Using the 100-1008 Multisegment Envelope Generator

0. Before you build…

The following changes should be made. Many of the part values for the components are printed on the legend for the component side, which sure makes building the board easier, but some of them may have been changed. Please make the following changes.

Rev A Board

Reference Designator	Legend Says	Should Be
R4	499K	100K

1. Construction Options

There are several ways you can build your 100-1008 envelope generator. This is unfortunate in a way, because now you will have to make a decision. On the other hand, I like things to have a lot of flexibility, so you will be able to both suffer and enjoy this board.

a. Sequence Length

The length of the sequence is controlled by pins 2, 4 and 6 on connector JP7. Use the following table to configure the sequence length.

Pin 1 and 2	Pin 3 and 4	Pin 5 and 6	Length
Open	Short	Short	2
Short	Open	Short	3
Open	Open	Short	4
Short	Short	Open	5
Open	Short	Open	6
Short	Open	Open	7
Open	Open	Open	8

b. Sustain Segment

The sustain segment is the state that the envelope generator will hold at when the gate signal is true (gate > 1.4votls). This function is selected by using pins 12, 14 and 16 on connector JP7. Use the following table below to configure the sustain segment.

Pin 11 and 12	Pin 13 and 14	Pin 15 and 16	Sustain Segment
Open	Short	Short	1 (Attack)
Short	Open	Short	2

Open	Open	Short	3
Short	Short	Open	4
Open	Short	Open	5
Short	Open	Open	6
Open	Open	Open	7

One thing you must be careful about is that if you have a 4 segment length, you do not want the sustain to occur at segment 5, although, this will cause no harm, I am not sure how it will affect the operation…although, feel free to try. Who knows, something interesting may happen. I suspect that it will just go through all the segments without stopping until it gets to the release segment. However, if the gate is still true, it may just start all over again until the gate is released.

2. Front panel wiring.

The wiring for the front panel is not at all critical. If you look at the front panel wiring diagram supplied, this is just one option. In fact, I guess you could say that this is the full blown option. Well, maybe not quite full blown. If you note, on JP7 I have shown a three deck rotary switch to select the sustain level segment. The length inputs are left floating so the length is fixed at 8. However, you could also add a rotary switch to the length inputs as well and get a lot of flexibility in how the module will work.

You could also hard wire the length to say just four segments. In this case, you could eliminate half of the rate pots and half of the level pots. You could also make the sustain segment fixed as well. If you set it to say four segments in length, you could have the sustain segment be on segment 3. This would make the module sort of like a real fancy ADSR, or, more like an ADDSR.

Also, if you have fewer than 8 segments, you can wire up fewer LEDs, if you choose to have LEDs that is. One thing to note, LED0 (pin 2 of JP2) is always the LED that indicates the Release State. And Led1 (pin 4 of JP2) always indicates the Attack State.

JP1 Pin Definitions. These inputs control the voltage level each segment decays to.

JP1 Pin 2 (A0)	Release Voltage Level
JP1 Pin 4 (A1)	Attack Voltage Level
JP1 Pin 6 (A2)	Segment 2 Voltage Level
JP1 Pin 8 (A3)	Segment 3 Voltage Level
JP1 Pin 10 (A4)	Segment 4 Voltage Level
JP1 Pin 12 (A5)	Segment 5 Voltage Level
JP1 Pin 14 (A6)	Segment 6 Voltage Level
JP1 Pin 16 (A7)	Segment 7 Voltage Level

JP3 Pin Definitions. These inputs control the rate each segment decays at.

JP3 Pin 2 (B0)	Release Rate Voltage
JP3 Pin 4 (B1)	Attack Rate Voltage
JP3 Pin 6 (B2)	Segment 2 Rate Voltage

JP3 Pin 8 (B3)	Segment 3 Rate Voltage
JP3 Pin 10 (B4)	Segment 4 Rate Voltage
JP3 Pin 12 (B5)	Segment 5 Rate Voltage
JP3 Pin 14 (B6)	Segment 6 Rate Voltage
JP3 Pin 16 (B7)	Segment 7 Rate Voltage

JP4 Pin Definitions.

JP4 Pin 2 (GATE)	Gate input to Trigger Envelope Sequence
JP4 Pin 10 (OUT_INV)	Inverted Envelope Output
JP4 Pin 12 (+10R)	+10 volts for Pots
JP4 Pin 14 (-10R)	-10 volts for Pots
JP4 Pin 16 (OUT)	Envelope Output

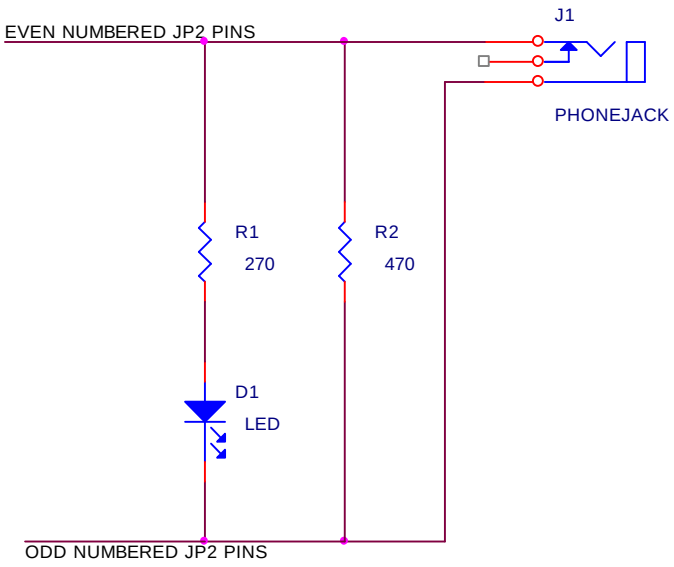
JP7 Pin Definitions. These inputs control the Length and Sustain segment.

JP7 Pin 2 (LEN0)	Bit 0 for Length
JP7 Pin 4 (LEN1)	Bit 1 for Length
JP7 Pin 6 (LEN2)	Bit 2 for Length
JP7 Pin 12 (SUS0)	Bit 0 for Sustain
JP7 Pin 14 (SUS1)	Bit 1 for Sustain
JP7 Pin 16 (SUS2)	Bit 2 for Sustain

JP2 Pin Definitions. These outputs are intended to light an LED indicating the current state.

