## Spindle Controller Operations

The heart of the spindle controller is the 600-160 controller card. This is where all of the action in the spindle controller takes place. The hardware functions on this card can be divided into the following catagories.

1. Central Communications Processing (CCP), which consists of the 68000 CPU, the 68681 DUART, the pcf8584p I2C interface, and the ram and rom. The CCP is responsible for communicating with the Host Computer (via the DUART), the spin stand (via the I2C interface), the Motion Control Processor (DSP56002), and the Spindle Phase Lock Loop Controller (XC4010D).
2. The Motion Control Processor (MCP), which consists of the MC56002, a Xilinx XC4010D and a bank of high speed ram make up the Motion Control Processing hardware. The XC4010D is used to interface to the variaous motion sensors, such as the Micro E processing Board (which is not covered in this document), and the spindle encoder.
3. The Spindle Phase Lock Loop Controller (SPLLC) consists of an XC4010D, a bank of RAM, and all the analog electronics needed to process the digital signals from the PLL that get fed to the three phase power amplifier.
4. The Rotary Actuator Power Amplifier takes the output form the MCP and feeds the indicated current to the Rotary Actuator Motor on the Spinstand.

The other systems in the Spindle Controller that are not on the 600-160 board are the following.

1. Three Phase Spindle Motor Amplifier. This is a linear amplifier that supplys the apropriate current to the windings of the spindle motor to make it rotate. This amplifier currently supplies up to 9 amps with enough bandwidth to rotate a motor at 30,000rpm.
2. Micro E Encoder Processor Board. This board is supplied by the Micro E corporation. It takes the output form the Micro E encoder head that is on the rotary actuator and turns it into a 32 bit digital word. This card will not be discussed.

Of equal importance to the hardware is the firmware that runs on the 600-160 board. There are 4 software programable devices on the 600-160 card. The MC68EC000, the MC56002, and a pair of XC4010Ds. The code for all of these devices lives in the eproms on the 600-160 board, U63 and U75. The firmware for the XC4010Ds will be discussed in the hardware section, since, their firmware describes the action of hardware. The firmware for the MC68EC000 and the MC56002 will be discussed in the firmware section.

The firmware for 68K is by far the most complicated. It consists of the following systems.

1. Real Time Kernel (RTK). This is the real time operating system that keeps everything running smoothly. The job of the RTK is to schedule the tasks based on priority. Resource allocation is handled by a system of Semaphores.
2. Central Input Output (CIO). This is the IO manager for the system. It consists of software than manages the device drivers for the various devices in the system.
3. Motion Control Communications. This is the code that communicates with the Motion Control Processor(MCP). This is a full duplex interface.
4. Spin Stand Communications. This is done through the I2C port. These are the routines for communicating with the spinstand. This is a full duplex communication.
5. High Level Spinstand Drivers. These are the functions that make the spinstand operate. These function include the routines for initializing the spinstand, loading heads, unloading heads, etc.
6. Host Interface Drivers. These are the routines that handle communications with the host computers. This is a strictly half duplex operation. Only the Host computer can initiate communications. The spindle controller will only respond to commands from the Host.

The firmware for the 56K consists of the following systems. The 56K firmware is very simple, it is just a basic foreground/background structure.

1. PID Loop. This is the main motion control software. This routine runs on a timer interrupt.
2. Communications Interface. This runs on another interrupt. When a message is comming in from the CCP, this routine is activated, it grabs the data comming in, processes it, and puts it where it belongs.
3. Event Loop. This is the main loop that runs while nothing else is happening. If the Communications Interface puts something into the Event Queue, it will then be processed.
4. Hardware Inteface. This is another routine that runs from an Interrupt. If one of the motion control sensors fires off an interrupt (such as a limit switch), that action is processed here.

# I.
# Hardware Description

## Central Communications Processor.

This system consists of the 68EC000 (CPU), 68681 (DUART), pcf8584p (I2C), 256K of RAM, and 512K of ROM, and glue logic.

The glue logic consists mostly of PLDs. U62 (XC7336 on Rev A, XC9536 Rev B) performs most of these duties. It contains the main portion of the address decoder, a timeout generator used for dectecting bus errors, boot logic, and a clock divider.

The boot logic consists of a counter that will count the first four bus cycles. While this count is occuring, the ROM is mapped into address space starting at 0 so that the CPU can get the start address and the stack top from the first four words in the rom. After the first four bus cycles, the ROM is then mapped to its normal address space, and the RAM is then mapped to address 0. The CPU then loads the data it got from the rom in its program counter and stack pointer and jumps to the first memory location of the firmware.

U54 performs the priority Interrupt managment tasks. It encodes the various interrupt lines for the 68000 and decodes the Interrupt acknoledgement cycles going back to the peripherals.

U57 provides some extra address decoding as well as generating a clock signal to use with the logic analyzer when it is needed.

U56 is the interface to the Xilinx Gate Arrays on the board. This is a 1 bit interface, connected to data line 0 of the 68000. There are 8 memory locations within this chip that provide various functions.

For more information on U54, U56 and U57, refer to the VHDL code for these devices.

The system RAM is used to store both permanent and transisent data. An example of the permament data are the various paramenters that are used to set up the spinstand, and another is the ram-disk that is used to store various bits of setup data. The RAM is backed up with a 3 volt litium battery, controlled by a MAX691 microprocessor supervisory chip.

The system ROM is where all of the firmware for the system is stored. This includes the 68K, or course, but also the 56K and the two Xilinx gate arrays. Data from the roms is uploaded into the 56K through the 56002's host interface at boot up time. The Xilinx gate arrays recieve their data through U56 at boot up time.

The pcf8584p I2C interface chip is used to communicate witht the spinstand. There is not a whole lot to say about the I2C port in that the only external components are a pair of pull up resistors on the data and clock lines.

The MC68681 (DUART) searves a dual purpose of providing communications with the host computer, at 38.4Kbaud (COM1), and with a RS232 type terminal for doing debug work at 9.6Kbaud (COM2). While the UARTs are capable of full handshaking, the current software does not implement this, as there is a slight imcompatability with the way that handshaking is implemented on a typical PC. The MC68681 also provides a few other minor services, such as a real time clock, and one of the output pins changes state on each real time clock tick to keep the watchdog set in the MAX691 microprocessor suppervisor.

The MC68EC000 is the workhorse of the spindle controller. Its sole purpose is to coordinate all activity in the spindle controller. It does this with the help of the Real Time Operating System that it runs (which is discussed later).

## Spindle Phase Lock Loop Controller

The Spindle Phase lock loop controller, while in reality is strictly a peripheral of the MC68EC000, can be considered a sub system all its own. The spindle phase lock loop controller consists of the controller chip (U96, XC4010D FPGA), lookup table (U66,U67), Dual DAC (U58), plus a collection of analog electronics that primarily makes up the filter for the loop.

The controller chip (U96) is implemented in a Xilinx XC4010D. Besides the phase lock loop, there are also other subsystems in this chip as well. The subsystems consist of Phase Lock Loop, Tachometer (for measuring spindle rotational rate), Commutation Logic (for making a brushless motor rotate), and Interupt Logic, for taking outputs from other various hardware and turning them into events that will trigger semaphores in the Real Time Operating System.

The look up table (U66,U67) are a pair of battery backed rams that contain a sine table that is used to commutate the spindle motor. The exact contents of the lookup table is determined by the number of poles the motor has, as well as the pitch of the encoder on the spindle motor.

The Dual DAC (U58) is connected to the look up table and is loaded from the look up table as the motor is spinning. The memory locations that are used depend on where in the rotational cycle the motor is, as well as which motor phase is being looked up at the time.

The collection of analog electronics has several small pieces. First, right out of the spindle controller chip (U96), there is a differential amplifier (U48B) that takes the outputs of the phase frequency detector and turns it into a voltage that is either 0 volts (phase/frequency difference = 0), or +/- 5 volts (phase/frequency difference != 0). The output of U48B then goes into a 100Hz, four pole butterworth filter (U60A, U60B). The purpose of this filter is to remove all of the high frequency elements created by the frequency synthesizer that is in the spindle controller chip (U96). The output of the low pass filter then goes through one channel of the 3:1 analog mux (U65A,U65B,U65C). The output of the Analog mux then goes through a gain controller (U104B,U105A). The output of the gain controller then goes into a network for selecting one of five different phase lead networks (U99,U48D, U65D,U98A,U98B,U98C,U99D). The output of the phase lead network then goes into a voltage clamp circuit (U74A,U74C,U74B).

The output of the voltage clamp then goes into the commutation circuit, which is located in the FPGA (U96), the look up table (U66,U67) and Dual DAC (u58) (See all above). The control voltage goes into the reference inputs of the DAC, which gives us the product (multiplication) of the phase and motor drive, at the outputs of U64A and

U64D. These signals are then sent over to the power amplifier board (600-118) where they control the amount of current flowing through the windings of the spindle motor.

## Motion Control Processor.

The motion control processor consists of a Motorolla DSP56002 Digital Signal Processor, a Xilinx XC4010D FPGA, a bank of 256Kx24 bit high speed ram, and miscilaneous circuits.

Central to everything in the Motion Control Processor is the Motorolla DSP56002. The DSP communicates to the Central Communications Processor through its host port. This host port is also used to initialize the program and data memory in the DSP when the system boots. The code for the DSP is stored in the main EPROMs on the board. What the DSP does will be discussed more below when we talk about the firmware.

The Xilinx XC4010D (U92) provides the hardware interfacing between the DSP and the various encoders that the system uses. The primary encoder is the Micro E interface, which is 32 bits wide. The FPGA provides the logic for latching and reading the position data from the Micro E processing board. The other encoder in the system comes from the spindle motor. The Motion Control Processor also provides the service of accelerating and decelerating the spindle motor (this is a job best done by a standard control loop, as opposed to a phase lock loop which is more suited to controlling the spindle at a constant speed). The Xilinx XC4010D has the circuits in it for decoding direction and keeping track of the current position. Also in the XC4010D are the circuits for handling various hardware exceptions from things like limit switches.

U92 also provides an interface to the 16 bit DACs that are used for various functions. U31 provides the control voltage for the Rotary Actuator. U102 provides the control voltage to a posible linear stage (which is currently not used). U107 provides the control voltage for the Spindle Motor, and is used only to start and stop the spindle motor. U36 is used to provide a test point for debugging purposes. Its most common use is to monitor the following error in the rotary motor PID control loop.

The high speed rams (U1,U7,U13) provide storage for most of the DSP code. Some of the memory is also used to store both X and Y data. For more information on DSP memory spaces, consult the Motorolla documentation.

## Rotary Actuator Power Amplifier.

The Rotary Acutuator Power Amplifier consists of U94,U93,U101,U2 (Rev B), and U95.

This is a fairly straight forward circuit. U93A provides level shifting services. Since the power amplifier runs on only +12 volts, the control signal must be shifted so that 0 volts corresponds to one half of the power supply voltage. On the REV B board, U2 also level shifts the offset control pot. U93B is the summing node for the power amplifier and is also where the frequency compensation is located. R231, R232 and

C206 are the frequency compensation components. U94 is the main part of the power amplifier. It is configured in a bridge. R187 provides current sensing for the amplifier. U93C amplifies this voltage and feeds it back to the summing node, U93B. U101A,B and C also use the current feedback voltage to do current limiting. R212 and R215 set the current limit. U93D is used to provide the ½ power supply voltage to bias the power amplifier.

U95 is a high frequency oscilator, whose output goes into a voltage doubler. This voltage is used to bias the power amplifier chip (U94) so that it has an extended output voltage swing.

## II.
## Firmware for the 600-160 board

The current firmware level, as of the writing of this document, is 1.69.41. This was the last version for the Rev A PC board.

The firmware, as noted above, is divided between the MC68EC000, DSP56002, and the XC4010D FPGAs. We will discuss the MC68EC000 first, since this makes up the majority of the firmware.

### The Real Time Kernel.

This is the heart of the firmware. The real time kernel is relatitively simple, consisting of a couple major components.

The task management module, which is mostly in the file task.cpp and pq.cpp, create, schedule, and destroy tasks. Each task has a **Task Control Block** (TCB). The TCB contains all of the information needed to run the task, such as a pointer to the task stack, current stack pointer, task priority, task status, task data (if used), pending semaphore (for tasks that are waiting for a resouce to become availiable) and a few assorted other bits of data. See task.h to find out all of the data members of the TCB structure.

The important routines in task management (located in task.cpp) are as follows:

**TCB *CreateTask(void (*task)(void),int stacksize,int priority,char *name);**

This routine is used to create a new task. The first parameter points to the function that will be the new task. The second paramter specifies the stack size, the third parameter specifies the priority (the larger the number, the higher the priority), and the fourth parameter is a name for the task. The name is arbitrary, and is used for debug purposes only. However, it must be unique.

**int TDelete(TCB *task);**

This routine is used to destroy as task. The parameter specifies the task to be destroyed. The return value is important. It is posible that the task might be pending on a

semaphore (if a task destroys itself, this will never be the case), in which case, the task cannot be destroyed. A message will be sent through the semaphore to let the task know that it should destroy itself.

**void EnterInterrupt(void);**

When writing interrupt handlers, this function must be called before the interrupt handler can do a POST to any semaphore. The function is very simple. It just increments a very simple counting semaphore.

**void ExitInterrupt(int IntLevel);**

This function is also used when writing interrupt handlers. This is called just before an interrupt handler does a RTS. Exit interrupt must be called from an assembly language program, because the first paramter, IntLevel is derived from the status register that was pushed on the stack when the interrupt occured. A typical calling sequence is as follows:

```
move.w      60(a7),d0           ;get status register from stack
andi.w      #0x0700,d0          ;mask off junk
asr.w       #8,d0               ;shift over 8 bits
move.w      d0,-(a7)            ;push onto stack
jsr         _ExitInterrupt      ;call routine to exit interrupt
```

It should also be noted that care should be taken when modifying this routine. The IRQSWAP routine that is called in this function needs to know exactly how many words are on the stack, as it cleans up the stack so that a normal return from interrupt never occures. If it removes too much, or too little, the firmware will not run at all.

**int TDelay(int time);**

This function gets used quite a bit in the firmware. The units of time is 10mSec. So it is posible to generate delays of 10mSec resolution. This delay function uses the Real Time Clock, so the minimum time will never be less than the specified time. However, this function also uses a semaphore to pend on the delay going to zero, so, depending on the priority of the task that is doing the time delay, it is posible that the delay could be longer.

## Semaphores

The other major component of the Real Time Kernal are the semaphores. In task.cpp, the base class for the semaphores, Tsemaphore, is defined. There are several other semaphores that are derived from Tsemaphore (see queue.cpp).

Understanding how Tsemaphore works is very important. You will find semaphores sprinkled liberally throughtout the entire piece of firmware. Semaphores are important because they control the timing of the firmware. Anytime a task needs to wait

for something to happen, a semaphore is used to do the waiting. When a task pends on a semaphore, the task esentially goes to sleep. The task will awake again when something posts the semaphore, generally an interrupt, to let the task know it can continue with what it is doing. Many things on the system are tied to semaphores. All of the buttons on the front panel activate semaphores. The limit switches on the linear stage activate semaphores. When the three phase power amplifier overheats, it activates a semaphore. Any mechanical or electrical event that occures in the spinstand system will have a semaphore associated with it. Just keep in mind, that if an event happens, it activates a semaphore.

Tsemaphore is a C++ base class. It has quite a few member functions, although, the two of most concern are **post** and **pend**. The following is a list of Tsemaphore member functions and what they do.

**TSemaphore(int InitCount,int Mode,char *n);**

This is the constructor for Tsemaphore. The most important parameter is probably **InitCount**. The initial value of the Data Member **EventCount** determines a lot of the bahaviour of the Tsemaphore object. EventCount is a variable that will be decremented every time **Pend** is called, and incremented every time **Post** is called. See the description of these function down below for more details.

Also, when you look at the code for the Tsemaphore constructor, you will notice that there is a lot of activity that takes place. One of the things that is done is that the Tsemaphore constructor will put each new instance of a Tsemaphore into a MasterList that keeps track of all current Tsemaphores. This is partly so that the task manager can search for Tasks that might be pending on a semaphore.

A second list called the TimeoutList, keeps track of Tsemaphore objects that have a finite timeout limit. Tsemaphore objects that have a finite timeout limit are determined by the value of **Mode** in the paramter list. On every time tick from the real time clock, the task manager will go through the timeout list looking for tasks that are pending on these semaphores. If any tasks have overstayed their welcome, the task manager will post the semaphore with a timeout error.

The final parameter, **n** is a pointer to a name for the semaphore. This is primarily used for debugging to search for a specific semaphore.

**~TSemaphore();**

This is the destrutor for the Tsemaphore object. Not too much of note happens here. The Tsemaphore object is removed from the MasterList and the TimoutList, if it was in there.

**void *operator new(size_t size);**

This function overrides the **new** operator for this object. Again, if you look at the code for this function, you will see it is quite involved. One of the things you will note is that within this operator, a new Tsemaphore object is created in an unorthodox way...or maybe klugey is a much better word. Still, the first semaphore that is going to end up in the MasterList will be the blocking semaphore for the Tsemaphore **new** operator. After the blocker is created, that portion of the code is ignored.

You will also note that before allocating memory the **new** operator tries to get the memory from a Tsemaphore pool first. If there are no unused objects in that pool, it then will create a new object from the heap using **malloc**. You should note that an extra 128 bytes is added onto the size of a Tsemaphore. This extra space is used to store the name of the Tsemaphore object.

**void operator delete(void *t);**

Just like **new**, the **delete** operator is also overloaded. Instead of returning the memory to the heap, the object is put into a pool list of Tsemaphore objects. This helps slow down the memory fragmentation problems you can have by constantly calling malloc and free (new and delete).

**int Pend(int Timeout);**

Now, for the real functions. How Pend behaves has a lot to do with the value of **EventCount**. If you look at the code, pratically the first thing that happens is the value of **EventCount** is checked. If **EventCount** is greater than 0, the function will return immeadiately.

Some examples are in order. One of the places where a Tsemaphore object is used is in the **malloc** function. Malloc maintains some static variables and therefore, can not be interrupted while it is operating. In a multitasking environment, you cannot garrentee that this won't happen, since new tasks are constantly starting up all the time. To prevent this, a blocking semaphore is put at the head of the function. The initial value of **EventCount** is set to 1. So, when malloc is called and encounters this semaphore, **EventCount** is checked and found to be 1. The value is decremented and then the **Pend** function returns. Now malloc begins to process. Now say, right in the middle of processing, another higher priority task is started up, and it wants to allocate some memory as well. So, it calls **malloc** and does a pend on the semaphore. Pend checks the value of **EventCount** and finds that the value is 0. The value of EventCount is then decremented but instead of returning, the task is put on the list of pending tasks for that semaphore and the task is put to sleep. Rescheduling takes place, and eventurally, the original task that called **malloc** will be running again, and the malloc function will finally finish. When malloc is done, the last thing it will do is post to the semaphore, which will increment the value of **EventCount**. See **Post** below for more details.

Another example of using **Pend** would be for controlling access to a buffer. If you look at the code for the RS232 drivers, you will notice that data going into and out of

the serial ports is buffered. There is a semaphore associated with each buffer. Every time a character is to be put into the buffer, **Pend** is called. So, if the buffer has 1024 characters spaces in it, the initial value of **EventCount** is also set to 1024. This way, when the buffer is full, access to the buffer will be suspended until the buffer has some space in it.

And the third example is a semaphore that always suspends when you call **Pend.** All of the semaphores that pend on a front panel switch are of this type. **EventCount** is set with an initial value of 0, which cause it to suspend the first time **Pend** is called. That way, the task will resume as soon as a message is recieved that the button has been pushed.

The **timeout** paramter will only have an effect if the **mode** of the semaphore was set to the timeout mode.

The value that gets returned by the semaphore will be determined by the **value** parameter of the **Post** function, see below for more details.

**int Post(int Value);**

The Post function, basically undoes, what the Pend function (see above) does. If you look at the code for **Post**, you will notice that the first thing it does is checks the value of **EventCount**. If this value is greater than or equal to 0, this function does nothing except return immeadiately and increment the value of **EventCount**.

If **EventCount** is less than 0, or inother words, negative, **EventCount** is incremented, and then, the Pending Task list is scanned. The pending task list is basically a first in first out buffer. So, the first task pending on the semaphore is removed, the **Task->Status** is set to **Value**, the task is put onto the active queue and then a reschedule occures if we are at the task level, as opposed to the interrupt level.

**int GetCount(void);**
**void SetCount(int C);**

These function peform the simple task of allowing the user to alter the value of **EventCount**. This sort of allows the user to reset some semaphores. In general, this should be avoided.

**void Kill(void);**

This member function does a **Post** with a value indicating that the pending task should terminate itself.

**void Timeout(void);**

This is a static function. This function scans the **TimeoutList** looking for semaphores that should be timed out. If a semaphore is found that has timed out, a timeout error message is sent to the task.

**TSemaphore *FindTask(TCB *task);**

This is a static function. This function goes through the master list, looking for the **task** to see if it is pending on any of the semaphores.
All of the other member functions are for debugging. See the code listing for a description of their function.

## Central I/O Manager

The Central I/O manager (CIO) provides access to various I/O devices in the spindle controller. The main part of the code for CIO is located in cio.cpp. Functions to access CIO can be found in trap.s.

CIO performs the following services:
ADD_HANDLER, OPEN, CLOSE, GET, READ, PUT, WRITE, STATUS, and XIO.
The XIO service handles anything that does not fit into any of the other functions.
There are two data structures that CIO uses. There is an IO Control Block (IOCB), and a Device Name Lookup Table (HTABS).

HTABS is used by the OPEN function. OPEN will search HTABS looking for the device name and when it is found will put a device id entry into IOCB. This will be used every time CIO is called with a handle.

IOCB is the device descriptor. This data structure contains all the information that the system needs to know to operate on a particular device. Among the info in the IOCB is a pointer to the jump table (pointed to by HTABS).
OPEN also locates a free IOCB entry to use. When open returns, it will return a handle to that IOCB.

CLOSE is used to release the IOCB and DEVICE. In general, it is a good idea to have only one channel open to any one device.

GET is used to get a character back from the device. Other than that, the behavior of this function is entirely dependent on the device driver.

READ is similar to GET, except the user can specify the number of bytes to read. READ will continue to attempt to retrieve the desired number of bytes until it has either accomplished its goal, or an EOF (End of File) terminates the action. READ is useful as it reduces the overhead in reading back large blocks of data.

PUT is used to send a single character to a device. Other than that, its behavior is entirely dependent on the device driver.

WRITE is similar to PUT, except the user can specify the number of bytes to send to the device. WRITE will output its entire contents to the device until it is empty. WRITE is useful as it reduces the overhead in writing large blocks of data. It should be noted that WRITE will not do its job indivisably. In other words, if two tasks are writing to the same device (which is allowed), the actual output will be indeterminate. If the WRITE function needs to be determinate, then the user needs to modify the device driver so that it uses a blocking semaphore to prevent indeterminate behavior. And example of this can be seen in the WRITE routine in the RS232 driver.

STATUS is used to get device status. What the status routine returns is dependent on the device driver. STATUS can be used in one of two ways. You can either pass a handle or a name of a device and status will do an implied OPEN.

XIO handles all other device services that are not handled by any of the above. What XIO does is entirely dependent on the device driver. Like STATUS, XIO can be used in one of two ways, either passing a handle or a name of a device and XIO will do an implied OPEN.

## Motion Control Communications

Communications with the MC56002 is handled by a Motion Control Communications Driver. This code lives in servo.cpp and dspirq.s. This piece of code is moderately complicated. It can be divided into Interrupt Handlers, Low Level Drivers, and User Interface routines. The user interface routines are far too numerous to list, so, they will only be covered in general terms.

**void HandleDspGet(void);**

This function is the interrupt routine for recieving data from the DSP chip. One thing that should be noted is that this routine will continue looping while there is data in the DSP Host Data Recieve Register (this is the register that is sending data back to the host). The purpose of this is to cut down on overhead. The DSP chip can send data much faster than the 68K can read it, so, once the data starts coming, the 68K might as well stay around until it is all read.

Inside of this routine is a state machine that is used to determine what it is that the DSP is sending back so that the input buffer semaphore can be posted at the correct time.

**void HandleDspPut(void);**

There is nothing really remarkable about this function. It is the interrupt handler for transmitting data to the DSP. Again, as long as there is data to transmit, this routine will keep sending data to the DSP, as the DSP can read the data much faster than the 68K can send it. This helps keep the overhead lower.

**void HandleDSPCommandsTask(void);**

This function is a task that decodes the data that it recieves from HandleDspGet and dispatches messages through the appropriate semaphores. There are four major classes of messages, **DSP_COMMANDACK** is a message that is sent to acknoledge the reciept of a command, **DSP_EXCEPTION** is a message that is sent to indicate some situation that needs attention, **DSP_MOVECOMPLETE** is a message that is sent to indicate that some commanded action has completed (generally, a move), and **DSP_DEBUGSTRING** is a message that is used in debugging the system.

**int telSendDataBlockCommand(CommandFrame cmd, long *d,int n);**

This function is the base function for sending a block of data to the DSP. The first parameter, **cmd**, is the command frame structure. This structure is filled in before this function is called with all of the various command data that will be used by the DSP to figure out what it is supposed to do. The second parameter **d** is a pointer to an array of longs that is sent to the DSP. It should be noted that the DSP uses a 24 bit word, so, the upper 8 bits of each data word end up getting lost. The final parameter, **n** is the number of longs to be sent in the array.

**int telSendDataCommand(CommandFrame cmd);**

This function is used to send just a command frame to the DSP. This is, in general, the most common way of communicating with the DSP and makes up about 50% of all of the functions.

**int telGetDataCommand(CommandFrame cmd,long *d,int n);**

This function is used to get information back from the DSP. This is also a very common function, and makes up about 48% of all of the transactions between the DSP and the 68K. The **cmd** parameter, again, is the command frame structure, and it is filled in before making this call. It will tell the DSP what kind of data is to be sent back. The **d** paramter points to an array of longs that will be filled with data by this function. The final parameter **n** lets the function know how much data to put in the array.

**int SendDSPBuff(CommandFrame cmd,long *b,int size);**

This function is used to send a command frame and a data buffer to the DSP. All of the data is transfered into the transmit buffer, and then the interrupt enable for the transmit register in the DSP is enabled. Doing it this way is much more efficient as the DSP can read the data much faster than it can be sent. This function is used by telSendDataBlockCommand function.

**int SendCommandFrame(CommandFrame cmd);**

This function is used to send just a command frame. Again, all of the data is loaded into the DSP transfer buffer and then interrupts are enabled. This function is used by telSendDataCommand and telGetDataCommand.

All the rest of the functions, for the most part, in servo.cpp make use of the above functions to execute the various commands.

## Spinstand Communications

Communications with the spinstand are primarily accomplished over an $I^2C$ port. The protocol used is based roughly on the ACCESS.BUS standard. Both sides of the communication are Bus Masters (the spindle controller and the spinstand). This allows a full duplex communication between the two devices. Care must be taken in modifying the code at either end, as the protocol is tricky in that both sides need to check to see if an arbitration is lost when both sides are trying to communicate. The way the protocol is set up, the spindle controller should always loose that arbitration.

The files that contains the $I^2C$ communications routines are I2C.CPP and I2CIRQ.S.

It should be noted that in looking at I2C.CPP that there are a lot of semaphores and tasks in this file. There is a lot of activity that takes place in the communication with the spinstand. Not all of this will be covered here. Refere to the code for more specific information.

**void I2CSendMessage(unsigned char *d,int size);**

This is the function that is used to send a buffer of data to the spinstand.

The first parameter **d** is a pointer to an array of bytes to transmit.

The second parameter **size** is the number of bytes to transmit.

What this function does is loads up the transmit buffer with the data. It then pops the first byte back out of the buffer and then waits for the $I^2C$ port to be not busy. This is nessesary to do it this way for several reasons. First, the $I^2C$ chip does not generate an interrupt when it is not busy. Second, even if it did, the latency would be so long that you could not be sure it was not busy anyway. Even the way it is done, with a very tight loop checking the status, you cannot be assured that the bus isn't busy when that first byte is stuffed in the $I^2C$ transmit buffer.

**void I2CHandleInterrupt(void);**

This is the interrupt handler for the $I^2C$ port. This is not the cleanest piece of code in the world, mostly because the interface chip that is used is not the cleanest piece of hardware ever designed. The interrupt handler handles both the transmitting and recieving of data.

**void DispatchI2CMessages(void);**

This function is used to decode the messages that are recieved via the $I^2C$ port. Once the command is decoded, the appropriate messase is sent through the appropriate semaphore. The command cattagories are FPCOMMAND_BUTTON_STATE which are the messages created by pushing buttons on the spinstand front panel, FPCOMMAND_STAGE_STATE which are messages sent out when the sensors on the stage change state, FPCOMMAND_VERSION which is sent in response to a version request command, and FPCOMMAND_SPINSTAND_STATE which is sent to a status request command.

It also should be noted that this function is a task. It will run always when a new message comes in.

**void I2CSendDemon(void);**

This function, which is a task, is used to send $I^2C$ packets to the spinstand at evenly spaced times. This is done not to be orderly, but because there is a bug in the code that runs in the micro controller in the spinstand. It will choke if too many messages come too fast, and it doesn't properly implement controlling speed on the $I^2C$ bus properly. So, in other words, this function is a kluge.

## High Level Spinstand Functions

These functions, which are located in acutrac.cpp, are the ones that manipulate the spinstand so that it can test the heads. The main job of these functions is to load the heads onto the disk, and unload the heads from the disk.

There are several C++ classes used in this code. The most important one is **AcutracDeviceTable**. This class contains all the information that is needed to operate with a particular head. A lot of the information has to do with the metrics of the system, such as track width, arm lenth, reference angle, etc. **PidParams** is another class, and is associated with AcutracDeviceTable. This class contains the tuning information for the servo. The functions in accutrac.cpp will automatically select the correct set of tuning parameters depending on weather the heads are mounted on the tooling or not.

**ACTION_STATUS** is a class used to retrieve status about the operation of some of the routines. Some routines are spawned as tasks which can take a considerable amount of time to execute. The host computer will retrieve system status back from the spindle controller through the ACTION_STATUS class.

The functions **angle_to_icts**, **icts_to_angle**, **track_to_rad**, **rad_to_track**, **rad_to_angle**, **angle_to_rad** are rather important as these are the functions that convert tracks into useful information that the spindle controller find useful and vice versa.

**int AcutracINIT(char *name);**

This function is used to initialize the data in acutrac.cpp. It does not do anything to the hardware. Its biggest job is to try to load the **.afg** file from the ramdisk that is pointed to by the parameter **name**.

**int AcutracRESET(char *fname);**

This function performs the full initialization. This function calls AcutracINIT, plus, it sets up and creates a task that will execute the hardware initialization. Once this task is created, AcutracRESET will return. It will be up to the original caller to monitor the progress of the hardware initialization task by using ACTION_STATUS.
The parameter **fname** points to the **.afg** file that contains the hardware information.

**int AcutracSTART(void);**

This function is used to spawn the **start** task, which loads the heads onto the disk. The task that is selected depends on what the contents of the **.afg** file were. HSA and HGA are the two selections. Once the start task begins, it can be monitored through the use of ACTION_STATUS.

**int AcutracSTOP(void);**

This function is used to spawn the stop task, which unloads the heads from the disk. The task selected depends on what the contents of the **.afg** file was. Once the task begins, it can be monitored through the use of ACTION_STATUS.

**int AcutracLIFTHEADS(void);**

This function is used to spawn the lift heads task, which, lifts the heads off of the disk. It does this by returning the heads to the load/unload radius. Once the task begins, it can be monitored through the use of the ACTION_STATUS structure.

**int AcutracRELOADHEADS(void);**

This function is used to spawn the reload heads task, which put the heads back onto the disk after they were lifted. Once the heads are back on the disk, the heads will return to their position before they were lifted. Once the task begins, it can be monitored thorugh the use of the ACTION_STATUS structure.

For each product, there will be a **Start, Stop, Lift, Reload and Reset** task. These tasks perform all the actions nessesary with the hardware. Many of the solenoids and sensors on the spinstand perform different functions depending of the product being tested. It should be noted that for every action that takes place, there needs to be a sensor to allow the software to know when an action is complete. This is needed because of the fact that this is a real time operating system. Tasks will suspend whenever they need to

wait for something, so that other tasks can keep doing their jobs. When a sensor fires, it will cause the apropriate semaphore to be fired so that the task can resume.

## Host Interface Drivers

The host interface rotines are located in serlprot.cpp and main.cpp. Main.cpp contains the main task for doing this job. Serlprot.cpp contains the meat of the code. First, we need to learn a little bit about the protocol that is used to communicate with the spinstand.

## Teletrac Spindle Serial Protocol

Teletrac supplies a non-ascii serial interface that has the added advantage of being more rigorous in how it checks the data flowing between the host and the spindle controller. This format is binary in nature with the data grouped in packets.
The format of the packet is quite simple and will be the same for all devices supporting this data format. Because of the fact that the packets are identical in format, any device from Teletrac that understands this data packet will be able to receive them without any problems and pass them on if they are not for the receiving device.

The data format for version 1 is as follows:

```
BYTE 0:0x80          ;Packet header indicator
BYTE 1:0x01          ;Packet version number
BYTE 2:<count>       ;Number of data bytes (excluding header and chksum)
BYTE 3:<device id>   ;Device data packet intended for
BYTE 4:start of data
        .
        .
        .
BYTE n-1:end of data ;<count> bytes of data
BYTE n:<checksum>    ;checksum of all previous data and header bytes
```

Because of the nature of the packet, they are completely independent of the type of data that is actually sent between the devices. This means that a common set of subroutines can be used to handle the packets regardless of the meaning of the data.

### Host Side Interface

Teletrac supplies the following routines for use under WINDOWS for communicating with Teletrac Packets.

**#include <protocol.h>**
**int TelInitProtocol(int port);**

This function is used to initialize the serial port for use with the teletrac protocol. Under MSDOS, the serial port uses an interrupt driven serial handler. It is very important therefor to insure that TelTerminateProtocol() is called before the program exits, or your system WILL crash, guaranteed.

This function returns 0 on success, non-zero on fail.

**#include <protocol.h>**
**void TelTerminateProtocol(void);**

This function terminates the Teletrac Protocol handler. It is very important that this function is called BEFORE your program exits or bad and evil things will happen to your computer, especially under MSDOS.

**#include <protocol.h>**
**int TelReceiveProtocol(RECPROT *r, int timeout);**

The value **timeout** is the maximum time that the function will wait for data to be sent back before it gives up and returns a timeout error. This value is a compromise between how long you want the computer hung up waiting for a message it will never get, and the amount of time that the transmitting device might take to process the command it received before sending back the reply.
For example, if you sent a commnad to some device and told it to move 10 inches at one inch per second and then tell me when you are done, you would probably want to set the timeout to 11 seconds, 10 seconds to give the device time to do the move, one second padding to make sure we don't get an erroneous timeout.
This function is used to receive a packet from a transmitting device. The data is put into a data structure that can then be operated on later. The data structure **RECPROT** has the following form:

```
typedef struct {
        int version;        //version number of packet
        int size;           //number of bytes in data buffer (not maximum size)
        int devID;          //id of device data was intended for
        char buff[BUFFSIZE]; //data buffer for data received
}RECPROT;
```

The novice C programmer should pay particular attention to this data structure. It may not be what it seems. Data member buff is the buffer where the received data will be put. The legnth of one may seem to be a bit short, especially considering the fact that up to 255 bytes can be transmitted with the protocol.
The user can allocate a **RECPROT** one of two ways. The easiest way is to use the supplied function **TelAllocateRECPROT()** (see below), or to use the function malloc. One would use malloc as follows:

```
RECPROT *r = (RECPROT *)malloc(sizeof(RECPROT) + BUFFSIZE);
```

BUFFSIZE would be a user supplied macro that would evaluate out to the desired maximum buffer size. If you want to be on the safe size, 256 is always a good number. When the variable **r** goes out of scope, or when the buffer is no longer needed, the user should use the standard **C** function **free** to give the memory back to the OS so as to

prevent memory leaks. This is also true if you use the function **TelAllocateRECPROT()**.

The function **TelReceiveProtocol()** can return the following values. Values less than zero indicate that an error has occured. These values can be:

**TEL_PROT_TIMEOUT** //Transmitting device did not send any data
**TEL_PROT_CHECKSUM** //data packet had a checksum error

If the packet was received ok, then a positive number is returned that indicates the number of data bytes there are in the data buffer.

**#include <protocol.h>**
**int TelSendProtocol(int devID, int count, char *buff, RECPROT *in, int timeout);**

This function sends a data packet out to a receiving unit. On the IBM PC, generally, this is done as the host, so provision has been made to get the reply back. After the data packet has been sent, this function will then call **TelReceiveProtocol** and return back the value received from that. **TelSendProtocol** does not itself generate any errors, so if **in == NULL**, this function will always return 0.

This function uses the parameters passed to build the packet. **devID** is the ID of the device that you want to send the data to. **buff** is the buffer that the data you want to send is stored in. **count** is the number of bytes in buff. **timeout** is used by TelReceiveProtocol, and is the maximum time to wait for data in milliseconds.

**#include <protocol.h>**
**RECPROT *TelAllocateRECPROT(int buffsize);**

This function is a convienient way to allocate memory for a receive buffer. This function calls malloc, so the buffer can be freed when it is done being used by calling **free()**. Generally, it is safest to set buffsize to 256.

## Controller Side Functions

**void TelSendProtocol(int handle,int devID,int count,char *buff);**

This function is used to send data back to the host. The **handle** parameter is the handle for the port to send the data through. This handle was obtained by using the **Open** function (see CIO above). The **devID** paramameter is the ID of the device to send the data to. The **count** parameter is the number of data bytes in the **buff** parameter. And the **buff** parameter points to an array of bytes to send back to the host.

**int TelReceiveProtocol(int handle,PROT_DATA *d);**

This function is used to recieve data from the host. The way this routine is set up is it will pend until a packet has arrived in the port buffer. The first parameter **handle** is a

handle for the device to recieve data from. This handle was obtained by using the **Open** function (see CIO above). The **d** parameter points to a PROT_DATA structure where the in coming packet data is stored.

**int TelCommandParser(int count,char *cmd,char *obuf,SYSTEM *sys);**

This is the function that is called to decode the data that came in the packet. The first parameter **count** is the number of characters that are in the command buffer. The second parameter **cmd** is the command buffer which holds the characters to be decoded. The third parameter **obuf** is an array of characters where all return data is stored. And the fourth parameter **sys** points to a SYSTEM structure that contains useful handles for variaous devices. The function itself returns the number of characters that were put in **obuf.**

# III.
# Motion Controller Firmware

The Motion Controller, which consists of the DSP56K, is responsible primarily for doing the servo control for the rotary actuator. It also provides the control for accelerating and stopping the spindle motor.
The major components of the DSP firmware are the Command Interface, the Event Manager, the Hardware Interrupts, and the PID routine.

## Pid Routine

The PID routine is by far the most complex and important routine in the DSP code, so that will be discussed first. The two files that are important are spid.h and pid.asm. Because a lot of the other code is written in C for the DSP, the file spid.h allows the C code to access functions and data structures used in pid.asm.
The PID routine runs on an interrupt. A timer in the hardware provides an interrupt at the sample rate, which can be as high as about 40KHz. For the current implementation, this is the maximum speed.
The entry point to the PID is at lable **Fpid**. The first thing that gets done is the profile generator is checked for its current status. The profile generator is very simple, but it does consume a rather hefty amount of code. This will, hopefully, be reduced in future versions.

Profiles are generated using the following algorythm:

$$A_n = A_{n-1} + J$$
$$V_n = V_{n-1} + A_n$$
$$X_n = X_{n-1} + V_n$$

Where A is the acceleration, V is the velocity, X is the distance, and J is the Jerk. If you look in the file spid.h for the structure PID_PARAMS, you will not the data members **Flags**, **Jerk**, **VCount**, and **SCount**. These variables are used to control the profile generator. The **Flags** variable indicates the current state. The states are

FLAGS_START, FLAGS_ACC1, FLAGS_ACC2, FLAGS_CVEL, FLAGS_ACC3, FLAGS_ACC4, and FLAGS_END. FLAGS_START is set by the main program when it wants a profile to start. The actuall move is divided up into 5 sections, FLAGS_ACC1, FLAGS_ACC2 are the initial acceleration. The difference between these two is that for FLAGS_ACC1 the Jerk is positive, and for FLAGS_ACC2 the Jerk is negative. The next section is optional, depending on what value the Jerk is and what the maximum velocity is. If the move would exceed the maximum velocity, the middle of the move is filled with a section where the stage moves with a constant velocity. FLAGS_CVEL is this section. The lenth of this section is specified before hand by setting **VCount** to a non zero value. The last two sections FLAGS_ACC3 and FLAGS_ACC4 are the deceleration sections, with the Jerk being negative for FLAGS_ACC3 and positive for FLAGS_ACC4. FLAGS_END indicates an idle state. The **SCount** variable indicates the number of cycles to execute the FLAGS_ACCx states. They all execute the in the same number of cycles and this value is a parameter that is set by the user. In the Xpress Filter Dialog box, this is called the **S Curve Time**. So, to recap, the above equations are executed, with only the value of the Jerk being changed to cause the stage to move.

At lable location **_PID512** is where the new position calculation is done. The old position is read in first, this is so that the velocity can be calculated (this will be needed later when it is combined with the derivative gain). Also, if there is a Position Error Signal (PES) this is also added in at this time. The current position itself is read from a register in the FPGA.

Following this is were the profile is calculated. As soon as the commanded position is determned, the following error is calculated by subtracting the commanded position and the current position. This value will be used later. Also calculated at this time will be the commanded velocity and the commanded acceleration. These are saved for later to be used with the feed forward gains.

The next section is where the **dither** is added in, if it is enabled. The dither is basically just a term that is tied to an internal sine wave generator. The frequency is determined by a **phase-accumulator** frequency generator in the FPGA. This register is read and used as an index into a 256 word long sine table that is built into the DSP. This is located in the Y memory at location $100. The dither oscilator is very important for doing the bode plots to determine the frequency response of the system.

The next section at lable **_PID501** is where the following error is calculated. Among the things done at this point is to determine if the following error exceeds a predetermined threshold and to make sure that the following error does not wrap (this would be disasterous).

The next section, starting at lable **_PID300** is where the following error is integrated. Among the things done here are the integrator is checked to make sure it never exceeds a certain predetermined value and that it doesn't wrap (which would cause no end of trouble). The value of the integrator will be saved for later. It should be noted that in one mode of operation, the integrator will check to see if the commanded velocity

is non zero. In this mode, the integrator will hold its value (i.e. stop integrating) until the commanded velocity returns to zero. This help the settling time because the integrator can build up quickly while there are large following errors while making a move.

The next section is where all of the various values are combined with their respective gains to create the output that will be sent to the motor.

The next section starting at lable **_Pid201** is the first of two notch filters. The input to the notch filter is the final output from the PID filter. The notch filter is a very simple state variable filter. The maximum notch frequency can be easily calculated by using the following formula:

Fc = <Sample Rate> / (2 * 3.14159).

So, for 19.5KHz sample rate, the maximum notch frequency will be 3.1KHz. There are basically three parameters that can be adjusted in the notch filter, Frequency, Q and depth. In general, you will want to keep the Q at about 1. The higher the Q, the more problems you are going to have with the filter.

The second notch filter begins at lable **_Pid202**. It is basically the same as the first notch filter.

At lable location _Pid222 we finally send the output of the PID filter to the motor.

The rest of the PID interrupt is dedicated to analysis. Right after the PID output is sent to the motor, we perform the Histogram function. In the Xpress program, you can veiw the histogram in the Filter Dialog. What the histogram does is it takes a number of samples, generally 4096, and uses the following error as an index into 32 bin counters. If the following error is 0, then the bin that corresponds to 0 following error is incremented by one. If the next sample the following error is –2, then the bin for –2 is incremented by 1. When all 4096 samples have cast their vote, ideally we should end up with a bell curve centered around 0 following error. A lot of information can be gained about the performance of the system by looking at the histogram.

Following the histogram there is a pair of spectrum analyzers. The spectrum analyzers are used in doing the bode plots. They are spectrum analyzers in the true sense of the word. A pointer is used to look at the signal to be measured. That signal is then multiplied by a unity gain sine / cosine wave from the dither oscilator. This will give us the real/imaginary components needed to get the magnitude/phase response. The outputs from the sine/cosine multiplication is then fed into a pair of simple low pass filters. The host computer can then read this data and further process it to create the bode plots.

## Command Processing

The command processing for the motion control processor is done in the file S_commnd.c. The command processor takes advantage of the structure of the host

interface port on the Motorolla DSP56002. The main processor (the 68K) sends the command data to the DSP, and then the final act is to write the correct vector in the the DSP command interrupt vector register. This generates an interrupt in the DSP and it ends up in the function **Command()** where it then decodes the command data and dispatches to the proper function. It should be noted that the Command() function has a lower priority than the PID function, so the PID loop will still be executing while Command() is trying to do things. In some of the critical areas, all interrupts are disabled, such as when the current position is trying to be read.

Some of the commands cannot be executed while in the Command() interrupt routine. In these cases, a message is sent to the **Event** processing. A good example of this is the callibrate and move commands.

### Event Processing

Event Processing runs all the time when nothing else is going on. The file that takes care of this is S_event.c. It is just a big software loop that constantly checks the status of various thing, especially the event message buffer. If a message is found in the event buffer, it is then decoded and the proper function is dispatched to process it. It should be noted that during critical sections of code interrupts are disabled. This is kept to a minimum.

### Hardware Interface

Hardware interfacing is handled in S_hrdwar.c. The main function in this file, HandleHardware(int flags), is an interrupt handler. It is called from an assembly language routine where it reads a status register in the FPGA and passes it to the HandleHardware function. The hardware interface is all done in the FPGA. About the only thing that interfaces to the DSP these days are the limit switches on the linear stage. If a message needs to be sent to the host processor (68K), it is put in the out going message queue before returning. Otherwise, status bits are set that are used by Event Processing.

## Bonus Section
# Spindle Gate Array (U96)

The current version of U96 (as of June 19, 2002), is implemented in a Xilinx XC4010D using the very old XACT 6.0 running under DOS. Schematic entry was done with Orcad SDT386+. Niether of these products is availiable any longer, so when updates are required, the design will have to be reimplemented with newer tools using a different gate array.

When I refer to a page number of the schematic, I am refering to page number that is in the title block of the schematic, and not the page number of this document. I

will be going throught the schematic page by page explaining what it is that you are looking at.

## Sheet 1

This is the top level schematic. All of the major blocks are on this sheet, with the interconnects shown between them. The major blocks consist of the **BUSIFC**, which is the bus interface controller. This block is what interfaces to the main host processor, the 68000 in this case. Next is the **ENCODER** interface. This block interfaces with the encoder that is on the spindle motor. Next is the **PLL**. This block contains the logic that makes up the digital portion of the Phase Lock Loop controller. Next is the **TACH**. The tach is what is used to measure the rotational rate of the spindle motor. Next is the **IO** interface. Besides being IO, this block contains logic that will generate interrupts when lines change states. Next is the **PHASE_ANGLE** block. This block contains the comutation logic for the spindle motor. The phase_angle title is perhaps not a very good one. And finally the **OUTPUTS** block. This block contains output drivers for some of the logic functions.

## Sheet 2

Sheet 2 is the top level schematic for the **IO** block. BX70 is an 8 bit register that will hold the data for the 4 8 bit DACs that are on the 600160 board. Down below there are a bunch of D flip flops starting with U181 that make up a delay for the data strobe that will eventually be passed on to the DAC that is going to actually latch the data. The DAC which actually latches the data is determined by the two lines DACSTBSEL and DAC_SEL.

Another 8 bit register, BX71, is used to hold the interrupt enable bits for the block that is called **ERROR**. U190,U192 and U193 are used to differentiate the signal called SCLK (which is a low frequency clock) so that it will generate a pulse that is 1 master clock time wide. This is also passed onto the module labled ERROR.

## Sheet 3

Sheet 3 is the top level shcmatic for the **ERROR** block. It contains the blocks **CHANGE** which is a block that detects when the inputs lines changes states (both low to high and high to low). And another block called **HOLD** which holds the current state of the interrupts during an interrupt acknowledge cycle.

As you can see, this is a very busy sheet. On the left are the input pins which go through a set of buffers before going into the change block. The outputs of the change block will be a pulse that is one clock time wide every time there is a transistion on the input. The outputs of the change block go into a bit of logic which consist of three and gates, an or gate, and a D flip flop which will generate an interrupt depending on the state of the interrupt enable signal (EN0..7). The interrupt can be cleared by writing tot he WD_CLRIRQ port and setting the data bit for that flip flop high on the data bus (D0..7).

The output of the interrupt registers then goes into a priority encoder consisting of U205 and U204. The output of the priority encoder then goes into the **HOLD** block where the state of the interrupts is frozen during an interrupt acknoledge cycle.

It is also posible to read out the state directly of the input lines as well as the state of the interrupt register, as can be seen in the lower right hand corner.

**Sheet 4**

Sheet 4 is the shcmatic for the **HOLD** block. It is very simple, just a standard data latch.

The tristate driver for the interrupt line is also on this page.

**Sheet 5**

Sheet five is the schematic for the **CHANGE** block. This is the logic that detects when the input level has made a transistion from low to high or high to low for inputs I0,I1,I2 and I3. These pins are a little different as they are sampled at a low clock frequency. This is to help in avoiding a lot of noise. Input I4 detects changes sampling at the master clock frequency. Inputs I6 and I7 use a quadrature detector (these inputs are used for the knob on the front of the spinstand). It is sampled at the master clock frequency divided by 16.

**Sheet 6**

Sheet six is the schematic for the **OUTPUTS** block. This is a very simple schematic. It basically takes some signals from a register on Sheet 1 and decodes them into one-hot lines.

**Sheet 7**

This is the top level schematic sheet for the **ENCODER** sheet. This sheet contains blocks **ANALYSIS** which provides signals useful for analyzing the spindle motor. It also has the POSITION block which maintains a 16 bit position for the spindle motor.

The main thing that happens on this page is the processing of the encoder inputs which consist of the **AQUAD**, **BQUAD** and **INDEX** signals. These signals are syncronized to the internal clock and then differentiated so that we get a pulse 1 clock wide every time they change state. It should be noted that we have the option of using either raw or processed signals for the PLL (U143,U144,U145). Normal operation uses the raw signal. This is the best way to get the lowest phase jitter.

The DIR (direction) and COUNT_EN (count enable) are sent from the **POSITION** block to other blocks. These signals are also combined with U146 (a DFF)

to create a true direction signal that is sent on to the **ERROR** block covered above. This is used to provide an interrupt every time the spindle changes direction. This is useful for determining when the spindle has stopped.

**Sheet 8**

Sheet 8 is the top level schematic for the **POSITION** block. It contains the blocks **PHASE_DETECTOR**, which determines not only which direction the spindle is going, by also detects when the edges of the encoder go by. And also the block **POSITION**, which was an unfortunate choice of lables, because that is going to just confuse things. This block contains the position register.

**Sheet 9**

Sheet 9 is the phase detector for the spindle encoder. Very straight forward. It provides both a direction signal and a count enable signal.

**Sheet 10**

Sheet 10 is the position counter (this is labled **POSITION** on sheet 8, sheet 8 is also labled **POSITION**, try not to get confused). BX56 is a 16 bit counter and BX55 is a 16 bit register that is used to hold the data so it can be read.

**Sheet 11**

Sheet 11 is the **ANALYSIS** block. This block is basically a bunch of counters and registers that allow you to generate a pulse when a particular encoder line comes around. You can set up the START output to make a trigger at line 55 and the STOP output to make a trigger at line 103 for instance.

**Sheet 12**

Sheet 12 is the **BUSIFC** block. This is the logic that interfaces up to the 68000. U61 is a DFF that is at the start of a chain of flip flops that are used for generating wait states, when needed. Write strobes are generated by U92 and U98. This page also contains the address decoding for the read and write strobes.

**Sheet 13**

Sheet 13 is the ENCODER block. This page also contains the block **PHASE_COUNT_LIMIT**. Also on this page is the block **RAM_DAC_CTL**. This is the page that generates the comutation signals for the spindle controller. BX13 and BX14 are the two phase registers. These are the registers that the phase relationship between the comutation signal and the index pulse are written. There is one register for each output on the spindle controller. On the 600-160 schematic, these outputs are labled sine and sine+120. BX5 is the phase counter. It is reset to zero on each index and is

incremented or decremented depending on the state of the COUNT_EN and DIR signals that come from the phase detector discussed above. The output of the phase counter is then added to the phase registers, one at a time, and then this value is used to look up a phase value in an external SINE LUT (SLUT) and then latched into an external 12 bit DAC.

**Sheet 14**

Sheet 14 is the **RAM_DAC_CTL** block. This block provides the logic for accessing the external Sine Look Up Table (SLUT) and also writing the contents of the SLUT into the external 12 bit DACs.

**Sheet 15**

Sheet 15 is the **PHASE_COUNT_LIMIT** block. This sheet has a register (BX18) that is used as a upper limit on the value that the Phase Counter on Sheet 13 can have. Also, the value that the limit can have depends on the direction of rotation. This is taken care of by a multiplexor (BX16).

**Sheet 16**

Sheet 16 is the **TACH** block. This sheet is the one that measures the rotational rate of the spindle motor. U401 and U402 prescale the clock by dividing by 3. BX24 then divides the clock further by 32768. This is then used as a time base to measure the frequency of the encoder.

**Sheet 17**

Sheet 17, which is a block of Sheet 16, is used to create a 16 bit value that is used as a system timer to record the execution times of the various tasks in the multitasking system.

**Sheet 18**

Sheet 18 is the PLL block. This is the digital portion of the Phase Lock Loop controller that controls the speed of the spindle motor. It contains a block PHFRCOMP which is the phase frequency detector for the loop.

There are three registers. BX52 controls the divide ratio of the divider that recieves the reference frequency from the referenec divider. BX50 controls the master reference divider which is a phase-accumulator type divider. This type of divider provides much finer control over the reference frequency. BX51 controls the divider that divides down the encoder frequency.

**Sheet 19**

Sheet 19 is the **PHFRCOMP** block. This is the phase frequency detector. U120 and U121 make up a standard D flip flop implementation of this circuit. The other logic on this page is an attempt to make a lock indicator. This is of marginal use.

## Motion Controller Gate Array (u92)

The Motion Controller Gate Array handles the interfacing between the Motion Control Processor and the hardware it is interested in. The primary purpose is to interface to the Micro E encoder interface, but there is also hardware that implements the frequency generator, real time clock, interrupt hardware, DAC interface, timer and spindle motor interface.

### DSP Interface

Sheet 11 of the schematic

This page is the interface between the DSP chip and the internal workings of the Motion Controller Gate Array. It contains the data bus buffer (BX5), Address Decoders (BX1(read),BX2(write)). Of special interest is the write strobe generator u14 and u23 and their associated logic. There is also the clock divider U34,U35 and U262. The primary frequency that the clocked logic runs at on the chip is 10MHz, but there is also some 5 MHz logic as well. The 20MHz clock is used to clock data off the data bus into the data registers throughout the chip.

### Micro E Interface

Sheet 6 of the schematic

This page contains the interface between the Micro E encoder and the Motion Control Processor. BX70 is the 32 bit input port that the micro E data comes in on. The sample pulse LATCH_POS fires off a onc shot consisting of u211, u210 and U214, u217. The signal then makes its way up to DATA_REG BX69 where it is latched. BX75 and BX76 is an offset register, which is added to the position. The offset register is used to zero the position. The output from the adder then is gated onto the data bus using tristate buffers.

### Frequency Generator

Sheet 12 of the schematic

The frequency generator is quite simple, and yet, complicated. At the heart of the frequency generator is a simple phase-accumulator type digital oscilator. It works simply by taking the contents of the frequency register, and adding it to the phase accumulator. The phase accumulator is updated at the system sample rate. To calculate the frequency use the following formula:

$$\mathbf{F = F_S * F_n / 2^{24}}$$

Where F is the frequency, $F_S$ is the sample rate and $F_n$ is the value in the frequency register.

There is also a phase offset register BX97 which is added to the outut of the phase accumulator so that we can get various phase shifts. The most common use is to get a 90

degree phase shift to generate the quadrature sine waves for the spectrum analyzer (see above).

It should be noted that the output of the phase accumulator is phase. Strictly speaking, this is a saw tooth wave. To generate sine waves, the DSP takes the phase output and uses that as an index into a look up table. The DSP56002 contains a 256 word sine lookup table in ROM.

There is also logic in the frequency generator so that the phase accumulator can be syncronized to the spindle motor. This feature is handy for determining how much of the noise on the following error is due to the unballance in the spindle motor. It can also be used to track ecentric tracks on the disk.

## Real Time Clock

Sheet 9 of the shcematic

The real time clock is kind of a hodge podge...this circuit dates back to the very original design that was on the Teletrac 600-060 card, and it hasn't changed a whole lot since.

BX26 scales the clock rate down. Its output then leaves the sheet as signal DAC_SHIFT. This signal goes to the DAC INTERFACE (see below). It gets scaled some more on that sheet for the amount of time it takes to shift out the data for the four DACs. The signal is the returned as RTC_TIME where it sets U80 which send an interrupt back to the DSP chip. RTC_TIME is also used to increment other counters that scale the real time clock further for sample signals for the LINEAR and SPINDLE MOTOR servo loops (note...there really is no long a linear servo loop).

## Interrupt Hardware

Sheet 14, 15 and 16 of the Schematic

The interrupt hardware is fairly simple. And, also, mostly unused these days. Again, this all dates back to the original servo card, the 600-060. It is basically just a collection of flip flops that latch external signals that can generate interrupts. The only inputs that are really used any longer are the Linear **LIM+** and **LIM-** signals which are used to let the spindle controller know when the linear stage is in position.

## DAC Interface Hardware

Sheet 10 of the Schematic

The DAC interface has four data registers, BX8, BX10, BX15 and BX16 where the data that is to go to the DACs is stored. When the DAC_SHIFT signal goes true, it starts a state machine that starts outputing the data. The data goes through a MUX and the selected register is loaded into shift register BX13 and then shifted out to the DACs.

BX14 keeps track of how many shifts have occured. U54 keeps track of which DAC is being loaded.

## Timer

Sheet 17 of the schematic

This is a very simple page. This is a timer that gets decremented every time the sample pulse occures. This was done to get a little more bandwidth in the DSP. Even though the DSP can do this job very quickly, it does it on every interrupt cycle and therefore does put a load on the system. The DSP can write a value into the counter and then read it until it goes to zero.

## Spindle Motor Inteface

Sheet 2 and 3 of the schematic

This is the interface to the spindle motor encoder. It is a very basic phase detector (sheet 3) and a position counter (sheet 2).

# Tiny Basic

The original intent of the Tiny Basic interpreter was to provide a means by which the user could create custom drivers for doing the RESET, START (load), STOP (unload), LIFT and RELOAD functions of the spinstand. Because of the fact that there was not enough room in the ROM on the 600-160 REV A PC board, this feature was removed and these functions were hard coded. And this is where things stand as of Version 1.70.10.

However, the REV B board has double the rom space, so even though the interpreter as such has no real use, it has been added back into the code. The current status of the Tiny Basic interpreter is that it functions, but it is useless since it has no language structure to interface to the various services in the spindle controller. For example, Tiny Basic cannot talk to the Motion Controller or start and stop the spindle.

These function can, however, be easily added to the language. It is just a matter of adding them to the grammar for the language and writing the action functions that will execute them. See **Tiny.syn** below.

**Implementation**

Tiny Basic is implemented using the Anagram parser generator. There are two sections to the code. The Grammar definition is at the top of the file. This defines the syntax that will be used in the tiny basic implementation. The bottom portion of the code are the reduction functions that execute the code. For more information about Anagram, you can visit their website: http://www.parsifal.com.

**Syntax**

This Tiny Basic implantation is not exactly like real Basic. Several modifications have been made to make the implementation of the language easier, as well as add some of the functionality that is needed to operate in the Spindle Controller.

**Data Types**

There are three data types that can be declared. There is a **long**, which is also the default type, which is 32 bits. There is also a **double**, which is a 64 bit floating point number. And there is a **string**, which is a value that can store a character string.

Using Tiny Basic

The Tiny Basic implementation takes full advantage of the fact that it is running in a multitasking environment. It is possible to have many threads running simultaneously, as long as there is plenty of random access memory. If you look in the make file for the firmware, you will notice that one of the command line options for the compiler is **–rA5=_data**. What this does is it makes all of global data for Tiny Basic accessible relative to address register A5. If you were to look in the SDS documentation,

you will find that they do not recommend mixing module that are compiled this way with ones that are compiled for absolute addressing. In fact, they say don't do it. However, I laugh in the face of danger. As long as all of the other modules do not access any variable declared in **tiny.syn**, or, **tiny.syn** does not access any globals outside of its module, there will be no problems. If you do, the linker will generate an error saying something to the effect that an address was out of range.

To start a tiny basic thread up, you need to call CreateRelTask. This function will allocate the memory needed for the global variables and pass a pointer to this area in register A5 when it starts up the tiny basic task. To get the sizes of global data areas in tiny basic, you need to call **RelRamSize** and **RelDataSize** (see region.s below). You will also need to create a memory space in which to put the basic code itself. You can see some examples of how this is done in **conrevb.syn**.

**Spindle Controller Errors**
or
**What to do when things go wrong**

The Accutrac Spindle Controller attempts to detect any error condition that can occure. This of course, in a very difficult task to do and there are probably conditions that cannot be detected. However, you will probably find that the errors that it does detect can be very annoying. The spindle controller, along with the spinstand, must be in 100% working order in order for the system to function.

Errors can come from two different places. The spindle controller itself can generate errors and the host interface software can also generate errors. Seperating these out may sometimes be difficult. With a little detective work, generally the source can be determined.

On the host computer side the main file is spindll.dll. This library has the interface routines that interfaces between the host program and the spindle controller. The Error handling is taken care of in the module timer.cpp. The function GenericError is used to process the errors recieved from the function and an attempt is made to translate this error number into something more or less meaningful. Unless you are familiar with the errors, however, this will be an extremely difficult task.

The main error that you will see that occures from the host side will be a command timeout error. This will generally occure because the spindle controller has become disconnected from the host somehow. So, we will first concentrate on spindle controller errors and what can cause them. The errors will be discussed in order of the value (each error has a negative numeric value).

On the firmware (spindle controller) side of things the file that contains the defines for the errors is Erorcode.h. On the host side (spinstan.exe,spindll.dll), the file that contains the defines for the errors is protocol.h. The differences between these files as fare as the error codes go are insignificant. One thing that should be noted is that the value of 0 (zero) indicates always that no error has occured.

**Multitasking Errors**

```
#define EVENT_TIMEOUT  -96              //event has timed out
```

This error occures when a **semaphore** has timed out. The text string that is generated even uses the word semaphore. This is a very broad error, and is one of the more common ones that you will see occure. Semaphores are discussed elsewhere in this document but we will discuss them as they concern timeouts.

A **semaphore** is a software object that basically counts events. They are used to suspend execution of a task when the task must wait for some external event to occure. In this way, other tasks can execute making for a much more efficient use of hardware

resources. When the event occures, the semaphore is posted, the suspended task is put back into the execution queue, and the task resumes its execution.

**Semaphores** have a built in timeout mechanism that is used for some events. A good example is when we turn on the spindle motor, we pend on a semphore that will be posted when the spindle motor comes up to speed. If the spindle motor never gets up to speed, the firmware would appear to be hung up since that task would never execute again. With the timeout, the firmware can set the amount of time allowed for the semaphore to be posted. If the semaphore is posted by the real time clock interrupt, this is the error value that is returned.

Common timeout errors in the spindle spindle controller include the following:

**Spindle Motor**. The spindle motor can cause a timeout error if it either does not start spinning or if it does not spin up fast enough. Not spinning at all can be caused either by a hardware problem with the motor drive circuitry which can range from broken wires to setup problems. Not spinning up fast enough can be caused by having the acceleration set to high. 5000rpm/sec is about the maximum that the spindle can accelerate. If there are a lot of disks on the spindle, 2000rpm/sec is the max. Accelerating too slow can also be due to the spindle parameters not being correctly set up.

**Ramp Loader.** The ramp loader can cause a timeout if it is moving too slowly. The semaphore for the ramps is hard coded for about 1 second. The speed for the ramp can be adjusted with the air valves up next to the ramp. Timeout errors can also be caused if the sensors are backwards. If the loaded sensor is in the unloaded position, it will not cause the correct semphore to fire. Another reason the semaphore might not fire is if the ramp loader sensors have not been properly set up.

**Linear Stage.** The linear stage can also cause a timeout if it is moving too slowly. Although, the timeout for the linear stage is hard coded at several seconds. Common causes for timeouts also include incorrect positioning of the optical interrupters, broken wires going to the interrupters, or improper setup of the DSP Linear Stage Interrupts. If you run spinstan.exe and pull down the **LINEAR | INTERRUPTS** menu be sure that for the **LIM+ and LIM-** interrupts that the **Run/Home** radio button is checked.

**Connector Clamp.** The connector clamp is very similar to the ramp loader and will have the same problems.

These are the most common errors to cause timeouts, however, there are others.

```
#define EVENT_OVERFLOW        -97             //too many events pending
```

Chances are you will never see this error. This error will occure when the semphore count has exceeded its maximum value. If this error ever occures, it should be considered a serious error.

#define EVENT_BUFFFULL -98 //pipeline buffer full

This is an error that can occure in an EVENT_QUEUE object. It will occure if the queue is full. This is an error that should never be seen. If this error occures, it is a serious problem.

#define EVENT_NOTASKS -99 //no tasks to reschedule to

This is an error that occures when an attempt was made to schedule a task but no tasks were found in the priority queue. This should never happen. There should always be an **Idle** task in the priority queue. If this error occures, it is a serious problem.

#define EVENT_TERMINATE -100 //task needs to be canceled

This is an error that is returned to a task to let the task know that it should kill itself. In theory, this error should never be reported back to the host computer. But, you never know. If the host computer ever sees this error, it should be considered a problem.

#define EVENT_ABORTED -101 //event was aborted by another task

This is an error that is sent to a task to indicate to the task that the event that was pending should be aborted. One of the places where this is used is for the spindle motor. If the user pushes the **STOP** button while the motor is spinning up, we need to abort the action and shut down the motor. This error, as far as I know, will in general not be reported back to the host, but, I could be wrong.

## Motion Control Errors

#define SERVO_GOTTIMEOUT -112 //TIMEOUT

This is an error indicating that there was a timeout between the spindle controller host processor (68EC000) and the DSP (56002). This is a very rare error. If it occures it probably indicates that there is a serious hardware or software problem with the motion controller.

#define SERVO_GOTNAK -113 //NAKked

This is an error indicating that the Motion Controller was unable to complete the indicated command. An example of this would be sending a command to clear the **LIM-** hardware flag while the limit switch is blocked. The Motion Controller will be unable to do this, so it will return a NAK error.

#define SERVO_GOTACKTIMEOUT -114 //timed out waiting for ACK

This is an error indicating that for the most part every thing was working fine, except the final "ACK" cycle never occured. This should be an extremely rare error. If it does occure, it could indicate a serious software problem.

#define SERVO_ERROR -115

This error is returned if the command attempted to execute but some error is detected in the process. As far as I know, this error code is not supported.

#define SERVO_UNKOWN_ERROR -116 // ???

This error should never occure. If it does, it indicates there is a serious problem somewhere.

**Spindle Controller Command Errors**

#define SPINDLE_ERROR -128 //some sort of error

This error indicates that some sort of error occured while a spindle command was being executed.

#define SPINDLE_NAK -129 //command refused

This error indicates that the spindle controller could not execute the command.

**Protocol Errors**

#define TEL_PROT_TIMEOUT -136

This error indicates that while waiting for characters to come in a timeout occured.

#define TEL_PROT_CHECKSUM -137

Pretty self explanitory. This error indicates that there was a checksum error in a data packet.

#define TEL_PROT_BADHEADER -138

When the first character in a packet has an unexpected value, you will get this error. The first character in a packet is always the value 0x80.

Ramdisk Errors

#define RAMDISK_FILENOTFOUND -64

This error is pretty self explanitory. A ramdisk command was attempting to execute an operation on a specific file and it could not find it. This error comes up most often when attempting to initialize the spinstand. During the initialization, a name of an **.afg** file is passed and if the system cannot locate that file, you will get this error. File names in the ramdisk are case sensitive. The slightest difference will cause an error.

#define RAMDISK_NOSECTORS -65

This error is indicating that there are no free secotors in the ram disk. This error should be extremely rare. The ram disk consists of 64K bytes of ram, which while not really very much, is more than pleanty to do the job. If this error occures, the operator will need to go in and clean up the ramdisk.

#define RAMDISK_FILEOPEN -66

This error is unsupported. You should never see this error occure.

#define RAMDISK_BADNAME -67

This error indicates that the file name is badly formed. This should be very rare in normal operation.

#define RAMDISK_NODIRECTORIES -68

The current status of this error is unknown. (7-2-2002). It is in the code, but it's placement seems to be meaningless. So for the time being, if this error occures, it indicates that a serious problem has occured.

#define RAMDISK_EOF -69

Not really an error, but it does indicate the status that the end of the file has been reached. This error will occure when reading files. Most likely, this will never be reported back to the host unless you are using one of the ramdisk read functions from the host.

**Error Codes for Miscilaneous Functions**

#define GENERAL_NAK -80

This error indicates that the command was refused. Most likely, it was an invalid command code.

#define GENERAL_ERROR -81

This error code indicates that an error occured while processing the command.

**Error codes for the High Level Functions**

#define ACUTRAC_ERROR -1 //you get this when there is some error

Like the comment says, you get this code when there is some sort of an error. An example, when the PivotToGap parameter is equal to zero, you will get this error (because it would cause a divide by zero exception if it tried). Another example, if you command the heads to load, and then command them to load again, you will get this error on the second try (heads must be unloaded before they load).

#define ACUTRAC_BUSYERROR -2

This error occures if you try to spawn too many tasks. The tasks that can be spawned are RESET, START (load), STOP (unload), LIFT, LOWER. Only one of these can run at any one time. This error should never be seen. If it occures, a host computer routine is not properly checking the status of spawned tasks.

#define ACUTRAC_CONFIGFILE -3

You get this error whenever there is a problem with the **.afg** file that is used for either INIT or RESET. The most common problem is a syntax error. It is posible to get the error message that the parser generated using other calls.

#define ACUTRAC_SERVO_NAK -4

This error, very similar to the SERVO_GOTNAK error above, is reported back when ever any of the high level functions get a NAK from the motion controller. This should be a rare error.

#define ACUTRAC_SERVO_TIMEOUT -5

This error occures when the high level functions get a timeout from the motion controller. This error should never occure. If this error occures it indicates a very serious problem.

#define ACUTRAC_SERVO_UNKNOWN -6

This error occures when the motion controller communication interface cannot figure out what is wrong. This error should never occure. If it does, it indicates that a very serious problem is present.

#define ACUTRAC_OUTOFRANGE -7

You get this error when some parameter is out of range. Most common, the system was told to seek to a track number that is greater than MaxTrack or less than Zero (there is some leaway at both ends for this example of about 100 tracks).

#define ACUTRAC_FILEERROR -8

You get this error when the system has a problem with the **.afg** config file. Again, most likely a syntax error.

#define ACUTRAC_FILEBAD -9

You get this error when the system cannot find the **.afg** file that was specified. Remember, the names are case sensitive and must be exact.

#define ACUTRAC_RAMPTIMEOUT -10

You get this error when there was some problem with the ramp lifters. There are generally two posibilities. Either the ramps did not move at all or fast enough, or, they are not connected up correctly. Generally for systems that have been up for a while, it is the former, for new systems, generally, it is the latter.

#define ACUTRAC_LINEARTIMEOUT -11

You get this error when there is a problem with the linear stage. There are several posibilities here. The most obvious is that the stage is either not moving or moving too slow. Another posibility is that optical interrupter is not functioning either by failure or not being adjusted correctly. A third posibility is the firmware is not set up correctly. Setting up the interrupts for the Linear Stage is described above.

#define ACUTRAC_MOTIONTIMEOUT -12

You get this error when there is a problem with moving the rotary actuator. It is posible that it is not moving at all, although, this is rare. If the rotary actuator is servoing, it will move. Another posibility is that something is causing the rotary actuator to not get to position. Excessive friction in the bearings can cause this, or a mechanical obstruction. When the servo is off, the rotary actuator must move freely. Another posibility is that there is insufficiant integrator gain, or the integrator is turned off. This could prevent the actuator from getting to position.

#define ACUTRAC_SPINDLETIMEOUT -13

You get this error when the spindle motor fails to come up to speed. There are many posible causes of this (that are discussed above). Most common is the acceleration is set to fast.

#define ACUTRAC_USERABORT -14

You get this error when you push the **STOP** button while an operation is underway.

#define ACUTRAC_TIMEOUT -15

You get this error when a **semaphore** generates a timeout. **Semaphore** timeouts are discussed at lenth above.

#define ACUTRAC_TERMINATE -16

You get this error when a task recieves a terminate command. This error should not be reported back to the host so most likely will never be seen.

#define ACUTRAC_NOTHANDLED -17

While not a serious problem, you will get this error when a problem handler that has not yet been implemented is encountered. Hopefully, this error will never be seen.

#define ACUTRAC_LOWAIR -18

This error indicates that an operation was attempted that required air pressure and the air pressure was low. This error can be eliminated only by restoring the air pressure. If there is air pressure and this error occures, one posibility is either an incorrectly connected air pressure sensor, or a faulty air pressure sensor.

#define ACUTRAC_NOINITINFO -19

This error occures when an operation is attempted that requires the information that would be in a **.afg** file. Hopefully, this will be an extremely rare error.

#define ACUTRAC_AMPFAULT -20

This error indicates that the spindle amplifier has faulted, most likely due to overtemperature. The most likely cause for this is either the air vents are blocked (never ever block the air vents, the amplifier disipates 600Watts and needs that air flow), or the bias is not correctly adjusted on the amplifier causing excessive idle current.

#define ACUTRAC_CLAMPTIMEOUT -21

You get this timeout when the connector clamp doesn't move or reach the correct destination. The posible causes for this could be insufficient air presure to the clamp actuator. This can be remidied by adjusting the air valves for actuator. Another posibility is that the sensors or the actuator are connected backwards so that up is down and down is up.

## Central I/O Error Codes

These are the error codes for the I/O manager.

```
#define CIO_ABORT           -144    /*      device IO was interrupted by user      */
```

This error code is unsupported. You should never see this error.

```
#define CIO_NO_HANDLES -145     /*      NO MORE IOCB'S AVAILIABLE    */
```

This error indicates that the system has run out of IO Control Blocks (IOCBs). This is a serious problem. It indicates some sort of resource leak in the firmware. This error should never occure.

```
#define CIO_NO_DEVICE     -146    /*      could not locate that device    */
```

This error indicates that an attempt was made to open a device that does not exist. This error should never occure. This is a very serious error.

```
#define CIO_WRONLY          -148    /* Read was attempted on write only */
```

This error indicates that attempt was made to read a device that was opened for write only. This error should be extremely rare. It is a serious error if it occures.

```
#define CIO_INVALID         -149  /* INVALID FUNCTION NUMBER   */
```

An invalid function number was passed to the CIO function. This should never happen. This is a serious problem.

```
#define CIO_NOT_OPEN      -150    /*      IOCB NOT OPEN      */
```

An attempt was made to use an IOCB that was never opened to a device. This is a serious problem.

```
#define CIO_INVLD_HNDL  -151    /*      INVALID HANDLE  */
```

An invalid handle was passed to the CIO routine. This is a serious problem.

```
#define CIO_RDONLY          -152    /*      Write was attempted on read only      */
```

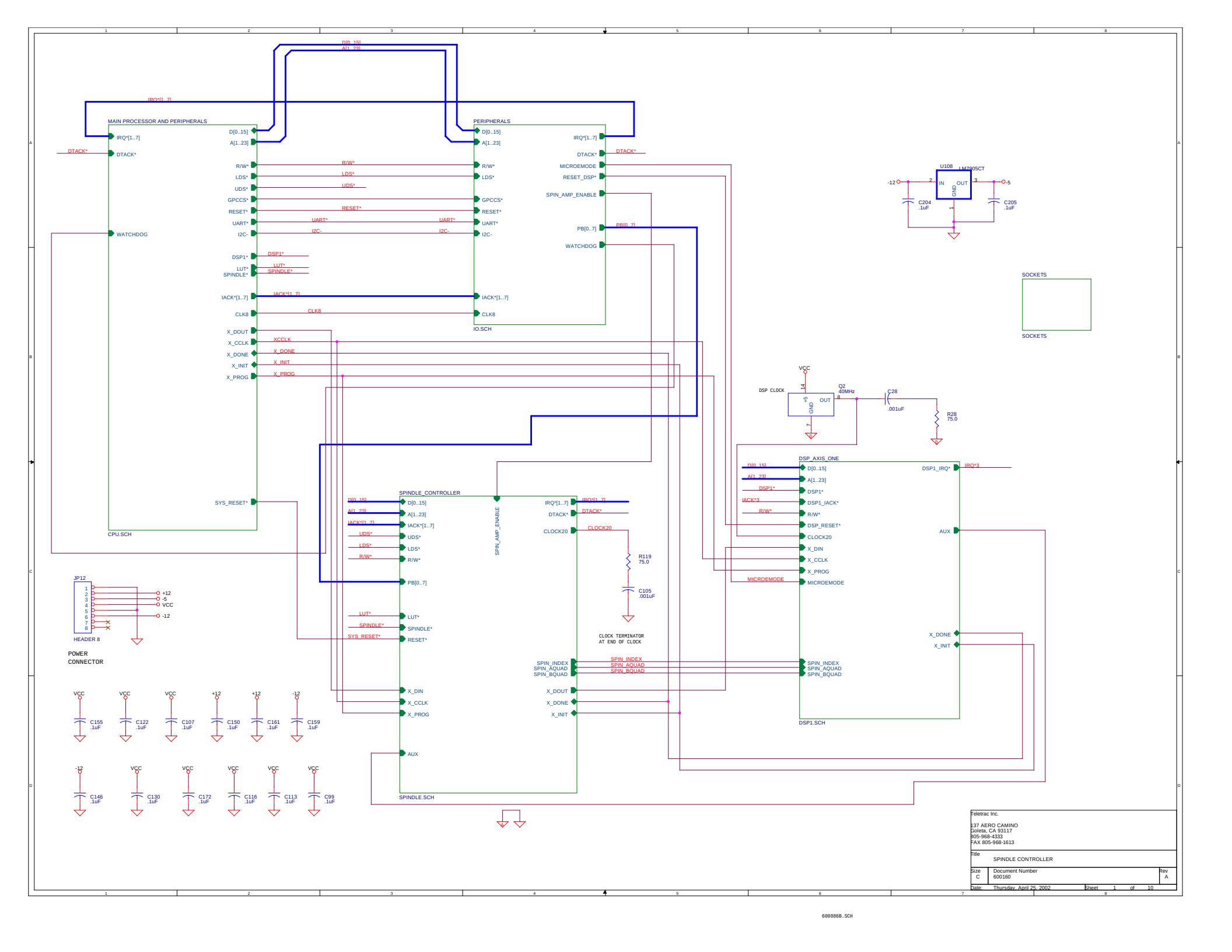
An attempt was made to write to a device that was opened for read only. This is a serious problem.

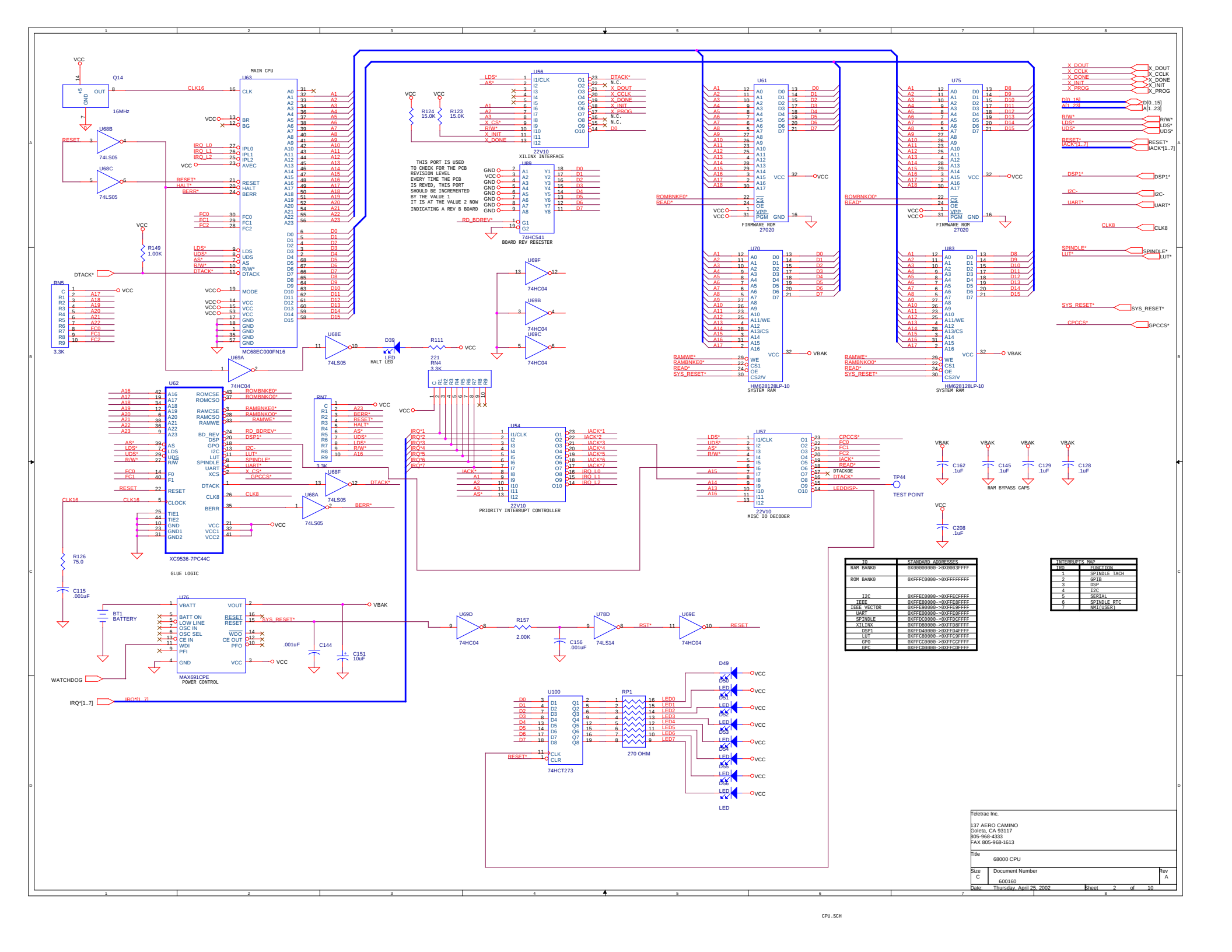
```
#define CIO_END_OF_FILE  -153    /*      end of file encountered          */
```

This error that the end of a file was encountered while reading. This error is generally not reported back to the host and will probably never be seen.

#define CIO_TRUNCATED -154

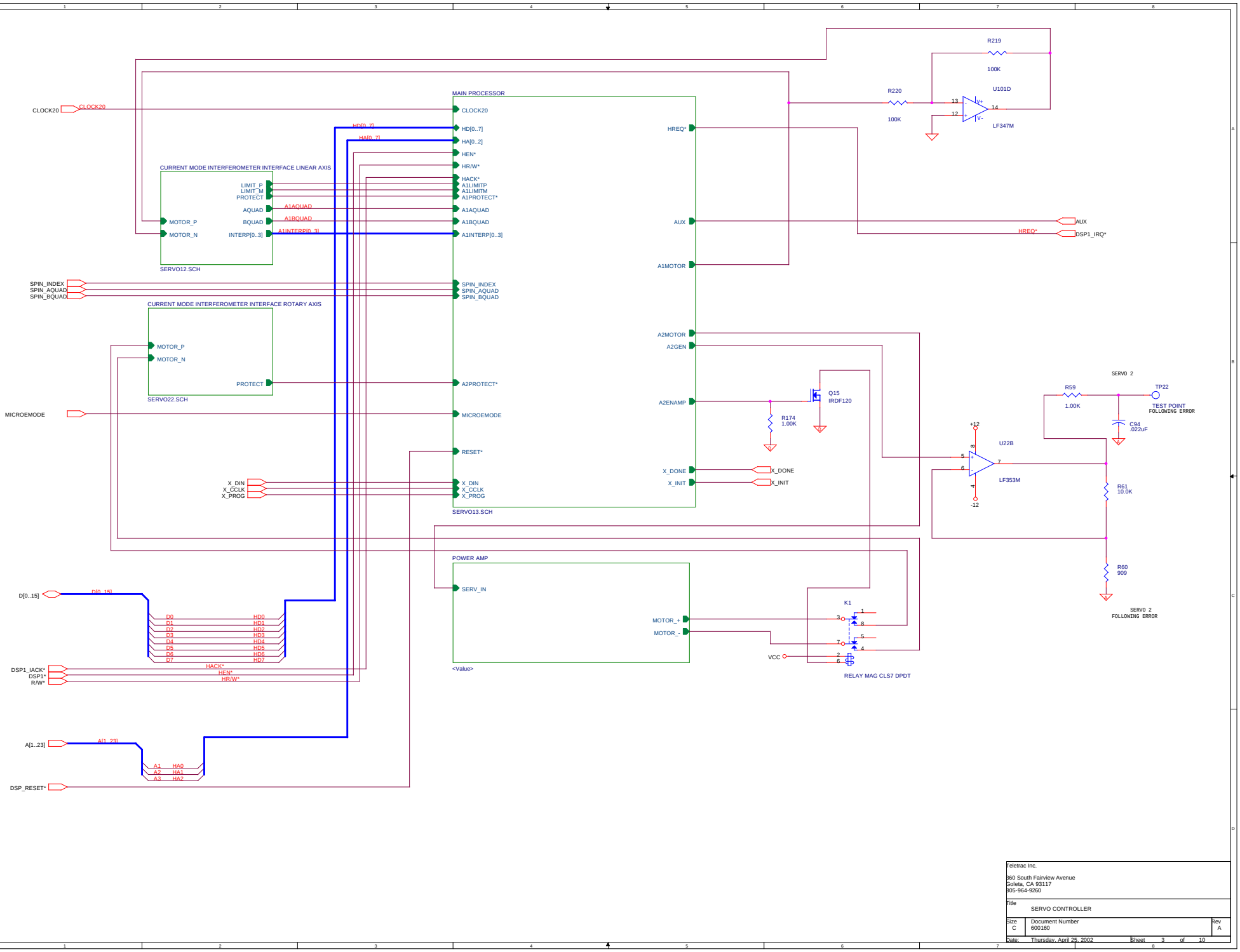
This error code is not supported.


MAIN PROCESSOR AND PERIPHERALS
CPU.SCH
PERIPHERALS
IO.SCH
SPINDLE_CONTROLLER
SPINDLE.SCH
DSP_AXIS_ONE
DSP1.SCH
SOCKETS
SOCKETS
U108
LM7905CT
C204 .1uF
C205 .1uF
Q2 40MHz
C28 .001uF
R28 75.0
R119 75.0
C105 .001uF
CLOCK TERMINATOR AT END OF CLOCK
JP12
HEADER 8
POWER CONNECTOR
C155 .1uF
C122 .1uF
C107 .1uF
C150 .1uF
C161 .1uF
C159 .1uF
C146 .1uF
C130 .1uF
C172 .1uF
C116 .1uF
C113 .1uF
C99 .1uF
Teletrac Inc.
137 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613
Title
SPINDLE CONTROLLER
Size C
Document Number 600160
Rev A
Date: Thursday, April 25, 2002
Sheet 1 of 10
600086B.SCH

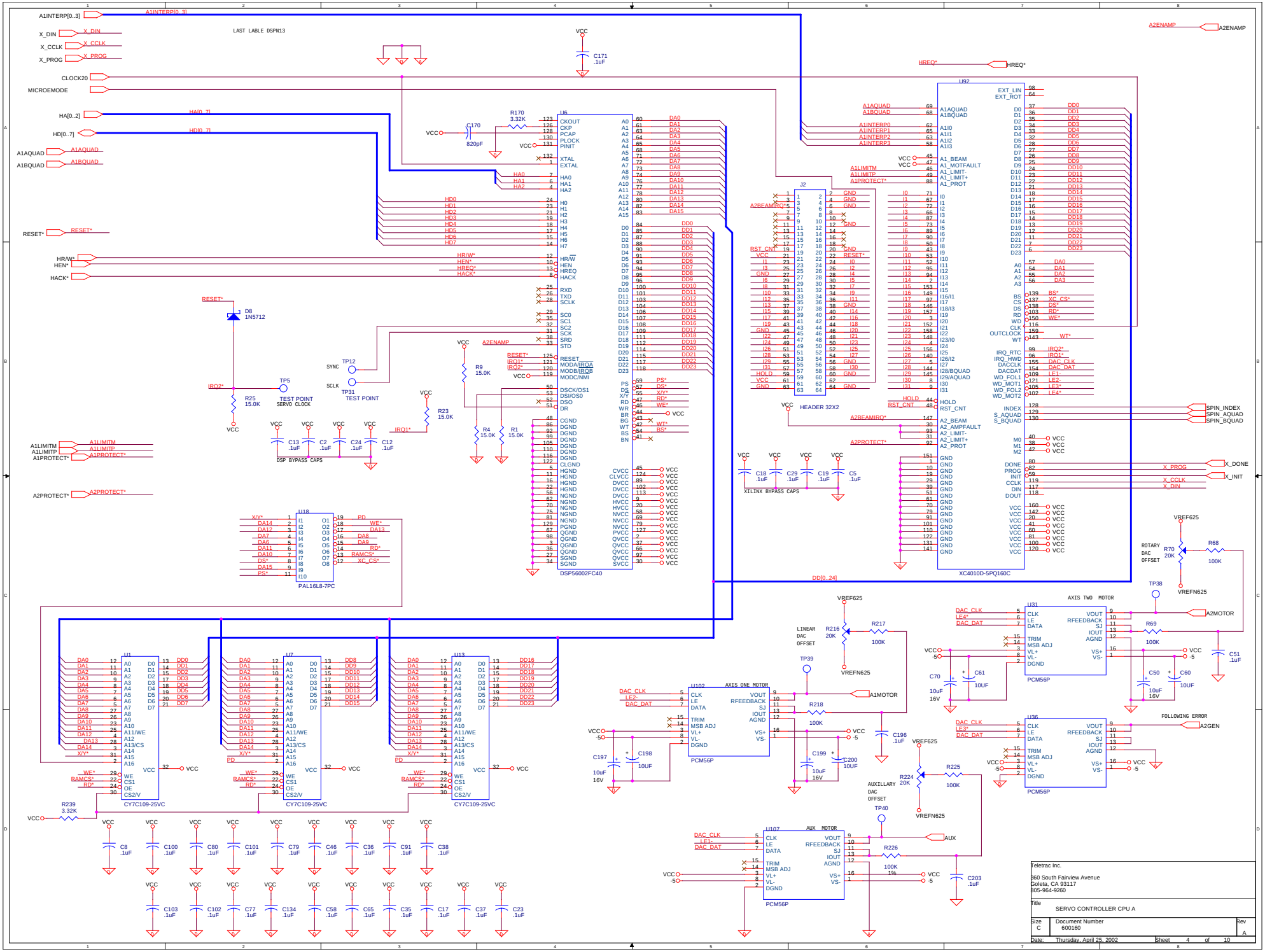



| IO | STANDARD ADDRESSES |
|---|---|
| RAM BANK0 | 0X00000000->0X0003FFFF |
| ROM BANK0 | 0XFFFC0000->0XFFFFFFFF |
| I2C | 0XFFEC0000->0XFFECFFFF |
| IEEE | 0XFFED0000->0XFFEDFFFF |
| IEEE VECTOR | 0XFFE90000->0XFFE9FFFF |
| UART | 0XFFE00000->0XFFE0FFFF |
| SPINDLE | 0XFFDC0000->0XFFDCFFFF |
| XILINX | 0XFFD80000->0XFFD8FFFF |
| DSP1 | 0XFFD40000->0XFFD4FFFF |
| LUT | 0XFFC80000->0XFFC9FFFF |
| GPO | 0XFFCC0000->0XFFCCFFFF |
| GPC | 0XFFCD0000->0XFFCDFFFF |

| INTERRUPTS MAP | |
|---|---|
| IRQ | FUNCTION |
| 1 | SPINDLE TACH |
| 2 | GPIB |
| 3 | DSP |
| 4 | I2C |
| 5 | SERIAL |
| 6 | SPINDLE RTC |
| 7 | NMI(USER) |




| | | |
|---|---|---|
| Teletrac Inc. | | |
| 360 South Fairview Avenue<br>Goleta, CA 93117<br>805-964-9260 | | |
| Title | SERVO CONTROLLER | |
| Size C | Document Number 600160 | Rev A |
| Date: Thursday, April 25, 2002 | Sheet 3 of 10 | |

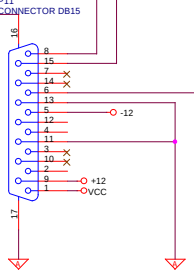

A1INTERP[0..3]
X_DIN
X_CCLK
X_PROG
LAST LABLE DSPN13
CLOCK20
MICROMODE
HA[0..2]
HD[0..7]
A1AQUAD
A1BQUAD
RESET*
HR/W*
HEN*
HACK*
A2ENAMP
HREQ*
C171 .1uF
R170 3.32K
C170 820pF
U6
DSP56002FC40
CKOUT
CKP
PCAP
PLOCK
PINIT
XTAL
EXTAL
HA0
HA1
HA2
HD0
HD1
HD2
HD3
HD4
HD5
HD6
HD7
HR/W
HEN
HREQ
HACK
RXD
TXD
SCLK
SC0
SC1
SC2
SCK
SRD
STD
RESET
MODA/IRQA
MODB/IRQB
MODC/NMI
DSCK/OS1
DSI/OS0
DSO
DR
CGND
DGND
CLGND
HGND
NGND
PGND
QGND
SGND
CVCC
CLVCC
DVCC
HVCC
NVCC
PVCC
QVCC
SVCC
D8 1N5712
TP12
SYNC
TP31
SCLK
TEST POINT
TP5
SERVO CLOCK
TEST POINT
R25 15.0K
R9 15.0K
R23 15.0K
R4 15.0K
R1 15.0K
C13 .1uF
C2 .1uF
C24 .1uF
C12 .1uF
DSP BYPASS CAPS
A1LIMITM
A1LIMITP
A1PROTECT*
A2PROTECT*
U18
PAL16L8-7PC
U92
XC4010D-5PQ160C
EXT_LIN
EXT_ROT
A1_BEAM
A1_MOTFAULT
A1_LIMIT-
A1_LIMIT+
A1_PROT
A1LIMITM
A1LIMITP
A1PROTECT*
A2BEAMIRQ*
RST_CNT
HOLD
HEADER 32X2
J2
A2_BEAM
A2_AMPFAULT
A2_LIMIT-
A2_LIMIT+
A2_PROT
A2BEAMIRQ*
A2PROTECT*
C18 .1uF
C29 .1uF
C19 .1uF
C5 .1uF
XILINX BYPASS CAPS
INDEX
S_AQUAD
S_BQUAD
SPIN_INDEX
SPIN_AQUAD
SPIN_BQUAD
DONE
PROG
INIT
CCLK
DIN
DOUT
X_DONE
X_INIT
X_PROG
X_CCLK
X_DIN
IRQ_RTC
IRQ_HWD
DACCLK
DACDAT
WD_FOL1
WD_MOT1
WD_FOL2
WD_MOT2
BS*
XC_CS*
DS*
RD*
WE*
LE1-
LE2-
LE3*
LE4*
DD[0..24]
U1
U7
U13
CY7C109-25VC
WE*
RAMCS*
RD*
R239 3.32K
C8 .1uF
C100 .1uF
C80 .1uF
C101 .1uF
C79 .1uF
C46 .1uF
C36 .1uF
C91 .1uF
C38 .1uF
C103 .1uF
C102 .1uF
C77 .1uF
C134 .1uF
C58 .1uF
C65 .1uF
C35 .1uF
C17 .1uF
C37 .1uF
C23 .1uF
VREF625
LINEAR DAC OFFSET
R216 20K
R217 100K
TP39
VREFN625
U102
AXIS ONE MOTOR
PCM56P
A1MOTOR
R218 100K
C196 .1uF
C197 10UF 16V
C198 10UF
C199 10uF 16V
C200 10UF
AUXILLARY DAC OFFSET
R224 20K
R225 100K
TP40
U107
AUX MOTOR
AUX
R226 100K 1%
C203 .1uF
U31
AXIS TWO MOTOR
A2MOTOR
R69 100K
C61 10UF
C70 10uF 16V
C50 10uF 16V
C60 10UF
C51 .1uF
U36
FOLLOWING ERROR
A2GEN
ROTARY DAC OFFSET
R70 20K
R68 100K
TP38
VREFN625
SERV013.SCH
Teletrac Inc.
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260
SERVO CONTROLLER CPU A
Size C
Document Number 600160
Rev A
Date: Thursday, April 25, 2002
Sheet 4 of 10

TP1
TEST POINT
GROUND

MOTOR_P

MOTOR_N

P11
CONNECTOR DB15




R206
10.0K

PROTECT

C185
.01uF

VCC

R227
220

R228
330

PROTECT
PROTECT

| | | |
|---|---|---|
| Teletrac<br>137 AREO CAMINO<br>GOLETA CA 93117 | | |
| Title: PROCESSING ELECTRONICS ROTARY AXIS | | |
| Size: C | Document Number: 600160 | Rev: B |
| Date: Friday, April 26, 2002 | Sheet 5 of 10 | |

LAST NODE LABLE CM50

TP30
TEST POINT
GROUND

VCC
D57 1N914
D58 1N914
D59 1N914
D60 1N914
D61 1N914
D62 1N914
D63 1N914
D64 1N914
INTERP0
INTERP1
INTERP2
INTERP3

LIMIT_P
LIMIT_M
PROTECT

INTERP[0..3]

MOTOR_P
MOTOR_N
R221100
R222
100

P10
MALE DB15
CONNECTOR DB15M

INPUT CONDITIONING

-12

R203 10.0K
R204 10.0K
R205 10.0K

LIMIT_P
LIMIT_M
PROTECT

C182 .01uF
C183 .01uF
C184 .01uF

R229 220
R230 330

R12 51.1
R10 51.1

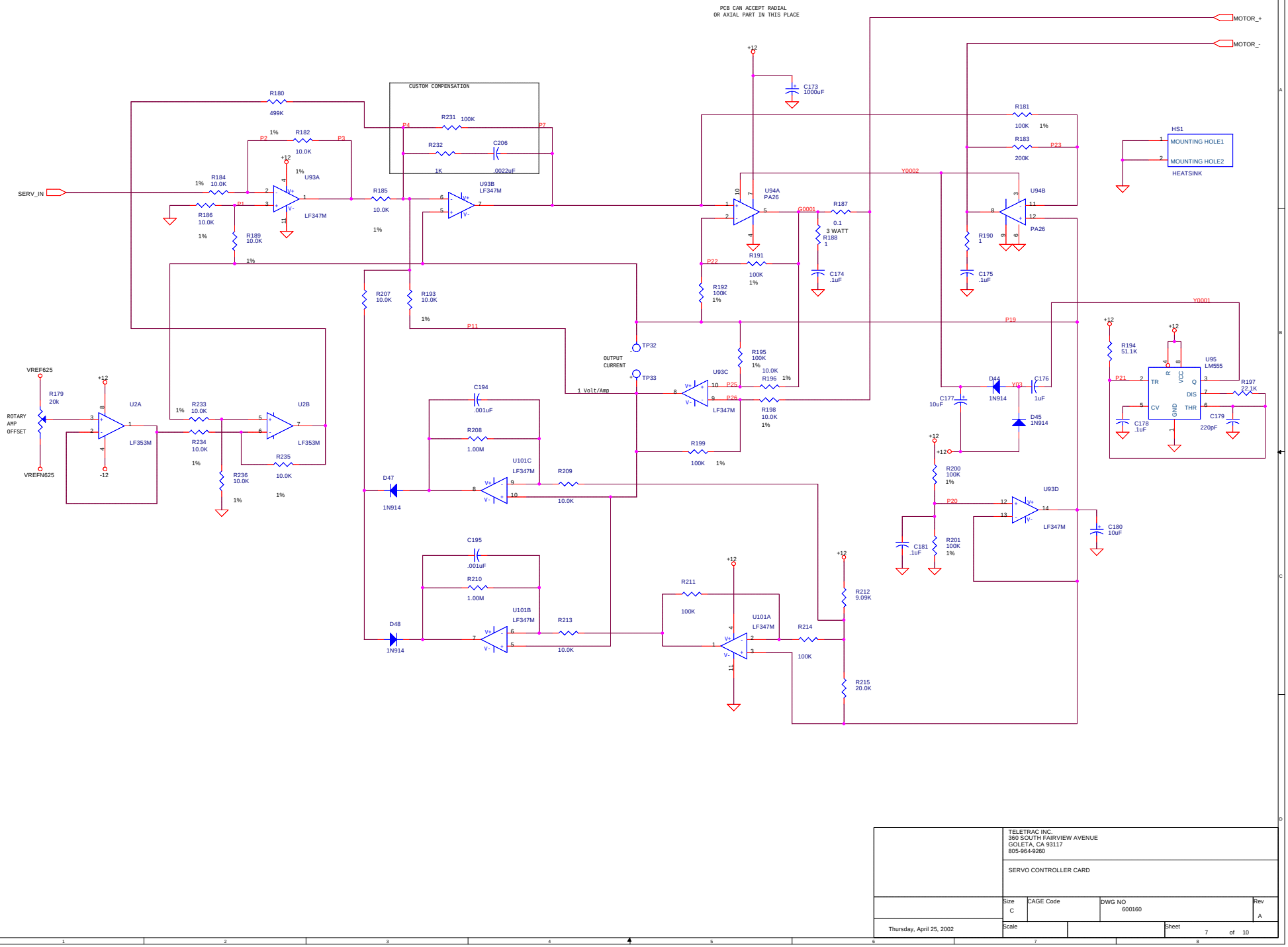
D1 1N914
D2 1N914
D3 1N914
D4 1N914

AQUAD
BQUAD

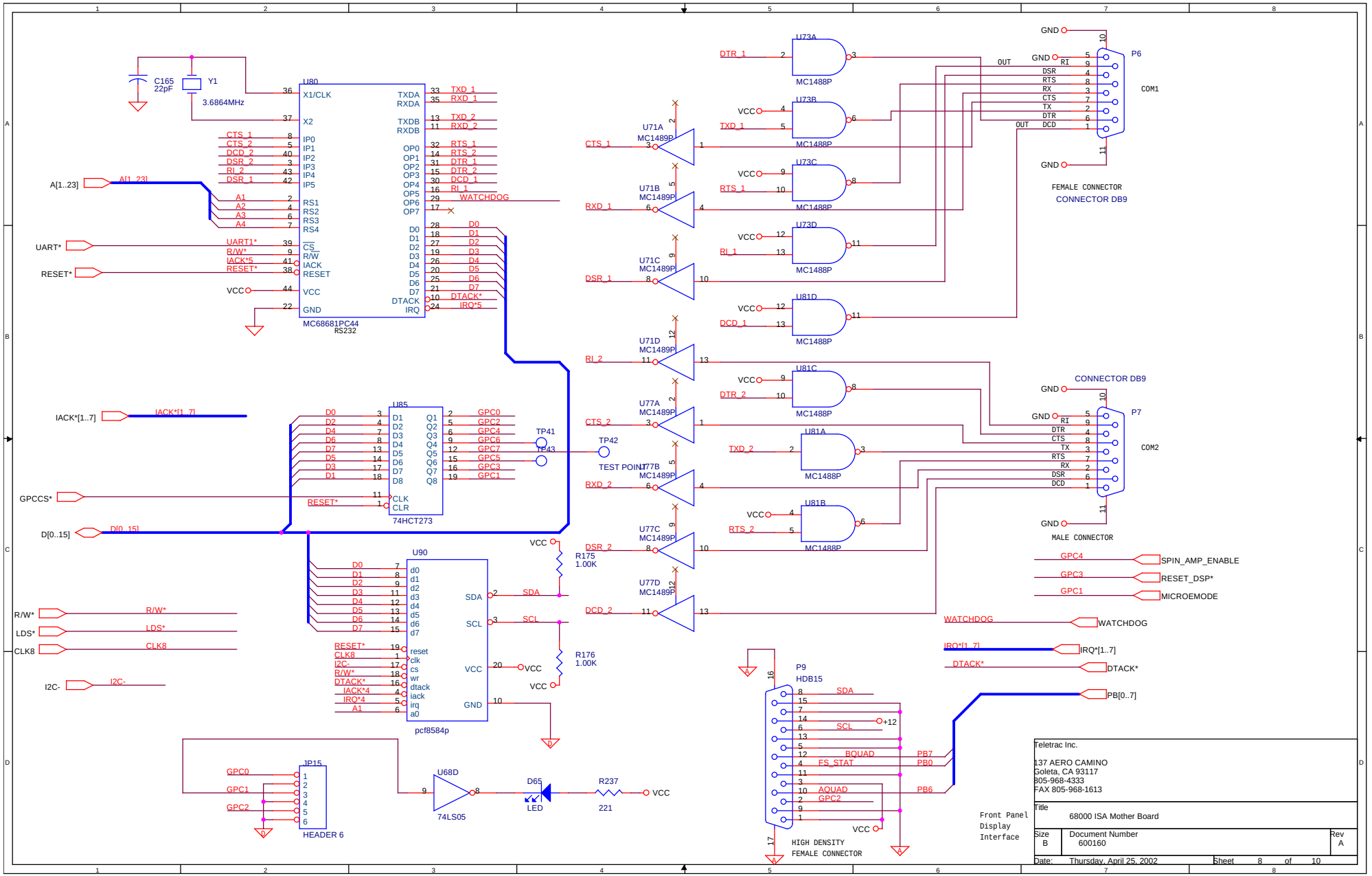
TP2
AQUAD
TEST POINT

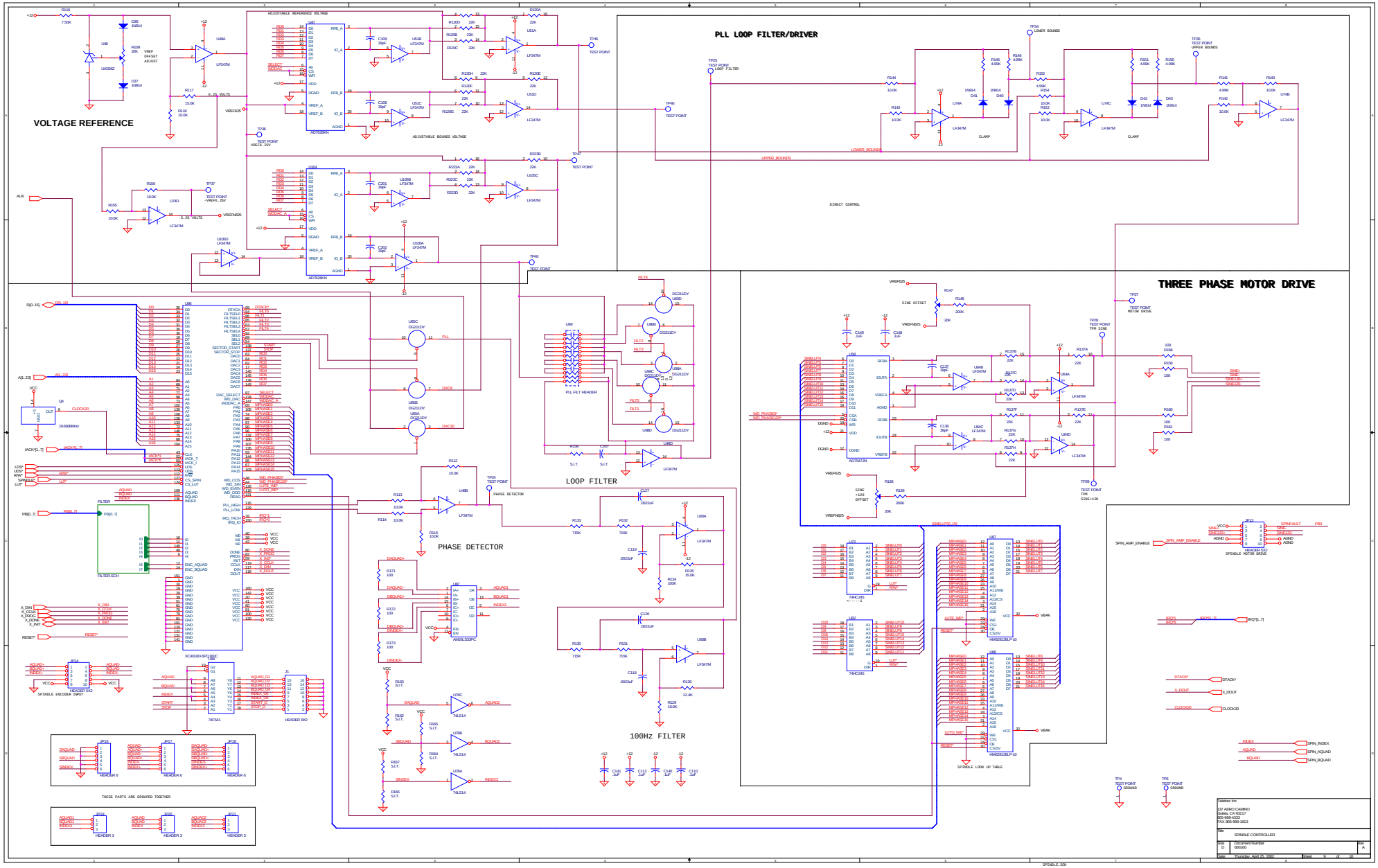
TP3
BQUAD
TEST POINT

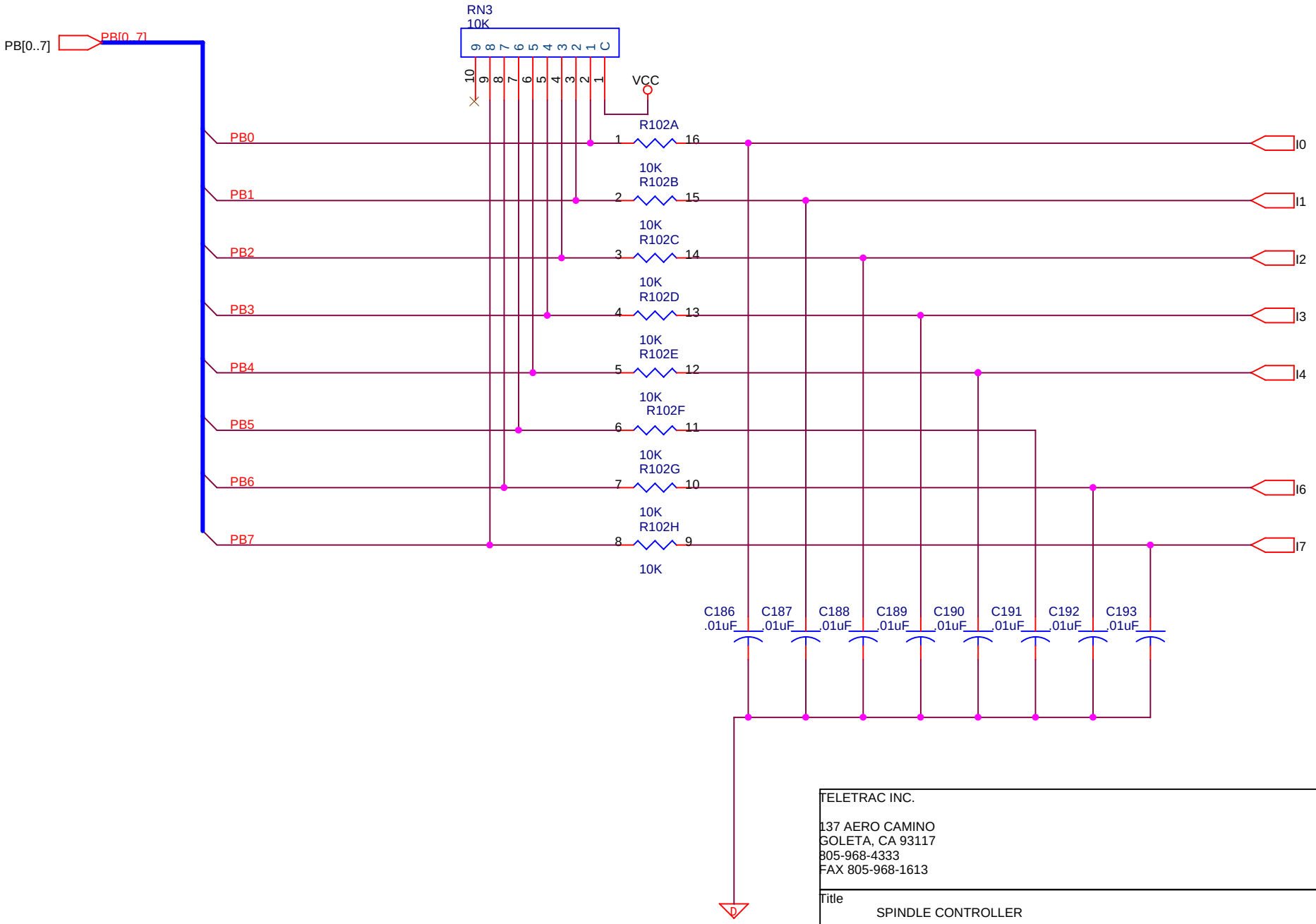
| | | |
|---|---|---|
| Teletrac | | |
| Title: PROCESSING ELECTRONICS LINEAR AXIS | | |
| Size: C | Document Number: 600160 | Rev: A |
| Date: Friday, April 26, 2002 | Sheet 6 of 10 | |




| | | | | |
|---|---|---|---|---|
| | TELETRAC INC.<br>360 SOUTH FAIRVIEW AVENUE<br>GOLETA, CA 93117<br>805-964-9260 | | | |
| | SERVO CONTROLLER CARD | | | |
| | Size C | CAGE Code | DWG NO 600160 | Rev A |
| Thursday, April 25, 2002 | Scale | | Sheet 7 of 10 | |


C165
22pF
Y1
3.6864MHz
U80
X1/CLK
X2
MC68681PC44
RS232
A[1..23]
UART*
RESET*
VCC
GND
TXDA
RXDA
TXDB
RXDB
WATCHDOG
U85
74HCT273
IACK*[1..7]
GPCCS*
D[0..15]
TP41
TP43
TP42
TEST POINT
U90
pcf8584p
SDA
SCL
R175
1.00K
R176
1.00K
R/W*
LDS*
CLK8
I2C-
JP15
HEADER 6
U68D
74LS05
D65
LED
R237
221
U73A
U73B
U73C
U73D
U81D
U71A
U71B
U71C
U71D
U81C
U77A
U77B
U81A
U77C
U81B
U77D
MC1488P
MC1489P
P6
COM1
FEMALE CONNECTOR
CONNECTOR DB9
P7
COM2
MALE CONNECTOR
SPIN_AMP_ENABLE
RESET_DSP*
MICROMODE
P9
HDB15
HIGH DENSITY
FEMALE CONNECTOR
Front Panel
Display
Interface
Teletrac Inc.
137 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613
Title
68000 ISA Mother Board
Size
B
Document Number
600160
Rev
A
Date: Thursday, April 25, 2002
Sheet 8 of 10


VOLTAGE REFERENCE
PLL LOOP FILTER/DRIVER
THREE PHASE MOTOR DRIVE
PHASE DETECTOR
LOOP FILTER
100Hz FILTER
ADJUSTABLE REFERENCE VOLTAGE
ADJUSTABLE BOUNDS VOLTAGE
DIRECT CONTROL
CLAMP
PLL FILT HEADER
SPINDLE ENCODER INPUT
SPINDLE MOTOR DRIVE
SPINDLE LOOK UP TABLE
THESE PARTS ARE GROUPED TOGETHER
Fabtronics Inc.
107 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613
SPINDLE CONTROLLER
Document Number 600160
Thursday, April 26, 2001
SPINDLE.SCH





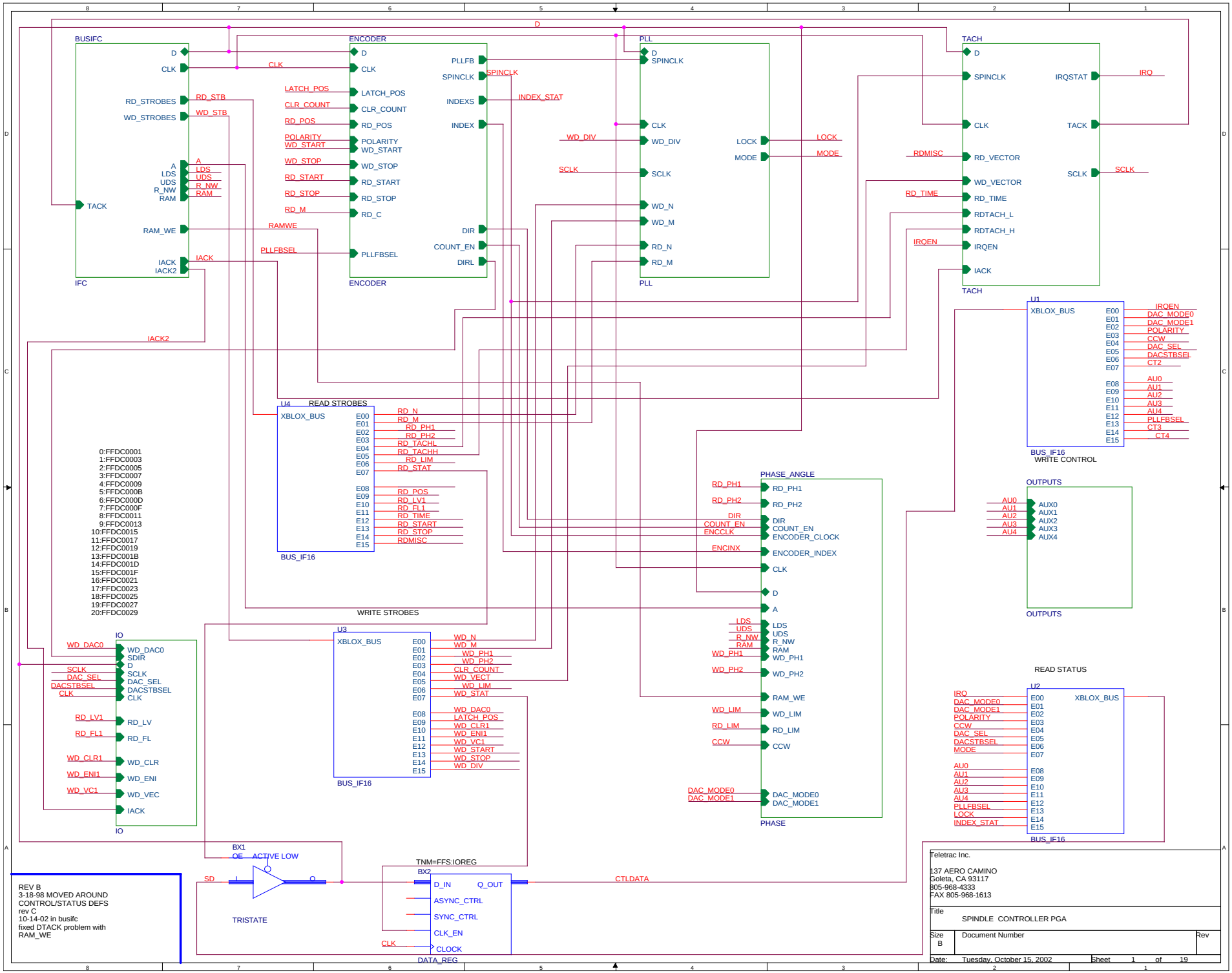
TELETRAC INC.

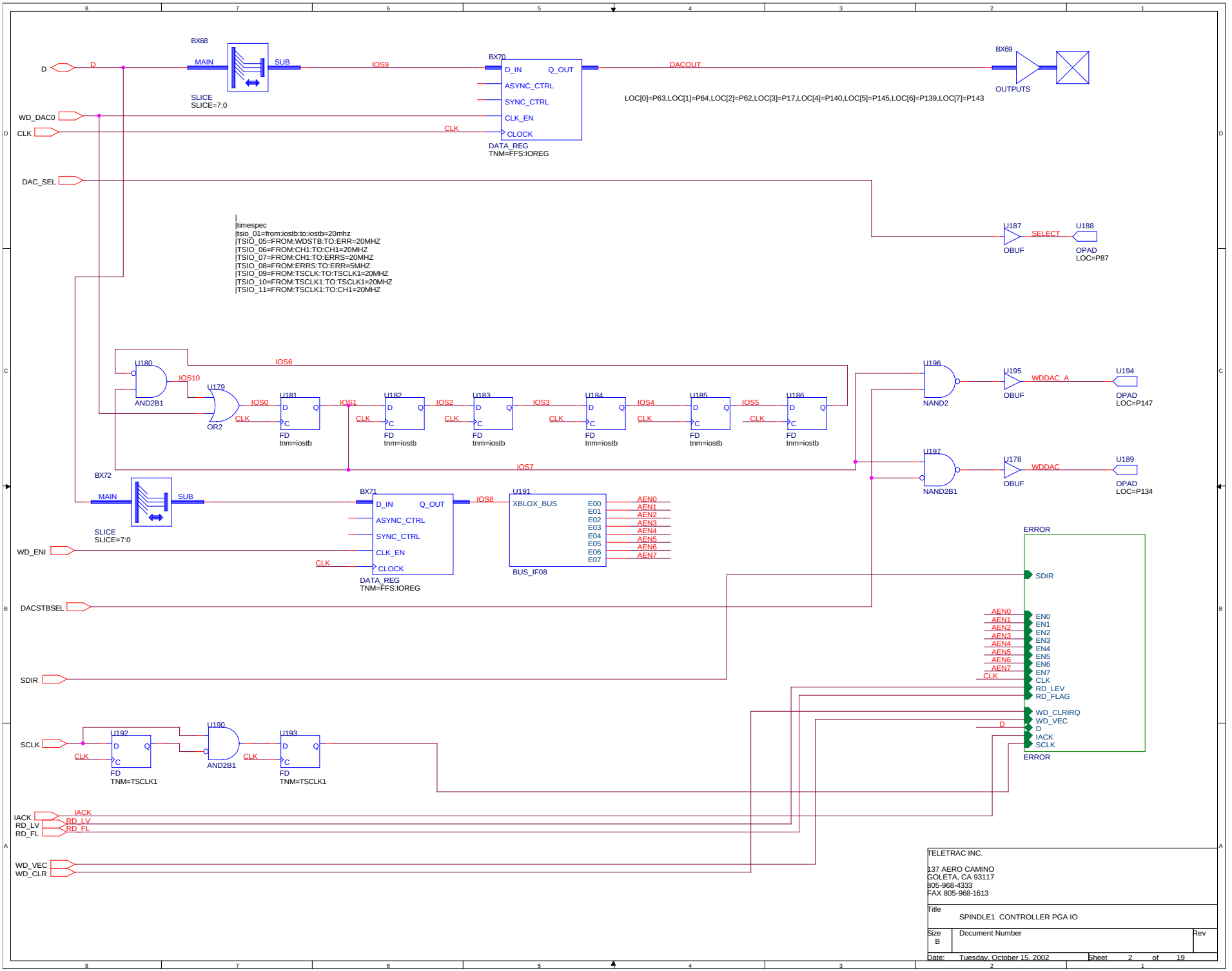
137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613


Title
SPINDLE CONTROLLER

| Size | Document Number | Rev |
|---|---|---|
| A | 600160 | A |

Date: Thursday, February 28, 2002 Sheet 10 of 10


BUSIFC
ENCODER
PLL
TACH
IFC
READ STROBES
WRITE STROBES
WRITE CONTROL
READ STATUS
PHASE_ANGLE
PHASE
OUTPUTS
IO
TRISTATE
DATA_REG
BUS_IF16
XBLOX_BUS
0:FFDC0001
1:FFDC0003
2:FFDC0005
3:FFDC0007
4:FFDC0009
5:FFDC000B
6:FFDC000D
7:FFDC000F
8:FFDC0011
9:FFDC0013
10:FFDC0015
11:FFDC0017
12:FFDC0019
13:FFDC001B
14:FFDC001D
15:FFDC001F
16:FFDC0021
17:FFDC0023
18:FFDC0025
19:FFDC0027
20:FFDC0029
REV B
3-18-98 MOVED AROUND
CONTROL/STATUS DEFS
rev C
10-14-02 in busifc
fixed DTACK problem with
RAM_WE
Teletrac Inc.
137 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613
SPINDLE CONTROLLER PGA
Size B
Document Number
Rev
Date: Tuesday, October 15, 2002
Sheet 1 of 19



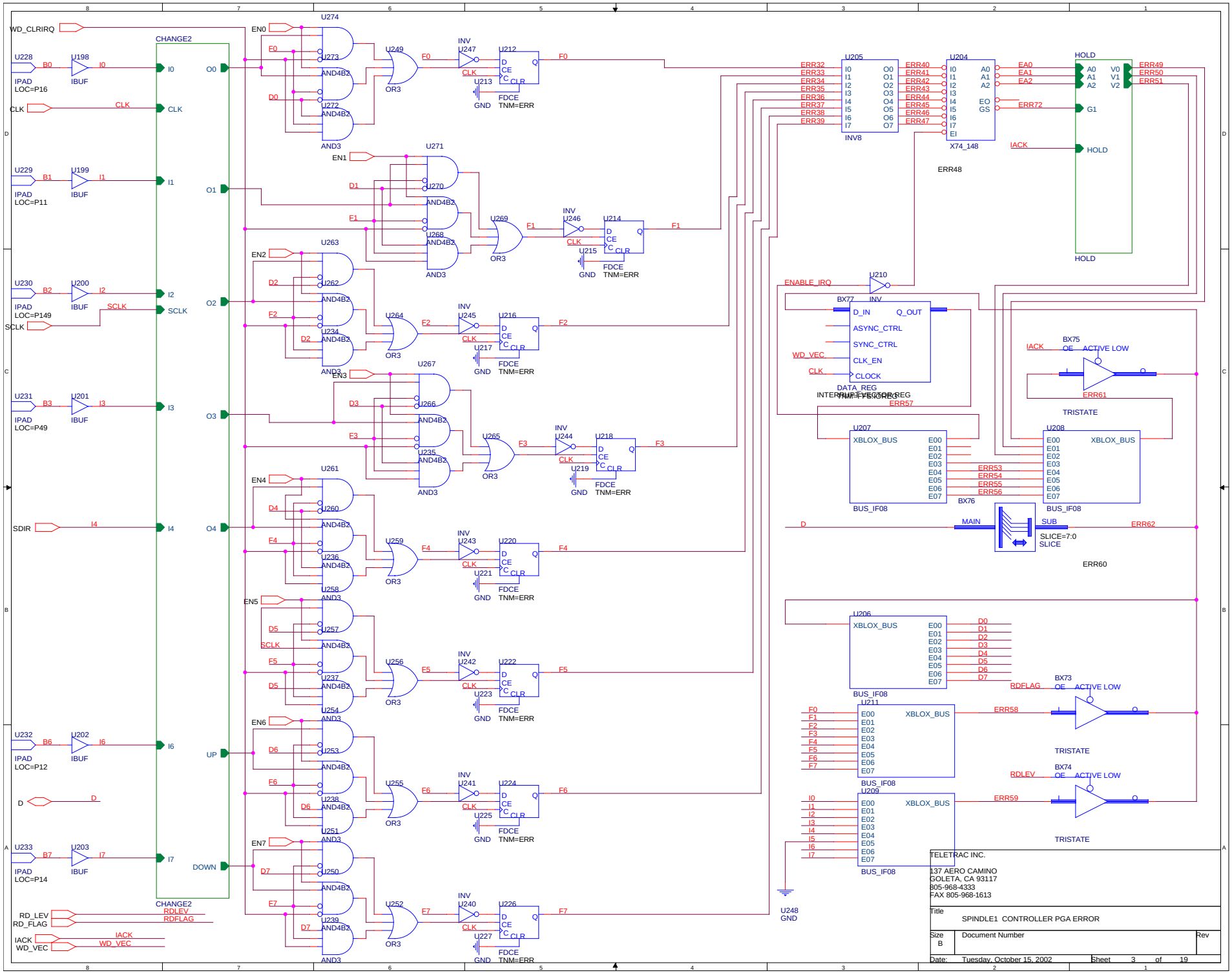

TELETRAC INC.

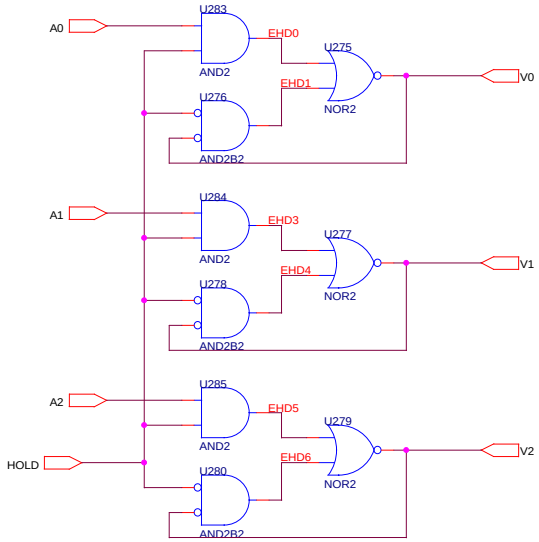
137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613

Title: SPINDLE1 CONTROLLER PGA IO

Size: B | Document Number | Rev

Date: Tuesday, October 15, 2002 | Sheet 2 of 19


WD_CLRIRQ
CHANGE2
U228
B0
U198
I0
IPAD
LOC=P16
IBUF
CLK
U229
B1
U199
I1
IPAD
LOC=P11
U230
B2
U200
I2
SCLK
IPAD
LOC=P149
U231
B3
U201
I3
IPAD
LOC=P49
SDIR
I4
U232
B6
U202
I6
IPAD
LOC=P12
D
U233
B7
U203
I7
IPAD
LOC=P14
RD_LEV
RD_FLAG
IACK
WD_VEC
RDLEV
RDFLAG
O0
O1
O2
O3
O4
UP
DOWN
EN0
EN1
EN2
EN3
EN4
EN5
EN6
EN7
U274
U273
U272
AND3
AND4B2
U249
OR3
INV
U247
U212
FDCE
TNM=ERR
U213
GND
U271
U270
U268
U269
U246
U214
U215
U263
U262
U234
U264
U245
U216
U217
U267
U266
U235
U265
U244
U218
U219
U261
U260
U236
U259
U243
U220
U221
U258
U257
U237
U256
U242
U222
U223
U254
U253
U238
U255
U241
U224
U225
U251
U250
U239
U252
U240
U226
U227
U248
U205
INV8
U204
X74_148
ERR48
HOLD
IACK
ERR72
ERR49
ERR50
ERR51
ENABLE_IRQ
U210
BX77
D_IN
Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
DATA_REG
WD_VEC
ERR57
BX75
OE
ACTIVE LOW
ERR61
TRISTATE
U207
U208
XBLOX_BUS
BUS_IF08
ERR53
ERR54
ERR55
ERR56
BX76
MAIN
SUB
SLICE=7:0
SLICE
ERR62
ERR60
U206
BX73
RDFLAG
ERR58
U211
U209
BX74
RDLEV
ERR59
TELETRAC INC.
137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613
Title
SPINDLE1 CONTROLLER PGA ERROR
Size
B
Document Number
Rev
Date: Tuesday, October 15, 2002
Sheet 3 of 19





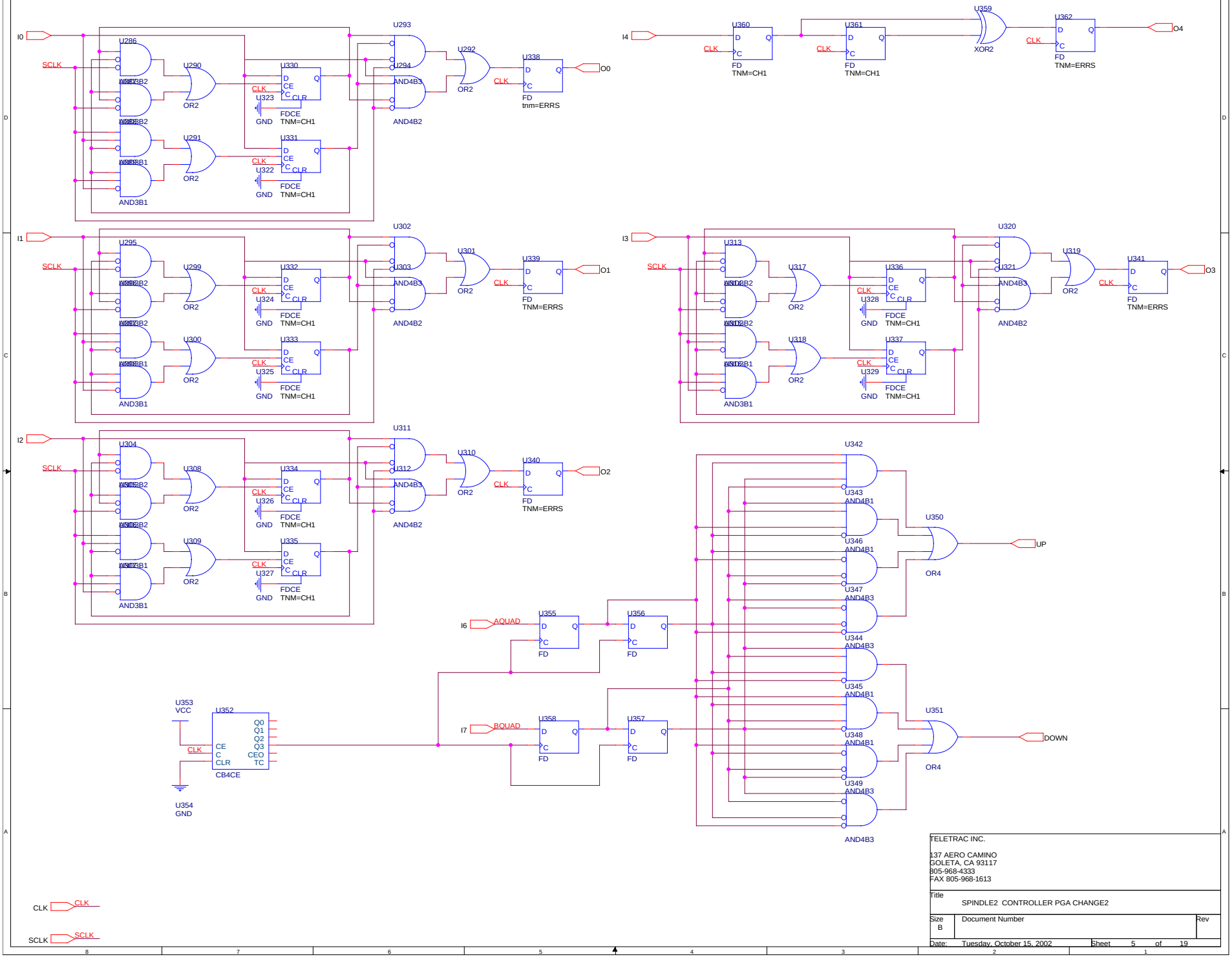
TELETRAC INC.

137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613

Title
SPINDLE1 CONTROLLER PGA

Size B | Document Number | Rev

Date: Tuesday, October 15, 2002 | Sheet 4 of 19



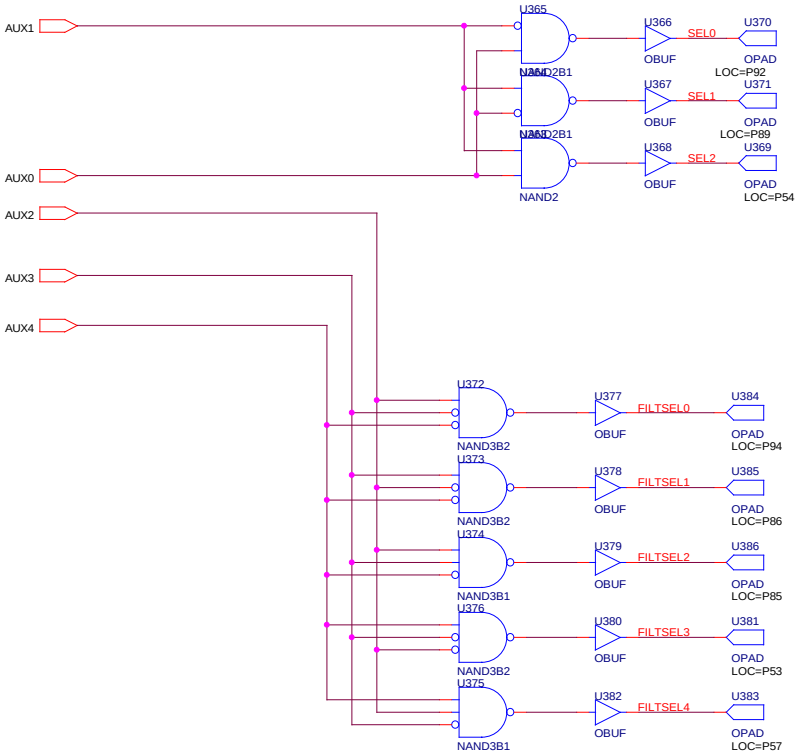

TELETRAC INC.

137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613

Title: SPINDLE2 CONTROLLER PGA CHANGE2

Size: B | Document Number | Rev

Date: Tuesday, October 15, 2002 | Sheet 5 of 19

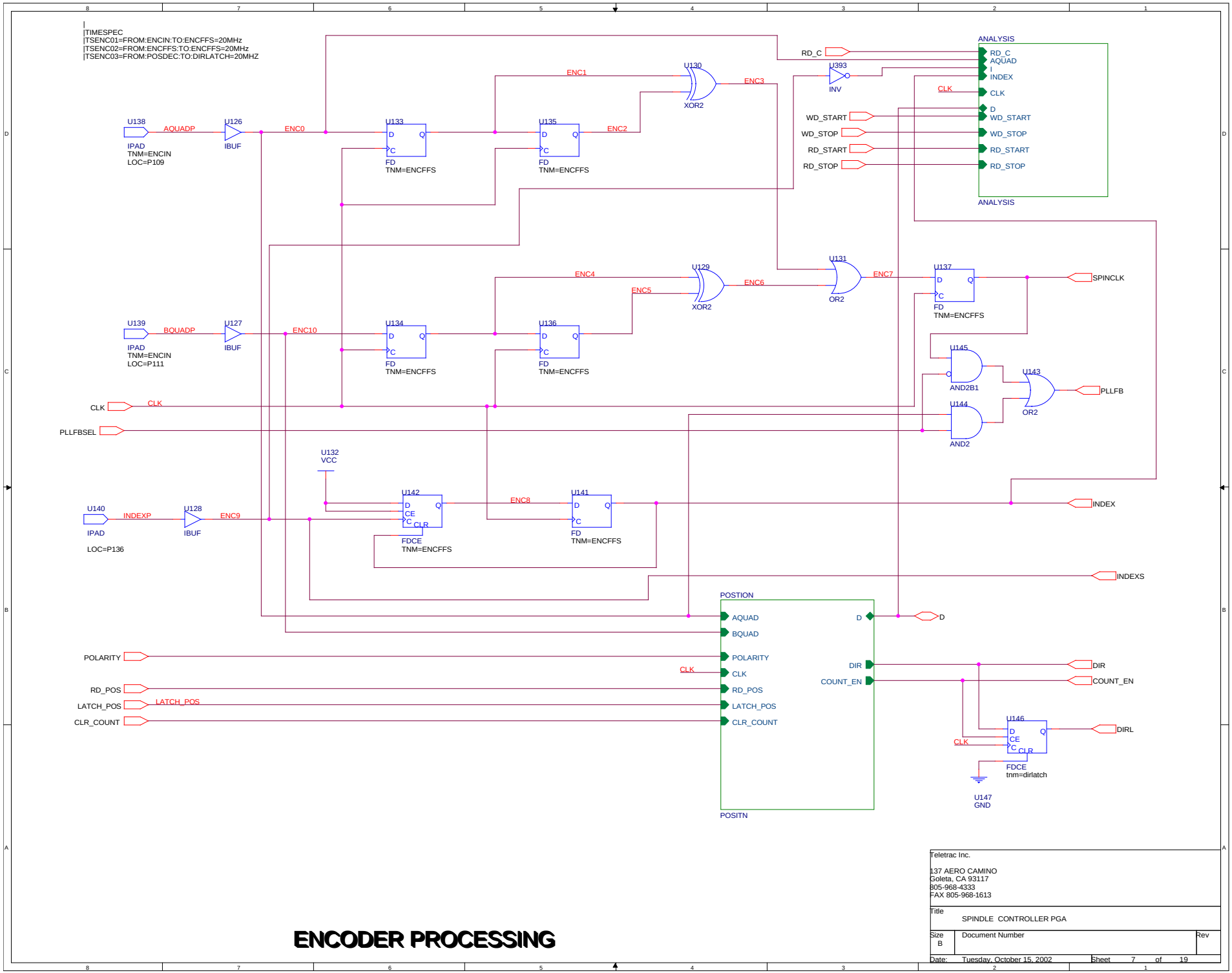



TELETRAC

137 AERO CAMINO
GOLETA, CA 93117 (805)-968-4333
FAX (805)-968-1613

| Size | Document Number | Rev |
|---|---|---|
| B | | |

Date: Tuesday, October 15, 2002 Sheet 6 of 19




# ENCODER PROCESSING

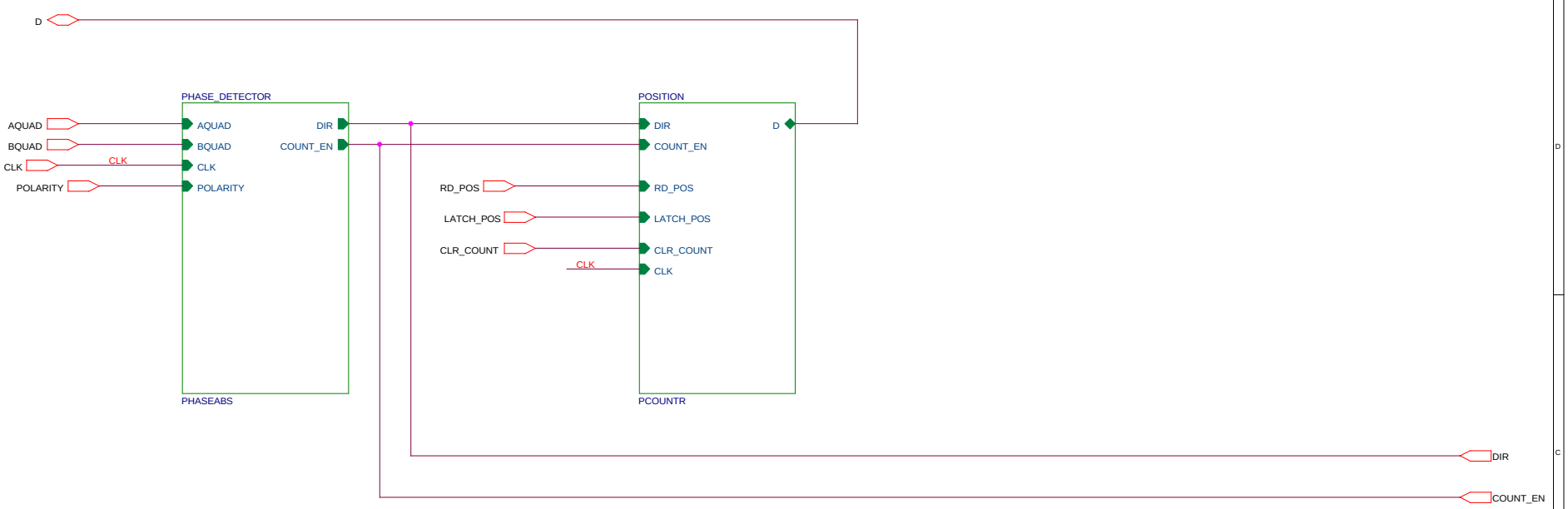

Teletrac Inc.

137 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613


Title: SPINDLE CONTROLLER PGA

Size: B | Document Number | Rev

Date: Tuesday, October 15, 2002 | Sheet 7 of 19



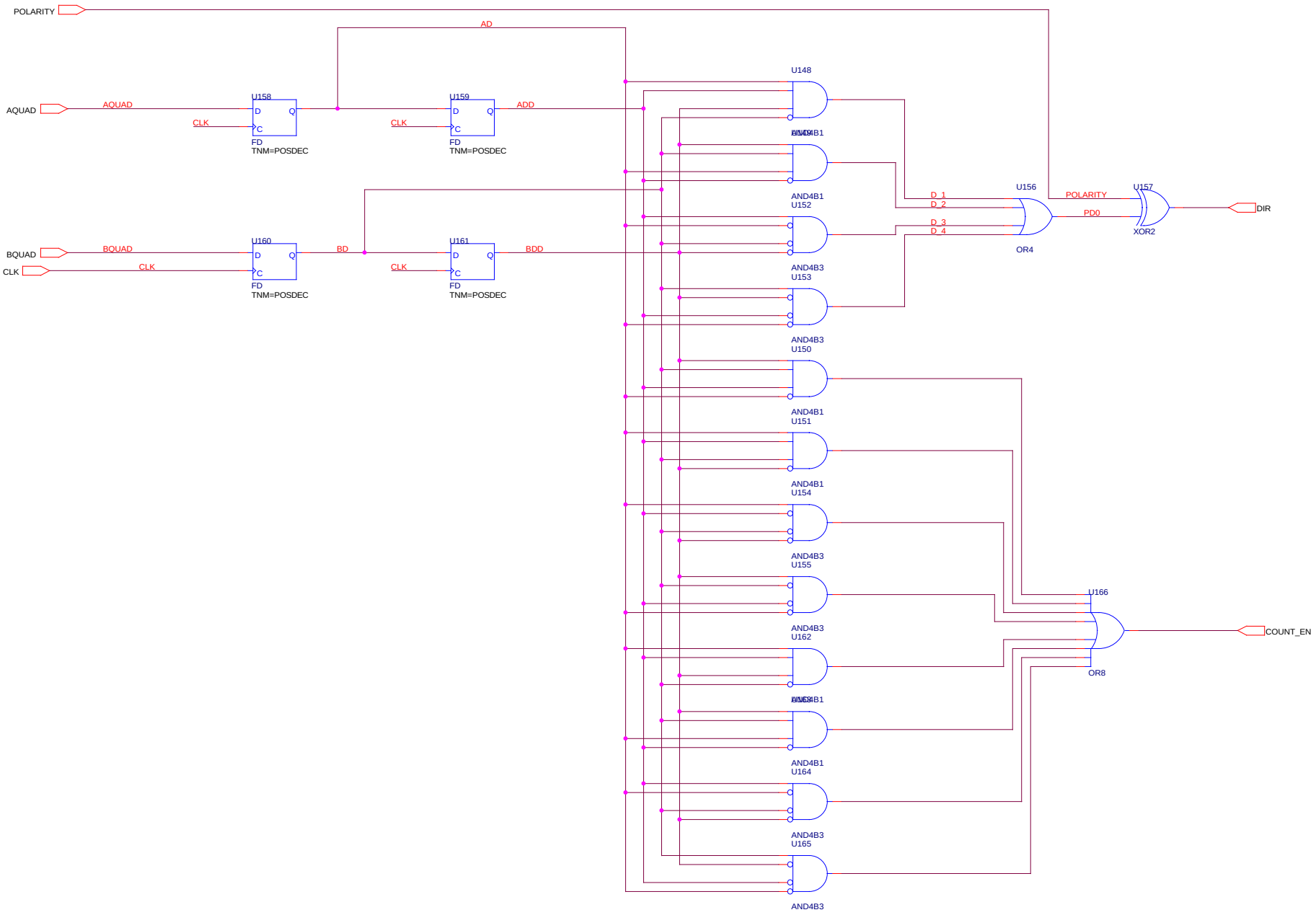

TELETRAC INC.

137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613

Title
SPINDLE CONTROLLER PGA

Size B | Document Number | Rev

Date: Tuesday, October 15, 2002 | Sheet 8 of 19




Teletrac Inc.

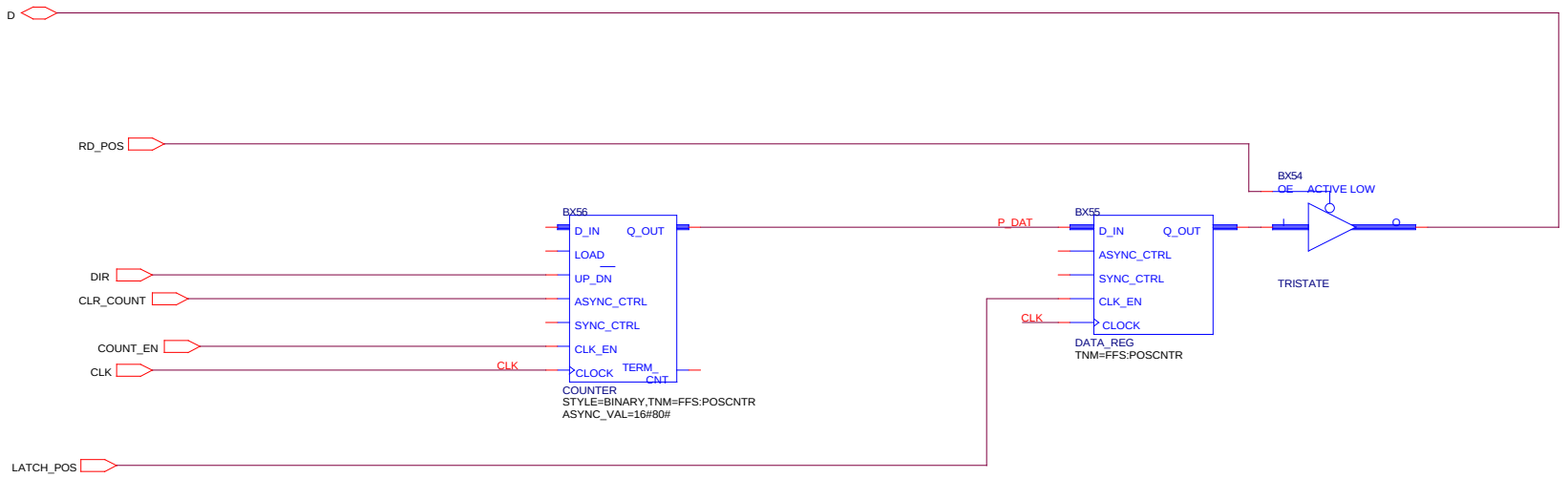
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260

Title: SERVO GATE ARRAY PHASE DETECTOR

| Size | Document Number | Rev |
|---|---|---|
| B | | B |

Date: Tuesday, October 15, 2002 Sheet 9 of 19

|
|TIMESPEC
|TSPC_01=FROM:POSCNTR:TO:POSCNTR=20MHz
|TSPC_02=FROM:POSDEC:TO:POSCNTR=20MHz





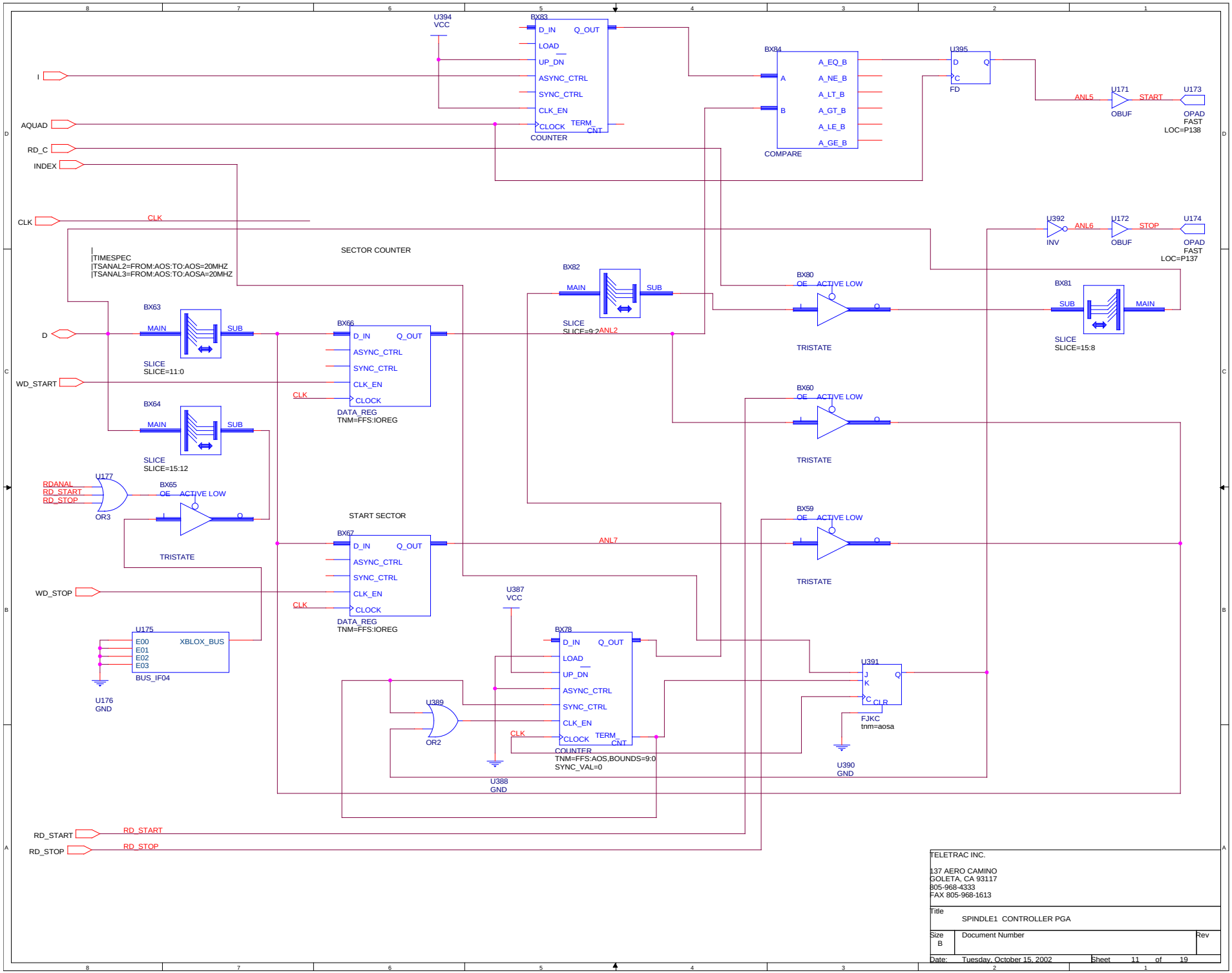
Teletrac Inc.

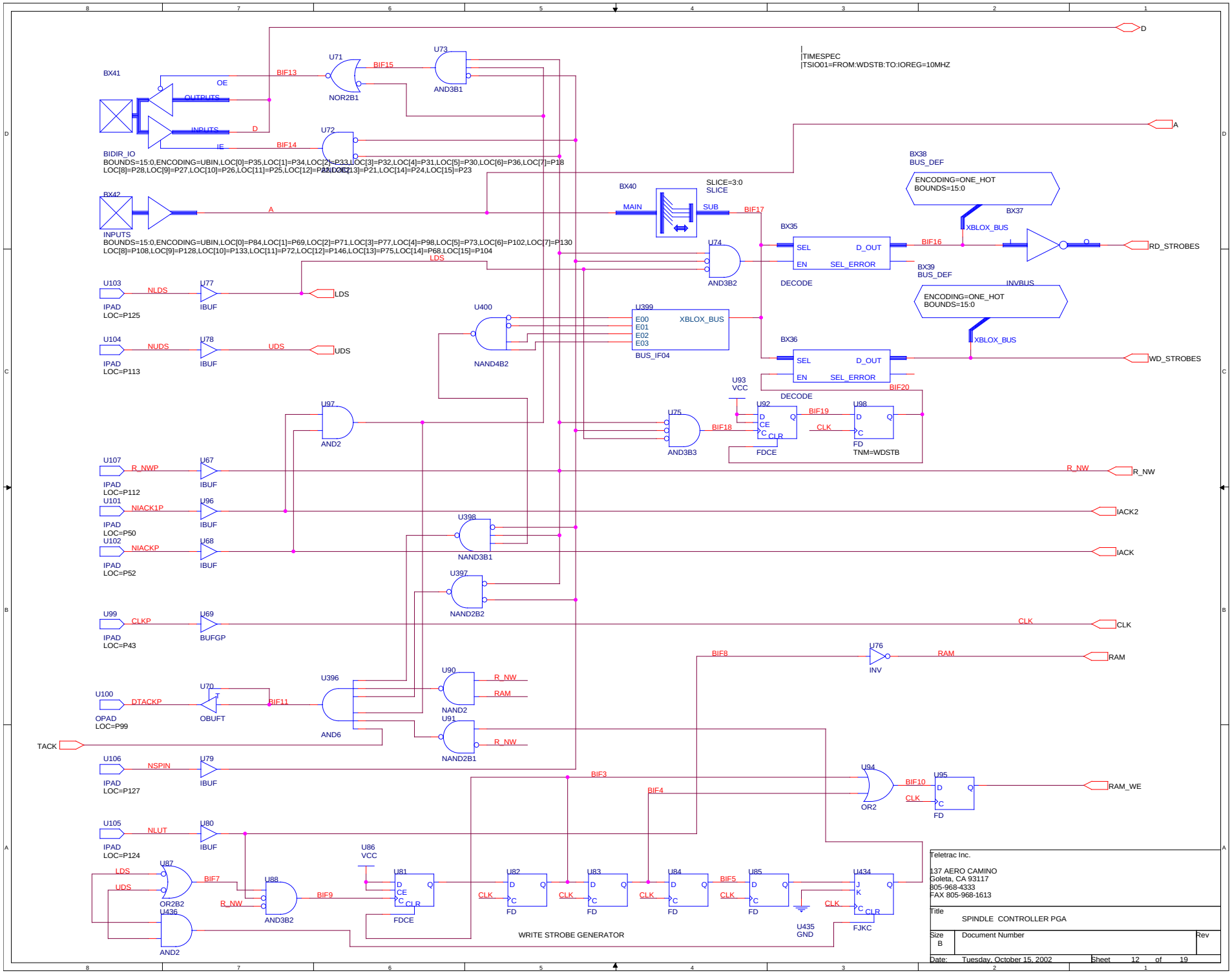
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260

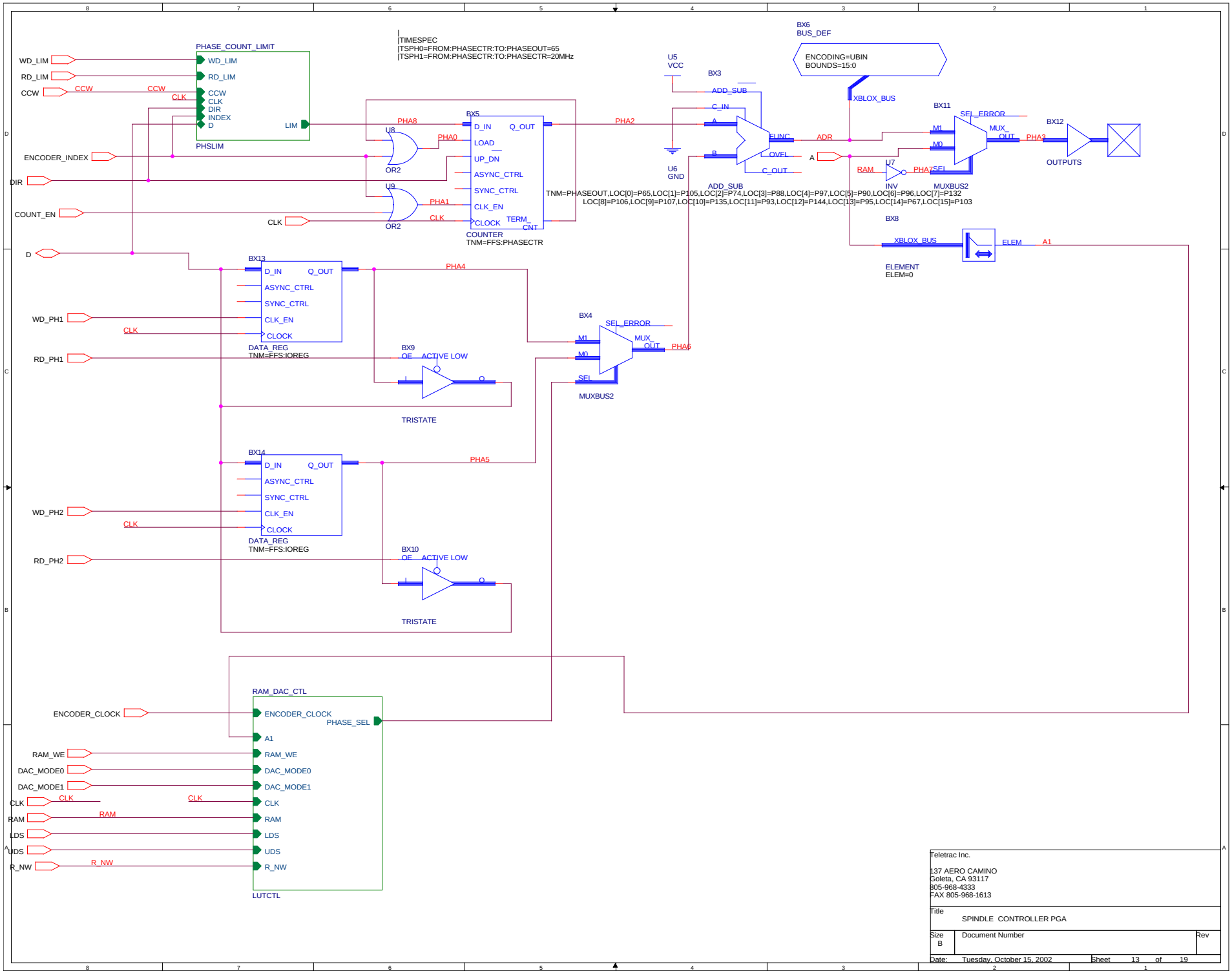

Title
SERVO GATE ARRAY POSITION COUNTER

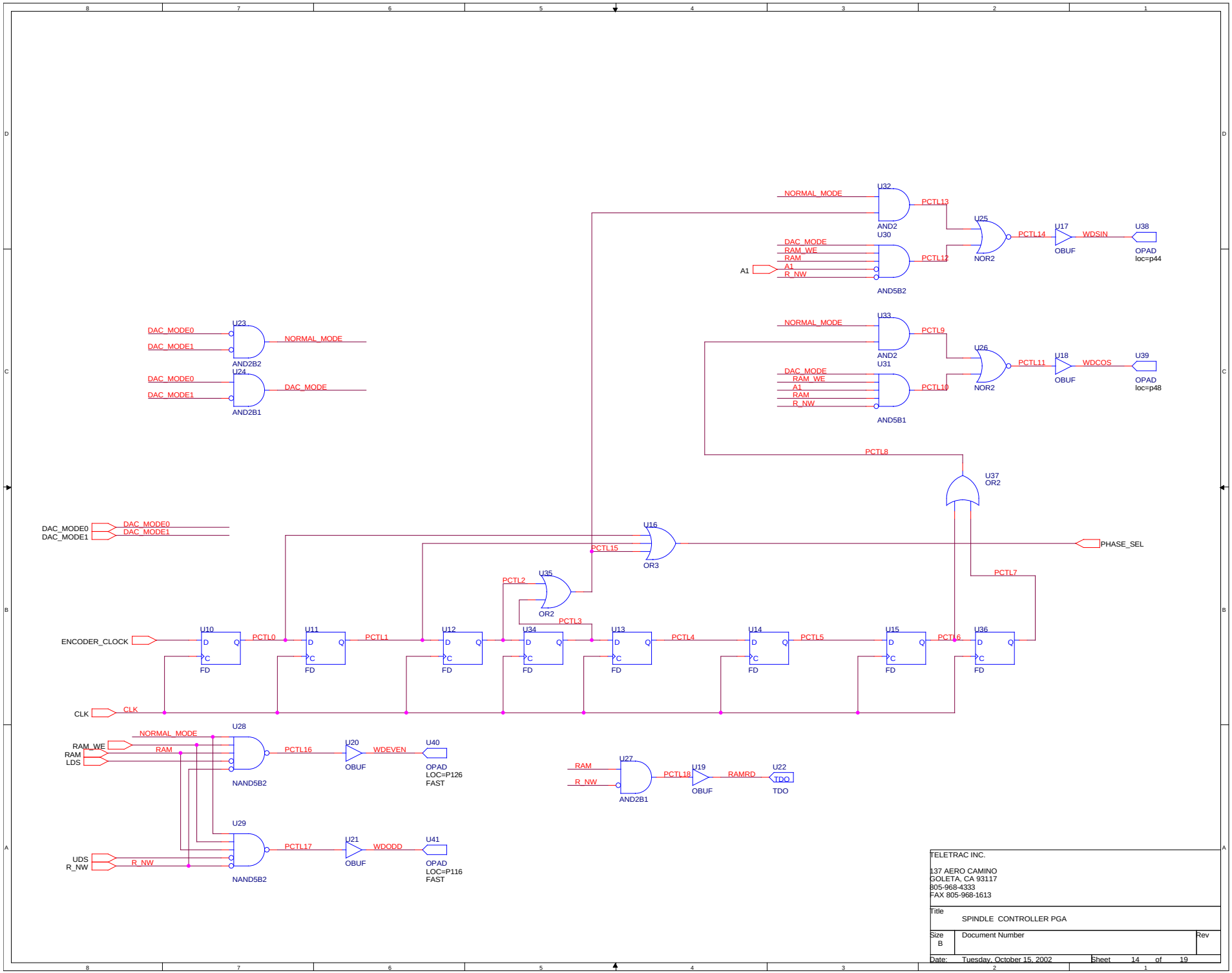
| Size | Document Number | Rev |
|---|---|---|
| B | | B |

Date: Tuesday, October 15, 2002 Sheet 10 of 19

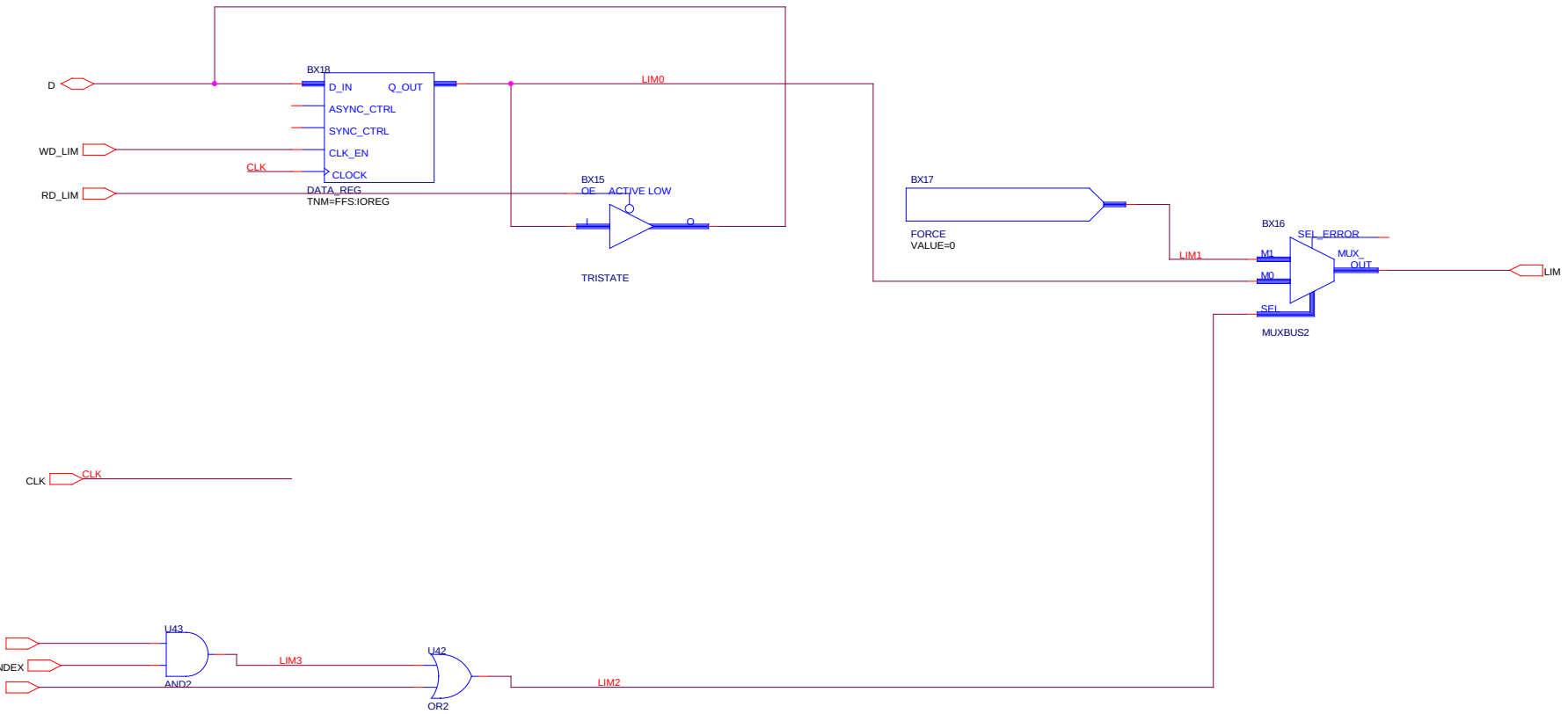

U394
VCC
BX83
D_IN
Q_OUT
LOAD
UP_DN
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
TERM
CNT
COUNTER
BX84
A
B
A_EQ_B
A_NE_B
A_LT_B
A_GT_B
A_LE_B
A_GE_B
COMPARE
U395
D
Q
C
FD
ANL5
U171
OBUF
START
U173
OPAD
FAST
LOC=P138
I
AQUAD
RD_C
INDEX
CLK
SECTOR COUNTER
|TIMESPEC
|TSANAL2=FROM:AOS:TO:AOS=20MHZ
|TSANAL3=FROM:AOS:TO:AOSA=20MHZ
U392
INV
ANL6
U172
OBUF
STOP
U174
OPAD
FAST
LOC=P137
BX82
MAIN
SUB
SLICE
SLICE=9:2
ANL2
BX80
OE
ACTIVE LOW
I
O
TRISTATE
BX81
SUB
MAIN
SLICE
SLICE=15:8
BX63
MAIN
SUB
SLICE
SLICE=11:0
D
WD_START
BX66
D_IN
Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
DATA_REG
TNM=FFS:IOREG
CLK
BX64
MAIN
SUB
SLICE
SLICE=15:12
BX60
OE
ACTIVE LOW
I
O
TRISTATE
RDANAL
RD_START
RD_STOP
U177
OR3
BX65
OE
ACTIVE LOW
I
O
TRISTATE
START SECTOR
BX67
D_IN
Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
DATA_REG
TNM=FFS:IOREG
CLK
ANL7
BX59
OE
ACTIVE LOW
I
O
TRISTATE
WD_STOP
U175
E00
E01
E02
E03
XBLOX_BUS
BUS_IF04
U176
GND
U387
VCC
BX78
D_IN
Q_OUT
LOAD
UP_DN
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
TERM
CNT
COUNTER
TNM=FFS:AOS,BOUNDS=9:0
SYNC_VAL=0
U389
OR2
CLK
U388
GND
U391
J
K
Q
C
CLR
FJKC
tnm=aosa
U390
GND
RD_START
RD_STOP
RD_START
RD_STOP
TELETRAC INC.
137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613
Title
SPINDLE1 CONTROLLER PGA
Size
B
Document Number
Rev
Date: Tuesday, October 15, 2002
Sheet 11 of 19


D
|TIMESPEC
|TSIO01=FROM:WDSTB:TO:IOREG=10MHZ
BX41
OE
OUTPUTS
INPUTS
D
IE
BIDIR_IO
BOUNDS=15:0,ENCODING=UBIN,LOC[0]=P35,LOC[1]=P34,LOC[2]=P33,LOC[3]=P32,LOC[4]=P31,LOC[5]=P30,LOC[6]=P36,LOC[7]=P18
LOC[8]=P28,LOC[9]=P27,LOC[10]=P26,LOC[11]=P25,LOC[12]=P23,LOC[13]=P21,LOC[14]=P24,LOC[15]=P23
U71
NOR2B1
BIF13
BIF15
U73
AND3B1
U72
BIF14
BX42
INPUTS
BOUNDS=15:0,ENCODING=UBIN,LOC[0]=P84,LOC[1]=P69,LOC[2]=P71,LOC[3]=P77,LOC[4]=P98,LOC[5]=P73,LOC[6]=P102,LOC[7]=P130
LOC[8]=P108,LOC[9]=P128,LOC[10]=P133,LOC[11]=P72,LOC[12]=P146,LOC[13]=P75,LOC[14]=P68,LOC[15]=P104
A
BX40
MAIN
SUB
SLICE=3:0
SLICE
BIF17
BX38
BUS_DEF
ENCODING=ONE_HOT
BOUNDS=15:0
BX37
XBLOX_BUS
BX35
SEL
EN
D_OUT
SEL_ERROR
DECODE
BIF16
RD_STROBES
U74
AND3B2
LDS
U103
NLDS
IPAD
LOC=P125
U77
IBUF
LDS
U104
NUDS
IPAD
LOC=P113
U78
UDS
U400
NAND4B2
U399
E00
E01
E02
E03
XBLOX_BUS
BUS_IF04
BX39
BUS_DEF
INVBUS
ENCODING=ONE_HOT
BOUNDS=15:0
XBLOX_BUS
BX36
WD_STROBES
U93
VCC
BIF20
U97
AND2
U75
AND3B3
BIF18
U92
CE
CLR
FDCE
BIF19
CLK
U98
FD
TNM=WDSTB
U107
R_NWP
IPAD
LOC=P112
U67
R_NW
U101
NIACK1P
IPAD
LOC=P50
U96
IACK2
U102
NIACKP
IPAD
LOC=P52
U68
IACK
U398
NAND3B1
U397
NAND2B2
U99
CLKP
IPAD
LOC=P43
U69
BUFGP
CLK
BIF8
U76
INV
RAM
U396
AND6
U90
NAND2
U100
DTACKP
OPAD
LOC=P99
U70
OBUFT
BIF11
U91
NAND2B1
TACK
U106
NSPIN
IPAD
LOC=P127
U79
BIF3
BIF4
U94
OR2
BIF10
U95
FD
RAM_WE
U105
NLUT
IPAD
LOC=P124
U80
U86
VCC
U87
OR2B2
BIF7
U88
AND3B2
BIF9
U436
AND2
U81
FDCE
U82
U83
U84
U85
BIF5
FD
U435
GND
U434
FJKC
WRITE STROBE GENERATOR
Teletrac Inc.
137 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613
Title
SPINDLE CONTROLLER PGA
Size
B
Document Number
Rev
Date: Tuesday, October 15, 2002
Sheet 12 of 19


PHASE_COUNT_LIMIT
WD_LIM
RD_LIM
CCW
CLK
DIR
INDEX
D
LIM
PHSLIM
TIMESPEC
TSPH0=FROM:PHASECTR:TO:PHASEOUT=65
TSPH1=FROM:PHASECTR:TO:PHASECTR=20MHz
BX6
BUS_DEF
ENCODING=UBIN
BOUNDS=15:0
XBLOX_BUS
U5
VCC
U6
GND
BX3
ADD_SUB
C_IN
A
B
FUNC
OVFL
C_OUT
ADR
ADD_SUB
BX5
D_IN
Q_OUT
LOAD
UP_DN
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
TERM_CNT
COUNTER
TNM=FFS:PHASECTR
U8
U9
OR2
PHA8
PHA0
PHA1
PHA2
ENCODER_INDEX
COUNT_EN
TNM=PHASEOUT,LOC[0]=P65,LOC[1]=P105,LOC[2]=P74,LOC[3]=P88,LOC[4]=P97,LOC[5]=P90,LOC[6]=P96,LOC[7]=P132
LOC[8]=P106,LOC[9]=P107,LOC[10]=P135,LOC[11]=P93,LOC[12]=P144,LOC[13]=P95,LOC[14]=P67,LOC[15]=P103
BX11
SEL_ERROR
M1
M0
MUX_OUT
PHA3
BX12
OUTPUTS
RAM
U7
INV
PHA7
SEL
MUXBUS2
BX8
XBLOX_BUS
ELEM
A1
ELEMENT
ELEM=0
D
BX13
PHA4
WD_PH1
RD_PH1
DATA_REG
TNM=FFS:IOREG
BX9
OE
ACTIVE LOW
I
O
TRISTATE
BX4
PHA6
MUXBUS2
BX14
PHA5
WD_PH2
RD_PH2
BX10
RAM_DAC_CTL
ENCODER_CLOCK
PHASE_SEL
A1
RAM_WE
DAC_MODE0
DAC_MODE1
CLK
RAM
LDS
UDS
R_NW
LUTCTL
Teletrac Inc.
137 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613
Title
SPINDLE CONTROLLER PGA
Size
B
Document Number
Rev
Date: Tuesday, October 15, 2002
Sheet 13 of 19




| | |
|---|---|
| TELETRAC INC. | |
| 137 AERO CAMINO<br>GOLETA, CA 93117<br>805-968-4333<br>FAX 805-968-1613 | |
| Title | SPINDLE CONTROLLER PGA |
| Size B | Document Number — Rev |
| Date: Tuesday, October 15, 2002 | Sheet 14 of 19 |



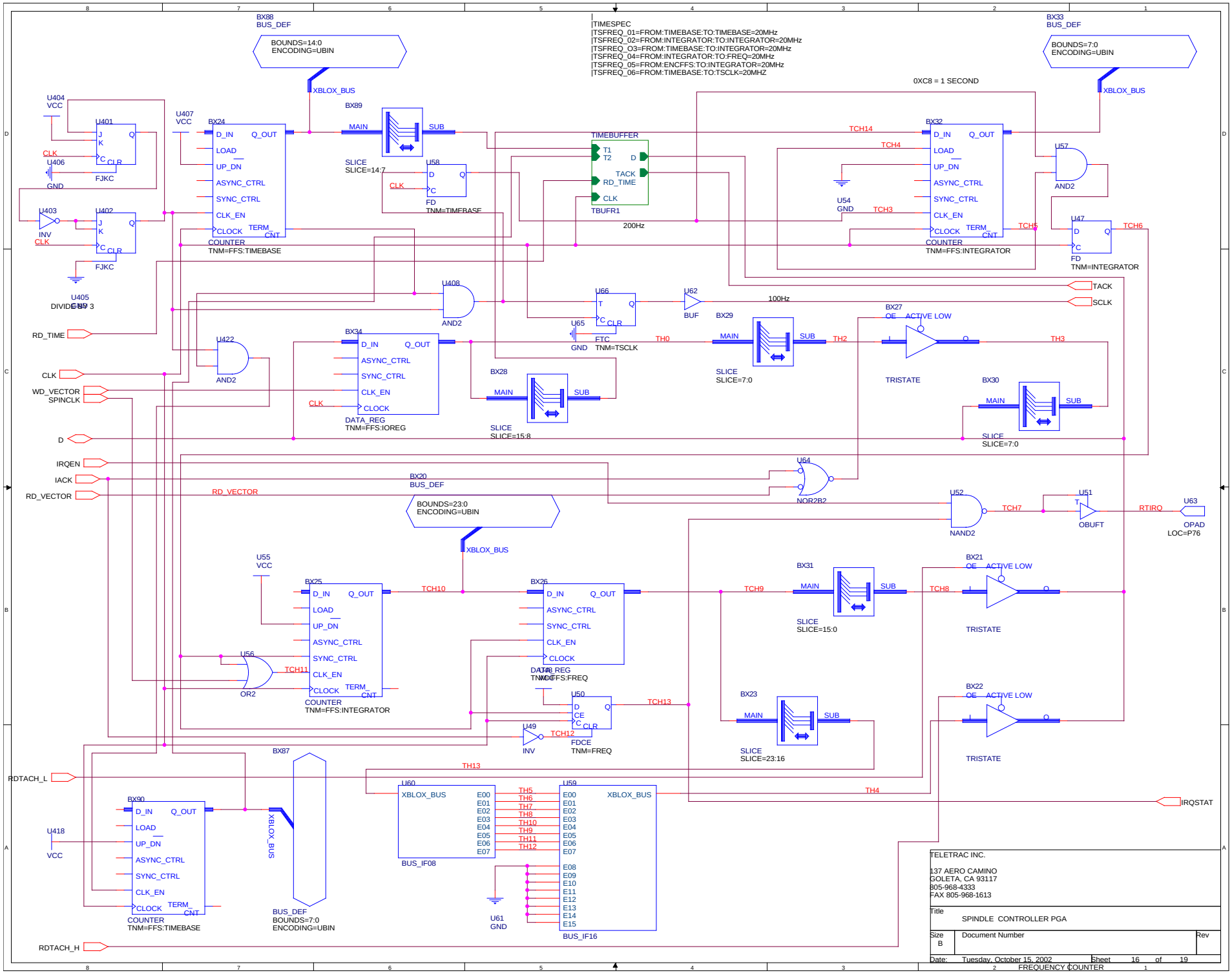

TELETRAC INC.

