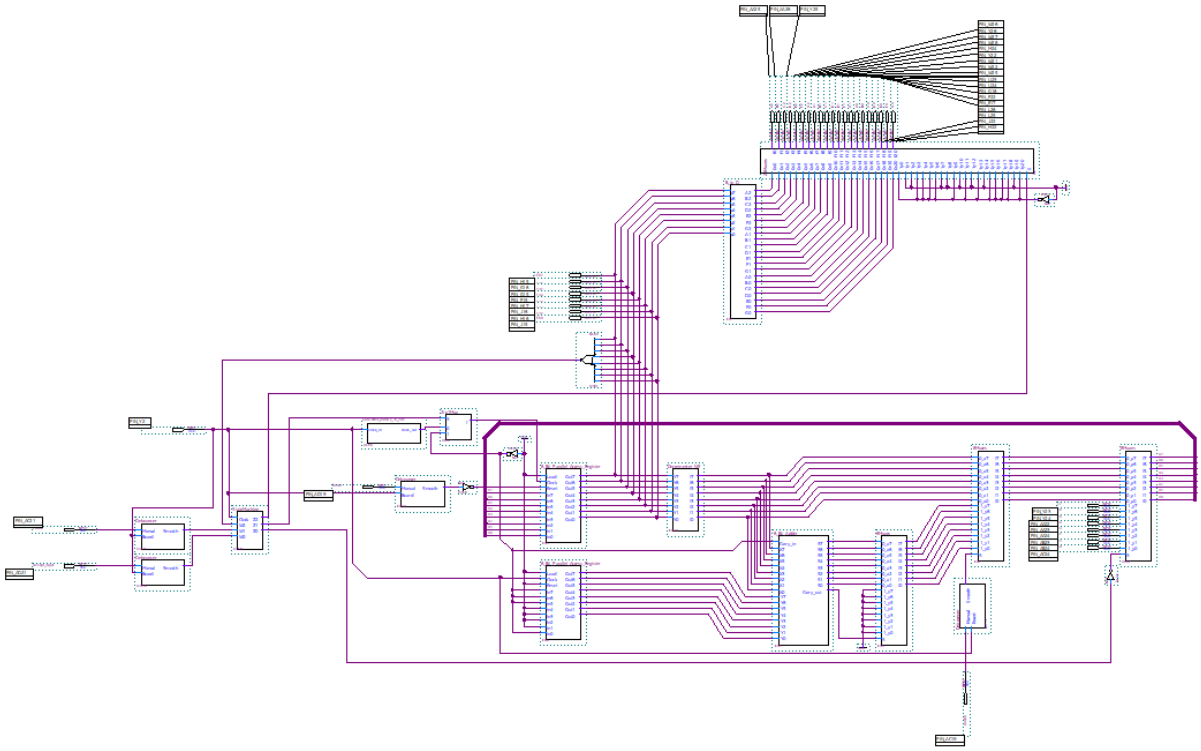
CprE 281 Final Project
8-bit Timer

Connor Hand and Zachary Scurlock



Lab Section 15

12/9/2022

Overview

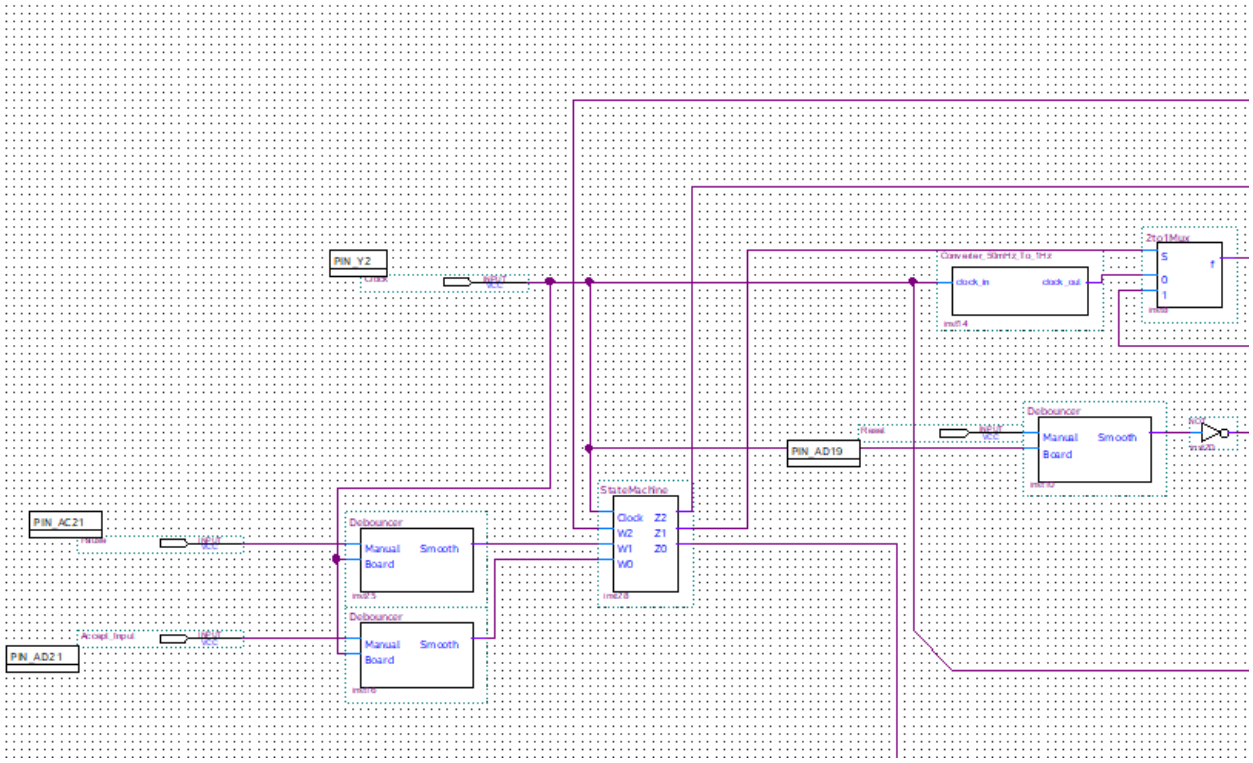
Our project is an 8-bit timer. The max value of the timer is 255 seconds. The timer includes a pause/play function, an add 30 seconds button, and a reset button. The register file consists of two 8-bit registers, one for holding the value of the timer at any given moment, and one for holding the value 30 for the add 30 seconds function. The state machine has four states, one for accepting input, one for counting down, one for pausing the timer, and lastly, one for when the timer reaches 0 seconds. The timer has four inputs for controlling the timer and eight inputs for setting the timer. The outputs include eight red LEDs that display the timer's current number in binary and three seven-segment displays that show the timer's current number in decimal.