JP2 Pin 2 (LEDO0)	Release State LED (red) (Anode)
JP2 Pin 4 (LEDO1)	Attack State LED (green) (Anode)
JP2 Pin 6 (LEDO2)	Segment 2 State LED (yellow) (Anode)
JP2 Pin 8 (LEDO3)	Segment 3 State LED (yellow) (Anode)
JP2 Pin 10 (LEDO4)	Segment 4 State LED (yellow) (Anode)
JP2 Pin 12 (LEDO5)	Segment 5 State LED (yellow) (Anode)
JP2 Pin 14 (LEDO6)	Segment 6 State LED (yellow) (Anode)
JP2 Pin 16 (LEDO7)	Segment 7 State LED (yellow) (Anode)

It should be noted that for JP2, the Cathode of the LEDs returns to the Odd pin numbers (which is ground). The user has some flexibility with JP2. It is possible to use it to generate a gate signal for each state. This can be done by replacing R24, R28, R33, R36, R38, R40, R42, and R44 with a short. You would then wire up to each pin of JP2 as shown below.



Gate Output Circuit

While this will involve making some labor intensive point to point wiring on your front panel (the parts can be mounted more or less on the Jack), this should provide a good solution if you want this functionality.

II. General Operational Considerations.

This is how the Envelope Generator works under normal conditions.

When the GATE signal goes TRUE, the STATE of the envelope generator goes to the ATTACK STATE (STATE1, STATE0 is the RELEASE STATE). The envelope generator will remain in the ATTACK STATE until one of two things happens:

1. The GATE signal goes FALSE, then Goto RELEASE STATE
2. The output voltage equals the ATTACK LEVEL voltage, then Goto STATE2

How long it takes for number 2 to happen depends on the ATTACK RATE control. If the time constant of the envelope generator is set to a long period, it will take longer than if the ATTACK RATE control is set to a short time constant. It should be noted that making the ATTACK RATE more positive will decrease (make faster) time constant.

The above process is repeated for each state, although an additional check is made to see if we are at the SUSTAIN SEGMENT. When the envelope generator reaches the SUSTAIN SEGMENT (which is set by the sustain bits on JP7), it will hold that state as long as the GATE signal is TRUE. When the GATE signal goes FALSE, it will then proceed until it reaches the end segment, which is set by the Length bits on JP7.

Until you get used to how this envelope generator actually works, you may have to remember to take a deep breath now and then before you get ready to through it out the window…speaking as the designer, I came close several time to doing the same thing. I am always trying to improve the device, and in the future, there may be different state equations for the PLD that controls this thing to make operation even smoother. But, as of right now, it seems to work about as smoothly as can be. But it does have quirks. On longer sequences, you

may note that it doesn't go through all of the states. Make sure first that the GATE signal is actually TRUE the entire time. If you have a setup where the full length is executed, and you see it go through ATTACK, 2, 3, 4, 5, the RELEASE (skipping 6 and 7), this could be due to several reasons…

1. The GATE signal went FALSE during STATE 5.

2. The Voltage Levels for STATE 6 and STATE 7 were the same (or very nearly so) to STATE 5. Under this condition, the envelope generator considers itself to have settled and will go onto the next state.

3. The ATTACK RATE is very fast.

It does take a bit of care to make sure you actually have the envelope generator set up to do what you really want it to.