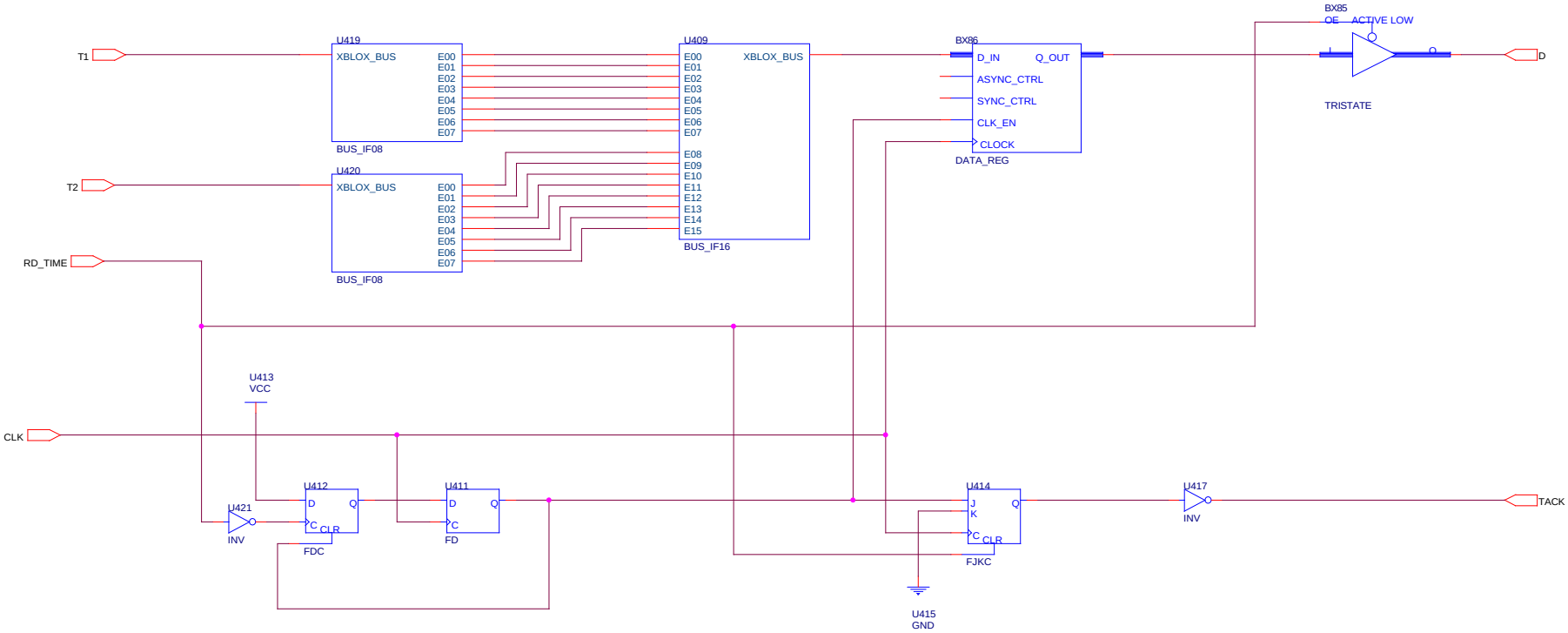
137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613

Title
SPINDLE CONTROLLER PGA

Size B | Document Number | Rev

Date: Tuesday, October 15, 2002 | Sheet 15 of 19


BX88
BUS_DEF
BOUNDS=14:0
ENCODING=UBIN
XBLOX_BUS
|TIMESPEC
|TSFREQ_01=FROM:TIMEBASE:TO:TIMEBASE=20MHz
|TSFREQ_02=FROM:INTEGRATOR:TO:INTEGRATOR=20MHz
|TSFREQ_O3=FROM:TIMEBASE:TO:INTEGRATOR=20MHz
|TSFREQ_04=FROM:INTEGRATOR:TO:FREQ=20MHz
|TSFREQ_05=FROM:ENCFFS:TO:INTEGRATOR=20MHz
|TSFREQ_06=FROM:TIMEBASE:TO:TSCLK=20MHz
BX33
BUS_DEF
BOUNDS=7:0
ENCODING=UBIN
XBLOX_BUS
0XC8 = 1 SECOND
U404
VCC
U401
J
K
Q
C
CLR
FJKC
CLK
U406
GND
U403
INV
U402
U405
DIVIDE BY 3
U407
VCC
BX24
D_IN
Q_OUT
LOAD
UP_DN
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
TERM_CNT
COUNTER
TNM=FFS:TIMEBASE
BX89
MAIN
SUB
SLICE
SLICE=14:7
U58
D
Q
C
FD
TNM=TIMEBASE
TIMEBUFFER
T1
T2
D
TACK
RD_TIME
CLK
TBUFR1
200Hz
TCH14
TCH4
U54
GND
TCH3
BX32
U57
AND2
U47
TCH5
TCH6
FD
TNM=INTEGRATOR
TACK
SCLK
U408
AND2
U66
T
Q
C
CLR
U62
BUF
100Hz
U65
FTC
TNM=TSCLK
TH0
BX29
TH2
BX27
OE
ACTIVE LOW
TRISTATE
TH3
BX30
SLICE
SLICE=7:0
RD_TIME
CLK
WD_VECTOR
SPINCLK
U422
AND2
BX34
DATA_REG
TNM=FFS:IOREG
BX28
SLICE
SLICE=15:8
D
IRQEN
IACK
RD_VECTOR
BX20
BUS_DEF
BOUNDS=23:0
ENCODING=UBIN
XBLOX_BUS
U64
NOR2R2
U52
NAND2
TCH7
U51
T
OBUFT
RTIRQ
U63
OPAD
LOC=P76
U55
VCC
BX25
TCH10
BX26
DATA_REG
TNM=FFS:FREQ
BX31
TCH9
SLICE
SLICE=15:0
BX21
TCH8
TRISTATE
U56
OR2
TCH11
COUNTER
TNM=FFS:INTEGRATOR
U50
D
CE
C
CLR
Q
TCH13
U49
INV
TCH12
FDCE
TNM=FREQ
BX23
SLICE
SLICE=23:16
BX22
TRISTATE
RDTACH_L
TH13
TH4
IRQSTAT
BX87
BX90
U418
VCC
COUNTER
TNM=FFS:TIMEBASE
BUS_DEF
BOUNDS=7:0
ENCODING=UBIN
U60
XBLOX_BUS
E00
E01
E02
E03
E04
E05
E06
E07
BUS_IF08
TH5
TH6
TH7
TH8
TH10
TH9
TH11
TH12
U59
XBLOX_BUS
E08
E09
E10
E11
E12
E13
E14
E15
BUS_IF16
U61
GND
RDTACH_H
TELETRAC INC.
137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
FAX 805-968-1613
Title
SPINDLE CONTROLLER PGA
Size
B
Document Number
Rev
Date: Tuesday, October 15, 2002
Sheet 16 of 19
FREQUENCY COUNTER

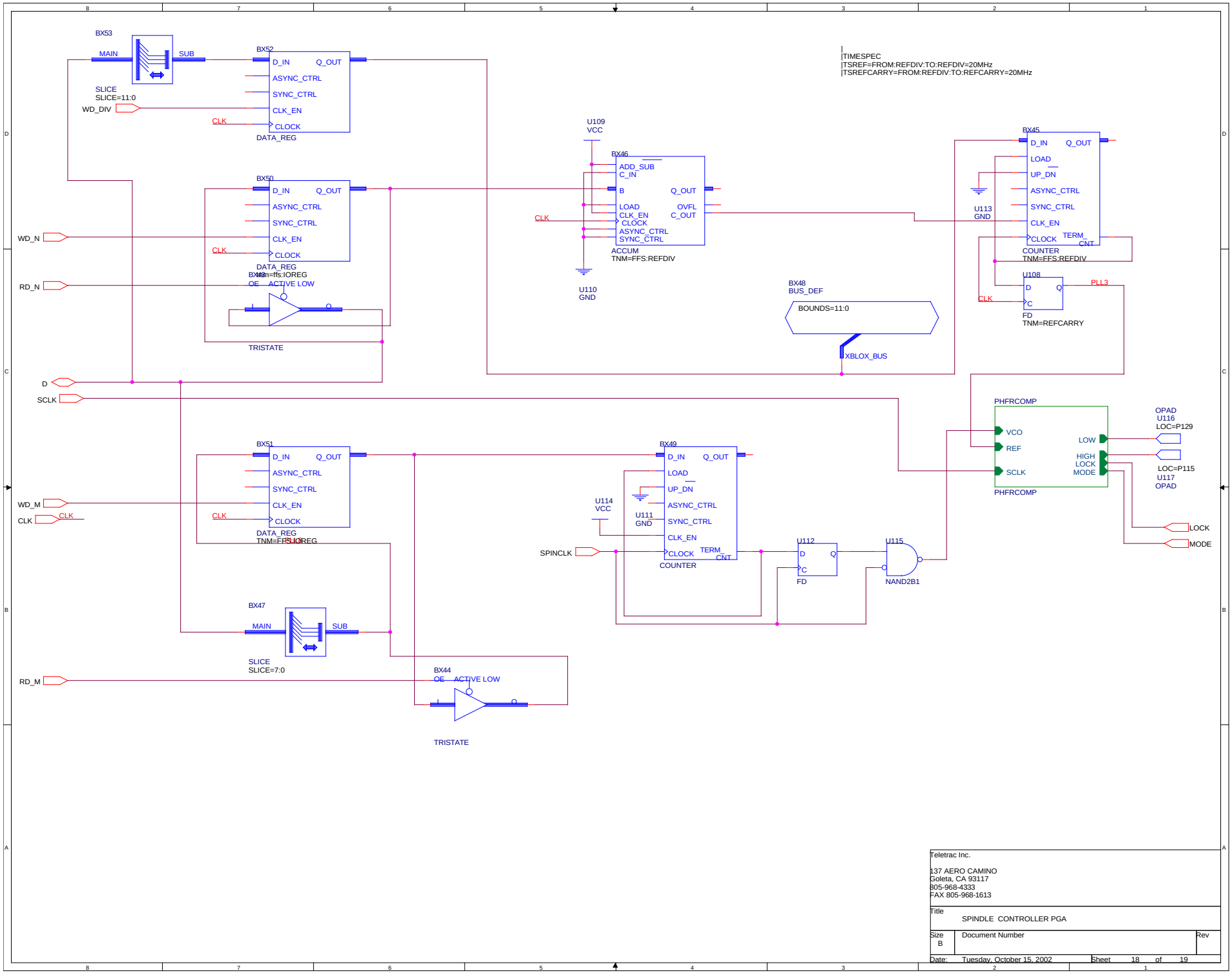



TELETRAC

137 AERO CAMINO
GOLETA, CA 93117 (805)-968-4333
FAX (805)-968-1613

| Size | Document Number | Rev |
|---|---|---|
| B | | |

Date: Tuesday, October 15, 2002 Sheet 17 of 19

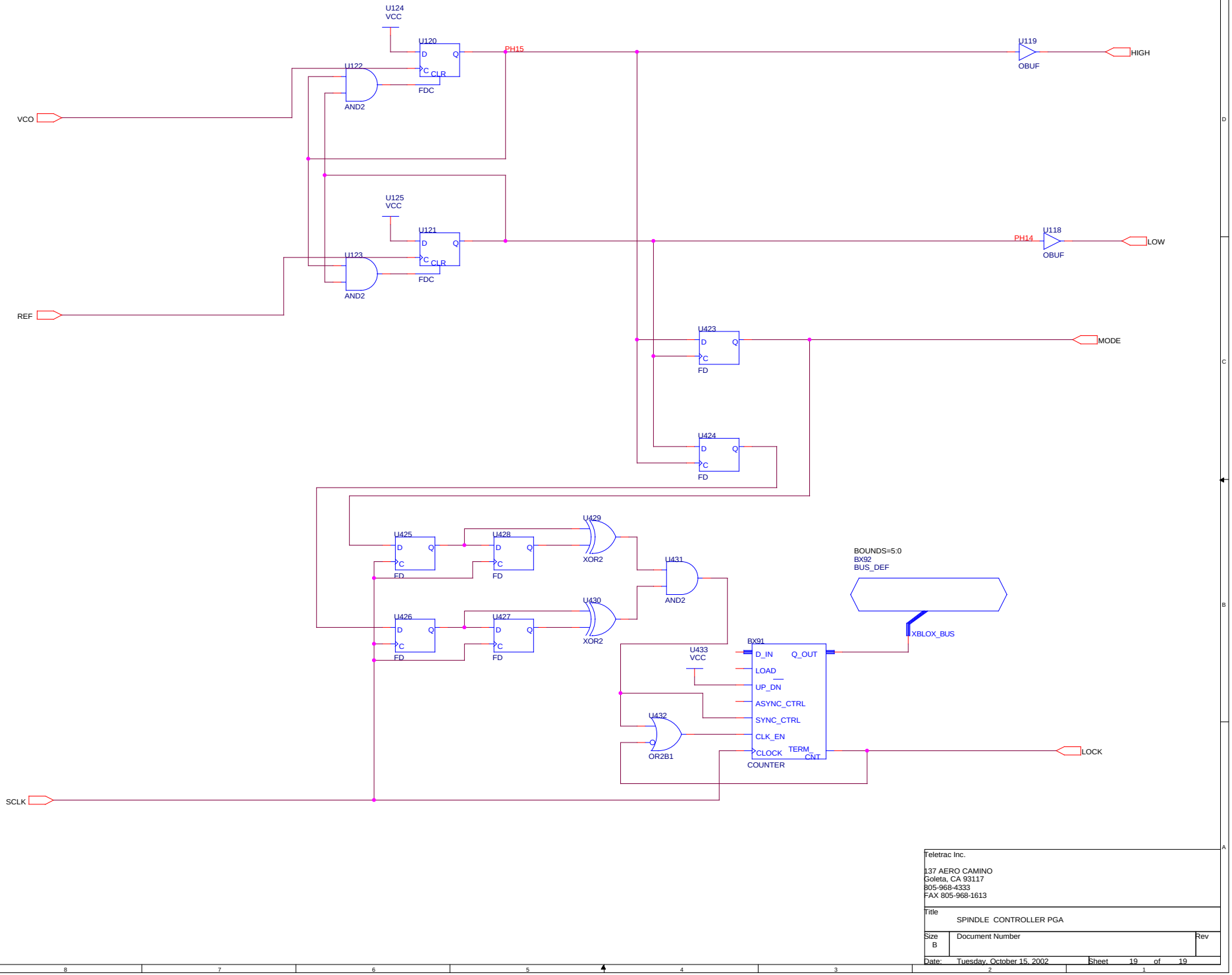



Teletrac Inc.
137 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613

Title: SPINDLE CONTROLLER PGA

Size: B | Document Number | Rev

Date: Tuesday, October 15, 2002 | Sheet 18 of 19




Teletrac Inc.

137 AERO CAMINO
Goleta, CA 93117
805-968-4333
FAX 805-968-1613

Title: SPINDLE CONTROLLER PGA

Size: B | Document Number | Rev

Date: Tuesday, October 15, 2002 | Sheet 19 of 19

Ver:10-31-00:added debug readback of rotary position
Ver:07-18-01:added latch in clear logic to prevent glitches:dsp_irq.sch
Ver:07-26-01:changed data reg clock to 20mhz
Ver:11-19-01:ADDED COUNTER TO COUNT DOWN AT RTC RATE

|
|TIMESPEC
|TSIO_01=FROM:WDSTROBE:TO:DATAREG=13MHz
|TSIO_02=FROM:CLKDIV:TO:CLKDIV=40MHz
|TSIO_04=FROM:WDSTROBE:TO:MEDATAREG=11MHz
|TSIO_05=FROM:WDSTROBE:TO:POSDATAREG=11MHz
|TSIO_06=FROM:WDSTROBE:TO:DACDATAREG=13MHz

RTC_COUNTER
WD_CNTR
RD_CNTR
D
CLK20
RTCEN
RTCOUNT

INTERFEROMETER_IFC
RD_POS2L
RD_POS2H
RD_POS1L
RD_POS1H
SAMP_ROT
RD_POS2L
RD_POS2H
RD_POS1L
RD_POS1H
D24
WD_POS1L
WD_POS1H
WD_POS2L
WD_POS2H
CLR_COUNT1
CLR_COUNT2
WD_POS1L
WD_POS1H
WD_POS2L
WD_POS2H
POLARITY1
POLARITY2
CLK20
CLK
INTER
SHEET 2

FREQGEN
CLK20
WD_FREQ
RD_FREQ
RD_OUT
WD_PHASE
RD_COS
D
CLK20
WD_FREQ
RD_FREQ
RD_OUT
SAMPLE
DIR
INC
WD_PHASE
INDEX
CLK
RD_COS
ACCUM

REAL_TIME_CLOCK
CLK20
WD_RTC
RD_RTC
CLR_RTC
CLK20
D
WD_RTC
RD_RTC
CLR_RTC
RTC_TIME
DAC_SHIFT
SAMP_SPIN
POLARITY1
POLARITY2
POLARITY3
CLK
RTC
SHEET 10

DSP_INTERFACE
D
CLK
CLK5
CLK20
RD
WD
RDB
SHEET 9
DSP_IFC

INTERRUPT_CONTROLLER
CLK20
IRQ_EN
CLEAR_IRQ
RD_IRQ
RD_LEV
D
CLK20
CLK
IRQ_EN
CLEAR_IRQ
RD_IRQ
RD_LEV
CLR_POS1
CLR_POS2
CLR_POS3
DSP_IRQ
SHEET 6

RD_ID
U3
BX1
OE ACTIVE LOW
BX2
INV
I
O
FORCE
value=16#5aa501#
TRISTATE

DAC_SHIFT

SPINDLE
CLOCK5
RD_POS3L
RD_POS3H
CLK
D
RD_POSL
RD_POSH
SAMPLE
POLARITY
CLR_POS
INC_OUT
DIR_OUT
INDEX_OUT
SPINDLE

CLK20
WDB
U2
XBLOX_BUS
E00 IRQ_EN $FFC0
E01 CLEAR_IRQ $FFC1
E02 WD_RTC $FFC2
E03 WD_DAC $FFC3
E04 WD_POS_L $FFC4
E05 WD_POS_H $FFC5
E06 WD_MAP $FFC6
E07 $FFC7
E08 WD_CNTR $FFC8
E09 $FFC9
E10 $FFCA
E11 $FFCB
E12 WD_PHASE $FFCC
E13 CLR_RTC $FFCD
E14 WD_FREQ $FFCE
E15 WD_FOLLOW $FFDF
BUS_IF16

WD_MAP
WD_D
WD_PL
WD_PH
WD_MAP
D
RD_MAP
CLK20
WD_D1
WD_P1L
WD_P1H
WD_D2
WD_P2L
WD_P2H
WD_D3
WD_P3L
WD_P3H
MAP
WD_DAC1
WD_POS1L
WD_POS1H
WD_DAC2
WD_POS2L
WD_POS2H
WD_DAC3
WD_POS3L
WD_POS3H
WD_MAP

16_BIT_DAC_INTERFACE
DAC_SHIFT
WD_DAC1
WD_DAC3
WD_DAC2
WD_FOLLOW
RD_DAC1
RD_DAC3
RD_DAC2
RD_FOLLOW
CLK20
CLK
D
WD_DAC1
WD_FOL1
WD_DAC2
WD_FOL2
RD_DAC1
RD_FOL1
RD_DAC2
RD_FOL2
CLK20
CLK
RTC_TIME
RTC_TIME
DAC_IFC
SHEET 5

U1
XBLOX_BUS
E00 RD_IRQ $FFC0
E01 RD_LEV $FFC1
E02 RD_RTC $FFC2
E03 RD_DAC $FFC3
E04 RD_POS_L $FFC4
E05 RD_POS_H $FFC5
E06 RD_MAP $FFC6
E07 RD_ID $ffc7
E08 RD_CNTR $ffc8
E09 $FFC9
E10 $FFCA
E11 $FFCB
E12 RD_COS $FFCC
E13 RD_OUT $FFCD
E14 RD_FREQ $FFCE
E15 RD_FOLLOW $FFDF
BUS_IF16

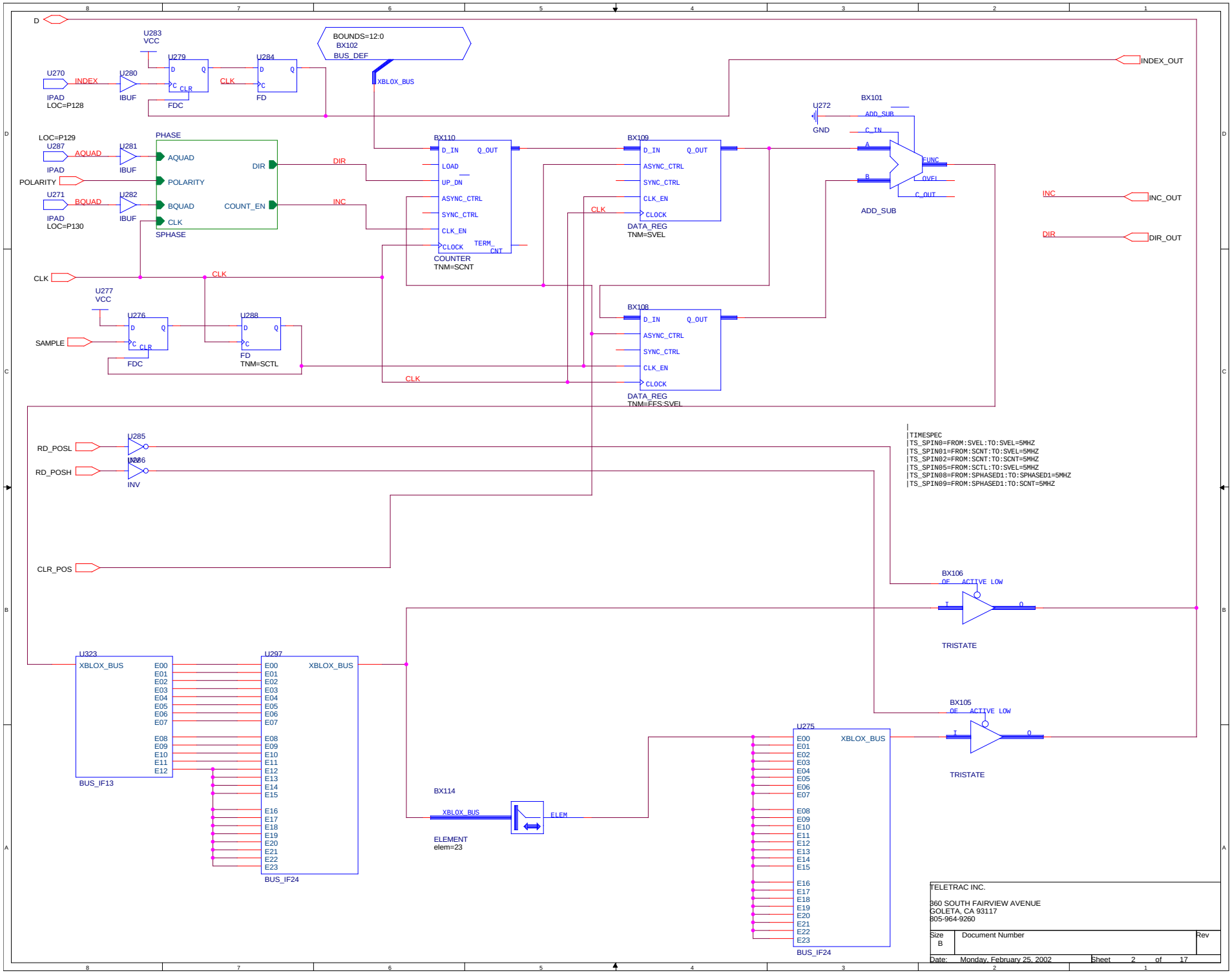
RD_MAP
MAP
RD_DAC
RD_PL
RD_PH
RD_D1
RD_P1L
RD_P1H
RD_D2
RD_P2L
RD_P2H
RD_D3
RD_P3L
RD_P3H
RD_DAC1
RD_POS1L
RD_POS1H
RD_DAC2
RD_POS2L
RD_POS2H
RD_DAC3
RD_POS3L
RD_POS3H
1
0
2
RD_MAP

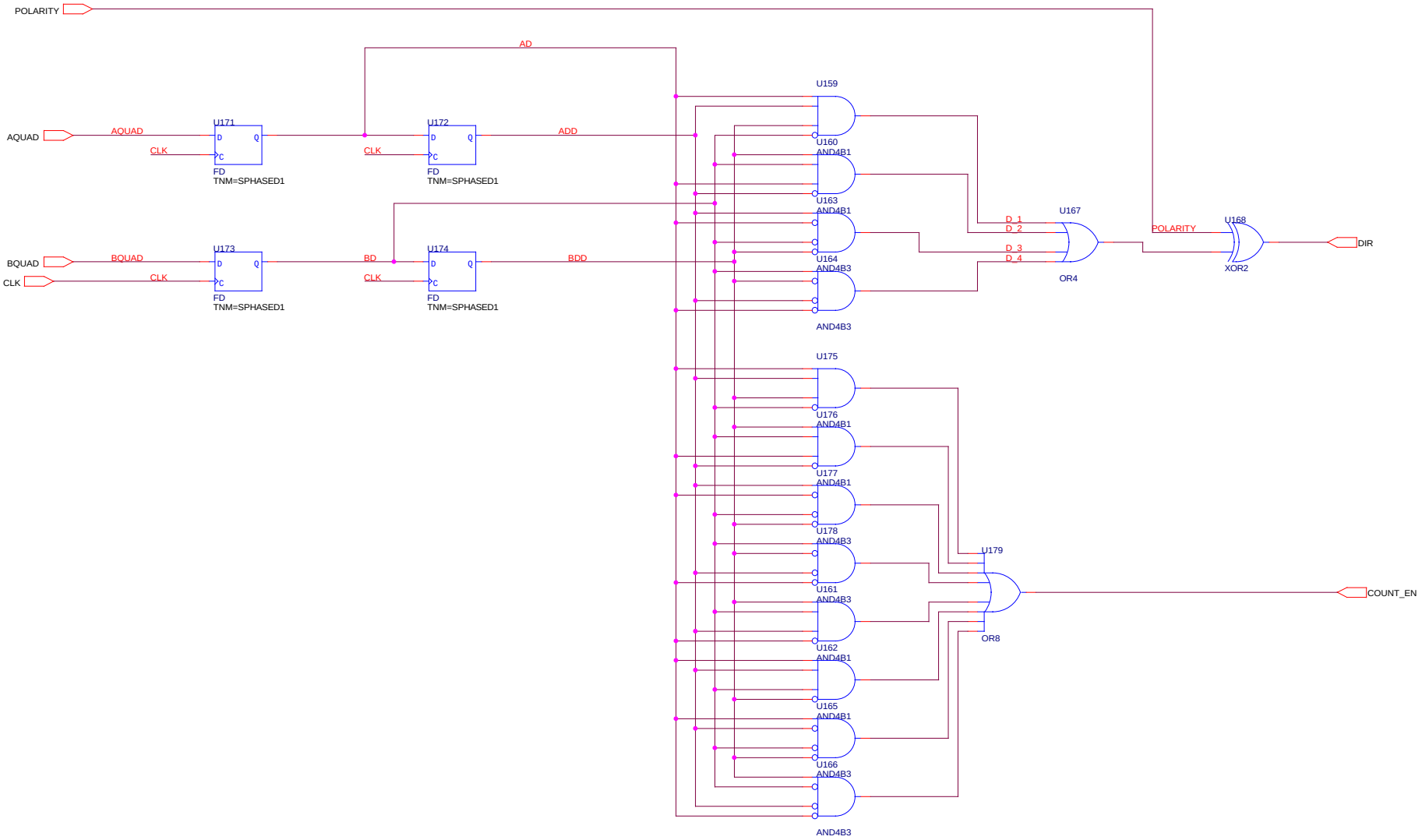
AXSYS TECHNOLOGIES
137 AERO CAMINO
Goleta, CA 93117
805-968-4333

Title
SERVO GATE ARRAY FOR SPINDLE CONTROLLER

| Size | Document Number | Rev |
|---|---|---|
| B | | C |

Date: Monday, February 25, 2002 Sheet 1 of 17


D
U283
VCC
BOUNDS=12:0
BX102
BUS_DEF
XBLOX_BUS
INDEX_OUT
U270
INDEX
IPAD
LOC=P128
U280
IBUF
U279
FDC
CLR
U284
FD
CLK
U272
GND
BX101
ADD_SUB
C_IN
A
B
FUNC
OVFL
C_OUT
ADD_SUB
LOC=P129
U287
AQUAD
IPAD
POLARITY
U271
BQUAD
IPAD
LOC=P130
U281
IBUF
U282
IBUF
PHASE
AQUAD
POLARITY
BQUAD
CLK
DIR
COUNT_EN
SPHASE
DIR
INC
BX110
D_IN
Q_OUT
LOAD
UP_DN
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
TERM_CNT
COUNTER
TNM=SCNT
BX109
D_IN
Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
DATA_REG
TNM=SVEL
INC
INC_OUT
DIR
DIR_OUT
CLK
U277
VCC
U276
FDC
SAMPLE
U288
FD
TNM=SCTL
BX108
D_IN
Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
DATA_REG
TNM=FFS:SVEL
RD_POSL
RD_POSH
U285
U286
INV
TIMESPEC
TS_SPIN0=FROM:SVEL:TO:SVEL=5MHZ
TS_SPIN01=FROM:SCNT:TO:SVEL=5MHZ
TS_SPIN02=FROM:SCNT:TO:SCNT=5MHZ
TS_SPIN05=FROM:SCTL:TO:SVEL=5MHZ
TS_SPIN08=FROM:SPHASED1:TO:SPHASED1=5MHZ
TS_SPIN09=FROM:SPHASED1:TO:SCNT=5MHZ
CLR_POS
BX106
OE ACTIVE LOW
TRISTATE
U323
XBLOX_BUS
BUS_IF13
U297
XBLOX_BUS
BUS_IF24
BX114
XBLOX_BUS
ELEM
ELEMENT
elem=23
BX105
OE ACTIVE LOW
TRISTATE
U275
XBLOX_BUS
BUS_IF24
TELETRAC INC.
360 SOUTH FAIRVIEW AVENUE
GOLETA, CA 93117
805-964-9260
Size
B
Document Number
Rev
Date: Monday, February 25, 2002
Sheet 2 of 17



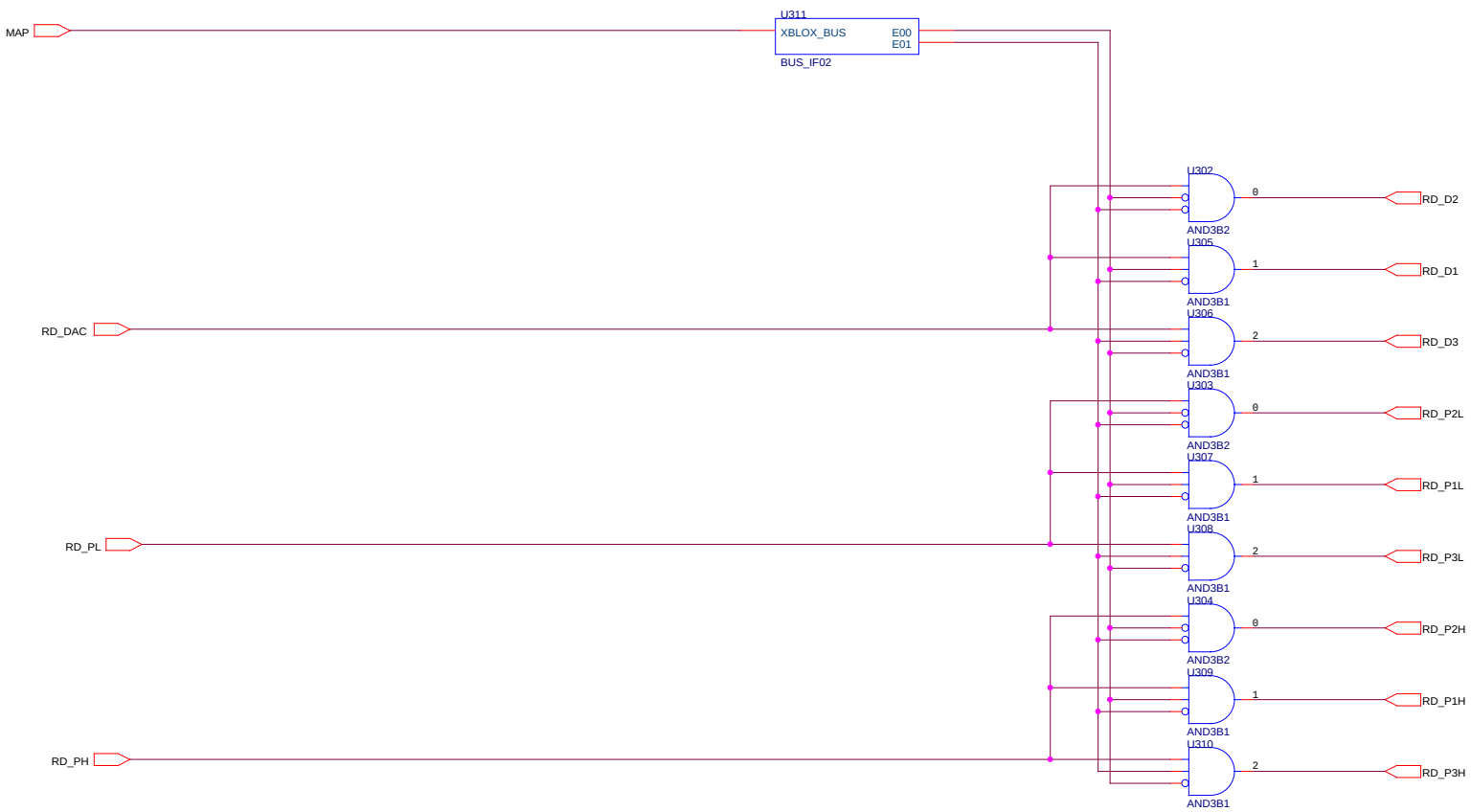

PHASE DETECTOR

TELETRAC INC.

360 SOUTH FAIRVIEW AVENUE
GOLETA, CA 93117
805-964-9260

| Size | Document Number | Rev |
|---|---|---|
| B | | |

Date: Monday, February 25, 2002 Sheet 3 of 17

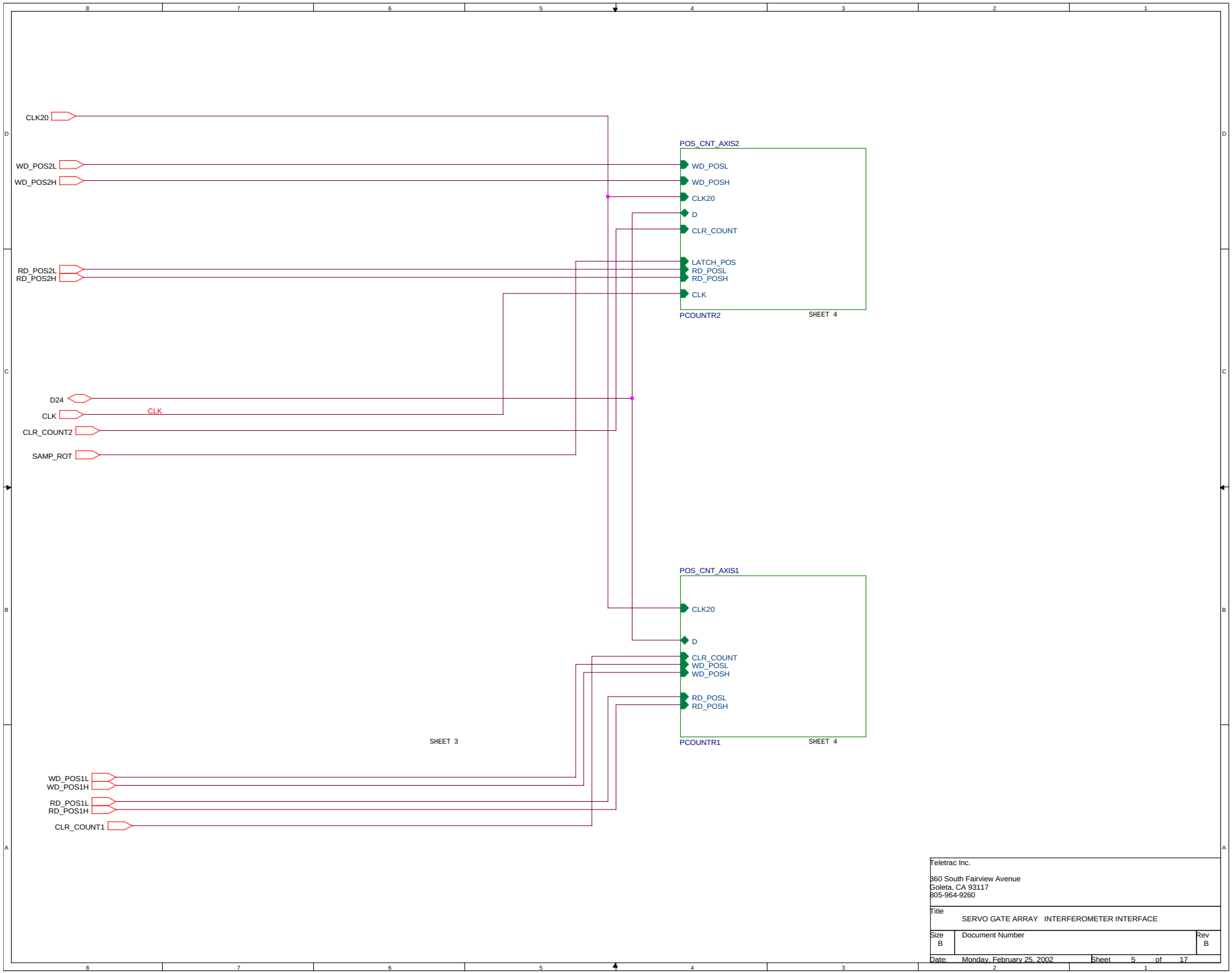



TELETRAC INC.

360 SOUTH FAIRVIEW AVENUE
GOLETA, CA 93117
805-964-9260

| Size | Document Number | Rev |
|---|---|---|
| B | | |

Date: Monday, February 25, 2002 Sheet 4 of 17



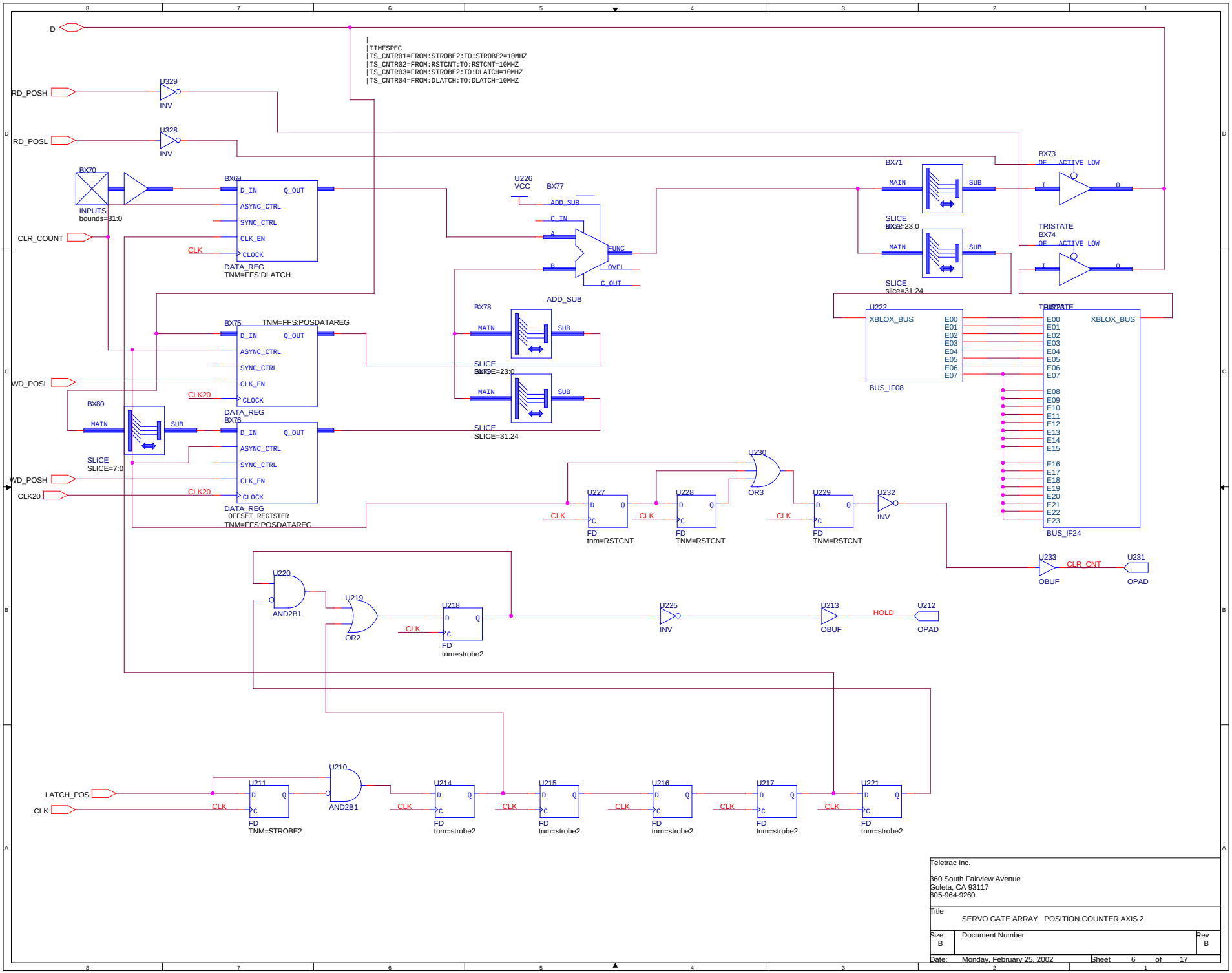

Teletrac Inc.

360 South Fairview Avenue
Goleta, CA 93117
805-964-9260

Title: SERVO GATE ARRAY INTERFEROMETER INTERFACE

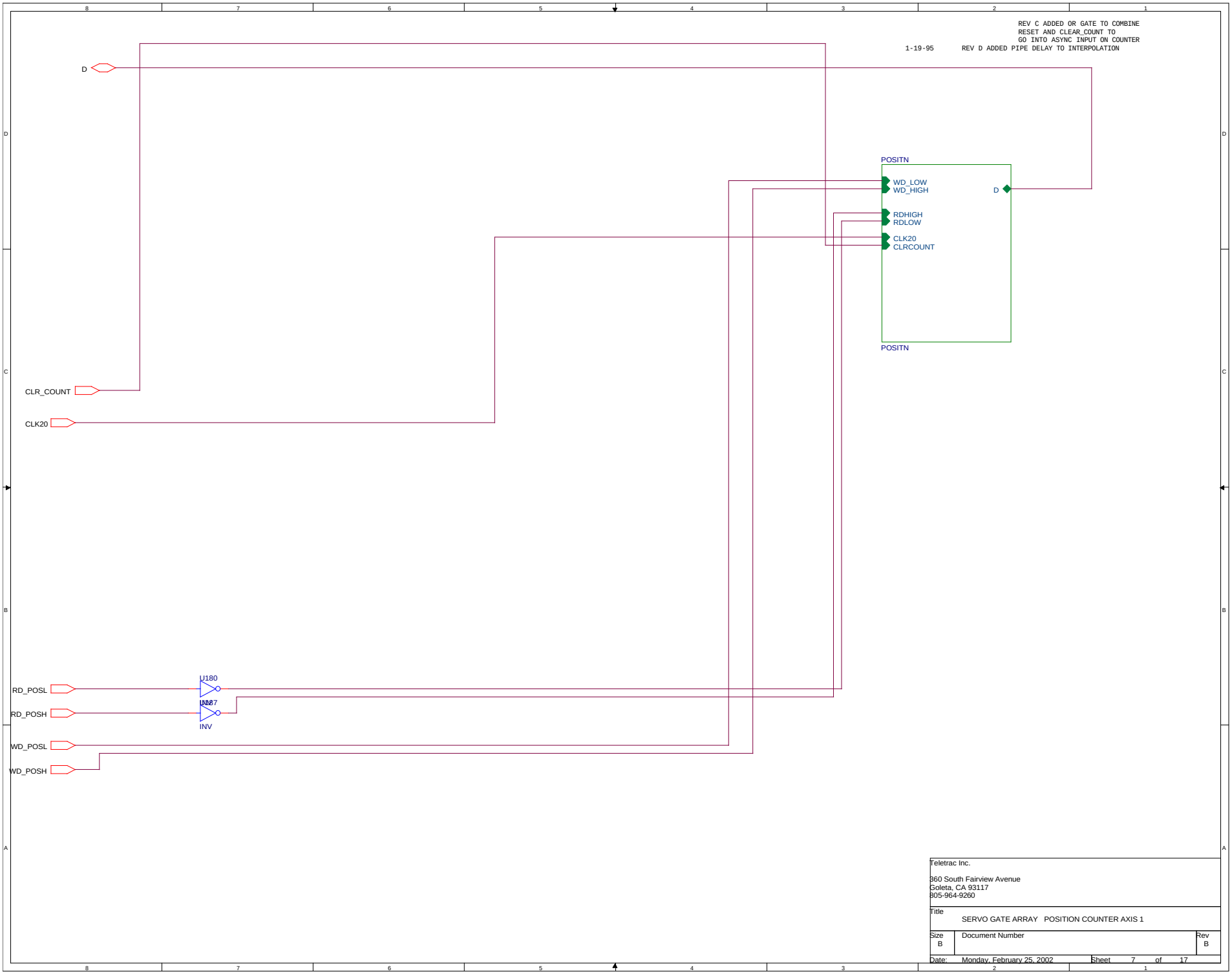
| Size | Document Number | Rev |
|---|---|---|
| B | | B |

Date: Monday, February 25, 2002 Sheet 5 of 17


D
|
|TIMESPEC
|TS_CNTR01=FROM:STROBE2:TO:STROBE2=10MHZ
|TS_CNTR02=FROM:RSTCNT:TO:RSTCNT=10MHZ
|TS_CNTR03=FROM:STROBE2:TO:DLATCH=10MHZ
|TS_CNTR04=FROM:DLATCH:TO:DLATCH=10MHZ
RD_POSH
U329
INV
RD_POSL
U328
INV
BX70
INPUTS
bounds=31:0
CLR_COUNT
BX69
D_IN
Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
CLK
DATA_REG
TNM=FFS:DLATCH
U226
VCC
BX77
ADD_SUB
C_IN
A
B
FUNC
OVFL
C_OUT
BX71
MAIN
SUB
SLICE
SLICE=23:0
BX73
OE
ACTIVE LOW
T
O
TRISTATE
BX74
BX78
SLICE=31:24
BX75
TNM=FFS:POSDATAREG
WD_POSL
CLK20
BX80
SLICE
SLICE=7:0
BX76
OFFSET REGISTER
TNM=FFS:POSDATAREG
U222
XBLOX_BUS
BUS_IF08
E00
E01
E02
E03
E04
E05
E06
E07
E08
E09
E10
E11
E12
E13
E14
E15
E16
E17
E18
E19
E20
E21
E22
E23
BUS_IF24
WD_POSH
CLK20
U227
U228
U229
U230
OR3
U232
INV
FD
tnm=RSTCNT
TNM=RSTCNT
U233
OBUF
CLR_CNT
U231
OPAD
U220
AND2B1
U219
OR2
U218
tnm=strobe2
U225
U213
HOLD
U212
LATCH_POS
U211
TNM=STROBE2
U210
U214
U215
U216
U217
U221
Teletrac Inc.
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260
Title
SERVO GATE ARRAY POSITION COUNTER AXIS 2
Size
B
Document Number
Rev
B
Date: Monday, February 25, 2002
Sheet 6 of 17

REV C ADDED OR GATE TO COMBINE
RESET AND CLEAR_COUNT TO
GO INTO ASYNC INPUT ON COUNTER

1-19-95 REV D ADDED PIPE DELAY TO INTERPOLATION



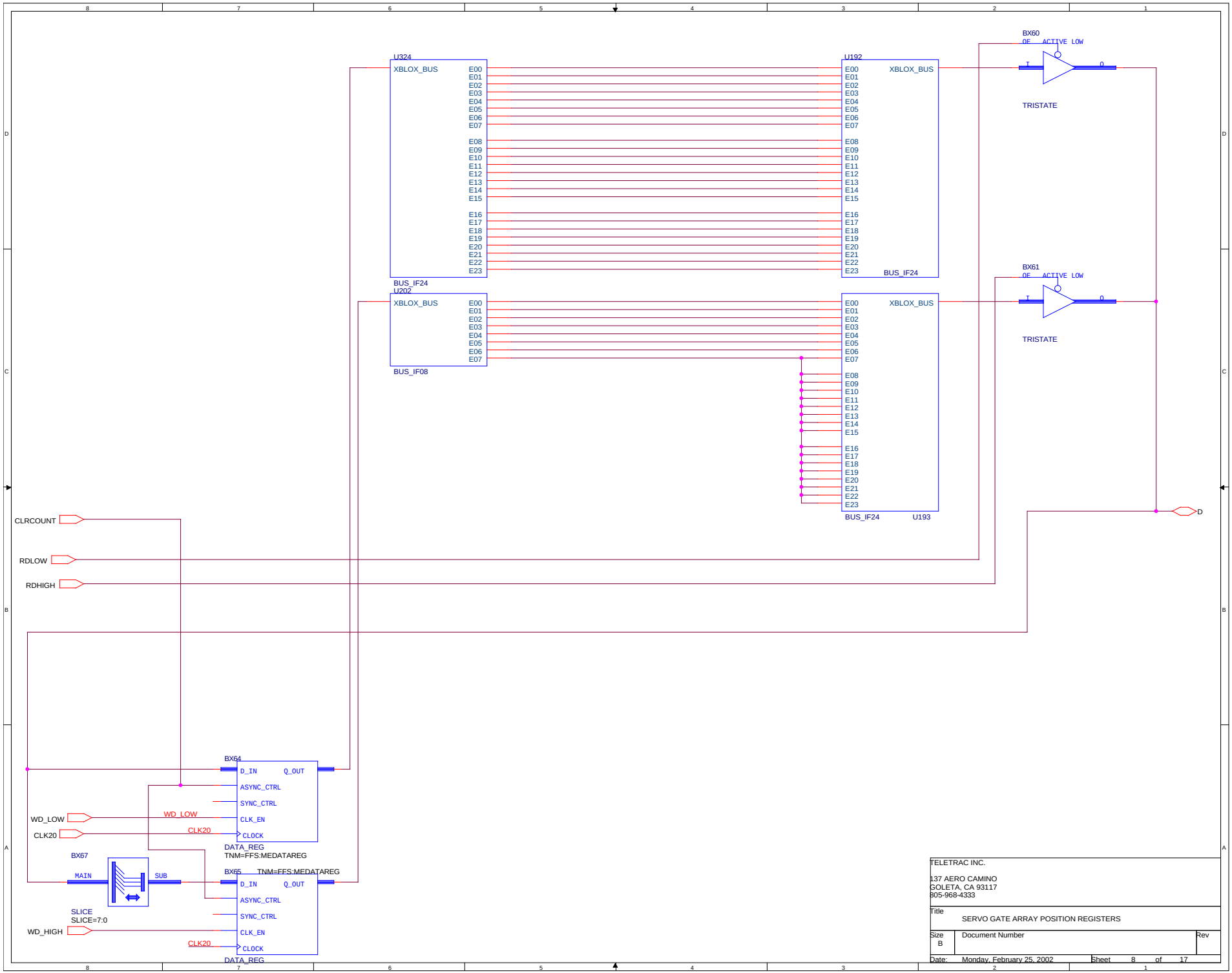

Teletrac Inc.

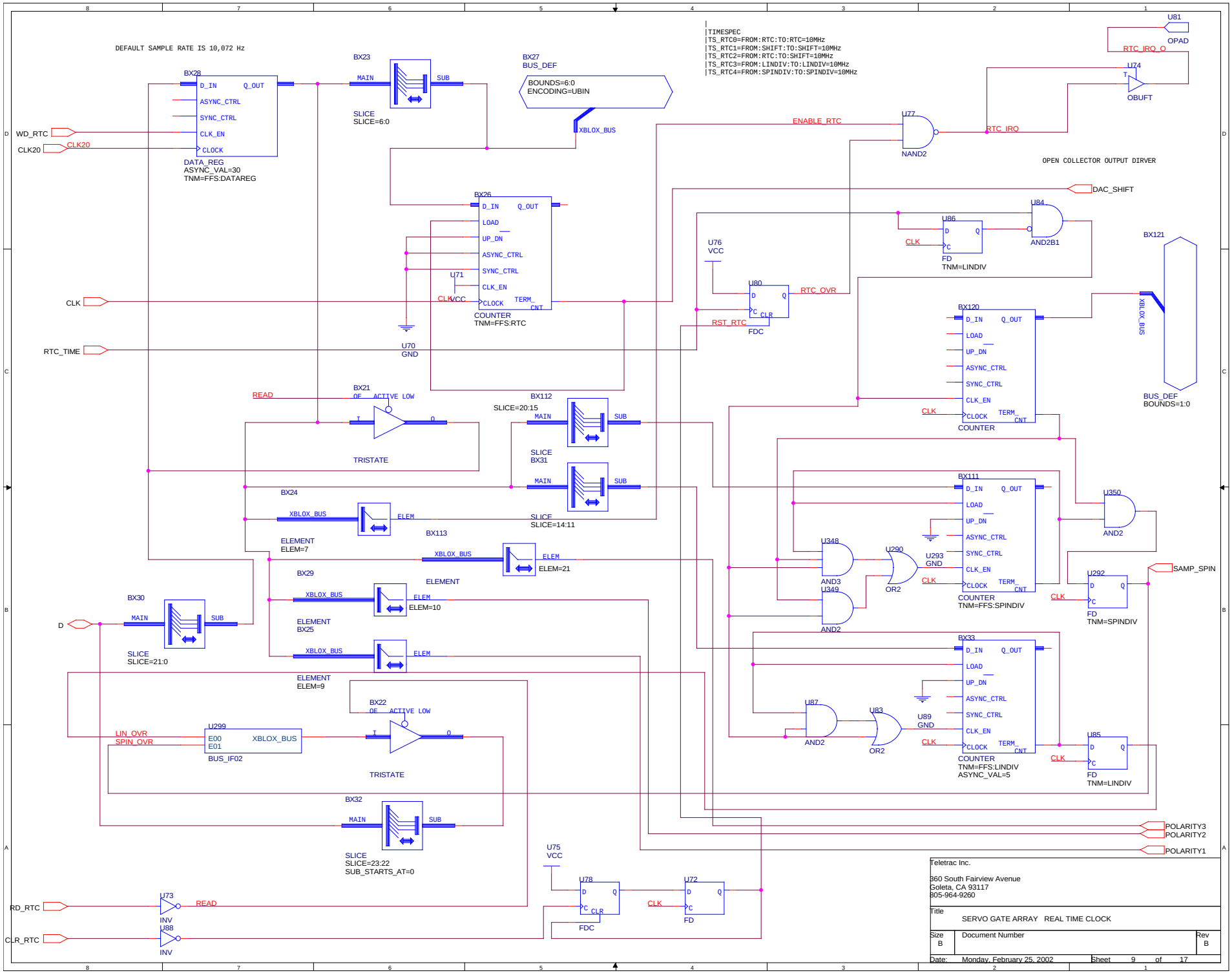
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260

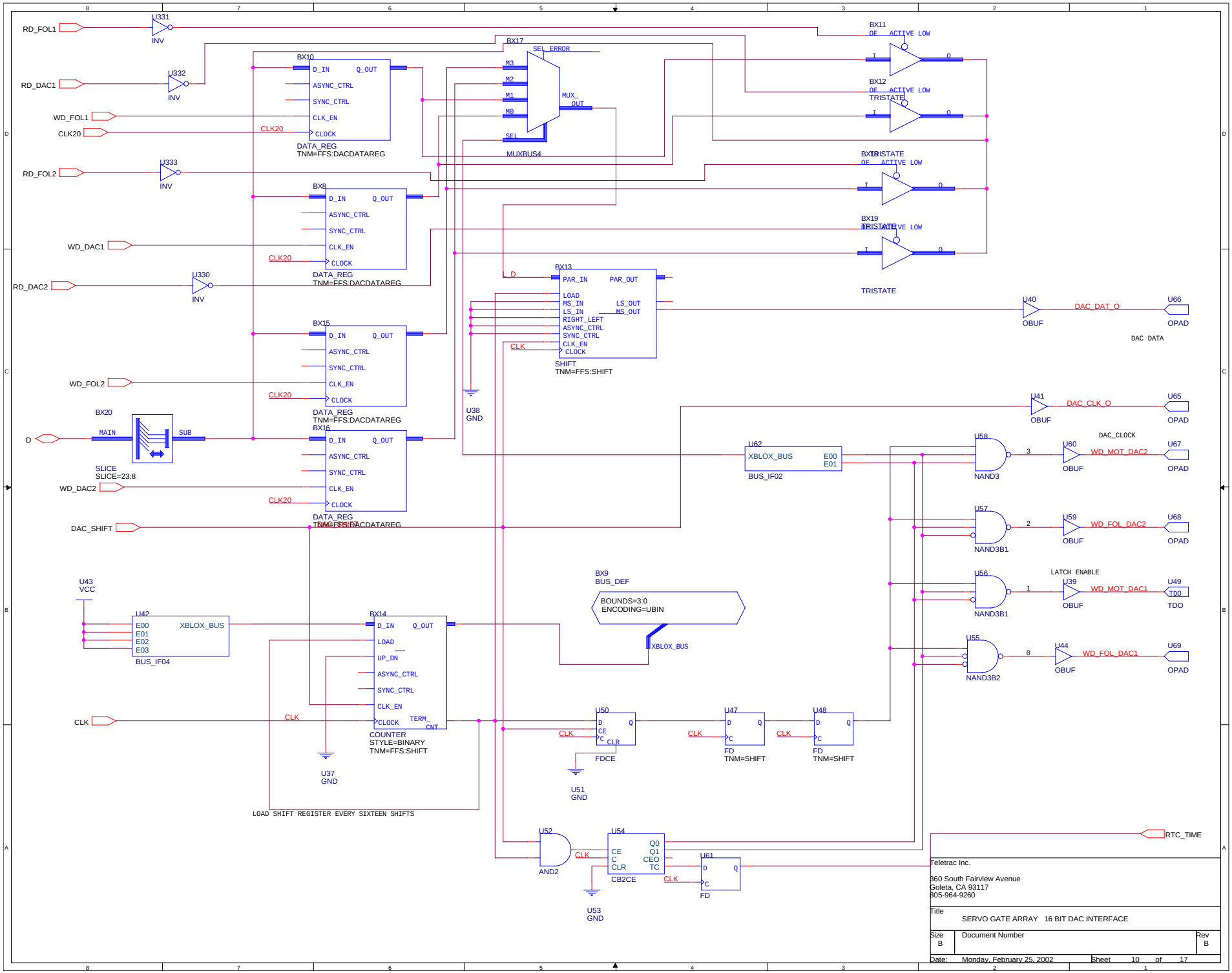
Title: SERVO GATE ARRAY POSITION COUNTER AXIS 1

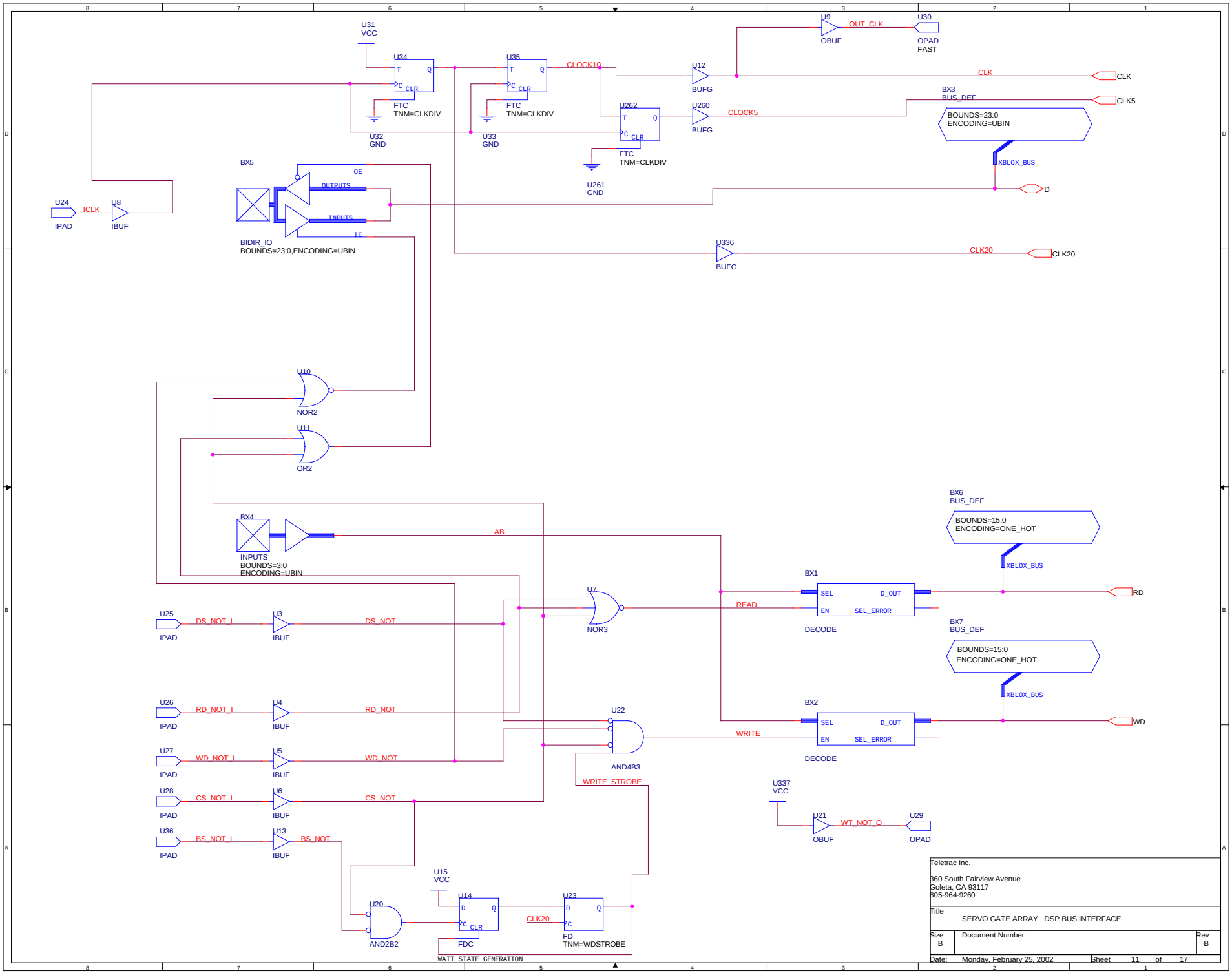
| Size | Document Number | Rev |
|---|---|---|
| B | | B |

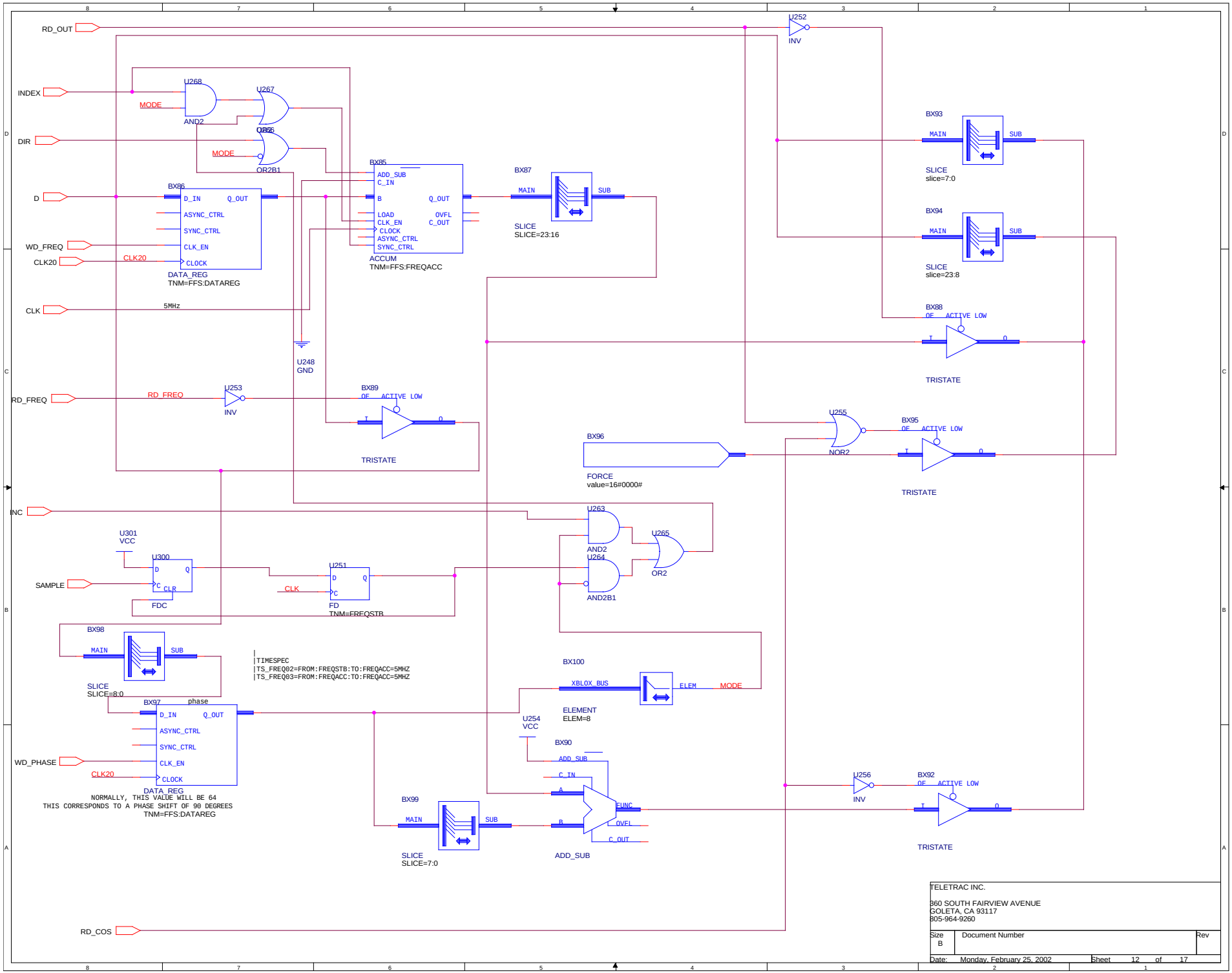
Date: Monday, February 25, 2002 Sheet 7 of 17

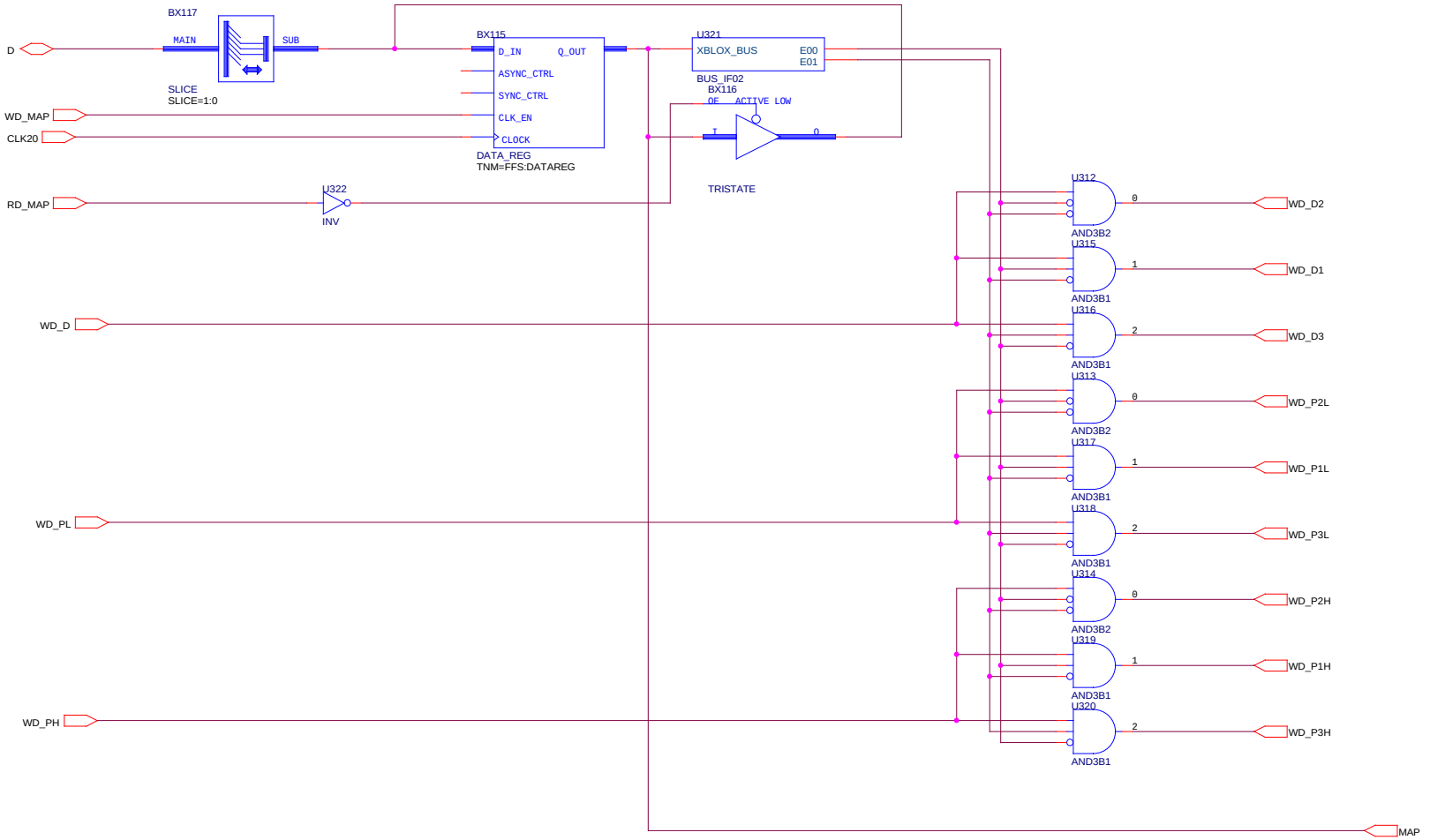

U324
XBLOX_BUS
E00 E01 E02 E03 E04 E05 E06 E07 E08 E09 E10 E11 E12 E13 E14 E15 E16 E17 E18 E19 E20 E21 E22 E23
BUS_IF24
U192
XBLOX_BUS
BUS_IF24
BX60
OE ACTIVE LOW
I O
TRISTATE
U202
XBLOX_BUS
BUS_IF08
BX61
OE ACTIVE LOW
I O
TRISTATE
XBLOX_BUS
BUS_IF24
U193
D
CLRCOUNT
RDLOW
RDHIGH
BX64
D_IN Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
WD_LOW
CLK20
DATA_REG
TNM=FFS:MEDATAREG
BX67
MAIN SUB
SLICE
SLICE=7:0
BX65 TNM=FFS:MEDATAREG
WD_HIGH
DATA_REG
TELETRAC INC.
137 AERO CAMINO
GOLETA, CA 93117
805-968-4333
Title
SERVO GATE ARRAY POSITION REGISTERS
Size B
Document Number
Rev
Date: Monday, February 25, 2002
Sheet 8 of 17


DEFAULT SAMPLE RATE IS 10,072 Hz
TIMESPEC
TS_RTC0=FROM:RTC:TO:RTC=10MHz
TS_RTC1=FROM:SHIFT:TO:SHIFT=10MHz
TS_RTC2=FROM:RTC:TO:SHIFT=10MHz
TS_RTC3=FROM:LINDIV:TO:LINDIV=10MHz
TS_RTC4=FROM:SPINDIV:TO:SPINDIV=10MHz
BX28
DATA_REG
ASYNC_VAL=30
TNM=FFS:DATAREG
BX23
SLICE
SLICE=6:0
BX27
BUS_DEF
BOUNDS=6:0
ENCODING=UBIN
XBLOX_BUS
BX26
COUNTER
TNM=FFS:RTC
U71
U70
GND
U76
VCC
U80
FDC
RST_RTC
RTC_OVR
U77
NAND2
ENABLE_RTC
RTC_IRQ
U81
OPAD
RTC_IRQ_O
U74
OBUFT
OPEN COLLECTOR OUTPUT DIRVER
DAC_SHIFT
U84
AND2B1
U86
FD
TNM=LINDIV
BX121
BUS_DEF
BOUNDS=1:0
BX120
COUNTER
BX21
TRISTATE
BX112
SLICE=20:15
SLICE
BX31
BX24
SLICE
SLICE=14:11
BX113
ELEMENT
ELEM=7
ELEM=21
BX29
ELEMENT
ELEM=10
ELEMENT
BX25
ELEMENT
ELEM=9
BX30
SLICE
SLICE=21:0
BX22
U299
BUS_IF02
TRISTATE
BX32
SLICE
SLICE=23:22
SUB_STARTS_AT=0
BX111
COUNTER
TNM=FFS:SPINDIV
U348
AND3
U349
AND2
U290
OR2
U293
GND
U350
AND2
U292
FD
TNM=SPINDIV
SAMP_SPIN
BX33
COUNTER
TNM=FFS:LINDIV
ASYNC_VAL=5
U87
AND2
U83
OR2
U89
GND
U85
FD
TNM=LINDIV
U75
VCC
U78
FDC
U72
FD
U73
INV
U88
INV
WD_RTC
CLK20
CLK
RTC_TIME
READ
LIN_OVR
SPIN_OVR
RD_RTC
CLR_RTC
D
POLARITY3
POLARITY2
POLARITY1
Teletrac Inc.
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260
Title
SERVO GATE ARRAY REAL TIME CLOCK
Size B
Document Number
Rev B
Date: Monday, February 25, 2002
Sheet 9 of 17


RD_FOL1
U331
INV
RD_DAC1
U332
INV
WD_FOL1
CLK20
RD_FOL2
U333
INV
WD_DAC1
RD_DAC2
U330
INV
WD_FOL2
D
BX20
MAIN
SUB
SLICE
SLICE=23:8
WD_DAC2
DAC_SHIFT
BX10
D_IN
Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
CLK20
DATA_REG
TNM=FFS:DACDATAREG
BX8
BX15
BX16
BX17
SEL_ERROR
M3
M2
M1
M0
SEL
MUX_OUT
MUXBUS4
BX11
OE_ACTIVE LOW
BX12
TRISTATE
BX18
BX19
T
O
BX13
PAR_IN
PAR_OUT
LOAD
MS_IN
LS_IN
LS_OUT
MS_OUT
RIGHT_LEFT
CLK
SHIFT
U38
GND
U40
OBUF
DAC_DAT_O
U66
OPAD
DAC DATA
U41
DAC_CLK_O
U65
DAC_CLOCK
U62
XBLOX_BUS
E00
E01
BUS_IF02
U58
NAND3
U60
WD_MOT_DAC2
U67
U57
NAND3B1
U59
WD_FOL_DAC2
U68
U56
LATCH ENABLE
U39
WD_MOT_DAC1
U49
TDO
U55
NAND3B2
U44
WD_FOL_DAC1
U69
U43
VCC
U42
E02
E03
BUS_IF04
BX14
UP_DN
TERM_CNT
COUNTER
STYLE=BINARY
TNM=FFS:SHIFT
U37
GND
LOAD SHIFT REGISTER EVERY SIXTEEN SHIFTS
BX9
BUS_DEF
BOUNDS=3:0
ENCODING=UBIN
U50
CE
C
CLR
FDCE
U51
U47
FD
U48
U52
AND2
U54
Q0
Q1
CEO
TC
CB2CE
U53
U61
RTC_TIME
Teletrac Inc.
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260
Title
SERVO GATE ARRAY 16 BIT DAC INTERFACE
Size
B
Document Number
Rev
B
Date: Monday, February 25, 2002
Sheet 10 of 17


U31
VCC
U34
T
Q
C
CLR
FTC
TNM=CLKDIV
U32
GND
U35
CLOCK10
U33
GND
U262
U261
GND
U9
OBUF
OUT_CLK
U30
OPAD
FAST
U12
BUFG
CLK
CLK
BX3
BUS_DEF
CLK5
U260
CLOCK5
BOUNDS=23:0
ENCODING=UBIN
XBLOX_BUS
D
U24
IPAD
ICLK
U8
IBUF
BX5
0E
OUTPUTS
INPUTS
TE
BIDIR_IO
BOUNDS=23:0,ENCODING=UBIN
U336
BUFG
CLK20
CLK20
U10
NOR2
U11
OR2
BX4
INPUTS
BOUNDS=3:0
ENCODING=UBIN
AB
BX6
BUS_DEF
BOUNDS=15:0
ENCODING=ONE_HOT
XBLOX_BUS
RD
BX1
SEL
D_OUT
EN
SEL_ERROR
READ
DECODE
U7
NOR3
BX7
BUS_DEF
BOUNDS=15:0
ENCODING=ONE_HOT
XBLOX_BUS
WD
BX2
SEL
D_OUT
EN
SEL_ERROR
WRITE
DECODE
U22
AND4B3
WRITE_STROBE
U25
IPAD
DS_NOT_I
U3
IBUF
DS_NOT
U26
IPAD
RD_NOT_I
U4
IBUF
RD_NOT
U27
IPAD
WD_NOT_I
U5
IBUF
WD_NOT
U28
IPAD
CS_NOT_I
U6
IBUF
CS_NOT
U36
IPAD
BS_NOT_I
U13
IBUF
BS_NOT
U337
VCC
U21
OBUF
WT_NOT_O
U29
OPAD
U15
VCC
U20
AND2B2
U14
D
Q
C
CLR
FDC
CLK20
U23
FD
TNM=WDSTROBE
WAIT STATE GENERATION
Teletrac Inc.
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260
Title
SERVO GATE ARRAY DSP BUS INTERFACE
Size
B
Document Number
Rev
B
Date: Monday, February 25, 2002
Sheet 11 of 17


RD_OUT
U252
INV
INDEX
U268
AND2
MODE
U267
OR2
DIR
MODE
U266
OR2B1
BX85
ADD_SUB
C_IN
B
Q_OUT
LOAD
OVFL
CLK_EN
C_OUT
CLOCK
ASYNC_CTRL
SYNC_CTRL
ACCUM
TNM=FFS:FREQACC
BX86
D_IN
Q_OUT
D
ASYNC_CTRL
SYNC_CTRL
CLK_EN
WD_FREQ
CLK20
CLOCK
DATA_REG
TNM=FFS:DATAREG
BX87
MAIN
SUB
SLICE
SLICE=23:16
BX93
SLICE
slice=7:0
BX94
SLICE
slice=23:8
CLK
5MHz
U248
GND
BX88
OE ACTIVE LOW
I
O
TRISTATE
RD_FREQ
U253
INV
BX89
TRISTATE
U255
NOR2
BX95
TRISTATE
BX96
FORCE
value=16#0000#
INC
U263
AND2
U265
OR2
U264
AND2B1
U301
VCC
U300
D
Q
C
CLR
FDC
SAMPLE
U251
FD
TNM=FREQSTB
CLK
BX98
SLICE
SLICE=8:0
TIMESPEC
TS_FREQ02=FROM:FREQSTB:TO:FREQACC=5MHZ
TS_FREQ03=FROM:FREQACC:TO:FREQACC=5MHZ
BX100
XBLOX_BUS
ELEM
MODE
ELEMENT
ELEM=8
BX97
phase
WD_PHASE
DATA_REG
NORMALLY, THIS VALUE WILL BE 64
THIS CORRESPONDS TO A PHASE SHIFT OF 90 DEGREES
TNM=FFS:DATAREG
U254
VCC
BX90
FUNC
BX99
SLICE
SLICE=7:0
ADD_SUB
U256
INV
BX92
TRISTATE
RD_COS
TELETRAC INC.
360 SOUTH FAIRVIEW AVENUE
GOLETA, CA 93117
805-964-9260
Size
B
Document Number
Rev
Date: Monday, February 25, 2002
Sheet 12 of 17

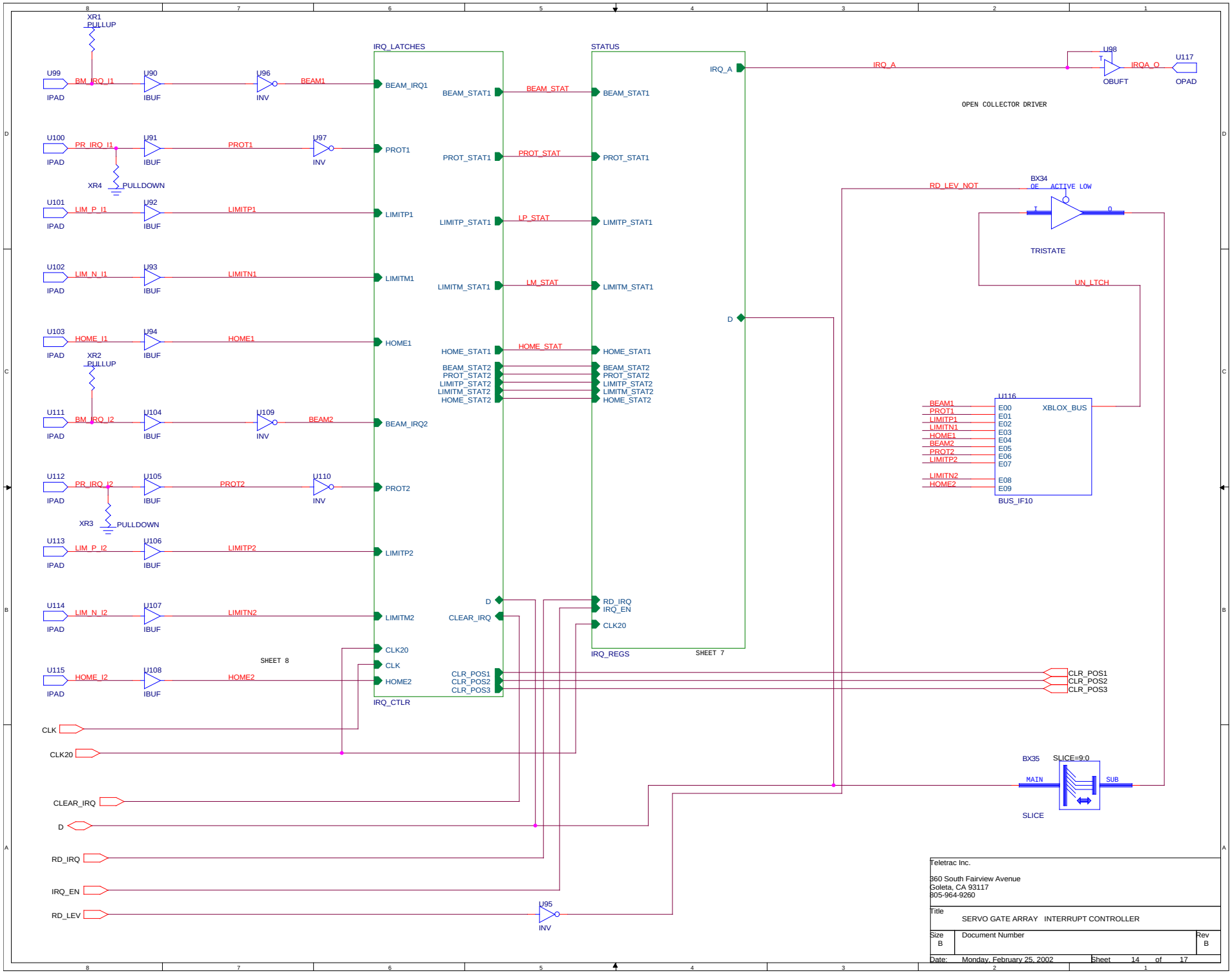


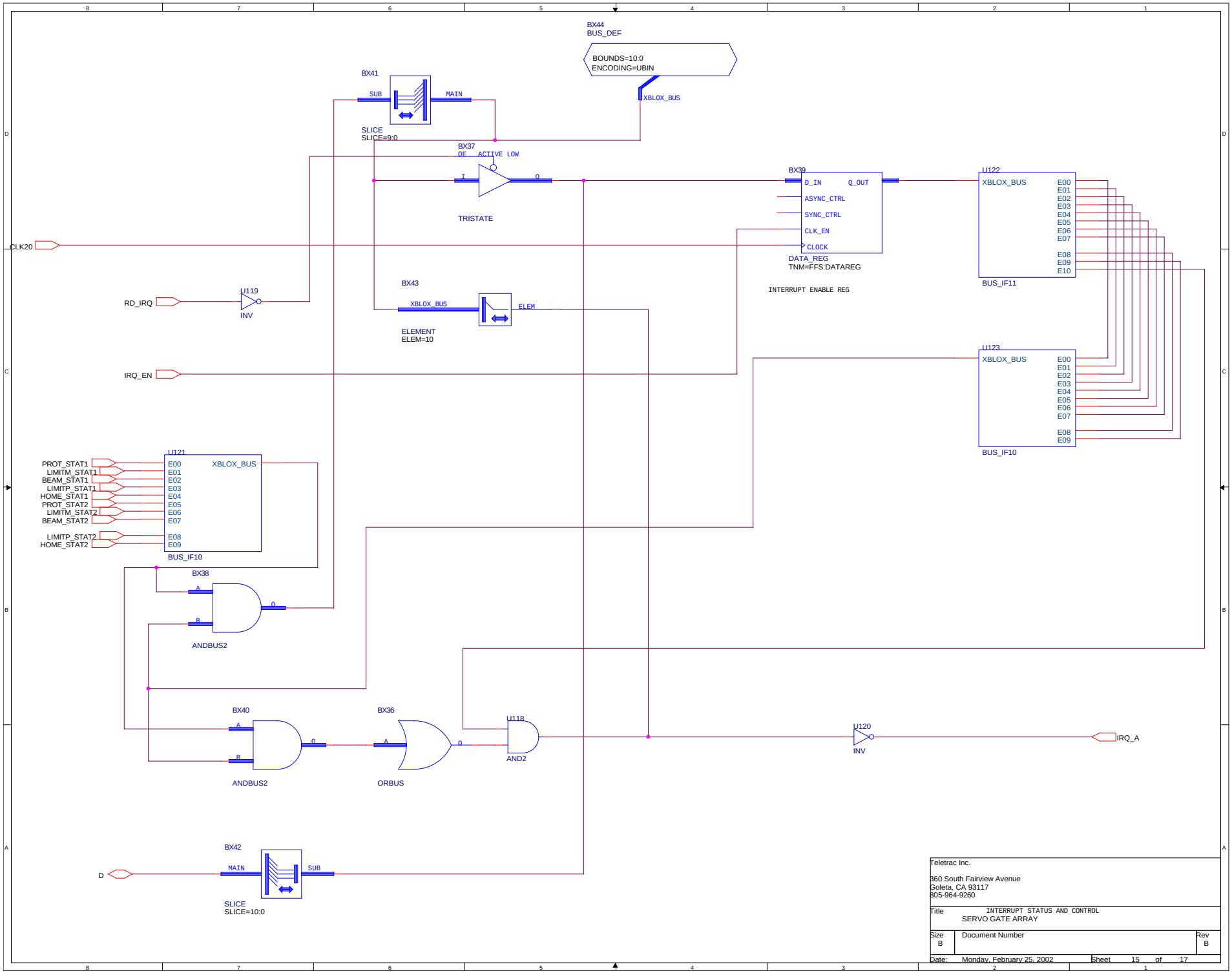

TELETRAC INC.

360 SOUTH FAIRVIEW AVENUE
GOLETA, CA 93117
805-964-9260

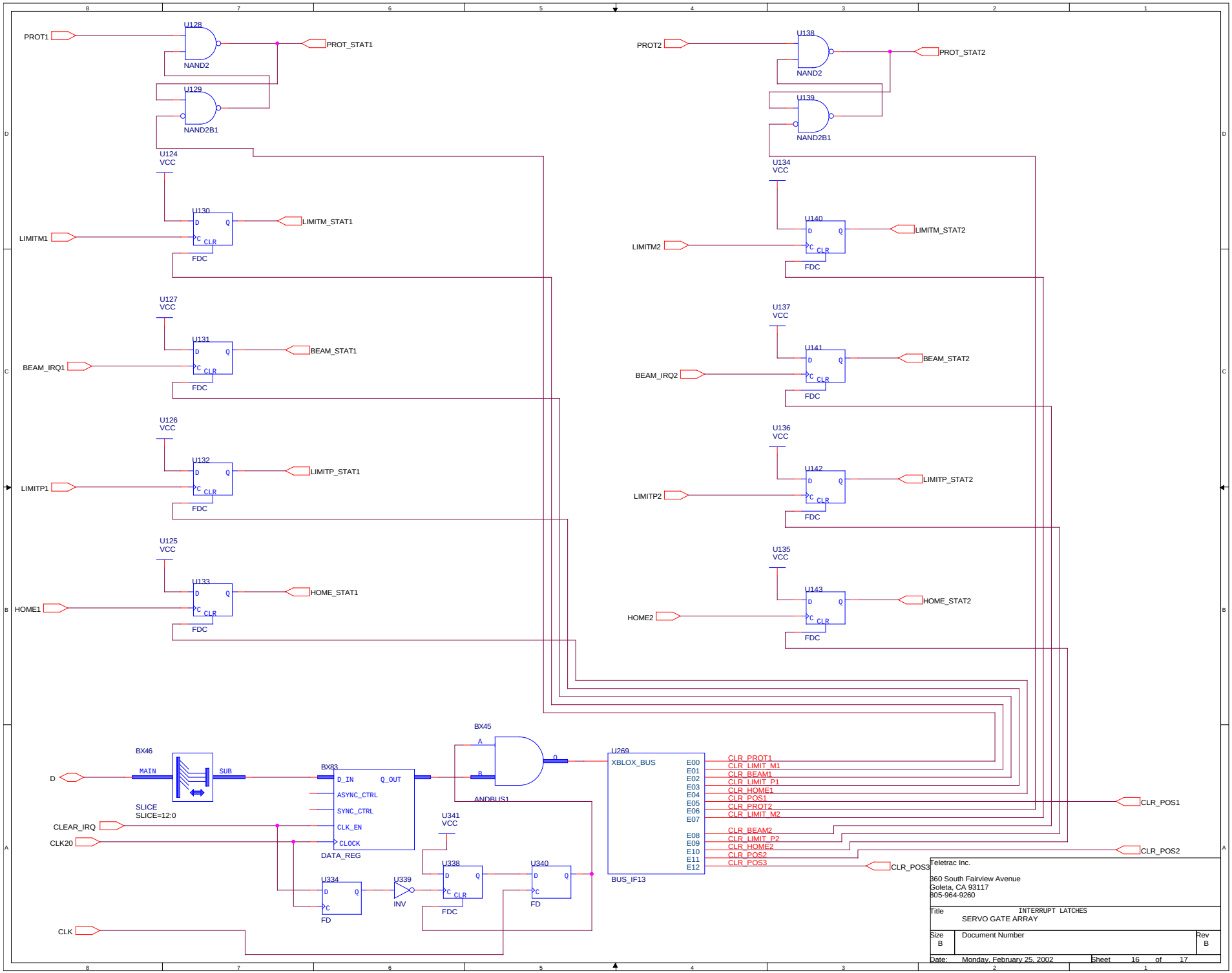
| Size | Document Number | Rev |
|---|---|---|
| B | | |

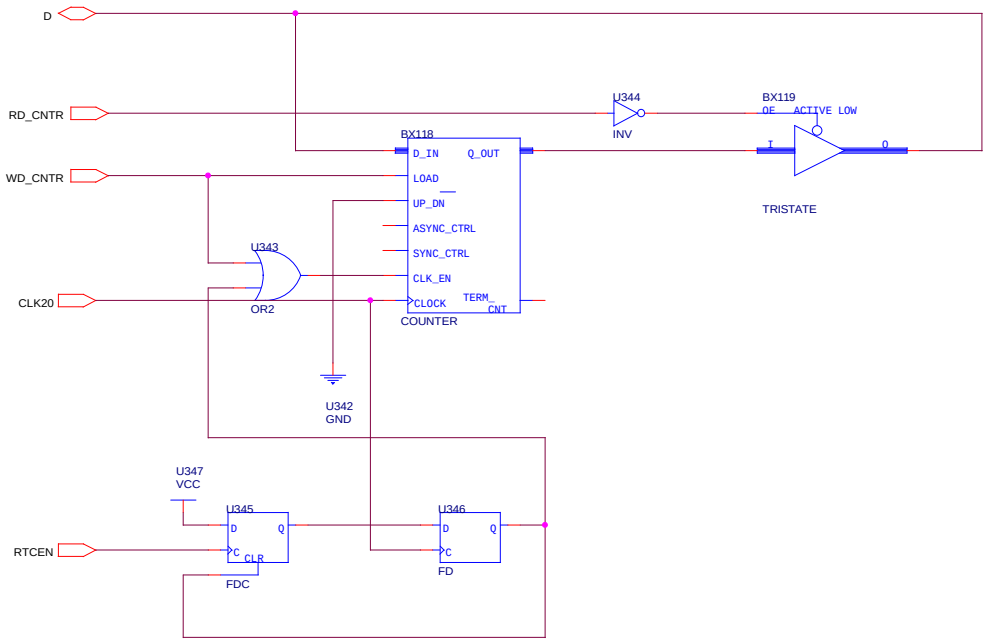
Date: Monday, February 25, 2002 Sheet 13 of 17


XR1
PULLUP
U99
BM_IRQ_I1
IPAD
U90
IBUF
U96
INV
BEAM1
BEAM_IRQ1
U100
PR_IRQ_I1
U91
PROT1
U97
PROT1
XR4
PULLDOWN
U101
LIM_P_I1
U92
LIMITP1
LIMITP1
U102
LIM_N_I1
U93
LIMITN1
LIMITM1
U103
HOME_I1
U94
HOME1
HOME1
XR2
PULLUP
U111
BM_IRQ_I2
U104
U109
BEAM2
BEAM_IRQ2
U112
PR_IRQ_I2
U105
PROT2
U110
PROT2
XR3
PULLDOWN
U113
LIM_P_I2
U106
LIMITP2
LIMITP2
U114
LIM_N_I2
U107
LIMITN2
LIMITM2
SHEET 8
U115
HOME_I2
U108
HOME2
CLK20
CLK
HOME2
IRQ_CTLR
IRQ_LATCHES
BEAM_STAT1
PROT_STAT1
LIMITP_STAT1
LIMITM_STAT1
HOME_STAT1
BEAM_STAT2
PROT_STAT2
LIMITP_STAT2
LIMITM_STAT2
HOME_STAT2
D
CLEAR_IRQ
CLR_POS1
CLR_POS2
CLR_POS3
BEAM_STAT
PROT_STAT
LP_STAT
LM_STAT
HOME_STAT
STATUS
IRQ_A
D
RD_IRQ
IRQ_EN
CLK20
IRQ_REGS
SHEET 7
IRQ_A
U98
T
OBUFT
IRQA_O
U117
OPAD
OPEN COLLECTOR DRIVER
RD_LEV_NOT
BX34
OE ACTIVE LOW
I
O
TRISTATE
UN_LTCH
U116
BEAM1
PROT1
LIMITP1
LIMITN1
HOME1
BEAM2
PROT2
LIMITP2
LIMITN2
HOME2
E00
E01
E02
E03
E04
E05
E06
E07
E08
E09
XBLOX_BUS
BUS_IF10
CLR_POS1
CLR_POS2
CLR_POS3
CLK
CLK20
CLEAR_IRQ
D
RD_IRQ
IRQ_EN
RD_LEV
U95
INV
BX35
SLICE=9:0
MAIN
SUB
SLICE
Teletrac Inc.
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260
Title
SERVO GATE ARRAY INTERRUPT CONTROLLER
Size
B
Document Number
Rev
B
Date: Monday, February 25, 2002
Sheet 14 of 17




| | |
|---|---|
| Teletrac Inc. | |
| 360 South Fairview Avenue<br>Goleta, CA 93117<br>805-964-9260 | |
| Title | INTERRUPT STATUS AND CONTROL<br>SERVO GATE ARRAY |
| Size B | Document Number |
| Rev B | |
| Date: Monday, February 25, 2002 | Sheet 15 of 17 |


PROT1
U128
NAND2
PROT_STAT1
U129
NAND2B1
U124
VCC
U130
D
Q
C
CLR
LIMITM1
LIMITM_STAT1
FDC
PROT2
U138
NAND2
PROT_STAT2
U139
NAND2B1
U134
VCC
U140
LIMITM2
LIMITM_STAT2
FDC
U127
VCC
U131
BEAM_IRQ1
BEAM_STAT1
FDC
U137
VCC
U141
BEAM_IRQ2
BEAM_STAT2
FDC
U126
VCC
U132
LIMITP1
LIMITP_STAT1
FDC
U136
VCC
U142
LIMITP2
LIMITP_STAT2
FDC
U125
VCC
U133
HOME1
HOME_STAT1
FDC
U135
VCC
U143
HOME2
HOME_STAT2
FDC
BX46
MAIN
SUB
SLICE
SLICE=12:0
D
CLEAR_IRQ
CLK20
BX83
D_IN
Q_OUT
ASYNC_CTRL
SYNC_CTRL
CLK_EN
CLOCK
DATA_REG
BX45
A
B
O
ANDBUS1
U341
VCC
U334
U339
INV
U338
U340
FD
FDC
FD
CLK
U269
XBLOX_BUS
BUS_IF13
E00
E01
E02
E03
E04
E05
E06
E07
E08
E09
E10
E11
E12
CLR_PROT1
CLR_LIMIT_M1
CLR_BEAM1
CLR_LIMIT_P1
CLR_HOME1
CLR_POS1
CLR_PROT2
CLR_LIMIT_M2
CLR_BEAM2
CLR_LIMIT_P2
CLR_HOME2
CLR_POS2
CLR_POS3
CLR_POS1
CLR_POS2
Teletrac Inc.
360 South Fairview Avenue
Goleta, CA 93117
805-964-9260
Title
INTERRUPT LATCHES
SERVO GATE ARRAY
Size
B
Document Number
Rev
B
Date:
Monday, February 25, 2002
Sheet
16
of
17




TELETRAC INC.

360 SOUTH FAIRVIEW AVENUE
GOLETA, CA 93117
805-964-9260

| Size | Document Number | Rev |
|---|---|---|
| B | | |

Date: Monday, February 25, 2002 Sheet 17 of 17

```
1 ----------------------------------------------------------------------------------------
2 -- U18---- DSP Glue Logic
3 ----------------------------------------------------------------------------------------
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7
8 entity u18 is port (
9     a:in std_logic_vector (15 downto 6);
10     ds:in std_logic;
11     ps:in std_logic;
12     we:in std_logic;
13     rd:in std_logic;
14     x_not_y:in std_logic;
15     xccs:out std_logic;
16     ramcs:out std_logic;
17     pd:out std_logic
18 );
19 attribute pin_numbers of u18:entity is
20     "a(14):2 "&
21     "a(12):3 "&
22     "a(13):17 "&
23     "a(11):6 "&
24     "a(15):9 "&
25     "ds:8 "&
26     "ps:11 "&
27     "x_not_y:1 "&
28     "we:18 "&
29     "rd:14 "&
30     "pd:19 "&
31     "a(6):5 "&
32     "a(7):4 "&
33     "a(8):16 "&
34     "a(9):15 "&
35     "a(10):7 "&
36     "ramcs:13 "&
37     "xccs:12";
38 end u18;
39
40 architecture archU18 of u18 is
41 begin
42     pd <= '0' when (a(15) = '0') and (ps = '0') else '1';
43     ramcs <= '0' when (a(15) = '0') and ( (ps='0') or (ds='0')) else '1';
44     xccs <= '0' when (a = "1111111111") and (ds='0') else '1';
45 end archU18;
46
```

```
1 ----------------------------------------------------------------------------------------
2 -- priority interrupt controller for the 600-160 rev B board
3 ----------------------------------------------------------------------------------------
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7 use work.priorityenc_pkg.all;
8 use work.decoder_pkg.all;
9
10 entity PRIORITY is port (
11     PI: IN std_logic_vector(6 downto 0);    --priority interrupt inputs
12     IACK:IN std_logic;                      --interrupt acknowledge input
13     ADR:IN std_logic_vector(2 downto 0);    --address input
14     AS: IN std_logic;                       --address strobe
15     PO: INOUT std_logic_vector(2 downto 0); --priority interrupt output
16     IACKO:OUT std_logic_vector(6 downto 0) --interrupt acknowledge outputs
17 );
18 attribute pin_numbers of PRIORITY:entity is
19     "PI(0):1 "&
20     "PI(1):2 "&
21     "PI(2):3 "&
22     "PI(3):4 "&
23     "PI(4):5 "&
24     "PI(5):6 "&
25     "PI(6):7 "&
26     "IACK:8 "&
27     "ADR(0):9 "&
28     "ADR(1):10 "&
29     "ADR(2):11 "&
30     "AS:13 "&
31     "PO(2):14 "&
32     "PO(1):15 "&
33     "PO(0):16 "&
34     "IACKO(6):17 "&
35     "IACKO(4):18 "& --iack 4 and 5 are out of order to account for the change in priority
36     "IACKO(5):19 "&
37     "IACKO(3):20 "&
38     "IACKO(2):21 "&
39     "IACKO(1):22 "&
40     "IACKO(0):23 ";
41 end PRIORITY;
42
43
44 architecture archPRIORITY of PRIORITY is
45
46     signal iacken:std_logic;
47     begin
48     ------------------------------------------------------------------------
49     -- Top level of priority interrupt controller
50     ------------------------------------------------------------------------
51     iacken <= '1' when IACK = '0' and AS = '0' else '0';
52     u1:decoder port map(a=>ADR,en=>iacken,dout=>IACKO);
53     u2:priorityenc port map (a=>PI,h=>IACK,o=>PO);
54 end archPRIORITY;
55
56
```

```
1 -- priority encoder
2 library ieee;
3 use ieee.std_logic_1164.all;
4
5 package priorityenc_pkg is
6     component priorityenc
7     port(
8         a: in std_logic_vector(6 downto 0);     --lines to encode
9         h: in std_logic;                        --hold input
10         o: inout std_logic_vector(2 downto 0)       --encoded outputs
11     );
12     end component;
13 end priorityenc_pkg;
14
15 library ieee;
16 use ieee.std_logic_1164.all;
17
18 library cypress;
19 use cypress.std_arith.all;
20 use cypress.lpmpkg.all;
21
22 entity priorityenc is
23     port(
24         a: in std_logic_vector(6 downto 0);     --lines to encode
25         h: in std_logic;                        --hold input
26         o: inout std_logic_vector(2 downto 0)       --encoded outputs
27     );
28 end;
29
30 architecture priorityenc_arch of priorityenc is
31 begin
32     prior: process (h,a)
33         begin
34             IF (h = '1') THEN
35                 IF a(6) = '0' THEN      --highest priority
36                     o <= "000";
37                 ELSIF a(4) = '0' THEN   --input 5 is the next hight priority
38                     o <= "001";
39                 ELSIF a(5) = '0' THEN   --followed by input 6...strange, but this is the way
40                     o <= "010";
41                 ELSIF a(3) = '0' THEN
42                     o <= "011";
43                 ELSIF a(2) = '0' THEN
44                     o <= "100";
45                 ELSIF a(1) = '0' THEN
46                     o <= "101";
47                 ELSIF a(0) = '0' THEN
48                     o <= "110";
49                 ELSE
50                     o <= "111";
51                 END IF;
52             END IF;
53     end process prior;
54 end priorityenc_arch;
55
56
```

```
1 -- decoder
2 library ieee;
3 use ieee.std_logic_1164.all;
4
5 package decoder_pkg is
6     component decoder
7     port(
8         a: in std_logic_vector(2 downto 0);
9         en: in std_logic;
10         dout: out std_logic_vector(7 downto 1)
11     );
12     end component;
13 end decoder_pkg;
14
15 library ieee;
16 use ieee.std_logic_1164.all;
17
18 library cypress;
19 use cypress.std_arith.all;
20 use cypress.lpmpkg.all;
21
22 entity decoder is
23     port(
24         a: in std_logic_vector(2 downto 0);
25         en: in std_logic;
26         dout: out std_logic_vector(7 downto 1)
27     );
28 end;
29
30 architecture decoder_arch of decoder is
31 begin
32     dout(7) <= '0' when a = "111" and en = '1' else '1';
33     dout(6) <= '0' when a = "110" and en = '1' else '1';
34     dout(5) <= '0' when a = "101" and en = '1' else '1';
35     dout(4) <= '0' when a = "100" and en = '1' else '1';
36     dout(3) <= '0' when a = "011" and en = '1' else '1';
37     dout(2) <= '0' when a = "010" and en = '1' else '1';
38     dout(1) <= '0' when a = "001" and en = '1' else '1';
39 end decoder_arch;
40
```

```
1 ----------------------------------------------------------------------------------------
2 -- u56--
3 -- This pld is used to send the configuration data to a set of Xilinx
4 -- Field programmable gate arrays.
5 ----------------------------------------------------------------------------------------
6
7 library ieee;
8 use ieee.std_logic_1164.all;
9 library cypress;
10 use cypress.std_arith.all;
11 use cypress.lpmpkg.all;
12
13 use work.combo_pkg.all;
14 use work.reg_pkg.all;
15
16 entity XILINX is port (
17     LDS: IN std_logic;      --lower data strobe active low
18     AS: IN std_logic;       --address strobe active low
19     CS: IN std_logic;       --chip select active low
20     ADR: IN std_logic_vector(2 downto 0);   --address lines active high
21     RNW: IN std_logic;      --read/not write input active HIHG
22     INIT_IN: IN std_logic;  --read back for INIT line on Xilinx
23     DONE_IN: IN std_logic;  --read back for DONE line on Xilinx
24     DIN: INOUT std_logic;   --data line to Xilinx
25     STROBE: INOUT std_logic;    --clock line to Xilinx
26     DONE: INOUT std_logic;  --done line to Xilinx
27     INIT: INOUT std_logic;  --init line to xilinx
28     PROGRAM:INOUT std_logic;    --program line to xilinx
29     TS_DONE:INOUT std_logic;    --tristate control of DONE line to xilinx
30     TS_INIT:INOUT std_logic;    --tristate control of INIT line to xilinx
31     D0:INOUT std_logic;     --data line to 68000
32     DTACKOE:INOUT std_logic;    --tristate control of DTACK to 68000
33     DTACK:INOUT std_logic   --DTACK to 68000, active low
34 );
35 type regselect is (dataout,strobeout,doneout,initout,programout,donets,initts,reset);
36 attribute pin_numbers of XILINX:entity is
37     "LDS:1 "&
38     "AS:2 "&
39     "ADR(0):6 "&
40     "ADR(1):7 "&
41     "ADR(2):8 "&
42     "CS:9 "&
43     "RNW:10 "&
44     "INIT_IN:11 "&
45     "DONE_IN:13 "&
46     "D0:14 "&
47     "TS_INIT:15 "&
48     "TS_DONE:16 "&
49     "PROGRAM:17 "&
50     "INIT:18 "&
51     "DONE:19 "&
52     "STROBE:20 "&
53     "DIN:21 "&
54     "DTACKOE:22 "&
55     "DTACK:23";
56 end XILINX;
57
58
59 architecture archXILINX of XILINX is
60 signal resetdata:std_logic;
61 signal data:std_logic;
62 signal dataoe:std_logic;
63 signal ack:std_logic;
64 signal ack_oe:std_logic;
65 signal done_p:std_logic;
66 signal init_p:std_logic;
67 begin
68     resetdata <= '1' when (ADR = "111") and (RNW = '0') and (CS = '0') else '0';    --reset sig
69     u1:reg port map (clk=>LDS,cs=>CS,rnw=>RNW,d0=>D0,reset=>resetdata,adr=>ADR,din=>DIN,strobe=
70                      done=>done_p,init=>init_p,donets=>TS_DONE,initts=>TS_INIT,program=>PROGRAM)
```

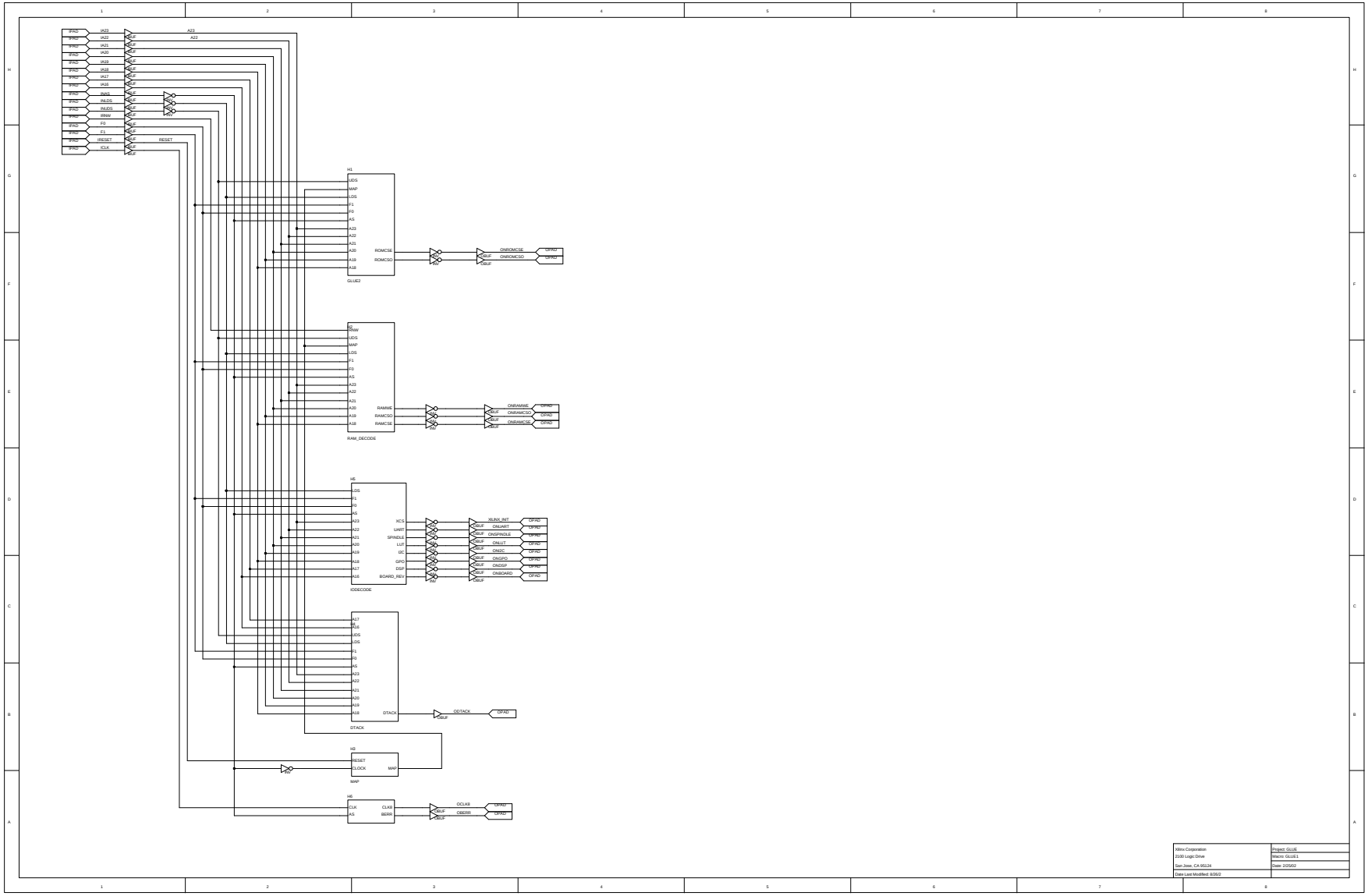
```
71      u2:combo port map (cs=>CS,adr=>ADR,rnw=>RNW,done_in=>DONE_IN,init_in=>INIT_IN,
72                         dtack_oe=>ack_oe,dtack=>ack,data=>data,data_oe=>dataoe);
73      D0 <= data when (dataoe = '1') else 'Z';
74      DTACKOE <= ack_oe;
75      DTACK <= ack when (DTACKOE = '1') else 'Z';
76      DONE <= done_p when (TS_DONE = '1') else 'Z';
77      INIT <= init_p when (TS_INIT = '1') else 'Z';
78  end archXILINX;
79
```

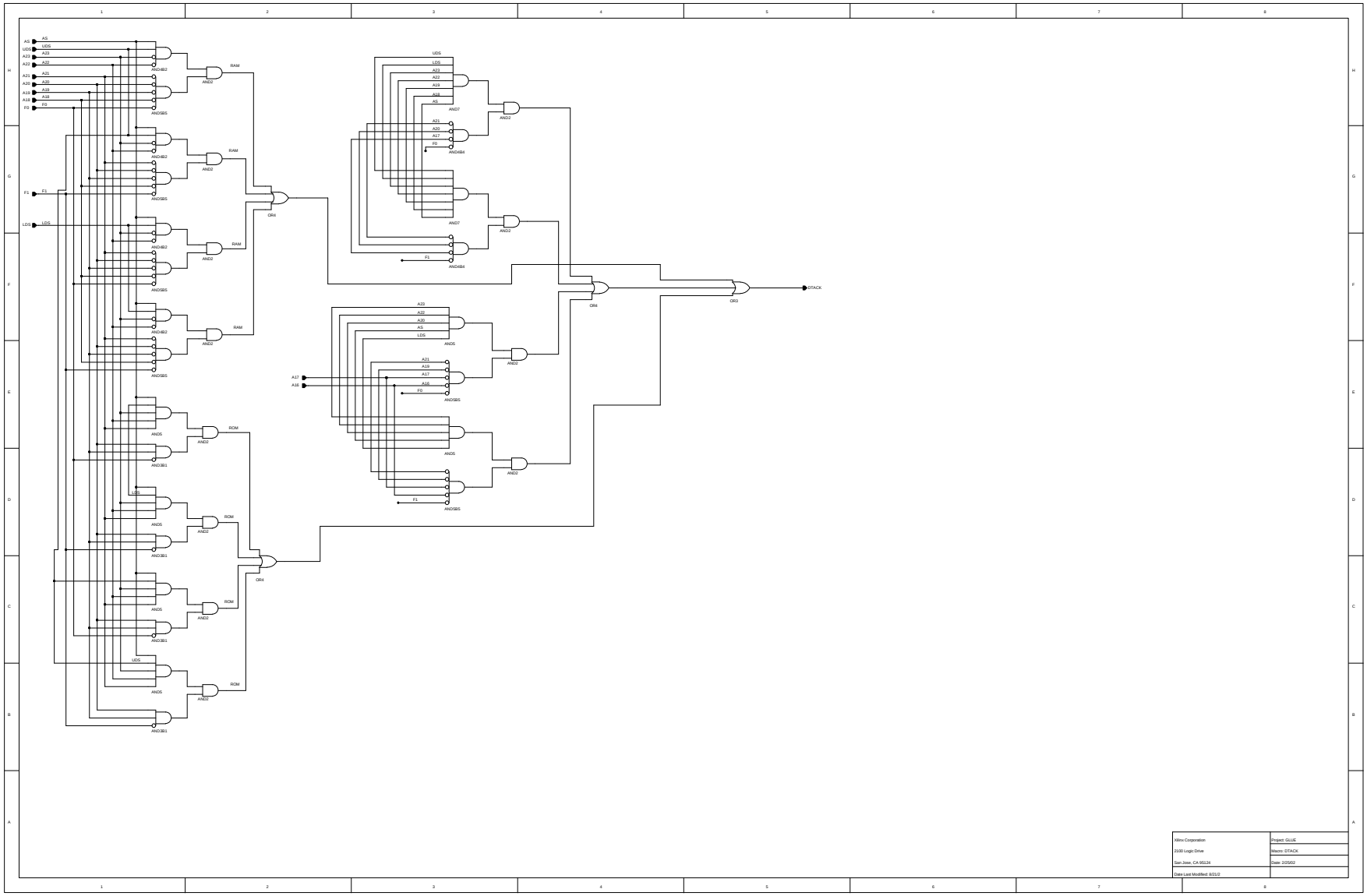
```
1 ----------------------------------------------------------------------------------------------
2 -- combinatorial logic for XILINX interface
3 ----------------------------------------------------------------------------------------------
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7
8 package combo_pkg is
9     component combo
10     port (
11         cs:in std_logic;        --chip select
12         adr:in std_logic_vector(2 downto 0);    --address lines
13         rnw:in std_logic;       --read not write input
14         done_in:in std_logic;   --done input
15         init_in:in std_logic;   --init input
16         dtack_oe:inout std_logic;   --tri state enable for dtack
17         dtack:out std_logic;        --dtack output
18         data:inout std_logic;           --data line
19         data_oe:inout std_logic --data line tri state control
20     );
21     end component;
22 end combo_pkg;
23
24 library ieee;
25 use ieee.std_logic_1164.all;
26
27 library cypress;
28 use cypress.std_arith.all;
29 use cypress.lpmpkg.all;
30
31 entity combo is
32     port (
33         cs:in std_logic;        --chip select
34         adr:in std_logic_vector(2 downto 0);    --address lines
35         rnw:in std_logic;       --read not write input
36         done_in:in std_logic;   --done input
37         init_in:in std_logic;   --init input
38         dtack_oe:inout std_logic;   --tri state enable for dtack
39         dtack:out std_logic;        --dtack output
40         data:inout std_logic;           --data line
41         data_oe:inout std_logic --data line tri state control
42     );
43 end;
44
45 architecture combo_arch of combo is
46 begin
47     data_oe <= '1' when (cs = '0') and (rnw = '1') else '0';
48     dtack_oe <=  '1' when (cs = '0') and (rnw = '0') else
49                  '1' when (cs = '0') and (rnw = '1') and (adr = "010") else
50                  '1' when (cs = '0') and (rnw = '1') and (adr = "011") else
51                  '0';
52     dtack <= '0' when (cs = '0') and (rnw = '0') else
53              '0' when (cs = '0') and (rnw = '1') and (adr = "010") else
54              '0' when (cs = '0') and (rnw = '1') and (adr = "011") else '1';
55     with adr select
56         data <= done_in when "010",
57                 init_in when "011",
58                 '0' when others;
59
60 end combo_arch;
61
```

```
1 ------------------------------------------------------------------------------------------
2 -- u56 registers
3 ------------------------------------------------------------------------------------------
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7
8 package reg_pkg is
9     component reg
10     port (
11         clk:in std_logic;   --clock input for registers
12         cs:in std_logic;    --clock enable for registers
13         rnw:in std_logic;   --write enable for registers
14         d0:inout std_logic; --data input for registers
15         reset:in std_logic; --reset input for registers
16         adr:in std_logic_vector;    --resister select input
17         din:inout std_logic;        --data register
18         strobe:inout std_logic; --strobe register
19         done:inout std_logic;   --done register
20         init:inout std_logic;   --init register
21         donets:inout std_logic; --tristate control for done
22         initts:inout std_logic; --tristate control for init
23         program:inout std_logic --program register
24     );
25     end component;
26 end reg_pkg;
27
28 library ieee;
29 use ieee.std_logic_1164.all;
30
31 library cypress;
32 use cypress.std_arith.all;
33 use cypress.lpmpkg.all;
34
35 entity reg is
36     port (
37         clk:in std_logic;   --clock input for registers
38         cs:in std_logic;    --clock enable for registers
39         rnw:in std_logic;   --write enable for registers
40         d0:inout std_logic; --data input for registers
41         reset:in std_logic; --reset input for registers
42         adr:in std_logic_vector;    --resister select input
43         din:inout std_logic;        --data register
44         strobe:inout std_logic; --strobe register
45         done:inout std_logic;   --done register
46         init:inout std_logic;   --init register
47         donets:inout std_logic; --tristate control for done
48         initts:inout std_logic; --tristate control for init
49         program:inout std_logic --program register
50     );
51 end;
52
53 architecture reg_arch of reg is
54 begin
55     sr:process(reset,clk)
56     begin
57         if(reset = '1') then
58             din <= '0';
59             strobe <= '0';
60             done <= '0';
61             init <= '0';
62             donets <= '0';
63             initts <= '0';
64             program <= '0';
65         elsif (clk'event and clk = '1') then
66             if(cs = '0') and (rnw = '0') then
67                 if(adr = "000") then
68                     din <= d0;      --write data into data register
69                 else
70                     din <= din;     --hold din
```

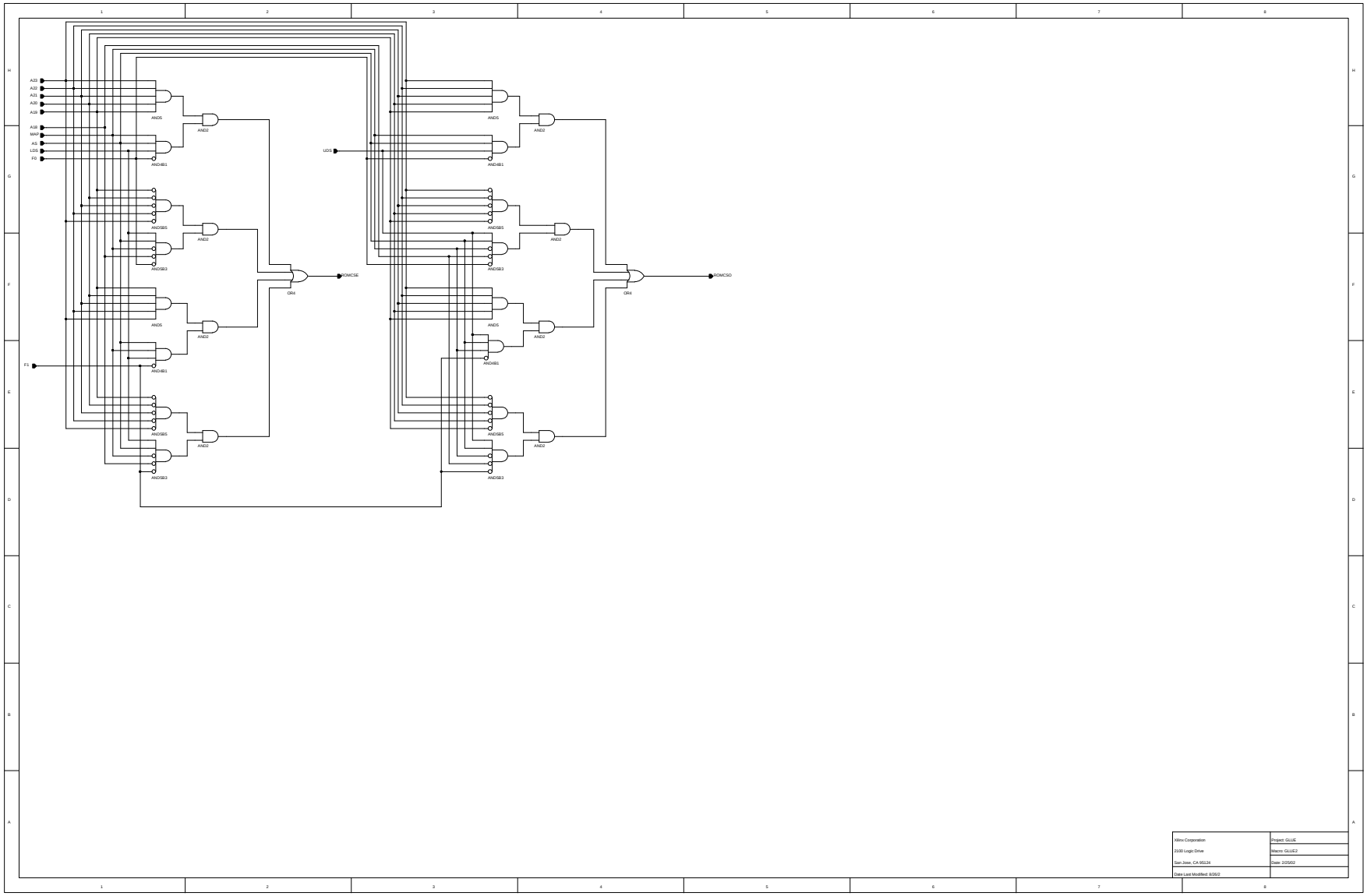
```
71                 end if;
72                 if(adr = "001") then
73                     strobe <= d0;   --write data into strobe register
74                 else
75                     strobe <= strobe;   --hold strobe
76                 end if;
77                 if(adr = "010") then
78                     done <= d0;     --write data into done register
79                 else
80                     done <= done;   --hold done
81                 end if;
82                 if(adr = "011") then
83                     init <= d0;     --write data into init register
84                 else
85                     init <= init;   --hold init
86                 end if;
87                 if(adr = "100") then
88                     program <= d0;  --write data into program register
89                 else
90                     program <= program; --hold program
91                 end if;
92                 if(adr = "101") then
93                     donets <= d0;   --write data into done tristate register
94                 else
95                     donets <= donets;   --hold done tristate
96                 end if;
97                 if(adr = "110") then
98                     initts <= d0;   --write data into init tristate register
99                 else
100                     initts <= initts;   --hold init tristate
101                end if;
102            end if;
103        end if;
104    end process;
105 end reg_arch;
106
```

```
1 ----------------------------------------------------------------------------------------
2 -- IEEE io decoder for spindle controller
3 -- U57 for Rev B version of 600160 board
4 -- Address "0000" is for the LED display
5 -- Address "0001" is for the PORT C register
6 ----------------------------------------------------------------------------------------
7
8 library ieee;
9 use ieee.std_logic_1164.all;
10
11 entity u57 is port (
12     lds:in std_logic;   --lower data strobe
13     uds:in std_logic;   --upper data strobe
14     as:in std_logic;    --address strobe
15     r_nw:in std_logic;  --read-notwrite
16     dspirqack:in std_logic; --interrupt ack for DSP chip
17     a:in std_logic_vector (3 downto 0); --address lines, A16->A13
18     fc:in std_logic_vector (2 downto 0);    --function code inputs
19     ieeeirqack:in std_logic;    --interrupt ack for ieee chip
20     gpibcs:in std_logic;    --chip select input for ieee chip
21     dtackoe:buffer std_logic;   --tristate control for DTACK output
22     dtack:out std_logic;    --DTACK output
23     laclk:out std_logic;    --clock to logic analyzer
24     wd_led:out std_logic;   --write LED register
25     iack:out std_logic;     --interrupt acknowledge output
26     read:out std_logic;     --read line to rams
27     portc:out std_logic     --write strobe for PORT C register
28 );
29 attribute pin_numbers of u57:entity is
30     "lds:1 "&
31     "uds:2 "&
32     "as:3 "&
33     "r_nw:4 "&
34     "gpibcs:6 "&
35     "a(2):7 "&
36     "dspirqack:8 "&
37     "a(1):9 "&
38     "a(0):10 "&
39     "a(3):11 "&
40     "wd_led:14 "&
41     "dtack:16 "&
42     "dtackoe:17 "&
43     "read:18 "&
44     "iack:19 "&
45     "fc(2):20 "&
46     "fc(1):21 "&
47     "fc(0):22 "&
48     "portc:23 "&
49     "laclk:15";
50 end u57;
51
52 architecture archU57 of u57 is
53 signal ack:std_logic;
54 begin
55     laclk <= '0' when (lds = '0') and (uds = '0') else '1';
56     portc <= '0' when (lds = '0') and (uds = '0') and (a = "0001") and (gpibcs = '0') else '1';
57     wd_led <= '0' when (r_nw = '0') and (lds = '0') and (gpibcs = '0') and (a = "0000") else '1
58     dtackoe <= '1' when ((gpibcs = '0') and (lds = '0') and ((a = "0001") or (a = "0000"))) or
59     ack <= '1'     when ((gpibcs = '0') and (lds = '0') and ((a = "0001") or (a = "0000"))) or
60     iack <= '0' when (fc = "111") else '1';
61     dtack <= '0' when (ack = '1') else 'Z';
62     read <= '0' when (r_nw = '1') else '1';
63
64 end archU57;
65
```

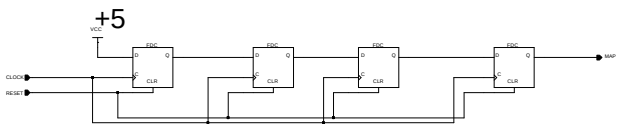

IPAD
IA23
IA22
IA21
IA20
IA19
IA18
IA17
IA16
IWAIT
INLDS
INUDS
IRNW
IFC
IF1
IRESET
ICLK
OBUF
A23
A22
RESET
H1
UDS
MAP
LDS
F1
F0
AS
A23
A22
A21
A20
A19
A18
ROMCSE
ROMCSO
GLUE2
ONROMCSE
ONROMCSO
OPAD
H5
RNW
RAMWE
RAMCSO
RAMCSE
RAM_DECODE
ONRAMWE
ONRAMCSO
ONRAMCSE
H6
A17
A16
XCS
UART
SPINDLE
LUT
IDC
GPO
DSP
BOARD_REV
IODECODE
XILINX_INT
ONUART
ONSPINDLE
ONLUT
ONIDC
ONGPO
ONDSP
ONBOARD
DTACK
ODTACK
H2
RESET
CLOCK
MAP
MAP
H6
CLK
AS
CLK8
8MSR
OCLK8
O8MSR
Xilinx Corporation
2100 Logic Drive
San Jose, CA 95124
Date Last Modified: 8/26/2
Project: GLUE
Macro: GLUE1
Date: 2/25/02



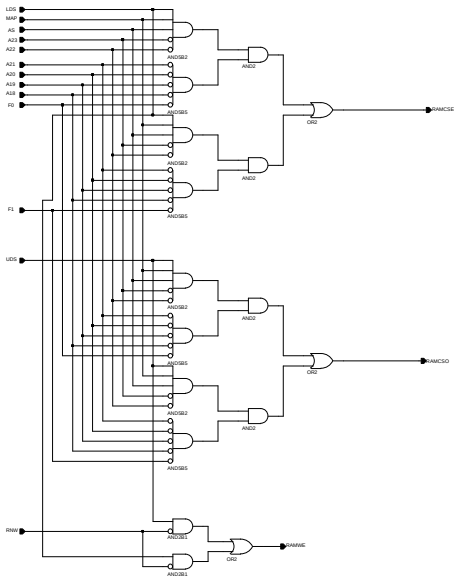

| | |
|---|---|
| Xilinx Corporation | Project: GLUE |
| 2100 Logic Drive | Macro: DTACK |
| San Jose, CA 95124 | Date: 2/25/02 |
| Date Last Modified: 8/21/12 | |



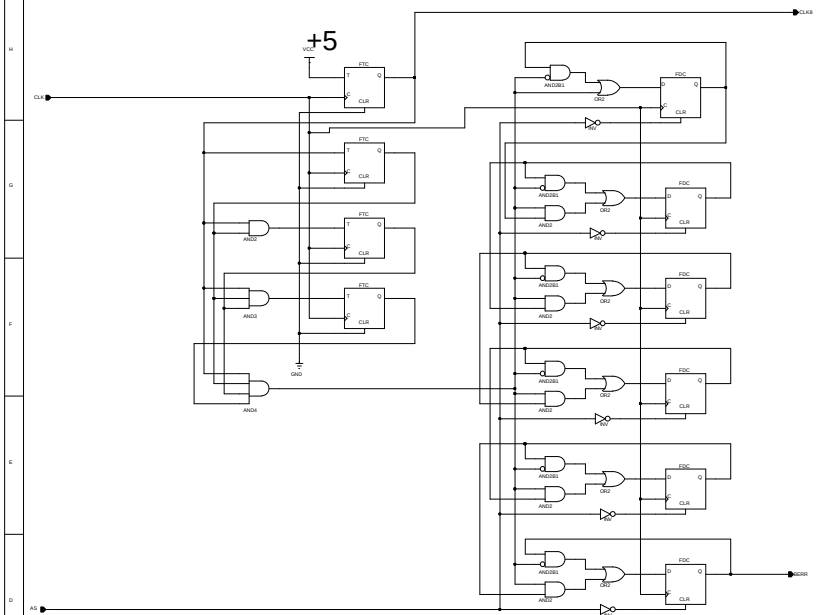

| | |
|---|---|
| Xilinx Corporation | Project: GLUE |
| 2100 Logic Drive | Macro: GLUE2 |
| San Jose, CA 95124 | Date: 2/25/02 |
| Date Last Modified: 8/26/2 | |



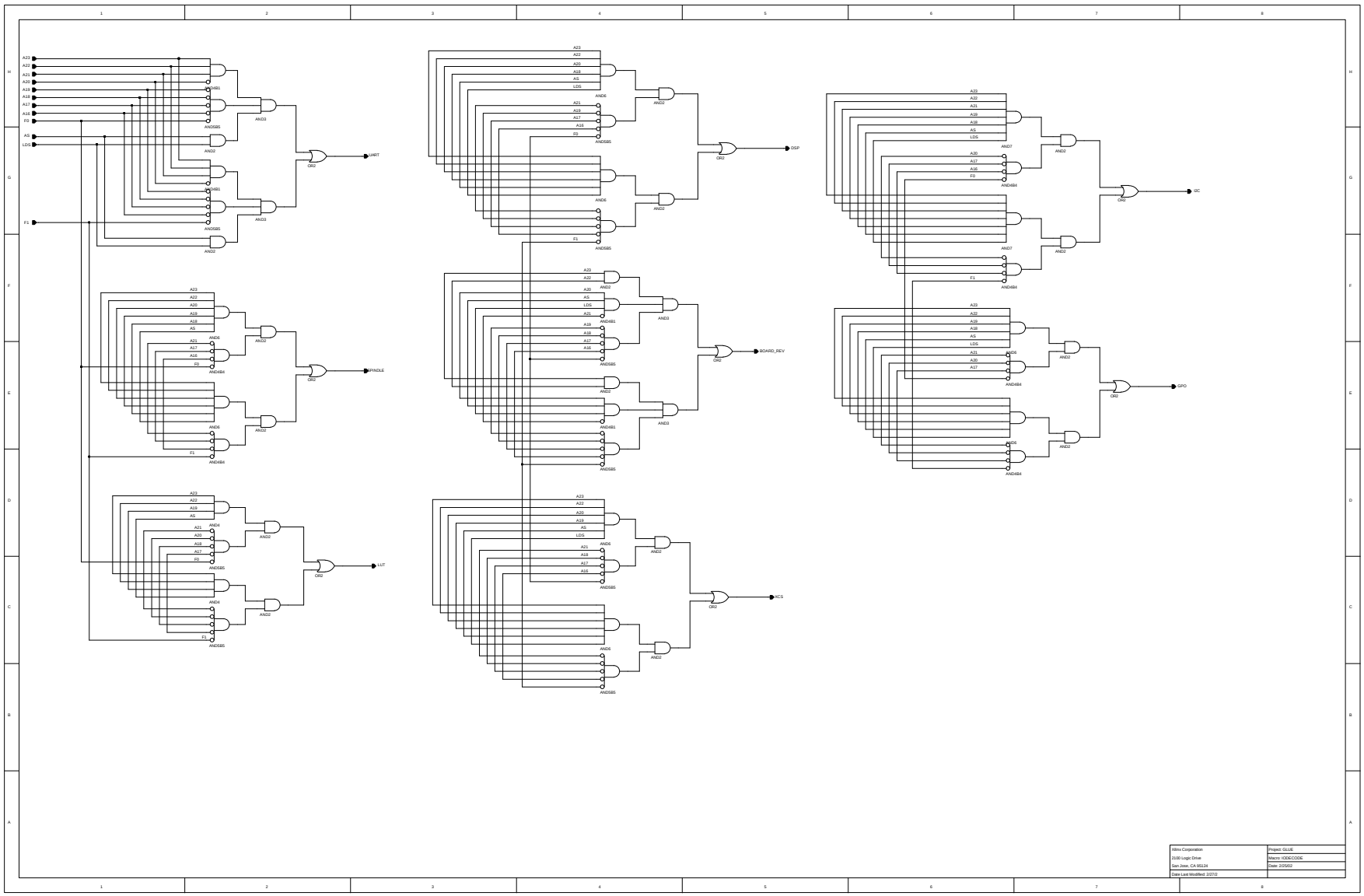

| | |
|---|---|
| Xilinx Corporation | Project: GLUE |
| 2100 Logic Drive | Macro: MAP |
| San Jose, CA 95124 | Date: 2/25/02 |
| Date Last Modified: 2/26/2 | |




| | |
|---|---|
| Xilinx Corporation | Project: GLUE |
| 2100 Logic Drive | Macro: RAM_DECODE |
| San Jose, CA 95124 | Date: 2/25/02 |
| Date Last Modified: 8/23/2 | |




| | |
|---|---|
| Xilinx Corporation | Project: GLUE |
| 2100 Logic Drive | Macro: DIV-TIMEOUT |
| San Jose, CA 95124 | Date: 2/25/02 |
| Date Last Modified: 2/27/2 | |