Instructions

1. Be sure Rocker Switch 0 and 1 are both off when starting the timer. While Rocker Switch 0 is off use the leftmost eight slide switches to input a binary number of your choice. The three rightmost seven-segment displays should display your binary number in decimal.
2. To start the timer, switch Rocker Switch 0 to the on position.
3. To pause the timer, switch Rocker Switch 1 to the on position, and to unpause the timer, switch Rocker Switch 1 back to the off position.
4. To add 30 seconds to the timer, the timer must be counting down and unpaused. While the timer is counting down, hold down Push-button 1 (non-debounced) until the timer's display increases by 30 seconds.
5. To reset the timer, press down Push-button 0 (non-debounced).
6. Watch the timer count down to 0 and display "End".
   (Sometimes the non-debounced Push-buttons don't work very well or are stuck outputting 1 if any of the features don't work change the pin assignment to a different button or switch).

Components of the Timer

Section 1



This section of the timer contains the control logic (except for the add 30 seconds function). It includes the finite state machine, the clock divider, the debouncer from Lab 11, and a custom 2-to-1 MUX. It also includes the logic for pausing the timer.

The Finite State Machine

W_0 - accept input / count down

W_1 - pause / play

W_2 - counted down and reached 0

Z_0 - controls whether to accept input or count down

Z_1 - controls pause function

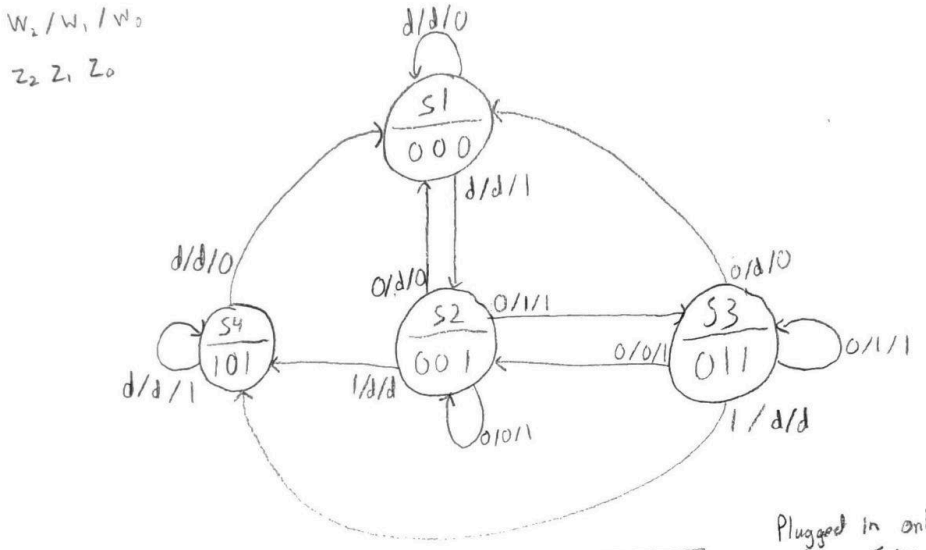
Z_2 - outputs End to seven segment display until user inputs new value