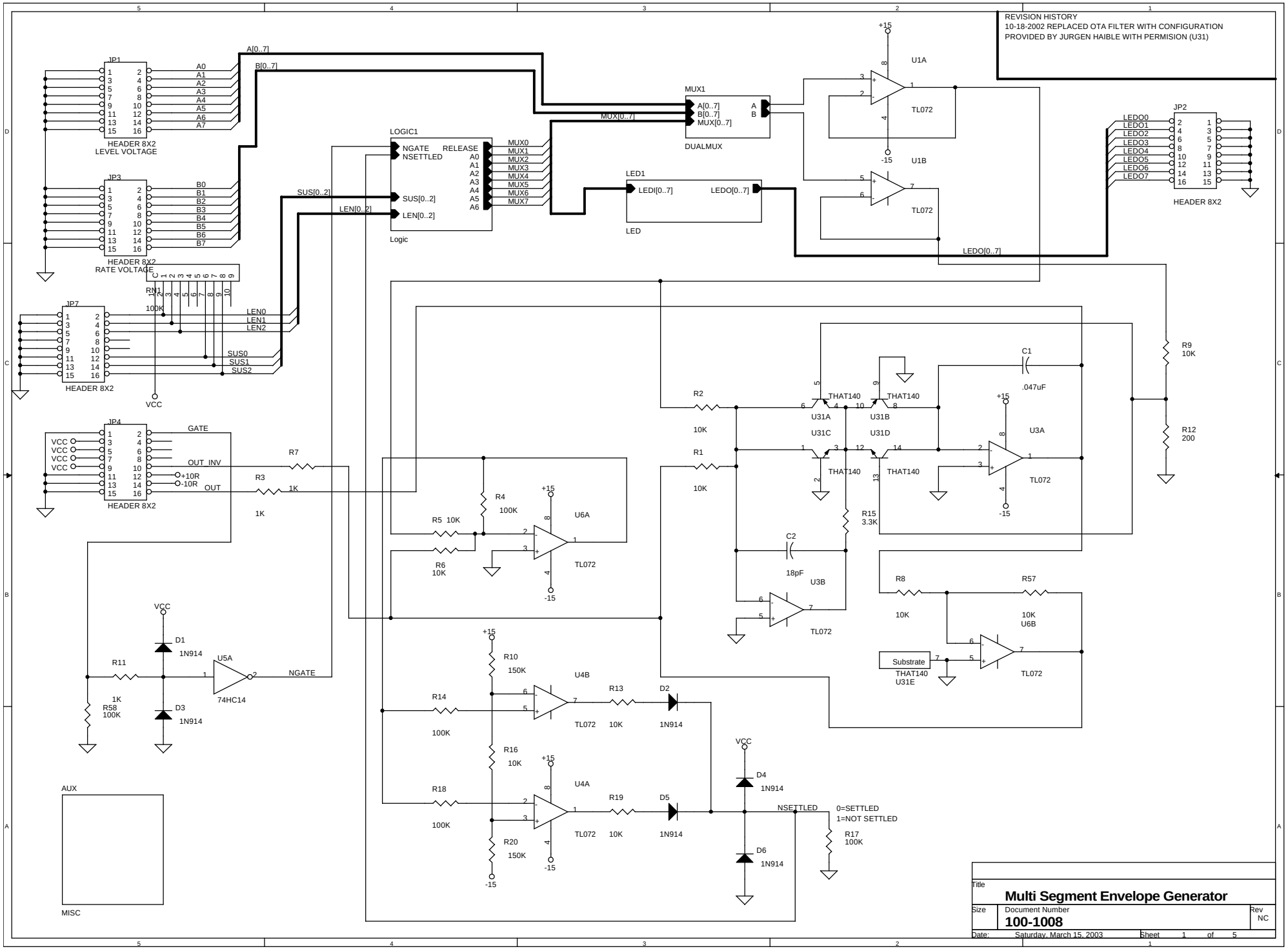
III. Assembling the PC board

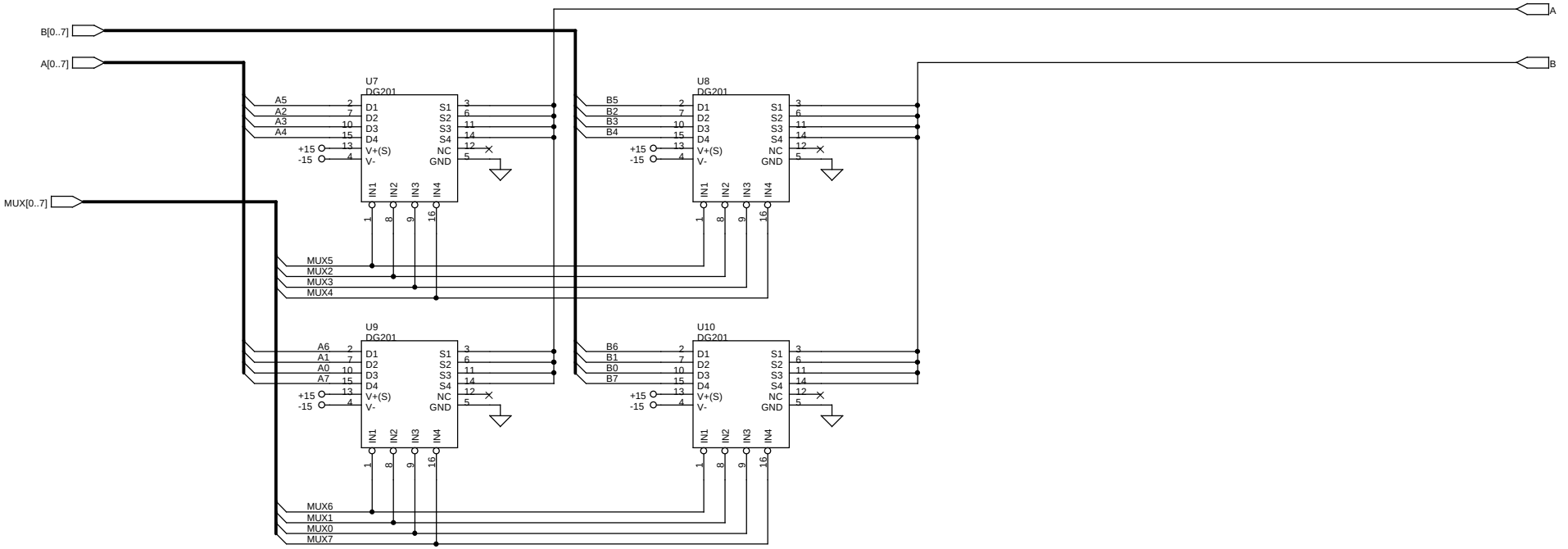
I admit it, there are a lot of parts on this little board. If you have the REV A board (first released in March of 2003), it should have come with the hard to find parts. Even at the release date, the THAT140 was no longer being produced, and only 25 of the REV A boards were made. So, hopefully, you have not lost either the THAT140 or the 22V10 that came pre programmed with the code for running the Envelope Generator. If you have lost the THAT140 for the REV A board, not all is lost. You can substitute matched sets of 2n3904 and 2n3906 for the pairs of NPN and PNP transistors, respectively. But I would not do this if you have the THAT140.

The 22V10 can be more easily replaced. The only thing special about it is the code programmed into it. If you have means of programming a 22V10, you should be able to download the code from the website and burn a new on. The 22V10 that came with the board is flash programmable so you can also update later if you have the means to program it.

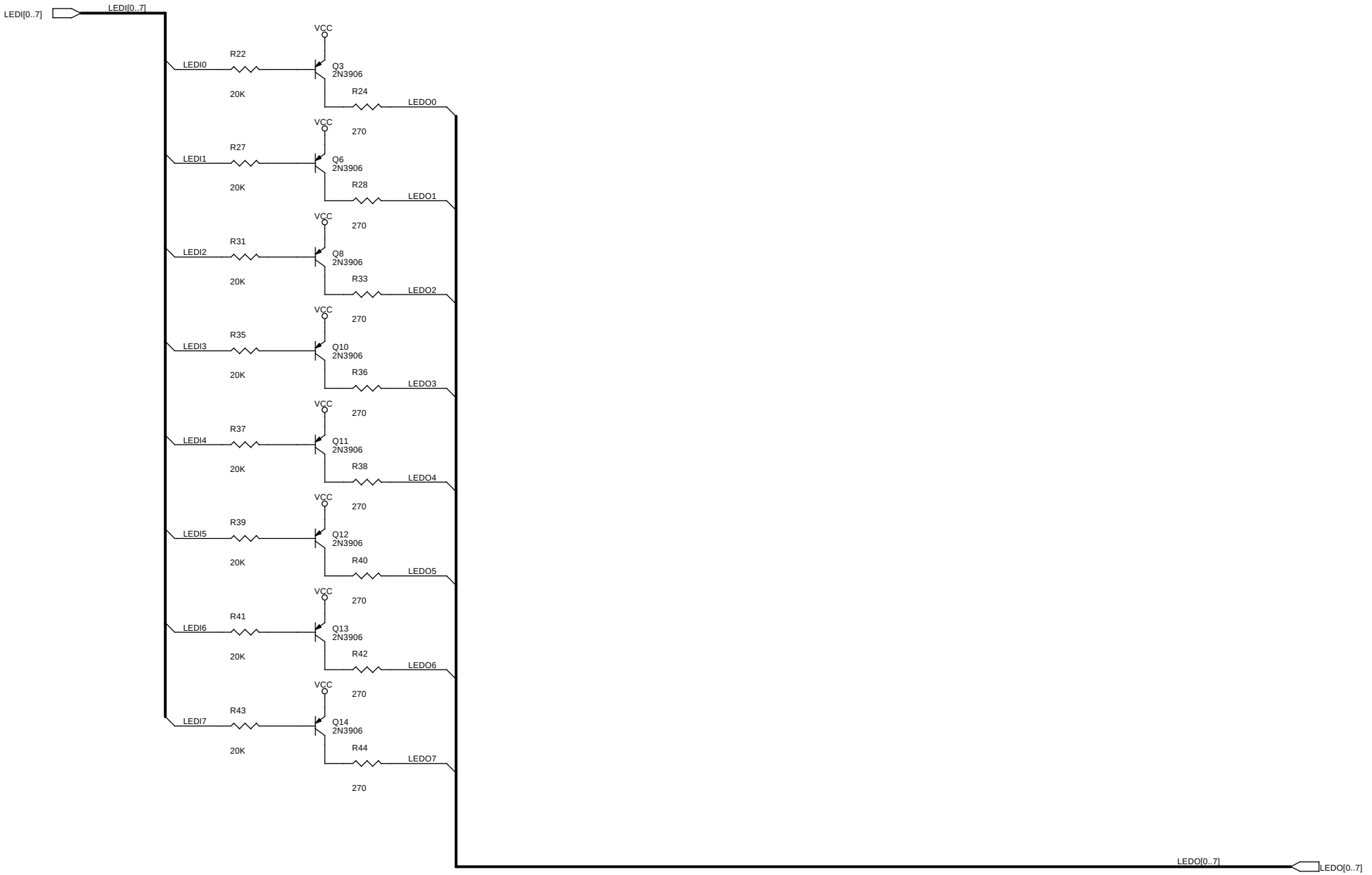
The board itself is a high quality double sided plated through PC board. User your favorite solder and soldering iron to construct the board. A temperature controlled iron will improve the way the solder flows, but it is not required. I would also highly recommend the use of 63/37 Tin/Lead solder. 60/40 will do in a pinch, but you will get better solder joints with the 63/37.