A23
A22
A21
A20
A19
A18
A17
A16
F0
AS
LDS
F1
AND5B5
AND4B1
AND4B4
AND2
AND3
AND6
OR2
UART
SPINDLE
LVT
DSP
BOARD_REV
NCS
SC
GPO
Xilinx Corporation
2100 Logic Drive
San Jose, CA 95124
Date Last Modified: 2/27/2
Project: GLUE
Macro: XDECODE
Date: 2/25/02

```
/*************************************************************************
**  C version of CIO
**  Copyright (c) 1991 by Jim Patchell
**
**  This is the Central Input Output device handler.  This is probably
**  a lame attempt to make access to various I/O devices a little more
**  orthagonal.  Maybe, maybe not.
**
**  The interface function, cio only requires one parameter, but accepts
**  a variable number of parameters depending on the type of function
**  it is that is being executed.
**
**  The functions that can be executed are:
**
**      add handler
**      open
**      close
**      get
**      get record (read)
**      put
**      put record (write)
**      status
**      special command (xio)
**
**
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include "cio.h"
#include "task.h"
#include "queue.h"

/*
    cio global variables
*/

static IOCB Iocb[32];   /*  space for 32 I/O channels   */
static HTABS htabs[20]; /*  space for 20 I/O devices    */
static int last_dev = 0;    /*  last device entry in table  */

static free_iocb(int handle);
static find_free_iocb(void);
static dev_search(int handle);

static Wait *IocbEvent = NULL;

extern "C" long cio(va_list argp)
{
    /*********************************************************************
    ** this is the Central I/O function (cio)
    ** This is where all of the high level IO access is handled.
    **
    ** Parameter:
    ** argp.....pointer to a list of arguments
    ** ReturnValue:
    ** Returns a long value depending on the function
    *********************************************************************/
    int handle;
    int a;      /*  temporary data storage  */
    long b;     //another temporary data storage
    IOCB *iocb;
    int cmd;
```

```
    cmd = va_arg(argp,int);     //get function command
    iocb = Iocb;
    switch(cmd)
    {
        case ADD_HANDLER:
            /*************************************************************
            ** This command is used to insert handler vector into HTABS
            ** called as: cio(cmd,devname,table)
            ** Also Called as:AddHandler(const char *dev_name,const H_JVEC *jump_table);
            **
            ** Parameter:
            ** cmd......Will have the value ADD_HANDLER
            ** devname..Points to a string that contains the device name
            ** table....Points to an H_JVEC structure that contains the
            **          driver entry points
            ** ReturnValue:
            ** Always returns 0.
            *************************************************************/
            IocbEvent->Pend();
            htabs[last_dev].dev_name = va_arg(argp,char *); /*  device name */
            htabs[last_dev].HtabsEntry = va_arg(argp,H_JVEC *); /*  address of table    */
            ++last_dev;
            IocbEvent->Post();      //unblock add handler
            break;
        case OPEN:
            /*************************************************************
            ** This command is used to open up the device.
            ** called as: cio(int cmd,char *dev_name,int mode);
            ** also called as:Open(char *dev_name,int mode);
            **
            ** Parameter:
            ** cmd.......Will have the value OPEN.
            ** dev_name..Name of device to open
            ** mode......Mode to open device as (Read/Write).
            ** ReturnValue:
            ** Returns handle to device (0 or positive), or negative on error
            *************************************************************/
            IocbEvent->Pend();
            if((handle = find_free_iocb()) == -1)
            {
                IocbEvent->Post();
                return(CIO_NO_HANDLES);
            }
            /*
               get open parrameters off of stack
               passed as cio(cmd,name,mode)
            */
            iocb += handle;
            iocb->dev_name = va_arg(argp,char *);
            iocb->mode = va_arg(argp,int);
            if(( iocb->ichid = dev_search(handle)) < 0)
            {
                IocbEvent->Post();
                return(CIO_NO_DEVICE);  /*  could not find device handle    */
            }
            /* we have laid claim to the IOCB, release iocbevent    */
            IocbEvent->Post();
            if((a = (*htabs[iocb->ichid].HtabsEntry->openv)(iocb,argp)) < 0)
            {
                handle = a;     //return ERROR
                iocb->ichid = -1;   //free up IOCB again
            }
            return(handle);
        case CLOSE:
            /*************************************************************
            ** This command is used to terminate (release) access to an
```

```
        ** IO device.
        ** called as: cio(int cmd,int handle);
        ** Also Called as:Close(int handle);
        **
        ** Parameter:
        ** cmd..........has the value of CLOSE
        ** handle........handle of device, aquired by using OPEN
        ** ReturnValue:
        ** Returns 0 on success, negative on error
        ***************************************************************/
         /*
            get close parrameters off of stack
            passes as: cio(cmd,handle);
        */
         handle = va_arg(argp,int);
         iocb += handle;
         if(handle > MAX_HANDLE || handle < 0)
             return(CIO_INVLD_HNDL);
         if(iocb->ichid < 0)
             return(CIO_NOT_OPEN);
         (*htabs[iocb->ichid].HtabsEntry->closv)(iocb,argp);
         return(free_iocb(handle));
    case GETC:
        /*************************************************************
        ** This command is used to get a character from a device
        ** called as: cio(int cmd, int handle);
        ** Also called as:Getc(int handle);
        **
        ** Parameter:
        ** cmd.....Has the value of GETC
        ** handle..Handle of device to get data from
        ** Return Value:
        ** Returns character (positive), or error (negative) value.
        ** It should be noted that the return value is a long.
        ** Most devices generally deal in bytes.  The byte value is
        ** not sign extended, so a valid return value is always positive
        *************************************************************/
         /*
        ** Get character from handler
        ** called as:  cio(GETC,handle)
        */
         handle = va_arg(argp,int);
         if(handle > MAX_HANDLE || handle < 0)
             return(CIO_INVLD_HNDL);
         iocb += handle;
         if(iocb->ichid < 0)
             return(CIO_NOT_OPEN);
         if((cmd & iocb->mode) == 0)
             return(CIO_WRONLY);
         //
         // look up function
         //
         return ( (*htabs[iocb->ichid].HtabsEntry->getv)(iocb,argp)  );
    case READ:
        /*************************************************************
        ** Read a block of data from a device
        ** called as: cio(READ,handle,buffer,count)
        ** Also called as:Read(int handle,char *buff,long count)
        **
        ** parameter:
        ** cmd......This value will be READ
        ** handle...Handle of device to be read.
        ** buffer...character array where data will be stored
        ** count....number of characters to read
        ** ReturnValue:
        ** Returns the number of characters read or negative on error
```

```
    ************************************************************/
    handle = va_arg(argp,int);
    if(handle > MAX_HANDLE || handle < 0)
        return(CIO_INVLD_HNDL);
    iocb += handle;
    if(iocb->ichid < 0)
        return(CIO_NOT_OPEN);
    if((cmd & iocb->mode) == 0)
        return(CIO_WRONLY);
    //
    //look up function
    //
    return (*htabs[iocb->ichid].HtabsEntry->readv)(iocb,argp);
case PUTC:
    /************************************************************
    ** This command outputs a character to the device
    ** Called As: cio(PUTC,handle,data)
    ** Also called as:Putc(int handle,int c);
    **
    ** Parameter:
    ** cmd........This value will be PUTC
    ** handle.....Handle of device to send character to
    ** data.......Character to send.
    ** ReturnValue:
    ** Returns 0 on success, negative value on failure
    ************************************************************/
    handle = va_arg(argp,int);
    if(handle > MAX_HANDLE || handle < 0)
        return(CIO_INVLD_HNDL);
    iocb += handle;
    if(iocb->ichid < 0)
        return(CIO_NOT_OPEN);
    if((iocb->mode & cmd) == 0)
        return(CIO_RDONLY); /*  read only error */
    a = va_arg(argp,int);
    //
    //look up function
    //
    return( (*htabs[iocb->ichid].HtabsEntry->putv)(iocb,a,argp));
case WRITE:
    /************************************************************
    ** This command writes a block of data to the device
    ** Called As: cio(WRITE,handle,buffer,count);
    ** Also Called As:Write(int handle,char *buff,long count);
    **
    ** Parameter:
    ** cmd.......this value is set as WRITE.
    ** handle....Handle of device to write block to
    ** buff......Pointer to array of characters to write
    ** count.....Number of characters to write to device
    ** ReturnValue:
    ** returns either an error or actual number of bytes written
    ************************************************************/
    handle = va_arg(argp,int);
    if(handle > MAX_HANDLE || handle < 0)
        return(CIO_INVLD_HNDL);
    iocb += handle;
    if(iocb->ichid < 0)
        return(CIO_NOT_OPEN);
    if((iocb->mode & cmd) == 0)
        return(CIO_RDONLY); /*  read only error */
    return( (*htabs[iocb->ichid].HtabsEntry->writev)(iocb,argp));
case STATUS:
    /************************************************************
    ** This command returns the device status
    ** called as: cio(cmd,handle,buffer,count,aux)
```

```
            ** called as: cio(cmd,-1,dev_name,buffer,count,aux)
            ** Also called as:Status(int handle,char *buff,long count,int aux);
            ** Or called as  :StatusO(char *dev_name,char *buff,long count,int aux);
            **
            ** Parameter:
            ** cmd.......this value is set as STATUS.
            ** handle....handle of device.  If -1, use next param as device name
            ** dev_name..pointer to string naming device to open (if handle < 0)
            ** buff......pointer to buffer to put status information
            ** count.....size of buffer
            ** aux.......aux information to determine what to return status of
            **
            ************************************************************/
             handle = va_arg(argp,int);
             if(handle < 0)
             {   /*  this is immeadiate operation    */
                 IocbEvent->Pend();
                 if((handle = find_free_iocb()) == -1)
                 {
                     IocbEvent->Post();
                     return(CIO_NO_HANDLES); /* open up an IOCB  */
                 }
                 iocb += handle;
                 iocb->dev_name = va_arg(argp,char *);
                 if(( iocb->ichid = dev_search(handle)) < 0)
                 {
                     IocbEvent->Post();
                     return(CIO_NO_DEVICE);  /*  could not find device handle    */
                 }
                 /* we have grabed this IOCB */
                 IocbEvent->Post();
                 b = (*htabs[iocb->ichid].HtabsEntry->statv)(iocb,argp);
                 free_iocb(handle);  /*  close iocb  */
                 return(b);
             }
             else
             {   /*  valid handle for already open iocb  */
                 iocb += handle;
                 if(handle > MAX_HANDLE || handle < 0)
                     return(CIO_INVLD_HNDL);
                 return((*htabs[iocb->ichid].HtabsEntry->statv)(iocb,argp) );
             }
        default:
            /************************************************************
            ** All other commands are processed here (special commands)
            **
            ** called as:  cio(cmd,handle,dev_name,buffer,count,aux)
             ** Also Called as:Xio(int cmd,int handle,char *dev_name,char *buff,long count,int au
x,...);
            **
            ** parameter:
            ** cmd.......special command to handle
            ** handle....handle of device.  If handle < 0, use dev_name to open device
            ** dev_name..name of device to open.  If handle >= 0, ignore.
            ** buff......data buffer pointer (if needed)
            ** count.....size of data buffer (if needed)
            ** aux.......aux data for handler
            ** ReturnValue:
             ** Returns negative value on error.  0 or positive, depending of function, on succes
            **
            ************************************************************/
             if((handle = va_arg(argp,int)) < 0)
             {
                 /*
                    this is an immediate operation
                */
```

```
                IocbEvent->Pend();
                if((handle = find_free_iocb()) == -1)
                {
                    IocbEvent->Post();
                    return(-1); /* open up an IOCB  */
                }
                iocb += handle;
                iocb->dev_name = va_arg(argp,char *);
                if(( iocb->ichid = dev_search(handle)) < 0) /*  locate handler entry table  */
                {
                    IocbEvent->Post();
                    return(-1); /*  could not find device handle    */
                }
                /*  we have grabbed the IOCB    */
                IocbEvent->Post();
                b = (*htabs[iocb->ichid].HtabsEntry->specv)(cmd,iocb,argp);
                free_iocb(handle);
                return(b);
            }
            else
            {
                /*
               ** iocb is already open
               */
                iocb += handle;
                va_arg(argp,char *);    //get rid of dummy device name
                return((*htabs[iocb->ichid].HtabsEntry->specv)(cmd,iocb,argp));
            }
    }   /*  end switch cmd  */
    return 0l;  //everything is peachy keen
}

static dev_search(int handle)
{
    /*********************************************************************
    ** This function is used to find a device name in the driver lookup
    ** table.
    **
    ** parameter:
    ** handle.....handle of device.  dev_name is stored in the IOCB prior
    **            to this function being called.
    ** ReturnValue:
    ** Returns Index into device table, negative on error
    *********************************************************************/
    int i;  //this value will be returned, index into device table
    HTABS *ht;  //pointer to device table
    IOCB *iocb; //pointer to IOCB for device
    char *n,un[4];      /*unit numbers up to 999    */
    int l;      /*  legnth of device name   */

    iocb = Iocb + handle;   //calculate address of IOCB for this device
    //-----------------------
    // Isolate the device name
    // from any unit numbers
    // that might be present
    //-----------------------
    l = strcspn(iocb->dev_name,":123456789");   /*  we need to find what is not unit number */
    //-----------------------
    // locate last entry in
    // the device table
    //-----------------------
    ht = htabs + last_dev - 1;                                  /*  point to handler table  */
    //-----------------------
    //search the device table
    //search from the end so
    //that if no device is
```

```
    //found, the index "i"
    //will end up with a neg
    //value.
    //----------------------
    for(i=last_dev -1;(i > -1) && strncmp(iocb->dev_name,ht->dev_name,l) ;--i,--ht);
    //----------------------
    //if the index "i" is
    //positive, complete
    // processing
    //----------------------
    if(i >= 0)
    {
        /*
        ** ok, now we need to locate the unit number
        */
        n = iocb->dev_name + l; //points to end of device name
        if((l = strcspn(n,":")) == 0)   //if the end char is ':', then
        {
            iocb->devnum = 1;       /*  default unit number */
        }
        else                        //else end is unit number
        {
            strncpy(un,n,l);        //copy unit number
            un[l] = '\0';
            iocb->devnum = atoi(un);    /*  convert unit number to int  */
        }
    }
    return(i);  //return index to CIO
}

static find_free_iocb(void)
{
    /*********************************************************************
    ** This function is used to find an unused IOCB block
    **
    ** Parameter:
    ** <none>
    ** ReturnValue:
    ** returns postive number if free IOCB found (handle), or negative
    ** on error.
    *********************************************************************/
    int i;
    IOCB *iocb;

    for(i=0,iocb = Iocb;i<32;++i,++iocb)
    {
        if(iocb->ichid == -1)   //is IOCB in use?
            return(i);
    }
    return(-1); //no IOCB's found.
}

static free_iocb(int handle)
{
    /*********************************************************************
    ** This function is used to release an IOCB (close)
    **
    ** parameter:
    ** handle....device handle of block to free
    ** ReturnValue:
    ** this function always returns 0
    *********************************************************************/
    Iocb[handle].ichid = -1;
    return(0);
}
```

```
extern "C" void init_iocb(void)
{
    /*********************************************************************
    ** This function initialized everything in this module
    **
    ** parameter:
    ** <none>
    ** ReturnValue:
    ** <none>
    *********************************************************************/
    int i;
    IOCB *iocb;

    IocbEvent = new Wait(1,"CIO_BLOCKING_SEMAPHORE");   //create blocking semaphore for CIO
    for(i=0,iocb = Iocb;i<32;++i,++iocb)
    {
        iocb->ichid = -1;
        iocb->p = (void *)0;
    }
    last_dev = 0;
}
```

```
/*
** Header file for Central I/O Handler
*/

#ifndef CIO__H
#define CIO__H

#include "erorcode.h"
#include <stdarg.h>
/*
** This data structure is used to describe the i/o device
** that has been opened.
*/

typedef struct {
    int ichid;      /*  handler ID                      0   */
    int devnum;     /*  device number                   2   */
    char *dev_name; /*  pointer to file name            4   */
    int mode;       /*  i/o mode file was opened with   8   */
    void *p;    /*  general purpose pointer             10  */
    int exp2;       /*  future expansion                14  */
}IOCB;

/*
** This vector table is used to locate the functions that
** perform the primitive I/O functions for a device
**
** A pointer to this vector table is located in the device
** handler lookup table.
*/

#ifdef __cplusplus
extern "C" {
#endif

#pragma function ( calling)
typedef struct {
    long (*openv)(IOCB *,va_list);              /*  0   open vector */
    long (*closv)(IOCB *,va_list);              /*  1   close vector    */
    long (*getv)(IOCB *,va_list);               /*  2   get byte vector */
    long (*readv)(IOCB *,va_list);              /*  3   read buffer */
    long (*putv)(IOCB *,int,va_list);           /*  4   put byte vector */
    long (*writev)(IOCB *,va_list);             /*  5   write a buffer  */
    long (*statv)(IOCB *,va_list);              /*  6   get status vector   */
    long (*specv)(int cmd,IOCB *,va_list);      /*  7   special vector  */
    long (*initv)(void);                        /*  8   init code vector    */
}H_JVEC;

#pragma function()

#ifdef __cplusplus
}
#endif

/*
** this is the structure of one element in the device driver
** lookup table.  Each device name is associtated with a vector
** table.
*/

#ifdef __cplusplus
extern "C" {
#endif

typedef struct {
    const char *dev_name;   /*  pointer to device name          */
```

```
    H_JVEC *HtabsEntry;     /*  pointer to handler jump table   */
}HTABS;

#ifdef __cplusplus
}
#endif

/*
    function numbers

    These functions are the predefined by CIO
*/

#define ADD_HANDLER 1   /*  add handler to device table */
#define OPEN        3   /*  open a device       */
#define READ        5   /*  read a buffer       */
#define GETC        7   /*  get a character     */
#define WRITE       9   /*  write a buffer      */
#define PUTC        11  /*  put a character     */
#define CLOSE       12  /*  close a device      */
#define STATUS      13  /*  get device status   */

#define MAX_HANDLE  31
#define EOL 0x0a


/*
** file modes
*/
#define READ_ONLY   4
#define WRITE_ONLY  8

#ifdef __cplusplus
extern "C" {
#endif

extern void init_iocb(void);
extern long cio(va_list argp);

extern long AddHandler(const char *dev_name,const H_JVEC *jump_table);
extern int  Open(char *dev_name,int mode);
extern int  Close(int handle);
extern int  Getc(int handle);
extern long Read(int handle,char *buff,long count);
extern int  Putc(int handle,int c);
extern long Write(int handle,char *buff,long count);
extern int  Status(int handle,char *buff,long count,int aux);
extern int  StatusO(char *dev_name,char *buff,long count,int aux);
extern long Xio(int cmd,int handle,char *dev_name,char *buff,long count,int aux,...);

#ifdef __cplusplus
}
#endif

#endif
```

```
//---------------------------------------------------------
//
// Priority Queue Class
//
// The elements of the Q consist of an Array of pointers
// To the various elements of the Q
// No real data actually exists in this thing.
//
//---------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pq.h"

//---------------------------------------------------------
//
// PQ constructor
//
// int ne;  //maximum number of elements in the priority queue
// int (*c)(void **,void**);    //compare function.  The compare
//                              //function,when called is passed
//                              //a pair of pointers to pointers
//                              //to the objects to be compared
// The constructor initializes all of the other parameters from
// these.
//
//---------------------------------------------------------
//
PQ::PQ(int ne,int (*c)(void **,void **))
{
    cmp = c;
    nitems = 0;
    maxitems = ne;
    heap = new void *[ne];
    bottom = &heap[-1];
}

//---------------------------------------------------------
//
// PQ destructor
//
// Just needs to delete the heap that was allocated
//
//---------------------------------------------------------
//
PQ::~PQ()
{
    delete[] heap;
}

//---------------------------------------------------------
//
// PQ::Insert
//
// This function puts a new object pointer into the heap
// if space is availiable.  It then reheaps
//
//---------------------------------------------------------
//
int PQ::Insert(void *item)
{
    int space_avail;

    if((space_avail = maxitems - nitems) > 0)
    {
```

```
        ++nitems;
        *(++bottom) = item;
        ReheapUp();
    }
    return space_avail;
}

//---------------------------------------------------------
//
// PQ::Delete
//
// This function deletes the highest proiority item from
// the heap.  It stores the pointer to the object in the
// location pointed to by target.  It then reheaps
//
//---------------------------------------------------------

int PQ::Delete(void **target)
{
    int SlotsInUse;

    if((SlotsInUse = nitems) > 0)
    {
        *target = *heap;        //get item from top of heap
        *heap = *bottom--;
        --nitems;
        ReheapDown();
    }
    return SlotsInUse;
}

//---------------------------------------------------------
//
// PQ::Replace
//
// This function removes the hightest priority item from the
// heap and puts it in the location that is pointed to by
// target.  It then replaces it with the object pointed to
// by item.  Then the function reheaps.
//
//---------------------------------------------------------

int PQ::Replace(void **target,void *item)
{
    int SlotsInUse;

    if((SlotsInUse = nitems) > 0)
    {
        if ((*cmp)(&item,heap) > 0) //only replace if priority is less
        {
            *target = item;     //item is higher priority
        }
        else
        {
            *target = *heap;
            *heap = item;
            ReheapDown();
        }
    }
    else
        *target = item;
    return SlotsInUse;
}

//---------------------------------------------------------
//
```

```
// PQ:Remove
//
// This function removes the object pointed to by item from the
// heap and puts it into the location pointed to by target.
// Then the function reheaps
//
//-------------------------------------------------------------
//
int PQ::Remove(void **target,int (*cmp)(void **,void **),void *item)
{
    int SlotsInUse;

    if((SlotsInUse = nitems)>0)
    {
        int i;
        int loop;

        for(i=0,loop=1;i<nitems && loop;++i)
        {
            if( (*cmp)(&item,&heap[i]) == 0)
            {
                *target = heap[i];
                memcpy(&heap[i],&heap[i+1],(nitems - i - 1) * sizeof(void *));
                bottom--;
                nitems--;
                ReheapDown();
                loop = 0;
            }
        }
    }
    return SlotsInUse;
}

/*************************************************************************
**
** These functions are private to the class and are not accessable
**
*************************************************************************/

void PQ::swap(void **s1,void **s2)
{
    void *temp;

    temp = *s1;
    *s1 = *s2;
    *s2 = temp;
}

void PQ::ReheapUp(void)
{
    int parent;     //index of parent
    int child;      //index of child
    void **pparent; //pointer to parent
    void **pchild;  //pointer to child

    child = nitems - 1;
    while((parent = (child - 1)/2) >= 0)
    {
        pchild = &heap[child];
        pparent = &heap[parent];
        if( (*cmp)(pparent,pchild) >= 0)
            break;
        swap(pparent,pchild);
        child = parent;
    }   //end of while statement
}
```

```
void PQ::ReheapDown(void)
{
    int parent; //index of parent
    int child;  //index of child
    void **pparent; //pointer to parent
    void **pchild;   //pointer to child
    void **psibling;    //pointer to sibling
    void **pheap;   //pointer to heap

    pheap = heap;
    for(parent=0,child=1;child < nitems;)
    {
        pparent = &pheap[parent];
        pchild = &pheap[child];
        if(child + 1 < nitems)
        {
            psibling = pchild + 1;
            if((*cmp)(pchild,psibling) < 0)
            {
                pchild = psibling;
                child++;
            }
        }   //end of if(child + 1 < nitems)
        if((*cmp)(pparent,pchild) >= 0)
            break;
        swap(pparent,pchild);
        parent = child;
        child = parent * 2 + 1;
    }   //end of for loop
}
```

```
//-----------------------------------------------------------
//
// Priority Queue Class
//
// The elements of the Q consist of an Array of pointers
// To the various elements of the Q
// No real data actuall exists in this thing.
//
//-----------------------------------------------------------

#ifndef PQ__H
#define PQ__H

class PQ {
    int (*cmp)(void **,void **);        /*  compare two objects     */
    int nitems;                         /*  number of items in heap     */
    int maxitems;                       /*  maximum number of items in heap */
    void **bottom;                      /*  prt to most recently added item */
    void **heap;                        /*  pointer to the heap itself      */
    void swap(void **,void **);
    void ReheapUp(void);
    void ReheapDown(void);
public:
    PQ(int ne,int (*c)(void **,void **));
    ~PQ();
    Insert(void *item);
    int Delete(void **target);
    void *Get(void) {return heap[0];}   //return pointer to highest priority item
    int NumElem(void){return nitems;}   //get total number of items
    int Replace(void **target,void *item);
    int Remove(void **target,int (*cmp)(void **,void **),void *item);
    void *Get(int i) {return heap[i];}  //return pointer to indexed item
};

#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "task.h"
#include "queue.h"
#include "global.h"
#include "cio.h"

//***************************************************************************
//
// This file contains the definitions for the various message queues that
// are needed by the teletrac spindle controller board.
// All of these queues are derived from TSemaphore in task.h
//
// Copyright (c) 1998 by Teletrac Inc.
//
//***************************************************************************

//---------------------------------------------------------------------------
// Class Definitions
// Class BQueue
// A queue of buffers
//
// This class creates a queue of buffers.  It is used by devices that
// need to send out atomic blocks of data.
//---------------------------------------------------------------------------

BQueue::BQueue(int size,int qsize,char *name):TSemaphore(0,TSEMAPHORE_MODE_WAIT,name)
{
    int i;
    Head = 0;
    nObj = 0;
    Tail = 0;
    Buff = new char * [qsize];      //allocate pointer array to buffers
    BTail = new int[qsize];         //allocate array of Tail pointers
    BnChars = new int[qsize];       //allocate array of Char Count
    Qsize = qsize;
    Size = size;
    for(i=0;i<Qsize;++i)
        Buff[i] = new char[Size];   //allocate buffers
}

BQueue::~BQueue()
{
    int i;

    for(i=0;i<Qsize;++i)
        delete[] Buff[i];
    delete[] BnChars;
    delete[] BTail;
    delete[] Buff;
}

void *BQueue::operator new(size_t size)
{
    void *t = malloc(size + 128);
    ((BQueue *)t)->name = ((char *)t) + size;
    return t;
}

void BQueue::operator delete(void *t)
{
    free(t);
}

int BQueue::SendMessage(char *m,int c)
```

```
{
    //-------------------------------------------------------------
    // this function puts the message pointed to by m into first
    // availiable buffer.  Then the semaphore is posted
    // individual buffers are not circular, so no head pointers
    // are required for individual buffers
    // returns 0 on success, negative value on fail
    //-------------------------------------------------------------
    TCB *t;

    int retval = -1;
    int sr = EnterCritical();   //disable interrupts for time being
    if(nObj < Qsize)
    {
        ++Head;                 //PREincrement Head Pointer
        if(Head == Qsize)
            Head = 0;           //wrap Head Pointer
        ++nObj;                 //increment number of messages in buffer
        memcpy(Buff[Head],m,c); //copy data into buffer
        BnChars[Head] = c;      //set number of characters
        BTail[Head] = 0;        //set tail pointer to begining of message
        retval = 0;
        if(task_h)              //is there really a task waiting?
        {
            t = task_h;     //get first task in
            task_h = t->next;       //remove from list
            t->next = (TCB *)0;
            t->status = EVENT_NOERROR;          //return this value
            t->pending = 0;         //task no longer pending
            ActiveTasks->Insert(t); //put task back onto active list
            ExitCritical(sr);
            Schedule();
        }
        else
        {
            ExitCritical(sr);
        }
    }
    else
        ExitCritical(sr);       //enable interrupts
    return retval;
}

int BQueue::GetMessage(void)
{
    //-------------------------------------------------------------
    // Gets a character from the current buffer
    // returns -1 when message is empty (does not cause pend)
    //-------------------------------------------------------------
    unsigned char *b = (unsigned char *)Buff[Tail];
    int retval = -1;        //assume end of buffer
    if(BnChars[Tail] > 0)
    {
        retval = b[BTail[Tail]++];
        //-----------------------------------------
        //The buffers are not circular buffers
        // no wrap is needed
        //-----------------------------------------
        BnChars[Tail]--;
    }
    return retval;      //return value
}

int BQueue::WaitMessage(void)
{
    //-------------------------------------------------------------
```

```
    // Wait for a message to be sent
    // if head == tail, pend on message
    //-------------------------------------------------------------
    int retval = EVENT_NOERROR;             //assume success
    int sr = EnterCritical();   //disable interrupts
    if(nObj)                    //is a message waiting?
    {
        nObj--;
        Tail++;                 //preincrement tail pointer
        if(Tail == Qsize)       //do we need to wrap
            Tail = 0;           //wrap tail pointer
        ExitCritical(sr);       //enable interrupts
    }
    else
    {
        //-------------------------------------------------------------
        // there are no messages waiting, so pend on message posted
        // and switch to another task
        //-------------------------------------------------------------
        /*  no  */
        if(!task_h)     //make linked list of waiting tasks
            task_h = CurrentTask;
        else
            task_t->next = CurrentTask;
        task_t = CurrentTask;
        CurrentTask->status = EVENT_NOERROR;
        CurrentTask->TimeStamp = TSemaphore::TStamp;
        if(ActiveTasks->Delete((void **)&ReadyTask))    //get highest priority task
        {
            ReadyTask->TcbSwaps++;
            asm(" trap #0");                //do context swap
            //-------------------------------------------------
            // This is where we end up when task is restarted
            // Set up Tail pointer and return
            //-------------------------------------------------
            nObj--;
            Tail++;
            if(Tail == Qsize)
                Tail = 0;
        }
        else
        {
            retval = -1;    //indicate some sort of error
        }
        ExitCritical(sr);
        retval = CurrentTask->status;   //return this value
    }
    return retval;
}


//-------------------------------------------------------------------------
// Class Definitions
// Class Wait
// Not much different from TSemaphore.  It just doesn't support timeouts
//-------------------------------------------------------------------------

Wait::Wait(int count,char *name):TSemaphore(count,TSEMAPHORE_MODE_WAIT,name)
{
    value = 0;
}

Wait::~Wait()
{
}
```

```
int Wait::Pend(int timeout)
{
    int retval = TSemaphore::Pend(timeout);
    if(retval > 0) retval = value;
    return retval;
}

int Wait::Post(int v)
{
    TCB *t;
    int sr;

    value = v;
    return TSemaphore::Post(v);
}

void *Wait::operator new(size_t size)
{
    void *t = malloc(size + 128);
    ((Wait *)t)->name = ((char *)t) + size;
    return t;
}

void Wait::operator delete(void *t)
{
    free(t);
}

//---------------------------------------------------------------------------
// Class Definitions
// Message QUEUE class
//
// Basically an "int" pipe to connect data between tasks
//---------------------------------------------------------------------------

MessageQueue::MessageQueue(int size,int mode,char *name):TSemaphore(0,mode,name)    //constructo
r Function
{
    Buff = new int[size];
    Head = 0;
    Tail = 0;
    nChars = 0;
    Size = size;
}

MessageQueue::~MessageQueue()
{
    if(Buff)
        delete[] Buff;  //destroy Buffer
}

int MessageQueue::Post(int v)
{
    return TSemaphore::Post(v);
}

int MessageQueue::Pend(int timeout)
{
    return TSemaphore::Pend(timeout);
}

int MessageQueue::Get(void)
{
    int retval;
    int sr;
```

```
    if(nChars)
    {
        sr = EnterCritical();
        retval = Buff[Tail++];
        if(Tail >= Size)
            Tail = 0;       //wrap tail pointer
        --nChars;
        ExitCritical(sr);
    }
    else
        retval = -1;
    return retval;
}

int MessageQueue::Put(int c)
{
    int retval = c;
    int sr;

    sr = EnterCritical();
    if(nChars < Size)
    {
        Buff[Head++] = c;
        if(Head >= Size)
            Head = 0;       //wrap head pointer
        ++nChars;           //increment number of characters
    }
    else
        retval = -1;        //could not put character into buffer
    ExitCritical(sr);
    return retval;
}

long MessageQueue::GetLong(void)
{
    //-------------------------------------
    // A way of sending a long through the
    // data pipe
    //-------------------------------------
    union {
        long retval;
        int i[2];
    } convert;

    int sr;

    sr = EnterCritical();
    if(nChars > 1)  //are there at least 2 words?
    {
        convert.i[0] = Buff[Tail++];
        if(Tail >= Size) Tail = 0;
        convert.i[1] = Buff[Tail++];
        if(Tail >= Size) Tail = 0;
        nChars -= 2;
    }
    ExitCritical(sr);
    return convert.retval;
}

int MessageQueue::PutLong(long c)
{
    //--------------------------------
    // a way to send a long through
    // the data pipe
    //--------------------------------
    union {
```

```
        long v;
        int i[2];
    }convert;
    int retval=0,sr;

    convert.v = c;
    sr = EnterCritical();
    if(nChars < (Size-1))   //is there space for two ints?
    {
        Buff[Head++] = convert.i[0];
        if(Head >= Size) Head = 0;
        Buff[Head++] = convert.i[1];
        if(Head >= Size) Head = 0;
        nChars += 2;
    }
    else
        retval = -1;
    ExitCritical(sr);
    return retval;
}

int MessageQueue::QueueStatus(void)
{
    return nChars;
}

int MessageQueue::Space(void)
{
    return Size-nChars;
}

void MessageQueue::Flush(void)
{
    Head = 0;       //get rid of all of the data
    nChars = 0;
    Tail = 0;
}

void *MessageQueue::operator new(size_t size)
{
    void *t = malloc(size + 128);
    ((MessageQueue *)t)->name = ((char *)t) + size;
    return t;
}

void MessageQueue::operator delete(void *t)
{
    free(t);
}

//---------------------------------------------------------------------
// Event Queue
//
// This is different from the Message Queue in that this is used when a
// Timeout in not desired.  This will help cut down on the overhead
// involved in using a semaphore.  Only one task can wait on an Event Queue,
// but many tasks can post to one.
//---------------------------------------------------------------------

EventQueue::EventQueue(int size,char *name):TSemaphore(0,TSEMAPHORE_MODE_WAIT,name) //constructo
r function
{
    Buff = new char[size];
    Head = 0;
    Tail = 0;
    nChars = 0;
```

```
    Size = size;
}

EventQueue::~EventQueue()
{
    if(Buff)
        delete[] Buff;
}

void *EventQueue::operator new(size_t size)
{
    void *t = malloc(size + 128);
    ((EventQueue *)t)->name = ((char *)t) + size;
    return t;
}

void EventQueue::operator delete(void *t)
{
    free(t);
}

int EventQueue::Post(int v)
{
    //this function will post the semaphore, but does not alter the queue
    //the value v will be returned by pend
    return TSemaphore::Post(v);
}


int EventQueue::PostMessage(int *d,int c)
{
    //this member function both writes data into the queue and performs a post
    TCB *t;
    int sr;

    sr = EnterCritical();
    if(EventCount < 32766)  //is it going to overflow??
    {
        //-----------------------
        //write data into queue
        //-----------------------
        while(c)
        {
            if(nChars < Size)
            {
                Buff[Head++] = (char)*d++;
                if(Head >= Size)
                    Head = 0;       //wrap head pointer
                ++nChars;           //increment number of characters
            }
            else
            {
                ExitCritical(sr);
                return EVENT_BUFFFULL;
            }
            --c;
        }
        if(EventCount++ >= 0)   //increment semaphore
        {
            ExitCritical(sr);
        }
        else
        {
            if(task_h)              //is there really a task waiting?
            {
                t = task_h;     //get first task in
```

```
                task_h = t->next;       //remove from list
                t->next = (TCB *)0;
                t->status = EVENT_NOERROR;          //return this value
                t->pending = 0;
                ActiveTasks->Insert(t); //put task back onto active list
                ExitCritical(sr);
                Schedule();
            }
            else
            {
                ExitCritical(sr);
            }
        }
        return EVENT_NOERROR;
    }
    else
    {
        ExitCritical(sr);
        return(EVENT_OVERFLOW);
    }
}

int EventQueue::Pend(int v)
{
    return TSemaphore::Pend(v);
}

int EventQueue::Get(void)
{
    int retval;
    int sr;

    if(nChars)
    {
        sr = EnterCritical();
        retval = Buff[Tail++];
        if(Tail >= Size)
            Tail = 0;       //wrap tail pointer
        --nChars;
        ExitCritical(sr);
    }
    else
        retval = -1;
    return retval;
}

int EventQueue::Put(int c)
{
    int retval = c;
    int sr;

    sr = EnterCritical();
    if(nChars < Size)
    {
        Buff[Head++] = (char)c;
        if(Head >= Size)
            Head = 0;       //wrap head pointer
        ++nChars;           //increment number of characters
    }
    else
        retval = -1;        //could not put character into buffer
    ExitCritical(sr);
    return retval;
}

int EventQueue::QueueStatus(void)   //number of char in queue
```

```
{
    return nChars;
}

int EventQueue::QueueSpace(void)    //how much space left in queue
{
    return Size - nChars;
}

void EventQueue::Kill(void)
{
    int sr;

    sr = EnterCritical();
    nChars = 0;
    Tail = Head;
    ExitCritical(sr);
}

//-----------------------------------------------
// Queue for talking to DSP 56002
//
// This is a queue of longs
//-----------------------------------------------

DspQueue::DspQueue(int size,char *name):TSemaphore(0,TSEMAPHORE_MODE_WAIT,name) //constructor fu
nction
{
    Buff = new long[size];
    Head = 0;
    Tail = 0;
    nChars = 0;
    Size = size;
}

DspQueue::~DspQueue()
{
    if(Buff)
        delete[] Buff;
}

void *DspQueue::operator new(size_t size)
{
    void *t = malloc(size + 128);
    ((DspQueue *)t)->name = ((char *)t) + size;
    return t;
}

void DspQueue::operator delete(void *t)
{
    free(t);
}

int DspQueue::Post(int v)
{
    return TSemaphore::Post(v);
}

int DspQueue::Pend(int t)
{
    return TSemaphore::Pend(t);
}

long DspQueue::Get(void)
{
    long retval;
```

```
    int sr;

    if(nChars)
    {
        sr = EnterCritical();
        retval = Buff[Tail++];
        if(Tail >= Size)
            Tail = 0;       //wrap tail pointer
        --nChars;
        ExitCritical(sr);
    }
    else
        retval = -1;
    return retval;
}

long DspQueue::Put(long c)  //this will always be done at interrupt level
{
    long retval = c;
    int sr;

    sr = EnterCritical();
    if(nChars < Size)
    {
        Buff[Head++] = c;
        if(Head >= Size)
            Head = 0;       //wrap head pointer
        ++nChars;           //increment number of characters
    }
    else
        retval = -1;        //could not put character into buffer
    ExitCritical(sr);
    return retval;
}

int DspQueue::QueueStatus(void)
{
    return nChars;
}
```

```
#include "task.h"

#ifndef QUEUE__H
#define QUEUE__H

//---------------------------------------------------------------------
// C++ Classes
//---------------------------------------------------------------------

class BiMessage:public TSemaphore {
    char **ToBuff;  //pointer to an array of character pointers
                    //sends data to the master process
    char **FromBuff;//pointer to an array of character pointers
                    //sends data back to requesting process
    TCB **Requestor;//pointer to an array of task pointers
                    //task that is requesting data
                    //Do I need this?
    int Head;       //head pointer for queue of messages
    int Tail;       //Tail pointer for queue of messages
    int nObj;       //number of objects in queue
    int QSize;      //max number of messages in Queue
    int Size;       //size of each message
    int *BTail;     //pointer to an array of tail pointers
    int *BnChars;   //pointer to an array of message char counts
public:
    BiMessage(int size,int qsize,char *name);
    ~BiMessage();
    int SendCommand(char *m,int c); //send a message to master process
    int SendReply(char *m,int c);   //send response back to requestor
    int GetCommand(void);           //master process gets char from requestor
    int GetReply(void);             //requestor gets char from master process
    void *operator new(size_t size);
    void operator delete(void *);
};

class BQueue:public TSemaphore {
    char **Buff;    //pointer to an array of character pointers
    int Head;       //head pointer for queue of messages
    int Tail;       //tail pointer for queue of messages
    int nObj;       //number of objects in queue
    int Qsize;      //size of queue
    int Size;       //size of messages
    int *BTail;     //pointer to array of tail pointers
    int *BnChars;   //number of chars in message
public:
    BQueue(int size,int qsize,char *name);
    ~BQueue();
    int SendMessage(char *m,int c); //send a message (does post)
    int GetMessage(void);               //get char from current message
    int WaitMessage(void);              //wait for message (pend)
    void *operator new(size_t size);
    void operator delete(void *);
};


class MessageQueue:public TSemaphore {
    int *Buff;      //buffer where data is stored
    int Head;       //head index
    int Tail;       //tail index
    int nChars;     //number of chars in queue
    int Size;       //size of queue
public:
    MessageQueue(int size,int mode,char *name); //constructor Function
    ~MessageQueue();
    virtual int Post(int v=0);
    virtual int Pend(int timeout=-1);
```

```
    int Get(void);
    int Put(int c);
    long GetLong(void);
    int PutLong(long c);
    int QueueStatus(void);
    int Space(void);
    void Flush(void);
    void *operator new(size_t size);
    void operator delete(void *);
};

class EventQueue:public TSemaphore {
    char *Buff;     //buffer where data is stored
    int Head;       //head index
    int Tail;       //tail index
    int nChars;     //number of chars in queue
    int Size;       //size of queue
public:
    EventQueue(int size,char *name);    //constructor function
    ~EventQueue();
    int PostMessage(int *d,int c);
    virtual int Post(int v=0);
    virtual int Pend(int t=0);
    int Get(void);
    int Put(int c);
    int QueueStatus(void);  //number of char in queue
    int QueueSpace(void);   //how much space left in queue
    void Kill(void);        //Delete Message from Queue
    void *operator new(size_t size);
    void operator delete(void *);
};

class DspQueue:public TSemaphore {
    long *Buff;     //buffer where data is stored
    int Head;       //head index
    int Tail;       //tail index
    int nChars;     //number of chars in queue
    int Size;       //size of queue
public:
    DspQueue(int size,char *name);  //constructor function
    ~DspQueue();
    virtual int Post(int v=0);
    virtual int Pend(int t=0);
    long Get(void);
    long Put(long c);
    int QueueStatus(void);
    void *operator new(size_t size);
    void operator delete(void *);
};

class Wait: public TSemaphore{
    //this is like a semaphore (ECB) but no timeout logic
    //for things where waiting forever is OK
    //these have pretty much zero CPU overhead
    int value;          /*  this could be dangerous */
public:
    Wait(int count,char *name);
    ~Wait();
    virtual int Pend(int timeout=0);
    virtual int Post(int v=0);
    void *operator new(size_t size);
    void operator delete(void *);
};

#endif
```

```
/*************************************************************************
**                                                                      **
**                                                                      **
** This is a rudimentary multitasking kernel that is integrated with    **
** the I/O module cio.cpp.  Provides the simplest services of           **
** creating tasks, both for absolute and relative data regions,         **
** creating and deleting events, interrupt processing.                  **
** This kernel is roughly based on uC/OS by Jean Labrosse, but by       **
** integrating it with I/O, expands it much more.                       **
**                                                                      **
** Other files that have to do with the Multitasking kernel are:        **
** CIO.CPP      .... provides I/O management                            **
** TRAP.S       .... provides access to CIO.CPP via software interrupt  **
** SWAP.S       .... handles context swaping and critical sections      **
** PQ.CPP       .... priority queue class for prioritizing tasks        **
** QUEUE.CPP    .... data pipes for communicating between tasks         **
**                                                                      **
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pq.h"
#include "task.h"
#include "cio.h"
#include "strings.h"

PQ *ActiveTasks;            //priority queue of active tasks
TCB *CurrentTask=0;         //current executing task
TCB *ReadyTask = 0;         //task that is ready to run
TCB *FreeTaskList = NULL;   //list of Tack Control Blocks that are not used
TCB *MasterTaskList = 0;    //master list of all current tasks

int InterruptCount=0;       //level of interrupt depth
int Blocking=0;             //level of task swap blocking
static int Multitasking=0;  //flag to indicate multitasking is enabled

//---------------------------------
// This is a list of all semaphores
// including those derived from
// the TSemaphore class
//---------------------------------

TSemaphore *TSemaphore::MasterList = (TSemaphore *)0;

//---------------------------------
// This is a list of the semaphores
// that have a timeout limit on how
// long they can stay pending.
//---------------------------------
TSemaphore *TSemaphore::TimeoutList = (TSemaphore *)0;
//---------------------------------
// This is a list of unused
// semaphores.  This keeps memory
// fragmentation down
//---------------------------------

TSemaphore *TSemaphore::PoolList = (TSemaphore *)0;

//---------------------------------
// This is a special semaphore which
// is used to control access to the
// constructor and destructor
// routines.
//---------------------------------
TSemaphore *TSemaphore::Blocker = (TSemaphore *)0;
```

```
long TSemaphore::PoolNewCount=0;    //number of times new was called
long TSemaphore::PoolDynamic=0; //number of times dynamic memory allocated

long TSemaphore::TStamp = 0l;   //time stamp counter

extern "C" void _Start(void);

//---------------------------------------------------------------------
//
// This function is used to compare two items in the priority queue
// to see which one has the highest priority
//
// The function is passes two pointers that point to the pointers
// that point to the actual objects
//
//---------------------------------------------------------------------
//
static int PriorityCompare(void **s1, void **s2)
{
    int r;

    //check priority levels
    if( (r = ((TCB *)*s1)->priority - ((TCB *)*s2)->priority) != 0)
        return r;
    //if priority levels are the same, then check time stamps
    return int(((TCB *)*s2)->TimeStamp - ((TCB *)*s1)->TimeStamp);
}

static int TaskCompare(void **s1,void **s2)
{
    int retval = 1;

    if(*s1 == *s2) retval = 0;
    return retval;
}

//---------------------------------------------------------------------
//
// Create a task
//
// void (*task)(void);      //this is the "main" routine for the task
// int stacksize;           //number of WORDs that are in the stack for the task
// int priority;            //0 is lowest priority, 32767 is highest priority
// char *name;              //this is used for debugging
//
// This function allocates space for the Stack in "stacksize" words
//
//---------------------------------------------------------------------

extern "C" TCB *CreateTask(void (*task)(void),int stacksize,int priority,char *name)
{

    TCB *t;
    int *stack;
    long *stk;
    int sr;

    sr = EnterCritical();   //disable interrupts while creatine TASK
    if((t = FreeTaskList))  //is there a free task?
    {
        FreeTaskList = t->next; //remove from free list
        if(t->datamem)      //did this used to be a relative task?
            free(t->datamem);
        if(t->stacksize < stacksize)
        {
```

```
            free(t->stacktop);  //get rid of memory
            stack = (int *)malloc(2 * stacksize);
        }
        else
        {
            stack = t->stacktop;
            stacksize = t->stacksize;   //larger stack is ok, but record it
        }
    }
    else    //no Task Blocks in pool, create a new task block
    {
        t = (TCB *)malloc((int)sizeof( TCB));   //allocate task
        stack = (int *)malloc(2 * stacksize);   //allocate stack space
    }
    if(t)   //if task blocks were successfully created, then....
    {
        t->TcbSwaps = 0;        //initialize swaps to 0
        t->datamem = (char *)0; //NULL pointer to data memory (unused)
        t->stacksize = stacksize;   //set stack size
        t->tasktype = TCB_ABSOLUTE; //set access mode to data memory
        t->priority = priority;     //set priority
        t->TimeStamp = TSemaphore::TStamp++;    //increment Time Stamp (kluge)
        t->name = name; //set task name
        t->stacktop = stack;    //set top of stack
        t->next = (TCB *)0;     //set next pointer to NULL
        t->list = MasterTaskList;   //add to master task list
        t->pending = 0;     //set pending semaphore to NULL
        t->TotalTime = 0l;  //set total execution time to 0
        t->StartTime = 0;   //set start time to 0
        MasterTaskList = t;         //complete addition to master task list
        stk = (long *)((int *)stack + stacksize);   //set stack pointer to bottom of stack
        *--stk = (long)task;    //push address of TASK function
        *--((int *)stk) = 0x2000;   /*  push status reg,supervisor mode,interrupts enabled  */
        *--stk = 0l;        /*  push a6 = 0 */
        *--stk = 0l;        /*  push a5 = 0 */
        *--stk = 0l;        /*  push a4 = 0 */
        *--stk = 0l;        /*  push a3 = 0 */
        *--stk = 0l;        /*  push a2 = 0 */
        *--stk = 0l;        /*  push a1 = 0 */
        *--stk = 0l;        /*  push a0 = 0 */
        *--stk = 0l;        /*  push d7 = 0 */
        *--stk = 0l;        /*  push d6 = 0 */
        *--stk = 0l;        /*  push d5 = 0 */
        *--stk = 0l;        /*  push d4 = 0 */
        *--stk = 0l;        /*  push d3 = 0 */
        *--stk = 0l;        /*  push d2 = 0 */
        *--stk = 0l;        /*  push d1 = 0 */
        *--stk = 0l;        /*  push d0 = 0 */
        t->stack = (int *)stk;  //save stack pointer
    }
    ExitCritical(sr);   //enable interrupts
    return t;           //return pointer to task
}

//---------------------------------------------------------------------
//
// Create a task that has position independent global data
//
// void (*task)(void);      //this is the "main" routine for the task
// int stacksize;           //number of WORDs that are in the stack for the task
// int priority;            //0 is lowest priority, 32767 is highest priority
// char *name;              //this is used for debugging
// long *data;              //this points to the intialization data for task
// long dsize;              //size of intialization data block in bytes
// long rsize;              //size of global ram in bytes
// void *param              //pointer sized parameter to pass to task
```

```
//
// This function allocates space for the Stack in "stacksize" words
// this function also creates a pointer to the data area and loads it into
// the stack location for address register 5 (a5).
//
// when you compile the code, use a #pragma to rename the 'ram' and 'data'
// regions to something else.  In the spec file you will need to find out the
// sizes of the the other data areas and use those sizes appropriately.
// The order of linking the data and ram area should be data area first.
// This function uses malloc to allocate a memory block that is
// 'dsize' + 'rsize' long.  The whole block is zeroed, and then the
// initialization data is downloaded.
//
// the file that contains the task to be created should be compiled with
// the following command line switch:
//  -rA5=_data
// This will cause the compiler to address global data relative to address
// register 5.
// When a task switch occures, this register is pushed onto the stack
// meaning that you can have as many independent instances of this task
// as you want.
//
//---------------------------------------------------------------------

//extern "C" TCB *CreateRelTask(void (*task)(void *),int stacksize,int priority,char *name,char
*initdata,int dsize,int rsize,void *param)
//{
//
//  TCB *t;
//  int *stack;
//  long *stk;
//  char *data;
//  int sr;
//
//  sr = EnterCritical();
//  if((t = FreeTaskList))  //is there a free task?
//  {
//      FreeTaskList = t->next; //remove from free list
//      if(t->datamem)      //did this used to be a relative task?
//          free(t->datamem);
//      if(t->stacksize < stacksize)
//      {
//          free(t->stacktop);  //get rid of memory
//          stack = (int *)malloc(2 * stacksize);
//      }
//      else
//      {
//          stack = t->stacktop;
//          stacksize = t->stacksize;   //larger stack is ok, but record it
//      }
//  }
//  else
//  {
//      t = (TCB *)malloc((int)sizeof( TCB));
//      stack = (int *)malloc(2 * stacksize);
//  }
//  if(t)
//  {
//      data = (char *)malloc(dsize + rsize);   /*  create data area    */
//      memset(data,0,dsize + rsize);   /*  zero memory     */
//      memcpy(data,initdata,dsize);    /*  copy in initialization data */
//      t->datamem = data;
//      t->TcbSwaps = 0;
//      t->tasktype = TCB_RELATIVE;
//      t->stacksize = stacksize;
//      t->priority = priority;
```

```
//      t->TimeStamp = TSemaphore::TStamp++;
//      t->name = name;
//      t->stacktop = stack;
//      t->next = (TCB *)0;
//      t->list = MasterTaskList;
//      MasterTaskList = t;
//      stk = (long *)((int *)stack + stacksize);
//      *--stk = (long)param;   /*  push parameter onto stack   */
//      *--stk = (long)task;    /*  simulates subroutine call, never used   */
//      *--stk = (long)task;    /*  address of routine to be run    */
//      *--((int *)stk) = 0x2000;   /*  supervisor mode,interrupts enabled  */
//      *--stk = 0l;        /*  a6 = 0  */
//      *--stk = (long)data;/*  a5 = data   */
//      *--stk = 0l;        /*  a4 = 0  */
//      *--stk = 0l;        /*  a3 = 0  */
//      *--stk = 0l;        /*  a2 = 0  */
//      *--stk = 0l;        /*  a1 = 0  */
//      *--stk = 0l;        /*  a0 = 0  */
//      *--stk = 0l;        /*  d7 = 0  */
//      *--stk = 0l;        /*  d6 = 0  */
//      *--stk = 0l;        /*  d5 = 0  */
//      *--stk = 0l;        /*  d4 = 0  */
//      *--stk = 0l;        /*  d3 = 0  */
//      *--stk = 0l;        /*  d2 = 0  */
//      *--stk = 0l;        /*  d1 = 0  */
//      *--stk = 0l;        /*  d0 = 0  */
//      t->stack = (int *)stk;
//  }
//  ExitCritical(sr);
//  return t;
//}


//*********************************************************************
//
// Delete Task
//
// This is REAL tricky
// memory cannot be free'd, technically, until the task is done using
// the stack, but it won't be done using the stack until it swaps
//
// parameter:
// task....pointer to the task to delete.  If task is a NULL pointer,
//         then the current task is deleted
// ReturnValue:
// Returns 0 on success, negative value on failure
// Returns -2 if it could not delete the task at all
// Returns -1 if it sent a notification to the pending semaphore that
// the task has been requested to kill itself
//-------------------------------------------------------------------
// Another tricky thing:  it is posible to delete a task that is pending
// on a semaphore.  It is nessesary to check all of the semaphores and
// remove any tasks waiting for a post, or else things will not be nice.
// 05-26-98
//
//-------------------------------------------------------------------
// This function still has some problems.  Need to come up with a way
// for a notification to be posted when the task is finally deleted,
// since it may not sometime happen right away.
//
//*********************************************************************


int TDelete(TCB *task)
{
    int retval=0;
```

```
    int sr;
    TCB *deletedtask=0;

    TBlock();
    TCB *t,**prev;
    int loop;
    if(task == NULL) task = CurrentTask;    //null value= delete current task
    for(prev = &MasterTaskList,t = MasterTaskList,loop=1;t && loop;)
    {
        if(t == task)   //is this the task?
        {
            *prev = t->list;
            loop = 0;
        }
        else
        {
            prev = &(t->list);
            t = t->list;
        }
    }
    TRelease();
    if(task == CurrentTask) //delete current task?
    {
        //if we end up here, the task can not posibly be pending
        //it cannot posibly be in the priority QUEUE
        //if we end up here, this call will NEVER return, tasks will
        //have to be resheduled
        EnterCritical();        //this task is going away, don't need to save SR
        CurrentTask->next = FreeTaskList;
        FreeTaskList = CurrentTask;
        ActiveTasks->Delete((void **)&ReadyTask);
        asm(" trap #0");    //reschedule
    }
    else    //delete a task that is not active
    {
        //we have to do one of several posible things.
        //1. check the priority QUEUE and see if the task lives in there
        //2. check the list of semaphores and see if the task is pending
        //This code has not been verified
        sr = EnterCritical();
        if(task->pending == 0)
        {
            /*
           ** the task is not pending on anything
           */
            ActiveTasks->Remove((void **)&deletedtask,TaskCompare,(void *)task);    //see if tas
k can be removed
            if(deletedtask) //we found the task
            {
                deletedtask->next = FreeTaskList;
                FreeTaskList = deletedtask;
            }
            else
            {
                extern int ConsolHandle;
//              char *s = new char[256];
//              int c;
//              c = sprintf(s,"Could not delete task %s\r\n",task->name);
//              Write(ConsolHandle,s,c);
//              delete[] s;
                retval = -2;
            }
        }
        else
        {
            /*
```

```
            ** the task is pending on a semaphore, for
            ** the semaphore to post and send a kill
            ** notification
            */
             task->pending->Kill();
             retval = -1;
        }
        ExitCritical(sr);
    }
    return retval;      //no errors
}

/*---------------------------------------------------------------------
**
** EnterInterrupt
**
**  This function keeps track of the nesting level of interrupts
**
**  This function requires no blocking BECAUSE the 68000 can do
**  This operation in one instruction so it is not interruptable
**
** parameter:
** <none>
** ReturnValue:
** <none>
**
**---------------------------------------------------------------------*/

extern "C" void EnterInterrupt(void)
{
    ++InterruptCount;
}

/*********************************************************************
**                                                                  **
** ExitInterrupt                                                    **
**                                                                  **
** This function keeps track of the nesting level of interrupts     **
** If nesting level is == 0, then do a context swap to the highest  **
** priority task                                                    **
**                                                                  **
** WARNING!                                                         **
**  Don't FUCK with this function, you will be sorry                **
**  Adding any local variables or passed parameters                 **
**  Messes with the stack.  IrqSwap alters the stack and            **
**  if it is different from what it expects, it will fuck           **
**  up the stack.                                                   **
**                                                                  **
** Hey, you don't even need to add any local variables, just add a  **
** Little bit more code and it will fuck things up.  Always check   **
** the listing file to see how much might get pushed onto the stack **
**                                                                  **
** Just as a point of reference                                     **
**  JSR to IrqSwap uses up 4 bytes on stack                         **
**  JSR to ExitInt uses up 4 bytes on stack                         **
**  IntLevel       uses up 2 bytes on stack                         **
**                        10 bytes total                            **
**                                                                  **
** ExitInterrupt will alocate at least 4 bytes on the stack probably**
** for sr, depending on how it does this.                           **
** Currently ExitInterrupt uses 4 bytes on the stack to save various**
** registers that it doesn't really have to                         **
**                                                                  **
** Currently IrqSwap() needs to remove 14 bytes from stack          **
**                                                                  **
*********************************************************************/
```

```
extern "C" void ExitInterrupt(int IntLevel)
{
    int sr;

    --InterruptCount;
    sr = EnterCritical();
    if(!IntLevel && !Blocking)  //don't swap if nested
    {
        CurrentTask->TimeStamp = TSemaphore::TStamp;
        ActiveTasks->Replace((void **)&ReadyTask,CurrentTask);
        if(ReadyTask != CurrentTask)
        {
            ReadyTask->TcbSwaps++;
            IrqSwap();  //interrupts will be re-enabled right after this
        }
    }
    ExitCritical(sr);
}

/*********************************************************************
** TimerTicker                                                      **
**                                                                  **
**  This function is called by the real time interrupt handler      **
**  This function Increments the clock, scans through the Event     **
**  Control Blocks looking for timed out events.                    **
** Parameter:                                                       **
** <none>                                                           **
** ReturnValue:                                                     **
** <none>                                                           **
**                                                                  **
*********************************************************************/

extern "C" void TimerTicker(void)
{
    TSemaphore::Timeout();
}

/*********************************************************************
**                                                                  **
** Start                                                            **
**                                                                  **
** This function dequeues the highest priority task and then starts **
** it.  This function gets the whole ball of wax moving             **
**                                                                  **
** Parameter:                                                       **
** <none>                                                           **
** ReturnValue:                                                     **
** This function will never return                                  **
**                                                                  **
*********************************************************************/

void Start(void)
{
    TCB *nexttask;

    EnterCritical();
    ActiveTasks->Delete((void **)&ReadyTask);   //get a task
    ReadyTask->TcbSwaps++;                      //our first swap
    Multitasking = 1;
    _Start();                                   //initial context swap
}

/*********************************************************************
**                                                                  **
** Schedule                                                         **
```

```
**                                                                    **
** This function reschedules a task at the task level.  It puts the **
** current task back into the active task list and dequeues the     **
** highest priority task.                                           **
**                                                                    **
** It updates the time stamp at the same time                       **
**                                                                    **
** Parameter:                                                       **
** <none>                                                           **
** ReturnValue:                                                     **
** <none>                                                           **
**                                                                    **
**********************************************************************/

extern "C" void Schedule(void)
{
    int sr;

    ++TSemaphore::TStamp;   //increment time stamp...this is kind of a kluge
    sr = EnterCritical();   //disable interrupts
    if(!InterruptCount && !Blocking)    //ok to swap?
    {
        CurrentTask->TimeStamp = TSemaphore::TStamp;    //set task timestamp to current
        ActiveTasks->Replace((void **)&ReadyTask,CurrentTask);  //get the next highest priority
task
        if(ReadyTask != CurrentTask)    //if new task is different, increment swap counter
        {
            ReadyTask->TcbSwaps++;
            asm(" trap #0");            //peform swap
        }
    }
    ExitCritical(sr);                   //exit here if no swap took place
}

/**********************************************************************
**                                                                    **
** InitOS                                                           **
**                                                                    **
** This function must be called before starting the multitasking    **
**                                                                    **
**********************************************************************/
void InitOS(void)
{
    InitDisable();
    ActiveTasks = new PQ(MAX_TASKS,PriorityCompare);    //initialize priority queue
}

/**********************************************************************
**                                                                    **
** Multitasking Control Routines                                    **
**                                                                    **
**********************************************************************
**                                                                    **
** TBlock                                                           **
**                                                                    **
**  This routine keeps track of blocking level                      **
**                                                                    **
**  If Blocking is non zero, multitasking swapping is blocked       **
**                                                                    **
**                                                                    **
**********************************************************************/

extern "C" void TBlock(void)
{
    ++Blocking;
}
```

```
/*************************************************************************
**                                                                      **
** TRelease                                                             **
**                                                                      **
** This routine keeps track of blocking level                           **
**                                                                      **
** If Blocking is non zero, mulitasking swapping is blocked             **
**                                                                      **
*************************************************************************/

extern "C" void TRelease(void)
{
    --Blocking;
}

/*************************************************************************
**
** TYeild
**
** Give up control of cpu to next highest priority task
**
*************************************************************************/

//extern "C" void TYeild(void)
//{
//  int sr;
//
//  sr = EnterCritical();
//  if(ActiveTasks->Delete((void **)&ReadyTask))
//  {
//      ActiveTasks->Insert(CurrentTask);
//      asm(" trap #0");        //do context swap
//  }
//  ExitCritical(sr);
//}

/*************************************************************************
**
** TDelay
**
** Give up control of cpu for 'time' system clock ticks
**
** Parameter:
** time....amount of time to put task to sleep for in "ticks"
** ReturnValue:
** Returns 0 on success, negative on error...more than likely the only
** error notification will be a request to kill the current task
**
*************************************************************************/

extern "C" int TDelay(int time)
{
    int error;
    static unsigned delaycount=0;
    String *S = new String;
    char *s = S->Get();

    sprintf(s,"TDelay%u",delaycount++);
    TSemaphore *d = new TSemaphore(0,1,s);
    if((error = d->Pend(time)) == EVENT_TIMEOUT)
        error = 0;
    delete d;
    delete S;
    return error;
}
```

```
/*************************************************************************
**                                                                      **
** TSemaphore Base Class Function Definitions                           **
**                                                                      **
** Class Public Memeber Functions:                                      **
**                                                                      **
**  TSemaphore(int InitCount,int mode,char *name);  //constructor       **
**  ~TSemaphore();                                  //destructor        **
**  virtual int Pend(int Timeout);                                      **
**  virtual int Post(int Value);                                        **
**  virtual int GetCount(void);                                         **
**  virtual void SetCount(int Count);                                   **
**                                                                      **
** Semaphore is named to make debugging just a little easier, maybe.    **
**                                                                      **
*************************************************************************/

TSemaphore::TSemaphore(int InitCount,int Mode,char *n)
{
    /*********************************************************************
    ** TSemaphore constructor
    ** This creates a TSemaphore object and initializes the data members
    ** that need to be initialized
    **
    ** Parameter:
    ** InitCount....This is used to set the EventCount data member.  The
    **              value will determine a lot of the behavior of the
    **              semaphore.  For example, if set to 32, you will be
    **              able to call Pend 32 times before the task will suspend
    ** Mode.........This is a flag.  If mode=0, then the semaphore will
    **              not be put into the list that checks for a timeout.
    **              If mode=1, the the semaphore is put into the list
    **              that is checked for timeouts.  This value is used to
    **              set the SemaphoreMode data member.
    ** n............Pointer to a character string that contains the name
    **              of the semaphore.
    ** ReturnValue:
    ** <none>
    *********************************************************************/
    int sr;

    EventCount = InitCount;
    SemaphoreMode = Mode;
//  name = new char[strlen(n)+1];   //allocate space for semaphore name
    if(n) strcpy(name,n);                    //copy name in
    task_h = 0;                       //no tasks waiting
    task_t = 0;                       //no tasks waiting
    //-----------------------------------
    // put new semaphore into master list
    //-----------------------------------
    sr = EnterCritical();   //do not disturbe this process
    if(MasterList)  //is list already started?
    {
        Mnext = MasterList;     //make current head, next in line
        Mprev = 0;              //there is no previous entry
        MasterList->Mprev = this;   //this will be prev for next entry
        MasterList = this;      //this is now the new head of list
    }
    else            //No, start list
    {
        MasterList = this;
        Mnext = 0;  //nothing follows
        Mprev = 0;  //nothing preceeds
    }
    //-------------------------------------------------------
```

```
    // if Mode==1, then put this semaphore into timeout list
    //-----------------------------------------------------
    if(SemaphoreMode == 1)
    {
        if(TimeoutList) //is list already started?
        {
            Tnext = TimeoutList;    //make current head next in line
            Tprev = 0;          //these is no previous entry
            TimeoutList->Tprev = this;  //this will be prev for next entry
            TimeoutList = this;         //this is now the new head of list
        }
        else            //no start list
        {
            TimeoutList = this;
            Tnext = 0;
            Tprev = 0;
        }
    }
    ExitCritical(sr);   //restore interrupts
}

TSemaphore::~TSemaphore()
{
    /*************************************************************************
    ** TSemaphore destructor
    **
    ** This function does all of the clean up needed when a TSemaphore object
    ** is no longer needed.
    *************************************************************************/
    int sr;

//  delete[] name;      //free up space used by semaphore name
    //-------------------------------------------
    // remove this instance from the master lists
    //-------------------------------------------
    sr = EnterCritical();   //don't want to mess this up
    if(this == MasterList)  //is it really the head pointer?
        MasterList = Mnext; //new head of list
    if(Mprev)
        Mprev->Mnext = Mnext;
    if(Mnext)
        Mnext->Mprev = Mprev;
    //-----------------------------------------------------
    // if timeout mode, remove instance from timeout list
    //-----------------------------------------------------
    if(SemaphoreMode)
    {
        if(this == TimeoutList)
            TimeoutList = Tnext;
        if(Tprev)
            Tprev->Tnext = Tnext;
        if(Tnext)
            Tnext->Tprev = Tprev;
    }
    ExitCritical(sr);
}

void *TSemaphore::operator new(size_t size)
{
    /*************************************************************************
    ** new operator for the TSemaphore class.
    ** This new operator allows TSemaphore to be allocated from a pool of
    ** TSemaphore objects that have previously been allocated.
    ** This will minimize the amount of memory fragmentation that will
    ** occure.
    **
```

```
    ** Parameter:
    ** size......size of memory block to allocate
    ** ReturnValue:
    ** pointer to block of memory that is allocated
    ************************************************************************/
    void *t;
    static char bs[256];    //this is where we will put the Blocker Semaphore

    PoolNewCount++; //number of times new was called
    if(!Blocker)    //has the blocking semaphore been created yet?
    {
        TSemaphore B(1,0,NULL); //this is to fool the compiler
                                //we need to create a semaphore
                                //before we create any semaphores
        //catch 22, we need to create a semaphore, but here we are
        //inside of the new operator for a semaphore, what do we
        //do, well, we fake it.
        //
        // Please note that this very well may NOT be the best way
        // to solve this problem.  But, it seemed to me to be the
        // best way currently.
        memcpy(bs,&B,sizeof(TSemaphore));       //copy semaphore data
        Blocker = (TSemaphore *)bs;             //set address of semaphore
        strcpy(&bs[sizeof(TSemaphore)],"SemBlock"); //copy semaphore name
        Blocker->name = &bs[sizeof(TSemaphore)];    //set name pointer to name
        // OK, don't laugh.  This is a REAL KLUGE!!!!  But for now,
        // I can't figure out anyother way of doing this relink.
        // This had better be the very first semaphore created.
        MasterList = (TSemaphore *)bs;  //put into master list
        MasterList->Mnext = 0;  //nothing follows
        MasterList->Mprev = 0;  //nothing preceeds

    }
    if(Multitasking)Blocker->Pend();
    if(!PoolList)   //nothing in pool
    {
        PoolDynamic++;  //number of times dynamic memory allocated
        t = malloc(size + 128); //allocate space, add an extra 128 bytes for name
        ((TSemaphore *)t)->name = ((char *)(t)) + size; //make pointer to name
    }
    else
    {
        t = (void *)PoolList;   //get TSemaphore object from pool
        PoolList = PoolList->next;  //unlink TSemaphore object from pool
    }
    if(Multitasking)Blocker->Post();
    return t;
}

void TSemaphore::operator delete(void *t)
{
    /************************************************************************
    ** delete operator for the TSemaphore class
    ** Rather than return the memory blocks to the malloc pool, create a
    ** pool of TSemaphore objects to be reallocated later
    **
    ** Parameter:
    ** t.......Semaphore object to delete
    ** ReturnValue:
    ** <none>
    **
    ************************************************************************/
    if(Multitasking)Blocker->Pend();
    ((TSemaphore *)t)->next = PoolList; //link t into pool list.
    PoolList = (TSemaphore *)t;         //put t at the head of the list
    if(Multitasking)Blocker->Post();
```

```
}

int TSemaphore::Pend(int Timeout)   //wait for semaphore availiable
{
    /*********************************************************************
    ** This is the PEND member function.
    ** This function checks the EventCount to see if it is greater than
    ** zero, and then it decrements the EventCount.  If EventCount was
    ** greater than zero, that means that the reasource was availiable
    ** and nothing happens.  If EventCount was less than zero, then the
    ** current task is put into a linked list, and a task swap is performed
    ** , thus suspending the current task.
    **
    ** Parameter:
    ** Timeout.....number of system ticks to wait before reposting a
    **             timeout notification
    ** ReturnValue:
    ** return 0 on no error, nonzero otherwise
    *********************************************************************/
    int sr;
    int retval;
    sr = EnterCritical();
    if(EventCount-- > 0)    /*  is resource availiable? */
    {
        /*  yes */
        ExitCritical(sr);
        retval = EVENT_NOERROR; //everything is peachy
    }
    else
    {
        /*  no  */
        if(!task_h)     //make linked list of waiting tasks
            task_h = CurrentTask;
        else
            task_t->next = CurrentTask;
        task_t = CurrentTask;
        CurrentTask->status = EVENT_NOERROR;
        CurrentTask->pending = this;    //keep track of which task is pending
        if(SemaphoreMode)
            CurrentTask->timeout = Timeout;

        //swap out current task

        CurrentTask->TimeStamp = TSemaphore::TStamp;    //set the time stamp
        if(ActiveTasks->Delete((void **)&ReadyTask))    //get highest priority task
        {
            ReadyTask->TcbSwaps++;
            asm(" trap #0");                            //do context swap
        }
        else
        {
            CurrentTask->status = EVENT_NOTASKS;
        }
        //-----------------------
        // Pend is over, check status
        //-----------------------

        ExitCritical(sr);
        retval = CurrentTask->status;   //return this value
    }
    return retval;
}

int TSemaphore::Post(int Value) //signal semaphore availiable
{
    /*********************************************************************
```

```
    ** This is the POST function.
    ** This function releases a PEND.
    ** If EventCount is less than zero, POST will unlink a waiting task
    ** from the linked list.  It does this on a first in, first out basis.
    ** If EventCount is greater than or equal to zero, it then does nothing
    ** In both cases, EventCount is incremented.
    **
    ** Parameter:
    ** Value.....This value is copied into TCB->status.  This value is then
    **           returned by PEND
    ** ReturnValue:
    ** Returns 0 on success, non-zero of otherwise
    ***************************************************************************/
    TCB *t;
    int sr;

    sr = EnterCritical();
    if(EventCount < 32766)  //is it going to overflow??
    {
        if(EventCount++ >= 0)   //increment semaphore
        {
            ExitCritical(sr);
        }
        else    //if the Event Count is less than 0, put waiting task into Active List
        {
            if(task_h)              //is there really a task waiting?
            {
                t = task_h;     //get first task in
                task_h = t->next;       //remove from list
                t->next = (TCB *)0;
                t->status = Value;          //return this value
                t->pending = 0;             //no longer pending
                ActiveTasks->Insert(t); //put task back onto active list
                ExitCritical(sr);
                Schedule();
            }
            else
            {
                ExitCritical(sr);
            }
        }
        return EVENT_NOERROR;
    }
    else
    {
        ExitCritical(sr);
        return(EVENT_OVERFLOW);
    }
}

int TSemaphore::GetCount(void)      //get event count
{
    /***************************************************************************
    ** This function returns the value of EventCount.
    **
    ** Parameter:
    ** <none>
    ** ReturnValue:
    ** Returns value of EventCount.
    ***************************************************************************/
    return EventCount;
}

void TSemaphore::SetCount(int C)    //set event count
{
    /***************************************************************************
```

```
    ** This function is used to set the value of EventCount.  DANGER!
    **
    ** Parameter:
    ** C..........This is the value to set EventCount to
    ** ReturnValue:
    ** <none>
    ***************************************************************/
    EventCount = C;
}

void TSemaphore::Kill(void)         //send a kill message
{
    /***************************************************************
    ** This fucntion is used to send a TERMINATE (DELETE) notification
    ** to a TASK.  This is primarily used on tasks that are pending on
    ** a semaphore.
    **
    ** Parameter:
    ** <none>
    ** ReturnValue:
    ** <none>
    ***************************************************************/
    Post(EVENT_TERMINATE);
}


void TSemaphore::Timeout(void)          //checks timeout of all semaphores
{
    //*********************************************************
    //
    // This function replaces TimerTicker
    // This function scans the semaphores in the timeout list
    // if a task(s) is pending on the semaphore, the task timeout
    // data member is decremented.  If the timeout goes to zero,
    // the task is posted and given an EVEN_TIMEOUT status.
    //
    // parameter:
    // <none>
    // returnValue:
    // <none>
    //
    //*********************************************************

    int sr;
    ++TSemaphore::TStamp;
    //we need to process the list of tasks waiting
    //for queues
    TSemaphore *e;
    TCB *t,**prev;

    sr = EnterCritical();
    for(e = TimeoutList;e;e=e->Tnext)   //go through whole list
    {
        for(prev = &(e->task_h),t = e->task_h;t;)
        {
            if(t->timeout > 0)  //if less than zero, infinate timeout
            {
                if( --(t->timeout) <= 0)    //decrement timeout counter
                {
                    e->EventCount++;            //just like a post V1.62.10
                    t->status = EVENT_TIMEOUT;  //event caused by semaphore
                    *prev = t->next;
                    t->next = 0;
                    t->pending = 0;
                    ActiveTasks->Insert(t);     //put task in active queue
                }
```

```
            }
            prev = &(t->next);
            t = t->next;
        }
    }
    ExitCritical(sr);
}

void TSemaphore::PrintInfo(void)
{
    //-----------------------------------------
    // print semaphore info for debug purposes
    //-----------------------------------------
    char *s = new char[256];
    int c;
    int i;
    TCB *t;
    extern int ConsolHandle;    //global handle for debug consol

    c = sprintf(s,"SEMAPHORE:%s\r\n",name);
    Write(ConsolHandle,s,c);
    c = sprintf(s,"\tEventCount:%d\r\n",EventCount);
    Write(ConsolHandle,s,c);
    for(i=1,t = task_h;t && (i < 10);++i,t = t->next)
    {
        c=sprintf(s,"TASK%d:%s Timeout->%d\r\n",i,t->name,t->timeout);
        Write(ConsolHandle,s,c);
    }
    Write(ConsolHandle,"-------Global Info--------------\r\n",34);
    c = sprintf(s,"Total Number of TSEMAPHORE news:%ld\r\n",PoolNewCount);
    Write(ConsolHandle,s,c);
    c = sprintf(s,"Number of allocations from heap:%ld\r\n",PoolDynamic);
    Write(ConsolHandle,s,c);
    Write(ConsolHandle,"--------------------------------\r\n",34);
    delete[] s;
}

TSemaphore *TSemaphore::FindSemaphore(char *n)
{
    //-----------------------------------------------------------------
    // This is a debug function
    //-----------------------------------------------------------------
    TSemaphore *e;
    int sr = EnterCritical();
    for(e = MasterList;e;e=e->Mnext)
    {
        if(strcmp(n,e->name) == 0)
            goto exit;
    }
exit:
    ExitCritical(sr);
    return e;
}

TSemaphore *TSemaphore::FindTask(TCB *task)
{
    /*****************************************************************
    **
    ** find the semaphore that "task" is waiting for
    ** This is a debug function
    **
    *****************************************************************/
    TSemaphore *e;
    TCB *t;
    int sr = EnterCritical();
```

```
    for(e = MasterList;e;e = e->Mnext)  //go through all of the semaphores
    {
        for(t = e->task_h;t;t = t->next)
        {
            if(t == task)
                goto exit;
        }
    }
exit:
    ExitCritical(sr);
    return e;
}
```

```
/*****************************************
** header file for TASK
*****************************************/

#ifndef TASK__H
#define TASK__H

#include "erorcode.h"
#include "pq.h"

//system defines


#define MAX_TASKS   32      //maximum number of tasks (arbitrary)

/*-------------------------------------------------------------------
** Definition of a Task Control Block (TCB)
**-------------------------------------------------------------------*/

typedef struct tcb {
    int *stack;         /* Offset 0 points to current position in stack */
    int *stacktop;      /* Offset 4 points to the end of the stack      */
    int StartTime;      /* Offset 8 Time when task started              */
    long TotalTime;     /* Offset 10    Total Execution time of task        */
    char *datamem;      /* Offset 14    pointer to relative data memory     */
    int stacksize;      /*  SIZE of the stack in words          */
    int tasktype;       /*  absolute or relative data           */
    int priority;       /*  task priority   */
    int status;         /*  task status, set by semaphore mostly    */
    long TimeStamp;     /*  system time stamp, set when swapped */
    long TcbSwaps;      /*  number of times tasks swapped in    */
    int timeout;        /*  timeout counter to trigger event    */
    char *name;         /*  name of task    */
    struct tcb *next;
    struct tcb *list;   /*  master linked list of tasks */
    class TSemaphore *pending;  /*  semaphore task is pending on    */
} TCB;

typedef struct {
    long Time;          //total exection time
    char *name;         //task name
}TSTATS;


#define TCB_ABSOLUTE    0   //absolute memory references
#define TCB_RELATIVE    1   //RElative memory references

/*-------------------------------------------------------------------
**
** Event Control Block
**
**-------------------------------------------------------------------*/

class TSemaphore {
    int SemaphoreMode;      //mode of semaphore
    static TSemaphore *Blocker;     //to prevent more than one access to new TSemaphore
    static TSemaphore *MasterList;  //head pointer for master semaphore list
    static TSemaphore *TimeoutList; //head pointer for timeout semaphore list
    static TSemaphore *PoolList;    //TSemaphore pool
    TSemaphore *next;               //next item in pool
    TSemaphore *Mnext,*Mprev;   //Link list pointers for master list
    TSemaphore *Tnext,*Tprev;   //Link list pointers for timeout list
    static long PoolNewCount;   //number of times new was called
    static long PoolDynamic;    //number of times dynamic memory allocated
protected:
    char *name;
```

```
    int EventCount;         //counter for semaphore
    TCB *task_h;        //head pointer for list of waiting tasks
    TCB *task_t;        //tail pointer for list of waiting tasks
public:
    TSemaphore(int InitCount,int Mode,char *name);
    ~TSemaphore();
    virtual int Pend(int Timeout=0);    //wait for semaphore availiable
    virtual int Post(int Value=0);  //signal semaphore availiable
    virtual int GetCount(void);     //get event count
    virtual void SetCount(int C);   //set event count
    virtual void Kill(void);        //send a kill message to task
    virtual void PrintInfo(void);       //print semaphore info
    static void Timeout(void);          //checks timeout of all semaphores
    static long TStamp;         //time stamp variable
    static TSemaphore *FindSemaphore(char *name);
    static TSemaphore *FindTask(TCB *task); //find semaphore that task is waiting for
    void *operator new(size_t size);
    void operator delete(void *);
};

//semaphore contructor modes
#define TSEMAPHORE_MODE_WAIT        0
#define TSEMAPHORE_MODE_TIMEOUT     1

//---------------------------------------------------------------------
/* global variables */
//---------------------------------------------------------------------

extern PQ *ActiveTasks;     //priority queue of active tasks
extern TCB *MasterTaskList;         //list of all tasks
extern TCB *CurrentTask;    //Current Task executing
extern TCB *ReadyTask;      //task that is ready to execute
extern int InterruptCount;
extern int Blocking;        //Blocking count, if true swaps blocked

/*************************************************************************
**                                                                      **
** Function Prototypes                                                  **
**                                                                      **
*************************************************************************/

#ifdef __cplusplus
extern "C" {
#endif

extern TCB *CreateTask(void (*task)(void),int stacksize,int priority,char *name);
extern TCB *CreateRelTask(void (*task)(void *),int stacksize,int priority,char *name,char *initd
ata,int dsize,int rsize,void *param);
extern int TDelete(TCB *task);      //delete task
extern void EnterInterrupt(void);   //called when interrupt routine entered
extern void ExitInterrupt(void);    //called when interrupt routine exited
extern void Swap(void);             //Context Swap (do not use)
extern void IrqSwap(void);          //Context swap for interrupt (do not use)
extern void Schedule(void);         //re schedule task (do not use)
extern void InitOS(void);           //Initialize Kernel
extern void Start(void);            //Start Multitasking
extern int EnterCritical(void);     //enter critical area
extern void ExitCritical(int);      //exit critical area
extern void TBlock(void);           //Block context swaps
extern void TRelease(void);         //unblock context swap
extern void TYeild(void);           //current tasks yield to next priority task
extern int TDelay(int time);        //delay a task for a period of time

extern void InitDisable(void);      //initialize semaphore to prevent
                                    //re-entrancy to malloc/free
extern void Disable(void);          //Pend on semaphore if malloc in use
```

```
extern void Enable(void);           //Post semaphore when malloc done

#ifdef __cplusplus
}
#endif

#endif
```

```
;---------------------------------------------------------------------
;;
; Swap.s
;
; Context Swapper and starter
; Enter and Exit critical section
;
;---------------------------------------------------------------------

    SECTION code
    xref _CurrentTask,_Schedule,_ReadyTask
    xdef _Swap,__Start,Swap,_IrqSwap,_EnterCritical
    .FREF   _ExitInterrupt,2    ;this function removes two bytes
    .FDEF   _ExitCritical,2     ;this function removes two bytes
    xref _EnterInterrupt

RD_TIME =   0xffdc0018          ;read timer in spindle controller chip
START_TIME  =   8
TOTAL_TIME  =   10
;---------------------------------------------------------------------
;
; _Swap can be called from the task level by a JSR to do a context swap
;
;---------------------------------------------------------------------
;
_Swap:
    move.w  sr,-(a7)                ;push status register on stack (simulate trap)
;
;---------------------------------------------------------------------
;
; Swap can be called from the task level by a TRAP #0 instruction
;
;---------------------------------------------------------------------
;
Swap:
    movem.l d0-d7/a0-a6,-(a7)       ;save current task's context
    move.l  _CurrentTask,a0         ;get pointer to stack pointer save area
    move.w  RD_TIME,d0              ;read timer to get end time
    sub.w   START_TIME(a0),d0       ;subtract start time
    and.l   #0x0000ffff,d0          ;is this needed? YES!
    add.l   d0,TOTAL_TIME(a0)       ;create sum
    move.l  a7,(a0)                 ;save the stack pointer

    move.l  _ReadyTask,a0           ;get pointer to stack pointer save area
    move.l  a0,_CurrentTask         ;make ready task current task
    move.w  RD_TIME,d0              ;get start time
    move.w  d0,START_TIME(a0)       ;save in start time
    move.l  (a0),a7                 ;get new stack pointer
    movem.l (a7)+,d0-d7/a0-a6       ;restore context
    rte                             ;return to new task
;
;---------------------------------------------------------------------
;
; This routine is called to do a context swap from the interrupt level
;
; It is called from ExitInterrupt via a JSR
; This routine assumes that all registers were pushed onto the stack
; when the interrupt service routine was entered
;
;---------------------------------------------------------------------
;
_IrqSwap:
    lea     14(a7),a7               ;Ignore returns from calls to
                                    ;_IrqSwap,ExitInterrupt
    move.l  _CurrentTask,a0         ;get pointer to stack pointer save area
    move.w  RD_TIME,d0              ;read timer to get end time
```

```
    sub.w   START_TIME(a0),d0       ;subtract start time
    and.l   #0x0000ffff,d0          ;is this needed?  YES!
    add.l   d0,TOTAL_TIME(a0)       ;create sum
    move.l  a7,(a0)                 ;save the stack pointer
    move.l  _ReadyTask,a0           ;get pointer to stack pointer of new task
    move.l  a0,_CurrentTask         ;Set Current Task=New Task
    move.w  RD_TIME,d0              ;get start time
    move.w  d0,START_TIME(a0)       ;save in start time
    move.l  (a0),a7                 ;get new stack pointer
    movem.l (a7)+,d0-d7/a0-a6       ;restore context
    rte                             ;return to new task

;-------------------------------------------------------------------------
;
; This routine is called to run the first task and get multitasking
; running.
; This is called from the task level with a JSR
;
;-------------------------------------------------------------------------

__Start:
    move.l  _ReadyTask,a0
    move.l  a0,_CurrentTask         ;set currently running task
    move.w  RD_TIME,d0              ;get start time
    move.w  d0,START_TIME(a0)       ;save in start time
    move.l  (a0),a7                 ;set stack pointer
    movem.l (a7)+,d0-d7/a0-a6       ;pop registers from stack
    rte                             ;run the task

;-------------------------------------------------------------------------
;
; Critical Enter and Exit routines.  We need to disable the interrupts
; and then re-enable them to the same interrupt level
;
;-------------------------------------------------------------------------


;-------------------------------------------------------------------------
; Disable interrupts, and return back interrupt level
;-------------------------------------------------------------------------
_EnterCritical:
    move    sr,d0           ;move status register to D0 to Return
    ori.w   #0x0700,sr      ;disable interrupts
    and.w   #0x0700,d0      ;mask off all other garbage
    rts

_ExitCritical:
;-------------------------------------------------------------------------
; this function recieves back the level to which to restore interrupts
;-------------------------------------------------------------------------
; %+4(A7) <- m
    move    sr,d0           ;get the status register
    and     #0xf8ff,d0      ;zero the interrupt level bits
    or      4(A7),d0        ;restore previous interrupt level
    move    d0,sr           ;restore status register
    MOVE.L  (A7)+,A0        ;get return address
    ADDQ.L  #2,A7           ;clean up stack
    JMP (A0)                ;return

    END
```

```
    XDEF Trap1
    .FREF _cio,4

#define ADD_HANDLER 1   /*  add handler to device table */
#define OPEN        3   /*  open a device       */
#define READ        5   /*  read a buffer       */
#define GETC        7   /*  get a character     */
#define WRITE       9   /*  write a buffer      */
#define PUTC        11  /*  put a character     */
#define CLOSE       12  /*  close a device      */
#define STATUS      13  /*  get device status   */

    SECTION code


;*************************************************************************
;
; Trap1
; CENTRAL I-O (CIO) ENTRY POINT
;
; Trap entry point.  Entry through the use of trap #1 function
;
; gets the status register off of the stack
; determines if system is in supervisor or user mode
; locates a posible parameter area
; pushes the pointer to the parameter area onto the stack
; Call the routine that is going to handle this trap
;
;*************************************************************************

Trap1
    move.w      (a7),d0                 ;get status register from stack
    btst.l      #13,d0                  ;test suppervisor bit
    beq.s       UserMode                ;if not supevisor, user mode
    lea         6(a7),a0                ;load a0 with address of parameters
    bra.s       Trap0_1
UserMode
    move.l      usp,a0                  ;load user stack pointer to a0
Trap0_1
    move.l      (a0),-(a7)              ;push parameter pointer onto stack
    jsr         _cio                    ;call cio processing routine
    rte

    ;*************************************************************
    ; long AddHandler(char *dev_name,H_JVEC *jump_table);
    ;
    ; This function is passed 8 bytes, and the compiler treats it
    ; like 8 bytes, but we clean up 10 bytes from stack
    ;*************************************************************

    .FDEF _AddHandler,8

_AddHandler
    move.l  (a7)+,d0                    ;remove return address
    move.w  #ADD_HANDLER,-(a7)          ;push command on stack add handler
    move.l  a7,a0                       ;pointer to top of params
    move.l  d0,-(a7)                    ;push return address back on stack
    move.l  a0,-(a7)                    ;push param pointer onto stack
    trap    #1                          ;call CIO
    addq.l  #4,a7                       ;clean up stack
    move.l  (a7)+,a0                    ;pop return address
    lea     10(a7),a7                   ;clean up stack
                                        ;8 bytes were passed
                                        ;we added another 2 bytes here
    jmp     (a0)                        ;return from subroutine
```

```
    ;************************************************************
    ; int Open(char *dev_name,int mode);
    ;
    ; This function gets 6 bytes and the compiler treats it like
    ; 6 bytes, but we clean up 8 from stack because we add 2
    ;************************************************************

    .FDEF   _Open,6

_Open
    move.l  (a7)+,d0                ;remove return address
    move.w  #OPEN,-(a7)             ;push OPEN command on stack
    move.l  a7,a0                   ;pointer to top of params on stack
    move.l  d0,-(a7)                ;push return address back on stack
    move.l  a0,-(a7)                ;push param pointer onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    move.l  (a7)+,a0                ;pop return address
    lea     8(a7),a7                ;clean up stack
    jmp     (a0)                    ;return from subroutine

    ;************************************************************
    ;int Close(int handle);
    ;
    ; This function gets 2 bytes and the compiler treats it like 2
    ; bytes, but we clean up 4 bytes from stack becasue we add 2
    ;************************************************************

    .FDEF _Close,2

_Close
    move.l  (a7)+,d0                ;remove return address
    move.w  #CLOSE,-(a7)            ;push CLOSE command on stack
    move.l  a7,a0                   ;pointer to top of params on stack
    move.l  d0,-(a7)                ;push return address back on stack
    move.l  a0,-(a7)                ;push param pointer onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    move.l  (a7)+,a0                ;pop return address
    addq.l  #4,a7                   ;clean up stack
    jmp     (a0)                    ;return from subroutine

    ;************************************************************
    ;int GetC(int handle);
    ;
    ;This function gets 2 bytes and the compiler treats it like 2 bytes
    ;but we clean up 4 bytes from the stack because we add 2
    ;
    ;************************************************************

    .FDEF   _Getc,2

_Getc
    move.l  (a7)+,d0                ;remove return address
    move.w  #GETC,-(a7)             ;push GETC command on stack
    move.l  a7,a0                   ;pointer to top of params on stack
    move.l  d0,-(a7)                ;push return address back on stack
    move.l  a0,-(a7)                ;push param pointer onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    move.l  (a7)+,a0                ;pop return address
    addq.l  #4,a7                   ;clean up stack
    jmp     (a0)                    ;return from subroutine

    ;************************************************************
```

```
    ;int PutC(int handle,int c);
    ;
    ; This function gets 4 bytes and the compiler treats it like 4 bytes
    ; but we clean up 6 bytes from the stack because we add 2
    ;*****************************************************************

    .FDEF   _Putc,4

_Putc
    move.l  (a7)+,d0                ;remove return address
    move.w  #PUTC,-(a7)             ;push PUTC command on stack
    move.l  a7,a0                   ;pointer to top of params on stack
    move.l  d0,-(a7)                ;push return address back on stack
    move.l  a0,-(a7)                ;push param pointer onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    move.l  (a7)+,a0                ;pop return address
    addq.l  #6,a7                   ;clean up stack
    jmp     (a0)                    ;return from subroutine

    ;*****************************************************************
    ;int Status(int handle,char *buff,long count,int aux);
    ;
    ; This function gets 12 bytes and the compiler treats it like 12 bytes
    ; but we clean up 14 bytes from the stack because we add 2
    ;*****************************************************************

    .FDEF   _Status,12

_Status
    move.l  (a7)+,d0                ;remove return address
    move.w  #STATUS,-(a7)           ;push STATUS command on stack
    move.l  a7,a0                   ;pointer to top of params on stack
    move.l  d0,-(a7)                ;push return address back on stack
    move.l  a0,-(a7)                ;push param pointer onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    move.l  (a7)+,a0                ;pop return address
    lea.l   14(a7),a7               ;clean up stack
    jmp     (a0)                    ;return from subroutine

    ;*****************************************************************
    ;int StatusO(char *buff,long count,int aux,char *name);
    ;
    ; This function gets 14 bytes and the compiler treats it like 14 bytes
    ; but we clean up 18 bytes from the stack because we add 4
    ;*****************************************************************

_StatusO
    move.l  (a7)+,d0                ;remove return address
    move.w  #-1,-(a7)               ;push negative handle on stack
    move.w  #STATUS,-(a7)           ;push STATUS command on stack
    move.l  a7,a0                   ;pointer to top of params on stack
    move.l  d0,-(a7)                ;push return address back on stack
    move.l  a0,-(a7)                ;push param pointer onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    move.l  (a7)+,a0                ;pop return address
    lea.l   18(a7),a7               ;clean up stack
    jmp     (a0)                    ;return from subroutine

    ;*****************************************************************
    ;long Xio(int cmd,int handle,char *name,char *buff,long count,int aux,...);
    ;
    ; We dont have to push anything extra onto the stack for this function
    ;
```

```
    ; Xio requires that a variable number of parameters can be passed
    ; so the calling function will clean up this stack mess
    ;*****************************************************************
    ;
    XDEF    _Xio

_Xio
    pea     4(a7)                   ;push start of param area onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    rts

    ;*****************************************************************
    ;long Read(int handle,char *buff,long count);
    ;
    ; This function gets 10 bytes and the compiler treats it like 10 bytes
    ; but we clean up 12 bytes from the stack because we add 2
    ;*****************************************************************
    ;


    .FDEF   _Read,10

_Read
    move.l  (a7)+,d0                ;remove return address
    move.w  #READ,-(a7)             ;push READ command on stack
    move.l  a7,a0                   ;pointer to top of params on stack
    move.l  d0,-(a7)                ;push return address back on stack
    move.l  a0,-(a7)                ;push param pointer onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    move.l  (a7)+,a0                ;pop return address
    lea.l   12(a7),a7               ;clean up stack
    jmp     (a0)                    ;return from subroutine

    ;*****************************************************************
    ;long Write(int handle,char *buff,long count);
    ;
    ; This function gets 10 bytes and the compiler treats it like 10 bytes
    ; but we clean up 12 bytes from the stack because we add 2
    ;*****************************************************************
    ;


    .FDEF   _Write,10
_Write
    move.l  (a7)+,d0                ;remove return address
    move.w  #WRITE,-(a7)            ;push WRITE command on stack
    move.l  a7,a0                   ;pointer to top of params on stack
    move.l  d0,-(a7)                ;push return address back on stack
    move.l  a0,-(a7)                ;push param pointer onto stack
    trap    #1                      ;call CIO
    addq.l  #4,a7                   ;clean up stack
    move.l  (a7)+,a0                ;pop return address
    lea.l   12(a7),a7               ;clean up stack
    jmp     (a0)                    ;return from subroutine
```

```
/*****************************************************************
**
** DAC device driver
**
*****************************************************************/

#include <stdarg.h>
#include <stdlib.h>
#include <string.h>
#include "cio.h"
#include "dac.h"
#include "spinchip.h"

#pragma function(calling)

static long DAC_open(IOCB *iocb,va_list argp);
static long DAC_close(IOCB *iocb,va_list argp);
static long DAC_get(IOCB *iocb,va_list argp);
static long DAC_read(IOCB *iocb,va_list argp);
static long DAC_put(IOCB *iocb,int a,va_list argp);
static long DAC_write(IOCB *iocb,va_list argp);
static long DAC_status(IOCB *iocb,va_list argp);
static long DAC_xio(int cmd,IOCB *iocb,va_list argp);

#pragma function()

static int DacVolts[4];     //thar is four of dem now

static const H_JVEC DAC_vec = {
    DAC_open,
    DAC_close,
    DAC_get,
    DAC_read,
    DAC_put,
    DAC_write,
    DAC_status,
    DAC_xio,
    DAC_init
};
static const char DAC_NAME[] = "DAC";

extern "C" {
long DAC_init(void)
{
    int v=0;
    /*
    ** all hardware initialization is done here
    */
    for(v=0;v<4;++v)
    {
        DacVolts[v] = 0;
        SpinChipDacSelect(v);
        *((volatile int *)SPIN_WRITEDAC) = 0x80;        /*  zero volts  */
    }
    /*  hardware initialized    */
    return AddHandler(DAC_NAME,&DAC_vec);   /*  install device driver   */
}
}

static long DAC_open(IOCB *iocb,va_list argp)
{
    /*
    ** this function opens up the DAC for either input (not very useful)
    ** and output
    */
    return 0l;
```

```
}

static long DAC_close(IOCB *iocb,va_list argp)
{
    return 0l;
}

static long DAC_get(IOCB *iocb,va_list argp)
{
    /*
    ** read back the shadow reg
    */
    long a = (long)DacVolts[iocb->devnum-1];

    return a;
}

static long DAC_read(IOCB *iocb,va_list argp)
{
    return 0l;
}

static long DAC_put(IOCB *iocb,int a,va_list argp)
{
    /*
    ** this function outputs data
    */
    DacVolts[iocb->devnum-1] = a;
    SpinChipDacSelect(iocb->devnum-1);

    *((volatile int *)SPIN_WRITEDAC) = a + ((iocb->mode & DACMODE_UNSIGNED)?0:0x80);
    return (long)0;
}

static long DAC_write(IOCB *iocb,va_list argp)
{
    long i;
    char *b = va_arg(argp,char *);
    long c = va_arg(argp,long);

    for(i=0;i<c;++i,++b)
        DAC_put(iocb,(int)*b,argp);
    return i;
}

static long DAC_status(IOCB *iocb,va_list argp)
{
    /*
    ** this function get the device status
    */
    return((long)0);
}

static long DAC_xio(int cmd,IOCB *iocb,va_list argp)
{
    /*
    ** this function is used to perform special functions
    */
    switch(cmd)
    {
    }
    return 0l;
}
```

```
/*****************************************************************
**
** Header file for Reference DAC device Driver
**
*****************************************************************/

#ifndef DAC__H
#define DAC__H

#define LOWERLIMITS "DAC1:"
#define UPPERLIMITS "DAC2:"
#define DIRECTCONTROL   "DAC3"
#define LOOPGAIN    "DAC4:"


#define DACMODE_UNSIGNED    0x100

#ifdef __cplusplus
extern "C" {
#endif

extern long DAC_init(void);

#ifdef __cplusplus
}
#endif

#endif
```

```
/*---------------------------------------------------------------------
**
** Quadrature Port device driver
**
** Copyright (c) 1998 teletrac inc.
**
**---------------------------------------------------------------------*/

#include <stdarg.h>
#include <stdlib.h>
#include <string.h>
#include "task.h"
#include "spinchip.h"
#include "cio.h"
#include "quad.h"

#pragma function(calling)

static long QUAD_open(IOCB *iocb,va_list argp);
static long QUAD_close(IOCB *iocb,va_list argp);
static long QUAD_get(IOCB *iocb,va_list argp);
static long QUAD_read(IOCB *iocb,va_list argp);
static long QUAD_put(IOCB *iocb,int a,va_list argp);
static long QUAD_write(IOCB *iocb,va_list argp);
static long QUAD_status(IOCB *iocb,va_list argp);
static long QUAD_xio(int cmd,IOCB *iocb,va_list argp);

#pragma function()

static const H_JVEC QUAD_vec = {
    QUAD_open,
    QUAD_close,
    QUAD_get,
    QUAD_read,
    QUAD_put,
    QUAD_write,
    QUAD_status,
    QUAD_xio,
    QUAD_init
};

static const char QUAD_NAME[] = "QUAD";

volatile long UpCount=0;
volatile long DownCount=0;

static volatile int QuadratureValue = 0;
static int QuadMode = QUAD_MODE_NORMAL;
static TSemaphore *QuadEvent;   //indicates quad is ready to read

/*********************************************************
** these are the interrupt routines for the quadrature port
**
** The value that the quadrature value is changed by depends
** on how long it has been since the last interrupt.  The longer
** the time, the less the increment, faster more
**
**
*********************************************************/

extern "C" {

void HandleUp(void)
{
    register int i;
```

```
    ++UpCount;
    QuadratureValue++;
    return;
}


void HandleDown(void)
{
    register int i;

    ++DownCount;
    QuadratureValue--;
    return;
}

}

extern "C" void QUAD_IRQ_UP(void);
extern "C" void QUAD_IRQ_DN(void);

long QUAD_init(void)
{
    int v;
    /*
    ** all hardware initialization is done here
    */

    QuadEvent = new TSemaphore(0,1,"QuadEvent");
    SpinIOEnableIrq(SPIN_ENUP | SPIN_ENDOWN);

    /*  hardware initialized    */
    return AddHandler(QUAD_NAME,&QUAD_vec); /*  install device driver   */
}

static long QUAD_open(IOCB *iocb,va_list argp)
{
    /*
    ** this function opens up the QUAD for either input (not very useful)
    ** and output
    */
    return 0l;
}

static long QUAD_close(IOCB *iocb,va_list argp)
{
    /*
    ** this function terminates i/o to the Quadrature Port and releases it
    ** to be used
    */
    return 0l;
}

static long QUAD_get(IOCB *iocb,va_list argp)
{
    /*
    ** this function is used for getting data from the Quadrature Port
    */
    long a;
    int sign;
    int t;
    int min;
    static int count;

    QuadratureValue = 0;    //reset value
    QuadEvent->Pend(2);
    if(QuadratureValue < 0) sign = 1; else sign=0;  //check sign
```

```
    if(QuadratureValue) min = 1; else min=0;        //minimum value
    QuadratureValue = abs(QuadratureValue);         //make positive

    if(QuadratureValue > 30) QuadratureValue = 30;
    t = QuadratureValue;
    QuadratureValue *= QuadratureValue;     //square it
    if(QuadMode)
        QuadratureValue *= t;                   //cube it
    QuadratureValue >>= 4;      //divide by two
    if(QuadratureValue==0)
    {
        if(count > 0)
        {
            --count;
            if(count == 0)
            {
                QuadratureValue = min;
                count = 4;
            }
        }
        else if(count == 0) count = 4;
        else QuadratureValue = min;
    }
    if(sign)QuadratureValue = -QuadratureValue; //re-negate
    a = (long)((unsigned )QuadratureValue);
    return a;
}

static long QUAD_read(IOCB *iocb,va_list argp)
{
    long i;
    char *b = va_arg(argp,char *);
    long c = va_arg(argp,long);

    for(i=0;i<c;++b,++i)
        *b = (char)QUAD_get(iocb,argp);
    return i;
}

static long QUAD_put(IOCB *iocb,int a,va_list argp)
{
    /*
    ** this function outputs data a to the Quadrature Port
    */
    return (long)0;
}

static long QUAD_write(IOCB *iocb,va_list argp)
{
    return 0l;
}

static long QUAD_status(IOCB *iocb,va_list argp)
{
    /*
    ** this function get the device status for the Quadrature Port
    */
    return((long)0);
}

static long QUAD_xio(int cmd,IOCB *iocb,va_list argp)
{
    /*
    ** this function is used to perform special functions to the Quadrature Port
    */
    int sr;
```

```
    char *b = va_arg(argp,char *);
    long c = va_arg(argp,long);
    int aux = va_arg(argp,int);

    switch(cmd)
    {
        case QUAD_RESET:
            QuadratureValue = 0;
            break;
        case QUAD_SET_SENSITIVITY:
            break;
        case QUAD_MODE:
            QuadMode = aux;
            break;
        case QUAD_FLUSH:
            //-----------------------------------------
            // This function call sets the Quadrature value
            // to zero and does a post semaphore
            //-----------------------------------------
            sr = EnterCritical();
            QuadratureValue = 0;
            if(QuadEvent->GetCount() < 0)   /*  if task waiting, post   */
                QuadEvent->Post();
            ExitCritical(sr);
            break;
    }
    return 0l;
}
```

```
/*-------------------------------------------------------------------------
**
** Header File for LCD device driver
**
**-------------------------------------------------------------------------*/

#ifndef QUAD__H
#define QUAD__H

/*
** this structure is used to control the sensitivity of the quadrature
** control knob.  It is used to create an array, shorter times are
** first in the array, going towards longer times at the end
*/


typedef struct {
    int time;           /*  Amount of time in milliseconds  */
    int increment;      /*  for this much change    */
}QUAD_SENSITIVITY;

#define QUAD_SENSITIVITY_SIZE   4

/* Hardware defines */

#define QUAD_VECTOR_UP      0x214
#define QUAD_VECTOR_DN      0x21C

/*  XIO function calls  */

#define QUAD_RESET              0x20
#define QUAD_SET_SENSITIVITY    0x21
#define QUAD_MODE               0x22
#define QUAD_FLUSH              0x23

/*  modes for Quadrature device */

#define QUAD_MODE_NORMAL        0
#define QUAD_MODE_FAST          1

#ifdef __cplusplus
extern "C" {
#endif

extern long QUAD_init(void);

#ifdef __cplusplus
}
#endif

#endif
```

```
/*************************************************************************
** RS232 Device Driver                                                  **
** Copyright (c) 1995 Jim Patchell                                      **
**                                                                      **
**                                                                      **
*************************************************************************/
#define RS232_BUILD //we need all defines

#include <stdio.h>
#include "task.h"
#include "queue.h"
#include "rs232.h"
#include "cio.h"
//#include <curses.h>
#include <stdarg.h>

static int BRIndex[2]={RS232_BAUDRATE_9600,RS232_BAUDRATE_9600};
IO_REC TRecs[2],RRecs[2];   //two transmit, two recieve buffers
char ImrShadow=0;
char acr_shad;

static const int BaudRates[13] = {
    BAUDRATE_75,
    BAUDRATE_110,
    BAUDRATE_134,
    BAUDRATE_150,
    BAUDRATE_300,
    BAUDRATE_600,
    BAUDRATE_1200,
    BAUDRATE_1800,
    BAUDRATE_2000,
    BAUDRATE_2400,
    BAUDRATE_4800,
    BAUDRATE_9600,
    BAUDRATE_19200
};

static const int Parity[3] = {
    PARITY_NONE,
    PARITY_EVEN,
    PARITY_ODD
};

static const int StopBits[2] = {
    STOPBIT_1,
    STOPBIT_2
};

#pragma function(calling)

static long R_open(IOCB *iocb,va_list argp);
static long R_close(IOCB *iocb,va_list argp);
static long R_get(IOCB *iocb,va_list argp);
static long R_read(IOCB *iocb,va_list argp);
static long R_put(IOCB *iocb,int a,va_list argp);
static long R_write(IOCB *iocb,va_list argp);
static long R_status(IOCB *iocb,va_list argp);
static long R_xio(int cmd,IOCB *iocb,va_list argp);

#pragma function()

static const H_JVEC r_vec = {
    R_open,
    R_close,
    R_get,
    R_read,
```

```
    R_put,
    R_write,
    R_status,
    R_xio,
    R_init
};

extern "C" void EnableInterrupt(void);
extern int p_error(int,...);

/*
** initialize the RS-232 handler
*/
static void init_uart(void);


static int LookupParity(int parity)
{
    /*-------------------------------------------------------
    ** this routine scans thru the parity table above and
    ** returns an INDEX NUMBER for the parity setting
    **-------------------------------------------------------*/
    int i;

    for(i=0;i<4;++i)
        if(parity == Parity[i])
            return i;
    return -1;

}

static int LookupStopBits(int sb)
{
    /*-------------------------------------------------------
    ** this routine scans thru the stop bit table above and
    ** returns an INDEX NUMBER for the stop bit setting
    **-------------------------------------------------------*/
    int i;

    for(i=0;i<2;++i)
        if(sb == StopBits[i])
            return i;
    return -1;
}

/*---------------------------------------------------------------------
** code for device driver entry points
**---------------------------------------------------------------------*/

extern "C" {

long R_init(void)
{
    static const int PutIrqMasks[2] = {0x01,0x10};
    static const int GetIrqMasks[2] = {0x02,0x20};
    static const int DsrBits[2] = {1,2};
    static const long Command[2] = {COMMAND_REG_A,COMMAND_REG_B};
    static const long Status[2] = {STATUS_REG_A,STATUS_REG_B};
    static const long Mode[2] = {MODE_REG_A,MODE_REG_B};
    static const long Data[2] = {DATA_REG_A,DATA_REG_B};

//  init_irq(); /*  initialize interrupts for UART  */
    /*
    ** initialize IO descriptors for Uarts
    */
    int i;
```

```
    init_uart();
    for(i=0;i<2;++i)
    {
        RRecs[i].buffer = new char[RS232_IOREC_BUFFERSIZE]; //make some big buffers
        TRecs[i].buffer = new char[RS232_IOREC_BUFFERSIZE];
        RRecs[i].size = RS232_IOREC_BUFFERSIZE;
        TRecs[i].size = RS232_IOREC_BUFFERSIZE;
        RRecs[i].head = 0;
        TRecs[i].head = 0;
        RRecs[i].tail = 0;
        TRecs[i].tail = 0;
        RRecs[i].nchars=0;
        TRecs[i].nchars=0;
        RRecs[i].high = 1000;
        RRecs[i].low = 0;
        TRecs[i].high = 1000;
        TRecs[i].low = 24;
        RRecs[i].eotchar = '\n';
        TRecs[i].eotchar = '\n';
        //---------------------------------------
        // create semaphore names
        //---------------------------------------
        sprintf(RRecs[i].buffer,"RRECS_RS232DEV%d",i);
        sprintf(TRecs[i].buffer,"TRECS_RS232DEV%d",i);
        RRecs[i].IOEvent = new TSemaphore(0,1,RRecs[i].buffer);
        TRecs[i].IOEvent = new TSemaphore(RS232_IOREC_BUFFERSIZE,1,TRecs[i].buffer);
        RRecs[i].putirqmsk = PutIrqMasks[i];
        TRecs[i].putirqmsk = PutIrqMasks[i];
        RRecs[i].getirqmsk = GetIrqMasks[i];
        TRecs[i].getirqmsk = GetIrqMasks[i];
        RRecs[i].dsrbit = DsrBits[i];
        TRecs[i].dsrbit = DsrBits[i];
        RRecs[i].uartdata = (char *)Data[i];
        TRecs[i].uartdata = (char *)Data[i];
        RRecs[i].statusreg = (char *)Status[i];
        TRecs[i].statusreg = (char *)Status[i];
        RRecs[i].bsreg = (char *)BIT_SET_COMMAND;
        TRecs[i].brreg = (char *)BIT_RESET_COMMAND;
        RRecs[i].modereg = (char *)Mode[i];
        TRecs[i].modereg = (char *)Mode[i];
        RRecs[i].cmdreg = (char *)Command[i];
        TRecs[i].cmdreg = (char *)Command[i];
        RRecs[i].timeout = 0;
        TRecs[i].timeout = 0;
        sprintf(TRecs[i].buffer,"TRECS_BLOCKER%d",i);
        TRecs[i].WritersBlock = new Wait(1,TRecs[i].buffer);    //blocking semaphore, one use
    }
    *(char *)INTERRUPT_VECTOR_REG = 0x40;
    EnableInterrupt();
    return AddHandler("R",&r_vec);
}

}

static void init_uart(void)
{
    *((char *)(INTERRUPT_MASK_REG)) = 0x0;  /* DISABLE INTERRUPTS */

    *((char *)(COMMAND_REG_A)) = 0x10;  /* reset MR ptr */
    *((char *)(COMMAND_REG_A)) = 0x20;   /* reset rxr   */
    *((char *)(COMMAND_REG_A)) = 0x30;   /* reset TXD */
    *((char *)(MODE_REG_A)) = 0x13;     /*  no parity, 8 bits   */
    *((char *)(MODE_REG_A)) = 0x07;     /*  1 stop bit  */
    *((char *)(STATUS_REG_A) ) = BAUDRATE_19200;    /*  9600 buad   */
    *((char *)(COMMAND_REG_A)) = 0x04;  /*  transmit enabled    */
```

```
    *((char *)(COMMAND_REG_A)) = 0x01;   /* recieve enabled */

    acr_shad = 0x0c;    //0x8c makes max baud 19200, 0x0c is 38.4k
    *((char *)(AUX_CONTROL_REG)) = acr_shad;    /*  enable change bits on ip2 and ip3   */

    *((char *)(COMMAND_REG_B)) = 0x10;  /* reset MR ptr */
    *((char *)(COMMAND_REG_B)) = 0x20;   /* reset rxr   */
    *((char *)(COMMAND_REG_B)) = 0x30;   /* reset TXD */
    *((char *)(MODE_REG_B)) = 0x93;     /*  no parity, 8 bits   */
    *((char *)(MODE_REG_B)) = 0x07;     /*  1 stop bit  */
    *((char *)(STATUS_REG_B) ) = BAUDRATE_9600;     /* 9600 Baud    */
    *((char *)(COMMAND_REG_B)) = 0x04;  /*  transmit enabled    */
    *((char *)(COMMAND_REG_B)) = 0x01;   /* recieve enabled */

    *((char *)(BIT_SET_COMMAND)) = 0x0f;    /*   set all handshake lines    */
}


static long R_open(IOCB *iocb,va_list argp)
{
    int dev = iocb->devnum - 1;
    int sr;

    if (iocb->mode & READ_ONLY)
    {
        /*
        **  enable system interrupts
        */
        sr = EnterCritical();   /*  disable interrupts  */
        RRecs[dev].head = 0;
        RRecs[dev].tail = 0;
        RRecs[dev].nchars = 0;
        ImrShadow |= RRecs[dev].getirqmsk;  /*  enable interrupt bit    */
        *((char *)(INTERRUPT_MASK_REG)) = ImrShadow;
        ExitCritical(sr);
    }
    if(iocb->mode & WRITE_ONLY)
    {
        TRecs[dev].head = 0;
        TRecs[dev].tail = 0;
        TRecs[dev].nchars = 0;
    }
    return(0);
}

static long R_close(IOCB *iocb,va_list argp)
{
    /*
    ** disable the UART interrupts
    */
    int dev = iocb->devnum - 1;
    int v;
    int sr;

    v = RRecs[dev].getirqmsk;
    v |= RRecs[dev].putirqmsk;
    sr = EnterCritical();
    ImrShadow &= ~v;
    *((char *)(INTERRUPT_MASK_REG)) = ImrShadow;
    ExitCritical(sr);
    return(0);
}

static long R_get(IOCB *iocb,va_list argp)
{
    /*********************************************************************
```

```
    ** get a character from the RS232 port buffer
    **
    ** If there are no characters in the buffer, pend until there is.
    ** Pending is required so that other tasks can be running when there
    ** are no characters and we don't tie things up waiting to run
    **
    ** 04-26-2001
    **
    ** get routine rewritten
    **
    ** RRecs[dev].IOEvent semaphore is now a counting semaphore so that
    ** hopefully a bug that was discovered will no longer occure.
    ** This change is allso reflected in the Recieve Interrupt (below)
    **
    ***********************************************************************/
    int dev;    //minor device number
    int error=0;    //multitasking error status
    int c;
    int sr;

    dev = iocb->devnum - 1; //get the device number
    //-------------------------------------------------------
    // check the recieve semaphore, pend if nothing is there
    //-------------------------------------------------------
    error = RRecs[dev].IOEvent->Pend(RRecs[dev].timeout);
    if(error < 0)   //was there an error?, then return
        return (long)error;
    sr = EnterCritical();
    if(!RRecs[dev].nchars)  //are there any characters to get, double check ?
    {
        ExitCritical(sr);
        return (long)RS232_TIMEOUT;
    }
    c = (unsigned char)RRecs[dev].buffer[RRecs[dev].tail++];    //get char
    if(RRecs[dev].tail == RRecs[dev].size)
        RRecs[dev].tail = 0;        //wrap circular buffer
    if(--RRecs[dev].nchars <= RRecs[dev].low)       //decrement total number of chars
    {
        //low water mark was reached.  Fiddle with flow control lines
        *RRecs[dev].bsreg = (char)RRecs[dev].dsrbit;    //open up floodgate
    }
    ExitCritical(sr);
    return (long)c;
}

static long R_read(IOCB *iocb,va_list argp)
{
    long i;
    char *b = va_arg(argp,char *);
    long count = va_arg(argp,long);

    for(i=0;i<count;++i,++b)
        *b = (char)R_get(iocb,argp);
    return i;
}

static long R_put(IOCB *iocb,int c,va_list argp)
{
    /***********************************************************************
    **
    ** Put a character into an RS232 buffer
    **
    **  If the buffer is full, pend on buffer empty.  This will allow
    ** other tasks to execute while waiting for the buffer to empty
    **
    ***********************************************************************/
```

```
    int error=0;    //multitasking error status
    IO_REC *dev = &TRecs[iocb->devnum - 1];
    int sr;

    error = dev->IOEvent->Pend(dev->timeout);
    if(error < 0)
        return (long)error;
    sr = EnterCritical();           //disable interrupts
    if(dev->nchars == dev->size)    //is buffer FULL?
    {
        if(!(ImrShadow & dev->putirqmsk))
        {
            ImrShadow |= dev->putirqmsk;
            *((char *)INTERRUPT_MASK_REG) = ImrShadow;
        }
    }
    //
    // put data into output buffer
    //
    dev->buffer[dev->head++] = (char)c;
    if(dev->head == dev->size)
        dev->head = 0;  //wrap buffer
    dev->nchars++;
    if(((c == dev->eotchar) || (dev->eotchar < 0)) && !(ImrShadow & dev->putirqmsk))
    {
        ImrShadow |= dev->putirqmsk;
        *((char *)INTERRUPT_MASK_REG) = ImrShadow;
    }
    ExitCritical(sr);
    return 0l;
}

static long R_write(IOCB *iocb,va_list argp)
{
    long i;
    IO_REC *dev = &TRecs[iocb->devnum - 1];
    char *b = va_arg(argp,char *);
    long count = va_arg(argp,long);
    int error;

    if((i = dev->WritersBlock->Pend()) < 0) goto exit;
    for(i=0;i<count;++i,++b)
    {
        if((error = R_put(iocb,(int)(unsigned char)*b,argp)) < 0)
        {
            dev->WritersBlock->Post(0); //clean up semaphore
            return (long)error;
        }
    }
exit:
    dev->WritersBlock->Post(0);
    return i;
}

static long R_xio(int cmd,IOCB *iocb,va_list argp)
{
    volatile int a; /*  temp variable   */
    int dev = iocb->devnum - 1;
    long retval=0;
    char *buffer = va_arg(argp,char *);
    long count = va_arg(argp,long);
    int aux = va_arg(argp,int);
    /*
    ** execute misc function needed by RS-232 port
    */
```

```
    switch(cmd) /*  decode command  */
    {
        case SET_BAUD_RATE:
            /*
            ** baud rate is in aux (an int)
            */
            BRIndex[dev] = aux; /*  store baudrate for later    */
            *(RRecs[dev].statusreg) = (char)BaudRates[aux];
            break;
        case SEND_BREAK:
            break;
        case WAIT_EMPTY:
            /*  wait for transmit buffer to empty   */
            while(TRecs[dev].tail != TRecs[dev].head);  /*  wait for index to be equal  */
            break;
        case SET_MODE:
            break;
        case SET_INPUT_BUFFER:
            break;
        case SET_OUTPUT_BUFFER:
            break;
        case MEM_PROTECT:   /*  set memprotect hardware line    */
            /*
            ** Aux = 0->mem protect off
            ** Aux = 1->mem protect on
            */
            if(aux)
                *((char *)(BIT_RESET_COMMAND)) = SET_PROTECT_BIT;
            else
                *((char *)(BIT_SET_COMMAND)) = SET_PROTECT_BIT;
            break;
        case SET_ERROR:
            /*
            ** Aux = 0->turn error light off
            ** Aux = 1->turn error light on
            */
            if(aux)
                *((char *)(BIT_RESET_COMMAND)) = SET_ERROR_BIT;
            else
                *((char *)(BIT_SET_COMMAND)) = SET_ERROR_BIT;
            break;
        case SET_ACTIVE:
            /*
            ** Aux = 0->turn active light off
            ** aux = 1->turn active light on
            */
            if(aux)
                *((char *)(BIT_RESET_COMMAND)) = SET_ACTIVE_BIT;
            else
                *((char *)(BIT_SET_COMMAND)) = SET_ACTIVE_BIT;
            break;
        case READ_INPORT:
            return *((char *)(INPUT_PORT_REG));
            break;
        case RS232_SET_PARITY:
            /*  reset MR pointer to MR1 */
            *(RRecs[dev].cmdreg) = 0x10;
            /*  get mode reg variable   */
            a = (int)*(RRecs[dev].modereg);
            a &= ~PARITY_MASK;
            a |= Parity[aux];
            /*  reset MR pointer to MR1 */
            *(RRecs[dev].cmdreg) = 0x10;
            *(RRecs[dev].modereg) = (char)a;
            break;
        case RS232_SET_BITS:
```

```
            /*
            ** set number of data bits
            */
            /*  reset MR pointer to MR1 */
            *(RRecs[dev].cmdreg) = 0x10;
            /*  get mode reg variable   */
            a = (int)*(RRecs[dev].modereg);
            a &= ~0x03; /*  mask off bits / character */
            a |= aux & 0x03;        /*  new bits per char   */
            /*  reset MR pointer to MR1 */
            *(RRecs[dev].cmdreg) = 0x10;
            *(RRecs[dev].modereg) = (char)a;
            break;
        case RS232_SET_STOP:
            /*
            ** set number of stop bits
            */
            /*  reset MR pointer to MR1 */
            *(RRecs[dev].cmdreg) = 0x10;
            /*  get mode reg variable   */
            a = (int)*(RRecs[dev].modereg);
            a = (int)*(RRecs[dev].modereg); /*  now point to MR2    */
            a &= ~0x0f;                     /*  mask off bits / character */
            a |= StopBits[aux];     /*  new bits per char   */
            *(RRecs[dev].modereg) = (char)a;
            break;
        case RS232_GET_BAUD:
            retval = (long)BRIndex[dev];
            break;
        case RS232_GET_PARITY:
            /*  reset MR pointer to MR1 */
            *(RRecs[dev].cmdreg) = 0x10;
            /*  get mode reg variable   */
            a = (int)*(RRecs[dev].modereg);
            a &= PARITY_MASK;
            retval = (long)LookupParity(a);
            break;
        case RS232_GET_BITS:
            /*  reset MR pointer to MR1 */
            *(RRecs[dev].cmdreg) = 0x10;
            /*  get mode reg variable   */
            retval = (long)*(RRecs[dev].modereg) & 0x03;
            break;
        case RS232_GET_STOP:
            /*  reset MR pointer to MR1 */
            *(RRecs[dev].cmdreg) = 0x10;
            /*  get mode reg variable   */
            a = (int)*(RRecs[dev].modereg);
            a = (int)*(RRecs[dev].modereg) & 0x0f;  /*  access of MR2   */
            retval = (long) LookupStopBits(a);
            break;
        case RS232_SET_PUTEOT:
            TRecs[dev].eotchar = aux;
            break;
        case RS232_SET_RTIMEOUT:
            RRecs[dev].timeout = aux;
            break;
        case RS232_SET_GETEOT:
            RRecs[dev].eotchar = aux;
            break;

    }   /*  end of switch command   */

    return(retval);
}
```

```
static long R_status(IOCB *iocb,va_list argp)
{
    /*
    ** This code returns the number of bytes in the
    ** recieve buffer
    */
    int dev = iocb->devnum - 1;
    char *buffer = va_arg(argp,char *);
    long count = va_arg(argp,long);
    int aux = va_arg(argp,int);

    switch(aux)     /*  which status function   */
    {
        case RECEIVE_STAT:
            return RRecs[dev].nchars;   /*  return number of characters */
        case SEND_STAT:
            return TRecs[dev].nchars;   /*  return number of xmit chars */
    }


    return(RRecs[dev].nchars);      /*  default, return recieve chars   */
}

#pragma function(calling)

extern "C" {

void HandlePutInterrupt(int d)
{
    //-------------------------------------
    // this function is called by the uart
    // isr handler in response to a PUT
    //-------------------------------------
    IO_REC *t = &TRecs[d];      //get address of device record

    int sr;

    while(*t->statusreg & 0x04)
    {
        sr = EnterCritical();   //are these needed?
        if(!t->nchars)      //is buffer empty?
        {
            ImrShadow &= ~t->putirqmsk;     //disable transmit interrupt
            *((char *)INTERRUPT_MASK_REG) = ImrShadow;
            ExitCritical(sr);   //are these needed?
            goto exit;  //just leave
        }
        else                //buffer still has data, put it out
        {
            *t->uartdata = t->buffer[t->tail++];    //stuff data into data reg
            if(t->tail == t->size)
                t->tail = 0;                        //wrap circular buffer
//          if(--t->nchars <= t->low)               //reach low water? or below?
//          {
//              if(t->IOEvent->GetCount() < 0)      //task pending?
            --t->nchars;                            //decrement number of chars
            t->IOEvent->Post(0);                    //Let calling task know
//          }
        }
        ExitCritical(sr);   //are these needed?
    }       //keep stuffing chars while you can
exit:
    return;
}

void HandleGetInterrupt(int d)
```

```
{
    //-------------------------------------
    // this function is called by the uart
    // isr handler in response to a GET
    //-------------------------------------
    /*
    ** 04-26-2001
    **
    ** get routine rewritten
    **
    ** RRecs[dev].IOEvent semaphore is now a counting semaphore so that
    ** hopefully a bug that was discovered will no longer occure.
    ** RRecs[dev].eotchar is no longer used
    ** This change is allso reflected in the Get routine (above)
    */
    IO_REC *t = &RRecs[d];      //address of IO descriptor
    char c;
    int sr;

    do
    {
        sr = EnterCritical();   //are these needed?
        c = *t->uartdata;
        if(t->nchars < t->size)     //is there space?
        {
            t->buffer[t->head++] = c;   //get data from uart
            if(t->head == t->size)
                t->head = 0;            //wrap buffer
            ++t->nchars;                //increment character count
        }
        ExitCritical(sr);           //are these needed?
        if(t->nchars == t->high)    //high water mark?
        {
            *t->brreg = (char)t->dsrbit;    //stop up flood gate
        }
        //-----------------------------------------
        // check end of transmission character
        // if it is less than zero, post semaphore anyway
        //-----------------------------------------
        t->IOEvent->Post(0);
    }while(*t->statusreg & 0x01);       //read as many as posible
}

}   //end of extern C
```

```
;*************************************************************************
;
; MC68681.s
;
; This is a companion file to RS232DEV.CPP
;
; This file handles all of the interrupts from the chip
; It also has the handler in it for all of the vaious spurious exceptions
;
; Interrupt handlers calls C functions in various files
; Interrupt handler calls :
;  EnterInterrupt
;  ExitInterrupt
;  HandleGetIrq
;  HandlePutIrq
;
;*************************************************************************
;

    .OPTION target=68000,flags=gG
    .OPTION char=1,short=2,int=2,long=4,float=4,double=8
    .OPTION ldouble=8,ptrd=4,ptrf=4,sizeof=4,cver=0x430
;   equates
    XDEF    _EnableInterrupt,_DisableInterrupt
    XREF _EnterInterrupt
    XDEF _rs232_irq
    XREF    _ImrShadow
    .FREF   _ExitInterrupt,2    ;this function removes two bytes

    XREF    START

MR1A:           EQU 0xffe00001  /*  mode control register A */
MR2A:           EQU 0xffe00001  /*  same address, different reg */
SRA:            EQU 0xffe00003  /*  status register port A  */
RBA:            EQU 0xffe00007  /*  recieve buffer A    */
IPCR:           EQU 0xffe00009  /*  input change register   */
ISR:            EQU 0xffe0000b  /*  interrupt status register   */
CUR:            EQU 0xffe0000d  /*  counter mode        */
CLR:            EQU 0xffe0000f  /*  counter mode        */
MR1B:           EQU 0xffe00011  /* mode control register B  */
MR2B:           EQU 0xffe00011  /* same address, different reg  */
SRB:            EQU 0xffe00013  /*  status register port B  */
RBB:            EQU 0xffe00017  /*  recieve buffer B    */
IVR:            EQU 0xffe00019  /*  interrupt vector reg    */
IPUL:           EQU 0xffe0001b  /*  input port, unlatched   */
START_COUNTER:  EQU 0xffe0001d
STOP_COUNTER:   EQU 0xffe0001f
BITSET:         EQU 0xffe0001d
BITRESET:       EQU 0xffe0001f

/* WRITE locations, where different from READ   */

CSRA:           EQU 0xffe00003  /*  clock select register A */
CRA:            EQU 0xffe00005  /*  command register A  */
TBA:            EQU 0xffe00007  /*  transmit buffer A   */
ACR:            EQU 0xffe00009  /*  Auxiliary Control Register  */
IMR:            EQU 0xffe0000b  /*  interrupt mask register */
CTUR:           EQU 0xffe0000d  /*  counter/timer upper reg */
CTLR:           EQU 0xffe0000f  /*  counter/timer lower reg */
CSRB:           EQU 0xffe00013  /*  clock select register B */
CRB:            EQU 0xffe00015  /*  command reg B   */
TRB:            EQU 0xffe00017  /*  transmit register B */
OPCR:           EQU 0xffe0001b  /*  output port config register */
OPRBS:          EQU 0xffe0001d  /*  output port bit set command */
OPRRS:          EQU 0xffe0001f  /*  output port bit reset command   */
```

```
RS232_VEC:  EQU 0x00000100  /*  user vector 64  */


EXCEPTION_VEC:  EQU 0x08
; equates for data structures

BUFFER: EQU 0       ;pointer to buffer area
HEAD:   EQU 4       ;head pointer
TAIL:   EQU 6       ;tail pointer
SIZE:   EQU 8       ;size of buffer
NCHARS:    = 10     ;number of characters in buffer
HIGH:   =   12      ;high water mark
LOW:    =   14      ;low water mark
EOTCHAR: = 16
TIMEOUT: = 18
IOEVENT: = 20
UARTDREG:   = 24    ;address of data register
STATUSREG:  = 28    ;address of status register
BITSETREG:  = 32    ;address of bit set register
BITRSETREG: = 36    ;address of bit reset register
MODEREG:    = 40    ;address of mode register
COMMANDREG: = 44    ;address of command register
PUTIRQMSK:  = 48    ;mask for IMR for put
GETIRQMSK:  = 50    ;mask for IMR for get
DSRBIT:     = 52    ;mask for bit set register for DSR

    SECTION ram
isr_temp:   DS.B    1

    SECTION code
_EnableInterrupt:
    andi    #0xf8ff,sr      ;   enable interrupts
    rts

_DisableInterrupt:
    ori     #0x07ff,sr      ;disable interrupts
    rts

_rs232_irq:
    movem.l d0-d7/a0-a6,-(a7)   ;save context on the stack
    jsr     _EnterInterrupt     ;indicate we are in an interrupt
    move.b  ISR,d0              ;get status of interrupts
    and.b   _ImrShadow,d0       ;mask off unwanted status bits
    move.b  d0,isr_temp         ;save a copy of interrupt status
;
;--------------------------------------------------------------------
; determine which interrupts were activated and call them
;--------------------------------------------------------------------
;
    btst.b  #5,isr_temp         ;* port B recive
    beq.s   L7                  ;highest priority
    move.w  #1,-(a7)            ;push device number on stack
    jsr     _HandleGetInterrupt ;call interrupt handler
    addq.l  #2,a7               ;clean up stack

L7:
    btst.b  #4,isr_temp         ;port B transmit
    beq.s   L6
    move.w  #1,-(a7)            ;push device number on stack
    jsr     _HandlePutInterrupt ;call interrupt handler
    addq.l  #2,a7

L6:
    btst.b  #0,isr_temp         ;*  port a transmit
    beq.s   L5
    move.w  #0,-(a7)            ;push device number on stack
```

```
    jsr     _HandlePutInterrupt ;call interrupt handler
    addq.l  #2,a7               ;clean up stack

L5:
    btst.b  #1,isr_temp         ; check interrupt status for UART port a
    beq.s   L8
    move.w  #0,-(a7)            ;push device number on stack
    jsr     _HandleGetInterrupt ;call interrupt handler
    addq.l  #2,a7               ;clean up stack

L8:
    btst.b  #3,isr_temp         ;check for a timer interrupt
    beq.s   L11
    jsr     _the_timer          ;goto timer interrupt handler
    move.b  STOP_COUNTER,d0     ;dummy read, clear interrupt

L11:
    btst.b  #7,isr_temp
    beq.s   L4
    jsr     _port_change        ;goto port change handler

L4:
    move.w  60(a7),d0               ;get status register
    andi.w  #0x0700,d0              ;mask off junk
    asr.w   #8,d0                   ;shift over 8 bits
    move.w  d0,-(a7)                ;push onto stack
    jsr     _ExitInterrupt      ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6   ;restore context from stack
    rte

;*********************************************************
; Misc interrupt routines
;*********************************************************

    XREF    _p_error
    XDEF buss_error,address_error
    XDEF illegal,zero_div,chk_instr,trapv_instr
buss_error:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    moveq #2,d0
    bra.s   goto_error
address_error:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    moveq #3,d0
    bra.s   goto_error
illegal:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    moveq #4,d0
    bra.s goto_error
zero_div:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    moveq #5,d0
    bra.s goto_error
chk_instr:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    moveq #6,d0
    bra.s goto_error
trapv_instr:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    moveq #7,d0
    bra.s goto_error

    XDEF privilege,trace,line_a,line_f,reserved

privilege:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
```

```
    moveq #8,d0
    bra.s goto_error
trace:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    moveq #9,d0
    bra.s goto_error
line_a:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    moveq #10,d0
    bra.s goto_error
line_f:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    moveq #11,d0
    bra.s goto_error
reserved:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    moveq #12,d0
    bra.s goto_error

    XDEF unitialized,spurious,autovector,trap_instr,user_vecs

unitialized:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    moveq #15,d0
    bra.s goto_error
spurious:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    moveq #24,d0
    bra.s goto_error
autovector:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    moveq #25,d0
    bra.s goto_error
trap_instr:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    moveq #32,d0
    bra.s goto_error
user_vecs:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    move.w #64,d0
goto_error:
    move.l  a7,d1
    add.l   #60,d1                     ;point stack to where it was
    move.l  d1,-(a7)
    move.w  d0,-(a7)
    jsr _p_error
    jmp START              ;Attempt to Reboot
```

```
/*************************************************************************
**                                                                      **
** MultiTasking RS-232 Device Driver                                    **
**                                                                      **
**  Driver manages a queue between the RS232 Port and a TASK.           **
**                                                                      **
** GET:                                                                 **
**                                                                      **
**  When doing a GET, if the buffer is empty, the driver will suspend   **
** and do a reschedule.  When the data in the buffer exceeds a high     **
** water mark, the waiting task will be woken up.  Also, if a character **
** is put into the buffer that matches the End Of Transmission character **
** (EOT), The waiting task if any will also be woken up.                **
**                                                                      **
** PUT:                                                                 **
**                                                                      **
**  When doing a PUT, if the buffer is full, the driver will suspend    **
** until the buffer is empty.  When the data buffer goes below the low  **
** water mark, the pending task will be woken up.                       **
*************************************************************************/

#ifndef RS232__H
#define RS232__H

/*
** hardware defines
*/

#define UART                        (0xffe00001l)

#define MODE_REG_A                  (UART + 0)
#define STATUS_REG_A                (UART + 1*2)
#define COMMAND_REG_A               (UART + 2*2)
#define DATA_REG_A                  (UART + 3*2)

#define MODE_REG_B                  (UART + 8*2)
#define STATUS_REG_B                (UART + 9*2)
#define COMMAND_REG_B               (UART + 10 *2)
#define DATA_REG_B                  (UART + 11 * 2)

#define AUX_CONTROL_REG             (UART + 4*2)
#define INPUT_PORT_CHANGE_REG       (UART + 4*2)
#define INTERRUPT_STATUS_REG        (UART + 5*2)
#define INTERRUPT_MASK_REG          (UART + 5*2)
#define COUNTER_UPPER_REG           (UART + 6*2)
#define COUNTER_LOWER_REG           (UART + 7*2)
#define INTERRUPT_VECTOR_REG        (UART + 12*2)
#define INPUT_PORT_REG              (UART + 13*2)
#define OUTPUT_PORT_CONFIG          (UART + 13*2)
#define START_COUNTER               (UART + 14*2)
#define BIT_SET_COMMAND             (UART + 14*2)
#define STOP_COUNTER                (UART + 15*2)
#define BIT_RESET_COMMAND           (UART + 15*2)
#define OUTPUT_PORT                 (UART + 14*2)
/*
** defines for port a UART
*/
#define TX_A_RDY_IRQ    0x01    /*  transmit ready interrupt mask   */
#define RX_A_RDY_IRQ    0x02    /*  reciever ready interrupt mask   */
#define DELTA_BREAK_A   0x04    /*  delta break a iqr mask  */

#define TX_B_RDY_IRQ    0x10    /*  transmit ready interrupt mask   */
#define RX_B_RDY_IRQ    0x20    /*  reciever ready interrupt mask   */
#define DELTA_BREAK_B   0x40    /*  delta break a iqr mask  */

#ifndef __IOREC__
```

```
#define TX_EMPTY 0x08

/*
** data structures used by device
*/

#ifdef RS232_BUILD

typedef struct {
    char *buffer;   /*  pointer to data buffer          4   */
    int head;       /*  head index in buffer            2   */
    int tail;       /*  tail index in buffer            2   */
    int size;       /*  size of buffer                  2   */
    int nchars;     /*  number of characters in buffer  2   */
    int high;       /*  high water mark                 2   */
    int low;        /*  low water mark                  2   */
    int eotchar;    /*  end of transmision character    2   */
    int timeout;    /*  how long to wait for buffer     2   */
    TSemaphore *IOEvent;    /*  semaphore controling IO         4   */
    char *uartdata; /*  address of uart data register   4   */
    char *statusreg;/*  address of status register      4   */
    char *bsreg;    /*  address of bit set register     4   */
    char *brreg;    /*  address of bit reset register   4   */
    char *modereg;  /*  address of mode register        4   */
    char *cmdreg;   /*  address of command register     4   */
    int putirqmsk;  /*  mask for put IMR reg            2   */
    int getirqmsk;  /*  mask for get IRM reg            2   */
    int dsrbit;     /*  bit mask for DSR line           2   */
    Wait *WritersBlock; /*  blocking semaphore for writing  4   */
    char dummy[6];  /*  reserved data pads to 64 bytes      */
}IO_REC;

#endif      //end of ifdef RS232_BUILD

#define __IOREC__
#endif

/*
** IO_REC defs
*/

#define RS232_IOREC_BUFFERSIZE  1024
/*
** Parameters to tell what kind of STATUS
*/

#define RECEIVE_STAT    1       /*  get number of characters in recieve buff    */
#define SEND_STAT       2       /*  get number of characters in transmit buff   */

/*
** function codes for XIO call
*/

#define SET_BAUD_RATE       0x20
#define SEND_BREAK          0x21
#define WAIT_EMPTY          0x22
#define SET_MODE            0x23
#define SET_INPUT_BUFFER    0x24
#define SET_OUTPUT_BUFFER   0x25
#define MEM_PROTECT         0x26
#define SET_ERROR           0x27
#define SET_ACTIVE          0x28
#define READ_INPORT         0x29
#define RS232_SET_PARITY    0x2a
#define RS232_SET_BITS      0x2b
```

```
#define RS232_SET_STOP      0x2c
#define RS232_GET_BAUD      0x2d
#define RS232_GET_PARITY    0x2e
#define RS232_GET_BITS      0x2f
#define RS232_GET_STOP      0x30
#define RS232_SET_PUTEOT    0x31
#define RS232_SET_RTIMEOUT  0x32
#define RS232_SET_GETEOT    0x33

/*
** buad rate defines
*/

#ifndef BAUD__RATES
#define BAUD__RATES

#define BAUDRATE_75 0x0
#define BAUDRATE_110    0x011
#define BAUDRATE_134    0x022   /*  actaully 134.5  */
#define BAUDRATE_150    0x033
#define BAUDRATE_300    0x044
#define BAUDRATE_600    0x055
#define BAUDRATE_1200   0x066
#define BAUDRATE_2000   0x077
#define BAUDRATE_2400   0x088
#define BAUDRATE_4800   0x099
#define BAUDRATE_1800   0x0aa
#define BAUDRATE_9600   0x0bb
#define BAUDRATE_19200  0x0cc

#define PARITY_NONE 0x10
#define PARITY_EVEN 0x00
#define PARITY_ODD  0x04
#define PARITY_MASK 0x1c

#define STOPBIT_1   7
#define STOPBIT_2   15

/* these are the defines to really talk to the machine with */
#define RS232_BAUDRATE_75       0
#define RS232_BAUDRATE_110      1
#define RS232_BAUDRATE_134      2   /*  actaully 134.5  */
#define RS232_BAUDRATE_150      3
#define RS232_BAUDRATE_300      4
#define RS232_BAUDRATE_600      5
#define RS232_BAUDRATE_1200     6
#define RS232_BAUDRATE_1800     7
#define RS232_BAUDRATE_2000     8
#define RS232_BAUDRATE_2400     9
#define RS232_BAUDRATE_4800     10
#define RS232_BAUDRATE_9600     11
#define RS232_BAUDRATE_19200    12

#define RS232_BAUDRATE_MAX      12

#define RS232_PARITY_NONE   0
#define RS232_PARITY_EVEN   1
#define RS232_PARITY_ODD    2

#define RS232_PARITY_MAX    2

#define RS232_BITS_5    0
#define RS232_BITS_6    1
#define RS232_BITS_7    2
#define RS232_BITS_8    3
```

```
#define RS232_BITS_MAX  3

#define RS232_STOP_1    0
#define RS232_STOP_2    1

#define RS232_STOP_MAX  1

#endif

/*
** bit patterns for output port
*/
#define SET_ERROR_BIT   0x20
#define SET_ACTIVE_BIT  0x40
#define SET_PROTECT_BIT 0x80

/*************************************************
**
** Return Values sent back by RS-232 handler
**
*************************************************/

#define RS232_TIMEOUT   -20     //handler errors start at -20

#ifdef __cplusplus
extern "C" {
#endif

/*
** misc gloabals
*/
extern char acr_shad;

/* function prototypes  */

#pragma function( calling)

extern long R_init(void);

#pragma function()


#ifdef __cplusplus
}
#endif

#endif  //end of RS232__H
```

```
/*********************************************************************
**
** Device driver for the spindle controller
**
** manages the spindle rotation rate and other functions
**
** manages the sine look up tables
**
** keeps a nonvolatile area of memory for parametrs
**
** This file was CHANGED for the experimental version to reflect the
** way that the phase lock loop will produce the frequency reference
** for the experimental gate array SPINDLE6.sch
**
** Serious changes have been made to this file for the REV B PCB
** 3-19-98
**
*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "cio.h"
#include "spindle.h"
#include "sinlut.h"
#include "spinchip.h"
#include "task.h"
#include "dac.h"
#include "frontled.h"
#include "global.h"
#include "rs232.h"
#include "mcfig.h"
#include "strings.h"

#pragma function(calling)

static long SPIN_open(IOCB *iocb,va_list argp);
static long SPIN_close(IOCB *iocb,va_list argp);
static long SPIN_get(IOCB *iocb,va_list argp);
static long SPIN_read(IOCB *iocb,va_list argp);
static long SPIN_put(IOCB *iocb,int a,va_list argp);
static long SPIN_write(IOCB *iocb,va_list argp);
static long SPIN_status(IOCB *iocb,va_list argp);
static long SPIN_xio(int cmd,IOCB *iocb,va_list argp);

#pragma function()

static const H_JVEC SPIN_vec = {
    SPIN_open,
    SPIN_close,
    SPIN_get,
    SPIN_read,
    SPIN_put,
    SPIN_write,
    SPIN_status,
    SPIN_xio,
    SPIN_init
};

static const char SPIN_NAME[] = "SPIN";
static int *SineLut;        /*  pointer used by Look Up table init routine  */
static char RPMstring[20];  /*  place to store accumulated text input   */
static int RPMindex = 0;
static int CurrentRPM = 0;
static int clamp=0;         /*  local clamping flag */
static int motorflag=0;         /*  local motor flag    */
static int head=0;          /*  local head flag     */
```

```
static Wait *SpindleExclude;    /*  exclusion semaphore for static data */
static int DacHandles[4];   //handles for PLL DACs

const int SinLut[256] = {
     0,   804,  1607,  2410,  3211,  4011,  4807,  5601,
  6392,  7179,  7961,  8739,  9511, 10278, 11038, 11792,
 12539, 13278, 14009, 14732, 15446, 16150, 16845, 17530,
 18204, 18867, 19519, 20159, 20787, 21402, 22004, 22594,
 23169, 23731, 24278, 24811, 25329, 25831, 26318, 26789,
 27244, 27683, 28105, 28510, 28897, 29268, 29621, 29955,
 30272, 30571, 30851, 31113, 31356, 31580, 31785, 31970,
 32137, 32284, 32412, 32520, 32609, 32678, 32727, 32757,
 32767, 32757, 32727, 32678, 32609, 32520, 32412, 32284,
 32137, 31970, 31785, 31580, 31356, 31113, 30851, 30571,
 30272, 29955, 29621, 29268, 28897, 28510, 28105, 27683,
 27244, 26789, 26318, 25831, 25329, 24811, 24278, 23731,
 23169, 22594, 22004, 21402, 20787, 20159, 19519, 18867,
 18204, 17530, 16845, 16150, 15446, 14732, 14009, 13278,
 12539, 11792, 11038, 10278,  9511,  8739,  7961,  7179,
  6392,  5601,  4807,  4011,  3211,  2410,  1607,   804,
     0,  -804, -1607, -2410, -3211, -4011, -4807, -5601,
 -6392, -7179, -7961, -8739, -9511,-10278,-11038,-11792,
-12539,-13278,-14009,-14732,-15446,-16150,-16845,-17530,
-18204,-18867,-19519,-20159,-20787,-21402,-22004,-22594,
-23169,-23731,-24278,-24811,-25329,-25831,-26318,-26789,
-27244,-27683,-28105,-28510,-28897,-29268,-29621,-29955,
-30272,-30571,-30851,-31113,-31356,-31580,-31785,-31970,
-32137,-32284,-32412,-32520,-32609,-32678,-32727,-32757,
-32767,-32757,-32727,-32678,-32609,-32520,-32412,-32284,
-32137,-31970,-31785,-31580,-31356,-31113,-30851,-30571,
-30272,-29955,-29621,-29268,-28897,-28510,-28105,-27683,
-27244,-26789,-26318,-25831,-25329,-24811,-24278,-23731,
-23169,-22594,-22004,-21402,-20787,-20159,-19519,-18867,
-18204,-17530,-16845,-16150,-15446,-14732,-14009,-13278,
-12539,-11792,-11038,-10278, -9511, -8739, -7961, -7179,
 -6392, -5601, -4807, -4011, -3211, -2410, -1607,  -804
};

#define SPINDLE_PI      128
#define SPINDLE_120     85

#pragma region("ram=nonvol")    /*  global defs after this will live in non volatile mem    */

SPINDLE_PARAMS SpinParams;  /*  place to store spindle paramters    */

static loopcomp LoopComp[10];
/* no more global memory after this point   */

//-----------------------------------------------------
// This is a TASK to initialize the motor
//-----------------------------------------------------

static int Isin(int theta)
{
    //returns back the the integer sine value of integer angle theta
    // uses the above lookup table.  Theta must have a value between
    // 0->255, or it will make a boo-boo
    return SinLut[theta];
}

void InitMotor(int ConsolHandle,int m)
{
    int loop=1;
    int a,i,j,s,s120,c;

    // initialize controller chip to find index pulse
```

```
    Xio(SPIN_MOTOR,sys.HSpindle,(char *)0,NULL,0l,0);   //turn off motor
    SpinChipDacMode(SPIN_DACMODEDIRECT);    //gain direct access to motor DACs
    Putc(DacHandles[SPIN_DIRECTCONTROL],m);                 //set drive level
    Xio(SPIN_SETPLLMODE,sys.HSpindle,(char *)0,NULL,0l,0);  //set to Direct Drive Mode
    Xio(SPIN_MOTOR,sys.HSpindle,(char *)0,NULL,0l,1);   //turn on motor
    //-------------------------------------
    //total open loop way to turn motor
    // just keep doing this until the index
    // fires
    //-------------------------------------
    i=0;
    while(loop)
    {
        s = Isin(i) ^ 0x8000;
        j = i + SPINDLE_120;    //calculate angle 120 from sin
        j %= 256;               //roll over if needed
        s120 = Isin(j) ^ 0x8000;
        *((int *)(SPIN_SLUTADDRESS)) = s;
        *((int *)(SPIN_SLUTADDRESS + 2)) = s120;
        TDelay(4);              //wait about 40 mSec

        if(Status(ConsolHandle,NULL,0l,RECEIVE_STAT) )
        {
            a = Getc(ConsolHandle);
            if(a == 'X');
                loop = 0;
        }
        ++i;
        if(i==256)
            i = 0;
    }
    //terminate initialization
    Xio(SPIN_MOTOR,sys.HSpindle,(char *)0,NULL,0l,0);   //turn off motor
    Xio(SPIN_SETPLLMODE,sys.HSpindle,(char *)0,NULL,0l,1);  //set to PLL Drive Mode
    SpinChipDacMode(SPIN_DACMODENORMAL);    //return back to normal operation
//  TDelete(0);     //delete self
}


/*-------------------------------------------------------
** functions to manage non-volatile memory
**-------------------------------------------------------*/

static int CheckChecksum(void)
{
    /*
    ** This function returns 0 if checksum of paramter
    ** area is intact, non-zero otherwise
    */
    int i;
    int l;
    unsigned *u,cs;

    if(SpinParams.magic != 0x12345678l)
        return 1;       /*  don't even bother with checksum, its wrong  */
    /*
    ** the checksum is calculated by summing the 16 bit unsigned values
    ** of everything except the checksum.  With the checksum added in,
    ** the result should be zero
    */
    l = sizeof(SPINDLE_PARAMS)/2;   /*  number of words */
    for(cs=0,i=0,u=(unsigned *)&SpinParams;i<l;++i)
        cs += *u++;     /*  calculate sum   */
    return (int)cs;
}
```

```
static void UpdateChecksum(void)
{
    /*
    ** this function recalculates the checksum of the parameter area
    */
    unsigned *u,cs;
    int i,l;

    l = sizeof(SPINDLE_PARAMS)/2 - 1;   /*  number of bytes to calculate    */
    for(i=0,u=(unsigned *)&SpinParams,cs=0;i<l;++i)
        cs += *u++;     /*  calculate sum   */
    SpinParams.checksum = ~cs + 1;      /*  store new checksum  */
}

/*-----------------------------------------------------
** Fucntions to manage the SLUT memory
**-----------------------------------------------------*/

static long CalculateLUTCheckSum(long size)
{
    /*
    ** Calculate the Checksum of the Sine lookup table
    ** checksum is a long, but is calculated by adding up
    ** the unsigned int values that are contained in the
    ** lookup rom
    */
    unsigned *lut;      /*  pointer to lookup memory    */
    long i,checksum=0;

    SpinChipDacMode(SPIN_DACMODENORMAL);    /*  make sure we have access to SLUT    */
    lut = (unsigned *)SPIN_SLUTADDRESS;     /*  address of Sin Look Up Table    */
    for(i=0;i<size;++i)
        checksum += (long)*lut++;   /*  add up all the values   */
    return checksum;
}

static long TCount;

static void SineCallback(double v)
{
    /*
    ** this is the function that stores the sine value v into ram
    ** need to compliment most significant upper bit for DAC
    */
    --TCount;
    *SineLut++ = ((int)(32767.0 * v)) ^ 0x8000;
}

static void InitSineLookupTable(long size,unsigned poles)
{
    SineLut = (int *)SPIN_SLUTADDRESS;  /*  init address of slut ram    */
    SpinChipDacMode(SPIN_DACMODENORMAL);    /*  make sure we have access to SLUT    */
    MakeSineLut(size,poles,SineCallback);
}
/*-----------------------------------------------------
** functions to manage spindle parameters
**-----------------------------------------------------*/

static int GetRange(int a)
{
    int retval;

    if(a < 128)
        retval = 0;
    else if(a < 256)
        retval = 1;
```

```
    else if(a < 512)
        retval = 2;
    else if(a < 1024)
        retval = 3;
    else if(a < 2048)
        retval = 4;
    else if(a < 4096)
        retval = 5;
    else if(a < 8192)
        retval = 6;
    else if(a < 16384)
        retval = 7;
    else
        retval = 8;
    return retval;
}

double round(double a)
{
    int v;

    v = (int)a;
    if((a - (double)v) > 0.5)
        v += 1;
    return (double)v;
}

static void CalculateEntries(int rpm,long reffreq,int counts,unsigned *m,unsigned *n)
{
    /*
    ** This function calculates the required divider ratio for the
    ** phase lock loop.
    ** counts:number of counts per revolution
    ** reffreq:frequency on input of n divider chain
    ** rpm:desired rpm
    ** m: pointer to place to put m value (divider from encoder)
    ** n: pointer to place to put n value (divider from reference)
    **
    ** both n and m are decremented by one to make up for the pipe
    ** delay in the divider chain
    */

    int a,b,s;
    long al;

    for(a=rpm,b=1,s=0;a>255;a>>=1,b<<=1,++s);
    a = 128 << s;
    a = rpm -a;
    al = (((long)a << 6) >> s) + 0x2000;

    *n = (unsigned)al;
    *m = b-1;
}

/*-----------------------------------------------------
** function to handle CIO calls
**-----------------------------------------------------*/

long SPIN_init(void)
{
    long t;
    /*  check the checksum of the parameters area   */
    DacHandles[SPIN_UPPERLIMITS] = Open(UPPERLIMITS,WRITE_ONLY | READ_ONLY);
    DacHandles[SPIN_LOWERLIMITS] = Open(LOWERLIMITS,WRITE_ONLY | READ_ONLY);
    DacHandles[SPIN_LOOPGAIN] = Open(LOOPGAIN,WRITE_ONLY | READ_ONLY | DACMODE_UNSIGNED);
    DacHandles[SPIN_DIRECTCONTROL] = Open(DIRECTCONTROL,WRITE_ONLY | READ_ONLY);
```

```
    if(CheckChecksum())     /*  are spindle parameters OK?  */
    {
        /*  no, complete reinit of everything to defualts   */
        SpinParams.magic = 0x12345678l; /*  Set magic number    */
        SpinParams.SinLUTSize = 2048l;  /*  standard table size */
        SpinParams.m = 0;
        SpinParams.n = 0;
        SpinParams.MotorPoles = 4;
        SpinParams.BaseFreq = 19660800;     /*  oscilator frequency */
        SpinParams.p1 = 0;
        SpinParams.p2 = (SpinParams.SinLUTSize) / 3;
        SpinParams.mode = 0;                /*  binary mode */
        SpinParams.accel = 1000;
        SpinParams.decel = 1000;
        SpinParams.upperlimit = 127;
        SpinParams.lowerlimit = 127;
        SpinParams.Direction = 0;   //0==CCW,1==CW
        InitSineLookupTable(SpinParams.SinLUTSize,SpinParams.MotorPoles);
        SpinParams.SinLUTChecksum = CalculateLUTCheckSum(SpinParams.SinLUTSize);
        UpdateChecksum();
    }

    /*  if not goofed up, check Sine Look Up Table (SLUT) memory    */
    else
    {
        if(SpinParams.SinLUTChecksum != CalculateLUTCheckSum(SpinParams.SinLUTSize))
        {
            /*  if SLUT is messed up, reinitialize SLUT */
            InitSineLookupTable(SpinParams.SinLUTSize,SpinParams.MotorPoles);
            SpinParams.SinLUTChecksum = CalculateLUTCheckSum(SpinParams.SinLUTSize);
            UpdateChecksum();
        }
    }
    /*
    ** do basic hardware initilizations
    */
    t = SpinParams.BaseFreq / 256l;
    t *= 15;
    t /= SpinParams.SinLUTSize / 4;
    *((volatile int *)SPIN_WDDIVIDE) = (int)(t-1);
    Putc(DacHandles[SPIN_LOWERLIMITS],SpinParams.lowerlimit);
    Putc(DacHandles[SPIN_UPPERLIMITS],SpinParams.upperlimit);
    Putc(DacHandles[SPIN_LOOPGAIN],SpinParams.LoopGain);
    *((int *)(SPIN_PHASELIMIT)) = SpinParams.SinLUTSize;
    *((int *)(SPIN_PHASE1)) = SpinParams.p1 + (SpinParams.Direction?((int)(SpinParams.SinLUTSize
/8)):0);
    *((int *)(SPIN_PHASE2)) = SpinParams.p2 + (SpinParams.Direction?((int)(SpinParams.SinLUTSize
/8)):0);
    //
    //This call has nothing to do with rotational direction
    //this must be done so that sine synthesis works correctly
    //
    SpinChipCCW(1);
    SpinChipPllFeedback(SPIN_ENCODERAQUAD);
    SpindleExclude = new Wait(1,"SpindleExclude");  //can only be used once
    SpinIOInitIrq();    /*  initialize io interrupts in spindle chip    */
    /*  add Spindle device driver to device table   */
    return AddHandler(SPIN_NAME,&SPIN_vec);
}

static long SPIN_open(IOCB *iocb,va_list argp)
{
    /*
    ** this function opens up the SPIN for either input (not very useful)
    ** and output
```

```
    */
    SpinParams.mode = iocb->mode;
    SpindleExclude->Pend();
    UpdateChecksum();
    SpindleExclude->Post();
    RPMindex = 0;   /*  initialize text buffer  */
    return 0l;
}

static long SPIN_close(IOCB *iocb,va_list argp)
{
    return 0l;
}

static long SPIN_get(IOCB *iocb,va_list argp)
{
    long a=(long)CurrentRPM;

    return a;
}

static long SPIN_read(IOCB *iocb,va_list argp)
{
    return 0l;
}

static long SPIN_put(IOCB *iocb,int a,va_list argp)
{
    /*
    ** This function will get the RPM in value a in binary mode
    ** In Ascii mode,it will recieve rpm as an ascii string
    ** terminated by a \n
    */
    int i;

    if(SpinParams.mode & SPIN_MODE_ASCII_IN)
    {
        /*  code for recieving RPM in ascii */
        if(a != '\n')
            RPMstring[RPMindex++] = (char)a;
        else
        {
            RPMstring[RPMindex] = '\0'; /*  terminate string    */
            a = atoi(RPMstring);
            goto stuffit;
        }
    }
    else
    {
stuffit:
        CurrentRPM=a;
        CalculateEntries(a,SpinParams.BaseFreq,SpinParams.SinLUTSize,
            &SpinParams.m,&SpinParams.n);
        //---------------------------------
        // calculate comp index
        //---------------------------------
        i = GetRange(a);
        SpindleExclude->Pend();
        Putc(DacHandles[SPIN_LOOPGAIN],LoopComp[i].Gain);
        SpinChipFilterSelect(LoopComp[i].Rolloff);
        Putc(DacHandles[SPIN_LOWERLIMITS],SpinParams.lowerlimit);
        Putc(DacHandles[SPIN_UPPERLIMITS],SpinParams.upperlimit);

        /*
        ** stuff values m and n into spindle controller chip
        */
```

```
        *((volatile int *)(SPIN_N)) = SpinParams.n;
        *((volatile int *)(SPIN_M)) = SpinParams.m;
        UpdateChecksum();
        SpindleExclude->Post();
    }
    return (long)0;
}

static long SPIN_write(IOCB *iocb,va_list argp)
{
    return 0l;
}

static long SPIN_status(IOCB *iocb,va_list argp)
{
    long v=0;

    v |= motorflag;
    v |= clamp << 1;
    v |= head << 2;
    return(v);
}

static long SPIN_xio(int cmd,IOCB *iocb,va_list argp)
{
    long retval=0;
    int error;
    int a,b;
    int *ip,*ip1;
    char *buffer = va_arg(argp,char *);
    long count = va_arg(argp,long);
    int aux = va_arg(argp,int);
    long t;

    switch(cmd)
    {
        case SPIN_INITLUT:
            SpindleExclude->Pend();
            SpinParams.SinLUTSize = (long)aux;
            InitSineLookupTable(SpinParams.SinLUTSize,SpinParams.MotorPoles);
            SpinParams.SinLUTChecksum = CalculateLUTCheckSum(SpinParams.SinLUTSize);
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_UPDATECHECKSUM:
            SpindleExclude->Pend();
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_GETCHECKSUM:
            retval = (long)CheckChecksum();
            break;
        case SPIN_SETBASEFREQ:  /*  set the base frequency used */
            /*
            ** need a long value to get the base frequency, so
            ** use the buffer count
            */
            SpindleExclude->Pend();
            SpinParams.BaseFreq = count;
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_SETENCODER:   /*  set the encoder pitch   */
            SpindleExclude->Pend();
            SpinParams.SinLUTSize = aux;
            UpdateChecksum();
```

```
            SpindleExclude->Post();
            t = SpinParams.BaseFreq / 256l;
            t *= 15;
            t /= SpinParams.SinLUTSize / 4;
            *((volatile int *)SPIN_WDDIVIDE) = (int)(t-1);
            break;
        case SPIN_SETMODE:      /*  set operating mode  */
            SpindleExclude->Pend();
            SpinParams.mode = aux;
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_POLES:
            SpindleExclude->Pend();
            SpinParams.MotorPoles = aux;    /*  set number of poles */
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_PHASING:
            SpindleExclude->Pend();
            SpinParams.p1 = aux;
            SpinParams.p2 = aux + SpinParams.SinLUTSize / 3 + 3;    //DEBUG CHANGE ME bACK
            /*  write data into registers   */
            *((int *)(SPIN_PHASE1)) = SpinParams.p1 + (SpinParams.Direction?((int)(SpinParams.Si
nLUTSize/8)):0);
            *((int *)(SPIN_PHASE2)) = SpinParams.p2 + (SpinParams.Direction?((int)(SpinParams.Si
nLUTSize/8)):0);
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_GETENCODER:
            retval = (long)SpinParams.SinLUTSize;
            break;
        case SPIN_GETMODE:
            retval = (long)SpinParams.mode;
            break;
        case SPIN_GETPOLES:
            retval = SpinParams.MotorPoles;
            break;
        case SPIN_GETPHASING:
            retval = (long)SpinParams.p1;   /*  p2 is related to p1, so only return this one
*/
            break;
        case SPIN_MOTOR:
            motorflag = aux;    /*  ok to do motor thing    */
            SetPortC(FRONTLED_SPINENAMP,aux?1:0);
            break;
        case SPIN_SETACCEL:
            SpindleExclude->Pend();
            if(aux <= 0) aux = 1;   //minimum acceleration
            SpinParams.accel = aux;
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_GETACCEL:
            retval = (long) SpinParams.accel;
            UpdateChecksum();
            break;
        case SPIN_SETDECEL:
            SpindleExclude->Pend();
            if(aux <= 0) aux = 1;   //minimum acceleration
            SpinParams.decel = aux;
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_GETDECEL:
```

```
            retval = (long)SpinParams.decel;
            UpdateChecksum();
            break;
        case SPIN_SETDAC:
            //the value to put into the dac is last value on stack
            // aux contains the DAC number to put the data into
            a = va_arg(argp,int);       //value to stick into dac
            SpindleExclude->Pend();
            switch(aux)
            {
                case SPIN_LOWERLIMITS:
                    SpinParams.lowerlimit = a;
                    UpdateChecksum();
                    break;
                case SPIN_UPPERLIMITS:
                    SpinParams.upperlimit = a;
                    UpdateChecksum();
                    break;
                case SPIN_DIRECTCONTROL:
                    break;
                case SPIN_LOOPGAIN:
                    SpinParams.LoopGain = a;
                    UpdateChecksum();
                    break;
            }
            SpindleExclude->Post();
            Putc(DacHandles[aux],a);
            break;
        case SPIN_GETDAC:
            //dac to read is in AUX
            switch(aux)
            {
                case SPIN_LOWERLIMITS:
                    retval = (long)SpinParams.lowerlimit;
                    UpdateChecksum();
                    break;
                case SPIN_UPPERLIMITS:
                    retval = (long)SpinParams.upperlimit;
                    UpdateChecksum();
                    break;
                case SPIN_DIRECTCONTROL:
                case SPIN_LOOPGAIN:
                    retval = (long)Getc(DacHandles[aux]);
                    break;
            }
            break;
        case SPIN_SETPORT_GPC:
            SetPortCBits(aux);
            break;
        case SPIN_CLEARPORT_GPC:
            ClearPortCBits(aux);
            break;
        case SPIN_GETPORT_GPC:
            retval = GetPortCBits();;
            break;
        case SPIN_SETSTART:
            *((volatile int *)SPIN_START) = aux;
            break;
        case SPIN_GETSTART:
            retval = (long) *((volatile int *)SPIN_START);
            break;
        case SPIN_SETSTOP:
            *((volatile int *)SPIN_STOP) = aux;
            break;
        case SPIN_GETSTOP:
            retval = (long) *((volatile int *)SPIN_STOP);
```

```
            break;
        case SPIN_SETPLL:
            SpinChipPllFeedback(aux?SPIN_ENCODERAQUAD:SPIN_ENCODEREDGES);
            break;
        case SPIN_GETPLL:
            retval = (long)SpinChipGetPllFeedback();
            break;
        case SPIN_SETROLLOFF:   /*  set pll rolloff         */
            SpinChipFilterSelect(aux);
            break;
        case SPIN_GETROLLOFF:   /*  get pll rolloff         */
            retval = SpinChipGetFilterSelect();
            break;
        case SPIN_SETPLLMODE:   /*  SET PLL MODE(DIRECT/PLL)    */
            //-----------------------------------------
            //aux = 2, Motor Drive in Direct Mode
            //aux = 1, Motor Drive in PLL Servo Mode
            //aux = 3, Motor Drive in DSP mode
            //-----------------------------------------
            SpinChipPllMode(aux);
            break;
        case SPIN_GETPLLMODE:   /*  get pll mode(direct/pll     */
            retval = (long)SpinChipGetPllMode();
            break;
        case SPIN_SETDIRECTION:     /*  set spindle direction       */
            SpindleExclude->Pend();
            SpinParams.Direction = aux;
            *((int *)(SPIN_PHASE1)) = SpinParams.p1 + (SpinParams.Direction?((int)(SpinParams.Si
nLUTSize/8)):0);
            *((int *)(SPIN_PHASE2)) = SpinParams.p2 + (SpinParams.Direction?((int)(SpinParams.Si
nLUTSize/8)):0);
            UpdateChecksum();
            SpindleExclude->Post();
            break;
        case SPIN_GETDIRECTION:     /*  get spindle direction       */
            retval = (long)SpinParams.Direction;
            break;
        case SPIN_SETLOOPCOMP:  /*  set loop compensation       */
            //aux contains the parameter
            //first value on stack contains index
            //second value on stack is parameter value
            a = va_arg(argp,int);   //get index of value
            b = va_arg(argp,int);   //get value
            SpindleExclude->Pend();
            switch(aux)
            {
                case SPIN_LOOPCOMP_GAIN:
                    LoopComp[a].Gain = b;
                    break;
                case SPIN_LOOPCOMP_ROLLOFF:
                    LoopComp[a].Rolloff = b;
                    break;
            }
            SpindleExclude->Post();
            break;
        case SPIN_GETLOOPCOMP:  /*  get loop compensation       */
            //aux contains the parameter
            //first value on stack contains index
            a = va_arg(argp,int);   //get index of value
            switch(aux)
            {
                case SPIN_LOOPCOMP_GAIN:
                    retval = (long)LoopComp[a].Gain;
                    break;
                case SPIN_LOOPCOMP_ROLLOFF:
                    retval = (long)LoopComp[a].Rolloff;
```

```
                    break;
                }
                break;
            case SPIN_SETMOTDACMODE:    /*  set access mode to motor dac    */
                //aux = 1:direct access to motor dacs
                //aux = 0:normal access to motor dacs
                SpinChipDacMode(aux?SPIN_DACMODEDIRECT:SPIN_DACMODENORMAL); //gain direct access to
motor DACs
                break;
            case SPIN_SETMOTDAC:    /*  set motor dacs  */
                //aux is motor dac a
                //next value on stack is motor dac b
                a = va_arg(argp,int);
                *((volatile int *)(SPIN_SLUTADDRESS)) = aux ^ 0x8000;
                *((volatile int *)(SPIN_SLUTADDRESS + 2)) = a ^ 0x8000;
                break;
            case SPIN_SETDAC_DIRECT:
                //the value to put into the dac is last value on stack
                // aux contains the DAC number to put the data into
                a = va_arg(argp,int);       //value to stick into dac
                Putc(DacHandles[aux],a);
                break;
            case SPIN_GETAMPENABLE:
                retval = (long)motorflag;
                break;
            case SPIN_WRITESLUT:
                //buffer contain the pointer to block of data
                //aux is the sector number
                ip = ((int *)SPIN_SLUTADDRESS) + aux * 64;  /*  address of slut ram sector  */
                ip1 = (int *)buffer;
                for(a=0;a<64;++a)   //write data to SLUT
                    *ip++ = *ip1++;
                break;
        }
        return retval;
}

//***************************************************************************
//
// Some AUX stuff to go along with the driver.
//
// Rebuilding the motor table takes a long......time.
// This needs to be a TASK
//
// So, this task will be spawned from the outside and do the
// appropriate thing.
//
//***************************************************************************

extern int ConsolHandle;

static void MotorInitTask(void)
{
    InitSineLookupTable(SpinParams.SinLUTSize,SpinParams.MotorPoles);
    SpinParams.SinLUTChecksum = CalculateLUTCheckSum(SpinParams.SinLUTSize);
    UpdateChecksum();

    TCount = 0;
    TDelete(0); //delete self
}

void DoMotorInit(void)
{
    TCB *t;

    TCount =  SpinParams.SinLUTSize * 2 + 1;
```

```
    t = CreateTask(MotorInitTask,512,5,"INIT_MOTOR");
    int sr = EnterCritical();
    ActiveTasks->Insert(t); //let it wait
    ExitCritical(sr);
}

long GetMotorInitStatus(void)
{
    //-------------------------------------------------------
    // just a way to get some indication of when it is done
    //-------------------------------------------------------
    return TCount;
}

int SaveSpindleConfig(char *name)
{
    int retval;
    String *S = new String;
    char *s = S->Get();
    sprintf(s,"D:%s",name); //create path
    retval = SaveMotorConfigFile(s,&SpinParams,LoopComp);
    delete S;
    return retval;
}

int LoadSpindleConfig(char *name)
{
    String *S = new String;
    char *s= S->Get();
    int c;
    long t;

    sprintf(s,"D:%s\r\n",name); //create path
    SpindleExclude->Pend();
    int retval = LoadMotorConfigFile(s,&SpinParams,LoopComp);
    UpdateChecksum();
    SpindleExclude->Post();
    Putc(DacHandles[SPIN_LOWERLIMITS],SpinParams.lowerlimit);
    Putc(DacHandles[SPIN_UPPERLIMITS],SpinParams.upperlimit);
    t = SpinParams.BaseFreq / 256l;
    t *= 15;
    t /= SpinParams.SinLUTSize / 4;
    *((volatile int *)SPIN_WDDIVIDE) = (int)(t-1);
    delete S;
    return retval;
}
```

```
/*********************************************************************
**
** Device driver for the spindle controller
**
** manages the spindle rotation rate and other functions
**
** manages the sine look up tables
**
** keeps a nonvolatile area of memory for parametrs
**
** Serious changes have been made to this file for the REV B PCB
** 3-19-98
**
*********************************************************************/

#ifndef SPINDLE__H
#define SPINDLE__H

struct loopcomp {
    int Gain;               //pll loop gain
    int Rolloff;            //rolloff selection
};


typedef struct {
    long magic;             /*  magic number for these parameters   */
    long SinLUTChecksum;    /*  checksum for sine lookup table  */
    long SinLUTSize;        /*  number of words in Sine lookup table and Counts per rev */
    unsigned MotorPoles;    /*  number of poles in the motor    */
    unsigned m;             /*  m divider calculated value          */
    unsigned n;             /*  n divider calculated value          */
    long BaseFreq;          /*  base frequence of pll reference     */
    unsigned p1;            /*  motor phase control */
    unsigned p2;            /*  motor phase control */
    int mode;               /*  controls mode of how rpm is set, etc    */
    int accel;              /*  spindle acceleration factor     */
    int decel;              /*  spindle de-celeration factor    */
    int upperlimit;         /*  upper voltage clamp for loop    */
    int lowerlimit;         /*  lower voltage clamp for loop    */
    int Direction;          //  rotation direction 0=CCW,1=CW
    int LoopGain;           //  place to store loop gain        */
    unsigned checksum;      /*  checksum for these parameters       */
}SPINDLE_PARAMS;


#define SPIN_MODE_ASCII_IN  0x01    /*  set true if ascii mode  */

#define SPIN_SLUTADDRESS    0xffc80000  /*  address of SLUT ram */

/*
** XIO function call numbers
*/

#define SPIN_INITLUT        0x20
#define SPIN_UPDATECHECKSUM 0x21
#define SPIN_GETCHECKSUM    0x22
#define SPIN_SETBASEFREQ    0x23    /*  set the base frequency used */
#define SPIN_SETENCODER     0x24    /*  set encoder pitch           */
#define SPIN_SETMODE        0x25    /*  set operating mode          */
#define SPIN_POLES          0x26    /*  set the number of motor poles   */
#define SPIN_PHASING        0x27    /*  set up the phase registers  */
#define SPIN_GETENCODER     0x28    /*  get the encoder pitch   */
#define SPIN_GETMODE        0x29    /*  get operating mode  */
#define SPIN_GETPOLES       0x2a    /*  get the number of motor poles   */
#define SPIN_GETPHASING     0x2b    /*  get the phase registers */
#define SPIN_MOTOR          0x2c    /*  activate spindle motor  */
```

```
#define SPIN_SETACCEL       0x2d    /*  set the acceleration for spindle    */
#define SPIN_GETACCEL       0x2e    /*  get the acceleration for spindle    */
#define SPIN_SETDECEL       0x2f    /*  set the decel for the spindle   */
#define SPIN_GETDECEL       0x30    /*  get the decel for the spindle   */

#define SPIN_SETDAC         0x31    /*  set control DAC */
#define SPIN_GETDAC         0x32    /*  get control DAC */

#define SPIN_SETPORT_GPC    0x33    /*  set bits in port GPC    */
#define SPIN_CLEARPORT_GPC  0x34    /*  clear bits in port GPC  */
#define SPIN_GETPORT_GPC    0x35    /*  get bits in port GPC    */
#define SPIN_SETSTART       0x36    /*  SET start sector        */
#define SPIN_GETSTART       0x37    /*  get start sector        */
#define SPIN_SETSTOP        0x38    /*  set stop sector         */
#define SPIN_GETSTOP        0x39    /*  get stop sector         */
#define SPIN_SETPLL         0x3a    /*  select pll source       */
#define SPIN_GETPLL         0x3b    /*  get pll source          */
#define SPIN_SETROLLOFF     0x3c    /*  set pll rolloff         */
#define SPIN_GETROLLOFF     0x3d    /*  get pll rolloff         */
#define SPIN_SETPLLMODE     0X3e    /*  SET PLL MODE(DIRECT/PLL)    */
#define SPIN_GETPLLMODE     0X3f    /*  get pll mode(direct/pll     */
#define SPIN_SETDIRECTION   0x40    /*  set spindle direction       */
#define SPIN_GETDIRECTION   0x41    /*  get spindle direction       */
#define SPIN_SETLOOPCOMP    0x42    /*  set loop compensation       */
#define SPIN_GETLOOPCOMP    0x43    /*  get loop compensation       */
#define SPIN_SETMOTDACMODE  0x44    /*  set access mode to motor dac    */
#define SPIN_SETMOTDAC      0x45    /*  set motor dacs  */
#define SPIN_SETDAC_DIRECT  0x46    /* change dac values without updating parameters    */
#define SPIN_GETAMPENABLE   0x47    /*  return back state of amplifier enable bit   */
#define SPIN_WRITESLUT      0x48    /*  write block of data to sine lookup table    */

//-----------------------------------------
// defines for LOOPCOMP commands
//-----------------------------------------

#define SPIN_LOOPCOMP_GAIN      0
#define SPIN_LOOPCOMP_ROLLOFF   1

/*-------------------------------------
** bit defines for port GPC
**-------------------------------------*/

#define SPIN_GPC_POWER      0x01    /*  controls power supply for motors    */
#define SPIN_GPC_BIT1       0x02
#define SPIN_GPC_I2C        0x04    /*  resets front panel display  */
#define SPIN_GPC_EXTAMPSEL  0x80    /*  select external amp input   */

/*
** aux parametrs for clamp function
*/

#define SPIN_CLAMP_ON   1
#define SPIN_CLAMP_OFF  0

/*
** aux paramters for motor control
*/

#define SPIN_MOTOR_ON   1
#define SPIN_MOTOR_OFF  0

/*
** errors for motor control
*/

#define SPIN_MOTOR_ERROR_CLAMP  1
```

```
#define SPIN_MOTOR_ERROR_HEAD   2

/*
** aux paramters for head loading control
*/

#define SPIN_HEAD_LOAD      1
#define SPIN_HEAD_UNLOAD    0

/*
** Status bit masks
*/

#define SPIN_CLAMP_STATUS   0x02
#define SPIN_MOTOR_STATUS   0x01
#define SPIN_HEAD_STATUS    0x04

/*-----------------------------------------
**
** Defines for the four dacs used by the PLL
**
**-----------------------------------------*/

#define SPIN_LOWERLIMITS    0
#define SPIN_UPPERLIMITS    1
#define SPIN_DIRECTCONTROL  2
#define SPIN_LOOPGAIN       3

/*******************************************************
**
** defines for PLL mode
**
*******************************************************/

#define SPIN_DIRECTDRIVE    2
#define SPIN_PLLDRIVE       1
#define SPIN_DSPDRIVE       3

#ifdef __cplusplus
extern "C" {
#endif

extern long SPIN_init(void);

#ifdef __cplusplus
}
#endif

//---------------------------------------------------------------------
// function prototypes for handling saving and loading of motor
// configurations
//---------------------------------------------------------------------
extern int LoadMotorConfigFile(char *name,SPINDLE_PARAMS *m,loopcomp *l);
extern char *GetMotorConfigFileErrorString(void);
extern int SaveMotorConfigFile(char *name,SPINDLE_PARAMS *mt,loopcomp *lc);
extern void GetLoadMotorConfigError(char *s);
extern void DoMotorInit(void);
long GetMotorInitStatus(void);
int SaveSpindleConfig(char *name);
int LoadSpindleConfig(char *name);

#endif  /* SPINDLE__H   */
```

```
/*-------------------------------------------------------------------
**
** Tachometer Port device driver
**
**-------------------------------------------------------------------*/

#include <stdarg.h>
#include <stdlib.h>
#include "cio.h"
#include "tach.h"
#include "spinchip.h"
#include "task.h"
#include "spindle.h"

#pragma function(calling)

extern "C" {

long TACH_open(IOCB *iocb,va_list argp);
long TACH_close(IOCB *iocb,va_list argp);
long TACH_get(IOCB *iocb,va_list argp);
long TACH_read(IOCB *iocb,va_list argp);
long TACH_put(IOCB *iocb,int a,va_list argp);
long TACH_write(IOCB *iocb,va_list argp);
long TACH_status(IOCB *iocb,va_list argp);
long TACH_xio(int cmd,IOCB *iocb,va_list argp);

}

#pragma function()

static const H_JVEC TACH_vec = {
    TACH_open,
    TACH_close,
    TACH_get,
    TACH_read,
    TACH_put,
    TACH_write,
    TACH_status,
    TACH_xio,
    TACH_init
};

static const char TACH_NAME[] = "TACH";
static Wait *TachEvent;
extern "C" void _TachIrq(void);

/*********************************************************
**
** This interrupt routine is used to display the tackometer
** on the front panel LED display
** This happens, once per second or so.
**
*********************************************************/

extern "C" {

void TACH_IRQ(void)
{
    if(TachEvent->GetCount() < 0)
        TachEvent->Post();  //indicate ready
}


long TACH_init(void)
{
```

```
    /*
    ** all hardware initialization is done here
    */

    TachEvent = new Wait(0,"TachEvent");    //make this not availiable
    *((long *)TACH_VECTOR) = (long)(_TachIrq);
    *((volatile int *)(SPIN_WDVECTOR)) = (TACH_VECTOR >> 2) | 0x1300;   //divide by 20 (.1Hz tim
ebase)

    SpinChipTachIrqEnable(1);   /*  enable interrupt    */

    /*  hardware initialized    */
    return AddHandler(TACH_NAME,&TACH_vec); /*  install device driver   */
}

long TACH_open(IOCB *iocb,va_list argp)
{
    /*
    ** this function opens up the TACH for either input (not very useful)
    ** and output
    */
    return 0l;
}

long TACH_close(IOCB *iocb,va_list argp)
{
    /*
    ** this function terminates i/o to the Tachometer Port and releases it
    ** to be used
    */
    return 0l;
}

long TACH_get(IOCB *iocb,va_list argp)
{
    /*
    ** this function is used for getting data from the Tachometer Port
    */
    long a=0;
    union {
        long v;
        int c[2];
    } cnvrt;
    int c;
    extern SPINDLE_PARAMS SpinParams;
    /*  read in data    */
    TachEvent->Pend();

    cnvrt.c[1] = *((volatile int *)SPIN_RDTACH_L);  /* big endian   */
    cnvrt.c[0] = *((volatile int *)SPIN_RDTACH_H);
    /*
    ** divide this number by the time base to get RPM
    */
    cnvrt.v *= 600;     /*  counts per minute, for .1 second timebase   */
    c = cnvrt.v % SpinParams.SinLUTSize;
    cnvrt.v /= SpinParams.SinLUTSize;   /*  counts per rev      */
    if(c >= 1024) cnvrt.v += 1; //round up
    a |= (unsigned )cnvrt.v;
    return a;
}

long TACH_read(IOCB *iocb,va_list argp)
{
    long i;
    char *b = va_arg(argp,char *);
    long c = va_arg(argp,long);
```

```
    for(i=0;i<c;++i,++b)
        *b = (char)TACH_get(iocb,argp);
    return i;
}

long TACH_put(IOCB *iocb,int a,va_list argp)
{
    /*
    ** this function outputs data a to the Tachometer Port
    ** (not very useful)
    */
    return (long)0;
}

long TACH_write(IOCB *iocb,va_list argp)
{
    return 0l;
}

long TACH_status(IOCB *iocb,va_list argp)
{
    /*
    ** this function get the device status for the Tachometer Port
    */
    return((long)0);
}

static long LongGetTach(void)
{
    union {
        long v;
        int c[2];
    } cnvrt;
    int c;

    extern SPINDLE_PARAMS SpinParams;
    /*  read in data    */
    TachEvent->Pend();

    cnvrt.c[1] = *((volatile int *)SPIN_RDTACH_L);  /* big endian   */
    cnvrt.c[0] = *((volatile int *)SPIN_RDTACH_H);
    /*
    ** divide this number by the time base to get RPM
    */
    cnvrt.v *= 6000;        /*  counts per minute, for .1 second timebase   */
    c = cnvrt.v % SpinParams.SinLUTSize;
    cnvrt.v /= SpinParams.SinLUTSize;   /*  counts per rev      */
    if(c >= 1024) cnvrt.v += 1; //round up
    return cnvrt.v;
}

long TACH_xio(int cmd,IOCB *iocb,va_list argp)
{
    long retval=0;
    /*
    ** this function is used to perform special functions to the Tachometer Port
    */
    switch(cmd)
    {
        case TACH_SET_TIME_BASE:
            break;
        case TACH_GET_LONG:
            retval = LongGetTach();
            break;
    }
```

```
    return retval;
}

}   //end of extern "C"
```

```
/*-------------------------------------------------------------------------
**
** Header File for Spindle Tachometer device driver
**
**-------------------------------------------------------------------------*/

#ifndef TACH__H
#define TACH__H



/* Hardware defines */
#define TACH_VECTOR     0x120

#define TACH_LOW    0xffdc0009
#define TACH_MID    0xffdc000b
#define TACH_HIGH   0xffdc000d
/*  XIO function calls  */

#define TACH_SET_TIME_BASE  0x20
#define TACH_GET_LONG       0x21

#ifdef __cplusplus
extern "C" {
#endif

#pragma function(calling)
extern int ConvertToBCD(long v,char * bcd_string);
#pragma function()

extern long TACH_init(void);

#ifdef __cplusplus
}
#endif

#endif
```

```
    ;*************************************************************************
    ;
    ; Spindle Chip interrupt handlers
    ;
    ; These are the assembly language interfaces for
    ; the interrupt ports in the spindle chip
    ;
    ; these don't handle the tachometer.
    ;
    ;   Interrupt   Vector
    ;   0           0
    ;   1           1
    ;   2           2
    ;   3           3
    ;   4           4
    ;   Ticker      5
    ;   Up          6
    ;   Down        7
    ;
    ;*************************************************************************

    XDEF SpinIrq0,SpinIrq1,SpinIrq2,SpinIrq3,SpinIrq4,SpinDown,SpinUp
    XDEF SpinTicker
    XREF _HandleIrq0
    XREF _HandleIrq1
    XREF _HandleIrq2
    XREF _HandleIrq3
    XREF _HandleIrq4
    XREF _HandleUp
    XREF _HandleDown
    XREF _HandleTickerIrq
    XREF _EnterInterrupt
    XREF _TimerTicker
    xdef __TachIrq
    xref _TACH_IRQ

    .FREF _ExitInterrupt,2

    ;
    ; Mask Bits to Clear Interrupts
    ;
CLEARIRQ0   =   0x0001
CLEARIRQ1   =   0x0002
CLEARIRQ2   =   0x0004
CLEARIRQ3   =   0x0008
CLEARIRQ4   =   0x0010
CLEARTICKER =   0x0020
CLEARUP     =   0x0040
CLEARDOWN   =   0x0080

    ; Spindle chip addresses

SPINCHIP    =   0XFFDC0000

SPIN_WDCLRFLAGS =       (SPINCHIP+20)

    SECTION code

SpinIrq0:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    jsr     _EnterInterrupt             ;increment interrupt count
    move.w  #CLEARIRQ0,d0               ;load bitmask to clear interrupt
    move.w  d0,SPIN_WDCLRFLAGS          ;write to clear interrupt register
    jsr     _HandleIrq0                 ;handle the interrupt
    move.w  60(a7),d0                   ;get status register
    andi.w  #0x0700,d0                  ;mask off junk
```

```
    asr.w   #8,d0                   ;shift over 8 bits
    move.w  d0,-(a7)                ;push onto stack
    jsr     _ExitInterrupt          ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6       ;restore processor context
    rte                             ;return to interrupted code

SpinIrq1:
    movem.l d0-d7/a0-a6,-(a7)       ;save processor context
    jsr     _EnterInterrupt         ;increment interrupt count
    move.w  #CLEARIRQ1,d0           ;load bitmask to clear interrupt
    move.w  d0,SPIN_WDCLRFLAGS      ;write to clear interrupt register
    jsr     _HandleIrq1             ;handle the interrupt
    move.w  60(a7),d0               ;get status register
    andi.w  #0x0700,d0              ;mask off junk
    asr.w   #8,d0                   ;shift over 8 bits
    move.w  d0,-(a7)                ;push onto stack
    jsr     _ExitInterrupt          ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6       ;restore processor context
    rte                             ;return to interrupted code

SpinIrq2:
    movem.l d0-d7/a0-a6,-(a7)       ;save processor context
    jsr     _EnterInterrupt         ;increment interrupt count
    move.w  #CLEARIRQ2,d0           ;load bitmask to clear interrupt
    move.w  d0,SPIN_WDCLRFLAGS      ;write to clear interrupt register
    jsr     _HandleIrq2             ;handle the interrupt
    move.w  60(a7),d0               ;get status register
    andi.w  #0x0700,d0              ;mask off junk
    asr.w   #8,d0                   ;shift over 8 bits
    move.w  d0,-(a7)                ;push onto stack
    jsr     _ExitInterrupt          ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6       ;restore processor context
    rte                             ;return to interrupted code

SpinIrq3:
    movem.l d0-d7/a0-a6,-(a7)       ;save processor context
    jsr     _EnterInterrupt         ;increment interrupt count
    move.w  #CLEARIRQ3,d0           ;load bitmask to clear interrupt
    move.w  d0,SPIN_WDCLRFLAGS      ;write to clear interrupt register
    jsr     _HandleIrq3             ;handle the interrupt
    move.w  60(a7),d0               ;get status register
    andi.w  #0x0700,d0              ;mask off junk
    asr.w   #8,d0                   ;shift over 8 bits
    move.w  d0,-(a7)                ;push onto stack
    jsr     _ExitInterrupt          ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6       ;restore processor context
    rte                             ;return to interrupted code

SpinIrq4:
    movem.l d0-d7/a0-a6,-(a7)       ;save processor context
    jsr     _EnterInterrupt         ;increment interrupt count
    move.w  #CLEARIRQ4,d0           ;load bitmask to clear interrupt
    move.w  d0,SPIN_WDCLRFLAGS      ;write to clear interrupt register
    jsr     _HandleIrq4             ;handle the interrupt
    move.w  60(a7),d0               ;get status register
    andi.w  #0x0700,d0              ;mask off junk
    asr.w   #8,d0                   ;shift over 8 bits
    move.w  d0,-(a7)                ;push onto stack
    jsr     _ExitInterrupt          ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6       ;restore processor context
    rte                             ;return to interrupted code

SpinUp:
    movem.l d0-d7/a0-a6,-(a7)       ;save processor context
    jsr     _EnterInterrupt         ;increment interrupt count
    move.w  #CLEARUP,d0             ;load bitmask to clear interrupt
```

```
    move.w  d0,SPIN_WDCLRFLAGS          ;write to clear interrupt register
    jsr     _HandleUp                   ;handle the interrupt
    move.w  60(a7),d0                   ;get status register
    andi.w  #0x0700,d0                  ;mask off junk
    asr.w   #8,d0                       ;shift over 8 bits
    move.w  d0,-(a7)                    ;push onto stack
    jsr     _ExitInterrupt              ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6           ;restore processor context
    rte                                 ;return to interrupted code

SpinDown:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    jsr     _EnterInterrupt             ;increment interrupt count
    move.w  #CLEARDOWN,d0               ;load bitmask to clear interrupt
    move.w  d0,SPIN_WDCLRFLAGS          ;write to clear interrupt register
    jsr     _HandleDown                 ;handle the interrupt
    move.w  60(a7),d0                   ;get status register
    andi.w  #0x0700,d0                  ;mask off junk
    asr.w   #8,d0                       ;shift over 8 bits
    move.w  d0,-(a7)                    ;push onto stack
    jsr     _ExitInterrupt              ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6           ;restore processor context
    rte                                 ;return to interrupted code

SpinTicker:
    movem.l d0-d7/a0-a6,-(a7)           ;save processor context
    jsr     _EnterInterrupt             ;increment interrupt count
    move.w  #CLEARTICKER,d0             ;load bitmask to clear interrupt
    move.w  d0,SPIN_WDCLRFLAGS          ;write to clear interrupt register
;
;   This handler is used to provide realtime interrupt
;   for the multitasking kernel
;
    jsr     _TimerTicker                ;handle timer interrupt
    move.w  60(a7),d0                   ;get status register
    andi.w  #0x0700,d0                  ;mask off junk
    asr.w   #8,d0                       ;shift over 8 bits
    move.w  d0,-(a7)                    ;push onto stack
    jsr     _ExitInterrupt              ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6           ;restore processor context
    rte                                 ;return to interrupted code


;*****************************************************************************
;
; Interrupt handler for the Tach
;
;*****************************************************************************
;
__TachIrq:
    movem.l d0-d7/a0-a6,-(a7)       ;save the current task's context
    jsr     _EnterInterrupt             ;increment interrupt count
    jsr     _TACH_IRQ               ;call tach handler
    move.w  60(a7),d0               ;get status register
    andi.w  #0x0700,d0              ;mask off junk
    asr.w   #8,d0                   ;shift over 8 bits
    move.w  d0,-(a7)                ;push onto stack
    jsr     _ExitInterrupt          ;indicate we are exiting an ISR
    movem.l (a7)+,d0-d7/a0-a6       ;restore interrupted tasks context
    rte
```

```
/*****************************************************************
**
** RAMDISK.CPP
**
**  This is a device driver for a RAM disk.  It uses RAM to make a
** virtual disk drive that has sectors and everything.
**
** The data for this thing must live in non-volatile memory.  Due to
** the nature of the beast, memory must be allocated before hand.  It
** cannot be dynamically allocated because of the way startup code works
**
** I admit it, some of this code is REALLY UGLY!!!
**
*****************************************************************/

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdarg.h>
#include "cio.h"
#include "ramdisk.h"
#include "stack.h"

#pragma function(calling)

static long D_open(IOCB *iocb,va_list argp);
static long D_close(IOCB *iocb,va_list argp);
static long D_get(IOCB *iocb,va_list argp);
static long D_read(IOCB *iocb,va_list argp);
static long D_put(IOCB *iocb,int a,va_list argp);
static long D_write(IOCB *iocb,va_list argp);
static long D_status(IOCB *iocb,va_list argp);
static long D_xio(int cmd,IOCB *iocb,va_list argp);

#pragma function()

static const H_JVEC d_vec = {
    D_open,
    D_close,
    D_get,
    D_read,
    D_put,
    D_write,
    D_status,
    D_xio,
    D_init
};

static RamDisk *RamDiskDescp;       //ram disk descriptor table
static unsigned char *RamFat;               //start of FAT, gets initialized at power up
static unsigned char *RamData;              //start of data, gets initialized at power up
static unsigned *Checksums;         //checksums for Sectors

#pragma region("ram = nonvol")

//all data declared here will be in non-volatile memory

static unsigned char RamDiskData[(RAMDISK_MAXSECTORS+2) * 512l];    //ram disk data

//-----------------------------------------------------------------
// The first 256 bytes of RamDiskData is for the Ram Disk Descriptor Table
// The second 256 bytes of RamDiskData is for the FAT.
// The third 512 bytes of RamDiskData is for sector checksums.
// This will use up the first two "clusters" of data.
// The fat is basically a table that points to the NEXT sector for a file
// Sectors have values from 0->0xfd.  0xfe indicates end of file.
```

```
// 0xff indicates free sector.
//---------------------------------------------------------------------

//---------------------------------------------------------------------
//
// Misc routines for controlling ram disk
//
//---------------------------------------------------------------------

static unsigned int CalculateFatChecksum(void)
{
    int i;
    unsigned cs=0;

    for(i=0;i<256;++i)
        cs += (unsigned)RamFat[i];
    return cs;
}

static unsigned int CalculateDataChecksum(int sector)
{
    int i;
    unsigned cs=0;
    unsigned char *p = RamData + ((long)sector << 9);

    for(i=0;i<512;++i)
        cs += (unsigned)*p++;
    return cs;
}

static void Format(void)    //initialize ram disk
{
    int i;
    long *p;
    unsigned char *f;
    unsigned char fatinit = 0xff;

    for(p=(long *)RamData,i=0;i< ((RAMDISK_MAXSECTORS) * 128);++i)
        *p++ = 0;
    for(f=RamFat,i=0;i<256;++i)
    {
        if(i == RAMDISK_MAXSECTORS) fatinit = 0xfe;
        *f++ = fatinit;     /*  initaialize Fat table to all availiable */
    }
    //
    // Directory is always the first file on the "DRIVE"
    //
    RamFat[0] = 1;
    RamFat[1] = 2;
    RamFat[2] = 3;
    RamFat[3] = 0xfe;   //four sectors for directory 128 files

    f = RamData;
    for(i=0;i<2048;i+=16)
        f[i] = '?';     //indicates spot is open for file
    //
    //initialize Ram Disk Descriptor
    //
    RamDiskDescp->magic = 0x12345678;
    RamDiskDescp->FatChecksum = CalculateFatChecksum();
    RamDiskDescp->Directory = 0;
    RamDiskDescp->DirSize = 4;
    for(i=0;i<RAMDISK_MAXSECTORS;++i)
        Checksums[i] = CalculateDataChecksum(i);
}
```

```
static DirEntry *DirSearch(char *name,int *sect)
{
    int i;
    DirEntry *d=NULL;
    int sector = RamDiskDescp->Directory;   //get first sector of directory
    while(sector != 0xfe)   //process all direcotry sectors
    {
        d = (DirEntry *)RamData + ((long)sector << 9);
        for(i=0;i<32;++i)
        {
            if(strncmp(d->name,name,8) == 0)
            {
                *sect = sector;
                return d;   //return start sector
            }
            d++;
        }
        sector = RamFat[sector];    //next sector please
    }
    return NULL;
}

static void DeallocateSectors(int sector)
{
    //------------------------------------------------------
    //this function dealocates a string of allocated sectors
    //------------------------------------------------------
    Stack *s = new Stack(RAMDISK_MAXSECTORS);

    do
    {
        s->Push(sector);
        sector = RamFat[sector];    /*  next sector */
    }while(sector != 0xfe);
    while((sector = s->Pop()) != -1)
        RamFat[sector] = 0xff;      //mark availiable
    RamDiskDescp->FatChecksum = CalculateFatChecksum();
    delete s;
}

static int GetSector(void)
{
    int i,loop,retval=-1;

    for(i=0,loop=1;i < (RAMDISK_MAXSECTORS) && loop;++i)    //search for a free sector
    {
        if(RamFat[i] == 0xff)   //free sector
        {
            loop = 0;
            retval = i;
        }
    }
    return retval;
}

static DirEntry *GetDirectoryEntry(int *sect)
{
    int i;
    DirEntry *d=NULL;
    int sector = RamDiskDescp->Directory;   //get first sector of directory

    while(sector != 0xfe)   //process all direcotry sectors
    {
        d = (DirEntry *)RamData + ((long)sector << 9);
        for(i=0;i<32;++i)
        {
```

```
            if(d->name[0] == '?')   //is this sector availiable?
            {
                *sect = sector;
                return d;   //return start sector
            }
            d++;
        }
        sector = RamFat[sector];    //next sector please
    }
    return d;
}

static int Flush(FCB *fcb)
{
    //flush any buffers to ram disk
    int retval=0;
    int i,j;
    unsigned char *d;
    int r;
    int sector;
    int s;

    if(fcb->buffer && (fcb->FileSize > fcb->Dir->size)) //is there a buffer to flush?
    {
        if((r = fcb->SectorIndex) == 0) /*  even block size */
        {
            d = RamData + fcb->CurrentSector * 512;
            memcpy(d,fcb->buffer,fcb->bufferindex); //copy data
            Checksums[fcb->CurrentSector] = CalculateDataChecksum(fcb->CurrentSector);
            fcb->Dir->size += fcb->bufferindex;
            fcb->SectorIndex = fcb->bufferindex;
            fcb->bufferindex = 0;
        }
        else
        {
            d = RamData + ((long)fcb->CurrentSector << 9) + r;  //point to end of data
            if(fcb->bufferindex < 512-r)
            {
                memcpy(d,fcb->buffer,fcb->bufferindex);
                fcb->SectorIndex = fcb->bufferindex + r;
                fcb->Dir->size +=fcb->bufferindex;
                fcb->bufferindex = 0;
            }
            else
            {
                memcpy(d,fcb->buffer,512-r);
                Checksums[fcb->CurrentSector] = CalculateDataChecksum(fcb->CurrentSector);
                sector = GetSector();   //get a new sector
                RamFat[fcb->CurrentSector] = sector;    //link new sector
                fcb->CurrentSector = sector;
                RamFat[fcb->CurrentSector] = 0xfe;  //mark end of file
                d = RamData + ((long)fcb->CurrentSector << 9);
                memcpy(d,fcb->buffer + (512 - r),fcb->bufferindex - (512 - r));
                fcb->Dir->size +=fcb->bufferindex;
                fcb->SectorIndex = fcb->bufferindex - (512 - r);
                fcb->bufferindex = 0;
            }
        }
        RamDiskDescp->FatChecksum = CalculateFatChecksum();
        Checksums[fcb->DirSector] = CalculateDataChecksum(fcb->DirSector);
    }
    return retval;
}

static void FixSpec(char *s)
{
```

```
    int i;
    int flag;

    for(i=0,flag=0;i<8;++i)
    {
        if(s[i] == '*')
            flag = 1;
        if(flag)
            s[i] = '?';
    }
}

static int MatchSpec(char *s,char *n)
{
    //does a string compare, returns 1 if match, 0 if no match
    int retval = 1; //assume it has matched already
    int i;

    if(n[0] == '?')     //deleted file?
        retval = 0;
    else
    {
        for(i=0;(i<8) && retval;++i)    //check while still matched
        {
            if(s[i] != '?') //this is an automatic match
                {
                if(s[i] != n[i])
                    retval = 0; //did not match
            }
        }
    }
    return retval;
}

//---------------------------------------------------------------------------
//
// These routines are the CIO entry points
//
//---------------------------------------------------------------------------
//
long D_init()
{
    RamData = &RamDiskData[RAMDISK_DATA];
    RamFat = &RamDiskData[RAMDISK_FAT];
    Checksums = (unsigned *)&RamDiskData[RAMDISK_CHECKSUMS];
    RamDiskDescp = (RamDisk *)&RamDiskData[RAMDISK_DESCRIPTOR];
    if(RamDiskDescp->magic != 0x12345678)
        Format();
    return AddHandler("D",&d_vec);
}

static long D_open(IOCB *iocb,va_list argp)
{
    //file name is pointed to by iocb->dev_name
    FCB *fcb = new FCB;
    DirEntry *d;
    long retval = 0;
    int sector;
    iocb->p = (void *)fcb;
    char *n = strchr(iocb->dev_name,':');   //find start of file name
    if((!n) || (!n[1])) //if null pointer or first character is 0
    {
        //then it is a bad file name
        retval = RAMDISK_BADNAME;   //something wrong with file name
    }
    else
```

```
    {
        //otherwise, process the open request
        ++n;
        if((d = DirSearch(n,&sector)) != NULL)  //search for file name
        {   //file was found, check to see what mode we are in
            fcb->DirSector = sector;
            fcb->StartSector = d->start;    //get start sector
            fcb->CurrentSector = d->start;  //current sector
            fcb->FilePointer = 0;           //always start at begin of file
            fcb->Dir = d;                   //remember directory entry
            fcb->SectorIndex = 0;
            fcb->bufferindex = 0;
            if(iocb->mode & WRITE_ONLY) //we are going to be writing, use buffer
            {
                fcb->buffer = new unsigned char[512];   //allocate buffer
                if(iocb->mode & READ_ONLY)  //opened for update?
                {
                    fcb->FileSize = d->size;    //get file size
                }
                else
                {
                    //purely write only, erase file, esentially
                    fcb->FileSize = 0;
                    DeallocateSectors(fcb->StartSector);
                    RamFat[fcb->StartSector] = 0xfe;    //now the last sector
                }
            }
            else
            {
                fcb->buffer = 0;
                fcb->FileSize = d->size;
            }
        }
        else    //could not file the desired file name
        {
            if(iocb->mode & WRITE_ONLY) //create file if write
            {
                if((d = GetDirectoryEntry(&sector)) != NULL)
                {
                    if( (d->start = GetSector()) >= 0)
                    {
                        fcb->DirSector = sector;
                        strncpy(d->name,n,8);
                        fcb->StartSector = d->start;
                        d->size = 0;
                        fcb->CurrentSector = d->start;
                        fcb->FileSize = 0;
                        fcb->FilePointer = 0;
                        fcb->Dir = d;
                        fcb->SectorIndex = 0;
                        fcb->buffer = new unsigned char[512];
                        fcb->bufferindex=0;
                        RamFat[fcb->StartSector] = 0xfe;
                    }
                    else
                    {
                        retval = RAMDISK_NOSECTORS;
                    }
                }

                else
                {
                    retval = RAMDISK_FILENOTFOUND;  //could not open file
                }
            }
            else
```

```
            {
                retval = RAMDISK_NODIRECTORIES;
            }
        }
    }
    if(retval)
    {
        delete fcb; //open failed, delete file control block
        iocb->p = (void *)0;
    }
    else
        RamDiskDescp->FatChecksum = CalculateFatChecksum();
    return retval;
}

static long D_close(IOCB *iocb,va_list argp)
{
    long retval=0;

    FCB *fcb = (FCB *)iocb->p;
    if(iocb->mode & WRITE_ONLY)
        retval = Flush(fcb);        //flush data;
    iocb->p = 0;                //get rid of fcb
    if(fcb->buffer)
        delete[] fcb->buffer;       //free buffer ram if any
    delete fcb;
    return retval;                  //free up ram
}

static long D_get(IOCB *iocb,va_list argp)
{
    long retval;
    unsigned char *p;

    FCB *fcb = (FCB *)iocb->p;
    p = RamData + ((long)fcb->CurrentSector << 9) + fcb->SectorIndex++;
    if(fcb->FilePointer >= fcb->Dir->size)
        retval = RAMDISK_EOF;
    else
    {
        fcb->FilePointer++;
        if(fcb->SectorIndex == 512)
        {
            fcb->CurrentSector = RamFat[fcb->CurrentSector];
            fcb->SectorIndex = 0;
        }
        retval = (long)*p;
    }
    return retval;
}

static long D_read(IOCB *iocb,va_list argp)
{
    FCB *fcb = (FCB *)iocb->p;
    long bytesread=0;
    int r;
    unsigned char *s = RamData + ((long)fcb->CurrentSector << 9) + fcb->SectorIndex;
    int len,l,loop;
    char *buff = va_arg(argp,char *);
    long count = va_arg(argp,long);
    long c = count;
    char *p = buff;

    //-------------------------------------------------------------
    //read chucks of 512 into buff as long as count is greater than 0
    //if the first sector is not a complete 512 bytes, then bring it
```

```
    //up to snuff
    //return actual number of bytes read
    //-------------------------------------------------------------
    if((r = fcb->SectorIndex) != 0) //less than 512 bytes in sector
    {
        if(c < (512-r)) len = c; else len = 512-r;
        if((l = fcb->FileSize - fcb->FilePointer) < len) len = l;
        memcpy(p,s,len);    //copy what is left in this sector
        c -= len;
        p += len;
        bytesread += len;
        fcb->FilePointer += len;
        if((fcb->SectorIndex += len) == 512)
        {
            fcb->SectorIndex = 0;
            fcb->CurrentSector = RamFat[fcb->CurrentSector];    //next sector
        }
    }
    loop = c;
    while(c > 0 && loop)
    {
        s = RamData + ((long)fcb->CurrentSector << 9);
        if((l = fcb->FileSize - fcb->FilePointer) < 512)
        {
            len = l;
            loop = 0;
        }
        else
            len = 512;
        if(c < len)
            len = c;
        if(len != 512) fcb->SectorIndex += len;
        memcpy(p,s,len);
        p+= len;
        c -= len;
        bytesread += len;
        fcb->FilePointer += len;
        if(len == 512)
            fcb->CurrentSector = RamFat[fcb->CurrentSector];    //next sector
    }
    return bytesread;
}

static long D_put(IOCB *iocb,int a,va_list argp)
{
    /*-------------------------------------------------------------------
    ** Write to a sector
    **-------------------------------------------------------------------*/
    int r;
    unsigned char *d;
    int sector;
    long retval=0;

    FCB *fcb = (FCB *)iocb->p;  //get file control block
    fcb->buffer[fcb->bufferindex++] = (unsigned char)a;
    fcb->FilePointer++;
    fcb->FileSize++;
    if(fcb->bufferindex == 512) //is it time to flush buffer
    {
        if((r = fcb->SectorIndex) == 0) /*  even block size */
        {
            d = RamData + (fcb->CurrentSector << 9);
            memcpy(d,fcb->buffer,512);  //copy data
            Checksums[fcb->CurrentSector] = CalculateDataChecksum(fcb->CurrentSector);
            sector = GetSector();
            RamFat[fcb->CurrentSector] = sector;
```

```
            fcb->CurrentSector = sector;
            RamFat[sector] = 0xfe;  //mark end of file;
        }
        else
        {
            d = RamData + ((long)fcb->CurrentSector << 9) + r;
            memcpy(d,fcb->buffer,512-r);
            Checksums[fcb->CurrentSector] = CalculateDataChecksum(fcb->CurrentSector);
            sector = GetSector();
            RamFat[fcb->CurrentSector] = sector;
            fcb->CurrentSector = sector;
            RamFat[fcb->CurrentSector] = 0xfe;  //mark end of file
            d = RamData + fcb->CurrentSector * 512;
            memcpy(d,fcb->buffer + (512 -r),r);
            Checksums[fcb->CurrentSector] = CalculateDataChecksum(fcb->CurrentSector);
        }
        fcb->bufferindex = 0;
        fcb->Dir->size += 512;  //update directory file size;
        RamDiskDescp->FatChecksum = CalculateFatChecksum();
        Checksums[fcb->DirSector] = CalculateDataChecksum(fcb->DirSector);

    }
    return retval;
}

static long D_write(IOCB *iocb,va_list argp)
{
    FCB *fcb = (FCB *)iocb->p;
    long byteswritten=0;
    int r;
    int len,l;
    int sector;
    char *buff = va_arg(argp,char *);
    long count = va_arg(argp,long);
    long c = count;
    char *p = buff;
    //---------------------------------------------------------------
    //read chucks of 512 into buff as long as count is greater than 0
    //if the first sector is not a complete 512 bytes, then bring it
    //up to snuff
    //return actual number of bytes read
    //---------------------------------------------------------------
    if(fcb->bufferindex)
        Flush(fcb);
    unsigned char *s = RamData + ((long)fcb->CurrentSector << 9) + fcb->SectorIndex;
    if((r = fcb->SectorIndex) != 0) //less than 512 bytes in sector
    {
        if(c < (512-r)) len = c; else len = 512-r;
        memcpy(s,p,len);    //copy what is left in this sector
        Checksums[fcb->CurrentSector] = CalculateDataChecksum(fcb->CurrentSector);
        c -= len;
        p += len;
        byteswritten += len;
        fcb->FilePointer += len;
        fcb->FileSize += len;
        fcb->Dir->size = fcb->FileSize;
        if((fcb->SectorIndex += len) == 512)
        {
            sector = GetSector();
            RamFat[fcb->CurrentSector] = sector;
            RamFat[sector] = 0xfe;
            fcb->CurrentSector = sector;    //next sector
            fcb->SectorIndex=0;
        }
    }
    while(c > 0)
```

```
    {
        s = RamData + ((long)fcb->CurrentSector << 9);
        if(c > 512)
            len = 512;
        else
            len = c;
        fcb->SectorIndex += len;
        memcpy(s,p,len);
        Checksums[fcb->CurrentSector] = CalculateDataChecksum(fcb->CurrentSector);
        p+= len;
        c -= len;
        byteswritten += len;
        fcb->FilePointer += len;
        fcb->FileSize += len;
        fcb->Dir->size = fcb->FileSize;
        if(fcb->SectorIndex == 512)
        {
            sector = GetSector();
            RamFat[fcb->CurrentSector] = sector;
            RamFat[sector] = 0xfe;
            fcb->CurrentSector = sector;    //next sector
            fcb->SectorIndex = 0;
        }
    }
    RamDiskDescp->FatChecksum = CalculateFatChecksum();
    Checksums[fcb->DirSector] = CalculateDataChecksum(fcb->DirSector);
    return byteswritten;
}

static long D_status(IOCB *iocb,va_list argp)
{
    FCB *fcb = (FCB *)iocb->p;
    long retval=0;
    int i;

    if(fcb)
        retval = fcb->FileSize - fcb->FilePointer;
    else    //if there is no FCB, then return free bytes in drive
        for(i=0;i<RAMDISK_MAXSECTORS;++i)
            if(RamFat[i] == 0xff) retval += 512;

    return retval;  //return number of bytes left in file
}

static long D_xio(int cmd,IOCB *iocb,va_list argp)
{
    long retval = 0;
    FCB *fcb = (FCB *)iocb->p;
    Ffblk *ff;
    DirEntry *d;
    int loop;
    char *p;
    char *buffer = va_arg(argp,char *);
    long count = va_arg(argp,long);
    int aux = va_arg(argp,int);

    switch(cmd)
    {
        case RAMDISK_FLUSH:
            if(fcb)
                retval = (long)Flush(fcb);
            else
                retval = RAMDISK_EOF;
            break;
        case RAMDISK_FINDFIRST:
            //-----------------------------------------------------
```

```
        // a pointer to a Ffblk is the first parameter of argp
        //-------------------------------------------------------
        ff = va_arg(argp,Ffblk *);
        strncpy(ff->spec,buffer,8);
        ff->sector = RamDiskDescp->Directory;
        ff->index = 0;
        d = (DirEntry *)(RamData + ((long)ff->sector << 9));
        FixSpec(ff->spec);  //process wild cards
        loop=1;
        do
        {
            if(MatchSpec(ff->spec,d->name))
            {
                loop = 0;
                strncpy(ff->name,d->name,8);
                ff->name[8] = 0;
                ff->size = d->size;
                ff->attrib = d->attrib;
            }
            else
            {
                ++d;
                ff->index++;
                if(ff->index == 32) //run out of sector
                {
                    if(RamFat[ff->sector] != 0xfe)  //end?
                    {
                        ff->sector = RamFat[ff->sector];
                        d = (DirEntry *)(RamData + ((long)ff->sector << 9));
                        ff->index = 0;
                    }
                    else
                    {
                        loop = 0;
                        retval = RAMDISK_EOF;
                    }
                }
            }
        }while(loop);   //search until first entry found
        break;
    case RAMDISK_FINDNEXT:
        //-------------------------------------------------------
        // a pointer to a Ffblk is the first parameter of argp
        //-------------------------------------------------------
        ff = va_arg(argp,Ffblk *);
        ff->index++;    //increment index
        if(ff->index == 32) //run out of sector
        {
            if(RamFat[ff->sector] != 0xfe)  //end?
            {
                ff->sector = RamFat[ff->sector];
                ff->index = 0;
            }
            else
            {
                retval = RAMDISK_EOF;
            }
        }
        if(!retval)
        {
            loop = 1;
            d = (DirEntry *)(RamData + ((long)ff->sector << 9) + (ff->index << 4));
        }
        while(loop)
        {
            if(MatchSpec(ff->spec,d->name))
```

```
            {
                loop = 0;
                strncpy(ff->name,d->name,8);
                ff->name[8] = 0;
                ff->size = d->size;
                ff->attrib = d->attrib;
            }
            else
            {
                ++d;
                ff->index++;
                if(ff->index == 32) //run out of sector
                {
                    if(RamFat[ff->sector] != 0xfe)  //end?
                    {
                        ff->sector = RamFat[ff->sector];
                        d = (DirEntry *)(RamData + ((long)ff->sector << 9));
                        ff->index = 0;
                    }
                    else
                    {
                        loop = 0;
                        retval = RAMDISK_EOF;
                    }
                }
            }
        }
        break;
    case RAMDISK_SEEK:
        //-----------------------------------------
        //new file pointer is passed in count
        //this is the only long availiable
        //-----------------------------------------
        if(fcb)
        {
            if(count > fcb->FileSize)
                retval = RAMDISK_EOF;
            else
            {
                long diff = count - fcb->FilePointer;
                if(diff < 0)
                {   //go backwards (hard)
                    //to go backwards, start at beginning of file and word forwards
                    fcb->CurrentSector = fcb->StartSector;
                    diff = count;
                    while(diff >= 512)
                    {
                        fcb->CurrentSector = RamFat[fcb->CurrentSector];
                        diff -= 512;
                        fcb->FilePointer += 512;
                    }
                    fcb->SectorIndex = diff;
                }
                else
                {   //go forward (easy)
                    while(diff >= 512)
                    {
                        if(RamFat[fcb->CurrentSector] != 0xfe)
                        {
                            fcb->CurrentSector = RamFat[fcb->CurrentSector];
                            fcb->FilePointer += 512;
                            diff -= 512;
                        }
                        else
                            return RAMDISK_EOF;
                    }
```

```
                    if((diff + fcb->SectorIndex) > 512)
                    {
                        diff -= 512 - fcb->SectorIndex;
                        fcb->FilePointer += 512 - fcb->SectorIndex;
                        if(RamFat[fcb->CurrentSector] != 0xfe)
                        {
                            fcb->CurrentSector = RamFat[fcb->CurrentSector];
                            fcb->FilePointer += diff;
                            fcb->SectorIndex = diff;
                        }
                        else
                            return RAMDISK_EOF;
                    }
                    else
                        fcb->SectorIndex += diff;
                }
            }
        }
        else
            retval = RAMDISK_EOF;
        break;
    case RAMDISK_TELL:
        if(fcb)
            retval = fcb->FilePointer;
        else
            retval = RAMDISK_EOF;
        break;
    case RAMDISK_FREE:
        int i;

        for(i=0;i<RAMDISK_MAXSECTORS;++i)
            if(RamFat[i] == 0xff) retval += 512;
        break;
    case RAMDISK_DELETE:
        /*
       ** file name is in buffer
       ** find file, then deallocate sectors
       */
        int sector;

        if(buffer == 0)
        {
            p = iocb->dev_name;
            p = strchr(p,':');
            p++;    //point to begining of name
        }
        else
            p = buffer;
        if((d = DirSearch(p,&sector)) != NULL)
        {
            DeallocateSectors(d->start);
            d->name[0] = '?';       //mark as unoccupied
        }
        else
            retval = RAMDISK_EOF;
        RamDiskDescp->FatChecksum = CalculateFatChecksum();
        Checksums[sector] = CalculateDataChecksum(sector);
        break;
    case RAMDISK_FORMAT:
        Format();
        break;
    case RAMDISK_FILESIZE:
        retval = fcb->FileSize;
        break;
}
return retval;
```

```
}
```

```
/*************************************************************************
**
** Device Driver for Ramdisk for Spindle Controller
**
**
*************************************************************************/

#ifndef RAMDISK__H
#define RAMDISK__H

#include "erorcode.h"

#define RAMDISK_MAXSECTORS  254

struct RamDisk {
    unsigned long magic;        //magic number to indicate formated ramdisk
    unsigned FatChecksum;       //checksum of fat table
    int Directory;              //Start Sector for Directory
    int DirSize;                //Size of Directory
};

struct DirEntry {
    char name[8];               //8 bytes for name
    long size;                  //4 bytes for size
    int start;                  //2 bytes for start sector
    unsigned attrib;            //2 bytes for attributes
};

//-------------------------------------------------------------------------
// DirEntry->attrib bits
//-------------------------------------------------------------------------

#define RAMDISK_READONLY    0x01    //locks file
#define RAMDISK_OPEN        0x02    //indicates file is in use

struct FCB {
    int StartSector;            //first data sector for file
    int CurrentSector;          //current active sector
    int SectorIndex;            //index in current sector where data points
    long FilePointer;           //index in file where data points
    long FileSize;              //total size of file
    unsigned char *buffer;      //pointer to file buffer if needed
    int bufferindex;            //index in file buffer
    DirEntry *Dir;              //pointer to directory entry
    int DirSector;              //sector directory entry is in
};

struct Ffblk {
    char name[10];              //name of file
    long size;                  //size of file
    unsigned attrib;            //file attributes
    char spec[8];               //user is not supposed to know about this
    int sector;                 //current sector
    int index;                  //index in sector
};

#define RAMDISK_DESCRIPTOR  0       //offset for ramdisk descriptor
#define RAMDISK_FAT         256     //offset for ramdisk FAT
#define RAMDISK_CHECKSUMS   512     //offset for ramdisk checksums
#define RAMDISK_DATA        1024    //offset for ramdisk data

/*  XIO function calls      */

#define RAMDISK_FLUSH       0x20    //flush buffers
#define RAMDISK_FINDFIRST   0x21    //find first entry in directory
#define RAMDISK_FINDNEXT    0x22    //find next entry in directory
```

```
#define RAMDISK_SEEK        0x23    //seek position in file
#define RAMDISK_TELL        0x24    //get position in file
#define RAMDISK_FREE        0x25    //free space in ramdisk
#define RAMDISK_DELETE      0x26    //delete file in ramdisk
#define RAMDISK_FORMAT      0x27    //format (initialize) ramdisk
#define RAMDISK_FILESIZE    0x28    //get size of file


//-----------------------------------------------
// various macros
//-----------------------------------------------

//  FindFirst(char *filespec,int attributes,Ffblk *ff);
#define FindFirst(n,a,f)    (int)Xio((RAMDISK_FINDFIRST),(-1),("D:"),((char *)(n)),(0l),(a),(f))
#define FindNext(f)         (int)Xio((RAMDISK_FINDNEXT),(-1),("D:"),((char *)0),(0l),(0),(f))
#define FormatRamdisk()     (int)Xio((RAMDISK_FORMAT),(-1),("D:"),((char *)0l),(0l),(0))
#define FileSize(h)         (int)Xio((RAMDISK_FILESIZE),(h),(char *)0,(char *)0,0l,0);

#ifdef __cplusplus
extern "C" {
#endif

#pragma function( calling)

extern long D_init(void);

#pragma function()

#ifdef __cplusplus
}
#endif

#endif  /* RAMDISK__H   */
```

```
/*************************************************************************
**
**  Teletrac Spindle Controller Board   June 13, 1995
**
**  Copyright (c) 1995 Teletrac Inc.
**
**  This is where we start execution on power up.
**  Initialize all the I/O devices
**  Call the main execution loop
**
*************************************************************************/

#define RS232_BUILD

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curses.h>
#include "cio.h"
#include "task.h"
#include "queue.h"
#include "timer.h"
#include "serlprot.h"
#include "xilinx.h"
#include "quad.h"
#include "spindle.h"
#include "tach.h"
#include "rs232.h"
#include "upload.h"
#include "frontled.h"
#include "spinchip.h"
#include "global.h"
#include "servo.h"
#include "ramdisk.h"
#include "dac.h"
#include "i2c.h"
#include "acutrac.h"

extern "C" void consol(void);
extern "C" void DisableInterrupt(void);
static void StartMultitasking(void);
extern void InitCrash(void);
volatile int GlobalSpindleRPM;
extern int ConsolHandle;

void main()
{
    InitCrash();
        *((volatile int *)0xffcc0000) = 0xffff;
    InitOS();
        *((volatile int *)0xffcc0000) = 0xfffe;
    TBlock();           //prevent task swaping until ready
        *((volatile int *)0xffcc0000) = 0xfffd;
    init_iocb();        /*  initialize CIO  */
        *((volatile int *)0xffcc0000) = 0xfffc;
    R_init();           /*  initialize first RS232 port */
        *((volatile int *)0xffcc0000) = 0xfffb;
    init_timer();       /*  initialize real time clock  */
        *((volatile int *)0xffcc0000) = 0xfffa;
    LoadXilinx(DATAspindle);
        *((volatile int *)0xffcc0000) = 0xfff9;
    InitSpinChip();
        *((volatile int *)0xffcc0000) = 0xfff8;
    DSPupload();
        *((volatile int *)0xffcc0000) = 0xfff7;
    QUAD_init();        /*  enable AQuad BQuad device driver    */
```

```
        *((volatile int *)0xffcc0000) = 0xfff6;
    DAC_init();
        *((volatile int *)0xffcc0000) = 0xfff5;
    SPIN_init();        /*  enable Spindle Device driver    */
        *((volatile int *)0xffcc0000) = 0xfff4;
    TACH_init();        /*  initialize Tachometer device driver */
        *((volatile int *)0xffcc0000) = 0xfff3;
    D_init();
        *((volatile int *)0xffcc0000) = 0xfff2;
    InitFrontPanelLeds();
        *((volatile int *)0xffcc0000) = 0xfff1;
    InitServoPort();
        *((volatile int *)0xffcc0000) = 0xfff0;
    I2CInit();
        *((volatile int *)0xffcc0000) = 0xffff;
    SystemStatus.spin.stat.system = 1;
    SetPortCBits(SPIN_GPC_I2C); //reset front panel
    StartMultitasking();
    while(1);       /*  just stop everything    */
}

SYSTEM sys;     //Io handles

//extern IO_REC TRecs[2];
volatile int LedCount=0;

int mhandle;

static void RunProtocol(void)
{
    PROT_DATA *inbuff;
    char *obuff;
    int ocount;
    int error;
    int logflag;

    InitProtocol();
    RestoreSettings(0);     //restore settings in DSP
    if((inbuff = (PROT_DATA *)malloc(sizeof(PROT_DATA))) == NULL)
    {
    }
    if((obuff = (char *)malloc(256)) == NULL)
    {
    }
    mhandle = Open("R1:",READ_ONLY | WRITE_ONLY);
    Xio(RS232_SET_PUTEOT,mhandle,(char *)0,(char *)0,0l,0x7f);  //set protocol tail character
    Xio(RS232_SET_GETEOT,mhandle,(char *)0,(char *)0,0l,0x7f);  //set protocol tail character
    do
    {
        error = TelReceiveProtocol(mhandle,inbuff); /*  get data    */
        if(error < 0)
        {
        }
        else
        {
            ocount = TelCommandParser(inbuff->bytecount,inbuff->buff,obuff,&sys,&logflag);
            TelSendProtocol(mhandle,inbuff->devID,ocount,obuff,logflag);    /*  send data back t
o host (device Zero)    */
        }
    }while(1);
}

static void TaskRs232(void)
{
    int i=0;
    char ostring[132];
```

```
    int a;
    extern int process_file(int h,char *s);

    Xio(SPIN_SETPLLMODE,sys.HSpindle,(char *)0,(char *)0,0l,1); //ready to PLL
    Xio(RS232_SET_GETEOT,ConsolHandle,(char *)0,(char *)0,0l,-1);   //no eot character
    Xio(RS232_SET_PUTEOT,ConsolHandle,(char *)0,(char *)0,0l,-1);   //no eot character
    I2CWriteLEDString(" Acutrac ][ ");
    I2CBeepTheBuzzer(2,20);
    Putc(ConsolHandle,'\r');
    Putc(ConsolHandle,'\n');
    Putc(ConsolHandle,'>'); //prompt
    while(1)
    {
        a = Getc(ConsolHandle); //get a character
        if(a == '\b')       //is character a backspace?
        {
            if(i > 0)
            {
                --i;            //decrement index by one
                Putc(ConsolHandle,a);
                Putc(ConsolHandle,' '); //erase character
                Putc(ConsolHandle,a);
            }
        }
        else if(a == '\n')      //eol?
        {
            ostring[i] = 0;
            i = 0;
            Putc(ConsolHandle,'\r');
            Putc(ConsolHandle,'\n');
            process_file(ConsolHandle,ostring);
            Putc(ConsolHandle,'>'); //prompt
        }
        else
        {
            Putc(ConsolHandle,a);       //put character to consol
            ostring[i++] = a;
        }
    }
}

static void IdleTask(void)
{
    volatile int i;

    while(1)
        ++i;
}

volatile int DisplayMode=0;
volatile int TrackMode=0;

Wait *WaitForDisplayCycle;
TSemaphore *WaitForAtSpeed;

//*********************************************************************
//
// This is the main task that does things to the front panel display
// on the spinstand
// Speed is more or less regulated by the Tach interrupt.  Getc(sys.HTach)
// will pend until the interrupt occures
//
//*********************************************************************

static void LedTask(void)
{
```

```
    long v;
    int c,d,e;
    char *s = new char[128];
    double t;
    DSP_status stat;
    SPINSTAT tstat;
    int sr;
    extern volatile int AdjustFlag;
    int ModeChange;
    int RpmState=0;

    SetMicroEAutoComp(MICROE_AUTOCOMP_ON);
    WaitForDisplayCycle = new Wait(0,"WaitForDispCyc");
    WaitForAtSpeed = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"WaitForSpeed");
    TDelay(50);
    //---------------------------------------
    // this is a low priority task.  Use it
    // for doing some miscilaneous initializing
    //---------------------------------------
    c = Xio(SPIN_GETENCODER,sys.HSpindle,(char *)0,(char *)0,0l,0); //get encoder pitch
    SetEncoderPitch(2, (long)c);    //tell dsp what the encoder pitch is
    //---------------------------------------
    // start of main task loop
    //---------------------------------------
    while(1)
    {
        if(WaitForDisplayCycle->GetCount() < 0)
            WaitForDisplayCycle->Post(0);
        if(DisplayMode==0)
        {
            if(RpmState)
            {
                if(StageSetRpm(e) != SPINDLE_NAK)
                    RpmState = 0;
            }
            ModeChange=1;
            GlobalSpindleRPM = Getc(sys.HTach);     //get tach reading
            if(AdjustFlag)      //display position
            {
                Read_Position(RSTAGE,&v);       //get current position
                if(TrackMode == 0)
                {
                    sprintf(s,"P=%10ld",v);
                }
                else if(TrackMode == 1)
                {
                    t = rad_to_track(angle_to_rad(DeviceTable->referenceangle - icts_to_angle(v
* DeviceTable->dutorientation)));
                    sprintf(s,"T=%10.3lf",t);
                }
            }
            else                //display RPM
            {
                c = sprintf(s,"RPM=%8d",GlobalSpindleRPM);
            }
            I2CWriteLEDString(s);
        }
        else if(DisplayMode==1)     //Set RPM
        {
            if(!RpmState)
            {
                e = Getc(sys.HSpindle);
                RpmState = 1;
            }
            else if(RpmState == 1)
            {
```

```
            if((d = Getc(sys.HQuad)) != 0)
            {
                if((e += d) < 0) e = 0;
                c = sprintf(s,"RPM:%8d",e);
                if(DisplayMode == 1)
                {
                    StageSetRpm(e);
                }
                I2CWriteLEDString(s);
            }
            else if(ModeChange)
            {
                c = sprintf(s,"RPM:%8d",e);
                I2CWriteLEDString(s);
                ModeChange = 0;
            }
        }
    }
    else if(DisplayMode==2)     //Set Position
    {
        //-------------------------------------------------
        // Would like to display TRACKS, will default to
        // Counts, will changed to TRACKS when config file
        // is loaded
        //-------------------------------------------------
        if(RpmState)
        {
            if(StageSetRpm(e) != SPINDLE_NAK)
                RpmState = 0;
        }
        if(TrackMode)   //display in tracks
        {
            AcutracGETTRACK(&t);    //get current track
            if((d = Getc(sys.HQuad)) != 0)
            {
                t += (double)d / 10.0;
                sprintf(s,"T:%10.3lf",t);
                I2CWriteLEDString(s);
                AcutracSEEKTRACK(t);
            }
            else if (ModeChange)
            {
                sprintf(s,"T:%10.3lf",t);
                I2CWriteLEDString(s);
                ModeChange = 0;
            }
        }
        else            //display in counts
        {
            Read_Goal(RSTAGE,&v);   //get goal position
            if((d = Getc(sys.HQuad)) != 0)
            {
                v += d;
                c = sprintf(s,"P:%10ld",v); //display current goal
                I2CWriteLEDString(s);
                Set_Goal(RSTAGE,v);
                Do_move(RSTAGE,MOVE_ABSOLUTE);
            }
            else if(ModeChange)
            {
                c = sprintf(s,"P:%10ld",v); //display current goal
                I2CWriteLEDString(s);
                ModeChange = 0;
            }
        }
    }
```

```
        else
        {
            GlobalSpindleRPM = Getc(sys.HTach);     //get tach reading
        }
        c = Getc(sys.HSpindle); //get set RPM
        if(abs(GlobalSpindleRPM - c) < 2)       //is it within 1 RPM?
        {
            if(WaitForAtSpeed->GetCount() < 0)
                WaitForAtSpeed->Post(0);
            sr = EnterCritical();
            SystemStatus.spin.stat.atspeed = 1;
            ExitCritical(sr);
        }
        else
        {
            sr = EnterCritical();
            SystemStatus.spin.stat.atspeed = 0;
            ExitCritical(sr);
        }
        //-----------------------------------------------------------------
        // Do other system related overhead
        // Like, Get Servo Status
        //-----------------------------------------------------------------
        if(ServoStatus(XSTAGE,&stat) == 0)  //is read status ok?
        {
            //-----------------------------
            //this call gets system data for both stages
            //-----------------------------
            tstat.servo1.stat.system = 1;
            tstat.servo1.stat.servo = (stat.system[XSTAGE] & SERVO_ENABLED)?1:0;
            tstat.servo1.stat.integrator = (stat.system[XSTAGE] & INTEGRATOR_ENABLED)?1:0;
            tstat.servo1.stat.beam = (stat.system[XSTAGE] & BEAM_INTERRUPTED)?1:0;
            tstat.servo1.stat.limit_p = (stat.system[XSTAGE] & LIMIT_PLUS_TRIPPED)?1:0;
            tstat.servo1.stat.limit_m = (stat.system[XSTAGE] & LIMIT_MINUS_TRIPPED)?1:0;
            tstat.servo1.stat.follow = (stat.system[XSTAGE] & FOLLOWING_ERROR)?1:0;
            tstat.servo1.stat.bumped = (stat.system[XSTAGE] & BUMP_ERROR)?1:0;
            tstat.servo1.stat.protect = (stat.system[XSTAGE] & PROTECTION_TRIPPED)?1:0;
            tstat.servo1.stat.inposition = (stat.system[XSTAGE] & SERVO_IN_POSITION)?1:0;

            tstat.servo2.stat.system = 1;
            tstat.servo2.stat.servo = (stat.system[RSTAGE] & SERVO_ENABLED)?1:0;
            tstat.servo2.stat.integrator = (stat.system[RSTAGE] & INTEGRATOR_ENABLED)?1:0;
            tstat.servo2.stat.beam = (stat.system[RSTAGE] & BEAM_INTERRUPTED)?1:0;
            tstat.servo2.stat.limit_p = (stat.system[RSTAGE] & LIMIT_PLUS_TRIPPED)?1:0;
            tstat.servo2.stat.limit_m = (stat.system[RSTAGE] & LIMIT_MINUS_TRIPPED)?1:0;
            tstat.servo2.stat.follow = (stat.system[RSTAGE] & FOLLOWING_ERROR)?1:0;
            tstat.servo2.stat.bumped = (stat.system[RSTAGE] & BUMP_ERROR)?1:0;
            tstat.servo2.stat.protect = (stat.system[RSTAGE] & PROTECTION_TRIPPED)?1:0;
            tstat.servo2.stat.inposition = (stat.system[RSTAGE] & SERVO_IN_POSITION)?1:0;

            tstat.servo3.stat.system = 1;
            tstat.servo3.stat.servo = (stat.system[SSTAGE] & SERVO_ENABLED)?1:0;
            tstat.servo3.stat.integrator = (stat.system[SSTAGE] & INTEGRATOR_ENABLED)?1:0;
            tstat.servo3.stat.beam = (stat.system[SSTAGE] & BEAM_INTERRUPTED)?1:0;
            tstat.servo3.stat.limit_p = (stat.system[SSTAGE] & LIMIT_PLUS_TRIPPED)?1:0;
            tstat.servo3.stat.limit_m = (stat.system[SSTAGE] & LIMIT_MINUS_TRIPPED)?1:0;
            tstat.servo3.stat.follow = (stat.system[SSTAGE] & FOLLOWING_ERROR)?1:0;
            tstat.servo3.stat.bumped = (stat.system[SSTAGE] & BUMP_ERROR)?1:0;
            tstat.servo3.stat.protect = (stat.system[SSTAGE] & PROTECTION_TRIPPED)?1:0;
            tstat.servo3.stat.inposition = (stat.system[SSTAGE] & SERVO_IN_POSITION)?1:0;

            sr = EnterCritical();
            SystemStatus.servo1.v = tstat.servo1.v;
            SystemStatus.servo2.v = tstat.servo2.v;
            SystemStatus.servo3.v = tstat.servo3.v;
            ExitCritical(sr);
```

```
        }
        else
        {
            //-------------------------------
            // indicate some sort of problem
            // with stage controller
            //-------------------------------
            sr = EnterCritical();
            SystemStatus.servo1.stat.system = 0;
            SystemStatus.servo2.stat.system = 0;
            SystemStatus.servo3.stat.system = 0;
            ExitCritical(sr);
        }
    }
}

TSTATS *TaskStatistics;

static void TaskStats(void)
{
    /*
    ** This task is used to get task statistics, such as execution
    ** times
    */

}

static void StartMultitasking(void)
{
    /*
    ** this is it, the REAL test
    **
    ** Just do the same thing that was done in the original test
    */
    TCB *t;
extern void InitAccutrac(void);
    /*
    ** create TASKS
    */
    t = CreateTask(RunProtocol,2048,13,"Host_Interface");
    ActiveTasks->Insert(t);

    t = CreateTask(TaskRs232,2048,7,"RS232");
    ActiveTasks->Insert(t);

    t = CreateTask(LedTask,2048,10,"LedTask");
    ActiveTasks->Insert(t);

    t = CreateTask(IdleTask,2048,1,"IdleTask");
    ActiveTasks->Insert(t);
    InitSaveRestore();      //initialize save/restore DSP tasks
    InitAccutrac();
    //
    // Open up channels to standard devices
    //

    ConsolHandle = Open("R2:",(READ_ONLY | WRITE_ONLY));
    sys.HSpindle = Open("SPIN:",READ_ONLY | WRITE_ONLY);    //spindle controller
    sys.HQuad = Open("QUAD:",READ_ONLY);                    //quadrature encoder know
    sys.HTach = Open("TACH:",READ_ONLY);                    //spindle tachometer

    Putc(sys.HSpindle,5400);    //set default RPM=5400
    DisableInterrupt();
    TRelease();     //release scheduler
    Start();        //start multitasker
}
```

```
/*************************************************************************
**                                                                      **
** Acutrac.cpp                                                          **
**                                                                      **
** Author:Jim Patchell                                                  **
** Date:8-29-96                                                         **
**                                                                      **
** This file is where the processing takes place for doing the spin stand **
** thing.                                                               **
**                                                                      **
** Functions include:                                                   **
**      int AcutracINIT(char * name);                                   **
**      int AcutracSAVEascii(char * name);                              **
**      int AcutracRESET(void);                                         **
**      int AcutracSTART(void);                                         **
**      int AcutracSTOP(void);                                          **
**      int AcutracGETRPM(int *rpm);                                    **
**      int AcutracGETMAXRPM(int *maxrpm);                              **
**      int AcutracGETMINRPM(int *minrpm);                              **
**      int AcutracSTARTSPINDLE(void);                                  **
**      int AcutracSTOPSPINDLE(void);                                   **
**      int AcutracSETROTDIR(int dir);                                  **
**      int AcutracGETROTDIR(int *dir);                                 **
**      int AcutracSETLOADRPM(int rpm);                                 **
**      int AcutracGETLOADRPM(int *rpm);                                **
**      int AcutracSEEKTRACK(double track);                             **
**      int AcutracSTEPIN(void);                                        **
**      int AcutracSTEPOUT(void);                                       **
**      int AcutracGETTRACK(double *track);                             **
**      int AcutracSETMAXTRACK(int maxtrack);                           **
**      int AcutracGETMAXTRACK(int *maxtrack);                          **
**      int AcutracSETRADIUS(char *radius);                             **
**      int AcutracSETTRACKSIZE(char *tracksize);                       **
**      int AcutracGETTRACKSIZE(char *tracksize);                       **
**      int AcutracGETERRORMESSAGE(char *message);                      **
**      int AcutracSETCORRECTION(char *c);                              **
**      int AcutracGETCORRECTION(char *c);                              **
**      int AcutracGETPIDPARAMSTYPE(void);                              **
**      int AcutracSELECTPARAMSTYPE(int type);                          **
**                                                                      **
** We will be recieving floating point data from an unknown quantity, so **
** floating point will arive in ascii format.                           **
**                                                                      **
** All functions return 0 when there is no error, non-zero for error    **
**                                                                      **
** All Acutrac functions will return acutrac ERRORS.  If some other     **
** function is called, it must have it's error code translated into     **
** acutrac error codes.  This is done with ChkErr function.  Just pass  **
** the return code to this function and you will get Acutrac Error codes **
**                                                                      **
**                                                                      **
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "cio.h"            /*  I/O manager */
#include "task.h"           /*  Multi Tasking Kernel    */
#include "ramdisk.h"        /*  ramdisk Device Driver   */
#include "acutrac.h"        /*  High Level Spinstand Functions  */
#include "scfig.h"          /*  Parser for AFG file */
#include "servo.h"          /*  Low Level Motion Control Functions  */
#include "global.h"         /*  Global Variables    */
#include "i2c.h"            /*  I2C spinstand access functions  */
#include "spindle.h"        /*  Device Driver for spindle motor */
```

```
#include "spinchip.h"       /*  Defines for spindle controller chip access  */
#include "strings.h"        /*  string class    */

//#include "serlprot.h"

/*
** Global variables for this module
*/

// there will be only one device table since we can handle only one at a time

extern volatile int TrackMode;  //flag..when Trackmode==1,track callibration has been set

AcutracDeviceTable *DeviceTable=(AcutracDeviceTable *)0;    //pointer to device parameter table
static char AcutracErrorString[128];    //place to write error messages
static char AcutracStateString[64];     //place to write state string messages

static TCB *ActionTask=0;   //pointer to task that gets invoked for RESET,START,STOP
static TCB *Abort=0;        //abort task, does all the things that need to
                            //be done when the panic button is pushed on
                            //the spinstand front panel
static volatile ACTION_STATUS ActionStatus; //indicates status of Reset Task,Start Task,Stop Tas
k
static volatile int Calibrated; //flag indicates if rotary is callibrated

static void HSAStart(void); //loads heads in HSA mode
static void HSAStop(void);  //unloads heads in HSA mode
static void HSALift(void);  //Lifts heads off of disk in HSA mode
static void HSAReload(void);    //Lowers heads back on disk in HSA mode
static void HSAReset(void); //Initialize spinstand for HSA mode

static void HGAReset(void); //initialize spinstand for HGA mode
static void HGAStart(void); //loads heads in HGA mode
static void HGAStop(void);  //unload heads in HGA mode
static void HGALift(void);  //Lifts heads off of disk in HGA mode
static void HGAReload(void);    //Lowers heads back on disk in HGA mode

static Wait *StringExclude; //Semaphore to control access to AcutracErrosString and AcutracState
String

extern volatile int DisplayMode;    //flag to set Spinstand Display Mode
extern int ConsolHandle;            //I/O handle for Debug RS232 port
extern Wait *WaitForDisplayCycle;   //Semaphore to indicate when DisplayMode has changed
extern TSemaphore *WaitForAtSpeed;  //Semaphore to indicate when spindle motor is at speed
int PidParams::currenttype;         //Flag:Tooling on Chuck/off Chuck, selects tuning paramter s
et


//---------------------------------------
// one place depository of various strings
//---------------------------------------
static const char * const EStrings[] = {
    "Everything A-O-K",             //the first 13 strings are errors
    "General Accutrac Error",       //-1
    "Spinstand Busy",               //-2
    "Bad Config File",              //-3
    "Motion Controller NAK error",  //-4
    "Motion Controller Timeout",    //-5
    "Motion Controller Unknown",    //-6
    "Out of Range",                 //-7
    "File Error",                   //-8
    "File Bad",                     //-9
    "Ramp Timeout",                 //-10
    "Linear Timeout",               //-11
    "Rotary Actuator Timeout",      //-12
    "Spindle Motor Timeout",        //-13
```

```
    "User Pushed Abort Button",     //-14
    "Semaphore Timeout",            //-15
    "Terminate Task",               //-16
    "Error Not Handled",            //-17
    "Air Pressure Low",             //-18
    "No Init File Loaded",          //-19
    "Spindle Amplifier Fault",      //-20
    "Connector Clamp Timeout"       //-21
};

//-----------------------------------------------------------------
//StateStrings is used to describe the various states that
//the spinstand functions START,STOP,LIFT,RELOAD,and RESET are in
//These strings are copied, with other information into
//AccutracStateString which can be read back by the host computer
//-----------------------------------------------------------------
static const char * const StateStrings[] = {
    "Activate Vacumm Chuck",
    "Install Device Under Test",
    "Wait for Vacuum Sensor",
    "Push the Continue Button",
    "Move Rotoary Stage Load/Unload",
    "Wait For In Position",
    "Move in Ramp Loader",
    "Wait for Ramp Loader",
    "Remove Comb",
    "Move X Stage to Run Position",
    "Start Spindle Motor",
    "Wait for Run Position",
    "Wait for Spindle At Speed",
    "Load Heads onto Disk",
    "Set Spindle Speed to Test RPM",
    "Move Head to Initial Track",
    "Move Rotary Stage to Unload Pos",
    "Set Spindle Speed to Unload RPM",
    "Stop Spindle Motor",
    "Move Linear Stage Home",
    "Wait for Linear Stage Home",
    "Install Comb",
    "Move Ramp Loader out",
    "Release Vacuum Chuck",
    "Remove Device Under Test",
    "Setup Linear Stage",
    "Wait for Linear Stage to move",
    "Calibrate Rotary Stage",
    "Build Track Lookup Table",
    "Check Condition of Stage",
    "               ",
    "Init Stage     ",
    "Load Heads     ",
    "Unload Heads   ",
    "Exersize Stage ",
    "Lift Heads     ",
    "Reload Heads   ",
    "Lifters Up",
    "Lifters Down",
    "Connector Clamp"
};

//-----------------------------------------------------------------
// Constant values that are used to index into the StateStrings Array
// These are only for readability of the code
//-----------------------------------------------------------------
const enum {
    LSS_ACTIVATEVACCHUCK,
    LSS_INSTALLDUT,
```

```
    LSS_WAITFORVACSENSE,
    LSS_PUSHCONTINUEBUTTON,
    LSS_MOVEROTSTAGELOADUNLOAD,
    LSS_WAITFORINPOSITION,
    LSS_MOVEINRAMPLOADER,
    LSS_WAITFORRAMPLOADER,
    LSS_REMOVECOMB,
    LSS_MOVEXSTAGETORUN,
    LSS_STARTSPINDLE,
    LSS_WAITFORRUNPOSITION,
    LSS_WAITFORSPINDLEATSPEED,
    LSS_LOADHEADSONTODISK,
    LSS_SETSPINDLETOTESTRPM,
    LSS_MOVEHEADTOINITTRACK,
    LSS_MOVEHEADTOUNLOADPOS,
    LSS_SETSPINDLETOUNLOADRPM,
    LSS_STOPSPINDLE,
    LSS_MOVEXSTAGEHOME,
    LSS_WAITXSTAGEHOME,
    LSS_INSTALLCOMB,
    LSS_MOVERAMPLOADEROUT,
    LSS_RELEASEVACCHUCK,
    LSS_REMOVEDUT,
    LSS_SETUPLINEARSTAGE,
    LSS_WAITFORLINSTAGETOMOVE,
    LSS_CALIBRATEROTARYSTAGE,
    LSS_BUILDTRACKLUT,
    LSS_CHECKCONDITION,
    LSS_BLANK15,
    LSS_INITSTAGE,
    LSS_LOADHEADS,
    LSS_UNLOADHEADS,
    LSS_EXERSIZE,
    LSS_LIFTHEADS,
    LSS_RELOADHEADS,
    LSS_LIFTERSUP,
    LSS_LIFTERSDOWN,
    LSS_CONNECTORCLAMP
};

//---------------------------------------------------------------------
// Message strings to display of the front panel of the spinstand
// Each string must be exactly 12 characters long
// It should be noted that the display on the spinstand is no longer
// an LCD, but an LED display
//---------------------------------------------------------------------

static const char * const LCDStrings[] = {
    "            ",     //blank
    "Insert D.U.T",
    "Press Contin",
    "Remove Comb ",
    "Replace Comb",
    "Remove D.U.T",
    "CannotUnload",
    "When Ready  ",
    "ERROR: SERVO",
    "INITIALIZING",
    "Lift Heads  ",
    "Lower Heads ",
    "Load Heads  ",
    "Unload Heads",
    "Calibrating."
};

//---------------------------------------------------------------------
```

```
// Constants for indexing into LCDStrings
//---------------------------------------------------------------
const enum {
    LCDS_BLANK,
    LCDS_INSERTDUT,
    LCDS_CONTINUE,
    LCDS_REMOVECOMB,
    LCDS_REPLACECOMB,
    LCDS_REMOVEDUT,
    LCDS_CANNOTUNLOAD,
    LCDS_WHENREADY,
    LCDS_ERRORSERVO,
    LCDS_INITIALIZING,
    LCDS_LIFTHEADS,
    LCDS_LOWERHEADS,
    LCDS_LOADHEADS,
    LCDS_UNLOADHEADS,
    LCDS_CALIBRATING
};



//-------------------------------------------------------------------------
// Misc Local functions
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// This is the constructor for the ACTION_STATUS structure
// It initializes all of the data members to Zero
//-------------------------------------------------------------------------

ACTION_STATUS::ACTION_STATUS()
{
    Function=0;
    Status=0;
    State=0;
    StartFlag=0;
}

//-------------------------------------------------------------------------
// implementation of AcutracDeviceTable Class;
//-------------------------------------------------------------------------

AcutracDeviceTable::AcutracDeviceTable()    //Constructor
{
    memset(this,0,sizeof(AcutracDeviceTable));  //initialize all of the data to zero
    Pid[ACUTRAC_HEADSON] = new PidParams(ROTARY_STAGE);     //create a PidParams for tooling on
chuck
    Pid[ACUTRAC_HEADSOFF] = new PidParams(ROTARY_STAGE);    //create a PidParams for tooling off
 chuck
}

AcutracDeviceTable::~AcutracDeviceTable()   //destructor (never used)
{
    delete Pid[ACUTRAC_HEADSON];    //delete PidParams memory
    delete Pid[ACUTRAC_HEADSOFF];
}

//-------------------------------------------------------------------------
// implementation of PidParams class
//
// The purpose of this class is to make it easy to switch the tuning
// parameters.  When the tooling is on the chuck, the rotary actuator
// may require differnt tuning parameters as opposed when the tooling
// is not on the chuck.  This class encapsulates this functionality.
//-------------------------------------------------------------------------
```

```
    //*************************************************
    // These functions is used to remap the Notch Filter
    // Servo Functions (see servo.h) into a function
    // that is more like the others.  The notch filter
    // functions have an extra paramter to indicate
    // which of the two filters is being operated
    // on.  Most of the servo function accept only
    // two parameters, and axis number and a value
    // By creating a new function, all of the servo
    // functions can be put into an array and accessed
    // the same way.
    //*************************************************

static int SetNotchFreq1(int a,long v)
{
    //*************************************************
    //
    // Set the notch filter frequency
    //
    // Parameter
    //  a.......Axis number
    //  v.......Value corresponding to filter frequency
    //
    // filter number is set to access filter
    // number 1 ( fn = 0)
    //
    // Return Value: Status of operation.
    //             : 0 :=> Operation success
    //             : Negative:=> Operation Failure
    //*************************************************

    return SetNotchFreq(a,v,0);
}

static int SetNotchQ1(int a,long v)
{
    //*************************************************
    //
    // Set the Notch Filter Q
    //
    // Parameter
    //  a.......Axis number
    //  v.......Value corresponding to filter Q (resonance)
    //
    // filter number is set to access filter
    // number 1 ( fn = 0)
    //
    // Return Value: Status of operation.
    //             : 0 :=> Operation success
    //             : Negative:=> Operation Failure
    //*************************************************

    return SetNotchQ(a,v,0);
}

static int SetNotchDepth1(int a,long v)
{
    //*************************************************
    //
    // Set the Notch Filter Depth
    //
    // Parameter
    //  a.......Axis number
    //  v.......Value corresponding to filter Notch Depth
    //          Value = 0 is infinate deap Notch
```

```
    //          Value = (2^23)-1 is flat response
    //
    // filter number is set to access filter
    // number 1 ( fn = 0)
    //
    // Return Value: Status of operation.
    //             : 0 :=> Operation success
    //             : Negative:=> Operation Failure
    //*************************************************

    return SetNotchDepth(a,v,0);
}

//*************************************************
// Lookup table for functions to upload PidParams to
// motion control processor
//*************************************************

static const int (* const uploadfunc[6])(int a,long v) = {
    Set_Kp,
    Set_Kv,
    Set_Ki,
    SetNotchFreq1,
    SetNotchQ1,
    SetNotchDepth1
};

static int ReadNotchFreq1(int a,long *v)
{
    //*************************************************
    //
    // Read the notch filter frequency
    //
    // Parameter
    //  a.......Axis number
    //  v.......Pointer To a Value corresponding to filter frequency
    //
    // filter number is set to access filter
    // number 1 ( fn = 0)
    //
    // Return Value: Status of operation.
    //             : 0 :=> Operation success
    //             : Negative:=> Operation Failure
    //*************************************************

    return ReadNotchFreq(a,v,0);
}

static int ReadNotchQ1(int a, long *v)
{
    //*************************************************
    //
    // Read the Notch Filter Q
    //
    // Parameter
    //  a.......Axis number
    //  v.......Pointer To a Value corresponding to filter Q (resonance)
    //
    // filter number is set to access filter
    // number 1 ( fn = 0)
    //
    // Return Value: Status of operation.
    //             : 0 :=> Operation success
    //             : Negative:=> Operation Failure
    //*************************************************
```

```
    return ReadNotchQ(a,v,0);
}

static int ReadNotchDepth1(int a, long *v)
{
    //*****************************************************
    //
    // Read the Notch Filter Depth
    //
    // Parameter
    //  a.......Axis number
    //  v.......Pointer to a Value corresponding to filter Notch Depth
    //          Value = 0 is infinate deap Notch
    //          Value = (2^23)-1 is flat response
    //
    // filter number is set to access filter
    // number 1 ( fn = 0)
    //
    // Return Value: Status of operation.
    //             : 0 :=> Operation success
    //             : Negative:=> Operation Failure
    //*****************************************************

    return ReadNotchDepth(a,v,0);
}

//*******************************************************
// Lookup table of functions to read the PidParams from
// the motion control processor
//*******************************************************

static const int (* const downloadfunc[6])(int a,long *v) = {
    Read_Kp,
    Read_Kv,
    Read_Ki,
    ReadNotchFreq1,
    ReadNotchQ1,
    ReadNotchDepth1
};


PidParams::PidParams(int c)     //constructor, needs axis information
{
    //*************************************
    // Create a PidParams Class
    //
    // Parameter:
    // c.......Motion Control Axis
    //*************************************
    axis = c;   //set axis number
    Kp = 0;     //initialize all parameters to zero
    Kv = 0;
    Ki = 0;
    NotchF = 0;
    NotchQ = 0;
    NotchD = 0;
    NotchEnableFlag = 0;
    currenttype = -1;   //Set the current type to unknown (negative)
                        //This variable is static, so all instances
                        // of this class will be reading back the
                        // same value.
}

PidParams::~PidParams() //destructor
{
}
```

```
int PidParams::Upload(int type)     // sends data to the DSP controller
{
    /*************************************************************
    ** Upload paramters to Motion Controller
    **
    ** Paramter:
    ** type......Used to set currenttype.  currenttype is a static
    **           variable and is used to keep track of which set
    **           of PIDParams are actually inside of the motion
    **           controller.
    **           It identifies the data as corresponding either to
    **           Heads on Chuck, or Heads off chuck
    *************************************************************/
    int retval; //value to return when we are all done.
    int i,loop; //loop variables
    long *v;    //pointer to parameters to upload

    for(i=0,loop=1,v = &Kp;(i<6) && loop;++i)
    {
        //*****************************************
        // this loop uploads the data.  The data in
        // the class that stores the paramters is in
        // the same order as the upload function
        // look up table.  Care must be taken not to
        // disturb this order when the code is modified
        //*****************************************
        if((retval = (*uploadfunc[i])(axis,*v++)) < 0)
            loop = 0;   //break out of loop if any of these fail
    }
    if(retval == 0) //if there was total success, change the notch filter
                    //state according to NotchEnableFlag
    {
        retval = EnableServoFunctions(axis,ENABLE_NOTCH,NotchEnableFlag);
    }
    currenttype = type; //set the current type
    return retval;      //and return
}

int PidParams::Download(int type)     // gets data from DSP controller
{
    /*************************************************************
    ** Download paramters from Motion Controller
    **