S1 - accepting input

S2 - counting down

S3 - paused

S4 - ended



Present State	$w_2w_1w_0$ = 000	001	010	011	100	101	110	111	Output z_2 z_1 z_0
00	S1	S2	S1	S2	S1	S2	S1	S2	0 0 0
01	S2	S1	S2	S1	S3	S4	S4	S4	0 0 1
10	S3	S1	S2	S1	S3	S4	S4	S4	0 1 1
11	S4	S1	S4	S1	S4	S1	S4	S1	1 0 1

Plugged in online instead of making 5-input K-maps.

$$Y_2 = \bar{W_2} W_0 y_2 y_1 + w_2 \bar{y_2} y_1 + w_2 y_2 \bar{y_1} + w_1 w_0 y_1 + w_1 w_0 y_2$$

$$Y_1 = w_0 \bar{y_2} \bar{y_1} + w_2 \bar{y_2} y_1 + w_2 y_2 \bar{y_1} + w_0 y_2 y_1 + \bar{w_1} w_0$$

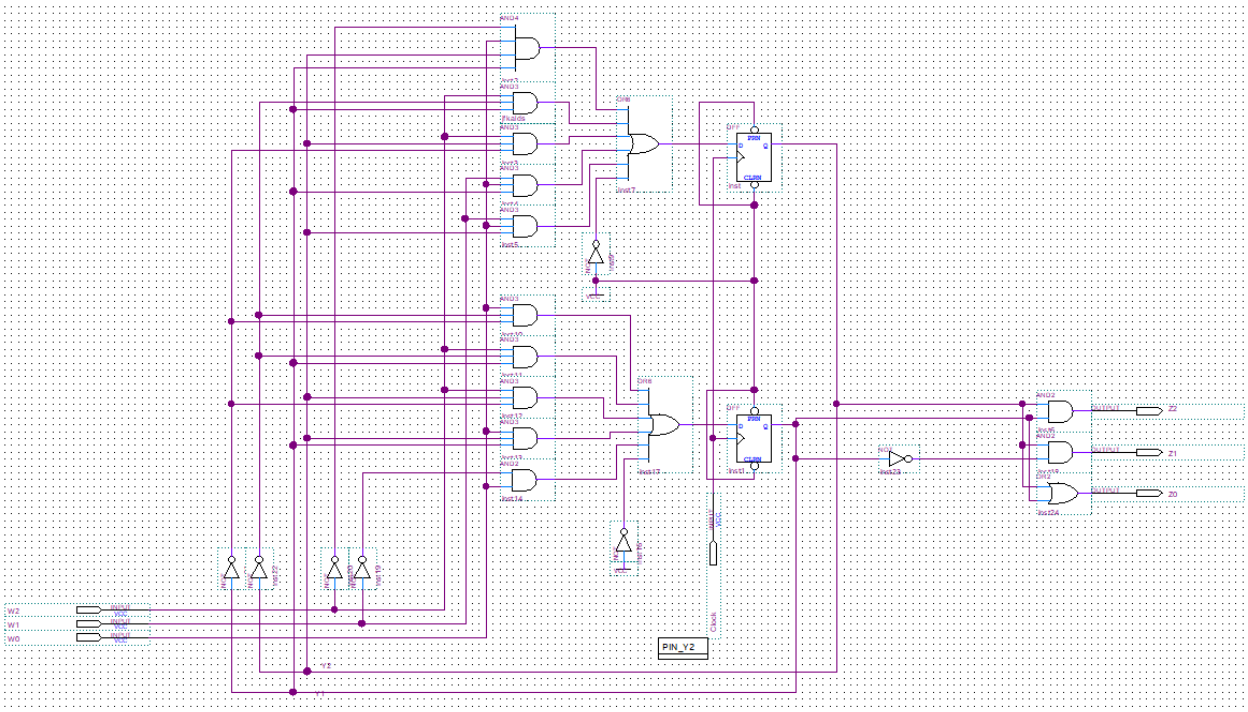
Present state y_2 y_1	$w_2w_1w_0$ = 000 (Y_2Y_1)	001 (Y_2Y_1)	010 (Y_2Y_1)	011 (Y_2Y_1)	100 (Y_2Y_1)	101 (Y_2Y_1)	110 (Y_2Y_1)	111 (Y_2Y_1)	Output z_2 z_1 z_0
00	00	01	00	01	00	01	00	01	0 0 0
01	00	01	00	10	11	11	11	11	0 0 1
10	00	01	00	10	11	11	11	11	0 1 1
11	00	11	00	11	00	11	00	11	1 0 1