Solder flux is also a big issue. If you can, use a high quality solder with a flux core. Kester solder is what I generally use. Read the instructions on the solder about cleaning. Most solder flux is actually cleaning optional. Cleaning Rosin type flux can be a pain as it requires chemicals that are not very pleasant. Kester 331 organic core flux can be cleaned with water, however, it should be noted that you must be very diligent with this flux. You will note that Kester classifies this stuff as a MUST CLEAN flux, and they mean it. If you use Kester 331, you must clean the board at the end of the work session. Leaving it until the next day will not work. Kester 331 is very corrosive, however, used properly, you will end up with a nice clean board, something that is difficult to do with the rosin fluxes.

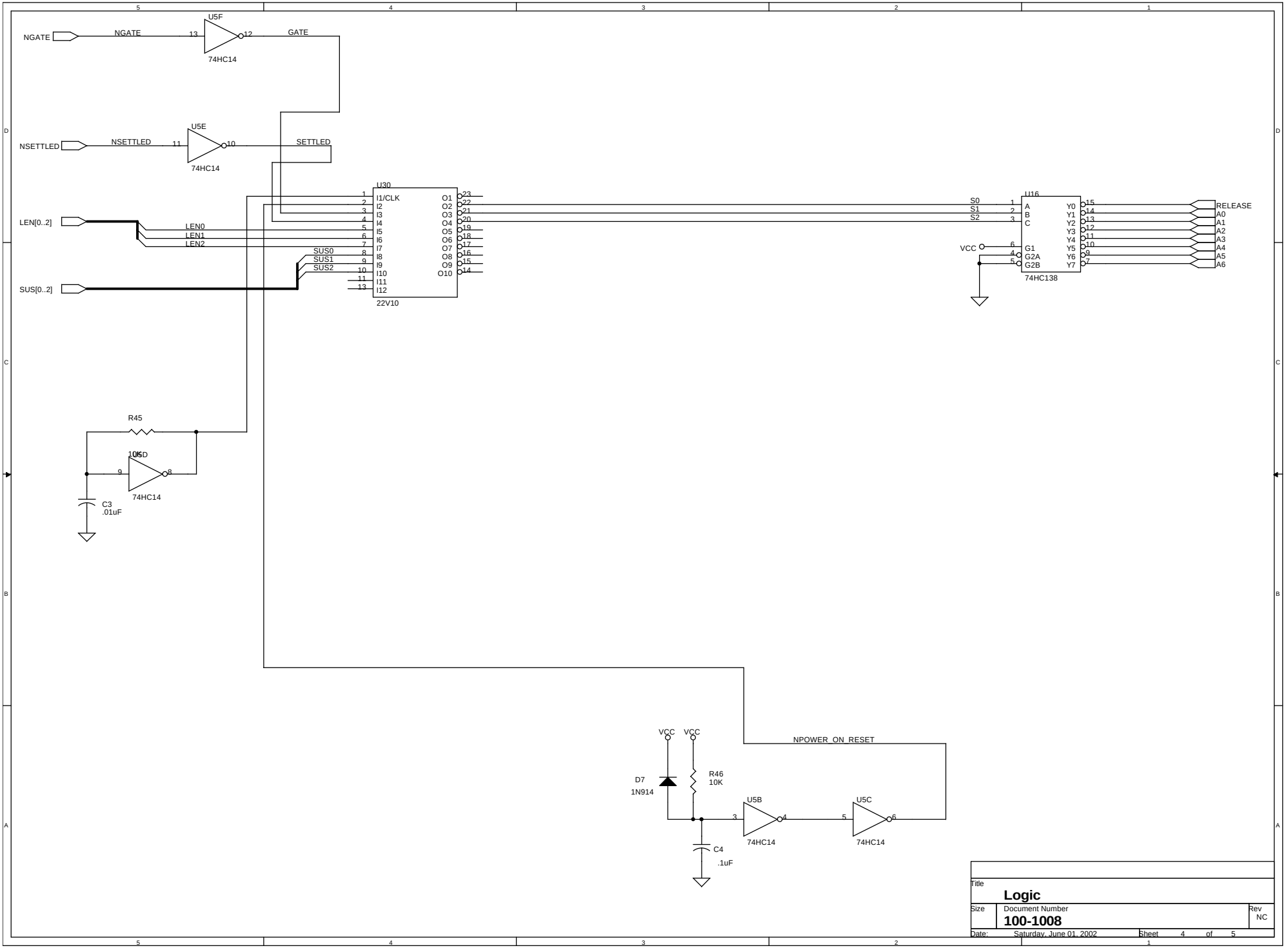

REVISION HISTORY
10-18-2002 REPLACED OTA FILTER WITH CONFIGURATION
PROVIDED BY JURGEN HAIBLE WITH PERMISION (U31)
JP1
HEADER 8X2
LEVEL VOLTAGE