    ** Paramter:
    ** type......Used to set currenttype.  currenttype is a static
    **           variable and is used to keep track of which set
    **           of PIDParams are actually inside of the motion
    **           controller.
    **           It identifies the data as corresponding either to
    **           Heads on Chuck, or Heads off chuck
    *************************************************************/
    int retval; //return value when done
    int i,loop; //loop parameters
    long *v;    //pointer to place to put data
    DSP_status stat;    //DSP Status Structure, used to determ notch status

    for(i=0,loop=1,v = &Kp;(i<6) && loop;++i)
    {
        //*****************************************
        // this loop downloads the data.  The data in
        // the class that stores the paramters is in
        // the same order as the download function
        // look up table.  Care must be taken not to
        // disturb this order when the code is modified
        //*****************************************
```

```
        if((retval = (*downloadfunc[i])(axis,v++)) < 0)
            loop = 0;   //terminate loop if there is an error
    }
    if(retval == 0) //if there was total success
    {
        //read back the current servo status
        if((retval = ServoStatus(axis,&stat)) == 0)
        {
            //Set the NotchEnableFlag according to the status
            NotchEnableFlag = (stat.system[axis] & DSPSTAT_NOTCHFILTER)?1:0;
        }
    }
    currenttype = type; //set current type to type
    return retval;      // and then return
}

void PidParams::Print(void)
{
    /*****************************************************************
    ** This is a DEBUG function
    ** This function will print out the contents of the PidParams
    ** data to the DEBUG consol (generally, this will be COM2 on
    ** the spindle controller).
    **
    ** Parmeters:
    ** <none>
    ** Return Value:
    ** <none>
    *****************************************************************/
    String *s = new String; //place to store string to be printed
    int c,i;
    long *lp;   //pointer to data

    c = sprintf(s->Get(),"Pid Params\r\n");
    Write(ConsolHandle,s->Get(),c);
    for(i=0,lp=&Kp;i<7;++i)
    {
        //*****************************************
        // This loop prints out the data
        // it should be noted that there are no
        // lables, only an index.  User will have to
        // know which order the data is in to make
        // sense of the display.
        //*****************************************
        c = sprintf(s->Get(),"%d:%ld\r\n",i,*lp++);
        Write(ConsolHandle,s->Get(),c);
    }
    delete s;   //return string to heep
}

int PidParams::GetCurrentType(void)
{
    //*************************************
    // Returns which type of data is contained
    // in the class (Heads On Tooling, or
    // Heads off Tooling).
    //
    // currenttype is a static variable
    // it keeps track of which set of data is
    // actually inside of the motion controller
    //*************************************
    return currenttype;
}

/*****************************************************************************
**
```

```
** The following functions provide access to the PidParams data without
** having to get too involved with using the various classes
**
*************************************************************************/

int AcutracGETPIDPARAMSTYPE(void)
{
    //*********************************************************
    // Get the current type of PidParams that are inside of
    // the motion controller
    //
    // Parmaters:
    // <none>
    // ReturnValue
    //  returns the current value PidParams::currenttype
    //*********************************************************
    int retval;

    if(DeviceTable)retval = DeviceTable->Pid[0]->GetCurrentType();
    else retval = ACUTRAC_NOINITINFO;
    return retval;
}

int AcutracSELECTPARAMSTYPE(int type)
{
    //*************************************************************
    // This function is used to select which set of data is going
    // to be used by the motion controller for the tuning parameters
    // It does this by uploading the appropriate set of data to
    // the motion controller.
    //
    // Parameter:
    // type......selector for the set of data to upload
    // Return Value
    // 0 on success, negative on failure
    //*************************************************************
    int retval;
    if(DeviceTable)
        retval = DeviceTable->Pid[type]->Upload(type);
    else
        retval = ACUTRAC_NOINITINFO;
    return retval;
}

int AcutracSETPIDPARAMS(int type,long *pidparams)
{
    /*************************************************************
    ** This function is used to update the PID paramters and then
    ** upload them to the motion controller
    **
    ** Parameter:
    ** type.......Either Heads On Chuck or Heads off Chuck parameter set
    ** pidparams..pointer to a long array that contains the PID parameters
    **            It should be noted that the PID params are assumed
    **            to be in the same order in the array as they are
    **            in the PidParams class
    ** Return Value:
    **              Status of operation, 0 = Success, Negative = Failure
    *************************************************************/
    int retval;
    if(DeviceTable)
    {
        DeviceTable->Pid[type]->Kp = *pidparams++;
        DeviceTable->Pid[type]->Kv = *pidparams++;
        DeviceTable->Pid[type]->Ki = *pidparams++;
        DeviceTable->Pid[type]->NotchF = *pidparams++;
```

```
        DeviceTable->Pid[type]->NotchQ = *pidparams++;
        DeviceTable->Pid[type]->NotchD = *pidparams++;
        DeviceTable->Pid[type]->NotchEnableFlag = *pidparams++;
        retval = DeviceTable->Pid[type]->Upload(type);
    }
    else
        retval = ACUTRAC_NOINITINFO;
    return retval;
}

int AcutracGETPIDPARAMS(int type,long *pidparams)
{
    /*************************************************************
    ** This function downloads the PID parameters from the motion
    ** controller into a PidParams class, and them copies them to a
    ** long array to be sent else where.
    **
    ** Parameter:
    ** type........Either Heads On Chuck, or Heads off Chuck parameter set
    ** pidparams...Pointer to long array to store data into.  It should
    **             be noted that the order of the data going into
    **             This array will be the same as in the PidParams class
    ** Return Value:
    ** 0 = Success, Negative indicates failure.
    *************************************************************/
    int retval;
    if(DeviceTable)
    {
        if((retval = DeviceTable->Pid[type]->Download(type)) == 0)
        {
            *pidparams++=DeviceTable->Pid[type]->Kp;
            *pidparams++=DeviceTable->Pid[type]->Kv;
            *pidparams++=DeviceTable->Pid[type]->Ki;
            *pidparams++=DeviceTable->Pid[type]->NotchF;
            *pidparams++=DeviceTable->Pid[type]->NotchQ;
            *pidparams++=DeviceTable->Pid[type]->NotchD;
            *pidparams=DeviceTable->Pid[type]->NotchEnableFlag;
        }
    }
    else
        retval = ACUTRAC_NOINITINFO;
    return retval;
}


//---------------------------------------------------------------

static int ChkErr(int servoerror)
{
    //-------------------------------------------------------------
    // translate all posible errors that a servo call or any thing else
    // can return
    /*************************************************************
    ** Ok, I admit, this might be kind of silly...but, this dates back
    ** to the early days of the Guzik Driver, and it is difficult to
    ** change.
    **
    ** Parameter:
    ** servoerror.....error number to translate
    ** Return Value:
    ** Translated Error number.  If the error number is not translated
    ** it is translated to ACUTRAC_NOTHANDLED...which means it was
    ** translated after all...:-)
    *************************************************************/
    //-------------------------------------------------------------
```

```
    int retval;

    switch(servoerror)
    {
        case 0:     //all functions return 0 on success
            retval = ACUTRAC_SUCCESS;
            break;
        case 1:     //and some functions can return either 0 or 1
            retval = 1;
            break;
        case ACUTRAC_ERROR:     //you get this when there is some error
        case ACUTRAC_BUSYERROR:     //you get this when spinstand doing something else
        case ACUTRAC_CONFIGFILE:        //you get this when something wrong with config file
        case ACUTRAC_SERVO_NAK:     //you get this when DSP doesn't want to do what you tell it
        case ACUTRAC_SERVO_TIMEOUT:     //you get this when comm to DSP fails
        case ACUTRAC_SERVO_UNKNOWN:     //you get this when it don't know
        case ACUTRAC_OUTOFRANGE:        //you get this when track is out of range
        case ACUTRAC_FILEERROR:     //you get this when RESET has problem with AFG file
        case ACUTRAC_FILEBAD:       //you get this when RESET can't find AFG file
        case ACUTRAC_RAMPTIMEOUT:       //you get this when ramp lifters don't move
        case ACUTRAC_LINEARTIMEOUT:     //you get this when X stage not moving
        case ACUTRAC_MOTIONTIMEOUT:     //you get this when rotary not moving
        case ACUTRAC_SPINDLETIMEOUT:        //you get this when motor not up to speed
        case ACUTRAC_USERABORT:     //you get this when stop button pushed
        case ACUTRAC_TIMEOUT:       //you get this when a semaphore times out
        case ACUTRAC_TERMINATE:     //you get this when task needs to terminate
            retval = servoerror;    //these should not get here, just don't process them
            break;
        //servo errors
        case SERVO_GOTTIMEOUT:
        case SERVO_GOTACKTIMEOUT:
            retval = ACUTRAC_SERVO_TIMEOUT;
            break;
        case SERVO_GOTNAK:
            retval = ACUTRAC_SERVO_NAK;
            break;
        case SERVO_UNKOWN_ERROR:
            retval = ACUTRAC_SERVO_UNKNOWN;
            break;
        //event errors
        case EVENT_TIMEOUT:         //event has timed out
            retval = ACUTRAC_TIMEOUT;
            break;
        case EVENT_OVERFLOW:        //too many events pending
            retval = ACUTRAC_NOTHANDLED;
            break;
        case EVENT_BUFFFULL:        //pipeline buffer full
            retval = ACUTRAC_NOTHANDLED;
            break;
        case EVENT_NOTASKS:         //no tasks to reschedule to
            retval = ACUTRAC_NOTHANDLED;
            break;
        case EVENT_TERMINATE:       //task needs to be canceled
            retval = ACUTRAC_TERMINATE;
            break;
        //what ever else might happen
        default:
            retval = ACUTRAC_NOTHANDLED;
            break;
    }
    return retval;
}

long angle_to_icts(double angle)
{
    /*****************************************************************
```

```
    ** Convert an angle (in radians) to ICounts (Interpolated Counts)
    ** The conversion is done by using the stored conversion factor
    ** CountsPerRadian.
    **
    ** Pamameter:
    ** angle.......angle in radians to convert
    ** return value:
    ** Value in I Counts.
    ********************************************************************/
    //-------------------------------------------------------------------
    // The Micro E makes it very easy to go from angle to Icounts
    // The Angle is Supposed To BE 39.5596 degrees or 0.690445 radians
    //-------------------------------------------------------------------
    return (long)(angle * DeviceTable->CountsPerRadian);
}

double icts_to_angle(long icts)
{
    /*******************************************************************
    ** Convert ICounts (Interpolated Counts) to an Angle (radians)  The
    ** Conversion is done by using the stored conversion factor
    ** CountsPerRadian
    **
    ** Parameter:
    ** icts............Raw Position in ICounts
    ** Return Value:
    ** Angle in Radians
    *******************************************************************/
    return (double)icts / DeviceTable->CountsPerRadian;
}

// convert track number to radius
double track_to_rad(double track)
{
    /*******************************************************************
    ** Convert track number (in tracks) to Radius (in inches)
    **
    ** Paramter:
    ** track......track number to convert(this value can be fractional).
    ** Return value:
    ** Radius in inches
    *******************************************************************/
    return DeviceTable->insideradius + (DeviceTable->maxtrack - track) * (DeviceTable->tracksize
*0.000001);
}

// convert radius to track number
double rad_to_track( double rad)
{
    /*******************************************************************
    ** Convert a Radius (in inches) to a Track (in tracks)
    **
    ** Parameter:
    ** rad.........radius (in inches) to convert
    **
    ** Return Value:
    ** Track in tracks (this value can be fractional)
    *******************************************************************/
    return DeviceTable->maxtrack - ((rad - DeviceTable->insideradius) / (DeviceTable->tracksize*
.000001));
}

// convert radius to angle
double rad_to_angle(double rad,int *error)
{
    /*******************************************************************
```

```
    ** Convert a radius (in inches) to an angle (in radians)
    **
    ** Parameter:
    ** rad........Radius (in inches) to convert
    ** error......pointer to int where status is stored
    **            0 == success, Negative == failure
    **            This fuction also updates the error string if there is
    **            an error
    ** Return Value:
    ** Angle in Radians
    ********************************************************************/
    *error = ACUTRAC_SUCCESS;

    double angle, eval;
    // protect against divide by zero
    if (DeviceTable->pivottogap == 0)
    {
        *error = ACUTRAC_ERROR;
        AcutracPrintErrorMessage(*error,"RADIUS to ANGLE:Pivtogap=0");
        return 0.0;
    }

    if (DeviceTable->pivottocenter == 0)
    {
        *error = ACUTRAC_ERROR;
        AcutracPrintErrorMessage(*error,"RADIUS to ANGLE:Pivtocntr=0");
        return 0.0;
    }
    //      A = ARCCOS ((b*b + c*c - a*a)/2*b*c) For HSA level
    //      A - angle between rotary arm and x axis
    //      b - distance from pivot to gap
    //      c - distance from pivot to center of disk
    //      a - distance from gap to center of disk (RADIUS)
    eval = ((DeviceTable->pivottogap*DeviceTable->pivottogap) + (DeviceTable->pivottocenter*Devi
ceTable->pivottocenter)
            - (rad*rad))/(2*DeviceTable->pivottogap*DeviceTable->pivottocenter);
    angle = acos(eval);

    return (angle);
}

// convert angle to radius
double angle_to_rad(double angle)
{
    /*******************************************************************
    ** Convert an anagle (in radians) to a Radius (in inches)
    **
    ** Parameter:
    ** angle.........angle (in radians) to convert
    ** Return Value:
    ** Radius (in inches).
    *******************************************************************/
    double rad;
    //      A - angle between rotary arm and x axis
    //      b - distance from pivot to gap
    //      c - distance from pivot to center of disk
    //      a - distance from gap to center of disk (RADIUS)
    //      rad = sqrt(b*b + c*c - 2*b*c*cos(A))
    rad = sqrt(pow(DeviceTable->pivottogap,2) + pow(DeviceTable->pivottocenter,2)
        - 2*DeviceTable->pivottogap*DeviceTable->pivottocenter*cos(angle));
    return (rad);
}

int CalculateAccelTime(int mode, int rpm)
{
    /*****************************************************************
```

```
    ** This function calculates the amount of time it will take to change
    ** the speed of the spindle motor.  The return value is then used to
    ** set the timeout value for the semaphore that pends on the motor
    ** comming up to speed.
    **
    ** Parameter:
    ** mode.............determines if we are speeding up or slowing down
    **                  Values can be ACUTRAC_ACCELTIME_DECEL or
    **                  ACUTRAC_ACCESLTIME_ACCEL
    ** rpm..............change in speed in rpms.
    ** Return Value:
    ** Returns a time tick value.  There are 100 ticks per second.  An
    ** extra 100 ticks is added to the value as a pad
    ********************************************************************/
    int time,a;
    long r;

    if(mode == ACUTRAC_ACCELTIME_DECEL)
        //get accel
        a = Xio(SPIN_GETDECEL,sys.HSpindle,(char *)0,(char *)0,0l,0);

    else
        a = Xio(SPIN_GETACCEL,sys.HSpindle,(char *)0,(char *)0,0l,0);
    time = rpm/a;
    r = (long)(rpm%a);
    time *= 100;    //one hundred ticks per second
    r *= 100;
    r /= (long)a;
    time += (int)r;
    return time + 100;  //add a one second pad
}

char *AcutracGetErrorBuffer(void)
{
    /*******************************************************************
    ** Return a pointer to AcutracErrorString
    **
    ** Paramter:none
    ** Return Value:
    ** Pointer to ActuracErrorString
    *******************************************************************/
    return AcutracErrorString;
}

int AcutracPrintErrorMessage(int ErrorNumber,char *s)
{
    /*****************************************************************
    ** Print a message to AcutracErrorString
    **
    ** Parameter:
    ** ErrorNumber.....this is an error value returned by a function
    ** s..............pointer to a string that may have more info
    ** Return Value:
    ** Returns the number of characters printed to the string
    *****************************************************************/
    int c;

    StringExclude->Pend();
    c = sprintf(AcutracErrorString,"Acutrac:Error:%s:%s",EStrings[-ErrorNumber],s);
    StringExclude->Post();
    return c;
}

static int PrintStateMessage(int index,int type)
{
    /*****************************************************************
```

```
    ** Create a string indicating the current state of some of the spinstand
    ** functions (Start, Stop, Reset, Lift, Lower).
    **
    ** Paramter:
    ** index.........usually indicates what is happening in the function
    ** type..........usually indicates which fuction is reporting.
    ** Return Value:
    ** returns the number of characters printed to AcutracStateString
    **
    ** It should be noted that both parameters index into the same set
    ** of strings, so the format of the result is somewhat at the wim
    ** of the programmer.  The index values are enumerated by the values
    ** listed above of the format LSS_xxxxxxx...
    **
    ***********************************************************************/
    int c;
    StringExclude->Pend();
    c = sprintf(AcutracStateString,"%s:%s",StateStrings[type],StateStrings[index]);
    StringExclude->Post();
    return c;
}

//---------------------------------------------------------------------------
// export functions for access primarily over the serial port
//---------------------------------------------------------------------------

TCB *GetActionTask(void)
{
    /***********************************************************************
    ** Return a pointer to the ActionTask.  This is the task pointer
    ** that points to either the Start, Stop, Reset, Lift, or Lower task
    ** that is running.  ActionTask is NULL if none of these is active.
    ** Only one of those tasks can be running at any one time.
    **
    ** Paramter:
    ** <none>
    ** ReturnValue:
    ** pointer to the current ActionTask, NULL if non are active
    ***********************************************************************/
    return ActionTask;
}

void AcutracSETStatus(int status)
{
    /***********************************************************************
    ** This function set the Status data member of ActionStatus
    **
    ** Parameter:
    ** status......This is the value to set data member Status to.
    ** return Value:
    ** <none>
    ***********************************************************************/
    ActionStatus.Status = status;
}

int AcutracGETSTATE(int *status,int *function,int *startflag)
{
    /***********************************************************************
    ** This function copies the values in ActionStatus.  This must be
    ** done during a critical section since these value may be changed
    ** by several different threads.
    **
    ** Parameter:
    ** status.....pointer to memory to put Status data member.
    ** function...pointer to memory to put Function data member.
    ** startflag..pointer to memory to put StartFlag data member.
```

```
    ** Return Value:
    ** returns value of data member State.
    ********************************************************************/
    int s;
    int sr;

    sr = EnterCritical();
    s = ActionStatus.State;
    *status = ActionStatus.Status;
    *function = ActionStatus.Function;
    *startflag = ActionStatus.StartFlag;
    ExitCritical(sr);
    return s;
}

int AcutracGotoLoadUnloadRadius(int rad)
{
    /*******************************************************************
    ** Send the Rotary Actuator to either the Load or the Unload Radius
    **
    ** Parameter:
    ** rad......This is a flag, its value can be ACUTRAC_LOADRADIUS or
    **          ACUTRAC_UNLOADRADIUS
    ** Return Value:
    ** Returns 0 on success, or a negative value on failure.
    *******************************************************************/
    double angle;
    long icnts;
    int error=0;
    double radius;

    /*  Are we going to move to the load or unload radius   */
    if(rad == ACUTRAC_LOADRADIUS)
        radius = DeviceTable->LoadRadius;
    else if(rad = ACUTRAC_UNLOADRADIUS)
        radius = DeviceTable->UnloadRadius;

    angle = rad_to_angle(radius,&error);
    if(error)
    {
        //perform some sort of error recovery
        goto exit;      //teminate load
    }
    icnts = angle_to_icts(DeviceTable->referenceangle - angle);
    if((error = ChkErr(Set_Goal(RSTAGE,icnts * DeviceTable->dutorientation) ) ) < 0)
    {
        //perform some sort of error recovery
        goto exit;
    }
    ClearMoveDoneSemaphore();
    if((error = ChkErr(Do_move(RSTAGE,MOVE_ABSOLUTE)) ) < 0)
    {
        //perform some sort of error recovery
        goto exit;
    }

    //Wait for Rotary Stage to get into position
    if((error = ChkErr(WaitForRotaryMove(200))) != 0)
    {
        //perform some sort of error recovery
        goto exit;
    }
exit:
    return error;
}
```

```
int AcutracUnloadHeads(void)
{
    /*****************************************************************
    ** This function clears the ramp semaphores, then moves the ramp
    ** lifters in to remove the heads off of the disk.  The function
    ** returns when the ramp semaphore is fired by the ramp sensor
    **
    ** Parameter:
    ** <none>
    ** Return Value:
    ** 0 on success, negative on utter failure.
    *****************************************************************/
    ClearRampFlags();
    StageLoadHeads(STAGELOADHEAD_RAMPIN);       //spread Fingers
    return ChkErr(WaitForRampIn(1000));
}

int AcutracINIT(char *name)
{
    /*****************************************************************
    ** This function initializes all of the software for the high level
    ** functions, but does not initialize any of the hardware (i.e., the
    ** rotary actuator is not calibrated, pnuematics are not set to their
    ** default positions, etc.).
    ** The primary action is to read in the afg file for the product
    ** being used.  This file must be located in the RAMDISK.  This
    ** function does not spawn a task.
    **
    ** parameter:
    ** name....name of the AFG file to load, located in RAMDISK
    ** Return Value
    ** returns 0 on success, a negative value on failure.
    *****************************************************************/
    int retval = ACUTRAC_SUCCESS;
//  int c;
    String *S = new String;
    char *s = S->Get();         //allocate temp string
    char *p;

    if(!DeviceTable) DeviceTable = new AcutracDeviceTable();
    sprintf(s,"D:%s",name);         //create device:filename,ramdisk device
    if((p = strchr(s,'.')) != NULL) //look for extension
    {
        ++p;    //point to first character of extension
        if(strcmp(p,"afg") == 0)            //ascii file extension?
        {
            if(LoadConfigFile(s,DeviceTable))
            {
                String *O = new String;
                char *o = O->Get();

                GetLoadConfigError(o);
                retval = ACUTRAC_FILEERROR;
                AcutracPrintErrorMessage(retval,o);
                delete O;
            }
            else
            {
                int v;
                DeviceTable->CountsPerRadian = AcutracGETCORRECTION();
                TrackMode = 1;  //tracks are defined now
                //----------------------------------------
                // new for version 1.52.00
                // we need to send a notification to
                // several routines that need to know what
                // kind of a setup we have
```

```
                //---------------------------------------
                if((DeviceTable->mode == HSA) || (DeviceTable->mode == HSAR))
                {
                    v = RAMPCOMMAND_HSAMODE;
                    RampInPipe->PostMessage(&v,1);
                    RampOutPipe->PostMessage(&v,1);
                }
                else if(DeviceTable->mode == HGA)
                {
                    v = RAMPCOMMAND_HGAMODE;
                    RampInPipe->PostMessage(&v,1);
                    RampOutPipe->PostMessage(&v,1);
                }
            }
        }
        else    //this is where one would have found the code to load a
                //binary file, but, this mode has never been implemented
                //and probably never will be.  One of these days
                //the remnants of this need to be removed.
            retval = ACUTRAC_FILEBAD;
    }
    //---------------------------------------------------------------
    // use table values to set up various things
    //---------------------------------------------------------------
    StageSetDirection(DeviceTable->rotation);   //set spindle direction
    delete S;                       //delete temp string
    return retval;
}

//-------------------------------------------------------------------------
//
// AcutracRESET
//
// Reset spinstand
//
// The name of the config file is passed to this function
// So that the new desired configuration is loaded
//
//-------------------------------------------------------------------------

int AcutracRESET(char *fname)
{
    /*****************************************************************
    ** This is the function that "resets" the spinstand.  This function
    ** causes a complete initialization of the spinstand to occure.
    ** It first will call AcutracINIT to initialize the software, and then
    ** it will spawn a task that will take care of the hardware initialization
    **
    ** Parameter:
    ** fname......name of the afg file that describes the product
    ** Return Value:
    ** 0 on success, negative on failure.
    *****************************************************************/
    void (*func)(void);
    int c;

    int retval = ACUTRAC_SUCCESS;

    //---------------------------------------------------------------
    // The global variable ActionTask is used sort of as a semaphore
    // Action Task points to a task that is doing something with the
    // spinstand.  It should be noted that this variable is not protected
    // with a real semaphore.  Since an Action Task can only be initiated
    // by a command from the outside world, it should not be posible to
    // try and start two action tasks at the same time.  This could, however
    // change in the future.
```

```
    // If ActionTask is 0, no task is running, so it is permisable to
    // spawn a task.
    // If ActionTask is non zero, an ActionTask is already running, so
    // in the case of RESET, we will Terminate the task.
    //-------------------------------------------------------------------
    if(ActionTask)  //is there already an ActionTask?
    {
        //---------------------------------------
        // if a spinstand task is already running
        // terminate it
        //---------------------------------------
        TDelete(ActionTask);
        ActionStatus.StartFlag = ACUTRAC_ACTION_START_NOT;
        ActionTask = 0; //Set to NULL
    }
    c = I2CGetStageStatus();    //get current status from the spinstand
    if(c & FPCOMMAND_AIR)   //is there air pressure?
    {
        retval = ACUTRAC_LOWAIR;    //low air presure error
        AcutracPrintErrorMessage(retval,"AcutracRESET");
        goto exit;
    }
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_NOT;
    //---------------------------------
    // load in new config file
    //---------------------------------
    if((retval = AcutracINIT(fname)) != ACUTRAC_SUCCESS) goto exit;

    //-------------------------------------------------
    // Select the function to use for reset proceedure
    //-------------------------------------------------
    if(ActionStatus.StartFlag == ACUTRAC_ACTION_START_NOT)
    {
        //select a function to spawn to do hardware initialization
        if(DeviceTable->mode == HSA)
            func = HSAReset;
        else if(DeviceTable->mode == HGA)
            func = HGAReset;
        else if (DeviceTable->mode == HSAR)
            func = HSAReset;
        ActionTask = CreateTask(func,512,12,"Accutrac_Reset");
        ActionStatus.Function = ACUTRAC_ACTION_PENDING;
        int sr = EnterCritical();
        ActiveTasks->Insert(ActionTask);    //put the task into priority queue
        ExitCritical(sr);
    }
    else
        retval = ACUTRAC_ERROR;
exit:
    return retval;
}

//---------------------------------------------------------------------------
// AcutracSTART
//
// This function starts up the START task.  That is, we load the heads onto
// the disk.  The actual start function that is called depends on the mode
// that is in the config file.  The only thing that prevents the start task
// from starting is if the state of the startflag is not set to start_ready
//
//---------------------------------------------------------------------------

int AcutracSTART(void)
{
    /*************************************************************************
    ** This function loads the heads on the disk.  The actual loading is
```

```
    ** done by a task that is spawed by this function.
    **
    ** Parameter:
    ** <none>
    ** Return Value
    ** 0 on success, negative of failure
    ***************************************************************************/
    void (*func)(void);

    int retval = ACUTRAC_SUCCESS;
    //-------------------------------------------------------------------------
    // The global variable ActionTask is used sort of as a semaphore
    // Action Task points to a task that is doing something with the
    // spinstand.  It should be noted that this variable is not protected
    // with a real semaphore.  Since an Action Task can only be initiated
    // by a command from the outside world, it should not be posible to
    // try and start two action tasks at the same time.  This could, however
    // change in the future.
    // If ActionTask is 0, no task is running, so it is permisable to
    // spawn a task.
    // If ActionTask is non zero, an ActionTask is already running, so
    // we cannot start another one.
    //-------------------------------------------------------------------------
    if(ActionTask)
        retval = ACUTRAC_BUSYERROR;
    else
    {
        if(ActionStatus.StartFlag == ACUTRAC_ACTION_START_READY)
        {
            //select which function to spawn based on product type
            if(DeviceTable->mode == HSA)
                func = HSAStart;
            else if(DeviceTable->mode == HGA)
                func = HGAStart;
            else if(DeviceTable->mode = HSAR)
                func = HSAStart;    //use the same driver for HSA for resonance test
            ActionTask = CreateTask(func,512,12,"Accutrac_Start");
            ActionStatus.Function = ACUTRAC_ACTION_PENDING;
            int sr = EnterCritical();
            ActiveTasks->Insert(ActionTask);    //insert task into priority queue
            ExitCritical(sr);
        }
        else
            retval = ACUTRAC_ERROR;

    }
    return retval;
}

//-----------------------------------------------------------------------------
// AcutracSTOP
//
// This function is used to start the STOP task, that is unloading the heads
// The only thing that will stop this function is if the START function has
// not been called or if action status is active
//
//-----------------------------------------------------------------------------

int AcutracSTOP(void)
{
    /*************************************************************************
    ** This function causes another function to spawn as a task that will
    ** handle the unloading of the heads off of the disk.
    **
    ** Paramter:
    ** <none>
```

```
    ** Return Value:
    ** Returns 0 on success, negative on failure.
    ************************************************************************/
    void (*func)(void);
    int retval;

    //-------------------------------------------------------------------
    // The global variable ActionTask is used sort of as a semaphore
    // Action Task points to a task that is doing something with the
    // spinstand.  It should be noted that this variable is not protected
    // with a real semaphore.  Since an Action Task can only be initiated
    // by a command from the outside world, it should not be posible to
    // try and start two action tasks at the same time.  This could, however
    // change in the future.
    // If ActionTask is 0, no task is running, so it is permisable to
    // spawn a task.
    // If ActionTask is non zero, an ActionTask is already running, so
    // we cannot start another one.
    //-------------------------------------------------------------------
    if(ActionTask)  //is a task already running?
        retval = ACUTRAC_BUSYERROR;
    else
    {
        retval = ACUTRAC_SUCCESS;
        if(ActionStatus.StartFlag == ACUTRAC_ACTION_START_YES)
        {
            //select function to spawn based on product type
            if(DeviceTable->mode == HSA)
                func = HSAStop;
            else if(DeviceTable->mode == HGA)
                func = HGAStop;
            else if(DeviceTable->mode == HSAR)
                func = HSAStop; //use HSA function for resonance test
            ActionTask = CreateTask(func,512,12,"Accutrac_Stop");
            ActionStatus.Function = ACUTRAC_ACTION_PENDING;
            int sr = EnterCritical();
            ActiveTasks->Insert(ActionTask);    //insert task into priority queue
            ExitCritical(sr);
        }
        else
            retval = ACUTRAC_ERROR;
    }
    return retval;
}

//-------------------------------------------------------------------------
//
// Function to lift heads off of disk
//
//-------------------------------------------------------------------------

static const void (* const Liftfunc[3])(void) = {
    HSALift,
    HGALift,
    HSALift
};


int AcutracLIFTHEADS(void)
{
    //---------------------------------------
    // this function moves heads to load/unload
    // radius and causes heads to be raised.
    // This is done indirectly by causing a
    // TASK to be spawned which will do this
    //
```

```
    // Paramter:
    // <none>
    // Return Value
    // returns 0 on success, negative on failure
    //---------------------------------------
    int retval = ACUTRAC_SUCCESS;

    //-------------------------------------------------------------------
    // The global variable ActionTask is used sort of as a semaphore
    // Action Task points to a task that is doing something with the
    // spinstand.  It should be noted that this variable is not protected
    // with a real semaphore.  Since an Action Task can only be initiated
    // by a command from the outside world, it should not be posible to
    // try and start two action tasks at the same time.  This could, however
    // change in the future.
    // If ActionTask is 0, no task is running, so it is permisable to
    // spawn a task.
    // If ActionTask is non zero, an ActionTask is already running, so
    // we cannot start another one.
    //-------------------------------------------------------------------
    if(ActionTask)  //is a task already running?
        retval = ACUTRAC_BUSYERROR;
    else
    {
        if(ActionStatus.StartFlag == ACUTRAC_ACTION_START_YES)
        {
            if(Liftfunc[DeviceTable->mode])
            {
                //select function based on product type
                ActionTask = CreateTask((void(*)(void))Liftfunc[DeviceTable->mode],512,12,"Accut
rac_Lift");
                ActionStatus.Function = ACUTRAC_ACTION_PENDING;
                int sr = EnterCritical();
                //insert task into priority queue
                ActiveTasks->Insert(ActionTask);
                ExitCritical(sr);
            }
            else
                retval = ACUTRAC_NOTHANDLED;
        }
        else
            retval = ACUTRAC_ERROR;
    }
    return retval;
}

//-------------------------------------------------------------------------
//
// Function to reload heads onto disk
//
//-------------------------------------------------------------------------

static const void (* const ReloadFunc[3])(void) = {
    HSAReload,
    HGAReload,
    HSAReload
};

int AcutracRELOADHEADS(void)
{
    //--------------------------------------
    // This function is used to reload the
    // heads back onto the disk after they
    // have been raised.  This is done
    // indirectly by causing a TASK to be
    // spawned which will do this.
```

```
    //---------------------------------------
    int retval = ACUTRAC_SUCCESS;

    //-------------------------------------------------------------------
    // The global variable ActionTask is used sort of as a semaphore
    // Action Task points to a task that is doing something with the
    // spinstand.  It should be noted that this variable is not protected
    // with a real semaphore.  Since an Action Task can only be initiated
    // by a command from the outside world, it should not be posible to
    // try and start two action tasks at the same time.  This could, however
    // change in the future.
    // If ActionTask is 0, no task is running, so it is permisable to
    // spawn a task.
    // If ActionTask is non zero, an ActionTask is already running, so
    // we cannot start another one.
    //-------------------------------------------------------------------
    if(ActionTask)
        retval = ACUTRAC_BUSYERROR;
    else
    {
        if(ActionStatus.StartFlag == ACUTRAC_ACTION_LIFTED)
        {
            if(ReloadFunc[DeviceTable->mode])
            {
                ActionTask = CreateTask((void(*)(void))ReloadFunc[DeviceTable->mode],512,12,"Acc
utrac_Reload");
                ActionStatus.Function = ACUTRAC_ACTION_PENDING;
                int sr = EnterCritical();
                ActiveTasks->Insert(ActionTask);
                ExitCritical(sr);
            }
            else
                retval = ACUTRAC_NOTHANDLED;
        }
        else
            retval = ACUTRAC_ERROR;
    }
    return retval;
}

//-------------------------------------------------------------------------
// Functions that maintain callibration data
//-------------------------------------------------------------------------

static const char DataCalFn[] = "D:DATA.CAL";

int AcutracSETCORRECTION(char *cal)
{
    /*********************************************************************
    ** This function is used to set the calibration factor for the encoder.
    ** The file that this value is written into is hard coded.
    **
    ** Parameter:
    ** cal.......Pointer to a string that represents the calibration factor
    ** ReturnValue:
    ** This function always returns 0
    *********************************************************************/
    int handle;
    int c;
    String *S = new String;
    char *s = S->Get();

    handle = Open((char *)DataCalFn,WRITE_ONLY);
    c = sprintf(s,"%s\n",cal);
    Write(handle,s,c);
    Close(handle);
```

```
    delete S;
    return 0;
}

int AcutracGETCORRECTION(char *cal)
{
    /*********************************************************************
    ** This function is used to read the calibration factor.
    **
    ** Paramter:
    ** cal.....pointer to a string to store the calibration string in
    ** ReturnValue:
    ** This function always returns 0
    *********************************************************************/
    int handle;

    handle = Open((char *)DataCalFn,READ_ONLY);
    //-------------------------------------------------------------
    // code to check for valid handle was added in version 1.52.00
    //-------------------------------------------------------------
    if(handle >= 0)     //is it a valid handle?
    {
        Read(handle,cal,20l);   //read 20 characters max into string
        Close(handle);
    }
    return 0;
}

//---------------------------------
// this function returns back counts per radian
//---------------------------------

double AcutracGETCORRECTION(void)
{
    /*********************************************************************
    ** This function is used to get the Calibration for the encode is the
    ** form of a double
    **
    ** Parameter:
    ** <none>
    ** ReturnValue:
    ** returns 0 on error, correction value otherwise
    *********************************************************************/
    int handle;
    String *S = new String;
    char *s = S->Get();
    double retval=0.0;

    //-------------------------------------------------------
    //code to check for valid handle added in version 1.52.00
    //-------------------------------------------------------
    if((handle = Open((char *)DataCalFn,READ_ONLY)) >= 0)
    {
        Read(handle,s,20l);
        Close(handle);
        sscanf(s,"%lf\n",&retval);
        //divide seconds/rev by radians/rev = seconds/radian
        double t = 1296000.0 /(2.0 * 3.141592654);
        //divide seconds/radian by seconds/count = counts/radian
        retval = t / retval;
    }
    delete S;
    return retval;
}

//-----------------------------------------------------------------------
```

```
// AcutracSELECTHEAD
//
// select head to do read/write on disk
//
//---------------------------------------------------------------------------

int AcutracSELECTHEAD(int head)
{
    DeviceTable->CurrentHead = head;
    return StageSelectHead(head);
}


//---------------------------------------------------------------------------
// AcutracGETSELECTEDHEAD
//
// get current selected head.  Reads back a shadow value of current head
//
//---------------------------------------------------------------------------

int AcutracGETSELECTEDHEAD(int *head)
{
    /*********************************************************************
    ** This function reads back the current selected head.  This value is
    ** stored in a shadow register in the DeviceTable data structure since
    ** we cannot physically read this value back from the hardware.
    **
    ** Parameter:
    ** head.....pointer to memory area to write head value to
    ** ReturnValue:
    ** This function always returns 0
    *********************************************************************/
    *head = DeviceTable->CurrentHead;
    return 0;
}

//---------------------------------------------------------------------------
// AcutracGETRPM
//
// this function is sort of a duplicate
//
//---------------------------------------------------------------------------

int AcutracGETRPM(int *rpm)
{
    /*********************************************************************
    ** This function reads back the Current Spindle Speed (in RPM).  This
    ** value is maintained in a global variable that is updated by a task.
    **
    ** Parameter:
    ** rpm.....pointer to a memory area where the current RPM is written
    ** ReturnValue:
    ** This function always returns 0 (ACUTRAC_SUCCESS)
    *********************************************************************/
    int retval = ACUTRAC_SUCCESS;
    *rpm = GlobalSpindleRPM;
    return retval;
}

//---------------------------------------------------------------------------
// AcutracGETMAXRPM
//
// Get the maximum allowable RPM
//
//---------------------------------------------------------------------------
```

```
int AcutracGETMAXRPM(int *maxrpm)
{
    /*********************************************************************
    ** This function gets the maximum allowable Spindle Speed (in RPM)
    **
    ** Parameter:
    ** maxrpm.....pointer to memory area to write maximum RPM
    ** ReturnValue:
    ** This function always returns 0
    *********************************************************************/
    int retval = ACUTRAC_SUCCESS;
    *maxrpm = DeviceTable->maxrpm;  //return maximum rpm
    return retval;
}

//---------------------------------------------------------------------------
// AcutracGETMINRPM
//
// Get the minumum allowable RPM
//
//---------------------------------------------------------------------------

int AcutracGETMINRPM(int *minrpm)
{
    /*********************************************************************
    ** This function gets the minimum allowable Spindle Speed (in RPM)
    **
    ** Parameter:
    ** minrpm.....pointer to memory area to write minimum RPM
    ** ReturnValue:
    ** This function always returns 0
    *********************************************************************/
    int retval = ACUTRAC_SUCCESS;
    *minrpm = DeviceTable->minrpm;  //return minimum rpm
    return retval;
}

//---------------------------------------------------------------------------
// AcutracSTARTSPINDLE
//
// This function is sort of a duplicate
//
//---------------------------------------------------------------------------

int AcutracSTARTSPINDLE(void)
{
    /*********************************************************************
    ** This function starts the spindle motor spinning
    **
    ** Parameter:
    ** <none>
    ** ReturnValue:
    ** Returns 0 on success, negative on error
    *********************************************************************/
    int retval = ACUTRAC_SUCCESS;
    if(StageSetMotor(1)) retval = ACUTRAC_ERROR;
    return retval;
}

//---------------------------------------------------------------------------
// AcutracSTOPSPINDLE
//
// This function is sort of a duplicate
//
//---------------------------------------------------------------------------
```

```
int AcutracSTOPSPINDLE(void)
{
    /*****************************************************************
    ** This function stops the spindle motor spinning
    **
    ** Parameter:
    ** <none>
    ** ReturnValue:
    ** Returns 0 on success, negative on error
    *****************************************************************/
    int retval = ACUTRAC_SUCCESS;
    if(StageSetMotor(0) ) retval = ACUTRAC_ERROR;
    return retval;
}

//-----------------------------------------------------------------------
// AcutracSETROTDIR
//
// This function is sort of a duplicate
//
//-----------------------------------------------------------------------

int AcutracSETROTDIR(int dir)
{
    /*****************************************************************
    ** This function is used to set the direction of rotation of the
    ** spindle motor.
    **
    ** Parameter:
    ** dir...determines the direction, CCW (0) or CW (1)
    ** ReturnValue:
    ** Returns 0 on success, negative on failure
    *****************************************************************/
    int retval = ACUTRAC_SUCCESS;
    if(StageSetDirection(dir)) retval = ACUTRAC_ERROR;
    return retval;
}

//-----------------------------------------------------------------------
// AcutracGETROTDIR
//
// this function is sort of a duplicate
//-----------------------------------------------------------------------

int AcutracGETROTDIR(int *dir)
{
    /*****************************************************************
    ** This function is used to get the direction of rotation of the
    ** spindle motor
    **
    ** Parameter:
    ** dir....pointer to memory area to write spindle direction, CCW=0,CW=1
    ** ReturnValue:
    ** This function always returns 0
    *****************************************************************/
    int retval = ACUTRAC_SUCCESS;
    *dir = Xio(SPIN_GETDIRECTION,sys.HSpindle,(char *)0,(char *)0,0l,0);
    return retval;
}

//-----------------------------------------------------------------------
// AcutracSETLOADRPM
//
// this function is used to the the RPM that the heads will load at
//
//-----------------------------------------------------------------------
```

```
int AcutracSETLOADRPM(int rpm)
{
    /*********************************************************************
    ** This function is used to set the RPM for when the heads are dropped
    ** onto the disk
    **
    ** Parameter:
    ** rpm.......value, in RPMs, of spindle motor speed
    ** ReturnValue:
    ** Returns 0 on Success, negative on failure
    *********************************************************************/
    int retval = ACUTRAC_SUCCESS;
    if((rpm > DeviceTable->maxrpm) || (rpm < 0))
    {
        retval = ACUTRAC_OUTOFRANGE;
        AcutracPrintErrorMessage(retval,"Load Rpm:Range");
    }
    else
        DeviceTable->loadrpm = rpm;
    return retval;
}

//-----------------------------------------------------------------------
// AcutracGETLOADRPM
//
// this function is used to get the load rpm
//
//-----------------------------------------------------------------------

int AcutracGETLOADRPM(int *rpm)
{
    /*********************************************************************
    ** This function is used to read back the load rpm.
    **
    ** Parameter:
    ** rpm....pointer to memory area where the RPM is written.
    ** ReturnValue:
    ** This function always returns 0
    *********************************************************************/
    int retval = ACUTRAC_SUCCESS;
    *rpm = DeviceTable->loadrpm;
    return retval;
}

//-----------------------------------------------------------------------
// AcutracSEEKTRACK
//
// This function is used to seek to a given track.  There are two functions
// with overloaded parameters.  The track can be passed either as a double or
// as a character string.  The character string is primarily for over the
// RS232 port.
//
//-----------------------------------------------------------------------

int AcutracSEEKTRACK(char *track)
{
    /*********************************************************************
    ** This function is used to convert a character string in the form:
    ** XXXXX.XXXX to a double value track, and then it seeks that track.
    ** It should be noted that this function is sort of obsolete now,
    ** Since it will only go up to track 99999.9999.  Use the function, below
    **
    ** Parameter:
    ** track....pointer to a character string representing the track
    ** ReturnValue:
```

```
    ** Returns 0 on success, negative on failure
    ************************************************************************/
    double trackgoal = atof(track);
    return AcutracSEEKTRACK(trackgoal);
}

int AcutracSEEKTRACK(double trackgoal)
{
    /************************************************************************
    ** This function is used to seek to a given track.
    **
    ** Paramter:
    ** track....track number to seek
    ** ReturnValue:
    ** Returns 0 on success, negative on failure.
    ************************************************************************/
    int retval = ACUTRAC_SUCCESS;
    if((trackgoal < -100.0) || (trackgoal > (DeviceTable->maxtrack + 100.0)))
        retval =  ACUTRAC_OUTOFRANGE;
    else
    {
        double radius, angle;
        long icts;
        radius = track_to_rad(trackgoal);
        angle = rad_to_angle(radius,&retval);
        if(retval != ACUTRAC_SUCCESS) goto exit;
        icts = (long) angle_to_icts(DeviceTable->referenceangle - angle);
        if((retval = ChkErr(Set_Goal(ROTARY_STAGE,icts * DeviceTable->dutorientation)) ) != ACUT
RAC_SUCCESS) goto exit;
        if((retval = ChkErr(Do_move(ROTARY_STAGE,MOVE_ABSOLUTE))) != ACUTRAC_SUCCESS) goto exit;
        DeviceTable->currenttrack = trackgoal;
    }
exit:
    return retval;
}

int AcutracGOTORADIUS(double radius)
{
    /************************************************************************
    ** This function is used to move the rotary actuator to a given radius
    **
    ** Parameter:
    ** radius....The radius to move to, in inches
    ** ReturnValue:
    ** Returns 0 on success, negative on failure.
    ************************************************************************/
    double angle;
    int error;
    long icnts;

    angle = rad_to_angle(radius,&error);
    if(error)
    {
        goto exit;      //teminate load
    }
    icnts = angle_to_icts(DeviceTable->referenceangle - angle);
    if((error = ChkErr(Set_Goal(RSTAGE,icnts * DeviceTable->dutorientation) ) ) < 0)
    {
        goto exit;
    }
    ClearMoveDoneSemaphore();
    if((error = ChkErr(Do_move(RSTAGE,MOVE_ABSOLUTE)) ) < 0)
    {
        if(error==ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
    }
```

```
exit:
    return error;
}

int AcutracGOTORADIUS(char *radius)
{
    /*************************************************************************
    ** This function is used to move the rotary actuator to a given radius,
    ** using an ascii string as the input.
    **
    ** Paramter:
    ** radius....pointer to a string of the form X.XXXXXXX that represents
    **           the desired radius, in inches
    ** ReturnValue:
    ** Returns 0 on success, negative on failure
    *************************************************************************/
    return AcutracGOTORADIUS(atof(radius));
}

//-------------------------------------------------------------------------
// ACutracSTEPIN
//
// this funtion increments the track by 1
//
//-------------------------------------------------------------------------

int AcutracSTEPIN(void)
{
    /*************************************************************************
    ** This function is used to move the actuator by 1 track, it does this
    ** by adding 1 to the current track value.  The result could have a
    ** fractional component.
    **
    ** Paramter:
    ** <none>
    ** ReturnValue:
    ** returns 0 on success, negative on failure
    *************************************************************************/
    double track = DeviceTable->currenttrack + 1.0;
    int retval;
    if(track > DeviceTable->maxtrack)
    {
        retval = ACUTRAC_OUTOFRANGE;
    }
    else
    {
        retval = AcutracSEEKTRACK(track);
    }
    return retval;
}

//-------------------------------------------------------------------------
// AcutracSTEPOUT
//
// This function decrements the track by 1
//
//-------------------------------------------------------------------------

int AcutracSTEPOUT(void)
{
    /*************************************************************************
    ** This function is used to move the actuator by -1 track, it does this
    ** by adding -1 to the current track value.  The result could have a
    ** fractional component.
    **
    ** Paramter:
```

```
    ** <none>
    ** ReturnValue:
    ** returns 0 on success, negative on failure
    ************************************************************************/
    double track = DeviceTable->currenttrack - 1.0;
    int retval;
    if(track > DeviceTable->maxtrack)
    {
        retval = ACUTRAC_OUTOFRANGE;
    }
    else
    {
        retval = AcutracSEEKTRACK(track);
    }
    return retval;
}

//-------------------------------------------------------------------------
// AcutracGETTRACK
//
// This function gets the current track number.  There are two different
// functions, one that returns track as a character string and one that
// returns track as a double.
//
//-------------------------------------------------------------------------

int AcutracGETTRACK(char *track)
{
    /************************************************************************
    ** This function retrieves the current track
    ** It returns the track as an ascii string in the form of XXXXX.XXXX
    **
    ** Parameter:
    ** track....pointer to a character string where the track number is
    **          to be stored.
    ** ReturnValue:
    ** This function always returns 0
    ************************************************************************/
    //--------------------------------
    // track points to an array of at
    // least 11 characters in which
    // to print the track number
    //--------------------------------
    int retval = ACUTRAC_SUCCESS;
    sprintf(track,"%5.4lf",DeviceTable->currenttrack);  //convert to float
    track[10] = 0;  //make sure it is 10 char long
    return retval;
}

int AcutracGETTRACK(double *track)
{
    /************************************************************************
    ** this function retrieves the current track value
    **
    ** Parameter:
    ** track....pointer to a double value where the track is stored
    ** ReturnValue:
    ** This function always returns 0
    ************************************************************************/
    *track = DeviceTable->currenttrack;
    return ACUTRAC_SUCCESS;
}

//-------------------------------------------------------------------------
// AcutracSETMAXTRACK
//
```

```
// Set the maximum number of tracks, well actually, set the total number of
// tracks.
//
//---------------------------------------------------------------------------

int AcutracSETMAXTRACK(char *s)
{
    /*************************************************************************
    ** This function set the value of the highest track number
    **
    ** Parameter:
    ** s....character string representing the track number
    ** ReturnValue:
    ** This function always returns 0
    *************************************************************************/
    int retval = ACUTRAC_SUCCESS;
    DeviceTable->maxtrack = atof(s);
    return retval;
}

//---------------------------------------------------------------------------
// AcutracGETMAXTRACK
//
// Get the maximum number of track, well acutally, the total number of tracks
//
//---------------------------------------------------------------------------

int AcutracGETMAXTRACK(char *maxtrack)
{
    /*************************************************************************
    ** This function returns the maximum track number.  It returns this value
    ** in the form of a string of format XXXXX.XXXX.  It should be noted that
    ** this will not be very useful shortly since the max track # will be
    ** 99,999.9999.
    **
    ** Parameter:
    ** maxtrack....pointer to a character string of at least 11 characters
    **             where the track value will be stored.
    ** ReturnValue:
    ** This function always returns a 0
    *************************************************************************/
    int retval = ACUTRAC_SUCCESS;
    sprintf(maxtrack,"%5.4lf",DeviceTable->maxtrack);
    return retval;
}


//---------------------------------------------------------------------------
// AcutracSETRADIUS
//
// This function is used to the the RADIUS of the disk.  The first parameter,
// if true, indicates that the inside radius is being operated on.
//
//---------------------------------------------------------------------------

int AcutracSETRADIUS(int IsId,char *radius)
{
    /*********************************************************************
    ** This function is used to set the inside and outside radius
    ** It should be noted that radius is represented by a character string
    **
    ** Parameter:
    ** IsId....Flag, IsId=0=>outside radius;IsId=1=>inside radius
    ** radius..pointer to a character string representing the radius in
    **         inches.
    ** ReturnValue:
```

```
    ** This function always returns 0
    **********************************************************************/
    int retval = ACUTRAC_SUCCESS;
    double rad = atof(radius);
    if(IsId)
        DeviceTable->insideradius = rad;
    else
        DeviceTable->outsideradius = rad;
    return retval;
}

//---------------------------------------------------------------------------
// AcutracSETTRACKSIZE
//
// This function probably should not be used.  Track size is determined by
// using other paramters
//
//---------------------------------------------------------------------------

int AcutracSETTRACKSIZE(char *tracksize)
{
    int retval = ACUTRAC_SUCCESS;
    DeviceTable->tracksize = atof(tracksize);
    return retval;
}

int AcutracGETTRACKSIZE(char *tracksize)
{
    int retval = ACUTRAC_SUCCESS;
    sprintf(tracksize,"%5.4lf",DeviceTable->tracksize);
    return retval;
}

//---------------------------------------------------------------------------
// AcutracGETERRORMESSAGE
//
// Return back the contents of the error message buffer.  This buffer will
// only store the error message for the last reported error message.  It
// will zap the buffer once the message is read
//
//---------------------------------------------------------------------------

int AcutracGETERRORMESSAGE(char *message)
{
    /**********************************************************************
    ** This function is used to retrieve an error message generated by
    ** various functions in this module.  These are primarily generated by
    ** the Start,Stop,Reset,Lift and Lower functions.
    **
    ** Parameter:
    ** message....pointer to a character string of at least 128 characters
    **            that will have the error message copied into.
    ** ReturnValue:
    ** This function always returns 0
    **********************************************************************/
    int retval = ACUTRAC_SUCCESS;
    StringExclude->Pend();
    strncpy(message,AcutracErrorString,128);
    memset(AcutracErrorString,0,128);   //erase message
    ActionStatus.Status = 0;            //clear action status
    StringExclude->Post();
    return retval;
}

int AcutracGETSTATEMESSAGE(char *message)
{
```

```
    /*********************************************************************
    ** This function is used to retrieve a message that coresponds to
    ** the current state of a spawned function for Start, Stop, Reset,
    ** Lift and Lower.
    **
    ** Parameter:
    ** message....pointer to a string of at least 48 bytes where the
    **            state message will be copied.
    ** ReturnValue:
    ** This function always returns 0.
    *********************************************************************/
    StringExclude->Pend();
    strncpy(message,AcutracStateString,48);
    StringExclude->Post();
    return ACUTRAC_SUCCESS;
}

//*****************************************************************
//
// Miscilaneous Faults, Such as Spindle Amplifier Fault
// About all we can do for this one is make sure that the heads
// are unloaded before we shut down the spindle motor
//*****************************************************************

void SpindleMotorFaultTask(void)
{
    /*********************************************************************
    ** This function is a TASK.
    **
    ** Pending Semaphore:SpindleAmplifierFault
    ** Posting Functions:HandleIrq3 (spinchip.cpp)
    **
    ** This task pends on the SpindleAmplifierFault semaphore until it is
    ** posted by the interrupt routine that handle the overtemp interrupt
    ** from the power amplifier.  If the transition is into the overtemp
    ** range, the spindle controller will attempt to shut down the spinstand
    ** if it is running.
    *********************************************************************/
    int v;
    int StageState;

    SystemStatus.spin.stat.overtemp = SpinGetInputLevel(SPIN_AMPFAULT);
    SpindleAmplifierFault = new Wait(0,"AmpFaultFlag"); //create semaphore for spindle fault
    SpinIOEnableIrq(SPIN_AMPFAULT);         //enable amp fault interrupt
    while(1)                                //this is a task, loop forever
    {
        v = SpindleAmplifierFault->Pend();      //wait for a fault
        //v == 1 BAD
        //v == 0 GOOD
        SystemStatus.spin.stat.overtemp = v;
        if(v)   //if bad unload heads
        {
            BigTimeShutDown(ACUTRAC_AMPFAULT);
        }
    }
}

//-----------------------------------------------------------------
//
// Spinstand functions for HSA
//
//-----------------------------------------------------------------


//*****************************************************
//
```

```
// This function needs to record where it is at each step
// of the way.  In this way, the host computer can get the
// status of the progress on each of these functions.
// also, where start left off at can be used when the user
// goes to do a stop.
//
//-------------------------------------------------------
//
// Start (Head Load) Sequence
//
// 1. Activate Vacuum Chuck to hold DUT
// 2. Prompt User to Install D.U.T.
// 3. Pend of Vacuum sensor == SUCK
// 4. pend on continue button
// 5. Move Rotary Stage to Comb Removal Position
// 6. Pend on rotary stage in position
// 7. move load ramp to H.S.A.
// 8. pend on ramp in
// 9. prompt for comb removal
// 10. activate x run solanoid
// 11. start spindle motor at load RPM
// 12. pend on x stage in position
// 13. pend on spindle motor at speed
// 14. move load ramp into garage
// 15. pend on load ramp out
// 16. Change spindle motor speed to test RPM
// 17. pend on motor at speed
// 18. move rotary stage to starting track
// 19. pend on rotary stage in position
//-------------------------------------------------------
// This function gets executed as a TASK
//
//*******************************************************

static void HSAStart(void)
{
    /*************************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    *************************************************************************/
    double angle;
    long icnts;
    int error;

    ActionStatus.State = 0;
    ActionStatus.Function = ACUTRAC_ACTION_START;
    ActionStatus.Status = 0;
    //-------------------------------------------------------
    // code for doing START
    //-------------------------------------------------------
//-------------------------------------
// Step 1
//-------------------------------------
    DisplayMode=3;          //disable RPM display to front panel
    if(WaitForDisplayCycle->Pend() == EVENT_TERMINATE)
        goto exit;
    I2CWriteLEDString((char *)LCDStrings[LCDS_LOADHEADS]);
    ClearChuckSuck();
    StageSuckHeads(1);      //open up vac on head chuck
    PrintStateMessage(LSS_INSTALLDUT,LSS_LOADHEADS);
```

```
    ActionStatus.State++;   //1
//-------------------------------------
// Step 2
//-------------------------------------
    I2CWriteLEDString((char *)LCDStrings[LCDS_INSERTDUT]);
    PrintStateMessage(LSS_WAITFORVACSENSE,LSS_LOADHEADS);
    ActionStatus.State++;   //2
//-------------------------------------
// Step 3
//-------------------------------------
    if(WaitForChuckSuck(WAITFORCHUCKSUCK_SUCK) == EVENT_TERMINATE)  //wait for head assembly
        goto exit;
    //heads are on chuck, change parameters
    if((error = ChkErr(DeviceTable->Pid[ACUTRAC_HEADSON]->Upload(ACUTRAC_HEADSON)) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ActionStatus.State++;   //3

//-------------------------------------
// Step 4
//-------------------------------------
    // pend on continue button
    I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
    PrintStateMessage(LSS_PUSHCONTINUEBUTTON,LSS_LOADHEADS);
    if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
        goto exit;
    I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
    if(DeviceTable->ConnectorClamp) //if there is a connector clamp, lower it
    {
        ClearConnectorClamps();
        StageConnectorClamp(STAGECONCLAMP_DOWN);
        error = WaitForConnectorClampDown(100);
        if(error)
        {
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_CONNECTORCLAMP]);
            ActionStatus.Status = error;
            goto exit;
        }
    }
    ActionStatus.State++;       //4

//-------------------------------------
// Step 5
//-------------------------------------
    //calculate angle to move to head to
    PrintStateMessage(LSS_MOVEROTSTAGELOADUNLOAD,LSS_LOADHEADS);
    angle = rad_to_angle(DeviceTable->LoadRadius,&error);
    if(error)
    {
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;      //teminate load
    }
    icnts = angle_to_icts(DeviceTable->referenceangle - angle);
    if((error = ChkErr(Set_Goal(RSTAGE,icnts * DeviceTable->dutorientation) ) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ClearMoveDoneSemaphore();
```

```
    if((error = ChkErr(Do_move(RSTAGE,MOVE_ABSOLUTE)) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        if(error==ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
        goto exit;
    }
    ActionStatus.State++;   //5

//-------------------------------------
// Step 6
//-------------------------------------
    //Wait for Rotary Stage to get into position
    PrintStateMessage(LSS_WAITFORINPOSITION,LSS_LOADHEADS);
    if((error = ChkErr(WaitForRotaryMove(200))) != 0)
    {
        AbortMove(RSTAGE);  //stop dead in our tracks
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ActionStatus.State++;   //6
//-------------------------------------
// Step 7
//-------------------------------------
    // move load ramp to H.S.A.
    PrintStateMessage(LSS_MOVEINRAMPLOADER,LSS_LOADHEADS);
    ClearRampFlags();
    StageLoadHeads(STAGELOADHEAD_RAMPIN);       //spread Fingers
    ActionStatus.State++;   //7

//-------------------------------------
// Step 8
//-------------------------------------
    PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_LOADHEADS);
    if((error = ChkErr(WaitForRampIn(1000))) != 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ActionStatus.State++;   //8


//-------------------------------------
// Step 9
//-------------------------------------
    PrintStateMessage(LSS_REMOVECOMB,LSS_LOADHEADS);
    I2CWriteLEDString((char *)LCDStrings[LCDS_REMOVECOMB]);
    if(( error = ChkErr(WaitForCombInOut(WAITFORCOMBINOUT_IN))) != 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
    ActionStatus.State++;   //9
//-------------------------------------
// Step 10
//-------------------------------------
    //activate x run solanoid
```

```
    PrintStateMessage(LSS_MOVEXSTAGETORUN,LSS_LOADHEADS);
    ClearWaitForLinearRun();
    StageSetXStage(STAGESETXSTAGE_RUN);
    ActionStatus.State++;   //10

//-------------------------------------
// Step 11
//-------------------------------------
    //start spindle motor at load RPM
    PrintStateMessage(LSS_STARTSPINDLE,LSS_LOADHEADS);
    StageSetRpm(DeviceTable->loadrpm);      //set motor RPM to load RPM
    if(DeviceTable->mode == HSA)    //turn on motor in HSA mode only
        StageSetMotor(1);
    ActionStatus.State++;   //11
//-------------------------------------
// Step 12
//-------------------------------------
    //pend on x stage in position
    PrintStateMessage(LSS_WAITFORRUNPOSITION,LSS_LOADHEADS);
    if((error = ChkErr(WaitForLinearRun(2000))) < 0)
    {
        if(error == ACUTRAC_TERMINATE)  //stop button was pushed durring load
        {
            StageSetXStage(STAGESETXSTAGE_HOME);
        }
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //handle error
        goto exit;
    }
    ActionStatus.State++;   //12
//-------------------------------------
// Step 13
//-------------------------------------
    //pend on spindle motor at speed
    PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_LOADHEADS);
    //do the following step for HSA mode only
    if(DeviceTable->mode == HSA)
    {
        error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_ACCEL,DeviceTab
le->loadrpm)));
        if(error)
        {
            if(error == ACUTRAC_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
            ActionStatus.Status = error;
            goto exit;
        }
    }
    ActionStatus.State++;   //13
//-------------------------------------
// Step 14
//-------------------------------------
    //move load ramp into garage
    PrintStateMessage(LSS_MOVERAMPLOADEROUT,LSS_LOADHEADS);
    ClearRampFlags();
    StageLoadHeads(STAGELOADHEAD_RAMPOUT);      //Close Fingers
    ActionStatus.State++;   //14
//-------------------------------------
// Step 15
//-------------------------------------

    //pend on load ramp out
    PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_LOADHEADS);
    if((error = ChkErr(WaitForRampOut(1000))) != 0)
    {
```

```
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ActionStatus.State++;   //15
//do this step for HSAR mode only
    if(DeviceTable->mode == HSAR)
    {
        StageSetMotor(1);   //start motor
        error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_ACCEL,DeviceTab
le->loadrpm)));
        if(error)
        {
            if(error == ACUTRAC_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
            ActionStatus.Status = error;
            goto exit;
        }
    }
//-------------------------------------
// Step 16
//-------------------------------------
    //Change spindle motor speed to test RPM
    PrintStateMessage(LSS_SETSPINDLETOTESTRPM,LSS_LOADHEADS);
    StageSetRpm(DeviceTable->testrpm);      //set motor RPM to load RPM
    ActionStatus.State++;   //16
//-------------------------------------
// Step 17
//-------------------------------------
    //pend on motor at speed
    PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_LOADHEADS);
    error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(((DeviceTable->testrpm-DeviceTable->l
oadrpm) > 0)?ACUTRAC_ACCELTIME_ACCEL:ACUTRAC_ACCELTIME_DECEL,abs(DeviceTable->testrpm-DeviceTabl
e->loadrpm))));
    if(error)
    {
        if(error == ACUTRAC_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }
    ActionStatus.State++;   //17
//-------------------------------------
// Step 18
//-------------------------------------
    //move rotary stage to starting track
    PrintStateMessage(LSS_MOVEHEADTOINITTRACK,LSS_LOADHEADS);
    ClearMoveDoneSemaphore();
    if((error = AcutracSEEKTRACK(DeviceTable->inittrack)) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        if(error = ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
        goto exit;
    }
    ActionStatus.State++;   //18
//-------------------------------------
// Step 19
//-------------------------------------
    //Wait for Rotary Stage to get into position

    PrintStateMessage(LSS_WAITFORINPOSITION,LSS_LOADHEADS);
    if((error = ChkErr(WaitForRotaryMove(100))) != 0)
```

```
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        AbortMove(RSTAGE);
        goto exit;
    }

    //-------------------------------------------------------
    // Terminate START
    //-------------------------------------------------------
exit:
    DisplayMode=0;          //Enable RPM display to front panel
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_YES;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}


//*********************************************************
//
// The stop routine uses the state of how far Start got to
// pick a starting point to do the stop.  There are no breaks
// in the switch statement so that everything that follows a
// case statement chosen will be executed
//
//-------------------------------------------------------
//
// Stop (Head Unload Sequence)
//
//  19. Move Rotary stage to unload position
//  18. Pend on Rotary Stage in Position
//  17. Change Spindle motor speed to unload RPM
//  16. Pend on motor at speed
//  15. Move Load Ramp to HSA
//  14. Pend on Load Ramp in
//  13. Stop Spindle Motor
//  12.
//  11. Activate X Home solanoid
//  10. Pend on X stage home
//   9. Prompt for Comb Install
//   8. Pend on Continue Button
//   7. Move Load Ramp Into Garage
//   6. Pend on Ramp Out
//   5. Release Vacuum Chuck
//   4. Prompt for D.U.T. removal
//   3. Pend On Continue Button
//
//-------------------------------------------------------
// This function gets executed as a TASK
//
//*********************************************************

static void HSAStop(void)
{
    /*************************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
```

```
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    ************************************************************************/
    double angle;
    long icnts;
    int error;

    ActionStatus.Function = ACUTRAC_ACTION_STOP;
    ActionStatus.Status = 0;
    DisplayMode=3;          //disable RPM display to front panel
    WaitForDisplayCycle->Pend();
    I2CWriteLEDString((char *)LCDStrings[LCDS_UNLOADHEADS]);
    switch(ActionStatus.State)
    {
        case 18:    //move to load unload radius
            StageSelectHead(-1);    //turn off preamp
            PrintStateMessage(LSS_MOVEHEADTOUNLOADPOS,LSS_UNLOADHEADS);
            ActionStatus.State--;
            angle = rad_to_angle(DeviceTable->UnloadRadius,&error);
            if(error)
            {
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;      //teminate load
            }
            icnts = angle_to_icts(DeviceTable->referenceangle - angle);
            if((error = ChkErr(Set_Goal(RSTAGE,icnts * DeviceTable->dutorientation) ) ) < 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;
            }
            ClearMoveDoneSemaphore();
            if((error = ChkErr(Do_move(RSTAGE,MOVE_ABSOLUTE)) ) < 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                if(error == ACUTRAC_TERMINATE)
                    AbortMove(RSTAGE);
                goto exit;
            }
        case 17:    //pend on rotary stage in position
            PrintStateMessage(LSS_WAITFORINPOSITION,LSS_UNLOADHEADS);
            ActionStatus.State--;
            if((error = ChkErr(WaitForRotaryMove(200))) != 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                AbortMove(RSTAGE);
                goto exit;
            }
        case 16:    //change spindle to unload rpm
            if(DeviceTable->mode == HSA)
            {
                PrintStateMessage(LSS_SETSPINDLETOUNLOADRPM,LSS_UNLOADHEADS);
                StageSetRpm(DeviceTable->unloadrpm);        //set motor RPM to load RPM
            }
            else if (DeviceTable->mode == HSAR)
            {   //stop motor now
                PrintStateMessage(LSS_STOPSPINDLE,LSS_UNLOADHEADS);
                BrakesDone->SetCount(0);    //just in case, clear semaphore
                StageSetMotor(0);       //stop motor
            }
```

```
            ActionStatus.State--;
        case 15:    //pend on AT SPEED
            if(DeviceTable->mode == HSA)
            {
                PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_UNLOADHEADS);
                error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime (((DeviceTable->unloadrpm
-DeviceTable->testrpm) > 0)?ACUTRAC_ACCELTIME_ACCEL:ACUTRAC_ACCELTIME_DECEL,abs(DeviceTable->tes
trpm-DeviceTable->unloadrpm))) );
                if(error)
                {
                    if(error == ACUTRAC_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
                    AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                    ActionStatus.Status = error;
                    //perform some sort of error recovery
                    goto exit;
                }
            }
            else if (DeviceTable->mode == HSAR) //wait for motor to stop
            {
                PrintStateMessage(LSS_STOPSPINDLE,LSS_UNLOADHEADS);
                if((error = ChkErr(BrakesDone->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_DECEL,3
0000))) ) < 0)  //wait for motor to stop
                {
                    AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_STOPSPINDLE]);
                    ActionStatus.Status = error;
                    //handle error
                    goto exit;
                }
            }
            ActionStatus.State--;
        case 14:
            PrintStateMessage(LSS_MOVEINRAMPLOADER,LSS_UNLOADHEADS);
            ActionStatus.State--;
            ClearRampFlags();
            StageLoadHeads(STAGELOADHEAD_RAMPIN);       //spread Fingers
        case 13:
            PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_UNLOADHEADS);
            ActionStatus.State--;
            if((error = ChkErr(WaitForRampIn(1000))) != 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;
            }
        case 12:
            PrintStateMessage(LSS_MOVEXSTAGEHOME,LSS_UNLOADHEADS);
            ActionStatus.State--;
            ClearWaitForLinearHome();   //clear semaphore
            StageSetXStage(STAGESETXSTAGE_HOME);
        case 11:
            if(DeviceTable->mode == HSA)
            {
                PrintStateMessage(LSS_STOPSPINDLE,LSS_UNLOADHEADS);
                StageSetMotor(0);       //stop motor
            }
            ActionStatus.State--;
        case 10:
            PrintStateMessage(LSS_WAITXSTAGEHOME,LSS_UNLOADHEADS);
            ActionStatus.State--;
            if((error = ChkErr(WaitForLinearHome(2000))) < 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //handle error
                goto exit;
```

```
            }
        case 9:
//          PrintStateMessage(LSS_INSTALLCOMB,LSS_UNLOADHEADS);
//          ActionStatus.State--;
//          if(DeviceTable->mode == HSA)
//          {
//              I2CWriteLEDString((char *)LCDStrings[LCDS_REPLACECOMB]);
//              if(( error = ChkErr(WaitForCombInOut(WAITFORCOMBINOUT_OUT))) != 0)
//              {
//                  AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
//                  ActionStatus.Status = error;
//                  //perform some sort of error recovery
//                  goto exit;
//              }
//              I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
//          }
        case 8:
            PrintStateMessage(LSS_PUSHCONTINUEBUTTON,LSS_UNLOADHEADS);
            ActionStatus.State--;
            I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
        case 7:
            ActionStatus.State--;
            if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                goto exit;
        case 6:
            PrintStateMessage(LSS_MOVERAMPLOADEROUT,LSS_UNLOADHEADS);
            ActionStatus.State--;
            ClearRampFlags();
            StageLoadHeads(STAGELOADHEAD_RAMPOUT);      //close Fingers
        case 5:
            PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_UNLOADHEADS);
            ActionStatus.State--;
            if((error = ChkErr(WaitForRampOut(1000))) != 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;
            }
        case 4:
            if(DeviceTable->ConnectorClamp)
            {
                ClearConnectorClamps();
                StageConnectorClamp(STAGECONCLAMP_UP);
                error = WaitForConnectorClampUp(100);
                if(error)
                {
                    AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_CONNECTORCLAMP]);
                    ActionStatus.Status = error;
                    goto exit;
                }
            }
            PrintStateMessage(LSS_RELEASEVACCHUCK,LSS_UNLOADHEADS);
            ActionStatus.State--;
            StageSuckHeads(0);      //Disable vac chuck
            //heads are off chuck, change paramters
            if((error = ChkErr(DeviceTable->Pid[ACUTRAC_HEADSOFF]->Upload(ACUTRAC_HEADSOFF)) ) <
 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;
            }
        case 3:
            ActionStatus.State--;
```

```
            if(DeviceTable->mode == HSAR)
            {
                if((error = ChkErr(Set_Goal(ROTARY_STAGE,0)) ) != ACUTRAC_SUCCESS)
                {
                    AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                    ActionStatus.Status = error;
                    goto exit;
                }
                if((error = ChkErr(Do_move(ROTARY_STAGE,MOVE_ABSOLUTE))) != ACUTRAC_SUCCESS)
                {
                    AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                    ActionStatus.Status = error;
                    goto exit;
                }
            }
            I2CWriteLEDString((char *)LCDStrings[LCDS_REMOVEDUT]);
            TDelay(200);
            I2CBeepTheBuzzer(1,35);
            I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
        case 2:
            ActionStatus.State--;
        case 1:
            ActionStatus.State--;
        case 0:
            break;
    }
exit:
    DisplayMode = 0;    //enable normal RPM display
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_READY;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

//*****************************************************************************
//
// This function is used to lift the heads off of the disk (for later replacement
//
//*****************************************************************************

static void HSALift(void)
{
    /*************************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    *************************************************************************/
    double angle;
    long icnts;
    int error;

    ActionStatus.Function = ACUTRAC_ACTION_LIFT;
    ActionStatus.Status = 0;
    DisplayMode=3;          //disable RPM display to front panel
    WaitForDisplayCycle->Pend();
    I2CWriteLEDString((char *)LCDStrings[LCDS_LIFTHEADS]);
    //action code goes here ||
    //                      \/
```

```
    //------------------------------
    // move to load/unload radius
    //------------------------------
    PrintStateMessage(LSS_MOVEHEADTOUNLOADPOS,LSS_LIFTHEADS);
    angle = rad_to_angle(DeviceTable->UnloadRadius,&error);
    if(error)
    {
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;      //teminate load
    }
    icnts = angle_to_icts(DeviceTable->referenceangle - angle);
    if((error = ChkErr(Set_Goal(RSTAGE,icnts * DeviceTable->dutorientation) ) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ClearMoveDoneSemaphore();
    if((error = ChkErr(Do_move(RSTAGE,MOVE_ABSOLUTE)) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        if(error == ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
        goto exit;
    }
    //-----------------------------------
    // wait for move to complete
    //-----------------------------------
    PrintStateMessage(LSS_WAITFORINPOSITION,LSS_LIFTHEADS);
    if((error = ChkErr(WaitForRotaryMove(200))) != 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        AbortMove(RSTAGE);
        goto exit;
    }
    //-----------------------------------
    // change spindle to load/unload rpm
    //-----------------------------------
    PrintStateMessage(LSS_SETSPINDLETOUNLOADRPM,LSS_LIFTHEADS);
    StageSetRpm(DeviceTable->unloadrpm);        //set motor RPM to load RPM
    PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_LIFTHEADS);
    error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime (((DeviceTable->unloadrpm-DeviceTable
->testrpm) > 0)?ACUTRAC_ACCELTIME_ACCEL:ACUTRAC_ACCELTIME_DECEL,abs(DeviceTable->testrpm-DeviceT
able->unloadrpm))) );
    if(error)
    {
        if(error == ACUTRAC_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    //-------------------------------------
    // remove head from disk with
    //-------------------------------------
    PrintStateMessage(LSS_MOVEINRAMPLOADER,LSS_LIFTHEADS);
    ClearRampFlags();
    StageLoadHeads(STAGELOADHEAD_RAMPIN);       //spread Fingers
    PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_LIFTHEADS);
```

```
    if((error = ChkErr(WaitForRampIn(1000))) != 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    //                    /\
    //action code above here||
exit:
    DisplayMode = 0;    //enable normal RPM display
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_LIFTED;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

//*************************************************************************
//
// This function is used to put the heads back onto the disk
//
//*************************************************************************

static void HSAReload(void)
{
    /*********************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    *********************************************************************/
    double angle;
    long icnts;
    int error;

    ActionStatus.Function = ACUTRAC_ACTION_RELOAD;
    ActionStatus.Status = 0;
    DisplayMode=3;          //disable RPM display to front panel
    WaitForDisplayCycle->Pend();
    I2CWriteLEDString((char *)LCDStrings[LCDS_LOWERHEADS]);
    //action code goes here ||
    //                     \/

    //---------------------------------
    // Put heads back onto disk
    //---------------------------------

    PrintStateMessage(LSS_MOVERAMPLOADEROUT,LSS_RELOADHEADS);
    ClearRampFlags();
    StageLoadHeads(STAGELOADHEAD_RAMPOUT);      //Close Fingers
    PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_RELOADHEADS);
    if((error = ChkErr(WaitForRampOut(1000))) != 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_RELOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    //-------------------------------------
```

```
    //Change spindle motor speed to test RPM
    //---------------------------------------
    PrintStateMessage(LSS_SETSPINDLETOTESTRPM,LSS_RELOADHEADS);
    StageSetRpm(DeviceTable->testrpm);      //set motor RPM to load RPM
    PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_RELOADHEADS);
    error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(((DeviceTable->testrpm-DeviceTable->l
oadrpm) > 0)?ACUTRAC_ACCELTIME_ACCEL:ACUTRAC_ACCELTIME_DECEL,abs(DeviceTable->testrpm-DeviceTabl
e->loadrpm))));
    if(error)
    {
        if(error == ACUTRAC_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_RELOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }
    //---------------------------------------
    //move rotary stage to current track
    //---------------------------------------
    PrintStateMessage(LSS_MOVEHEADTOINITTRACK,LSS_RELOADHEADS);
    ClearMoveDoneSemaphore();
    if((error = AcutracSEEKTRACK(DeviceTable->currenttrack)) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_RELOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        if(error = ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
        goto exit;
    }
    PrintStateMessage(LSS_WAITFORINPOSITION,LSS_RELOADHEADS);
    if((error = ChkErr(WaitForRotaryMove(100))) != 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_RELOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        AbortMove(RSTAGE);
        goto exit;
    }

    //                      /\
    //action code above here||
exit:
    DisplayMode = 0;    //enable normal RPM display
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_YES;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

//*********************************************************
//
// The reset function gets called to initialize the stages.
// This function records where it is at so that the host can
// keep track of progress if desired.
//
// This function gets executed as a TASK
//
//*********************************************************

static void HSAReset(void)
{
    /*********************************************************************
    ** This function is a TASK
```

```
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    ***************************************************************************/
    long pos,lastpos;
    int loop;
    int error;

    ActionStatus.State = 0;
    ActionStatus.Function = ACUTRAC_ACTION_RESET;
    ActionStatus.Status = 0;
    //-------------------------------------
    // code to do a reset follows
    //-------------------------------------

    DisplayMode=3;          //disable RPM display to front panel
    if(WaitForDisplayCycle->Pend() == EVENT_TERMINATE)
        goto exit;      //terminate process
    I2CWriteLEDString((char *)LCDStrings[LCDS_INITIALIZING]);

    ActionStatus.State++;       //State=1
    /*-----------------------------------------------
    **
    ** check to see if there is a head on the vacume chuck
    **
    **-----------------------------------------------*/
    StageSelectHead(-1);    //turn off preamp
    StageSuckHeads(1);      //turn on vac chuck
    if(TDelay(50) == EVENT_TERMINATE)               //wait just a bit, just in case
        goto exit;      //terminate process
    error = I2CGetStageStatus();    //see if we get SUCK
    if(error == EVENT_TERMINATE)
        goto exit;
    PrintStateMessage(LSS_CHECKCONDITION,LSS_INITSTAGE);
    if(!(error & FPCOMMAND_VAC))
    {
        //-----------------------------------------------------------
        // Check to see if the comb is in the comb corral
        //-----------------------------------------------------------
        if((error & FPCOMMAND_CORAL) == 0)
        {
            //OK, it looks like the head might be loaded
            //now check to see if the X stage is in the run position
            if(GetXStagePosition())
            {
                if(SystemStatus.spin.stat.motor)    //is motor running?
                {
                    if(AcutracGotoLoadUnloadRadius(ACUTRAC_LOADRADIUS) == ACUTRAC_TERMINATE)
        //goto unload radius
                        goto exit;  //task is terminated, just quit
                    if(AcutracUnloadHeads() == ACUTRAC_TERMINATE)
                        goto exit;                  //spread fingers
                    StageSetXStage(STAGESETXSTAGE_HOME);    //move stage to Home position
                    I2CWriteLEDString((char *)LCDStrings[LCDS_REPLACECOMB]);
                    if(WaitForCombInOut(WAITFORCOMBINOUT_OUT) == EVENT_TERMINATE)
                        goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
                    if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                        goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
                    StageSetMotor(0);       //turn off motor
                }
```

```
            else    //no motor is not spining
            {
                //check if heads are loaded?
                if((error & FPCOMMAND_RAMP_OUT))    //are heads loaded
                {
                    //this is very BAD
                    I2CWriteLEDString((char *)LCDStrings[LCDS_CANNOTUNLOAD]);
                    if(TDelay(60) == EVENT_TERMINATE) goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_WHENREADY]);
                    if(TDelay(60) == EVENT_TERMINATE) goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
                    if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                        goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
                }
                else
                {
                    StageSetXStage(STAGESETXSTAGE_HOME);    //move stage to Home position
                    I2CWriteLEDString((char *)LCDStrings[LCDS_REPLACECOMB]);
                    if(WaitForCombInOut(WAITFORCOMBINOUT_OUT) == EVENT_TERMINATE)
                        goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
                    if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                        goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
                    StageSetMotor(0);       //turn off motor
                }
            }
        }
        else
        {
            //stage is in the home position
            //heads are not loaded on the disk, fingers better
            //be spread, so prompt for replacement of comb
            I2CWriteLEDString((char *)LCDStrings[LCDS_REPLACECOMB]);
            if(WaitForCombInOut(WAITFORCOMBINOUT_OUT) == EVENT_TERMINATE)
                goto exit;
            I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
            if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                goto exit;
            I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
            StageSetMotor(0);       //turn off motor
        }
    }
    //-----------------------------------------------------
    // this is where we need to end up if the comb has not
    // been removed yet
    //-----------------------------------------------------
    PrintStateMessage(LSS_MOVEINRAMPLOADER,LSS_INITSTAGE);
    loop = I2CGetStageStatus();
    if(loop & FPCOMMAND_RAMP_IN)
    {
        ClearRampFlags();
        StageLoadHeads(STAGELOADHEAD_RAMPOUT);      //close Fingers
        if((error = WaitForRampOut(1000)) != 0)
        {
            goto exit;
        }
    }
    StageConnectorClamp(STAGECONCLAMP_UP);
    StageSuckHeads(0);      //close up vac on head chuck
    I2CWriteLEDString((char *)LCDStrings[LCDS_REMOVEDUT]);  //promt user
    if(TDelay(200) == EVENT_TERMINATE)
        goto exit;
    if(I2CBeepTheBuzzer(1,35) == EVENT_TERMINATE)
        goto exit;
```

```
        I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
        if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
            goto exit;
        I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
    }
    else if(GetXStagePosition())    //is stange in run position
    {
        StageSetXStage(STAGESETXSTAGE_HOME);    //move stage to Home position
    }
    if(SystemStatus.spin.stat.motor)
    {
        BrakesDone->SetCount(0);    //just in case, clear semaphore
        StageSetMotor(0);
        if((error = ChkErr(BrakesDone->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_DECEL,30000)))
) < 0)  //wait for motor to stop
        {
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_STOPSPINDLE]);
            ActionStatus.Status = error;
            //handle error
            goto exit;
        }
    }
    StageSuckHeads(0);
    StageLoadHeads(STAGELOADHEAD_RAMPOUT);      //close Fingers
    ActionStatus.State++;       //State = 9
    PrintStateMessage(LSS_CALIBRATEROTARYSTAGE,LSS_INITSTAGE);
    if((error = ChkErr(DeviceTable->Pid[ACUTRAC_HEADSOFF]->Upload(ACUTRAC_HEADSOFF)) ) < 0)
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    if((error = ChkErr(EnableServoFunctions(ROTARY_STAGE,ENABLE_PID,1l))) < 0)
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        goto exit;
    }
    if((error = ChkErr(EnableServoFunctions(ROTARY_STAGE,ENABLE_INTEGRATOR,1l))) < 0)
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        goto exit;
    }
    TDelay(50);     //wait for servo to stabalize
    if(!Calibrated)
    {
        SetMicroEAutoComp(MICROE_AUTOCOMP_ON);  //turn on auto comp
        I2CWriteLEDString((char *)LCDStrings[LCDS_CALIBRATING]);
        Calibrated = 1;
        ClearWaitForRotaryCalibrate();
        if((error = ChkErr(Do_Calibrate(ROTARY_STAGE))) < 0)
        {
            I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
            ActionStatus.Status = error;
            goto exit;
        }
        PrintStateMessage(LSS_CALIBRATEROTARYSTAGE,LSS_INITSTAGE);
        if(WaitForRotaryCalibrate(5000) == EVENT_TERMINATE) //wait for callibrate to finish
        {
            //Ok, we are in the middle of doing something with the servo, probably, so
```

```
            //send an abort move there
            AbortMove(RSTAGE);
            goto exit;
        }
    }
    ActionStatus.State++;           // State = 10
    if((error = ChkErr(Get_Calibration(ROTARY_STAGE,&pos))) < 0)    //get callibration distance
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        ActionStatus.Status = error;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        goto exit;
    }
    //calculate out rotary calibration factor
    // The Angle is Supposed To BE 39.5596 degrees or 0.690445 radians
    //-------------------------------------------------------------
    PrintStateMessage(LSS_BUILDTRACKLUT,LSS_INITSTAGE);
    DeviceTable->CountsPerRadian = AcutracGETCORRECTION();
    if(DeviceTable->mode == HSA)
    {
        if(AcutracGotoLoadUnloadRadius(ACUTRAC_LOADRADIUS) == ACUTRAC_TERMINATE)            //go
to unload radius
            goto exit;  //task is terminated, just quit
    }
    else if(DeviceTable->mode == HSAR)
    {
        if((error = ChkErr(Set_Goal(ROTARY_STAGE,0l)) ) != ACUTRAC_SUCCESS)
        {
            ActionStatus.Status = error;
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
            goto exit;
        }
        if((error = ChkErr(Do_move(ROTARY_STAGE,MOVE_ABSOLUTE))) != ACUTRAC_SUCCESS)
        {
            ActionStatus.Status = error;
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
            goto exit;
        }
    }
    StageSetDirection(DeviceTable->rotation);   //set spindle direction (this might have failed
earlier)
    SetMicroEAutoComp(MICROE_AUTOCOMP_OFF); //turn off autocomp before exit

exit:
    DisplayMode = 0;    //restore display mode back to normal
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_READY;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

//-------------------------------------------------------------------
//
// Spinstand functions for HGA Heads
//
//-------------------------------------------------------------------

//-------------------------------------------------------------------------
// AcutracSTART
//
//  Put device in the state when it is ready for read/write
//
//
```

```
// Sequence of events for HGA spinstand head load
//
//  1.  Promt User to Install Device
//  2.  Spin Up Motor
//  3.  Move Rotary Motor to Load/Unload Position
//  4.  Move X Stage to Run Position
//
//---------------------------------------------------------------------------

static int HGACheckRampFlags(int which,int one,int timeout)
{
    //-------------------------------------------------------
    // "which" refers to orientation (ie,FRONT, BACK)
    // "one"   refers to position, IN, OUT)
    // "timeout"-how long to wait in system ticks (10mS, I think)
    // returns 0 or 1.  1 is good, 0 is mismatch, less than zero is
    // an error
    //-------------------------------------------------------
    int v;

    if(which == DUT_BACK)
        v = RampInFlag->Pend(timeout);
    else
        v = RampOutFlag->Pend(timeout);
    if(v >= 0)
    {
        if(v==one) v = 1;else v = 0;    //return 1 if equal, 0 if not
    }
    return v;
}

static void HGAClearRampFlags(void)
{
    RampInFlag->SetCount(0);
    RampOutFlag->SetCount(0);
}

static int HGACheckLifterFlags(int which,int one,int timeout)
{
    int v;

    if(which == DUT_BACK)
        v = ClampUpFlag->Pend(timeout);
    else
        v = ClampDownFlag->Pend(timeout);
    if(v >= 0)
    {
        if(v==one) v = 1;else v = 0;    //return 1 if equal, 0 if not
    }
    return v;
}

static void HGAClearLifterFlags(void)
{
    ClampDownFlag->SetCount(0);
    ClampUpFlag->SetCount(0);
}

//*********************************************************
//
// This function needs to record where it is at each step
// of the way.  In this way, the host computer can get the
// status of the progress on each of these functions.
// also, where start left off at can be used when the user
// goes to do a stop.
//
```

```
// This function gets executed as a TASK
//
//*******************************************************


static void HGAStart(void)
{
    /*************************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    *************************************************************************/
    double angle;
    long icnts;
    int error;

    ActionStatus.State = 0;
    ActionStatus.Function = ACUTRAC_ACTION_START;
    ActionStatus.Status = 0;
    //-----------------------------------------------
    // Code for doing start
    //-----------------------------------------------
//-------------------------------------
// Step 1 Activate Vacume chuck
//-------------------------------------
    DisplayMode=3;          //disable RPM display to front panel
    WaitForDisplayCycle->Pend();
    I2CWriteLEDString((char *)LCDStrings[LCDS_LOADHEADS]);
    PrintStateMessage(LSS_INSTALLDUT,LSS_LOADHEADS);
    ActionStatus.State++;   //1

    ClearChuckSuck();
    StageSuckHeads(1);      //open up vac on head chuck
    PrintStateMessage(LSS_INSTALLDUT,LSS_LOADHEADS);
    ActionStatus.State++;   //2
//-------------------------------------
// Step 2 Display Insert DUT message
//-------------------------------------
    I2CWriteLEDString((char *)LCDStrings[LCDS_INSERTDUT]);
    PrintStateMessage(LSS_WAITFORVACSENSE,LSS_LOADHEADS);
    ActionStatus.State++;   //3
//-------------------------------------
// Step 3 Pned on Vacume chuck semaphore
// and upload heads on tuning parameters
//-------------------------------------
    if(WaitForChuckSuck(WAITFORCHUCKSUCK_SUCK) == EVENT_TERMINATE)  //wait for head assembly
        goto exit;
    //heads are on chuck, change parameters
    if((error = ChkErr(DeviceTable->Pid[ACUTRAC_HEADSON]->Upload(ACUTRAC_HEADSON)) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ActionStatus.State++;   //4

//-------------------------------------
// Step 3
//-------------------------------------
    // pend on continue button
```

```
//  TDelay(300);        //Wait 3 Seconds
    I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
    PrintStateMessage(LSS_PUSHCONTINUEBUTTON,LSS_LOADHEADS);
    if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
        goto exit;
    ActionStatus.State++;       //5
//-------------------------------------
// Step 4
//-------------------------------------
    //calculate angle to move to head to
    PrintStateMessage(LSS_MOVEROTSTAGELOADUNLOAD,LSS_LOADHEADS);
    angle = rad_to_angle(DeviceTable->LoadRadius,&error);
    if(error)
    {
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;      //teminate load
    }
    icnts = angle_to_icts(DeviceTable->referenceangle - angle);
    if((error = ChkErr(Set_Goal(RSTAGE,icnts * DeviceTable->dutorientation) ) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ClearMoveDoneSemaphore();
    if((error = ChkErr(Do_move(RSTAGE,MOVE_ABSOLUTE)) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        if(error==ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
        goto exit;
    }
    ActionStatus.State++;   //6

//-------------------------------------
// Step 5
//-------------------------------------
    //Wait for Rotary Stage to get into position
    PrintStateMessage(LSS_WAITFORINPOSITION,LSS_LOADHEADS);
    if((error = ChkErr(WaitForRotaryMove(200))) != 0)
    {
        AbortMove(RSTAGE);  //stop dead in our tracks
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ActionStatus.State++;   //7
//-------------------------------------
// Step 6
//-------------------------------------
    //start spindle motor at load RPM
    PrintStateMessage(LSS_STARTSPINDLE,LSS_LOADHEADS);
    StageSetRpm(DeviceTable->loadrpm);      //set motor RPM to load RPM
    StageSetMotor(1);
    ActionStatus.State++;   //8
//-------------------------------------
// _step_7_
//
// raise lifters
//-------------------------------------
    HGAClearRampFlags();
```

```
    StageLoadHeads(STAGELOADHEAD_RAMPOUT);  //move lifters to engage
    PrintStateMessage(LSS_MOVEINRAMPLOADER,LSS_LOADHEADS);
    if((error = ChkErr(HGACheckRampFlags(DeviceTable->dutorientation,STAGELOADHEAD_RAMPOUT,500))
) <=0)
    {
        if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        if(error == 0) error = ACUTRAC_SERVO_UNKNOWN;   //mismatch, unexpected state
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }
    HGAClearLifterFlags();
    StageConnectorClamp(STAGECONCLAMP_DOWN);
    PrintStateMessage(LSS_LIFTERSUP,LSS_LOADHEADS);
    if((error = ChkErr(HGACheckLifterFlags(DeviceTable->dutorientation,STAGECONCLAMP_DOWN,500)))
 <= 0)
    {
        if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        if(error == 0) error = ACUTRAC_SERVO_UNKNOWN;   //mismatch, unexpected state
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }

//-------------------------------------
// Step 8
//-------------------------------------
    //activate x run solanoid
    PrintStateMessage(LSS_MOVEXSTAGETORUN,LSS_LOADHEADS);
    ClearWaitForLinearRun();
    StageSetXStage(STAGESETXSTAGE_RUN);
    ActionStatus.State++;   //9

//-------------------------------------
// Step 8
//-------------------------------------
    //pend on spindle motor at speed
    PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_LOADHEADS);

    error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_ACCEL,DeviceTable->
loadrpm)));
    if(error)
    {
        if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }
    ActionStatus.State++;   //10

//-------------------------------------
// Step 9
//-------------------------------------
    //pend on x stage in position
    PrintStateMessage(LSS_WAITFORRUNPOSITION,LSS_LOADHEADS);
    if((error = ChkErr(WaitForLinearRun(2000))) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //handle error
        goto exit;
    }
    ActionStatus.State++;   //11
//-------------------------------------
// Step 10, Drop heads on disk
//-------------------------------------
```

```
    PrintStateMessage(LSS_LOADHEADSONTODISK,LSS_LOADHEADS);
    HGAClearRampFlags();
    StageLoadHeads(STAGELOADHEAD_RAMPIN);   //load heads
    if((error = ChkErr(HGACheckRampFlags(DeviceTable->dutorientation,STAGELOADHEAD_RAMPIN,500)))
 <=0)
    {
        if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        if(error == 0) error = ACUTRAC_SERVO_UNKNOWN;   //mismatch, unexpected state
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }

    ActionStatus.State++;   //12
//-------------------------------------
// Step 11
//-------------------------------------
    //Change spindle motor speed to test RPM
    PrintStateMessage(LSS_SETSPINDLETOTESTRPM,LSS_LOADHEADS);
    StageSetRpm(DeviceTable->testrpm);      //set motor RPM to load RPM
    ActionStatus.State++;   //13
//-------------------------------------
// Step 12
//-------------------------------------
    //pend on motor at speed
    PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_LOADHEADS);
    error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(((DeviceTable->testrpm-DeviceTable->l
oadrpm) > 0)?ACUTRAC_ACCELTIME_ACCEL:ACUTRAC_ACCELTIME_DECEL,abs(DeviceTable->testrpm-DeviceTabl
e->loadrpm))));
    if(error)
    {
        if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }
    ActionStatus.State++;   //14
//-------------------------------------
// Step 13
//-------------------------------------
    //move rotary stage to starting track
    PrintStateMessage(LSS_MOVEHEADTOINITTRACK,LSS_LOADHEADS);
    ClearMoveDoneSemaphore();
    if((error = AcutracSEEKTRACK(DeviceTable->inittrack)) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        if(error = ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
        goto exit;
    }
    ActionStatus.State++;   //15
//-------------------------------------
// Step 14
//-------------------------------------
    //Wait for Rotary Stage to get into position

    PrintStateMessage(LSS_WAITFORINPOSITION,LSS_LOADHEADS);
    if((error = ChkErr(WaitForRotaryMove(100))) != 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        AbortMove(RSTAGE);
        goto exit;
```

```
    }

//-------------------------------------------------------
// Terminate START
//-------------------------------------------------------

exit:
    DisplayMode=0;          //Enable RPM display to front panel
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_YES;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

static void HGAStop(void)
{
    /*************************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    *************************************************************************/
    double angle;
    long icnts;
    int error;

    ActionStatus.Function = ACUTRAC_ACTION_STOP;
    ActionStatus.Status = 0;
    DisplayMode=3;          //disable RPM display to front panel
    WaitForDisplayCycle->Pend();
    I2CWriteLEDString((char *)LCDStrings[LCDS_UNLOADHEADS]);
    //-------------------------------------------------
    // Code for doing Stop
    //-------------------------------------------------
    switch(ActionStatus.State)
    {
        case 15:
            //goto load/unload radius
            PrintStateMessage(LSS_MOVEHEADTOUNLOADPOS,LSS_UNLOADHEADS);
            ActionStatus.State--;
            angle = rad_to_angle(DeviceTable->UnloadRadius,&error);
            if(error)
            {
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;      //teminate load
            }
            icnts = angle_to_icts(DeviceTable->referenceangle - angle);
            if((error = ChkErr(Set_Goal(RSTAGE,icnts * DeviceTable->dutorientation) ) ) < 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;
            }
            ClearMoveDoneSemaphore();
            if((error = ChkErr(Do_move(RSTAGE,MOVE_ABSOLUTE)) ) < 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
```

```
                ActionStatus.Status = error;
                //perform some sort of error recovery
                if(error == ACUTRAC_TERMINATE)
                    AbortMove(RSTAGE);
                goto exit;
            }
        case 14:
            //wait for move complete
            PrintStateMessage(LSS_WAITFORINPOSITION,LSS_UNLOADHEADS);
            ActionStatus.State--;
            if((error = ChkErr(WaitForRotaryMove(200))) != 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                AbortMove(RSTAGE);
                goto exit;
            }
        case 13:
            //set motor to unload rpm
            PrintStateMessage(LSS_SETSPINDLETOUNLOADRPM,LSS_UNLOADHEADS);
            ActionStatus.State--;
            StageSetRpm(DeviceTable->unloadrpm);        //set motor RPM to load RPM
        case 12:
            //wait for motor at speed
            PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_UNLOADHEADS);
            ActionStatus.State--;
            error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime (((DeviceTable->unloadrpm-Dev
iceTable->testrpm) > 0)?ACUTRAC_ACCELTIME_ACCEL:ACUTRAC_ACCELTIME_DECEL,abs(DeviceTable->testrpm
-DeviceTable->unloadrpm))) );
            if(error)
            {
                if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;
            }
        case 11:
            PrintStateMessage(LSS_MOVERAMPLOADEROUT,LSS_UNLOADHEADS);
            HGAClearRampFlags();
            StageLoadHeads(STAGELOADHEAD_RAMPOUT);
            if((error = ChkErr(HGACheckRampFlags(DeviceTable->dutorientation,STAGELOADHEAD_RAMPO
UT,1000))) <= 0)
            {
                if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
                if(error == 0) error = ACUTRAC_SERVO_UNKNOWN;   //mismatch
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
            }
        case 10:
            //move x stage home
            PrintStateMessage(LSS_MOVEXSTAGEHOME,LSS_UNLOADHEADS);
            ActionStatus.State--;
            ClearWaitForLinearHome();   //clear semaphore
            StageSetXStage(STAGESETXSTAGE_HOME);
        case 9:
            //wait for xstage home
            PrintStateMessage(LSS_WAITXSTAGEHOME,LSS_UNLOADHEADS);
            ActionStatus.State--;
            if((error = ChkErr(WaitForLinearHome(2000))) < 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                //handle error
                goto exit;
```

```
            }
        case 8:
            //turn off motor
            PrintStateMessage(LSS_STOPSPINDLE,LSS_UNLOADHEADS);
            BrakesDone->SetCount(0);    //just in case, clear semaphore
            StageSetMotor(0);       //stop motor
            ActionStatus.State--;
        case 7:
            HGAClearLifterFlags();
            StageConnectorClamp(STAGECONCLAMP_UP);
            PrintStateMessage(LSS_LIFTERSDOWN,LSS_UNLOADHEADS);
            if((error = ChkErr(HGACheckLifterFlags(DeviceTable->dutorientation,STAGECONCLAMP_UP,
1000))) <= 0)
            {
                if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
                if(error == 0) error = ACUTRAC_SERVO_UNKNOWN;   //mismatched
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_UNLOADHEADS]);
                ActionStatus.Status = error;
                goto exit;
            }
        case 6:
            //wait for motor to stop
            if((error = ChkErr(BrakesDone->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_DECEL,Devic
eTable->loadrpm))) ) < 0)   //wait for motor to stop
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_STOPSPINDLE]);
                ActionStatus.Status = error;
                //handle error
                goto exit;
            }
        case 5:
            PrintStateMessage(LSS_RELEASEVACCHUCK,LSS_UNLOADHEADS);
            ActionStatus.State--;
            StageSuckHeads(0);      //Disable vac chuck
            //heads are off chuck, change paramters
            if((error = ChkErr(DeviceTable->Pid[ACUTRAC_HEADSOFF]->Upload(ACUTRAC_HEADSOFF)) ) <
 0)
            {
                AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
                ActionStatus.Status = error;
                //perform some sort of error recovery
                goto exit;
            }
        case 4:
            ActionStatus.State--;
            I2CWriteLEDString((char *)LCDStrings[LCDS_REMOVEDUT]);
            TDelay(200);
            I2CBeepTheBuzzer(1,35);
            I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
        case 3:
            PrintStateMessage(LSS_PUSHCONTINUEBUTTON,LSS_UNLOADHEADS);
            ActionStatus.State--;
            if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                goto exit;
        case 2:
            StageLoadHeads(STAGELOADHEAD_RAMPIN);
            ActionStatus.State--;
            break;
    }   //end of switch ActionStatus.State
exit:
    DisplayMode = 0;    //enable normal RPM display
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
```

```
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_READY;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

//---------------------------------------------------------------------
// This function is used to remove the heads from the disk
//---------------------------------------------------------------------

static void HGALift(void)
{
    /*********************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    *********************************************************************/
    double angle;
    long icnts;
    int error;

    ActionStatus.Function = ACUTRAC_ACTION_LIFT;
    ActionStatus.Status = 0;
    DisplayMode=3;          //disable RPM display to front panel
    WaitForDisplayCycle->Pend();
    I2CWriteLEDString((char *)LCDStrings[LCDS_LIFTHEADS]);
    //action code goes here ||
    //                      \/

    //-------------------------------
    // move to load/unload radius
    //-------------------------------
    PrintStateMessage(LSS_MOVEHEADTOUNLOADPOS,LSS_LIFTHEADS);
    angle = rad_to_angle(DeviceTable->UnloadRadius,&error);
    if(error)
    {
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;      //teminate load
    }
    icnts = angle_to_icts(DeviceTable->referenceangle - angle);
    if((error = ChkErr(Set_Goal(RSTAGE,icnts * DeviceTable->dutorientation) ) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    ClearMoveDoneSemaphore();
    if((error = ChkErr(Do_move(RSTAGE,MOVE_ABSOLUTE)) ) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        if(error == ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
        goto exit;
    }
    //-----------------------------------
    // wait for move to complete
    //-----------------------------------
    PrintStateMessage(LSS_WAITFORINPOSITION,LSS_LIFTHEADS);
    if((error = ChkErr(WaitForRotaryMove(200))) != 0)
```

```
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        AbortMove(RSTAGE);
        goto exit;
    }
    //-------------------------------------
    // change spindle to load/unload rpm
    //-------------------------------------
    PrintStateMessage(LSS_SETSPINDLETOUNLOADRPM,LSS_LIFTHEADS);
    StageSetRpm(DeviceTable->unloadrpm);        //set motor RPM to load RPM
    PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_LIFTHEADS);
    error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime (((DeviceTable->unloadrpm-DeviceTable
->testrpm) > 0)?ACUTRAC_ACCELTIME_ACCEL:ACUTRAC_ACCELTIME_DECEL,abs(DeviceTable->testrpm-DeviceT
able->unloadrpm))) );
    if(error)
    {
        if(error == ACUTRAC_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LIFTHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    //-------------------------------------
    // remove head from disk with
    //-------------------------------------
    PrintStateMessage(LSS_MOVEINRAMPLOADER,LSS_LIFTHEADS);
    StageLoadHeads(STAGELOADHEAD_RAMPOUT);      //spread Fingers
    //                    /\
    //action code above here||
exit:
    DisplayMode = 0;    //enable normal RPM display
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_LIFTED;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

//---------------------------------------------------------------------
// This function is used to put the heads back onto the disk
//---------------------------------------------------------------------

static void HGAReload(void)
{
    /*********************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    *********************************************************************/
    double angle;
    long icnts;
    int error;

    ActionStatus.Function = ACUTRAC_ACTION_RELOAD;
    ActionStatus.Status = 0;
    DisplayMode=3;          //disable RPM display to front panel
    WaitForDisplayCycle->Pend();
```

```
    I2CWriteLEDString((char *)LCDStrings[LCDS_LOWERHEADS]);
    //action code goes here ||
    //                      \/

    //---------------------------------
    // Put heads back onto disk
    //---------------------------------

    PrintStateMessage(LSS_MOVERAMPLOADEROUT,LSS_RELOADHEADS);
    StageLoadHeads(STAGELOADHEAD_RAMPIN);       //Close Fingers
    PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_RELOADHEADS);
    if((error = ChkErr(HGACheckRampFlags(DeviceTable->dutorientation,STAGELOADHEAD_RAMPIN,500)))
 <=0)
    {
        if(error == EVENT_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        if(error == 0) error = ACUTRAC_SERVO_UNKNOWN;   //mismatch, unexpected state
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_LOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }
    //---------------------------------------
    //Change spindle motor speed to test RPM
    //---------------------------------------
    PrintStateMessage(LSS_SETSPINDLETOTESTRPM,LSS_RELOADHEADS);
    StageSetRpm(DeviceTable->testrpm);      //set motor RPM to load RPM
    PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_RELOADHEADS);
    error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(((DeviceTable->testrpm-DeviceTable->l
oadrpm) > 0)?ACUTRAC_ACCELTIME_ACCEL:ACUTRAC_ACCELTIME_DECEL,abs(DeviceTable->testrpm-DeviceTabl
e->loadrpm))));
    if(error)
    {
        if(error == ACUTRAC_TIMEOUT) error = ACUTRAC_SPINDLETIMEOUT;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_RELOADHEADS]);
        ActionStatus.Status = error;
        goto exit;
    }
    //---------------------------------------
    //move rotary stage to current track
    //---------------------------------------
    PrintStateMessage(LSS_MOVEHEADTOINITTRACK,LSS_RELOADHEADS);
    ClearMoveDoneSemaphore();
    if((error = AcutracSEEKTRACK(DeviceTable->currenttrack)) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_RELOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        if(error = ACUTRAC_TERMINATE)
            AbortMove(RSTAGE);
        goto exit;
    }
    PrintStateMessage(LSS_WAITFORINPOSITION,LSS_RELOADHEADS);
    if((error = ChkErr(WaitForRotaryMove(100))) != 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_RELOADHEADS]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        AbortMove(RSTAGE);
        goto exit;
    }

    //                      /\
    //action code above here||
exit:
    DisplayMode = 0;    //enable normal RPM display
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
```

```
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_YES;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

static void HGAReset(void)
{
    /*********************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:various
    ** Posting Functions:various
    **
    ** This task does not continue to run.  When it has completed its
    ** operation, it terminates itself.  This function can also be
    ** terminated by outside forces (all pending semaphores
    *********************************************************************/
    int error;
    long pos,lastpos;
    int loop;

    ActionStatus.State = 0;
    ActionStatus.Function = ACUTRAC_ACTION_RESET;
    ActionStatus.Status = 0;
    //-----------------------------------------------------------
    // Code to do reset follows
    //-----------------------------------------------------------
    DisplayMode = 3;    //disable RPM display to front panel
    if(WaitForDisplayCycle->Pend() == EVENT_TERMINATE)
        goto exit;      //terminate process
    I2CWriteLEDString((char *)LCDStrings[LCDS_INITIALIZING]);

    ActionStatus.State++;   //State = 1;

    /*-----------------------------------------------
    **
    ** check to see if there is a head on the vacume chuck
    **
    ***********************************************
    **
    ** this function is going to be complicated just a
    ** little more than the other.  It matters if we
    ** are front/back because we need to look at different
    ** sensors for each case.  The sensors are much
    ** more limited on this system for many reasons.
    **
    **-----------------------------------------------*/
    StageSelectHead(-1);    //turn off preamp
    StageSuckHeads(1);      //turn on vac chuck
    if(TDelay(30) == EVENT_TERMINATE)               //wait just a bit, just in case
        goto exit;      //terminate process
    error = I2CGetStageStatus();    //see if we get SUCK
    if(error == EVENT_TERMINATE)
        goto exit;
    PrintStateMessage(LSS_CHECKCONDITION,LSS_INITSTAGE);
    if(!(error & FPCOMMAND_VAC))
    {
        //-------------------------------------------------------------
        // Check to see if the comb is in the comb corral
        //-------------------------------------------------------------

        //OK, it looks like the head might be loaded
        //now check to see if the X stage is in the run position
        if(GetXStagePosition())
```

```
        {
            if(SystemStatus.spin.stat.motor)    //is motor running?
            {
                if(AcutracGotoLoadUnloadRadius(ACUTRAC_LOADRADIUS) == ACUTRAC_TERMINATE)
    //goto unload radius
                    goto exit;  //task is terminated, just quit
                HGAClearRampFlags();
                StageLoadHeads(STAGELOADHEAD_RAMPOUT);  //move lift finger in
                if((error = HGACheckRampFlags(DeviceTable->dutorientation,STAGELOADHEAD_RAMPOUT,
1000)) == EVENT_TERMINATE)
                    goto exit;
                StageSetXStage(STAGESETXSTAGE_HOME);    //move stage to Home position

                I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
                StageSetMotor(0);       //turn off motor
            }
            else    //no motor is not spining
            {
                //check if heads are loaded?
                if(((error & FPCOMMAND_RAMP_OUT) == 0) || ((error & FPCOMMAND_RAMP_IN) == 0))
//are heads loaded
                {
                    //this is very BAD
                    I2CWriteLEDString((char *)LCDStrings[LCDS_CANNOTUNLOAD]);
                    if(TDelay(200) == EVENT_TERMINATE) goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_WHENREADY]);
                    if(TDelay(200) == EVENT_TERMINATE) goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
                    if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                        goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
                    StageSetXStage(STAGESETXSTAGE_HOME);    //move stage to Home position
                }
                else
                {
                    StageSetXStage(STAGESETXSTAGE_HOME);    //move stage to Home position
                    I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
                    if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                        goto exit;
                    I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
                    StageSetMotor(0);       //turn off motor
                }
            }
        }
        else
        {
            //stage is in the home position
            //heads are not loaded on the disk, fingers better
            //be spread, so prompt for replacement of comb
            I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
            if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
                goto exit;
            I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
            StageSetMotor(0);       //turn off motor
        }
        //-----------------------------------------------------
        // this is where we need to end up if the comb has not
        // been removed yet
        //-----------------------------------------------------
        StageSuckHeads(0);      //close up vac on head chuck
        I2CWriteLEDString((char *)LCDStrings[LCDS_REMOVEDUT]);  //promt user
        if(TDelay(200) == EVENT_TERMINATE)
            goto exit;
        if(I2CBeepTheBuzzer(1,35) == EVENT_TERMINATE)
            goto exit;
        I2CWriteLEDString((char *)LCDStrings[LCDS_CONTINUE]);
```

```
        if(WaitForContinueButtonPushed() == EVENT_TERMINATE)
            goto exit;
        I2CWriteLEDString((char *)LCDStrings[LCDS_BLANK]);
    }
    else if(GetXStagePosition())    //is stange in run position
    {
        StageSetXStage(STAGESETXSTAGE_HOME);    //move stage to Home position
    }
    if(SystemStatus.spin.stat.motor)
    {
        BrakesDone->SetCount(0);    //just in case, clear semaphore
        StageSetMotor(0);
        if((error = ChkErr(BrakesDone->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_DECEL,30000)))
) < 0)  //wait for motor to stop
        {
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_STOPSPINDLE]);
            ActionStatus.Status = error;
            //handle error
            goto exit;
        }
    }
    StageSuckHeads(0);
    StageLoadHeads(STAGELOADHEAD_RAMPIN);       //Back out Fingers
    StageConnectorClamp(STAGECONCLAMP_UP);                  //lower Fingers
    ActionStatus.State++;       //State = 9
    PrintStateMessage(LSS_CALIBRATEROTARYSTAGE,LSS_INITSTAGE);
    if((error = ChkErr(DeviceTable->Pid[ACUTRAC_HEADSOFF]->Upload(ACUTRAC_HEADSOFF)) ) < 0)
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        //perform some sort of error recovery
        goto exit;
    }
    if((error = ChkErr(EnableServoFunctions(ROTARY_STAGE,ENABLE_PID,1l))) < 0)
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        goto exit;
    }
    if((error = ChkErr(EnableServoFunctions(ROTARY_STAGE,ENABLE_INTEGRATOR,1l))) < 0)
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        goto exit;
    }
    TDelay(50);     //wait for servo to stabalize
    ClearWaitForRotaryCalibrate();
    if((error = ChkErr(Do_Calibrate(ROTARY_STAGE))) < 0)
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        goto exit;
    }
    PrintStateMessage(LSS_CALIBRATEROTARYSTAGE,LSS_INITSTAGE);
    if(WaitForRotaryCalibrate(5000) == EVENT_TERMINATE) //wait for callibrate to finish
    {
        //Ok, we are in the middle of doing something with the servo, probably, so
        //send an abort move there
        AbortMove(RSTAGE);
        goto exit;
    }
    ActionStatus.State++;           // State = 10
```

```
    if((error = ChkErr(Get_Calibration(ROTARY_STAGE,&pos))) < 0)    //get callibration distance
    {
        I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
        ActionStatus.Status = error;
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        goto exit;
    }
    PrintStateMessage(LSS_BUILDTRACKLUT,LSS_INITSTAGE);
    DeviceTable->CountsPerRadian = AcutracGETCORRECTION();
    if(AcutracGotoLoadUnloadRadius(ACUTRAC_LOADRADIUS) == ACUTRAC_TERMINATE)           //goto u
nload radius
        goto exit;  //task is terminated, just quit
    StageSetDirection(DeviceTable->rotation);   //set spindle direction (this might have failed
earlier)

exit:
    DisplayMode = 0;    //restore display mode back to normal
    ActionTask = 0;     //it is OK to do this now, since all spin
                        //spinstand operations are done, a new
                        //spinstand task can be spawned.  It is only
                        //spinstand operations that are none reentrant
    ActionStatus.Function = ACUTRAC_ACTION_IDLE;
    ActionStatus.StartFlag = ACUTRAC_ACTION_START_READY;
    TDelete(ActionTask);//pass null pointer to Delete task routine
}

/*****************************************************************************
**
**                      DisCon Exersize Routine
**
** This is a TASK that will exercise the spinstand
**
**
*****************************************************************************/

Wait *WaitExersize;


void ExersiseTask(void)
{
    /*************************************************************************
    ** This function is a TASK
    **
    ** Pending Semaphore:WaitExersize
    ** Posting Functions:ACUTRAC_EXERSIZE command in ProcessHighLevelCommands
    **                    (serlprot.cpp)
    **
    ** This function is used as a "burn in" function for the spinstand.
    ** It does most of the basic functions of operating the spinstand,
    ** without requiring human intervention.
    **
    *************************************************************************/

    long pos,lastpos;
    int loop,i;
    int error;
    char *s = new char[256];
    int c;
    //---------------------------------------
    // Start off by setting up system
    // Make sure that X Stage is home
    // Make sure head loader is in corral
    // Prompt for removeal of head stack
    // Calibrate X stage
    // Calibrate Rotary Stage
    //---------------------------------------
```

```
    ActionStatus.State = 0;
    WaitExersize = new Wait(0,"Exercise_Flag");

    while(1)    //this is a task, do forever
    {
    //----------------------------------------
    // Move Rotary to Load/Unload Position
    // Spread Fingers
    // Move XStage to Run Position
    // Unspread Fingers
    // Move Heads Back and forth on disk
    // Move to Load/Unload Position
    // Spread Fingers
    // Move XStage to Home Position
    // Unspread Fingers
    // Wait about 1 Second
    // Repeat Process
    //----------------------------------------
        WaitExersize->Pend();                   //wait for Exercise command
        c = sprintf(s,"Pass=%d\r\n",WaitExersize->GetCount());
        ActionStatus.Function = WaitExersize->GetCount();
        Write(ConsolHandle,s,c);
        if((DeviceTable->mode == HSA) || (DeviceTable->mode == HGA))
        {
            ClearRampFlags();                       //clear RAMP flags
            StageLoadHeads(STAGELOADHEAD_RAMPIN);   //move ramp in
            PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_EXERSIZE);
            ActionStatus.State++;
            if((error = ChkErr(WaitForRampIn(1000)) ) < 0)  //wait for RAMP to get there
            {
                ActionStatus.Status = error;
                c = sprintf(s,"Wait For RAMPIN 1 Timeout:%d\r\n",error);
                Write(ConsolHandle,s,c);
                AcutracPrintErrorMessage(error,s);
                TDelay(20); //wait just a little bit for things to catch up
            }
        }
        if((DeviceTable->mode == HSA) && (DeviceTable->ConnectorClamp == 1))
        {
            ClearConnectorClamps();
            StageConnectorClamp(STAGECONCLAMP_DOWN);
            if((error = ChkErr(WaitForConnectorClampDown(100)) ) < 0)
            {
                ActionStatus.Status = error;
                c = sprintf(s,"Connector Clamp Error:%d\r\n",error);
                Write(ConsolHandle,s,c);
                AcutracPrintErrorMessage(error,s);
                TDelay(20);
            }
        }
        StageSetRpm(DeviceTable->loadrpm);      //set motor RPM to load RPM
        StageSetMotor(1);
        ClearWaitForLinearRun();                //Clear Wait for Linear RUN
        StageSetXStage(STAGESETXSTAGE_RUN);     //Move to RUN position
        PrintStateMessage(LSS_WAITFORRUNPOSITION,LSS_EXERSIZE);
        ActionStatus.State++;
        if((error = ChkErr(WaitForLinearRun(500)) ) < 0)//wait for Linear to Run position
        {
            ActionStatus.Status = error;
            c = sprintf(s,"Wait For RUN 1 Timeout:%d\r\n",error);
            Write(ConsolHandle,s,c);
            AcutracPrintErrorMessage(error,s);
            TDelay(20); //wait just a little bit for things to catch up
        }
        PrintStateMessage(LSS_WAITFORSPINDLEATSPEED,LSS_EXERSIZE);
```

```
        ActionStatus.State++;
        if((error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_ACCEL,Devic
eTable->loadrpm)) ))< 0)
        {
            ActionStatus.Status = error;
            c = sprintf(s,"Wait For MOTOR 1 Timeout:%d\r\n",error);
            Write(ConsolHandle,s,c);
            AcutracPrintErrorMessage(error,s);
            TDelay(20); //wait just a little bit for things to catch up
        }
        if((DeviceTable->mode == HSA) || (DeviceTable->mode == HGA))
        {
            ClearRampFlags();
            StageLoadHeads(STAGELOADHEAD_RAMPOUT);
            PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_EXERSIZE);
            ActionStatus.State++;
            if((error = ChkErr(WaitForRampOut(1000))) < 0)
            {
                ActionStatus.Status = error;
                c = sprintf(s,"Wait For RAMPOUT 1 Timeout:%d\r\n",error);
                Write(ConsolHandle,s,c);
                AcutracPrintErrorMessage(error,s);
                TDelay(20); //wait just a little bit for things to catch up
            }
        }
        StageSetRpm(DeviceTable->testrpm);      //set motor RPM to run RPM
        if((error = ChkErr(WaitForAtSpeed->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_ACCEL,Devic
eTable->testrpm-DeviceTable->loadrpm))))< 0)
        {
            ActionStatus.Status = error;
            c = sprintf(s,"Wait For MOTOR 1 Timeout:%d\r\n",error);
            Write(ConsolHandle,s,c);
            AcutracPrintErrorMessage(error,s);
            TDelay(20); //wait just a little bit for things to catch up
        }
        for(i=0;i<26;++i)   //move the head around a whole lot
        {
            AcutracSEEKTRACK(DeviceTable->maxtrack/2);      //move rotary stage around
            PrintStateMessage(LSS_WAITFORINPOSITION,LSS_EXERSIZE);
            ActionStatus.State++;
            if((error = ChkErr(WaitForRotaryMove(200)) ) < 0)
            {
                c = sprintf(s,"Wait For MOVE Timeout:%d\r\n",error);
                Write(ConsolHandle,s,c);
                ActionStatus.Status = error;
                AcutracPrintErrorMessage(error,s);
                TDelay(20); //wait just a little bit for things to catch up
            }
            AcutracSEEKTRACK(DeviceTable->maxtrack - 1.0);
            PrintStateMessage(LSS_WAITFORINPOSITION,LSS_EXERSIZE);
            ActionStatus.State++;
            if((error = ChkErr(WaitForRotaryMove(200)) ) < 0)
            {
                c = sprintf(s,"Wait For MOVE Timeout:%d\r\n",error);
                Write(ConsolHandle,s,c);
                ActionStatus.Status = error;
                AcutracPrintErrorMessage(error,s);
                TDelay(20); //wait just a little bit for things to catch up
            }
            AcutracSEEKTRACK(DeviceTable->maxtrack/2);
            PrintStateMessage(LSS_WAITFORINPOSITION,LSS_EXERSIZE);
            ActionStatus.State++;
            if((error = ChkErr(WaitForRotaryMove(200)) ) < 0)
            {
                c = sprintf(s,"Wait For MOVE Timeout:%d\r\n",error);
                Write(ConsolHandle,s,c);
```

```
                ActionStatus.Status = error;
                AcutracPrintErrorMessage(error,s);
                TDelay(20); //wait just a little bit for things to catch up
            }
            AcutracSEEKTRACK(0.0);
            PrintStateMessage(LSS_WAITFORINPOSITION,LSS_EXERSIZE);
            ActionStatus.State++;
            if((error = ChkErr(WaitForRotaryMove(200)) ) < 0)
            {
                c = sprintf(s,"Wait For MOVE Timeout:%d\r\n",error);
                Write(ConsolHandle,s,c);
                ActionStatus.Status = error;
                AcutracPrintErrorMessage(error,s);
                TDelay(20); //wait just a little bit for things to catch up
            }
        }   //end of loop to move heads a whole lot
        if((DeviceTable->mode == HSA) || (DeviceTable->mode == HGA))
        {
            ClearRampFlags();
            StageLoadHeads(STAGELOADHEAD_RAMPIN);
            PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_EXERSIZE);
            ActionStatus.State++;
            if((error = ChkErr(WaitForRampIn(1000)) ) < 0)
            {
                c = sprintf(s,"Wait For RAMPIN 2 Timeout:%d\r\n",error);
                Write(ConsolHandle,s,c);
                ActionStatus.Status = error;
                AcutracPrintErrorMessage(error,s);
                TDelay(20); //wait just a little bit for things to catch up
            }
        }
        BrakesDone->SetCount(0);    //just in case, clear semaphore
        StageSetMotor(0);
        ClearWaitForLinearHome();
        StageSetXStage(STAGESETXSTAGE_HOME);
        PrintStateMessage(LSS_WAITXSTAGEHOME,LSS_EXERSIZE);
        ActionStatus.State++;
        if((error = ChkErr(WaitForLinearHome(1000))) < 0)
        {
            c = sprintf(s,"Wait For HOME 1 Timeout:%d\r\n",error);
            Write(ConsolHandle,s,c);
            ActionStatus.Status = error;
            AcutracPrintErrorMessage(error,s);
            TDelay(20); //wait just a little bit for things to catch up
        }
        if((error = ChkErr(BrakesDone->Pend(CalculateAccelTime(ACUTRAC_ACCELTIME_DECEL,DeviceTab
le->loadrpm))) ) < 0)   //wait for motor to stop
        {
            c = sprintf(s,"Wait For Brakes 1 Timeout:%d\r\n",error);
            Write(ConsolHandle,s,c);
            ActionStatus.Status = error;
            AcutracPrintErrorMessage(error,s);
            TDelay(20); //wait just a little bit for things to catch up
        }

        if((DeviceTable->mode == HSA) || (DeviceTable->mode == HGA))
        {
            ClearRampFlags();
            StageLoadHeads(STAGELOADHEAD_RAMPOUT);
            PrintStateMessage(LSS_WAITFORRAMPLOADER,LSS_EXERSIZE);
            if((error = ChkErr(WaitForRampOut(1000))) < 0)
            {
                c = sprintf(s,"Wait For RAMPOUT 2 Timeout:%d\r\n",error);
                Write(ConsolHandle,s,c);
                ActionStatus.Status = error;
                AcutracPrintErrorMessage(error,s);
```

```
                TDelay(20); //wait just a little bit for things to catch up
            }
        }
        if((DeviceTable->mode == HSA) && (DeviceTable->ConnectorClamp == 1))
        {
            ClearConnectorClamps();
            StageConnectorClamp(STAGECONCLAMP_UP);
            if((error = ChkErr(WaitForConnectorClampUp(100)) ) < 0)
            {
                ActionStatus.Status = error;
                c = sprintf(s,"Connector Clamp Error:%d\r\n",error);
                Write(ConsolHandle,s,c);
                AcutracPrintErrorMessage(error,s);
                TDelay(20);
            }
        }
        TDelay(3000);       //delay for 3000 centiseconds
        ActionStatus.State = 0;
    }
}

void InitAccutrac(void)
{
    /*********************************************************************
    ** This function performs all of the needed initializations for this
    ** module.  This function is called before multitasking is enabled
    **
    ** Parameter:
    ** <none>
    ** ReturnValue:
    ** <none>
    *********************************************************************/
    TCB *t;

    StringExclude = new Wait(1,"String Blocker");   //exclusion semaphore for error strings
    t = CreateTask(ExersiseTask,2048,7,"Exercise");
    ActiveTasks->Insert(t);
    //---------------------------------------------------------
    // Create task to handle Spindle Motor Amplifier Fault
    // This task will have a HIGH priority
    //---------------------------------------------------------
    t = CreateTask(SpindleMotorFaultTask,2048,256,"MotorFault");
    ActiveTasks->Insert(t);
}

int CalibrateLinearStage(void)
{
    /*********************************************************************
    ** This function is used to calibrate the "linear" stage.
    ** This is an obsolete function, but remains because you never know
    ** when you might need it again.  This dates back to when there was an
    ** encoder on the linear stage to find out where it was.  This has now
    ** been replaced with limit switches.
    **
    ** Parameter:
    ** <none>
    ** ReturnValue:
    ** returns 0 on success, negative on failure.
    *********************************************************************/
    int error;
    long pos,lastpos;
    int loop;

    if((error = ChkErr(RecordLinearPosition(0,LINEARSTAGE_INPOSENABLE,0)) )== ACUTRAC_TERMINATE)
    //disable lin in pos
        goto exit;
```

```
    if((error = ChkErr(Read_Position(0,&lastpos))) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        goto exit;
    }
    ActionStatus.State++;       //State = 2
    PrintStateMessage(LSS_MOVEXSTAGEHOME,LSS_INITSTAGE);
    if(DeviceTable->mode == HGA)
        StageLoadHeads(STAGELOADHEAD_RAMPIN);   //move ramp in for HGA
    StageSetXStage(0);      //Send X stage to home position
    if((error = TDelay(50)) == EVENT_TERMINATE)         //wait just a little bit
        goto exit;
    loop = 1;
    ActionStatus.State++;       //State = 3
    PrintStateMessage(LSS_WAITFORLINSTAGETOMOVE,LSS_INITSTAGE);
    while(loop)             //wait for X stage to get to most negative position
    {
        if((error = ChkErr(Read_Position(0,&pos))) < 0)
        {
            I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
            ActionStatus.Status = error;
            goto exit;
        }
        if(labs(pos-lastpos) < 10)
            loop = 0;
        else
            TDelay(50);     //wait just a bit more
        lastpos = pos;
    }
    if((error = ChkErr(ZeroCount(0))) < 0)
    {
        AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
        ActionStatus.Status = error;
        goto exit;
    }
    ActionStatus.State++;       //State = 4
    if((error = ChkErr(RecordLinearPosition(0,LINEARSTAGE_RECORDHOME,0)) )== ACUTRAC_TERMINATE)
//Record Home Position
        goto exit;
    PrintStateMessage(LSS_MOVEXSTAGETORUN,LSS_INITSTAGE);
    StageSetXStage(1);      //go run
    if((error = TDelay(50)) == EVENT_TERMINATE) goto exit;  //wait just a little bit
    loop = 1;
    ActionStatus.State++;       //State = 5
    PrintStateMessage(LSS_WAITFORLINSTAGETOMOVE,LSS_INITSTAGE);
    while(loop)             //wait for X stage to get to most negative position
    {
        if((error = ChkErr(Read_Position(0,&pos))) < 0)
        {
            I2CWriteLEDString((char *)LCDStrings[LCDS_ERRORSERVO]);
            ActionStatus.Status = error;
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_INITSTAGE]);
            goto exit;
        }
        if(labs(pos-lastpos) < 10)
            loop = 0;
        else
            if((error = TDelay(50) == EVENT_TERMINATE)) goto exit;      //wait just a bit more
        lastpos = pos;
    }
    if((error = ChkErr(RecordLinearPosition(0,LINEARSTAGE_RECORDRUN,0))) == ACUTRAC_TERMINATE)
//Record Run Position
        goto exit;
    if((error = ChkErr(RecordLinearPosition(0,LINEARSTAGE_INPOSENABLE,1))) == ACUTRAC_TERMINATE)
```

```
    //enable lin in pos
        goto exit;
    ActionStatus.State++;   //State = 6
    PrintStateMessage(LSS_MOVEXSTAGEHOME,LSS_INITSTAGE);
    ClearWaitForLinearHome();
    StageSetXStage(0);  //return back home
    if((DeviceTable->mode) == HGA || (DeviceTable->mode == HSA))
    {
        ActionStatus.State++;       //State = 7
        if(DeviceTable->mode == HSA)
        {
            StageLoadHeads(STAGELOADHEAD_RAMPOUT);
            ActionStatus.State++;       //State = 8
        }
        PrintStateMessage(LSS_WAITXSTAGEHOME,LSS_INITSTAGE);
        if((error = ChkErr(WaitForLinearHome(2000))) < 0)
        {
            AcutracPrintErrorMessage(error,(char *)StateStrings[LSS_WAITXSTAGEHOME]);
            ActionStatus.Status = error;
            goto exit;
        }
        ActionStatus.State++;       //State = 9
    }
exit:
    return error;
}
```

```
//---------------------------------------------------------------------------
// Header File for Acutrac Spinstand interface functions
//---------------------------------------------------------------------------

#ifndef ACUTRAC__H
#define ACUTRAC__H

#include "erorcode.h"

struct PidParams {
    int axis;
    long Kp;    //proportional gain
    long Kv;    //derivitive gain
    long Ki;    //integral gain
    long NotchF;    //notch frequency
    long NotchQ;    //notch Q
    long NotchD;    //notch depth
    long NotchEnableFlag;   //yeah, what it says
    static currenttype;
public:
    PidParams(int c);       //constructor, needs axis information
    ~PidParams();
    int Upload(int type);       // sends data to the DSP controller
    int Download(int type);     // gets data from DSP controller
    void Print(void);       //print params to console
    int GetCurrentType(void);
};

struct AcutracDeviceTable {
    int mode;                   //HSA,HGA,etc.
    int maxrpm;                 //maximum rpm
    int minrpm;                 //minimum rpm
    int testrpm;                //rpm to test media at
    int unloadrpm;              //rpm to unload heads at
    int loadrpm;                //rpm to load heads at
    int rotation;               //driver rotation direction
    int TrackPitchMode;         //constant radius/constant angular track pitch
    double inittrack;           //init track
    double currenttrack;        //current track number
    double maxtrack;            //maximum track number
    double tracksize;           //tracksize in microinches
    double insideradius;        //inside radius in inches
    double outsideradius;       //outside radius in inches
    double pivottogap;          //pivot to gap in inches
    double pivottocenter;       //pivot to center in inches
    double actuatorradius;      //effective actuator radius in inches
    double referenceangle;      //actuator angle from spindle center to beam normal in radians
    double LoadRadius;          //load radius in inches
    double UnloadRadius;        //unload radius in inches
    double preloadunloadradius; //used for HGA, where to position before going to load/unload ra
dius
    double xdistancehome;       //distance from x-home to head load poisition in inches
    double xdistancerun;        //distance from head load to operatin position in inches
    int trackoffset;            //current track offset step
    int maxhead;                //maximum number of heads
    int currenthead;            //current head number
    int dutorientation;         //DUT orientation (either FRONT or BACK of spindle)
    double CountsPerRadian;     //counts per radian of Rotary Encoder
    char ServoParams[10];       //name of Positioner param file
    int CurrentHead;            //current selected head
    int ConnectorClamp;         //indicates if connector clamp is used
    int ConnectorClampTimeout;  //timeout value for connector clamp semaphore
    long Overshoot;             //overshoot position by this much to eliminate hysterisis
    PidParams *Pid[2];          //array of pid params that are used in various situations
public:
    AcutracDeviceTable();
```

```
    ~AcutracDeviceTable();
};

struct ACTION_STATUS {
    unsigned State;     //internal state of action
    unsigned Status;        //internal status of action
    unsigned Function;  //which function is being performed
    unsigned StartFlag; //indicates if start can be run
    ACTION_STATUS();        //constructor
};

//---------------------------------------------------------------------------
//
// various constant defines
//
//---------------------------------------------------------------------------

#define XSTAGE  0   //axis number for linear stage
#define RSTAGE  1   //axis number for rotary stage
#define SSTAGE  2   //axis number for Spindle Motor

#define STD     0   //for standard HSA mounting
#define TLOCK   1   //for swan type twist-lock mount

#define DUT_FRONT   0   //for DUT orientation on front of spindle
#define DUT_BACK    1   //for DUT orientation on back of spindle

#define CCW     0   //spindle direction
#define CW      1   //spindle direction

#define HSA     0   //types of product to test
#define HGA     1
#define HSAR    2   //HSA test, resonance

//#define MOP       2   //Magneto Optical

#define ACUTRAC_HEADSON                 0   //indicates heads are on tooling
#define ACUTRAC_HEADSOFF                1   //indicates heads are not on tooling

//function
#define ACUTRAC_ACTION_IDLE             0
#define ACUTRAC_ACTION_RESET            1
#define ACUTRAC_ACTION_START            2
#define ACUTRAC_ACTION_STOP             3
#define ACUTRAC_ACTION_PENDING          4
#define ACUTRAC_ACTION_LIFT             5
#define ACUTRAC_ACTION_RELOAD           6

//start flag
#define ACUTRAC_ACTION_START_NOT        0   //start not run yet, reset not run either
#define ACUTRAC_ACTION_START_READY      1   //ready for start to be run, but not stop
#define ACUTRAC_ACTION_START_YES        2   //start run, not again, reset by stop
#define ACUTRAC_ACTION_LIFTED           3   //heads are lifted off the disk

//-------------------------------------------------
// defines for AcutracGotoLoadUnloadRadius(int rad)
//-------------------------------------------------

#define ACUTRAC_LOADRADIUS              0
#define ACUTRAC_UNLOADRADIUS            1

/*************************************************************************\
|*                         function prototypes                           *|
\*************************************************************************/

extern int AcutracINIT(char *name);
```

```
extern int AcutracRESET(char *name);
extern int AcutracSTART(void);
extern int AcutracSTOP(void);
extern int AcutracGETRPM(int *rpm);
extern int AcutracGETMAXRPM(int *maxrpm);
extern int AcutracGETMINRPM(int *minrpm);
extern int AcutracSTARTSPINDLE(void);
extern int AcutracSTOPSPINDLE(void);
extern int AcutracSETROTDIR(int dir);
extern int AcutracGETROTDIR(int *dir);
extern int AcutracSETLOADRPM(int rpm);
extern int AcutracGETLOADRPM(int *rpm);
extern int AcutracSEEKTRACK(char *s);
extern int AcutracSEEKTRACK(double trackgoal);
extern int AcutracSTEPIN(void);
extern int AcutracSTEPOUT(void);
extern int AcutracGETTRACK(char *s);
extern int AcutracGETTRACK(double *t);
extern int AcutracSETMAXTRACK(char *maxtrack);
extern int AcutracGETMAXTRACK(char *maxtrack);
extern int AcutracSETRADIUS(int IsId,char *radius);
extern int AcutracSETTRACKSIZE(char *tracksize);
extern int AcutracGETTRACKSIZE(char *tracksize);
extern int AcutracGETERRORMESSAGE(char *message);
extern int AcutracGETSTATE(int *status, int *function,int *startflag);
extern int AcutracGETSTATEMESSAGE(char *message);
extern void AcutracSETStatus(int status);
extern int AcutracPrintErrorMessage(int ErrorNumber,char *s);
extern char *AcutracGetErrorBuffer(void);
extern int AcutracSELECTHEAD(int head);
extern int AcutracGETSELECTEDHEAD(int *head);
extern int AcutracSETCORRECTION(char *cal);
extern int AcutracGETCORRECTION(char *cal);
extern double AcutracGETCORRECTION(void);
extern int AcutracSETPIDPARAMS(int type,long *pid);
extern int AcutracGETPIDPARAMS(int type,long *pid);
extern int AcutracGETPIDPARAMSTYPE(void);
extern int AcutracSELECTPARAMSTYPE(int type);

extern TCB *GetActionTask(void);
extern ACTION_STATUS *GetActionStatus(void);
extern int AcutracGotoLoadUnloadRadius(int r);
extern int AcutracUnloadHeads(void);

extern AcutracDeviceTable *DeviceTable;

extern int CalculateAccelTime(int mode, int rpm);

extern int AcutracLIFTHEADS(void);
extern int AcutracRELOADHEADS(void);
extern int AcutracGOTORADIUS(double radius);
extern int AcutracGOTORADIUS(char *radius);

#define ACUTRAC_ACCELTIME_ACCEL     0
#define ACUTRAC_ACCELTIME_DECEL     1

//--------------------------------
// conversion routines
//--------------------------------
extern long angle_to_icts(double angle);
extern double icts_to_angle(long icts);
extern double track_to_rad(double track);
extern double rad_to_track( double rad);
extern double rad_to_angle(double rad,int *error);
extern double angle_to_rad(double angle);
```

```
//--------------------------------
// misc crap
//--------------------------------

extern int CalibrateLinearStage(void);

#endif
```

```
//---------------------------------------------------------------------
//
// These routines are used to protect non-rentrant functions
//
// Uses a Semaphore
//
// copyright (c) 1998 teletrac inc
//
//---------------------------------------------------------------------

#include <stdio.h>
#include "task.h"
#include "queue.h"

static Wait *MemoryEvent = (Wait *)0;
static Flag = 0;

extern "C" {
void InitDisable(void)
{
    MemoryEvent = new Wait(1,"MallocBlocker");  //can be used once before pending
    Flag = 1;       //ok to start using Disable and Enable
}

void Disable(void)
{
    if(!Flag) return;   //very dangerous
    MemoryEvent->Pend();
}

void Enable(void)
{
    if(!Flag) return;   //very dangerous
    MemoryEvent->Post();
}
}
```

```
/*
** header file for distance conversions
*/
#ifndef DISTANCE__H
#define DISTANCE__H

/*--------------------------------------------------------------------------**
**                                                                          **
** Conversion Types                                                         **
**                                                                          **
**--------------------------------------------------------------------------*/

#define DISTANCE_COUNTS     0   /*  divide ICnts by 16  */
#define DISTANCE_ICOUNTS    1   /*  no change in value  */
#define DISTANCE_MM         2   /*  converts icounts to milimeters  */
#define DISTANCE_INCHES     3   /*  converts icounts to inches      */
#define DISTANCE_MICRONS    4   /*  converts icounts to microns     */
#define DISTANCE_MILS       5   /*  converts icounts to miliinches  */
#define DISTANCE_MICROINCH  6   /*  converts icounts to micro inches    */
#define DISTANCE_CM         7   /*  converts icounts to centimeters */

#define DISTANCE_SINGLE     0   /*  single pass conversion  */
#define DISTANCE_DOUBLE     1   /*  double pass conversion  */

#ifdef __cplusplus
extern "C" {
#endif

extern double ICountsToDistance(double icnts, double presure, double temperature, double humidit
y,int type, int units);
extern double DistanceToICounts(double dist, double presure, double temperature, double humidity
,int type, int units);
extern char *ICountsToStr(char *s,double icnts, double presure, double temperature, double humid
ity,int type, int units);

#ifdef __cplusplus
}
#endif

#endif  /*  distance__h */
```

```
/*************************************************************************
**
** Convert BigEndian <-> LittleEndian
**
*************************************************************************/

extern "C" {
int IntLittleEndian(int v)
{
    /*
    ** v is little endian, returns big endian
    */
    union{
        unsigned char cv[2];
        int v;
    }convert;
    unsigned char t;

    convert.v = v;
    t = convert.cv[0];
    convert.cv[0] = convert.cv[1];
    convert.cv[1] = t;
    return convert.v;
}

int GetInt(char *s)
{
    /*
    ** This routine returns back the INT value it finds at the address
    ** pointed to by s.  We need this because we will not know the
    ** memory boundry that is being used
    */
    union {
        char cv[2];
        int v;
    }convert;

    convert.cv[1] = *s++;
    convert.cv[0] = *s++;
    return convert.v;
}

char *SaveInt(char *s,int v)
{
    union {
        unsigned char cv[2];
        int v;
    }convert;

    convert.v = v;
    *s++ = convert.cv[1];
    *s++ = convert.cv[0];
    return s;
}

long GetLong(char *s)
{
    union {
        char cv[4];
        long v;
    }convert;
    int i;

    for(i=3;i>=0;--i)
        convert.cv[i] = *s++;
    return convert.v;
```

```
}

double GetDouble(char *s)
{
    union {
        char cv[8];
        double v;
    }convert;
    int i;

    for(i=7;i>=0;--i)
        convert.cv[i] = *s++;
    return convert.v;
}

char *SaveLong(char *s,long v)
{
    union {
        char cv[4];
        long v;
    }convert;

    convert.v = v;
    *s++ = convert.cv[3];
    *s++ = convert.cv[2];
    *s++ = convert.cv[1];
    *s++ = convert.cv[0];
    return s;
}

}
```

```
//*************************************************************************
//
// ERROR CODES FOR ALL SYSTEMS in the SPINDLE CONTROLLER
//
// Errors can be generated by the following systems:
// High Level Spinstand Functions:ACUTRAC.CPP
// Central Input Output Handler:CIO.CPP
// DSP SERVO INTERFACE:SERVO.CPP-COMM.H
// Ramdisk Handler:RAMDISK.CPP
// Serial Protocol Handler:SERLPROT.CPP
// MultiTasking Kernel:TASK.CPP
//
//*************************************************************************

#ifndef ERRORCODES__H
#define ERRORCODES__H

//-----------------------------------------------------------------
// Error codes for Multi Tasking Kernel
//-----------------------------------------------------------------

#define EVENT_NOERROR   0
#define EVENT_TIMEOUT   -96     //event has timed out
#define EVENT_OVERFLOW  -97     //too many events pending
#define EVENT_BUFFFULL  -98     //pipeline buffer full
#define EVENT_NOTASKS   -99     //no tasks to reschedule to
#define EVENT_TERMINATE -100        //task needs to be canceled
#define EVENT_ABORTED   -101        //event was aborted by another task

//-----------------------------------------------------------------
// Error Codes for Servo Controller
//-----------------------------------------------------------------

#define SERVO_GOTACK        0       //error codes from WaitACK(), OK
#define SERVO_GOTTIMEOUT    -112    //TIMEOUT
#define SERVO_GOTNAK        -113    //NAKked
#define SERVO_GOTACKTIMEOUT -114    //timed out waiting for ACK
#define SERVO_ERROR         -115
#define SERVO_UNKOWN_ERROR  -116    //        ???

//-----------------------------------------------------------------
// Error Codes for Spindle Commands
//-----------------------------------------------------------------

#define SPINDLE_OK          0       //everything is peachy keen
#define SPINDLE_ERROR       -128    //some sort of error
#define SPINDLE_NAK         -129    //command refused

//-----------------------------------------------------------------
// Error Codes for Serial Protocol Handler
//-----------------------------------------------------------------

/*
** return values from TelReceiveProtocol
*/

#define TEL_PROT_OK     0
#define TEL_PROT_TIMEOUT    -136
#define TEL_PROT_CHECKSUM   -137
#define TEL_PROT_BADHEADER  -138

//-----------------------------------------------------------------
// Error Codes for Ramdisk Driver
//-----------------------------------------------------------------

/*  Device Driver Errors    */
```

```
#define RAMDISK_FILENOTFOUND        -64
#define RAMDISK_NOSECTORS           -65
#define RAMDISK_FILEOPEN            -66
#define RAMDISK_BADNAME             -67
#define RAMDISK_NODIRECTORIES       -68
#define RAMDISK_EOF                 -69

//---------------------------------------------------------------
// Error returned by misc commands
//---------------------------------------------------------------

#define GENERAL_OK                  0
#define GENERAL_NAK                 -80
#define GENERAL_ERROR               -81

//---------------------------------------------------------------
// Error Codes for High Level Spindstand Functions
//---------------------------------------------------------------

    /*  defines for ACTION_STATUS   */
    /*  status error codes  */

#define ACUTRAC_SUCCESS         0       //this is what you really want
#define ACUTRAC_ERROR           -1      //you get this when there is some error
#define ACUTRAC_BUSYERROR       -2      //you get this when spinstand doing something else
#define ACUTRAC_CONFIGFILE      -3      //you get this when something wrong with config file
#define ACUTRAC_SERVO_NAK       -4      //you get this when DSP doesn't want to do what you tell
 it
#define ACUTRAC_SERVO_TIMEOUT   -5      //you get this when comm to DSP fails
#define ACUTRAC_SERVO_UNKNOWN   -6      //you get this when it don't know
#define ACUTRAC_OUTOFRANGE      -7      //you get this when track is out of range
#define ACUTRAC_FILEERROR       -8      //you get this when RESET has problem with AFG file
#define ACUTRAC_FILEBAD         -9      //you get this when RESET can't find AFG file
#define ACUTRAC_RAMPTIMEOUT     -10     //you get this when ramp lifters don't move
#define ACUTRAC_LINEARTIMEOUT   -11     //you get this when X stage not moving
#define ACUTRAC_MOTIONTIMEOUT   -12     //you get this when rotary not moving
#define ACUTRAC_SPINDLETIMEOUT  -13     //you get this when motor not up to speed
#define ACUTRAC_USERABORT       -14     //you get this when stop button pushed
#define ACUTRAC_TIMEOUT         -15     //you get this when a semaphore times out
#define ACUTRAC_TERMINATE       -16     //you get this when task needs to terminate
#define ACUTRAC_NOTHANDLED      -17     //haven't figured out what to do in this case yet
#define ACUTRAC_LOWAIR          -18     //you get this when air pressure in system is low
#define ACUTRAC_NOINITINFO      -19     //no init file has been uploaded
#define ACUTRAC_AMPFAULT        -20     //you get this when the ampilfier faults
#define ACUTRAC_CLAMPTIMEOUT    -21     //you get this when the connector clamp times out

//---------------------------------------------------------------
// Error Codes for Central I/O Handler
//---------------------------------------------------------------

/*
    cio error codes
*/
#define CIO_ABORT       -144    /*  device IO was interrupted by user   */
#define CIO_NO_HANDLES  -145    /*  NO MORE IOCB'S AVAILIABLE   */
#define CIO_NO_DEVICE   -146    /*  could not locate that device    */
#define CIO_WRONLY      -148    /*  Read was attempted on write only    */
#define CIO_INVALID     -149    /*  INVALID FUNCTION NUMBER */
#define CIO_NOT_OPEN    -150    /*  IOCB NOT OPEN   */
#define CIO_INVLD_HNDL  -151    /*  INVALID HANDLE  */
#define CIO_RDONLY      -152    /*  Write was attempted on read only    */
#define CIO_END_OF_FILE -153    /*  end of file encountered             */
#define CIO_TRUNCATED   -154    /*  NO EOL FOUND BEFORE END OF BUFFER   */

#endif
```

```
/*************************************************************************
**
** Front Panel LED's
**
** Copyright (c) 1998 teletrac inc
**
*************************************************************************/

#include <stdio.h>
#include "task.h"
#include "queue.h"
#include "global.h"
#include "frontled.h"
#include "cio.h"

static int PortCShadow = 0x08;          /*  initial PORT_C value    */

static Wait *Exclude;
static TCB *FrontPanelLedTask;


extern "C" void InitFrontPanelLeds(void)
{
    int i;

    PORT_C = PortCShadow;
    Exclude = new Wait(1,"PortCBlocker");       /*  allow one access at a time  */
}

/*************************************************************************
**
** API to front panel LED's
**
** This function is re entrant at the task level.  It is NOT!!! rentrant at
** the interrupt level (PendSemaphore can not be called from an interrupt)
**
** Input Parameters:
**  Led:Specifies which LED to operate on, a number greater than or equal
**      to zero.
**  time:specifies what to do to led.
**      :positive value->time in 2 time system ticks to cycle led
**      :0 ->turn off led
**      :negative value->turn on LED
** Returns:
**  Nothing.
**
*************************************************************************/

extern "C" int GetPortCBits(void)
{
    int v = PortCShadow;
    return v;       //return only 3 lsbs
}


extern "C" void SetPortCBits(int v)
{
    v &= 0x07;
    Exclude->Pend();
    PortCShadow |= v;
    PORT_C = PortCShadow;
    Exclude->Post();
}

extern "C" void ClearPortCBits(int v)
{
```

```
    int e;

    v &= 0x07;
    Exclude->Pend();
    PortCShadow &= ~v;
    PORT_C = PortCShadow;
    Exclude->Post();
}

extern "C" void SetPortC(int Bit,int value)
{
    int e;
    int mask;

/*-------------------------------------------------------------------------
** Led values from  0 to 7**
**-------------------------------------------------------------------------*/

    if(Bit < 0 || Bit > 8 )
        return;     //come on guys, let's have a valid input, prevents BOMB's
    mask = 1 << Bit;
    Exclude->Pend();
    if(!value)
    {
        PortCShadow &= ~mask;   //sets bit to 0
    }
    else
        PortCShadow |= mask;
    PORT_C = PortCShadow;
    SystemStatus.LedStates = ~PortCShadow;
    Exclude->Post();
}
```

```
/*****************************************************************************
**
** Header file for front panel Leds
**
*****************************************************************************/

#ifndef FRONTLED__H
#define FRONTLED__H

//
// leds for front panel switchs
//
#define FRONTLED_ERROR          0
#define FRONTLED_ENABLE_MICRO_AUTOCOMP          1
#define FRONTLED_UNDEF2         2
#define FRONTLED_RSTDSP         3
#define FRONTLED_SPINENAMP      4
#define FRONTLED_UNDEF3         5
#define FRONTLED_UNDEF4         6
#define FRONTLED_EXTAMPSEL      7

//
//individual LEDs
//
//

/*************************************************************
**
** Port Defines
**
*************************************************************/

#define PORT_C      *((volatile int *)0xffcd0000)   /*  address of PORT out to spin stand   */

//-------------------------------------------------------------
// function prototypes
//-------------------------------------------------------------
extern "C" void SetPortCBits(int v);
extern "C" void ClearPortCBits(int v);
extern "C" int GetPortCBits(void);

extern "C" void InitFrontPanelLeds(void);
extern "C" void SetPortC(int Bit,int value);

#endif //FRONTLED__H
```

```
/*****************************************************************************
**
** I2C Interface Code
**
** Copyright (c) 1997 Teletrac Inc.
** Global Functions:
**
** Initialize I2C port
** Send Message from I2C port
** Get Message from I2C port
**
** Local Functions:
**
** Interrupt Handler
** Decode Message
**
*****************************************************************************
**
** Signals recieved from spinstand via I2C
**
** Vacuum Sensor
** Air Presure
** Ramp In
** Ramp Out
** Comb Removed (this is a sensor in the comb corral, manually removed/replaced
**
*****************************************************************************
**
** Controls that can be activated
**
** Vacuum chuck
** Disk Clamp/Unclamp
** RampIn/Out
** LinearRun/Home
**
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include "task.h"   //defines for multi tasking kernel
#include "i2c.h"    //defines for I2C port
#include "cio.h"
#include "global.h"
#include "spindle.h"
#include "serlprot.h"
#include "quad.h"
#include "spinchip.h"
#include "acutrac.h"
#include "frontled.h"
#include "servo.h"
#include "timer.h"
#include "strings.h"

extern int ConsolHandle;    //debug, delete me someday


//---------------------------------------------------------------------------
// Local Globals
//---------------------------------------------------------------------------
static BQueue *I2COutputMessageQueue;   //queue of I2C mesages to send

static EventQueue *I2CPipe;             //pipe of data from I2C controller
static EventQueue *I2CSendPipe;         //pipe of data to I2C controller

static EventQueue *XStagePipe;          //semaphore for XStage
static EventQueue *HeadLoaderPipe;      //semaphore for head loader
```

```
static EventQueue *SpindleMotorPipe;    //spindle motor on/off semaphore
static EventQueue *ContinuePipe;        //Semaphore for continue button
static EventQueue *StopPipe;            //semaphore for stop button
static EventQueue *VacuumButtonPipe;    //semaphore for vacuum button
static EventQueue *ConnClampPipe;       //semaphore for connector clamp
static EventQueue *AdjustEnablePipe;    //semaphore for adjust enable button
static EventQueue *PanelEnablePipe;     //semaphore for adjust panel button
static EventQueue *AirPresurePipe;      //semaphore for air pressure low sensor
static EventQueue *CombCoralPipe;       //semaphore for the comb corral
EventQueue *RampInPipe;                 //semaphore for Ramp In Sensor
EventQueue *RampOutPipe;                //semaphore for Ramp Out Sensor
EventQueue *ClampUpPipe;                //semaphore for Connector Clamp Sensor
EventQueue *ClampDownPipe;              //semaphore for Connector Clamp Sensor
static EventQueue *VaccumPipe;          //semaphore for Vacuum Sensor
static EventQueue *SpinstandState;      //semaphore for spinstand status
static EventQueue *SpinstandVersion;    //semaphore for getting spinstand version

static EventQueue *LastCommandGarbagePipe;  //semaphore for garbage command task
static Wait *WaitForContinueButton;     //semaphore for Continue Button
static Wait *WaitForSuck;               //semaphore for vac sensor
static Wait *WaitForComb;               //semaphore for waiting for comb

static EventQueue *CombRemoved;         //Sensor for Comb Tender

TSemaphore *RampInFlag;         //Semaphore for Comb In Sensor
TSemaphore *RampOutFlag;            //Semaphore for Comb out Sensor
TSemaphore *ClampUpFlag;            //Semaphore for Clamp Up sensor
TSemaphore *ClampDownFlag;      //Semaphore for Clamp Down sensor

static int XmitFlag = 0;
static int RecFlag = 0;
static Wait *WaitForI2CXmit;            //semaphore for I2C transmit

//Wait *ApplyBrakes;
TSemaphore *BrakesDone;

static const char * const ActionTaskStrings[4] = {
    "Accutrac_Start",
    "Accutrac_Stop",
    "Accutrac_Reset",
    NULL
};

static int XStagePosition;  //0=home,1=run

//---------------------------------------------------------------------------


extern "C"  {
static void DispatchI2CMessages(void);  //task to dispatch messages from I2C port
void I2CSendDemon(void);
}


//---------------------------------------------------------------------------
// Misc functions
//---------------------------------------------------------------------------

int WaitForCombInOut(int combinout)
{
    int retval;
    WaitForComb->SetCount(0);   //clear semaphore before waiting
    if((retval = WaitForComb->Pend()) == combinout)
        retval = 0;
    else if(retval != EVENT_TERMINATE)
        retval = 1;
```

```
    return retval;
}

/*******************************************************************************
** int WaitForConnectorClampUp(int timeout);
** int WaitForConnectorClampDown(int timeout);
**
*******************************************************************************/
void ClearConnectorClamps(void)
{
    ClampUpFlag->SetCount(0);
    ClampDownFlag->SetCount(0);
}

int WaitForConnectorClampUp(int timeout)
{
    int error;
    error = ClampUpFlag->Pend(timeout);
    if(error == EVENT_TIMEOUT) error = ACUTRAC_CLAMPTIMEOUT;
    return error;
}

int WaitForConnectorClampDown(int timeout)
{
    int error;
    error = ClampDownFlag->Pend(timeout);
    if(error == EVENT_TIMEOUT) error = ACUTRAC_CLAMPTIMEOUT;
    return error;
}

/*******************************************************************************
**
** void WaitForChuckSuck(void)
**
** Waits for vaccumme sensor to indicate when the vaccumme chuck has
** something on it and is sucking hard
** returns 0 if we got the event we wanted
** return 1 if we did not get what we wanted
**
*******************************************************************************/

int WaitForChuckSuck(int suckblow)
{
    int retval;
    if((retval = WaitForSuck->Pend()) == suckblow)  //pend on suck event
        retval = 0;
    return retval;
}

void ClearChuckSuck(void)
{
    WaitForSuck->SetCount(0);   //clear semaphore
                            //This could be a very dangerous thing
                            //to do
}

/*******************************************************************************
**
** void WaitForContinueButtonPushed(void)
**
** Waits for a complete cycle of the CONTINUE button to be pushed and
** released
**
*******************************************************************************/

int WaitForContinueButtonPushed(void)
```

```
{
    WaitForContinueButton->SetCount(0);     //reset back to ZERO
    return WaitForContinueButton->Pend();   //wait for button to be released
}

/*****************************************************************************
**
** int WaitForRampIn(int timeout)
**
** Waits for Ramp to Move Into Ramp Corral
** Returns 0 on success
** Returns Negative on failure (timeout)
**
*****************************************************************************/

int WaitForRampIn(int timeout)
{
    int error;

    error = RampInFlag->Pend(timeout);
    if(error == EVENT_TIMEOUT) error = ACUTRAC_RAMPTIMEOUT;
    return error;
}

/*****************************************************************************
**
** int WaitForRampOut(int timeout)
**
** Waits for Ramp to Move Out of Ramp Corral and Spread Fingers
**
** Returns 0 on success
** Returns Negative on failure (timeout)
**
*****************************************************************************/

int WaitForRampOut(int timeout)
{
    int error;

    error = RampOutFlag->Pend(timeout);
    if(error == EVENT_TIMEOUT) error = ACUTRAC_RAMPTIMEOUT;
    return error;
}

void ClearRampFlags(void)
{
    RampInFlag->SetCount(0);
    RampOutFlag->SetCount(0);
}

//---------------------------------------------------------------------------
// Tasks that handle the various events caused by a message from the I2C port
//---------------------------------------------------------------------------

volatile int PanelEnableFlag=0;
volatile int NotLockedOutFlag=0;
volatile int AdjustFlag = 0;
volatile int AirPressureFlag=0;     //true indicates good air pressure

static void MoveXStage(int State)
{
    I2CSetControlStage(FPCOMMAND_HOME_X_LED,State ^ 1);
    I2CSetControlStage(FPCOMMAND_STAGEDRV,State);
    I2CSetControlStage(FPCOMMAND_RUN_X_LED,State);
}
```

```
int GetXStagePosition(void)
{
    return XStagePosition;
}

static void XStage(void)
{
    int Cmd;
    int State=0;
    int Type;

    SetIrqParams(LINEAR_STAGE,HDWIRQ_LIMIT_M | HDWIRQ_SET,HDWIRQ_ENABLE);       /*  set interrup
t parameter for irq vector  */
    SetIrqParams(LINEAR_STAGE,HDWIRQ_LIMIT_P | HDWIRQ_SET,HDWIRQ_ENABLE);       /*  set interrup
t parameter for irq vector  */
    ClearInterrupt(LINEAR_STAGE,CLEAR_LIMIT_M);
    ClearInterrupt(LINEAR_STAGE,CLEAR_LIMIT_P);
    I2CSetControlStage(FPCOMMAND_RUN_X_LED,0);
    I2CSetControlStage(FPCOMMAND_STAGEDRV,0);
    I2CSetControlStage(FPCOMMAND_HOME_X_LED,1);
    while(1)        //this is a TASK, loop forever
    {
        XStagePipe->Pend();
        Type = XStagePipe->Get();   //where did command come from
        Cmd = XStagePipe->Get();    //check state of switch
        if((PanelEnableFlag && NotLockedOutFlag) || Type)           //ARE we allowed to do anyth
ing?
        {
            if(Type == 0)
            {
                I2CBeepTheBuzzer(1,2);
                if(Cmd == 0)            //was button pushed down?
                {
                    State ^= 1;     //change state of state
                    MoveXStage(State);
                }
            }
            else
            {
                State = Cmd;
                MoveXStage(State);
            }
            XStagePosition = State;
            SystemStatus.spin.stat.xstage = State;
        }
    }
}

static void LoadHeads(int State)
{
    I2CSetControlStage(FPCOMMAND_RAMP,State);
    I2CSetControlStage(FPCOMMAND_UNLOADED_LED,State ^ 1);
    I2CSetControlStage(FPCOMMAND_LOADED_LED,State);
}

static void HeadLoader(void)
{
    int Cmd;
    int State=0;
    int Type;

    TDelay(30);
    State = I2CGetStageStatus();
    if(State & FPCOMMAND_RAMP_IN)
        State = 0;
    else if (State & FPCOMMAND_RAMP_OUT)
```

```
        State = 1;
    else
        State = 1;  //don't really know, just choose one
    SystemStatus.spin.stat.head = State;
    LoadHeads(State);
//  I2CSetControlStage(FPCOMMAND_UNLOADED_LED,1);
    while(1)        //this is a TASK, loop forever
    {
        HeadLoaderPipe->Pend();
        Type = HeadLoaderPipe->Get();   //where did command come from
        Cmd = HeadLoaderPipe->Get();
        if((PanelEnableFlag && NotLockedOutFlag) || Type)           //ARE we allowed to do anyth
ing?
        {
            if(Type==0)
            {
                I2CBeepTheBuzzer(1,2);
                if(Cmd == 0)            //is button pushed down?
                {
                    State ^= 1;         //toggle state
                    LoadHeads(State);
                }
            }
            else    //type == 1
            {
                State = Cmd;
                LoadHeads(State);
            }
            SystemStatus.spin.stat.head = State?1:0;
        }
    }
}

static void SpindleMotor(void)
{
    //-----------------------------------------------------------------
    // Very Complicated Function, now.
    // Changed September 30, 1998
    // Will use a velocity control loop to control speed during accel
    // and decel times
    // When accel is finished, will switch over to pll.  If the speed
    // switched to 0, it will just stop
    //-----------------------------------------------------------------
    int Cmd;
    int SubType;
    int RunState=0;         //keep track for front panel LED's
    int DirectionState=0;   //keep track for front panel LED's
    int Type;
    int error;
    int OtherFlag;
    int SpindleDirection;   //used for braking
    int Accel;
    long Goal;
    //------------------------------
    // initialize direction states
    //------------------------------
    DirectionState = Xio(SPIN_GETDIRECTION,sys.HSpindle,(char *)0,(char *)0,0l,0);
    SystemStatus.spin.stat.direction = DirectionState?1:0;
    if(DirectionState)
    {
        I2CSetControlStage(FPCOMMAND_CCW_LED,0);
        I2CSetControlStage(FPCOMMAND_CW_LED,1);
    }
    else
    {
        I2CSetControlStage(FPCOMMAND_CCW_LED,1);
```

```
        I2CSetControlStage(FPCOMMAND_CW_LED,0);
    }
    //--------------------------------
    // initialize run state
    //--------------------------------
    I2CSetControlStage(FPCOMMAND_STOP_SPIN_LED,1);
    //--------------------------------
    // set up braking semaphores
    //--------------------------------
    BrakesDone = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"BrakesDone");
    DirectionChanged = new Wait(0,"DirectionChanged");  //create semaphore
    SpinIOEnableIrq(SPIN_DIR);          //enable spindle direction interrupt

    //--------------------------------
    // Main Task Body
    //--------------------------------
    while(1)
    {
        //-----------------------------
        // Get message from pipe
        //-----------------------------

        SpindleMotorPipe->Pend();
        Type = SpindleMotorPipe->Get(); //where did command come from
        SubType = SpindleMotorPipe->Get();  //what action are we to do
        Cmd = SpindleMotorPipe->Get();      //and what do we do
        //----------------------------
        // check for fault
        //----------------------------
        OtherFlag = SpinGetInputLevel(SPIN_AMPFAULT) ^ 0x01;
        switch(SubType) //process commands
        {
            case MOTORSPINDLE_STARTSTOP_COMMAND:
                if((PanelEnableFlag && NotLockedOutFlag) || Type)           //ARE we allowed to
do anything?
                {
                    if(Type==0)
                    {
                        if(Cmd == 0)    //is button pushed down?
                        {
                            RunState ^= 1;      //toggle state
                        }
                    }
                    else    //internal command, set State Absolute
                    {
                        RunState = Cmd;
                    }
                    if(RunState && AirPressureFlag && OtherFlag)
                    {
                        //check motor fault before turning on
                        I2CSetControlStage(FPCOMMAND_STOP_SPIN_LED,0);
                        I2CSetControlStage(FPCOMMAND_RUN_SPIN_LED,1);
                    }
                    else
                    {
                        I2CSetControlStage(FPCOMMAND_STOP_SPIN_LED,1);
                        I2CSetControlStage(FPCOMMAND_RUN_SPIN_LED,0);
                    }
                    SystemStatus.spin.stat.motor = RunState & AirPressureFlag & OtherFlag;
                    if(RunState && AirPressureFlag && OtherFlag)
                    {       //turn on motor? this is a long proceedure
                        ZeroCount(2);
//                      AbortMove(2);   //this causes goal velocity to be set to 0
                        SpindleDirection = Xio(SPIN_GETDIRECTION,sys.HSpindle,(char *)0,(char *)
0,0l,0);    //get current direction
                        //get acceleration parameter
```

```
                    Accel = Xio(SPIN_GETACCEL,sys.HSpindle,(char *)0,(char *)0,0l,0);
                    //set the accel by using the SCurve Time
                    Set_SCurve(2,(long)Accel);
                    Goal = (long)Getc(sys.HSpindle);    //get current set rpm
                    Set_Goal(2,-Goal);      //set desired RPM
                                            //just the way it worked out
                                            //Goal needs to be negated
                    Set_PhaseDetectorPolarity(2,(long)(SpindleDirection ^ 1));
                    SpinChipPllMode(SPIN_PLLDSP);   //select DSP for processing
                    //set gain to max
                    Xio(SPIN_SETDAC,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_LOOPGAIN,255);
                    //set limits to max
                    Xio(SPIN_SETDAC_DIRECT,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_LOWERLIM
ITS,127);
                    Xio(SPIN_SETDAC_DIRECT,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_UPPERLIM
ITS,127);
                    //set filter rolloff to none
                    Xio(SPIN_SETROLLOFF,sys.HSpindle,(char *)0,(char *)0,0l,0);
                    //enable servo loop
                    ZeroIntegrator(2);  //zap integrator first
                    EnableServoFunctions(2,ENABLE_PID,1);
                    //Turn On Power Amp
                    Xio(SPIN_MOTOR,sys.HSpindle,(char *)0,(char *)0,0l,RunState);
                    //Clear Semaphore
                    SpinDoneSemaphore->SetCount(0);
                    //Command Move
                    DoMoveRel(2);
                    //Wait for move done
                    if(SpinDoneSemaphore->Pend() == EVENT_ABORTED)
                    {
                        AbortMove(2);
                        Set_Goal(2,Goal*2); //set desired RPM
                                            //just the way it worked out
                                            //Goal needs to be negated
                                            //add just a bit more to insure direction change
s
                        DirectionChanged->SetCount(0);
                        //Command Move
                        DoMoveRel(2);
                        //Wait for move done
                        DirectionChanged->Pend();
                        AbortMove(2);
                        Xio(SPIN_MOTOR,sys.HSpindle,(char *)0,(char *)0,0l,0);  //stop motor
                        I2CSetControlStage(FPCOMMAND_STOP_SPIN_LED,1);
                        I2CSetControlStage(FPCOMMAND_RUN_SPIN_LED,0);
                        RunState = 0;   //motor is not running
                    }
                    else
                    {
                        Putc(sys.HSpindle,(int)Goal);   //set rpm
                        if(Goal > 511)
                            SpinChipPllMode(SPIN_PLL);      //set to pll mode
                    }
                }
                else    //turn off motor
                {
                    error = Status(sys.HSpindle,(char *)0,0l,0);    //is motor enabled?
                    if(error & SPIN_MOTOR_STATUS)
                    {
                        Accel = Xio(SPIN_GETDECEL,sys.HSpindle,(char *)0,(char *)0,0l,0);
                        //set the accel by using the SCurve Time
                        Set_SCurve(2,(long)Accel);
                        Goal = (long)Getc(sys.HSpindle);    //get current set rpm
                        Set_Goal(2,Goal*2); //set desired RPM
                                            //just the way it worked out
                                            //Goal needs to be negated
```

```
                                                    //add just a bit more to insure direction change
s
                            ZeroIntegrator(2);              //zap integrator
                            SpinChipPllMode(SPIN_PLLDSP);   //select DSP for processing
                            //set gain to max
                            Xio(SPIN_SETDAC,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_LOOPGAIN,25
5);
                            //set limits to max
                            Xio(SPIN_SETDAC_DIRECT,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_LOWE
RLIMITS,127);
                            Xio(SPIN_SETDAC_DIRECT,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_UPPE
RLIMITS,127);
                            //set filter rolloff to none
                            Xio(SPIN_SETROLLOFF,sys.HSpindle,(char *)0,(char *)0,0l,0);
                            //Clear Semaphore
                            SpinDoneSemaphore->SetCount(0);
                            DirectionChanged->SetCount(0);
                            //Command Move
                            DoMoveRel(2);
                            //Wait for move done
                            DirectionChanged->Pend();   //we don't care why this was fired off (
                                                        //direction change, abort, who cares
                            AbortMove(2);   //prevent erroneous post of SpinDoneSemaphore
                            Xio(SPIN_MOTOR,sys.HSpindle,(char *)0,(char *)0,0l,0);  //stop motor
                            BrakesDone->Post(0);        //indicate braking proceedure done
                        }
                    }
                }
                break;
            case MOTORSPINDLE_DIRECTION_COMMAND:
                if(!RunState)
                {
                    if((PanelEnableFlag && NotLockedOutFlag) || Type)          //ARE we allowed
 to do anything?
                    {
                        if(Type == 0)   //front panel command
                        {
                            I2CBeepTheBuzzer(1,2);
                            if(Cmd == 0)    //is button pushed down?
                            {
                                DirectionState ^= 1;        //toggle state
                            }
                        }
                        else    //internal command
                        {
                            DirectionState = Cmd;
                        }
                        if(DirectionState)
                        {
                            I2CSetControlStage(FPCOMMAND_CCW_LED,0);
                            I2CSetControlStage(FPCOMMAND_CW_LED,1);
                        }
                        else
                        {
                            I2CSetControlStage(FPCOMMAND_CCW_LED,1);
                            I2CSetControlStage(FPCOMMAND_CW_LED,0);
                        }
                        SystemStatus.spin.stat.direction = DirectionState?1:0;
                        Xio(SPIN_SETDIRECTION,sys.HSpindle,(char *)0,(char *)0,0l,DirectionState
);
                    }
                }
                break;
            case MOTORSPINDLE_SETRPM_COMMAND:
                {
                    union {
```

```
                    int a;
                    char c[2];
                }convert;
                int nvel;

                convert.c[0] = Cmd;
                convert.c[1] = SpindleMotorPipe->Get();
                //-----------------------------------------------------
                // start sequence to change motor speed
                //-----------------------------------------------------
                if(RunState)    //is motor already running?
                {
                    Goal = (long)Getc(sys.HSpindle);    //get current set rpm
                    //get acceleration parameter
                    if((nvel = ((int)Goal - convert.a)) < 0)    //speed increase
                        Accel = Xio(SPIN_GETACCEL,sys.HSpindle,(char *)0,(char *)0,0l,0);
                    else if (nvel > 0) //speed decrease
                        Accel = Xio(SPIN_GETDECEL,sys.HSpindle,(char *)0,(char *)0,0l,0);
                    else        //no speed change
                        goto nochange;
                    //set the accel by using the SCurve Time
                    Set_SCurve(2,(long)Accel);
                    Set_Goal(2,(long)nvel);     //set new velocity
                    ZeroIntegrator(2);          //zap integrator
                    SpinChipPllMode(SPIN_PLLDSP);   //select DSP for processing
                    //set limits to max
                    Xio(SPIN_SETDAC_DIRECT,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_LOWERLIM
ITS,127);
                    Xio(SPIN_SETDAC_DIRECT,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_UPPERLIM
ITS,127);
                    //set gain to max
                    Xio(SPIN_SETDAC,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_LOOPGAIN,255);
                    //set filter rolloff to none
                    Xio(SPIN_SETROLLOFF,sys.HSpindle,(char *)0,(char *)0,0l,0);
                    //Clear Semaphore
                    SpinDoneSemaphore->SetCount(0);
                    //Command Move
                    DoMoveRel(2);
                    //Wait for move done
                    if(SpinDoneSemaphore->Pend() == EVENT_ABORTED)
                    {
                        AbortMove(2);
                        goto nochange;
                    }
                }
                Putc(sys.HSpindle,convert.a);
                if(RunState && (convert.a > 511))
                    SpinChipPllMode(SPIN_PLL);      //set to pll mode
            }
            break;
        case MOTORSPINDLE_EMSTOP_COMMAND:   //emergency Stop command
            error = Status(sys.HSpindle,(char *)0,0l,0);    //is motor enabled?
            if(error & SPIN_MOTOR_STATUS)
            {
                //set limits to max
                Xio(SPIN_SETDAC_DIRECT,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_LOWERLIMITS,
127);
                Xio(SPIN_SETDAC_DIRECT,sys.HSpindle,(char *)0,(char *)0,0l,SPIN_UPPERLIMITS,
127);
                DirectionChanged->SetCount(0);  //clear semaphore before use
                SpindleDirection = Xio(SPIN_GETDIRECTION,sys.HSpindle,(char *)0,(char *)0,0l
,0);    //get current direction
                TDelay(5);
                Xio(SPIN_SETDIRECTION,sys.HSpindle,(char *)0,(char *)0,0l,SpindleDirection ^
 0x01); //change direction
                DirectionChanged->Pend(20); //wait for direction to actually change
```

```
                    Xio(SPIN_MOTOR,sys.HSpindle,(char *)0,(char *)0,0l,0);  //stop motor
                    Xio(SPIN_SETDIRECTION,sys.HSpindle,(char *)0,(char *)0,0l,SpindleDirection);
 //change direction
                    BrakesDone->Post(0);        //indicate braking proceedure done
                    I2CSetControlStage(FPCOMMAND_STOP_SPIN_LED,1);
                    I2CSetControlStage(FPCOMMAND_RUN_SPIN_LED,0);
                    AbortMove(2);   //stop anything happening in profile generator
                    ZeroCount(2);   //zip zap position, may have to change latter
                    RunState = 0;
                }
                break;
        }
nochange:;

    }
}


static void ContinueButton(void)
{
    //-----------------------------------------
    //
    // this function will "send" a message through
    // another "event" to let a waiting task know
    // that the continue button has been pushed
    //
    //-----------------------------------------
    //
    int cmd;
    int Type;
//  char *s = new char[256];
//  int c;

    while(1)        //this is a TASK, loop forever
    {
        ContinuePipe->Pend();
        Type = ContinuePipe->Get(); //where did command come from
        cmd = ContinuePipe->Get();  //get state of switch
//      c = sprintf(s,"Continue Button:%d\r\n");
//      Write(ConsolHandle,s,c);
        if(cmd)     //is switch released?
            WaitForContinueButton->Post();  //post semaphore
        else
            I2CBeepTheBuzzer(1,2);

    }
}

static void VacuumButton(void)
{
    //This is the task to handle the VacuumButton
    int cmd,Type,State;

    State=0;
    I2CSetControlStage(FPCOMMAND_VACCUUM_OFF_LED,State ^ 1);
    I2CSetControlStage(FPCOMMAND_VACUUM,State);
    I2CSetControlStage(FPCOMMAND_VACCUUM_ON_LED,State);
    while(1)
    {
        VacuumButtonPipe->Pend();
        Type = VacuumButtonPipe->Get();
        cmd = VacuumButtonPipe->Get();
        if((PanelEnableFlag && NotLockedOutFlag) || Type)          //ARE we allowed to do anyth
ing?
        {
            if(Type == 0)
```

```
            {
                I2CBeepTheBuzzer(1,2);
                if(cmd == 0)            //was button pushed down?
                {
                    State ^= 1;     //change state of state
                    I2CSetControlStage(FPCOMMAND_VACCUUM_OFF_LED,State ^ 1);
                    I2CSetControlStage(FPCOMMAND_VACUUM,State);
                    I2CSetControlStage(FPCOMMAND_VACCUUM_ON_LED,State);
                }
            }
            else
            {
                State = cmd;
                I2CSetControlStage(FPCOMMAND_VACCUUM_OFF_LED,State ^ 1);
                I2CSetControlStage(FPCOMMAND_VACUUM,State);
                I2CSetControlStage(FPCOMMAND_VACCUUM_ON_LED,State);
            }
            SystemStatus.spin.stat.vac = State?1:0;
        }
    }
}

static void ConClampButton(void)
{
    //This is the task to handle the connector clamp button
    int cmd,Type,State;

    State=0;
    I2CSetControlStage(FPCOMMAND_CONCLAMP_UP_LED,State ^ 1);
    I2CSetControlStage(FPCOMMAND_CONCLAMP,State);
    I2CSetControlStage(FPCOMMAND_CONCLAMP_DOWN_LED,State);
    while(1)
    {
        ConnClampPipe->Pend();
        Type = ConnClampPipe->Get();
        cmd = ConnClampPipe->Get();
        if((PanelEnableFlag && NotLockedOutFlag) || Type)           //ARE we allowed to do anyth
ing?
        {
            if(Type == 0)
            {
                I2CBeepTheBuzzer(1,2);
                if(cmd == 0)            //was button pushed down?
                {
                    State ^= 1;     //change state of state
                    I2CSetControlStage(FPCOMMAND_CONCLAMP_UP_LED,State ^ 1);
                    I2CSetControlStage(FPCOMMAND_CONCLAMP,State);
                    I2CSetControlStage(FPCOMMAND_CONCLAMP_DOWN_LED,State);
                }
            }
            else
            {
                State = cmd;
                I2CSetControlStage(FPCOMMAND_CONCLAMP_UP_LED,State ^ 1);
                I2CSetControlStage(FPCOMMAND_CONCLAMP,State);
                I2CSetControlStage(FPCOMMAND_CONCLAMP_DOWN_LED,State);
            }
            SystemStatus.spin.stat.conclamp = State?1:0;
        }
    }
}

static void StopButton(void)
{
    //----------------------------------------
    //
```

```
    // This function must check to see what is
    // running and execute any sequence that is
    // needed to bring the spinstand to some
    // benign state.  I.E., stop spindle motor,
    // stop stages, etc.
    //
    //-----------------------------------------
    //
    int cmd;
    int Type;
    int i,loop;

    while(1)        //this is a TASK, loop forever
    {
        StopPipe->Pend();
        Type = StopPipe->Get(); //where did command come from
        cmd = StopPipe->Get();  //was switch pushed or released
        if(cmd==0)  //if pushed
        {
            I2CBeepTheBuzzer(1,2);
            BigTimeShutDown(ACUTRAC_USERABORT);
            SystemStatus.spin.stat.stopbutton = 1;  //set stop button flag
        }
    }
}

void BigTimeShutDown(int why)
{
    /*************************************************************************
    ** this function is used by the STOP BUTTON, AIR PRESURE and AMP FAULT
    ** tasks to shut down the system
    *************************************************************************/
    TCB *AT;        //pointer to action task
    int i,loop,StageState;

    switch(DeviceTable->mode)   //what mode are we in
    {
        case HSA:
        case HGA:
            //-------------------------------------------------------------
            // start of HSA shut down sequence
            //-------------------------------------------------------------
        // check to see if ActionTask is active (allocated)
            if((AT = GetActionTask()) != NULL)
            {
        // check to see which Action Task is being used
                for(i=0,loop=1;ActionTaskStrings[i] && loop;++i)
                {
                    if(strcmp(AT->name,ActionTaskStrings[i]) == 0)
                    loop = 0;
                }
        // Determine state of spinstand
                switch(i-1)
                {
                    case 0:     //start
                        break;
                    case 1:     //stop
                        break;
                    case 2:     //reset
                        break;
                    case 3:     //error
                        break;
                    default :
                        break;
                }
        // Terminate ActionTask
```

```
                TDelete(AT);    //delete task
                AcutracSETStatus(why);
                AcutracPrintErrorMessage(why,AcutracGetErrorBuffer());

        // Take appropriate action
                StageState = I2CGetStageStatus();

                if(!(StageState & FPCOMMAND_VAC))   //is vacume engaged
                {
                    //Yes
                    if((StageState & FPCOMMAND_RAMP_OUT))   //are heads loaded
                    {
                        //-------------------------------
                        // goto load / unload radius
                        //-------------------------------
                        AcutracGotoLoadUnloadRadius(ACUTRAC_UNLOADRADIUS);
                        //-------------------------------
                        // Unload Heads
                        //-------------------------------
                        AcutracUnloadHeads();
                    }
                }

                StageEmergencyMotorStop();  //turn off motor
            }
            else
            {  // if no actiontask, check current state of spinstand


                StageState = I2CGetStageStatus();

        // take appropriate action
                if(!(StageState & FPCOMMAND_VAC))   //is vacume engaged
                {
                    //Yes
                    if((StageState & FPCOMMAND_RAMP_OUT))   //are heads loaded
                    {
                        //-------------------------------
                        // goto load / unload radius
                        //-------------------------------
                        AcutracGotoLoadUnloadRadius(ACUTRAC_UNLOADRADIUS);
                        //-------------------------------
                        // Unload Heads
                        //-------------------------------
                        AcutracUnloadHeads();
                    }
                }
                StageEmergencyMotorStop();  //turn off motor
            }
            //------------------------------------------------------------
            // end of HSA shut down sequence
            //------------------------------------------------------------
            break;
        default :    //no device is loaded
            StageEmergencyMotorStop();
            break;
    }   //end of switch(DeviceTable->mode)
}

static void AdjustEnable(void)
{
    int Cmd;
    int State = 0;
    int Type;

    I2CSetControlStage(FPCOMMAND_RPM_LED,1);
```

```
    while(1)        //this is a TASK, loop forever
    {
        AdjustEnablePipe->Pend();
        Type = AdjustEnablePipe->Get(); //where did command come from
        Cmd = AdjustEnablePipe->Get();
        if(Cmd == 0)    //is button pushed down?
        {
            State ^= 1;     //toggle state
            if(State)
            {
                I2CSetControlStage(FPCOMMAND_RPM_LED,0);
                I2CSetControlStage(FPCOMMAND_RADIUS_LED,1);
            }
            else
            {
                I2CSetControlStage(FPCOMMAND_RPM_LED,1);
                I2CSetControlStage(FPCOMMAND_RADIUS_LED,0);
            }
            AdjustFlag = State;
        }
    }
}

static void PanelEnable(void)
{
    int Type;
    extern volatile int DisplayMode;

    while(1)        //this is a TASK, loop forever
    {
        PanelEnablePipe->Pend();
        Type = PanelEnablePipe->Get();  //where did command come from
        PanelEnableFlag = PanelEnablePipe->Get() ^ 0x01;    //complement flag
        if(PanelEnableFlag)
        {
            if(DisplayMode == 0)
            {
                if(AdjustFlag==0)
                {
                    Xio(QUAD_MODE,sys.HQuad,(char *)0,(char *)0,0l,QUAD_MODE_NORMAL);
                    DisplayMode = 1;
                }
                else if(AdjustFlag == 1)
                {
                    Xio(QUAD_MODE,sys.HQuad,(char *)0,(char *)0,0l,QUAD_MODE_FAST);
                    DisplayMode=2;
                }
            }
        }
        else
        {
            if(DisplayMode != 3)

                DisplayMode = 0;
            //--------------------------------
            // flush Quad Knob
            //--------------------------------
        }
    }
}

static void AirPresure(void)
{
    //-----------------------------------------------
    //
    // This is sort of like a STOP, but even more critical
```

```
    // This will unload the heads immeadiately and
    // Then stop the spindle motor
    // Then it will do anything else it may need to do
    //
    //-------------------------------------------------
    int cmd;
    int c;
    //-------------------------------------------------
    // check air pressure switch
    //-------------------------------------------------
    TDelay(30);
    c = I2CGetStageStatus();
    if(!(c & FPCOMMAND_AIR))    //is there air pressure?
    {
        AirPressureFlag = 1;
        SystemStatus.spin.stat.air = 0;
    }
    else
    {
        SystemStatus.spin.stat.air = 1;
    }
    NotLockedOutFlag = 1;   //no need to lock out front panel

    while(1)        //this is a TASK, loop forever
    {
        AirPresurePipe->Pend();
        cmd = AirPresurePipe->Get();    //get state of air switch
        SystemStatus.spin.stat.air = cmd;
        if(cmd == 1)    //did air pressure go off?
        {
            BigTimeShutDown(ACUTRAC_LOWAIR);
        }
        AirPressureFlag = cmd ^ 1;
    }
}

static void RampIn(void)
{
    //-------------------------------------------------
    // When the ramp is in a position that the heads
    // will be on the disk, this function will Post
    // a semaphore
    //*************************************************
    // New for ver 1.52.00
    // This function needs to be configurable.
    // for cmd values greater than 15, these will be
    // configuration commands
    // They will be defined in i2c.h
    //-------------------------------------------------
    int cmd;
    char *s = new char[256];
    int c;
    int mode = RAMPMODE_HSA;    //default is HSA mode

    while(1)    //This is a TASK, so loop forever
    {
        RampInPipe->Pend();
        cmd = RampInPipe->Get();
        switch(cmd)
        {
            case 0:
            case 1:     //these are the values recieved from
                        //the I2C dispatcher indicating an
                        //event has occured on the RampIn sensor
                if(cmd && (mode == RAMPMODE_HSA))   //always post true value
                {
```

```
                    c = sprintf(s,"HSA Ramp In\r\n");
                    Write(ConsolHandle,s,c);
                    RampInFlag->Post(0);
                }
                else if(mode == RAMPMODE_HGA)
                {
//                  c = sprintf(s,"HGA Ramp In:%d\r\n",cmd);
//                  Write(ConsolHandle,s,c);
                    RampInFlag->Post(cmd);  //only in HGA mode
                }
                break;
            case RAMPCOMMAND_HSAMODE:
//              c = sprintf(s,"RampIn=HSA mode\r\n");
//              Write(ConsolHandle,s,c);
                mode = RAMPMODE_HSA;
                break;
            case RAMPCOMMAND_HGAMODE:
//              c = sprintf(s,"RampIn=HGA mode\r\n");
//              Write(ConsolHandle,s,c);
                mode = RAMPMODE_HGA;
                break;
        }   //end of switch(cmd)

    }
}

static void RampOut(void)
{
    //-------------------------------------------------
    // When the ramp is in a position to spread the
    // fingers, this function will post a semaphore
    //-------------------------------------------------
    int cmd;
    char *s = new char[256];
    int c;
    int mode = RAMPMODE_HSA;    //default is HSA mode

    while(1)    //This is a TASK, so loop forever
    {
        RampOutPipe->Pend();
        cmd = RampOutPipe->Get();
        switch(cmd)
        {
            case 0:
            case 1:     //these are the values recieved from
                        //the I2C dispatcher indicating an
                        //event has occured on the RampIn sensor
                if(cmd && (mode == RAMPMODE_HSA))   //always post true value
                {
                    c = sprintf(s,"HSA Ramp Out\r\n");
                    Write(ConsolHandle,s,c);
                    RampOutFlag->Post(0);
                }
                else if(mode == RAMPMODE_HGA)
                {
//                  c = sprintf(s,"HGA Ramp Out:%d\r\n",cmd);
//                  Write(ConsolHandle,s,c);
                    RampOutFlag->Post(cmd);
                }
                break;
            case RAMPCOMMAND_HSAMODE:
//              c = sprintf(s,"RampOut=HSA mode\r\n");
//              Write(ConsolHandle,s,c);
                mode = RAMPMODE_HSA;
                break;
            case RAMPCOMMAND_HGAMODE:
```

```
//              c = sprintf(s,"RampOut=HGA mode\r\n");
//              Write(ConsolHandle,s,c);
                mode = RAMPMODE_HGA;
                break;
        }   //end of switch(cmd)
    }
}

static void ClampUp(void)
{
    //-------------------------------------------------
    // When the connector clamp has released the
    // connector, this function will post a
    // semphore
    //-------------------------------------------------

    int cmd;
    char *s = new char[256];
    int c;

    while(1)    //this is a TASK, loop forever
    {
        ClampUpPipe->Pend();
        cmd = ClampUpPipe->Get();
        if(cmd == 0)    //we only care if clamp is up
        {
            c = sprintf(s,"ClampUp:%d\r\n",cmd);
            Write(ConsolHandle,s,c);
        }
        if(DeviceTable)
        {
            if(DeviceTable->mode == HGA)
                ClampUpFlag->Post(cmd);
            else if(DeviceTable->mode == HSA)
            {
                if(cmd == 0)    //only post when clamp has arrived at dest
                    ClampUpFlag->Post(cmd);
            }
        }
    }
}

static void ClampDown(void)
{
    //-------------------------------------------------
    // When the connector clamp has secured the
    // connector, this function will post a semaphore
    //-------------------------------------------------

    int cmd;
    char *s = new char[256];
    int c;
    while(1)    //this is a TASK, loop forever
    {
        ClampDownPipe->Pend();
        cmd = ClampDownPipe->Get();
        if(cmd == 0)    //we only care if clamp is down
        {
            c = sprintf(s,"ClampDown:%d\r\n",cmd);
            Write(ConsolHandle,s,c);
        }
        if(DeviceTable)
        {
            if(DeviceTable->mode == HGA)
                ClampDownFlag->Post(cmd);
            else if(DeviceTable->mode == HSA)
```

```
            {
                if(cmd == 0)    //only post when clamp has arrived at dest
                    ClampDownFlag->Post(cmd);
            }
        }

    }
}

static void Vaccuum(void)
{
    int cmd;
    while(1)    //This is a TASK, so loop forever
    {
        VaccumPipe->Pend();
        cmd = VaccumPipe->Get();
        //-------------------------------------
        // cmd == 0 is SUCK
        // cmd == 1 is BLOW
        //-------------------------------------
        WaitForSuck->Post(cmd);
    }
}

static void CombCorral(void)
{
    char *s = new char[256];
    int c;

    int cmd;
    while(1)    //This is a TASK, so loop forever
    {
        CombCoralPipe->Pend();
        cmd = CombCoralPipe->Get();
        //-----------------------------------------------------
        // if cmd==0, something just went into the comb coral
        // if cmd==1, something just came out of the comb coral
        //-----------------------------------------------------
//      c = sprintf(s,"Comb CMD = %d\r\n",cmd);
//      Write(ConsolHandle,s,c);
        WaitForComb->Post(cmd);
    }
}

static void LastCommandGarbage(void)
{
    while(1)        //this is a TASK, loop forever
    {
        LastCommandGarbagePipe->Pend();
    }
}


//-------------------------------------------------------------------------
// Misc Functions
//-------------------------------------------------------------------------

void I2CInitQueue(int size)
{
    I2CPipe = new EventQueue(size,"I2CPipe");               //create data pipe from I2C controll
er
    I2CSendPipe = new EventQueue(size,"I2CSendPipe");           //create data pipe to I2C contro
ller

    XStagePipe = new EventQueue(8,"XStagePipe");                //semaphore for XStage
    HeadLoaderPipe = new EventQueue(8,"HeadLoadPipe");          //semaphore for head loader
```

```
//  SpindleDirectionPipe = new EventQueue(8,"SpinDirPipe"); //semaphore for spindle direction
    ContinuePipe = new EventQueue(8,"ContinuePipe");            //Semaphore for continue button
    StopPipe = new EventQueue(8,"StopPipe");                 //semaphore for stop button
    VacuumButtonPipe = new EventQueue(8,"VacButPipe");  //semaphore for vacuum button
    ConnClampPipe = new EventQueue(8,"ConnClampPipe");       //semaphore for connector clamp

    AdjustEnablePipe = new EventQueue(8,"AdjEnPipe");        //semaphore for adjust enable button
    PanelEnablePipe = new EventQueue(8,"PanelEnPipe");       //semaphore for adjust panel button
    AirPresurePipe = new EventQueue(8,"AirPresPipe");            //semaphore for air pressure low
 sensor
    WaitForContinueButton = new Wait(0,"WaitForCont");       //semaphore for waiting for continue
 button
    SpindleMotorPipe = new EventQueue(256,"SpinMotPipe");        //semaphore for starting spindle
 motor
    CombCoralPipe = new EventQueue(8,"CombCoralPipe");           //semaphore for the comb corral
    RampInPipe = new EventQueue(8,"RampInPipe");                 //semaphore for Ramp In Sensor
    RampOutPipe = new EventQueue(8,"RampOutPipe");           //semaphore for Ramp Out Sensor
    ClampUpPipe = new EventQueue(8,"ClampUpPipe");               //semaphore for Connector Clamp
Sensor
    ClampDownPipe = new EventQueue(8,"ClampDownPipe");               //semaphore for Connector Cl
amp Sensor

    VaccumPipe = new EventQueue(8,"VacPipe");                //semaphore for Vacuum Sensor
    SpinstandState = new EventQueue(8,"SpinstandState");             //semaphore for spinstand st
atus
    SpinstandVersion = new EventQueue(8,"SpinStandVersion");         //semaphore for getting spin
stand version

    I2COutputMessageQueue = new BQueue(32,32,"I2COutputQueue"); //32 messages, 32 bytes long
    RampInFlag = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"RampInFlag");             //semaphore
for RampIn flag
    RampOutFlag = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"RampOutFlag");           //semaphore
for RampOut flag
    ClampUpFlag = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"ClampUpFlag");           //semaphore
for ClampUp flag
    ClampDownFlag = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"ClampDownFlag");       //semaphore
for ClampDown flag


    WaitForI2CXmit = new Wait(0,"WaitForI2CXmit");               //semaphore for I2C transmit Dea
mon
    WaitForSuck = new Wait(0,"WaitForSuck");                     //semaphore for Vac Sensor
    WaitForComb = new Wait(0,"WaitForComb");                     //semaphore for waiting for comb
}

//---------------------------------------------------------------------------
// Initialize I2C port
//---------------------------------------------------------------------------

extern "C" {

void I2CInit(void)
{
    TCB *t;

    I2CInitQueue(256);

    t = CreateTask(XStage,512,13,"Move_XStage");
    ActiveTasks->Insert(t);
    t = CreateTask(HeadLoader,512,13,"Load_Head");
    ActiveTasks->Insert(t);
    t = CreateTask(SpindleMotor,512,13,"Spindle_Motor");
    ActiveTasks->Insert(t);
    t = CreateTask(ContinueButton,512,13,"Continue");
    ActiveTasks->Insert(t);
    t = CreateTask(StopButton,512,100,"Stop");  //very high priority
```

```
    ActiveTasks->Insert(t);
    t = CreateTask(AdjustEnable,512,13,"Adjust_Enable");
    ActiveTasks->Insert(t);
    t = CreateTask(PanelEnable,512,13,"Panel_Enable");
    ActiveTasks->Insert(t);
    t = CreateTask(AirPresure,512,101,"Air_Pressure");  //very high priority
    ActiveTasks->Insert(t);
    t = CreateTask(RampIn,512,13,"RampIn");
    ActiveTasks->Insert(t);
    t = CreateTask(RampOut,512,13,"RampOut");
    ActiveTasks->Insert(t);
    t = CreateTask(Vaccuum,512,13,"Vac");
    ActiveTasks->Insert(t);
    t = CreateTask(CombCorral,512,13,"CombCorral");
    ActiveTasks->Insert(t);
    t = CreateTask(VacuumButton,512,13,"VaccumButton");
    ActiveTasks->Insert(t);
    t = CreateTask(ConClampButton,512,13,"ClampButton");
    ActiveTasks->Insert(t);
    t = CreateTask(ClampUp,512,13,"ClampUp");
    ActiveTasks->Insert(t);
    t = CreateTask(ClampDown,512,13,"ClampDown");
    ActiveTasks->Insert(t);
    //
    //-----------------------------------------------------------------
    //
    // The I2C message Dispatcher needs to have the highest priority
    // So that it can always post message semaphores
    //
    //-----------------------------------------------------------------
    //
    t = CreateTask(DispatchI2CMessages,512,127,"I2CDispatch");
    ActiveTasks->Insert(t);
    //
    //-----------------------------------------------------------------
    // The I2C Message sending Deamon needs to have a high priority
    // So that Messages that are sent will get sent
    //-----------------------------------------------------------------
    t = CreateTask(I2CSendDemon,512,126,"I2CDemon");
    ActiveTasks->Insert(t);
    //
    //-----------------------------------------------------------------
    //
    // initialize I2C hardware and interrupts
    //
    //-----------------------------------------------------------------
    I2C_CONTROLSTATUS = I2C_PENDING_INTERRUPT_NOT;  //select S0'
    asm(" nop");
    asm(" nop");        //delays needed for pcf8584
    I2C_DATA = I2C_OWNADDRESS >> 1;
    asm(" nop");
    asm(" nop");        //delays needed for pcf8584
    I2C_CONTROLSTATUS = I2C_PENDING_INTERRUPT_NOT | I2C_ES1;    //select reg S2
    asm(" nop");
    asm(" nop");        //delays needed for pcf8584
    I2C_DATA = 0x19;    //8MHz,45khz SCL
    asm(" nop");
    asm(" nop");        //delays needed for pcf8584
    I2C_CONTROLSTATUS = I2C_PENDING_INTERRUPT_NOT | I2C_ES2;    //select reg S3
    asm(" nop");
    asm(" nop");        //delays needed for pcf8584
    I2C_DATA = 0xd1;    //interrupt vector for vector 0x344
    asm(" nop");
    asm(" nop");        //delays needed for pcf8584
    I2C_CONTROLSTATUS = I2C_PENDING_INTERRUPT_NOT | I2C_ENABLE_SERIAL_OUTPUT  | I2C_ENABLE_INTER
RUPT  | I2C_ACK;
```

```
}

//---------------------------------------------------------------------------
// Send a Message from the I2C port
// this function takes a string and size and sends it to the I2C port
// This function calculates the checksum to be sent.
// size would be one less that the lenth you would find in the
// message packet.
//---------------------------------------------------------------------------


void I2CSendMessage(unsigned char *d,int size)
{
    int i;
    unsigned char checksum = 0;
    for(i=0;i<size;++i,++d)
    {
        I2CSendPipe->Put(*d);
        checksum ^= *d;
    }
    I2CSendPipe->Put(checksum); //send checksum
    i = I2CSendPipe->Get(); //get first byte and send it.
*((volatile int *)0xffcc0000) = ~0x00a0;
    while(!(I2C_CONTROLSTATUS & I2C_BUS_BUSY)); //wait for bus not busy
*((volatile int *)0xffcc0000) = ~0x02;
    I2C_DATA = i;   //get first byte and send it.
    I2C_CONTROLSTATUS = I2C_PENDING_INTERRUPT_NOT | I2C_ENABLE_SERIAL_OUTPUT | I2C_ACK | I2C_STA
RT | I2C_ENABLE_INTERRUPT;  //maybe need more, 0xc0
    XmitFlag=1;
}

//---------------------------------------------------------------------------
// Write a string to the LED front panel, up to 12 characters
//---------------------------------------------------------------------------

void I2CWriteLEDString(char *s)
{
    unsigned char *t = new unsigned char[32];
    int c;

    c = sprintf((char *)t,"%c%c%c%c%12s",I2C_FRONTPANEL,I2C_OWNADDRESS,17,FPCOMMAND_DISPLAY_DATA
,s);
    I2COutputMessageQueue->SendMessage((char *)t,c);
    delete[] t;
}

//---------------------------------------------------------------------------
// Beep the buzzer on the front panel
// c=number of times.
// t = time delay
//---------------------------------------------------------------------------

int I2CBeepTheBuzzer(int count,int t)
{
    int i,c;
    unsigned char *s = new unsigned char[20];
    int retval;

    for(i=0;i<count;++i)
    {
        if((retval = TDelay(t)) < 0) goto exit;
        c = sprintf((char *)s,"%c%c%c%c%c",I2C_FRONTPANEL,I2C_OWNADDRESS,6,FPCOMMAND_BUZZER_STAT
E,1);
        I2COutputMessageQueue->SendMessage((char *)s,c);
        if((retval = TDelay(t)) < 0) goto exit;
        c = sprintf((char *)s,"%c%c%c%c%c",I2C_FRONTPANEL,I2C_OWNADDRESS,6,FPCOMMAND_BUZZER_STAT
```

```
E,0);
        I2COutputMessageQueue->SendMessage((char *)s,c);

    }
exit:
    delete[] s;
    return retval;
}

//---------------------------------------------------------------------------
// Set the state of a control line]
//---------------------------------------------------------------------------

void I2CSetControlStage(int control, int state)
{
    String *S = new String;
    char *s = S->Get();
    int c;

    c = sprintf(s,"%c%c%c%c%c%c",I2C_FRONTPANEL,I2C_OWNADDRESS,7,FPCOMMAND_SET_CONTROL_STATE,con
trol,state);
    I2COutputMessageQueue->SendMessage(s,c);
    delete S;
}

//---------------------------------------------------------------------------
// Get the Status of the stage port
//---------------------------------------------------------------------------

int I2CGetStageStatus(void)
{
    String *S = new String;
    char *s = S->Get();
    int c;
    int retval;

    c = sprintf(s,"%c%c%c%c",I2C_FRONTPANEL,I2C_OWNADDRESS,5,FPCOMMAND_SPINSTAND_STATE_SEND);
    I2COutputMessageQueue->SendMessage(s,c);    //send command
    if((retval = SpinstandState->Pend()) != EVENT_TERMINATE)
        retval = SpinstandState->Get();
    delete S;
    return retval;
}

//---------------------------------------------------------------------------
// get the version number of the Front Panel Software
//---------------------------------------------------------------------------

unsigned I2CGetStageVersion(void)
{
    String *S = new String;
    char *s = S->Get();
    int c;
    union {
        char ca[2];
        unsigned rv;
    } retval;

    c = sprintf(s,"%c%c%c%c",I2C_FRONTPANEL,I2C_OWNADDRESS,5,FPCOMMAND_VERSION_SEND);
    I2COutputMessageQueue->SendMessage(s,c);    //send command
    SpinstandVersion->Pend();                   //wait for version into to return
    retval.ca[0] = SpinstandVersion->Get();     //get MSB
    retval.ca[1] = SpinstandVersion->Get();     //get LSB
    delete S;
    return retval.rv;
}
```

```
//---------------------------------------------------------------------------
// Interface Functions for manipulating the Stage
//---------------------------------------------------------------------------

int StageSetXStage(int c)
{
    int Msg[2];
    int retval = 0;

    if(!AirPressureFlag)
        retval = SPINDLE_NAK;       //refuse if no air
    else
    {
        Msg[0] = I2C_MESSAGE_INTERNAL;
        Msg[1] = c?1:0;
        XStagePipe->PostMessage(Msg,2);
    }
    return retval;
}

int StageSetDirection(int c)
{
    int Msg[3];
    int retval = 0;

    Msg[0] = I2C_MESSAGE_INTERNAL;
    Msg[1] = MOTORSPINDLE_DIRECTION_COMMAND;
    Msg[2] = c?1:0;
    if(SpindleMotorPipe->QueueSpace() > 3)
        SpindleMotorPipe->PostMessage(Msg,3);
    else
        retval = SPINDLE_NAK;
    return retval;
}

extern int StageSelectHead(int c)
{
    String *S = new String;
    char *s = S->Get();
    int count;

    count = sprintf(s,"%c%c%c%c%c%c",I2C_FRONTPANEL,I2C_OWNADDRESS,6,FPCOMMAND_SELECT_HEAD,c);
    I2COutputMessageQueue->SendMessage(s,count);
    delete S;
    return 0;
}

int StageEmergencyMotorStop(void)
{
    //makes sure that the motor will stop
    int Msg[3];
    int retval = 0;
    //----------------------------------------
    // first, check to see if we are waiting
    // for servo loop to spin up motor
    //----------------------------------------
    if(SpinDoneSemaphore->GetCount() < 0)
        SpinDoneSemaphore->Post(EVENT_ABORTED); //cause event to abort
    else if(DirectionChanged->GetCount() < 0)
        DirectionChanged->Post(EVENT_ABORTED);  //cause event to abort
    else if(DirectionChanged->GetCount() >= 0)  //not already waiting for stop
    {
        Msg[0] = I2C_MESSAGE_INTERNAL;
        Msg[1] = MOTORSPINDLE_EMSTOP_COMMAND;
        Msg[2] = 0;
```

```
        SpindleMotorPipe->PostMessage(Msg,3);
    }
    return retval;
}

int StageSetMotor(int c)
{
    int Msg[3];
    int retval = 0;
    //---------------------------------------------------------------
    // check the status of the air pressure
    // if there is no air pressure, do not allow motor to start
    // allways allow motor to shut off
    //---------------------------------------------------------------
    if(!(AirPressureFlag) && c) //is the air off and motor on command?
        retval = SPINDLE_NAK;       //indicate command refused
    else
    {
        Msg[0] = I2C_MESSAGE_INTERNAL;
        Msg[1] = MOTORSPINDLE_STARTSTOP_COMMAND;
        Msg[2] = c?1:0;
        if(SpindleMotorPipe->QueueSpace() > 3)
            SpindleMotorPipe->PostMessage(Msg,3);
        else
            retval = SPINDLE_NAK;
    }
    return retval;
}

int StageSetRpm(int rpm)
{
    int Msg[4];
    int retval = 0;
    union {
        int a;
        char c[2];
    } convert;
    static int state=0;

    convert.a = rpm;
    Msg[0] = I2C_MESSAGE_INTERNAL;
    Msg[1] = MOTORSPINDLE_SETRPM_COMMAND;
    Msg[2] = convert.c[0];
    Msg[3] = convert.c[1];  //crude way to pass int
    int n = SpindleMotorPipe->QueueStatus();    //get number of bytes
    if(!state)  //check state
    {
        if(n > 200)     //past high water mark?
        {
            state = 1;  //flip state
            retval = SPINDLE_NAK;
        }
        else
        {
            SpindleMotorPipe->PostMessage(Msg,4);
        }
    }
    else
    {
        retval = SPINDLE_NAK;
        if(n < 50)  //low water mark
        {
            state = 0;      //flip state back
        }
    }
    return retval;
```

```
}


int StageLoadHeads(int c)
{
    int Msg[2];
    int retval = 0;

    if(!AirPressureFlag)
        retval = SPINDLE_NAK;       //indicate command refused
    else
    {
        Msg[0] = I2C_MESSAGE_INTERNAL;
        Msg[1] = c?1:0;
        HeadLoaderPipe->PostMessage(Msg,2);
    }
    return retval;
}

int StageConnectorClamp(int c)
{
    int retval=0;
    int Msg[2];

    if(!AirPressureFlag)
        retval = SPINDLE_NAK;
    else
    {
        Msg[0] = I2C_MESSAGE_INTERNAL;
        Msg[1] = c?1:0;
        ConnClampPipe->PostMessage(Msg,2);
    }
    return retval;
}

int StageSuckHeads(int c)
{
    int retval=0;
    int Msg[2];

    if(!AirPressureFlag)
        retval = SPINDLE_NAK;
    else
    {
        Msg[0] = I2C_MESSAGE_INTERNAL;
        Msg[1] = c?1:0;
        VacuumButtonPipe->PostMessage(Msg,2);
    }
    return retval;
}

//---------------------------------------------------------------------------
// Local Functions
//---------------------------------------------------------------------------

void I2CHandleInterrupt(void)
{
    //-------------------------------------------------------
    //This code is going to be fun
    //
    //Recieve Interrupt:
    //If this is the addressed device, then stuff data into
    //a message pipe.  When Pipe is full, post semaphore
    //to activate dispatcher of messages
    //
    //Transmit Interrupt:
```

```
    //Send data from pipe until the pipe is empty, then send
    //STOP.
    //
    //-------------------------------------------------------
    int stat;
    int clearflag=0;

    stat = I2C_CONTROLSTATUS;   //read in status port
    if(stat & I2C_LOST_ARBITRATION)
    {
        clearflag = 1;              //we must do software reset later
        XmitFlag = 0;               //clear transmit flag
        WaitForI2CXmit->Post(1);    //non zero value means retransmit
    }
    if(stat & I2C_ADDRESSED_AS_SLAVE)   //start of recieve frame?
    {
        RecFlag = 1;
    }
    if(RecFlag)
    {
        int c = I2C_DATA;               //read data
        if(clearflag)
        {
            asm(" nop");        //must do software reset but first
            asm(" nop");
            asm(" nop");        //delays needed for pcf8584
            I2C_CONTROLSTATUS = I2C_PENDING_INTERRUPT_NOT | I2C_ENABLE_SERIAL_OUTPUT | I2C_ACK |
 I2C_ENABLE_INTERRUPT;
        }
        if(stat & I2C_STOP_SENSED)  //check stop flag first
        {
            RecFlag = 0;
            c = I2C_DATA;
            I2CPipe->Post();                //process I2C Message
        }
        else
        {
            I2CPipe->Put(c);                    //get data, put in pipe
        }
    }
    else
    {
        if(I2CSendPipe->QueueStatus()==0)
        {
            I2C_CONTROLSTATUS = I2C_PENDING_INTERRUPT_NOT | I2C_ENABLE_SERIAL_OUTPUT | I2C_ACK |
 I2C_STOP | I2C_ENABLE_INTERRUPT;
*((volatile int *)0xffcc0000) = 0x00;
            XmitFlag = 0;   //end of transmit mode
            WaitForI2CXmit->Post(0);
        }
        else
        {
            I2C_DATA = I2CSendPipe->Get();  //get first byte and send it.
        }
    }
}

static void DispatchI2CMessages(void)
{
    //-------------------------------------------------------
    // This code will get a message from a pipe.  That message
    // will tell this code what it is that needs to be done.
    // This function will then activate the appropriate
    // Semaphore to do the particular Job.
    //-------------------------------------------------------
    int Msg[20];    //message buffer
```

```
    int OMsg[3];    //output message

    int i;
    unsigned char checksum;
    unsigned char destadr;
    unsigned char souradr;
    unsigned count;
    unsigned char c;

    OMsg[0] = I2C_MESSAGE_FP;   //mark message from front panel
    while(1)    //this is a TASK, loop forever
    {
        I2CPipe->Pend();    //wait for Message

        //-------------------------------------------------------
        // Check message
        //-------------------------------------------------------
        checksum=0;                 //initialize checksum
        destadr = I2CPipe->Get();   //get destination address
        checksum ^= destadr;
        souradr = I2CPipe->Get();   //get source address
        checksum ^= souradr;
        count = I2CPipe->Get();     //get message lenth
        checksum ^= (unsigned char)count;
        count -= 4;                 //there are four bytes of overhead

        for(i=0;(i<count)&&(i<20);++i)
        {
            c = I2CPipe->Get();  //get data and store temp
            checksum ^= c;      //create checksum
            Msg[i] = (int)c;    //store in message buffer
        }
        checksum ^= I2CPipe->Get();     //get checksum
        OMsg[1] = Msg[2];
        switch(Msg[0])              //decode command
        {
            case FPCOMMAND_BUTTON_STATE:
                switch(Msg[1])
                {
                    case FPCOMMAND_MANUAL_ENABLE:
                        if(PanelEnablePipe->QueueSpace() > 2)
                            PanelEnablePipe->PostMessage(OMsg,2);
                        break;
                    case FPCOMMAND_SPINDLE_DIR:
                        OMsg[2] = OMsg[1];
                        OMsg[1] = MOTORSPINDLE_DIRECTION_COMMAND;
                        if(SpindleMotorPipe->QueueSpace() > 3)
                            SpindleMotorPipe->PostMessage(OMsg,3);
                        break;
                    case FPCOMMAND_SPINDLE:
                        OMsg[2] = OMsg[1];
                        OMsg[1] = MOTORSPINDLE_STARTSTOP_COMMAND;
                        if(SpindleMotorPipe->QueueSpace() > 3)
                            SpindleMotorPipe->PostMessage(OMsg,3);
                        break;
                    case FPCOMMAND_HEADS:
                        if(HeadLoaderPipe->QueueSpace() > 2)
                            HeadLoaderPipe->PostMessage(OMsg,2);
                        break;
                    case FPCOMMAND_XSTAGE:
                        if(XStagePipe->QueueSpace() > 2)
                            XStagePipe->PostMessage(OMsg,2);
                        break;
                    case FPCOMMAND_ADJ_SELECT:
                        if(AdjustEnablePipe->QueueSpace() > 2)
                            AdjustEnablePipe->PostMessage(OMsg,2);
```

```
                        break;
                    case FPCOMMAND_CONTINUE:
                        if(ContinuePipe->QueueSpace() > 2)
                            ContinuePipe->PostMessage(OMsg,2);
                        break;
                    case FPCOMMAND_STOP:
                        if(StopPipe->QueueSpace() > 2)
                            StopPipe->PostMessage(OMsg,2);
                        break;
                    case FPCOMMAND_VACBUTTON:
                        if(VacuumButtonPipe->QueueSpace() > 2)
                            VacuumButtonPipe->PostMessage(OMsg,2);
                        break;
                    case FPCOMMAND_CONN_CLAMP:
                        if(ConnClampPipe->QueueSpace() > 2)
                            ConnClampPipe->PostMessage(OMsg,2);
                        break;
                }
                break;
            case FPCOMMAND_STAGE_STATE:
                switch(Msg[1])
                {
                    case FPCOMMAND_RAMP_OUT:
                        RampOutPipe->PostMessage(&Msg[2],1);                //semaphore for Ramp
 Out Sensor
                        break;
                    case FPCOMMAND_CLAMP_UP:
                        ClampUpPipe->PostMessage(&Msg[2],1);
                        break;
                    case FPCOMMAND_RAMP_IN:
                        RampInPipe->PostMessage(&Msg[2],1);             //semaphore for Ramp In
Sensor
                        break;
                    case FPCOMMAND_VAC:
                        VaccumPipe->PostMessage(&Msg[2],1);             //semaphore for Vacuum S
ensor
                        break;
                    case FPCOMMAND_AIR:
                        AirPresurePipe->PostMessage(&Msg[2],1);
                        break;
                    case FPCOMMAND_CORAL:
                        CombCoralPipe->PostMessage(&Msg[2],1);          //semaphore for the comb
 corral
                        break;
                    case FPCOMMAND_CLAMP_DOWN:
                        ClampDownPipe->PostMessage(&Msg[2],1);
                        break;
                    default:
                        break;
                }
                break;
            case FPCOMMAND_VERSION:
                SpinstandVersion->PostMessage(&Msg[1],2);
                break;
            case FPCOMMAND_SPINSTAND_STATE:                 //get back status data from spinstan
d
                SpinstandState->PostMessage(&Msg[1],1);
                break;
        }
    }   //end of while(1)
}

void I2CSendDemon(void)
{
    //-----------------------------------------------------------------
    //
```

```
    // This taks is used to send messages to the I2C port, and, hopefully
    // space them out in time so that the front panel is not overloaded
    //
    //---------------------------------------------------------------------
    unsigned char *s = new unsigned char[256];      //temp storage for message
    int i;
    int loop;
    int c;

    while(1)    //this is a task, loop forever
    {
        I2COutputMessageQueue->WaitMessage();   //wait for a message
        for(i=0,loop=1;loop;++i)
        {
            if((c = I2COutputMessageQueue->GetMessage()) < 0)
            {
                loop = 0;       // interrupt loop
                c = i;      //c will be count
            }
            else
            {
                s[i] = c;
            }
        }
        loop = 1;
        while(loop)             //attempt to send the message
        {
            I2CSendMessage(s,c);    //send the message
            if(WaitForI2CXmit->Pend() == 0)
                loop = 0;       //stop transmitting
            else
            {
                I2CSendPipe->Kill();    //kill current message befor resend
            }
            TDelay(3);              //delay for 20 milliseconds
        }   //end of while(loop)
    }   // end of while (1)
}

//end of extern C block
}
```

```
/*****************************************************************************
**
** I2C device driver
**
** This is a block device.  Lowest level driver will send a block of Data
**
** Or recieve a block of data.
**
*****************************************************************************/

#ifndef I2C__H
#define I2C__H

#include "queue.h"

//---------------------------------------------------------------------------
//
// Defines for Control/Status Register
//
//---------------------------------------------------------------------------

#define I2C_PENDING_INTERRUPT_NOT   0x80
#define I2C_ENABLE_SERIAL_OUTPUT    0x40
#define I2C_ES1                     0x20    //used for selecting registers
#define I2C_ES2                     0x10    //used for selecting registers
#define I2C_ENABLE_INTERRUPT        0x08
#define I2C_START                   0x04
#define I2C_STOP                    0x02
#define I2C_ACK                     0x01

//status bits if different

#define I2C_STOP_SENSED             0x20
#define I2C_BUS_ERROR               0x10
#define I2C_LRB_AD0                 0x08    //Last Recieved bit/addressed as 0
#define I2C_ADDRESSED_AS_SLAVE      0x04
#define I2C_LOST_ARBITRATION        0x02
#define I2C_BUS_BUSY                0x01


//---------------------------------------------------------------------------
//
// Defines for PCF8584 Registers
//
//---------------------------------------------------------------------------

#define I2C_BASE                    0xffec0001
#define I2C_CONTROLSTATUS           (*((char *)(I2C_BASE + 2)))
#define I2C_DATA                    (*((char *)(I2C_BASE + 0)))

//---------------------------------------------------------------------------
//
// Misc Defines for I2C port
//
//---------------------------------------------------------------------------

#define I2C_OWNADDRESS          0x6E    //effective own address of 0x20

//---------------------------------------------------------------------------
//
// High Level Commands for Son Of Accutrac Communications with Front
// Panel, recieved commands
//
//---------------------------------------------------------------------------

#define FPCOMMAND_BUTTON_STATE              0x10
```

```
#define FPCOMMAND_STAGE_STATE               0x20
#define FPCOMMAND_VERSION                   0x88
#define FPCOMMAND_SPINSTAND_STATE           0x89

//
// defines for button states
//

#define FPCOMMAND_MANUAL_ENABLE             0x01
#define FPCOMMAND_SPINDLE_DIR               0x02
#define FPCOMMAND_SPINDLE                   0x04
#define FPCOMMAND_HEADS                     0x08
#define FPCOMMAND_XSTAGE                    0x10
#define FPCOMMAND_ADJ_SELECT                0x20
#define FPCOMMAND_CONTINUE                  0x40
#define FPCOMMAND_STOP                      0x80
#define FPCOMMAND_VACBUTTON                 0x11
#define FPCOMMAND_CONN_CLAMP                0x12

//
// defines for stage states
//

#define FPCOMMAND_RAMP_OUT                  0x01
#define FPCOMMAND_CLAMP_UP                  0x02
#define FPCOMMAND_RAMP_IN                   0x04
#define FPCOMMAND_VAC                       0x08
#define FPCOMMAND_AIR                       0x10
#define FPCOMMAND_CORAL                     0x20
#define FPCOMMAND_CHUCKVAC                  0x40
#define FPCOMMAND_CLAMP_DOWN                0x80


//-----------------------------------------------------------------------
//
// High level commands to be sent to the Front panel
//
//-----------------------------------------------------------------------

#define FPCOMMAND_RESET                     0x80
#define FPCOMMAND_DISPLAY_DATA              0x81
#define FPCOMMAND_DISPLAY_ATTRIBUTES        0x82
#define FPCOMMAND_SET_CONTROL_STATE         0x83
#define FPCOMMAND_ECHO                      0x84
#define FPCOMMAND_BUZZER_STATE              0x85
#define FPCOMMAND_SELECT_HEAD               0x86
#define FPCOMMAND_VERSION_SEND              0x88
#define FPCOMMAND_SPINSTAND_STATE_SEND      0x89

//
// defines for control state
//

#define FPCOMMAND_LOADED_LED                0x01
#define FPCOMMAND_UNLOADED_LED              0x02
#define FPCOMMAND_RUN_SPIN_LED              0x03
#define FPCOMMAND_STOP_SPIN_LED             0x04
#define FPCOMMAND_CW_LED                    0x05
#define FPCOMMAND_CCW_LED                   0x06
#define FPCOMMAND_RUN_X_LED                 0x07
#define FPCOMMAND_HOME_X_LED                0x08
#define FPCOMMAND_RPM_LED                   0x09
#define FPCOMMAND_RADIUS_LED                0x0a

#define FPCOMMAND_VACCARRIER                0x0b
#define FPCOMMAND_VACUUM                    0x0c
```

```
#define FPCOMMAND_RAMP                      0x0d
#define FPCOMMAND_CONCLAMP                  0x0e
#define FPCOMMAND_STAGEDRV                  0x0f

#define FPCOMMAND_CONCLAMP_UP_LED           0x11
#define FPCOMMAND_CONCLAMP_DOWN_LED         0x12
#define FPCOMMAND_VACCUUM_ON_LED            0x13
#define FPCOMMAND_VACCUUM_OFF_LED           0x14

#define FPCOMMAND_STATE_ON                  1
#define FPCOMMAND_STATE_OFF                 0
#define FPCOMMAND_LED_ON                    1
#define FPCOMMAND_LED_OFF                   0

//---------------------------------------------------------------------------
// External Peripheral Addresses
//---------------------------------------------------------------------------

#define I2C_FRONTPANEL                  0xaa

//---------------------------------------------------------------------------
// Definitions for Messages sent by I2C dispatcher to function tasks
//
// This is to differentiate between Internal messages and Messages from
// the front panel.  Front panel messages are only valid when the panel
// enable button is down.
//---------------------------------------------------------------------------

#define I2C_MESSAGE_FP                  0       //message originated from FP
#define I2C_MESSAGE_INTERNAL            1       //message originated internally

//---------------------------------------------------------------------------
// Definitions for Messages sent to deal with the spindle motor
//
// Commands include : starting/stoping
//                  : direction changing
//                  : setting rpm
//
//---------------------------------------------------------------------------

#define MOTORSPINDLE_STARTSTOP_COMMAND      0
#define MOTORSPINDLE_DIRECTION_COMMAND      1
#define MOTORSPINDLE_SETRPM_COMMAND         2
#define MOTORSPINDLE_EMSTOP_COMMAND         3

//---------------------------------------------------------------------------
// Function Prototypes
//---------------------------------------------------------------------------

#ifdef __cplusplus
extern "C" {
#endif

extern void I2CInit(void);
extern void I2CSendMessage(unsigned char *d,int size);
extern void I2CWriteLEDString(char *s);
extern int I2CBeepTheBuzzer(int c,int t);
extern void I2CSetControlStage(int control, int state);
extern int StageSetXStage(int c);
extern int GetXStagePosition(void);
extern int StageSetDirection(int c);
extern int StageSetMotor(int c);
extern int StageEmergencyMotorStop(void);
extern int StageSetRpm(int rpm);
extern int StageLoadHeads(int c);
extern int StageSuckHeads(int c);
```

```
extern int StageSelectHead(int c);
extern int StageConnectorClamp(int c);
extern int I2CGetStageStatus(void);
extern unsigned I2CGetStageVersion(void);
extern void BigTimeShutDown(int why);


extern Wait *ApplyBrakes;
extern TSemaphore *BrakesDone;
extern EventQueue *RampInPipe;      //semaphore for Ramp In Sensor
extern EventQueue *RampOutPipe;     //semaphore for Ramp Out Sensor
extern EventQueue *ClampUpPipe;     //semaphore for Connector Clamp Sensor
extern EventQueue *ClampDownPipe;   //semaphore for Connector Clamp Sensor

extern TSemaphore *RampInFlag;          //Semaphore for Comb In Sensor
extern TSemaphore *RampOutFlag;         //Semaphore for Comb out Sensor
extern TSemaphore *ClampUpFlag;         //Semaphore for Clamp Up sensor
extern TSemaphore *ClampDownFlag;       //Semaphore for Clamp Down sensor

//----------------------------------------
// PARAMETERS FOR StageSetXStage
//----------------------------------------

#define STAGESETXSTAGE_HOME     0
#define STAGESETXSTAGE_RUN      1

//----------------------------------------
// PARAMETERS for StageLoadHeads
//----------------------------------------

#define STAGELOADHEAD_RAMPIN    0
#define STAGELOADHEAD_RAMPOUT   1


//----------------------------------------
// paramters for StageConnectorClamp
//----------------------------------------

#define STAGECONCLAMP_UP        0
#define STAGECONCLAMP_DOWN      1

#ifdef __cplusplus
}
#endif

/*
** functions associated with events that are generated by I2C port
*/

extern int WaitForContinueButtonPushed(void);
extern int WaitForRampIn(int timeout);
extern int WaitForRampOut(int timeout);
extern void ClearRampFlags(void);
extern int WaitForChuckSuck(int suckblow);
extern int WaitForCombInOut(int combinout);
extern void ClearChuckSuck(void);
extern int WaitForConnectorClampUp(int timeout);
extern int WaitForConnectorClampDown(int timeout);
extern void ClearConnectorClamps(void);

//----------------------------------------
// parameter for WaitForChuckSuck
//----------------------------------------

#define WAITFORCHUCKSUCK_SUCK   0
#define WAITFORCHUCKSUCK_BLOW   1
```

```
//----------------------------------------
// paramter for WaitForCombInOut
//----------------------------------------

#define WAITFORCOMBINOUT_IN     0
#define WAITFORCOMBINOUT_OUT    1

//*************************************************
// defines for RampIn Task
// these are commands sent to the RampIn/out task via
// the RampIn(out)Pipe message pipe
//*************************************************

#define RAMPCOMMAND_HSAMODE 16  //these are commands that you
#define RAMPCOMMAND_HGAMODE 17  //send via RampInPipe to set mode

#define RAMPMODE_HSA        0   //these are values that the
#define RAMPMODE_HGA        1   //local mode varible can take on



#endif
```

```
;*****************************************************************************
;
; Assembly Code to handle I2C interrupts
; Calls various kernel functions before calling interrupt handler
;
;*****************************************************************************

    XDEF I2CIrq
    XREF _I2CHandleInterrupt
    XREF _EnterInterrupt
    XREF _TimerTicker
    .FREF _ExitInterrupt,2

    SECTION code

I2CIrq:
    movem.l d0-d7/a0-a6,-(a7)          ;save processor context
    jsr     _EnterInterrupt            ;increment interrupt count
    jsr     _I2CHandleInterrupt        ;handle the interrupt
    move.w  60(a7),d0                  ;get status register
    andi.w  #0x0700,d0                 ;mask off junk
    asr.w   #8,d0                      ;shift over 8 bits
    move.w  d0,-(a7)                   ;push onto stack
    jsr     _ExitInterrupt             ;call routine to exit interrupt
    movem.l (a7)+,d0-d7/a0-a6          ;restore processor context
    rte                                ;return to interrupted code
```

```
/*-------------------------------------------------------------
**
** Copyright (c) 1995 Teletrac Inc.
**
** This file contains the code for the Teletrac Serial Protocol
**
**-------------------------------------------------------------*/

#include <stdio.h>
#include "cio.h"
#include "task.h"
#include "queue.h"
#include <string.h>
#include "servo.h"
#include "rs232.h"
#include "timer.h"
#include "serlprot.h"
#include "spindle.h"
#include "spincopr.h"
#include "hostcmd.h"    /*  list of protocol commands   */
#include "global.h"
#include "endian.h"
#include "ramdisk.h"
#include "acutrac.h"
#include "frontled.h"
#include "scfig.h"
#include "i2c.h"
#include "strings.h"

static long prevGoal;

/*************************************************************************
** the following data structure will contain information about various
** features so that the host software will know how to deal with things to
** keep things software compatable
*************************************************************************/

static FEATURES *features;


/*************************************************************************
** back to our regular programming
*************************************************************************/

extern volatile int GlobalSpindleRPM;
extern volatile int DisplayMode;
static char *ProcessSpindleCommands(int count,char *cmd,int *index,char *obuf,int *ocnt,SYSTEM *
sys);
static char *ProcessServoCommands(int count,char *cmd,int *index,char *obuf,int *ocnt);
static char *ProcessHighLevelCommands(int count,char *cmd,int *index,char *obuf,int *ocnt);
static char *ProcessRamDiskCommands(int count,char *cmd,int *index,char *obuf,int *ocnt);

static int SerialSetGetRunoutComp(int axis,char *name);
static int SerialSetSetRunoutComp(int axis,char *name);

static char XmitBuffer[MAX_DATA];

#pragma region("ram=logdata")

#define LOGDEPTH    32      //depth of command log

long SequenceNumber;
int LogIndex;
LOG Logdata[LOGDEPTH];

/*************************************************************************
```

```
** Logging Functions
*****************************************************************************/

void InitLog(void)
{
    int i;

    for(i=0;i<LOGDEPTH;++i)
    {
        Logdata[i].InCommand = 0;
        Logdata[i].InPacketId = 0;
        Logdata[i].InByteCount = 0;
        Logdata[i].InTimeStamp = 0;
        Logdata[i].ErrorNumber = 0;
        Logdata[i].OutCommand = 0;
        Logdata[i].OutPacketId = 0;
        Logdata[i].OutByteCount = 0;
        Logdata[i].OutTimeStamp = 0;
        Logdata[i].SequenceNumber = 0;
    }
    LogIndex = 0;
    SequenceNumber = 0;
}

/*****************************************************************************
** Serial Functions
*****************************************************************************/

extern "C" {
char *TelBuildStatusBuffer(int status,int command,char *b)
{
    *b++ = 's';     /*  status command  */
    *b++ = (char)command;
    Logdata[LogIndex].OutCommand = command;
    *b++ = (char) -status;
    return b;
}
}

/*****************************************************************************
**
** member functions for FEATURES data structure
**
*****************************************************************************/

FEATURES::FEATURES()        //constructor
{
    numofnotchfilters = 2;  //current number is two v1.62
    ConnectorClamp = 1;     //current number is 1 (YES) v1.69.41
}

//FEATURES::~FEATURES();    //destructor
char * FEATURES::Save(char *p)  //save the data
{
    p = SaveInt(p,numofnotchfilters);
    return SaveInt(p,ConnectorClamp);
}

void InitProtocol(void)
{
    features = new FEATURES;
    prevGoal = 0;
}

/*****************************************************************************
**
```

```
**                    Teletrac Serial Protocol
**
** These routines form the basic serial protocol for the teletrac
** spindle controller board.  Data is transmitted in Packets.  What
** is inside the packet is totally imaterial to what the protocol manager
** does.  The protocol manager does not care what is in the data
**
** A Packet consists of the following elements:
**
**  Byte 0:0x80         ;Packet Header Indicator
**  Byte 1:0x01         ;Packet Version Number
**  Byte 2:<count>      ;Number of bytes of Data
**  Byte 3:<dev Id>     ;Identity of device data is for
**  Byte 4-> Byte n-1   ;<count> number of bytes of binary data
**  Byte n:<checksum>   ;sum of all bytes in data packet
**  Byte n:0x7f         ;packet tail indicator
**
*************************************************************************/

extern "C" {
void TelSendProtocol(int handle,int devID,int count,char *buff,int logflag)
{
    int i,n;
    unsigned int checksum=0;        /*  need to accumulate checksum */
    /*---------------------------------------
    ** handle:      handle of IO device to send data
    ** count:       number of bytes in data packet
    ** buff:        pointer to data buffer to send
    **
    ** package the data together into a packet
    ** and then send it out the device pointed
    ** to by handle
    **---------------------------------------*/
    XmitBuffer[0] = 0x80;   /*  packet header indicator */
    XmitBuffer[1] = '1';    /*  version number          */

    XmitBuffer[2] = (char)count;
    XmitBuffer[3] = (char)devID;
    for(i=0;i<count;++i)
        XmitBuffer[4+i] = buff[i];  /*  copy data into output buffer    */
    n = i+4;
    for(i=0;i<n;++i)
        checksum += (unsigned)XmitBuffer[i];    /*  calculate checksum  */
    XmitBuffer[n++] = checksum & 0x0ff;
    XmitBuffer[n++] = 0x7f; //packet tail indicator
    if(logflag) Logdata[LogIndex].OutTimeStamp = time_ms;   //log time stamp on output
    Write(handle,XmitBuffer,(long)n);
    if(logflag)
    {
        Logdata[LogIndex].SequenceNumber = SequenceNumber++;    //log misc stuff
        Logdata[LogIndex].OutByteCount = count;
        Logdata[LogIndex].OutPacketId = devID;
        LogIndex++;
        if(LogIndex == LOGDEPTH) LogIndex = 0;  //check for log index wrap
    }
}
}

extern "C" {
int TelReceiveProtocol(int handle,PROT_DATA *d)
{
    /*---------------------------------------
    **
    ** handle:      handle of IO device to recieve protocol
    ** d:       place to put data
    **
```

```
    **
    ** Recieve Data, put it into the buffer
    ** Check to makesure that the checksum was correct
    **
    **--------------------------------------*/
//  int loop = 1;
    int c;
    int return_status = TEL_PROT_OK;
    int i;
    int checksum,csum=0;
    int eop;    //packet tail indicator

    //-------------------------------------------------
    // note for version 1.62.06 and above
    // since we do not know if we wish to actually log this
    // transaction, we have to start doing it anyway
    // this will mean that we will always loose the very
    // oldest entry, but, I can think of no other way
    //-------------------------------------------------

    Xio(RS232_SET_RTIMEOUT,handle,(char *)0,(char *)0,0l,0);    //infinate timeout
    c = Getc(handle);   /*  get data    */
    if(c == 0x80)       /*  header start?   */
    {
        /*  ok, we have a start of header, fill in all the parts    */
        /*  version */

        //check LogIndex to make sure it is withing range
        //It can go out of range on a dead battery
        if(LogIndex < 0 || LogIndex > LOGDEPTH) LogIndex = 0;
        Logdata[LogIndex].InTimeStamp = time_ms;    //log command in timestamp
        csum += c;      /*  accumulate checksum */
        Xio(RS232_SET_RTIMEOUT,handle,(char *)0,0l,20);
        if((c = Getc(handle)) < 0) return TEL_PROT_TIMEOUT;
        d->version = c;
        csum += c;      /*  accumulate checksum */
        /*  byte count  */
        if((c = Getc(handle)) < 0) return TEL_PROT_TIMEOUT;
        d->bytecount = c;
        Logdata[LogIndex].InByteCount = c;  //log byte count
        csum += c;      /*  accumulate checksum */
        /*  Device ID   */
        if((c = Getc(handle)) < 0) return TEL_PROT_TIMEOUT;
        d->devID = c;
        Logdata[LogIndex].InPacketId = c;   //log packet ID
        csum += c;
        /*  get data    */
        for(i=0;i<d->bytecount;++i)
        {
            if((c = Getc(handle)) < 0) return TEL_PROT_TIMEOUT;
            d->buff[i] = (char)c;
            csum += c;  /*  accumulate checksum */
        }
        if((checksum = Getc(handle)) < 0) return TEL_PROT_TIMEOUT;
        /*  end of packet tail  */
        if((eop = Getc(handle)) < 0) return TEL_PROT_TIMEOUT;
        if((csum & 0x0ff) - checksum)
            return_status = TEL_PROT_CHECKSUM;
//      loop = 0;   /*  bump out of loop, we got the poop   */
    }
    else
        return_status = TEL_PROT_BADHEADER;
    return return_status;
}
}
```

```
/*-------------------------------------------------------------------
**
** Command Parser
**
** Receives a character pointer that points to a buffer that contains
** the command(s) string
**
** Commands are executed in the order found
**
**-------------------------------------------------------------------*/

extern "C" {
int TelCommandParser(int count,char *cmd,char *obuf,SYSTEM *sys,int *logflag)
{
    /*
    ** count:       number of characters in commnad string
    ** cmd:         pointer to command string
    ** obuf:        pointer to string to build return in
    ** returns number of characters in obuf
    */
    int command;
    int loop = 1;
    int i=0;
    char *p,*s;
    int ocnt = 0;
    int baud;
    int error_flag;
    int Y,X;

    p = obuf;
    do
    {
        command = cmd[i++];     /*  first character better be command   */
        Logdata[LogIndex].InCommand = command;
        switch(command)         /*  decode command  */
        {
            case 'D':   //process Ram disk commands
                if(ocnt)    /*  add concat character if needed  */
                {
                    *p++ = '&';
                    ++ocnt;
                }
                p = ProcessRamDiskCommands(count,cmd,&i,p,&ocnt);
                *logflag = 1;
                break;
            case 'E':   //return back size of feature set structure
                if(ocnt)    //add concat character if needed
                {
                    *p++ = '&';
                    ++ocnt;
                }
                p = TelBuildStatusBuffer(GENERAL_OK,'E',p);
                ocnt += 3;
                p = SaveInt(p,sizeof(FEATURES));
                ocnt += sizeof(int);
                *logflag = 1;
                break;
            case 'F':   //get feature set
                if(ocnt)    //add concat character if needed
                {
                    *p++ = '&';
                    ++ocnt;
                }
                p = TelBuildStatusBuffer(GENERAL_OK,'F',p);
                ocnt += 3;
                p = features->Save(p);
```

```
        ocnt += sizeof(FEATURES);
        *logflag = 1;
        break;
    case 'G':           /*  GET_REV(A,B,C,D)    */
        /*
           returns back Board Info
           A:  Board Name
           B:  Board Revision
           C:  Firmware Revision
           D:  Firmware Date
        */
        if (ocnt)
        {
            *p++ = '&';     /*  add concat character    */
            ++ocnt;
        }
        p = TelBuildStatusBuffer(GENERAL_OK,'G',p);
        ocnt += 3;
        strcpy(p,tel_board_name);
        command = strlen(tel_board_name) + 1;
        p += command;   /*  increment pointer   */
        ocnt += command;    /*  increment bytecount */

        command =  sprintf(p,tel_board_rev,'A' + *((char *)0xffd00001) );

        command++;
        p += command;   /*  increment pointer   */
        ocnt += command;    /*  increment bytecount */

        strcpy(p,tel_version);
        command = strlen(tel_version) + 1;
        p += command;   /*  increment pointer   */
        ocnt += command;    /*  increment bytecount */
        strcpy(p,tel_date);
        command = strlen(tel_date) + 1;
        p += command;   /*  increment pointer   */
        ocnt += command;    /*  increment bytecount */
        *logflag = 1;
        break;
    case 'H':   /*  process high level commands     */
        if(ocnt)    /*  add concat character if needed  */
        {
            *p++ = '&';
            ++ocnt;
        }
        p = ProcessHighLevelCommands(count,cmd,&i,p,&ocnt);
        *logflag = 1;
        break;
    case 'I':   /*  init log data   */
        if(ocnt)
        {
            *p++ = '&';
            ++ocnt;
        }
        p = TelBuildStatusBuffer(GENERAL_OK,'I',p);
        *logflag = 0;   //do not log this command
        break;
    case 'L':   // send back log data
        if(ocnt)
        {
            *p++ = '&';
            ++ocnt;
        }
        X = GetInt(&cmd[i]);    //get log entry 'n'
        i += sizeof(int);
        if(X > LOGDEPTH)
```

```
        {
            p = TelBuildStatusBuffer(GENERAL_NAK,'L',p);
            ocnt += 3;
        }
        else
        {
            p = TelBuildStatusBuffer(GENERAL_OK,'L',p);
            ocnt += 3;
            p = SaveInt(p,Logdata[X].InCommand);    //data to send back
            p = SaveInt(p,Logdata[X].InSubCommand);
            p = SaveInt(p,Logdata[X].InPacketId);
            p = SaveInt(p,Logdata[X].InByteCount);
            p = SaveLong(p,Logdata[X].InTimeStamp);
            p = SaveInt(p,Logdata[X].ErrorNumber);
            p = SaveInt(p,Logdata[X].OutCommand);
            p = SaveInt(p,Logdata[X].OutPacketId);
            p = SaveInt(p,Logdata[X].OutByteCount);
            p = SaveLong(p,Logdata[X].OutTimeStamp);
            p = SaveLong(p,Logdata[X].SequenceNumber);
            ocnt += sizeof(LOG);
        }
        *logflag = 0;   //do not log this command
        break;
    case 'P':   /*  servo positioner commands   */
        if(ocnt)    /*  add concat character if needed  */
        {
            *p++ = '&';
            ++ocnt;
        }
        p = ProcessServoCommands(count,cmd,&i,p,&ocnt);
        *logflag = 1;
        break;
    case 'R':   /*  Set system serial number    */
        s = &cmd[i];
        i += strlen(s) + 1;
        SetSerialNumber(s);
        p = TelBuildStatusBuffer(GENERAL_OK,'R',p);
        ocnt += 3;
        *logflag = 1;
        break;
    case 'S':   /*  spindle controller commands */
        if(ocnt)    /*  add concat character if needed  */
        {
            *p++ = '&';
            ++ocnt;
        }
        p = ProcessSpindleCommands(count,cmd,&i,p,&ocnt,sys);
        *logflag = 1;
        break;
    case 'T':   /*  Get the System Serial number    */
        if (ocnt)
        {
            *p++ = '&';     /*  add concat character    */
            ++ocnt;
        }
        if(CheckSerialNumber())
            p = TelBuildStatusBuffer(GENERAL_ERROR,'T',p);  /*  bad check sum   */
        else
            p = TelBuildStatusBuffer(GENERAL_OK,'T',p);
        ocnt += 3;
        p = GetSerialNumber(&ocnt,p);
        *logflag = 1;
        break;
    case '&':           /*  command concatanate */
        break;
    default :           /*  illegal command */
```

```
                if(ocnt)
                {
                    *p++ = '&';
                    ++ocnt;
                }
                p = TelBuildStatusBuffer(GENERAL_NAK,command,p);
                ocnt += 3;
                *logflag = 1;
                break;
            }   /*  end of switch command   */
            if(i >= count)  /*  at the end of the buffer?   */
                loop = 0;
        }while(loop);
        return ocnt;    /*  update output buffer count  */
}

}       //end of extern "C"

static char *BuildSpindleStatus(int errorflag,int command,char *obuf,int *ocnt)
{
    char *o;

    o = TelBuildStatusBuffer(errorflag,'S',obuf);
    *o++ = (char)command;
    *ocnt += 4;     /*  increment output count  */
    return o;
}

static char *ProcessSpindleCommands(int count,char *cmd,int *index,char *obuf,int *ocnt,SYSTEM *
sys)
{
    /*****************************************************************
    ** This function processes spindle commands
    **
    ** count:number of bytes total in command input buffer
    ** cmd  :command input buffer
    ** index:pointer to index in input buffer
    ** obuf :response output buffer
    ** ocnt :pointer to byte count in output buffer
    ** sys  :pointer to system device handles
    **
    **  Returns back pointer to first free character space in obuf
    **
    ** Command format as seen in the buffer is:
    **
    **  Byte 0:Spindle Command
    **  Byte 1->n:Aux data if any
    **
    *****************************************************************/
    int Command;
    int i,*ip;
    int error_flag = 0;
    char *o;
    int outcount=0;
    int v,v1,v2;
    long lv;
    char *s;
    String *S;

    o = obuf;
    i = *index;
    Command = (int)cmd[i++];
    Logdata[LogIndex].InSubCommand = Command;

    switch(Command) /*  decode command  */
    {
```

```
        case SCMD_MOTOR:                /*  Motor On/Off Command    */
            v = cmd[i++];       /*  get motor command flag  */
            /*  create reply status */
            error_flag = StageSetMotor(v);
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_CLAMP:                /*  Clamp/Unclamp command   */
            //-----------------------------------------------------
            // This is going to be used for the XStage for now
            //-----------------------------------------------------
            v = cmd[i++];       /*  get clamp control flag  */
            if((error_flag = StageSetXStage(v)) == 0)
                SystemStatus.spin.stat.clamp = v;
            /*  create reply status */
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_DIRECTION:            /*  Clockwise/CounterClockwise rotation */
            v = cmd[i++];
            /*  create reply status */
            StageSetDirection(v);
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_SETSPEED:         /*  Set rotational Rate     */
            v = GetInt(&cmd[i]);
            i += sizeof(int);
            StageSetRpm(v);
            /*  create reply status */
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_SETACCEL:         /*  Set acceleration rate   */
            v = GetInt(&cmd[i]);
            i += sizeof(int);
            Xio(SPIN_SETACCEL,sys->HSpindle,(char *)0,(char *)0,0l,v);
            /*  create reply status */
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_SETDECEL:         /*  Set slowdown rate       */
            v = GetInt(&cmd[i]);
            i += sizeof(int);
            Xio(SPIN_SETDECEL,sys->HSpindle,(char *)0,(char *)0,0l,v);
            /*  create reply status */
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_STATUS:           /*  Get Spindle Status      */
            /*  create reply status */
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            o = SaveInt(o,SystemStatus.spin.v);
            o = SaveInt(o,SystemStatus.servo1.v);
            o = SaveInt(o,SystemStatus.servo2.v);
            o = SaveInt(o,SystemStatus.servo3.v);
            o = SaveInt(o,SystemStatus.LedStates);
            outcount += sizeof(unsigned) * 5;
            break;
        case SCMD_GETACTSPEED:      /*  Get Actual Speed        */
            v = GlobalSpindleRPM;
            /*  create reply status */
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            o = SaveInt(o,v);
            outcount += sizeof(unsigned);
            break;
        case SCMD_GETSETSPEED:      /*  Get the Set Speed       */
            /*  create reply status */
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            v = Getc(sys->HSpindle);
            o = SaveInt(o,v);
            outcount += sizeof(int);
```

```
        break;
    case SCMD_GETACCEL:         /*  get acceleration       */
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        v = (int)Xio(SPIN_GETACCEL,sys->HSpindle,(char *)0,(char *)0,0l,0);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_GETDECEL:         /*  Get Deceleration       */
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        v = (int)Xio(SPIN_GETDECEL,sys->HSpindle,(char *)0,(char *)0,0l,0);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_SETENCODER:       /*  set encoder pitch   */
        v = GetInt(&cmd[i]);
        i += sizeof(int);
        Xio(SPIN_SETENCODER,sys->HSpindle,(char *)0,(char *)0,0l,v);
        if(SetEncoderPitch(2,(long)v)< 0)       //tell dsp what pitch is too
            error_flag = SPINDLE_ERROR;
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETENCODER:       /*  get encoder pitch   */
        v = (int)Xio(SPIN_GETENCODER,sys->HSpindle,(char *)0,(char *)0,0l,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_SETPOLES:         /*  set number of motor poles   */
        v = GetInt(&cmd[i]);
        i += sizeof(int);
        Xio(SPIN_POLES,sys->HSpindle,(char *)0,(char *)0,0l,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETPOLES:         /*  get number of motor poles   */
        v = (int)Xio(SPIN_GETPOLES,sys->HSpindle,(char *)0,(char *)0,0l,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_SETPHASING:       /*  set motor drive phasing     */
        v = GetInt(&cmd[i]);
        i += sizeof(int);
        Xio(SPIN_PHASING,sys->HSpindle,(char *)0,(char *)0,0l,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETPHASING:       /*  get motor drive phasing     */
        v = (int)Xio(SPIN_GETPHASING,sys->HSpindle,(char *)0,(char *)0,0l,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_HEADLOAD:
        v = cmd[i++];       /*  get headload command flag   */
        error_flag = StageLoadHeads(v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETLOWERLIMIT:    /*  get pll filter lower limit  */
```

```
        v = (int)Xio(SPIN_GETDAC,sys->HSpindle,(char *)0,(char *)0,0l,SPIN_LOWERLIMITS);