y_2 y_1	z_2 z_1 z_0
0 0	0 0 0
0 1	0 0 1
1 0	0 1 1
1 1	1 0 1

$Z_2 = y_2 y_1$

$Z_1 = y_2 \bar{y_1}$

$Z_0 = y_2 + y_1$



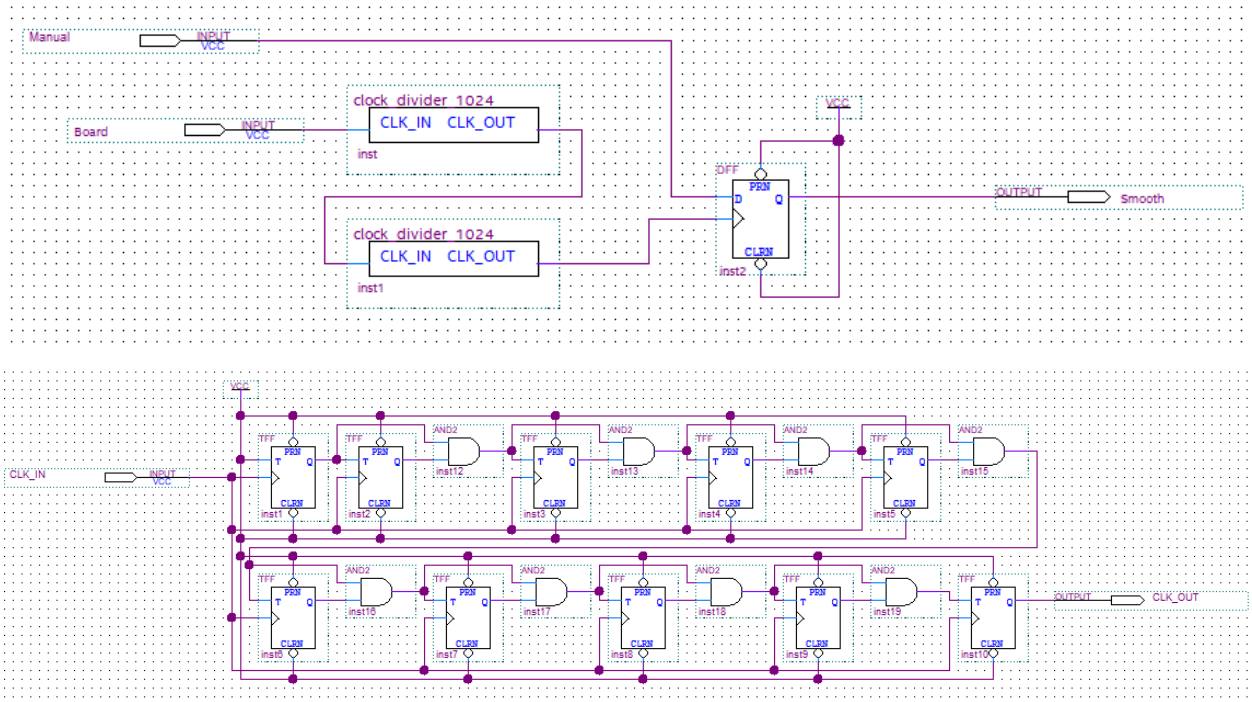
The finite state machine has three inputs, three outputs, and four states. The three inputs are for accepting input, pausing the timer, and displaying “End”. Two of the inputs come from switches and one of the inputs comes from when the value of the timer reaches 0. The three outputs control whether or not to accept user input, pause the timer, or display “End” to the seven-segment displays. All three of the outputs go to select lines of multiplexers elsewhere in the circuit. The four states have four different instructions. State 1 is for when the timer is accepting input, state 2 is for when the timer is counting down, state 3 is for when the timer is paused, and state 4 is for when the timer has reached 0 and it displays “End”.

The Clock Divider

```
1   module Converter_50mHz_To_1Hz(clock_in, clock_out);
2       input clock_in;
3       output clock_out;
4
5       wire clock_in;
6       reg clock_out = 0;
7
8       localparam div_value = 24999999;
9
10      integer counter_value = 0;
11
12      // counter
13      always@ (posedge clock_in)
14      begin
15          if (counter_value == div_value)
16              counter_value <= 0;
17          else
18              counter_value <= counter_value+1;
19          end
20
21       // clock divider
22       always@ (posedge clock_in)
23       begin
24          if(counter_value == div_value)
25             clock_out <= ~clock_out;
26          else
27             clock_out <= clock_out;
28          end
29       endmodule
```

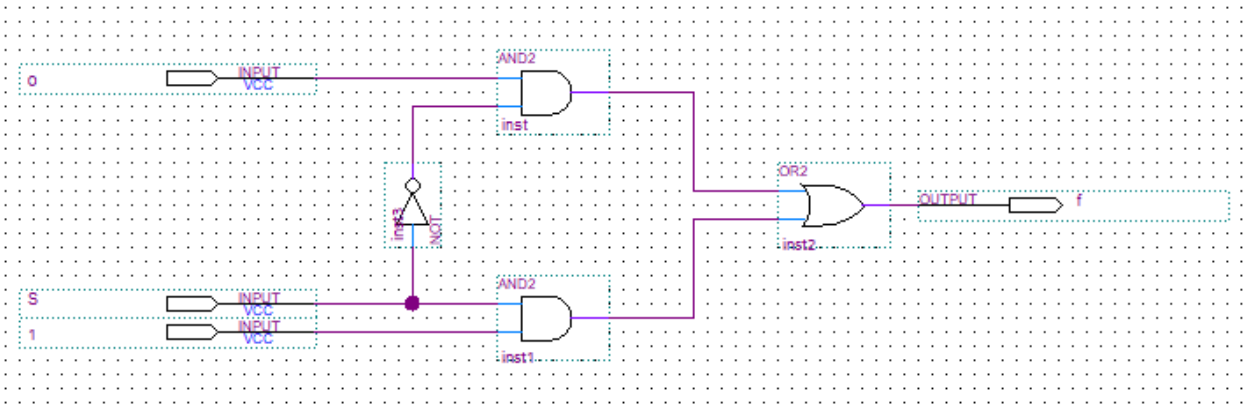
This clock divider converts 50 MHz to 1 Hz. It increases the variable counter_value by one for every positive edge of the 50 MHz clock. Once counter_value reaches div_value, clock_out is negated (div_value starts at 24,999,999 because counting starts at 0). This creates a clock that switches from 0 to 1 and then from 1 to 0 every half second, which makes every second a positive edge of the output clock.