A[0..7]
B[0..7]
JP3
HEADER 8X2
RATE VOLTAGE
LOGIC1
NGATE
NSETTLED
RELEASE
SUS[0..2]
LEN[0..2]
Logic
MUX1
DUALMUX
MUX[0..7]
LED1
LEDI[0..7]
LEDO[0..7]
LED
U1A
U1B
TL072
+15
-15
JP2
HEADER 8X2
LEDO0
LEDO1
LEDO2
LEDO3
LEDO4
LEDO5
LEDO6
LEDO7
JP7
HEADER 8X2
LEN0
LEN1
LEN2
SUS0
SUS1
SUS2
VCC
RN1
100K
JP4
HEADER 8X2
GATE
OUT_INV
OUT
+10R
-10R
R7
R3
1K
R2
10K
R1
10K
R4
100K
R5 10K
R6
10K
U6A
R9
10K
R12
200
C1
.047uF
U31A
U31B
U31C
U31D
THAT140
U3A
R15
3.3K
C2
18pF
U3B
R8
R57
10K
U6B
Substrate
U31E
D1
1N914
D2
D3
D4
D5
D6
R11
1K
R58
100K
U5A
74HC14
NGATE
R10
150K
R13
10K
R14
100K
R16
10K
R18
100K
R19
10K
R20
150K
U4B
U4A
NSETTLED
0=SETTLED
1=NOT SETTLED
R17
100K
AUX
MISC
Title
Multi Segment Envelope Generator
Size
Document Number
100-1008
Rev
NC
Date:
Saturday, March 15, 2003
Sheet 1 of 5

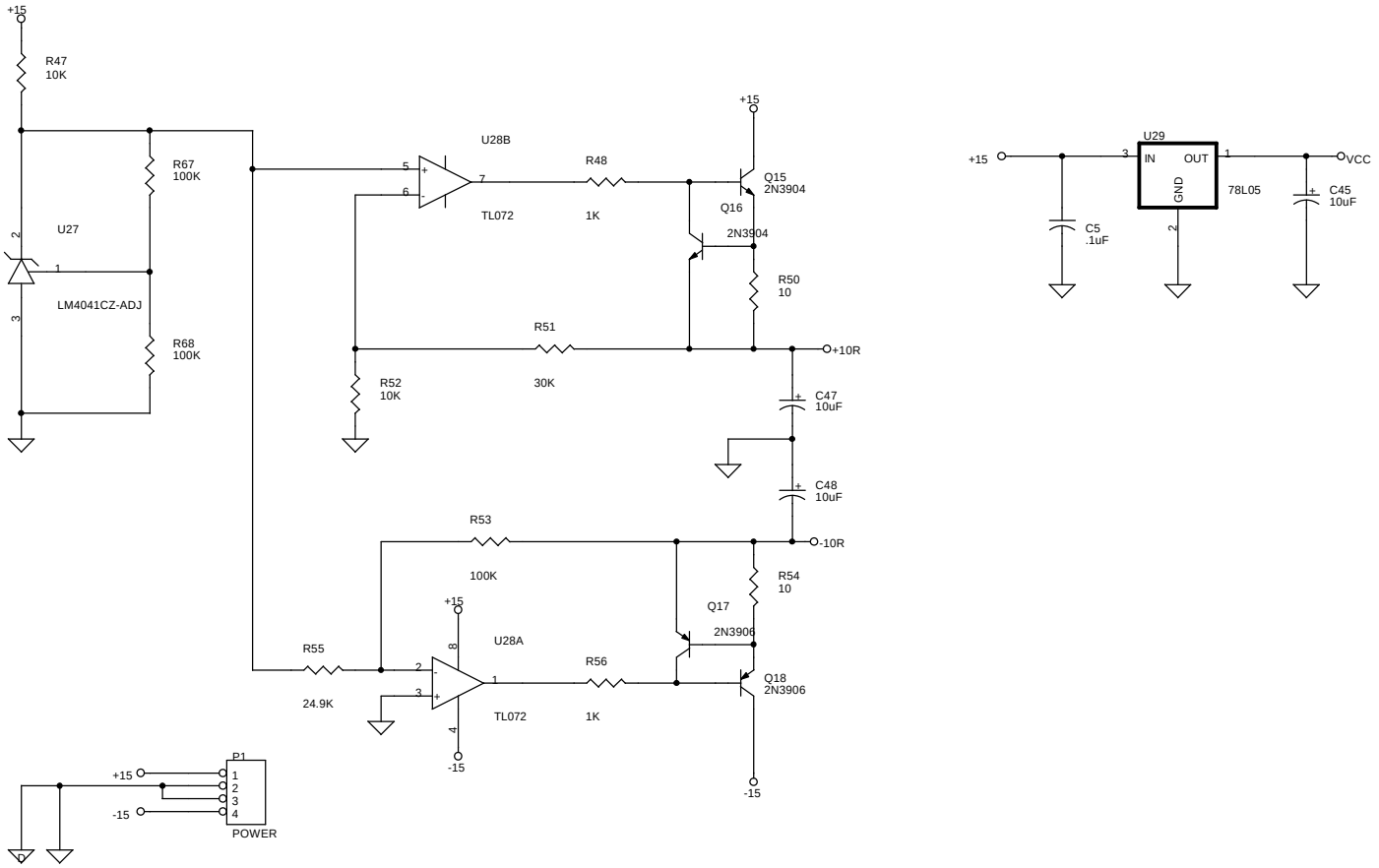


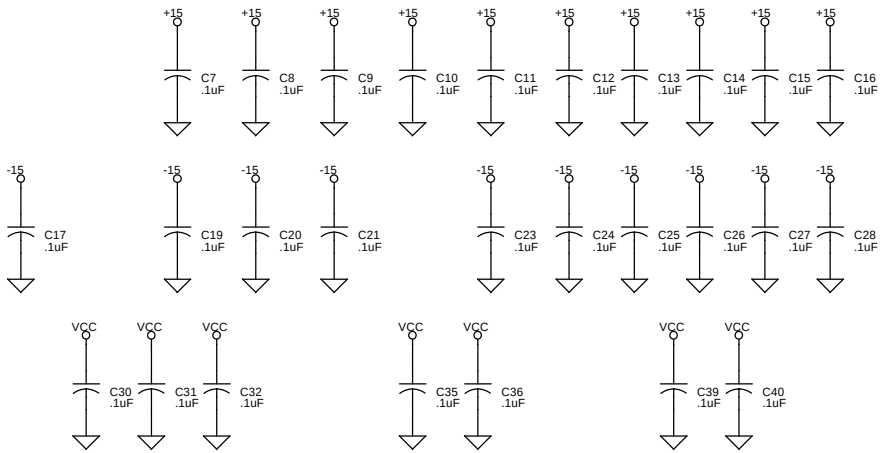
Title	**DUAL 8 TO 1 MUX**	
Size	Document Number **100-1008**	Rev NC
Date: Saturday, June 01, 2002	Sheet 2 of 5	