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_SETLOWERLIMIT:    /*  set pll filter lower limit  */
        v = cmd[i++];           /*  get lower limit */
        Xio(SPIN_SETDAC,sys->HSpindle,(char *)0,(char *)0,0l,SPIN_LOWERLIMITS,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETUPPERLIMIT:    /*  get pll filter upper limit  */
        v = (int)Xio(SPIN_GETDAC,sys->HSpindle,(char *)0,(char *)0,0l,SPIN_UPPERLIMITS);
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_SETUPPERLIMIT:    /*  set pll filter upper limit  */
        v = cmd[i++];           /*  get lower limit */
        Xio(SPIN_SETDAC,sys->HSpindle,(char *)0,(char *)0,0l,SPIN_UPPERLIMITS,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETDISPLAYMODE:
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,DisplayMode);
        outcount += sizeof(int);
        break;
    case SCMD_SETDISPLAYMODE:
        DisplayMode = cmd[i++];
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_VACCHUCK:
        v = cmd[i++];       /*  get headload command flag   */
        error_flag = StageSuckHeads(v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETDACDIRECT:
        v = (int)Xio(SPIN_GETDAC,sys->HSpindle,(char *)0,(char *)0,0l,SPIN_DIRECTCONTROL);
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_SETDACDIRECT:
        v = cmd[i++];
        Xio(SPIN_SETDAC,sys->HSpindle,(char *)0,(char *)0,0l,SPIN_DIRECTCONTROL,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETLOOPGAIN:
        v = (int)Xio(SPIN_GETDAC,sys->HSpindle,(char *)0,(char *)0,0l,SPIN_LOOPGAIN);
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_SETLOOPGAIN:
        v = cmd[i++];
        Xio(SPIN_SETDAC,sys->HSpindle,(char *)0,(char *)0,0l,SPIN_LOOPGAIN,v);
        /*  create reply status */
        o = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_SETLOOPCOMP:  /*  set loop compensation params    */
        v = GetInt(&cmd[i]);    //compensation type
        i+=sizeof(int);
        v1 = GetInt(&cmd[i]);   //index value
```

```
        i+=sizeof(int);
        v2 = GetInt(&cmd[i]);   //compensation value
        i += sizeof(int);
        Xio(SPIN_SETLOOPCOMP,sys->HSpindle,(char *)0,(char *)0,0l,v,v1,v2);
        /*  create reply status */
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETLOOPCOMP:  /*  get the loop compensation params    */
        v = GetInt(&cmd[i]);    //compensation type
        i+=sizeof(int);
        v1 = GetInt(&cmd[i]);   //index value
        i+=sizeof(int);
        v2 = (int)Xio(SPIN_GETLOOPCOMP,sys->HSpindle,(char *)0,(char *)0,0l,v,v1);
        /*  create reply status */
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v2);
        outcount += sizeof(int);
        break;
    case SCMD_SAVEMOTORPARAMS:  /*  save the motor parameters to file   */
        S = new String;
        s = S->Get();
        memcpy(s,&cmd[i],8);
        s[8] = 0;
        i+= 8;
        SaveSpindleConfig(s);
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        delete S;
        break;
    case SCMD_LOADMOTORPARAMS:  /*  load the motor parameters from file */
        S = new String;
        s = S->Get();
        memcpy(s,&cmd[i],8);
        s[8] = 0;
        i+= 8;
        LoadSpindleConfig(s);
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        delete S;
        break;
    case SCMD_INITLUT:  /*  rebuild the motor drive look up table   */
        DoMotorInit();  /*  start up motor initialization task  */
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETINITSTATUS:    /*  find out how much further to go for motor init  */
        lv = GetMotorInitStatus();
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case SCMD_SETSPINDLEMODE:   /*  set source of motor drive for spindle motor */
        v = GetInt(&cmd[i]);    //pll mode
        i+=sizeof(int);
        Xio(SPIN_SETPLLMODE,sys->HSpindle,(char *)0,NULL,0l,v); //set Pll Drive Mode
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
    case SCMD_GETSPINDLEMODE:   /*  get source of motor drive for spindle motor */
        v = (int)Xio(SPIN_GETPLLMODE,sys->HSpindle,(char *)0,NULL,0l,0);    //get Pll Mode
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case SCMD_MOTORONOFFLL:
        v = GetInt(&cmd[i]);
        i+=sizeof(int);
        Xio(SPIN_MOTOR,sys->HSpindle,(char *)0,NULL,0l,v);
        o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
        break;
```

```
        case SCMD_SETMOTDACMODE:    /*  set access to motor dac */
            v = GetInt(&cmd[i]);
            i += sizeof(int);
            Xio(SPIN_SETMOTDACMODE,sys->HSpindle,(char *)0,NULL,0l,v);
            o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_SETMOTORDAC:      /*  set motor dac values    */
            v = GetInt(&cmd[i]);
            i += sizeof(int);
            v1 = GetInt(&cmd[i]);
            i += sizeof(int);
            Xio(SPIN_SETMOTDAC,sys->HSpindle,(char *)0,NULL,0l,v,v1);
            o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_GETAMPENABLE:
            v = (int)Xio(SPIN_GETAMPENABLE,sys->HSpindle,(char *)0,NULL,0l,0);
            o  = BuildSpindleStatus(error_flag,Command,o,&outcount);
            o = SaveInt(o,v);
            outcount += sizeof(int);
            break;
        case SCMD_CONNECTORCLAMP:
            v = cmd[i++];       /*  get motor command flag  */
            /*  create reply status */
            error_flag = StageConnectorClamp(v);
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_SETSTART: /*  set start pulse position    */
            v = GetInt(&cmd[i]);        /*  get pulse position  */
            Xio(SPIN_SETSTART,sys->HSpindle,(char *)0,NULL,0l,v);
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_GETSTART: /*  get start pulse position    */
            v = (int)Xio(SPIN_GETSTART,sys->HSpindle,(char *)0,NULL,0l,0);
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            o = SaveInt(o,v);
            outcount += sizeof(int);
            break;
        case SCMD_RESETSTOPSTAT:
            SystemStatus.spin.stat.stopbutton = 0;
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            break;
        case SCMD_UPLOADSLUT:   //upload sine lut to spindle controller */
            //the sector number is the first int on the stack
            v = GetInt(&cmd[i]);
            i+= sizeof(int);
            ip = new int[64];   //retrieve the 64 words of data
            memcpy(ip,&cmd[i],128); //copy data
            i+= sizeof(int) * 64;
            Xio(SPIN_WRITESLUT,sys->HSpindle,(char *)0,(char *)ip,0l,v);
            o = BuildSpindleStatus(error_flag,Command,o,&outcount);
            delete[] ip;
            break;
        default:
            o = BuildSpindleStatus(SPINDLE_NAK,Command,o,&outcount);
            break;
    }   /*  end of decode command   */
    /* send back the information calling routine needs  */
    Logdata[LogIndex].ErrorNumber = error_flag;
    *ocnt += outcount;
    *index += i;
    return o;
}

char *BuildServoStatus(int errorflag, int Card,int Command,char *obuf,int *ocnt)
{
    char *o;
```

```
    o = TelBuildStatusBuffer(errorflag,'P',obuf);
    *o++ = (char)(Card + '1');
    *o++ = (char)Command;
    *ocnt += 5;     /*  increment output count  */
    return o;
}

static char *ProcessServoCommands(int count,char *cmd,int *index,char *obuf,int *ocnt)
{
    /*****************************************************************
    ** this function processes servo commands
    **
    **  count:number of bytes total in command input buffer
    **  cmd  :command input buffer
    **  index:pointer to index in input buffer
    **  obuf :response output buffer
    **  ocnt :pointer to byte count in output buffer
    **
    **  Returns back pointer to first free character space in obuf
    **
    ** Command format as seen in the buffer is:
    **
    **  BYTE 0:Controller Number in ASCII (0,1,2,3,....)
    **  BYTE 1:Command Number in binary
    **  BYTE 2->n:Aux Data, if any, depends on the command
    **
    *****************************************************************/
    int ControllerNumber;
    int Command;
    int i,j,iv;
    char *o;
    int errorflag = 0;
    int outcount = 0;
    long lv,lv1,*lp;
    static long ld[32];
    double dv;
    HOMEPARAMS Hp;
    JOGPARAMS Jp;
    DSP_status stat;

    o = obuf;
    i = *index;
    ControllerNumber = (int) cmd[i++] - '0';
    Command = (int) cmd[i++];
    Logdata[LogIndex].InSubCommand = Command;

    switch(Command)     /*  decode command  */
    {
        case SET_KP:                    /*  set proportional gain   */
            lv = GetLong(&cmd[i]);
            i+= sizeof(long);
            errorflag = Set_Kp(ControllerNumber,lv);
                    /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SET_KV:                /*  set differential gain   */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = Set_Kv(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SET_KI:                /*  set integral gain       */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
```

```
        errorflag = Set_Ki(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_KVFF:              /*  set feed forward velocity gain  */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = Set_Kvff(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_KAFF:              /*  set feed forward acceleration gain  */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = Set_Kaff(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_OFFSET:            /*  set servo offset    */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = Set_Offset(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_GOAL:              /*  set desired position    */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = Set_Goal(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_MAXVEL:            /*  set maximum velocity    */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = Set_MaxVel(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_SCURVE:            /*  set time to get up to speed */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = Set_SCurve(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case DO_MOVE:               /*  execute profile generator and do move   */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        /*  build status response   */
        errorflag = Do_move(ControllerNumber,(int)lv);
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case MOVE_ABORT:            /*  abort current move  */
        /*  build status response   */
        errorflag = AbortMove(ControllerNumber);
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_PHASE_DETECT_POL:  /*  set the polarity of the phase detector  */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = Set_PhaseDetectorPolarity(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_SAMPLE_RATE:       /*  Set servo sample rate       */
```

```
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = Set_SampleRate(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case ZERO_INTEGRATOR:       /*  reset the integrator accumulator    */
            /*  build status response   */
            errorflag = ZeroIntegrator(ControllerNumber);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SET_INT_MODE:          /*  for firmware v2.10 and greater  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetIntegratorMode(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case START_JOG:             /*  set up and start a jog for tuning purposes  */
            errorflag = StartJog(ControllerNumber);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case END_JOG:   /*  cancel the jog  */
            errorflag = EndJog(ControllerNumber);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SET_JOG_PARAM:         /*  set joging parameter        */
            Jp.start = GetLong(&cmd[i]);
            i += sizeof(long);
            Jp.end = GetLong(&cmd[i]);
            i += sizeof(long);
            Jp.index = GetLong(&cmd[i]);
            i += sizeof(long);
            Jp.mode = GetLong(&cmd[i]);
            i += sizeof(long);
            Jp.time = GetLong(&cmd[i]);
            errorflag = SetJogParams(ControllerNumber,Jp);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case CLEAR_INTERRUPT:       /*  clears the beam interrupt flag and restarts the servo
*/
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            /*  build status response   */
            errorflag = ClearInterrupt(ControllerNumber,lv);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case ENABLE_SERVO_FUNCTIONS:/*  enable/disable the servo loop   */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            lv1 = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = EnableServoFunctions(ControllerNumber,lv,lv1);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SET_INT_LIMITS:        /*  set intergrator limits  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = Set_IntegratorLimits(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
```

```
        case ZERO_COUNT:            /*  zero the position counter   */
            errorflag = ZeroCount(ControllerNumber);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SET_MAX_FOLLOW:        /*  set the maximum following error */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = Set_MaxFollowingError(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SET_FOL_GAIN:          /*  set the following error DAC gain    */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = Set_FollowGain(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case HOME_SETUP:            /*  load parameters for home command    */
            Hp.flags = GetLong(&cmd[i]);
            i += sizeof(long);
            Hp.init_velocity = GetLong(&cmd[i]);
            i += sizeof(long);
            Hp.init_dwell = GetLong(&cmd[i]);
            i += sizeof(long);
            Hp.backout_dist = GetLong(&cmd[i]);
            i += sizeof(long);
            Hp.finedwell = GetLong(&cmd[i]);
            i += sizeof(long);
            Hp.finevel = GetLong(&cmd[i]);
            i += sizeof(long);
            Hp.zerodist = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = Set_HomeParams(ControllerNumber,Hp);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case DO_HOME:               /*  go to "home" position               */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            switch((int)lv)
            {
                case 0: /*  home    */
                    errorflag = Do_Home(ControllerNumber);
                    break;
                case 1: /*  calibrate   */
                    errorflag = Do_Calibrate(ControllerNumber);
                    break;
            }
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case RESTORE_SETTINGS:      /*  restore settings from NV rom    */
            /*  build status response   */
            errorflag = RestoreSettings(ControllerNumber);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SAVE_SETTINGS:         /*  save settings into NV rom       */
            /*  build status response   */
            errorflag = SaveSettings(ControllerNumber);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SET_INTERRUPTS:        /*  enable/disable interrupts       */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
```

```
        lv1 = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = SetIrqParams(ControllerNumber,lv,lv1);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case TRIGGER_SERVO:         /*  set trigger flag true   */
        /*  build status response   */
        /*  I can't beleive it, this was never implemented  */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_INPOS_THRESH:      /*  set in position threshold   */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = SetInPositionThreshold(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SET_BUMP:              /*  set the bump threshold  */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = SetBump(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;

    /*
    ** send commands
    */
    case READ_POSITION:         /*  read current position   */
        /*  build status response   */
        errorflag = Read_Position(ControllerNumber,&lv);
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_GOAL:             /*  read back the desired goal  */
        /*  build status response   */
        errorflag = Read_Goal(ControllerNumber,&lv);
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_SCURVE_TIME:      /*  read back SCURVE time   */
        errorflag = Read_SCurve(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_MAX_VEL:          /*  read back maximum veloctiy  */
        errorflag = Read_MaxVel(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_KP:               /*  read back proportional gain     */
        errorflag = Read_Kp(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_KV:               /*  read back differential gain     */
        errorflag = Read_Kv(ControllerNumber,&lv);
```

```
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_KI:               /*  read back integral gain         */
        errorflag = Read_Ki(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_KVFF:             /*  read back feed forward velocity */
        errorflag = Read_Kvff(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_KAFF:             /*  read back feed forward acceleration */
        errorflag = Read_Kaff(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_OFFSET:           /*  read back servo offset          */
        errorflag = Read_Offset(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case SERVO_STATUS:              /*  read back servo card status     */
        errorflag = ServoStatus(ControllerNumber,&stat);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        for(i=0,lp = (long *)&stat;i<6;++i,++lp)
        {
            o = SaveLong(o,*lp);
            outcount += sizeof(long);
        }
        break;
    case READ_PHASE_DETECT_POL: /*  read the phase detector polarity    */
        errorflag = Read_PhaseDetectorPolarity(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case GET_SAMPLE_RATE:       /*  read the servo sample rate  */
        errorflag = Read_SampleRate(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case GET_INT_MODE:          /*  for firmware v2.10 and greater  */
        errorflag = GetIntegratorMode(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_DAC_LEV:          /*  reads the level of the Motor DAC    */
        errorflag = Read_DacLev(ControllerNumber,&lv);
```

```
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case GET_INT_LIMITS:        /*  read the integrator limits  */
        errorflag = Read_IntegratorLimits(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case GET_MAX_FOLLOW:        /*  read the maximum following error    */
        errorflag = Read_MaxFollowingError(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case GET_FOL_GAIN:          /*  get the following error DAC gain    */
        errorflag = Read_FollowGain(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case GET_SN:
        {
            String *S = new String;
            char *s = S->Get();
            /*  get serial number and revision      */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = GetVersionNumber(ControllerNumber,s,COMM_VERSION);
            /*  there will be no serial number in the DSP   */
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            for(i=0;i<16;++i,++o,++outcount)
                *o = s[i];
            delete S;
        }
        break;
    case GET_HOME_PARAMS:       /*  get home parrameters    */
        errorflag = Get_HomeParams(ControllerNumber,&Hp);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,Hp.flags);
        o = SaveLong(o,Hp.init_velocity);
        o = SaveLong(o,Hp.init_dwell);
        o = SaveLong(o,Hp.backout_dist);
        o = SaveLong(o,Hp.finedwell);
        o = SaveLong(o,Hp.finevel);
        o = SaveLong(o,Hp.zerodist);
        outcount += sizeof(long) * 7;
        break;
    case GET_INTERRUPTS:        /*  get interrupt settings  */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = GetIrqParams(ControllerNumber,lv,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_PMEM:             /*  for debug, read program memory  */
        lv = GetLong(&cmd[i]);
```

```
        i += sizeof(long);
        errorflag = ReadPmem(ControllerNumber,&lv,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_YMEM:             /*  for debug, read y data memory   */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = ReadYmem(ControllerNumber,&lv,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_XMEM:             /*  for debug, read x data memory   */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = ReadXmem(ControllerNumber,&lv,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_CALDIST:          /*  read calibration distance   */
        errorflag = Get_Calibration(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case READ_JOG_PARAMS:       /*  read jogging parameters     */
        errorflag = GetJogParams(ControllerNumber,&Jp);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,Jp.start);
        outcount += sizeof(long);
        o = SaveLong(o,Jp.end);
        outcount += sizeof(long);
        o = SaveLong(o,Jp.index);
        outcount += sizeof(long);
        o = SaveLong(o,Jp.mode);
        outcount += sizeof(long);
        o = SaveLong(o,Jp.time);
        outcount += sizeof(long);
        break;
    case READ_INPOSTHRESH:      /*  read in position threshold  */
        errorflag = GetInPositionThreshold(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case GET_BUMP:              /*  read the bump threshold */
        errorflag = ReadBump(ControllerNumber,&lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case SERIAL_CLEAR_FLAG:
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = SetGoalReg(ControllerNumber,lv);
        /*  build status response   */
```

```
//          errorflag = ClearFlag(ControllerNumber);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_FREQUENCY:  /*  get frequency of internal oscilator */
            errorflag = GetFrequency(ControllerNumber,&lv);
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_GET_MAGNITUDE:  /*  get magnitude of position diviation */
            iv = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = GetMagnitude(ControllerNumber,ld,long(iv));
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,ld[0]);
            o = SaveLong(o,ld[1]);
            outcount += sizeof(long) * 2;
            break;
        case SERIAL_GET_FILTFREQ:   /*  get the filter cutoff frequency */
            iv = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = GetFilterBandwidth(ControllerNumber,&lv,long(iv));
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_SET_FREQUENCY:  /*  set frequency of internal sin oscilator */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetFrequency(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_SET_FILTERFREQ: /*  set the cutoff freq of lp filter    */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            iv = GetInt(&cmd[i]);   // get analyzer number
            i += sizeof(int);
            errorflag = SetFilterBandwidth(ControllerNumber,lv,long(iv));
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_SET_AMPLITUDE:
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetAmplitude(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_AMPLITUDE:
            errorflag = GetAmplitude(ControllerNumber,&lv);
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_GET_PHASE:      /*  get phase of cosine output  */
            errorflag = GetPhase(ControllerNumber,&lv);
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
```

```
        case SERIAL_SET_PHASE:      /*  set phase of cosine output  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetPhase(ControllerNumber,lv);
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_DITHERMODE: /*  get mode of dither oscilator    */
            errorflag = GetDitherMode(ControllerNumber,&lv);
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_SET_DITHERMODE: /*  set mode of dither  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetDitherMode(ControllerNumber,lv);
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_SET_GET_RUNOUTCOMP: /*  get runout compensation table   */
            //
            // this command writes the runout comp data to a file
            //
            {
                String *S = new String;
                char *s = S->Get();
                memcpy(s,&cmd[i],8);    //get name of data file
                s[8] = 0;
                i+= 8;
                errorflag = SerialSetGetRunoutComp(ControllerNumber,s);
                delete S;
            }
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_SET_SET_RUNOUTCOMP: /*  recieve runout compensation table   */
            //
            // this command reads the runout comp data from a file
            // and then sends it to the dsp chip
            //
            {
                String *S = new String;
                char *s = S->Get();
                memcpy(s,&cmd[i],8);    //get name of data file
                s[8] = 0;
                i+= 8;
                errorflag = SerialSetSetRunoutComp(ControllerNumber,s);
                delete S;
            }
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_SET_RUNOUTLUT:  /*  select runout comp table    */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetLookupTableSelection(ControllerNumber,lv);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_RUNOUTLUT:  /*  get selected runout comp table  */
            errorflag =  GetLookupTableSelection(ControllerNumber,&lv);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
```

```
        case SERIAL_START_HISTOGRAM:    /*  start histogram data collection */
            lv = GetLong(&cmd[i]);      //get number of samples to take
            i += sizeof(long);
            errorflag = StartHistogram(ControllerNumber,lv);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_HISTOGRAM:  /*  get data from histogram */
            {
                String *S = new String;
                char *s = S->Get();
                errorflag = GetHistogram(ControllerNumber,(long *)s);   //get histogram data
                o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
                for(j=0,lp = (long *)&s[0];j<32;++j,++lp)
                    o = SaveLong(o,*lp);
                delete S;
            }
            outcount += sizeof(long) * 32;
            break;
        case SERIAL_GET_NOTCHQ: /*  get notch filter Q  */
            errorflag = ReadNotchQ(ControllerNumber,&lv,0); //compat with old software
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_SET_NOTCHQ: /*  set notch filter Q  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetNotchQ(ControllerNumber,lv,0);   //compat with old software
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_NOTCHFREQ:  /*  get notch filter frequency  */
            errorflag = ReadNotchFreq(ControllerNumber,&lv,0);  //compat with old software
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_SET_NOTCHFREQ:  /*  set notch filter frequency  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetNotchFreq(ControllerNumber,lv,0);    //compat with old software
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_NOTCHDEPTH: /*  get notch filter depth  */
            errorflag = ReadNotchDepth(ControllerNumber,&lv,0); //compat with old software
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_SET_NOTCHDEPTH: /*  set notch filter depth  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetNotchDepth(ControllerNumber,lv,0);   //compat with old software
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_SET_ANALMODE:
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            errorflag = SetAnalMode(ControllerNumber,lv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
```

```
        break;
    case SERIAL_SETGOALANDMOVE:
        lv = GetLong(&cmd[i]);
        i+= sizeof(long);
        dv = GetDouble(&cmd[i]);
        i+= sizeof(double);
        /*  build status response   */
        if((errorflag = Set_Goal(ControllerNumber,lv)) == 0)
            if((errorflag = Do_move(ControllerNumber,MOVE_ABSOLUTE)) == 0)
                if(DeviceTable) DeviceTable->currenttrack = dv; //set current track
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SERIAL_GET_PESSCALE:   /*  get scale factor for PES    */
        errorflag = GetPesScale(ControllerNumber,&lv);
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case SERIAL_SET_PESSCALE:   /*  set scale factor for PES    */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = SetPesScale(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SERIAL_GET_PESLIMIT:   /*  get limit for PES   */
        errorflag = GetPesLimit(ControllerNumber,&lv);
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case SERIAL_SET_PESLIMIT:   /*  set limit for PES   */
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = SetPesLimit(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SERIAL_RESET_PES:      /*  reset PES counter   */
        errorflag = ResetPesCounter(ControllerNumber);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SERIAL_SET_PESMODE:
        lv = GetLong(&cmd[i]);
        i += sizeof(long);
        errorflag = SetPesMode(ControllerNumber,lv);
        /*  build status response   */
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
    case SERIAL_GET_FPGAID:
        errorflag = GetFPGAID(ControllerNumber,&lv);
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        o = SaveLong(o,lv);
        outcount += sizeof(long);
        break;
    case SERIAL_WRITE_MEM:
        lv = GetLong(&cmd[i]);  //address
        i+=sizeof(long);
        lv1 = GetLong(&cmd[i]); //data
        i+=sizeof(long);
        iv = GetInt(&cmd[i]);   //function
        i += sizeof(int);
        errorflag = WriteDSPMem(ControllerNumber,lv,lv1,iv);
        o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
        break;
```

```
        case SERIAL_GET_NOTCHQ_N:   /*  get notch filter Q  */
            iv = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = ReadNotchQ(ControllerNumber,&lv,iv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_SET_NOTCHQ_N:   /*  set notch filter Q  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            iv = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = SetNotchQ(ControllerNumber,lv,iv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_NOTCHFREQ_N:    /*  get notch filter frequency  */
            iv = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = ReadNotchFreq(ControllerNumber,&lv,iv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_SET_NOTCHFREQ_N:    /*  set notch filter frequency  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            iv = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = SetNotchFreq(ControllerNumber,lv,iv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_NOTCHDEPTH_N:   /*  get notch filter depth  */
            iv = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = ReadNotchDepth(ControllerNumber,&lv,iv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,lv);
            outcount += sizeof(long);
            break;
        case SERIAL_SET_NOTCHDEPTH_N:   /*  set notch filter depth  */
            lv = GetLong(&cmd[i]);
            i += sizeof(long);
            iv = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = SetNotchDepth(ControllerNumber,lv,iv);
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_GET_DATA_STRUCT:
            iv = GetInt(&cmd[i]);   //get type of structure
            i+= sizeof  (int);
            errorflag = GetDataStructures(ControllerNumber,iv,ld);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,ld[0]);
            o = SaveLong(o,ld[1]);
            outcount += sizeof(long) * 2;
            break;
        case SERIAL_GET_MAGNITUDE1:
            iv = GetInt(&cmd[i]);   //get analyzer number
            i+=sizeof(int);
```

```
            errorflag = GetMagnitude1(ControllerNumber,&ld[0],&ld[1],long(iv));
            /* build status response    */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveLong(o,ld[0]);
            o = SaveLong(o,ld[1]);
            outcount += sizeof(long) * 2;
            break;
        case SERIAL_SETGOALANDMOVE2:
            lv = GetLong(&cmd[i]);
            i+= sizeof(long);
            dv = GetDouble(&cmd[i]);
            i+= sizeof(double);
            if(lv < prevGoal)
            {
                if((errorflag = SetGoalAndMove(ControllerNumber,lv-DeviceTable->Overshoot)) == 0
)
                    if((errorflag = SetGoalAndMove(ControllerNumber,lv)) == 0)
                        if(DeviceTable) DeviceTable->currenttrack = dv; //set current track
            }
            else
            {
                if((errorflag = SetGoalAndMove(ControllerNumber,lv)) == 0)
                    if(DeviceTable) DeviceTable->currenttrack = dv; //set current track
            }
            prevGoal = lv;
            /*  build status response   */
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_MICROE_SETAUTOCOMP:     /*  turn on/off micro autocomp  */
            iv = GetInt(&cmd[i]);
            i+= sizeof(int);
            errorflag = SetMicroEAutoComp(iv);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            break;
        case SERIAL_MICROE_GETAUTOCOMP:     /*  get autocomp status */
            errorflag = GetMicroEAutoComp(&iv);
            o = BuildServoStatus(errorflag,ControllerNumber,Command,o,&outcount);
            o = SaveInt(o,iv);
            outcount += sizeof(int);
            break;
        default:
            o = BuildServoStatus(SERVO_GOTNAK,ControllerNumber,Command,o,&outcount);
            break;
    }   /*  end of decode command   */
    /* return back important info   */
    Logdata[LogIndex].ErrorNumber = errorflag;
    *ocnt += outcount;
    *index += i;
    return o;
}

char *BuildHighLevelStatus(int errorflag,int Command,char *obuf,int *ocnt)
{
    char *o;

    o = TelBuildStatusBuffer(errorflag,'H',obuf);
    *o++ = (char)Command;
    *ocnt += 4;     /*  increment output count  */
    return o;
}

static char *ProcessHighLevelCommands(int count,char *cmd,int *index,char *obuf,int *ocnt)
{
    int i;
    char *o;
    int errorflag = 0;
```

```
    int outcount = 0;
    String *S = new String;
    char *s= S->Get();
    int v,v2,v3,v4;
    long lv;
    static long lp[16];

    o = obuf;
    i = *index;
    int Command = (int) cmd[i++];
    Logdata[LogIndex].InSubCommand = Command;

    switch(Command)     /*  decode command  */
    {
        case ACUTRAC_INIT:
            memcpy(s,&cmd[i],8);
            s[8] = 0;
            i+= 8;
            errorflag = AcutracINIT(s); //initialize with file <s>
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_RESET:
            memcpy(s,&cmd[i],8);
            s[8] = 0;
            i+= 8;
            errorflag = AcutracRESET(s);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_START:
            errorflag = AcutracSTART();
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_STOP:
            errorflag = AcutracSTOP();
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_GETRPM:
            errorflag = AcutracGETRPM(&v);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            o = SaveInt(o,v);
            outcount += sizeof(int);
            break;
        case ACUTRAC_GETMAXRPM:
            errorflag = AcutracGETMAXRPM(&v);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            o = SaveInt(o,v);
            outcount += sizeof(int);
            break;
        case ACUTRAC_GETMINRPM:
            errorflag = AcutracGETMINRPM(&v);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            o = SaveInt(o,v);
            outcount += sizeof(int);
            break;
        case ACUTRAC_STARTSPINDLE:
            errorflag = AcutracSTARTSPINDLE();
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_STOPSPINDLE:
            errorflag = AcutracSTOPSPINDLE();
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_SETROTDIR:
            v = GetInt(&cmd[i]);
            i+= sizeof(int);
            errorflag = AcutracSETROTDIR(v);
```

```
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_GETROTDIR:
        errorflag = AcutracGETROTDIR(&v);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case ACUTRAC_SETLOADRPM:
        v = GetInt(&cmd[i]);
        i+=sizeof(int);
        errorflag = AcutracSETLOADRPM(v);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_GETLOADRPM:
        errorflag = AcutracGETLOADRPM(&v);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case ACUTRAC_SEEKTRACK:
        memcpy(s,&cmd[i],10);
        s[10] = 0;
        i+= 10;
        errorflag = AcutracSEEKTRACK(s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_STEPIN:
        errorflag = AcutracSTEPIN();
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_STEPOUT:
        errorflag = AcutracSTEPOUT();
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_GETTRACK:
        errorflag = AcutracGETTRACK(s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        memcpy(o,s,10);
        o += 10;
        outcount += 10;
        break;
    case ACUTRAC_SETMAXTRACK:
        memcpy(s,&cmd[i],10);
        i+=10;
        s[10] = 0;
        errorflag = AcutracSETMAXTRACK(s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_GETMAXTRACK:
        errorflag = AcutracGETMAXTRACK(s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        memcpy(o,s,10);
        o += 10;
        outcount += 10;
        break;
    case ACUTRAC_SETRADIUS:
        v = GetInt(&cmd[i]);
        i += sizeof(int);
        memcpy(s,&cmd[i],8);
        s[8] = 0;
        i+= 8;
        errorflag = AcutracSETRADIUS(v,s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_SETTRACKSIZE:
```

```
        memcpy(s,&cmd[i],10);
        s[10] = 0;
        i+= 10;
        errorflag = AcutracSETTRACKSIZE(s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_GETTRACKSIZE:
        errorflag = AcutracGETTRACKSIZE(s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        memcpy(o,s,10);
        o += 10;
        outcount += 10;
        break;
    case ACUTRAC_GETERRORMESSAGE:
        errorflag = AcutracGETERRORMESSAGE(s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        memcpy(o,s,128);
        o+=128;
        outcount += 128;
        break;
    case ACUTRAC_GETCONFIGMESSAGE:
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        GetLoadConfigError(o);
        o+= 80;
        outcount +=80;
        break;
    case ACUTRAC_GETSTATE:
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        v = AcutracGETSTATE(&v2,&v3,&v4);
        o = SaveInt(o,v);
        o = SaveInt(o,v2);
        o = SaveInt(o,v3);
        o = SaveInt(o,v4);
        outcount += sizeof(int) * 4;
        break;
    case ACUTRAC_GETSTATEMESSAGE:
        errorflag = AcutracGETSTATEMESSAGE(s);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        memcpy(o,s,48);
        o+= 48;
        outcount += 48;
        break;
    case ACUTRAC_SETHEAD:
        v = GetInt(&cmd[i]);
        i += sizeof(int);
        errorflag = AcutracSELECTHEAD(v);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_GETHEAD:
        errorflag = AcutracGETSELECTEDHEAD(&v);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        o = SaveInt(o,v);
        outcount += sizeof(int);
        break;
    case ACUTRAC_EXERSIZE:
        extern Wait *WaitExersize;
        v = GetInt(&cmd[i]);
        i+= sizeof(int);
        WaitExersize->Post();
        WaitExersize->SetCount(v-1);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_STOPEXERSIZE:
        WaitExersize->SetCount(0);
        o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
        break;
```

```
        case ACUTRAC_SETCORRECTION:
            memcpy(s,&cmd[i],20);
            s[20] = 0;
            i+= 20;
            errorflag = AcutracSETCORRECTION(s);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_GETCORRECTION:
            errorflag = AcutracGETCORRECTION(s);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            memcpy(o,s,20);
            o += 20;
            outcount += 20;
            break;
        case ACUTRAC_LIFTHEADS:
            errorflag = AcutracLIFTHEADS();
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_RELOADHEADS:
            errorflag = AcutracRELOADHEADS();
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_GETPIDPARAMS:
            v = GetInt(&cmd[i]);    //param set type
            i += sizeof(int);
            errorflag = AcutracGETPIDPARAMS(v,lp);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            for(v2 = 0;v2<7;++v2)
                o = SaveLong(o,lp[v2]);
            outcount += sizeof(long) * 7;
            break;
        case ACUTRAC_SETPIDPARAMS:
            v = GetInt(&cmd[i]);    //param set type
            i += sizeof(int);
            for(v2=0;v2<7;++v2) //extract data
            {
                lp[v2] = GetLong(&cmd[i]);
                i += sizeof(long);
            }
            errorflag = AcutracSETPIDPARAMS(v,lp);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_SETACTRADIUS:  //set actuator to radius
            memcpy(s,&cmd[i],8);    //get radius from command buffer (ascii)
            s[8] = 0;
            i+= 8;
            errorflag = AcutracGOTORADIUS(s);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_GETPIDPARAMSTYPE:
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            o = SaveInt(o,AcutracGETPIDPARAMSTYPE());
            outcount += sizeof(int);
            break;
        case ACUTRAC_SELECTPARAMSTYPE:
            v = GetInt(&cmd[i]);
            i += sizeof(int);
            errorflag = AcutracSELECTPARAMSTYPE(v);
            o = BuildHighLevelStatus(errorflag,Command,o,&outcount);
            break;
        default:
            o = BuildHighLevelStatus(ACUTRAC_NOTHANDLED,Command,o,&outcount);
            break;
    }   //end of switch(cmd);
    Logdata[LogIndex].ErrorNumber = errorflag;
    *ocnt += outcount;
```

```
    *index += i;
    delete S;
    return o;
}

char *BuildRamDiskStatus(int errorflag,int Command,char *obuf,int *ocnt)
{
    char *o;

    o = TelBuildStatusBuffer(errorflag,'D',obuf);
    *o++ = (char)Command;
    *ocnt += 4;     /*  increment output count  */
    return o;
}

static char *ProcessRamDiskCommands(int count,char *cmd,int *index,char *obuf,int *ocnt)
{
    int i;
    char *o;
    int errorflag = 0;
    int outcount = 0;
    int handle;
    int mode;
    int bytecount;
    char *s;
    String *S;
    static Ffblk Ff;    //this needs to be static for finding directory entries

    o = obuf;
    i = *index;
    int Command = (int) cmd[i++];
    Logdata[LogIndex].InSubCommand = Command;

    switch(Command)     /*  decode command  */
    {
        case ACUTRAC_OPENFILE:
            S = new String;
            s = S->Get();
            memcpy(&s[2],&cmd[i],8);    //copy 8 characters for name
            i+=8;
            s[0]='D';
            s[1]=':';
            s[10] = '\0';
            mode = GetInt(&cmd[i]);
            i+= sizeof(int);
            if((handle = Open(s,mode)) < 0) errorflag = handle;
            o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
            o = SaveInt(o,handle);  //return handle back to host
            outcount += sizeof(int);
            delete S;
            break;
        case ACUTRAC_CLOSEFILE:
            handle = GetInt(&cmd[i]);
            i+=sizeof(int);
            errorflag = Close(handle);
            o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
            break;
        case ACUTRAC_READFILE:
            handle = GetInt(&cmd[i]);
            i+=sizeof(int);
            bytecount = GetInt(&cmd[i]);
            i+=sizeof(int);
            s = new char[bytecount];    //make space for buffer
            bytecount = Read(handle,s,bytecount);   //read data
            //this if statement looks very wierd, but I don't dare change it
            if((bytecount < 0) || (bytecount > (outcount+bytecount+6) ) )
```

```
        {
            bytecount = 0;
            errorflag = 1;
        }
        o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
        o = SaveInt(o,bytecount);
        outcount += sizeof(int);
        if(bytecount)
        {
            memcpy(o,s,bytecount);
            outcount += bytecount;
        }
        delete[] s;
        break;
    case ACUTRAC_WRITEFILE:
        handle = GetInt(&cmd[i]);
        i+=sizeof(int);
        bytecount = GetInt(&cmd[i]);
        i+=sizeof(int);
        s = new char[bytecount];
        memcpy(s,&cmd[i],bytecount);
        i += bytecount;
        errorflag = (int)Write(handle,s,(long)bytecount);
        if(errorflag > 0) errorflag = 0;
        //ideally, this function will say how many bytes were writen
        //but, that seems to be imcompatable
        o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
        delete[] s;
        break;
    case ACUTRAC_FLUSHFILE:
        handle = GetInt(&cmd[i]);
        i+=sizeof(int);
        errorflag = Xio(RAMDISK_FLUSH,handle,(char *)0,(char *)0,0l,0);
        o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
        break;
    case ACUTRAC_DELETEFILE:
        S = new String;
        s = S->Get();
        s[0]='D';
        s[1]=':';
        memcpy(&s[2],&cmd[i],8);
        s[10]='\0';
        i+=8;
        errorflag = Xio(RAMDISK_DELETE,-1,s,(char *)0,0l,0);
        o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
        delete S;
        break;
    case ACUTRAC_FINDFIRST:
        S = new String;
        s = S->Get();
        memcpy(s,&cmd[i],8);
        s[8]='\0';
        i+=8;
        mode = GetInt(&cmd[i]);
        i+=sizeof(int);
        errorflag = FindFirst(s,mode,&Ff);
        o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
        if(!errorflag)
        {
            bytecount = sprintf(s,"%8s  %6ld",Ff.name,Ff.size);
            o = SaveInt(o,bytecount);
            outcount+= sizeof(int);
            memcpy(o,s,bytecount);
            o+= bytecount;
            outcount += bytecount;
        }
```

```
            else
            {
                o = SaveInt(o,0);
                outcount+=sizeof(int);
            }
            delete S;
            break;
        case ACUTRAC_FINDNEXT:
            S = new String;
            s = S->Get();
            errorflag = FindNext(&Ff);
            o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
            if(!errorflag)
            {
                bytecount = sprintf(s,"%8s  %6ld",Ff.name,Ff.size);
                o = SaveInt(o,bytecount);
                outcount+= sizeof(int);
                memcpy(o,s,bytecount);
                o+= bytecount;
                outcount += bytecount;
            }
            else
            {
                o = SaveInt(o,0);
                outcount+=sizeof(int);
            }
            delete S;
            break;
        case ACUTRAC_FORMAT:
            errorflag = Xio(RAMDISK_FORMAT,-1,"D:",(char *)0,0l,0);
            o = BuildRamDiskStatus(errorflag,Command,o,&outcount);
            break;
    }   //end of switch command
    Logdata[LogIndex].ErrorNumber = errorflag;
    *ocnt += outcount;
    *index += i;
    return o;
}

//---------------------------------------------------------------------------
// Misc Support Routines that I cannot figure out where else to put
//---------------------------------------------------------------------------

static int SerialSetGetRunoutComp(int axis,char *name)
{
    String *S = new String;
    char *s = S->Get();
    int handle;
    long *data;
    int retval=0;

    sprintf(s,"D:%s",name);     //create file name
    if((handle = Open(s,WRITE_ONLY)) >= 0)  //open up file
    {
        data = new long[256];
        if((retval = GetRunoutCompTable(axis,data)) == 0)
        {
            Write(handle,(char *)data,1024l);           //write data to file
        }
        Close(handle);
        delete[] data;
    }
    else
    {
        retval = RAMDISK_FILENOTFOUND;
    }
```

```
    delete S;
    return retval;
}

static int SerialSetSetRunoutComp(int axis,char *name)
{
    String *S = new String;
    char *s = S->Get();
    int handle;
    long *data;
    int retval;

    sprintf(s,"D:%s",name);     //create file name
    if((handle = Open(s,READ_ONLY)) >= 0)   //open up file
    {
        data = new long[256];
        Read(handle,(char *)data,1024l);        //read data from file
        Close(handle);
        retval = SetRunoutCompTable(axis,data);
        delete[] data;
    }
    else
    {
        retval = RAMDISK_FILENOTFOUND;
    }
    delete S;
    return retval;
}
```

```
/*
**
** Header file for serlprot.c
**
*/

#ifndef TELPROT__H
#define TELPROT__H

#include "erorcode.h"

#define MAX_DATA    256

typedef  struct  {
    int bytecount;
    int version;
    int devID;
    char buff[MAX_DATA];
}PROT_DATA;

/*
** this data structure matches the one for the IBM PC, except
** the order of the bit fields is reversed.  This is very bad
** since we really do not know if or when bit field may be
** changed
*/

typedef  struct  {
    union  {
        struct  {
            unsigned  dummy:4;        //filler
            unsigned  stopbutton:1;   //indicates if stop button was pushed
            unsigned  overtemp:1;     //power amplifier over temp
            unsigned  xstage:1;       //xstage is in run position
            unsigned  conclamp:1;     //clamp for connector is down
            unsigned  vac:1;          //is vac suck or blow, 1= suck
            unsigned  head:1;         //head is loaded/unloaded
            unsigned  atspeed:1;      //phase locked status
            unsigned  air:1;          //air pressure status
            unsigned  direction:1;    //motor direction
            unsigned  clamp:1;        //spindle clamp on/off status
            unsigned  motor:1;        //motor on/off status
            unsigned  system:1;       //system OK
        }stat;
        unsigned  v;
    }spin;
    union  {
        struct  {
            unsigned  dummy:5;            //filler
            unsigned  inposition:1;       //indicates servo in position
            unsigned  histogram:1;        //indicates histogram finished
            unsigned  protect:1;          //protect error
            unsigned  bumped:1;           //bumped error
            unsigned  follow:1;           //following error
            unsigned  limit_m:1;          //negative limit
            unsigned  limit_p:1;          //positive limit
            unsigned  beam:1;             //Beam interrupted
            unsigned  integrator:1;       //integrator status
            unsigned  servo:1;            //pid loop status
            unsigned  system:1;           //system OK
        }stat;
        unsigned  v;
    }servo1;
    union  {
        struct  {
            unsigned  dummy:5;            //filler
```

```
            unsigned inposition:1;      //indicates servo in position
            unsigned histogram:1;       //indicates histogram finished
            unsigned protect:1;         //protect error
            unsigned bumped:1;          //bumped error
            unsigned follow:1;          //following error
            unsigned limit_m:1;         //negative limit
            unsigned limit_p:1;         //positive limit
            unsigned beam:1;            //Beam interrupted
            unsigned integrator:1;      //integrator status
            unsigned servo:1;           //pid loop status
            unsigned system:1;          //system OK
        }stat;
        unsigned v;
    }servo2;
    union {
        struct {
            unsigned dummy:5;           //filler
            unsigned inposition:1;      //indicates servo in position
            unsigned histogram:1;       //indicates histogram finished
            unsigned protect:1;         //protect error
            unsigned bumped:1;          //bumped error
            unsigned follow:1;          //following error
            unsigned limit_m:1;         //negative limit
            unsigned limit_p:1;         //positive limit
            unsigned beam:1;            //Beam interrupted
            unsigned integrator:1;      //integrator status
            unsigned servo:1;           //pid loop status
            unsigned system:1;          //system OK
        }stat;
        unsigned v;
    }servo3;
    unsigned LedStates;
}SPINSTAT;

typedef struct {
    int HSpindle;       /*  handle for spindle controller   */
    int HTach;          /*  handle for tachometer           */
    int HQuad;          /*  handle for quadrature encoder   */
}SYSTEM;

/*
** structure that describes features of spindle controller
** that may change from version to version
*/

struct FEATURES{
    //data members
    int numofnotchfilters;
    int ConnectorClamp;
    //member functions
    FEATURES();     //constructor
//  ~FEATURES();    //destructor
    char *Save(char *p);    //save the data
};

/*************************************************************************
** Serial Log Classes
*************************************************************************/

struct LOG {
    int InCommand;
    int InSubCommand;
    int InPacketId;
    int InByteCount;
    long InTimeStamp;
    int ErrorNumber;
```

```
    int OutCommand;
    int OutPacketId;
    int OutByteCount;
    long OutTimeStamp;
    long SequenceNumber;
};

/*
** Misc protocol defines
*/

#define HOST_ID     0       //serial ID of host controller


/*
** Status Return Values
*/

//#define STATUS_OK         'A'         /*  Ack */
//#define STATUS_ERROR      'E'         /*  Error       */
//#define STATUS_UNDEFINED  'N'         /*  NAK     */


#ifdef __cplusplus
extern "C" {
#endif

extern int TelReceiveProtocol(int handle,PROT_DATA *buff);
extern void TelSendProtocol(int handle,int devID,int count,char *buff,int logflag);
extern char *TelBuildStatusBuffer(int status,int command,char *b);
extern int TelCommandParser(int count,char *cmd,char *obuf,SYSTEM *sys,int *logflag);
extern void InitProtocol(void);

extern SYSTEM sys;

#ifdef __cplusplus
}
#endif

#endif
```

```
#ifndef HOSTCMD__H
#define HOSTCMD__H

//-------------------------------------------------------------
// Spindle Commands
//-------------------------------------------------------------

#define SCMD_MOTOR              0   /*  Motor On/Off Command    */
#define SCMD_CLAMP              1   /*  Clamp/Unclamp command   */
#define SCMD_DIRECTION          2   /*  Clockwise/CounterClockwise rotation */
#define SCMD_SETSPEED           3   /*  Set rotational Rate     */
#define SCMD_SETACCEL           4   /*  Set acceleration rate   */
#define SCMD_SETDECEL           5   /*  Set slowdown rate       */
#define SCMD_LOCK               6   /*  Set lock at stop on/off */
#define SCMD_SETLOCKCOMP        7   /*  Set lock compensation   */
#define SCMD_STATUS             8   /*  Get Spindle Status      */
#define SCMD_GETACTSPEED        9   /*  Get Actual Speed        */
#define SCMD_GETSETSPEED        10  /*  Get the Set Speed       */
#define SCMD_GETACCEL           11  /*  get acceleration        */
#define SCMD_GETDECEL           12  /*  Get Deceleration        */
#define SCMD_GETLOCKCOMP        13  /*  get lock compensation   */
#define SCMD_SETENCODER         14  /*  Set the Encoder Pitch   */
#define SCMD_GETENCODER         15  /*  Get the Encoder Pitch   */
#define SCMD_SETPOLES           16  /*  Set the number of motor poles   */
#define SCMD_GETPOLES           17  /*  Get the number of motor poles   */
#define SCMD_SETPHASING         18  /*  Set motor phasing       */
#define SCMD_GETPHASING         19  /*  Get motor phasing       */
#define SCMD_USERCOMMAND        20  /*  activate a user command sequence    */
#define SCMD_HEADLOAD           21  /*  load head on disk       */
#define SCMD_GETLOWERLIMIT      22  /*  get pll filter lower limit  */
#define SCMD_SETLOWERLIMIT      23  /*  set pll filter lower limit  */
#define SCMD_GETUPPERLIMIT      24  /*  get pll filter upper limit  */
#define SCMD_SETUPPERLIMIT      25  /*  set pll filter upper limit  */
#define SCMD_AMPLIFIERPOWER     26  /*  turn on amplifier power     */
#define SCMD_GETAMPPOWER        27  /*  get amplifier power status  */
#define SCMD_GETDISPLAYMODE     28  /*  get display mode    */
#define SCMD_SETDISPLAYMODE     29  /*  set display mode    */
#define SCMD_VACCHUCK           30  /*  turn on/off vaccum chuck    */
#define SCMD_GETDACDIRECT       31  /*  get the direct control dac  */
#define SCMD_SETDACDIRECT       32  /*  set the direct control dac  */
#define SCMD_GETLOOPGAIN        33  /*  get the loop gain dac       */
#define SCMD_SETLOOPGAIN        34  /*  set the loop gain dac       */
#define SCMD_SETLOOPCOMP        35  /*  set loop compensation params    */
#define SCMD_GETLOOPCOMP        36  /*  get the loop compensation params    */
#define SCMD_SAVEMOTORPARAMS    37  /*  save the motor parameters to file   */
#define SCMD_LOADMOTORPARAMS    38  /*  load the motor parameters from file */
#define SCMD_INITLUT            39  /*  rebuild the motor drive look up table   */
#define SCMD_GETINITSTATUS      40  /*  find out how much further to go for motor init  */
#define SCMD_SETSPINDLEMODE     41  /*  set source of motor drive for spindle motor */
#define SCMD_GETSPINDLEMODE     42  /*  get source of motor drive for spindle motor */
#define SCMD_MOTORONOFFLL       43  /*  low level motor on off command  */
#define SCMD_SETMOTDACMODE      44  /*  set access to motor dac */
#define SCMD_SETMOTORDAC        45  /*  set motor dac values    */
#define SCMD_GETAMPENABLE       46  /*  get state of amplifier enable bit   */
#define SCMD_CONNECTORCLAMP     47  /*  activate connector clamp    */
#define SCMD_SETSTART           48  /*  set start pulse position    */
#define SCMD_GETSTART           49  /*  get start pulse position    */
#define SCMD_RESETSTOPSTAT      50  /*  reset the status bit for stop button    */
#define SCMD_UPLOADSLUT         51  /*  upload commutator table for spindle */

//-------------------------------------------------------------
// Servo Commands
//-------------------------------------------------------------


#define SET_KP                  0   /*  set proportional gain   */
```

```
#define SET_KV                  1   /*  set differential gain   */
#define SET_KI                  2   /*  set integral gain       */
#define SET_KVFF                3   /*  set feed forward velocity gain  */
#define SET_KAFF                4   /*  set feed forward acceleration gain  */
#define SET_OFFSET              5   /*  set servo offset    */
#define SET_GOAL                6   /*  set desired position    */
#define SET_MAXVEL              7   /*  set maximum velocity    */
#define SET_SCURVE              8   /*  set time to get up to speed */
#define DO_MOVE                 9   /*  execute profile generator and do move   */
#define MOVE_ABORT              10  /*  abort current move  */
#define SET_PHASE_DETECT_POL    11  /*  set the polarity of the phase detector  */
#define SET_SAMPLE_RATE         12  /*  Set servo sample rate       */
#define ZERO_INTEGRATOR         13  /*  reset the integrator accumulator    */
#define SET_INT_MODE            14  /*  for firmware v2.10 and greater  */
#define START_JOG               15  /*  set up and start a jog for tuning purposes  */
#define END_JOG                 16  /*  cancel the jog  */
#define SET_JOG_PARAM           17  /*  set joging parameter        */
#define CLEAR_INTERRUPT         18  /*  clears the beam interrupt flag and restarts the servo
*/
#define ENABLE_SERVO_FUNCTIONS  19  /*  enable/disable the servo loop   */
#define SET_INT_LIMITS          20  /*  set intergrator limits  */
#define ZERO_COUNT              21  /*  zero the position counter   */
#define SET_MAX_FOLLOW          22  /*  set the maximum following error */
#define SET_FOL_GAIN            23  /*  set the following error DAC gain    */
#define HOME_SETUP              24  /*  load parameters for home command    */
#define DO_HOME                 25  /*  go to "home" position               */
#define RESTORE_SETTINGS        26  /*  restore settings from NV rom    */
#define SAVE_SETTINGS           27  /*  save settings into NV rom       */
#define SET_INTERRUPTS          28  /*  enable/disable interrupts       */
#define TRIGGER_SERVO           29  /*  set trigger flag true   */
#define SET_INPOS_THRESH        30  /*  set in position threshold   */
#define SET_BUMP                31  /*  set the bump threshold  */
#define READ_POSITION           32  /*  read current position   */
#define READ_GOAL               33  /*  read back the desired goal  */
#define READ_SCURVE_TIME        34  /*  read back SCURVE time   */
#define READ_MAX_VEL            35  /*  read back maximum velocty  */
#define READ_KP                 36  /*  read back proportional gain     */
#define READ_KV                 37  /*  read back differential gain     */
#define READ_KI                 38  /*  read back integral gain         */
#define READ_KVFF               39  /*  read back feed forward velocity */
#define READ_KAFF               40  /*  read back feed forward acceleration */
#define READ_OFFSET             41  /*  read back servo offset          */
#define SERVO_STATUS            42  /*  read back servo card status     */
#define READ_PHASE_DETECT_POL   43  /*  read the phase detector polarity    */
#define GET_SAMPLE_RATE         44  /*  read the servo sample rate  */
#define GET_INT_MODE            45  /*  for firmware v2.10 and greater  */
#define READ_DAC_LEV            46  /*  reads the level of the Motor DAC    */
#define GET_INT_LIMITS          47  /*  read the integrator limits  */
#define GET_MAX_FOLLOW          48  /*  read the maximum following error    */
#define GET_FOL_GAIN            49  /*  get the following error DAC gain    */
#define GET_SN                  50  /*  get serial number and revision      */
#define GET_HOME_PARAMS         51  /*  get home parrameters    */
#define GET_INTERRUPTS          52  /*  get interrupt settings  */
#define READ_PMEM               53  /*  for debug, read program memory  */
#define READ_YMEM               54  /*  for debug, read y data memory   */
#define READ_XMEM               55  /*  for debug, read x data memory   */
#define READ_CALDIST            56  /*  read calibration distance   */
#define READ_JOG_PARAMS         57  /*  read jogging parameters     */
#define READ_INPOSTHRESH        58  /*  read in position threshold  */
#define GET_BUMP                59  /*  read the bump threshold */
#define SERIAL_CLEAR_FLAG       60  /*  clear single step flag in DSP   */
#define SERIAL_GET_FREQUENCY    61  /*  get frequency of internal oscilator */
#define SERIAL_GET_MAGNITUDE    62  /*  get magnitude of position diviation */
#define SERIAL_GET_FILTFREQ     63  /*  get the filter cutoff frequency */
#define SERIAL_SET_FREQUENCY    64  /*  set frequency of internal sin oscilator */
#define SERIAL_SET_FILTERFREQ   65  /*  set the cutoff freq of lp filter    */
```

```
#define SERIAL_SET_AMPLITUDE    66  /*  set the amplitude of the sine oscilator */
#define SERIAL_GET_AMPLITUDE    67  /*  get the amplitude of the sine oscilator */
#define SERIAL_GET_PHASE        68  /*  get phase of cosine output  */
#define SERIAL_SET_PHASE        69  /*  set phase of cosine output  */
#define SERIAL_GET_DITHERMODE   70  /*  get mode of dither oscilator    */
#define SERIAL_SET_DITHERMODE   71  /*  set mode of dither  */
#define SERIAL_SET_GET_RUNOUTCOMP   72  /*  get runout compensation table   */
#define SERIAL_SET_SET_RUNOUTCOMP   73  /*  recieve runout compensation table   */
#define SERIAL_SET_RUNOUTLUT    74  /*  select runout comp table    */
#define SERIAL_GET_RUNOUTLUT    75  /*  get selected runout comp table  */
#define SERIAL_START_HISTOGRAM  76  /*  start histogram data collection */
#define SERIAL_GET_HISTOGRAM    77  /*  get data from histogram */
#define SERIAL_GET_NOTCHQ       78  /*  get notch filter Q  */
#define SERIAL_SET_NOTCHQ       79  /*  set notch filter Q  */
#define SERIAL_GET_NOTCHFREQ    80  /*  get notch filter frequency  */
#define SERIAL_SET_NOTCHFREQ    81  /*  set notch filter frequency  */
#define SERIAL_GET_NOTCHDEPTH   82  /*  get notch filter depth  */
#define SERIAL_SET_NOTCHDEPTH   83  /*  set notch filter depth  */
#define SERIAL_SET_ANALMODE     84  /*  set analisys input mode */
#define SERIAL_SETGOALANDMOVE   85  /*  set goal, do move, set current track    */
#define SERIAL_GET_PESSCALE     86  /*  get scale factor for PES    */
#define SERIAL_SET_PESSCALE     87  /*  set scale factor for PES    */
#define SERIAL_GET_PESLIMIT     88  /*  get limit for PES   */
#define SERIAL_SET_PESLIMIT     89  /*  set limit for PES   */
#define SERIAL_RESET_PES        90  /*  reset PES counter   */
#define SERIAL_SET_PESMODE      91  /*  set mode of PES counter */
#define SERIAL_GET_FPGAID       92  /*  get id descriptor from FPGA */
#define SERIAL_WRITE_MEM        93  /*  write data to DSP memory    */
#define SERIAL_GET_NOTCHQ_N     94  /*  get notch filter Q  */
#define SERIAL_SET_NOTCHQ_N     95  /*  set notch filter Q  */
#define SERIAL_GET_NOTCHFREQ_N  96  /*  get notch filter frequency  */
#define SERIAL_SET_NOTCHFREQ_N  97  /*  set notch filter frequency  */
#define SERIAL_GET_NOTCHDEPTH_N 98  /*  get notch filter depth  */
#define SERIAL_SET_NOTCHDEPTH_N 99  /*  set notch filter depth  */
#define SERIAL_GET_DATA_STRUCT  100 /*  get address and size of data structures */
#define SERIAL_GET_MAGNITUDE1   101 /*  get magnitude of spetrum filter */
#define SERIAL_SETGOALANDMOVE2  102 /*  alternate set goal and move */
#define SERIAL_MICROE_SETAUTOCOMP   103 /*  turn on/off micro autocomp  */
#define SERIAL_MICROE_GETAUTOCOMP   104 /*  get autocomp status */

/*-------------------------------------------------------------------------
** High Level Commands
**-------------------------------------------------------------------------*/

#define ACUTRAC_INIT            0
#define ACUTRAC_RESET           1
#define ACUTRAC_START           2
#define ACUTRAC_STOP            3
#define ACUTRAC_SETRPM          4
#define ACUTRAC_GETRPM          5
#define ACUTRAC_GETMAXRPM       6
#define ACUTRAC_GETMINRPM       7
#define ACUTRAC_STARTSPINDLE    8
#define ACUTRAC_STOPSPINDLE     9
#define ACUTRAC_SETROTDIR       10
#define ACUTRAC_GETROTDIR       11
#define ACUTRAC_SETLOADRPM      12
#define ACUTRAC_GETLOADRPM      13
#define ACUTRAC_SEEKTRACK       14
#define ACUTRAC_STEPIN          15
#define ACUTRAC_STEPOUT         16
#define ACUTRAC_GETTRACK        17
#define ACUTRAC_SETMAXTRACK     18
#define ACUTRAC_GETMAXTRACK     19
#define ACUTRAC_SEEKOFFSET      20
#define ACUTRAC_GETCURRENTOFF   21
```

```
#define ACUTRAC_GETMAXOFFSET    22
#define ACUTRAC_SETMAXOFFSET    23
#define ACUTRAC_SETAUTOCENTER   24
#define ACUTRAC_GETAUTOCENTER   25
#define ACUTRAC_SETRADIUS       26
#define ACUTRAC_SETTRACKSIZE    27
#define ACUTRAC_GETTRACKSIZE    28
#define ACUTRAC_SETTRACKSTEPS   29
#define ACUTRAC_GETTRACKSTEPS   30
#define ACUTRAC_GETRADIUS       31
#define ACUTRAC_GETERRORMESSAGE 32
#define ACUTRAC_GETCONFIGMESSAGE 33
#define ACUTRAC_GETSTATE        34
#define ACUTRAC_GETSTATEMESSAGE 35
#define ACUTRAC_SETHEAD         36
#define ACUTRAC_GETHEAD         37
#define ACUTRAC_EXERSIZE        38
#define ACUTRAC_STOPEXERSIZE    39
#define ACUTRAC_SETCORRECTION   40
#define ACUTRAC_GETCORRECTION   41
#define ACUTRAC_LIFTHEADS       42
#define ACUTRAC_RELOADHEADS     43
#define ACUTRAC_GETPIDPARAMS    44
#define ACUTRAC_SETPIDPARAMS    45
#define ACUTRAC_SETACTRADIUS    46
#define ACUTRAC_GETPIDPARAMSTYPE    47
#define ACUTRAC_SELECTPARAMSTYPE    48

/*-------------------------------------------------------------------------
** Ramdisk Commands
**-----------------------------------------------------------------------*/

#define ACUTRAC_OPENFILE        0
#define ACUTRAC_CLOSEFILE       1
#define ACUTRAC_READFILE        2
#define ACUTRAC_WRITEFILE       3
#define ACUTRAC_FLUSHFILE       4
#define ACUTRAC_DELETEFILE      5
#define ACUTRAC_FINDFIRST       6
#define ACUTRAC_FINDNEXT        7
#define ACUTRAC_FORMAT          8

#endif  /*  hostcmd__h  */
```

```c
#include <math.h>
#include <stdio.h>
#include "sinlut.h"

void MakeSineLut(long n,unsigned poles,void (*callback)(double v))
{
    /*
    ** n is the number of elements in LUT
    **
    ** callback is a function that is called to put the value where it
    ** belongs
    */
    double twopi;
    long i;

    twopi = (double)poles * 8.0 * atan(1) / (double)n;
    n = n * 2 + 1;
    for(i=0;i<n;++i)
    {
        (*callback)(sin(twopi * (double)i));
    }
}
```

```c
#ifndef SINLUT__H
#define SINLUT__H


#ifdef __cplusplus
extern "C"{
#endif
extern void MakeSineLut(long n,unsigned poles,void (*callback)(double v));

#ifdef __cplusplus
}
#endif

#endif /*  SINLUT__H  */
```

```
/*****************************************************************************
**
** Interface to the spindle controller chip
**
** When this code is converted to MULTITASKING, it will require a semaphore
** to make sure that the Control Shadow is not interfered with, or have
** task switching blocked, or something.
**
** Serious changes have been made to this file for the REV B PCB
** 3-19-98
**
*****************************************************************************/

#include <stdio.h>
#include "spinchip.h"
#include "task.h"
#include "frontled.h"
#include "cio.h"
#include "rs232.h"
#include "strings.h"

extern int ConsolHandle;

union SCTL {
    SPIN_CR cr;
    unsigned v;
} SpindleCTLShadow;

Wait *CtlExclude;
Wait *DirectionChanged;
Wait *SpindleAmplifierFault;
Wait *FrontPanelFault;
static void FrontPanelFaultTask(void);

extern "C" void InitSpinChip(void)
{
    TCB *t;

    CtlExclude = new Wait(1,"CTLBlocker");
    SpindleCTLShadow.v = 0; //zap this value
    //create task to do front panel, very high priority
    t = CreateTask(FrontPanelFaultTask,2048,2000,"Front Panel");
    ActiveTasks->Insert(t);
}

extern "C" void SpinChipPllFeedback(unsigned c)
{
    /*************************************************************
    ** this function selects where the feedback for the PLL comes
    ** from.  It selects either the AQUAD input from the encoder
    ** wheel or the edges derived from AQuadB.
    *************************************************************/

    CtlExclude->Pend();
    SpindleCTLShadow.cr.PllFeedback = c;
    *((volatile unsigned *)(SPIN_CONTROL)) = SpindleCTLShadow.v;
    CtlExclude->Post();
}

extern "C" void SpinChipFilterSelect(unsigned c)
{
    CtlExclude->Pend();
    SpindleCTLShadow.cr.FilterSelect = c;
    *((volatile unsigned *)(SPIN_CONTROL)) = SpindleCTLShadow.v;
    CtlExclude->Post();
}
```

```
extern "C" unsigned SpinChipGetFilterSelect(void)
{
    return SpindleCTLShadow.cr.FilterSelect;
}

extern "C" unsigned SpinChipGetPllFeedback(void)
{
    return SpindleCTLShadow.cr.PllFeedback;
}

extern "C" void SpinChipPllMode(unsigned c)
{
    CtlExclude->Pend();
    SpindleCTLShadow.cr.PllMode = c;
    *((volatile unsigned *)(SPIN_CONTROL)) = SpindleCTLShadow.v;
    CtlExclude->Post();
}

extern "C" unsigned SpinChipGetPllMode(void)
{
    return SpindleCTLShadow.cr.PllMode;
}

extern "C" void SpinChipDacSelect(unsigned c)
{
    CtlExclude->Pend();
    SpindleCTLShadow.cr.DacStrobeSelect = c;
    *((volatile unsigned *)(SPIN_CONTROL)) = SpindleCTLShadow.v;
    CtlExclude->Post();
}

extern "C" void SpinChipCCW(unsigned c)
{
    CtlExclude->Pend();
    SpindleCTLShadow.cr.Ccw = c;
    *((volatile unsigned *)(SPIN_CONTROL)) = SpindleCTLShadow.v;
    CtlExclude->Post();
}

extern "C" void SpinChipPolarity(unsigned c)
{
    CtlExclude->Pend();
    SpindleCTLShadow.cr.Polarity = c;
    *((volatile unsigned *)(SPIN_CONTROL)) = SpindleCTLShadow.v;
    CtlExclude->Post();
}

extern "C" void SpinChipDacMode(unsigned c)
{
    CtlExclude->Pend();
    SpindleCTLShadow.cr.DacMode = c;
    *((volatile unsigned *)(SPIN_CONTROL)) = SpindleCTLShadow.v;
    CtlExclude->Post();
}

extern "C" void SpinChipTachIrqEnable(unsigned c)
{
    CtlExclude->Pend();
    SpindleCTLShadow.cr.TachIrqEnable = c;
    *((volatile unsigned *)(SPIN_CONTROL)) = SpindleCTLShadow.v;
    CtlExclude->Post();
}

/*****************************************************************
**
```

```
** Misc Interrupts for Spindle
**
*****************************************************************/

static unsigned SpinIOIrqEn = 0;
volatile long STimer=0;

extern "C" void SpinIOInitIrq(void)
{

    SpinIOEnableIrq(SPIN_IOTICKER);         /*  enable real time clock  */
    *((unsigned *)SPIN_WDIOVECTOR) = 0xc1;  /*  spin vector is at 0x300, and
                                             the extra 1 enables interrupt   */
}

extern "C" void SpinIOEnableIrq(unsigned mask)
{
    SpinIOIrqEn |= mask;
    *((unsigned *)SPIN_WDINTENABLE) = SpinIOIrqEn;
}

extern "C" void SpinIODisableIrq(unsigned mask)
{
    SpinIOIrqEn &= ~mask;
    *((unsigned *)SPIN_WDINTENABLE) = SpinIOIrqEn;
}

extern "C" unsigned SpinGetInputLevel(unsigned mask)
{
    //---------------------------------------
    // This function returns 0 if switch is not pushed
    // returns 1 if switch is pushed
    //---------------------------------------
    unsigned v;

    v = *((volatile unsigned *)SPIN_RDINTLEVEL);    //read interrupt levels
    return (v & mask)?0:1;
}

extern "C" void HandleIrq0(void)
{
    //--------------------------------
    // Indicates Front Panel state
    // has changed
    //--------------------------------
    if(FrontPanelFault)
    {
        if(FrontPanelFault->GetCount() < 0)
            FrontPanelFault->Post(SpinGetInputLevel(SPIN_FPFAULT));
    }
}

extern "C" void HandleIrq1(void)
{
    //--------------------------------
    // Not Used
    //--------------------------------
}

extern "C" void HandleIrq2(void)
{
    //--------------------------------
    // Not Used
    //--------------------------------
}
```

```
extern "C" void HandleIrq3(void)
{
    //-----------------------------------
    // spindle amplifier interrupt
    // Low Input= FAULT
    // High Input = Good
    // Interrupt occurs every time state
    // changes
    //-----------------------------------
    if(SpindleAmplifierFault)   //is there a semaphore?
    {
        if(SpindleAmplifierFault->GetCount() < 0)
            SpindleAmplifierFault->Post(SpinGetInputLevel(SPIN_AMPFAULT));
    }

}

extern "C" void HandleIrq4(void)
{
    //-----------------------------------
    // this interrupt supposedly handles
    // a change in interrupt direction
    //-----------------------------------
    if(DirectionChanged)
    {
//      if(DirectionChanged->GetCount() < 0)
        //return back level of interrupt line
            DirectionChanged->Post();
    }
}

//*********************************************************************
//
// Don't know where else to put this.
//
// Task for handling a front panel FAULT
//
//*********************************************************************

static void FrontPanelFaultTask(void)
{
    int v;
    int c;
    char *s = new char[256];

    FrontPanelFault = new Wait(0,"FrontPanelFault");        //create semaphore for front panel
//
// Pain in the neck, enable this later
//
//  SpinIOEnableIrq(SPIN_FPFAULT);      //enable front panel interrupt
    while(1)                              //this is a TASK, loop forever
    {
        v = FrontPanelFault->Pend();    //wait for a front panel fault
        c = sprintf(s,"Front Panel FAULT!:%d\r\n",v);
        Write(ConsolHandle,s,c);
    }
}
```

```
/*****************************************************************
**
** Header file for routines that deal with various aspects of the
** Spindle Controller chip
**
** Serious changes have been made to this file for the REV B PCB
** 3-19-98
**
*****************************************************************/

#ifndef SPINCHIP__H
#define SPINCHIP__H

#include "queue.h"

/*
** defines for control register
*/
    /*  Access to SinLUT and Sin Dacs --SpinChipDacMode()   */

#define SPIN_DACMODENORMAL  0
#define SPIN_DACMODEDIRECT  1

    /*  PLL Feedback Source */

#define SPIN_ENCODEREDGES   0
#define SPIN_ENCODERAQUAD   1

    /*  PLL Loop Modes  */

#define SPIN_PLL            1
#define SPIN_PLLDIRECT      2
#define SPIN_PLLDSP         3

/*****************************************************************
**
** Structure for spindle chip control register
**
*****************************************************************/

typedef  struct  {
    unsigned  ct4:2;             //unused          (Msb)
    unsigned  PllFeedback:1;     //select AQuad or edges
    unsigned  FilterSelect:3;    //Select loop filter for pll
    unsigned  PllMode:2;         //Select Pll Mode
    unsigned  ct2:1;             //unused
    unsigned  DacStrobeSelect:2; //select 8 bit dac
    unsigned  Ccw:1;             //direction control bit
    unsigned  Polarity:1;        //polarity for position decoder
    unsigned  DacMode:2;         //mode of output dac
    unsigned  TachIrqEnable:1;       //enable interrupts
} SPIN_CR;

/*
** defines for status register
** same as the control register, except as noted
*/

#define SPIN_IRQSTAT        0x01    /*  indicates interrupt active  */

/*
** addressing defines
*/

#define SPIN_BASEADDRESS    0xffdc0000
```

```
#define SPIN_N              (SPIN_BASEADDRESS+0)        /*  pll 'n' divider reg */
#define SPIN_M              (SPIN_BASEADDRESS+2)        /*  pll 'm' divider reg */
#define SPIN_PHASE1         (SPIN_BASEADDRESS+4)        /*  phase offset #1     */
#define SPIN_PHASE2         (SPIN_BASEADDRESS+6)        /*  phase offset #2     */
#define SPIN_RDTACH_L       (SPIN_BASEADDRESS+8)        /*  tach register read low  */
#define SPIN_RDTACH_H       (SPIN_BASEADDRESS+10)       /*  tach register read high */
#define SPIN_WDCLRCOUNT     (SPIN_BASEADDRESS+8)        /*  clear position counter  */
#define SPIN_WDVECTOR       (SPIN_BASEADDRESS+10)       /*  write to interrupt vector for tach
                                                            upper byte is divider for TACH timeb
ase */
#define SPIN_PHASELIMIT     (SPIN_BASEADDRESS+12)       /*  number of words/cycle */
#define SPIN_CONTROL        (SPIN_BASEADDRESS+14)       /*  control register for spindle control
 (wd)   */
#define SPIN_STATUS         (SPIN_BASEADDRESS+14)       /*  status register for spindle control
(rd)    */
#define SPIN_RDPHASECOUNT   (SPIN_BASEADDRESS+16)       /*  read the phase counter  */
#define SPIN_WRITEDAC       (SPIN_BASEADDRESS+16)       /*  write to DAC port   */
#define SPIN_RDPOSITION     (SPIN_BASEADDRESS+18)       /*  read position register  */
#define SPIN_WDLATCHPOS     (SPIN_BASEADDRESS+18)       /*  latch position register */
#define SPIN_RDINTLEVEL     (SPIN_BASEADDRESS+20)       /*  read external interrupt levels  */
#define SPIN_RDINTFLAGS     (SPIN_BASEADDRESS+22)       /*  read external interrupt flags   */
#define SPIN_WDCLRFLAGS     (SPIN_BASEADDRESS+20)       /*  write data to clear flags   */
#define SPIN_WDINTENABLE    (SPIN_BASEADDRESS+22)       /*  write to interrupt enable register
*/
#define SPIN_WDIOVECTOR     (SPIN_BASEADDRESS+24)       /*  write to interrupt vector register f
or IO   */
#define SPIN_RDTIME         (SPIN_BASEADDRESS+24)       /*  read high res real time */
#define SPIN_START          (SPIN_BASEADDRESS+26)       /*  start sector register   */
#define SPIN_STOP           (SPIN_BASEADDRESS+28)       /*  stop sector register    */
#define SPIN_RDTACHVECTOR   (SPIN_BASEADDRESS+30)       /*  read vector register for TACH interr
upt */
#define SPIN_WDDIVIDE       (SPIN_BASEADDRESS+30)       /*  write to pll divider    */

/*
** bit masks for interrupt enables for spindle chip IO
*/

#define SPIN_FPFAULT    0x01        /*  enable for Front Panel fault    */
#define SPIN_IOEN1      0x02
#define SPIN_IOEN2      0x04
#define SPIN_AMPFAULT   0x08        //enable for Amplifier Fault
#define SPIN_DIR        0x10        //enable for Spindle Direction change
#define SPIN_IOTICKER   0x20
#define SPIN_ENUP       0x40
#define SPIN_ENDOWN     0x80


/*
** function prototypes
*/

extern Wait *DirectionChanged;
extern Wait *SpindleAmplifierFault;
extern Wait *FrontPanelFault;


#ifdef __cplusplus
extern "C" {
#endif

extern void InitSpinChip(void);
extern void SpinChipPllFeedback(unsigned c);
extern void SpinChipFilterSelect(unsigned c);
extern void SpinChipPllMode(unsigned c);
extern void SpinChipDacSelect(unsigned c);
extern void SpinChipCCW(unsigned c);
```

```
extern void SpinChipPolarity(unsigned c);
extern void SpinChipDacMode(unsigned c);
extern void SpinChipTachIrqEnable(unsigned c);
extern unsigned SpinChipGetFilterSelect(void);
extern unsigned SpinChipGetPllFeedback(void);
extern unsigned SpinChipGetPllMode(void);

extern void SpinIOInitIrq(void);
extern void SpinIOEnableIrq(unsigned mask);
extern void SpinIODisableIrq(unsigned mask);
extern unsigned SpinGetInputLevel(unsigned mask);

extern void InitSwitchTasks(void);

extern TSemaphore *ContinueSwitch,*StopSwitch,*AirPressure;     //semaphores for Spin Stand switches

#ifdef __cplusplus
}
#endif

#endif
```

```
/*
** This file starts off with various peices of info
*/

/*************************************************************************
                                Revision History

    Version 1.01.00 Aug 17, 1998
        Task.cpp
        Swap.s
                Removed extraneous "Blocking" action from all of the
                routines that did context swapping.  Revised amount
                stack was cleaned up by in IrqSwap().

    Version 1.01.01 Sept 14, 1998
        I2C.cpp
                Reversed polarity of sense on air pressure switch in the
                AirPressure task.

    Version 1.02.00 Oct 9, 1998
        I2C.cpp
                Added acceleration/deceleration to motor starting
                and stopping
        Accutrac.cpp
                Added Magneto Optical Load and Unload routines
                Extended Range of SEEKTRACK +- 100 tracks beyond min and
                max track
        Servo.cpp
                Fixed ZeroCount function.
    Version 1.02.01 Oct 15, 1998
            I2C.cpp
                In SpindleMotor task,initialized motor direction status bit.
                In HeadLoader task, initialized head status bit
    Version 1.02.02 Oct 27, 1998
            Acutrac.cpp
                Fixed bugs in errors returned by various function and
                ran the result through ChkErr to prevent system lockup
                Fixed bug that caused syncronization problem in Head Load/
                Unload and Exersize functions
            I2C.cpp
                Made single function to process system shut downs.  Called
                by Stop Button, Air Pressure and Amplifier Fault exceptions
    Version 1.02.03 Jan 6, 1999
            Acutrac.cpp
                In Stop and Reset for HSA driver, added call to send
                -1 to head select.  This is supposed to turn things off
            Serlprot.cpp
                Added commands to upload and download files for runout
                compensation table.
    Version 1.03.00 Jan 20, 1999
            Acutrac.cpp
                Completed HGA drivers

    Version 1.04.00 March 1, 1999
            Serlprot.cpp
                Changed the way that error codes are returned to host
                This will make it easier to find out what the problem
                was.
                ***This revision will not be compatable with host software
                entirely.***
    Version 1.04.01 March 8, 1999
            Main.cpp
                Moved initialization of ConsolHandle from TaskRs232 to
                StartMultitasking.
    Version 1.04.02 March 15,1999
            Accutrac.cpp
```

```
            HSAReset() Fixed problem with not initializing under certain
            circumstances with device still on chuck.
        Servo.cpp
            Raised priority of restore servo parameters task to make
            restore happen first.
Version 1.04.03 March 18,1999
        Serlprot.cpp
            Added new spindle command to turn motor on and off
            directly.  This is a low low level command to be
            used primarily in setting up the spindle motor.
Version 1.04.04 March 24, 1999
        Spindle.cpp
            Fixed bug that would allow user to set acceleration and
            deceleration to negative value or worse, 0
            Fixed bug that when you change encoder pitch, hardware
            was not updated unless you turned off the power.

Version 1.05.00 June 7, 1999
        Accutrac.cpp
            Added functions for lifting and reloading heads
        Serlprot.cpp
            added commands for lifting and reloading heads
Version 1.06.00 July 19, 1999
        Servo.cpp
            added code for histogram
        Added code for histogram into DSP code
Version 1.07.00 August 26, 1999
        SCFIG.SYN
            Added field in parser for ACCELERATION
Version 1.07.01 August 31, 1999
        S_HRDWAR.C
            Made DSP firmware so that a soft config bit
            chooses between using a linear scale and
            a pair of limit switchs to decide where linear
            stage is.
        S_EVENT.C
            made same changes here.
Version 1.08.00 Sept 8, 1999
        spinpga.c
            Version 1.02 implemented of spga023 gate array.
            still use spindl11 for spindle gate array.
Version 1.10.00 Sept 23,1999
        Added Notch Filter to PID loop in servo control.
        Many files changed.
        Added commands for setting Notch filter parameters
        Added command for enabling Notch filter
Version 1.10.01 Nov 1, 1999
        I2C.CPP
            Changed way motor is started and stop.  When under PID
            control set upper and lower limits to MAX
        Spindle.cpp
            in Putc handler, updates upper/lower limits dacs
Version 1.10.02 Nov 2, 1999
        I2C.cpp
            Added more calls to change max motor drive when changing
            RPM and when doing an ermergency stop
        spindle.cpp
            Made SPIN_GETDAC command return values of limits properly
Version 1.10.03 Nov 29, 1999
        spindle.cpp
            SetLowerLimits did not update checksum of spindle data
            added command to get state of amplifier enable
        serprot.cpp
            added command to get status of amp enable
Version 1.20.00 December 10, 1999
        serprot.cpp
```

```
        accutrac.cpp
            Added code to allow firmware to change tuning parameters
            depending on what the conditions are.
    Version 1.20.01 Feb 1, 2000
        accutrac.cpp
            Actually complete the above change
            Blanked display when message is no longer needed
    Version 1.20.02 Feb 23, 2000
        accutrac.cpp
            Changed way DeviceTable is allocated.  Created constructor
            and destructor for SpinstandDeviceTable structure.
        many other files
            Fixed various minor bugs
    Version 1.20.03 Feb 24, 2000
        i2c.cpp
            Fixed big time shutdown for HGA
    Version 1.20.04 Feb 28, 2000
        scfig.syn
            Removed extraneous "eol" from grammer following ROTARY
            setup production
    Version 1.20.05 March 1, 2000
        accutrac.cpp
            AcutracRESET: checked for air pressure before doing
            reset sequence
        errorcode.h
            Added ACUTRAC_LOWAIR error
    Version 1.30.00 March 10, 2000
        many files
            changeds have been made to be compatable with
            spinstand version 2.2
    Version 1.30.01 March 17, 2000
        quad.cpp
            made modification to make fine adjustments hopefully
            a little easier
    Version 1.30.02
        accutrac.cpp
            AcutracDeviceTable::AcutracDeviceTable():
                set memory to zero
    Version 1.40.00 April 13, 2000
        rs232.cpp
            changed baud rate of host port to 38.4KB
        protocol.cpp
            added new command to set goal, seek to goal
            and set current track to speed things up.
        servo.cpp
            made small improvement in recieve interrupt routine
            to increase throughput
    Version 1.50.00 April 20, 2000
        Eliminated a lot of unnesesary or duplicate code to shrink
        rom image.  Some commands were eliminated.
    Version 1.51.00 June 8, 2000
        Task.cpp
            Added "pending" member to TCB
            Added "Kill" function to TSemaphore
            Deleted "SendKill" function from TSemaphore
            changed way TDelete function deleted tasks.
            Deleted "RemoveTask" function from TSemaphore
    Version 1.51.01 June 19, 2000
        Strings.cpp
            New file.  Creates a pool of char strings
            that are 256 bytes long so as to take a load
            off of malloc (new) to slow down heap fragmentation
    Version 1.51.02 June 22, 2000
        Task.cpp
            Created pool for TSemaphore object to eliminate
            large number of calls to allocate from heap.
```

```
            Added operator new and operator delete functions.
            allocated memory for name object at same time.
            Created way of making blocking semaphore at the
            first time a TSemaphore object is created.
        Queue.cpp
            Made all of the derived classes compatable with the
            new way that TSemaphore is allocated, but no memory pool
            for these objects has been created (because so far, none
            of these objects has been deleted).
Version 1.51.03 August 11, 2000
        Serlprot.cpp
            Added commands for reading and writing to the START
            register in the spindle gate array.
        Spindl11.sch
            Added counter to generate a pulse at a particular sector
Version 1.51.04 September 29, 2000
        Acutrac.cpp
            Fixed bug in
                int AcutracGETPIDPARAMSTYPE(void)
                int AcutracSELECTPARAMSTYPE(int type)
                int AcutracSETPIDPARAMS(int type,long *pidparams)
                int AcutracGETPIDPARAMS(int type,long *pidparams)
            DeviceTable was being used before it was allocated.
            Functions now return an error if table not allocated.
Version 1.51.05 October 3, 2000
        Serlprot.cpp
            Removed commands for changing baud rate.  These probably
            don't work anymore anyhow, and they are not being used
            or supported.
Version 1.52.00 October 6, 2000
        i2c.cpp
            added code for the connector clamp sensors
        accutrac.cpp
            changed the HSA routines to HSA only (these used to
            support the DAFFY DUCK HGA tooling.
            removed the MOP support
            added routines for supporting the new HGA tooling
            It should be noted that the sensors for these are
            not optimal.
Version 1.60.00 October 24, 2000
        spga029: This is the new standard gate array.  It was created
        from spga023.  spga029=>xc4010d fpga.  This gate array was
        created so that the firmware would be campatable with both
        the standard gate array and the servo on position gate array
        (spga028).  Addressing was converted over to bank switching
        to make things more simple in the fpga.
        spga028: Same as spga029 except this one includes the servo
        on position hardware and is compiled into an xc4013e part.
        pid.asm:
            modifications for doing servo on position
        rtc.asm
            modifications for doing servo on position.  Made use
            of bank switching to get at position registers.
        commands added:
            Enable Servo On Position
            Clear Servo On Position register
            Set Servo On Position Scale
            Read Servo On Position Scale
Version 1.60.01 November 21, 2000
        Accutrac.cpp
            Modified SpindleMotorFaultTask(void) so that it would
            read in the current state of the amplifier fault line.

Version 1.60.10 Dec 1, 2000
        This is the first release where the PES stuff works.
Version 1.60.20 Dec 19, 2000
```

```
        main.cpp
            This version has the priority of the RunProtocol routine
            changed from 13 to 15
    Version 1.60.21 Jan 23, 2001
        pid.asm
            SetGoal
                fixed problem, would do unknown things if spindle
                axis goal was set
    Version 1.60.22 Jan 26, 2001
        s_command.c
            added code for writing data to memory command for DSP
        servo.cpp
            added function for writing data to memory in DSP
        serlprot.cpp
            added code for writing data to memory command for DSP
    Version 1.61.00 Jan 29, 2001
        Start revision to move Home and Jog paramters from high speed
        memory, to standard memory in DSP to make room for more PID
        paramters.  This will require modifications in the way
        paramters are saved.
        spid.asm
            moved home and jog paramters out of high speed ram
        servo.cpp
            added new command to get home/jog params
            modified save settings task to get pid and home/jog
            pagams and save them to ramdisk
        s_command.c
            added new command to send home/jog params
    Version 1.62.00
        Start revision to add second notch filter to PID
            Acutrac.cpp
                In stop sequences, remove the very last Wait until
                continue button pushed commands.
            Servo.cpp
                Modified save and restore param functions for new data
            scommand.c
                added new functions for extra notch filters
            protocol.cpp
                added functions for getting feature set
            Acutrac.cpp
                In HSAReset and HGAReset:
                    -fixed problem in setting spindle direction if
                    the spindle was spinning durring init process
                    -fixed problem where linear stage sometimes does
                    not retract depending on the state of the stages
                    -made display read initializing.
            i2c.cpp & acutrac.cpp
                StopButton
                HSAStart
                HGAStart
                    when the stop button is pressed while the linear
                    stage is moving, the stage will return back to home
    Version 1.62.01
            S_Command.C
                added command for reading back the full amplitude register
                from the spectrum analyzer (GET_MAGNITUDE1)
            Servo.cpp
                added command for GetMagnitude1
            protocol.cpp
                added command for GetMagnitude1
    Version 1.62.02
            Cleaned up various files (eliminated unused functions)
    Version 1.62.03 March 27, 2001
            Fixed bug in notch filters in files:
    Version 1.62.04 April 9, 2001
            added code for connector clamp FEATURES
```

```
Version 1.62.05 April 11, 2001
            fixed bug in serlprot.cpp, function TelCommandParse
            if there was an invalid command, it did not send
            back the appropriate status (in fact, no status, ooppss)
Version 1.62.06 April 18, 2001
            Serlprot.cpp
                Added commands for logging and getting logging information
                on serial command activity.
                Added logflag to SendProtocol
                Added logflag pointer to TelParseProtocol
Version 1.62.10 May 1, 2001
            RS232DEV.cpp
                Fixed Problem with Semaphores.
                1. GetC and HandleGetInterrupt, Semaphore now counts
                number of characters availiable.
                2. PutC and HandlePutInterrupt, Semaphore now counts
                number of characters to transmit.
                3. Write, blocking semaphore would not return a proper
                error if the task was killed
            Task.cpp
                TSemaphore::Timeout
                    when a timeout is posted, the event counter should
                    be incremented
            Serlprot.cpp
                Logging function could cause crash when battery dies.
                In TelReceiveProtocol, logging index is check to make
                sure it is within range before it is used.
            Accutrac.cpp
                Calls to ActiveTasks->Insert() needed to be protected
                with a critical section in frunctions
                AcutracRESET
                AcutracSTART
                AcutracSTOP
                AcutracLIFTHEADS
                AcutracRELOADHEADS
            Spindle.cpp
                Calls to ActiveTasks->Insert() needed to be protected
                with critical sections in
                DoMotorInit
Version 1.62.20 May 14, 2001
            Removed Debug code from:
                Accutrac.cpp
                I2c.cpp
                Verified that changes made for version 1.62.10
                apparently fixed the problems.
Version 1.62.21 May 22, 2001
            Serlprot.cpp
                Commands for getting the feature set were commented
                out for debug purposes, these commands are now
                restored (opps!).
Version 1.63.00 June 4, 2001
            pid.asm
                Added code to PID loop to enable doing open loop
                frequency responses
            scommand.c
                added or modified commands for doing open loop
                frequency responses
                In Function RB_config:
                    Zero out MiscParams[i].Status
            servo.cpp
                added functions for doing open loop frequency responses
            serlprot.cpp
                added functions for doing open loop frequency responses
Version 1.64.00 June 27,2001
            accutrac.cpp
            accutrac.h
```

```
                Added new data member to AcutracDeviceTable
                    UnloadRadius
                    LoadRadius
                commented out AcutracSAVEascii
            scfig.syn
                added new keywords to afg file parser
                    LOADRAD
                    UNLOADRAD
                commented out SaveConfigFile
Version 1.64.10 July 10, 2001
            task.cpp
            swap.s
                Added code to record total execution time for
                each task.
            spindl12
                created spindl12 fpga to add hardware to record
                total execution time
Version 1.64.11 July 13, 2001
            acutrac.cpp
                In function HSAStart, Step 12, there was in =
                instead of an == operator in the if statement
Version 1.64.12 July 18, 2001
            many files
                cleaned up unused global variables
Version 1.65.00 July 26, 2001
            logic timeing problem in SPGA029A FPGA
            circuit changed to eliminate glitch in position clear strobe
            new version for std system is spga030
            new version for PES system not yet done
Version 1.65.10
            Minor problem fixed in SPINDL12 FPGA
                address 0xffdc0006 would not generate DTACK on read
                address 0xffdc0018 would generate erouneous DTACK
Version 1.65.20 Aug 9 2001
            MCFIG.SYN
                Made grammar more tolerent of extraneous new lines
                in the config files.
Version 1.66.00 Sept 7, 2001
            Accutrac.cpp
                HSAStop()
                    remove step where code waits for user to remove
                    comb from comb corral.
Version 1.67.00 Sept 19,2001
            Accutrac.cpp
                HSAStart()
                HSAStop()
                    Added code to load connector clamp
                SpindleMotorFaultTask(void)
                    Added paramter to BigTimeShutDown to indicate amp fault
            I2C.cpp
                BitTimeShutDown
                    Added paramter to indicate error that caused shutdown
                AirPressure
                    Added parameter in call to BigTimeShutDown to indicate Air Fault
                StopButton
                    Added parameter in call to BigTimeShutDown to indicate User Abort
            Serlprot.h
                Add Status bit to indicate when the stop button was pushed
            Serlprot.cpp
                Added command to clear stop button status
Version 1.67.10 Sept 25,2001
            SCFIG.SYN
                Problem with parser generator and compiler.  Defualt size of enum
                is 1 byte.  Parser generator was using values up to 153,
                compiler sees this as a signed char, big trouble.
                Changed OPTCC in \sds663\lib68000 as follows:
```

```
                    this was added to OPTCC
                    -s enum=2
    Version 1.67.50 Nov 5, 2001
                Serlprot.cpp
                    Added command to upload SLUT to spindle controller
                Spindle.cpp
                    Added code to upload SLUT
    Version 1.68.00 Nov 13, 2001
                Accutrac.cpp
                    added new device driver for doing resonance testing
    Version 1.69.00 Nov 20, 2001
                xilinx.c
                    New gate array for motion control.  SPGA032 has a
                    pre scale for the spindle motor sample rate so
                    that we can get the 300hz sample rate with a master
                    rotary sample rate of 40khz.
    Version 1.69.01 Nov 27, 2001
                accutrac.cpp
                    Bug in HSAStop, case 11:
                    Motor stop command was only executed for HSAR,
                    instead of only executing for HSA..fixed
    Version 1.69.02 Nov 28, 2001
                s_profil.c
                    In spindle profile function, multiplied sample rate
                    by 3 to make up for prescale circuit in new gate array
                    SPGA032
                xilinx.c
                    Fixed bug in prescale circuit.  Status would be valid
                    for 3 sample times for spindle sample rate
                accutrac.cpp
                    HSAReset
                        moved rotary actuator to center of travel after reset
                    HSAStop
                        for HSAR, move rotary to center on retract
    Version 1.69.03
                accutrac.cpp
                    HSAStop
                        moved place where rotary goes to center on retract
    Version 1.69.20
                servo.cpp
                serlprot.cpp
                s_command.c
                s_event.c
                    created new DSP command SetGoalAndMove and new
                    serial command SetGoalAndMove2
    Version 1.69.30 Jan 8, 2002
                ramdisk.cpp
                    D_open
                        fixed problem where a file name with zero lenth
                        could be opened because it found plenty of matches
                        This would cause the ramdisk to scramble it's
                        directory when a write was done.
    Version 1.69.40 Jan 11, 2002
                servo.cpp
                serlprot.cpp
                    added code to support enabling/disabling of micro E
                    autocomp mode
    Version 1.69.41 April 2,2002
                serlprot.cpp
                    fixed bug in FEATURES->SAVE
                    did not save value for connector clamp
*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
const char tel_board_name[] = "SOA 600-160";
const char tel_board_rev[] = "REV %c";
const char tel_version[] = "1.69.41";
const char tel_date[] = __DATE__" "__TIME__;
const char tel_copyright[] = "Copyright (c) 1997/2002 Teletrac Inc";

#pragma region("ram=sernum")    /*  global defs after this will live in non volatile mem    */

static char tel_SerialNumber[34];
static unsigned tel_SerialChecksum;

static unsigned CalcChecksum(void)
{
    char *s = tel_SerialNumber;
    unsigned a=0;

    while(*s)
    {
        a += ((unsigned)*s++);
    }
    return a;
}

void SetSerialNumber(char *s)
{
    int l;

    strncpy(tel_SerialNumber,s,32);     /*  only copy first 32 characters   */
    l = strlen(s);
    if(l > 32)
        l=32;
    tel_SerialNumber[l] = 0;            /*  null terminate string   */
    tel_SerialChecksum = CalcChecksum();    /*  recalc checksum */
}

char *GetSerialNumber(int *c,char *b)   /*  copy checksum into buffer   */
{
    /*
    ** return 1 if checksum is not valid
    */
    int l;

    *c += (l = strlen(tel_SerialNumber) + 1);
    strcpy(b,tel_SerialNumber);
    return(b + l);
}

int CheckSerialNumber(void)
{
    return (tel_SerialChecksum - CalcChecksum());
}
```

```
#ifndef SPINCOPR__H
#define SPINCOPR__H

extern const char tel_board_name[];
extern const char tel_board_rev[];
extern const char tel_version[];
extern const char tel_date[];
extern const char tel_copyright[];

#ifdef __cplusplus
extern "C" {
#endif

extern void SetSerialNumber(char *s);
extern char *GetSerialNumber(int *c,char *b);
extern int CheckSerialNumber(void);

#ifdef __cplusplus
}
#endif

#endif
```

```
#include "stack.h"

//-----------------------------------
// A simple stack class
//-----------------------------------


Stack::Stack(int size)
{
    stack = new unsigned  char[size];
    stacksize = size;
    stackpointer = 0;
}

Stack::~Stack()
{
    delete[] stack;
}

void Stack::Push(int d)
{
    if(stackpointer < stacksize)
        stack[stackpointer++] = d;
}

int Stack::Pop(void)
{
    if(stackpointer > 0)
        return (int)stack[--stackpointer];
    else
        return -1;
}
```

```
//---------------------------------------------------------------------
// A simple Stack Class
//---------------------------------------------------------------------

#ifndef STACK__H
#define STACK__H

class Stack {
    unsigned char *stack;
    int stackpointer;
    int stacksize;
public:
    Stack(int size);
    ~Stack();
    void Push(int d);
    int Pop(void);
};

#endif // STACK__H
```

```
;*****************************************************************
;
; Startup code for Spindle Controller
; Initialize memory and call main.
; Contains vectors for interrupts
;
;*****************************************************************
;
    XDEF    START,__brkp,__brksz,__exit
    XREF    STKTOP,DATA,_main
    XREF    MALLOC_SIZE,MALLOC_START

;*****************************************************************
;   If C++ style startup code is not desired, then
;   define CPLUSPLUS to 0
;*****************************************************************
;
CPLUSPLUS   = 1

;*****************************************************************
;   Define vector area
;*****************************************************************
;
    SECTION vectors
__vectors   DS.L    256 ;256 vectors at start of ram

;*****************************************************************
;   Define variables to track memory allocations for mbrk().
;*****************************************************************
;
    SECTION data
__brkp  DC.L    MALLOC_START            ; point to available memory
__brksz DC.L    MALLOC_SIZE         ; number of available bytes

;*************************************************************
;   The reset vector will point to this code (START).
;   By the time we get here, A7 will have been initialized.
;*************************************************************
;
    SECTION code
    .STK    "none"          ; terminate call chain for -OG
START   MOVE.L  #STKTOP,A7      ; set the stack pointer
    MOVE.L  #0,A6           ; end of stack frame chain

    ;*********************************************************
    ;       Zero out uninitialized RAM.
    ;*********************************************************
    MOVE.L  #`BASE(ram),A1      ; A1 = base of region ram
    MOVE.L  #`SIZE(ram),D0      ; D0 = size of region ram
    LSR.L   #2,D0           ; compute size in longs
    BRA ZDBF            ; enter a fast loop
  ZLP:  CLR.L   (A1)+           ; clear four bytes at a time
  ZDBF: DBF D0,ZLP          ; up to 256K in inner loop
    SUB.L   #$10000,D0      ; rest in outer loop
    BHS ZLP

    ;*********************************************************
    ;       Initialize other RAM from ROM.
    ;*********************************************************
    MOVE.L  #DATA,A0        ; A0 = ROM base of region data
    MOVE.L  #`BASE(data),A1     ; A1 = RAM base of region data
    MOVE.L  #`SIZE(data),D0     ; D0 = size of region data
    LSR.L   #2,D0           ; compute size in longs
    BRA IDBF            ; enter a fast loop
  ILP:  MOVE.L  (A0)+,(A1)+     ; move four bytes at a time
  IDBF: DBF D0,ILP          ; up to 256K in inner loop
    SUB.L   #$10000,D0      ; rest in outer loop
    BHS ILP
    ;
    ;*********************************************************
    ;       Zero out Heap
```

```
    ;*******************************************************
    move.l  __brkp,a1       ;a1 = base of heap
    move.l  __brksz,d0      ;d0 = size of heap
    lsr.l   #2,d0           ;compute size in longs
    move.l  #$deadbeaf,d1   ;load in data
    bra zhpdbf
zhp:
    move.l  d1,(a1)+            ;clear four bytes at a time
zhpdbf:
    dbf d0,zhp              ;up to 256k in inner loop
    sub.l   #$10000,d0      ;rest in outer loop
    bhs zhp

    ;
    ;*******************************************************
    ;       Initialize RAM vectors from ROM
    ;*******************************************************
    ;
;   XREF    EXCEPTION_VECTORS
    MOVE.L  #exception_vecs,A0  ;point A0 to ROM image of exception vector data
    MOVE.L  #`BASE(vectors),A1  ;point A1 to RAM base of region vectors
    MOVE.L  #`SIZE(vectors),D0  ;set D0 to size of region vectors
    LSR.L   #2,D0           ;compute the size in longs
    BRA EDBF            ;enter a fast loop
ELP:    MOVE.L  (A0)+,(A1)+     ;move four bytes at a time
EDBF:   DBF D0,ELP
    ;
    ;*******************************************************
    ;       Initialize memory allocator.
    ;*******************************************************
;    MOVE.L #`BASE(malloc),D0   ; address of malloc region
;   MOVE.L  #`SIZE(malloc),D1   ; size of malloc region
;    .IF "ptrd"?"2"
;   SWAP    D0          ; accommodate 2-byte C "pointers"
;    .ENDIF
;    .IF "long"?"2"
;   SWAP    D1          ; accommodate 2-byte C "longs"
;    .ENDIF
;   MOVE.L  D0,__brkp       ; variables referenced by mbrk()
;   MOVE.L  D1,__brksz
;
 .IF CPLUSPLUS
    ;*******************************************************
    ;       Call any C++ initializer thunks.
    ;*******************************************************
    MOVE.L  #`BASE(init),A5     ; A5 = base of region init
    MOVE.L  #`SIZE(init),D7     ; D7 = size of region init
    .IF "ptrf"?"2"
    LSR.L   #1,D7           ; number of initializer thunks
    .ELSE
    LSR.L   #2,D7           ; number of initializer thunks
    .ENDIF
    BRA NDBF            ; enter loop
  NLP:
    .IF "ptrf"?"2"
    MOVE.W  (A5)+,A0        ; get address of thunk
    .ELSE
    MOVE.L  (A5)+,A0        ; get address of thunk
    .ENDIF
    CMP.W   #0x4E71,A0      ; NOP may pad region
    BEQ NDBF
    JSR (A0)            ; call initializer thunk
  NDBF: DBF D7,NLP          ; up to 256K in inner loop
    SUB.L   #$10000,D7      ; rest in outer loop
    BHS NLP
 .ENDIF
```

```
    ;*************************************************************
    ;       Invoke main() with no arguments.
    ;*************************************************************
    JSR _main           ; any return value is "int" in D0
    ;*************************************************************
    ;       Call any C++ exit thunks.
    ;*************************************************************
__exit
 .IF CPLUSPLUS
    MOVE.L  #`BASE(exit),A5     ; A5 = base of region exit
    MOVE.L  #`SIZE(exit),D7     ; D7 = size of region exit
    ADD.L   D7,A5           ; reverse order
    .IF "ptrf"?"2"
    LSR.L   #1,D7           ; number of exit thunks
    .ELSE
    LSR.L   #2,D7           ; number of exit thunks
    .ENDIF
    BRA XDBF            ; enter loop
  XLP:
    .IF "ptrf"?"2"
    MOVE.W  -(A5),A0        ; get address of thunk
    .ELSE
    MOVE.L  -(A5),A0        ; get address of thunk
    .ENDIF
    CMP.W   #0x4E71,A0      ; NOP may pad region
    BEQ XDBF
    JSR (A0)            ; call exit thunk
  XDBF: DBF D7,XLP          ; up to 256K in inner loop
    SUB.L   #$10000,D7      ; rest in outer loop
    BHS XLP
 .ENDIF

  DONE: BRA DONE            ; infinite loop if main ever returns

;*************************************************************************
;   OTHER EXCEPTION VECTORS: to supervisor data space at address 8,
;   or 8 bytes beyond where the vector base register will point.
;   This table is commented out because no actual interrupt routines
;   are provided.
;*************************************************************************
    SECTION vects
    XREF buss_error,address_error
    XREF illegal,zero_div,chk_instr,trapv_instr,privilege
    XREF trace,line_a,line_f,reserved,unitialized
    XREF spurious,autovector,trap_instr,user_vecs
    XREF _rs232_irq,Swap
    XREF SpinIrq0,SpinIrq1,SpinIrq2,SpinIrq3,SpinIrq4,SpinUp,SpinDown
    XREF SpinTicker
    XREF DSPirq,I2CIrq
    XREF Trap1

exception_vecs:
    DC.L    STKTOP          ; initial stack pointer
    DC.L    START           ; initial execution address
    DC.L    buss_error,address_error        ; 0x08
    DC.L    illegal,zero_div,chk_instr,trapv_instr  ; 0x10
    DC.L    privilege,trace,line_a,line_f       ; 0x20
    DC.L    reserved,reserved,reserved,unitialized  ; 0x30
    DCB.L   8,reserved              ; 0x40
    DC.L    spurious,autovector,autovector  ; 0x60
    DC.L    autovector
    DCB.L   4,autovector                ; 0x70
    DC.L    Swap                        ; 0x80
    DC.L    Trap1                       ; 0x84
    DCB.L   14,trap_instr               ; 0x88
    DCB.L   16,reserved                 ; 0xc0
```

```
    DC.L    _rs232_irq              ; 0x100
    DCB.L   127,user_vecs           ; 0x104
    DC.L    SpinIrq0                ; 0x300
    DC.L    SpinIrq1                ; 0x304
    DC.L    SpinIrq2                ; 0x308
    DC.L    SpinIrq3                ; 0x30c
    DC.L    SpinIrq4                ; 0x310
    DC.L    SpinTicker              ; 0x314
    DC.L    SpinUp                  ; 0x318
    DC.L    SpinDown                ; 0x31c
    DCB.L   8,user_vecs             ; 0x320
    DC.L    DSPirq                  ; 0x340
    DC.L    I2CIrq                  ; 0x344
    DCB.L   46,user_vecs            ; 0x344
```

```
//*********************************************************
// implementation of string class
//*********************************************************

#include <stdio.h>
#include "strings.h"
#include "task.h"
#include "queue.h"

static Wait *blocker=0;

String *String::Master;     //master link to string pool

String::String()
{
}

String::~String()
{
}

void *String::operator new(size_t size) //new operator for this class
{
    char *p;

    if(!blocker)    //no blocker has been defined
    {
        blocker = new Wait(1,"NewStringsBlocker");
    }
    blocker->Pend();
    if(!Master)     //nothing in pool, create new object
    {
        p = new char[size + 256];
        ((String *)p)->s = p + size;
    }
    else
    {

        p = (char *)Master;
        Master = Master->next;  //unlink object
    }
    blocker->Post();
    return (void *)p;
}

void String::operator delete(void *p)       //delete operator for this class
{
    int sr;

    blocker->Pend();
    ((String *)p)->next = Master;
    Master = (String *)p;
    blocker->Post();
}

char *String::Get(void)     //get pointer to character data
{
    return s;
}
```

```
//*********************************************************
//
// Header file for string class manager
//
// the purpose of this class is to aid in preventing
// the fragmentation of heap memory
// this function will allocate a large pool of memory for
// use as strings.  Most of the strings that are used are
// 256 bytes or less, so just allocate a pool of 256 byte
// long strings.
//
//*********************************************************

#ifndef STRINGS__H
#define STRINGS__H


class String{
    char *s;
    String *next;   //used to make a linked list of string objects
    static String *Master;  //head pointer to linked list
public:
    String();
    ~String();
    void *operator new(size_t size);    //new operator for this class
    void operator delete(void *p);      //delete operator for this class
    char *Get(void);        //get pointer to character data
};

#endif
```

```c
/*
** This is the timer routines
** For Spindle Controller board 2-17-95
*/
#include <stdio.h>
#include "timer.h"
#include "rs232.h"

int volatile ticks;

extern char ImrShadow;

long volatile time_ms;

void the_timer()
{
    static int flag = 0;
    /*
    ** this routine gets called from the interrupt service routine
    */
    --ticks;    /*  decrement global ticker */
    time_ms++;  /*  increment time  */
    /*
    ** toggle the watchdog line
    */

    if(flag)
    {
        flag = 0;
        *((volatile char *)(BIT_RESET_COMMAND)) = 0x40;
    }
    else
    {
        flag = 1;
        *((volatile char *)(BIT_SET_COMMAND)) = 0x40;
    }
}

void init_timer()
{
    register int dummy;

    *((volatile char *)(COUNTER_UPPER_REG)) = 4;
    *((volatile char *)(COUNTER_LOWER_REG)) = 128;  /*  10 mSEC timer default   */
    acr_shad |= 0x70;
    *((volatile char *)(AUX_CONTROL_REG)) = acr_shad;
    ImrShadow |= 0x08;  /*  enable bit for timer and input port change  */
    *((volatile char *)(INTERRUPT_MASK_REG)) = ImrShadow;   /*  enable timer    */
    dummy = *((volatile char *)(START_COUNTER));
    dummy = *((volatile char *)(INPUT_PORT_CHANGE_REG));
}

void port_change(void)
{
    //-------------------------------------------------
    // this function is called by the 68681 isr routine
    //-------------------------------------------------

    char in_port_change_reg;

    in_port_change_reg = *((char *)(INPUT_PORT_CHANGE_REG));
}
```

```c
/*
** header file for timer
    Teletrac Spindle Controller
*/
extern int volatile ticks;
extern long volatile time_ms;   /* scalar time  */

#ifdef __cplusplus
extern "C" {
#endif

void the_timer(void);
void init_timer(void);

#ifdef __cplusplus
}
#endif
```

```
/*
** this is the main routine for UPLOAD
*/
#include <stdio.h>
#include <stdlib.h>
#include "host.h"
#include <curses.h>
#include "upload.h"
#include "frontled.h"
#include "timer.h"

#include "dspdata.c"
#include "dspcode.c"

static int download(const unsigned char *mem,size_t max);
void DSPreset(void);

extern "C" {

int DSPupload(void)
{

    int retrys = 0;
    int loop = 1;
    /*
    ** download data
    */
    do
    {
        DSPreset();
        if(download(DATAdspcode,sizeof(DATAdspcode) ) )
        {
            ++retrys;   //try again
            continue ;
        }
        if(download(DATAdspdata,sizeof(DATAdspdata) ) )
        {
            ++retrys;
            continue ;
        }
        else
        {
            loop = 0;   //stop looping , all done
        }
    }while(loop && (retrys < 10) ); //only try 10 times
    if(retrys == 10)
        return(1);
    return(0);
}

void DSPreset(void)
{
    PORT_C = 0;                 //put DSP into RESET state
    ticks = 2;
    while(ticks > 0);           //wait just a bit (20mS)
    PORT_C = 0x08;  //take DSP out of RESET state
}

}

static int download(const unsigned char *mem,size_t size)
{
    /*
    ** move the data to the host port
    */
    extern volatile int ticks;
```

```
    long i;     /*  index into memory   */
    for(i=0;i<size;)
    {
        ticks = 500;        /*  one second timeout  */
        while( ! ( *((volatile  char *)DSP_ISR) & ISR_TXDE) && ticks);
        if(ticks <= 0)
        {
            return(1);      //timeout error
        }
        *((char *)DSP_DATAH) = mem[i+2];
        *((char *)DSP_DATAM) = mem[i+1];
        *((char *)DSP_DATAL) = mem[i];
        i+=3;
    }
    ticks = 2;
    while(ticks > 0);
    *((char *)DSP_IRQCTL) = *((char *)DSP_IRQCTL) | ICR_HF0;
    ticks = 2;
    while(ticks > 0);
    *((char *)DSP_IRQCTL) = *((char *)DSP_IRQCTL) & ~ICR_HF0;
    return(0);
}
```

```
//
//header file for upload.cpp and srec.cpp

#ifndef UPLOAD__H
#define UPLOAD__H

#ifdef __cplusplus
extern "C" {
#endif
extern int DSPupload(void);
extern void DSPreset(void);

#ifdef __cplusplus
}
#endif

#endif
```

```
/*************************************************************************
**
** Xilinx upload interface for Spindle Controller Board
**
** Author:Jim Patchell
** Date  :2-25-95
**
*************************************************************************/
#include <stdio.h>

#define XILINX_BASE     0xffd80001      /*  base address of xilinx ifc  */

#define XILINX_DATA_OUT XILINX_BASE + 0
#define XILINX_STROBE   XILINX_BASE + 2
#define XILINX_DONE     XILINX_BASE + 4
#define XILINX_INIT     XILINX_BASE + 6
#define XILINX_PROGRAM  XILINX_BASE + 8
#define XILINX_DONE_TS  XILINX_BASE + 10
#define XILINX_INIT_TS  XILINX_BASE + 12
#define XILINX_RESET    XILINX_BASE + 14

#include "spinpga.c"

int LoadXilinx(const unsigned char *d)
{
    int data,j;
    int loop = 1;
    int error=0;

    *((volatile char *)XILINX_RESET) = 1;   /*  reset interface */
    *((volatile char *)XILINX_PROGRAM) = 1; /*  raise program bit   */
    while(! (*((volatile char *)XILINX_INIT) & 0x01) ); /*  wait for init to be high    */
    do
    {
        if(! (*((volatile char *)XILINX_INIT) & 0x01) )
        {
            loop = 0;   /*  stop looping this is an error   */
            error = 1;
        }
        else if(*((volatile char *)XILINX_DONE) & 0x01)
            loop = 0;   /*  stop looping, load is done      */
        else
        {
            data = *d++;    /*  get data    */
            for(j=0;j<8;++j)    /*  shift out data  */
            {
                if(data & 0x01)
                    *((volatile char *)XILINX_DATA_OUT) = 1;
                else
                    *((volatile char *)XILINX_DATA_OUT) = 0;
                *((volatile char *)XILINX_STROBE) = 1;  /*  STROBE  */
                *((volatile char *)XILINX_STROBE) = 0;
                data >>= 1; /* shift data right */
            }
        }
    }while(loop);
    *((volatile char *)XILINX_STROBE) = 1;  /*  STROBE  */
    *((volatile char *)XILINX_STROBE) = 0;
    *((volatile char *)XILINX_STROBE) = 1;  /*  STROBE  */
    *((volatile char *)XILINX_STROBE) = 0;
    return(error);
}
```

```c
/*
** header file for the xilinx upload routines
*/

#ifndef XILINX__H
#define XILINX__H

extern const unsigned char DATAspindle[];


#ifdef __cplusplus
extern "C" {
#endif

int LoadXilinx(const unsigned char *d);

#ifdef __cplusplus
}
#endif

#endif
```

```
[
    disregard white space
    ~case sensitive
    lexeme {eol,longint}
    line numbers
    parser file name = "motcfig.cpp"
    header file name = "motcfig.h"
]

eof = -69
letter = 'a-z'+'A-Z'+'_'+'.'
digit = '0-9'+'.'
intdigit = '0-9'
noteol = ~(eof + '\n')

eol -> [';',noteol?...],'\n'

(void)white space
    -> ' ' + '\t' + '\r'
    -> "/*", ~eof?...,"*/"

(void)mcfig $
    -> [mc]...,eof

(void)mc
    -> "MOTOR",eol, '{',eol,[motor config,eol]...,'}',eol
    -> "PLL",eol,'{',eol,pll config lines,'}',eol
    -> eol

motor config
    ->"ENCODERPITCH",'=',longint:v  =mt->SinLUTSize=v;
    ->"MOTORPOLES",'=',longint:v    =mt->MotorPoles=v;
    ->"MOTORPHASE",'=',longint:v    ={mt->p1 = (int)v;
                                        mt->p2 = (int)v + mt->SinLUTSize / 3;
                                    }
    ->"ACCEL",'=',longint:v         =mt->accel = v;
    ->"DECEL",'=',longint:v         =mt->decel = v;
    ->"CLAMPHI",'=',longint:v       =mt->upperlimit = (int) v;
    ->"CLAMPLO",'=',longint:v       =mt->lowerlimit = (int) v;
    ->

pll config lines
    -> pll config lines, pll config
    -> pll config

pll config
    ->'[',longint:i,']',longint:g,',',longint:c,eol ={lc[i].Gain = g;
                                                      lc[i].Rolloff = c;
                                                      }
    -> eol

(long)longint
    ->simple longint        ={StringAccumulator[StringAccumLength]='\0';return atol(StringAccumu
lator);}
    ->'-',simple longint    ={StringAccumulator[StringAccumLength]='\0';return -atol(StringAccum
ulator);}

(void)simple longint
    ->intdigit:c                    =ins(c);
    ->simple longint,intdigit:c     =pcn(c);

{
    //embeded c code

#include "task.h"
#include "acutrac.h"
```

```
#include "cio.h"
#include "ramdisk.h"
#include "spindle.h"
#include "strings.h"

#ifdef toupper
#undef toupper
#endif

#define GET_INPUT ((PCB).input_code = Getc(InFile))

#define SYNTAX_ERROR sprintf(error,"%s, line %d, column %d\n", \
  (PCB).error_message, (PCB).line, (PCB).column)

#define PARSER_STACK_OVERFLOW {sprintf(error, \
   "\nParser stack overflow, line %d, column %d\n",\
   (PCB).line, (PCB).column);}

#define REDUCTION_TOKEN_ERROR {sprintf(error, \
    "\nReduction token error, line %d, column %d\n", \
    (PCB).line, (PCB).column);}

static int InFile;
static char StringAccumulator[80];
static char StringAccumLength=0;
static char error[80];
static loopcomp *lc;
static SPINDLE_PARAMS *mt;

static void ins(int c)
{
    //init and ut char to name
    StringAccumulator[0] = c;
    StringAccumLength=1;
}

static void pcn(int c)
{
    StringAccumulator[StringAccumLength++]=c;
}

int LoadMotorConfigFile(char *name,SPINDLE_PARAMS *m,loopcomp *l)
{
    int retval = 0;
    mt = m;
    lc = l;
    if((InFile = Open(name,READ_ONLY)) >= 0)
    {
        mcfig();
        if((PCB).exit_flag != AG_SUCCESS_CODE)
            retval = 1;
        Close(InFile);
    }
    else
    {
        sprintf(error,"Could not open %s",name);
        retval = 1;
    }
    return retval;
}

char *GetMotorConfigFileErrorString(void)
{
    return error;
}
```

```
int SaveMotorConfigFile(char *name,SPINDLE_PARAMS *mt,loopcomp *lc)
{
    int retval = 0;
    int handle;
    String *S = new String;
    char *s = S->Get();
    int c;
    int i;

    if((handle = Open(name,WRITE_ONLY)) >= 0)
    {
        c = sprintf(s,"MOTOR\r\n{\r\n");
        Write(handle,s,long(c));
        c = sprintf(s,"ENCODERPITCH=%ld\r\n",mt->SinLUTSize);
        Write(handle,s,long(c));
        c = sprintf(s,"MOTORPOLES=%d\r\n",mt->MotorPoles);
        Write(handle,s,long(c));
        c = sprintf(s,"MOTORPHASE=%d\r\n",mt->p1);
        Write(handle,s,long(c));
        c = sprintf(s,"ACCEL=%d\r\n",mt->accel);
        Write(handle,s,long(c));
        c = sprintf(s,"DECEL=%d\r\n",mt->decel);
        Write(handle,s,long(c));
        c = sprintf(s,"CLAMPHI=%d\r\n",mt->upperlimit);
        Write(handle,s,long(c));
        c = sprintf(s,"CLAMPLO=%d\r\n}\r\n",mt->lowerlimit);
        Write(handle,s,long(c));
        c = sprintf(s,"PLL\r\n{\r\n");
        Write(handle,s,long(c));
        for(i=0;i<10;++i)
        {
            c=sprintf(s,"[%d]%d,%d\r\n",i,lc[i].Gain,lc[i].Rolloff);
            Write(handle,s,long(c));
        }
        c = sprintf(s,"}\r\n");
        Write(handle,s,long(c));
        Close(handle);
    }
    else
    {
        sprintf(error,"Could not open %s for output",name);
        retval = 1;
    }
    delete S;
    return retval;
}

void GetLoadMotorConfigError(char *s)
{
    memcpy(s,error,80);
}

}   //end of embeded C
```

```
//---------------------------------------------------------------------
//
// Header file for motor configuration
//
//---------------------------------------------------------------------

#ifndef MCFIG__H
#define MCFIG__H

extern int LoadMotorConfigFile(char *name,SPINDLE_PARAMS *m,loopcomp *l);
extern char *GetMotorConfigFileErrorString(void);
extern int SaveMotorConfigFile(char *name,SPINDLE_PARAMS *mt,loopcomp *lc);
extern void GetLoadMotorConfigError(char *s);

#endif
```

```
/*************************************************************************
** Parse Configuration files for spindle controller
*************************************************************************/

[
    disregard white space
    ~case sensitive
    lexeme {eol,longint,real,string}
    line numbers
    parser file name = "spincfig.cpp"
    header file name = "spincfig.h"
]

eof = -69
letter = 'a-z'+'A-Z'+'_'+'.'
digit = '0-9'+'.'
intdigit = '0-9'
noteol = ~(eof + '\n')

eol -> [';',noteol?...],'\n'

(void)white space
    -> ' ' + '\t' + '\r'
    -> "/*", ~eof?...,"*/"

(void)scfig $
    -> "ASCII",eol,[configitems,eol]...,eof

(void) configitems
    ->"MAXRPM",'=',longint:v        =dt->maxrpm = (int)v;
    ->"MINRPM",'=',longint:v        =dt->minrpm = (int)v;
    ->"TESTRPM",'=',longint:v       =dt->testrpm = (int)v;
    ->"UNLOADRPM",'=',longint:v     =dt->unloadrpm = (int)v;
    ->"LOADRPM",'=',longint:v       =dt->loadrpm = (int)v;
    ->"ROTATION",'=',longint:v      =dt->rotation = (int)v;
    ->"INITTRACK",'=',longint:v     =dt->inittrack = v;
    ->"MAXTRACK",'=',longint:v      =dt->maxtrack = v;
    ->"INSIDERADIUS",'=',real:v     =dt->insideradius = v;
    ->"OUTSIDERADIUS",'=',real:v    =dt->outsideradius = v;
    ->"PIVOTTOGAP",'=',real:v       =dt->pivottogap = v;
    ->"PIVOTTOCENTER",'=',real:v    =dt->pivottocenter = v;
    ->"ACTUATORRADIUS",'=',real:v   =dt->actuatorradius = v;
    ->"REFERENCEANGLE",'=',real:v   =dt->referenceangle = v;
    ->"LOADUNLOADRADIUS",'=',real:v ={
                                        dt->LoadRadius = v;
                                        dt->UnloadRadius = v;
                                    }
    ->"LOADRAD",'=',real:v          =dt->LoadRadius = v;
    ->"UNLOADRAD",'=',real:v        =dt->UnloadRadius = v;
    ->"PRELOADRAD",'=',real:v   =dt->preloadunloadradius = v;
    ->"XDISTANCEHOME",'=',real:v    =dt->xdistancehome = v;
    ->"XDISTANCERUN",'=',real:v     =dt->xdistancerun = v;
    ->"MAXHEAD",'=',longint:v       =dt->maxhead = (int)v;
    ->"DUTORIENTATION",'=',orient:v=dt->dutorientation = (int)v;
    ->"POSPARAMS",'=',string:s      ={strcpy(dt->ServoParams,s);}
    ->"MODE",'=',modetype:t         =dt->mode = t;
    ->"PITCHMODE",'=',pitchmode:t   =dt->TrackPitchMode=t;
    ->"ACCELERATION",'=',longint:v  ={
                                        Xio(SPIN_SETACCEL,sys.HSpindle,(char *)0,(char *)0,0l,(i
nt)v);
                                        Xio(SPIN_SETDECEL,sys.HSpindle,(char *)0,(char *)0,0l,(i
nt)v);
                                    }
    ->"DECELERATION",'=',longint:v  =Xio(SPIN_SETDECEL,sys.HSpindle,(char *)0,(char *)0,0l,(int)
v);
    ->"ROTARY",'{',eol,rotary pid terms...,'}'
```

```
    ->"CONCLAMP",'=',longint:v      =dt->ConnectorClamp = (int)v;
    ->"CONCLMPTO",'=',longint:v     =dt->ConnectorClampTimeout = (int)v;
    ->"OVERSHOOT",'=',longint:v     =dt->Overshoot = v;
    ->                              //blank line

rotary pid terms
    ->"SET",'[',settype:s,']','{',eol,pid set...,'}',eol ={
        pp[s]->Kp = Kp;
        pp[s]->Kv = Kv;
        pp[s]->Ki = Ki;
        pp[s]->NotchF = NotchF;
        pp[s]->NotchQ = NotchQ;
        pp[s]->NotchD = NotchD;
        pp[s]->NotchEnableFlag = NotchEn;
}


pid set
    ->"KP",'=',longint:v,eol    ={Kp =v;}
    ->"KV",'=',longint:v,eol    ={Kv =v;}
    ->"KI",'=',longint:v,eol    ={Ki = v;}
    ->"NOTCHF",'=',longint:v,eol    ={NotchF = v;}
    ->"NOTCHQ",'=',longint:v,eol    ={NotchQ = v;}
    ->"NOTCHD",'=',longint:v,eol    ={NotchD = v;}
    ->"NOTCHEN",'=',longint:v,eol   ={NotchEn = v;}

(int) settype
    ->"HEADSON" =0;
    ->"HEADSOFF" = 1;

(int) pitchmode
    ->"RADIUSCONSTANT"  =0;
    ->"ANGLECONSTANT"       =1;

(int) modetype
    ->"HSA"                 =HSA;
    ->"HGA"                 =HGA;
    ->"HSAR"                =HSAR;

(int) orient
    ->"FRONT"                   =1;
    ->"BACK"                    =-1;
    ->"1"                       =1;
    ->"-1"                      =-1;

(double) real
    ->simple real           ={StringAccumulator[StringAccumLength]='\0';return atof(StringAccumu
lator);}
    ->'-',simple real       ={StringAccumulator[StringAccumLength]='\0';return -atof(StringAccum
ulator);}

(void) simple real
    ->digit:c               =ins(c);
    ->simple real,digit:c   =pcn(c);

(long)longint
    ->simple longint        ={StringAccumulator[StringAccumLength]='\0';return atol(StringAccumu
lator);}
    ->'-',simple longint    ={StringAccumulator[StringAccumLength]='\0';return -atol(StringAccum
ulator);}

(void)simple longint
    ->intdigit:c                    =ins(c);
    ->simple longint,intdigit:c     =pcn(c);

(char *)string
```

```
    ->namestring        ={StringAccumulator[StringAccumLength]='\0';return StringAccumulator;}

(void)namestring
    ->letter:c          =ins(c);
    ->string,letter:c   =pcn(c);

{
    //embeded c code
#include <math.h>
#include "task.h"
#include "acutrac.h"
#include "cio.h"
#include "ramdisk.h"
#include "global.h"
#include "spindle.h"
#include "serlprot.h"   //for SYSTEM sys variable
#include "strings.h"

#ifdef toupper
#undef toupper
#endif

#define GET_INPUT ((PCB).input_code = Getc(InFile))

#define SYNTAX_ERROR sprintf(error,"%s, line %d, column %d\n", \
  (PCB).error_message, (PCB).line, (PCB).column)

#define PARSER_STACK_OVERFLOW {sprintf(error, \
   "\nParser stack overflow, line %d, column %d\n",\
   (PCB).line, (PCB).column);}

#define REDUCTION_TOKEN_ERROR {sprintf(error, \
    "\nReduction token error, line %d, column %d\n", \
    (PCB).line, (PCB).column);}

static int InFile;
static char StringAccumulator[80];
static char StringAccumLength=0;
static AcutracDeviceTable *dt;
static char error[80];
static long Kp,Kv,Ki,NotchF,NotchQ,NotchD,NotchEn;
static PidParams *pp[2];

static void ins(int c)
{
    //init and ut char to name
    StringAccumulator[0] = c;
    StringAccumLength=1;
}

static void pcn(int c)
{
    StringAccumulator[StringAccumLength++]=c;
}

int LoadConfigFile(char *name,AcutracDeviceTable *t)
{
    int retval = 0;
    dt = t;
    pp[0] = dt->Pid[0];
    pp[1] = dt->Pid[1];
    if((InFile = Open(name,READ_ONLY)) >= 0)
    {
        scfig();
        if((PCB).exit_flag != AG_SUCCESS_CODE)
            retval = 1;
```

```
        Close(InFile);
//  char *s = new char[256];
//  int c;
//  c = sprintf(s,"%s\r\n",error);
//  Write(ConsolHandle,s,c);
//  delete[] s;
        //-----------------------------------------------------
        // Take care of some of the last minute cleanup and initialization
        //-----------------------------------------------------
        t->tracksize = ((t->outsideradius - t->insideradius)/t->maxtrack) * 1000000.0;
    }
    else
    {
        sprintf(error,"Could not open %s",name);
        retval = 1;
    }
    return retval;
}

char *GetConfigFileErrorString(void)
{
    return error;
}


//static const char * const PMStrings[] = {
//  "RADIUSCONSTANT",
//  "ANGLECONSTANT",
//  NULL
//};

//static const char * const modestrings[] = {
//  "HSA",
//  "HGA"
//};

//static const char * const OStrings[] = {
//  "BACK",
//  "FRONT"
//};

//static const char * const PidModeStrings[] = {
//  "HEADSON",
//  "HEADSOFF"
//};

//int SaveConfigFile(char *name,AcutracDeviceTable *t)
//{
//  int retval = 0;
//  int handle;
//  String *S = new String;
//  char *s = S->Get();
//  int size;
//
//  if((handle = Open(name,WRITE_ONLY)) >= 0)
//  {
//      size = sprintf(s,"ASCII\r\n");
//      Write(handle,s,long(size));
//      size = sprintf(s,"MAXRPM=%d\r\n",dt->maxrpm );
//      Write(handle,s,long(size));
//      size = sprintf(s,"MINRPM=%d\r\n",dt->minrpm );
//      Write(handle,s,long(size));
//      size = sprintf(s,"TESTRPM=%d\r\n",dt->testrpm );
//      Write(handle,s,long(size));
//      size = sprintf(s,"UNLOADRPM=%d\r\n",dt->unloadrpm );
//      Write(handle,s,long(size));
```

```
//      size = sprintf(s,"LOADRPM=%d\r\n",dt->loadrpm );
//      Write(handle,s,long(size));
//      size = sprintf(s,"ROTATION=%d\r\n",dt->rotation );
//      Write(handle,s,long(size));
//      size = sprintf(s,"INITTRACK=%lf\r\n",dt->inittrack );
//      Write(handle,s,long(size));
//      size = sprintf(s,"MAXTRACK=%lf\r\n",dt->maxtrack );
//      Write(handle,s,long(size));
//      size = sprintf(s,"INSIDERADIUS=%lf\r\n",dt->insideradius );
//      Write(handle,s,long(size));
//      size = sprintf(s,"OUTSIDERADIUS=%lf\r\n",dt->outsideradius );
//      Write(handle,s,long(size));
//      size = sprintf(s,"PIVOTTOGAP=%lf\r\n",dt->pivottogap );
//      Write(handle,s,long(size));
//      size = sprintf(s,"PIVOTTOCENTER=%lf\r\n",dt->pivottocenter );
//      Write(handle,s,long(size));
//      size = sprintf(s,"ACTUATORRADIUS=%lf\r\n",dt->actuatorradius );
//      Write(handle,s,long(size));
//      size = sprintf(s,"REFERENCEANGLE=%lf\r\n",dt->referenceangle );
//      Write(handle,s,long(size));
//      size = sprintf(s,"LOADRAD=%lf\r\n",dt->LoadRadius );
//      Write(handle,s,long(size));
//      size = sprintf(s,"UNLOADRAD=%lf\r\n",dt->UnloadRadius);
//      Write(handle,s,long(size));
//      size = sprintf(s,"XDISTANCEHOME=%lf\r\n",dt->xdistancehome );
//      Write(handle,s,long(size));
//      size = sprintf(s,"XDISTANCERUN=%lf\r\n",dt->xdistancerun );
//      Write(handle,s,long(size));
//      size = sprintf(s,"MAXHEAD=%d\r\n",dt->maxhead );
//      Write(handle,s,long(size));
//      size = sprintf(s,"DUTORIENTATION=%s\r\n",OStrings[(dt->dutorientation+1)?1:0] );
//      Write(handle,s,long(size));
//      size = sprintf(s,"POSPARAMS=%s\r\n",dt->ServoParams);
//      Write(handle,s,long(size));
//      size = sprintf(s,"MODE=%s\r\n",modestrings[dt->mode]);
//      Write(handle,s,long(size));
//      size = sprintf(s,"PITCHMODE=%s\r\n",PMStrings[dt->TrackPitchMode]);
//      Write(handle,s,long(size));
        //---------------------------------------------------------
        // the number of iterations for printing out PidParams is going
        // to be just hard coded right now.
        //---------------------------------------------------------
//      int i;
//      size = sprintf(s,"ROTARY {\r\n");
//      Write(handle,s,long(size));
//      for(i=0;i<2;++i)
//      {
//          size = sprintf(s,"\tSET[%d] {\r\n",i);
//          Write(handle,s,long(size));
//          size = sprintf(s,"\t\tKP=%ld\r\n",dt->Pid[i]->Kp);
//          Write(handle,s,long(size));
//          size = sprintf(s,"\t\tKV=%ld\r\n",dt->Pid[i]->Kv);
//          Write(handle,s,long(size));
//          size = sprintf(s,"\t\tKI=%ld\r\n",dt->Pid[i]->Ki);
//          Write(handle,s,long(size));
//          size = sprintf(s,"\t\tNOTCHF=\r\n",dt->Pid[i]->NotchF);
//          Write(handle,s,long(size));
//          size = sprintf(s,"\t\tNOTCHQ=\r\n",dt->Pid[i]->NotchQ);
//          Write(handle,s,long(size));
//          size = sprintf(s,"\t\tNOTCHD=\r\n",dt->Pid[i]->NotchD);
//          Write(handle,s,long(size));
//          size = sprintf(s,"\t}\r\n");
//          Write(handle,s,long(size));
//      }
//      size = sprintf(s,"}\r\n");
//      Write(handle,s,long(size));
```

```
//      Close(handle);
//  }
//  else
//  {
//      sprintf(error,"Could not open %s for output",name);
//      retval = 1;
//  }
//  delete S;
//  return retval;
//}

void GetLoadConfigError(char *s)
{
    memcpy(s,error,80);
}

}   //end of embedded code
```

```
/*****************************************************************
**
** Header file for Spin Stand Config File
**
*****************************************************************/

#ifndef SCFIG__H
#define SCFIG__H

extern int LoadConfigFile(char *name,AcutracDeviceTable *t);
extern char *GetConfigFileErrorString(void);
extern int SaveConfigFile(char *name,AcutracDeviceTable *t);
extern void GetLoadConfigError(char *s);

#endif
```

```
/*
** define ALLOC and include global.h
*/

#define GLOBAL_ALLOC    1
#include "global.h"     /*  allocate memory for global variables    */
```

```
/*************************************************************************
**
** This file contains all of the global variable declarations, that is
** Data that is truely global
**
*************************************************************************/

#ifndef GLOBAL__H
#define GLOBAL__H

#include "serlprot.h"

#define LINEAR_STAGE    0
#define ROTARY_STAGE    1

/*
** This is the data structure that holds all of the system parameters
*/

extern volatile int GlobalSpindleRPM;

#ifdef GLOBAL_ALLOC
#define EXTERN
#define VALUE   =-1
#else
#define EXTERN extern
#define VALUE
#endif

EXTERN SPINSTAT SystemStatus;
EXTERN int ConsolHandle VALUE;

#endif
```

```
//---------------------------------------------------------------------
// comunications library for 56000 DSP
//
// Copyright (c) 1998 teletrac inc
//
//---------------------------------------------------------------------

#include <stdio.h>
#include "cio.h"
#include "servo.h"
#include "host.h"
#include "task.h"
#include "queue.h"
#include "dsp.h"
#include "acutrac.h"
#include "spindle.h"
#include "frontled.h"
#include "timer.h"
#include "global.h"
#include "strings.h"

extern ConsolHandle;    //debug, delete me someday


static void BeamInterruptedTask(void);
static void LaserNotReadyTask(void);
static void AmplifierFaultTask(void);
static void RequestAmpTask(void);

static int WriteDSPWord(long data,int time);    //write a "word" to host port
static int WaitACK(int time);                   //wait for ACKKNOLEDGE
static int ReadDSPWord(long *data,int time);    //read a "word" from host port
static int SendDSPBuff(CommandFrame cmd,long *b,int size);
static int ReadDSPBuff(long *b,int size);
static int ClearInterface(void);
static int SendCommandFrame(CommandFrame cmd);

static Wait *ServoAvailiable;   //blocking semaphore for servo port
static TSemaphore *MoveDoneSemaphore;
TSemaphore *SpinDoneSemaphore;  //indicates when spindle profile done
static TSemaphore *HomeDoneSemaphore;
static TSemaphore *CalibrateDoneSemaphore;
static Wait *WaitforWaitMove;
static Wait *WaitforWaitHome;
static Wait *WaitforWaitCal;
static Wait *WaitForProtectError;
static Wait *WaitForBeamInterrupted;
static Wait *WaitforLinearHome;
static Wait *WaitforLinearRun;
static TSemaphore *LinearHomeSemaphore;
static TSemaphore *LinearRunSemaphore;
static EventQueue *RequestAmpEvent;

extern volatile int ticks;


/*********************************************************************
**
** Device driver for talking to the DSP56002 on the spindle controller
** board
**
** This device driver will not be connected into CIO like other device
** drivers, since it is not character oriented as far as the data is
** concerned, long words need to be buffered and sent rather than bytes,
** plus, when the buffer has been sent, and command interrupt must be
** sent to the DSP to start the command sequence.
```

```
**
**-----------------------------------------------------------------------
**
**  Messaging is going to GET a little more complicated with this
**
**  When the host sends a message to the DSP, no immediate reply will
** be forth coming.  The host will pend for a reply.  The reply will be
** a message indicating the status of the command.
**
** The DSP will be able to send it's own commands back to the host to
** indicate serious conditions like Amplifier Faults, Beam Interrupts,
** moves being completed, etc.
**
************************************************************************/


/*static*/ DspQueue *InPipe;    /*  pointers to IO records  */
/*static*/ DspQueue *ToDsp;     /*  data to send to DSP     */
/*static*/ MessageQueue *OutPipe;
static EventQueue *LimitPipe;

//-----------------------------------------------------------------------
//
// interrupt handlers for DSP chip
//
//-----------------------------------------------------------------------

extern "C" void HandleDspGet(void)
{
    //-------------------------------------------------------------
    // this function needs to be a state machine.
    // as it fills up the queue of message data, it needs to know when
    // to post the end of the command so that the command can be processed
    //-------------------------------------------------------------

    static int state = 0;       //state machine to determine when to post
    long v;
    static int count;

    while(CheckRxcv())
    {
        v = GetDSP();               //get word from DSP, this clears interrupt
        InPipe->Put(v);             //put data into pipe
        switch(state)
        {
            case 0:
                if(v == DSP_COMMANDACK) state = 3;
                else state = 1;
                break;
            case 1:
                state = 2;      //exception or move complete
                break;
            case 2:
                state = 0;      //exception or move complete, post
                InPipe->Post();
                break;
            case 3:
                state=4;        //commandack
                break;
            case 4:
                if(v > 0)       //is this is a word count
                {
                    count = (int)v;
                    state = 5;
                }
                else
```

```
                {           //done, post
                    state = 0;
                    InPipe->Post();
                }
                break;
            case 5:
                --count;
                if(count == 0)
                {
                    state = 0;  //all data read, post
                    InPipe->Post();
                }
                break;
        }   //end of switch state
    }   //end of while(CheckRxcv())
}

extern "C" void HandleDspPut(void)
{
    int Loop=1;

    while(CheckXMit() && Loop)  //while the transmit registers are empty
    {
        if(ToDsp->QueueStatus())
        {
            PutDSP(ToDsp->Get());
        }
        else
        {
            DisableDSPXmit();   //disable DSP transmit interrupt
            ToDsp->Post();
            Loop=0;
        }
    }
}

static void HandleDSPCommandsTask(void)
{
    //-----------------------------------------------------------------
    // This task recieves data from the HandleDSPGet interrupt routine
    // it decodes the commands and then determines what to do with them
    //-----------------------------------------------------------------
    long cmd;
    long subcmd;
    long aux;
    int i;
    char *s = new char[256];

    while(1)    //loop forever, we are a task
    {
        InPipe->Pend();     //wait for data from DSP pipe
        cmd = InPipe->Get();    //get the command
        subcmd = InPipe->Get(); //get sub command
        aux = InPipe->Get();    //get aux, done getting mandatory data
        switch((int)cmd)
        {
            case DSP_COMMANDACK:
                OutPipe->Put((int)subcmd);
                OutPipe->Put((int)aux);     //error or word count
                if(aux > 0)
                {
                    for(i=0;i<(int)aux;++i)
                    {
                        OutPipe->PutLong(InPipe->Get());    //stuff pipe
                    }
                }
```

```
                OutPipe->Post();    //notify that ACK has occurred
                break;
            case DSP_EXCEPTION:     //exceptions will include things
                                    //like laser beam interrupted
                switch(subcmd)      //which exception was it
                {
                    case DSPEXCEP_BEAMINTERRUPTED:  //indicates beam interrupted error, very bad
 thing
                        WaitForBeamInterrupted->Post();
                        break;
                    case DSPEXCEP_AMPLIFIERFAULT:   //indicates amplifier faulted, bad thing
                        break;
                    case DSPEXCEP_LIMITHIT:         //indicates that a limit was hit, not real b
ad
                        i = (int)aux;
                        LimitPipe->PostMessage(&i,1);   //tell limit task what got limited
                        break;
                    case DSPEXCEP_PROTECTFAULT:     //indicates that a protection fault occured,
very bad thing
                        WaitForProtectError->Post();
                        break;
                    case DSPEXCEP_AMPREQ:
                        i = (int)aux;
                        RequestAmpEvent->PostMessage(&i,1);
                        break;
                    case DSPEXCEP_HISTOGRAM:        //indicates histogram is complete
                        break;
                }   //end of switch subcmd
                break;
            case DSP_MOVECOMPLETE:  //post the semaphore to indicate
                                    //a move has been completed
                switch(subcmd)
                {
                    case DSPMOVE_MOVECOMPLETE:      //indicates move finished
                        if(aux == 1)    //rotary actuator
                            MoveDoneSemaphore->Post(0);
                        else if(aux == 2)   //spindle axis
                            SpinDoneSemaphore->Post(0);
                        break;
                    case DSPMOVE_HOMECOMPLETE:      //indicates home finished
                        HomeDoneSemaphore->Post(0);
                        break;
                    case DSPMOVE_CALIBRATECOMPLETE: //indicates calibrate finished
                        CalibrateDoneSemaphore->Post(0);
                        break;
                    case DSPMOVE_LINEARHOME:    /*  indicates linear is at home */
                        i=sprintf(s,"Linear Home\r\n");
                        Write(ConsolHandle,s,i);
                        LinearHomeSemaphore->Post(0);
                        break;
                    case DSPMOVE_LINEARRUN:     /*  indicates linear is at run  */
                        i=sprintf(s,"Linear Run\r\n");
                        Write(ConsolHandle,s,i);
                        LinearRunSemaphore->Post(0);
                        break;
                }   //end of switch subcmd
                break;
            case DSP_DEBUGSTRING:
                //------------------------------
                //recieve a debug string from
                //the DSP and display it someplace
                //sub command is zero, aux is number
                // of characters (max 256)
                //------------------------------
                for(i=0;i<(int)aux;++i)
                    s[i] = InPipe->Get();   //just get data and throw away for now
```

```
                s[i] = '\0';        //teminate end of string
                Write(ConsolHandle,s,((int)aux));

                break;
        }
    }
}

/*-------------------------------------------------------------
**
** library interface to DSP Host port
**
**-------------------------------------------------------------*/

int telSendDataBlockCommand(CommandFrame cmd, long *d,int n)
{
    int error;
    int SentCommand;
    int Ack;

    if((error = ServoAvailiable->Pend()) == EVENT_NOERROR)  //block access
    {
        SendDSPBuff(cmd,d,n);
        if((error = OutPipe->Pend(40)) == EVENT_NOERROR)    //wait for ACK
        {
            SentCommand = OutPipe->Get();               //get sub command
            Ack = OutPipe->Get();                       //Get ACK
            if(Ack == -1)   //was it NAKED?
                error = SERVO_GOTNAK;
        }
        else if(error == EVENT_TIMEOUT)
        {
            OutPipe->Flush();
            error = SERVO_GOTTIMEOUT;               //convert to servo timeout
        }
        else if(error == EVENT_TERMINATE)
            OutPipe->Flush();
    }
    ServoAvailiable->Post();                //unblock access
    return error;
}

int telSendDataCommand(CommandFrame cmd)
{
    int error;
    int SentCommand;
    int Ack;

    if((error = ServoAvailiable->Pend()) == EVENT_NOERROR)  //block access
    {
        SendCommandFrame(cmd);
        if((error = OutPipe->Pend(40)) == EVENT_NOERROR)    //.1 second timeout
        {
            SentCommand = OutPipe->Get();       //get sub command
            Ack = OutPipe->Get();           //get either bytecount or error
            if(Ack == -1)               //was it NAKED?
                error = SERVO_GOTNAK;
        }
        else if(error == EVENT_TIMEOUT)
        {
            OutPipe->Flush();
            error = SERVO_GOTTIMEOUT;       //convert to timeout value for servo
        }
        else if(error == EVENT_TERMINATE)
            OutPipe->Flush();
    }
```

```
    ServoAvailiable->Post();    //unblock access
    return error;
}

int telGetDataCommand(CommandFrame cmd,long *d,int n)
{
    //
    //this function reads data from the host port of the DSP
    //
    int error;
    int SentCommand;
    int ByteCount;
    int i;

    if((error = ServoAvailiable->Pend()) == EVENT_NOERROR)  //block access
    {
        SendCommandFrame(cmd);
        if((error = OutPipe->Pend(40)) == EVENT_NOERROR)            //wait for ACK from DSP
        {
            SentCommand = OutPipe->Get();       //get sub command
            ByteCount = OutPipe->Get();         //get either bytecount or error

            if(ByteCount == n)
            {
                for(i=0;i<n;++i)
                    *d++ = OutPipe->GetLong();
            }
            else if(ByteCount == -1)
                error = SERVO_GOTNAK;
            else
            {
                error = SERVO_ERROR;
            }
        }
        else if(error == EVENT_TIMEOUT)
        {
            OutPipe->Flush();
            error = SERVO_GOTTIMEOUT;
        }
        else if(error == EVENT_TERMINATE)
            OutPipe->Flush();
    }
    ServoAvailiable->Post();        //unblock access
    return(error);
}

//---------------------------------------------------------------------------
// Initialize the DSP port interface for the servo controllers
//---------------------------------------------------------------------------

void InitServoPort(void)
{
    TCB *t;

    ServoAvailiable = new Wait(1,"ServoBlocker");               /* one use only */
    MoveDoneSemaphore = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"MoveDone");
    SpinDoneSemaphore = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"SpinupDone"); //indicates when
 spindle profile done
    HomeDoneSemaphore = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"HomeDone");
    CalibrateDoneSemaphore = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"CalibrateDone");
    WaitforWaitMove = new Wait(1,"WaitForMoveBlocker");             //use only once blocking sem
aphore
    WaitforWaitHome = new Wait(1,"WaitForHomeBlocker");             //use only once blocking sem
aphore
    WaitforWaitCal = new Wait(1,"WaitForCalBlocker");               //use only once blocking sem
aphore
```

```
    WaitForProtectError = new Wait(0,"WaitForProtectError");
    WaitForBeamInterrupted = new Wait(0,"WaitForBeamInterrupted");
    LimitPipe = new EventQueue(8,"LimitPipe");
    WaitforLinearHome = new Wait(1,"WaitForLinearHome");           //use only once blocking sem
aphore
    WaitforLinearRun = new Wait(1,"WaitForLinearRun");             //use only once blocking sem
aphore
    LinearHomeSemaphore = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"LinearHome");
    LinearRunSemaphore = new TSemaphore(0,TSEMAPHORE_MODE_TIMEOUT,"LinearRun");
    RequestAmpEvent = new EventQueue(16,"RequestAmp");

    //-------------------------------------------------------------
    // The DSP Command Dispatcher must be a very high priority
    // It will post semaphores to other tasks
    //-------------------------------------------------------------
    t = CreateTask(HandleDSPCommandsTask,512,254,"DSP_MESSAGE_TASK");
    ActiveTasks->Insert(t);     //add this task to list of waiting tasks
    t = CreateTask(BeamInterruptedTask,512,128,"BEAM_INTERRUPTED");
    ActiveTasks->Insert(t);     //add this task to list of waiting tasks
    t = CreateTask(RequestAmpTask,512,12,"REQUEST_AMP");
    ActiveTasks->Insert(t);
    InPipe = new DspQueue(512,"DspInPipe");     /*  data from DSP   */
    ToDsp = new DspQueue(512,"ToDspPipe");      /*  data to send to DSP     */
    OutPipe = new MessageQueue(sizeof(long)*512,1,"DataFromDSP");   /*  data to DSP Task    */

    //init hardware on DSP chip
    *((char *)DSP_IRQVEC) = 208;    /*  vector number of DSP irq    */
    *((char *)DSP_IRQCTL) = ICR_RREQ;   //enable host recieve irq   */
}

//-------------------------------------------------------------------------
//private member functions
//-------------------------------------------------------------------------

static int SendDSPBuff(CommandFrame cmd,long *b,int size)
{
    int i;
    //first send command frame
    ToDsp->Put(cmd.command);
    ToDsp->Put(cmd.axis);
    ToDsp->Put(cmd.func);
    ToDsp->Put(cmd.aux[0]);
    ToDsp->Put(cmd.aux[1]);
    //now send data
    for(i=0;i<size;++i)
        ToDsp->Put(*b++);
    EnableDSPXmit();        /*  enable DSP transmit interrupt   */
    ToDsp->Pend();

    ticks = 100;    //wait for 10 mS
    while(!( *((char *)DSP_ISR) & ISR_TRDY) )
    {
        if(ticks < 0)       //check for a timeout
        {
            return(-1);
        }
    }

    *((char *)DSP_CMDVEC) = 0x80 | 18;  //vector 18

    return 0;
}

static int SendCommandFrame(CommandFrame cmd)
{
    //send command frame to servo card through host port
```

```
    //return 0 if ok, -1 if there was an error
    int error=0;

    ToDsp->Put(cmd.command);
    ToDsp->Put(cmd.axis);
    ToDsp->Put(cmd.func);
    ToDsp->Put(cmd.aux[0]);
    ToDsp->Put(cmd.aux[1]);
    EnableDSPXmit();        /*  enable DSP transmit interrupt   */
    ToDsp->Pend();
    //
    //wait for the host port to GRAB all of the data that we gave it
    //
    ticks = 100;    //wait for 10 mS
    while(!( *((char *)DSP_ISR) & ISR_TRDY) )
    {
        if(ticks < 0)       //check for a timeout
        {
            return(-1);
        }
    }

    //
    //if we got this far, set the command bit and vector in the
    //command vector register
    //
    *((char *)DSP_CMDVEC) = 0x80 | 18;  //vector 18

    return(error);
}


static int ClearInterface(void)
{
    //This routine attempts to clear the host interface if there is an error
    //It will attempt to to clear the port if there is any data stuck in it
    int i,loop;
    volatile int dummy;

    for(i=34,loop = 1;(i > 0) && loop;--i)
    {
        if(!( *( (char *)DSP_ISR) & ISR_RXDF) )
            loop = 0;   //break loop
        else
            dummy = *((char *)DSP_DATAL);       //attempt to clear port
    }
    return(i);
}

//---------------------------------------------------------------------------
// Command Access Functions
//---------------------------------------------------------------------------

//***************************************************************************
// PES functions
//***************************************************************************

int SetPesMode(int axis,long mode)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_PESMODE;
    cmd.axis = axis;
    cmd.aux[0] = mode;
    return telSendDataCommand(cmd);
}
```

```
int SetPesScale(int axis,long scale)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_PESSCALE;
    cmd.axis = axis;
    cmd.aux[0] = scale;
    return telSendDataCommand(cmd);
}

int GetFPGAID(int axis,long *id)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_FPGAID;
    cmd.axis = axis;
    return telGetDataCommand(cmd,id,1);
}

int GetPesScale(int axis,long *scale)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_PESSCALE;
    cmd.axis = axis;
    return telGetDataCommand(cmd,scale,1);
}

int SetPesLimit(int axis,long limit)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_PESLIMIT;
    cmd.axis = axis;
    cmd.aux[0] = limit;
    return telSendDataCommand(cmd);
}

int GetPesLimit(int axis,long *limit)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_PESLIMIT;
    cmd.axis = axis;
    return telGetDataCommand(cmd,limit,1);
}

int ResetPesCounter(int axis)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_RESET_PES;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

//*************************************************************************
// Special Linear Access commands for pnumatic slide
//*************************************************************************

int RecordLinearPosition(int axis,long func,long aux)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_RECORD_LINPOS;
```

```
    cmd.axis = axis;
    cmd.aux[0] = aux;
    cmd.func = func;
    return telSendDataCommand(cmd);
}

//************************************************************************
//
// Histogram functions
//
//************************************************************************

int GetHistogram(int axis,long *buff)
{
    //-------------------------------------------------
    // this function assumes that buff points to an
    // array of longs that is at least 32 deap
    //-------------------------------------------------
    CommandFrame cmd;
    int error;

    cmd.command = DSPCOM_GET_HISTOGRAM;
    cmd.axis = axis;
    return telGetDataCommand(cmd,buff,32);
}

int StartHistogram(int axis,long nsamples)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_START_HISTOGRAM;
    cmd.aux[0] = nsamples;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

//************************************************************************
// Motion Functions
//************************************************************************

int AbortMove(int axis)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_MOVE_ABORT;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetHandshakeLine(int level)
{

    CommandFrame cmd;

    cmd.command = DSPCOM_SET_HANDSHAKE;
    cmd.aux[0] = (long)level;
    cmd.axis = 0;
    return telSendDataCommand(cmd);
}

int DoMoveRel(int axis)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_DO_MOVE;
    cmd.aux[1] = MOVE_RELATIVE;
```

```
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Do_move(int axis,int move_mode)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_DO_MOVE;
    cmd.aux[1] = move_mode;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Read_Goal(int axis,long *Goal)
{
    CommandFrame cmd;
    int error;
    long v[2];

    cmd.command = DSPCOM_READ_GOAL;
    cmd.axis = axis;
    error = telGetDataCommand(cmd,v,2);
    *Goal =(v[0] & 0x00ffffff) | (v[1] << 24);
    return(error);
}

int Read_MaxVel(int axis,long *bits_per_decimillisec)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_MAX_VEL;
    cmd.axis = axis;
    return telGetDataCommand(cmd,bits_per_decimillisec,1);
}

int Read_SCurve(int axis,long *stime)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_SCURVE_TIME;
    cmd.axis = axis;
    return telGetDataCommand(cmd,stime,1);
}

int SetGoalAndMove(int axis,long Goal)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SETGOALANDMOVE2;
    cmd.aux[0] = Goal;
    cmd.aux[1] = Goal>>24;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_Goal(int axis,long Goal)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_GOAL;
    cmd.aux[0] = Goal;
    cmd.aux[1] = Goal>>24;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}
```

```
int Set_SCurve(int axis,long s_curve_time)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_SCURVE;
    cmd.aux[0] = s_curve_time;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

//***************************************************************************
// Interrupt Functions
//***************************************************************************

int ClearInterrupt(int axis,long func)
{
    //Clears a BEAM interrupt error
    CommandFrame cmd;

    cmd.command = DSPCOM_CLEAR_INTERRUPT;       //clear interrupt
    cmd.func = func;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int GetIrqParams(int axis,long vector,long *val)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_INTERRUPTS;
    cmd.axis = axis;
    cmd.aux[0] = vector;
    return telGetDataCommand(cmd,val,1);
}

//***************************************************************************
// Homing and calibration functions
//***************************************************************************

int Do_Calibrate(int axis)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_DO_HOME;
    cmd.aux[0] = 1;     /*  do a system calibrate   */
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Do_Home(int axis)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_DO_HOME;
    cmd.aux[0] = 0;     /*  do a normal homing thing    */
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Get_Calibration(int axis,long *val)
{
    CommandFrame cmd;
    int error;
    long v[2];
```

```
    cmd.command = DSPCOM_READ_CALDIST;
    cmd.axis = axis;
    error = telGetDataCommand(cmd,v,2);
    *val = v[0] & 0x00ffffff | (v[1] << 24);    /*  convert value   */
    return(error);
}

int Get_HomeParams(int axis,HOMEPARAMS *HomeParams)
{
    //this routine gets a block of home parameters
    long *v;
    long data[8];   //for DSP_long conversion
    CommandFrame cmd;
    int error = 0;
    int i;


    cmd.command = DSPCOM_GET_HOME_PARAMS;
    cmd.axis = axis;
    for(i=0;i<6;++i)
    {
        cmd.func = i;
        if((error = telGetDataCommand(cmd,&data[i],1)) < 0)
            return(error);
    }
    cmd.func = i;
    if((error = telGetDataCommand(cmd,&data[i],2)) < 0)
        return error;
    for(v = (long *)HomeParams,i=0;i<6;++i)
        *v++ = data[i];
    HomeParams->zerodist = (data[6] & 0x00ffffff) | (data[7] << 24);
    return(error);
}

int Set_HomeParams(int axis,HOMEPARAMS HomeParams)  /*  set home parameters */
{
    CommandFrame cmd;
    int error = 0;
    int i;
    long *v;
    long data[8];

    //This routine sends a block of data to the servo axis

    cmd.command = DSPCOM_HOME_SETUP;
    cmd.axis = axis;
    for(v = (long *)&HomeParams,i=0;i < 6;++i,++v)  /*  set long values */
    {
        cmd.aux[0] = *v;
        cmd.func = i;
        if((error = telSendDataCommand(cmd)) < 0)
            return(error);
    }
    cmd.aux[0] = HomeParams.zerodist;
    cmd.aux[1] = HomeParams.zerodist>>24;
    cmd.func = i;
    return telSendDataCommand(cmd);
}

//***************************************************************************
// Jogging Functions
//***************************************************************************

int StartJog(int axis)
{
    CommandFrame cmd;
```

```
    cmd.command = DSPCOM_START_JOG;     //send distance and start jog
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int EndJog(int axis)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_END_JOG;       //send time
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int GetJogParams(int axis,JOGPARAMS *JogParams)
{
    CommandFrame cmd;
    int error = 0;
    long data[7];
    int i;

    cmd.command = DSPCOM_READ_JOG_PARAMS;
    cmd.axis = axis;
    cmd.func = JOGPARAM_START;
    if((error = telGetDataCommand(cmd,&data[0],2)) < 0)
        return(error);
    cmd.func = JOGPARAM_END;
    if((error = telGetDataCommand(cmd,&data[2],2)) < 0)
        return(error);
    for(i=2;i<5;++i)
    {
        cmd.func = i;
        if((error = telGetDataCommand(cmd,&data[2+i],1)) < 0)
            return(error);
    }
    JogParams->start = (data[0] & 0x00ffffff) | (data[1] << 24);
    JogParams->end = (data[2] & 0X00ffffff) | (data[3]<<24);
    JogParams->index = data[4];
    JogParams->mode = data [5];
    JogParams->time = data[6];
    return(error);
}

//***************************************************************************
// PID loop filter parameters
//***************************************************************************
extern int ReadNotchDepth(int axis,long *d,int filtn)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_NOTCH_D;
    cmd.axis = axis;
    cmd.aux[0] = (long)filtn;   //filter number (0/1)
    return telGetDataCommand(cmd,d,1);
}

extern int SetNotchDepth(int axis,long d,int filtn)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_NOTCH_D;
    cmd.aux[0] = d;
    cmd.aux[1] = (long)filtn;   //filter number (0/1)
    cmd.axis = axis;
    return telSendDataCommand(cmd);
```

```
}

int SetNotchQ(int axis,long q,int filtn)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_NOTCH_Q;
    cmd.aux[0] = q;
    cmd.aux[1] = (long)filtn;   //filter number (0/1)
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetNotchFreq(int axis,long f, int filtn)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_NOTCH_FREQ;
    cmd.aux[0] = f;
    cmd.aux[1] = (long)filtn;   //filter number (0/1)
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int ReadNotchQ(int axis,long *q,int filtn)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_NOTCH_Q;
    cmd.axis = axis;
    cmd.aux[0] = (long)filtn;   //filter number (0/1)
    return telGetDataCommand(cmd,q,1);
}

int ReadNotchFreq(int axis,long *f,int filtn)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_NOTCH_FREQ;
    cmd.axis = axis;
    cmd.aux[0] = (long)filtn;   //filter number (0/1)
    return telGetDataCommand(cmd,f,1);
}

int Read_Kaff(int axis,long *kaff)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_KAFF;
    cmd.axis = axis;
    return telGetDataCommand(cmd,kaff,1);
}

int Read_Ki(int axis,long *ki)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_KI;
    cmd.axis = axis;
    return telGetDataCommand(cmd,ki,1);
}

int Read_Kp(int axis,long *kp)
{
    CommandFrame cmd;
```

```
    cmd.command = DSPCOM_READ_KP;
    cmd.axis = axis;
    return telGetDataCommand(cmd,kp,1);
}

int Read_Kv(int axis,long *kv)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_KV;
    cmd.axis = axis;
    return telGetDataCommand(cmd,kv,1);
}

int Read_Kvff(int axis,long *kvff)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_KVFF;
    cmd.axis = axis;
    return telGetDataCommand(cmd,kvff,1);
}

int Read_Offset(int axis,long *offset)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_OFFSET;
    cmd.axis = axis;
    return telGetDataCommand(cmd,offset,1);
}

int EnableServoFunctions(int axis,long func,long flag)
{
    //Enables/disables the servo loop
    CommandFrame cmd;

    cmd.command = DSPCOM_ENABLE_SERVO_FUNCTIONS;
    cmd.aux[0] = func;
    cmd.aux[1] = flag;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int GetIntegratorMode(int axis,long *mode)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_INT_MODE;
    cmd.axis = axis;
    return telGetDataCommand(cmd,mode,1);
}

int Read_IntegratorLimits(int axis,long *limit)
{
    //Reads the integrator limits
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_INT_LIMITS;
    cmd.axis = axis;
    return telGetDataCommand(cmd,limit,1);
}

int Set_Kaff(int axis,long kaff)
{
    CommandFrame cmd;
```

```
    cmd.command = DSPCOM_SET_KAFF;
    cmd.aux[0] = kaff;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_Ki(int axis,long ki)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_KI;
    cmd.aux[0] = ki;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_Kp(int axis,long kp)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_KP;
    cmd.aux[0] = kp;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_Kv(int axis,long kv)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_KV;
    cmd.aux[0] = kv;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_Kvff(int axis,long kvff)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_KVFF;
    cmd.aux[0] = kvff;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_Offset(int axis,long offset)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_OFFSET;
    cmd.aux[0] = offset;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_IntegratorLimits(int axis,long limit)
{
    //Sets the integrator limits
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_INT_LIMITS;
    cmd.aux[0] = limit;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
```

```
}

int SetIntegratorMode(int axis,long mode)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_INT_MODE;
    cmd.aux[0] = mode;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

//*************************************************************************
//
// These are functions that are used for generating BODE diagrams of
// the response of the servo loops.
// From the information that can be retrieved by these functions, phase and
// amplitude response can be calcualted
//*************************************************************************

int SetAnalMode(int axis,long mode)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_ANAL;
    cmd.aux[0] = mode;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetFrequency(int axis,long v)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_FREQUENCY;
    cmd.aux[0] = v;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetFilterBandwidth(int axis,long w,long n)
{
    //---------------------------------------
    // parameter:axis--which motion axis to set
    //          :w-----cuttoff frequency of filter
    //          :n-----analyzer number
    //---------------------------------------
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_FILTERFREQ;
    cmd.aux[0] = w;
    cmd.axis = axis;
    cmd.func = n;
    return telSendDataCommand(cmd);
}

int SetAmplitude(int axis,long v)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_AMPLITUDE;
    cmd.aux[0] = v;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}
```

```
int GetFrequency(int axis,long *v)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_FREQUENCY;
    cmd.axis = axis;
    return telGetDataCommand(cmd,v,1);
}

int GetFilterBandwidth(int axis,long *w,long n)
{
    //-----------------------------------------
    // parameter:axis--which motion axis to set
    //          :w-----cuttoff frequency of filter
    //          :n-----analyzer number
    //-----------------------------------------
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_FILTFREQ;
    cmd.axis = axis;
    cmd.func = n;
    return telGetDataCommand(cmd,w,1);
}

extern int GetMagnitude1(int axis,long *real,long *imag,long n)
{
    //-------------------------------------------------
    //parameter:axis--which motion axis
    //         :real--real part of analyzer output
    //         :imag--imaginary part of analyzer output
    //         :n-----which analyzer to get output from
    //-------------------------------------------------
    CommandFrame cmd;
    int retval;
    long a[4];

    cmd.command = DSPCOM_GET_MAGNITUDE1;
    cmd.axis = axis;
    cmd.func = n;
    if((retval = telGetDataCommand(cmd,a,4)) == 0)
    {
        *real = ((a[1] << 24) & 0xff000000l) | (a[0] & 0x00ffffffl);
        *imag = ((a[3] << 24) & 0xff000000l) | (a[2] & 0x00ffffffl);
    }
    return retval;
}

int GetMagnitude(int axis,long *a,long n)
{
    /*****************************************************
    ** This function returns back two long to a space pointed
    ** to by a.  User must assure that there is enough space
    ** to put these two values.
    ** The Sine value is returned at index 0,
    ** The cosine value is returned at index 1.
    ** n indicates which analyzer to get data from
    *****************************************************/
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_MAGNITUDE;
    cmd.axis = axis;
    cmd.func = n;
    return telGetDataCommand(cmd,a,2);
}

int GetAmplitude(int axis,long *a)
```

```
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_AMPLITUDE;
    cmd.axis = axis;
    return telGetDataCommand(cmd,a,1);
}

int SetPhase(int axis,long a)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_PHASE;
    cmd.aux[0] = a;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int GetPhase(int axis,long *a)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_PHASE;
    cmd.axis = axis;
    return telGetDataCommand(cmd,a,1);
}

int SetDitherMode(int axis,long mode)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_DITHERMODE;
    cmd.aux[0] = mode;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int GetDitherMode(int axis,long *mode)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_DITHERMODE;
    cmd.axis = axis;
    return telGetDataCommand(cmd,mode,1);
}

int SetLookupTableSelection(int axis,long table)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_RUNOUTLUT;
    cmd.aux[0] = table;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int GetLookupTableSelection(int axis,long *table)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_RUNOUTLUT;
    cmd.axis = axis;
    return telGetDataCommand(cmd,table,1);
}

//***************************************************************************
```

```
// Miscilaneous Functions
//***************************************************************************

int GetEncoderPtich(int axis, long *val)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_ENCODERPITCH;
    cmd.axis = axis;
    return telGetDataCommand(cmd,val,1);
}

int SetEncoderPitch(int axis, long val)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_ENCODERPITCH;
    cmd.aux[0] = val;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int GetInPositionThreshold(int axis, long *val)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_INPOSTHRESH;
    cmd.axis = axis;
    return telGetDataCommand(cmd,val,1);
}

int GetVersionNumber(int axis,char *s,int n)
{
    CommandFrame cmd;
    int error = 0;
    long temp[16];      /*  temp long array to put data */
    int i;

    cmd.command = DSPCOM_GET_SN;
    cmd.aux[0] = n;     /*  which string to get */
    cmd.axis = axis;
    if(!(error = telGetDataCommand(cmd,temp,16)) )
    {
        for(i=0;i<16;++i)
            s[i] = (char)temp[i];   /*  convert longs to char   */
        s[i] = 0;                   //null terminate string
    }
    return(error);
}

int Read_DacLev(int axis,long *dacval)
{
    // read the value that is in the motor DAC
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_DAC_LEV;
    cmd.axis = axis;
    return telGetDataCommand(cmd,dacval,1);
}

int Read_FollowGain(int axis,long *gain)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_FOL_GAIN;
    cmd.axis = axis;
```

```
    return telGetDataCommand(cmd,gain,1);
}

int Read_MaxFollowingError(int axis,long *limit)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_MAX_FOLLOW;
    cmd.axis = axis;
    return telGetDataCommand(cmd,limit,1);
}

int Read_PhaseDetectorPolarity(int axis,long *pol)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_PHASE_DETECT_POL;
    cmd.axis = axis;
    return telGetDataCommand(cmd,pol,1);
}

int ReadPmem(int axis,long *v, long addr)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_PMEM;
    cmd.axis = axis;
    cmd.aux[0] = addr;
    return telGetDataCommand(cmd,v,1);
}

int Read_Position(int axis,long *val)
{
    CommandFrame cmd;
    int error;
    long v[2];

    cmd.command = DSPCOM_READ_POSITION;
    cmd.axis = axis;
    error = telGetDataCommand(cmd,v,2);
    *val = (v[0] & 0x00ffffff) | (v[1] << 24);
    return(error);
}

int Read_SampleRate(int axis,long *rate)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_SAMPLE_RATE;
    cmd.axis = axis;
    return telGetDataCommand(cmd,rate,1);
}

int ReadXmem(int axis,long *v, long addr)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_READ_XMEM;
    cmd.aux[0] = addr;
    cmd.axis = axis;
    return telGetDataCommand(cmd,v,1);
}

int ReadYmem(int axis,long *v, long addr)
{
    CommandFrame cmd;
```

```
    cmd.command = DSPCOM_READ_YMEM;
    cmd.aux[0] = addr;
    cmd.axis = axis;
    return telGetDataCommand(cmd,v,1);
}

int GetDataStructures(int axis,int type,long *v)
{
    //--------------------------------
    // v points to an array of two longs
    // v[0]->address of structure
    // v[1]->size of structure
    //--------------------------------
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_DATA_STRUCT;
    cmd.func = (long)type;
    cmd.axis = axis;
    return telGetDataCommand(cmd,v,2);
}

int ReadBump(int axis,long *bump)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_BUMP;
    cmd.axis = axis;
    return telGetDataCommand(cmd,bump,1);
}

int ServoStatus(int axis,DSP_status *stat)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SERVO_STATUS;  //Send the status command
    cmd.axis = axis;
    return telGetDataCommand(cmd,(long *)stat,sizeof(DSP_status) / sizeof(long) );
}

int SetBump(int axis,long bump)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_BUMP;
    cmd.aux[0] = bump;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetInPositionThreshold(int axis,long val)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_INPOS_THRESH;
    cmd.aux[0] = val;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetIrqParams(int axis,long vector,long val)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_INTERRUPTS;
    cmd.aux[0] = vector;
```

```
    cmd.aux[1] = val;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetJogParams(int axis,JOGPARAMS JogParams)
{
    CommandFrame cmd;
    int error = 0;

    cmd.command = DSPCOM_SET_JOG_PARAM;
    cmd.axis = axis;
    cmd.func = JOGPARAM_START;
    cmd.aux[0] = JogParams.start;
    cmd.aux[1] = JogParams.start>>24;
    if((error = telSendDataCommand(cmd)) < 0)
        return(error);
    cmd.func = JOGPARAM_END;
    cmd.aux[0] = JogParams.end;
    cmd.aux[1] = JogParams.end>>24;
    if((error = telSendDataCommand(cmd)) < 0)
        return(error);
    cmd.func = JOGPARAM_INDEX;
    cmd.aux[0] = JogParams.index;
    if((error = telSendDataCommand(cmd)) < 0)
        return(error);
    cmd.func = JOGPARAM_MODE;
    cmd.aux[0] = JogParams.mode;
    if((error = telSendDataCommand(cmd)) < 0)
        return(error);
    cmd.func = JOGPARAM_DWELL;
    cmd.aux[0] = JogParams.time;
    if((error = telSendDataCommand(cmd)) < 0)
        return(error);
    return(error);
}

int Set_MaxVel(int axis,long bits_per_decimillisec)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_MAXVEL;
    cmd.aux[0] = bits_per_decimillisec;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_PhaseDetectorPolarity(int axis,long pol)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_PHASE_DETECT_POL;
    cmd.aux[0] = pol;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_SampleRate(int axis,long rate)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_SAMPLE_RATE;
    cmd.aux[0] = rate;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}
```

```
int Set_FollowGain(int axis,long gain)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_FOL_GAIN;
    cmd.aux[0] = gain;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int Set_MaxFollowingError(int axis,long limit)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_MAX_FOLLOW;
    cmd.aux[0] = limit;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int ZeroCount(int axis)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_ZERO_COUNT;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int ZeroIntegrator(int axis)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_ZERO_INTEGRATOR;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetGoalReg(int axis,long Goal)
{
    //-------------------------------
    // this is a debug function
    // results may vary
    //-------------------------------
    CommandFrame cmd;

    cmd.command = DSPCOM_CLEAR_FLAG;
    cmd.aux[0] = Goal;
    cmd.aux[1] = Goal>>24;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int WriteDSPMem(int axis,long addr,long d,int func)
{
    //-------------------------------
    // this is a debug function
    // results may vary
    //-------------------------------
    CommandFrame cmd;

    cmd.command = DSPCOM_WRITE_MEM;
    cmd.aux[0] = addr;      //address
    cmd.aux[1] = d;         //data
    cmd.func = func;
```

```
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}


int ClearFlag(int axis)
{
    //------------------------------
    // this is a debug function
    // results may vary
    //------------------------------
    CommandFrame cmd;

    cmd.command = DSPCOM_CLEAR_FLAG;
    cmd.axis = axis;
    return telSendDataCommand(cmd);
}

int SetRunoutCompTable(int axis,long *data)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_RUNOUTCOMP;
    cmd.axis = axis;
    cmd.aux[0] = 256l;
    return telSendDataBlockCommand(cmd,data,256);
}

int GetRunoutCompTable(int axis,long *data)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_RUNOUTCOMP;
    cmd.axis = axis;
    return telGetDataCommand(cmd,data,256);
}

//---------------------------------------------------------------------------
//
// Functions that Wait for Something to Happen
//
//---------------------------------------------------------------------------

void ClearWaitForLinearHome(void)
{
    LinearHomeSemaphore->SetCount(0);
}

int WaitForLinearHome(int timeout)
{
    int error;

    error = LinearHomeSemaphore->Pend(timeout);
    return error;
}

void ClearWaitForLinearRun(void)
{
    LinearRunSemaphore->SetCount(0);
}

int WaitForLinearRun(int timeout)
{
    int error;

    error = LinearRunSemaphore->Pend(timeout);
```

```
    return error;
}

void ClearMoveDoneSemaphore(void)
{
    MoveDoneSemaphore->SetCount(0); //Clear the Semaphore back to netral
}

int WaitForRotaryMove(int timeout)
{
    int error;

    WaitforWaitMove->Pend();        //only want to wait for this stuff one at a time
    error = MoveDoneSemaphore->Pend(timeout);   //pend of move complete
    WaitforWaitMove->Post();
//  if(error == EVENT_TIMEOUT) error = ACUTRAC_MOTIONTIMEOUT;
    return error;
}

void ClearWaitForRotaryHome(void)
{
    HomeDoneSemaphore->SetCount(0);
}

int WaitForRotaryHome(int timeout)
{
    int error;

    WaitforWaitHome->Pend();
    error = HomeDoneSemaphore->Pend(timeout);
    WaitforWaitHome->Post();
//  if(error == EVENT_TIMEOUT) error = ACUTRAC_MOTIONTIMEOUT;
    return error;
}

void ClearWaitForRotaryCalibrate(void)
{
    CalibrateDoneSemaphore->SetCount(0);
}

int WaitForRotaryCalibrate(int timeout)
{
    int error;

    WaitforWaitCal->Pend();
    error = CalibrateDoneSemaphore->Pend(timeout);
    WaitforWaitCal->Post();
    if(error == EVENT_TIMEOUT) error = ACUTRAC_MOTIONTIMEOUT;
    return error;
}

void DisplayWaitForRotaryCalibrateSemaphores(void)
{
    WaitforWaitCal->PrintInfo();
    CalibrateDoneSemaphore->PrintInfo();
}

/*************************************************************************
**
** These are two TASKS that are used to save and restore Servo parameters
** Servo Parameters will be stored in RAMDISK.  A file will use up 512 bytes
** of data
**
*************************************************************************/

Wait *SaveFlag,*RestoreFlag;
```

```
static long ConFigBuff[256];

int GetConfigParams(long *data)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_CONFIG;
    cmd.axis = 0;
    return telGetDataCommand(cmd,data,207); //change when dsp code is changed
}

int GetOtherConfigParams(long *data)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_GET_OTHER_CONFIG;
    cmd.axis = 0;
    return telGetDataCommand(cmd,data,48);  //change when dsp code is changed
}

int SetConfigParams(long *data)
{
    CommandFrame cmd;

    cmd.command = DSPCOM_SET_CONFIG;
    cmd.axis = 0;
    cmd.aux[0] = 207l;      //every time the DSP code is changed, these
    cmd.aux[1] = 48l;       //values have to change
    return telSendDataBlockCommand(cmd,data,255);
}

static void SaveConfigParams(void)
{
    int error;
    int handle;

    while(1)    //this is a TASK, it loops forever
    {
        SaveFlag->Pend();                                   //go to sleep
        if((handle = Open("D:SERVCFIG",WRITE_ONLY)) >= 0)   //open up file
        {
            if((error = GetConfigParams(ConFigBuff)) == 0)  //get config data
                Write(handle,(char *)ConFigBuff,828l);          //write data to file
            if((error = GetOtherConfigParams(ConFigBuff)) == 0)
                Write(handle,(char *)ConFigBuff,192l);
            Close(handle);
        }
    }
}

static void RestoreConfigParams(void)
{
    int error;
    int handle;

    while(1)    //this is a TASK, it loops forever
    {
        RestoreFlag->Pend();                                //wait for task to wake up
        if((handle = Open("D:SERVCFIG",READ_ONLY)) >= 0)    //open up file
        {
            Read(handle,(char *)ConFigBuff,1020l);              //read data into buffer
            Close(handle);                                  //close file
            SetConfigParams(ConFigBuff);                    //write data to DSP
        }
    }
}
```

```
void InitSaveRestore(void)
{
    TCB *t;

    SaveFlag = new Wait(0,"SaveFlag");
    RestoreFlag = new Wait(0,"RestoreFlag");
    t = CreateTask(SaveConfigParams,2048,13,"Save_Config");
    ActiveTasks->Insert(t);
    t = CreateTask(RestoreConfigParams,2048,14,"Restore_Config");   //make this task slightly hi
gher
    ActiveTasks->Insert(t);
}

int SaveSettings(int axis)
{
    SaveFlag->Post();   //start up task to save settings
    return 0;
}

int RestoreSettings(int axis)
{
    RestoreFlag->Post();    //start up task to restore settings
    return 0;
}

//---------------------------------------------------------------------
//
// Special functions for dealing with the MicroE encoder board
// for this version, do it simple, do not go through the DSP to turn
// the auto compensation on or off
//
//---------------------------------------------------------------------

int SetMicroEAutoComp(int flag)
{
    //flag = 1:auto comp on
    //flag = 0:auto comp off
    if(flag == MICROE_AUTOCOMP_ON)
        SetPortCBits(0x02); //
    else if(flag == MICROE_AUTOCOMP_OFF)
        ClearPortCBits(0x02);   //
    return 0;
}

int GetMicroEAutoComp(int *flag)
{
    int v = GetPortCBits();
    *flag = (v & 0x02)?1:0;
    return 0;
}

//---------------------------------------------------------------------
//
// These Tasks are used to handle various exceptions from the servo card
//
// They include tasks for handling Beam Interrupt, Laser Not Ready, etc.
//
//---------------------------------------------------------------------
//

static void RequestAmpTask(void)
{
    int m;

    while(1)    //this is a task, loop forever
```

```
    {
        RequestAmpEvent->Pend();
        m = RequestAmpEvent->Get(); //get message value
        SetPortC(FRONTLED_EXTAMPSEL,(m?1:0));
        SetHandshakeLine(m?1:0);

    }
}

static void BeamInterruptedTask(void)
{

    while(1)    //this is a task, loop forever
    {
        WaitForBeamInterrupted->Pend(); //we can wait for this forever
    }
}

static void LaserNotReadyTask(void)
{
    while(1)    //this is a task, loop forever
    {
        WaitForProtectError->Pend();    //we can wait for this forever
    }
}

static void AmplifierFaultTask(void)
{
    while(1)    //this is a task, loop forever
    {
    }
}
```

```
    ;*****************************************************************
    ;
    ; Interrupt Handler for DSP Chip
    ;
    ;*****************************************************************

DSP_IRQCTL  =   0xffd40001
DSP_CMDVEC  =   0xffd40003
DSP_ISR     =   0xffd40005
DSP_IRQVEC  =   0xffd40007
DSP_DATA    =   0xffd40009
DSP_DATAH   =   0xffd4000b
DSP_DATAM   =   0xffd4000d
DSP_DATAL   =   0xffd4000f

HEAD    =   4
TAIL    =   6
SIZE    =   8
NCHAR   =   10
LOW     =   12
HIGH    =   14
TIMEOUT =   16
EVENT   =   18

    SECTION code

        ;*****************************************************************
        ;
        ; Handle and Dispatch DSP chip interrupts
        ;
        ;*****************************************************************
    XREF _HandleDspGet,_HandleDspPut
    .FREF   _ExitInterrupt,2    ;this function removes two bytes
    XREF _EnterInterrupt

    XDEF    DSPirq

DSPirq
    movem.l d0-d7/a0-a6,-(a7)   ;push all registers onto stack
    jsr     _EnterInterrupt     ;Increment Interrupt Semaphore
    move.b  DSP_ISR,d7          ;get the status word
    btst    #0,d7               ;recieve buffer full?
    beq.s   DSPirq01            ;no try transmit
    jsr     _HandleDspGet       ;handle get interrupt request
DSPirq01
    btst    #1,d7               ;test transit data register empty
    beq.s   DSPirq02            ;no, nothing more to try
    jsr     _HandleDspPut       ;handle put interrupt request
DSPirq02
    move.w  60(a7),d0               ;get status register
    andi.w  #0x0700,d0              ;mask off junk
    asr.w   #8,d0                   ;shift over 8 bits
    move.w  d0,-(a7)                ;push onto stack
    jsr     _ExitInterrupt      ;Check to see if swap is needed
    movem.l (a7)+,d0-d7/a0-a6   ;pop all registers from stack
    rte                         ;return from interrupt

    XDEF    _EnableDSPXmit
    ;*****************************************************************
    ;Enable the DSP transmit interrupt
    ;*****************************************************************
_EnableDSPXmit
    bset.b  #1,DSP_IRQCTL
    rts

    XDEF    _DisableDSPXmit
```

```
    ;*****************************************************************
    ;Disable the DSP transmit interrupt
    ;*****************************************************************
_DisableDSPXmit
    bclr.b  #1,DSP_IRQCTL
    rts

    XDEF    _CheckXMit
    ;*****************************************************************
    ; Returns TRUE if transmit register ready to accept character
    ;*****************************************************************
_CheckXMit
    move.b  DSP_ISR,d0          ;get isr register
    ext.w   d0                  ;convert to int
    and     #0x02,d0            ;mask off TXDE bit
    rts                         ;return

    XDEF    _CheckRxcv
    ;*****************************************************************
    ; Returns TRUE if recieve register is full
    ;*****************************************************************
_CheckRxcv
    move.b  DSP_ISR,d0          ;get isr register
    ext.w   d0
    and     #0x01,d0            ;mask off RXDF bit
    rts

    .FDEF   _PutDSP,4
    ;*****************************************************************
    ; This routine takes a long and stores it into the DSP
    ;*****************************************************************
_PutDSP
    MOVE.L  #0xFFD40009,A0      ;load address of DSP port into A0
    MOVE.L  %+4(A7),D0          ;data to save into D0
    MOVEP.L D0,0(A0)            ;move data to DSP
    MOVE.L  (A7)+,A0            ;get return address
    ADDQ.L  #4,A7               ;clean up Stack
    JMP (A0)                    ;return

    XDEF    _GetDSP
    ;*****************************************************************
    ; This routine returns a long from the DSP
    ;*****************************************************************
_GetDSP
    MOVE.L  #0xFFD40009,A0      ;load address of DSP port into A0
    MOVEP.L 0(A0),D0            ;move DSP data to D0
    ASL.L   #8,D0               ;sign extend data
    ASR.L   #8,D0
    RTS                         ;return data
```

```
/*
** defines for the HOST INTERFACE on the 56000
**
** HOST
*/

#ifndef HOST__H
#define HOST__H

//---------------------------------------------------------------
// port addresses for DSP
//---------------------------------------------------------------

#define DSP_BASE    0xffd40001
#define DSP_IRQCTL  (DSP_BASE + 0)  //interrupt control register
#define DSP_CMDVEC  (DSP_BASE + 2)  //command vector register
#define DSP_ISR     (DSP_BASE + 4)  //interrupt status register
#define DSP_IRQVEC  (DSP_BASE + 6)  //interrupt vector register
#define DSP_DATA    (DSP_BASE + 8)  //start of data register
#define DSP_DATAH   (DSP_BASE + 10)
#define DSP_DATAM   (DSP_BASE + 12)
#define DSP_DATAL   (DSP_BASE + 14)


/*************************************
**  Interrupt control Reg bit defines **
*************************************/

#define ICR_RREQ    1       /*  Controls HREQ pin for Host Receive data transfers   */
#define ICR_TREQ    2       /*  Controls HREQ pin for Host Transmit data transfers  */
#define ICR_HF0     8       /*  Host control flag 0, Used to send flags to DSP      */
#define ICR_HF1     0x10    /*  Host control flag 1, Used to send flags to DSP      */
#define ICR_MODE_BITS   0x60    /*  Used to control DMA as follows: */
#define ICR_MODE_IRQ    0   /*  No dma, interrupt mode on   */
#define ICR_MODE_24BIT  0x20    /*  DMA mode, 24 bits   */
#define ICR_MODE_16BIT  0x40    /*  DMA mode, 16 bits   */
#define ICR_MODE_8BIT   0x60    /*  DMA mode, 8 bits    */
#define ICR_INIT    0x80        /*  Initialize Host Interface according to control bits */

/***********************************
** Command Vector Reg bit defines  **
***********************************/

#define CVR_HV_BITS 0x1f    /*  bit mask for command vector */
#define CVR_HC      0x80    /*  Host command bit, set by HOST, cleared by DSP   */

/*************************************
**  Interrupt Status Reg bit defines  **
*************************************/

#define ISR_RXDF    1       /*  Recieve Data Register Full  */
#define ISR_TXDE    2       /*  Transmit Data Register Empty    */
#define ISR_TRDY    4       /*  Transmit Data Register Ready    */
#define ISR_HF2     8       /*  Indicates state of flag 2 on DSP side   */
#define ISR_HF3     0x10    /*  Indicates state of flag 3 on DSP side   */
#define ISR_DMA     0x40    /*  DMA status  */
#define ISR_HREQ    0x80    /*  indicates state of external HREQ pin    */

#endif
```

```
//
//
// Code for talking to servo controller card
//
// comm.h
//

#ifndef COMM___H
#define COMM___H

#include "erorcode.h"
#include "task.h"

//-----------------------------------------------------------------
// Host port commands
//-----------------------------------------------------------------

    /*
    ** recieve commands
    */

#define DSPCOM_SET_KP                   0   /*  set proportional gain   */
#define DSPCOM_SET_KV                   1   /*  set differential gain   */
#define DSPCOM_SET_KI                   2   /*  set integral gain       */
#define DSPCOM_SET_KVFF                 3   /*  set feed forward velocity gain  */
#define DSPCOM_SET_KAFF                 4   /*  set feed forward acceleration gain  */
#define DSPCOM_SET_OFFSET               5   /*  set servo offset    */
#define DSPCOM_SET_GOAL                 6   /*  set desired position    */
#define DSPCOM_SET_MAXVEL               7   /*  set maximum velocity    */
#define DSPCOM_SET_SCURVE               8   /*  set time to get up to speed */
#define DSPCOM_DO_MOVE                  9   /*  execute profile generator and do move   */
#define DSPCOM_MOVE_ABORT               10  /*  abort current move  */
#define DSPCOM_SET_PHASE_DETECT_POL     11  /*  set the polarity of the phase detector  */
#define DSPCOM_SET_SAMPLE_RATE          12  /*  Set servo sample rate       */
#define DSPCOM_ZERO_INTEGRATOR          13  /*  reset the integrator accumulator    */
#define DSPCOM_SET_INT_MODE             14  /*  for firmware v2.10 and greater  */
#define DSPCOM_START_JOG                15  /*  set up and start a jog for tuning purposes  */
#define DSPCOM_END_JOG                  16  /*  cancel the jog  */
#define DSPCOM_SET_JOG_PARAM            17  /*  set joging parameter        */
#define DSPCOM_CLEAR_INTERRUPT          18  /*  clears the beam interrupt flag and restarts the
servo   */
#define DSPCOM_ENABLE_SERVO_FUNCTIONS   19  /*  enable/disable the servo loop   */
#define DSPCOM_SET_INT_LIMITS           20  /*  set intergrator limits  */
#define DSPCOM_ZERO_COUNT               21  /*  zero the position counter   */
#define DSPCOM_SET_MAX_FOLLOW           22  /*  set the maximum following error */
#define DSPCOM_SET_FOL_GAIN             23  /*  set the following error DAC gain    */
#define DSPCOM_HOME_SETUP               24  /*  load parameters for home command    */
#define DSPCOM_DO_HOME                  25  /*  go to "home" position               */
#define DSPCOM_RESTORE_SETTINGS         26  /*  restore settings from NV rom    */
#define DSPCOM_SAVE_SETTINGS            27  /*  save settings into NV rom       */
#define DSPCOM_SET_INTERRUPTS           28  /*  enable/disable interrupts       */
#define DSPCOM_TRIGGER_SERVO            29  /*  set trigger flag true   */
#define DSPCOM_SET_INPOS_THRESH         30  /*  set in position threshold   */
#define DSPCOM_SET_BUMP                 31  /*  set the bump threshold  */
#define DSPCOM_CLEAR_FLAG               32  /*  clear step flag */
#define DSPCOM_RECORD_LINPOS            33  /*  record current linear position  */
#define DSPCOM_SET_FREQUENCY            34  /*  set frequency of internal sin oscilator */
#define DSPCOM_SET_FILTERFREQ           35  /*  set the cutoff freq of lp filter    */
#define DSPCOM_SET_AMPLITUDE            36  /*  set the amplitude of the oscilator  */
#define DSPCOM_SET_PHASE                37  /*  set phase of cosine output  */
#define DSPCOM_SET_DITHERMODE           38  /*  set mode of dither  */
#define DSPCOM_SET_ENCODERPITCH         39  /*  set the encoder pitch   */
#define DSPCOM_SET_RUNOUTLUT            40  /*  select default/user lut */
#define DSPCOM_SET_HANDSHAKE            41  /*  set handshake line for seagate thingy   */
#define DSPCOM_START_HISTOGRAM          42  /*  start histogram aquisition  */
#define DSPCOM_SET_NOTCH_FREQ           43  /*  set frequency of notch filter   */
```

```
#define DSPCOM_SET_NOTCH_Q              44  /*  set notch filter Q  */
#define DSPCOM_SET_NOTCH_D              45  /*  set notch filter depth  */
#define DSPCOM_SET_ANAL                 46  /*  set DSP analysis input  */
#define DSPCOM_SET_PESSCALE             47  /*  set scale factor for PES    */
#define DSPCOM_SET_PESLIMIT             48  /*  set limit for PES   */
#define DSPCOM_RESET_PES                49  /*  reset counter for PES   */
#define DSPCOM_SET_PESMODE              50  /*  set mode of PES counter */
#define DSPCOM_WRITE_MEM                51  /*  write data word to DSP memory   */
#define DSPCOM_SETGOALANDMOVE2          52  /*  set goal and move to absolute position  */

/*
** send commands
*/

#define DSPCOM_READ_POSITION            256         /*  read current position   */
#define DSPCOM_READ_GOAL                257 /*  read back the desired goal  */
#define DSPCOM_READ_SCURVE_TIME         258 /*  read back SCURVE time   */
#define DSPCOM_READ_MAX_VEL             259 /*  read back maximum velocty  */
#define DSPCOM_READ_KP                  260 /*  read back proportional gain     */
#define DSPCOM_READ_KV                  261 /*  read back differential gain     */
#define DSPCOM_READ_KI                  262 /*  read back integral gain         */
#define DSPCOM_READ_KVFF                263 /*  read back feed forward velocity */
#define DSPCOM_READ_KAFF                264 /*  read back feed forward acceleration */
#define DSPCOM_READ_OFFSET              265 /*  read back servo offset          */
#define DSPCOM_SERVO_STATUS             266 /*  read back servo card status     */
#define DSPCOM_READ_PHASE_DETECT_POL    267 /*  read the phase detector polarity    */
#define DSPCOM_GET_SAMPLE_RATE          268 /*  read the servo sample rate  */
#define DSPCOM_GET_INT_MODE             269 /*  for firmware v2.10 and greater  */
#define DSPCOM_READ_DAC_LEV             270 /*  reads the level of the Motor DAC    */
#define DSPCOM_GET_INT_LIMITS           271 /*  read the integrator limits  */
#define DSPCOM_GET_MAX_FOLLOW           272 /*  read the maximum following error    */
#define DSPCOM_GET_FOL_GAIN             273 /*  get the following error DAC gain    */
#define DSPCOM_GET_SN                   274 /*  get serial number and revision      */
#define DSPCOM_GET_HOME_PARAMS          275 /*  get home parrameters    */
#define DSPCOM_GET_INTERRUPTS           276 /*  get interrupt settings  */
#define DSPCOM_READ_PMEM                277 /*  for debug, read program memory  */
#define DSPCOM_READ_YMEM                278 /*  for debug, read y data memory   */
#define DSPCOM_READ_XMEM                279 /*  for debug, read x data memory   */
#define DSPCOM_READ_CALDIST             280 /*  read calibration distance   */
#define DSPCOM_READ_JOG_PARAMS          281 /*  read jogging parameters     */
#define DSPCOM_READ_INPOSTHRESH         282 /*  read in position threshold  */
#define DSPCOM_GET_BUMP                 283 /*  read the bump threshold */
#define DSPCOM_GET_CONFIG               284 /*  dump config parameters  */
#define DSPCOM_GET_LINEAR_ENDPOINT      285 /*  get recorded endpoint   */
#define DSPCOM_GET_FREQUENCY            286 /*  get frequency of internal oscilator */
#define DSPCOM_GET_MAGNITUDE            287 /*  get magnitude of position diviation */
#define DSPCOM_GET_FILTFREQ             288 /*  get the filter cutoff frequency */
#define DSPCOM_GET_AMPLITUDE            289 /*  get the amplitude of the oscilator  */
#define DSPCOM_GET_PHASE                290 /*  get phase of cosine output  */
#define DSPCOM_GET_DITHERMODE           291 /*  get mode of dither oscilator    */
#define DSPCOM_GET_ENCODERPITCH         292 /*  get the encoder pitch       */
#define DSPCOM_GET_RUNOUTCOMP           293 /*  get runout compensation table   */
#define DSPCOM_GET_RUNOUTLUT            294 /*  get runout comp table selection */
#define DSPCOM_GET_HISTOGRAM            295 /*  get histogram data  */
#define DSPCOM_GET_NOTCH_FREQ           296 /*  get notch filter frequency  */
#define DSPCOM_GET_NOTCH_Q              297 /*  get notch filter Q  */
#define DSPCOM_GET_NOTCH_D              298 /*  get notch filter depth  */
#define DSPCOM_GET_PESSCALE             299 /*  get scale factor for PES    */
#define DSPCOM_GET_PESLIMIT             300 /*  get limit for PES   */
#define DSPCOM_GET_FPGAID               301 /*  get id for DSP gate array   */
#define DSPCOM_GET_OTHER_CONFIG         302 /*  get jog/home params */
#define DSPCOM_GET_DATA_STRUCT          303 /*  get data structure address and size */
#define DSPCOM_GET_MAGNITUDE1           304 /*  get magnitude of spetrum filter */

/*
**  WriteDSPMem modes
```

```
*/

#define DSPCOM_PMEM 0
#define DSPCOM_XMEM 1
#define DSPCOM_YMEM 2

/*
** PES modes
*/

#define DSPCOM_PESMODE_NORMAL           0
#define DSPCOM_PESMODE_TEST             1

/*
** Get Data Buffer Commands
*/

#define DSPCOM_SET_CONFIG               512 /*  recieve parameter dump  */
#define DSPCOM_SET_RUNOUTCOMP           513 /*  recieve runout compensation table   */


/*
**  Get Serial Number defines
*/

#define COMM_VERSION    1
#define COMM_SN         0

/*
** jogging defines
*/

#define JOG_PARAM_DIRECTION         0x01    /*  jog in positive direction   */
#define JOG_PARAM_MODE_NORETRACE    0x02    /*  if set, does not retrace quickly, just reverses
*/

#define JOGPARAM_START  0
#define JOGPARAM_END    1
#define JOGPARAM_INDEX  2
#define JOGPARAM_MODE   3
#define JOGPARAM_DWELL  4

typedef  struct  {
    long  start;          //  Where the jog is going to start
    long  end;            //  Where the jog is going to end
    long  index;          // Distance to move on each step
    long  mode;           // Jogging Mode
    long  time;           // Dwell time between indexes
}JOGPARAMS;

/*
**  Clear Interrupt Function Values
*/

#define CLEAR_PROTECT   0   /*  clear progection interrupt  */
#define CLEAR_LIMIT_M   1   /*  clear LIMIT Minus       */
#define CLEAR_BEAM      2   /*  clear BEAM interrupt    */
#define CLEAR_LIMIT_P   3   /*  clear LIMIT Plus        */
#define CLEAR_AMP_FAULT 4   /*  claer amplifier fault   */
#define CLEAR_FOLLOW    5   /*  clear Fatal Follow Error    */
#define CLEAR_GENERAL   6   /*  clear general servo fault   */
#define CLEAR_BUMP      7   /*  clear bumped error      */

#define CLEAR_MAX       CLEAR_BUMP

/*
```

```
** hardware interrupt vector numbers
*/

#define HDWIRQ_PROTECT      0   /*  vector value for protect interrupt  */
#define HDWIRQ_LIMIT_M      1
#define HDWIRQ_BEAM         2
#define HDWIRQ_LIMIT_P      3
#define HDWIRQ_AMP_FAULT    4
#define HDWIRQ_HOME         5
#define HDWIRQ_TRIGGER      6
#define HDWIRQ_CTS          7

/*
** hardware interrupt bit defines
*/
#define HDWIRQ_ENABLE   0x01    /*  Interrupt Enabled   */
#define HDWIRQ_DISPID   0x02    /*  Enable PID disable on interrupt */
#define HDWIRQ_IRQSTAT  0x04    /*  Interrupt Status Read Only      */
#define HDWIRQ_ABORT    0x08    /*  causes move to abort on interrupt   */
#define HDWIRQ_MOVE     0x10    /*  causes move complete when turned on */
                                /*  disables interrupt exception message    */

#define HDWIRQ_SET  0x08    /*  or this with vector number if bits are to be set    */

/*
** Servo Functions
*/
#define ENABLE_PID          (0l)    /*  enable the PID loop     */
#define ENABLE_INTEGRATOR   (1l)    /*  enable the PID integrator   */
#define ENABLE_FATAL_FOLLOW (2l)    /*  enable the fatal following error    */
#define ENABLE_DITHER       (4l)    /*  enable the dither oscilator */
#define ENABLE_NOTCH        (5l)    /*  enable notch filter     */
#define ENABLE_PES          (6l)    /*  enable PES loop */
/*
** index parameter defs
*/
#define DWELL_TIME  0
#define INDEX_SIZE  1

/*
** home command parrameters
*/
    /*  Meaning of flags bits in HOMEPARAMS */
#define HOME_NONE           0   /*  there is no limit defined   */
#define HOME_LIMIT_PLUS     1   /*  limit on the plus limit switch  */
#define HOME_LIMIT_MINUS    2   /*  limit on the minus limit switch */
#define HOME_LIMIT_HOME     3   /*  limit on the home limit switch  */
#define HOME_LIMIT_DAC      4   /*  limit on DAC drive      */

#define HOME_LIMIT_MASK     0x0f

#define MOTOR_CURRENT_MASK  0x0ff0
#define MOTOR_CURRENT_SHIFT 4   /*  shift number 8 bits before oring    */

#define HOME_POSITIVE_DIR       0x1000  /*  home to the east    */

typedef  struct  {
    long flags;         //contains information about how to do home
    long init_velocity; //initial seek velocity used
    long init_dwell;    //dwell time after servo hits stop
    long backout_dist;  //distance to back out after stop was hit
    long finedwell;     //time to wait before doing fine seek
    long finevel;       //velocity to do fine seek at
    long zerodist;  //distance to move before zeroing servo
} HOMEPARAMS;
```

```
    /*  home commands   */
#define HOME_DOHOME     0       /*  performs home command   */
#define HOME_DOCAL      1       /*  performs callibrate command */


/*
** move functions
*/

#define MOVE_ABSOLUTE   0
#define MOVE_RELATIVE   1

/*
** Record Linear Position Functions
*/

#define LINEARSTAGE_RECORDHOME      0
#define LINEARSTAGE_RECORDRUN       1
#define LINEARSTAGE_INPOSENABLE     2


//
// status defs
//
#define SYSTEM_SRx1 1                   /*  standard sample rate    */
#define SYSTEM_SRx2 2                   /*  sample rate is half speed   */
#define BEAM_INTERRUPTED    0x4         /*  beam interrupted flag   */
#define INTEGRATOR_ENABLED  0x8         /*  integrator flag         */
#define SERVO_ENABLED       0x10        /*  servo enabled flag  */
#define LIMIT_PLUS_TRIPPED  0x20        /*  plus limit has been tripped */
#define LIMIT_MINUS_TRIPPED 0x40        /*  minus limit has been tripped    */
#define HOME_TRIPPED        0x80        /*  home sensor has been tripped    */
#define AMP_FAULT_TRIPPED   0x100       /*  amplifier fault has been tripped    */
#define TRIGGER_TRIPPED     0x200       /*  trigger has been tripped    */
#define PROTECTION_TRIPPED  0x400       /*  protection line has been tripped    */
#define CTS_TRIPPED         0x800       /*  CTS line tripped    */
#define FOLLOWING_ERROR     0x1000      /*  fatal following error   */
#define F_ERROR_ENABLED     0x2000      /*  fatal following error enabled   */
#define GENERAL_FAULT       0x4000      /*  Some general fault has occured  */
#define ENABLE_TRIGGER      0x8000      /*  Indicates profile generator will wait */
                                        /*  for a trigger before generating move    */
#define SERVO_IN_POSITION   0x10000     /*  indicates following error less than preset value
*/
#define DIS_INT_CMDVNEZ     0x20000     /*  disable integrator durring moves    */
#define BUMP_ERROR          0x40000     /*  bump Error Has Occured          */
#define SYS_DITHER_ENABLED  0x80000     /*  dither is enabled               */
#define MOVE_IDLE           0x100000    /*  indicates profile generator not active  */
#define DSPSTAT_NOTCHFILTER 0x200000    /*  indicates notch filter enabled  */

/*
** Definitions of the various events for type data member
*/

#define MOVE_SERVO  1       /*  this event moves the servo to position  */
#define DO_JOG      2       /*  this event jogs the motor back and forth    */
#define DODEBUG     3       /*  do a DEBUG event    */
#define MOVE_SERVO_REL  4   /*  move servo relative to current position */
#define DO_HOME_EVENT   5   /*  trying to find home position            */
#define MOVE_SERVO_INDEXED  6   /*  move the servo in an indexed fashion    */
#define SAVE_PARAMS     7
#define RESTORE_PARAMS  8
#define EVENT_ARMED     9   /*  indicates system waiting for a trigger  */

typedef struct  {
    long status;
```

```
    long type;
    long pending;
    long system[3];
}DSP_status;

//-----------------------------------------------------------------
//
// parameters for MICROE management
//
//-----------------------------------------------------------------

#define MICROE_AUTOCOMP_ON  1
#define MICROE_AUTOCOMP_OFF 0


//
//-----------------------------------------------------------------
// ACK defines
//-----------------------------------------------------------------
//

#define ACK     0x41434b
#define NAK     0x4e414b


//
// timeout defines
//

#define READ_DATA_TIMEOUT   100     //50 milisecond timeout
#define WRITE_DATA_TIMEOUT  100     //50 milisecond timeout


//-----------------------------------------------------------------
//comm class definitions
//-----------------------------------------------------------------

typedef  struct{
    long command;       //actually, we use only the first 24 bits
    long axis;          //servo axis being talk to
    long func;          //command sub function
    long aux[2];        //aux data to be sent
}CommandFrame;

extern  int telSendDataCommand(CommandFrame);        //private functions for host communications
extern  int telGetDataCommand(CommandFrame cmd,long *d,int n);
extern  void InitServoPort(void);

/*-----------------------------------------------------------------
** PES functions
**-----------------------------------------------------------------*/

extern  int SetPesScale(int axis,long scale);
extern  int GetPesScale(int axis,long *scale);
extern  int SetPesLimit(int axis,long limit);
extern  int GetPesLimit(int axis,long *limit);
extern  int ResetPesCounter(int axis);
extern  int SetPesMode(int axis,long mode);
extern  int GetFPGAID(int axis,long *id);

/*-----------------------------------------------------------------
** filter parameter functions
**-----------------------------------------------------------------*/


extern  int Read_Kv(int axis,long *kv);
```

```
extern int Read_Kp(int axis,long *kp);
extern int Read_Ki(int axis,long *ki);
extern int Read_Kvff(int axis,long *kvff);
extern int Read_Kaff(int axis,long *kaff);
extern int Read_Offset(int axis,long *offset);
extern int Read_IntThresh(int axis,long *thresh);
extern int Read_DacLimit(int axis,long *limit);
extern int GetIntegratorMode(int axis,long *v);
extern int Read_MaxFollowingError(int axis,long *limit);
extern int Read_DacLev(int axis,long *dacval);
extern int Read_IntegratorLimits(int axis,long *limit);
extern int ReadNotchQ(int axis,long *q,int fn);
extern int ReadNotchFreq(int axis,long *f,int fn);
extern int ReadNotchDepth(int axis,long *d,int fn);

extern int Set_Kp(int axis,long kp);
extern int Set_Kv(int axis,long kv);
extern int Set_Ki(int axis,long ki);
extern int Set_Kvff(int axis,long kvff);
extern int Set_Kaff(int axis,long kaff);
extern int Set_Offset(int axis,long offset);
extern int Set_IntThresh(int axis,long thresh);
extern int Set_DacLimit(int axis,long limit);
extern int SetIntegratorMode(int axis,long v);
extern int Set_MaxFollowingError(int axis,long limit);
extern int Set_IntegratorLimits(int axis,long limit);
extern int ZeroIntegrator(int axis);
extern int EnableServoFunctions(int axis,long func,long flag);
extern int SetNotchQ(int axis,long q,int fn);
extern int SetNotchFreq(int axis,long f,int fn);
extern int SetNotchDepth(int axis,long d,int fn);

/*-------------------------------------------------------------------------
** Jogging Functions
**-------------------------------------------------------------------------*/



extern int StartJog(int axis);
extern int EndJog(int axis);
extern int GetJogParams(int axis,JOGPARAMS *param);
extern int SetJogParams(int axis,JOGPARAMS param);


/*-------------------------------------------------------------------------
** Interrupt functions
**-------------------------------------------------------------------------*/

/*  standard C interface    */


extern int ClearInterrupt(int axis,long func);
extern int GetIrqParams(int axis,long vector,long *val);    /*  get interrupt parameter for irq
vector  */
extern int SetIrqParams(int axis,long vector,long val);     /*  set interrupt parameter for irq
vector  */


/*-------------------------------------------------------------------------
** Motion Functions
**-------------------------------------------------------------------------*/

extern int SetIndexParams(int axis,long val,int index);     /*  set indexed move params */
extern int DoMoveIndexed(int axis);
extern int Set_SCurve(int axis,long stime);
extern int Set_MaxVel(int axis,long bits_per_deicmillisec);
```

```
extern int Do_move(int axis,int move_mode);
extern int DoMoveRel(int axis);
extern int AbortMove(int axis);
extern int Set_Goal(int axis,long Goal);
extern int Read_Position(int axis,long *);
extern int Read_Goal(int axis,long *Goal);
extern int Read_SCurve(int axis,long *stime);
extern int Read_MaxVel(int axis,long *bits_per_decimillisec);
extern int GetIndexParams(int axis,long *val,int index);    /*  set indexed move params */
extern int SetGoalAndMove(int axis,long Goal);

/*---------------------------------------------------------------------
** Homing Functions
**---------------------------------------------------------------------*/

/*  standard C interface    */

extern int Do_Home(int axis);
extern int Do_Calibrate(int axis);
extern int Set_HomeParams(int axis,HOMEPARAMS HomeParams);  /*  set home parameters */
extern int Get_HomeParams(int axis,HOMEPARAMS *HomeParams); /*  get home parameters */
extern int RecordLinearPosition(int axis,long func,long aux);

//---------------------------------------------------------------------
// servo card status
//---------------------------------------------------------------------

extern int ServoBusy(int axis);         /*  determines if servo card is too busy to talk    */
extern int GetVersionNumber(int axis,char *s,int n);
extern int ServoStatus(int axis,DSP_status *stat);

//---------------------------------------------------------------------
// bode plot functions
//---------------------------------------------------------------------

extern int SetAnalMode(int axis,long mode);
extern int SetFrequency(int axis,long v);
extern int SetFilterBandwidth(int axis,long w,long n);
extern int GetFrequency(int axis,long *v);
extern int GetFilterBandwidth(int axis,long *w,long n);
extern int GetMagnitude(int axis,long *a,long n);
extern int GetMagnitude1(int axis,long *real,long *imag,long n);
extern int GetAmplitude(int axis,long *a);
extern int SetAmplitude(int axis,long a);
extern int SetPhase(int axis,long a);
extern int GetPhase(int axis,long *a);
extern int SetDitherMode(int axis,long mode);
extern int GetDitherMode(int axis,long *mode);

//---------------------------------------------------------------------
//
// System analysis functions
//
//---------------------------------------------------------------------

extern int GetHistogram(int axis,long *buff);
extern int StartHistogram(int axis,long nsamples);

//---------------------------------------------------------------------
//misc function prototypes
//---------------------------------------------------------------------
extern int GetEncoderPtich(int axis, long *val);
extern int SetEncoderPitch(int axis, long val);

extern int ZeroCount(int axis);
extern int Set_FollowGain(int axis,long gain);
```

```
extern int SetDacOffset(int axis,long beam);
extern int Set_AmpOffset(int axis,long offset);
extern int SaveSettings(int axis);
extern int RestoreSettings(int axis);
extern int SetInPositionThreshold(int axis,long val);
extern int Set_SineOffset(int axis,long sine_offset);
extern int Set_CosOffset(int axis,long cosine_offset);
extern int Set_PhaseDetectorPolarity(int axis,long pol);
extern int Set_SampleRate(int axis,long rate);
extern int SetFrequency(int axis,long freq);
extern int SetPortBits(int axis,long val);
extern int ClearPortBits(int axis,long val);
extern int SetBump(int axis,long bump);
extern int ClearFlag(int axis);                 /*  debug function  */

extern int Get_Calibration(int axis,long *val);
extern int Read_FollowGain(int axis,long *gain);
extern int ReadDacOffset(int axis,long *beam);
extern int Read_SampleRate(int axis,long *rate);
extern int Read_PhaseDetectorPolarity(int axis,long *pol);
extern int Read_CosOffset(int axis,long *cosine_offset);
extern int Read_SineOffset(int axis,long *sine_offset);
extern int GetInPositionThreshold(int axis, long *val);
extern int Get_ADCVal(int axis,int chan,long *val);         /*  get ADC value       */
extern int Read_AmpOffset(int axis,long *offset);
extern int ReadFrequency(int axis,long *freq);
extern int GetPort(int axis,long *val);
extern int GetOutPort(int axis,long *val);
extern int ReadBump(int axis,long *bump);

extern int SetRunoutCompTable(int axis,long *data);
extern int GetRunoutCompTable(int axis,long *data);
extern int SetLookupTableSelection(int axis,long table);
extern int GetLookupTableSelection(int axis,long *table);

//
//-------------------------------------------------------
// Micro E functions
//
//-------------------------------------------------------

extern int SetMicroEAutoComp(int flag);
extern int GetMicroEAutoComp(int *flag);

//-------------------------------------------------------
// Debug Functions
//-------------------------------------------------------

extern int SetGoalReg(int axis,long Goal);
extern int ReadPmem(int axis,long *v, long addr);
extern int ReadXmem(int axis,long *v, long addr);
extern int ReadYmem(int axis,long *v, long addr);
extern int WriteDSPMem(int axis,long addr,long d,int func);
extern int GetDataStructures(int axis,int type,long *v);

//-------------------------------------------------------
// system functions
//-------------------------------------------------------

extern void InitSaveRestore(void);

extern void ClearWaitForLinearHome(void);
extern void ClearWaitForLinearRun(void);
extern void ClearWaitForRotaryHome(void);
extern void ClearWaitForRotaryCalibrate(void);
extern void ClearMoveDoneSemaphore(void);
```

```
extern int WaitForRotaryMove(int timeout);
extern int WaitForRotaryHome(int timeout);
extern int WaitForRotaryCalibrate(int timeout);

extern int WaitForLinearHome(int timeout);
extern int WaitForLinearRun(int timeout);

extern TSemaphore *SpinDoneSemaphore;   //indicates when spindle profile done

#endif  //COMM__H
```

```
#include <stdio.h>
#include "cio.h"
#include "task.h"
#include <curses.h>
#include <stdarg.h>
#include "crash.h"
#include <string.h>
#include "rs232.h"

#pragma region("ram = nonvol")
CRASH crashdata;
void BombWait(void);

void InitCrash(void)
{
    if(crashdata.Magic != 0x12345678)
    {
        crashdata.Magic = 0x12345678;
        crashdata.flag = 0;
    }
}

int GetCrash(void)
{
    int Handle;
    char *s = new char[256];
    int c;

    if((Handle = Open("D:CRASH",WRITE_ONLY)) < 0)
        return Handle;
    c = sprintf(s,"Exception=%d\n",crashdata.except);
    Write(Handle,s,c);
    c = sprintf(s,"Status Reg=%04x\n",crashdata.sr);
    Write(Handle,s,c);
    c = sprintf(s,"Instruction=%04x\n",crashdata.instruction);
    Write(Handle,s,c);
    c = sprintf(s,"Access Address=%08lx\n",crashdata.access);
    Write(Handle,s,c);
    c = sprintf(s,"Program Counter=%08lx\n",crashdata.adr);
    Write(Handle,s,c);
    c = sprintf(s,"Function Code=%04x\n",crashdata.function);
    Write(Handle,s,c);
    if(crashdata.task)
    {
        c = sprintf(s,"-----Task Information-----\n");
        Write(Handle,s,c);
        c = sprintf(s,"TASK:%s\n",crashdata.task->name);
        Write(Handle,s,c);
        c = sprintf(s,"Prioity:%d\n",crashdata.task->priority);
        Write(Handle,s,c);
        c = sprintf(s,"Stack Top:%08lx\n",crashdata.task->stacktop);
        Write(Handle,s,c);
    }
    c = sprintf(s,"-----68000 Registers-----\n");
    Write(Handle,s,c);
    c = sprintf(s,"D0:%08lx   A0:%08lx\n",crashdata.d0,crashdata.a0);
    Write(Handle,s,c);
    c = sprintf(s,"D1:%08lx   A1:%08lx\n",crashdata.d1,crashdata.a1);
    Write(Handle,s,c);
    c = sprintf(s,"D2:%08lx   A2:%08lx\n",crashdata.d2,crashdata.a2);
    Write(Handle,s,c);
    c = sprintf(s,"D3:%08lx   A3:%08lx\n",crashdata.d3,crashdata.a3);
    Write(Handle,s,c);
    c = sprintf(s,"D4:%08lx   A4:%08lx\n",crashdata.d4,crashdata.a4);
    Write(Handle,s,c);
    c = sprintf(s,"D5:%08lx   A5:%08lx\n",crashdata.d5,crashdata.a5);
```

```
    Write(Handle,s,c);
    c = sprintf(s,"D6:%08lx   A6:%08lx\n",crashdata.d6,crashdata.a6);
    Write(Handle,s,c);
    c = sprintf(s,"D7:%08lx\n",crashdata.d7);
    Write(Handle,s,c);
    Close(Handle);
    delete[] s;
    memset(&crashdata,0,sizeof(CRASH));
    return 0;
}

extern "C" void p_error(int except,void *stack,long d0,long d1,long d2,long d3,
                        long d4, long d5, long d6, long d7,long a0, long a1,
                        long a2, long a3, long a4, long a5, long a6,...)
{
    switch(except)
    {
        case 2:
        case 3:
            if(crashdata.flag == 0)
            {
                crashdata.function = *((int *)stack)++;
                crashdata.access = *((long *)stack)++;
                crashdata.instruction = *((int *)stack)++;
                crashdata.sr = *((int *)stack)++;
                crashdata.adr = *((long *)stack)++;
            }
            break;
        default:
            if(crashdata.flag == 0)
            {
                crashdata.sr = *((int *)stack)++;
                crashdata.adr = *((long *)stack)++;
            }
            break;
    }
    if(crashdata.flag == 0)
    {
        crashdata.d0 = d0;
        crashdata.d1 = d1;
        crashdata.d2 = d2;
        crashdata.d3 = d3;
        crashdata.d4 = d4;
        crashdata.d5 = d5;
        crashdata.d6 = d6;
        crashdata.d7 = d7;
        crashdata.a0 = a0;
        crashdata.a1 = a1;
        crashdata.a2 = a2;
        crashdata.a3 = a3;
        crashdata.a4 = a4;
        crashdata.a5 = a5;
        crashdata.a6 = a6;
        crashdata.except = except;
        crashdata.flag = 1;
        if((except == 2) || (except == 3))
            crashdata.task = CurrentTask;
        else
            crashdata.task = 0;

        *((volatile int *)0xffcc0000) = ~except;
    }
    BombWait();
}

void BombWait(void)
```

```
{
    int sr = EnterCritical();   //disble interrupts
    while(1)
    {
        //attempt to do everything posible to keep system alive
        volatile int a;
        a = 10000;
        while(a--);
        *((volatile char *)(BIT_RESET_COMMAND)) = 0x40; //toggle watchdog
        a = 10000;
        while(a--);
        *((volatile char *)(BIT_SET_COMMAND)) = 0x40;   //toggle watchdog
    }

}
```

```
[
    disregard white space
    ~case sensitive
    ~line numbers
    lexeme {hex number}
    pointer input
    ~near functions
    parser file name = "#.cpp"
]

eof = 0
letter = 'a-z' + 'A-Z' + '_' + '.' + ':'
digit = '0-9'

(long) hexdigit
    -> '0-9':c      =c - '0';
    -> 'a-f' + 'A-F':c =9 + (c & 0x07);


(long) hex number
    -> "0X",hex num:v           =v;

(long) hex num
    -> hexdigit:c   =c;
    -> hex num:v, hexdigit:c    =(v<<4) + c;

(long) decimal number
    -> digit:c = c-'0';
    -> decimal number:v,digit:c =(v*10) + c-'0';

(char *)name
    ->name string                       =identify_name();

(void)name string
    ->letter:c                          =ins(c);
    ->name string, letter+digit:c       =pcn(c);

(void)white space
    -> '\t' + '\r' + ' ' + '\n'

//(void)space
//  -> ' '
//  -> space,' '

(int)consol $
    -> cmds:v,eof   =v;
    -> eof          =0;

(int)cmds
    -> consol commands:v        =v;

(int) consol commands
    -> "DUMP",'(', hex number:start,',',hex number:stop,')' =Dump(start,stop,0);
    -> "PEEKB",'(',hex number:address,')'   =PeekB(address);
    -> "PEEKW",'(',hex number:address,')'   =PeekW(address);
    -> "PEEKL",'(',hex number:address,')'   =PeekL(address);
    -> "POKEB",'(',hex number:address,',',hex number:value,')'  =PokeB(address,value);
    -> "POKEW",'(',hex number:address,',',hex number:value,')'  =PokeW(address,value);
    -> "POKEL",'(',hex number:address,',',hex number:value,')'  =PokeL(address,value);
    -> "CRASH" =DoCrash(0x12345678l);
    -> "GETCRASH" = GetCrash();
    -> "TYPE",'(',name:s,')'    =DoType(s);
    -> "TASKLIST"               =DoTaskList();
    -> "TASK",'(',name:s,')'    =DoTaskDump(s);
    -> "FIND",'(',name:s,')'    =DoFind(s);
    -> "DISPSEM",'(',name:s,')'                     =DoDispSem(s);
```

```
//  -> "TESTLUT"                                  =DoLutTest();
    -> "HOSTREG"                                     =DoDspHostReg();
    -> "VERSION"                                     =DoVersion();
//  -> "QUADKNOB"                                         =DoQuadKnob();
    -> "EXERSIZE",'(',decimal number:d,')'               =DoExersize(d);
    -> "ZAPCRASH"                                        =DoInitCrash();
    -> "ACTIONSTATUS"                                    =DoActionStatus();
    -> "SERVOSTATUS"                                     =DoServoStatus();
    -> "SPINSTANDSTATE"                                  =DoSpinstandState();
    -> "TESTRAMP",'(',decimal number:d,')'               =DoRampTest(d);
    -> "FDUMP",'(',name:s,')'                            =DoFDump(s);
    -> "DIR"                                             =DoDir();
    -> "I2C",'(',decimal number:c,',',decimal number:s,')'  ={
                                                             I2CSetControlStage(c,s);
                                                             return 0;
                                                             }
    -> "DUMPY",'(', hex number:start,')'    =DumpY(start);
    -> "DUMPX",'(', hex number:start,')'    =DumpX(start);
    -> "DEBUG"                                           =DoDebug();

{
/*************************************************************************/
/*                                                                       */
/*    A C T I O N   C O D E                                              */
/*    Son of Accutrac Consol Code                                    */
/*    Copyright (c) 1997 by Teletrac Inc.                                */
/*************************************************************************/

#include "cio.h"
#include <string.h>
#include <ctype.h>
#include "task.h"
#include "queue.h"
#include "crash.h"
#include "ramdisk.h"
#include "rs232.h"
#include "timer.h"
#include "spincopr.h"
#include "servo.h"
#include "spindle.h"
#include "serlprot.h"
#include "frontled.h"
#include "i2c.h"
#include "acutrac.h"
#include "scfig.h"
#include "global.h"
#include "spinchip.h"

#define SYNTAX_ERROR (DoSynTaxError(PCB.error_message))
#define PARSER_STACK_OVERFLOW (DoStackOverflow())
#define REDUCTION_TOKEN_ERROR ((DoReductionTokenError())


static char CONFIGname[82];
static int  CONFIGname_length;
static char *ErrorMessage = {"ERROR:"};
extern int ConsolHandle;
static char OString[512];


static void ins(int c)
{
    /*  Init and Put char to Name   */
    CONFIGname[0] = c;
    CONFIGname_length = 1;
}
```

```
static void pcn(int c)
{
    CONFIGname[CONFIGname_length++] = c;
}

static char *identify_name(void)
{
    CONFIGname[CONFIGname_length] = '\0';   /*  termnate name   */
    return(CONFIGname);
}

static void Task1(void)
{
    char *s = new char[256];
    int c;

    while(1)
    {
        c = sprintf(s,"TASK1:Printing out a really long string of characters:0:TASK1\r\n");
        if(Write(ConsolHandle,s,c) < 0)
            TDelete(0);
    }
}

static void Task2(void)
{
    char *s = new char[256];
    int c;

    while(1)
    {
        c = sprintf(s,"TASK2:Printing out a really long string of characters:2:TASK2\r\n");
        if(Write(ConsolHandle,s,c) < 0)
            TDelete(0);
        c = sprintf(s,"TASK2:Printing out a really long string of characters:1:TASK2\r\n");
        if(Write(ConsolHandle,s,c) < 0)
            TDelete(0);
    }
}

static int DoDebug(void)
{
    static TCB *t1,*t2;
    static int f=0;

    if(f)
    {
        TDelete(t1);
        TDelete(t2);
        f=0;
    }
    else
    {
        int sr;

        f = 1;
        t1 = CreateTask(Task1,2048,4,"Task1");
        sr = EnterCritical();
        ActiveTasks->Insert(t1);
        ExitCritical(sr);
        t2 = CreateTask(Task2,2048,3,"Task2");
        sr = EnterCritical();
        ActiveTasks->Insert(t2);
        ExitCritical(sr);
    }
```

```
    //print out the status of the RS232 buffers
//  int blen;
//  extern int mhandle;

//  int c;
//  blen = Status(mhandle,(char *)0,0l,SEND_STAT);
//  c = sprintf(OString,"Number of bytes in XMIT buffer =%d\r\n",blen);
//  Write(ConsolHandle,OString,c);
    return 0;
}

static int DoDir(void)
{
    //do a directory listing
    Ffblk fblk;
    int retval,c;

    retval = FindFirst("????????",0,&fblk);
    while(retval != RAMDISK_EOF)
    {
        c = sprintf(OString,"%8s....%ld\r\n",fblk.name,fblk.size);
        Write(ConsolHandle,OString,c);
        retval = FindNext(&fblk);
    }
    return 0;
}

static int ReadNotchFreq1(int a,long *v)
{
    return ReadNotchFreq(a,v,1);
}

static int ReadNotchQ1(int a,long *v)
{
    return ReadNotchQ(a,v,1);
}

static const int (* const func[])(int,long *d) =
{
    Read_Kp,
    Read_Kv,
    Read_Ki,
    Read_IntegratorLimits,
    ReadNotchFreq1,
    ReadNotchQ1
};

static const char *const PStrings[] = {
    "KP",
    "KV",
    "KI",
    "LIM",
    "NOTCHF",
    "NOTCHQ",
    NULL
};

static int DoRampTest(long d)
{
    long i;
    int c;
    char *s = new char[256];
    int error;

    for(i=0;i<d;++i)
```

```
    {
        ClearRampFlags();
        StageLoadHeads(STAGELOADHEAD_RAMPIN);
        if((error = WaitForRampIn(1000)) < 0)
        {
            c = sprintf(s,"Wait For RAMPIN 2 Timeout:%d\r\n",error);
            Write(ConsolHandle,s,c);
        }
        ClearRampFlags();
        StageLoadHeads(STAGELOADHEAD_RAMPOUT);
        if((error = WaitForRampOut(1000)) < 0)
        {
            c = sprintf(s,"Wait For RAMPOUT 2 Timeout:%d\r\n",error);
            Write(ConsolHandle,s,c);
        }
    }
    delete[] s;
    return 0;
}

static int DoSpinstandState(void)
{
    int status;
    int c;

    status = I2CGetStageStatus();
    c = sprintf(OString,"%4x::RAMP OUT:%d  RAMPIN %d  VAC:%d  AIR:%d  COMB:%d\r\n",status,
        (status & FPCOMMAND_RAMP_OUT)?1:0,(status & FPCOMMAND_RAMP_IN)?1:0,
        (status & FPCOMMAND_VAC)?1:0,(status & FPCOMMAND_AIR)?1:0,
        (status & FPCOMMAND_CORAL)?1:0);
    Write(ConsolHandle,OString,c);
    return 0;
}

static int DoActionStatus()
{
    int c,v,function,startflag;
    extern volatile int DisplayMode;

    c = sprintf(OString,"STATE=%d\r\n",AcutracGETSTATE(&v,&function,&startflag));
    Write(ConsolHandle,OString,c);
    c = sprintf(OString,"STATUS=%d\r\n",v);
    Write(ConsolHandle,OString,c);
    c = sprintf(OString,"FUNCTION=%d\r\n",function);
    Write(ConsolHandle,OString,c);
    c = sprintf(OString,"STARTFLAG=%d\r\n",startflag);
    Write(ConsolHandle,OString,c);
    c = sprintf(OString,"DisplayMode=%d\r\n",DisplayMode);
    Write(ConsolHandle,OString,c);
    return 0;
}

static int DoInitCrash(void)
{
    crashdata.Magic = 0;
    InitCrash();
    return 0;
}

static void DoSynTaxError(char *msg)
{
    sprintf(OString,"%s\r\n",msg);
    Write(ConsolHandle,OString,strlen(OString));
}

static void DoReductionTokenError(void)
```

```
{
    sprintf(OString,"\n\rReduction token error, line %d, column %d\r\n",(PCB).line, (PCB).column
);
    Write(ConsolHandle,OString,strlen(OString));
}

static void DoStackOverflow(void)
{
    sprintf(OString,"\r\nParser stack overflow, line %d, column %d\r\n",(PCB).line, (PCB).column
);
    Write(ConsolHandle,OString,strlen(OString));
}

int Dump(long start,long end,int mode)
{
    char *AS = new char[128];
    int i,index;
    int line=0;

    while(start < end)
    {
        index = sprintf(OString,"%08lx:",start);
        for(i=0;i<8;++i)
        {
            index += sprintf(&OString[index],"%04x ",*((int *)(start + 2*i)) );
            AS[2*i] = *((char *)(start + 2*i));
            AS[2*i+1] = *((char *)(start + 2*i+1));
        }
        for(i=0;i<16;++i)
        {
            if(!isprint(AS[i])) AS[i] = '.';
            else if(AS[i] < ' ') AS[i] = '.';
            else if(AS[i] >= 0x7f) AS[i] = '.';
        }
        AS[i] = 0;
        sprintf(&OString[index],"%s\r\n",AS);
        Write(ConsolHandle,OString,strlen(OString));
        ++line;
        if((line == 20) && mode)
        {
            Getc(ConsolHandle); //wait for any character
            line = 0;           //reset line count
        }
        start += 16;
    }
    delete[] AS;
    return 0;
}

static void DSPDump(long adr,long *b)
{
    int i;
    int c;
    int j;
    char *p;

    for(i=0;i<16;++i)
    {
        c = sprintf(OString,"%04lx:",adr + i * 8);
        for(j=0;j<8;++j)
            c += sprintf(&OString[c],"%06lx-",*b++ & 0x0ffffff);
        --c;
        c += sprintf(&OString[c],"\r\n");
        Write(ConsolHandle,OString,c);
    }
}
```

```
static int DumpX(long start)
{
    long *b = new long[128];
    int i;
    int error;
    int c;

    for(i=0;i<128;++i)
    {
        if((error = ReadXmem(0,&b[i],start + long(i)) ) < 0)
        {
            c = sprintf(OString,"Error:%d in ReadXmem\r\n",error);
            Write(ConsolHandle,OString,c);
            delete[] b;
            return 0;
        }
    }
    DSPDump(start,b);
    delete[] b;
    return 0;
}

static int DumpY(long start)
{
    long *b = new long[128];
    int i;
    int error;
    int c;

    for(i=0;i<128;++i)
    {
        if((error = ReadYmem(0,&b[i],start + long(i)) ) < 0)
        {
            c = sprintf(OString,"Error:%d in ReadXmem\r\n",error);
            Write(ConsolHandle,OString,c);
            delete[] b;
            return 0;
        }
    }
    DSPDump(start,b);
    delete[] b;
    return 0;
}

static int DoFDump(char *s)
{
    //---------------------------
    // do hex dump of file
    //---------------------------
    int handle;
    int size;
    int c;
    char *buff;
    long start,end;

    handle = Open(s,READ_ONLY);
    if(handle >= 0)
    {
        size = FileSize(handle);
        c = sprintf(OString,"File Size = %d\r\n",size);
        Write(ConsolHandle,OString,c);
        buff = new char[size];
        Read(handle,buff,(long)size);
        Close(handle);
        start = (long)buff;
```

```
        end = start + (long)size;
        Dump(start,end,1);
        delete[] buff;
    }
    return 0;
}

static int PeekB(long address)
{
    sprintf(OString,"PEEKB:%02x\r\n",*((char *)address) & 0x0ff);
    Write(ConsolHandle,OString,strlen(OString));
    return 0;
}

static int PeekW(long address)
{
    if(address & 1)
        sprintf(OString,"ERROR:Address MUST be Even\r\n");
    else
        sprintf(OString,"PEEKW:%04x\r\n",*((int *)address));
    Write(ConsolHandle,OString,strlen(OString));
    return 0;
}

static int PeekL(long address)
{
    if(address & 1)
        sprintf(OString,"ERROR:Address MUST be Even\r\n");
    else
        sprintf(OString,"PEEKL:%08lx\r\n",*((long *)address));
    Write(ConsolHandle,OString,strlen(OString));
    return 0;
}

static int PokeB(long address,long value)
{
    *((char *)address) = (char)value;
    return 0;
}

static int PokeW(long address,long value)
{
    if(address & 1)
    {
        sprintf(OString,"ERROR:Address MUST be Even\r\n");
        Write(ConsolHandle,OString,strlen(OString));
    }
    else
        *((int *)address) = (int)value;

    return 0;
}

static int PokeL(long address,long value)
{
    if(address & 1)
    {
        sprintf(OString,"ERROR:Address MUST be Even\r\n");
        Write(ConsolHandle,OString,strlen(OString));
    }
    else
        *((long *)address) = value;
    return 0;
}

static int DoCrash(long x)
```

```
{
    long *r = (long *)0x40001;

    r[6] = x+0x50;  //cause address error
    return (int)x;

}

static int DoType(char *s)
{
    int FHandle;
    int c;

    if((FHandle = Open(s,READ_ONLY)) < 0)
    {
        c = sprintf(OString,"ERROR: Could not Open %s\r",s);
        Write(ConsolHandle,OString,c);
        return 0;
    }
    while((c = Getc(FHandle)) != RAMDISK_EOF)
    {
        if(c == '\n')
            Putc(ConsolHandle,'\r');        //add in CR
        Putc(ConsolHandle,c);
    }
    Close(FHandle);
    return 0;
}

static int DoDispSem(char *name)
{
    TSemaphore *e;

    if(e = TSemaphore::FindSemaphore(name))
        e->PrintInfo();

    return 0;
}

static int DoFind(char *name)
{
    //locate the semaphore that a task is waiting at
    TSemaphore *e;
    TCB *t;

    t = MasterTaskList;
    while(t)
    {
        if(strcmp(t->name,name) == 0)
            goto exit;
        t = t->next;
    }
    int c = sprintf(OString,"Could not find %s in Semaphores\r\n",name);
    Write(ConsolHandle,OString,c);
    //search priority queue
    int sr = EnterCritical();
    int j = ActiveTasks->NumElem();
    int i;
    for(i=0;i<j;++i)
    {
        t = (TCB *)ActiveTasks->Get(i);
        if(strcmp(t->name,name) == 0)
        {
            c = sprintf(OString,"Task is Priority Queue\r\n");
            Write(ConsolHandle,OString,c);
        }
```

```
    }
    ExitCritical(sr);
    return 0;
exit:
    if(e = TSemaphore::FindTask(t))
        e->PrintInfo();
    return 0;
}


static int DoTaskList(void)
{
    TCB *l = MasterTaskList;            //get address of task list
    int c;
    int i=0;
    int count=0;

    while(l)
    {
        c = sprintf(OString,"%2d:%s::%d:Total Time:%ld\r\n",++i,l->name,l->priority,l->TotalTime
);
        Write(ConsolHandle,OString,c);
        l = l->list;
        ++count;
        if(count == 23)
        {
            c = sprintf(OString,"Press any Key to Continue");
            Write(ConsolHandle,OString,c);
            Getc(ConsolHandle);
            Putc(ConsolHandle,'\r');
            Putc(ConsolHandle,'\n');
            count = 0;
        }
    }
    c = sprintf(OString,"Total Tasks=%d\r\n",i);
    Write(ConsolHandle,OString,c);
    return 0;
}

static int DoTaskDump(char *name)
{
    TCB *l = MasterTaskList;            //get address of task list
    int c;

    while(l)
    {
        if(strcmp(name,l->name) == 0)   //find the name?
        {
            c = sprintf(OString,"Current Stack Pointer:0x%08lx\r\n",l->stack);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"Stack Top:           :0x%08lx\r\n",l->stacktop);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"Data Memory          :0x%08lx\r\n",l->datamem);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"Stack Size           :%d\r\n",l->stacksize);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"Priority             :%d\r\n",l->priority);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"Status               :%d\r\n",l->status);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"TimeStamp            :%ld\r\n",l->TimeStamp);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"Swaps                :%ld\r\n",l->TcbSwaps);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"Timeout              :%d\r\n",l->timeout);
            Write(ConsolHandle,OString,c);
```

```
            c = sprintf(OString,"Start Time           :%u\r\n",l->StartTime);
            Write(ConsolHandle,OString,c);
            c = sprintf(OString,"Total Time           :%lu\r\n",l->TotalTime);
            Write(ConsolHandle,OString,c);
            if(l->pending)
            {
                l->pending->PrintInfo();
            }
            goto exit;
        }
        else
            l = l->list;
    }
    c = sprintf(OString,"Could not find %s\r\n",name);
    Write(ConsolHandle,OString,c);
exit:
    return 0;
}

//static int DoLutTest(void)
//{
//  int *m;
//  int *b = (int *)0xc80000;   //base address of Look Up table memory
//  int loop = 1;
//  int a;
//  int c;
//  unsigned d,data;
//  int state = 0;      //0=write,1=verify
//  int dir = 0;        //0=shift left,1=shift right
//  long errors=0l;
//  long pass = 0l;
//  int count = 0;

//  c = sprintf(OString,"Look Up Table Memory Test\r\n");
//  Write(ConsolHandle,OString,c);
//  d = 1;
//  data = d;
//  m = b;
//  while(loop)
//  {
//      if(!state)          //do we write?
//      {
//          *m++ = d;           //store data, increment pointer
//          if(m == (int *)0xca0000)    //end of memory?
//          {
//              state = 1;      //switch to verify
//              m = b;
//              d = data;
//              dir = 0;
//              continue;
//          }
//      }
//      else                //do verify
//      {
//          if(*m != d)
//          {
//              ++errors;
//              c = sprintf(OString,"%08lx:Got %04x:Expected %04x\r\n",m,*m,d);
//              Write(ConsolHandle,OString,c);
//          }
//          m++;
//          if(m == (int *)0xca0000)    //end of memory
//          {
//              state = 0;  //put into write mode
//              m = b;
//              data = (data << 1) | 1;
```

```
//              if(data == 0xffff) data = 1;
//              d = data;
//              dir = 0;
//              c = sprintf(OString,"PASS:%8ld::ERRORS=%ld\r\n",pass++,errors);
//              Write(ConsolHandle,OString,c);
//              continue;
//          }
//      }
//      if(dir)
//      {
//          d >>= 1;
//          if(d & 0x0001)
//              dir = 0;
//      }
//      else
//      {
//          d <<= 1;
//          if(d & 0x8000)
//              dir = 1;
//      }
//      if(++count == 4096)
//      {
//          if(Status(ConsolHandle,(char *)0,0l,RECEIVE_STAT) )
//          {
//              a = Getc(ConsolHandle);
//              if(a == 'X');
//                  loop = 0;
//          }
//          count = 0;
//      }
//  }
//  return 0;
//}

static int DoDspHostReg(void)
{
    int c;
    char *dsp = (char *)0xffd40001;
    int i;

    c = sprintf(OString,"------Dump of DSP host Port------\r\n");
    Write(ConsolHandle,OString,c);
    for(i=0;i<8;++i,dsp+=2)
    {
        c = sprintf(OString,"REG%d:%02x\r\n",i,(int)(*dsp) & 0x0ff);
        Write(ConsolHandle,OString,c);
    }
    return 0;
}

static int DoVersion(void)
{
    int c;
    int error;
    char *s = new char[256];

    c = sprintf(OString,"%s\r\n",tel_board_name);
    Write(ConsolHandle,OString,c);
    c = sprintf(OString,"%s\r\n",tel_board_rev);
    Write(ConsolHandle,OString,c);
    c = sprintf(OString,"%s\r\n",tel_version);
    Write(ConsolHandle,OString,c);
    c = sprintf(OString,"%s\r\n",tel_date);
    Write(ConsolHandle,OString,c);
    c = sprintf(OString,"%s\r\n",tel_copyright);
    Write(ConsolHandle,OString,c);
```

```
    if((error = GetVersionNumber(0,s,1)) < 0)   //get version number
        c = sprintf(OString,"DSP ERROR:%d\r\n",error);
    else
        c = sprintf(OString,"%s\r\n",s);
    Write(ConsolHandle,OString,c);

    union {
        unsigned v;
        struct {
            unsigned dummy:2;
            unsigned year:5;
            unsigned month:4;
            unsigned day:5;
        } date;
    }version;

    version.v = I2CGetStageVersion();
    c = sprintf(OString,"%d-%d-%d\r\n",version.date.year,version.date.month,version.date.day);
    Write(ConsolHandle,OString,c);

    delete[] s;
    return 0;
}

//static int DoQuadKnob(void)
//{
//  int loop = 1;
//  int c;
//  int d;
//  extern volatile long UpCount,DownCount;
//
//  while(loop)
//  {
//      d = Getc(sys.HQuad);
//      c = sprintf(OString,"VALUE=::%5d::DOWN:%12ld::UP:%12ld\r",d,DownCount,UpCount);
//      Write(ConsolHandle,OString,c);
//      if(Status(ConsolHandle,(char *)0,0l,RECEIVE_STAT) )
//      {
//          d = Getc(ConsolHandle);
//          if(d == 'X');
//              loop = 0;
//      }
//      Xio(WAIT_EMPTY,ConsolHandle,(char *)0,(char *)0,0l,0);
//  }
//  Putc(ConsolHandle,'\n');
//  return 0;
//}

int DoExersize(long v)
{
    extern Wait *WaitExersize;
//  long count=0;
//  int loop = 1;
//  int c;
//  int d;

    WaitExersize->Post();
    WaitExersize->SetCount((int)(v-1));
//  while(loop)
//  {
//      if(WaitExersize->GetCount() < 0)
//      {
//          c = sprintf(OString,"Passes=%10ld\r\n",count++);
//          Write(ConsolHandle,OString,c);
//          WaitExersize->Post();
//      }
```

```
        //-----------------------------------------------------
        // A Bug is Evident in the following line
        // Sometimes a bad value comes back in D0
        // and indicates there is data waiting when there is
        // infact not
        //-----------------------------------------------------

//      if(c = Status(ConsolHandle,(char *)0,0l,RECEIVE_STAT) )
//      {
//          c = sprintf(OString,"Status=%d\r\n",c);
//          Write(ConsolHandle,OString,c);
//          d = Getc(ConsolHandle);
//          if(d == 'X')
//              loop = 0;
//      }
//  }
    return 0;
}

int DoServoStatus(void)
{
    int error;
    DSP_status stat;
    int c;

    if((error = ServoStatus(RSTAGE,&stat)) < 0)
    {
        c = sprintf(OString,"ERROR:%d\r\n",error);
        Write(ConsolHandle,OString,c);
    }
    else
    {
        c = sprintf(OString,"STATUS:%lx\r\n",stat.status);
        Write(ConsolHandle,OString,c);
        c = sprintf(OString,"TYPE:%lx\r\n",stat.type);
        Write(ConsolHandle,OString,c);
        c = sprintf(OString,"PENDING:%lx\r\n",stat.pending);
        Write(ConsolHandle,OString,c);
        c = sprintf(OString,"SYSTEM[0]:%lx\r\n",stat.system[XSTAGE]);
        Write(ConsolHandle,OString,c);
        c = sprintf(OString,"SYSTEM[1]:%lx\r\n",stat.system[RSTAGE]);
        Write(ConsolHandle,OString,c);
        c = sprintf(OString,"SYSTEM[2]:%lx\r\n",stat.system[SSTAGE]);   //spindle
        Write(ConsolHandle,OString,c);
        c = sprintf(OString,"SYSTEMSTATUS[0]:%lx\r\n",SystemStatus.servo1.v);
        Write(ConsolHandle,OString,c);
        c = sprintf(OString,"SYSTEMSTATUS[1]:%lx\r\n",SystemStatus.servo2.v);
        Write(ConsolHandle,OString,c);
        c = sprintf(OString,"SYSTEMSTATUS[2]:%lx\r\n",SystemStatus.servo3.v);   //spindle
        Write(ConsolHandle,OString,c);
    }
    return 0;
}
int process_file(int handle,char *s)
{
    /*
    ** s is the string to be parsed by the parser
    */
    ConsolHandle = handle;
    PCB.pointer = (unsigned char *)s;   /*  load pointer into parser control block  */
    consol();                                   /*  parse string    */
    return(consol_value());             /*  return grammar token value  */
}

} //end of embed C code
```

```
#ifndef CONSOL_H
#define CONSOL_H

void init_consol(void);
void consol(void);

int consol_value(void);
typedef union {
  int alignment;
  char ag_vt_2[sizeof(int)];
  char ag_vt_3[sizeof(long)];
  char ag_vt_4[sizeof(char *)];
} consol_vs_type;

typedef enum {
  consol_white_space_token = 1, consol_hexdigit_token = 3,
  consol_hex_num_token = 7, consol_decimal_number_token,
  consol_name_token = 10, consol_name_string_token,
  consol_consol_token = 15, consol_cmds_token, consol_eof_token,
  consol_consol_commands_token, consol_digit_token = 52,
  consol_letter_token, consol_hex_number_token = 56
} consol_token_type;

typedef struct {
  consol_token_type token_number, reduction_token, error_frame_token;
  int input_code;
  int input_value;
  int line, column;
  int ssx, sn, error_frame_ssx;
  int drt, dssx, dsn;
  int ss[32];
  consol_vs_type vs[32];
  int bts[32], btsx;
  unsigned char * pointer;
  unsigned char * la_ptr;
  int lab[15], rx, fx;
  const unsigned char *key_sp;
  int save_index, key_state;
  char *error_message;
  char read_flag;
  char exit_flag;
} consol_pcb_type;


#ifndef PRULE_CONTEXT
#define PRULE_CONTEXT(pcb)  (&((pcb).cs[(pcb).ssx]))
#define PERROR_CONTEXT(pcb) ((pcb).cs[(pcb).error_frame_ssx])
#define PCONTEXT(pcb)       ((pcb).cs[(pcb).ssx])
#endif


#ifndef AG_RUNNING_CODE_CODE
/* PCB.exit_flag values */
#define AG_RUNNING_CODE         0
#define AG_SUCCESS_CODE         1
#define AG_SYNTAX_ERROR_CODE    2
#define AG_REDUCTION_ERROR_CODE 3
#define AG_STACK_ERROR_CODE     4
#define AG_SEMANTIC_ERROR_CODE  5
#endif

extern consol_pcb_type consol_pcb;
#endif
```

```
//-------------------------------------------------------------------------
//
// Data structure for CRASH data
//
// This data structure attempts to be as comprehensive as posible
// so that some idea of what happened can be determinited
//
//-------------------------------------------------------------------------
#include "task.h"

#ifndef CRASH__H
#define CRASH__H

typedef struct {
    int except;     //exception number that occured
    int sr;         //status register
    int instruction;    //instruction that caused problem
    long access;        //address that was accessed to cause problem
    long adr;           //address of program counter
    int function;       //function code
    TCB *task;          //pointer to task that crashed
    long Magic;         //magic number (must be valid for data to be valid)
    int flag;           //if 0, no crash has occured
    long d0;
    long d1;
    long d2;
    long d3;
    long d4;
    long d5;
    long d6;
    long d7;
    long a0;
    long a1;
    long a2;
    long a3;
    long a4;
    long a5;
    long a6;
}CRASH;

extern int GetCrash(void);
extern void InitCrash(void);
extern CRASH crashdata;

#endif
```

```
{
/*************************************************************************
**
** Tiny Basic Interpreter
**
** Copyright (c) 1995 Jim Patchell
**
** Requires the use of Parsifal Software's Anagram LALR compiler.
**
*************************************************************************
**
** This version of Tiny Basic can run as a thread under the operating
** system that I wrote.
**
** This file should be compiled with the -rA5=_data switch turned on
** when using Cross code C++.  This makes the compiler generate code
** that will access the global variables in this file relative to
** address register A5.  It should be noted that you cannot access
** global variables that are outside of tiny basic.  If you need to
** do that, you will need to somehow pass a pointer to the routine
** so that it can have the address of the variable.
**
** by using relative addressing, it is posible to have as many basic
** threads running at the same time as you think you can manage.  Each
** instance will have its very own global memory area.
**
*************************************************************************/
#include <stdio.h>
#include <stdlib.h>
//#include <assert.h>
#include <string.h>
#include "cio.h"
#include "task.h"
#include "queue.h"
#include "tiny.h"

}
[
    ~allow macros
    ~declare pcb
    grammar token = basic
    disregard white space
    ~case sensitive
    lexeme {eol, integer,name string,String Value}
    distinguish lexemes
    line numbers
    ~near functions
    sticky {then part,all character string}
    pointer input
    left {"||"}
    left {"&&"}
    left {'>','<'}
    left {'=',"!="}
    left {'|'}
    left {'&'}
    left {"<<",">>"}
    left {'+','-'}
    left {'*','/','%'}
    parser file name = "tinybasc.cpp"
    header file name = "tinybasc.h"
]

eof = 0
all letters = 1..126 - '"'
letter = 'a-z' + 'A-Z' + '_' + '.'
digit = '0-9'
```

```
noteol = ~(eof + '\n')

eol
    ->'\n'
    ->'\r','\n'

(void)white space
    -> ' ' + '\t'

(int) basic
    ->basic statement list:v , eof  =v;

(int) basic statement list
    ->basic line:v                      =v;
    ->basic statement list,basic line:v =v;

(int) basic line
    -> opt line number,basic statement:v,eol    =v;
    -> opt line number,declaration,eol

(void) opt line number
    ->
    -> line number:ln           ={CurrentLineNumber=ln;}

(void) declaration
    -> "LONG", decl var list:l  ={
                                    if(RunFlag)
                                    {
                                        symbol *s = l;
                                        while(s)
                                        {
                                            s->SetType(SYMBOLTYPE_LONG);
                                            s = s->next;
                                        }
                                    }
                                }
    -> "DOUBLE",decl var list:l={
                                    if(RunFlag)
                                    {
                                        symbol *s = l;
                                        while(s)
                                        {
                                            s->SetType(SYMBOLTYPE_DOUBLE);
                                            s = s->next;
                                        }
                                    }
                                }

    -> "STRING",string decl var list

(symbol *) decl var list
    ->variable name:s       =s;
    ->decl var list:l,',',variable name:s   ={
                                        if(RunFlag)
                                        {
                                            s->next = l;
                                        }
                                        return  s;
                                    }


(symbol *) string decl var list
    ->variable name:s,'[',integer:d,']' ={
                                    if(RunFlag)
                                    {
                                        s->SetMaxStringLen(d);
```

```
                                    s->SetType(SYMBOLTYPE_STRING);
                                }
                                return s;

                                }
    ->string decl var list:l,',',variable name:s,'[',integer:d,']'  ={
                                        if(RunFlag)
                                        {
                                        s->next = l;
                                        s->SetMaxStringLen(d);
                                        s->SetType(SYMBOLTYPE_STRING);
                                        }
                                        return s;
                                    }

(int) basic statement
    ->"STATS"   ={
                char *s = new char[256];
                int c;
                c = sprintf(s,"Total Symbols\r\n+%d\r\n-%d\r\n",TotalSymsAllocated,TotalSymsFree
d);
                Write(Consol,s,c);
                delete[]s;
                return 0;
                }

    ->"DELAY",'(',expression:s,')' = {
                            TDelay(s->GetLongVal());
                            if((s->GetType() & 1) == 0)
                                delete s;
                            return 0;
                            }
    ->"DUMP"        ={Symbols->dump(Consol);return 0;}
    ->"LIST"        ={Write(Consol,ProgramSpace,strlen(ProgramSpace));return 0;}
    ->"NEW"         ={memset(ProgramSpace,0,ProgSpaceSize);return 0;}
    ->"SUSPEND"     ={
                        Tp->ConsolSuspend->Post(0);
                        Tp->BasicSuspend->Pend();
                        return 0;
                    }
    ->"RUN"             ={
            buffer_selector = PROGRAM;
            PCB.pointer = (unsigned char *)ProgramSpace;
            tiny();
            buffer_selector = COMMAND_LINE;
            return 0;
            }

    ->"SAVE" , name string  ={
                        int out;    //handle of output file

                        name[name_index] = 0;
                        char *oname = new char[256];
                        sprintf(oname,"D:%s",name);
                        if((out = Open(oname,WRITE_ONLY)) < 0)
                        {
                            char *os = new char[256];   //output string
                            int l;
                            l = sprintf(os,"Could not open %s for save\r\n",oname);
                            Write(Consol,os,l);
                            delete[] os;
                        }
                        else
                        {
                            int l = strlen(ProgramSpace);
                            Write(out,ProgramSpace,l);
```

```
                        Close(out);
                    }
                    delete[] oname;
                    return 0;
                }
    ->"LOAD" , name string  ={
                    int in; //handle of input file
                    char *s;

                    int c;
                    name[name_index] = 0;
                    char *oname = new char[256];
                    sprintf(oname,"D:%s",name);
                    s = ProgramSpace;
                    if((in = Open(oname,READ_ONLY)) < 0)
                    {
                        char *os = new char[256];   //output string
                        int l;
                        l = sprintf(os,"Could not open %s for load\r\n",oname);
                        Write(Consol,os,l);
                        delete[] os;
                    }
                    else
                    {
                        while((c = Getc(in)) != RAMDISK_EOF)
                            *s++ = (char)c;
                    }
                    *s = 0;
                    Close(in);
                    delete[] oname;
                    return 0;
                }
    ->assignment    ={return 0;}
    ->conditional   ={return 0;}
    ->"GOTO",line number:l  ={
                        char *s;
                        int flag;

                        if(RunFlag)
                        {
                            s = FindLine(l,ProgramSpace,&flag);
                            if(flag)
                            {
                                int l;
                                char *s = new char[256];
                                l = sprintf(s,"Could not find line %d\n",l);
                                Write(Consol,s,l);
                                delete[] s;
                                PCB.exit_flag = AG_SEMANTIC_ERROR_CODE;
                            }
                            else
                            {
                                PCB.pointer = (unsigned char *)s;
                                PCB.ssx = 0;
                                PCB.la_ptr = (unsigned char *) s;
                                PCB.sn = 0; /*  do the goto */
                            }
                        }
                        return 0;

            }
    ->"GOSUB",line number:l   ={
                    char *s;
                    int flag;
                    /*  Save State  */
                    if(RunFlag)
```

```
                    {
                        RetStack[++RetStackPointer].return_line = (char *) PCB.pointer;
                        RetStack[RetStackPointer].laptr = (char *) PCB.la_ptr;
                        RetStack[RetStackPointer].sn = PCB.sn;
                        RetStack[RetStackPointer].ssx = PCB.ssx;

                        s = FindLine(l,ProgramSpace,&flag);
                        if(flag)
                        {
                            int l;
                            char *s = new char[256];
                            l = sprintf(s,"Could not find line %d\n",l);
                            Write(Consol,s,l);
                            delete[] s;
                            PCB.exit_flag = AG_SEMANTIC_ERROR_CODE;
                        }
                        else
                        {
                            PCB.pointer = (unsigned char *)s;
                            PCB.ssx = 0;
                            PCB.la_ptr = (unsigned char *)s;
                            PCB.sn = 0; /*  do the goto */
                        }
                    }
                    return 0;
                }
    ->"RETURN"                  ={
                                    if(RunFlag)
                                    {
                                        PCB.pointer = (unsigned char *)RetStack[RetStackPointer].ret
urn_line;
                                        PCB.la_ptr = (unsigned char *)RetStack[RetStackPointer].lapt
r;
                                        PCB.sn = RetStack[RetStackPointer].sn;
                                        PCB.ssx = RetStack[RetStackPointer--].ssx;
                                    }
                                    return 0;
                                }
    ->forloop                   ={return 0;}
    ->"NEXT", variable name ={
                long v;
                if(RunFlag)
                {
                    v = ForStack[ForStackPointer].loopvar->GetLongVal();
                    v += ForStack[ForStackPointer].step;
                    ForStack[ForStackPointer].loopvar->SetVal(v);
                    if(v <= ForStack[ForStackPointer].count)
                    {
                        //mess with input stream
                        PCB.pointer = (unsigned char *)ForStack[ForStackPointer].start;
                        PCB.ssx = 0;
                        PCB.la_ptr = PCB.pointer;
                        PCB.sn = 0;
                    }
                    else
                    {
                        ForStackPointer--;
                    }
                }
                return 0;
            }
    ->input                     ={return 0;}
    ->output                    ={return 0;}
    ->"POKEB",'(',expression:p,',',expression:v,')' ={DoPoke(p,v,BYTE_POKE);return 0;}
    ->"POKEW",'(',expression:p,',',expression:v,')' ={DoPoke(p,v,WORD_POKE);return 0;}
    ->"POKEL",'(',expression:p,',',expression:v,')' ={DoPoke(p,v,LONG_POKE);return 0;}
```

```
    ->"BYE"                             ={return 1;}

(void)input
    ->"INPUT",variable name list:s  ={
                if(RunFlag)
                {
                    while(s)
                    {
                        char *d;

                        d = getline('?');
                        s->SetVal(strtol(d,(char **)0,10));
                        s = s->next;
                    }
                }
    }

(symbol *) variable name list
    ->variable name:s       =s;
    ->variable name list:l,',', variable name:s     ={if(RunFlag)s->next = l;return s;}

(void) output
    -> "PRINT", expression list:el  ={ExpressionList *l = el;

                                if(RunFlag)
                                {
                                    char *s = new char[1024];
                                    int c=0;
                                    while(l)
                                    {
                                        c += DoPrint(l->v,&s[c]);
                                        l = l->next;
                                        delete el;  //delete members of expression list
                                        el = l;
                                    }
                                    c += sprintf(&s[c],"\r\n");
                                    Write(Consol,s,c);
                                    delete[] s;
                                }
                            }

(ExpressionList *) expression list
    -> expression:v                 ={ExpressionList *el = new ExpressionList;
                                        el->v = v;
                                        return el;
                                    }
    -> expression list:list,',',expression:v    ={ExpressionList *el = new ExpressionList;
                                        ExpressionList *l=list;
                                        while(l->next)
                                            l = l->next;
                                        el->v = v;
                                        l->next = el;
                                        return list;
                                    }


(symbol *) assignment
    ->variable name:s, '=',expression:v ={if(RunFlag)DoAssign(s,v);return s;}

(void) conditional
    -> if part, then part, nothing
    -> if part, then part, else part

(void) nothing
    ->      ={RunFlag = 1;}
```

```
(void) if part
    -> "IF", expression:v   ={if(ConvertToLong(v,1) == 0)RunFlag = 0;}

(void) then part
    -> "THEN", basic statement ={if(RunFlag)RunFlag = 0;else RunFlag=1;}

(void) else part
    -> "ELSE",basic statement  ={RunFlag = 1;}

(void) forloop
    ->"FOR", assignment:s,"TO", expression:limit, optional step:step = {
            if(RunFlag)
            {
                DoForLoop(s,limit,step);
            }
        }

(symbol *) optional step
    ->                      =MakeLongValue(1l);     /*  default step size   */
    ->"STEP", expression:v  =v;

(symbol *)expression
    ->urnary expression
    ->expression:v1,'=',expression:v2       ={symbol *s;s = DoEquals(v1,v2);return s;}
    ->expression:v1,"!=",expression:v2      ={symbol *s;s = DoNotEquals(v1,v2);return s;}
    ->expression:v1,"&&",expression:v2      ={symbol *s;s = DoLogicalAnd(v1,v2);return s;}
    ->expression:v1,"||",expression:v2      ={symbol *s;s = DoLogicalOr(v1,v2);return s;}
    ->expression:v1,'+',expression:v2       ={symbol *s;s = DoAdd(v1,v2);return s;}
    ->expression:v1,'-',expression:v2       ={symbol *s;s = DoSubtract(v1,v2);return s;}
    ->expression:v1,'*',expression:v2       ={symbol *s;s = DoMult(v1,v2);return s;}
    ->expression:v1,'/',expression:v2       ={symbol *s;s = DoDivide(v1,v2);return s;}
    ->expression:v1,'%',expression:v2       ={symbol *s;s = DoMod(v1,v2);return s;}
    ->expression:v1,'>',expression:v2       ={symbol *s;s = DoGreaterThan(v1,v2);return s;}
    ->expression:v1,'<',expression:v2       ={symbol *s;s = DoLessThan(v1,v2);return s;}
    ->expression:v1,">>",expression:v2      ={symbol *s;s = DoShiftRight(v1,v2);return s;}
    ->expression:v1,"<<",expression:v2      ={symbol *s;s = DoShiftLeft(v1,v2);return s;}
    ->expression:v1,'&',expression:v2       ={symbol *s;s = DoBitAnd(v1,v2);return s;}
    ->expression:v1,'|',expression:v2       ={symbol *s;s = DoBitOr(v1,v2);return s;}

(symbol *) urnary expression
    -> postfix expression
    -> '-', expression:v    ={
                            symbol *s;
                            s = DoNeg(v);
                            return s;
                            }

(symbol *) postfix expression
    -> primary expression
    -> "ABS" ,'(',expression:v,')'  ={
                                symbol *s;
                                if(RunFlag)s = DoAbs(v);
                                return s;
                                }
    -> "RND" ,'(',expression:v,')'  ={
                                symbol *s;
                                if(RunFlag)s = DoRandom(v);
                                return s;
                                }
    -> "PEEKB",'(',expression:v,')' ={
                                symbol *s;
                                s = DoPeekB(v);
                                return s;
                                }
    -> "PEEKW",'(',expression:v,')' ={
                                symbol *s;
```

```
                                s = DoPeekW(v);
                                return s;
                                }
    -> "PEEKL",'(',expression:v,')' ={
                                symbol *s;
                                s = DoPeekL(v);
                                return s;
                                }

(symbol *) primary expression
    -> variable name:s      =s;
    -> integer:v            ={
                            //create temporary symbol for integer value
                            symbol *s = new symbol("temp");
                            s->SetType(SYMBOLTYPE_TEMPLONG);
                            s->SetVal(v);
                            return s;
                            }
    -> String Value:s       =s;
    -> '(', expression:v , ')'      =v;

(symbol *)variable name
    ->name string           ={
                        symbol *s=NULL;
                        if(RunFlag)
                        {
                            name[name_index] = 0;
                            if((s = (symbol *)Symbols->findsym(name))==NULL)
                            {
                                s = (symbol *)Symbols->newsym();
                                s->Init();          //fake constructor
                                s->SetName(name);
                                s->SetVal(0l);      //defualt value
                                s->SetType(SYMBOLTYPE_LONG);
                                Symbols->addsym(s);
                            }
                        }
                        return s;