The Debouncer



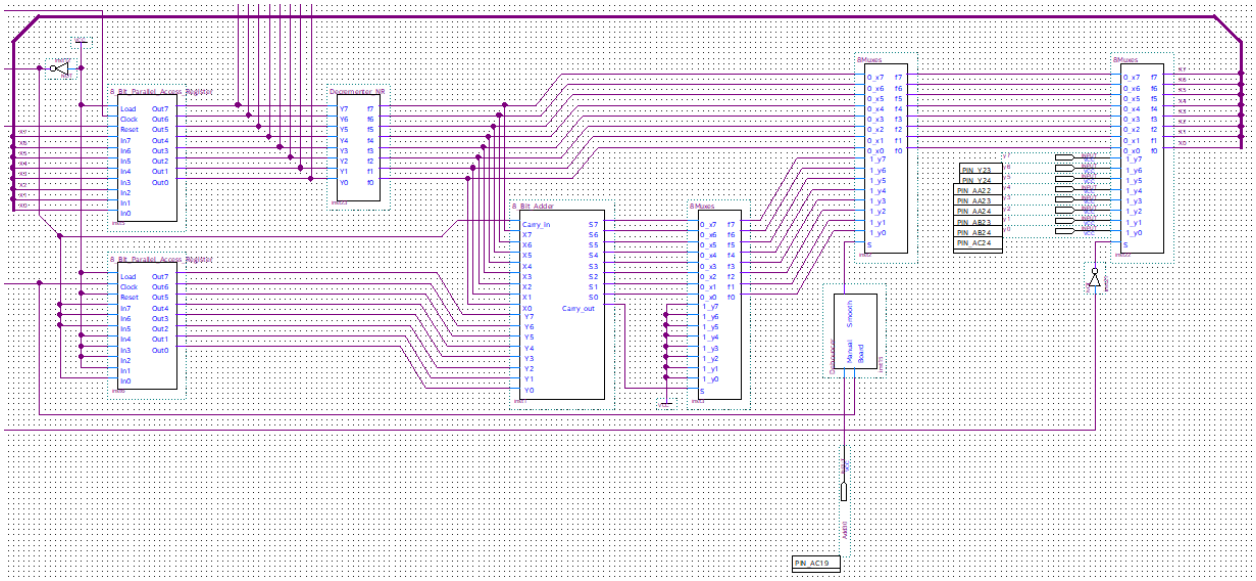
This is the debouncer that we made in Lab 11. It works by using two clock dividers and a D-flip-flop. The manual input of the switches from the board can quickly go from 1 to 0 when switching them which can cause errors. The manual input goes into a DFF that only updates every so often. This allows a smooth output from the DFF.

2-to-1 MUX



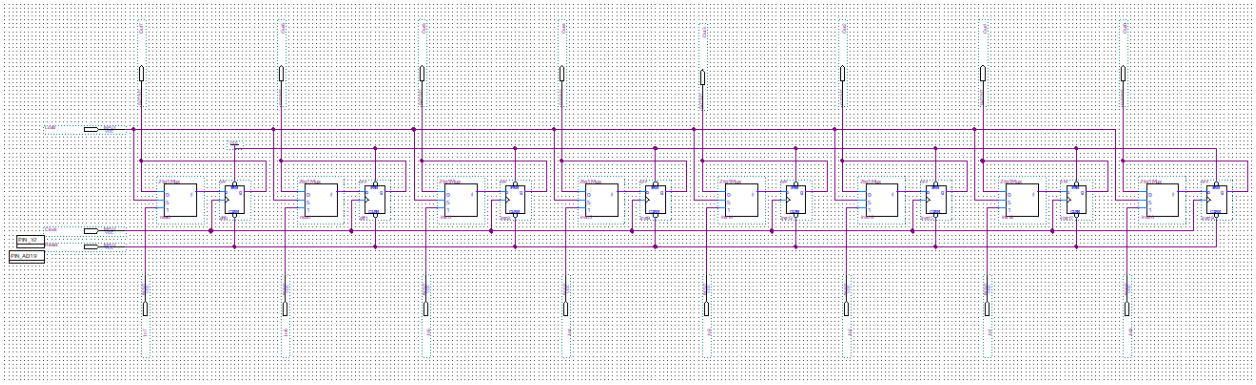
This component has two lines of data and a select line. If the select line is 1, data is taken from line 1. If the select line is 0, data is taken from line 0.