Title	**LED DRIVERS**	
Size	Document Number **100-1008**	Rev NC
Date: Thursday, May 30, 2002	Sheet 3 of 5	

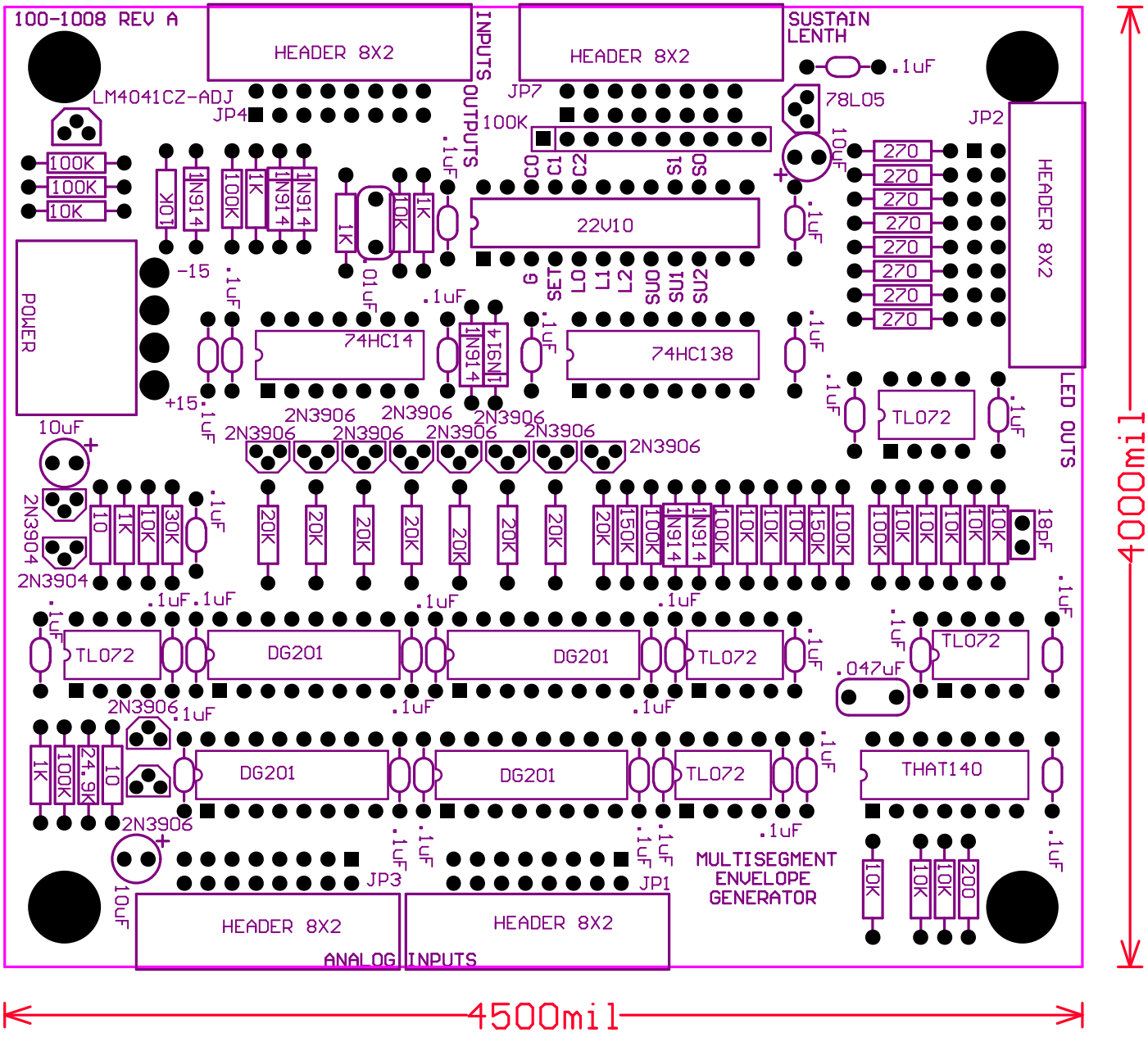

NGATE
NGATE
U5F
13
12
GATE
74HC14
NSETTLED
NSETTLED
U5E
11
10
SETTLED
74HC14
LEN[0..2]
LEN0
LEN1
LEN2
SUS[0..2]
SUS0
SUS1
SUS2
U30
I1/CLK
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
I12
O1
O2
O3
O4
O5
O6
O7
O8
O9
O10
22V10
S0
S1
S2
VCC
U16
A
B
C
G1
G2A
G2B
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
74HC138
RELEASE
A0
A1
A2
A3
A4
A5
A6
R45
10K
U5D
74HC14
C3
.01uF
VCC
VCC
D7
1N914
R46
10K
U5B
U5C
74HC14
74HC14
C4
.1uF
NPOWER_ON_RESET
Title
Logic
Size
Document Number
100-1008
Rev
NC
Date:
Saturday, June 01, 2002
Sheet
4
of
5





Title	**AUX CIRCUITS**	
Size	Document Number **100-1008**	Rev NC
Date: Friday, October 25, 2002	Sheet 5 of 5	