                }

(symbol *)String Value
    -> all character string,'"' ={
                                    symbol *s = new symbol("temp");
                                    s->SetType(SYMBOLTYPE_TEMPSTRING);
                                    name[name_index] = 0;
                                    s->SetVal(name);
                                    return s;
                                }

(void)all character string
    -> '"'                              =iins();
    ->all character string,all letters:c =pcn(c);

(void)name string
    -> letter:c                         =ins(c);
    -> name string, letter+digit:c      =pcn(c);

(long) integer
    -> digit:d                          =d - '0';
    -> integer:n, digit:d               =10*n + d - '0';

(long) line number
    -> integer:d                            =d;

{
    /*
```

```
    ** these are the C functions for the above basic grammar
    */
static char *FindNextLine(int linenumber,char *s,int *flag);

#define SYNTAX_ERROR SyntaxError((PCB).error_message,(PCB).line,(PCB).column)

#define PARSER_STACK_OVERFLOW {ParserStackOverflow((PCB).line, (PCB).column);}

#define REDUCTION_TOKEN_ERROR {ReductionTokenError((PCB).line, (PCB).column);}

#pragma region("ram = relram")


static tiny_pcb_type tiny_pcb[2];
static int buffer_selector;

#define PCB tiny_pcb[buffer_selector]
#define COMMAND_LINE    0
#define PROGRAM         1

static char name[256];
static int name_index;
static hashtab *Symbols;
static char *ProgramSpace;
static int ProgSpaceSize;
static FOR_STACK ForStack[20];  /*  20 loops deep   */
static int ForStackPointer;
static long CurrentLineNumber;
static int RunFlag; /*  when true, parser actions really do something   */
static SubroutineStack RetStack[100];   /*  one hundred levels of subroutine    */
static int RetStackPointer;
static TinyParams *Tp;
static int TotalSymsAllocated;
static int TotalSymsFreed;

static int Consol;      /*  consol device pointer   */

void ReductionTokenError(int line,int col)
{
    char *s = new char[256];    //create space to write syntax error
    int sl;

    sl = sprintf(s,"\r\nReduction Token Error:line %d, column %d\r\n",line,col);
    Write(Consol,s,sl);

    delete[] s;
}

void ParserStackOverflow(int line,int col)
{
    char *s = new char[256];    //create space to write syntax error
    int sl;

    sl = sprintf(s,"\r\nParser stack overflow:line %d, column %d\r\n",line,col);
    Write(Consol,s,sl);

    delete[] s;
}

void SyntaxError(char *m,int l,int c)
{
    char *s = new char[256];    //create space to write syntax error
    int sl;

    sl = sprintf(s,"Syntax Error:%s, line %d, column %d\r\n",m,l,c);
    Write(Consol,s,sl);
```

```
    delete[] s;
}

static void iins(void)
{
    name_index = 0;
}

static void ins(int c)
{
    name[0] = c;
    name_index = 1;
}

static void pcn(int c)
{
//  assert(name_index < 256);
    name[name_index++] = c;
}


//---------------------------------------------------------------------
//
// member functions for hashtab class
//
//---------------------------------------------------------------------

unsigned hashtab::hash(unsigned char *name)
{
    //-----------------------------------------------------
    // this is the hashing function.  It takes the character
    // string name and creates a value from that...sort of
    // like a CRC
    //-----------------------------------------------------
    unsigned h = 0;
    unsigned g;

    for(;*name;++name)
    {
        h = (h << 2) + * name;
        if(g = h & 0xc000)
            h = (h ^ (g >> 12)) & 0x3fff;
    }
    return h;
}

hashtab::hashtab(int size)
{
    //-----------------------------------------------------
    //Constructor
    //besides creating the hash table object, we need to create
    // an array of BUCKET pointers.
    // size: number of elements in the BUCKET pointer array
    //     : It should be noted that this should be a prime
    //     : number...it is a mathematic thingy
    //-----------------------------------------------------
    table = (BUCKET **)new char[size * sizeof(BUCKET *)];
    memset(table,0,size * sizeof(BUCKET *));
    tabsize = size;
}

hashtab::~hashtab()
{
    //-----------------------------------------------------
    //first, we need to scan through table, and delete all of the
```

```
    //symbols that were allocated
    //-------------------------------------------------------
    int i;
    symbol *s;
    BUCKET *b;

    for(i=0;i<tabsize;++i)
    {
        if(table[i])    //valid pointer, delete some stuff
        {
            b = table[i];
            while(b)
            {
                s = (symbol *)(b+1);
                delete s;
                b = b->next;
            }
        }
    }
    delete table;
}

void *hashtab::operator new(size_t size)
{
    GetCurrentTask()->TotalMemory += size;
    return malloc(size);
}

void hashtab::operator delete(void *m)
{
    GetCurrentTask()->TotalMemory -= sizeof(hashtab);
    free(m);
}

void *hashtab::newsym(void)
{
    //-----------------------------------------------------
    // Create a new symbol
    //
    // This function has been changed starting in version 1.70.03
    // The symbol class is now responsible for added on the
    // extra memory for the bucket portion.  This should be
    // much cleaner
    // This function will return a pointer to the symbol portion
    //-----------------------------------------------------
    symbol *s = new symbol("");
    return (void *)s;
}

void *hashtab::addsym(void *isym)
{
    //-------------------------------------------------------------
    // add a symbol to the hash table
    //
    // isym: pointer to the symbol to add
    //     : the symbol should have been created with newsym (above)
    // action:
    //-------------------------------------------------------------
    BUCKET **p,*tmp;
    BUCKET *sym = (BUCKET *)isym;

    //-------------------------------------------------------------
    // the next line is pretty wierd...
    // the first element in the symbol is the symbol name
    // so by passing sym as a character pointer is
    // passing the name to the hash function
```

```
    // sym is then decremented, which then points us to the BUCKET
    // the hash value is the MODed with the hash table size to get
    // and index into the hash table, which we then use to get a
    // BUCKET pointer to the first element.
    //-------------------------------------------------------------

    p = &table[hash((unsigned char *)sym--) % tabsize]; //get element in has table
    //link in new symbol
    tmp = *p;   //save current symbol (if any, this could be NULL
    *p = sym;   //store new bucket pointer
    sym->prev = p;  //do cross links
    sym->next = tmp;
    if(tmp)         //if temp was not null, do what ever this is
        tmp->prev = &sym->next;
    ++numsyms;      //increment total number of symbols
    return (void *)(isym);  //return back origianl pointer
}

void *hashtab::nextsym(void *ilast)
{
    BUCKET *last = (BUCKET *)ilast;

    for(--last;last->next;last = last->next)
        if(strcmp((char *)ilast,(char *)(last->next + 1)) == 0)
            return (void *)(last->next + 1);
    return (void *)0;
}

void *hashtab::findsym(char *isym)
{
    BUCKET *p;

    p = table[hash(((unsigned char *)isym)/*--*/) % tabsize];
    while(p && strcmp((char *)isym,(char *)(p+1)))
        p = p->next;
    return (void *)(p ? p + 1: (void *)0);
}

void hashtab::delsym(void *isym)
{
    BUCKET *sym = (BUCKET *)isym;

    if(sym)
    {
        --numsyms;
        --sym;
        if(*(sym->prev) = sym->next)
            sym->next->prev = sym->prev;
    }
}

void hashtab::dump(int handle)
{
    //-------------------------------------------------------
    // dump all of the symbols
    //-------------------------------------------------------
    int i;
    BUCKET *b;
    symbol *s;

    for(i=0;i<tabsize;++i)
    {
        if(table[i])    //anything is this one?
        {
            b = table[i];
            while(b)
```

```
            {
                s = (symbol *)(b+1);
                s->PrintInfo(handle);   //print symbol information
                b = b->next;
            }
        }
    }
}

//---------------------------------------------------------------------------
//
// member functions for symbol class
//
//---------------------------------------------------------------------------
symbol::symbol(char *s)
{
    strcpy(name,s);
    next=NULL;
    sv = NULL;
    value.lv = 0l;
}

symbol::~symbol()
{
    if(sv)
        delete[] sv;
}

void *symbol::operator new(size_t size)
{
    BUCKET *b;
    size += sizeof(BUCKET); //add on extra memory for bucket
    GetCurrentTask()->TotalMemory += size;
    b = (BUCKET *)malloc(size);
    b++;        //increment pointer to start of symbol
    TotalSymsAllocated++;
    return (void *)b;   //return pointer to symbol
}

void symbol::operator delete(void *m)
{
    //-----------------------------------------
    //m points to symbol.  To free memory, must point
    // to BUCKET
    //-----------------------------------------
    BUCKET *b = (BUCKET *)m;
    --b;
    free(b);
    TotalSymsFreed--;
    GetCurrentTask()->TotalMemory -= sizeof(symbol) + sizeof(BUCKET);
}

void symbol::Init(void) //fake constructor
{
    next=NULL;
    sv = NULL;
}

void symbol::SetName(char *s)
{
    strcpy(name,s);
}

void symbol::SetVal(long v)
{
    value.lv = v;
```

```
}

long symbol::GetLongVal(void)
{
    long val;

    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            val = value.lv;
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            val = (long)value.dv;
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            val = 0;
            break;
    }
    return val;
}

void symbol::SetVal(double v)
{
    value.dv = v;
}

double symbol::GetDoubleVal(void)
{
    double val;

    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            val = (double)value.lv;
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            val = value.dv;
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            val = 0.0;
            break;
    }
    return val;
}

void symbol::SetType(int t)
{
    type = t;
}

int symbol::GetType(void)
{
    return type;
}

void symbol::SetVal(symbol *v)
{
    type = v->GetType();
    switch(type)
    {
```

```
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            value.lv = v->GetLongVal();
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            value.dv = v->GetDoubleVal();
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            if(sv)
            {
                if(v->GetStringLen()+1 > (int)(MallocBlockSize(sv) - sizeof(void *)))
                {
                    delete[] sv;
                    sv = new char[v->GetStringLen()+1];
                }
            }
            else
            {
                sv = new char[v->GetStringLen()+1];
            }
            strcpy(sv,v->GetStringVal());
            break;
    }
}

char *symbol::GetStringVal(void)
{
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            if(!sv) sv = new char[256];
            sprintf(sv,"%ld",value.lv);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            if(!sv) sv = new char[256];
            sprintf(sv,"%lf",value.dv);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    return sv;
}

int symbol::GetStringLen(void)
{
    int l = 0;
    if(sv)
        l = strlen(sv);
    return l;
}

void symbol::SetMaxStringLen(long d)
{
    if(sv)
    {
        if(value.lv < d)
        {
            delete[] sv;    //remove old memory
            sv = new char[d];
            value.lv = d;
        }
```

```
    }
    else
    {
        sv = new char[d];
        value.lv = d;
    }
    memset(sv,0,d); //zero new memory
}

void symbol::SetVal(char *s)
{
    if(!sv) //if string has not been allocated yet
        sv = new char[strlen(s)+1];
    strcpy(sv,s);
}

static const char * const symtypes[] = {
    "No type",
    "Long",
    "Long",
    "Double",
    "Double",
    "String",
    "String"
};

void symbol::PrintInfo(int handle)
{
    char *s = new char[1024];
    int c;
    c = sprintf(s,"---------------\"%s\"----------------\r\n",name);
    c += sprintf(&s[c],"Type = %s\r\n",symtypes[type]);
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            c += sprintf(&s[c],"%ld\r\n",value.lv);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            c += sprintf(&s[c],"%lf\r\n",value.dv);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            if(sv)
                c += sprintf(&s[c],"MaxSize:%ld->%s\r\n",value.lv,sv);
            else
                c += sprintf(&s[c],"No String Yet\r\n");
            break;
    }
    c += sprintf(&s[c],"-----------------------------------------\r\n");
    Write(handle,s,c);
    delete[] s;
}

//---------------------------------------------------------------------
// Misc Program Utility Routines
//---------------------------------------------------------------------

symbol *MakeLongValue(long v)
{
    symbol *s = new symbol("temp");
    s->SetVal(v);
    s->SetType(SYMBOLTYPE_TEMPLONG);
    return s;
}
```

```
long ConvertToLong(symbol *s,int deleteflag)
{
    int type = s->GetType();
    long v;
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            v = s->GetLongVal();
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            v = (long)s->GetDoubleVal();
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
//          printf("Big Time Error!\n");
            break;
    }
    if( ((type & 1) == 0) && deleteflag)    //delete temporary symbol if desired
        delete s;
    return v;
}

void DoForLoop(symbol *s,symbol *limit,symbol *step)
{
    int flag;

    ++ForStackPointer;      //increment stack pointer
    ForStack[ForStackPointer].start = FindNextLine((int)CurrentLineNumber,ProgramSpace,&flag);
    ForStack[ForStackPointer].count = ConvertToLong(limit,1);
    ForStack[ForStackPointer].step = ConvertToLong(step,1);
    ForStack[ForStackPointer].loopvar = s;
}

void DoAssign(symbol *s,symbol *v)
{
    int type1 = v->GetType();
    switch(s->GetType() )
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal(v->GetLongVal());
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v->GetDoubleVal());
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetVal(v->GetStringVal() );
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v;
}

int DoPrint(symbol *v,char *s)
{
    int c;
    int type = v->GetType();
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
```

```
            c = sprintf(s,"%ld",v->GetLongVal());
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            c = sprintf(s,"%lf",v->GetDoubleVal());
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            c = sprintf(s,"%s",v->GetStringVal());
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete v;
    return c;
}

void DoPoke(symbol *a,symbol *v,int optype)
{
    long address;
    long val;
    int type = v->GetType();
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            val = v->GetLongVal();
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            val = (long)v->GetDoubleVal();
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete v;
    type = a->GetType();
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            address = v->GetLongVal();
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            address = (long)v->GetDoubleVal();
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete a;
    switch(optype)
    {
        case BYTE_POKE:
            *((char *)a) = (char)val;
            break;
        case WORD_POKE:
            *((int *)a) = (int)val;
            break;
        case LONG_POKE:
            *((long *)a) = val;
            break;
    }
```

```
}

symbol *DoEquals(symbol *v1,symbol *v2)
{
    long v;
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetLongVal() == v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = (double)v1->GetLongVal() == v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetDoubleVal() == (double)v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = v1->GetDoubleVal() == v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = 0l;
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = 0l;
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
```

```
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    s->SetVal(v);
    s->SetType(SYMBOLTYPE_TEMPLONG);
    return s;
}

symbol *DoNotEquals(symbol *v1,symbol *v2)
{
    long v;
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetLongVal() != v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = (double)v1->GetLongVal() != v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v=0l;
                    break;
            }
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetDoubleVal() != (double)v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = v1->GetDoubleVal() != v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = 0l;
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = 0l;
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
```

```
                    v = 0l;
                    break;
            }
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    s->SetVal(v);
    s->SetType(SYMBOLTYPE_TEMPLONG);
    return s;
}

symbol *DoLogicalAnd(symbol *v1,symbol *v2)
{
    long v;
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            v = v1->GetLongVal() && v2->GetLongVal();
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            v = 0l;
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    s->SetVal(v);
    s->SetType(SYMBOLTYPE_TEMPLONG);
    return s;
}

symbol *DoLogicalOr(symbol *v1,symbol *v2)
{
    long v;
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetLongVal() || v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = v1->GetLongVal() || (long)v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
```

```
            }
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = (long)v1->GetDoubleVal() || v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = (long)v1->GetDoubleVal() || (long)v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = 0l;
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = 0l;
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    s->SetVal(v);
    s->SetType(SYMBOLTYPE_TEMPLONG);
    return s;
}

symbol *DoAdd(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal(v1->GetLongVal() + v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetDoubleVal() + v2->GetDoubleVal());
            s->SetType(SYMBOLTYPE_TEMPDOUBLE);
            break;
        case SYMBOLTYPE_STRING:
```

```
        case SYMBOLTYPE_TEMPSTRING:
            s->SetMaxStringLen(v1->GetStringLen() + v2->GetStringLen() + 1);
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            strcpy(s->GetStringVal(),v1->GetStringVal());
            strcat(s->GetStringVal(),v2->GetStringVal());
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoSubtract(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal(v1->GetLongVal() - v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetDoubleVal() - v2->GetDoubleVal());
            s->SetType(SYMBOLTYPE_TEMPDOUBLE);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetVal(0l);
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoMult(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal(v1->GetLongVal() * v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetDoubleVal() * v2->GetDoubleVal());
            s->SetType(SYMBOLTYPE_TEMPDOUBLE);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetVal(0l);
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            break;
```

```
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoDivide(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal(v1->GetLongVal() / v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetDoubleVal() / v2->GetDoubleVal());
            s->SetType(SYMBOLTYPE_TEMPDOUBLE);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetVal(0l);
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoMod(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetLongVal() % v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetVal(0l);
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoGreaterThan(symbol *v1,symbol *v2)
```

```
{
    long v;
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetLongVal() > v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = v1->GetDoubleVal() > v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetDoubleVal() > v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = v1->GetDoubleVal() > v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = 0l;
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = 0l;
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
```

```
    s->SetVal(v);
    s->SetType(SYMBOLTYPE_TEMPLONG);
    return s;
}

symbol *DoLessThan(symbol *v1,symbol *v2)
{
    long v;
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetLongVal() < v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = (double)v1->GetLongVal() < v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = v1->GetDoubleVal() < (double)v2->GetLongVal();
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = v1->GetDoubleVal() < v2->GetDoubleVal();
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            switch(type2)
            {
                case SYMBOLTYPE_LONG:
                case SYMBOLTYPE_TEMPLONG:
                    v = 0l;
                    break;
                case SYMBOLTYPE_DOUBLE:
                case SYMBOLTYPE_TEMPDOUBLE:
                    v = 0l;
                    break;
                case SYMBOLTYPE_STRING:
                case SYMBOLTYPE_TEMPSTRING:
                    v = 0l;
                    break;
            }
```

```
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    s->SetVal(v);
    s->SetType(SYMBOLTYPE_TEMPLONG);
    return s;
}

symbol *DoShiftRight(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetLongVal() >> v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoShiftLeft(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetLongVal() << v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoBitAnd(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
```

```
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetLongVal() & v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoBitOr(symbol *v1,symbol *v2)
{
    int type1 = v1->GetType();
    int type2 = v2->GetType();
    symbol *s = new symbol("temp");
    switch(type1)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal(v1->GetLongVal() | v2->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            s->SetType(SYMBOLTYPE_TEMPSTRING);
            break;
    }
    if((type1 & 1) == 0)    //check to see if input symbol is a temp
        delete v1;
    if((type2 & 1) == 0)    //check to see if input symbol is a temp
        delete v2;
    return s;
}

symbol *DoNeg(symbol *v)
{
    int type = v->GetType();
    symbol *s = new symbol("temp");
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal( -v->GetLongVal());
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            s->SetVal( - v->GetDoubleVal());
            s->SetType(SYMBOLTYPE_TEMPDOUBLE);
            break;
        case SYMBOLTYPE_STRING:
```

```
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete v;
    return s;
}

symbol *DoPeekL(symbol *v)
{
    int type = v->GetType();
    symbol *s = new symbol("temp");
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal(  *((long *)(v->GetLongVal() ) ) );
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete v;
    return s;
}

symbol *DoPeekW(symbol *v)
{
    int type = v->GetType();
    symbol *s = new symbol("temp");
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal( long( *((int *)(v->GetLongVal() )) ) );
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete v;
    return s;
}

symbol *DoPeekB(symbol *v)
{
    int type = v->GetType();
    symbol *s = new symbol("temp");
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal( long( *((char *)(v->GetLongVal() )) ) );
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
```

```
        case SYMBOLTYPE_TEMPDOUBLE:
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete v;
    return s;
}

symbol *DoRandom(symbol *v)
{
    int type = v->GetType();
    symbol *s = new symbol("temp");
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal((long) (rand() % (int)(v->GetLongVal()) )   );
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete v;
    return s;
}

symbol *DoAbs(symbol *v)
{
    int type = v->GetType();
    symbol *s = new symbol("temp");
    switch(type)
    {
        case SYMBOLTYPE_LONG:
        case SYMBOLTYPE_TEMPLONG:
            s->SetVal(labs(v->GetLongVal() ));
            s->SetType(SYMBOLTYPE_TEMPLONG);
            break;
        case SYMBOLTYPE_DOUBLE:
        case SYMBOLTYPE_TEMPDOUBLE:
            break;
        case SYMBOLTYPE_STRING:
        case SYMBOLTYPE_TEMPSTRING:
            break;
    }
    if((type & 1) == 0) //check to see if input symbol is a temp
        delete v;
    return s;
}

static char *getline(int prompt)
{
    static char buff[80];

    int c;
    int i = 0,j;    //i is the index, j is scratch
    int len=0;

    memset(buff,0,80);
```

```
    Putc(Consol,prompt);
    while(1)
    {
        c = Getc(Consol);
        switch(c)
        {
            case 0x1b:  /*  ESCAPE erase line   */
                for(j=0;j<i;++j)
                {
                    Putc(Consol,'\b');
                    Putc(Consol,' ');
                    Putc(Consol,'\b');
                }
                i = 0;
                len = 0;
                break;
            case '\b':
                if(i > 0)
                {
                    --i;
                    --len;  /* one shorter  */
                    Putc(Consol,'\b');
                    Putc(Consol,' ');
                    Putc(Consol,'\b');

                }
                break;
            case '\r':
                break;  //just throw away a CR
            case '\n':
                /*
               ** End of the string, terminate string and exit
               */
                buff[i++] = '\r';   //put in CR
                buff[i++] = c;      //this is the line feed
                buff[i] = '\0';     //terminate string
                Putc(Consol,c);
                Putc(Consol,'\r');
                Putc(Consol,'\n');
                return buff;
            default:
                if(i == len)
                {
                    buff[i] = c;    /*  store character in buffer   */
                    ++i;        /*  increment i     */
                    ++len;
                    Putc(Consol,c); /*  put character to screen */
                }
                break;
        }   /*  end of switch statement */
    }
}

static int LineNumber(char *s)
{
    /* figure out if there is a line number or not  */
    /*  returns 0 if no line number */
    return atoi(s);
}

static char *FindLine(int linenumber,char *s,int *flag)
{
    int ln;

    while(*s)
    {
```

```
        if((ln = LineNumber(s)) == linenumber)
        {
            *flag = 0;
            return(s);
        }
        if(ln > linenumber)
        {
            *flag = 1;
            return s;
        }
        /*  not right line number   */
        s = strchr(s,'\n') ;    /*  find next new line  */
        ++s;                /*  increment to beginning of line  */
    }
    return s;
}

static char *FindNextLine(int linenumber,char *s,int *flag)
{
    char *p = FindLine(linenumber,s,flag);
    char *q = strchr(p,'\n');
    ++q;
    return q;       /*  well, there should be a line number here    */
}

static int InsertLine(char *s,char *newline,char *space,unsigned size)
{
    //s is pointer of place to put new line
    //newline is pointer to line to insert
    //space is the area where it is to be put
    //size is number of bytes in space
    int l = strlen(newline);
    int ms = strlen(s);
    if(s + l + ms > space + size)
    {
        char *em = new char[256];
        int lem;
        lem = sprintf(em,"Not Enough Memory\r\n");
        Write(Consol,em,lem);
        delete[] em;
        return(0);
    }
    if(l)
        memmove(s + l,s,ms);
    else
    {
        char *em = new char[256];
        int lem;
        lem = sprintf(em,"Did not have to move memory\r\n");
        delete[] em;
    }
    memcpy(s,newline,l);    /*  copy in new line    */
    return 1;               /*  yep, we could copy  */
}

static void DelLine(char *s,char *space,unsigned size)
{
    char *p;

    if((p = strchr(s,'\n')) != NULL)
    {
        memmove(s,p+1,size - (p - space) + 1);
    }
    else
    {
        *s = 0;
```

```
    }
}

//---------------------------------------------------------------------------
//
// entry into main
//
//---------------------------------------------------------------------------

extern "C" {
void TinyRunTask(void *p)
{
    //-----------------------------------------------------------------
    // This tiny task takes a file name of a basic program and
    // runs it.
    // There is no editor
    //
    // If the basic program were to stop for some reason, the task
    // will delete itself
    //-----------------------------------------------------------------
    int InHandle;           //handle for input program
    Tp = (TinyParams *)p;   //get info on what to run
    ForStackPointer = -1;
    RunFlag = 1;    /*  when true, parser actions really do something   */
    RetStackPointer = -1;
    Consol = Tp->Handle;
    Symbols = (hashtab *) new hashtab(47);
    ProgramSpace = new char[Tp->Workspace];
    ProgSpaceSize = Tp->Workspace;
    memset(ProgramSpace,0,Tp->Workspace);   /*  zero workspace  */
    if((InHandle = Open(Tp->name,READ_ONLY)) < 0)   //could we open input?
    {
        char *s = new char[256];
        int c;
        c = sprintf(s,"Could Not open %s for input\r\n",Tp->name);
        Write(Consol,s,c);
        delete[] s;
    }
    else
    {
        Read(InHandle,ProgramSpace,ProgSpaceSize);
        Close(InHandle);
        buffer_selector = PROGRAM;
        PCB.pointer = (unsigned char *)ProgramSpace;
        tiny();
    }
    delete[] ProgramSpace;
    delete Symbols;
    TDelete(0); //delete self
}

void TinyTask(void *p)
{
    /*************************************************************
    ** This is a full basic task.  You have to enter the program
    ** and then type run, etc...just like any regular basic
    *************************************************************/

    int loop=1;
    char *e = new char[256];    //space to write error messages
    char *s = new char[256];    //space to get new line of code
    int l;
    int workspace;
    Tp = (TinyParams *)p;

    TotalSymsAllocated=0;
```

```
    TotalSymsFreed=0;
    workspace = Tp->Workspace;
    //initialize global variables
    ForStackPointer = -1;
    RunFlag = 0;    /*  when true, parser actions really do something   */
    RetStackPointer = -1;
    Consol = Tp->Handle;
    Symbols = (hashtab *) new hashtab(47);
    ProgramSpace = new char[workspace];
    ProgSpaceSize = workspace;
    memset(ProgramSpace,0,workspace);   /*  zero workspace  */
    while(loop) //loop forerver
    {
        char *s,*p;
        int ln;
        int flag;

        s = getline('*');
        if(ln = LineNumber(s))
        {
        /*
        ** PARSE THE LINE
        */
            buffer_selector = COMMAND_LINE;
            PCB.pointer = (unsigned char *)s;
            tiny();
            if(PCB.exit_flag == AG_SYNTAX_ERROR_CODE)
            {
                l = sprintf(e,"Syntax Error\r\n");
                Write(Consol,e,l);
            }
            else
            {
                p = FindLine(ln,ProgramSpace,&flag);
                if(p)
                {
                    if(!flag)
                    {
                        DelLine(p,ProgramSpace,workspace);
                    }
                    InsertLine(p,s,ProgramSpace,workspace);
                }
                else
                {
                    l = sprintf(e,"Could Not Insert that line\r\n");
                    Write(Consol,e,l);
                }
            }
        }
        else
        {
            RunFlag = 1;
            buffer_selector = COMMAND_LINE;
            PCB.pointer = (unsigned char *)s;
            tiny();
            if(tiny_value() == 1)
                loop = 0;
            RunFlag = 0;
        }
    }
    delete[] e;
    delete[] s;
    delete[] ProgramSpace;
    delete Symbols;
    (Tp->ConsolSuspend)->Post(1);
    TDelete(NULL);
```

```
}
}   /*  end of Extern "C"   */
}   /*  end of embedded C code  */
```

```
//
// tiny.h
//
// header file for tiny basic
//

#ifndef TINY__H
#define TINY__H

#define BYTE_POKE       0
#define WORD_POKE       1
#define LONG_POKE       2

typedef struct BUCKET {
    struct BUCKET *next;
    struct BUCKET **prev;
}BUCKET;

class hashtab {
    int tabsize;
    int numsyms;
    BUCKET **table;
    unsigned hash(unsigned char *name);
public:
    hashtab(int size);
    ~hashtab();
    void *operator new(size_t size);
    void operator delete(void *);
    void *newsym(void);
    void *addsym(void *isym);
    void *findsym(char *isym);
    void delsym(void *isym);
    void *nextsym(void *ilast);
    void dump(int handle);
};

struct string {
    int size;
    char *s;
};

#define SYMBOLTYPE_LONG         1
#define SYMBOLTYPE_TEMPLONG     2
#define SYMBOLTYPE_DOUBLE       3
#define SYMBOLTYPE_TEMPDOUBLE   4
#define SYMBOLTYPE_STRING       5
#define SYMBOLTYPE_TEMPSTRING   6

class symbol {
    char name[32];
    int type;
public:
    union{
        long lv;
        double dv;
    }value;
    char *sv;
    symbol *next;
    symbol(char *s);    //constructor
    ~symbol();          //destructor
    void *operator new(size_t size);
    void operator delete(void *);
    void Init(void);    //fake constructor
    void SetVal(long v);
    long GetLongVal(void);
    void SetVal(double v);
```

```
    double GetDoubleVal(void);
    void SetName(char *s);
    void PrintInfo(int handle);
    void SetType(int t);
    int GetType(void);
    void SetVal(symbol *v);
    char *GetStringVal(void);
    int GetStringLen(void);
    void SetMaxStringLen(long d);
    void SetVal(char *s);
};

struct FOR_STACK {
    char *start;    /*  start of loop   */
    long count;
    long step;
    symbol *loopvar;
};

struct SubroutineStack {
    char *return_line;
    char *laptr;
    int ssx,sn;
};

struct ExpressionList {
    symbol *v;
    ExpressionList *next;
    ExpressionList(){next = NULL;}
};

struct TinyParams {
    int flag;
    int Handle;             //handle for standard output
    Wait *ConsolSuspend;
    Wait *BasicSuspend;
    int Workspace;          //size of workspace
    char *name;             //name of basic program to run
};


/* Function Prototypes  */

#ifdef __cplusplus
extern "C" {
#endif

extern void TinyRunTask(void *p);
extern void TinyTask(void *p);

#ifdef __cplusplus
}
#endif


#endif
```

```
;*****************************************************************
; Region.s
;
; This file is used to get the sizes of various data regions
;
;*****************************************************************

        SECTION code

        XDEF    _RelRamSize,_RelDataSize,_DataAddress,_HeapSize
        XREF    DATA2,MALLOC_SIZE

_RelRamSize
        move.l  #`SIZE(relram),D0
        rts

_RelDataSize
        move.l  #`SIZE(reldata),D0
        rts

_DataAddress
        move.l  #DATA2,D0
        rts

_HeapSize
        move.l  #MALLOC_SIZE,D0
        rts
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "shost.h"
#include "spid.h"
#include "srtc.h"
#include "sirq.h"
#include "snv.h"
#include "sevent.h"
#include "servhw.h"

/* char os[256]; */

main()
{
    int i;

    InitPID();
    HInit();            /*  Init host Port  */
    ClearPGA();
    init_event();
    InitXICOR();    /*  init code for PORT C direction, etc.    */
    HIcmdEnable();  /*  enable host commands    */
    /*
    ** check ServoFlags and init event.system accordingly
    */
    for(i=0;i<3;++i)    /*  initialize various things for both axis */
    {
        MiscParams[i].ServoFlags &= EN_FATAL_FOLLOWING_ERROR | PID_DISABLE_INTEGRATOR;
        if(MiscParams[i].ServoFlags & EN_FATAL_FOLLOWING_ERROR)
            event_status.system[i] |= F_ERROR_ENABLED;
        else
            event_status.system[i] &= ~F_ERROR_ENABLED;
        if(MiscParams[i].ServoFlags & PID_DISABLE_INTEGRATOR)
            event_status.system[i] |= DIS_INT_CMDVNEZ;
        FiltParams[i].Ki = 0;       /*  default is integrator disabled on power up  */
        MiscParams[i].LinInPosEn = 0;
        MiscParams[i].RunoutLut = (int *)0x100; /* default look up table    */
        MiscParams[i].AnalInput = (int *)0;     /*  default anal input  */
    }
    MiscParams[1].FollErrReg = FOLLOWERROR_REG_P;
    MiscParams[0].FollErrReg = (int *)0x5500;
    MiscParams[2].FollErrReg = (int *)0x5501;

    RTCInit();
    InitHWIRQ();        /*  enable hardware Interrupts  */
    E_enable_pid(0,0);      /*  disable servo loop  */
/*  ZeroCount(0);       */    /*  zero position   */
    E_enable_pid(1,0);      /*  disable servo loop  */
    E_enable_pid(2,0);      /*  disable servo loop  */
/*  ZeroCount(0);       */    /*  zero position   */
    HIFClr(0x18);               /*  clear host port flags   */
    event_loop();
}
```

```
/*
**
** this file contains the routines for doing the host command
** processing.  This is sort of MULTITASKING
** but these routines do not have thier own stack
**
** Just a real fancy interrupt handler
*/
#include <stdio.h>
#include "scommand.h"
#include "spid.h"   /*  for external filter parameters  */
#include "sevent.h"  /* for externals of event manager  */
#include "squeues.h"    /*  for queues definitions  */
#include "srtc.h"
#include "sirq.h"
#include "shome.h"
#include "servhw.h"

typedef union {
    int auxi[2];
    long auxl;
}AUX;       /*  place to store AUX information  */

volatile int MoveAbort[3];      /*  abort move flags    */

int *RunoutComp[3];     /*  pointers for 3 blocks of runout comp    */

static int EnableFunctions(int axis,int func,int flag);

void E_enable_pid(int axis,int);
void E_enable_integrator(int axis,int);
static void E_fatal_follow(int axis,int);
static void E_dither(int axis,int flag);
static void E_Notch(int axis,int flag);
static void E_NAK(int axis,int flag);
static void SendAck(int cmd,int status);
static void SendData(int *data,int wcount);
static void GetData(int *data,int wcount);
static void E_PES(int axis,int falg);
static void E_Notch2(int axis,int flag);
static void E_PostDither(int axis,int flag);

static void (*e_func[9])(int axis,int flag) = {
    E_enable_pid,           /*  0   */
    E_enable_integrator,    /*  1   */
    E_fatal_follow,         /*  2   */
    E_NAK,                  /*  3   */
    E_dither,               /*  4   */
    E_Notch,                /*  5   */
    E_PES,                  /*  6   */
    E_Notch2,               /*  7   */
    E_PostDither            /*  8   */
    };

static int C_beam(int axis);
static int C_limitP(int axis);
static int C_limitM(int axis);
static int C_fault(int axis);
static int C_protect(int axis);

static int C_follow(int axis);
static int C_genfault(int axis);

int (*c_func[7])(int axis) = {
    C_protect,      /*  0   */
    C_limitM,       /*  1   */
```

```
    C_beam,         /*  2   */
    C_limitP,       /*  3   */
    C_fault,        /*  4   */
    C_follow,       /*  5   */
    C_genfault,     /*  6   */
    };

int IrqStatMasks[] = {
    PROT1_IRQSTAT,          /* axis1 mask for protection input  */
    LIMN1_IRQSTAT,          /* axis1 - Limit switch status      */
    BEAM1_IRQSTAT,          /* axis1 Beam Interrupted Status        */
    LIMP1_IRQSTAT,          /* axis1 + Limit switch status      */
    FAULT1_IRQSTAT,         /* axis1 Amp Fault status           */
    PROT2_IRQSTAT,          /* axis2 mask for protection input  */
    LIMN2_IRQSTAT,          /* axis2 - Limit switch status      */
    BEAM2_IRQSTAT,          /* axis2 Beam Interrupted Status        */
    LIMP2_IRQSTAT,          /* axis2 + Limit switch status      */
    FAULT2_IRQSTAT          /* axis2 Amp Fault status           */
};

int IrqClearMasks[] = {
    PROT1_IRQCLEAR,         /* axis1 mask for protection input  */
    LIMN1_IRQCLEAR,         /* axis1 - Limit switch status      */
    BEAM1_IRQCLEAR,         /* axis1 Beam Interrupted Status        */
    LIMP1_IRQCLEAR,         /* axis1 + Limit switch status      */
    FAULT1_IRQCLEAR,        /* axis1 Amp Fault status           */
    POS1_CLEAR,             /* axis1 clear position registers   */
    PROT2_IRQCLEAR,         /* axis2 mask for protection input  */
    LIMN2_IRQCLEAR,         /* axis2 - Limit switch status      */
    BEAM2_IRQCLEAR,         /* axis2 Beam Interrupted Status        */
    LIMP2_IRQCLEAR,         /* axis2 + Limit switch status      */
    FAULT2_IRQCLEAR,        /* axis2 Amp Fault status           */
    POS2_CLEAR              /* axis2 clear position registers   */
};

int IrqLevelMasks[] = {
    BEAM1_LEVEL,            /* axis1 level of beam interrupted line */
    PROT1_LEVEL,            /* axis1 level of protection line   */
    LIMP1_LEVEL,            /* axis1 plus limit line                */
    LIMN1_LEVEL,            /* axis1 minus limit line           */
    FAULT1_LEVEL,           /* axis1 Amp Fault level                    */
    BEAM2_LEVEL,            /* axis2 level of beam interrupted line */
    PROT2_LEVEL,            /* axis2 level of protection line   */
    LIMP2_LEVEL,            /* axis2 plus limit line                */
    LIMN2_LEVEL,            /* axis2 minus limit line           */
    FAULT2_LEVEL,           /* axis2 amp Fault level                    */
};

/*
** prototypes for functions for recieve data commands
*/

static int R_setkp(int axis,int func,AUX a);
static int R_setkv(int axis,int func,AUX a);
static int R_setki(int axis,int func,AUX a);
static int R_setkvff(int axis,int func,AUX a);
static int R_setkaff(int axis,int func,AUX a);
static int R_setoffset(int axis,int func,AUX a);
static int R_setgoal(int axis,int func,AUX a);
static int R_setmaxvel(int axis,int func,AUX a);
static int R_setscurve(int axis,int func,AUX a);
static int R_domove(int axis,int func,AUX a);
static int R_abort(int axis,int func,AUX a);
static int R_phasepol(int axis,int func,AUX a);
static int R_samplerate(int axis,int func,AUX a);
static int R_zeroint(int axis,int func,AUX a);
```

```
static int R_setintmode(int axis, int func,AUX aux);
static int R_startjog(int axis,int func,AUX a);
static int R_endjog(int axis,int func,AUX a);
static int R_clearirq(int axis,int func,AUX a);
static int R_servofunc(int axis,int func,AUX a);
static int R_intlimits(int axis,int func,AUX a);
static int R_zerocount(int axis,int func,AUX a);
static int R_maxfollow(int axis,int func,AUX a);
static int R_folgain(int axis,int func,AUX a);
static int R_home_params(int axis,int func,AUX a);
static int R_home(int axis,int func,AUX a);         /*  go home command */
int R_setirq(int axis,int func,AUX a);
static int R_jog_param(int axis,int func,AUX a);        /*  set the jog parameter          */
static int R_inpostion(int axis,int func,AUX a);
static int R_clearflag(int axis,int func,AUX aux);  /*  clear step flag */
static int R_recordlinpos(int axis,int func,AUX aux);   /*  record linear position for future re
f   */
static int R_setfreq(int axis,int func,AUX aux);
static int R_setcutofffreq(int  axis,int func,AUX aux); /*  set the cuttoff frequency of low pas
s filter    */
static int R_amplitude(int  axis,int func,AUX aux);     /* set amplitude of oscilator   */
static int R_phase(int axis,int func,AUX aux);          /*  set phase of cosine output  */
static int R_dithermode(int axis,int func,AUX aux);     /*  set mode of dither oscialtor    */
static int R_encoderpitch(int axis,int func,AUX aux);   /*  set the encoder pitch   */
static int R_lutselect(int axis,int func,AUX aux);      /*  select look up table for dither */
static int R_handshake(int axis,int func,AUX aux);      /*  handshake with special seagate thing
y   */
static int R_histogram(int axis,int func,AUX aux);      /*  start histogram aquisition  */
static int R_notchfreq(int axis,int func,AUX aux);      /*  set notch filter frequency  */
static int R_notchQ(int axis,int func,AUX aux);         /*  set notch filter Q  */
static int R_notchD(int axis,int func,AUX aux);         /*  set notch filter depth  */
static int R_setanalmode(int axis,int func,AUX aux);        /*  set source of analysis  */
static int R_NAK(int axis,int func,AUX aux);            /*  for commands that don't exist   */
static int R_PesScale(int axis,int func,AUX aux);       /*  set PES scale for servo on track
*/
static int R_PesLimit(int axis,int func,AUX aux);       /*  set PES limit   */
static int R_ResetPes(int axis,int func,AUX aux);       /*  reset PES counter   */
static int R_SetPESMode(int axis,int func,AUX aux);     /*  set PES test mode bit   */
static int R_SetMem(int axis,int func,AUX aux);         /*  set memory location to value    */
static int R_SetGoalAndMove(int axis,int func,AUX aux); /*  set goal and move   */

static int (*r_func[53])(int axis,int func,AUX a) = {
    R_setkp,            /*  0   */
    R_setkv,            /*  1   */
    R_setki,            /*  2   */
    R_setkvff,          /*  3   */
    R_setkaff,          /*  4   */
    R_setoffset,        /*  5   */
    R_setgoal,          /*  6   */
    R_setmaxvel,        /*  7   */
    R_setscurve,        /*  8   */
    R_domove,           /*  9   */
    R_abort,            /*  10  */
    R_phasepol,         /*  11  */
    R_samplerate,       /*  12  */
    R_zeroint,          /*  13  */
    R_setintmode,       /*  14  */
    R_startjog,         /*  15  */
    R_endjog,           /*  16  */
    R_jog_param,        /*  17  */
    R_clearirq,         /*  18  */
    R_servofunc,        /*  19  */
    R_intlimits,        /*  20  */
    R_zerocount,        /*  21  */
    R_maxfollow,        /*  22  */
    R_folgain,          /*  23  */
```

```
    R_home_params,      /*  24  */
    R_home,             /*  25  */
    R_NAK,              /*  26  */
    R_NAK,              /*  27  */
    R_setirq,           /*  28  */
    R_NAK,              /*  29  */
    R_inpostion,        /*  30  */
    R_NAK,              /*  31  */
    R_clearflag,        /*  32  */
    R_recordlinpos,     /*  33  */
    R_setfreq,          /*  34  */
    R_setcutofffreq,    /*  35  */
    R_amplitude,        /*  36  */
    R_phase,            /*  37  */
    R_dithermode,       /*  38  */
    R_encoderpitch,     /*  39  */
    R_lutselect,        /*  40  */
    R_handshake,        /*  41  */
    R_histogram,        /*  42  */
    R_notchfreq,        /*  43  */
    R_notchQ,           /*  44  */
    R_notchD,           /*  45  */
    R_setanalmode,      /*  46  */
    R_PesScale,         /*  47  */
    R_PesLimit,         /*  48  */
    R_ResetPes,         /*  49  */
    R_SetPESMode,       /*  50  */
    R_SetMem,           /*  51  */
    R_SetGoalAndMove    /*  52  */
};

/*
** Servo Commands that recieve a data block
*/

static int RB_config(int **buff,int axis,int func,AUX a);
static int RB_runout(int **buff, int axis, int func, AUX aux);

static int (*rb_func[2])(int **buff,int axis,int func,AUX a) = {
    RB_config,          /*  0   */
    RB_runout           /*  1   */
};

/*
** servo commands that send data back
*/
static int S_NAK(int **buff,int axis,int func,AUX a);   /*  return error for bad function   */
static int S_readpos(int **buff,int axis,int func,AUX a);
static int S_readgoal(int **buff,int axis,int func,AUX a);
static int S_scurvetime(int **buff,int axis,int func,AUX a);
static int S_maxvel(int **buff,int axis,int func,AUX a);
static int S_kp(int **buff,int axis,int func,AUX a);
static int S_kv(int **buff,int axis,int func,AUX a);
static int S_ki(int **buff,int axis,int func,AUX a);
static int S_kvff(int **buff,int axis,int func,AUX a);
static int S_kaff(int **buff,int axis,int func,AUX a);
static int S_offset(int **buff,int axis,int func,AUX a);
static int S_status(int **buff,int axis,int func,AUX a);
static int S_phasepos(int **buff,int axis,int func,AUX a);
static int S_samplerate(int **buff,int axis,int func,AUX a);
static int S_intmode(int **buff,int axis,int func,AUX a);
static int S_daclev(int **buff,int axis,int func,AUX a);
static int S_intlimit(int **buff,int axis,int func,AUX a);
static int S_maxfollow(int **buff,int axis,int func,AUX a);
static int S_folgain(int **buff,int axis,int func,AUX a);
static int S_serialnumber(int **buff,int axis,int func,AUX a);
```

```
static int S_home_params(int **buff,int axis,int func,AUX a);
static int S_getirq(int **buff,int axis,int func,AUX a);
static int S_readpmem(int **buff,int axis,int func,AUX a);
static int S_readymem(int **buff,int axis,int func,AUX a);
static int S_readxmem(int **buff,int axis,int func,AUX a);
static int S_caldist(int **buff,int axis,int func,AUX a);
static int S_jog_params(int **buff,int axis,int func,AUX a);
static int S_inposition(int **buff,int axis,int func,AUX a);
static int S_sendsettings(int **buff,int axis,int func,AUX aux);
static int S_frequency(int **buff,int axis,int func,AUX aux);
static int S_magnitude(int **buff,int axis,int func,AUX aux);
static int S_cutofffreq(int **buff,int axis,int func,AUX aux);
static int S_linearendpoint(int **buff,int axis,int func,AUX aux);
static int S_amplitude(int **buff,int axis,int func,AUX aux);
static int S_phase(int **buff,int axis,int func,AUX aux);
static int S_dithermode(int **buff,int axis,int func,AUX aux);
static int S_encoderpitch(int **buff,int axis,int func,AUX aux);
static int S_runoutcomp(int **buff,int axis,int func,AUX aux);
static int S_lutselect(int **buff,int axis,int func,AUX aux);
static int S_histogram(int **buff,int axis,int func,AUX aux);
static int S_notchfreq(int **buff,int axis,int func,AUX aux);
static int S_notchQ(int **buff,int axis,int func,AUX aux);
static int S_notchD(int **buff,int axis,int func,AUX aux);
static int S_PesScale(int **buff,int axis,int func,AUX aux);
static int S_PesLimit(int **buff,int axis,int func,AUX aux);
static int S_FPGAID(int **buff,int axis,int func,AUX aux);
static int S_otherconfig(int **buff,int axis,int func,AUX aux);
static int S_datastruct(int **buff,int axis,int func,AUX aux);
static int S_magnitude1(int **buff,int axis,int func,AUX aux);

static int (*s_func[49])(int **buff,int axis,int func,AUX a) = {
    S_readpos,          /*  0/256   */
    S_readgoal,         /*  1/257       */
    S_scurvetime,       /*  2/258       */
    S_maxvel,           /*  3/259       */
    S_kp,               /*  4/260       */
    S_kv,               /*  5/261       */
    S_ki,               /*  6/262       */
    S_kvff,             /*  7/263       */
    S_kaff,             /*  8/264       */
    S_offset,           /*  9/265       */
    S_status,           /*  10/266      */
    S_phasepos,         /*  11/267      */
    S_samplerate,       /*  12/268      */
    S_intmode,          /*  13/269      */
    S_daclev,           /*  14/270      */
    S_intlimit,         /*  15/271      */
    S_maxfollow,        /*  16/272      */
    S_folgain,          /*  17/273      */
    S_serialnumber,     /*  18/274      */
    S_home_params,      /*  19/275      */
    S_getirq,           /*  20/276      */
    S_readpmem,         /*  21/277      */
    S_readymem,         /*  22/278      */
    S_readxmem,         /*  23/279      */
    S_caldist,          /*  24/280      */
    S_jog_params,       /*  25/281      */
    S_inposition,       /*  26/282      */
    S_NAK,              /*  27/283      */
    S_sendsettings,     /*  28/284      */
    S_linearendpoint,   /*  29/285      */
    S_frequency,        /*  30/286      */
    S_magnitude,        /*  31/287      */
    S_cutofffreq,       /*  32/288      */
    S_amplitude,        /*  33/289      */
    S_phase,            /*  34/290      */
```

```
    S_dithermode,       /*  35/291      */
    S_encoderpitch,     /*  36/292      */
    S_runoutcomp,       /*  37/293      */
    S_lutselect,        /*  38/294      */
    S_histogram,        /*  39/295      */
    S_notchfreq,        /*  40/296      */
    S_notchQ,           /*  41/297      */
    S_notchD,           /*  42/298      */
    S_PesScale,         /*  43/299      */
    S_PesLimit,         /*  44/300      */
    S_FPGAID,           /*  45/301      */
    S_otherconfig,      /*  46/302      */
    S_datastruct,       /*  47/303      */
    S_magnitude1        /*  48/304      */
};

/* misc globals */
long DesiredGoal[3] = {0l,0l,0l};
int beam_flag = 0;
int General_Servo_Fault = 0;
TEL_PROFILE MoveProfiles[3];

static int val;     /*  used by some of the send functions  */
static long lval;   /*  used by some of the send functions  */
static int aval[2]; /*  used by some of the send functions  */

void Command(void)
{
    unsigned cmd;   /*  place to store command          */
    int axis;       /*  axis to be operated on          */
    int func;       /*  command sub function            */
    AUX aux;        /*  place to store AUX information  */
    int size,*buff; /*  params for send functions       */

    cmd = HIGet();          /*  get the command         */
    axis = HIGet();         /*  get Axis Number         */
    func = HIGet();         /*  get function number     */
    aux.auxi[0] = HIGet();  /*  get data                */
    aux.auxi[1] = HIGet();
    if(cmd > 511 && cmd < (SET_RUNOUTCOMP+1))
    {
        /*
        ** We recieve a buffer of data
        */
        cmd -= 512; /* offset command   */
        if((size = (*rb_func[cmd])(&buff,axis,func,aux) ) > 0)
        {
            SendAck(cmd+512,0);     /*  no data is returned */
        }
        else
            SendAck(cmd+512,-1);    /*  NAK */
    }
    else if(cmd > 255 && cmd < (GET_MAGNITUDE1+1))          /*  this is a send type command */
    {
        cmd -= 256;
        if((size = (*s_func[cmd])(&buff,axis,func,aux) ) > 0)
        {
            SendAck(cmd+256,size);
            SendData(buff,size);
        }
        else
            SendAck(cmd+256,-1);
    }
    else if (cmd < (SETGOALANDMOVE + 1) )
    {
        if( (*r_func[cmd])(axis,func,aux) )
```

```
        {
            SendAck(cmd,-1);    /*  send NAK    */
        }
        else
            SendAck(cmd,0);
    }
    else
        SendAck(-1,-1);         /*  unknown command */
}

static void SendAck(int cmd,int ack)
{
    HIPut(DSP_COMMANDACK);      /*  send ACK command    */
    HIPut(cmd);                 /*  send command that was processed */
    HIPut(ack);                 /*  ack status word */
}

static void SendData(int *data,int wcount)
{
    /*  send count data words pointed to by data    */
    int i;

    for(i=0;i<wcount;++i,++data)
        HIPut(*data);
}

static void GetData(int *data,int wcount)
{
    int i;

    for(i=0;i<wcount;++i,++data)
        *data = HIGet();
}

/*
** functions to enable various servo functions
*/

static int EnableFunctions(int axis,int func,int flag)
{
    if(func > ENABLE_POSTDITHER)
        return(0);
    if((axis > 2) || (axis < 0))
        return 0;
    (*e_func[func])(axis,flag); /*  call function   */
    return 1;
}

void E_enable_pid(int axis,int flag)
{
    if(flag)            /*  enable servo loop   */
    {
        if(!PidError[axis])
        {
            MiscParams[axis].ServoFlags |= PID_ENABLE_SERVO;
            event_status.system[axis] |= SERVO_ENABLED;
            Enable_Amp(axis);
        }
    }
    else                    /*  disable servo loop  */
    {
        MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
        event_status.system[axis] &= ~SERVO_ENABLED;
        Disable_Amp(axis);
    }
}
```

```
void E_enable_integrator(int axis,int flag)
{
    if(flag)            /*  enable  */
    {
        if(!(event_status.system[axis] & INTEGRATOR_ENABLED))
        {
            MiscParams[axis].ServoFlags |= PID_ENABLE_INTEGRATOR;
            FiltParams[axis].Ki = MiscParams[axis].Ki;
            event_status.system[axis] |= INTEGRATOR_ENABLED;
        }
    }
    else                    /*  disable */
    {
        if(event_status.system[axis] & INTEGRATOR_ENABLED)
        {
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_INTEGRATOR;
            FiltParams[axis].Ki = 0;                    /*  this will disable it    */
            event_status.system[axis] &= ~INTEGRATOR_ENABLED;
        }
    }
}

static void E_fatal_follow(int axis,int flag)
{
    if(flag)
    {
        MiscParams[axis].ServoFlags |= EN_FATAL_FOLLOWING_ERROR;
        event_status.system[axis] |= F_ERROR_ENABLED;
    }
    else
    {
        MiscParams[axis].ServoFlags &= ~EN_FATAL_FOLLOWING_ERROR;
        event_status.system[axis] &= ~F_ERROR_ENABLED;
    }
}

static void E_NAK(int axis,int flag)
{
    return;
}

static void E_dither(int axis,int flag)
{
    if(flag)
    {
        MiscParams[axis].ServoFlags |= EN_DITHER;
        event_status.system[axis] |= SYS_DITHER_ENABLED;
    }
    else
    {
        MiscParams[axis].ServoFlags &= ~EN_DITHER;
        event_status.system[axis] &= ~SYS_DITHER_ENABLED;
    }
}

static void E_Notch(int axis,int flag)
{
    /*
    ** Function to enable notch filter in PID loop
    */
    if(flag)
    {
        MiscParams[axis].ServoFlags |= EN_NOTCHFILTER;
        event_status.system[axis] |= DSPSTAT_NOTCHFILTER;
    }
```

```
    else
    {
        MiscParams[axis].ServoFlags &= ~EN_NOTCHFILTER;
        event_status.system[axis] &= ~DSPSTAT_NOTCHFILTER;
    }
}

static void E_PES(int axis,int flag)
{
    if(flag)
    {
        MiscParams[axis].ServoFlags |= EN_PES;
        event_status.system[axis] |= DSPSTAT_PES;
    }
    else
    {
        MiscParams[axis].ServoFlags &= ~EN_PES;
        event_status.system[axis] &= ~DSPSTAT_PES;
    }
}

static void E_Notch2(int axis,int flag)
{
    /*
    ** Function to enable notch filter in PID loop
    */
    if(flag)
    {
        MiscParams[axis].ServoFlags |= EN_NOTCHFILTER2;
        event_status.system[axis] |= DSPSTAT_NOTCHFILT2;
    }
    else
    {
        MiscParams[axis].ServoFlags &= ~EN_NOTCHFILTER2;
        event_status.system[axis] &= ~DSPSTAT_NOTCHFILT2;
    }
}

static void E_PostDither(int axis,int flag)
{
    if(flag)
    {
        MiscParams[axis].ServoFlags |= EN_POSTDITHER;
        event_status.system[axis] |= DSPSTAT_POSTDITHER;
    }
    else
    {
        MiscParams[axis].ServoFlags &= ~EN_POSTDITHER;
        event_status.system[axis] &= ~DSPSTAT_POSTDITHER;
    }
}

/*
** functions to clear interrupt functions
*/

static int C_beam(int axis)
{
    int mask;

    mask = IrqLevelMasks[axis * 5]; /*  get bit pos of interrupt level  */
    if(RDIRQLEVEL_REG & mask)   /*  is beam still broken?   */
    {
        return 0;                       /*  say, nope, won't clear  */
    }
    else
```

```
    {
        /*
        ** Beam is good, clear exception
        */
        mask = IrqClearMasks[axis * 6 + 2]; /*  get bit position of interrupt clear */
        CLEARIRQ_REG = mask;    /*  clear Beam Stat */
        mask = IrqStatMasks[axis * 5 + 2];  /*  get bit pos of interrupt enable */
        if(MiscParams[axis].IrqParams[CLEAR_BEAM] & 0x01)
            EnableHWIRQ(mask);              /*  enable the Beam Interrupt   */
        event_status.system[axis] &= ~BEAM_INTERRUPTED;
        MiscParams[axis].IrqParams[CLEAR_BEAM] &= ~0x04;
        PidError[axis] &= ~mask;
    }
    return(1);
}

static int C_limitP(int axis)
{
    int mask;

    mask = IrqLevelMasks[axis * 5 + 2]; /*  get bit pos of interrupt level  */
    if(RDIRQLEVEL_REG & mask)   /*  is limit switch still on?   */
        return(0);
    else
    {
        mask = IrqClearMasks[axis * 6 + 3]; /*  get bit position of interrupt clear */
        CLEARIRQ_REG = mask;    /*  clear Limit stat    */
        mask = IrqStatMasks[axis * 5 + 3];  /*  get bit pos of interrupt enable */
        if(MiscParams[axis].IrqParams[CLEAR_LIMIT_P] & 0x01)
            EnableHWIRQ(mask);
        event_status.system[axis] &= ~LIMIT_PLUS_TRIPPED;
        MiscParams[axis].IrqParams[CLEAR_LIMIT_P] &= ~0x04;
        PidError[axis] &= ~mask;
    }
    return(1);
}

static int C_limitM(int axis)
{
    int mask;

    mask = IrqLevelMasks[axis * 5 + 3]; /*  get bit pos of interrupt level  */
    if(RDIRQLEVEL_REG & mask)   /*  is limit switch still on?   */
        return(0);
    else
    {
        mask = IrqClearMasks[axis * 6 + 1]; /*  get bit position of interrupt clear */
        CLEARIRQ_REG = mask;        /*  clear Limit stat    */
        mask = IrqStatMasks[axis * 5 + 1];  /*  get bit pos of interrupt enable */
        if(MiscParams[axis].IrqParams[CLEAR_LIMIT_M] & 0x01)
            EnableHWIRQ(mask);
        event_status.system[axis] &= ~LIMIT_MINUS_TRIPPED;
        MiscParams[axis].IrqParams[CLEAR_LIMIT_M] &= ~0x04;
        PidError[axis] &= ~mask;
    }
    return(1);
}

static int C_fault(int axis)
{
    int mask;

    mask = IrqLevelMasks[axis * 5 + 4]; /*  get bit pos of interrupt level  */
    if(RDIRQLEVEL_REG & mask)
        return(0);
    else
```

```
    {
        mask = IrqClearMasks[axis * 6 + 4]; /*  get bit position of interrupt clear */
        CLEARIRQ_REG = mask;    /*  clear home flag */
        mask = IrqStatMasks[axis * 5 + 4];  /*  get bit pos of interrupt enable */
        if(MiscParams[axis].IrqParams[CLEAR_AMP_FAULT] & 0x01)
            EnableHWIRQ(mask);
        event_status.system[axis] &= ~HOME_TRIPPED;
        MiscParams[axis].IrqParams[CLEAR_AMP_FAULT] &= ~0x04;
        PidError[axis] &= ~mask;
    }
    return(1);
}

static int C_protect(int axis)
{
    int mask;

    mask = IrqLevelMasks[axis * 5 + 1]; /*  get bit pos of interrupt level  */
    if(RDIRQLEVEL_REG & mask)
        return(0);
    else
    {
        mask = IrqClearMasks[axis * 6]; /*  get bit position of interrupt clear */
        CLEARIRQ_REG = mask;    /*  clear home flag */
        mask = IrqStatMasks[axis * 5];  /*  get bit pos of interrupt enable */
        if(MiscParams[axis].IrqParams[CLEAR_PROTECT] & 0x01)
        {
            EnableHWIRQ(mask);
        }
        event_status.system[axis] &= ~PROTECTION_TRIPPED;
        MiscParams[axis].IrqParams[CLEAR_PROTECT] &= ~0x04;
        PidError[axis] &= ~mask;
    }
    return(1);
}

static int C_follow(int axis)
{
    MiscParams[axis].Status &= ~FATAL_FOLLOWING_ERROR;       /* clear flag for pid loop */
    event_status.system[axis] &= ~FOLLOWING_ERROR;    /*    clear flag for system status    */
    return(1);
}

static int C_genfault(int axis)
{
    event_status.system[axis] &= ~GENERAL_FAULT;
    return(1);
}

/*
** Recieve commands
*/

static R_NAK(int axis,int func,AUX a)
{
    return 1;
}

static int R_setkp(int axis, int func,AUX a)
{
    /*  gain value is in aux0   */
    FiltParams[axis].Kp = a.auxi[0];
    return(0);
}

static int R_setkv(int axis, int func,AUX a)
```

```
{
    /*  differential gain in aux0   */
    FiltParams[axis].Kv = a.auxi[0];
    return(0);
}

static int R_setki(int axis, int func,AUX a)
{
    /*  integral gain in aux0   */
    if((event_status.system[axis] & INTEGRATOR_ENABLED) || (axis == 2))
        FiltParams[axis].Ki = a.auxi[0];
    MiscParams[axis].Ki = a.auxi[0];
    return(0);
}

static int R_setkvff(int axis, int func,AUX a)
{
    /* feedforward velocity in aux0     */
    FiltParams[axis].Kvff = a.auxi[0];
    return(0);
}

static int R_setkaff(int axis, int func,AUX aux)
{
    /*  set feed forward acceleration gain  */
    FiltParams[axis].Kaff = aux.auxi[0];
    return(0);
}

static int R_setoffset(int axis, int func,AUX aux)
{
    /*  set servo offset    */
    FiltParams[axis].Offset = aux.auxi[0];
    return(0);
}

static int R_setgoal(int axis, int func,AUX aux)
{
    /*  goal is a long in aux0 & aux1   */
    DesiredGoal[axis] = aux.auxl;
    if(status_queue(message) > 14)
        return(1);  /*  queue is full   */
    else
    {
        put_queue(message,EVENT_PROFILE);
        put_queue(message,axis);
        event_status.pending++;
    }
    return(0);
}

static int R_setmaxvel(int axis, int func,AUX aux)
{
    /*  set maximum velocity    */
    MiscParams[axis].MaximumVel = aux.auxi[0];
    if(status_queue(message) > 14)
        return(1);  /*  queue is full   */
    else
    {
        put_queue(message,EVENT_PROFILE);
        put_queue(message,axis);
        event_status.pending++;
    }
    return(0);
}
```

```
static int R_setscurve(int axis, int func,AUX aux)
{
    /*  set time to get up to speed */
    MiscParams[axis].SCurveTime = aux.auxi[0];
    if(status_queue(message) > 14)
        return(1);  /*  queue is full   */
    else
    {
        put_queue(message,EVENT_PROFILE);
        put_queue(message,axis);
        event_status.pending++;
    }
    return(0);
}

static int R_domove(int axis, int func,AUX aux)
{
    /*  execute profile generator and do move   */
    /*  this command is special, since it generates
    **  an event.  It never completes before this routine
    ** is exited
    */
    int cmd;

    cmd = aux.auxi[1] & 0x03;
    if(!(event_status.system[axis] & GENERAL_FAULT))
    {
        if(event_status.system[axis] & SERVO_ENABLED)
        {
            if(status_queue(message) > 13)
                return(1);  /*  queue is full   */
            else
            {
                    put_queue(message,MOVE_SERVO);  /*  Move Servo Event    */
                    put_queue(message,axis);    /*  axis to move    */
                    put_queue(message,cmd);     /*  move mode   */
            }
            event_status.pending++;
            return(0);              /*  message sent    */
        }
    }
    return(1);      /*  servo is not enabled OR fault   */
}

static int R_abort(int axis, int func,AUX aux)
{
    /*
    ** in order to abort a move, this function will need to do something
    ** totally different now
    */
    MoveAbort[axis] = 1;    /*  set abort flag true */
    AbortMove(axis);        /* special assemlby program to stop move    */
    return(0);
}

static int R_phasepol(int axis, int func,AUX aux)
{
    /*  set the polarity of the phase detector  */
    MiscParams[axis].Polarity = aux.auxi[0];
    RTC_REG = 0x80 | (MiscParams[0].Polarity?0x200:0) | (MiscParams[1].Polarity?0x400:0) | (Misc
Params[2].Polarity?0x200000:0) | (MiscParams[1].SampleRate & 0x7f) | ((MiscParams[0].SampleRate
& 0xf) << 11) | ((MiscParams[2].SampleRate & 0x3f) << 15);
    return(0);
}

static int R_samplerate(int axis, int func,AUX aux)
```

```
{
    /*  Set servo sample rate       */
    int t;

    MiscParams[axis].SampleRate = aux.auxi[0];
    RTC_REG = 0x80 | (MiscParams[0].Polarity?0x200:0) | (MiscParams[1].Polarity?0x400:0) | (Misc
Params[2].Polarity?0x200000:0) | (MiscParams[1].SampleRate & 0x7f) | ((MiscParams[0].SampleRate
& 0xf) << 11) | ((MiscParams[2].SampleRate & 0x3f) << 15);
    return(0);
}

static int R_zeroint(int axis, int func,AUX aux)
{
    /*  reset the integrator accumulator    */
    ZeroIntegrator();
    return(0);
}

static int R_setintmode(int axis, int func,AUX aux)
{
    if(aux.auxi[0] > 0)
    {
        MiscParams[axis].ServoFlags |= PID_DISABLE_INTEGRATOR;
        event_status.system[axis] |= DIS_INT_CMDVNEZ;
    }
    else
    {
        MiscParams[axis].ServoFlags &= ~PID_DISABLE_INTEGRATOR;
        event_status.system[axis] &= ~DIS_INT_CMDVNEZ;
    }
    return(0);
}

static int R_startjog(int axis, int func,AUX aux)
{
    /*  start doing a jog   */
    if(event_status.system[axis] & SERVO_ENABLED)
    {
        if(status_queue(message) > 14)
            return(1);          /*  queue full  */
        else
        {
            put_queue(message,DO_JOG);  /*  jog command */
            put_queue(message,axis);
            event_status.pending++;
            return(0);
        }
    }
    return(1);      /*  servo not enabled   */
}

static int R_endjog(int axis, int func,AUX aux)
{
    /* stop doing a jog */
    if(status_queue(message) > 14)
        return(1);      /*  queue is full   */
    else
    {
        put_queue(message,EVENT_END_JOG);   /*  send end jog message    */
        put_queue(message,axis);
        event_status.pending++;
        return(0);
    }
}

static int R_clearirq(int axis, int func,AUX aux)
```

```
{
    /*  try to clear the beam interrupted interrupt */
    if(func > CLEAR_MAX)
        return(1);
    else
    {
        if( (*c_func[func])(axis) )
            return(0);
    }
    return(1);
}

static int R_servofunc(int axis, int func,AUX aux)
{
    if(EnableFunctions(axis,aux.auxi[0],aux.auxi[1]) )
        return(0);
    return(1);
}

static int R_intlimits(int axis, int func,AUX aux)
{
    /*  set intergrator limits  */
    FiltParams[axis].Kil = aux.auxi[0];
    return(0);
}

static int R_zerocount(int axis, int func,AUX aux)
{
    ZeroCount(axis);
    return(0);
}

static int R_maxfollow(int axis, int func,AUX aux)
{
    /*  set the maximum following error */
    FiltParams[axis].FolLimit = aux.auxi[0];
    return(0);
}

static int R_folgain(int axis, int func,AUX aux)
{
    /*  set the following error DAC gain    */
    MiscParams[axis].FollowGain = aux.auxi[0];
    return(0);
}

static int R_home_params(int axis, int func,AUX aux)
{
    /*
    ** parameter number is in func
    ** parameter is in AUX 0 or l
    */
    switch(func)
    {
        case HOME_FLAGS:
            HomeParams[axis].Flags = aux.auxi[0];
            break;
        case HOME_INITVEL:
            HomeParams[axis].InitVelocity = aux.auxi[0];
            break;
        case HOME_INITDWELL:
            HomeParams[axis].InitDwell = aux.auxi[0];
            break;
        case HOME_BACKOUT:
            HomeParams[axis].Backout = aux.auxi[0];
            break;
```

```
        case HOME_FINEDWELL:
            HomeParams[axis].FineDwell = aux.auxi[0];
            break;
        case HOME_FINEVEL:
            HomeParams[axis].FineVelocity = aux.auxi[0];
            break;
        case HOME_ZERDIST:
            HomeParams[axis].HomingDistance = aux.auxl;
            break;
    }
    return(0);
}

static int R_home(int axis, int func,AUX aux)           /*  go home command */
{
    /*
    ** perform the home operation according to home parrameters
    */
    if(event_status.system[axis] & SERVO_ENABLED)
    {
        if(status_queue(message) > 13)
            return(1);      /*  queue full  */
        else
        {
            put_queue(message,DO_HOME_EVENT);
            put_queue(message,axis);
            put_queue(message,aux.auxi[0]);     /*  type of home command    */
            event_status.pending++;
            return(0);
        }
    }
    return(1);
}

int R_setirq(int axis, int func,AUX aux)
{
    int flag;
    int mask;

    if(aux.auxi[0] >= 8 && aux.auxi[0] < 16)
    {
        flag = 1;       /*  we are going to enable interrupts   */
        aux.auxi[0] -= 8;   /*  make a real interrupt number    */
    }
    else if(aux.auxi[0] < 8)
        flag = 0;       /*  we are going to disable interrupts  */
    else
        return(1);      /*  no interrupt up there, man  */
    if(flag)            /*  enable interrupts   */
    {
        if((aux.auxi[1] & 0x08) && (MiscParams[axis].IrqParams[aux.auxi[0]] & 0x02))
            MiscParams[axis].IrqParams[aux.auxi[0]] &= ~0x02;   /*  exclude actions */
        else if ((aux.auxi[1] & 0x02) && (MiscParams[axis].IrqParams[aux.auxi[0]] & 0x08))
            MiscParams[axis].IrqParams[aux.auxi[0]] &= ~0x08;   /*  exclude actions */
        MiscParams[axis].IrqParams[aux.auxi[0]] |= aux.auxi[1]; /*  set bits    */
        if(MiscParams[axis].IrqParams[aux.auxi[0]] & 0x01)      /*  do we need to enable hardwar
e?  */
            EnableHWIRQ(IrqStatMasks[axis * 5 + aux.auxi[0] ]); /*  enable irq  */
    }
    else                /*  disable interrupts  */
    {
        MiscParams[axis].IrqParams[aux.auxi[0]] &= ~aux.auxi[1];    /*  clear bits  */
        if(MiscParams[axis].IrqParams[aux.auxi[0]] & 0x01 == 0)
            DisableHWIRQ(IrqStatMasks[axis * 5 + aux.auxi[0] ]);    /*  disable IRQ */
    }
    return(0);
```

```
}

static int R_jog_param(int axis, int func,AUX aux)      /*  set the jog parameter          */
{
    /*  aux 0 is index into parameter area  */
    /*  aux 1 is word value                 */
    switch(func)
    {
        case JOGPARAM_START:
            JogParams[axis].Start = aux.auxl;   /*  set parameter   */
            break;
        case JOGPARAM_END:
            JogParams[axis].End = aux.auxl; /*  set parameter   */
            break;
        case JOGPARAM_INDEX:
            JogParams[axis].Index = aux.auxi[0];    /*  set parameter   */
            break;
        case JOGPARAM_MODE:
            JogParams[axis].Mode = aux.auxi[0]; /*  set parameter   */
            break;
        case JOGPARAM_DWELL:
            JogParams[axis].Dwell = aux.auxi[0];    /*  set parameter   */
            break;
    }
    return(0);
}

static int R_inpostion(int axis, int func,AUX aux)
{
    MiscParams[axis].InPosition = aux.auxi[0];
    return(0);
}

static int R_clearflag(int axis,int func,AUX aux)   /*  clear step flag, now used for debug */
{
/*  if(status_queue(message) > 13)  */
/*      return(1);  */    /*  queue full  */
/*  else
    {
        put_queue(message,DODEBUG);
        put_queue(message,axis);
        event_status.pending++;
    }   */
    *((volatile int *)0xffca) = 0;  /* dummy write  */
/*  SetGoal(axis,aux.auxl);     */
    return 0;
}

static int R_recordlinpos(int axis,int func,AUX aux)    /*  record linear position for future re
f   */
{
    int sr;

    if(func == LINEARSTAGE_INPOSENABLE)
    {
        MiscParams[axis].LinInPosEn = aux.auxi[0];  /*  enable/disable function */
    }
    else if(func==LINEARSTAGE_RECORDRUN)
    {
        sr = Disable();
        MiscParams[axis].LinRun = FiltParams[axis].position;
        Enable(sr);
    }
    else
    {
        sr = Disable();
```

```
        MiscParams[axis].LinHome = FiltParams[axis].position;
        Enable(sr);
    }
    return 0;
}

static int R_setfreq(int axis,int func,AUX aux)
{
    FREQUENCY_REG = aux.auxi[0];
    return 0;
}

static int R_setcutofffreq(int  axis,int func,AUX aux)  /*  set the cuttoff frequency of low pas
s filter    */
{
    ANAL *pa = (ANAL *)&FiltParams[axis].Zfc0;  /*  base pointer of ANAL structure  */
    pa[func].MeterFilter = aux.auxi[0];
    return 0;
}

static int R_amplitude(int  axis,int func,AUX aux)
{
    FiltParams[axis].Amplitude = aux.auxi[0];
    return 0;
}

static int R_phase(int axis,int func,AUX aux)           /*  set phase of cosine output  */
{
    MiscParams[axis].Phase = aux.auxi[0];
    WDPHASE_REG = (MiscParams[axis].Phase & 0x0ff) | (MiscParams[axis].DitherMode << 8);
    return 0;
}

static int R_dithermode(int axis,int func,AUX aux)      /*  set mode of dither oscialtor    */
{
    MiscParams[axis].DitherMode = aux.auxi[0];
    WDPHASE_REG = (MiscParams[axis].Phase & 0x0ff) | (MiscParams[axis].DitherMode << 8);
    return 0;
}

static int R_encoderpitch(int axis,int func,AUX aux)    /*  set the encoder pitch   */
{
    MiscParams[axis].EncoderPitch = aux.auxi[0];
    return 0;
}

static int R_lutselect(int axis,int func,AUX aux)       /*  select look up table for dither */
{
    if(aux.auxi[0] == 0)    /*  set default table   */
        MiscParams[axis].RunoutLut = (int *) 0x100;
    else
        MiscParams[axis].RunoutLut = RunoutComp[axis];  /* user table   */
    return 0;
}

static int R_handshake(int axis,int func,AUX aux)       /*  handshake with special seagate thing
y   */
{
    if(aux.auxi[0])
        CLEARIRQ_REG = 0x14000;     /*  set this bit high   */
    else
        CLEARIRQ_REG = 0x10000;     /*  set this bit low    */
    return 0;
}

static int R_histogram(int axis,int func,AUX aux)       /*  start histogram aquisition  */
```

```
{
    int i;

    for(i=0;i<32;++i)
        Histogram[axis].Histogram[i] = 0;
    Histogram[axis].count = aux.auxi[0];    /*  start histogram */
    HistFlags[axis] = 1;
    if(axis != 1)   /*  only do histogram for rotary axis   */
        i = 1;
    else
        i = 0;
    return i;
}

static int R_notchfreq(int axis,int func,AUX aux)       /*  set notch filter frequency  */
{
    /*  create a pointer to the start of the notch filter parameters    */
    NOTCH_PARAMS *n = (NOTCH_PARAMS *)&FiltParams[axis].NotchDepth0;
    n[aux.auxi[1]].NotchFreq = aux.auxi[0]; /*  store frequency data    */
    return 0;
}

static int R_notchQ(int axis,int func,AUX aux)          /*  set notch filter Q  */
{
    /*  create a pointer to the start of the notch filter parameters    */
    NOTCH_PARAMS *n = (NOTCH_PARAMS *)&FiltParams[axis].NotchDepth0;
    n[aux.auxi[1]].NotchQ = aux.auxi[0];    /*  store frequency data    */
    return 0;
}

static int R_notchD(int axis,int func,AUX aux)          /*  set notch filter depth  */
{
    NOTCH_PARAMS *n = (NOTCH_PARAMS *)&FiltParams[axis].NotchDepth0;
    n[aux.auxi[1]].NotchDepth = aux.auxi[0];    /*  store frequency data    */
    return 0;
}


static int R_setanalmode(int axis,int func,AUX aux)     /*  set source of analysis  */
{
    switch(aux.auxi[0])
    {
        case DSP_ANAL_CLOSEDLOOP:
            FiltParams[axis].MeterIn1 = (int *)&filtvals[axis].RelativePosition;
            break;
        case DSP_ANAL_OPENLOOP:
            FiltParams[axis].MeterIn1 = (int *)&filtvals[axis].MotorInput;
            FiltParams[axis].MeterIn0 = (int *)&filtvals[axis].PIDOutput;
            break;
        case DSP_ANAL_NOTCH1:
            FiltParams[axis].MeterIn1 = (int *)&filtvals[axis].Notch1In;
            FiltParams[axis].MeterIn0 = (int *)&filtvals[axis].Notch0In;
            break;
        case DSP_ANAL_NOTCH0:
            FiltParams[axis].MeterIn1 = (int *)&filtvals[axis].Notch0In;
            FiltParams[axis].MeterIn0 = (int *)&filtvals[axis].PIDOutput;
            break;
        case DSP_ANAL_PID:
            FiltParams[axis].MeterIn1 = (int *)&filtvals[axis].FollowError;
            FiltParams[axis].MeterIn0 = (int *)&filtvals[axis].Notch1In;
            break;
    }
    return 0;
}

static int R_PesScale(int axis,int func,AUX aux)        /*  set PES scale for servo on track
```

```
*/
{
    FiltParams[axis].PESScale = aux.auxi[0];
    return 0;
}

static int R_PesLimit(int axis,int func,AUX aux)        /*  set PES limit   */
{
    FiltParams[axis].PESLimit = aux.auxi[0];
    return 0;
}

static int R_ResetPes(int axis,int func,AUX aux)        /*  reset PES counter   */
{
    int rv;

    if( (FPGA_ID) == 0x5aa502)
    {
        rv = 0;
        CLR_PES = 0;
    }
    else rv = 1;
    return rv;
}

static int R_SetPESMode(int axis,int func,AUX aux)      /*  set PES test mode bit   */
{
    int rv = 0;

    if((FPGA_ID) == 0x5aa502)
    {
        WD_PESCR = aux.auxi[0];
    }
    else
        rv = 1;
    return rv;
}

static int R_SetMem(int axis,int func,AUX aux)          /*  set memory location to value    */
{
    switch(func)
    {
        case DSP_PMEM:
            WritePMem((int *)aux.auxi[0],aux.auxi[1]);
            break;
        case DSP_XMEM:
            WriteXMem((int *)aux.auxi[0],aux.auxi[1]);
            break;
        case DSP_YMEM:
            WriteYMem((int *)aux.auxi[0],aux.auxi[1]);
            break;
    }
    return 0;
}

static int R_SetGoalAndMove(int axis,int func,AUX aux)  /*  set goal and move   */
{
    if(status_queue(message) > 12)
        return(1);  /*  queue is full   */
    else
    {
        put_queue(message,EVENT_SETGOALANDMOVE);
        put_queue(message,axis);
        put_queue(message,aux.auxi[0]); /*  goal is in two parts    */
        put_queue(message,aux.auxi[1]); /*  and this is the other part  */
        event_status.pending++;
```

```
    }
    return(0);
}

/*************************************************************\
** Send commands                                             **
\*************************************************************/

static int S_NAK(int **buff,int axis,int func,AUX aux)
{
    return 0;
}

static int S_readpos(int **buff,int axis, int func,AUX aux)
{
    int sr;

    sr = Disable();
    lval = FiltParams[axis].position;
    Enable(sr);
    *buff = (int *)&lval;
    return(2);
}

static int S_readgoal(int **buff,int axis, int func,AUX aux)
{
    /*  read back the desired goal  */
    *buff = (int *)&DesiredGoal[axis];
    return(2);
}

static int S_scurvetime(int **buff,int axis, int func,AUX aux)
{
    /*  read back SCURVE time   */
    *buff = (int *)&MiscParams[axis].SCurveTime;
    return(1);
}

static int S_maxvel(int **buff,int axis, int func,AUX aux)
{
    /*  read back maximum velocity  */
    *buff = (int *)&MiscParams[axis].MaximumVel;
    return(1);
}

static int S_kp(int **buff,int axis, int func,AUX aux)
{
    /*  read back proportional gain     */
    *buff = (int *)&FiltParams[axis].Kp;
    return(1);
}

static int S_kv(int **buff,int axis, int func,AUX aux)
{
    /*  read back differential gain     */
    *buff = (int *)&FiltParams[axis].Kv;
    return(1);
}

static int S_ki(int **buff,int axis, int func,AUX aux)
{
    /*  read back integral gain         */
    *buff = (int *)&MiscParams[axis].Ki;
    return(1);
}
```

```
static int S_kvff(int **buff,int axis, int func,AUX aux)
{
    /*  read back feed forward velocity */
    *buff = (int *)&FiltParams[axis].Kvff;
    return(1);
}

static int S_kaff(int **buff,int axis, int func,AUX aux)
{
    /*  read back feed forward acceleration */
    *buff = (int *)&FiltParams[axis].Kaff;
    return(1);
}

static int S_offset(int **buff,int axis, int func,AUX aux)
{
    /*  read back servo offset          */
    *buff = (int *)&FiltParams[axis].Offset;
    return(1);
}

static int S_status(int **buff,int axis, int func,AUX aux)
{
    /*  read back servo card status     */
    *buff = (int *)&event_status;
    return 6;       /*  status structure is 6 words */
}

static int S_phasepos(int **buff,int axis, int func,AUX aux)
{
    /*  read the phase detector polarity    */
    *buff = (int *)&MiscParams[axis].Polarity;
    return(1);
}

static int S_samplerate(int **buff,int axis, int func,AUX aux)
{
    /*  read the servo sample rate  */
    *buff = (int *)&MiscParams[axis].SampleRate;
    return(1);
}

static int S_intmode(int **buff,int axis, int func,AUX aux)
{
    *buff = &val;
    if(event_status.system[axis] & DIS_INT_CMDVNEZ)
        val = 1;
    else
        val = 0;
    return(1);
}

static int S_daclev(int **buff,int axis, int func,AUX aux)
{
    return(0);          /*  NAK */
}

static int S_intlimit(int **buff,int axis, int func,AUX aux)
{
    /*  read the integrator limits  */
    *buff = (int *)&FiltParams[axis].Kil;
    return(1);  /*  send limits */
}

static int S_maxfollow(int **buff,int axis, int func,AUX aux)
{
```

```
    /*  read the maximum following error    */
    *buff = (int *)&FiltParams[axis].FolLimit;
    return(1);  /*  send following error limit data */
}

static int S_folgain(int **buff,int axis, int func,AUX aux)
{
    /*  get the following error DAC gain    */
    *buff = (int *)&MiscParams[axis].FollowGain;
    return(1);  /*  send following error gain   */
}

static int S_serialnumber(int **buff,int axis, int func,AUX aux)
{
    extern char version[];

    *buff = (int *)version;
    return(16);
}

static int S_home_params(int **buff,int axis, int func,AUX aux)
{
    int retval = 1;
    switch(func)
    {
        case HOME_FLAGS:
            *buff = (int *)&HomeParams[axis].Flags; /*  address of data to send */
            break;
        case HOME_INITVEL:
            *buff = (int *)&HomeParams[axis].InitVelocity;  /*  address of data to send */
            break;
        case HOME_INITDWELL:
            *buff = (int *)&HomeParams[axis].InitDwell; /*  address of data to send */
            break;
        case HOME_BACKOUT:
            *buff = (int *)&HomeParams[axis].Backout;   /*  address of data to send */
            break;
        case HOME_FINEDWELL:
            *buff = (int *)&HomeParams[axis].FineDwell; /*  address of data to send */
            break;
        case HOME_FINEVEL:
            *buff = (int *)&HomeParams[axis].FineVelocity;  /*  address of data to send */
            break;
        case HOME_ZERDIST:
            *buff = (int *)&HomeParams[axis].HomingDistance;    /*  address of data to send */
            retval = 2;
            break;
    }
    return(retval);                         /*  send one word           */
}

static int S_getirq(int **buff,int axis, int func,AUX aux)
{
    *buff = (int *)&MiscParams[axis].IrqParams[aux.auxi[0]];    /*  point to interrupt of interr
est */
    return(1);                          /*  say we gonna send back one word */
}

static int S_readpmem(int **buff,int axis, int func,AUX aux)
{
    val = ReadPmem(aux.auxi[0]);
    *buff = &val;
    return(1);
}

static int S_readymem(int **buff,int axis, int func,AUX aux)
```

```
{
    val = ReadYmem(aux.auxi[0]);
    *buff = &val;
    return(1);
}

static int S_readxmem(int **buff,int axis, int func,AUX aux)
{
    val = ReadXmem(aux.auxi[0]);
    *buff = &val;
    return(1);
}

static int S_caldist(int **buff,int axis, int func,AUX aux)
{
    *buff = (int *)&MiscParams[axis].CalDistance;
    return(2);      /*  this is a long  */
}

static int S_jog_params(int **buff,int axis, int func,AUX aux)
{
    int retval = 1;
    /*  aux 0 is parameter to read  */
    switch(func)
    {
        case JOGPARAM_START:
            *buff = (int *)&JogParams[axis].Start;
            retval = 2;
            break;
        case JOGPARAM_END:
            *buff = (int *)&JogParams[axis].End;
            retval = 2;
            break;
        case JOGPARAM_INDEX:
            *buff = (int *)&JogParams[axis].Index;
            break;
        case JOGPARAM_MODE:
            *buff = (int *)&JogParams[axis].Mode;
            break;
        case JOGPARAM_DWELL:
            *buff = (int *)&JogParams[axis].Dwell;
            break;
    }
    return(retval);
}

static int S_inposition(int **buff,int axis, int func,AUX aux)
{
    *buff = (int *)&MiscParams[axis].InPosition;
    return(1);
}

static int S_getbump(int **buff,int axis, int func,AUX aux)
{
    *buff = (int *)&MiscParams[axis].BumpThresh;
    return (1);
}

static int S_sendsettings(int **buff,int axis,int func,AUX aux)
{
    /* return only PID and MISC params, this changed with version 1.61  */
    *buff = (int *)&FiltParams[0].Zfc0;
    return ((sizeof(PID_PARAMS) + sizeof(PID_STATUS)) * 3);
}

static int S_linearendpoint(int **buff,int axis,int func,AUX aux)
```

```
{
    int retval = 1;

    if(func==LINEARSTAGE_RECORDRUN)
    {
        *buff = (int *)&MiscParams[axis].LinRun;
    }
    else if (func == LINEARSTAGE_RECORDHOME)
    {
        *buff = (int *)&MiscParams[axis].LinHome;
    }
    else
        retval = 0;
    return retval;
}

static int S_frequency(int **buff,int axis,int func,AUX aux)
{
    /*
    ** returns the frequency of internal oscilator
    */
    val = FREQUENCY_REG;
    *buff = (int *)&val;
    return 1;
}

static int S_magnitude(int **buff,int axis,int func,AUX aux)
{
    /*
    ** returns two words, the sine magnitude followed by
    ** the cosine magnitude
    */
    int *a,*b;
    ANAL *pa;
    a = (int *)&lval;
    pa = (ANAL *)&FiltParams[axis].Zfc0;    /*  base pointer of meter params    */
    b = (int *)&(pa[func].ZFc);
    a[0] = b[1];
    a[1] = b[3];
    *buff = a;
    return 2;
}

static int S_cutofffreq(int **buff,int axis,int func,AUX aux)
{
    /*
    ** return the cutoff frequency of the filter
    */
    ANAL *pa = (ANAL *)&FiltParams[axis].Zfc0;  /*  base pointer of meter params    */
    *buff = (int *)&pa[func].MeterFilter;
    return 1;
}

static int S_amplitude(int **buff,int axis,int func,AUX aux)
{
    *buff = ((int *)&FiltParams[axis].Amplitude);
    return 1;
}

static int S_phase(int **buff,int axis,int func,AUX aux)
{
    *buff = ((int *)&MiscParams[axis].Phase);
    return 1;
}

static int S_dithermode(int **buff,int axis,int func,AUX aux)
```

```
{
    *buff = (int *)&MiscParams[axis].DitherMode;
    return 1;
}

static int S_encoderpitch(int **buff,int axis,int func,AUX aux)
{
    *buff = (int *)&MiscParams[axis].EncoderPitch;
    return 1;
}

static int S_runoutcomp(int **buff,int axis,int func,AUX aux)
{
    /*
    ** return back 256 word runout comp table
    */
    *buff = RunoutComp[axis];   /*  address of table    */
    return 256;
}

static int S_lutselect(int **buff,int axis,int func,AUX aux)
{
    if(MiscParams[axis].RunoutLut == (int *)0x100)
        val = 0;        /*  default lookup table    */
    else
        val = 1;        /*  user lookup table       */
    *buff = &val;
    return 1;
}

static int S_histogram(int **buff,int axis,int func,AUX aux)
{
    *buff = (int *)&Histogram[axis].Histogram;
    return 32;
}

static int S_notchfreq(int **buff,int axis,int func,AUX aux)
{
    NOTCH_PARAMS *n = (NOTCH_PARAMS *)&FiltParams[axis].NotchDepth0;
    *buff = (int *)&n[aux.auxi[0]].NotchFreq;
    return 1;
}

static int S_notchQ(int **buff,int axis,int func,AUX aux)
{
    NOTCH_PARAMS *n = (NOTCH_PARAMS *)&FiltParams[axis].NotchDepth0;
    *buff = (int *)&n[aux.auxi[0]].NotchQ;
    return 1;
}

static int S_notchD(int **buff,int axis,int func,AUX aux)
{
    NOTCH_PARAMS *n = (NOTCH_PARAMS *)&FiltParams[axis].NotchDepth0;
    *buff = (int *)&n[aux.auxi[0]].NotchDepth;
    return 1;
}

static int S_PesScale(int **buff,int axis,int func,AUX aux)
{
    *buff = (int *)&FiltParams[axis].PESScale;
    return 1;
}

static int S_PesLimit(int **buff,int axis,int func,AUX aux)
{
    *buff = (int *)&FiltParams[axis].PESLimit;
```

```
    return 1;
}

static int S_FPGAID(int **buff,int axis,int func,AUX aux)
{
    *buff = (int *)FPGA_ID_P;
    return 1;
}

static int S_otherconfig(int **buff,int axis,int func,AUX aux)
{
    *buff = (int *)JogParams;
    return (sizeof(JOG_PARAMS) + sizeof(HOME_PARAMS)) * 3;
}

static int S_datastruct(int **buff,int axis,int func,AUX aux)
{
    int retval=2;

    switch(func)
    {
        case DSP_SEND_PID_PARAMS:
            aval[0] = (int)&FiltParams[axis];
            aval[1] = sizeof(PID_PARAMS);
            break;
        case DSP_SEND_PID_STATUS:
            aval[0] = (int)&MiscParams[axis];
            aval[1] = sizeof(PID_STATUS);
            break;
        case DSP_SEND_PID_PROFILE:
            aval[0] = (int)&filtvals[axis];
            aval[1] = sizeof(PID_PROFILE);
            break;
        case DSP_SEND_JOG_PARAMS:
            aval[0] = (int)&JogParams[axis];
            aval[1] = sizeof(JOG_PARAMS);
            break;
        case DSP_SEND_HOME_PARAMS:
            aval[0] = (int)&HomeParams[axis];
            aval[1] = sizeof(HOME_PARAMS);
            break;
        default:
            retval = 0;
            break;
    }
    *buff = aval;
    return retval;
}

static int S_magnitude1(int **buff,int axis,int func,AUX aux)
{
    /*  this function is similar to S_magnitude, but
    ** it returns back all of Zfc and Zfs, rather than just
    ** the MSB
    */
    ANAL *pa = (ANAL *)&FiltParams[axis].Zfc0;  /*  base pointer of ANAL structure  */
    *buff = (int *)&pa[func].ZFc;
    return 4;
}


/*************************************************************
**
** Functions that recieve a block of data
**
*************************************************************/
```

```
static int RB_config(int **buff,int axis,int func,AUX aux)
{
    /*
    ** This function set things up so that data is stored in
    ** the block of memory that contains all of the configuration
    ** for the servo
    */
    int i;
    int retval = aux.auxi[0];   /*  get word count  */
    E_enable_pid(0,0);
    E_enable_pid(1,0);

    *buff = (int *)FiltParams;  /*  start of config block   */
    GetData(*buff,aux.auxi[0]); /* get data from host   */
    *buff = (int *)JogParams;
    GetData(*buff,aux.auxi[1]);
    /*
    ** ok, fix up data
    */
    MiscParams[1].FollErrReg = (int *)FOLLOWERROR_REG_P;
    MiscParams[0].FollErrReg = (int *)0x5500;
    MiscParams[2].FollErrReg = (int *)0x5501;

    for(i=0;i<3;++i)
    {
        if(MiscParams[i].ServoFlags & EN_FATAL_FOLLOWING_ERROR)
            event_status.system[i] |= F_ERROR_ENABLED;
        else
            event_status.system[i] &= ~F_ERROR_ENABLED;

        if(MiscParams[i].ServoFlags & PID_DISABLE_INTEGRATOR)
            event_status.system[i] |= DIS_INT_CMDVNEZ;
        else
            event_status.system[i] &= ~DIS_INT_CMDVNEZ;

        if(MiscParams[i].ServoFlags & EN_NOTCHFILTER)
            event_status.system[i] |= DSPSTAT_NOTCHFILTER;
        else
            event_status.system[i] &= ~DSPSTAT_NOTCHFILTER;

        MiscParams[i].ServoFlags &= EN_FATAL_FOLLOWING_ERROR | PID_DISABLE_INTEGRATOR | EN_NOTCH
FILTER;
        event_status.system[i] &= ~SERVO_ENABLED;
        MiscParams[i].LinInPosEn = 0;
        FiltParams[i].Flags = FLAGS_END;    /*  make sure we can move   */
        MiscParams[i].RunoutLut = (int *)0x100; /* default look up table    */
        MiscParams[i].AnalInput = (int *)0;     /*  default anal input  */
        MiscParams[i].Status = 0;           /*  zap out status  */
    }
    UpdateHWIRQ();
    RTC_REG = 0x80 | (MiscParams[0].Polarity?0x200:0) | (MiscParams[1].Polarity?0x400:0) | (Misc
Params[2].Polarity?0x200000:0) | (MiscParams[1].SampleRate & 0x7f) | ((MiscParams[0].SampleRate
& 0xf) << 11) | ((MiscParams[2].SampleRate & 0x3f) << 15);
    return aux.auxi[0]+aux.auxi[1];             /*  return back word count  */
}

static int RB_runout(int **buff, int axis, int func, AUX aux)
{
    /*
    ** this function reads 256 bytes into a block that is stored in the
    ** user runout compensation memory.
    */
    /*  get pointer to user comp data for this axis */
    int *r = RunoutComp[axis];
    GetData(r,256);     /*  block is 256 bytes long */
```

```
    return 256;
}

/*************************************************************************
**
** Routines for sending message commands back to host processor
**
*************************************************************************/

void DspException(int subcommand,int aux)
{
    HIcmdDisAble();     /* disable Host Command Interrupt   */
    HIPut(DSP_EXCEPTION);   /*  Send Exception command  */
    HIPut(subcommand);
    HIPut(aux);         /*  aux word, used for Limit Switches   */
    HIcmdEnable();      /* enable Host Command Interrupt    */
}

void DspMoveComplete(int subcommand,int aux)
{
    HIcmdDisAble();     /* disable Host Command Interrupt   */
    HIPut(DSP_MOVECOMPLETE);    /*  Send Move Complete Command  */
    HIPut(subcommand);
    HIPut(aux);         /*  aux word, used to indicate axis     */
    HIcmdEnable();      /* enable Host Command Interrupt    */
}

void DspDebugString(char *s,int l)
{
    /*
    ** This routine cannot be called from within the host
    ** interface interrupt level
    ** If you DO, bad things will happen
    */
    int i;

    HIcmdDisAble(); /*  disable host command interrupt  */
    HIPut(DSP_DEBUGSTRING); /*  send debug string command   */
    HIPut(0);               /*  sub command is zero */
    HIPut(l);               /*  aux is string lenth */
    for(i=0;i<l;++i)
        HIPut(*s++);        /*  send string to host */
    HIcmdEnable();
}
```

```c
/*
** This is the event loop for the servo card
** it looks at a message QUEUE and performs the operations
** dictated by those messages
*/

#include <stdio.h>
#include <stdlib.h>
#include "squeues.h"
#include "sevent.h"
#include "scommand.h"
#include "srtc.h"
#include "spid.h"
#include "sirq.h"
#include "shome.h"
#include "snv.h"
#include "servhw.h"

#define BUMP_STATE_DISABLED         0
#define BUMP_STATE_INPOSITION       1
#define BUMP_STATE_MOVED            2


QUEUE *message;     /*  pointer to message queue    */
volatile STATUS_CLASS event_status;
volatile int LinearState;
volatile int HistFlags[3];  /*  flags for histogram state   */
extern int *RunoutComp[];       /*  pointers for 3 blocks of runout comp    */

static int Between(int axis);

void init_event()
{
    message = make_queue(8);    /*  queue up to 8 messages  */
    event_status.system[0] = 0; /*  initialize status   */
    event_status.system[1] = 0;
    event_status.system[2] = 0;
    /*
    ** yep, the following addresses are hard coded.
    ** This will hopefully be OK
    ** This saves about 2300 bytes of ROM
    ** Also, 0x1000 allows for pleanty of growing space in the
    ** y memory
    */

    RunoutComp[0] = (int *)0x1000;
    RunoutComp[1] = (int *)0x1100;
    RunoutComp[2] = (int *)0x1200;
}

int MoveState[3]={0,0,0};

void event_loop()
{
    /*
    ** this is the main event loop
    ** this is where the program will execute from in the forground
    **
    ** it gets is messages from a message queue and uses that as an
    ** index into the event function table to decide what it will execute
    */
    int index,t;
    long steps;
    long delta_goal;
    int loop_flag;
    long start,end;
```

```
    int following_error;
    int axis;
    int sr;
    TEL_PROFILE jog;
    union {
        long lv;
        int iv[2];
    }convert;

    while(1)    /*  we do this forever and never leave until we die */
    {
        if(status_queue(message) )  /*  any thing is message queue? */
        {
            index = get_queue(message); /*  get message from queue  */
            axis = get_queue(message);
            if((index & 0x0ff) < MAX_EVENTS)
            {
                HIcmdDisAble();
                event_status.status = EVENT_IN_PROGRESS;
                event_status.type = index & 0xff;
                HIcmdEnable();
                switch(index & 0x0ff)
                {
                    case MOVE_SERVO:    /*  this is a RELATIVE move */
                        while(FiltParams[axis].Flags != FLAGS_END); /*  wait for move to finish
*/
                        if((t = get_queue(message)) == 0)       /*  get move mode   */
                        {
                            /*  absolute move   */
                            if((delta_goal = DesiredGoal[axis]-GetGoal(axis)) != 0)
                            {
                                TelProfile(MiscParams[axis].SCurveTime,delta_goal,MiscParams[axi
s].MaximumVel,&jog);
                                sr = Disable();     /*  disable all interrupts  */
                                FiltParams[axis].Flags = FLAGS_START;   /*  start profile genera
tor */
                                FiltParams[axis].Jerk = jog.Jerk;   /*  set jerk value  */
                                FiltParams[axis].VCount = jog.VSteps;   /*  set number of const
vel steps   */
                                FiltParams[axis].SCount = jog.SSteps;   /*  set number of S Curv
e steps */
                                event_status.system[axis] &= ~MOVE_IDLE;
                                Enable(sr); /*  re-Enable all interrupts    */
                            }
                        }
                        else    /*  relative move   */
                        {
                            sr = Disable(); /*  disable all interrupts  */
                            FiltParams[axis].Flags = FLAGS_START;
                            FiltParams[axis].Jerk = MoveProfiles[axis].Jerk;
                            FiltParams[axis].VCount = MoveProfiles[axis].VSteps;
                            FiltParams[axis].SCount = MoveProfiles[axis].SSteps;
                            event_status.system[axis] &= ~MOVE_IDLE;
                            Enable(sr); /*  enable all interrupts   */
                        }
                        MoveState[axis]=1;
                        HIcmdDisAble();
                        event_status.pending--;
                        HIcmdEnable();
                        break;
                    case DO_JOG:
                        /* This event is going to be a little tricky    */
                        /* It is going to continuously move back and
                       ** forth, and monitor the event queue for a
                       ** message to tell it to quit.
                       ** it will not quit until the current move
```

```
                    ** is over
                    */
                     if(!(JogParams[axis].Mode & JOG_DIRECTION))
                     {       /*  reverse */
                         start = JogParams[axis].End;
                         end = JogParams[axis].Start;
                     }
                     else    /*  normal  */
                     {
                         start = JogParams[axis].Start;
                         end = JogParams[axis].End;
                     }
                     if(start > end)
                     {
                         steps = -(long)JogParams[axis].Index;
                     }
                     else
                     {
                         steps = (long)JogParams[axis].Index;
                     }
                     TelProfile(MiscParams[axis].SCurveTime,steps,MiscParams[axis].MaximumVel
,&jog);

                     loop_flag = 1;
                     do
                     {
                         /*
                        ** start doing the jog
                        */
                         sr = Disable(); /*  disable all interrupts  */
                         FiltParams[axis].Flags = FLAGS_START;
                         FiltParams[axis].Jerk = jog.Jerk;
                         FiltParams[axis].VCount = jog.VSteps;
                         FiltParams[axis].SCount = jog.SSteps;
                         Enable(sr); /*  enable all interrupts   */
                         /*  wait for move to complete   */
                         SetTP12();
                         while(!(FiltParams[axis].Flags & FLAGS_END));
                         ClearTP12();
                         RT_COUNT = JogParams[axis].Dwell;
                         do
                         {
                             if(status_queue(message) )
                             {
                                 index = get_queue(message);
                                 axis = get_queue(message);
                                 if(index == EVENT_END_JOG)
                                 {
                                     HIcmdDisAble();
                                     event_status.pending--;
                                     HIcmdEnable();
                                     loop_flag = 0;
                                     RT_COUNT = 0;
                                 }
                             }
                         }while(RT_COUNT > 0);
                         if(!Between(axis))
                         {
                             steps = - steps;
                             TelProfile(MiscParams[axis].SCurveTime,steps,MiscParams[axis].Ma
ximumVel,&jog);

                         }
                     }while(loop_flag);
                     HIcmdDisAble();
                     event_status.pending--;
                     HIcmdEnable();
```

```
                        break;
                    case DODEBUG:
                        HIcmdDisAble();
                        event_status.pending--;
                        HIcmdEnable();
                        break;
                    case DO_HOME_EVENT:     /*  try to locate home  */
                        Home(axis,get_queue(message));
                        HIcmdDisAble();
                        event_status.pending--;
                        HIcmdEnable();
                        break;
                    case END_JOG:
                        HIcmdDisAble();
                        event_status.pending--;
                        HIcmdEnable();
                        break;
                    case EVENT_PROFILE:
                        if(axis < 2)    /*   normal profile */
                            TelProfile(MiscParams[axis].SCurveTime,DesiredGoal[axis],MiscParams[
axis].MaximumVel,&MoveProfiles[axis]);
                        else    /*  profile for spindle motor   */
                            TelSpinProfile(MiscParams[axis].SCurveTime,DesiredGoal[axis],MiscPar
ams[axis].MaximumVel,&MoveProfiles[axis]);
                        HIcmdDisAble();
                        event_status.pending--;
                        HIcmdEnable();
                        break;
                    case EVENT_SETGOALANDMOVE:
                        while(FiltParams[axis].Flags != FLAGS_END); /*  wait for move to finish
*/
                        convert.iv[0] = get_queue(message);
                        convert.iv[1] = get_queue(message);
                            /*  absolute move   */
                        if((delta_goal = convert.lv-GetGoal(axis)) != 0)
                        {
                            TelProfile(MiscParams[axis].SCurveTime,delta_goal,MiscParams[axis].M
aximumVel,&jog);
                            sr = Disable();     /*  disable all interrupts  */
                            FiltParams[axis].Flags = FLAGS_START;   /*  start profile generator
*/
                            FiltParams[axis].Jerk = jog.Jerk;   /*  set jerk value  */
                            FiltParams[axis].VCount = jog.VSteps;   /*  set number of const vel
steps   */
                            FiltParams[axis].SCount = jog.SSteps;   /*  set number of S Curve st
eps */
                            event_status.system[axis] &= ~MOVE_IDLE;
                            Enable(sr); /*  re-Enable all interrupts    */
                        }
                        MoveState[axis]=1;
                        HIcmdDisAble();
                        event_status.pending--;
                        HIcmdEnable();
                        break;
                }   /*  end switch index    */
                HIcmdDisAble();
                event_status.status = 0;
                event_status.type = 0;
                HIcmdEnable();
            }
        }
        /*
        ** Check following Error flag in ServoFlags
        ** and set status appropriately
        */
        for(axis=0;axis<3;++axis)
```

```
        {
            HIcmdDisAble();
            if(MiscParams[axis].Status & FATAL_FOLLOWING_ERROR)
            {
                sr = Disable();
                event_status.system[axis] |= FOLLOWING_ERROR;   /*  set flag    */
                Enable(sr);
                if(event_status.system[axis] & F_ERROR_ENABLED)
                {
                    Disable_Amp(axis);
                    MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
                    sr = Disable();
                    event_status.system[axis] &= ~SERVO_ENABLED;
                    Enable(sr);
                }
            }
            /*
            ** check the following error and see if we are in position
            */
            if((following_error = MiscParams[axis].FollowError ) < 0)
                following_error = - following_error;    /*  absolute value  */
            if(following_error < MiscParams[axis].InPosition)           /*  are we in position?
*/
            {
                sr = Disable();
                event_status.system[axis] |= SERVO_IN_POSITION;
                Enable(sr);
            }
            else
            {
                sr = Disable();
                event_status.system[axis] &= ~SERVO_IN_POSITION;
                Enable(sr);
            }
            HIcmdEnable();
        }   /*  end of for loop */
        sr = Disable(); /* disable all interrupts   */
        start = FiltParams[0].position; /* get linear axis position */
        Enable(sr);     /*  enable all interrupts   */
        if(MiscParams[0].LinInPosEn)    /*  check to see if linear stage is enabled */
        {
            switch(LinearState)
            {
                case 0:     /* initial state    */
                    end = start - MiscParams[0].LinHome;
                    if(end < 0) end = -end;
                    if(end < 100)   /*  is position within threshold    */
                    {
                        LinearState = 1;    /*  set to next state   */
                        DspMoveComplete(DSPMOVE_LINEARHOME,0);
                    }
                    else
                    {
                        end = start - MiscParams[0].LinRun;
                        if(end < 0) end = -end;
                        if(end < 100)   /*  is position within threshold?   */
                        {
                            LinearState = 2;
                            DspMoveComplete(DSPMOVE_LINEARRUN,0);
                        }
                    }
                    break;
                case 1:     /*  we are at linera home   */
                    end = start - MiscParams[0].LinHome;
                    if(end < 0) end = -end;
                    if(end > 200)
```

```
                        LinearState = 0;    /*  between, find out where we are  */
                    break;
                case 2:     /*  we are at linear run    */
                    end = start - MiscParams[0].LinRun;
                    if(end < 0) end = -end;
                    if(end > 200)
                        LinearState = 0;    /*  between, find out where we are  */
                    break;
            }   /*  end of switch Linear State  */
        }
        for(axis=0;axis<3;++axis)
        {
            if(MoveState[axis])
            {
                sr = Disable(); /* disable all interrupts   */
                index = FiltParams[axis].Flags;
                Enable(sr);     /*  enable all interrupts   */
                if(index == FLAGS_END)  /*  index contains profile flags    */
                {
                    DspMoveComplete(DSPMOVE_MOVECOMPLETE,axis);
                    MoveState[axis]=0;
                    sr = Disable();
                    event_status.system[axis] |= MOVE_IDLE;
                    Enable(sr);
                }
            }
        }
        for(axis=0;axis<3;++axis)
        {
            sr = Disable(); /*  disable interrupts  */
            if(HistFlags[axis])
            {
                if(Histogram[axis].count == 0)
                {
                    /* indicate histogram is done   */
                    DspException(DSPEXCEP_HISTOGRAM,axis);
                    HistFlags[axis] = 0;    /*  reset histogrm flag */
                }
            }
            Enable(sr);     /*  enable interrupts   */
        }
    }   /*  end of while(1) */
}


static int Between(int axis)
{
    long Goal,*start,*end,*swap;

    start = (long *)&JogParams[axis].Start;
    end = (long *)&JogParams[axis].End;
    Goal = GetGoal(axis);
    if (*end < *start)
    {
        swap = start;
        start = end;
        end = swap;
    }
    if((*start < Goal) && (Goal < *end) )
        return(1);
    return(0);
}
```

```c
/*
** this routine is used to perform the homing operation
**
*/
#include <stdio.h>
#include <stdlib.h>
#include "scommand.h"
#include "spid.h"   /*  for external filter parameters  */
#include "sevent.h"  /* for externals of event manager  */
#include "srtc.h"
#include "shome.h"
#include "sirq.h"
#include "servhw.h"

static int motor_current;
extern void E_enable_pid(int,int);
extern volatile int MoveAbort[3];   /*  flags indicate when an Abort occured    */

static int Limit_LimitPlus(int axis);
static int Limit_LimitMinus(int axis);
static int Limit_LimitHome(int axis);
static int Limit_Dac(int axis);

extern int (*c_func[9])(int axis);  /*  functions that clear various flags  */
                                    /*  returns 1 if function was able to clear flag    */
                                    /*  returns 0 if function was not able to clear flag    */


static int (*LimFuncs[])(int axis) = {
    NULL,
    Limit_LimitPlus,
    Limit_LimitMinus,
    Limit_LimitHome,
    Limit_Dac
};

typedef union {
        long v;
        int d[2];
    }CONVERT;

static int dummy(int axis)
{
    return(1);  /*  make Home think flag was always cleared */
}

static long SeekInc(int axis,int val)
{
    long pos_inc;

    if(HomeParams[axis].Flags & HOME_POSITIVE)  /*  which direction?    */
        pos_inc = (long)val << 24;
    else
        pos_inc = -(long)val << 24;
    return(pos_inc);
}

static int SlowMove(int axis,long start_pos,long pos_inc,long distance,int dwell,int (*clear_fun
c)(int axis))
{
    long x;

    SetVelocity(axis,pos_inc);  /*  set volocity and start move */
    do      /*  back away from stop     */
    {
        x = GetGoal(axis);
```

```
/*      if(pos_inc > 0)
            distance = x - *start_pos;
        else
            distance = *start_pos - x;*/
         if(MoveAbort[axis])
             return 1;   /*  indicate that move was aborted  */
    }while(distance > labs(x-start_pos));
    SetVelocity(axis,0l);   /*  stop move   */
    (*clear_func)(axis);                        /*  clear flag  */
    RT_COUNT = dwell;

    while((RT_COUNT > 0))       /*  wait dwell time (in sample time units   */
         if(MoveAbort[axis])
             return 1;
    return 0;           /*  normal return   */
}

static int Seek(int axis,long pos_inc, int dwell, int (*limit_func)(int axis),int (*clear_func)(
int axis) )
{
    int loop = 1;

    (*clear_func)(axis);
    SetVelocity(axis,pos_inc);  /*  start move  */
    do                          /*  initially locate stop   */
    {
         if((*limit_func)(axis) )
             loop = 0;
         if(MoveAbort[axis])
         {
             return 1;          /*  indicate that home was aborted  */
         }
    }while(loop);
    SetVelocity(axis,0l);       /*  stop move   */

    RT_COUNT = dwell;       /*  set the dwell time      */

    while((RT_COUNT > 0))       /*  wait dwell time (in sample time units   */
         if(MoveAbort[axis])
             return 1;
    return 0;               /*  normal exit     */
}


static int Calibrate(int axis)
{
    long pos1,pos2,home;        /*  end positions   */
    int (*lim_p)(int axis);
    int (*lim_m)(int axis);
    int (*clr_p)(int axis);
    int (*clr_m)(int axis);
    long pos_inc,start_pos;

    switch(HomeParams[axis].Flags & 0x0f)
    {
         case HOME_NONE:     /*  not valid   */
             return 2;
         case HOME_LIMIT_PLUS:       /*  Limit Plus  */
         case HOME_LIMIT_MINUS:      /*  Limit Minus , both the same */
             lim_m = LimFuncs[2];
             lim_p = LimFuncs[1];
             clr_p = c_func[CLEAR_LIMIT_P];
             clr_m = c_func[CLEAR_LIMIT_M];
             break;
         case HOME_DAC:
             lim_p = LimFuncs[4];
```

```
            clr_p = dummy;
            lim_m = LimFuncs[4];
            clr_m = dummy;
            break;
        default:    /*  not valid   */
            return 2;
    }
    pos_inc =-( (long)( HomeParams[axis].InitVelocity) << 24);

    MoveAbort[axis] = 0;
    motor_current = (HomeParams[axis].Flags & MOTOR_CURRENT_MASK) >> MOTOR_CURRENT_SHIFT;
    if((HomeParams[axis].Flags & 0x0f) == HOME_DAC)
        motor_current *= 255;       /*  scale this number for DAC   */
    if(Seek(axis,pos_inc,HomeParams[axis].InitDwell,lim_m,clr_m))
        return 1;
    if(SlowMove(axis,GetGoal(axis),-pos_inc,(long)HomeParams[axis].Backout,HomeParams[axis].Init
Dwell,clr_m))
        return 3;
    if(Seek(axis,-((long)(HomeParams[axis].FineVelocity) << 24),HomeParams[axis].FineDwell,lim_m
,clr_m) )
        return 4;
    pos1 = FiltParams[axis].position;       /*  record this position    */

    if(SlowMove(axis,GetGoal(axis),-pos_inc,(long)HomeParams[axis].Backout,HomeParams[axis].Init
Dwell,clr_m))
        return 5;

    if(Seek(axis,-pos_inc,HomeParams[axis].InitDwell,lim_p,clr_p))
        return 6;
    if(SlowMove(axis,GetGoal(axis),pos_inc,(long)HomeParams[axis].Backout,HomeParams[axis].InitD
well,clr_p))
        return 7;
    if(Seek(axis,((long)(HomeParams[axis].FineVelocity) << 24),HomeParams[axis].FineDwell,lim_p,
clr_p) )
        return 8;
    pos2 = FiltParams[axis].position;       /*  record this position    */

    home = pos1 + pos2;
    home /= 2l;
    MiscParams[axis].CalDistance = labs(pos2-pos1);
    HomeParams[axis].HomingDistance = MiscParams[axis].CalDistance >> 1;

    pos1 = home - GetGoal(axis);
    MoveRel(axis,pos1);
    RT_COUNT = HomeParams[axis].FineDwell;

    while((RT_COUNT > 0))       /*  wait dwell time (in sample time units   */             /*  retu
rn on Abort     */
        if(MoveAbort[axis])
            return 9;

    ZeroCount(axis);            /*  we are home, zero count here    */
    (*clr_m)(axis);                 /*  clear flag  */
    (*clr_p)(axis);
    return(0);
}

static int LimpSave;
static int LimmSave;
static int HomeSave;

void Home(int axis,int cmd)     /*  called from the event manager   */
{
    long pos_inc,start_pos;
    int (*limit_func)(int axis);    /*  function to determine if limit was hit  */
    int (*clear_func)(int axis);    /*  function used to clear limit flag   */
```

```
    long home_position;         /*  position to force when limit encountered    */

    MoveAbort[axis]=0;  /*  clear move abort flag   */
    LimpSave = MiscParams[axis].IrqParams[IVECT_LIMIT_P];   /*  save IRQ flags  */
    LimmSave = MiscParams[axis].IrqParams[IVECT_LIMIT_M];
    HomeSave = MiscParams[axis].IrqParams[IVECT_HOME];
    MiscParams[axis].IrqParams[IVECT_LIMIT_P] &= ~(0x0a);
    MiscParams[axis].IrqParams[IVECT_LIMIT_M] &= ~(0x0a);
    MiscParams[axis].IrqParams[IVECT_HOME] &= ~(0x0a);

    if(cmd == HOME_DOCAL)
    {
        if(Calibrate(axis))     /*  we want to do a calibrate first */
            goto exit;
    }

    /*
    ** we use a very simple crude profile generator, for speed
    ** to move the plant until a stop is hit.
    */

    /*
    ** first figure out how far to move each sample period, and which way
    */
    switch(HomeParams[axis].Flags & 0x0f)
    {
        case HOME_NONE:     /*  not valid   */
            goto exit;
        case HOME_LIMIT_PLUS:       /*  Limit Plus  */
            limit_func = LimFuncs[1];
            clear_func = c_func[CLEAR_LIMIT_P];
            home_position = MiscParams[axis].CalDistance / 2l;
            break;
        case HOME_LIMIT_MINUS:      /*  Limit Minus */
            limit_func = LimFuncs[2];
            clear_func = c_func[CLEAR_LIMIT_M];
            home_position =  MiscParams[axis].CalDistance / -2l;
            break;
        case HOME_DAC:
            limit_func = LimFuncs[4];
            clear_func = dummy;
            if(HomeParams[axis].Flags & HOME_POSITIVE)
                home_position = MiscParams[axis].CalDistance/2l;
            else
                home_position = MiscParams[axis].CalDistance/-2l;
            break;
        default:    /*  not valid   */
            goto exit;
    }

    (*clear_func)(axis);                    /*  clear flag  */
    pos_inc = SeekInc(axis,HomeParams[axis].InitVelocity);
    motor_current = (HomeParams[axis].Flags & MOTOR_CURRENT_MASK) >> MOTOR_CURRENT_SHIFT;
    if((HomeParams[axis].Flags & 0x0f) == HOME_DAC)
        motor_current *= 256;

    if(Seek(axis,pos_inc,HomeParams[axis].InitDwell,limit_func,clear_func) )
        goto exit;

    if(SlowMove(axis,GetGoal(axis),-pos_inc,(long)HomeParams[axis].Backout,HomeParams[axis].Fine
Dwell,clear_func) )
        goto exit;

    if(Seek(axis,SeekInc(axis,HomeParams[axis].FineVelocity),HomeParams[axis].FineDwell,limit_fu
nc,clear_func) )
        goto exit;
```

```
    SetGoal(axis,home_position);

    if(HomeParams[axis].HomingDistance == 0)
    {
        /*
        ** if the backout distance is zero, then we look for the home
        ** limit switch
        */

/*      (*c_func[CLEAR_HOME])(axis);    */        /*  clear home flag */
/*      if(SlowMove(axis,&start_pos,-pos_inc,(long)HomeParams[axis].Backout,HomeParams[axis].Fin
eDwell,clear_func) )
            goto exit;
        if(Seek(axis,&start_pos,-SeekInc(axis,HomeParams[axis].FineVelocity),HomeParams[axis].Fi
neDwell,Limit_LimitHome,clear_func))
            goto exit;  */
    }
    else
    {
        if(HomeParams[axis].Flags & HOME_POSITIVE)
            MoveRel(axis,-HomeParams[axis].HomingDistance);
        else
            MoveRel(axis,HomeParams[axis].HomingDistance);
    }
    (*clear_func)(axis);                        /*  clear flag  */
exit:
    MiscParams[axis].IrqParams[IVECT_LIMIT_P] = LimpSave;   /*  save IRQ flags  */
    MiscParams[axis].IrqParams[IVECT_LIMIT_M] = LimmSave;
    MiscParams[axis].IrqParams[IVECT_HOME] = HomeSave;
    MoveAbort[axis] = 0;    /*  clear abort flag in case it was set */
    if(cmd == HOME_DOCAL)
        DspMoveComplete(DSPMOVE_CALIBRATECOMPLETE,axis);
    else
        DspMoveComplete(DSPMOVE_HOMECOMPLETE,axis);
}

static int Dac_Desc[3] = {DREG_LINEAR,DREG_ROTARY,DREG_SPINDLE};

static int Limit_Dac(int axis)
{
    int d;
    int sr;

    sr = Disable(); /*  disable interrupts  */
    AXIS_DESC_REG = Dac_Desc[axis];
    /*  This function checks the motor DAC level as the limit   */
    d = DAC_REG & 0xffff00; /*  get value from DAC  */
    Enable(sr);     /*  enable interrupts   */
    d >>= 8;    /*  sign extend */
    d = abs(d);
    if(d > motor_current)
        return 1;
    return 0;
}

static int Limit_LimitPlus(int axis)
{
    if(MiscParams[axis].IrqParams[CLEAR_LIMIT_P] & 0x04)
        return(1);
    return(0);
}

static int Limit_LimitMinus(int axis)
{
    if(MiscParams[axis].IrqParams[CLEAR_LIMIT_M] & 0x04)
```

```
        return(1);
    return(0);
}

static int Limit_LimitHome(int axis)
{
/*  if(MiscParams[axis].IrqParams[CLEAR_HOME] & 0x04)
        return(1);  */
    return(0);
}
```

```
/*
** this file contains the code that services hardware specific interrupts
*/

#include "sirq.h"
#include "sevent.h"
#include "spid.h"
#include "scommand.h"
#include "servhw.h"

typedef union {
    int auxi[2];
    long auxl;
}AUX;       /*  place to store AUX information  */

static int IrqEnShad = 0;
int PidError[3] = {1,1,1};
extern int R_setirq(int axis,int func,AUX a);

HandleHardware(int flags)
{
    /*
    ** flags contains the contents of the interrupt status register
    */
    int axis = 0;

    if(flags & BEAM1_IRQSTAT)   /*  was it a Beam interrupt?    */
    {
        /*
        ** a beam interrupt is supposed to be a real nasty event
        ** one of the things that needs to happen is to have the
        ** servo shut down, pronto
        ** need to set a flag for the event manager
        ** set some sort of status
        */
        event_status.system[axis] |= BEAM_INTERRUPTED;
        DisableHWIRQ(BEAM1_IRQSTAT);    /*  disable further beam interrupted interrupts */
        MiscParams[axis].IrqParams[IVECT_BEAM] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_BEAM] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            Disable_Amp(axis);
            PidError[axis] |= BEAM1_IRQSTAT;
        }
        else if (MiscParams[axis].IrqParams[IVECT_BEAM] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
        DspException(DSPEXCEP_BEAMINTERRUPTED,0);
    }
    if(flags & LIMN1_IRQSTAT)
    {
        /*
        ** set flag indicating that the Negative Limit has been hit
        */
        event_status.system[axis] |= LIMIT_MINUS_TRIPPED;
        DisableHWIRQ(LIMN1_IRQSTAT);    /*  disable further LIMIT exceptions    */
        MiscParams[axis].IrqParams[IVECT_LIMIT_M] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_LIMIT_M] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            PidError[axis] |= LIMN1_IRQSTAT;
            Disable_Amp(axis);
```

```
        }
        else if (MiscParams[axis].IrqParams[IVECT_LIMIT_M] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
        if(MiscParams[axis].IrqParams[IVECT_LIMIT_P] & HDWIRQ_MOVE)
        {
            /*
            ** if this bit is set, then use limit switches to indicate
            ** when linear stage is in position
            */
            DspMoveComplete(DSPMOVE_LINEARHOME,0);
            if(MiscParams[axis].IrqParams[CLEAR_LIMIT_P] & HDWIRQ_ENABLE)
                EnableHWIRQ(LIMP1_IRQSTAT);
            event_status.system[axis] &= ~LIMIT_PLUS_TRIPPED;
            MiscParams[axis].IrqParams[CLEAR_LIMIT_P] &= ~HDWIRQ_IRQSTAT;
            CLEARIRQ_REG = LIMP1_IRQCLEAR;  /*  clear positive limit interrupt  */
        }
        else
        {
            DspException(DSPEXCEP_LIMITHIT,DSPEXAUX_NEGATIVELIMIT);
        }
    }
    if(flags & LIMP1_IRQSTAT)
    {
        /*
        ** set the flag intidacting that the positive limit has been hit
        */
        /*************************************************************
        ** This function now takes on special meaning
        ** if the compiler flag SEAGATE is on, then this IRQ is
        ** going to be used to send a message indicating that the user
        ** is requesting the use of the power amp and micro E encoder
        **
        *************************************************************/
        event_status.system[axis] |= LIMIT_PLUS_TRIPPED;
        DisableHWIRQ(LIMP1_IRQSTAT);    /*  disable further LIMIT exceptions    */
        MiscParams[axis].IrqParams[IVECT_LIMIT_P] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_LIMIT_P] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            PidError[axis] |= LIMP1_IRQSTAT;
            Disable_Amp(axis);
        }
        else if (MiscParams[axis].IrqParams[IVECT_LIMIT_P] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
        if(MiscParams[axis].IrqParams[IVECT_LIMIT_P] & HDWIRQ_MOVE)
        {
            /*
            ** if this bit is set, then use limit switches to indicate
            ** when linear stage is in position
            */
            DspMoveComplete(DSPMOVE_LINEARRUN,0);
            if(MiscParams[axis].IrqParams[CLEAR_LIMIT_M] & HDWIRQ_ENABLE)
                EnableHWIRQ(LIMN1_IRQSTAT);
            event_status.system[axis] &= ~LIMIT_MINUS_TRIPPED;
            MiscParams[axis].IrqParams[CLEAR_LIMIT_M] &= ~HDWIRQ_IRQSTAT;
            CLEARIRQ_REG = LIMN1_IRQCLEAR;  /*  clear negative limit interrupt  */
        }
        else
        {
```

```
            DspException(DSPEXCEP_LIMITHIT,DSPEXAUX_POSITIVELIMIT);
        }
    }
    if(flags & FAULT1_IRQSTAT)
    {
        /*
        ** Set the flag indicating that the Amplifier has faulted
        */
        event_status.system[axis] |= AMP_FAULT_TRIPPED;
        DisableHWIRQ(FAULT1_IRQSTAT);   /*  disable further AMP exceptions  */
        MiscParams[axis].IrqParams[IVECT_AMP_FAULT] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_AMP_FAULT] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            PidError[axis] |= FAULT1_IRQSTAT;
            Disable_Amp(axis);
        }
        else if (MiscParams[axis].IrqParams[IVECT_AMP_FAULT] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
    }
    if(flags & PROT1_IRQSTAT)
    {
        /*
        ** Set the flag that a PROTECTION fault has occured
        */
        event_status.system[axis] |= PROTECTION_TRIPPED;
        DisableHWIRQ(PROT1_IRQSTAT);    /*  disable further PROTECTION exceptions   */
        MiscParams[axis].IrqParams[IVECT_PROTECT] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_PROTECT] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            PidError[axis] |= PROT1_IRQSTAT;
            Disable_Amp(axis);
        }
        else if (MiscParams[axis].IrqParams[IVECT_PROTECT] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
    }
    axis = 1;
    if(flags & BEAM2_IRQSTAT)   /*  was it a Beam interrupt?    */
    {
        /*
        ** a beam interrupt is supposed to be a real nasty event
        ** one of the things that needs to happen is to have the
        ** servo shut down, pronto
        ** need to set a flag for the event manager
        ** set some sort of status
        */
        event_status.system[axis] |= BEAM_INTERRUPTED;
        DisableHWIRQ(BEAM2_IRQSTAT);    /*  disable further beam interrupted interrupts */
        MiscParams[axis].IrqParams[IVECT_BEAM] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_BEAM] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            Disable_Amp(axis);
            PidError[axis] |= BEAM2_IRQSTAT;
        }
        else if (MiscParams[axis].IrqParams[IVECT_BEAM] & HDWIRQ_ABORT)
```

```
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
        DspException(DSPEXCEP_BEAMINTERRUPTED,1);
    }
    if(flags & LIMN2_IRQSTAT)
    {
        /*
        ** set flag indicating that the Negative Limit has been hit
        */
        event_status.system[axis] |= LIMIT_MINUS_TRIPPED;
        DisableHWIRQ(LIMN2_IRQSTAT);    /*  disable further LIMIT exceptions    */
        MiscParams[axis].IrqParams[IVECT_LIMIT_M] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_LIMIT_M] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            PidError[axis] |= LIMN2_IRQSTAT;
            Disable_Amp(axis);
        }
        else if (MiscParams[axis].IrqParams[IVECT_LIMIT_M] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
    }
    if(flags & LIMP2_IRQSTAT)
    {
        /*
        ** set the flag intidacting that the positive limit has been hit
        */
        event_status.system[axis] |= LIMIT_PLUS_TRIPPED;
        DisableHWIRQ(LIMP2_IRQSTAT);    /*  disable further LIMIT exceptions    */
        MiscParams[axis].IrqParams[IVECT_LIMIT_P] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_LIMIT_P] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            PidError[axis] |= LIMP2_IRQSTAT;
            Disable_Amp(axis);
        }
        else if (MiscParams[axis].IrqParams[IVECT_LIMIT_P] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
    }
    if(flags & FAULT2_IRQSTAT)
    {
        /*
        ** Set the flag indicating that the Amplifier has faulted
        */
        event_status.system[axis] |= AMP_FAULT_TRIPPED;
        DisableHWIRQ(FAULT2_IRQSTAT);   /*  disable further AMP exceptions  */
        MiscParams[axis].IrqParams[IVECT_AMP_FAULT] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_AMP_FAULT] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            PidError[axis] |= FAULT2_IRQSTAT;
            Disable_Amp(axis);
        }
        else if (MiscParams[axis].IrqParams[IVECT_AMP_FAULT] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
```

```
            event_status.system[axis] |= GENERAL_FAULT;
        }
    }
    if(flags & PROT2_IRQSTAT)
    {
        /*
        ** Set the flag that a PROTECTION fault has occured
        */
        event_status.system[axis] |= PROTECTION_TRIPPED;
        DisableHWIRQ(PROT2_IRQSTAT);    /*  disable further PROTECTION exceptions   */
        MiscParams[axis].IrqParams[IVECT_PROTECT] |= HDWIRQ_IRQSTAT;
        if(MiscParams[axis].IrqParams[IVECT_PROTECT] & HDWIRQ_DISPID)
        {
            event_status.system[axis] &= ~SERVO_ENABLED;
            MiscParams[axis].ServoFlags &= ~PID_ENABLE_SERVO;
            PidError[axis] |= PROT2_IRQSTAT;
            Disable_Amp(axis);
        }
        else if (MiscParams[axis].IrqParams[IVECT_PROTECT] & HDWIRQ_ABORT)
        {
            AbortMove(axis);        /*  cause move to be aborted    */
            event_status.system[axis] |= GENERAL_FAULT;
        }
        DspException(DSPEXCEP_PROTECTFAULT,1);
    }
}

void EnableHWIRQ(int mask)
{
    /*
    ** use the interrupt enable shadow reg to enable an interrupt
    */
    IrqEnShad |= mask;
    IRQEN_REG = IrqEnShad;
}

void DisableHWIRQ(int mask)
{
    /*
    ** use the interrupt enable shadow reg to disable an interrupt
    */
    IrqEnShad &= ~mask;
    IRQEN_REG = IrqEnShad;
}

void UpdateHWIRQ()
{
    int i;
    AUX aux;
    int axis;

    for(axis = 0;axis<2;++axis)
    {
        PidError[axis] = 0;
        for(i=0;i<5;++i)
        {
            aux.auxi[0] = i | 0x8;
            aux.auxi[1] = MiscParams[axis].IrqParams[i];
            R_setirq(axis,axis,aux);    /*  set up interrupts   */
        }
    }
    PidError[2] = 0;        /*  klugey fix  */
}

void InitHWIRQ()
{
```

```
    /*
    ** initialize the hardware interrupts
    */
    int i;
    int axis;

    for(axis = 0;axis<2;++axis)
    {
        PidError[axis] = 0;
        for(i=0;i<5;++i)
        {
            MiscParams[axis].IrqParams[i] = 0;
        }
    }

    DoHWInit();          /*  call assembly language low level enable routine */
    EnableHWIRQ(HW_INTERRUPT_ENABLE);
    PidError[2] = 0;         /*  klugey fix  */
}
```

```
/*************************************************************************
**
** Title:profile.cpp
** Author:Jim Patchell
** Date:3-20-96
** Description:
**      This routine is used to calculate the required JERK, SCurve
**      Iterations, and Constant Velocity Interations to do a move
**      With firmware version 4.XX of the 600-100 Xpress Servo Card
**
**  Parameters:
**      S-Curve Time:type long.  Restrictions:must be multiple of 2
**      Maximum Velocity:type int Restrictions:none
**      Distance:type long Restrictions:none
**      Profile:Pointer to Profile Structure Restrictions:none
**  Returns:
**      type int.
**      0=No Error
**
**                  Copyright (c) 1996 Teletrac Inc.
**
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "spid.h"
#include "sprofile.h"


static double how_far(double jerk, double t)
{
    return(jerk * t * t * t/8.0l);
}

static double x_jerk(double time,double distance)
{
    /*********************************
    **
    ** this function calculates the required JERK
    ** to move a body from point a to point b
    ** "time" is the amount of time to reach maximum
    ** velocity.  distance is how far to move
    ** in the "linear" case, the amount of time
    ** to move distance is equal to two times
    ** the max velocity time
    *********************************/
    return( 4.0 * distance / (time * time * time) );
}

static double max_vel(double jerk, double time)
{
    /*************************************
    ** this function calculates the maximum
    ** velocity that will be attained given
    ** the jerk required to get a certain
    ** distance in two times time.
    *************************************/
    double vmax;

    vmax = jerk * time * time / 4.0;
    return(vmax );
}

static double v_jerk(double vmax, double time)
{
```

```
    /************************************
    ** given time and maximum velocity,
    ** calculate the JERK required to
    ** reach maximum velocity
    ************************************/

    return(4.0 * vmax / ( time * time) );
}

int TelProfile(int SCurveTime, long Distance, int MaxVel, TEL_PROFILE *Profile)
{
    int ReturnVal = TELPROFILE_OK;
    double jerk;    /* Temp value for calculating Jerk  */
    double peekvel; /* fastest servo will move          */
    double distance;    /* temporary place to store distance calculation    */
    int negFlag = 0;
    /*
    ** First check to see if SCurve is Multiple of 2
    */
    if(Distance < 0)
    {
        Distance = -Distance;
        negFlag = 1;
    }
    if(SCurveTime % 2l) ReturnVal = TELPROFILE_SCURVE;
    else
    {
        /*
        ** Calculate the Jerk and Steps Required to perform this profile
        */
        Profile->SSteps = SCurveTime / 2;   /*  number of SCurve Steps  */
        jerk = x_jerk((double)SCurveTime,(double)Distance);
        peekvel = max_vel(jerk, (double)SCurveTime);
        if((long)peekvel > MaxVel)  /* does this exceed maximum velocity?   */
        {
            /*Yes, TOO FAST, recalc to slew between S-Curves*/
            jerk = v_jerk((double)(MaxVel),(double)(SCurveTime));   /*get estimate of jerk  */
            distance = (double)Distance - 2.0l * how_far(jerk,(double)(SCurveTime)); /*estimate
of slew distance*/
            distance = distance/((double)MaxVel);   /*estimate of slew intterations*/
            distance = ceil(distance);      /*round up to ensure max-v not exceeded*/
            Profile->VSteps = (int)(distance);  /*set VSteps and Convert to long*/
            peekvel = (double)(Distance)/(double)(SCurveTime + Profile->VSteps);
            jerk = v_jerk(peekvel,(double)(SCurveTime));    /*calculate real jerk*/
        }
        else
        {
            Profile->VSteps = 0;    /*not needed for pure SCurve*/
        }
        Profile->Jerk = (long)(jerk * 16777216.0l); /*Set Jerk Value*/
        if(negFlag)
            Profile->Jerk = -Profile->Jerk;

    }   /*end of if(SCurveTime % 2)*/
    return ReturnVal;
}

/*************************************************************************
**
** These two functions are utility functions that are similar to TelProfile
**
*************************************************************************/

/*
double TelPeekVelocity(int SCurveTime, long Distance, int MaxVel)
{
```

```
*/
    /*
    ** This function is used to find out what the peek velocity is going
    ** to be for the given parameters
    */
/*  long VSteps;
    double distance;
    double peekvel;
    double jerk;

    if(SCurveTime % 2l) return 0.0;
    else
    {
*/
        /*
        ** Calculate the Jerk and Steps Required to perform this profile
        */
/*
        jerk = x_jerk((double)SCurveTime,(double)Distance);
        peekvel = max_vel(jerk, (double)SCurveTime);
        if((long)peekvel > MaxVel) */     /*does this exceed maximum velocity?*/
/*      {   */
            /*Yes, TOO FAST, recalc to slew between S-Curves*/
/*          jerk = v_jerk((double)(MaxVel),(double)(SCurveTime));   */      /*get estimate of jerk*/
/*          distance = (double)Distance - 2.0l * how_far(jerk,(double)(SCurveTime)); */      /*estima
te of slew distance*/
/*          distance = distance/((double)MaxVel);*/        /*estimate of slew intterations*/
/*          distance = ceil(distance);  */     /*round up to ensure max-v not exceeded*/
/*          VSteps = (long)(distance); */     /*set VSteps and Convert to long*/
/*          peekvel = (double)(Distance)/(double)(SCurveTime + VSteps);
        }
*/
/*  }   */  /*end of if(SCurveTime % 2)*/
/*  return peekvel;
}
*/


void MoveRel(int axis,long goal)
{
    TEL_PROFILE profile;
    int sr;
    /*
    ** Calculate out the nessesary parameters for the move
    */
    TelProfile(MiscParams[axis].SCurveTime,goal,MiscParams[axis].MaximumVel,&profile);
    sr = Disable();
    FiltParams[axis].Flags = FLAGS_START;
    FiltParams[axis].Jerk = profile.Jerk;
    FiltParams[axis].VCount = profile.VSteps;
    FiltParams[axis].SCount = profile.SSteps;
    Enable(sr);
    while(FiltParams[axis].Flags != FLAGS_END);
}

/*************************************************************************
**
** profile functions for spindle motor control
**
*************************************************************************/

int TelSpinProfile(int acceleration, long Velocity, int MaxVel, TEL_PROFILE *Profile)
{
    /*****************************************************************
    **
    ** acceleration: rpm/second.
```

```
    ** velocity: rpm
    ** MaxVel:rpm
    **
    ***********************************************************************/

    /*
    ** calculate the number of steps needed.  Must look at the sample rate
    ** to do this calculation.  Must know the encoder count as well.
    ** assume for the moment 4096 per rev.
    */
    double f;
    double samplerate;
    double a;

    if(Velocity < 0)
    {
        Velocity = -Velocity;
        f = -1.0;
    }
    else
        f = 1.0;
    samplerate = (double)((MiscParams[2].SampleRate+1) * (MiscParams[1].SampleRate + 1))*3.0;
    samplerate /= 156250.0;
    Profile->SSteps =(int)( ( (double)Velocity / (double)acceleration) / samplerate );
    if(Profile->SSteps == 0)
    {
        Profile->SSteps = 1;
    }
    a = ((double)Velocity /((double)Profile->SSteps * samplerate));
    a = a / 60 * (double)MiscParams[2].EncoderPitch;    /*  counts/sec-sec  */
    a *= samplerate * samplerate * f;           /*  counter per sample  */
    Profile->Jerk = (long)(a * 16777216.0);
}
```

```c
/*
** these are the routines for managing simple circular queues
*/

#include "squeues.h"
#include <stdio.h>
#include <stdlib.h>

/* #define TEST */

static QUEUE data_queue;

QUEUE *make_queue(int size) /*  constructor */
{
    QUEUE *q;

/*  if((q = (QUEUE *)calloc(1,sizeof(QUEUE) + size)) == NULL)
    {
        exit(1);
    }
    q->size = size; */
    q = &data_queue;
    q->size = 16;
    q->head = 0;
    q->tail = 0;
    q->num_elem = 0;
    return(q);
}


int put_queue(QUEUE *q,int val) /*  put data into queue */
{
    if(q->num_elem == q->size)      /*  is buffer full? */
    {
        return(BUFFER_FULL);
    }
    q->data[q->head++] = (char)val; /*  put data into buffer    */
    ++q->num_elem;
    if(q->head == q->size)
    {
        q->head = 0;                    /*  wrap around     */
    }
    return(OK);
}

int get_queue(QUEUE *q)         /*  get data from queue */
{
    int val;

    if(q->num_elem == 0)
    {
        return(BUFFER_EMPTY);
    }
    val = (int)q->data[q->tail++];
    if(q->tail == q->size)
        q->tail = 0;            /*  wrap around */
    --q->num_elem;
    return(val);
}

int status_queue(QUEUE *q)          /*  how many things in que  */
{
    return(q->num_elem);
}

/*
```

```
    Queues are not malloced at this time

void free_queue(QUEUE *q)
{
    free(q);
}

*/
```

```c
/*
** routines to setup and handle the Real time clock
*/

#include "srtc.h"
#include "spid.h"
#include "servhw.h"


void RTCInit(void)
{
    rtcI();
    MiscParams[0].Polarity = 0;
    MiscParams[0].SampleRate = 7;
    MiscParams[1].Polarity = 0;
    MiscParams[1].SampleRate = 7;
    MiscParams[2].Polarity = 0;
    MiscParams[2].SampleRate = 63;
    RTC_REG = 0x80 | (MiscParams[0].Polarity?0x200:0) | (MiscParams[1].Polarity?0x400:0) | (Misc
Params[2].Polarity?0x200000:0) | (MiscParams[1].SampleRate & 0x7f) | ((MiscParams[0].SampleRate
& 0xf) << 11) | ((MiscParams[2].SampleRate & 0x3f) << 15);
}
```

```
/*****************************************************************************
**
** This version of the SERVO CODE is for the Spindle Controller.
** It is different in that it handles two axis instead of one
** The history of previous incarnations that ran on the 600-060 and
** 600-100 boards are included.
**
*****************************************************************************
**
** This file contains the serial number and rev number
**
** 6-29-93 ver 1.03
**
**   Command.c and command.h, added trigger commands for starting move
**   profile.c added code to do trigger of move
**   hardware.c added code to set global trigger from hardware trigger
**   host.asm  added code to use HF2 to indicate servo moving
**
** 8-24-93 ver 1.10
**   Command.c and command.h added send and recieve command for jogging
**   parameters.  Removed the Jog Time command.
**
**   Event.c changed the way jogging was done
**
** 8-27-93 ver 1.11
**  Command.c  event_status.pending was not being incremented when an
**  event was being commanded
**
** 9-15-93 ver 1.12
**  Added code to set a bit in the host flag to indicate profile generator
**  is busy.
**
** 9-17-93 ver 1.13
** Added code in command.c that would check to see if the queue was full
** before stuffing characters into it
**
** 9-22-93 ver 1.14
** Corrected logic problem that allowed servo to be enabled before clearing
** Beam Interrupted Interrupt, or for that matter, any interrupt that caused
** the servo to be disabled.
**
** 9-27-93 ver 1.15
** Corrected problem with the fact that the detector PHASE was not saved in
** NV Rom.
** Fixed HOME so that it would also set HOST flag to indicate a real busy
** do not disturb situation.
**
** 9-29-93 ver 1.16
** Corrected posible stack problem in HOST.ASM in function GetI
** Fixed problem in routines in HOST.ASM that set and clear HCR reg.
** Interrupts are disabled while change is being made
**
** 10-1-93 ver 1.17
** While Jogging, pending events count was not decremented properly
** added event for END_JOG and added event_status.pending--; to DO_JOG
** event in Event.c
**
** 10-12-93 ver 1.18
** Command.c had bug in reading the value in the motor DAC.  did not mask off
** upper 8 bits so it was garbage.
** Added in position status bit in servo status.
**
** 11-18-93 ver 2.00 Alpha
**  Profile generator was converted to assembly language to increase speed.
**  Floating point ADD and Multiply were changed for speed so that the
**  routines would not check for floating point overflow or underflow, since,
```

```
** in doing so, They would still just spit out bogus values anyway.
** PID loop was changed so that profile generator would have to send one
** position sample per cycle.
** Code was added to detect error in profile generator/pid interface
**
** 11-23-93 ver 2.01 Alpha
**  Clear Following error code in Command.c sequence was changed to prevent
**  Servo PID loop from generating second following error exception
**
** 12-8-93 ver 2.02 Alpha
**  Fixed problem in EVENT.C, COMMAND.C and PID.ASM that caused problems
**  With the ENABLE FATAL FOLLOWING error command.  Cleaned up minor problems
**  in inter-task communication.
**
**  12-13-93 ver 2.02
**   Released
**
** 12-16-93 ver 2.03
**   Hardware.c  If interrupt mode was set to disable PID, it did not
**   Disable the Amplifier as well.
**   Home.c  Added to to save condition of Interrupts, then set up IRQ's
**   So that Homing or Calibration will not get accidently aborted.
** 12-21-93 ver 2.10
**   Command.c Fixed being able to HOME when servo is disabled
**   Pid.asm   Changed integrator so that it disables when Command velocity
**             is not zero (no integrate on moves).
**   Main.c    Added to to initialize event_status.system for new flag
**             indicating that that integrator will disable when Command
**             Velocity is not zero.
** 01-03-94 ver 2.11
**   Home.c     After calibrate, the final move disables the don't bother me
**                  flag, so flag is re-enabled after the move.
**
** 01-05-94 ver 3.00
**   Prifile.c  Problem with new profile generator.  Restored old profile
**              generator
**   Pid.asm    Changed PID so that it averages two points from profile
**              generator.
** 01-11-94 ver 3.01
**   Pid.asm    Typo in immediate move was #>$512 but should have been
**              #>512
** 02-19-94 ver 3.02
**  Event.c     Changed ServoFlags EN_FATAL_FOLLOWING_ERROR to fix problem
**  main.c
**  Home.c      Cleared Limit Functions before SEEK
** 02-24-94 ver 3.03
**  Event.c     Made JOG work a bit better for single step jogs
** 03-01-94 ver 3.04
**  Home.c      After Calibrate, Don't bother me flag cleared by move fixed
**
** 06-27-94     3.10
**  Command.c   Added commands to set and clear bits on digital port and
**              To read back bits on input port.
**
** 08-18-94     3.11
**  Command.c   Added code in move command to implement speed factor
**  Event.c     Added code in move commands to implement speed factors
**
** 09-16-94     3.12
**  Command.c   Added new command to read back shadow of output port
**
** 09-20-94     3.13
** Profile.c    Added conditional compile statements to allow for long
**              Slow moves.  Uses macro PROFILE_LONG_DISTANCE
**
** 11-28-94     3.14
```

```
** Home.c       Added in code to limit on DAC drive
**
** 02-11-95     3.15
** Event.c      Added in code for "BUMP" detection
**              Removed code for LEAD_SCREW
** command.c    Added code for BUMP detection
** pid.asm      Added memory location for Bump threshold
** nv.c nv.h    Increased size of Save and load for Bump Threshold
**
** 04-12-95     3.16
** command.c    Code Clean up, no real changes
** event.c      Code Clean up, no real changes
** queues.c     Code Clean up, no real changes
** initcode.asm FInitPID stack poped once too often
** rtc.c        RTCInit() Enabled RTC hardware IRQ AFTER! RTC initialized
** event.c      variables changed while interrupt was enabled, interrupt
**              enable fixed so this would not happen
**
** 08-18-95     3.17
** nv.c         Restoring parameters did not set real time clock register
**
** 12-20-95     3.20
** all files    revised for use on 600-100 servo card
**
**  3-18-96     4.00
**              Extensive Revision.
**              Profile.C: eliminated!
**              RTC.ASM:
**                  modified to load address registers to point to
**              PID parameters before PID called.  This is so we can have
**              multi axis controller card.
**              PID.ASM:
**                  complete redesigned.  Profile generator moved to
**              inside of PID loop to increase speed.  Profile Generator
**              will operate autonomously.  It will no longer chew up all
**              of the CPU bandwidth.  Don't bother me flag will no longer
**              be set while doing a move.
**              HOME.C:
**                  extensively modified.  Changes made to take advantage
**              of new profile generator for doing homes.  Modifications
**              made so that multi axis version of card can home all axis
**              at the same time.
**              COMMAND.C:
**                  Eliminated MoveIndexed Commands
**                  Eliminated Indexed Setup Commands
**              SN.C:
**                  removed init_version and serial_number
**  07-17-96    4.01
**              Host.asm  initialization problem fixed for recieve
**              record
**
** 12-05-96     4.02
**              S_COMMND.C Function R_startjog eroneously set the desired
**              goal.
** 12-23-96     4.03
**              Further Revisions of pid.asm based on IBMYAG code
**              Further Revisions of event.c based on IBMYAG code
**              Further Revisions of Home.c based on IBMYAG code
**
**
** 08-18-97     1.00
**              Son Of Accutrac
**              Extensive revisions to almost all code.
**              Added Messages back to host processor for Exceptions
**              Removed code for non-volatile memory.
**              Code for non-volatile memory still in place for now, but
```

```
**              not functional.
**              Code for DUAL AXIS operation.
**
** 6-15-98      1.01
**              Modified PID.asm to properly scale following error gain
**
** 7-08-98      1.02
**              Modifide all files to convert to 3 axis controller.
**              Axis 0= Linear
**              Axis 1= Rotary
**              Axis 2= Spindle
**
** 10-13-98     1.03
**              Added control code for spindle motor
**              Added Encoder pitch parameters
**
** 01-15-99     1.04
**              host.asm:increased buffer size from 256 to 512
**
** 01-22-99     1.05
**              s_command.c
**              in retstore parameters, added code to fix problem with
**              disable integrator on velocity non zero
**
** 03-15-99     1.06
**              s_event.c
**              s_event.h
**              added MOVE_IDLE to system events and added to set
**              MOVE_IDLE status bit when profile generators are not
**              active.
**
** 03-26-99     1.07
**              pid.asm
**              modified pid so that difference between goal and desired
**              position is really output.  Dither is not figured into
**              this.
**
** 03-29-99     1.08
**              pid.asm
**              Found and repaired bug in spectrum analyzer part of
**              PID code.
**
** 04-08-99     1.09
**              event.c
**              Protected event_status.pending from command interrupts
**              Added code to get axis when checking for end jog in
**              jogging function
**
** 04-12-99     1.10
**              s_hrdwar.c
**              In init function, removed code to initalize non-existant
**              axis 3 interrupts.
**              put kluge in to clear PidError[2] flag.
**
** 07-19-99     1.11
**              s_command.c
**                  added code for histogram commands
**              pid.asm
**                  added histogram code
**              s_event.c
**                  added code to send message when histogram is done
*/

static int dummy;

char version[] = "SOA 1.11 0719 99";
```

```
    ; $Id: crt056.asm,v 1.12 91/06/14 14:30:36 jeff Exp $
    ; this is the startup code for bus boot
    opt so,xr
    page 132,66,3,3

TOP_OF_MEMORY   equ $7fff

    section no_ram
    org Y:$100
    ds  $100        ;ram starts at address 200
    endsec
    section stack
    org Y:$7000
    global Stack
Stack   ds $1000
    endsec

    ;   This section should be loaded at P:0 that way reset will
    ;   cause a jump to main.

;   The reset section is located at p:0 in the mapfile. This section is
;   intended to hold system reset code. This code should jump to start
;   when it is done.

    section reset

    org p:$0
    jmp F__start

    org p:$2
    jsr Fabort      ;1
    jsr Fabort      ;2
    jsr Fabort      ;3
    jsr HWIrq       ;4   Hardware interrupt IRQA
    jsr RTCirq      ;5   Real time interrupt IRQB
    jsr Fabort      ;6
    jsr Fabort      ;7
    jsr Fabort      ;8
    jsr Fabort      ;9
    jsr Fabort      ;10
    jsr Fabort      ;11
    jsr Fabort      ;12
    jsr Fabort      ;13
    jsr Fabort      ;14
    jsr Fabort      ;15
    jsr GetI        ;16  Host in interrupt
    jsr PutI        ;17  Host out interrupt
    jsr HICmdIrq    ;18  Host command
    jsr Fabort      ;19
    jsr Fabort      ;20
    jsr Fabort      ;21
    jsr Fabort      ;22
    jsr Fabort      ;23
    jsr Fabort      ;24
    jsr Fabort      ;25
    jsr Fabort      ;26
    jsr Fabort      ;27
    jsr Fabort      ;28
    jsr Fabort      ;29
    jsr Fabort      ;30
    jsr Fabort      ;31

    endsec

    section crt0
```

```
;   org     y:

; The following varaibles are used for dynamic memory allocation
; __stack_safty: Since dynamic memory and the stack grow towards each other
; This constant tells brk and sbrk what the minimum amount of space should
; be left between the top of stack during the brk or sbrk call and the end
; of any allocated memory.
; __mem_limit: a constant telling brk and sbrk where the end of available
; memory is.
; __break: pointer to the next block of memory that can be allocated
; The heap may be moved by changing the initial value of __break.
; This is the base of the heap.
; __y_size: the base of dynamic memory.
; errno: error type: set by some libraries
; __max_signal the last possible signal.

;   global  Ferrno
;Ferrno     dc  $0

;   global F__stack_safety
;F__stack_safety    dc  $10

;   global F__mem_limit
;F__mem_limit   dc  TOP_OF_MEMORY

;   global F__break
;F__break   dc  TOP_OF_MEMORY

;   global F__y_size
;F__y_size  dc  DSIZE

;   global  F__max_signal
;F__max_signal  dc  $3e ; what the quint monitor.


    org p:

F__start
    global  F__start
        and #$f3,mr
        and #$bf,ccr
;   Must finish boot process, this is done by loading the rest of the
;   program code.
;   Host indicates when end of program code has occured, then
;   the data load is done, loading initialized values into memory
;
;************************************************************************
;
;   finish boot of program code.  This starts loading at $200
;
;************************************************************************
;
_load_pmem
        bset    #0,X:$ffe0      ;port b as host port
        move    #$ffe9,r2       ;r2 is the host address
        move    #$200,r0        ;start loading at $200
_loop1  bset    #0,X:$ffe0      ;port b as host port
_l001   jclr    #3,X:$ffe9,_l002 ;stop loading
        jmp     <_load_ymem
_l002   jclr    #0,X:(r2),_l001 ;wait for hrdf to go high
        move    X:$ffeb,a1      ;put 24 bit host data in a1
        move    a1,P:(r0)+      ;store into P memory
        jmp     _loop1
;
;************************************************************************
;
```

```
; load data into y data memory
;
;************************************************************************
;
;
_load_ymem
        jset    #3,X:$ffe9,_load_ymem   ;wait for flag to be reset
        bset    #4,X:$ffe8      ;inicate waiting for data
        move    #$ffe9,r2       ;r2 is the host address
        move    #0,r0           ;start loading at 0
_loop3  bset    #0,X:$ffe0      ;port b as host port
_l003   jclr    #3,X:$ffe9,_l004    ;stop loading
        jmp     bootend
_l004   jclr    #0,X:(r2),_l003 ;wait for hrdf to go high
        move    X:$ffeb,a1      ;put 24 bit host data in a1
        move    a1,Y:(r0)+      ;store into P memory
        jmp     _loop3
;
;   To change the base of the stack, change the value loaded into the
;   stack pointer here.
;
bootend
    and #$f3,mr
    and #$bf,ccr
    move    #Stack,r6   ; initialize the stack pointer
    move    #$0,r0      ; funny value to terminate a backtrace.
    movep   #$0004,x:$fffe  ;zero external wait states
                            ;4 wait states for Xilinx
    bclr    #4,x:$ffe8
    jsr Fmain       ; run user program

F__crt0_end
    global  F__crt0_end
    bset    #4,X:$ffe8      ;indicate STOP
    bset    #3,X:$ffe8      ;indicate STOP
    stop            ; all done

    endsec

    section crt0__time

    org y:0

    global  F__time
F__time dc  $0  ; used to time execution with simulator

    org p:

Fclock
    global  Fclock

    move    y:F__time,a
    tst a
    rts

    endsec

;   section crt0__printf_end

;   org p:$3ff      ; we must reserve this location so that
                ; the instruction fetch mechanism will not
                ; automatically trigger the printf.
;   nop

;   org p:$400      ; this location is recognized by the
                ; exec program.
```

```
;   global  F__printf_end
;F__printf_end          ; dummy printf catch routine.
;   rts

;   endsec

    section crt0__fp_shift

    org y:

;
; floating point table setup
;

    global  F__fp_shift
F__fp_shift dc      $800000,$c00000,$e00000,$f00000,$f80000,$fc0000
        dc      $fe0000,$ff0000,$ff8000,$ffc000,$ffe000,$fff000
        dc      $fff800,$fffc00,$fffe00,$ffff00,$ffff80,$ffffc0
        dc      $ffffe0,$fffff0,$fffff8,$fffffc,$fffffe

    endsec
```

```
    section junk_c

    org p:
    global  FFloatTo72
FFloatTo72
    move    r0,y:(r6)+      ;save frame pointer
    lua (r6)+,r0            ;update frame pointer
    move    r1,y:(r6)+      ;save r1 onto stack
    move    b2,y:(r6)+      ;save b accumulator
    move    b1,y:(r6)+
    move    b0,y:(r6)+
    move    x1,y:(r6)+      ;save x1 register
; ************************************************
; void FloatTo72(double f, int *w72)
; {
;   *((long *)w72) = (long)f;
; ************************************************
    move    #65532,n0
    move    (r0)+
    move    (r0)+n0
    move    y:(r0)-,a       ;move mantisa to a1
    move    y:(r0)-n0,b     ;move exponent to b1


;************************************************
;
;   Code to convert double into 72 bit 24.48 number
;
;************************************************
    move    #>$2000,x1
    asl     a
    asl     a
    sub     x1,b            ;remove offset from exponent
    jeq     _fto72exit
    jgt     _fto72_0
                            ;exponet was negative, shift right
    neg     b               ;make exponent positive
    rep     b
    asr     a               ;shift accumulator "b" times
    jmp     _fto72exit

_fto72_0                    ;exponent was positive, shift left
    rep     b
    asl     a

_fto72exit

;************************************************
;
; Save converted number into value pointed to by w72
;
;************************************************
    move    #65531,n0       ;get pointer to w72
    move    y:(r0+n0),r1
    move    a2,y:(r1)+      ;save data to w72
    move    a1,y:(r1)+
    move    a0,y:(r1)

; ************************************************
; }
; ************************************************
    move    (r6)-
    move    y:(r6)-,x1      ;restore x1 register
    move    y:(r6)-,b0      ;restore b accumulator
    move    y:(r6)-,b1
    move    y:(r6)-,b2
```

```
    move    y:(r6)-,r1      ;restore r1
    move    y:(r6),r0       ;restore frame pointer
    rts


    endsec

;
```

```
;;!cc
    PAGE    132,66,4,4,5
;*********************************************************************
;                                                                    **
; host interface routines.  Gets/Puts data to/from the host port     **
;                                                                    **
;   HInit:  initialize the host port                                 **
;   HIGet:  get a single word from the host port                     **
;   HIPut:  put a single word to the host port                       **
;   HIiStatus:  status of input port                                 **
;   HIoStatus:  status of output port                                **
;   HIcmdStatus: status of command register                          **
;                                                                    **
;*********************************************************************
;
HEAD    = 0
TAIL    = 1
MOD     = 2
NCHARS  = 3
IPR     = $ffff     ;interrupt priority register
HCR     = $ffe8     ;control register
HSR     = $ffe9     ;status register
HRX     = $ffeb     ;recieve register
HTX     = $ffeb     ;transmit register
HCP     = 2         ;bit number for host command pending

    section Rec_data
    org x:$400
    global rbuf
    global xbuf
    global rec_rec
    global xmit_rec

rbuf    ds  512     ;recieve buffer
xbuf    ds  512     ;transmit buffer
rec_rec ds  4       ;space for recieve descriptor
                        ;offset 0 head pointer
                        ;offset 1 tail pointer
                        ;offset 2 modulus
                        ;offset 3 number of characters

xmit_rec    ds  4   ;space for transmit descriptor
                        ;offset 0 head pointer
                        ;offset 1 tail pointer
                        ;offset 2 modulus
                        ;offset 3 number of characters
    endsec

    section hostsec

    org p:
    global  FHInit
FHInit
    move    r0,y:(r6)+
    lua     (r6)+,r0        ;lua does not update r6

;
;the real stuff
;
    move    n1,y:(r6)+      ;save index register
    move    m1,y:(r6)+
    move    r1,y:(r6)+
    move    x0,y:(r6)+
    bclr    #4,X:HCR
    bclr    #3,x:HCR
;
```

```
;setup the buffer descriptors for transmit and
;recieve routines
;
    move    #rec_rec,r1     ;load address of recieve descriptor
    move    #rbuf,a1
    move    a1,X:(r1)+      ;head pointer
    move    a1,X:(r1)+      ;tail pointer
    move    #511,a1
    move    a1,X:(r1)+      ;modulus, fixed was move a,X:(r1)+
    clr     a
    move    a1,X:(r1)       ;number of characters

    move    #xmit_rec,r1    ;load address of transmit descriptor
    move    #xbuf,a1
    move    a1,X:(r1)+      ;head pointer
    move    a1,X:(r1)+      ;tail pointer
    move    #511,a1
    move    a1,X:(r1)+      ;modulus
    clr     a
    move    a1,X:(r1)       ;number of characters

;
; set up interrupt priorities for the host interface
;
    move    X:IPR,a         ;get interrupt priority register
    move    #>$f3ff,x0
    and     x0,a
    move    #>$0400,x0      ;priority level 0 for host interface
    or      x0,a
    move    a,X:IPR         ;save updated IPR reg

    andi    #$fc,mr         ;enable interrupts
    bset    #0,X:HCR        ;enable recieve interrupt
    move    (r6)-
    move    y:(r6)-,x0
    move    y:(r6)-,r1      ;restore index register
    move    y:(r6)-,m1
    move    y:(r6)-,n1
;
;exit code
;

    move    Y:(r6),r0
    tst     a
    rts

    global FHIGet
FHIGet
;
;get a character from the host interface
;if no characters are availiable, wait until one is
;
    move    r0,y:(r6)+
    lua     (r6)+,r0

    move    r1,y:(r6)+

    move    m3,y:(r6)+      ;save another index register
    move    r3,y:(r6)+
    move    b0,y:(r6)+      ;save b register
    move    b1,y:(r6)+
    move    x0,y:(r6)+

    move    #rec_rec+NCHARS,r1  ;address of recieve record
    clr     a               ;clear the accumulator
_get_wait
```

```
    move    X:(r1),a
    tst     a               ;are there any characters?
    jne     _Get01          ;yes, go get them
    jset    #0,X:HCR,_get_wait  ;if rec interrupt enabled, keep waiting
    bset    #0,X:HCR        ;enable recieve interrupt
    jmp     _get_wait       ;keep looping
;
;if there is a character, set up a circular buffer
;
_Get01
    move    sr,y:(r6)+      ;save status register
    ori     #3,mr           ;disable
    move    #rec_rec+MOD,r1
    move    X:(r1),m3       ;load modulus into mod reg
    move    #rec_rec+TAIL,r1            ;offset for tail pointer
    move    X:(r1),r3       ;load address of buffer
    move    X:(r3)+,a       ;get data
    nop
    move    r3,X:(r1)       ;save tail pointer

    move    #rec_rec+NCHARS,r1
    move    X:(r1),b
    move    #>-1,x0
    add     x0,b            ;decrement number of characters
    move    b,X:(r1)
    move    (r6)-
    move    y:(r6)-,sr      ;reload sr and enable interrupts
;
;restore saved registers
;
    move    y:(r6)-,x0
    move    y:(r6)-,b1
    move    y:(r6)-,b0
    move    y:(r6)-,r3
    move    y:(r6)-,m3
    move    y:(r6)-,r1      ;restore index register
;
;exit code
;
    move    Y:(r6),r0
    tst     a
    rts

    global FHIPut
FHIPut
;
;put a character into the host port
;
    move    r0,y:(r6)+
    lua     (r6)+,r0
    move    r1,y:(r6)+
    move    r3,y:(r6)+      ;save another index register
    move    n0,y:(r6)+      ;index from frame pointer
    move    b0,y:(r6)+
    move    b1,y:(r6)+
    move    b2,y:(r6)+
;
;check to see if there is room in the transmit registers first
;
;   jclr    #1,X:HSR,_check_buff    ;if no space in reg, check buffer
;   move    #-3,n0                  ;offset into stack frame
;   move    y:(r0+n0),a
;   move    a,x:HTX                 ;load transmit reg from stack frame
;   jmp     _p_exit
;
;find out if there is any room in the buffer
```

```
;
_check_buff
    move    #xmit_rec+NCHARS,r1         ;offset to number of characters
_wait_xbuff
    move    X:(r1),a
    move    #>512,b
    cmp     b,a
    jeq     _wait_xbuff
;
;disable interrupts and the add character to buffer
;
    move    sr,y:(r6)+          ;save status register
    move    m3,y:(r6)+          ;push modulus reg here
    ori     #3,mr               ;disable interrupts
    jset    #1,X:HCR,_no_set
    bset    #1,X:HCR            ;set transmit interrupt bit
_no_set
    move    #xmit_rec+MOD,r1
    move    X:(r1),m3           ;modulus of circular buffer
    move    #xmit_rec+HEAD,r1   ;head pointer offset
    move    X:(r1),r3           ;head pointer into index register
    move    #-3,n0              ;offset into stack frame of data
    move    y:(r0+n0),a1        ;get data
    move    a1,X:(r3)+          ;put data into buffer
    move    r3,X:(r1)           ;save head pointer
    move    #xmit_rec+NCHARS,r1 ;load number of characters
    move    x:(r1),a
    move    #>1,b
    add     b,a                 ;increment accumulator
    move    a,x:(r1)            ;save number of characters
    move    (r6)-
    move    y:(r6)-,m3          ;so it can be restored her
                                ;before interrupts are enabled
    move    y:(r6),sr           ;reload sr and enable interrupts
;
;restore saved registers
;
_p_exit
    move    (r6)-
    move    y:(r6)-,b2
    move    y:(r6)-,b1
    move    y:(r6)-,b0
    move    y:(r6)-,n0
    move    y:(r6)-,r3
    move    y:(r6)-,r1      ;restore index register
;
;exit code
;
    move    Y:(r6),r0
    tst     a
    rts
;
;code for the host recieve interrupt
;
    global  GetI
GetI
    move    (r6)+
    move    ssl,y:(r6)+
    move    ssh,y:(r6)+
    move    r1,y:(r6)+
    move    m3,y:(r6)+
    move    r3,y:(r6)+
    move    a0,y:(r6)+
    move    a1,y:(r6)+
    move    a2,y:(r6)+
    move    x0,y:(r6)+
```

```
;
;check to see if there is any room in the recieve buffer
;
    move    #rec_rec+NCHARS,r1  ;load number of characters in buffer
    move    X:(r1),a
    move    #>512,x0                ;compare buffer lenth to max
    cmp     x0,a
    jlt     _GI001              ;branch if plenty of room
;
;if there is not enough room, disable the recieve interrupt to make
;it stop annoying the main processes
;
    bclr    #0,x:HCR        ;clear recieve interrupt
    jmp     GIexit
;
;get data from recieve register, and put it into the input buffer
;
_GI001
    move    #rec_rec+MOD,r1 ;load buffer modulus
    move    X:(r1),m3
    move    #rec_rec+HEAD,r1 ;load head pointer
    move    X:(r1),r3
    move    x:HRX,a     ;get character from host port
    move    a,x:(r3)+   ;place it into buffer
    move    r3,x:(r1)   ;save head pointer

    move    #rec_rec+NCHARS,r1
    move    x:(r1),a
    move    #>1,x0
    add     x0,a            ;increment number of characters
    move    a,x:(r1)        ;save number of characters

;
;restore the registers
;
GIexit
    move    (r6)-
    move    y:(r6)-,x0
    move    y:(r6)-,a2
    move    y:(r6)-,a1
    move    y:(r6)-,a0
    move    y:(r6)-,r3
    move    y:(r6)-,m3
    move    y:(r6)-,r1
    move    y:(r6)-,ssh
    move    y:(r6)-,ssl
;   move    (r6)-
    rti

    global PutI
PutI
;
;save registers on stack
;
    move    (r6)+
    move    ssl,y:(r6)+
    move    ssh,y:(r6)+
    move    m3,y:(r6)+
    move    n3,y:(r6)+
    move    a0,y:(r6)+
    move    a1,y:(r6)+
    move    a2,y:(r6)+
    move    r3,y:(r6)+
;
;get character from buffer and place in transmit register
;
```

```
    move    x:xmit_rec+MOD,m3   ;offset of modulus
    move    x:xmit_rec+TAIL,r3  ;offset to tail pointer
    move    x:(r3)+,a           ;load character from buffer into a
    move    a,X:HTX             ;put it into the transmit register
    move    r3,x:xmit_rec+TAIL      ;put tail pointer back into descriptor
    move    x:xmit_rec+NCHARS,r3    ;offset of number of characters
    move    #$ffff,m3
    move    (r3)-           ;number of characters into index reg
    move    r3,x:xmit_rec+NCHARS
    move    r3,a
    tst     a
    jne     _IPexit         ;if count is non-zero, exit
    bclr    #1,X:HCR        ;clear interrupt enable flag for transmit port
;
;restore the registers
;
_IPexit
    move    (r6)-
    move    y:(r6)-,r3
    move    y:(r6)-,a2
    move    y:(r6)-,a1
    move    y:(r6)-,a0
    move    y:(r6)-,n3
    move    y:(r6)-,m3
    move    y:(r6)-,ssh
    move    y:(r6)-,ssl
;   move    (r6)-
    rti

    global  FHIiStatus
FHIiStatus
;
;Get status of Host input port
;
    move    r0,y:(r6)+
    lua     (r6)+,r0
    move    r1,y:(r6)+

    move    #rec_rec+NCHARS,r1  ;offset for number of characters
    move    X:(r1),a            ;move number of characters to accumulator
    tst     a
    jne     _iS_xit
    bset    #0,x:HCR            ;if buffer is empty, enable rec interrupt
;
;exit code
;
_iS_xit
    move    (r6)-
    move    y:(r6)-,r1      ;restore index register
    move    Y:(r6),r0
    tst     a
    rts

    global  FHIoStatus
FHIoStatus
    move    x:xmit_rec+NCHARS,a ;offset for number of characters
;
;exit code
;
    tst     a
    rts

;
;get status of HOST command
;
    global FHIcmdStatus
```

```
FHIcmdStatus
    jclr    #HCP,x:HSR,_nocmd   ;check to see if command bit flipped
    move    #>1,a               ;set TRUE
    jmp     <_cmd_xit
_nocmd
    clr     a                   ;set to false
_cmd_xit
    tst     a
    rts

    global FHIFSet
    global FHIFClr
;
;---------------------------------------------------------------
;
; This routine will set/clear bits in the host control register
;
;---------------------------------------------------------------
;
FHIFSet
    move    r0,y:(r6)+      ;set up stack frame
    lua     (r6)+,r0
    move    n0,y:(r6)+
    move    x0,y:(r6)+      ;save registers that are to be used
    move    x1,y:(r6)+
    move    sr,y:(r6)+      ;push status reg on stack

    ori     #3,mr           ;disable interrupts
    move    #-3,n0
    move    y:(r0+n0),a1    ;get bit mask
    move    #>$18,x0        ;strip off invalid bits
    and     x0,a1
    move    a1,x1
    move    x:HCR,a1        ;get host control register
    move    #>$e7,x0
    and     x0,a1
    or      x1,a1           ;set bits
    move    a1,x:HCR        ;set bits in register

    move    (r6)-
    move    y:(r6)-,sr      ;pop status reg
    move    y:(r6)-,x1
    move    y:(r6)-,x0
    move    y:(r6)-,n0
    move    y:(r6),r0
    tst     a
    rts

FHIFClr
    move    r0,y:(r6)+      ;set up stack frame
    lua     (r6)+,r0
    move    n0,y:(r6)+
    move    x0,y:(r6)+      ;save registers that are to be used
    move    sr,y:(r6)+      ;push status reg on stack

    ori     #3,mr           ;disable interrupts
    move    #-3,n0
    move    y:(r0+n0),a1    ;get bit mask
    move    #>$18,x0        ;strip off invalid bits
    and     x0,a1
    not     a
    move    a1,x0
    move    x:HCR,a1        ;get host control register
    and     x0,a1
    move    a1,x:HCR        ;set bits in register
```

```
    move    (r6)-
    move    y:(r6)-,sr      ;pop status reg
    move    y:(r6)-,x0
    move    y:(r6)-,n0
    move    y:(r6),r0
    tst     a
    rts

    global FEnable
    global FDisable

FEnable
    move    r0,y:(r6)+      ;set up stack frame
    lua     (r6)+,r0
    move    n0,y:(r6)+
    move    x0,y:(r6)+      ;save registers that are to be used
    move    x1,y:(r6)+

    move    #-3,n0
    move    y:(r0+n0),a         ;get interrupt mask
    move    #>$0300,x0
    and     x0,a                ;STRIP off garbage
    move    a,x0                ;save mask

    move    sr,a                ;get status register
    move    #$fcff,x1           ;load another mask
    and     x1,a                ;zap interrupt mask bits
    or      x0,a                ;restore interrupt mask bit
    move    a1,sr               ;save interrupt mask bit

    move    (r6)-               ;restore stack
    move    y:(r6)-,x1
    move    y:(r6)-,x0
    move    y:(r6)-,n0
    move    y:(r6),r0
    tst     a
    rts

FDisable
    move    sr,a                ;get old interrupt mask
    ori     #$03,mr             ;disable interrupts
    tst     a
    rts

    global FHIcmdEnable         ;enable command interrupt

FHIcmdEnable
    bset    #HCP,x:HCR          ;enable command interrupt bit
    rts

    global FHIcmdDisAble        ;disable command interrupt
FHIcmdDisAble
    bclr    #HCP,x:HCR          ;clear interrupt enable bit
    rts

    global HICmdIrq
    ;
    ;this is the interrupt entry point for the command interrupt
    ;processing, we push every single register
    ;
HICmdIrq
    move    (r6)+       ;dummy push
    move    ssl,y:(r6)+
    move    ssh,y:(r6)+
    move    r0,y:(r6)+
```

```
    move    n0,y:(r6)+
    move    r1,y:(r6)+
    move    n1,y:(r6)+
    move    r2,y:(r6)+
    move    n2,y:(r6)+
    move    r3,y:(r6)+
    move    m3,y:(r6)+
    move    n3,y:(r6)+
    move    r4,y:(r6)+
    move    n4,y:(r6)+
    move    r5,y:(r6)+
    move    n5,y:(r6)+
    move    r7,y:(r6)+
    move    n7,y:(r6)+
    move    a0,y:(r6)+
    move    a1,y:(r6)+
    move    a2,y:(r6)+
    move    b0,y:(r6)+
    move    b1,y:(r6)+
    move    b2,y:(r6)+
    move    x0,y:(r6)+
    move    x1,y:(r6)+
    move    y0,y:(r6)+
    move    y1,y:(r6)+
    move    n6,y:(r6)+  ;all registers that need to be saved are saved

    move    #$ffff,m3   ;this is the register we use for MODULUS operation
                        ;in both the RS232 and HI interface buffers

    ;
    ; call function to do command processing
    ;
    jsr     FCommand
    ;
    ;
    move    (r6)-
    move    y:(r6)-,n6
    move    y:(r6)-,y1
    move    y:(r6)-,y0
    move    y:(r6)-,x1
    move    y:(r6)-,x0
    move    y:(r6)-,b2
    move    y:(r6)-,b1
    move    y:(r6)-,b0
    move    y:(r6)-,a2
    move    y:(r6)-,a1
    move    y:(r6)-,a0
    move    y:(r6)-,n7
    move    y:(r6)-,r7
    move    y:(r6)-,n5
    move    y:(r6)-,r5
    move    y:(r6)-,n4
    move    y:(r6)-,r4
    move    y:(r6)-,n3
    move    y:(r6)-,m3
    move    y:(r6)-,r3
    move    y:(r6)-,n2
    move    y:(r6)-,r2
    move    y:(r6)-,n1
    move    y:(r6)-,r1
    move    y:(r6)-,n0
    move    y:(r6)-,r0      ;all registers that need to be restored, restored
    move    y:(r6)-,ssh
    move    y:(r6)-,ssl
    rti
```

```
    endsec
```

```
;*****************************************************************
;
; put the INIT code for the critical sections here, since they do not
; need to run fast
;
;*****************************************************************
;

    section initcode
    org p:
IPR =   $ffff       ;interrupt priorty register
RTC =   $ffc2       ;interrupt control register

POSREG  = $ffc8

RD_IRQ  = $ffc0
RD_RTC  = $ffc2
CLEAR_IRQ = $ffc1

ZI_V    =   5       ;storage for integrator temp , int



    global FZeroIntegrator
FZeroIntegrator
    ;
    ;zero the value accumulated by the integrator
    ;
    clr     a
    move    a,x:Ffiltvals+ZI_V  ;FIX ME!!!! BOGUS
    rts


;
;Initialize the Real Time Clock Interrupt
;
    global FrtcI

FrtcI
    move    x0,y:(r6)+
    move    r0,y:(r6)+
    move    x1,y:(r6)+
    move    x:IPR,a
    move    #>$18,x0        ;set the priority level to 2
    or      x0,a
    move    a,x:IPR

    move    (r6)-
    move    y:(r6)-,x1
    move    y:(r6)-,r0
    move    y:(r6),x0
    rts

    global FrtcDisable

FrtcDisable
    move    x0,y:(r6)+
    move    r0,y:(r6)+
    move    x1,y:(r6)+
    move    x:IPR,a
    move    #>$ffffc7,x0        ;set the priority level to 2
    and     x0,a
    move    a,x:IPR

    move    (r6)-
    move    y:(r6)-,x1
```

```
    move    y:(r6)-,r0
    move    y:(r6),x0
    rts

    endsec
```

```
;;!cc
;*****************************************************************
;
; This file contains the interrupt routine that handles any hardware
; exceptions that are generated
;
;*****************************************************************
;
STATUSREG = $ffc0       ;hardware interrupt status reg
IPR = $ffff             ;interrupt priority reg
    section HardIRQ
    org p:

    global HWIrq        ;hardware interrupt service

HWIrq
    move    (r6)+       ;dummy push
    move    ssl,y:(r6)+
    move    ssh,y:(r6)+
    move    r0,y:(r6)+  ;push important registers
    move    n0,y:(r6)+
    move    r1,y:(r6)+
    move    n1,y:(r6)+
    move    r2,y:(r6)+
    move    n2,y:(r6)+
    move    r3,y:(r6)+
    move    m3,y:(r6)+
    move    n3,y:(r6)+
    move    r4,y:(r6)+
    move    n4,y:(r6)+
    move    r5,y:(r6)+
    move    n5,y:(r6)+
    move    r7,y:(r6)+
    move    n7,y:(r6)+
    move    a0,y:(r6)+
    move    a1,y:(r6)+
    move    a2,y:(r6)+
    move    b0,y:(r6)+
    move    b1,y:(r6)+
    move    b2,y:(r6)+
    move    x0,y:(r6)+
    move    x1,y:(r6)+
    move    y0,y:(r6)+
    move    y1,y:(r6)+
    move    n6,y:(r6)+  ;all registers that need to be saved are saved

    move    #$ffff,m3   ;this is the register we use for MODULUS operation
                        ;in both the RS232 and HI interface buffers

    ;
    ; call function to do command processing
    ;
    move    y:STATUSREG,a   ;push status reg on stack
    move    a,y:(R6)+
    jsr     FHandleHardware ;call service routine
    move    (r6)-           ;clean up stack
    ;
    ;
    move    (r6)-
    move    y:(r6)-,n6
    move    y:(r6)-,y1
    move    y:(r6)-,y0
    move    y:(r6)-,x1
    move    y:(r6)-,x0
    move    y:(r6)-,b2
    move    y:(r6)-,b1
```

```
    move    y:(r6)-,b0
    move    y:(r6)-,a2
    move    y:(r6)-,a1
    move    y:(r6)-,a0
    move    y:(r6)-,n7
    move    y:(r6)-,r7
    move    y:(r6)-,n5
    move    y:(r6)-,r5
    move    y:(r6)-,n4
    move    y:(r6)-,r4
    move    y:(r6)-,n3
    move    y:(r6)-,m3
    move    y:(r6)-,r3
    move    y:(r6)-,n2
    move    y:(r6)-,r2
    move    y:(r6)-,n1
    move    y:(r6)-,r1
    move    y:(r6)-,n0
    move    y:(r6)-,r0      ;all registers that need to be restored, restored
    move    y:(r6)-,ssh
    move    y:(r6)-,ssl
;   move    (r6)-       ;dummy pop
    rti

    global FDoHWInit        ;initialize this interrupt
FDoHWInit
    ;
    ;This interrupt is going to level 0
    ;
    move    x0,y:(r6)+      ;save X0 reg
    move    x:IPR,a         ;get interrupt priortity register
    move    #>$1,x0         ;priority level 0, level trigger mode
    or      x0,a            ;set bits
    move    a,x:IPR         ;save interrupt priority register
    move    (r6)-
    move    y:(r6),x0       ;restore X0 register
    rts

    endsec
```

```
;
;*****************************************************************
;
; code for the PID loop algorythm
;
; This is the NEW code that uses a built in profile generator for
; the PID to speed up the code by a terrific amount.
; This is the PID generator that will be used in version 4.0 of the
; Servo Controller code.
;
; This code was changed 3-18-96, and, many times since
;
;*****************************************************************
;
;----------------------------------------------------------------------
; Misc Hardware Addresses
;----------------------------------------------------------------------
;
CLEARIRQ = $ffc1
FREQUENCY_REG = $ffce
RDSIN_REG = $ffcd       ;this is the output of the dig oscilator 0->255
RDCOS_REG = $ffcc       ;this is the output of the phase shifted sin
POS_H_REG = $ffc5   ;high address of the position registers
LINEAR  =   1       ;hardware descriptors for position registers
ROTARY  =   0
SPIN    =   2
POS_H_DESC  = $ffc6 ;address of hardware descriptor register
RDPES_REG   =   $ffcb   ;address of PES offset register
PCD     = $ffe5         ;port D address

;
;----------------------------------------------------------------------
;
; bit definitions for SERVOFLAGS
;
;----------------------------------------------------------------------
;

SERVFLAG = 0        ;turns on the servo loop
INTEGFLAG = 1       ;keep accumulating, but don't use
INTOFF = 2          ;stop integrator from accumulating
EN_FATAL_FOLLOWING = 3  ;this bit enables death on fatal following error
FATAL_FOLLOW = 4        ;this bit indicates following error exceeded
ENABLE_DITHER = 5       ;this bit indicates dither oscilator enabled
ENABLE_NOTCH1 = 8       ;this bit indicates that notch filter 1 is enabled
ENABLE_PES = 7          ;this bit indicates that the PES function is enabled
ENABLE_NOTCH0 = 6       ;this bit indicates that notch filter 0 is enabled
ENABLE_OUTDITH = 9      ;this bit indicates to add dither oscilator to output of PID

;
;----------------------------------------------------------------------
;
; offsets for the status structure (PID_STATUS)
;
;  PID loop only needs to know about the first four values
;
;----------------------------------------------------------------------
;

FOLLOW_ERROR = 0
STATUS       = 1
FOLLOWGAIN   = 2
FOLLOWREG    = 3
SERVOFLAGS   = 4

;
```

```
;------------------------------------------------------------------------
;
;offsets for filter parameters structure (PID_PARAMS)
;
;------------------------------------------------------------------------
;
SIZEOF_FILTPARAMS   =   37
FLAGS  = 36         ;Control Flags for Motion Profile
JERK   = 34         ;Value of Jerk for Profile
VCOUNT = 33         ;Constant Vel Count
SCOUNT = 32         ;S Curve count Register
PES_SCALE = 31      ;Position Error Signal Scale
PES_LIMIT = 30      ;Position Error Signal Limit
POS_P   =   28      ;position at offset 12, long
AMPLITUDE = 27      ;oscilator amplitude
FOL_LIMIT  = 26     ;maximum following error allowed
INTEGRATE_FLAG = 25 ;Set to zero if command vel != 0, 1 otherwise
INT_LIM =   24      ;integrator limit  ,int
INT_P   =   23      ;integral gain ,int
FFWV_P  =   22      ;feed forward vel gain , int
FFWA_P  =   21      ;feed forward acc gain , int
DIFF_P  =   20      ;differential gain , int
PROP_P  =   19      ;proportional gain , int
OFF_P   =   18      ;offset , int
NOTCHQ1 = 17            ;notch filter Q
NOTCHF1 = 16            ;notch filter Frequency
NOTCHD1 = 15            ;notch filter depth
NOTCHQ0 = 14            ;notch filter Q
NOTCHF0 = 13            ;notch filter Frequency
NOTCHD0 = 12        ;notch filter depth
METERIN1 = 11       ;pointer to meter input
METERFILT1 = 10     ;frequency of meter filter
ZFCOS1 = 8          ;Cosine Filter State Variable
ZFSIN1 = 6          ;Sin Filter State Variable
METERIN0 = 5        ;pointer to meter input
METERFILT0 = 4      ;frequency of meter filter
ZFCOS0 = 2          ;Cosine Filter State Variable
ZFSIN0 = 0          ;Sin Filter State Variable

;
; DEFINES FOR BITS IN FLAGS
;
FLAGS_START =   0   ;indicates profile is to START
FLAGS_ACC1  =   1   ;Acceleration number 1
FLAGS_ACC2  =   2   ;Deceleration number 2
FLAGS_CVEL  =   3   ;Constant Velocity
FLAGS_ACC3  =   4   ;Deceleration number 3
FLAGS_ACC4  =   5   ;Acceleration number 4
FLAGS_END   =   6   ;indicates that profile has stopped
;
;------------------------------------------------------------------------
;
;offsets into filter values data structure  (PID_PROFILE)
;
;------------------------------------------------------------------------
;
SIZEOF_PID_PROFILE  =   27
PROFCNT =       26      ;Profile State Machine Counter
JERK    =       24      ;storage for jerk register
ACCEL   =       22      ;storage for accel register
VELOC   =       20      ;storage for velocity
GOAL    =       17      ;storage for GOAL from Profile Generator
RELPOS  =       16      ;Relative Position
DGOAL   =       14      ;This is the REAL goal
FOLLOWERROR =   13      ;storage for following error
ZI_V    =       12      ;storage for integrator temp , int
```

```
NOTCH1IN    =   11      ;Response Input for Notch Filter 1
ZBP1 =          9       ;storage for Band Pass State Variable
ZLP1 =          7       ;Storage for Low Pass State Variable
NOTCH0IN    =   5       ;Response Input to Notch Filter 0
ZBP0 =          4       ;storage for Band Pass State Variable
ZLP0 =          2       ;Storage for Low Pass State Variable
PIDOUT =        1       ;Output from PID loop
MOTRIN =        0       ;Input to the motor
;
;------------------------------------------------------------------------
;
    section pid_ydata
    org Y:
    global FFiltParams
    global FMiscParams
    global FRTCShadow       ;shadow of RTC reg
    global FSStatus         ;Various Status for servo loop WRITE only

FFiltParams ds SIZEOF_FILTPARAMS*3  ;space for filter parameters
FMiscParams ds 96   ;space for misc parameters
    endsec

    section pid_ydata_slow
    org Y:
    global FHistogram
    global FJogParams
    global FHomeParams      ;space for homeing parameters

FJogParams  ds 24   ;eight words to store jog params
FHomeParams ds 24   ;space for homeing parameters

FHistogram  ds 99   ;33 words to store histogram (3 axis)

    endsec
;
;------------------------------------------------------------------------
;
;internal data for the filter
;
;------------------------------------------------------------------------
    section pid_xdata
    org x:

    global Ffiltvals

Ffiltvals ds    SIZEOF_PID_PROFILE*3        ;storage space for various filter variables
    endsec

    section pid_code
    org P:
    global FPid
;
;************************************************************************
;
; PID filter, called from real time interrupt
;
;************************************************************************
;
FPid
    ;
    ;-------------------------------------------------------------
    ;
    ; first thing to do is calculate new goal position
    ;
    ;
    ;
```

```
    ; Profile generator uses this algorythm
    ;
    ; An = An-1 + Jerk
    ; Vn = Vn-1 + An
    ; Xn = Xn-1 + Vn
    ;
    ; An is used for feed forward acceleration
    ; Vn is used for feed rorward velocity
    ;
    ; The profile Status Flags are as follows
    ;
    ; Bit 0:    Enable Motion Generator
    ;           This flag allows outside program to manipulate
    ;           Motion parameters without generating bogus motions
    ; Bit 1:    Acceleration Profile Segment 1
    ; Bit 2:    Acceleration Profile Segment 2
    ; Bit 3:    Constant Velocity Profile Segment
    ; Bit 4:    Acceleration Profile Segment 3
    ; Bit 5:    Acceleration Profile Segment 4
    ; Bit 6:    Profile Generation Stopped
    ;
    ; r1- points to X Memory TOP image of Profile Data
    ; r2- points to hardware registers
    ; r3- points to Y Memory BOTTOM image of Axis Status
    ; r5- points to Y Memory TOP image of Profile Parameters
    ;
    ; uses registers a,b,x,y,r0,n0,r1,n1,r2,r3,n3,r4,r5,n5
    ;
    ;---------------------------------------------------------------
    ;
    move    #1,n1                       ;offset to profile count
    move    x:(r1)-,a                   ;get profile count,r1 points to jerk
    jclr    #FLAGS_START,y:(r5),_PID101 ;Start Profile Generation
    bclr    #FLAGS_START,y:(r5)         ;clear start flag
    bset    #FLAGS_ACC1,y:(r5)-         ;set inprocess flag point r5 to Jerk
    move    y:(r5)-,b                   ;jerk->b    point r5 to VCount
    move    y:(r5)-,b0
    move    b,x:(r1)-                   ;b->jerk for profile
    move    (r5)-                       ;point r5 to SCount
    move    b0,x:(r1)+
    move    y:(r5)-,x0                  ;SCount->x0 point r5 to position
    move    x0,x:(r1+n1)                ;x0->ProfileCount
    jmp     _PID107                     ;jump around
_PID101
    ;
    jclr    #FLAGS_ACC1,y:(r5),_PID102  ;Acceleration #1 segment
    tst     a                           ;is ProfileCount == 0?
    jne     _PID106                     ;nothing to do yet
    bclr    #FLAGS_ACC1,y:(r5)          ;clear ACC1 flag
    bset    #FLAGS_ACC2,y:(r5)-         ;set ACC2 flag, r5 points to jerk
    move    y:(r5)-,b                   ;get jerk   r5 points VCount
    move    y:(r5)-,b0
    neg     b   (r5)-                   ;make negative  r5 points to SCount
    move    b,x:(r1)-                   ;b->jerk
    move    b0,x:(r1)+
    move    y:(r5)-,x0                  ;SCount->x0, r5 points to position
    move    x0,x:(r1+n1)                ;x0->ProfileCount
    jmp     _PID107
_PID102
    jclr    #FLAGS_ACC2,y:(r5),_PID103  ;deceleration #2 segment
    tst     a                           ;is ProfileCount == 0?
    jne     _PID106                     ;nothing to do yet
    bclr    #FLAGS_ACC2,y:(r5)          ;clear ACC2 flag
    move    #-3,n5                      ;-3->n5 to point to VCount
    move    y:(r5+n5),b
    tst     b                           ;is there a constant vel count?
```

```
    jne     <_PID202                    ;yes there is
    bset    #FLAGS_ACC3,y:(r5)-         ;no set ACC3 flag   r5 points jerk
    move    (r5)-                       ;r5 still points to jerk
    move    (r5)-                       ;r5 points to vcount
    ;
    ; no need to change jerk
    ;
    move    (r5)-                       ;r5 points to SCount
    move    y:(r5),x0
    move    x0,x:(r1+n1)                ;save in profile count
    move    (r5)-                       ;r5 points to position
    jmp     _PID107
_PID202
    bset    #FLAGS_CVEL,y:(r5)-         ;set CVEL flag  r5 points to jerk
    move    (r5)-                       ;r5 still points to jerk
    move    (r5)-                       ;r5 points to vcount
    clr     a   y:(r5)-,x0              ;vcount ->x0, r5 points to SCount
    move    (r5)-                       ;r5 points to position
    move    a,x:(r1)-                   ;jerk = 0
    move    a0,x:(r1)+
    move    x0,x:(r1+n1)                ;x0->ProfileCount
    jmp     _PID107
_PID103
    jclr    #FLAGS_CVEL,y:(r5),_PID104  ;constant velocity segment
    tst     a                           ;is ProfileCount == 0?
    jne     _PID106                     ;nothing to do yet
    bclr    #FLAGS_CVEL,y:(r5)          ;clear CVEL flag
    bset    #FLAGS_ACC3,y:(r5)-         ;Set ACC3 flags,r5 points to jerk
    move    y:(r5)-,b                   ;get jerk   r5 points VCount
    move    y:(r5)-,b0
    neg     b   (r5)-                   ;make negative  r5 points to SCount
    move    b,x:(r1)-                   ;b->jerk
    move    b0,x:(r1)+                  ;r1 points to jerk
    move    y:(r5)-,x0                  ;SCount->x0, r5 points to position
    move    x0,x:(r1+n1)                ;x0->ProfileCount
    jmp     _PID107
_PID104
    jclr    #FLAGS_ACC3,y:(r5),_PID105  ;deceleration #3 segment
    tst     a                           ;is ProfileCount == 0?
    jne     _PID106                     ;nothing to do yet
    bclr    #FLAGS_ACC3,y:(r5)          ;clear ACC3 flag
    bset    #FLAGS_ACC4,y:(r5)-         ;set ACC4 flag
    move    y:(r5)-,b                   ;jerk->b    point r5 to VCount
    move    y:(r5)-,b0
    move    b,x:(r1)-                   ;b->jerk for profile
    move    (r5)-                       ;point r5 to SCount
    move    b0,x:(r1)+
    move    y:(r5)-,x0                  ;SCount->x0 point r5 to position
    move    x0,x:(r1+n1)                ;x0->ProfileCount
    jmp     _PID107
_PID105
    jclr    #FLAGS_ACC4,y:(r5),_PID106  ;acceleration #4 segment
    tst     a                           ;is ProfileCount == 0?
    jne     _PID106                     ;nothing to do yet
    bclr    #FLAGS_ACC4,y:(r5)          ;clear ACC4 flag
    bset    #FLAGS_END,y:(r5)-          ;end of motion flag, r5 points to jerk
    clr     a   (r5)-                   ;r5 still points to jerk
    move    (r5)-                       ;r5 points to vcount
    move    (r5)-                       ;r5 points to scount
    move    (r5)-                       ;r5 points to position
    move    a,x:(r1)-                   ;jerk = 0;
    move    a0,x:(r1)+
    jmp     _PID107
_PID106
    ;
    ; profile is idle (done).
```

```
    ;
    move    (r5)-                       ;r5 points to jerk
_PID116
    move    (r5)-                       ;r5 points to jerk still
    move    (r5)-                       ;r5 points to vcount
    move    (r5)-                       ;r5 points to scount
    move    (r5)-                       ;r5 points to position
_PID107
    move    x:(r1+n1),a
    move    #>1,x0
    sub     x0,a
    move    a,x:(r1+n1)         ;decrement ProfileCount
    ;
    ;*************************************************************
    ;
    ; This is where we use the PES info to correct for drift
    ;
    ; r5 points to PES Scale
    ;
    ;*************************************************************
    ;
    ; always read in the PES params
    move    #SERVOFLAGS,n3                  ;index to ServoFlags
    move    y:(r5)-,y0      ;PES Scale ->y0,r5 points to PES Limit
    clr a   y:(r5)-,b       ;PES Limit ->B, r5 points to Amplitude
    jclr    #ENABLE_PES,y:(r3+n3),_PID510   ;jump if PES not enabled
    ;
    ; do PES calculation
    ;
    move    y:RDPES_REG,a
    tst     a           ;if PES value is negative, negate limit in B
    jge     >_PID511
    neg     b
_PID511
    cmpm    b,a         ;check for bounds
    tgt     b,a         ;perform limit
    move    a,y1        ;move PES to y1 to prepare for scaling
    mpy     y1,y0,a     ;put scaled value back into a
    clr a   a,x0        ;move scaled PES ->x0
    jmp     >_PID512
_PID510
    move    a,x0        ;not using PES, zero offset
    ;*************************************************************
    ;
    ; Calculate new position
    ; A is zero
    ; B is garbage
    ; r2 points to Position Register
    ; r5 points to Position PidParams
    ; x0 contains PES offset for new position
    ;
    ;*************************************************************
    ;
    ;
_PID512
    clr     b   y:(r5)-,a2  ;old position -> A
    move    y:(r5)+,a1      ;read in old position

    move    y:(r2)-,b2      ;read position register, r2 points low position
    move    y:(r2)-,b1      ;read position register, r2 points to following error
    add     x0,b            ;add in PES offset
    move    b2,y:(r5)-      ;save new position
    sub     b,a b1,y:(r5)+  ;calculate velocity r5 points to position
    move    a,y:(r6)+       ;push differential onto stack

    ;*************************************************************
    ;
```

```
    ; do profile calculation, this is done idle or not so that other
    ; routines can manipulate servo positioning, such as HOME
    ;
    ; Also, check to see if frequency synthesizer needed to be added
    ; into goal position
    ; An = An-1 + Jerk
    ; Vn = Vn-1 + An
    ; Xn = Xn-1 + Vn
    ; r1 points to Jerk in X-Memory Space
    ; r5 points to Position in Y-Memory Space
    ;
    ;*************************************************************
    ;
    move    x:(r1)-,x1              ;load in jerk
    move    x:(r1)-,x0
    move    x:(r1)-,a               ;load in acc
    move    x:(r1)+,a0
    add     x,a                     ;a = acc + jerk
    move    a,x:(r1)-               ;save new acc
    move    a0,x:(r1)-              ;acc is in A
    move    x:(r1)-,b               ;load in vel
    move    x:(r1)+,b0
    add     a,b a,y:(r6)+           ;b = vel + acc ,push Acceleration on stack
                                    ;if velocity is non-zero
    clr a   #INTEGRATE_FLAG-POS_P-1,n5      ;index to IFlag
    jclr    #INTEGFLAG,y:(r3+n3),_PID403    ;is integrator disabled on non-zero velocity
    jclr    #INTOFF,y:(r3+n3),_PID401       ;is integrator always on?
    tst     b                       ;check velocity
    jeq     <_PID401                ;zero velocity, integrate else
    move    a,y:(r5+n5)             ;set IFlga = 0 to disable integrator
    jmp     <_PID402

_PID403
    move    a,y:(r5+n5)             ;set IFlag to 0
    jmp     <_PID402                ;end of flag setting

_PID401
    jclr    #INTEGFLAG,y:(r3+n3),_PID402    ;is integrator enabled?
    bset    #22,y:(r5+n5)
_PID402

    move    b,x:(r1)-               ;save new vel
    move    b,y:(r6)+               ;push velocity on stack
    move    b0,x:(r1)-              ;vel is in B
    move    x:(r1)-,a2              ;load in position
    move    x:(r1)-,a1
    move    x:(r1)+,a0
    add     b,a (r1)+               ;a = pos + vel
    clr     b a2,x:(r1)-            ;save position
    move    a1,x:(r1)-
    move    a0,x:(r1)-              ;goal position is in a, r1 points to FollowError
    ;
    ;*************************************************************
    ;
    ; do dither calculation, if enabled
    ; A contains Goal Position
    ; X is Garbage
    ; B is zero
    ; r5 points to real position
    ; r1 points to RelativePosition
    ; r2 points to follow dac
    ; r3 points to following error
    ; r3+n3 points to servo flags
    ; check to see if dither is enabled
    ;
    ;*************************************************************
```

```
    ;
    move    y:(r5)-,x1          ;load in real position to X
    move    y:(r5)-,x0
    move    a2,b                ;copy goal into B
    move    a1,b0
                                ;r5 now points to dither amplitude
    sub     X,B                 ;Calculate relative position
    clr b   b0,x:(r1)-          ;save relative position
    ;
    ;
    ;*****************************************************************
    ;
    ; This is where we add dither
    ; A contains Goal Position
    ; B contains 0
    ; X contains Real position
    ; r5 points to dither amplitude
    ; r1 points to dGoal
    ; r2 points to follow dac
    ; r3 points to following error
    ; r3+n3 points to servo flags
    ;
    ;*****************************************************************
    ;
    move    y:(r5)-,y0          ;y0 contains dither amplitude,r5 points to follow limit
    jclr    #ENABLE_DITHER,y:(r3+n3),_PID501    ;don't modifiy goal if not enabled
    move    #29,n3              ;index to look up table address
    move    y:(r3+n3),r4        ;r4 points to runout lookup table
    move    y:RDCOS_REG,n4      ;read freq into n4 to get index to Sin LUT
    move    y:(r4+n4),y1        ;load in sine value
    mac     y1,y0,a             ;multiply by gain and add to goal
    ;
    ;*****************************************************************
    ;
    ; A contains Goal Position
    ; X contains the REAL position
    ; B is zero
    ; Y is garbage
    ; r5 points to follow limit
    ; r4 points to runout lookup table
    ; n4 is index into sine table (phase)
    ; r2 points to following error dac
    ; r3 points to following error
    ; r3+n3 runout lookup table
    ;
    ;*****************************************************************
    ;
_PID501
    rnd     a                   ;round off goal before saving
    clr b   a2,x:(r1)-          ;save goal to dgoal
    move    a1,x:(r1)-          ;r1 points to following error
    ;
    ;*****************************************************************
    ;
    ; calculate following error
    ; from here, r3 will be incremented n3 is garbage
    ; A contains Goal Position
    ; X contains the REAL position
    ; B is zero
    ; Y is garbage
    ; r5 points to follow limit
    ; r4 points to runout lookup table
    ; n4 is index into runout table (phase)
    ; r2 points to following error dac
    ; r3 points to following error
    ; r3+n3 runout lookup table
```

```
    ;
    ;*****************************************************************
    ;
    ;
    move    x1,b2                   ;load real position into b
    move    x0,b1
    sub     b,a  y:(r5)-,y0         ;goal-position->a, move limit into x0,r5 points IFlag
    move    a,y:(r3)+               ;save following error for later use, r3 points to status
    clr b   a,x:(r1)-               ;save following error, r1 point to Zi
                                    ;r3 points to Status
    cmpm    y0,a    a,y1            ;check limit follow error->y1
    jlt     <_PID300                ;don't set flags if inside limit
    bset    #FATAL_FOLLOW,y:(r3)    ;Set fatal following error flag
    move    #3,n3                   ;index to servo flags
    jclr    #EN_FATAL_FOLLOWING,y:(r3+n3),_PID300   ;????????????????????
    bclr    #SERVFLAG,y:(r3)    ;disable servo loop if fatal following error enabled
    clr     b   #>$000040,y0        ;load mask for servo flags
    move    r1,y:(r6)+          ;push r1 onto stack
    move    #12,n1
    move    (r1)+n1             ;r1 points to jerk
    move    #FLAGS-INTEGRATE_FLAG,n5
    move    y0,y:(r5+n5)        ;set flags to profile done
    rep     #6
    move    b,x:(r1)-           ;zero profile parameters
    move    (r6)-
    clr b   y:(r6),r1           ;restore r1

_PID300
    ;
    ;*****************************************************************
    ;
    ; Time to use integrator hold flag to condition following error
    ; before integration
    ;
    ; A is following error
    ; B is zero
    ; y1 is following error
    ; r1 points to zi
    ; n3 = 3;
    ; r3 points to Status
    ; r3+n3 points to ServoFlags
    ;
    ;*****************************************************************
    ;
    clr a   y:(r5)-,y0  x:(r1),b    ;Zero a, Iflag->y0,zi->b,r5 points Int Limit
    mac     y0,y1,b  y:(r5)-,a  ;integrator limit in a
    jge     <_PID302
    neg     a
_PID302
    cmpm    a,b     (r3)+           ;r3 points to follow gain
    tgt     a,b                     ;limit integrator
    move    b,x0                    ;save integrator
    move    b,x:(r1)-               ;save integration,r1 points to Notch1In
    ;
    ;*****************************************************************
    ;
    ; combine all the terms for the final output
    ;
    ; the stack looks like this:
    ; velocty
    ; aff
    ; vff
    ;
    ; A is garbage
    ; B is garbage
    ; x0 is integrator
    ; y1 is following error
```

```
    ; y0 is garbage
    ; r1 points to Notch1In
    ; r3 points to Follow Gain
    ; n3 is garbage
    ;
    ;*****************************************************************
    ;
    clr     a   (r6)-                   ;decrement stack pointer
    clr     b   y:(r5)-,x1              ;zero b accum, Ki->x1
    mac     x0,x1,b #>1,y0              ;Accum Integrator,foll err SCALE->Y0
    mac     y0,y1,a y:(r6)-,x0          ;Accum follow error, ffvel -> x0
    move    y:(r5)-,x1                  ;Kvff-> x1
    mac     x1,x0,a y:(r6)-,y0          ;Accum KvFF, ffacc->y0
    move    y:(r5)-,y1                  ;Kaff-> y1
    mac     y0,y1,a y:(r6),x0           ;Accum KaFF, Vel->x0
    move    y:(r5)-,x1                  ;Kv->x1
    mac     x0,x1,b y:(r5)-,y0          ;Accum Velocity, Kp->y0
    rep     #8
    asl     a   #2,n3                   ;r3+n3 point to servo flags
    add     a,b                         ;
    clr     a
    move    #>$7fffff,a0                ;limit factor for accumulator
    rep     #6
    asr     b                           ;scale multiply
    jge     <_PID303                    ;check sign of result
    neg     a                           ;make limit same sign
_PID303
    cmpm    a,b
    tgt     a,b                         ;limit result

    move    b0,x0                       ;zero a, sum->x0
    mpy     x0,y0,b y:(r5)-,y1          ;Proportional Gain,Offset->y1
    asr     b                           ;r5 points to NotchQ1
    cmpm    a,b                         ;r5 points to position
    tgt     a,b                         ;limit result again
    move    b0,b                        ;safe to move
    add     y1,b                        ;add in offset
    ;
    ;---------------------------------------------------------------
    ;
    ; b is output of PID
    ; check flags to see what to do with it
    ; r5 points to NotchQ1
    ; r1 points to top of Notch1In
    ;
    ;---------------------------------------------------------------
    ;
    jset    #SERVFLAG,y:(r3+n3),_Pid201 ;Jump if we servo
    move    #POS_P-NOTCHQ1+1,n5         ;don't servo if disabled
    clr     a #GOAL-NOTCH1IN+2,n1
    move    (r5)+n5                     ;r5 points to Pos
    move    (r1)+n1                     ;r1 points to goal top
    move    y:(r5)-,a2                  ;set goal to position
    move    y:(r5)+,a1                  ;this will keep servo from
    move    (r5)-n5                     ;r5 points to NotchQ1
    move    a2,x:(r1)-                  ;banging when enabled
    move    a1,x:(r1)-                  ;
    move    a0,x:(r1)-                  ;
    move    (r1)-                       ;skip over RelativePosition
    move    a2,x:(r1)-                  ;update dGoal
    move    a1,x:(r1)-                  ;r1 points to follow error
    clr     b  (r1)-                    ;output zero to motor,r1 points Zi
    clr     a  (r1)-                    ;r1 points to Notch1In

_Pid201
    ;*****************************************************************
```

```
    ; check to see if we need to do a notch filter
    ; r1 points to top of Zbp1
    ; r5 points to top of position
    ; r3 points to followgain
    ; r3+n3 points to servo flags
    ; b is output to motor
    ;
    ;*********************************************************************
    ;
    move    b,x:(r1)-               ;Save Output of PID/Input to Notch1,r1 points to Zbp1
    move    #>4,n1                  ;constant for r1
    move    #>3,n5                  ;constant for r5
    jclr    #ENABLE_NOTCH1,y:(r3+n3),_Pid202    ;don't notch if not enabled
    ;
    ;*********************************************************************
    ; r1 points to Zbp1
    ; r5 points to NotchQ1
    ;                                       Zbp == Band Pass State Var
    ;                                       Zlp == Low Pass State Var
    ;                                       D   == Depth Variable
    ;                                       Q   == Q variable (resonance)
    ;                                       W   == Frequency Varialbe
    ;                                       h   == High Pass Output
    ;                                       N   == Notch Output
    ; so called Dipole Filter (both boost and notch) Notch[1]
    ;*********************************************************************
    move    x:(r1)-,x1 y:(r5)-,y1           ;Zbp[msb]->x1,Q->y1
    move    x:(r1)-,x0 y:(r5)-,y0           ;Zbp[lsb]->x0,W->y0
    mac -y1,x1,b    x:(r1)-,a               ;Zlp[msb]->a,h=I-Q*Zbp
    move    x:(r1)+,a0                      ;Zlp[lsb]->a0,Zbp->b
    mac     x1,y0,a #3,n1                   ;Zlp = Zlp + W * Zbp
    sub     a,b     a,x:(r1)-               ;h=h-Zlp,save Zlp[msb]
    move    a0,x:(r1)+n1                    ;save Zlp[lsb],r1 points to Zbp[msb]
    clr a   b,y1                            ;h[msb]->y1
    add     x,a #4,n1                       ;Zbp->a
    mac     y1,y0,a y:(r5)-,x0              ;Zbp=W * h + Zbp,D->x0 (depth)
    move    a,x:(r1)-                       ;Save Zbp[msb]
    move    a0,x:(r1)-                      ;save Zbp[lsb]
    move    x:(r1)-,b                       ;Zlp[msb]->b N=Zlp
    add     y1,b    (r1)-                   ;N+=hp
    rep     #6
    asr     b                               ;scale sum
    mac     x0,x1,b                         ;N+=D * Zbp
    rep     #6
    asl     b                               ;rescale sum
    jmp     <_Pid203
    ;
    ;*********************************************************************
    ; second notch filter   (Normal Filter) Notch[0]
    ;*********************************************************************
    ;
_Pid202
    move    (r1)-n1                         ;update pointers to
    move    (r5)-n5                         ;next notch filter
_Pid203
    ;
    move    b,x:(r1)-       ;save Input to Notch0,r1 points to Zbp0
    jclr    #ENABLE_NOTCH0,y:(r3+n3),_Pid222    ;don't notch if not enabled
    ;
    ;*********************************************************************
    ; r1 points to Zbp0
    ; r5 points to NotchQ0
    ;                                       Zbp == Band Pass State Var
    ;                                       Zlp == Low Pass State Var
    ;                                       D   == Depth Variable
    ;                                       Q   == Q variable (resonance)
```

```
    ;                                       W   == Frequency Varialbe
    ;                                       h   == High Pass Output
    ;                                       N   == Notch Output
    ;*************************************************************************
    move    x:(r1)-,x1 y:(r5)-,y1           ;Zbp[msb]->x1,Q->y1
    move    x:(r1)-,x0 y:(r5)-,y0           ;Zbp[lsb]->x0,W->y0
    mac -y1,x1,b    x:(r1)-,a               ;Zlp[msb]->a,h=I-Q*Zbp
    move    x:(r1)+,a0                      ;Zlp[lsb]->a0,Zbp->b
    mac     x1,y0,a #3,n1                   ;Zlp = Zlp + W * Zbp
    sub     a,b     a,x:(r1)-               ;h=h-Zlp,save Zlp[msb]
    move    a0,x:(r1)+n1                    ;save Zlp[lsb]
    clr a   b,y1                            ;h[msb]->y1
    add     x,a                             ;Zbp->a
    mac     y1,y0,a y:(r5)-,x0              ;Zbp=W * h + Zbp,D->x0 (depth)
    move    a,x:(r1)-                       ;Save Zbp[msb]
    move    a0,x:(r1)-                      ;save Zbp[lsb]
    move    x:(r1)-,b                       ;N=Zlp  (Zlp->b)
    add     y1,b    (r1)-                   ;N+=hp
    mac     x0,x1,b                         ;N+=D * Zbp
    jmp     <_Pid223

    ;
    ;*************************************************************************
    ; output finally to the motor
    ;*************************************************************************
_Pid222
    move    (r1)-n1
    move    (r5)-n5                 ;r5 points to METERIN1
_Pid223
    move    b,x:(r1)-               ;save output of PID loop
    clr     a   #$100,r4            ;address of SLUT->r4
    jclr    #ENABLE_OUTDITH,y:(r3+n3),_Pid310   ;jump if we do not add dither
    move    y:RDSIN_REG,n4          ;read freq phase into n4 to get index to Sin LUT
    move    #AMPLITUDE-METERIN1,n5  ;index to amplitude
    move    y:(r4+n4),x1            ;load in sine value
    move    y:(r5+n5),x0            ;load in sine amplitude
    mac     x1,x0,b                 ;accumulate to output
_Pid310
    move    #FOLLOWERROR,n1
    move    b,x:(r1)+n1             ;save input to motor
    move    b,y:(r2)                ;output to motor
    ;
    ;
    ;*************************************************************************
    ;
    ; calculate spectrum analyzer outputs
    ; r4 points to sine table
    ; n4 is index into sine table
    ; r5 points to MeterIn1
    ; r3 points to follow gain
    ; r2 is no longer used, it will be used for other purposes now
    ; r1 points to follow error
    ;   FIRST FREQUENCY ANALYZER
    ;*************************************************************************
    ;
    clr a   y:(r5)-,r2              ;load pointer to input data->r2
    move    y:(r5)-,x1              ;cutoff frequency->x1
    move    y:(r5)-,b               ;sine Z-1->b
    move    y:(r5)+,b0              ;r5 points to top of sine Z-1
    move    x:(r2),x0               ;read in thing to measure
    ;---------------------------------
    ; check input to analysis
    ; if non zero use that
    ; pointer
    ;---------------------------------
    move    y:RDCOS_REG,n4          ;read freq into n4 to get index to Sin LUT
```

```
    clr a   b,y0                    ;sine Z-1 msb -> y0
    move    y:(r4+n4),y1            ;sine table value -> y1
    mac     x0,y1,a                 ;syncronous detector
    sub     y0,a                    ;subtract Z-1
    move    a,y0                    ;result -> y0
    mac     x1,y0,b                 ;perform filter operation
    clr a   b,y:(r5)-               ;save filter output
    move    b0,y:(r5)-              ;r5 points to cosine Z-1
    ;*************************************************************
    ; y0 is garbage
    ; y1 is garbage
    ; x0 is deviation
    ; x1 is cutoff frequency
    ; a is 0
    ; b is garbage
    ; r3 points to followgain
    ;*************************************************************
    move    y:RDSIN_REG,n4          ;index of sine oscilator
    move    y:(r5)-,b               ;cosine Z-1 -> b
    move    y:(r5)+,b0              ;r5 points to cosize Z-1
    move    y:(r4+n4),y1            ;cosine table value -> y1
    move    b,y0                    ;cosine Z-1 msb -> y0
    mac     x0,y1,a                 ;syncronous detector
    sub     y0,a                    ;subtract Z-1 follow gain->y1
    move    a,y0                    ;result -> y0
    mac     x1,y0,b                 ;perform filter opration
    move    b,y:(r5)-               ;save filter result
    move    b0,y:(r5)-
    ;
    ;*************************************************************************
    ; SECOND FREQUENCY ANALYZER
    ;*************************************************************************
    ;
    move    y:(r5)-,r2              ;load pointer to input data->r2
    move    y:(r5)-,x1              ;cutoff frequency->x1
    move    y:(r5)-,b               ;sine Z-1->b
    move    y:(r5)+,b0              ;r5 points to top of sine Z-1
    move    x:(r2),x0               ;read in thing to measure
    ;---------------------------------
    ; check input to analysis
    ; if non zero use that
    ; pointer
    ;---------------------------------
    move    y:RDCOS_REG,n4          ;read freq into n4 to get index to Sin LUT
    clr a   b,y0                    ;sine Z-1 msb -> y0
    move    y:(r4+n4),y1            ;sine table value -> y1
    mac     x0,y1,a                 ;syncronous detector
    sub     y0,a                    ;subtract Z-1
    move    a,y0                    ;result -> y0
    mac     x1,y0,b                 ;perform filter operation
    clr a   b,y:(r5)-               ;save filter output
    move    b0,y:(r5)-              ;r5 points to cosine Z-1
    ;*************************************************************
    ; y0 is garbage
    ; y1 is garbage
    ; x0 is deviation
    ; x1 is cutoff frequency
    ; a is 0
    ; b is garbage
    ; r3 points to followgain
    ;*************************************************************
    move    y:RDSIN_REG,n4          ;index of sine oscilator
    move    y:(r5)-,b               ;cosine Z-1 -> b
    move    y:(r5)+,b0              ;r5 points to cosize Z-1
    move    y:(r4+n4),y1            ;cosine table value -> y1
    move    b,y0                    ;cosine Z-1 msb -> y0
```

```
    mac     x0,y1,a                 ;syncronous detector
    sub     y0,a                    ;subtract Z-1 follow gain->y1
    move    a,y0                    ;result -> y0
    mac     x1,y0,b                 ;perform filter opration
    move    b,y:(r5)-               ;save filter result
    move    b0,y:(r5)-              ;r5 points to garbage
    ;
    ;*********************************************************************
    ; position histogram
    ; r1 points to follow error
    ; r3 points to follow gain
    ;*********************************************************************
    ;
    move    #FHistogram+33,r4           ;pointer to histogram
    move    y:(r4),a                    ;load histogram counter
    tst     a   #>1,y1                  ;check value load 1 into x0
    jle     <_Pid211                    ;if zero bypass
    sub     y1,a    #>31,x1             ;decrement counter
    move    a,y:(r4)+                   ;save count, r0 points to hist data
    move    #>16,x0
    move    x:(r1),a                    ;following error->a
    add     x0,a    #$ffffe0,y0         ;offset following error
    and     x1,a    a,b                 ;strip off bits
    and     y0,b    a1,n4               ;check for overflow condition
    jne     <_Pid211                    ;don't histogram on overflow
    move    y:(r4+n4),a
    add     y1,a
    move    a,y:(r4+n4)                 ;increment histogram bin
_Pid211
    ;
    clr b   y:(r3)+,x1                  ;Follow Error Gain->y1,r3 point to follow error out
    move    y:(r3)+,r2                  ;follow out pointer->r2
    move    x:(r1),x0
    mpy     x0,x1,a #>$7fffff,b0        ;calculate following error out
    tst a
    jge >_Pid221
    neg b
_Pid221
    cmpm    b,a
    tgt     b,a
    move    a0,y:(r2)               ;output following error
    rts

;*************************************************************************
;
; Special Control Loop
;
;  This control loop code is special for the spindle motor.  It is intended
; to be used for speeding up and slowing down the spindle motor.
;
;   It uses a much cruder profile generator.
;
;   Vn = Vn-1 + An
;
;   This will cause the velocity to slew up and down in speed.
;
; The parameter blocks used for the normal PID are also used for this
; control loop.  Some of the parameters have changed their name, many more
; are just not used.
;
; in PID_PARAMS:
;
;   Jerk is turned into acceleration
;   Amplitude is turned into old velocity (this might cause a problem)
;
; in PID_PROFILE:
```

```
;
;   Jerk is turned into acceleration
;   Accel is turned into velocity
;
; Register usage is the same as PID
; r4 is used as general pointer
; r3 points to FollowingError
; r5 points to PID data     (FLAGS)
; r1 points to profile data (Profile Count)
;
;*************************************************************************

    global FVID

FVID
    bset    #6,x:PCD
    move    #PROFCNT-ACCEL-1,n1     ;offset from profile count to accel
    move    x:(r1)-n1,a             ;load profile count into a, r1 points to accel
    jclr    #FLAGS_START,y:(r5),_VID101 ;start profile?
    bclr    #FLAGS_START,y:(r5)     ;reset start flag
    bset    #FLAGS_ACC1,y:(r5)-     ;set accel flag, r5 points to accel (JERK)
    move    y:(r5)-,b               ;(PID_PARAMS)accel -> b
    move    y:(r5)-,b0              ;r5 now points to VCount (not used)
    move    b,x:(r1)-               ;b -> (PID_PROFILE)accel
    move    b0,x:(r1)+              ;r1 points to accel
    move    (r5)-                   ;r5 points to SCount
    move    y:(r5)-,a               ;SCount -> a,r5 points to PESScale
    move    (r5)-                   ;r5 points to PESLimit
    move    (r5)-                   ;r5 points to position
    jmp     _VID107
_VID101
    jclr    #FLAGS_ACC1,y:(r5),_VID106  ;acceleration mode?
    tst     a                       ;is profile count == 0?
    jne     _VID106                 ;no, keep wait
    bclr    #FLAGS_ACC1,y:(r5)      ;yes time to stop motion
    bset    #FLAGS_END,y:(r5)-      ;set profile done flag, r5 points accel
    clr     b   (r5)-               ;r5 still points accel
    move    b,x:(r1)-
    move    b0,x:(r1)+              ;r1 still points to accel
    move    (r5)-                   ;r5 points to VCount
    move    (r5)-                   ;r5 points to SCount
    move    (r5)-                   ;r5 points to PESScale
    move    (r5)-                   ;r5 points to PESLimit
    move    (r5)-                   ;r5 points to position
    jmp     _VID107                 ;jump arount
_VID106
    move    #7,n5
    move    (r5)-n5                 ;r5 points to position
_VID107
    move    #>1,x0                  ;prepare to subtract one from profile count
    sub     x0,a
    move    a,x:(r1+n1)             ;decrement profile count
;*************************************************************************
;
; Calculate Position, velocity, and acceleration
;
;*************************************************************************

    clr a                           ;clear accumulators
    clr b   y:(r5)-,a2              ;load in old pos, r5 points to pos low
    move    y:(r5)+,a1              ;r5 points to position
    move    y:(r2)-,b2              ;load in velocity register
    move    y:(r2)-,b1              ;r2 now points to following error
    add     b,a                     ;calculate new position
    move    a2,y:(r5)-              ;save new position
    move    a1,y:(r5)-              ;r5 points to Amplitude (not used)
```

```
    move    b,y1                    ;save velocity in y1
    move    (r5)-                   ;r5 points to follow limit (not used)

;*************************************************************************
;
; Do Profile calculation
;
; y0 contains junk
; y1 contains velocity
; a is junk
; b is junk
; r5 points to follow limit
; r1 points to acceleration
;
;*************************************************************************

    move    x:(r1)-,x1              ;accel->X
    move    x:(r1)-,x0              ;r1 points to velocity
    move    x:(r1)-,b               ;r1 points to velocity
    move    x:(r1)+,b0              ;r1 points to veloc
    add     x,b     #6,n1           ;calculate new velocity
    move    b,x:(r1)-               ;save new velocity
    move    b0,x:(r1)-              ;r1 points to other Goal[2] (not used)

;*************************************************************************
;
; b contains goal velocity
; y0 contains junk
; y1 contains velocity
; r5 points to follow limit
; r1 points really to nothing (Goal member in PID_PROFILE)
; r3 points to following error
;
;*************************************************************************
;   move    #6,n1
;   move    (r1)-n1                 ;r1 points to follow error
    move    y1,a
    sub     b,a (r1)-n1             ;calculate velocity following error
    move    a,y:(r3)+               ;save following error, r3 points to status
    move    a,x:(r1)-               ;save following error, r1 points to zi
    move    (r3)+                   ;r3 points to follow gain
    clr b   a,x0                    ;save following error in x0
    clr a   y:(r3)+,x1
                                    ;follow error gain->x1, r3 points follow address
    mac     x0,x1,b y:(r3)+,r4      ;scale following error for output
                                    ;load address of follow error output
                                    ;r3 points to servo flags
    tst b   #>$7fffff,a0
    jge     <_VID401
    neg     a
_VID401
    cmpm    a,b (r5)-               ;r5 points to iflag
    tgt     a,b
;   asl     b   (r5)-               ;r5 points to iflag
;   asl     b   (r5)-               ;r5 points to integrator limit
;   rep     #29
;   asl     b                       ;more scaling
    move    (r5)-                   ;r5 points to integrator limit
    move    b0,y:(r4)               ;output following error
    move    x:(r1),b                ;load in integrator (stored in other velocity)
    add     x0,b y:(r5)-,a          ;add following error, limit ->a,r5 points Ki
    jge     <_VID202
    neg     a                       ;limit maximum value of integrator
_VID202
    cmpm    a,b
    tgt     a,b
```

```
    move    b,x:(r1)                ;store back in integrator
    move    b,y0


;************************************************************************
;
; combine data and output to motor dac
;
; y0 contains integrator
; x0 contains following error
; r5 points to Ki
;
;************************************************************************

    clr b   #3,n5                   ;r5 points to integral gain
    clr a   y:(r5)-n5,y1            ;load integral gain, r5 points velocity gain
    move    y:(r5)-,x1              ;load velocity gain->x1,r5 points to offset
    mac     y0,y1,a                 ;calculate differential contribution
    mac     x0,x1,a                 ;following error contribution
    rep     #24
    asl     a                       ;final output is in a
    jset    #SERVFLAG,y:(r3),_VID201    ;don't servo if disabled
    clr     a                       ;zero accum
_VID201
    move    a,y:(r2)                ;output to servo dac
    bclr    #6,x:PCD
    rts


    endsec

    section pid_aux_code
    org P:

;
;************************************************************************
;
; For the Linear Axis, we need only get the position
;
; pretty much same entry conditions as for Pid
;
;************************************************************************

    global Pos
Pos
    clr     b               ;Clear accumulators

    move    y:(r2)-,b2      ;read position register, r2 points low position
    move    y:(r2),b1       ;read position register, r2 points to following error

    move    b2,y:(r5)-      ;save new position
    move    b1,y:(r5)       ;calculate velocity r5 points to position
    rts
;
;************************************************************************
;
; initialize the PID filters working data
;
;************************************************************************
;
    global FInitPID
FInitPID
    move    r1,y:(r6)+      ;save index register 1
    move    #Ffiltvals,r1   ;get address
    clr     a               ;clear accumulator
    do      #SIZEOF_PID_PROFILE+SIZEOF_PID_PROFILE+SIZEOF_PID_PROFILE,_loop     ;clear data
```

```
    move    a,x:(r1)+       ;write zero into ram 32 times
_loop
    move    #FFiltParams,r1
    rep     #SIZEOF_FILTPARAMS*3
    move    a,y:(r1)+       ;clear out y memory
    ori     #4,OMR          ;enable internal ROM
    move    (r6)-
    move    y:(r6),r1       ;restore index register
    tst     a
    rts



    global FGetGoal

FGetGoal
    ;----------------------------------------
    ; this function returns goal as a long
    ; accepts (int)axis as parameter
    ;----------------------------------------
    move    r0,y:(r6)+
    move    ssh,y:(r6)+
    move    #3,n6
    move    r6,r0
    move    (r6)+n6
    move    x0,y:(r6)+  ;save X register
    move    x1,y:(r6)+
    move    r1,y:(r6)+  ;save address register
    move    n1,y:(r6)+  ;save index register
    move    sr,y:(r6)+  ;save status register so interrupts can be disabled
    ;------------------------------------------------
    ; prepare to get goal
    ;------------------------------------------------
    move    #65533,n0       ;offset of axis number
    move    y:(r0+n0),x1    ;get the axis number
    move    #>SIZEOF_PID_PROFILE,x0         ;scale axis to make index
    mpy     x0,x1,a         ;calculate array index
    asr     a               ;scale computation
    move    a0,n1           ;move index into r1
    move    #>Ffiltvals+GOAL+2,r1 ;address of array
    move    (r1)+n1         ;move address to address register
    ;------------------------------------------------
    ; r1 points to top of goal
    ;------------------------------------------------
    ori     #$3,mr      ;DISABLE INTERRUPTS
    move    x:(r1)-,a2  ;lives in X memory
    move    x:(r1)-,a1  ;load in all of goal
    move    x:(r1),a0
    rnd     a           ;round off result
    rep #24
    asr     a           ;convert to a long

    move    (r6)-       ;restore stack frame
    move    y:(r6)-,sr
    move    y:(r6)-,n1
    move    y:(r6)-,r1
    move    y:(r6)-,x1
    move    y:(r6),x0
    move    (r0)-
    move    y:(r0)-,ssh
    move    r0,r6
    tst a   y:(r0),r0
    rts

    ;
;-----------------------------------------------
```

```
;void SetGoal(int axis,long v)
;   This function sets both the goal and position
; To the value V
;-----------------------------------------------
    global FSetGoal
FSetGoal
    move    r0,y:(r6)+
    lua     (r6)+,r0
    move    x0,y:(r6)+
    move    x1,y:(r6)+
    move    y0,y:(r6)+
    move    y1,y:(r6)+
    move    r1,y:(r6)+
    move    n1,y:(r6)+
    move    r2,y:(r6)+
    move    n2,y:(r6)+
    move    r3,y:(r6)+
    move    n3,y:(r6)+
    move    sr,y:(r6)+
    ;
    ;---------------------------------------
    ; Get Stuff From off of Stack
    ;---------------------------------------
    ;
    move    #>SIZEOF_PID_PROFILE,x0             ;scale axis to get index
    move    #65533,n0           ;offset of axis on stack
    move    y:(r0+n0),y0        ;y0 contains axis number
    mpy     x0,y0,a             ;calculate index
    asr     a                   ;scale multiply
    move    a0,n1
    move    #Ffiltvals+GOAL+2,r1        ;address of top of goal into r1
    move    (r1)+n1                 ;r1 points to top of filtvals[axis].goal
    move    #FFiltParams+POS_P+1,r2 ;address of top of FiltParams.position
    move    #>SIZEOF_FILTPARAMS,x0  ;scale to get position index
    mpy     x0,y0,a
    asr     a
    move    a0,n2
    move    (r2)+n2             ;r2 now points to top of FiltParams[axis].position
    ;---------------------------------------
    ;y0 = axisnumber
    ;r1 points to goal
    ;r2 points to position
    ;test axis and generate bit patern to clear
    ;position counters
    ;---------------------------------------
    move    #POS_H_REG,r3   ;address of position register rotary
    move    y0,a
    tst     a
    jeq     <_SetGoal01         ;axis is zero,(LINEAR)
    move    #>1,x0              ;is this rotary axis?
    cmp     x0,a                ;compare axis number to one
    jne     <_SetGoal11         ;assume that it is spindle axis
    move    #>$800,x0           ;bit pattern for rotary axis
    move    #>ROTARY,x1         ;hardware descriptor for rotary axis
    jmp     <_SetGoal02
_SetGoal01                      ;LINEAR axis processing
    move    #>$20,x0            ;bit patern for Linear Axis
    move    #>LINEAR,x1         ;hardware descriptor for linear axis
    jmp     <_SetGoal02
_SetGoal11
    move    #>$1000,x0          ;bit pattern for spindle axis
    move    #>SPIN,x1           ;hardware descriptor for spindle axis
    ;---------------------------------------
    ;load and scale goal value
    ;---------------------------------------
_SetGoal02
```

```
    move    #65531,n0           ;index of value V
    move    (r0)+
    move    (r0)+n0
    move    y:(r0)-,y1          ;register Y contains new goal
    move    y:(r0)-n0,y0
    move    y1,a
    move    y0,a0
    rep     #24
    asl     a                   ;scale A for 32.24
    ori     #3,mr               ;DISABLE INTERRUPTS

    move    x0,y:CLEARIRQ   ;clear position
    nop                     ;this should fix this problem as well, FUCK!
    nop
    move    x1,y:POS_H_DESC ;write hardware descriptor

    move    a2,x:(r1)-  ;save into X memory space (goal)
    move    a1,x:(r1)-
    move    a0,x:(r1)-
    move    (r1)-       ;skip over RelativePosition
    move    a2,x:(r1)-  ;save into X memory space (dgoal)
    move    a1,x:(r1)-
    move    a2,y:(r2)-  ;save into Y memory space (position)
    move    a1,y:(r2)-
    move    a2,y:(r3)-  ;save into position register
    nop                 ;sure hope these nop's work
    nop                 ;yep, they do, bypasses problem in gate array
    move    a1,y:(r3)-

    move    (r6)-
    move    y:(r6)-,sr  ;enable interrupts
    move    y:(r6)-,n3
    move    y:(r6)-,r3
    move    y:(r6)-,n2
    move    y:(r6)-,r2
    move    y:(r6)-,n1
    move    y:(r6)-,r1
    move    y:(r6)-,y1
    move    y:(r6)-,y0
    move    y:(r6)-,x1
    move    y:(r6),x0
    move    y:-(r6),r0
    rts

    ;---------------------------------------------------------
    ; void SetVelocity(int axis,long v)
    ;
    ; This function sets the velocity parameter
    ;
    ;---------------------------------------------------------

    global  FSetVelocity
FSetVelocity
    move    r0,y:(r6)+
    lua     (r6)+,r0
    move    b0,y:(r6)+
    move    b1,y:(r6)+
    move    x0,y:(r6)+
    move    y0,y:(r6)+
    move    y1,y:(r6)+
    move    r1,y:(r6)+
    move    n1,y:(r6)+
    move    sr,y:(r6)+

    ;-----------------------------------
    ; Get the AXIS number
```

```
    ;-------------------------------------
    move    #>SIZEOF_PID_PROFILE,x0         ;scale axis by this to get index
    move    #65533,n0       ;offset on stack
    move    y:(r0+n0),y0    ;axis number is in y0
    mpy x0,y0,a             ;calculate index
    asr a                   ;scale result
    move    a0,n1           ;save result in index register
    move    #Ffiltvals+VELOC+1,r1 ;base address of velocity
    move    (r1)+n1         ;r1 points to top of filtvals[axis].velocity
    ;-------------------------------------
    ;get velocity value
    ;-------------------------------------
    move    #65531,n0
    move    (r0)+
    move    (r0)+n0
    move    y:(r0)-,y1      ;put velocity into Y
    move    y:(r0)-n0,y0
    ori     #$3,mr          ;DISABLE Interrupts
    ;-------------------------------------
    ;save value
    ;-------------------------------------
    move    y1,x:(r1)-      ;save into x memory space
    move    y0,x:(r1)

    move    (r6)-
    move    y:(r6)-,sr      ;enable interrupts
    move    y:(r6)-,n1
    move    y:(r6)-,r1
    move    y:(r6)-,y1
    move    y:(r6)-,y0
    move    y:(r6)-,x0
    move    y:(r6)-,b
    move    y:(r6),b0
    move    y:-(r6),r0
    rts


    global FZeroCount

FZeroCount
    move    r0,y:(r6)+
    lua (r6)+,r0
    move    x0,y:(r6)+
    move    x1,y:(r6)+
    move    r1,y:(r6)+
    move    n1,y:(r6)+
    move    r2,y:(r6)+
    move    n2,y:(r6)+
    move    sr,y:(r6)+          ;save status
    ;---------------------------------
    ; Calculate index from axis
    ;---------------------------------
    move    #>SIZEOF_PID_PROFILE,x0             ;scale axis by this for index
    move    #65533,n0
    move    y:(r0+n0),x1        ;axis->x1
    mpy x0,x1,a                 ;calculate index
    asr a                       ;scale result
    move    a0,n1               ;index->n1
    move    #>SIZEOF_FILTPARAMS,x0              ;sizeof FiltParams is 27
    mpy x0,x1,a
    asr a
    move    a0,n2
    move    #Ffiltvals+GOAL+2,r1        ;address of GOAL
    move    #FFiltParams+POS_P+1,r2 ;address of Position
    move    (r1)+n1
    move    (r2)+n2
```

```
    move    x1,a                ;test which axis
    tst     a
    jeq     <_ZeroCount01
    move    #>2,x1
    cmp     x1,a
    jne     <_ZeroCount03
    move    #>$1000,x1          ;bit pattern for spindle axis
    jmp     <_ZeroCount02
_ZeroCount03
    move    #>$800,x1           ;bit pattern for rotary axis reg clear
    jmp     <_ZeroCount02
_ZeroCount01
    move    #>$20,x1            ;bit pattern for linear axis reg clear
_ZeroCount02
    clr     a                   ;zero accumulator
    ori     #$3,mr              ;DISABLE INTERRUPTS
    ;-----------------------------------
    ; Zero Everyting
    ;-----------------------------------
    move    a,x:(r1)-           ;zero goal
    move    a,x:(r1)-
    move    a,x:(r1)-
    move    (r1)-               ;skip over RelativePosition
    move    a,x:(r1)-           ;zero dgoal
    move    a,x:(r1)-
    move    a,y:(r2)-           ;zero position
    move    a,y:(r2)
    move    x1,y:CLEARIRQ       ;reset hardware registers

    move    (r6)-
    move    y:(r6)-,sr          ;enable interrupts
    move    y:(r6)-,n2
    move    y:(r6)-,r2
    move    y:(r6)-,n1
    move    y:(r6)-,r1
    move    y:(r6)-,x1
    move    y:(r6)-,x0
    move    y:(r6),r0
    rts
;---------------------------------------------------------------------------
;
; Abort a Move
;
;---------------------------------------------------------------------------

    global  FAbortMove
FAbortMove
    move    r0,y:(r6)+
    lua (r6)+,r0
    move    x0,y:(r6)+
    move    x1,y:(r6)+
    move    r1,y:(r6)+
    move    n1,y:(r6)+
    move    r2,y:(r6)+
    move    n2,y:(r6)+
    move    sr,y:(r6)+          ;save status
    ;-----------------------------------
    ; Calculate index from axis
    ;-----------------------------------
    move    #>SIZEOF_PID_PROFILE,x0             ;scale axis by this for index
    move    #65533,n0
    move    y:(r0+n0),x1        ;axis->x1
    mpy x0,x1,a                 ;calculate index
    asr a                       ;scale result
    move    a0,n1               ;index->n1
    move    #>SIZEOF_FILTPARAMS,x0              ;FiltParams size is 27
```

```
    mpy x0,x1,a
    asr a
    move    a0,n2
    move    #Ffiltvals+JERK+1,r1        ;address of jerk
    move    #FFiltParams+FLAGS,r2   ;address of Flags
    move    (r1)+n1
    move    (r2)+n2
    move    #>64,x1
    ori     #$3,mr      ;disable interrupts
    move    x1,y:(r2)   ;Set Motion Status to FLAGS_END
;  ************************************************
;   filtvals[axis].Jerk = 0;
;  ************************************************

    clr a
    move    a1,x:(r1)-  ;Zero Jerk
    move    a0,x:(r1)-
    move    a1,x:(r1)-  ;Zero Accel
    move    a0,x:(r1)-
    move    a1,x:(r1)-  ;Zero Veloc
    move    a0,x:(r1)-


    move    (r6)-
    move    y:(r6)-,sr          ;enable interrupts
    move    y:(r6)-,n2
    move    y:(r6)-,r2
    move    y:(r6)-,n1
    move    y:(r6)-,r1
    move    y:(r6)-,x1
    move    y:(r6)-,x0
    move    y:(r6),r0
    rts

    endsec
```

```
;;!cc
;;
;---------------------------------------
;
; this function reads a memory location from PMEM
;
;---------------------------------------

    section readpmem

    org p:
    global FReadPmem
    global FReadYmem
    global FReadXmem

FReadPmem
    move    r0,y:(r6)+  ;build stack frame
    lua     (r6)+,r0
    move    n0,y:(r6)+  ;save n0 on stack
    move    r1,y:(r6)+
    move    #-3,n0      ;get pointer to string
    move    y:(r0+n0),r1
    move    p:(r1),a    ;read pmem
    move    (r6)-
    move    y:(r6)-,r1
    move    y:(r6)-,n0
    move    y:(r6),r0
    tst     a
    rts

FReadYmem
    move    r0,y:(r6)+  ;build stack frame
    lua     (r6)+,r0
    move    n0,y:(r6)+  ;save n0 on stack
    move    r1,y:(r6)+
    move    #-3,n0      ;get pointer to string
    move    y:(r0+n0),r1
    move    y:(r1),a    ;read pmem
    move    (r6)-
    move    y:(r6)-,r1
    move    y:(r6)-,n0
    move    y:(r6),r0
    tst     a
    rts

FReadXmem
    move    r0,y:(r6)+  ;build stack frame
    lua     (r6)+,r0
    move    n0,y:(r6)+  ;save n0 on stack
    move    r1,y:(r6)+
    move    #-3,n0      ;get pointer to string
    move    y:(r0+n0),r1
    move    x:(r1),a    ;read pmem
    move    (r6)-
    move    y:(r6)-,r1
    move    y:(r6)-,n0
    move    y:(r6),r0
    tst     a
    rts


;*************************************************
; routines to write into memory
;*************************************************

    global  FWriteYMem
```

```
FWriteYMem
    move    r0,y:(r6)+  ;build stack frame
    lua (r6)+,r0
    move    x0,y:(r6)+  ;save registers onto stack
    move    r1,y:(r6)+
    move    #65533,n0
    move    y:(r0+n0),r1    ;load pointer of address to write
    move    #65532,n0
    move    y:(r0+n0),x0    ;load data to write
    move    x0,y:(r1)       ;save data to memory
    move    (r6)-           ;clean up stack
    move    y:(r6)-,r1
    move    y:(r6),x0
    move    y:-(r6),r0
    rts

    global  FWriteXMem
FWriteXMem
    move    r0,y:(r6)+  ;build stack frame
    lua (r6)+,r0
    move    x0,y:(r6)+  ;save registers onto stack
    move    r1,y:(r6)+
    move    #65533,n0
    move    y:(r0+n0),r1    ;load pointer of address to write
    move    #65532,n0
    move    y:(r0+n0),x0    ;load data to write
    move    x0,x:(r1)       ;save data to memory
    move    (r6)-           ;clean up stack
    move    y:(r6)-,r1
    move    y:(r6),x0
    move    y:-(r6),r0
    rts

    global  FWritePMem
FWritePMem
    move    r0,y:(r6)+  ;build stack frame
    lua (r6)+,r0
    move    x0,y:(r6)+  ;save registers onto stack
    move    r1,y:(r6)+
    move    #65533,n0
    move    y:(r0+n0),r1    ;load pointer of address to write
    move    #65532,n0
    move    y:(r0+n0),x0    ;load data to write
    move    x0,p:(r1)       ;save data to memory
    move    (r6)-           ;clean up stack
    move    y:(r6)-,r1
    move    y:(r6),x0
    move    y:-(r6),r0
    rts

    endsec
```

```
IPR =   $ffff       ;interrupt priorty register

RD_RTC  = $ffc2
CLR_RTC = $ffcd
POS_H_REG = $ffc5   ;high address of the position registers
LINEAR  =   1       ;hardware descriptors for position registers
ROTARY  =   0
SPIN    =   2
POS_H_DESC  = $ffc6 ;address of hardware descriptor register

;
;---------------------------------------------------------------------
;
;offsets for filter parameters structure (PID_PARAMS)
;
;---------------------------------------------------------------------
;
SIZEOF_FILTPARAMS   =   37
FLAGS  = 36         ;Control Flags for Motion Profile
JERK   = 34         ;Value of Jerk for Profile
VCOUNT = 33         ;Constant Vel Count
SCOUNT = 32         ;S Curve count Register
PES_SCALE = 31      ;Position Error Signal Scale
PES_LIMIT = 30      ;Position Error Signal Limit
POS_P   =   28      ;position at offset 12, long
AMPLITUDE = 27      ;oscilator amplitude
FOL_LIMIT  = 26     ;maximum following error allowed
INTEGRATE_FLAG = 25 ;Set to zero if command vel != 0, 1 otherwise
INT_LIM =   24      ;integrator limit  ,int
INT_P   =   23      ;integral gain ,int
FFWV_P  =   22      ;feed forward vel gain , int
FFWA_P  =   21      ;feed forward acc gain , int
DIFF_P  =   20      ;differential gain , int
PROP_P  =   19      ;proportional gain , int
OFF_P   =   18      ;offset , int
NOTCHQ1 = 17            ;notch filter Q
NOTCHF1 = 16            ;notch filter Frequency
NOTCHD1 = 15            ;notch filter depth
NOTCHQ0 = 14            ;notch filter Q
NOTCHF0 = 13            ;notch filter Frequency
NOTCHD0 = 12        ;notch filter depth
METERIN1 = 11       ;pointer to meter input
METERFILT1 = 10     ;frequency of meter filter
ZFCOS1 = 8          ;Cosine Filter State Variable
ZFSIN1 = 6          ;Sin Filter State Variable
METERIN0 = 5        ;pointer to meter input
METERFILT0 = 4      ;frequency of meter filter
ZFCOS0 = 2          ;Cosine Filter State Variable
ZFSIN0 = 0          ;Sin Filter State Variable

;
;---------------------------------------------------------------------
;
;offsets into filter values data structure  (PID_PROFILE)
;
;---------------------------------------------------------------------
;
SIZEOF_PID_PROFILE  =   27
PROFCNT =       26      ;Profile State Machine Counter
JERK    =       24      ;storage for jerk register
ACCEL   =       22      ;storage for accel register
VELOC   =       20      ;storage for velocity
GOAL    =       17      ;storage for GOAL from Profile Generator
RELPOS  =       16      ;Relative Position
DGOAL   =       14      ;This is the REAL goal
FOLLOWERROR =   13      ;storage for following error
```

```
ZI_V    =       12      ;storage for integrator temp , int
NOTCH1IN    =   11      ;Response Input for Notch Filter 1
ZBP1 =          9       ;storage for Band Pass State Variable
ZLP1 =          7       ;Storage for Low Pass State Variable
NOTCH0IN    =   5       ;Response Input to Notch Filter 0
ZBP0 =          4       ;storage for Band Pass State Variable
ZLP0 =          2       ;Storage for Low Pass State Variable
PIDOUT =        1       ;Output from PID loop
MOTRIN =        0       ;Input to the motor


    section irq
    org p:
    global  RTCirq
    global  FrtcI


RTCirq
    ;
    ;------------------------------------------------
    ;
    ; push registers onto the STACK
    ;
    ;------------------------------------------------
    ;
    move    (r6)+       ;stupid instruction
    move    x0,y:(r6)+  ;save X
    move    x1,y:(r6)+
    move    y0,y:(r6)+  ;save Y
    move    y1,y:(r6)+
    move    a0,y:(r6)+  ;Save A
    move    a1,y:(r6)+
    move    a2,y:(r6)+
    move    b0,y:(r6)+  ;Save B
    move    b1,y:(r6)+
    move    b2,y:(r6)+
    move    r1,y:(r6)+  ;save r1
    move    n1,y:(r6)+  ;save n1
    move    r2,y:(r6)+  ;save r2
    move    r3,y:(r6)+  ;save r3
    move    n3,y:(r6)+  ;save n3
    move    r4,y:(r6)+  ;save r4
    move    n4,y:(r6)+  ;same n4
    move    r5,y:(r6)+  ;save r5
    move    n5,y:(r6)+  ;save n5
    ;
    ;
    ;------------------------------------------------
    ;
    ; load up registers and call PID loop
    ;
    ; r1- points to X Memory image of Profile Data
    ; r5- points to Y Memory image of Profile Parameters
    ; r2- points to hardware registers
    ; r3- points to Y Memory image of Axis Status
    ;
    ;------------------------------------------------
    ;----------------------------------------
    ; calling sequence to Pid Axis 2 (rotary)
    ;
    ; r1--profile structure pointer
    ; r2--hardware register pointers
    ; r3--misc parameters
    ; r5--filter parameters pointer
    ;----------------------------------------
    move    #Ffiltvals+PROFCNT+SIZEOF_PID_PROFILE,r1
```

```
    move    #FFiltParams+FLAGS+SIZEOF_FILTPARAMS,r5   ;r5 points to top of PidParams
    move    #POS_H_REG,r2           ;r2 points to position reg axis 2
    move    #>ROTARY,x0
    move    x0,y:POS_H_DESC         ;set hardware descriptor for rotary
    move    #FMiscParams+32,r3      ;r3 points to bottom of MiscParams
    jsr     FPid
    ;
    ;-----------------------------------------
    ; Decrement Sample Rate Counter
    ;-----------------------------------------
    ;
    move    y:RD_RTC,a  ;Read Sample Rate Counter Status
    ;
    ;-----------------------------------------
    ; calling sequence to Pid Axis 1  (linear)
    ;-----------------------------------------
;   move    #FFiltParams+POS_P+1,r5 ;r5 points to top of position
;   move    #POS_H_REG,r2           ;r2 points to position reg axis 1
;   move    #>LINEAR,x0
;   move    x0,y:POS_H_DESC         ;set hardware descriptor for Linear
;   jsset   #22,a,Pos           ;jump to subroutine if bit set
    ;
    ;-----------------------------------------
    ; calling sequence to PID asix 3  (spindle)
    ;-----------------------------------------
    move    #Ffiltvals+PROFCNT+SIZEOF_PID_PROFILE+SIZEOF_PID_PROFILE,r1      ;r1 points to top o
f filtvals
    move    #FFiltParams+FLAGS+SIZEOF_FILTPARAMS+SIZEOF_FILTPARAMS,r5   ;r5 points to top of Pid
Partams
    move    #POS_H_REG,r2           ;r2 points to position reg axis 2
    move    #>SPIN,x0
    move    x0,y:POS_H_DESC         ;set hardware descriptor for rotary
    move    #FMiscParams+64,r3      ;r3 points to bottom of MiscParams
    jsset   #23,a,FVID              ;jump to subroutine if bit set
    ;-------------------------------------------------
    ; clear rtc interrupt
    ;-------------------------------------------------
    ;
    move    a,y:CLR_RTC
    ;
    ;-------------------------------------------------
    ;
    ;pop all saved registers from stack
    ;
    ;-------------------------------------------------
    ;
    move    (r6)-
    move    y:(r6)-,n5  ;restore n5
    move    y:(r6)-,r5  ;restore r5
    move    y:(r6)-,n4  ;restore n4
    move    y:(r6)-,r4  ;restore r4
    move    y:(r6)-,n3  ;restore n3
    move    y:(r6)-,r3  ;restore r3
    move    y:(r6)-,r2  ;restore r2
    move    y:(r6)-,n1  ;restore n1
    move    y:(r6)-,r1  ;restore r1
    move    y:(r6)-,b2  ;restore B
    move    y:(r6)-,b1
    move    y:(r6)-,b0
    move    y:(r6)-,a2  ;restore A
    move    y:(r6)-,a1
    move    y:(r6)-,a0
    move    y:(r6)-,y1  ;restore Y
    move    y:(r6)-,y0
    move    y:(r6)-,x1  ;restore X
    move    y:(r6)-,x0
```

```
    rti
;
; This is the routine to handle the real time clock
;

    endsec
```

```
;
;*******************************************************
;
; These routines talk to the various registers
; In the Gate Array
;
; This does NOT do what one would think anylonger
;
;*******************************************************
;
CLEAR   = $ffcc
PCDDR   = $ffe3
PCD     = $ffe5

    section ssicode
    org p:
    global FClearPGA
    global Fabort

FClearPGA
;
;programable hardware reset
;
    move    a1,y:CLEAR
;
; setup SSI port for SYNC pulses
; Bit 5 is TP12
; Bit 6 is TP31
;
    bset    #5,x:PCDDR  ;make output ports
    bset    #6,x:PCDDR

    rts

    global FSetTP12

FSetTP12
    bset    #5,x:PCD
    rts

    global FClearTP12

FClearTP12
    bclr    #5,x:PCD
    rts

    global FSetTP31

FSetTP31
    bset    #6,x:PCD
    rts

    global FClearTP31

FClearTP31
    bclr    #6,x:PCD
    rts

Fabort
_loop   jmp _loop   ;infinate loop (this is a CRASH condition)

    endsec
```

```
;;!cc
;---------------------------------------------------------------------
;
; functions that interface with the xicor 2402 nv ram chip
;
;---------------------------------------------------------------------

PCD =   $ffe5
PCDDR = $ffe3
PCC =   $ffe1
HSR =   $ffe9

DATA = 4        ;data is on SC0
CLOCK = 3       ;clock is on SC1
AMPEN1 = 7      ;amplifier enable axis 1
AMPEN2 = 8      ;amplifier enable axis 2
SYNC1 = 5       ;sync output axis 1
SYNC2 = 6       ;sync output axis 2
ADDRESS = -3    ;defines for parameters on stack
CHIP = -4
DATAWORD = -3

    section XICOR_
    org p:
    global FInitXICOR   ;routine to initialize the port for XICOR

FInitXICOR
    bset    #CLOCK,x:PCDDR  ;set clock port to output
    bset    #DATA,x:PCDDR   ;set data port to output
    bset    #CLOCK,x:PCD    ;clear clock bit
    bset    #DATA,x:PCD     ;clear data bit
    bset    #AMPEN1,x:PCDDR ;set amp enable to output axis 1
    bset    #AMPEN2,x:PCDDR ;set amp enable to output axis 2
    bset    #SYNC1,x:PCDDR  ;set sync to output axis 1
    bset    #SYNC2,x:PCDDR  ;set sync to output axis 2
    rts


;---------------------------------------------------------------------
;
; amplifier enables
;
;---------------------------------------------------------------------

    global  FDisable_Amp
FDisable_Amp
    move    r0,y:(r6)+
    lua (r6)+,r0
    move    b0,y:(r6)+
    move    b1,y:(r6)+
    move    x0,y:(r6)+
    move    r1,y:(r6)+
; ************************************************
;
; void Disable_Amp(int axis)
; {
;   *(int *)0xffe5 &= ~(1 << (7 + axis) );
; ************************************************
    move    #65509,r1
    move    #>7,a
    move    #65533,n0
    move    y:(r0+n0),b
    add a,b
    move    #>1,a
    tst b
    jeq L2
```

```
    rep b
    asl a
    move    a1,a
L2
    not a
    move    a1,a
    move    x:(r1),x0
    and x0,a
    move    a1,a
    move    a1,x:(r1)
; ************************************************
; }
; ************************************************
;
    move    (r6)-
    move    y:(r6)-,r1
    move    y:(r6)-,x0
    move    y:(r6)-,b
    move    y:(r6),b0
    move    y:-(r6),r0
    rts

    global  FEnable_Amp
FEnable_Amp
    move    r0,y:(r6)+
    lua (r6)+,r0
    move    b0,y:(r6)+
    move    b1,y:(r6)+
    move    x0,y:(r6)+
    move    r1,y:(r6)+
; ************************************************
; ************************************************
; ************************************************
;
; void Enable_Amp(int axis)
; {
;   *(int *)0xffe5 |= 1 << (7 + axis);
; ************************************************
;
    move    #65509,r1
    move    #>7,a
    move    #65533,n0
    move    y:(r0+n0),b
    add a,b
    move    #>1,a
    tst b
    jeq L4
    rep b
    asl a
    move    a1,a
L4
    move    x:(r1),x0
    or  x0,a
    move    a1,a
    move    a1,x:(r1)
; ************************************************
; }
; ************************************************
;
    move    (r6)-
    move    y:(r6)-,r1
    move    y:(r6)-,x0
    move    y:(r6)-,b
    move    y:(r6),b0
    move    y:-(r6),r0
    rts

    endsec
```

```
/*
**
** Header file for Event.c
*/

#ifndef EVENT__H
#define EVENT__H

#include "squeues.h"

#define MAX_EVENTS  12      /*  maximum number of diffent events    */

#define EVENT_IN_PROGRESS   0x01

/*
** Definitions of the various events
*/

#define MOVE_SERVO      1   /*  this event moves the servo to position  */
#define DO_JOG          2   /*  this event jogs the motor back and forth    */
#define DODEBUG         3   /*  do a DEBUG event    */
#define DO_HOME_EVENT   4   /*  trying to find home position            */
#define SAVE_PARAMS     5
#define RESTORE_PARAMS  6
#define EVENT_ARMED     7   /*  indicates system waiting for a trigger  */
#define EVENT_CLEARBUMP 8   /*  clears bump error in event processor    */
#define EVENT_END_JOG   9   /*  end a jogging event */
#define EVENT_PROFILE   10  /*  do profile planning */
#define EVENT_SETGOALANDMOVE    11  /*  set goal and do absolute move   */

/*
** "system" status defines, system info
*/

#define DSPSTAT_POSTDITHER  0x2         /*  Post Dither is Enabled  */
#define BEAM_INTERRUPTED    0x4         /*  beam interrupted flag   */
#define INTEGRATOR_ENABLED  0x8         /*  integrator flag         */
#define SERVO_ENABLED       0x10        /*  servo enabled flag  */
#define LIMIT_PLUS_TRIPPED  0x20        /*  plus limit has been tripped */
#define LIMIT_MINUS_TRIPPED 0x40        /*  minus limit has been tripped    */
#define HOME_TRIPPED        0x80        /*  home sensor has been tripped    */
#define AMP_FAULT_TRIPPED   0x100       /*  amplifier fault has been tripped    */
#define TRIGGER_TRIPPED     0x200       /*  trigger has been tripped    */
#define PROTECTION_TRIPPED  0x400       /*  protection line has been tripped    */
#define CTS_TRIPPED         0x800       /*  CTS line tripped    */
#define FOLLOWING_ERROR     0x1000      /*  Fatal Following Error   */
#define F_ERROR_ENABLED     0x2000      /*  Fatal Following Error Enabled   */
#define GENERAL_FAULT       0x4000      /*  Some general fault has occured  */
#define ENABLE_TRIGGER      0x8000      /*  Indicates profile generator will wait */
                                        /*  for a trigger before generating move    */
#define SERVO_IN_POSITION   0x10000     /*  indicates following error less than preset value
*/
#define DIS_INT_CMDVNEZ     0x20000     /*  disable integrator durring moves    */
#define BUMP_ERROR          0x40000     /*  bump Error Has Occured          */
#define SYS_DITHER_ENABLED  0x80000     /*  dither is enabled               */
#define MOVE_IDLE           0x100000    /*  indicates motion profile is idle    */
#define DSPSTAT_NOTCHFILTER 0x200000    /*  indicates notch filter enabled  */
#define DSPSTAT_PES         0x400000    /*  indicates PES is enabled    */
#define DSPSTAT_NOTCHFILT2  0x800000    /*  indicates notch filter two is enabled   */

typedef struct {
    int status;     /*  event in progress status    */
    int type;       /*  current event being executed    */
    int pending;    /*  events pending execution    */
    int system[3];      /*  contains system info        */
} STATUS_CLASS;
```

```
/* external globals */

extern QUEUE *message;
extern volatile STATUS_CLASS event_status;
extern volatile int HistFlags[];

/* function prototypes  */

extern void init_event(void);
extern void event_loop(void);

extern void SetTP12(void);
extern void SetTP31(void);
extern void ClearTP12(void);
extern void ClearTP31(void);

#endif
```

```
/*
**
** Header file for Command.c
*/

#ifndef COMMAND__H
#define COMMAND__H

#include "sprofile.h"

/*
** defines for Commands
** commands that do not send are from 0 -> 255
** commands that send data back are from 256 -> 511
*/

    /*
    ** recieve commands
    */

#define SET_KP                  0   /*  set proportional gain   */
#define SET_KV                  1   /*  set differential gain   */
#define SET_KI                  2   /*  set integral gain       */
#define SET_KVFF                3   /*  set feed forward velocity gain  */
#define SET_KAFF                4   /*  set feed forward acceleration gain  */
#define SET_OFFSET              5   /*  set servo offset    */
#define SET_GOAL                6   /*  set desired position    */
#define SET_MAXVEL              7   /*  set maximum velocity    */
#define SET_SCURVE              8   /*  set time to get up to speed */
#define DO_MOVE                 9   /*  execute profile generator and do move   */
#define MOVE_ABORT              10  /*  abort current move  */
#define SET_PHASE_DETECT_POL    11  /*  set the polarity of the phase detector  */
#define SET_SAMPLE_RATE         12  /*  Set servo sample rate       */
#define ZERO_INTEGRATOR         13  /*  reset the integrator accumulator    */
#define SET_INT_MODE            14  /*  set lower DAC limit     */
#define START_JOG               15  /*  set up and start a jog for tuning purposes  */
#define END_JOG                 16  /*  cancel the jog  */
#define SET_JOG_PARAM           17  /*  set joging parameter        */
#define CLEAR_INTERRUPT         18  /*  clears the beam interrupt flag and restarts the servo
*/
#define ENABLE_SERVO_FUNCTIONS  19  /*  enable/disable the servo loop   */
#define SET_INT_LIMITS          20  /*  set intergrator limits  */
#define ZERO_COUNT              21  /*  zero the position counter   */
#define SET_MAX_FOLLOW          22  /*  set the maximum following error */
#define SET_FOL_GAIN            23  /*  set the following error DAC gain    */
#define HOME_SETUP              24  /*  load parameters for home command    */
#define DO_HOME                 25  /*  go to "home" position               */
#define RESTORE_SETTINGS        26  /*  restore settings from NV rom    */
#define SAVE_SETTINGS           27  /*  save settings into NV rom       */
#define SET_INTERRUPTS          28  /*  enable/disable interrupts       */
#define TRIGGER_SERVO           29  /*  set trigger flag true   */
#define SET_INPOS_THRESH        30  /*  set in position threshold   */
#define SET_BUMP                31  /*  set the bump threshold  */
#define CLEAR_FLAG              32  /*  clear step flag */
#define RECORD_LINPOS           33  /*  record current linear position  */
#define SET_FREQUENCY           34  /*  set frequency of internal sin oscilator */
#define SET_FILTERFREQ          35  /*  set the cutoff freq of lp filter    */
#define SET_AMPLITUDE           36  /*  set the amplitude of oscilator  */
#define SET_PHASE               37  /*  set phase of cosine output  */
#define SET_DITHERMODE          38  /*  set mode of dither  */
#define SET_ENCODERPITCH        39  /*  set the encoder pitch   */
#define SET_RUNOUTLUT           40  /*  select default/user lut */
#define SET_HANDSHAKE           41  /*  set handshake line for seagate thingy   */
#define START_HISTOGRAM         42  /*  start histogram aquisition  */
#define SET_NOTCH_FREQ          43  /*  set frequency of notch filter   */
#define SET_NOTCH_Q             44  /*  set notch filter Q  */
```

```
#define SET_NOTCH_D             45  /*  set notch filter depth  */
#define SET_ANAL                46  /*  set analysis input mode */
#define SET_PESSCALE            47  /*  set scale for PES   */
#define SET_PESLIMIT            48  /*  set limit for PES   */
#define RESET_PES               49  /*  reset PES counters  */
#define SET_PESMODE             50  /*  set mode of PES counter */
#define WRITE_MEM               51  /*  write data word to memory   */
#define SETGOALANDMOVE          52  /*  set goal and do move    */

/*
** send commands
*/

#define READ_POSITION           256 /*  read current position   */
#define READ_GOAL               257 /*  read back the desired goal  */
#define READ_SCURVE_TIME        258 /*  read back SCURVE time   */
#define READ_MAX_VEL            259 /*  read back maximum veloctiy  */
#define READ_KP                 260 /*  read back proportional gain     */
#define READ_KV                 261 /*  read back differential gain     */
#define READ_KI                 262 /*  read back integral gain         */
#define READ_KVFF               263 /*  read back feed forward velocity */
#define READ_KAFF               264 /*  read back feed forward acceleration */
#define READ_OFFSET             265 /*  read back servo offset          */
#define STATUS                  266 /*  read back servo card status     */
#define READ_PHASE_DETECT_POL   267 /*  read the phase detector polarity    */
#define GET_SAMPLE_RATE         268 /*  read the servo sample rate  */
#define GET_INT_MODE            269 /*  get lower DAC limit     */
#define READ_DAC_LEV            270 /*  reads the level of the Motor DAC    */
#define GET_INT_LIMITS          271 /*  read the integrator limits  */
#define GET_MAX_FOLLOW          272 /*  read the maximum following error    */
#define GET_FOL_GAIN            273 /*  get the following error DAC gain    */
#define GET_SN                  274 /*  get serial number and revision      */
#define GET_HOME_PARAMS         275 /*  get home parrameters    */
#define GET_INTERRUPTS          276 /*  get interrupt settings  */
#define READ_PMEM               277 /*  FOR debug purposes      */
#define READ_YMEM               278 /*  for debug purposes      */
#define READ_XMEM               279 /*  for debug purposes      */
#define READ_CALDIST            280 /*  read calibration distance   */
#define READ_JOG_PARAMS         281 /*  read jogging parameters     */
#define READ_INPOSTHRESH        282 /*  read in position threshold  */
#define GET_BUMP                283 /*  read the bump threshold */
#define GET_CONFIG              284 /*  dump config parameters  */
#define GET_LINEAR_ENDPOINT     285 /*  get recorded endpoint   */
#define GET_FREQUENCY           286 /*  get frequency of internal oscilator */
#define GET_MAGNITUDE           287 /*  get magnitude of position diviation */
#define GET_FILTFREQ            288 /*  get the filter cutoff frequency */
#define GET_AMPLITUDE           289 /*  get the amplitude of sine oscilator */
#define GET_PHASE               290 /*  get phase of cosine output  */
#define GET_DITHERMODE          291 /*  get mode of dither oscilator    */
#define GET_ENCODERPITCH        292 /*  get the encoder pitch       */
#define GET_RUNOUTCOMP          293 /*  get runout compensation table   */
#define GET_RUNOUTLUT           294 /*  get runout comp table selection */
#define GET_HISTOGRAM           295 /*  get histogram data block    */
#define GET_HISTOGRAM           295 /*  get histogram data  */
#define GET_NOTCH_FREQ          296 /*  get notch filter frequency  */
#define GET_NOTCH_Q             297 /*  get notch filter Q  */
#define GET_NOTCH_D             298 /*  get notch filter depth  */
#define GET_PESSCALE            299 /*  get PES scale   */
#define GET_PESLIMIT            300 /*  get PES limit   */
#define GET_FPGAID              301 /*  get FPGA gate array */
#define GET_OTHER_CONFIG        302 /*  get jog and home params */
#define GET_DATA_STRUCT         303 /*  get data structure address and size */
#define GET_MAGNITUDE1          304 /*  get magnitude of spetrum filter */

/*
** Get Data Buffer Commands
```

```
*/

#define SET_CONFIG              512 /*  recieve parameter dump  */
#define SET_RUNOUTCOMP          513 /*  recieve runout compensation table   */

/*
** Commands Sent Back to Host
*/

#define DSP_COMMANDACK      0       /*indicates that a command ACK command sent            */
#define DSP_EXCEPTION       1       /*indicates that some problem needs to be taken care of */
#define DSP_MOVECOMPLETE    2       /*indicates that a move has been completed      */
#define DSP_DEBUGSTRING     3       /*indicates that DSP is sending debug text to host  */

/*  sub commands for DSP_EXCEPTION  */

#define DSPEXCEP_BEAMINTERRUPTED    0   /*indicates beam interrupted error          */
#define DSPEXCEP_AMPLIFIERFAULT     1   /*indicates amplifier faulted               */
#define DSPEXCEP_LIMITHIT           2   /*indicates that a limit was hit            */
#define DSPEXCEP_PROTECTFAULT       3   /*indicates that a protection fault occured */
#define DSPEXCEP_AMPREQ             4   /*external request for amplier  */
#define DSPEXCEP_HISTOGRAM          5   /*histogram is done exception   */

/* Aux defines for DSPEXCEP_AMPREQ  */

#define DSPEXAUX_REQUESTAMP         0   /*  request use of amp  */
#define DSPEXAUX_RELEASEAMP         1   /*  release use of amp  */

/* Aux defines for DSPEXCEP_LIMITHIT    */

#define DSPEXAUX_POSITIVELIMIT      0   /*  indicates positive limit tripped    */
#define DSPEXAUX_NEGATIVELIMIT      1   /*  indicates negative limit tripped    */

/* sub commands for DSP_MOVECOMPLETE        */

#define DSPMOVE_MOVECOMPLETE        0   /*indicates move finished       */
#define DSPMOVE_HOMECOMPLETE        1   /*indicates home finished       */
#define DSPMOVE_CALIBRATECOMPLETE   2   /*indicates calibrate finished  */
#define DSPMOVE_LINEARHOME          3   /*  indicates linear is at home */
#define DSPMOVE_LINEARRUN           4   /*  indicates linear is at run  */

/*
** jogging defines
*/

#define JOG_DIRECTION           0x01    /*  jog in clock wise direction */
#define JOG_MODE_NORETRACE      0x02    /*  if set, does not retrace quickly, just reverses */

#define JOG_START_L             0       /*  start position of jog   */
#define JOG_START_H             1
#define JOG_END_L               2       /*  end positon of jog      */
#define JOG_END_H               3
#define JOG_INDEX               4       /*  index is function 0 */
#define JOG_MODE                5       /*  mode is function 1  */
#define JOG_TIME                6       /*  dwell time is function 2    */

/*
**  Clear Interrupt Function Values
*/

#define CLEAR_PROTECT   0   /*  clear progection interrupt  */
#define CLEAR_LIMIT_M   1   /*  clear LIMIT Minus       */
#define CLEAR_BEAM      2   /*  clear BEAM interrupt    */
#define CLEAR_LIMIT_P   3   /*  clear LIMIT Plus        */
#define CLEAR_AMP_FAULT 4   /*  clear amplifier fault   */
#define CLEAR_FOLLOW    5   /*  clear Fatal Follow Error    */
```

```
#define CLEAR_GENERAL   6   /*  clear general servo fault   */
#define CLEAR_BUMP      7   /*  clear the bump indicator    */

#define CLEAR_MAX       CLEAR_BUMP

/*
**  Interrupt Vectors
*/

#define IVECT_PROTECT   0   /*  clear progection interrupt  */
#define IVECT_LIMIT_M   1   /*  clear LIMIT Minus       */
#define IVECT_BEAM      2   /*  clear BEAM interrupt    */
#define IVECT_LIMIT_P   3   /*  clear LIMIT Plus        */
#define IVECT_AMP_FAULT 4   /*  clear amplifier fault   */
#define IVECT_HOME      5   /*  clear HOME interrupt    */
#define IVECT_TRIGGER   6   /*  clear TRIGGER interrupt */
#define IVECT_CTS       7   /*  clear CTS interrupt     */


/*
** hardware interrupt bit defines
*/
#define HDWIRQ_ENABLE   0x01    /*  Interrupt Enabled   */
#define HDWIRQ_DISPID   0x02    /*  Enable PID disable on interrupt */
#define HDWIRQ_IRQSTAT  0x04    /*  Interrupt Status Read Only      */
#define HDWIRQ_ABORT    0x08    /*  causes move to abort on interrupt   */
#define HDWIRQ_MOVE     0x10    /*  causes move complete when turned on */
                                /*  disables interrupt exception message    */

#define HDWIRQ_SET  0x08    /*  or this with vector number if bits are to be set    */

/*
** Servo Functions
*/
#define ENABLE_PID          0   /*  enable the PID loop     */
#define ENABLE_INTEGRATOR   1   /*  enable the PID integrator   */
#define ENABLE_FATAL_FOLLOW 2   /*  enable the fatal following error    */
#define ENABLE_MOVE_TRIGGER 3   /*  move will start on a trigger    */
#define ENABLE_DITHER       4   /*  enable sinusoid oscilator       */
#define ENABLE_NOTCH        5   /*  enable notch filter     */
#define ENABLE_PES          6   /*  enable servo on track mode  */
#define ENABLE_NOTCH2       7   /*  enable notch filter #2  */
#define ENABLE_POSTDITHER   8   /*  enable dither signal after PID  */

/*
** home command parrameters
*/
    /*  limit switches  */
#define HOME_NONE           0   /*  there is no limit defined   */
#define HOME_LIMIT_PLUS     1   /*  limit on the plus limit switch  */
#define HOME_LIMIT_MINUS    2   /*  limit on the minus limit switch */
#define HOME_LIMIT_HOME     3   /*  limit on the home limit switch  */
#define HOME_DAC            4   /*  limit on Dac Drive      */

#define MOTOR_CURRENT_MASK  0x0ff0
#define MOTOR_CURRENT_SHIFT 4   /*  shift number 8 bits before oring    */

#define HOME_POSITIVE       0x1000  /*  home to the east    */

    /*  home commands   */
#define HOME_DOHOME     0       /*  performs home command   */
#define HOME_DOCAL      1       /*  performs callibrate command */

/*
** move functions
*/
```

```
#define MOVE_ABSOLUTE   0
#define MOVE_RELATIVE   1

/*
** Record Linear Position Functions
*/

#define LINEARSTAGE_RECORDHOME      0
#define LINEARSTAGE_RECORDRUN       1
#define LINEARSTAGE_INPOSENABLE     2

/*
** Read Amplitude Functions
*/

#define READAMPLITUDE_COSINE        0
#define READAMPLITUDE_SINE          1

/*
** function codes for write mem
*/

#define DSP_PMEM    0
#define DSP_XMEM    1
#define DSP_YMEM    2

/*
** functions for Send Data Structures
*/

#define DSP_SEND_PID_PARAMS     0
#define DSP_SEND_PID_STATUS     1
#define DSP_SEND_PID_PROFILE    2
#define DSP_SEND_JOG_PARAMS     3
#define DSP_SEND_HOME_PARAMS    4


/*
** functions for SET_ANALysys mode
*/

#define DSP_ANAL_CLOSEDLOOP     0
#define DSP_ANAL_OPENLOOP       1
#define DSP_ANAL_NOTCH1         2
#define DSP_ANAL_NOTCH0         3
#define DSP_ANAL_PID            4

/*
** External data definitions
*/

extern long DesiredGoal[3];
extern int volatile Global_Trigger;
extern int IrqStatMasks[];
extern int IrqClearMasks[];
extern int IrqLevelMasks[];
extern TEL_PROFILE MoveProfiles[3];

/*
** index parameter defs
*/
#define DWELL_TIME  0
#define INDEX_SIZE  1
```

```
/*
** function prototype, although, this function is only called by an
** interrupt
*/


extern void Command(void);
extern void Enable_Amp(int axis);
extern void Disable_Amp(int axis);
extern void DspException(int subcommand,int aux);
extern void DspMoveComplete(int subcommand,int aux);

extern int ReadPmem(int adr);
extern int ReadYmem(int adr);
extern int ReadXmem(int adr);
extern void WriteXMem(int *adr,int v);
extern void WriteYMem(int *adr,int v);
extern void WritePMem(int *adr,int v);

#endif
```

```
/*****************************
** Home.h  Header file for   **
** home command              **
*****************************/

#ifndef HOME__H
#define HOME__H


typedef struct {
    int Flags;              /*  misc home parameters    */
    int InitVelocity;       /*  initial seek velocity towards limit */
    int InitDwell;          /*  dwell time to wait before backing out   */
    int Backout;            /*  Backout distance to move from stop  */
    int FineDwell;          /*  time to Wait before starting fine seek  */
    int FineVelocity;       /*  fine seek velocity  */
    long HomingDistance;    /*  distance to move from stop for home */
}HOME_PARAMS;


#define HOME_FLAGS      0       /*  misc home parameters    */
#define HOME_INITVEL    1       /*  initial seek velocity towards limit */
#define HOME_INITDWELL  2       /*  time to wait before backing out */
#define HOME_BACKOUT    3       /*  distance to back away from stop */
#define HOME_FINEDWELL  4       /*  time to wait before starting fine seek  */
#define HOME_FINEVEL    5       /*  fine seek velocity  */
#define HOME_ZERDIST    6       /*  zero distance from home */

/* global variables     */
extern volatile HOME_PARAMS HomeParams[3];  /*  place where home parameters live    */
/* function prototypes  */
extern void Home(int axis,int cmd);
#endif
```

```
/*
** header file for HOST interface routines
*/
#ifndef HOST__H
#define HOST__H

extern void HInit(void);         /*  initializes the host interface  */
extern int HIGet(void);          /*  get a character from host interface */
extern void HIPut(int);          /*  put a character into host interface */
extern int HIiStatus(void);      /*  Returns number of characters waiting    */
                                 /*  in input buffer */
extern int HIoStatus(void);      /*  returns number of characters in     */
                                 /*  transmit buffer             */
extern void HIcmdClear(void);    /*  tell host to clear pending commnad  */
extern void HIcmdEnable(void);   /*  enable host command interrupt   */
extern void HIcmdDisAble(void);  /*  disable host command interrupt  */
extern void HIFSet(int);     /*  set bits in host port control reg   */
extern void HIFClr(int);     /*  clear bits in host port control reg */
extern void Enable(int);         /*  enable interrupts   */
extern int Disable(void);        /*  disable interrupts  */

#endif
```