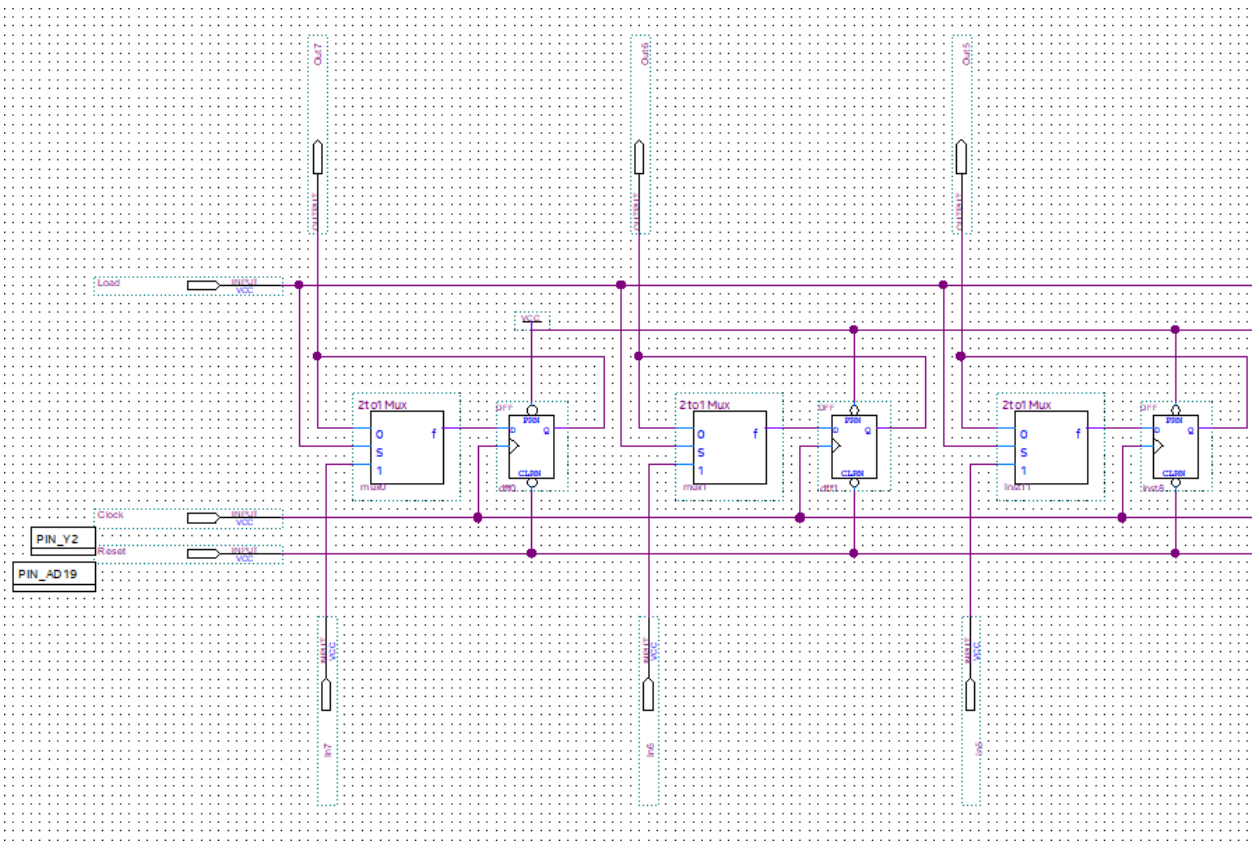
Section 2



Section 2 contains most of the functionality of the timer. This section is where user input is taken for setting the timer and adding 30 seconds. Section 2 contains two 8-bit parallel access registers, one for storing the value of 30, and the other for holding the value of the timer. Section 2 also includes a decrementer with no rollback, an 8-bit adder, three sets of eight 2-to-1 multiplexers, and a debouncer.