100-1008-6 REV A FAB DRAWING
100-1008-5 REV A COMPONENT LEGNED



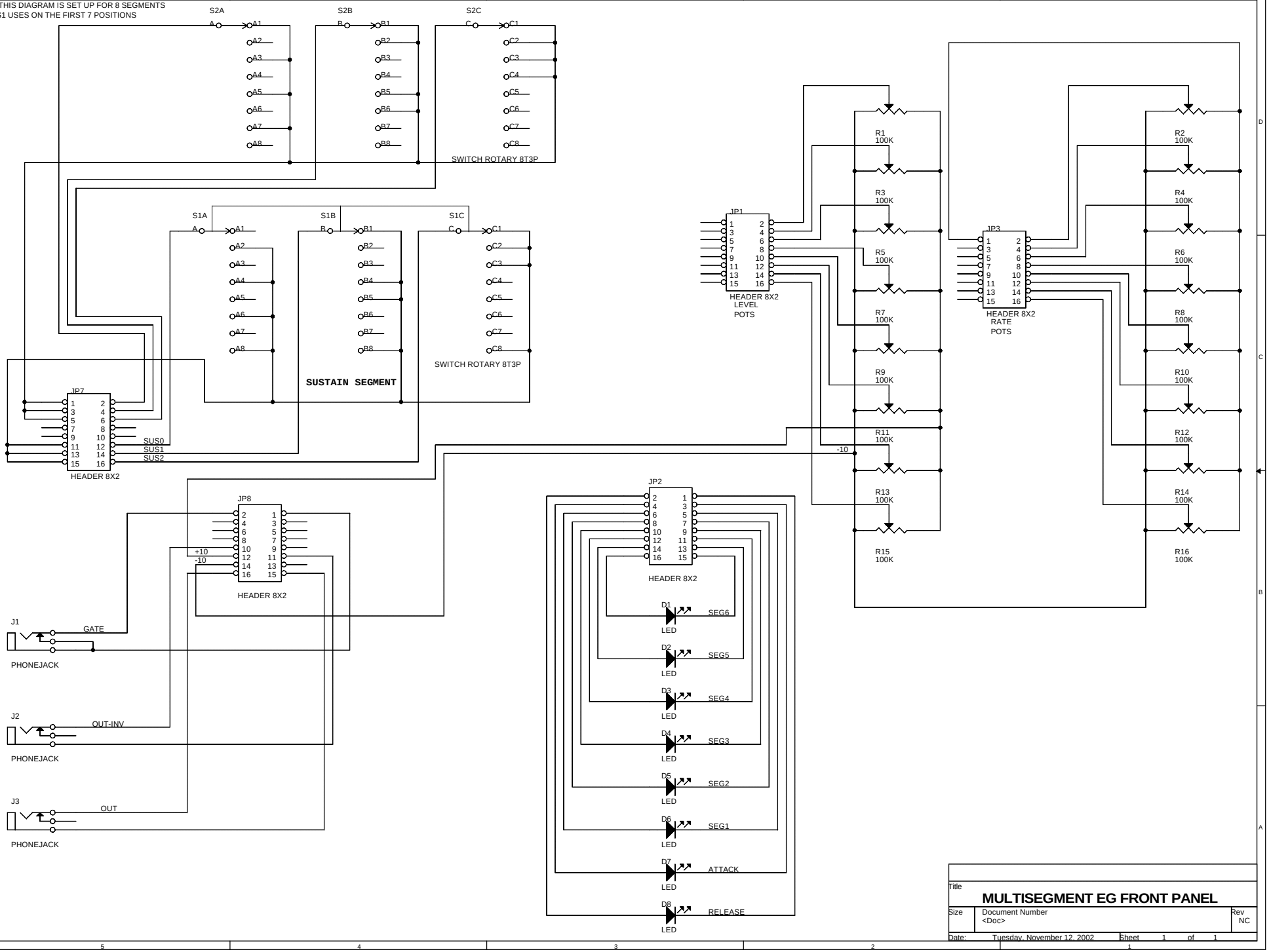
Multi Segment Envelope Generator Revised: Saturday, March 15, 2003
100-1008 Revision: NC

Jim Patchell
patchell@silcom.com

Bill Of Materials March 15,2003 16:16:12 Page1

Item	Quantity	Reference	Part	Part Number
1	1	C1	.047uF	23PS310
2	1	C2	18pF	
3	1	C3	.01uF	271-PF2A103J
4	29	C4,C5,C7,C8,C9,C10,C11, C12,C13,C14,C15,C16,C17, C19,C20,C21,C23,C24,C25, C26,C27,C28,C30,C31,C32, C35,C36,C39,C40	.1uF	80-C410C104M5U
5	3	C45,C47,C48	10uF	140-XRL35V10
6	7	D1,D2,D3,D4,D5,D6,D7	1N914	
7	5	JP1,JP2,JP3,JP4,JP7	HEADER 8X2	
8	1	P1	POWER	
9	10	Q3,Q6,Q8,Q10,Q11,Q12,Q13, Q14,Q17,Q18	2N3906	512-2N3906
10	2	Q15,Q16	2N3904	512-2N3904
11	1	RN1	100K	
12	14	R1,R2,R5,R6,R8,R9,R13, R16,R19,R45,R46,R47,R52, R57	10K	271-10K
13	5	R3,R7,R11,R48,R56	1K	271-1K
14	8	R4,R14,R17,R18,R53,R58, R67,R68	100K	271-100K
15	2	R20,R10	150K	271-150K
16	1	R12	200	271-200
17	1	R15	3.3K	271-3.3K
18	8	R22,R27,R31,R35,R37,R39, R41,R43	20K	271-20K
19	8	R24,R28,R33,R36,R38,R40, R42,R44	270	271-270
20	2	R50,R54	10	271-10
21	1	R51	30K	271-30K
22	1	R55	24.9K	271-24.9K
23	4	U1,U3,U4,U28	TL072	511-TL072CP
24	1	U5	74HC14	511-M74HC14
25	1	U6	TL072	
26	4	U7,U8,U9,U10	DG201	DG201ACJ
27	1	U16	74HC138	511-M74HC138
28	1	U27	LM4041CZ-ADJ	
29	1	U29	78L05	
30	1	U30	22V10	
31	1	U31	THAT140	