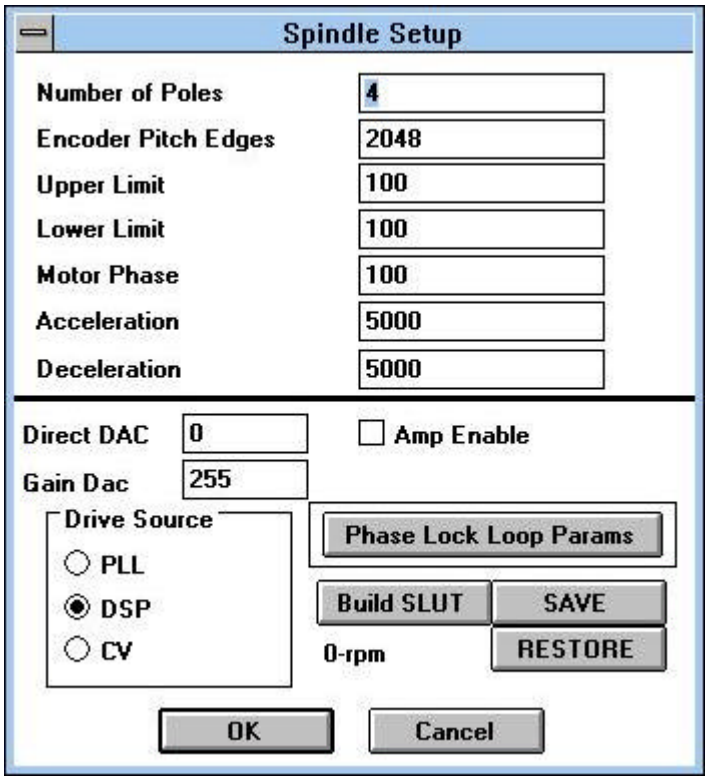
```
/*********************************************************************
**
** Header File for profile.cpp
**
*********************************************************************/

#ifndef PROFILE__H
#define PROFILE__H


typedef struct {
    long Jerk;      /*  Jerk required to do move                      */
    int SSteps;    /*   number of steps needed for S Curve            */
    int VSteps; /*  number of steps needed for Constant Velocity    */
}TEL_PROFILE;

/*
** values returned by TelProfile
*/

#define TELPROFILE_OK       0   /*  no Error    */
#define TELPROFILE_SCURVE   1   /*  S Curve not a Multiple of 2 */


/*---------------------------------------
** function prototypes
**---------------------------------------*/

#ifdef __cplusplus
extern "C" {
#endif

extern int TelProfile(int SCurveTime, long Distance, int MaxVel, TEL_PROFILE *Profile);
extern double TelPeekVelocity(int SCurveTime, long Distance, int MaxVel);
extern double TelProfileTotalTime(int SCurveTime, long Distance, int MaxVel);
extern void MoveRel(int axis,long goal);
extern void FloatTo72(double f, int *w72);

#ifdef __cplusplus
}
#endif

#endif
```

```c
/*
** this is a header file for the routines to do a bunch of queues
*/

#ifndef QUEUES__H
#define QUEUES__H

typedef struct queue {
    int size;       /*  total size of queue */
    unsigned head;  /*  head pointer    */
    unsigned tail;  /*  tail pointer    */
    int num_elem;   /*  number of elements is queue */
    char data[16];   /* this is where the data goes, must be last   */
    }QUEUE;

#define BUFFER_FULL 1
#define BUFFER_EMPTY    -1
#define OK          0

/*
** global accessable routines
*/

extern QUEUE *make_queue(int size); /*  constructor */
extern int put_queue(QUEUE *q,int val); /*  put data into queue */
extern int get_queue(QUEUE *q);         /*  get data from queue */
extern int status_queue(QUEUE *q);          /*  how many things in que  */
extern void free_queue(QUEUE *q);


#endif
```

```
/*************************************************************************
**
** Header file for DSP hardware defines
**
** Gate Array Addresses
**
*************************************************************************/

#ifndef SERVOHARDWARE__H
#define SERVOHARDWARE__H

/*-----------------------------------------------------------------------
    Defines for various hardware registers in gate array
_____________________________________________________________________*/

#define IRQEN_REG       (*((volatile int *)0xffc0)) /*  write to IRQ enable register    */
#define RDIRQSTATUS_REG (*((volatile int *)0xffc0)) /*  read IRQ status flip flops  */
#define CLEARIRQ_REG    (*((volatile int *)0xffc1)) /*  clear IRQ status flip flop(s)   */
#define RDIRQLEVEL_REG  (*((volatile int *)0xffc1)) /*  read level of input to IRQ flipflops
*/
#define RTC_REG         (*((volatile int *)0xffc2)) /*  read/write to real time clock control re
g   */
#define DAC_REG         (*((volatile int *)0xffc3)) /*  read/write dac motor drive  */
#define POS_L_REG       (*((volatile int *)0xffc4)) /*  read/write position LOW word    */
#define POS_H_REG       (*((volatile int *)0xffc5)) /*  read/write position HIGH word   */
#define AXIS_DESC_REG   (*((volatile int *)0xffc6)) /*  points to axis selector register    */
#define FPGA_ID         (*((volatile int *)0xffc7)) /*  read FPGA id code   */
#define WD_PESCR        (*((volatile int *)0xffc9)) /*  write control reg for PES   */
#define CLR_PES         (*((volatile int *)0xffcb)) /*  clear PES counter   */
#define RD_PESREG       (*((volatile int *)0xffcb)) /*  read pes register   */
#define RDCOS_REG       (*((volatile int *)0xffcc)) /*  read cosine oscilator output    */
#define RDSIN_REG       (*((volatile int *)0xffcd)) /*  read sin oscilator output   */
#define FREQUENCY_REG   (*((volatile int *)0xffce)) /*  read/write frequency control reg    */
#define FOLLOWERROR_REG (*((volatile int *)0xffcf)) /*  read/write following error output   */
#define WDPHASE_REG     (*((volatile int *)0xffcc)) /*  write to phase control reg for cosine ou
tput    */
#define WDCLEARRTC      (*((volatile int *)0xffcd)) /*  clear RTC interrupt */

#define DAC_REG_P   (((volatile int *)0xffc3))  /*  read/write dac motor drive  */
#define FOLLOWERROR_REG_P (((volatile int *)0xffcf))    /*  pointer to following error register
*/
#define FPGA_ID_P   (((volatile int *)0xffc7))  /*  pointer to FPGA ID  */

/* descriptor values for axis descriptor register   */

#define DREG_LINEAR     1
#define DREG_ROTARY     0
#define DREG_SPINDLE    2

#endif
```

```c
/*
** header file for real time clock
*/

#ifndef RTC_H
#define RTC_H

#define RATE_BITS   0x7f    /*  mask for rtc rate   */

#define RT_COUNT    (*((volatile int *)0xffc8))

extern volatile int RtcCount;

extern int RTC_status(void);
extern void RTChandle(void);
extern void RTCInit(void);
extern void RTC_Set_Polarity(int axis,int pol);
extern int RTC_Get_Polarity(int axis);



#endif
```