8-bit Parallel Access Register

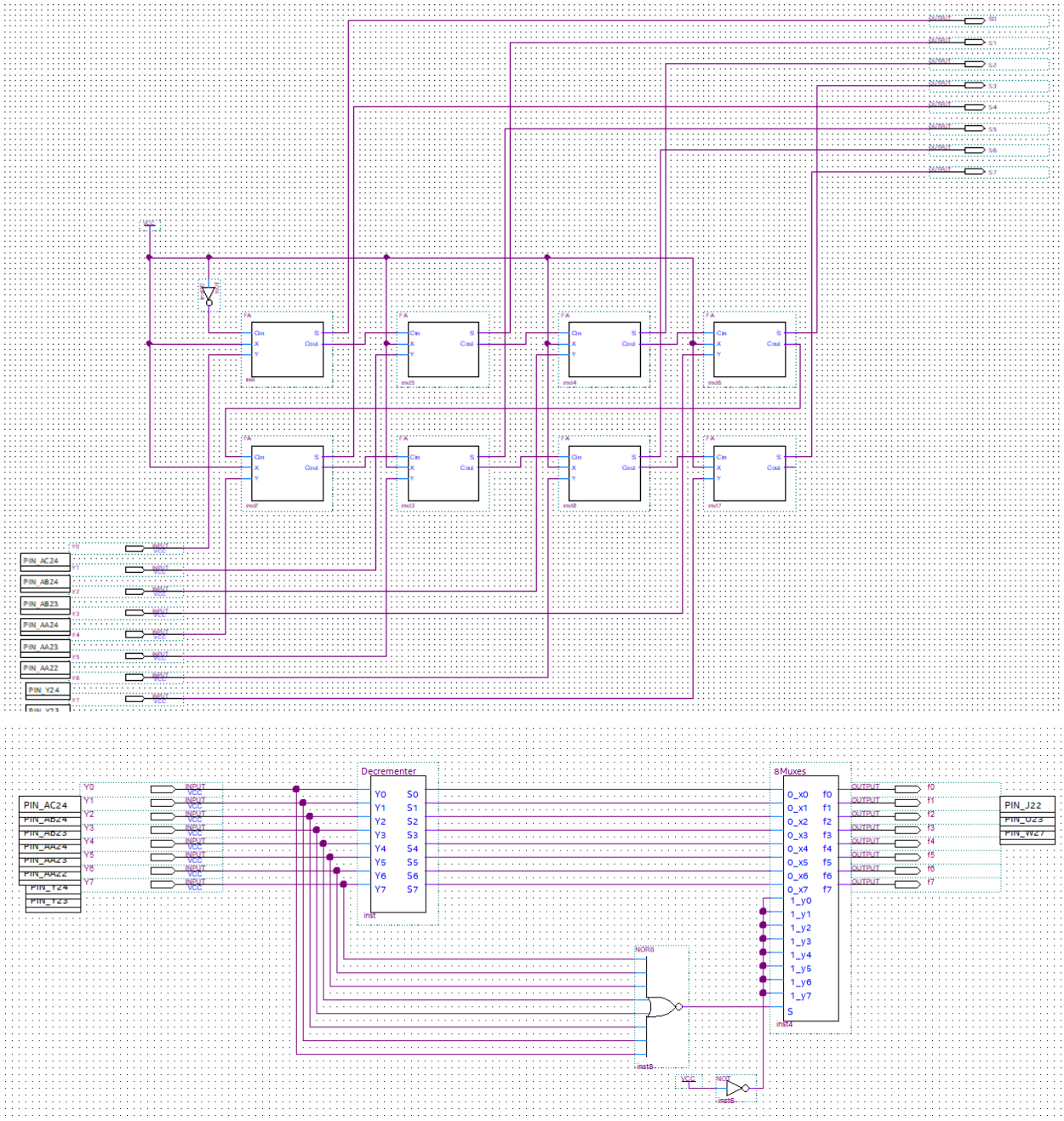




(Smaller section of the register for readability).

Our 8-bit parallel access register stores an 8-bit number and has a load and reset capability.

Decrementer



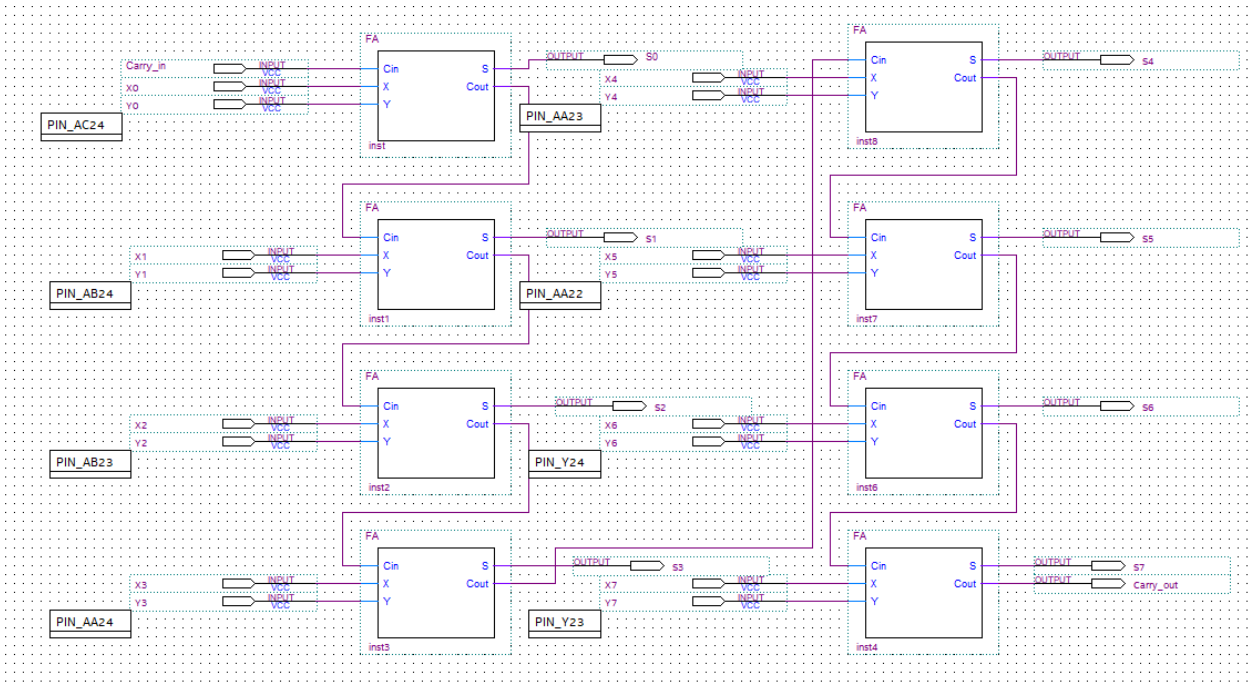
The decrementer adds 11111111 to the current value of the timer ignoring the final carry-out bit causing the value of the timer to decrement by 1 with each clock cycle. The no-rollback detects when the input is all zeros and stops adding, preventing the displayed time from rolling back to 255 seconds.