NOTE: THIS DIAGRAM IS SET UP FOR 8 SEGMENTS
NOTE:S1 USES ON THE FIRST 7 POSITIONS



Title	**MULTISEGMENT EG FRONT PANEL**	
Size	Document Number <Doc>	Rev NC
Date: Tuesday, November 12, 2002	Sheet 1 of 1	

```
-------------------------------------------------------------------------------
-
-- Top level for Multisegment Envelope Generator Controller
-------------------------------------------------------------------------------
-

library ieee;
use ieee.std_logic_1164.all;

library cypress;
use cypress.std_arith.all;

use work.compare_pkg.all;
use work.statemachine_pkg.all;
use work.counter_pkg.all;
use work.StateType_pkg.all;

entity multisegtop is
        port (
                CLOCK:in std_logic;
                RESET: in std_logic;
                COUNT: inout std_logic_vector(2 downto 0);
                GATE:in std_logic;
                SETTLED:in std_logic;
                MAX:in std_logic_vector(2 downto 0);
                HOLD:in std_logic_vector(2 downto 0);
--              MAXI:in std_logic;
--              HOLDI:in std_logic;
                MAXO:buffer std_logic;
                HOLDO:buffer std_logic;
                STATE:buffer std_logic_vector(1 downto 0)
                );
        attribute pin_numbers of multisegtop:entity is
                "CLOCK:1 "&
                "RESET:2 "&
                "GATE:3 "&
                "SETTLED:4 "&
                "MAX(0):5 "&
                "MAX(1):6 "&
                "MAX(2):7 "&
                "HOLD(0):8 "&
                "HOLD(1):9 "&
                "HOLD(2):10 "&
--              "MAXI:11 "&
--              "HOLDI:13 "&
                "COUNT(0):22 "&
                "COUNT(1):21 "&
                "COUNT(2):20 "&
                "STATE(0):15 "&
                "STATE(1):16 "&
                "MAXO:14 "&
                "HOLDO:23 ";
end multisegtop;

architecture archmultisegtop of multisegtop is
signal zerocount:std_logic;
signal enablecount:std_logic;
signal states:StateType;
begin
        u1:compare port map(a=>HOLD,b=>COUNT,c=>HOLDO);
        u2:compare port map(a=>MAX,b=>COUNT,c=>MAXO);
        u3:counter port
map(clk=>CLOCK,reset=>RESET,dout=>COUNT,enable=>enablecount,clr=>zerocount);
```

```
        u4:statemachine port
map(clk=>CLOCK,reset=>RESET,gate=>GATE,settled=>SETTLED,envend=>MAXO,hold=>HOLDO,inc
=>enablecount,clear=>zerocount,state=>states);
        STATE <= "00" when (states = idle) else
                 "01" when (states = attack) else
                         "10" when (states = sustain) else
                         "11";

end archmultisegtop;
```

```
--multi segment controller ic

library ieee;
use ieee.std_logic_1164.all;

package counter_pkg is
        component counter
        port (
                clk:in std_logic;
                reset: in std_logic;
                dout: inout std_logic_vector(2 downto 0);
                enable:in std_logic;
                clr:in std_logic
                );
        end component;
end counter_pkg;

library ieee;
use ieee.std_logic_1164.all;

library cypress;
use cypress.std_arith.all;
use cypress.lpmpkg.all;

entity counter is
        port (
                clk:in std_logic;
                reset: in std_logic;
                dout: inout std_logic_vector(2 downto 0);
                enable:in std_logic;
                clr:in std_logic
                );
end;

architecture counter_arch of counter is
begin
        sr:process(reset,clk)
        begin
                if(reset = '0') then
                        dout <= "000";          --reset the counter
                elsif (clk'event and clk = '1') then
                        if(clr = '1') then
                                dout <= "000"; -- clear counter
                        elsif(enable = '1') then
                                dout <= dout +1;                --increment counter
                        else
                                dout <= dout;   --hold
                        end if;
                end if;
        end process;
end counter_arch;
```

```
--------------------------------------------------------------------------------
------
-- State machine for multisegment envelope generator
--------------------------------------------------------------------------------
------

library ieee;
use ieee.std_logic_1164.all;

package StateType_pkg is
type StateType is (idle,attack,sustain,release);
end StateType_pkg;

library ieee;
use ieee.std_logic_1164.all;
use work.StateType_pkg.all;

package statemachine_pkg is
        component statemachine
        port (
                clk:in std_logic;
                reset: in std_logic;
                gate:in std_logic;
                settled:in std_logic;
                envend:in std_logic;
                hold:in std_logic;
                inc:out std_logic;
                clear:out std_logic;
                state:buffer StateType
                );
        end component;
end statemachine_pkg;

library ieee;
use ieee.std_logic_1164.all;

library cypress;
use cypress.std_arith.all;
use cypress.lpmpkg.all;
use work.statemachine_pkg.all;
use work.StateType_pkg.all;

entity statemachine is
        port (
                clk:in std_logic;
                reset: in std_logic;
                gate:in std_logic;
                settled:in std_logic;
                envend:in std_logic;
                hold:in std_logic;
                inc:out std_logic;
                clear:out std_logic;
                state:buffer StateType
                );
end;

architecture statemachine_arch of statemachine is
signal current_state,next_state: StateType;
begin
        state_clock:process(reset,clk)
        begin
                if(reset = '0') then
                        current_state <= idle;
```

```
        elsif (clk'event and clk = '1') then
                current_state <= next_state;
        end if;
    end process state_clock;
    state_comb:process(current_state,gate,settled,envend,hold)
    begin
        case current_state is
            when idle =>
                if(gate = '1') then
                    next_state <= attack;
                    inc <= '1';
                    clear <= '0';
                else
                    next_state <= idle;
                    inc <= '0';
                    clear <= '0';
                end if;
            when attack =>
                if(gate = '0') then
                    next_state <= idle;
                    inc <= '0';
                    clear <= '1';
                elsif (gate = '1') and (settled = '0') and (hold =
'0') then
                    next_state <= attack;
                    inc <= '0';
                    clear <= '0';
                elsif (gate = '1') and (settled = '1') and (hold =
'0') then
                    next_state <= attack;
                    inc <= '1';
                    clear <= '0';
                elsif (gate = '1') and (hold = '1') then
                    next_state <= sustain;
                    inc <= '0';
                    clear <= '0';
                else
                    next_state <= attack;
                    inc <= '0';
                    clear <= '0';
                end if;
            when sustain =>
                if(gate = '1') then
                    next_state <= sustain;
                    inc <= '0';
                    clear <= '0';
                elsif (gate = '0') and (envend = '1') then
                    next_state <= idle;
                    inc <= '0';
                    clear <= '1';
                elsif (gate = '0') and (envend = '0') then
                    next_state <= release;
                    inc <= '1';
                    clear <= '0';
                else
                    next_state <= sustain;
                    inc <= '0';
                    clear <= '0';
                end if;
            when release =>
                if(settled = '0') then
                    next_state <= release;
                    inc <= '0';
```

```
                clear <= '0';
            elsif(settled = '1') and (envend = '0') then
                next_state <= release;
                inc <= '1';
                clear <= '0';
            elsif(settled = '1') and (envend = '1') then
                next_state <= idle;
                inc <= '0';
                clear <= '1';
            else
                next_state <= idle;
                inc <= '0';
                clear <= '0';
            end if;
        end case;
    end process state_comb;
    state <= current_state;
end statemachine_arch;
```

```
--------------------------------------------------------------------------------
-- 3 bit comparator
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

package compare_pkg is
        component compare
        port (
                a:in std_logic_vector(2 downto 0);
                b:in std_logic_vector(2 downto 0);
                c:buffer std_logic
                );
        end component;
end compare_pkg;

library ieee;
use ieee.std_logic_1164.all;

library cypress;
use cypress.std_arith.all;
use cypress.lpmpkg.all;

entity compare is
        port (
                a:in std_logic_vector(2 downto 0);
                b:in std_logic_vector(2 downto 0);
                c:buffer std_logic
                );
end;

architecture compare_arch of compare is
begin
        c <= '1' when (a = b) else '0';
end compare_arch;
```