**Procedure for Adjusting Motor Phasing on Acutrac ][**

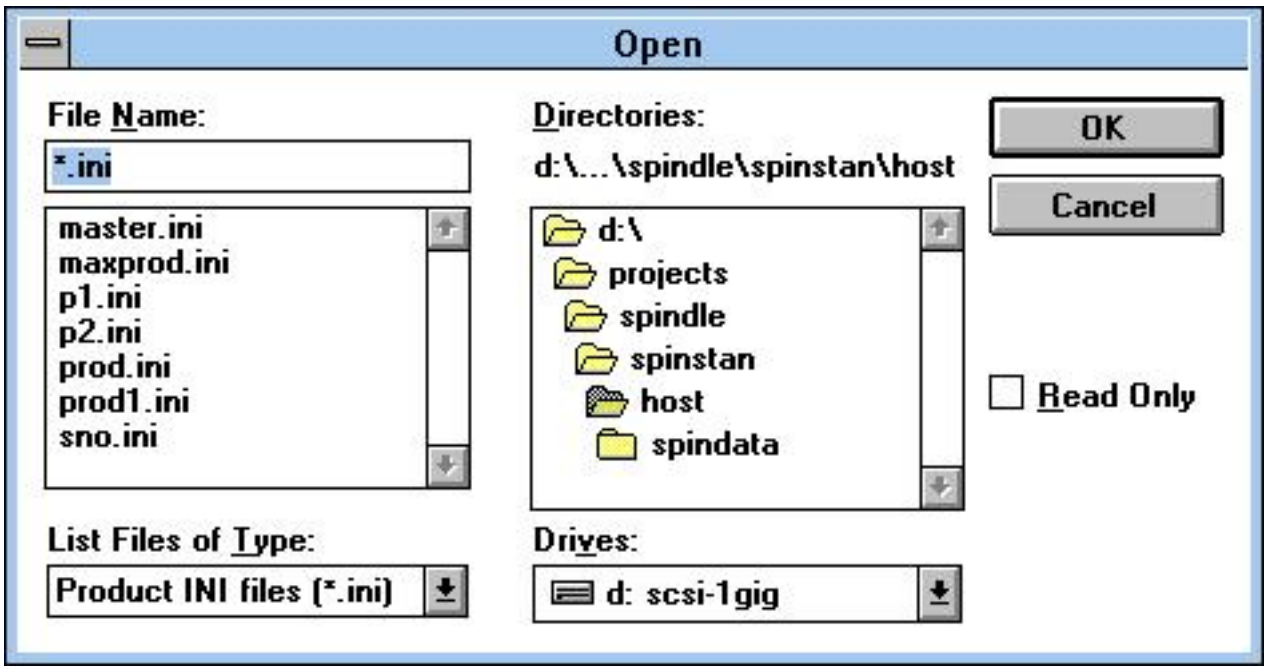



1. **The above dialog box can be reached by pulling down the <u>spindle</u> menu and selecting the <u>spindle setup</u> menu item.**
2. If the Gain Dac edit box has not been set to 255, then do so. In order for this entry to take affect, the Enter key must be pressed.
3. If the Direct DAC edit box is not set to 8, then do so. In order for this entry to take affect, the Enter key must be pressed.
4. In the Drive Source box, select the CV radio button.
5. Make sure that there are no obstructions around the spindle motor. If there are any disks on the spindle, make sure they are tight. The spindle could be spinning at some very high speeds.
6. Select the Amp Enable checkbox. This will turn on the spindle power amplifier and the motor should start spinning. If the motor refuses to spin, sometimes a little help may get it going. If this doesn't make it spin, change the motor phase by either plus or minus 100 (negative values are not valid, you cannot set it to less than 0). To do this, click on the motor phase edit box, change the number and then press the Enter key to make the new value take affect.
7. Now that the motor is spinning, check to make sure that it is spinning in the correct direction. If on the previous (main) dialog, the CCW radio button was checked, the motor should be going counter clock wise. If the motor is going in the wrong direction, either add or subtract 100 from the motor phase as described in step six until the motor is running in the correct direction.
8. It is now time to optimize the motor phase. The goal is to get the motor spinning as fast as it can go. The easiest way to do this is to add or subtract the value 10 to the motor phase. This is done the same way as outlined in step 6. After changing the value of the motor phase, watch the RPM display right below the Build SLUT (Sin Look Up Table) button. If adding the value 10 to the motor phase causes the RPM to decrease, try subtracting. Once you have found the direction to go with the motor phase, keep changing it until you cannot make it go any faster. It may take about 30 seconds or more for the speed to stabilize.
9. After the optimal value for motor phase has been found, press the SAVE button so you won't have to repeat this procedure in the future.

# Junior Spinstand Help Files

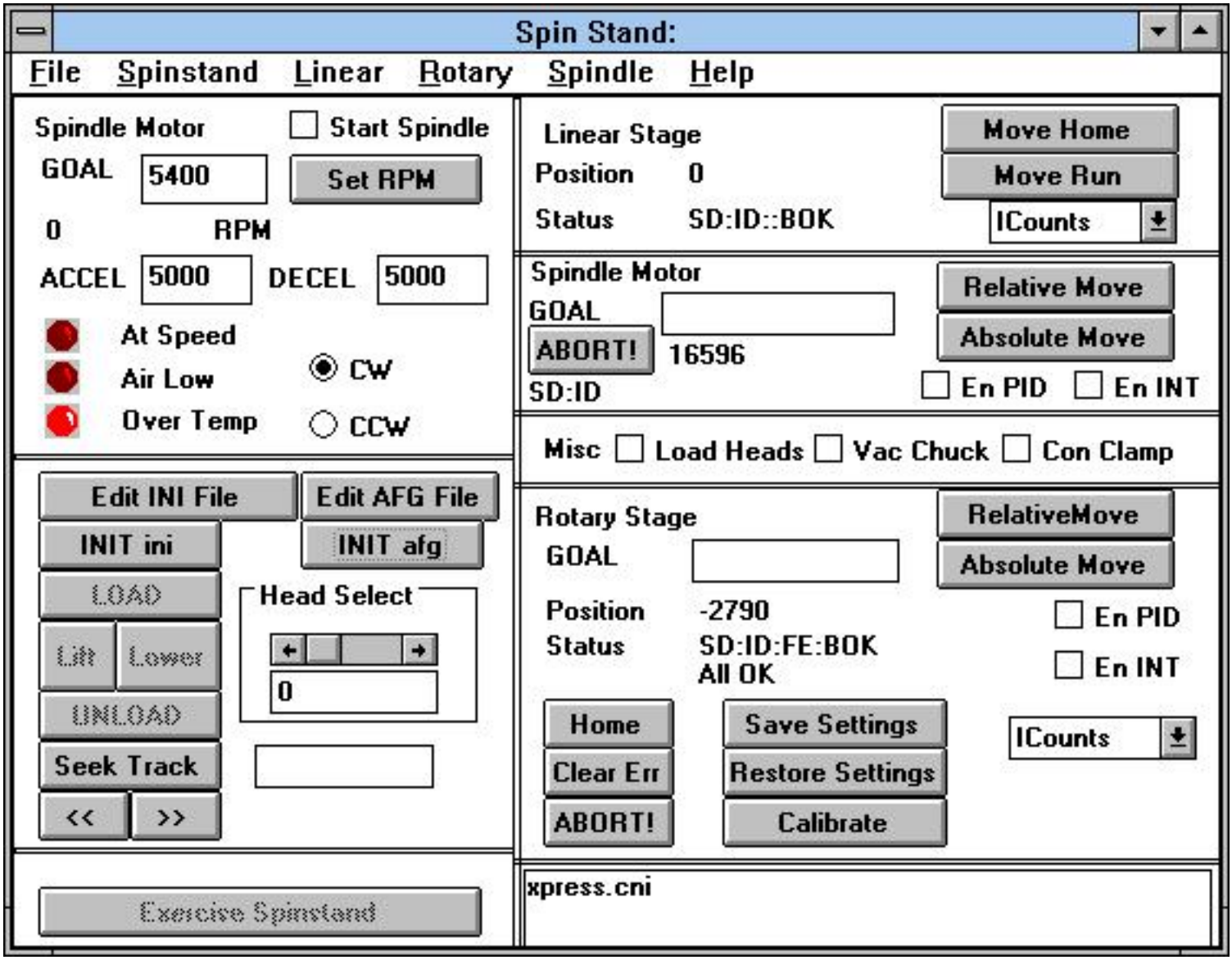
*revised 6-6-2002*

The first view you will get when you start up spinstan.exe is:




The program needs to know the name of the product.ini file you wish to use before the program can be started. This is because there is some important information in the product.ini file, such as the serial com port to be used, that is needed before the program can run. After you have selected the product.ini file you wish to use you will get the following dialog box:


You can answer this dialog NO if you are starting up the spinstan.exe program when the spinstand is already been initialized (see below). You can answer NO also if you just don't want to initialize the spinstand. YES will cause the initialization sequence to be started.




This is the main dialog of spinstan.exe.

---

**What do all of the Config Files do?**
**And, where do they come from?**

---

**Spindle Motor**

In the upper left hand corner are the spindle controller functions

**Start Spindle**

This check box is used to start and stop the spindle motor. When the box is checked, the spindle motor will run, when the check box is unchecked, the spindle motor will be stopped.

**GOAL**

There are two controls associated with this field. The edit box is where you enter the desired spindle rate in RPM, and the button to the right sets the rpm. By entering the data into the edit box and hitting the enter key, you can also set the rpm.

### RPM Display

Right below the **GOAL** field is the rpm display. This is a text box that is updated at the rate of about once every 200 milliseconds.

### ACCEL

This is the spindle acceleration field. There is only an edit box associated with this field. The units are in rpm/second. To enter the data, you must hit the enter key.

### DECEL

This is the spindle deceleration field. This is the rate that is used while the spindle motor is slowing down. The units are in rpm/second. To enter the data, you must hit the enter key.

### At Speed

This is a status indicator for when the spindle motor has achieved the entered goal. When the "led" is bright, the spindle is at speed. When the led is dull, it is not at speed.

### Air Low

This is a status indicator for when the air pressure is too low to operate the spindle motor. When the led is bright, the air pressure is low and the spindle motor should not be operated. When the led is dull, the air pressure is adequate and the spindle motor can be operated.

### Over Temp

This is a status indicator for when the power amplifier has exceeded it's save operating temperature. When the led is bright, the power amplifier has exceeded its safe operating temperature and the spindle motor should not be operated. When the led is dull, it is safe to operate the spindle motor.

### CW/CCW

These two controls indicate the direction that the motor will spin when the spindle motor is run (see above). Clicking these controls while the motor is running will not change the motor direction. The motor must be stopped for these controls to have any affect.

---

## Calibration Data

### Spinstand

The spinstand section is right below the spindle section. This group of controls allows the user to operate the spin stand system.

### Edit Ini File

pushing this button brings up the **Product Ini file editor**. Using this dialog box, the user can set up the parameters that describe the device that is to be tested.

### Edit Afg File

Do not push this button, trust me....

**INIT ini**

Pushing this button brings up the **Spinstand Initialization Dialog** box. This is where you will choose the product.ini file that will be used to initialize the spinstand.

**INIT afg**

Do not push this button, trust me.....

**LOAD**

Pushing this button brings up the **Load Heads Dialog** box. This will initiate the head loading sequence for the particular product that is selected in the product.ini file.

**Lift**

This is the lift heads button. Pushing this button will cause the heads to go to the load unload position. The heads will then be lifted off the disk, but the linear stage will not retract. This was intended so that noise measurements could be made on the head.

**Lower**

This is the lower heads button. Pushing this button will cause the heads to be lowered back onto the disk.

**UNLOAD**

This button brings up the **Unload Heads Dialog** box. This will initiate the head unloading sequence for the particular product that is selected in the product.ini file.

**Seek Track**

There are two controls associated with this field. The edit box is used to enter the desired track to seek to. The seek track button is then pushed to cause the seek to occure. You can also use the enter key after the data has been entered into the edit box to accomplish the same thing.

**<< and >>**

These two button cause the track to increment ( >> ) and to decrement ( << ) by one track.

---

The next section down is the exercise section. Pushing on the **Exersize Spinstand** button will bring up the **Exersize Spinstand dialog** box.

---

**Linear Stage Interrupts**

In the **Linear** menu, there is an entry for setting up the **Linear Interrupts**

**Linear Stage**

This section controls the linear stage.

**Move Home**

This button causes the Linear stage to move to the home position (heads unloaded).

**Move Run**

This button causes the Linear stage to move to the run position (heads loaded).

**Position**

Not used.

**Status**

Not used.

**Units Selector**

Not used.

---

**Spindle Motor**

It is better if you just ignore this section.

---

**PID Filter**

**System**

**Interrupts**

**Homing**

**Dither**

**Bode Plot**

**P.E.S.**

**Rotary Stage**

This is the section that controls the stage that moves the head.

**GOAL**

There are three controls associated with this field. The edit box is the desired position. The value in this box can be in sevreal different units depending on what the units selector box is set to. The default setting is **iCounts**, which, for the standard setup, each icount represents aproximately 1 micro degree (this figure is not at all acurate, and is only to give you an idea of how much an **iCount** is.). Units can

also be in **seconds**. A calibration factor is used to calculate this distance so this is completely accurate. The other two controls are **Move Relative** and **Move Absolute**. Pushing these controls cause the rotary stage to move. A relative move will add the goal to the current postion. Absolute move will move the stage to that position.

**Position**

This is a display of the current postion. The units are selected by the units selection box.

**Status**

This is a display of the current status of the rotary stage servo. The first sub field indicates if the PID enable status. **SD**=disabled, **SE** = enabled. The second subfield is the integrator enable status. **ID =** disabled, **IE** = enabled. The third sub field indicates the status of the following error flag. **FE** = following error flag tripped, blank = following error flag not tripped. The last field indicates a fault condition. **BOK** = everything is fine, otherwise, the appropriate fault is indicated.

**En PID**

This control enables the PID loop. When the check box is checked, the loop is enabled. When the check box is unchecked, the loop is disabled.

**En INT**

This control enables the Integrator in the PID loop. When the check box is checked, the integrator is enabled. When the check box is unchecked, the loop is disabled.

**Home**

Pushing this button will cause the homeing sequence to be executed.

**Save Settings**

Pushing this button will save all of the motion control parameters to non volatile memory (even the parameters for the linear stage and the spindle motor).

**Clear Err**

Pushing this button will cause the host computer to attempt to clear all faults that it encounters in the rotary stage.

**Restore Settings**

This button will cause the settings that are stored in non volatile memory to be restored to the motion controller. This includes the settings for the linear stage as well as the spindle motor.

**ABORT!**

Pushing this button will cause the motion controller to abort the last commanded move, home, or calibrate.

**Calibrate**

Pushing this button will cause the rotary stage to execute a calibration sequence. After the calibrate is done, the user should push the **Save Settings** button (see above) in order to save the calibration data.

**Units Selector Box**

This box is used to select the units displayed for the rotary stage. The units can be in either **iCounts**, which are the native units of the rotary stage, or in **seconds**.

---

Right below the **rotary stage** box is a general message box.

---

**Getting Revision Information**

Go to the help menu and pull it down and click on the **ABOUT** entry.

---

**Ramdisk Untilities**

---

# How to tune the Rotary Actuator

---

# Configuration Files for Accutrac ][

There are several configuration files for Accutrac ][ that are needed to make it work. What these files do, and where they go, and where they come from can be a bit confusing, to say the very least.

**servo.mni**

This is a file that store the parameters for the motion controller. The motion controller is primarily responsible for operating the rotary positioner that holds the heads. It is also responsible for the linear stage, and for starting and stopping the spindle motor. **servo.mni** is the name of the default set of parameters. When you recieve your system, they should also be a file with a name related to the particular project it is being configured for.

This file is created on the host computer by pulling down the **FILE** menu and selecting the **Save Motion Params** menu item. This takes all of the important motion parameters and saves them into an ascii format ini file on the host computer. You will be prompted to name this file. You can name it anything you want so that you can have many different configurations for various products.

To reload these parameters, you pull down the **FILE** menu and select the **Load Motion Params** menu item. This will take the contents of the **servo.mni** file and load it into the motion controller in the spindle controller. After this operation is complete, you should then press the **Save Settings** button in the main window. This will create a **SERVCFIG** file in the spindle controller ramdisk (see below).

**SERVCFIG**

This is a file that is stored inside of the spindle controller that contains all of the motion control parameters. This file is loaded into the motion controller every time power is recycled, or when you push the **Restore Settings** button on the main window. To access this file, you will pull down the **FILE** menu and click on the **Ramdisk Utils** menu item. This file can be downloaded from the spindle controller to the host to back it up. It can also be uploaded back to the spindle controller to restore it. However, the format of this file is dependent on the version of the firmware. You cannot use a file created from one version of spindle controller firmware in a controller with another version. This file is created/updated whenever the **Save Settings** button is pushed in the main window.

**MOTRCFIG**

This is a file that is stored inside of the spindle controller. It contains the information needed to run the spindle motor. Unlike **SERVCFIG** (see above), this file is not reloaded every time the power is recycled. This file can be accessed from the ramdisk utilities (**FILE | Ramdisk Utils**). This file can be downloaded to the host computer, or uploaded from the host computer.

The hardware that manages the spindle controller keeps it's own copy of this information. To update the info kept by the spindle controller, pull down the **Spindle** menu and select the **Spindle Setup** menu item. Down near the bottom of the dialog box are **SAVE** and **RESTORE** buttons. Pressing the **RESTORE** button will take the data out of the **MOTRCFIG** file in the ramdisk and load it into the spindle controller memory. Pressing the **SAVE** button will update/create the **MOTRCFIG** file in the spindle controller ramdisk.

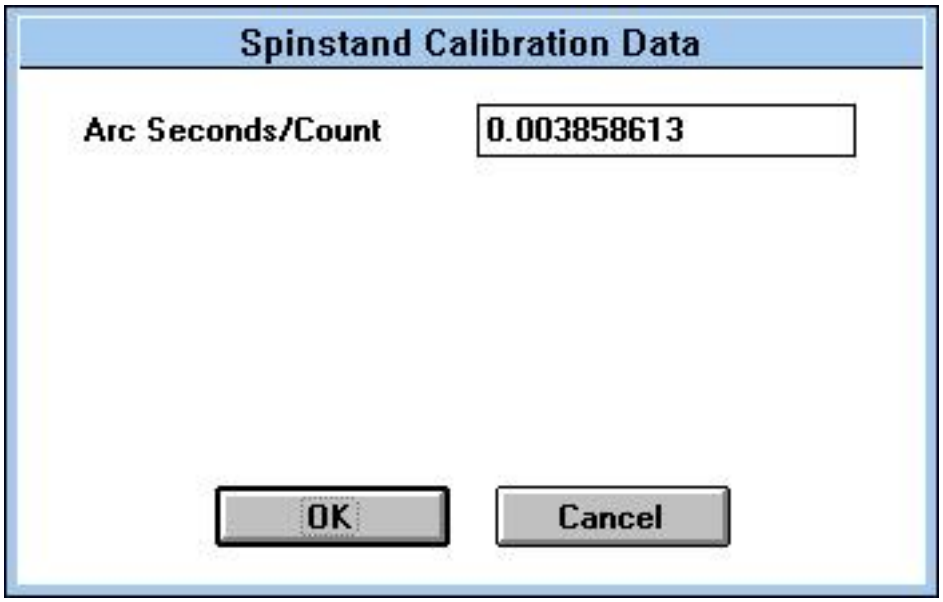
**PROD.AFG**

This is a data file that is created by the host software from the **Product.ini** file (see elsewhere). This file contains basically the same information in a different format. This is uploaded when ever an **Init** (see elsewhere) operation is initiated. The Prod.afg file is created first on the host computers hard drive, and then is uploaded to the spindle controllers ram disk. It is then used to do the spindstand initialization process.

**DATA.CAL**

This file is located on the spindle controller ramdisk. It contains the data that describes the units of micro E encoder used on the rotary actuator. This file is created/updated by pulling down the **Spinstand** menu and clicking on the **Calibration Data** menu item. A default value is always suggested.

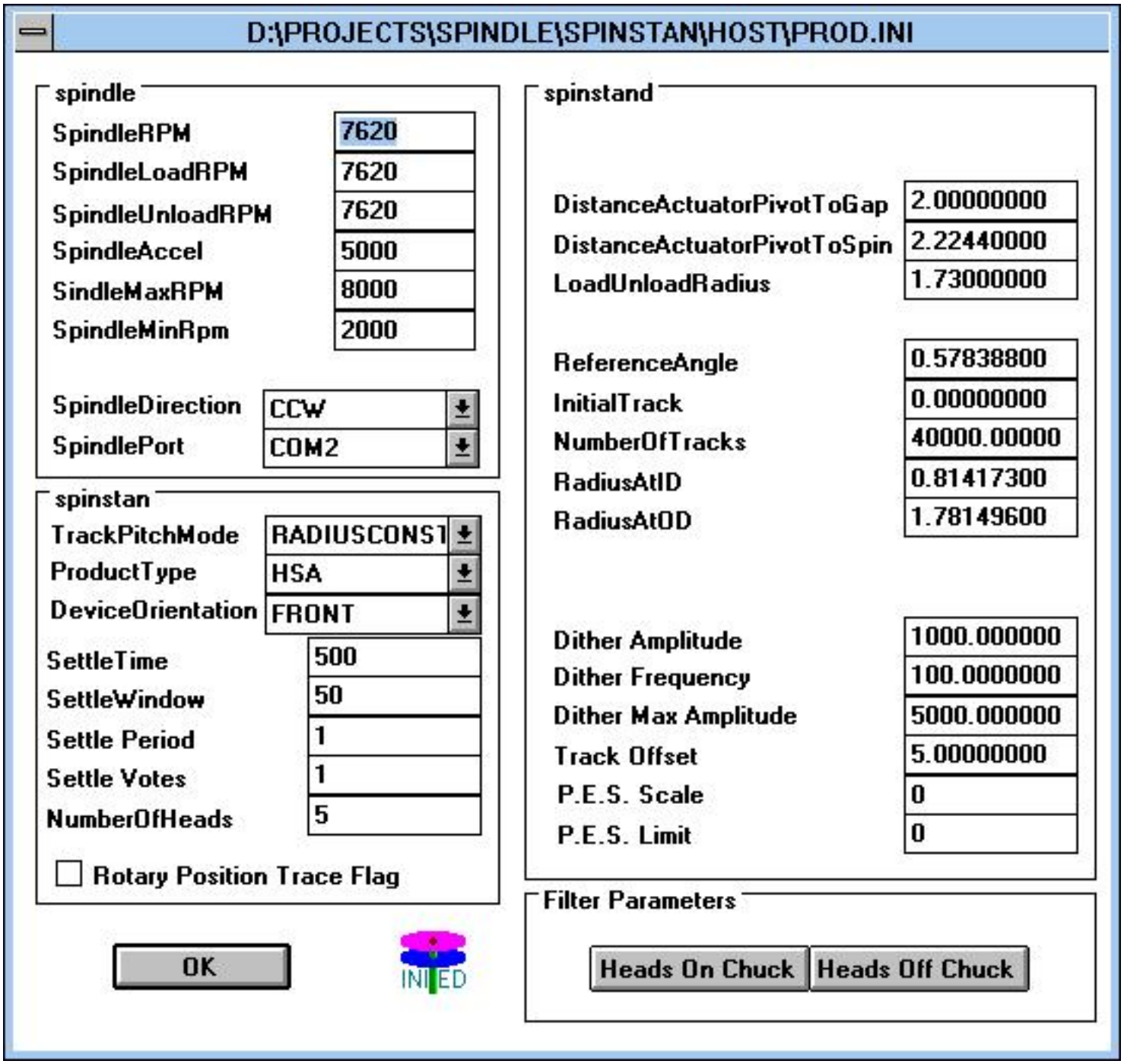
# Calibration Data Dialog Box




This is the calibration data for the MicroE encoder. The units of the data displayed is in Arc Seconds/**iCount**.

This data is stored permanently (more or less) inside the spindle controller.

# Product.ini File Editor




This dialog box is used to edit the product.ini files that define the geometry and other parameters of the test. The product.ini file is a standard windows ini file, which can be edited with a standard text editor such as notepad. This dialog box just makes it a little more convenient in that it will check to make sure that the data you enter is at least somewhat valid.

**SpindleRPM**

This is the rpm that the motor will be turning when the heads are actually being tested.

**SpindleLoadRPM**

This is the rpm that the motor will be turning when the heads are loaded onto the disk.

**SpindleUnloadRPM**

This is the rpm that the motor will be turning when the heads are unloaded from the disk.

**SpindleAccel**

This is the rate at which the spindle motor will change speed. The units are in rmp/second.

**SpindleMaxRPM**

This is the maximum allowable rpm for the spindle to rotate at.

**SpindleMinRPM**

This is the minimum allowable rpm for the spindle to rotate at (sort of useless).

**SpindleDirection**

This selector box allows you to select either clockwise (**CW**) or counter clockwise (**CCW**).

**SpindlePort**

This selector box allows you to select the com port on the PC that the spindle controller will be connected to.

**TrackPitchMode**

This is the way the tracks are space on the disk. This can be either constant radius, or constant angle.

**ProductType**

This selector box allows you to select the general type of product to be tested.

**Device Orientation**

This selector box allows you to select which side of the spindle the head will land on. **FRONT** causes the the head to land on the disk towards the back of the spinstand, and **BACK** allows the head to land on the disk towards the front of the spinstand. (don't ask).

**Settle Time**

This is the amount of time, in milliseconds, to wait for the rotary stage to settle to within the settling window.

**Settle Window**

This is the size of the window that the rotary stage must be within, in **iCounts**, in order to be considered settled.

**Settle Period**

This is the number of milliseconds to wait between checking to see if the rotary stage is within the

settle window.

**Settle Votes**

This is the number of times in a row that the settle criteria is met in a row.

**Number of Heads**

This is the total number of heads that can be tested.

**Rotary Position Trace Flag**

This records all of the goal positions and actual positions during a test.

**DistanceActuatorPivotToGap**

The distance, in inches, from the rotary stage pivot to the head gap.

**DistanceActuatorPivotToSpin**

The distance, in inches, from the rotary stage pivot to the spindle axis.

**LoadUnloadRadius**

The distance, in inches, from the spindle axis to where the heads will land on the disk.

**Reference Angle**

This is a value supplied by Axis. The units are radians.

**Initial Track**

This is the track that the rotary stage will seek to after the heads are loaded.

**NumberOfTracks**

This is the total number of tracks between the inside and outside radius, in tracks.

**RadiusAtID**

The inside radius, in inches.

**RadiusAtOD**

The outside radius, in inches.

**Dither Amplitude**

Amplitude of the dither oscillator.

**Dither Frequency**

Frequency of the dither oscillator.

**Dither Max Amplitude**

Maximum allowable dither amplitude.

**Track Offset**

Normally, this value is 0.  It can be used as a fudge factor if needed.

**P.E.S. Scale**

Scale factor for the Position Error Signal

**P.E.S. Limit**

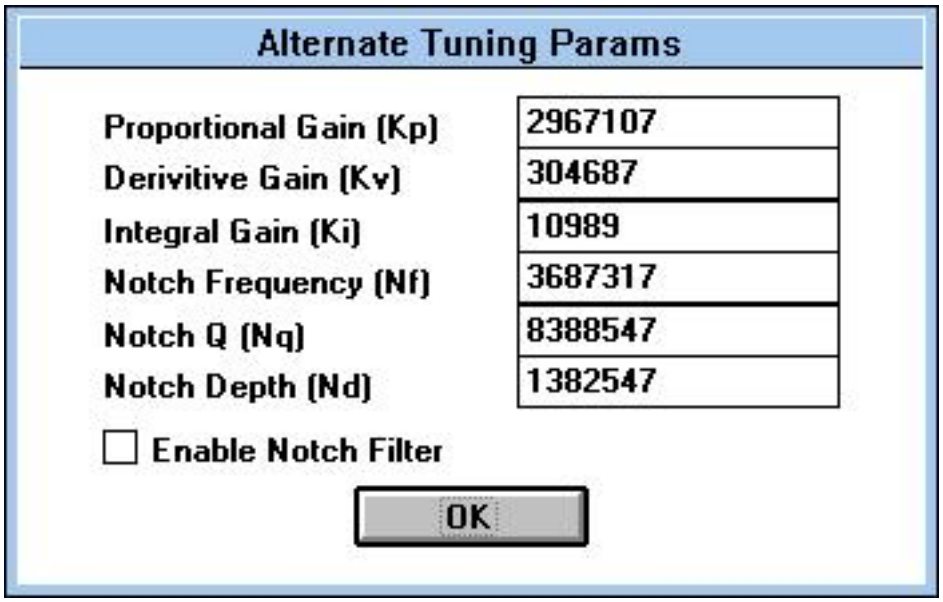
Maximum allowable amplitude of the Position Error Signal.

---

## Filter Parameters

There are different filter parameters depending whether the heads are on the chuck or not.

# Alternate Tuning Parameters



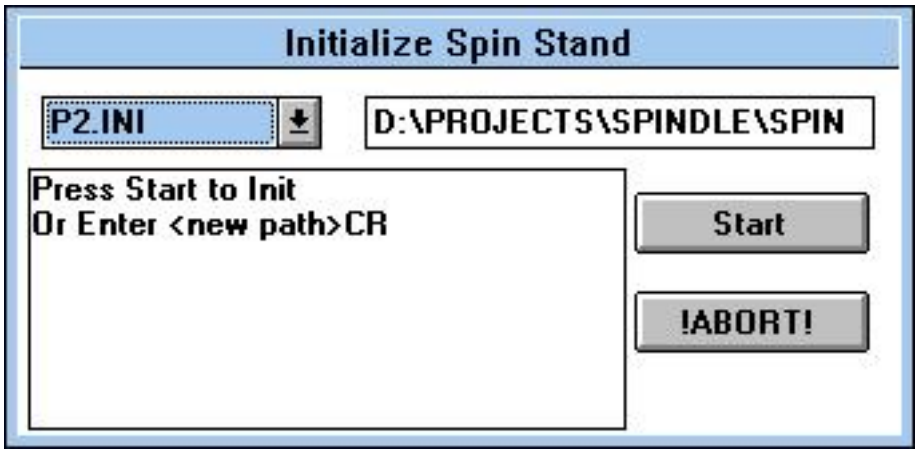

**Note:The values you see in the dialog box above are NOT typeical values.**

This dialog box is called up from the Product.ini file edit dialog.

**Proportional Gain (Kp)** Proportional gain for PID filter
**Derivative Gain (Kv)** Derivative Gain (Velocity) for PID filter
**Integral Gain (Ki)** Integral Gain for PID filter
**Notch Frequency** Notch Filter Resonant frequency
**Notch Q (Nq)** Notch Filter Q
**Notch Depth (Nd)** Notch Filter depth
**Enable Notch Filter** If checked, notch filter in enabled, unchecked, notch filter is disabled.

This dialog box is used for both heads on and heads off of chuck.

# Initialize Spin Stand Dialog Box




This is the dialog box that appears when the Initialize button is pushed.

**Product INI File Selector Box**

This is the box that selects the product.ini file to be used. You must select a file.

**Path Edit Box**

This is the path to find the product.ini files. After typing in the path, you must use the enter key to make it take affect.

**Start**

Pushing this button starts the process. When the process is done, the dialog box will close itself.

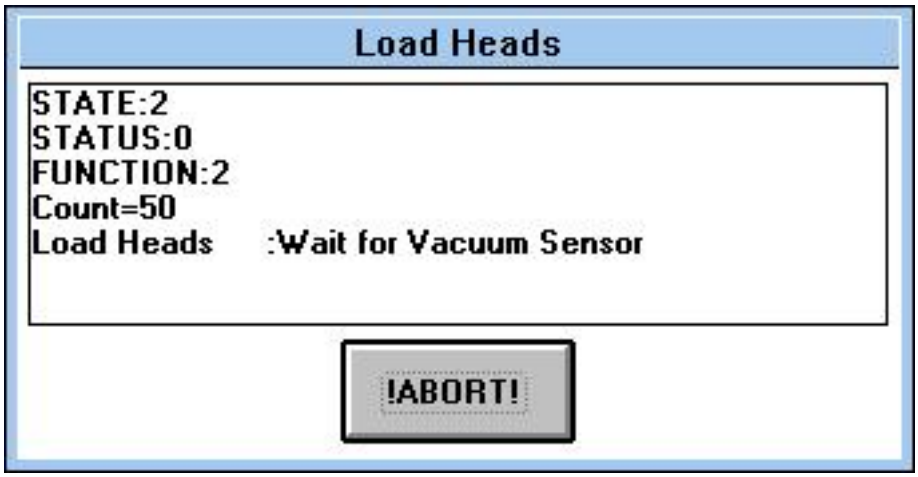
**!ABORT!**

This button aborts the initialization process and closes the dialog box.

**Text Display Box**

This box does a running commentary on the events that are happening.

# Load Heads Dialog Box




This is the load heads dialog box. This box will automatically close when the process is done.

**!ABORT!**

This button will abort the load heads process.

**Text Display Box**

This box gives a running commentary on the status of events that are happening. A lot of this data is for debug purposes.

# Unload Heads Dialog Box


This dialog appears when the unload heads button is pushed.

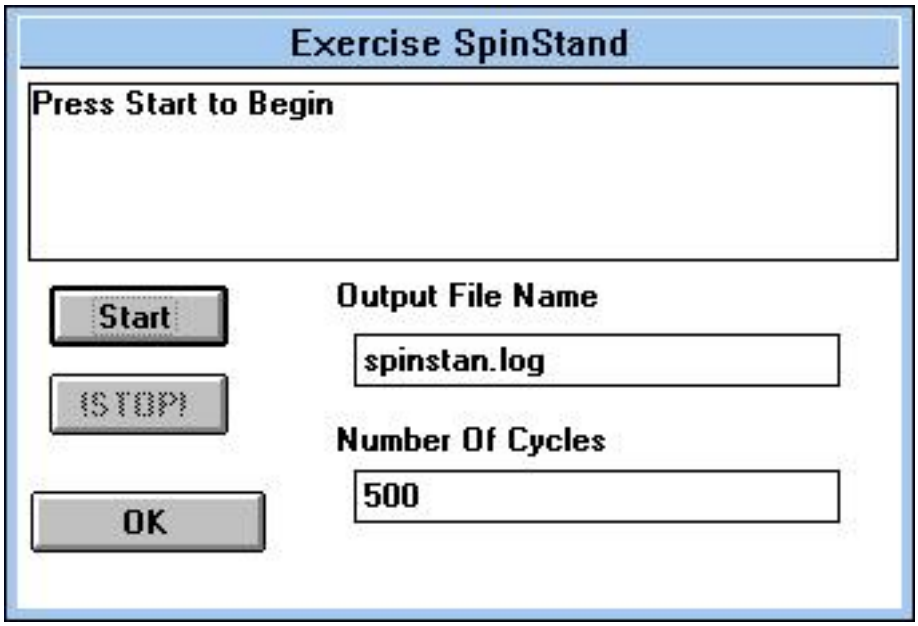
**Static Text Display Box**

This contains status information about the unload process.  Most of this data is for debug purposes.

**!ABORT!**

This button aborts the unload process.

# Exercise Dialog Box




This dialog box is used to give the hardware a pretty good workout. It will not check out the entire system, as, the system require human interface to function, so, some things are bypassed just to make the test more convenient. For example, the comb sensor is not exercised because this requires a person to stand there and put the comb into the sensor. But it does work very important components, such as the ramp loader, the linear stage, the spindle motor, and the rotary stage.

The text box at the top of the dialog box will give a running status of the test. Errors that are encountered will be displayed for only a short time. The Exercise routine will for the most part ignore errors and keep on plodding forward.

**Output File Name**

This field at this time is not used.

**Number Of Cycles**

This is the number of cycles the test will go through. 500 cycles will take about 2 hours, depending on how the pneumatics are set up. Any value up to about 32000 can be entered here.

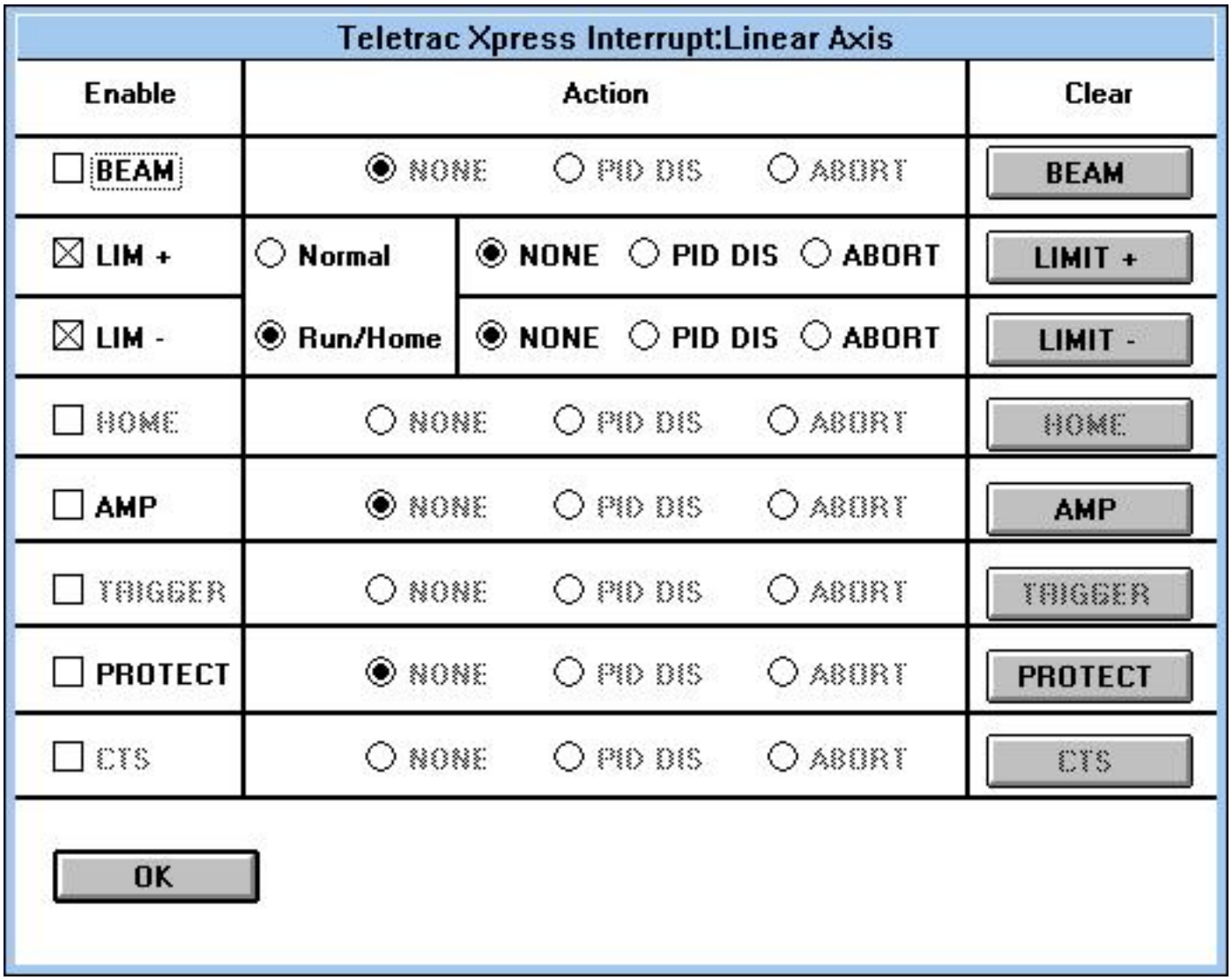
**Start**

This button starts the test.

**!STOP!**

This button will stop the test. It will not stop the test instantly. When the system is back in the home position, the test will stop.

Linear Stage Interrupts Dialog Box




This dialog box has three main sections.

**Enable**

This section has a check box that will enable/disable any particular interrupt. If the box is checked, that interrupt is enabled. If the box is unchecked, that interrupts is disabled.

**Action**

This section tells the DSP what to do when an interrupt occures. If the **NONE** radio button is selected, then nothing will be doen. If the **PID DIS** button is selected, the PID loop will be disabled when the interrupt occures, and a fault will be generated. If the **ABORT** radio button is selected, then an abort move event is generated, and if the stage is moving, the motion will be stopped immeadiately. This will also generate a **general fault**. The only exceptions to these rules are the **LIM+ and LIM-** interrupts. There is an additional mode that is selected by two radio buttons. The **Normal** radio button causes the Action modes to behave just like all of the reset. If the **Run/Home** radio button is selected, this will cause a message to be sent to the main microprocessor to let it know that the linear stage has tripped a limit. These actions will not generate any faults.

**Clear**

These buttons can be used to clear an interrupt (and it's associated faults). If the interrupt cannot be cleared, because what ever caused the problem is still doing so, the command will be **NAK**ed (**N**ot **A**c**K**nowleged).

---

The follwing interrupts are:

**BEAM**

This is the Laser Beam Interrupted Interrupt. This interrupt is also used for any other encoder fault.

**LIM+**

This is the limit switch that is in the positive direction of travel.

**LIM-**

This is the limit switch that is in the negative direction of travel.

**HOME**

This interrupt is not used.

**AMP**

This is an interrupt that is generated by the power amplifier.

**TRIGGER**

This interrupt is not used.

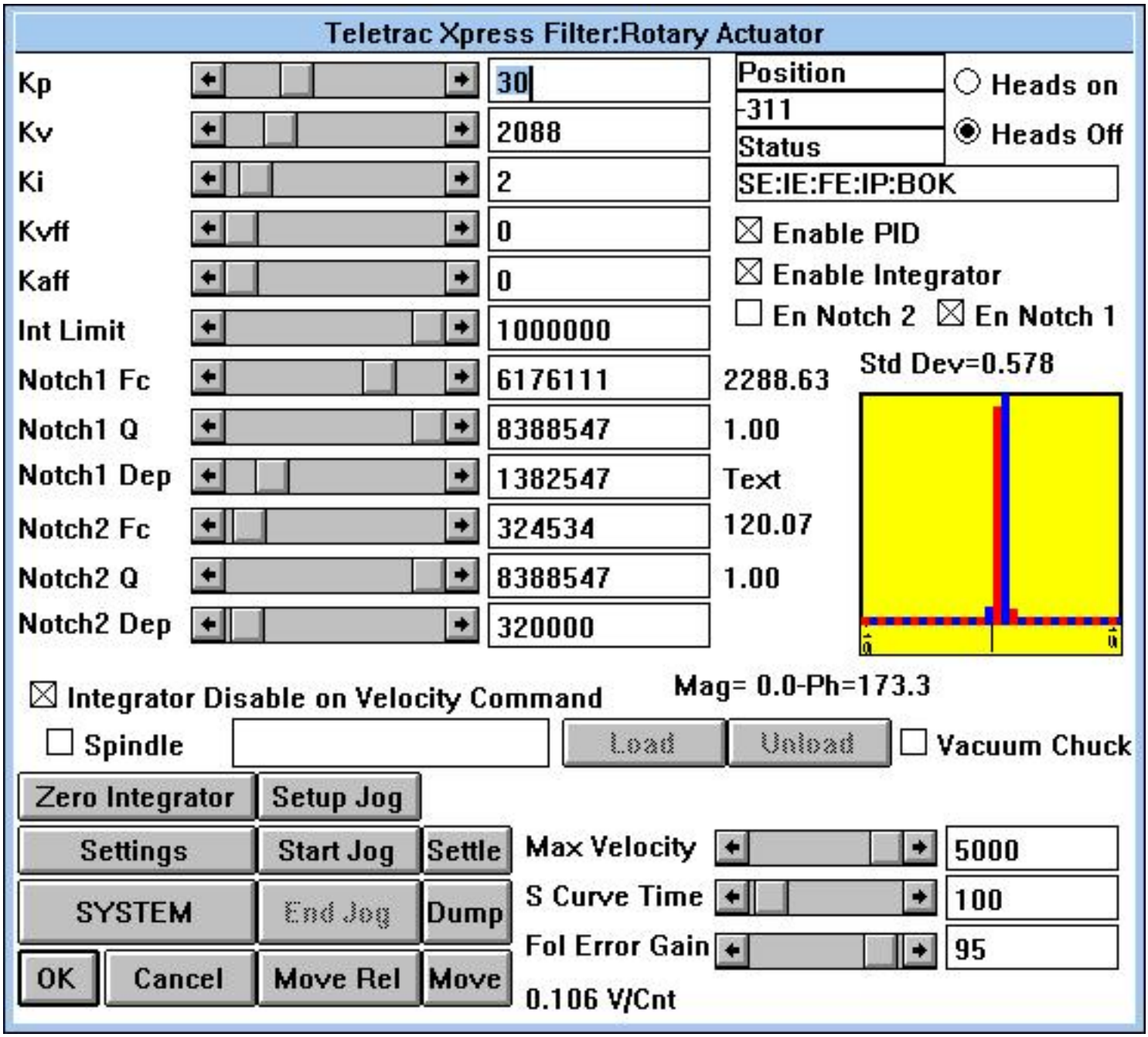
**PROTECT**

This is a general purpose interrupt. Generally, this is tied to the laser lock signal.

**CTS**

This interrupt is not used.

# Filter Dialog Box




The filter dialog box is used for tuning the PID filter in the servo loop.

**Kp**

Proportional Gain. There are two controls in this field. The slider and edit box are connected together. Changing one will change the other.

**Kv**

Velocity (derivative) Gain. There are two controls in this field. The slider and edit box are connected together. Changing one will change the other.

**Ki**

Integral Gain. There are two controls in this field. The slider and edit box are connected together. Changing one will chage the other.

**Kvff**

Velocity Feed Forward Gain. There are two controls in this field. The slider and edit box are connected together. Changing one will change the other.

**Kaff**

Acceleration Feed Forward Gain. There are two controls in this field. The slider and edit box are connected together. Changing one will change the other.

**Int Limit**

Integrator Limit. This is the maximum value that the integrator can attain. There are two controls in this field. The slider and edit box are connected together. Changing one will change the other.

**Notch Fc**

Notch Filter Frequency. This will be the resonant frequency of the notch filter. There are three controls in this field. The slider and edit box are connected together. Changing one will change the other. There is a text control just to the right that indicates the true resonant frequency (in Hz) of the filter.

**Notch Q**

Notch Filter Q. This will adjust the filter quality factor (damping). Low Qs have high damping, high Qs have low damping. There are three controls in this field. The slider and edit box are connected together. Changing one will change the other. There is a text control just to the right that indicates the true filter Q.

**Notch Dep**

Notch Filter Depth. This control adjust the depth of the notch. When at 0, the notch depth is infinate. When it is at its maximum value, the notch depth is 0 (flat). There are two controls in this field. The slider and edit box are connected together. Changing one will change the other.

**Integrator Disable on Velocity Command**

This control turns this option on and off. When it is on (check box is checked), when ever the command velocity is non zero (the stage is being commanded to move), the integrator will stop accumulating following error. This is to prevent the integrator from saturating during moves.

**Max Velocity**

This field sets the maximum allowable velocity the stage can go. The units are in **iCounts/servo cycle**. In general, it is a good idea to set this to a nice round number. There are two controls in this field. The slider and edit box are connected together. Changing one will change the other.

**S Curve Time**

This field sets the number of **servo cycles** that the DSP will use to perform the **S Curve**. This is the time for a single **S Curve** and it takes two in order to complete a move (one to speed up, one to slow down). Idealy, this number should divide evenly into **Max Velocity** and this value should also be divisable by 2. There are two controls in this field. The slider and edit box are connected together. Changing one will change the other.

**Fol Error Gain**

This field sets the gain for the following error test point on the motion controller. This field has 3 controls. The slider and the edit box are connected together so that changing one will change the other. The value displayed by these two controls only has meaning to the machine. The third control, a text display box, displays the gain in **volts/iCounts**.

**Goal Edit Box**

Right next to the static text display for the follow error gain, is an edit box for entering in a position goal. The units for the box are **icounts** only.

**Zero Integrator**

This button resets the integrator accumulator to zero.

**Setup Jog**

This button brings up the **Jog Setup Dialog Box**.

**Load**

This button start the **Load Heads** sequence. It should be noted that in order for this button to function properly, the spinstand must have been put through the initialization process.

**Spindle**

This check box will start/stop the spindle motor. When the box is checked, the spindle motor will be spinning. When the box is unchecked, this spindle motor will be stopped.

**Settings**

Pressing this button brings up the settings dialog box for the filter dialog box. This dialog controls the range over which the sliders in this view will operate.

**Start Jog**

Pressing this button will start the jogging proceedure as setup by the **Jog Setup Dialog Box**.

**Settle**

This button serves no purpose.

**Unload**

This button starts the **Unload Heads** sequence. It should be noted that in order for this button to function properly, the spinstand must have been put through the initialization process.

### Vacuum Chuck

This control activates the vacuum chuck solonoid. When the check box is checked, the vacuum chuck will suck. When the check box is unchecked, the vacuum chuck will no longer suck.

### System

This button will bring up the **System Configuration Dialog Box**.

### End Jog

This button will terminate the jogging process.

### Dump

This button causes a memory dump from the DSP. This is for debug operations, and the functionality is undefined.

### Move Rel

This button cause the host program to read the **Goal Edit Box** (see above), add this value to the current commanded position, and move to that new position.

### Move

This button causes the host program to read the **Goal Edit Box** (see above), and move to that position.

### Position

This is a static text box that displays the postion. Units are always in **iCounts**.

### Status

This is a static text box that displays the current status. The first sub field shows the PID status. **SE**= servo enabled, **SD** = servo disabled. The second sub field shows the integrator status. **IE**= integrator enabled, **ID**= integrator disabled. The third sub field indicates if the following error flag has been tripped. **FE**= following error flag tripped, blank is following error flag not tripped. The forth subfield indicates the in position status. **IP**= position is in position, blank is position is not in position. The fifth sub field indicates the fault status. **BOK**=no faults have been tripped, otherwise the appropriate message will be displayed indicating the fault.

### Heads on/Heads off

These two radio buttons select which set of tuning parameters will be displayed. There is a different set of parameters depending on weather the heads are on or off of the chuck. This is needed as the extra mass can change the resonant frequency of the motor shaft, thus changing the frequency that the notch filter must be tuned. When the filter dialog box is exited by using the **OK** button, the dialog box will ask if you wish to update the alternate tuning paramters.

### Enable PID

This checkbox will enable/disable the PID loop. When this box is checked, the PID will be

enabled. When this box is unchecked, the PID will be disabled.

**Enable Integrator**

This checkbox will enable/disable the Integrator. When this box is checked, the Integrator will be enabled. When this box is unchecked, the Integrator will be disabled.

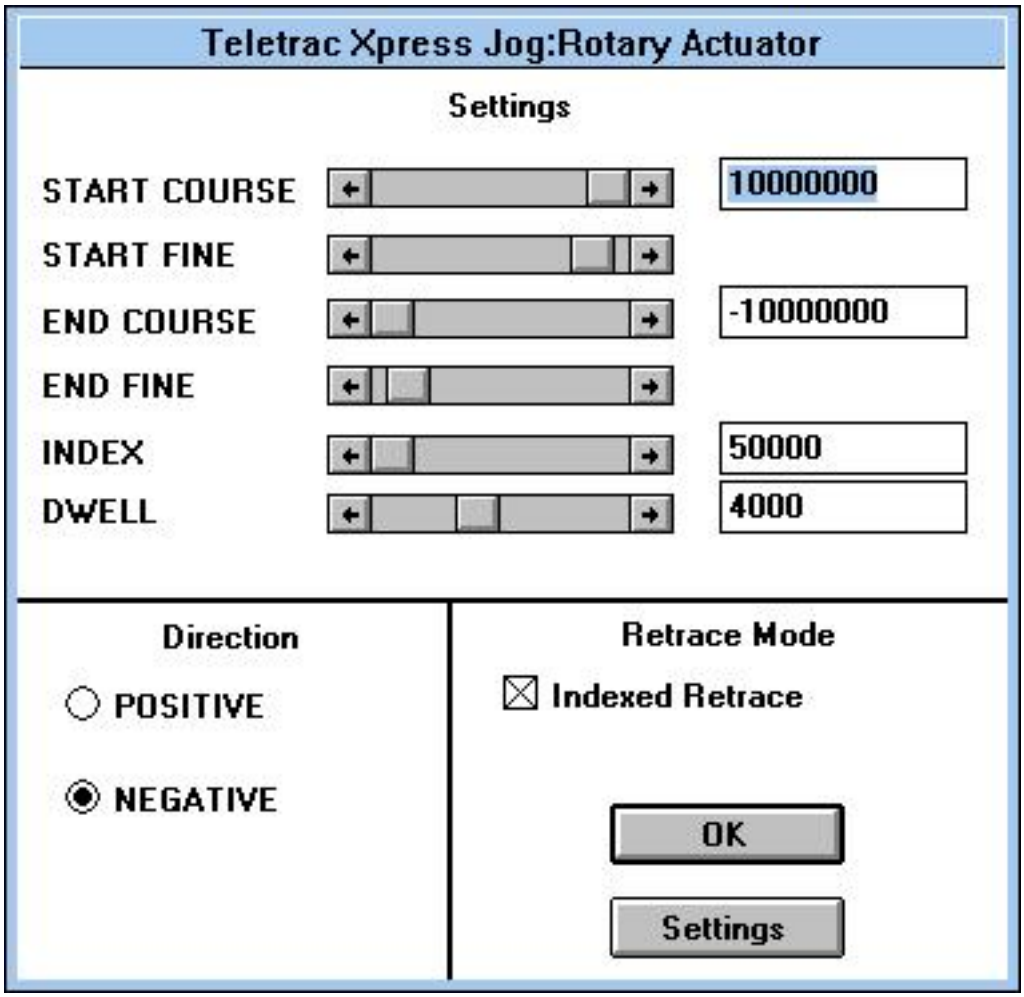
**Enable Notch Filter**

This checkbox will enable/disable the Notch Filter. When this box is checked, the Notch Filter will be enabled. When this box is unchecked, the Notch Filter will be disabled.

**Position Histogram**

The big yellow box displays the position histogram. The histogram consists of 32 bins. The bins range from -16 to +15 **iCounts** relative to the current Goal position. What we want to see is the 0 bin to be full magnitude, and all of the other bins to be at 0. In normal operation, you will see a bell curve distribution of positions. Above the histogram display is a standard deviation display. This is a 1 sigma standard deviation. This is meant to only be a guide as to how well the stage is holding position.

# Jog Settup Dialog Box




**START COURSE**
**START FINE**

These fields adjusts the start position of the jog. There are three controls associated with these fields. The two sliders and the edit box are all connected together. Changing any control will change the others as well as the value in the controller card.

**END COURSE**
**END FINE**

These fields adjusts the end position of the jog. There are three controls associated with these fields. The two sliders and the edit box are all connected together. Changing any control will change the others as well as the value in the controller card.

**INDEX**

This field sets the distance that will be moved on each jog iteration. There are two controls associated with this field. The slider and edit box are connected together so that changing one will change the other, as well as the value in the controller card.

**DWELL**

This field sets the time that the positioner will remain at each position while jogging. There are three controls associated with these fields. The two sliders and the edit box are all connected together. Changing any control will change the others as well as the value in the controller card.

**DIRECTION**

There are two controls associated with this box. Clicking on the **Postive** radio button will cause the positioner to start going in the positive direction when the **Start Jog** button is pushed in the **Filter** dialog box. Clicking on the **Negative** radio button will cause the positioner to start going in the negative direction when the **Start Jog** button is pushed in the **Filter** dialog box.

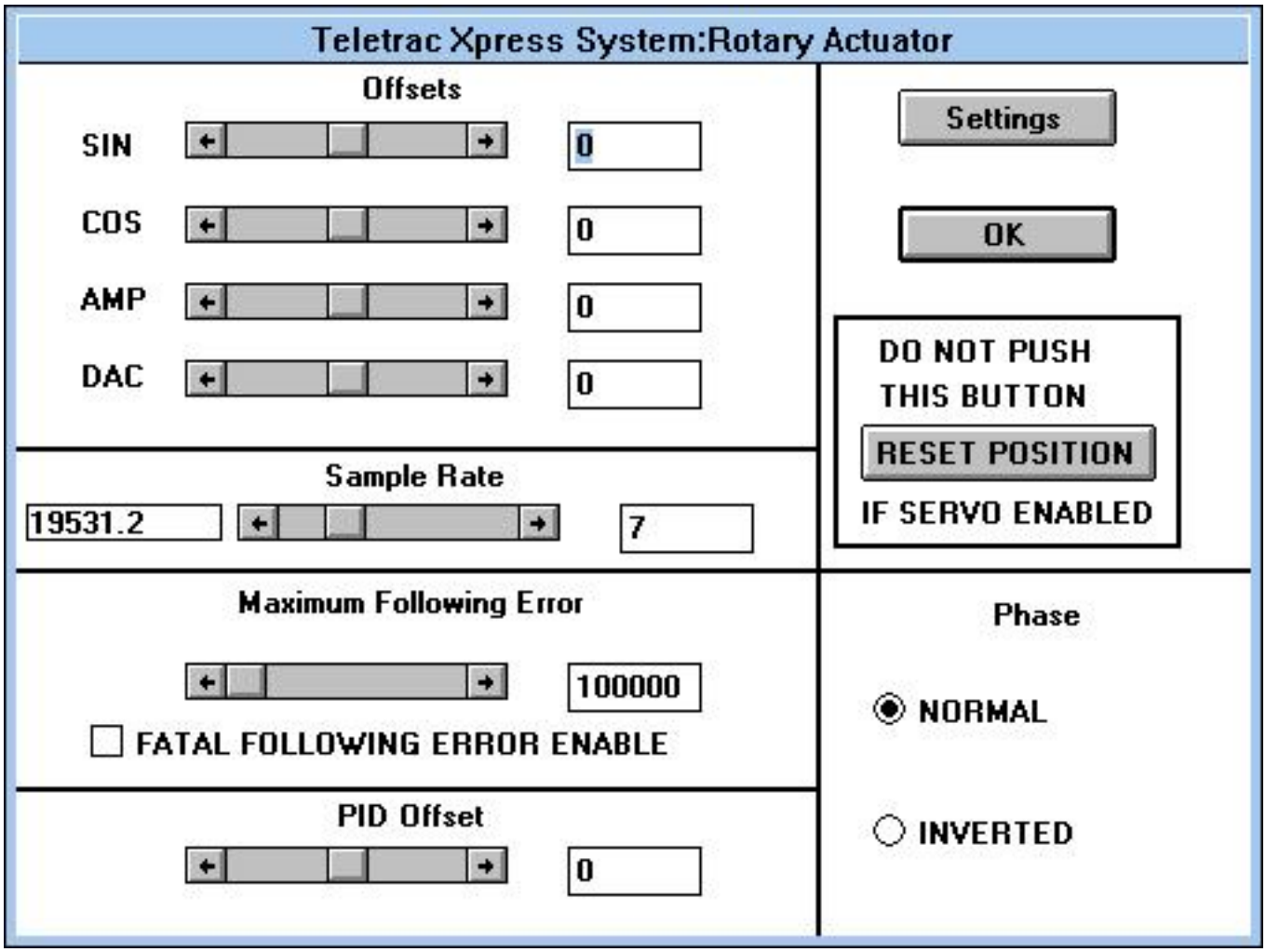
**MODE**

There is one control associated with this box. Checking the **Indexed Retrace** checkbox will cause the positioner to step back towards the start position once it has reached the end position. Unchecking this box will cause the positioner to make one large move back towards the start position once it has reached the end position.

**Settings**

This button brings up the settings dialog box. This is where the ranges of the sliders can be set.

# System Setup Dialog Box




### Offsets

None of these controls do anything.

### Sample Rate

This field sets the number of **servo cycles** / second. There are three controls in this field. The slider and edit box are connected together so that if one changes the other changes as well. The third control is a static text display that shows the actual sample rate. It should be noted, that for the **Rotary Actuator**, the sample rate value must be set to 7 (19531.2Hz), or the system will not operate properly.

### Maximum Following Error

This value is set by two controls. The slider and edit box are connected together so that changing one will change the other. This is the maximum value that the PID loop will allow the following error to be, but it cheats. If the following error is greater, it just limits the following error.

### FATAL FOLLOWING ERROR ENABLE

This checkbox will enable the fatal following error fault. When this box is check, the PID loop will be disabled whenever the following error exceeds the **maximum following error** (see above). This option is more of an irritation rather than useful.

### PID Offset

There are two controlls in this field. The slider and edit box are connected together so that changing one will change the other. This value should be set to 0. It has some uses in system debug.

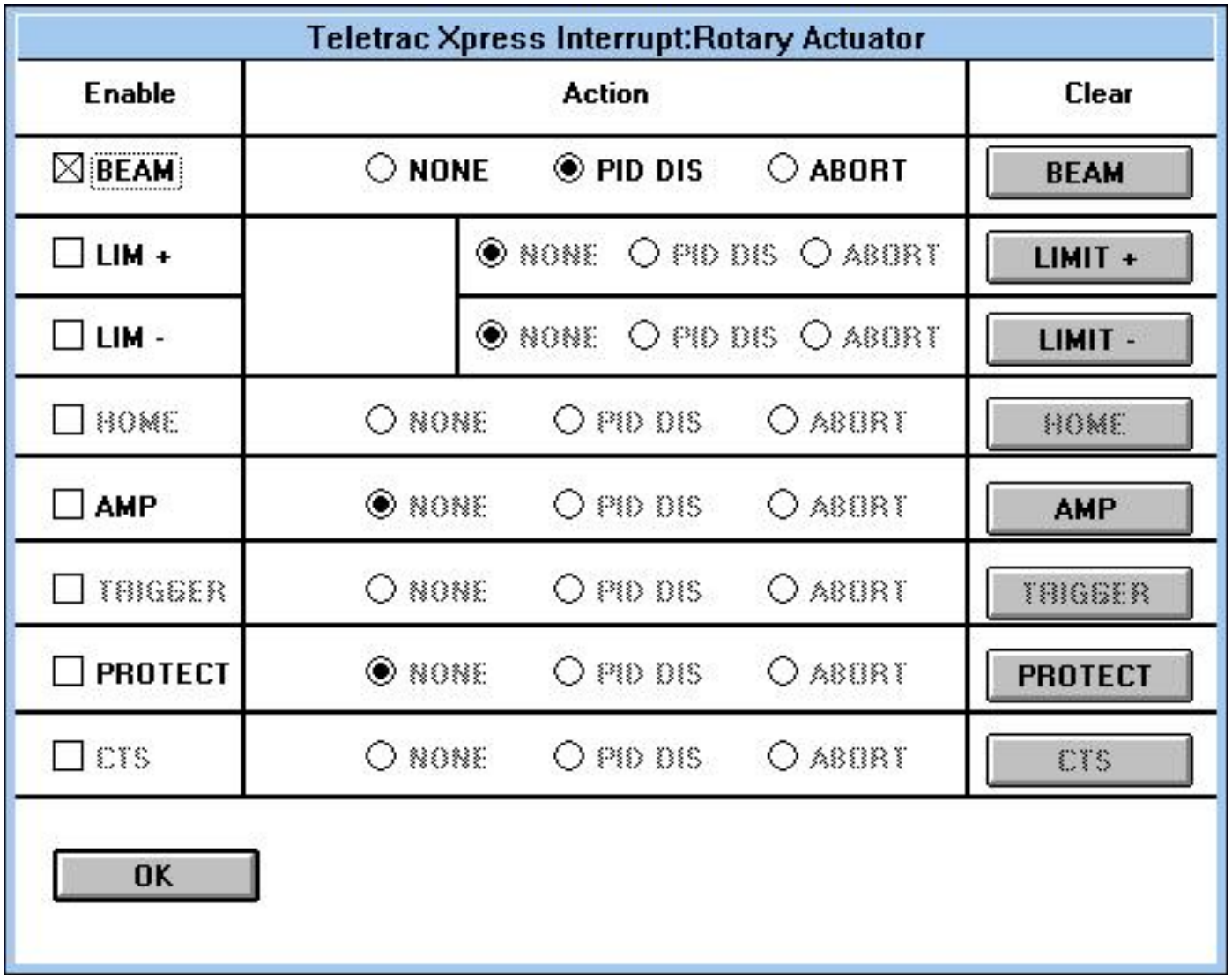
### RESET POSITION

This button resets the position counter to 0. In newer versions of the firmware, nothing bad will happen when you push this with the PID enabled. However, this is not recomended.

### Phase

This field inverts the phase detector for the positioner. This field has no affect on the rotary positioner.

# Rotary Stage Interrupts Dialog Box




This dialog box has three main sections.

**Enable**

This section has a check box that will enable/disable any particular interrupt. If the box is checked, that interrupt is enabled. If the box is unchecked, that interrupts is disabled.

**Action**

This section tells the DSP what to do when an interrupt occures. If the **NONE** radio button is selected, then nothing will be doen. If the **PID DIS** button is selected, the PID loop will be disabled when the interrupt occures, and a fault will be generated. If the **ABORT** radio button is selected, then an abort move event is generated, and if the stage is moving, the motion will be stopped immeadiately. This will also generate a **general fault**.

**Clear**

These buttons can be used to clear an interrupt (and it's associated faults). If the interrupt cannot be cleared, because what ever caused the problem is still doing so, the command will be **NAK**ed (**N**ot

**Ac**K**n**owleged).

---

The follwing interrupts are:

**BEAM**

This is the Laser Beam Interrupted Interrupt. This interrupt is also used for any other encoder fault.

**LIM+**

This is the limit switch that is in the positive direction of travel.

**LIM-**

This is the limit switch that is in the negative direction of travel.

**HOME**

This interrupt is not used.

**AMP**

This is an interrupt that is generated by the power amplifier.

**TRIGGER**

This interrupt is not used.

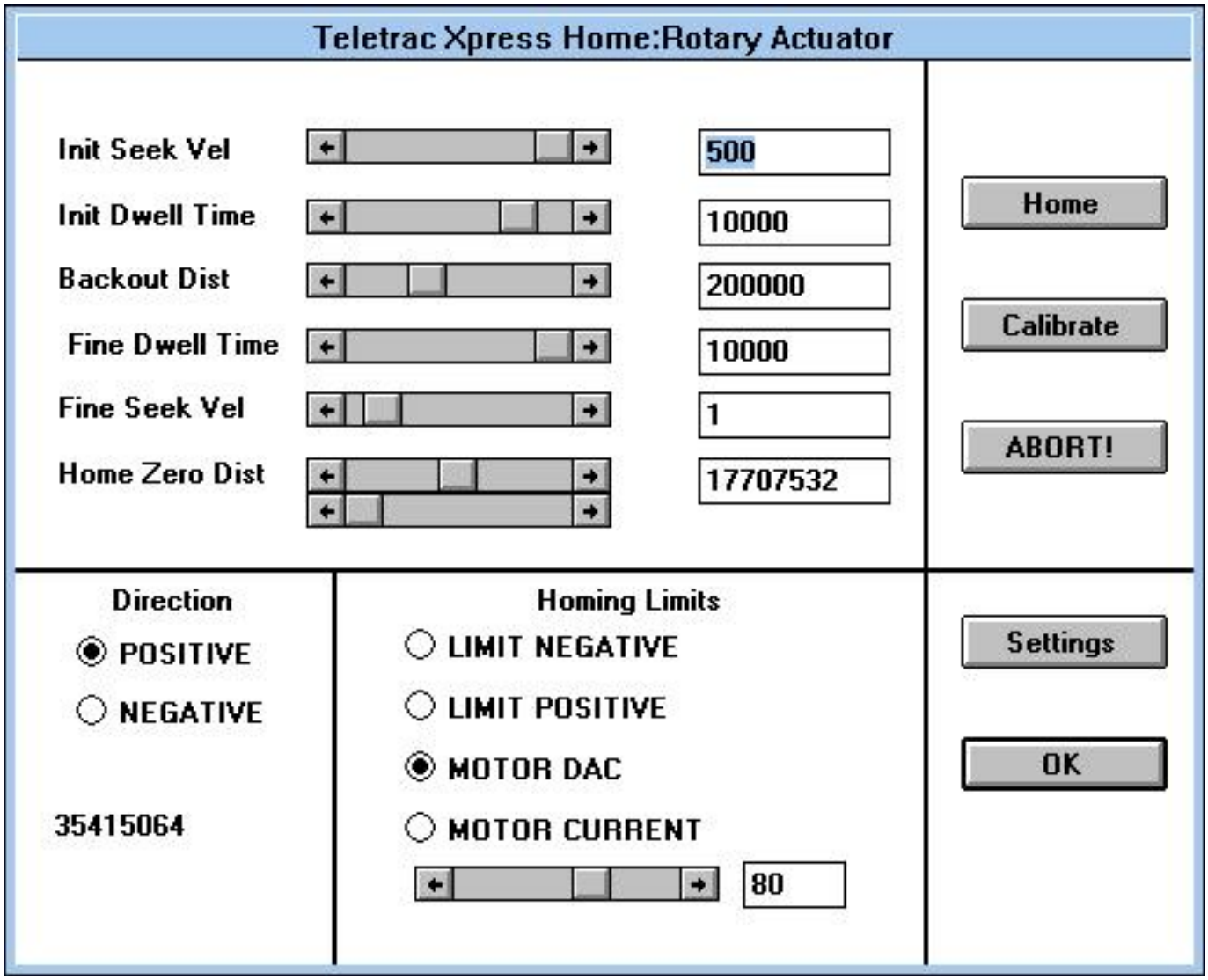
**PROTECT**

This is a general purpose interrupt. Generally, this is tied to the laser lock signal.

**CTS**

This interrupt is not used.

# Homing Dialog Box




**Init Seek Vel**

This field sets the initial velocity the the actuator will move to find its first limit. There are two controls in this field. The slider and edit box are connected togethers so that when one is changed, the other changes, along with the value on the controller card.

**Init Dwell Time**

After finding the limit, this is the amount of time, in servo clock cycles, that the controller will wait before backing away from the limit. There are two controls in this field. The slider and edit box are connected togethers so that when one is changed, the other changes, along with the value on the controller card.

**Backout Dist**

This is the distance that the controller will move the actuator away from the limit. There are two controls in this field. The slider and edit box are connected togethers so that when one is changed, the other changes, along with the value on the controller card.

**Fine Seek Vel**

This field sets the velocity the actuator will move to find the limit with fine precision. There are two controls in this field. The slider and edit box are connected togethers so that when one is changed, the other changes, along with the value on the controller card.

**Fine Dwell Time**

After finding the limit with fine precision, this is the amount of time the controller will wait, in servo clock cycles, before backing away from the limit. There are two controls in this field. The slider and edit box are connected togethers so that when one is changed, the other changes, along with the value on the controller card.

**Home Zero Distance**

This is the distance to move away from the limit after it has been found.

**Direction**

Home can be done in a positive or negative direction.

**Travel Limit**

This selects the type of limit. You can use the **positive limit** (generally an interrupter), **negative limit** (generally an interrupter), or the **motor dac**. When using the limits, you must make sure that the direction you choose matches the limit. When the motor dac is used, use the slider to set the threshold to determine when you have hit something hard (the motor dac is used to find hard stops). **Motor Current** is no longer supported.

**Home**

Pushing this button causes a homing sequence to be done.

**Calibrate**

Pushing this button causes a calibration sequence to be done.

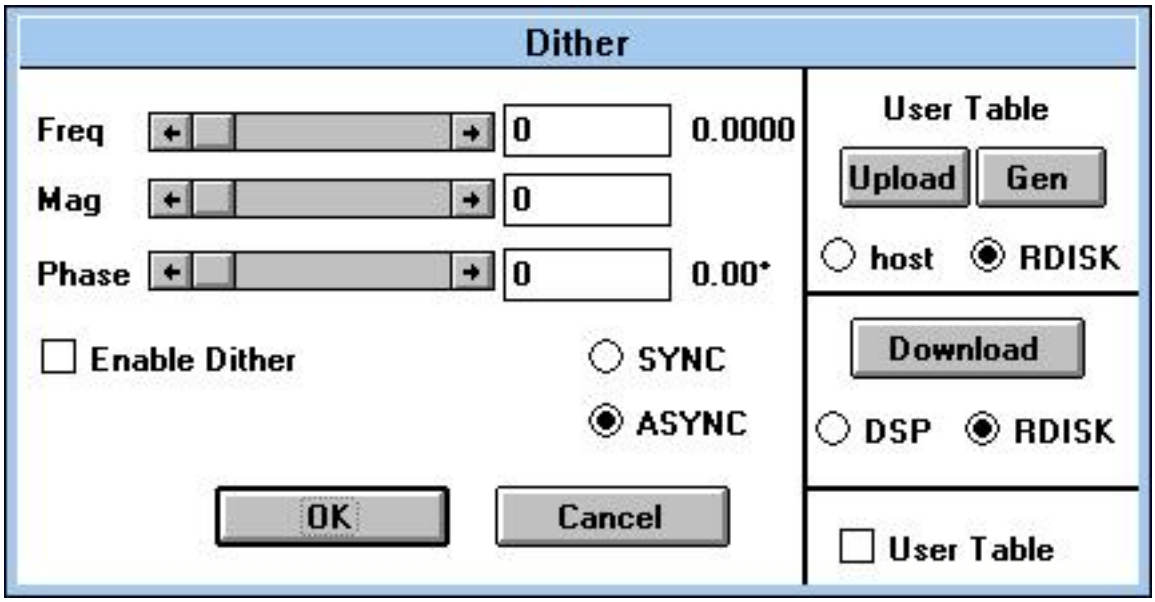
**ABORT!**

Pushing this button will cause a move, home, or calibrate sequence to be terminated.

**Calibration Distance**

Right below the direction radio buttons is a number that will indicate the total distance between the two limits.

# Dither Dialog Box




The dither can be operated in two different modes. The mode is selected by the **SYNC/ASYNC** radio buttons. We will first consider the Async mode.

The async mode is just like having a function generator inside of your spindle controller. The default waveform is a sinusoid. It should be noted that this signal is added to the goal position.

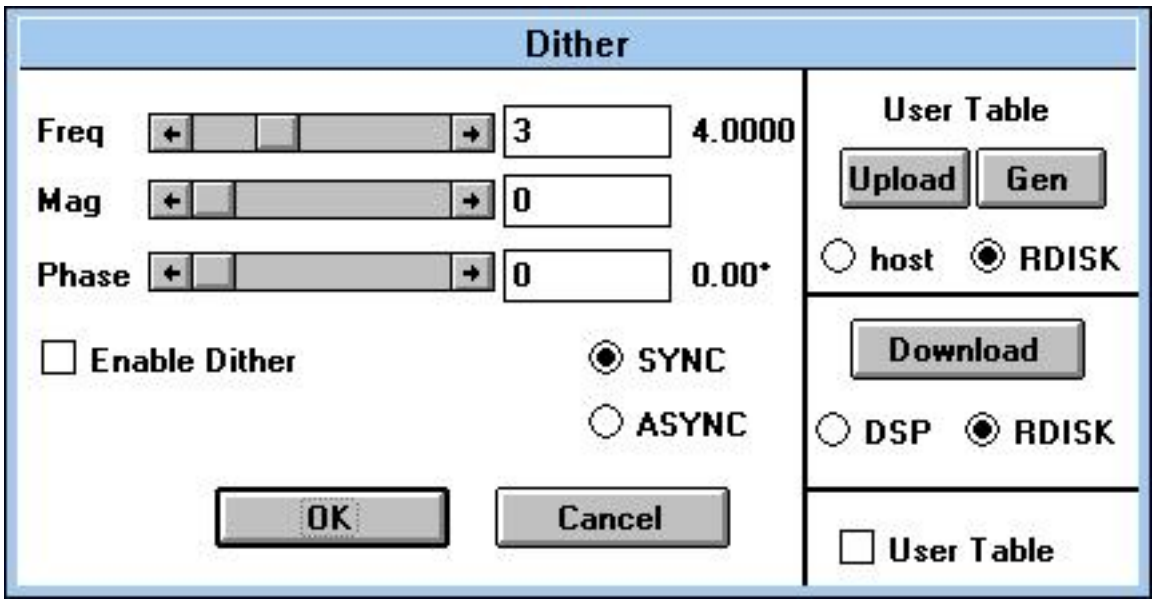
Only the functions that differ from the sync mode will be discussed for the momemt. For all the functions that are identical, see below.

**Freq**

There are three controls associated with this field. The slider and the edit box are connected together, so that if you change one, the other will change as well. To the right is a static control that will display the actual frequency of the signal.

**Phase**

The phase has no real affect on anything in the async mode. It does shift the phase, but, since the signal is not referenced to anything, you will not see anything happen.




The sync mode, the dither oscilator is slaved to the spindle motor encoder. This will allow you to coordinate the movement of the rotary stage to the rotation of the spindle motor. Posible uses of this mode is to eliminate repeatable runout of the spindle motor.

**Freq**

In the sync mode, this is a prescale value. The number displayed on the far right divides into the number of encoder line. The result should be 256. For example, if you have a 512 line encoder.....hmmm...don't remember how this works...oh well.

**Phase**

This is the phase angle with relation to the index pulse coming out of the spindle motor. The reslolution of this adjustment is 1.406 degrees.

---

**Mag**

This is the magnitude of the dither oscilator, in "peek" **icounts**. So, if this value is set to 2000, the amplitude of the dither oscilator will be plus and minus 2000 **iCounts**.

**Enable Dither**

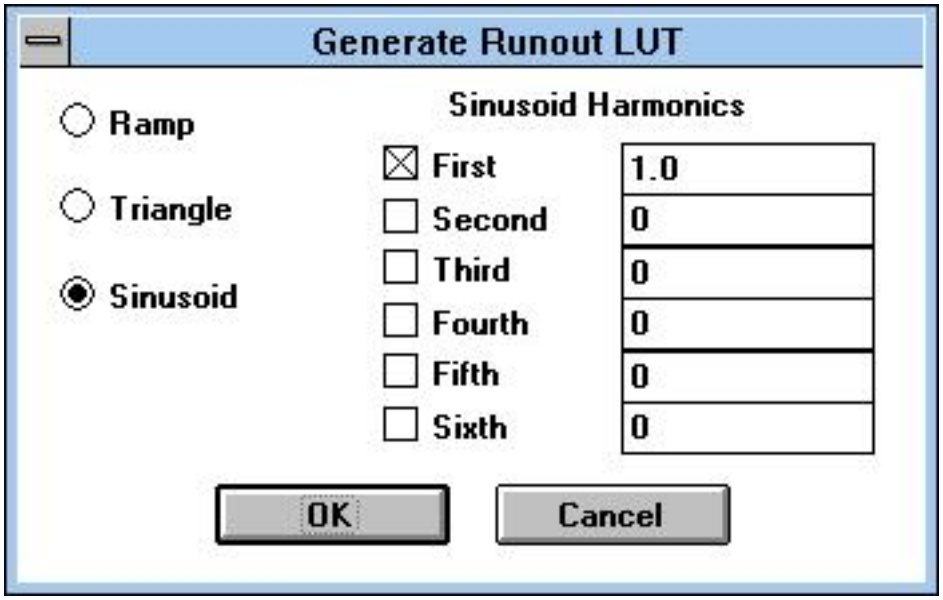
This turns the dither oscilator on or off. When this check box is checked, the dither is on, when this check box is unchecked, the dither oscilator is off.

---

**GEN**

I highly recomend that the rest of this dialog box be ignored.

# Generate Runout Look Up Table (L.U.T.)




This dialog box is used to generate lookup tables for the dither oscilator (which can be used to correct for runout...).

**Waveform select**

The user can select sine, triange, or ramp wave forms by selecting the coresponding radio button. If sinusoid is selected, the user can select which harmonics are to be used.

**Sinusoid Harmonics**

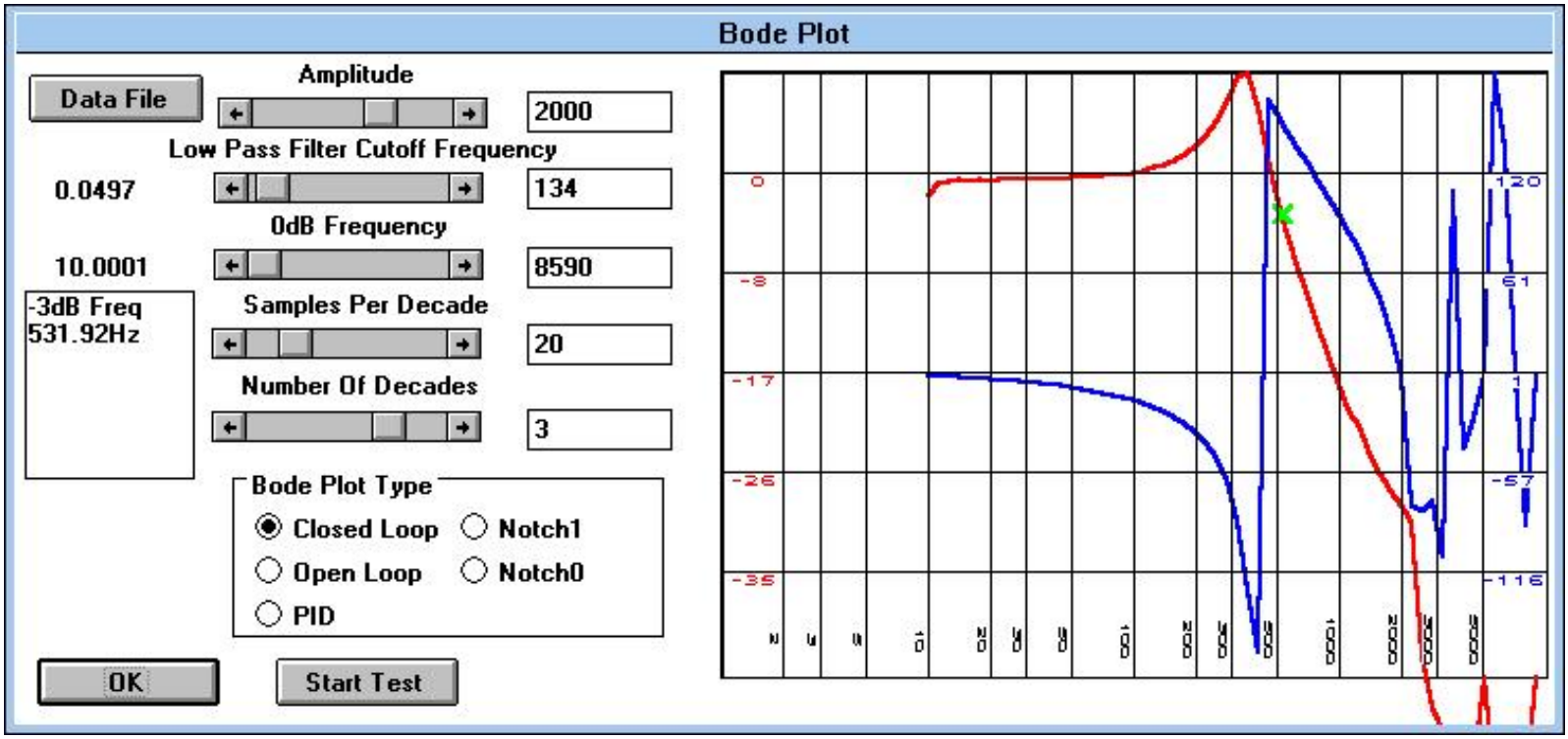
The check boxes to the left determine which harmonics are to be used. At least one box must be selected. If multiple boxes are selected, the algebraic sum of the selected harmonics is created. The edit boxes on the left are the relative amplitudes of the harmonics. These values must be equal to or less than 1.

**OK**

Pressing the OK button causes the table to be generated.

# Bode Plot Dialog Box

*updated 6-6-2002*




This dialog box is used to analyze the closed loop frequency response of the servo system. A typical resonse is shown above. The red line is the amplitude response, and the blue line is the phase response.

**Data File**

Use this button to bring up a file dialog box to select the file to save the frequency response data into. The file is a text file.

**Amplitude**

This is the amplitude of the stimulating signal for doing the bode plot. In general, it is recomended that this value be set to 2000. The units of the field are **iCounts**.

**Low Pass Filter Cutoff Frequency**

This is the cutoff frequency of the low pass filter at the output of the hetrodyne converter that is used to measure the response. There are three controls in this field. The slider and edit box are connected together so that when one changes, the other does as well. The units displayed by these two controls are in internal units and have no real meaning. On the left, there is a static control that displays the real cuttoff frequency in Hz. This is the initial cutoff frequency. The test frequency alsways starts at 1Hz. As the test frequency increases, the bandwidth of the low pass filter is increased in proportion, this is to help speed up the bode plot process.

**0 db Frequency**

This field is no longer used.

**Samples per decade**

This is the number of test frequencies per decade. The frequencies are spaced out logrithymlicly.

**Number of Decades**

This is the number of decades to run the bode plot. The only real useful values here are 3 or 4.

**Bode Plot Type**

Closed Loop:generates the closed loop response of the system.
Open Loop: generates the open loop response (more or less) of the system.
PID: generates the frequency response of the PID filter.

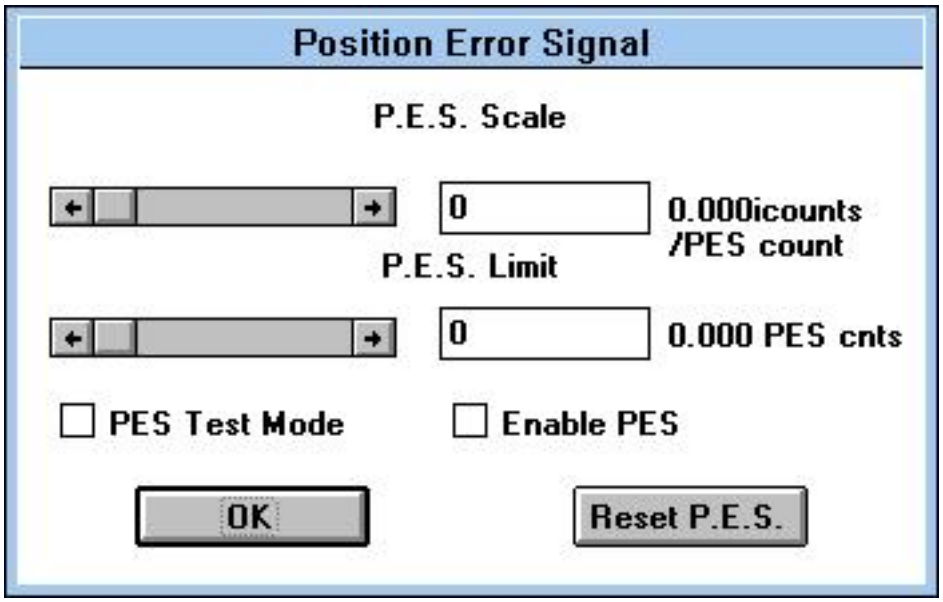
Notch0:Generates a frequency response of the Notch Filter
Notch1:Generates a frequency response of the other Notch Filter

Once the plots are made, you can switch between them by selecting the desired response.

**Start Test**

Starts the bode plot test.
It takes a while to run this test, go off and get a cup of coffee.
There is a known bug in this test. Sometime, especially the first time it is run, there is sometimes a divide by zero fault. Just restart the program, and try again. There is no need to turn off the spindle controller, in fact, this may cause the fault to occure the next time you run the test.

# Position Error Signal Dialog Box




This dialog box is for setting up the Position Error Signal hardware.

**P.E.S. Scale**

This field sets the scaling of the number of **iCounts** per PES count. The slider and edit box controls are connected together so that when one changes, the other changes as well.

**P.E.S. Limit**

This is the maximum limit on the number of PES counts.

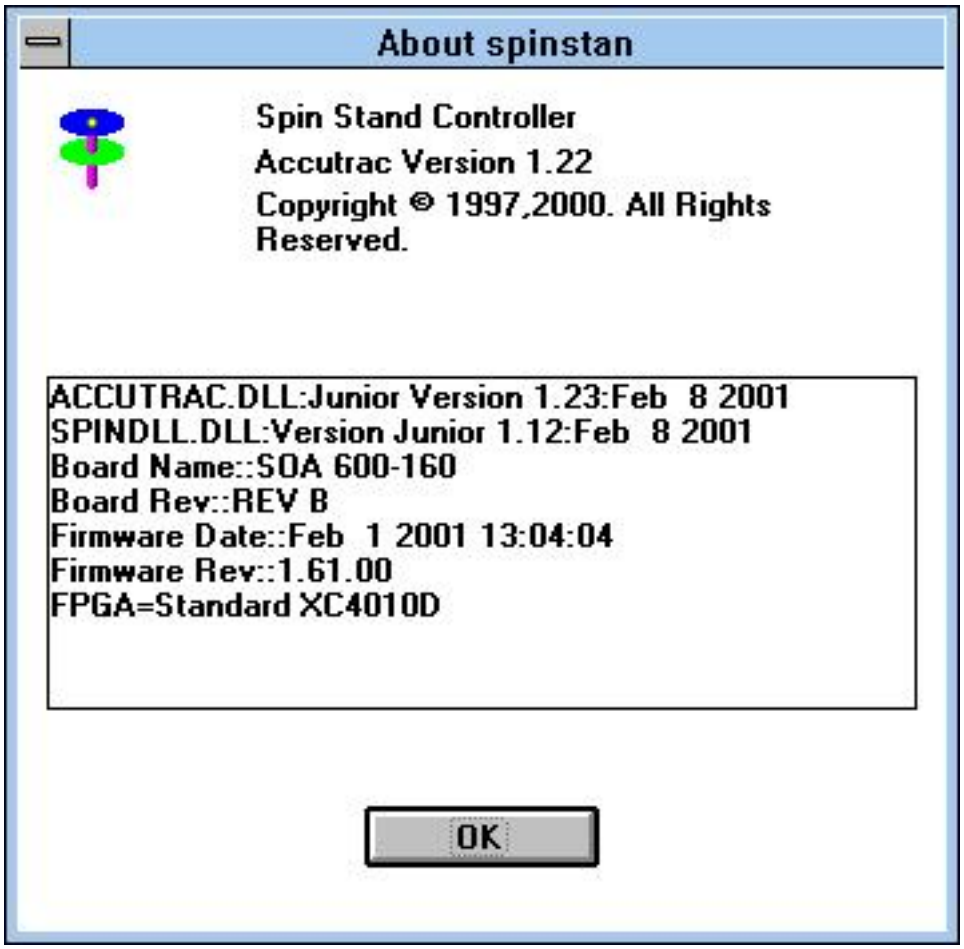
**PES Test Mode**

This enables the test mode in the hardware. The PES count is scanned back and forth so that the hardware can be checked.

**Enable PES**

This enables the PES mode. The PES counts are added to the current position when this box is checked.

# About Dialog Box




This dialog box shows a lot about the versions of the various components in the system.

**ACCUTRAC.DLL**

This is the version of the accutrac.dll file. The version is followed by the date that it was last compiled. The date is bound to be different despite the fact that the version number has not changed. This is due to the fact that from time to time, a fresh complete rebuild is done on the exectuables.

**SPINDLL.DLL**

This is the version of the spindll.dll file.

**Board Name**

This is the name of the controller board in the spindle controller.

**Board Rev**

This is the revision level of the artwork for the spindle controller PCB.

**Firmware Date**

This is the date and time that the firmware was last compiled.

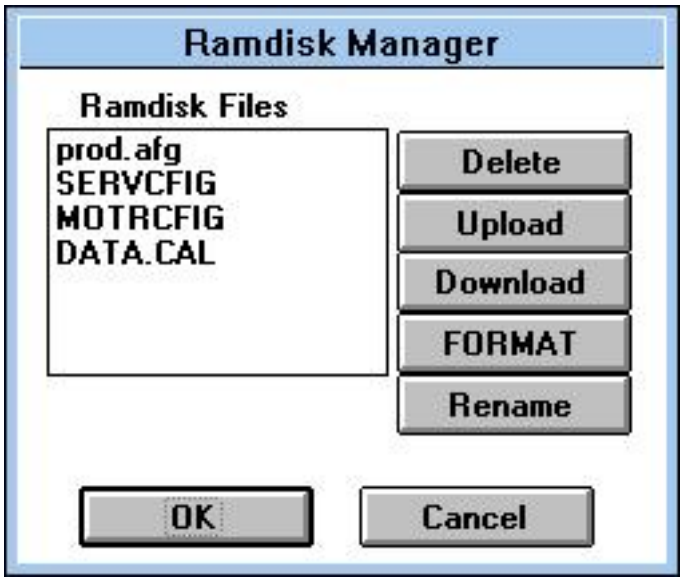
**Firmware Rev**

This is the revision level of the firmware.

**FPGA**

This is the gate array type used in the motion controller.

# Ramdisk Utilities Dialog Box




**Ramdisk Files**

This is a list of the files that are in the ramdisk that lives in the spindle controller.

**Delete**

If you highlight one of the ramdisk files and then push the Delete button, that file in the ramdisk will be deleted.

**Upload**

This is a way to get a file from the host computer to the ramdisk. Press the upload button and you will get a file dialog to select the file you wish to upload. The next dialog will ask you for a name that will be used to identify the file in the ramdisk. The rule for ramdisk names are that the name can be 8 characters long, and this means all characters, even periods (there are no extensions to file names in the ramdisk, but periods are valid). If you enter more than 8 characters, the name will be truncated.

**Download**

This is a way to get files from the spindle controller back into the host computer. You first highlight the file you wish to download, then press the download button. You will then get a dialog box to select the file name to download the data into.

**FORMAT**

This button will erase the ramdisk.

**Rename**

This button will rename a file on the ramdisk. Highlight the file to rename, then click the button. Remember the rules for naming files on the ramdisk (see above).

**System File Names**

### SERVCFIG

This is the data that the motion controller saves to the ramdisk. This data is uploaded to the DSP motion controller when the power is turned on. It is updated when you push the **Save Settings** button in the main dialog.

### MOTRCFIG

This is the data for the phase lock loop spindle motor controller. This is a backup copy. The primary data for the spindle controller is stored elsewhere.

### DATA.CAL

This is the calibration data for the rotary actuator encoder.

# How to tune the Rotary Actuator

*Created 6-6-2002*

Tuning the rotary actuator is more of an art, rather than a science, in many ways. You will learn only by doing. Knowing how feedback systems work is a real big help. There are many things that must be traded off, and, when things go wrong, you need to learn how to recognize what each one is so that you can take corrective action. Fortunately, there are really only two parameters that you must adjust, and only 5 that you should adjust, in order to get optimal performance. However, all of the controls interact with each other, to some degree, so, if you are not careful, you can end up chasing your tail. I will try to convey as much information as is posible here, but entire books have been written on this subject, so it is difficult to say how well I will do in a few pages.

Some of the information presented on this page is duplicated elsewhere. Links will be provided to that information.

---

## Objective

The objective to tuning can be expressed in one of two ways. You want to create a system that has a flat frequency response (frequency domain), or, you want to create a system that is critically damped (time domain). These are, it turns out, the same objective, just accomplished in different ways. Each methode has its advantages.

---

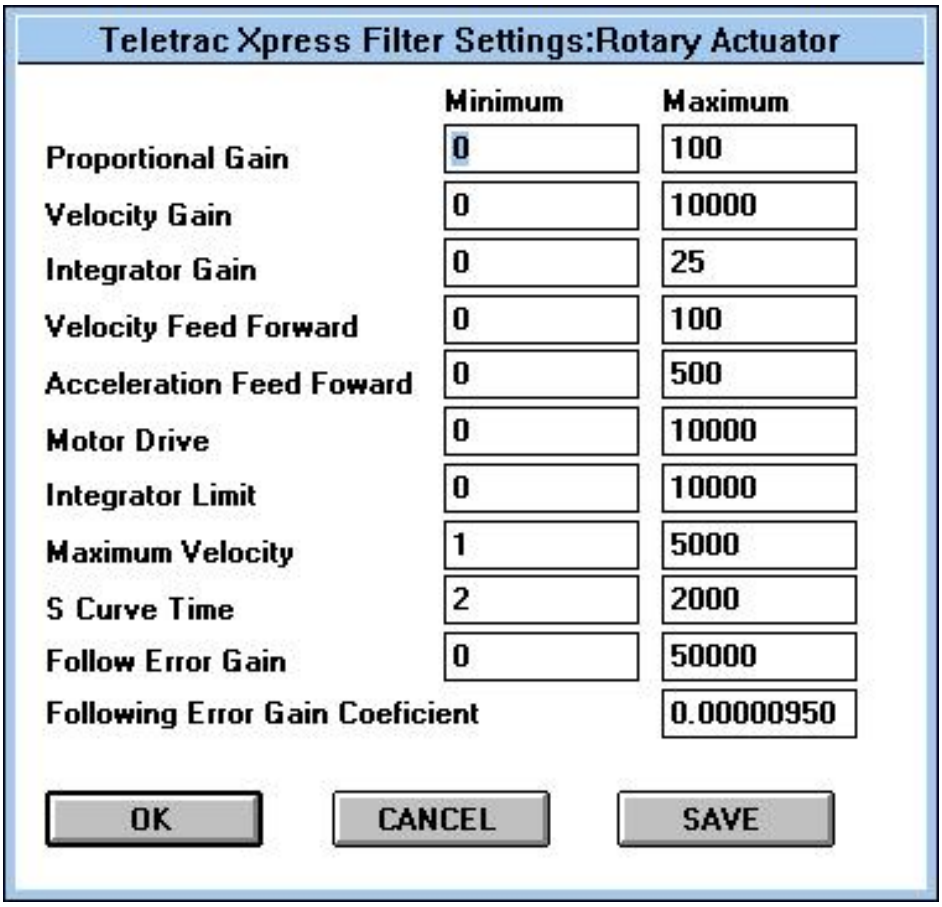
## Time Domain Tuning

The time domain methode is the fastest way to do the tuning. Setting up the spindle controller to do this test is not difficult, but it does require an osciliscope.
Almost any scope will do for this job. The only requirement is that it be DC coupled and have dual trace capability. A DSO is a good choice as the repitition rate of the signal is fairly low, about 5 times a second.

Connnect one channel of the scope to TP12 (sync) and the other to TP22 (following error). Set the scope to trigger on the channel connected to TP12.

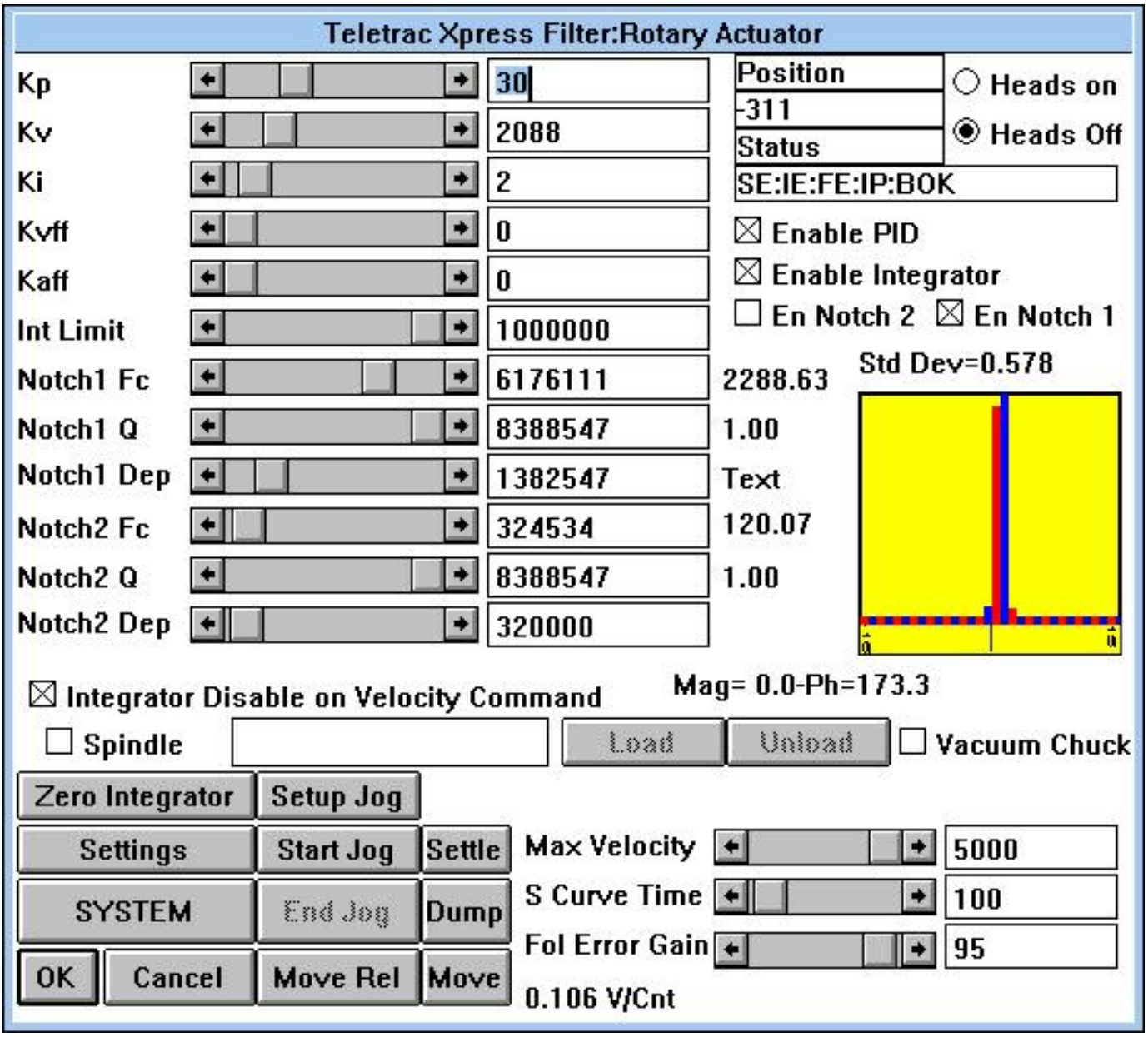
Setting up the spindle controller to do this test is going to be a little tricky. As I was writing this proceedure, I discovered that a change made in the firmware makes the following error gain come out different than it did for previous versions, but, this is easily fixed.

If you have Firmware Version 1.69.41, Spindll.dll Version 1.22, Accutrac.dll Version 1.35 or later, (bring up the About Box if you are not sure), you may want to check the filter settings dialog box.




To get to the filter settings dialog, you must be in the Fileter Dialog. There is a button labled Settings. The settings should look like the above box, especially the Follow Error Gain and the Following Error Gain Coeficient. The older following error gain coeficient had a value something like 0.001xxxx, and it will be rather obvious when you look at the scope displays that there is not a whole lot of detail. Below are a couple of examples of what you should be seeing when everything is set up and running.

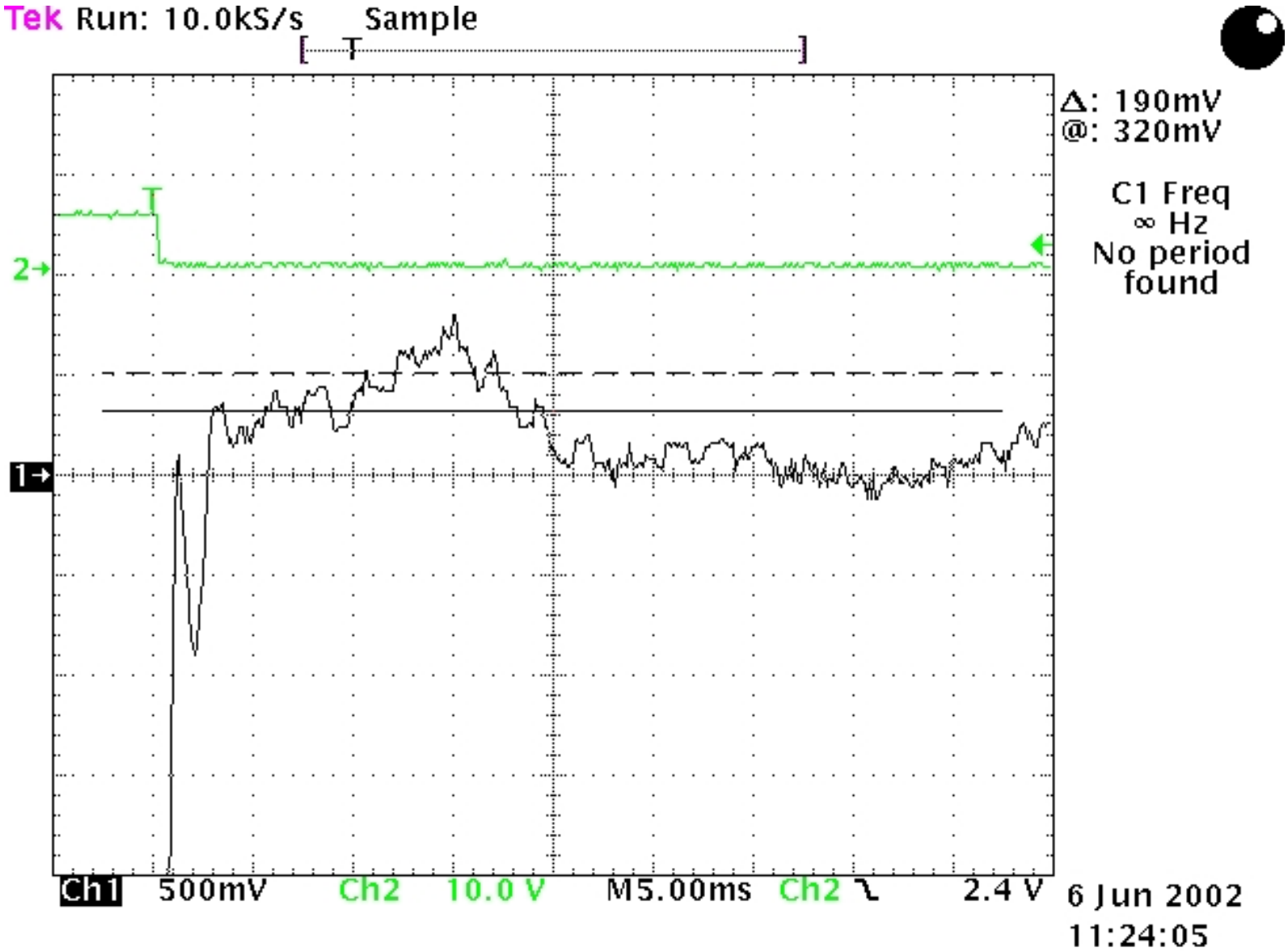
Now, Back to the filter dialog box.



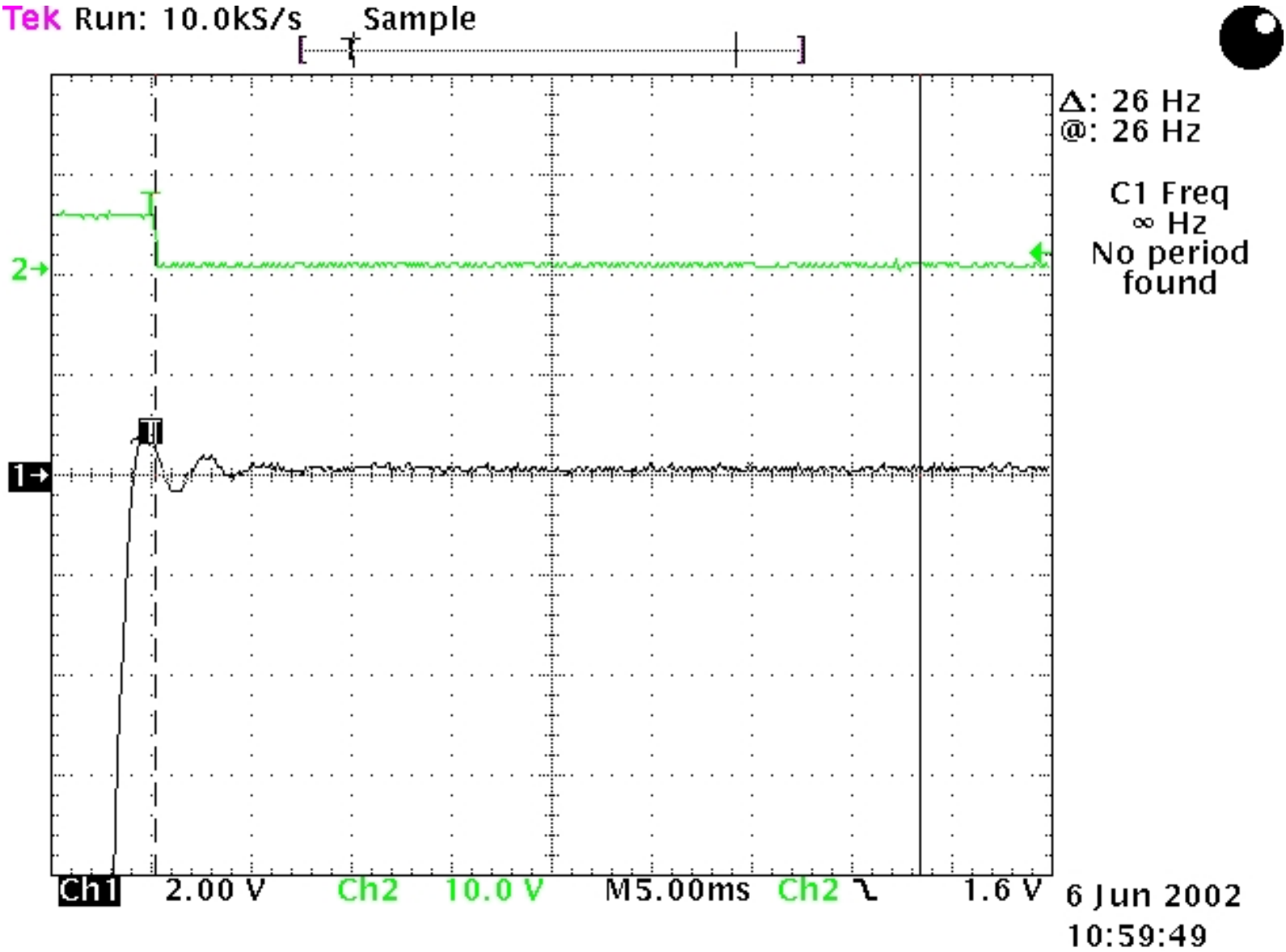

You are going to need to set up the Jogging. Click on the Setup Jog button. You will want to make the following settings. **Start** should be set to 10,000,000, **End** should be set to -10,000,000, **Index** should be set to 50,000 (if you know the track width, you may want to use this number instead. The step size has a profound effect on the step response), and **Dwell** should be set to 4000. Also, make sure the **Indexed Retrace** check box is checked.

When you click the **Start Jog** button, the rotary actuator will start to move. You should see something on your scope that looks like the screen shots below. Although, there will more than likely be a lot of ringing if the system has never been tuned before. If you grab the Kv slider and move it to increase the value of Kv, you will note that the ringning will gradually go away.

One thing you will note is that once you get it tuned more or less pretty good, each time it makes the jog it may look a bit different. Sometimes it will ring more, sometimes it will ring less. This seems to be due to mechanical differences from one position to the next. About all you can to is try to get a good average. In the scope photo right below, you will note two horizontal lines, the top one is dashed, the bottom one is solid. This is the spacing for 1 count.




Above is a critically damped system.



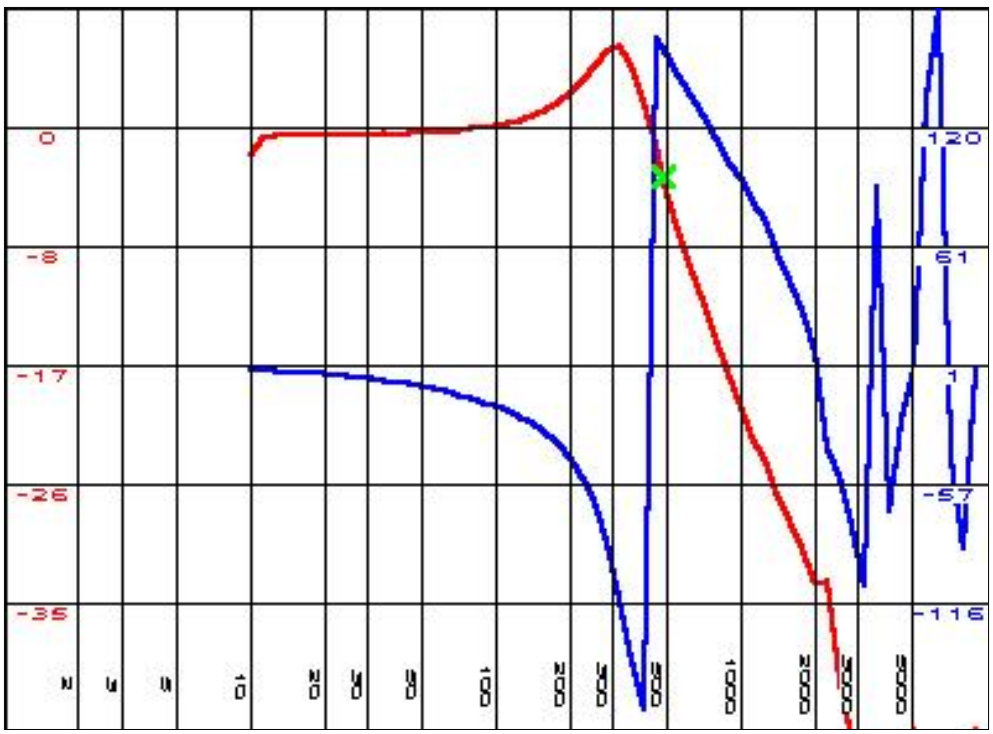

The above is under damped. The following error gain is much lower here, so it is not quite as pronounced.

---

## Frequency Domain Tuning

Even if you used the time domain tuning, you will also want to verify that the frequency response is flat as well. Sometimes the noise in the time domain displays can fool you into thinking it is better tuned than it is. So, to do a bode plot you will need to pull down the bode plot dialog box.


This first picture shows an underdamped response. There is about 4 dB of peaking in the response around 500Hz. It should also be noted that the Notch filter is turned off. You can see a small peak at about 1800Hz which can cause problems with high frequency oscilations. By setting the notch filter to about 1800Hz (in this case, other cases will differ), setting the Q to 1 and the notch depth to about 200,000, the plot below was obtained.




The plot above are the same parameters except Notch Filter 1 was enabled with the above described parameters. It has pretty much knocked that notch out of the picture and should not cause any more problem. Now, by advancing the derivative (velocity) gain, we can increase the damping even more. You should not how

the peaking has increased with the addition of the notch filter. Adding the notch filter adds two more poles and two more zeros to the transfer function, and they will interract with the other poles and zeros in the system.


This response is looking pretty good. The scope picture above of the critically damped system was made from the parameters used to make this plot.

---

## Tuning Proceedure

One thing you should note is that more than likely, no two systems will ever tune up exactly the same. Once you have tuned up one particular type of system, you can probably use those tuning parameters as a starting point that will be very close to what you need, but you will probably have to tweek the values a bit to get an optimal system. Any changes in tooling on the rotary actuator will drastically affect the tuning. Any time the tooling is changed, the system will have to be most likely retuned.

You will always probably have to start out with the frequency domain tuning, since you will need to find the first resonant peak so that the notch filter can be applied to it. Some of the newer motors have a very high Q resonance in them at about 5KHz.

So, the first thing we need to do is check the sample rate in the system dialog box. The default sample rate is 19531Hz. You will need to set this to 39062Hz. This is nessesary because a digital notch filter cannot go up to 5Kz with a 19531 sample rate.

Before you enable the PID loop, determine if the loop has ever been tuned before. If not, good starting values are Kp=20, Kv=4000,Ki=2. Kvff and Kaff should be zero. The notch filter should be off. Now, enable the PID loop and enable the Integrator. Hopefully, the system should now be servoing. If the actuator is making a very high pitched squeel, try to decrease Kv. If this doesn't help and the servo breaking into a very low frequency oscilation, we will have to try to make a best guess at the high frequency resonance.

Try setting Fc of Notch filter #1 to 5KHz, Q=1, and notch depth to about 300,000. Enable the notch filter, then enable the PID and integrator. Hopefully, it will now be quiet. You have a bit of a catch 22 here. In order to

make the system quiet, you need to notch out the resonance. In order to find the resonanc, you need to run a BODE plot. In order to run a BODE plot, they system must not be oscilating.....and so on.

After you have got the system more or less stabalizied, run a BODE plot. I would do a closed loop first. After you do the closed loop, it is also sometimes helpful to run the open loop response as well. Locate the first resonant peak. Depending on the motor, this can be anywhere from 2KHz to 7KHz. If the resonant peak proves to be a problem, you can play with the center frequency to home in on it a little better later. Return to the filter menu. Set the filter frequency and set the notch depth to about 150,000. Now it is time to set Kp and Kv.

It should be noted that while overly critical, you do have to be careful with the notch depth. Ideally, the notch depth should match the peaking in the resonance exactly. This however, is probably imposible. A notch depth of 0 is probably not a good idea. This will more than likely cause more problems than it cures.

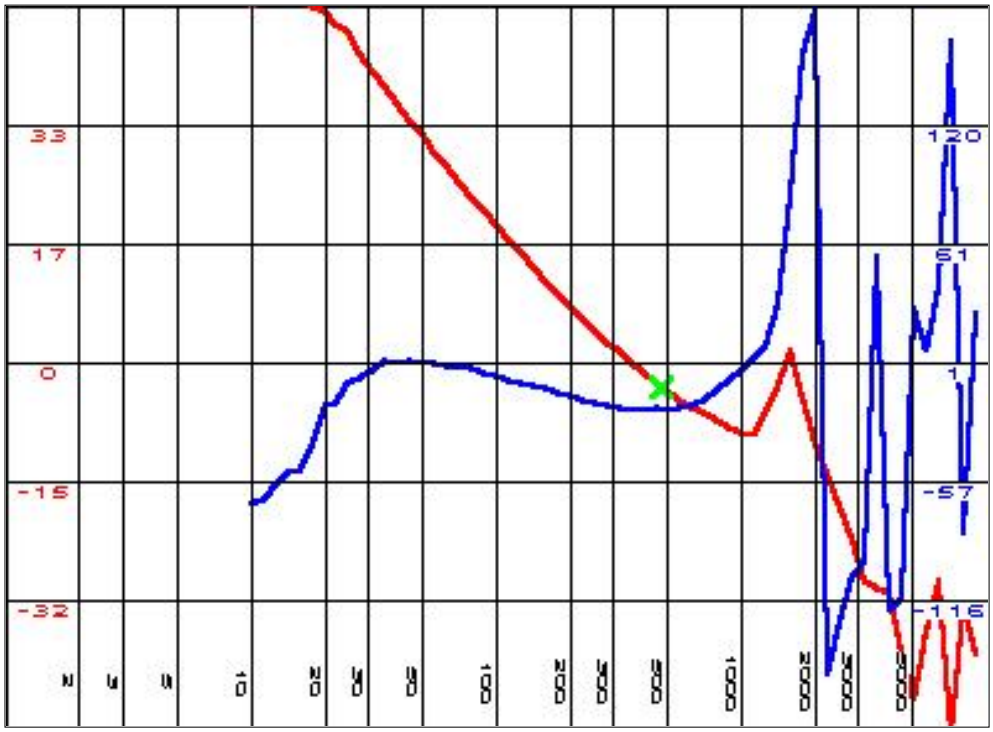
The goal is to adjust Kp as high as posible, however, just because you can get Kp to some level, does not mean that you want to use that value. In general, Kp will be in the range of 20 to 30. Kp has a direct affect of the bandwidth of the system. The higher Kp, the higher the bandwidth. In general, you should have no problem in getting a -3dB point of about 500Hz. So, start with 20. Adjust Kv until you can get the frequency response to be flat. If it is already flat, reduce Kv until it isn't and tune it back so that it is. Like I said, you want Kp to be as large as posible, but Kv needs to be as small as posible. After you have gotten the frequency response flat, run an open loop response. Check the level of the first resonant peak. It must be more than 6dB below the 0 dB line for the system to be stable. You might be able to play with the Notch center frequency to improve this a bit. If you have more than 15dB of margin, increase Kp by 5 and retune. For examples of what the open loop resonses look like, see Open Loop Response Plots below.

There will be several warning signs that Kp is too high. One will be the gain margin of the first resonant peak (must always be 6dB below 0 dB). Others are more subjective. If the frequency response just doesn't look good, that might be a sign that Kp is probably too high. If the time domain response and the frequency response don't seem to match each other, that could also be a sign. If the system works good one day and not the next, that could also be a sign.

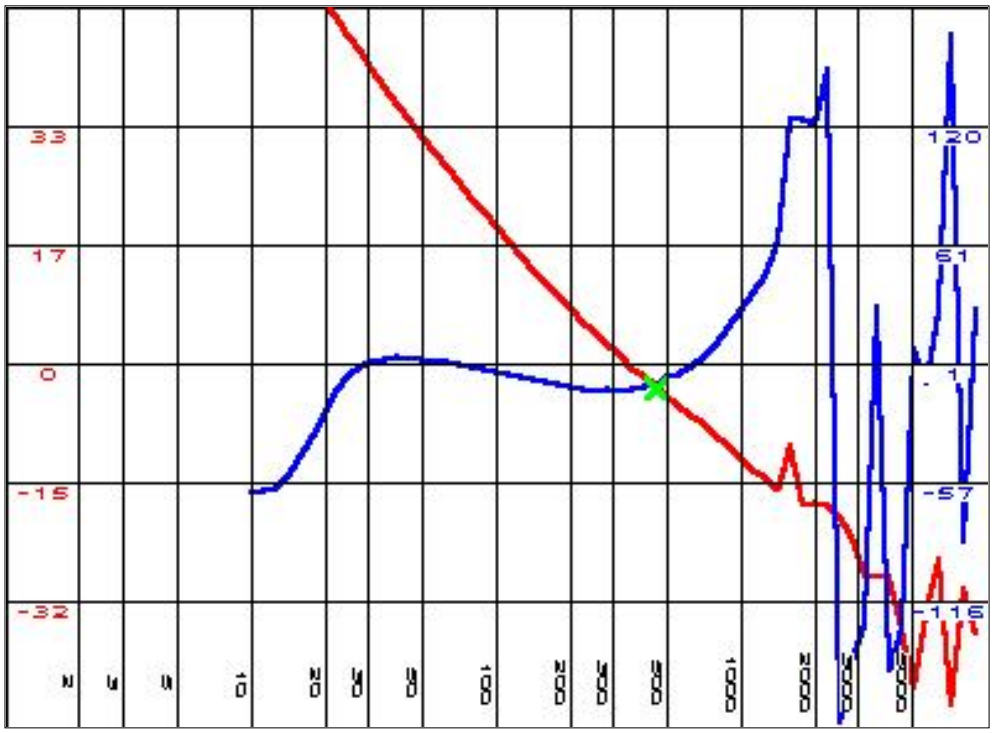
You should also note that there are a couple of radio buttons in the filter dialog box labled heads on and heads off. You should tune the rotary actuator under both of these conditions. Make sure that the vacume is on when you have the heads on. Also, use the OK button when you exit the filter menu and answer YES to the dialog box that asks if you wish to same the parameters to the product.ini file. This way, when the heads are being tested, the correct set of tuning parameters will be used by the spinstand at all times.

---

# Open Loop Response Plots




The above image is an example of an open loop response (the RED line is the gain). You will notice that at about 1600-1700 Hz, there is a sharp peak that coresponds to the first mechanical resonance of this particular motor. Please note that the newer motors generally have their first resonant peak at about 5000Hz. It should also be noted that the peak is not only greater than the 6dB gain margine, the gain actually crosses the 0dB line, which is an indication that the system is potentially very unstable.




This is the same system, and the same tuning parameters with the addition of the notch filter. The Filter Frequency (Fc) was set to about 1600Hz and the notch depth was set to about 130000. Notice how this brings the peak significantly below the 0 db line (the margin is about 10dB).

---

# Heads On/Heads Off

If you look at the **filter dialog** box, in the uppper right hand corner, you will notice a couple of radio buttons labled **Heads On** and **Heads Off**. What these do is set up tuning parameters for when the heads are on the chuck and when they are off the chuck. Some systems are so senitive, that they will start to oscilate when you take the heads off the chuck. They oscilate because of the fact that the resonant frequency shifts and is no longer properly compensated for.

How you set this up is select either the **Heads On** or **Heads Off** button. If you select **Heads Off**, it would, of course, be best that the heads were not on the chuck. To assist you there is a check box labled **Vacuum Chuck** so that when you do put the heads on, you can hold them on securely, or, when you want to get them off, you can release them.

So, you tune the system up, following the above proceedures. The **Heads Off** mode is not very critical. You are interested only in having a stable system under these conditions. You will never be doing any testing using the **Heads Off** parameters.

After you are done, you should be able to toggle the **Heads On** and **Heads Off** radio buttons and see the parameters change. Now you need to save these parameters. These are stored in the **product.ini** file. To do this, you **must** click on the OK button in the filter dialog box. A box will come up asking if you wish to save the data, and you must answer **yes**. A file dialog box will come up. Locate the **product.ini** file for the product that you are setting up and then click the OK button. These parameters are now saved in the product.ini file. You can verify this by exiting out of the program and starting over and selecting the product.ini file you saved the data in. When you enter the filter dialog, you should be able to click on the **Heads On** and **Heads Off** radio buttons and see the data you just saved.

---

# Misc

It is posible to calculate the Notch Filter Q and Center Frequency, however, it is a lot easier to use the Filter Dialog box to find out this information.

**Q = 8388608 / ( <Q Value>); *** <Q Value> is the internal number representing Q**

**Fc = <Fc Value> /( 8388608 * 2 * 3.14159 * Sample_Rate); *** <Fc Value> is the internal number representing frequency**

**Sample_Rate is the sample rate in Hz.**

**Adjustment Proceedure for the 600-160 Board**
**(This is for the REV A Version)**

The 600-160 Board has very few adjustments. None of them are truely critical, but, it does help to have these all properly adjusted. These adjustments are only posible to perfrom on a fully functional board, in other words, one that is fully communicating with a host computer via the RS-232 connection and using the **Spinstan.exe** software.

If the board is not communicating with the host, see below for trouble shooting.

The first thing we need to do is adjust the voltage reference (U46). This is done by connecting a volt meter to TP36 and adjusting R109. This voltage should be set to 6.25 volts +/- .01 volts. If this voltage differs widely from this value, or cannot be adjusted, check the components around U46 and U48A.

Next, we need to check the negative reference. This voltage is not adjustable. However, it should be close to –6.25 volts. The range on this can probably vary from -6.1 to –6.3 volts, depending on the tolerence of the components (Ideally, it will be –6.25 volts). If the voltage on this test point varies widely from the expected value, check the components around U74D.

Now, we are going to get into a much more tricky proposition. We need to adjust the offests on the spindle comutating DACs (U58), which will require the use Spinstan.exe. Also, it is a very good idea to disconnect the power to the 600-118 board (otherwise, damage could occure to that board).

The easiest way, currently, to adjust the offsets of U58 is to spin the spindle by hand. Clip a scope to TP28. You will need to bring up the **Spindle Setup** dialog box in the Spinstan.exe program. In the Edit box labled **Direct DAC** type in the value 127 and press the **Enter** key (the Enter key is what causes the data to be sent to the spindle controller). Next, in the box labled **Drive Source**, select the **CV** radio button.

Now, if you spin the spindle motor with your hand, you should see a sinusoid apear on the screen. Adjust R147 until the sinusoid is centered about zero volts. Repeat this proceedure for TP29 using R138 to adjust the offset voltage.

The last adjustments are not too tricky. We are going to be setting the DAC offsets for the motion controller outputs.

Connect a volt meter to TP38 (on the REV A board, this is Pin 9 of U31). Make sure that the Rotary Axis PID is disabled. Adjust R70 for zero volts.

Adjusting the offset voltage for the power amplifier is tricky. You will need to connect a load to the amplifier. The rotary motor will work, or, you can make a jig that has a 1 ohm resistor it in. Go into the filter menu and set the proportional gain to 0 (Kp).

Enable the PID loop. Connect a voltmeter from pin 5 to pin 8 of U94. Adjust R179 for zero volts.

## Trouble Shooting

Trouble shooting, under the best of conditions, is very difficult to do on the 600-160 card. On the REV A card, start with the modifications that should be made. Make sure that they are properly wired before turning the power on to the card. On the REV A board, there is one mod, that if it is not made, will cause damage to U47 and U104.

Pretty much, you are dependent on the card coming up the first time when it is powered up. Please note that it can take a considerable amount of time for this to happen, as the main CPU must initialize a lot of battery backed ram before the real work can begin (this takes about 5 minutes). However, if the LED status bar is present on the board, you should be able to see the lights go through their sequence as the system is booting.

Other common failures are static damage to the parts. The parts on the 600-160 board are very static sensitive. Even after the board is assembled, you must take precautions to prevent static discharge to the board. The MC56002 DSP is in particular very sensitive. The FPGA's are also fairly sensitive.

For new boards (especially the REV A), there are several places under the chips where solder bridges can occure. The REV B board, hopefully, will not have this problem. These can be very difficult to find and remove. But, in general, they have been removed succesfully in the past without having to remove any of the surface mount IC's.

On the rear of the card (the portion next to the rear panel), there is a RED led that indicates when the 68EC000 is in the HALT state. This state will only be acheived when there is a double bus fault. If the RED led remains lit, this is an indication that there is a serious hardware fault present (although, it is posible for a firmware fault to cause the same symptom, as far as is known at this time this situation should not arrise).