Full Adder

```
1   module FA(Cin, X, Y, S, Cout);
2      input Cin, X, Y;
3      output S, Cout;
4
5      assign S = (~X & ~Y & Cin) | (~X & Y & ~Cin) | (X & Y & Cin) | (X & ~Y & ~Cin);
6      assign Cout = (X & Y) | (Y & Cin) | (X & Cin);
7   endmodule
```

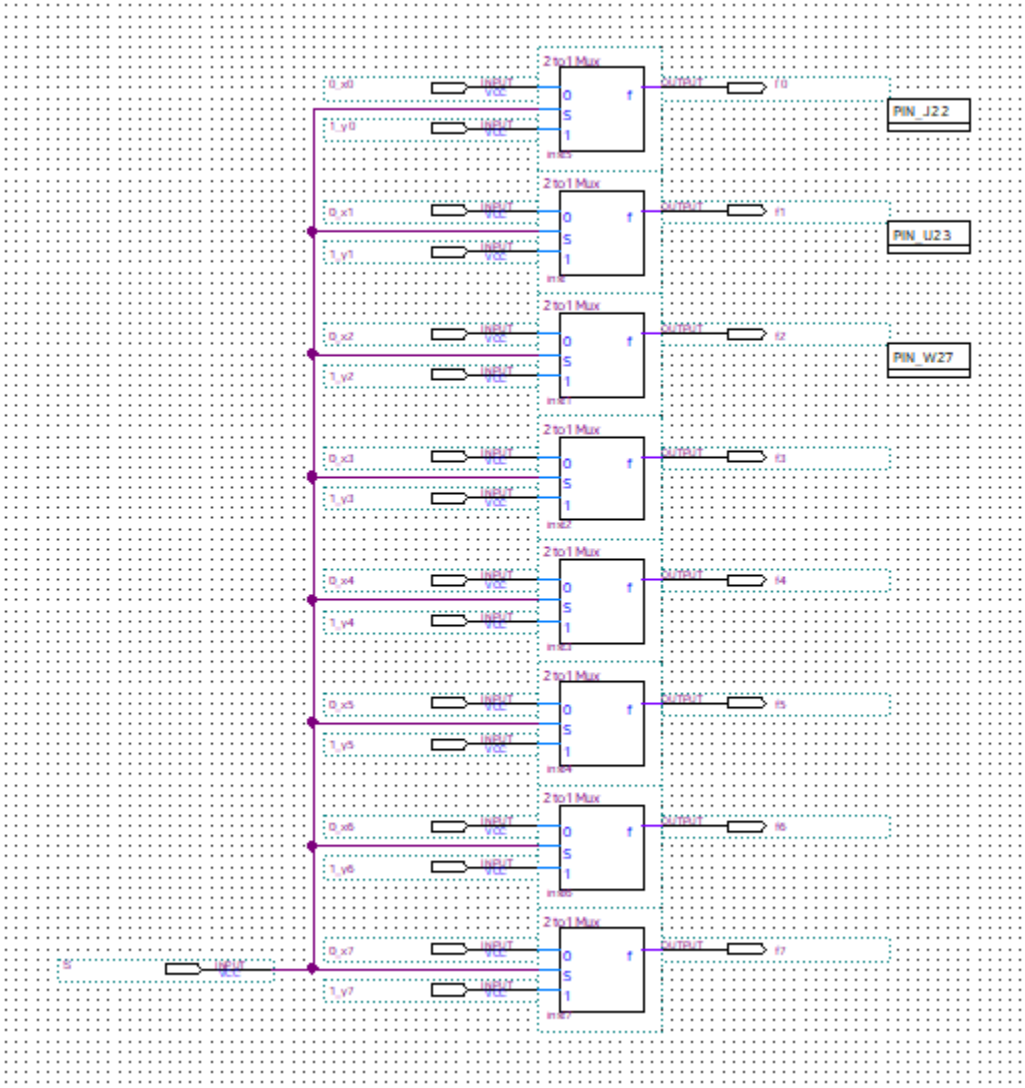
This is the Verilog code for a full adder which takes three inputs, Cin, X, and Y, and has two outputs, S, and Cout.

8-bit Adder



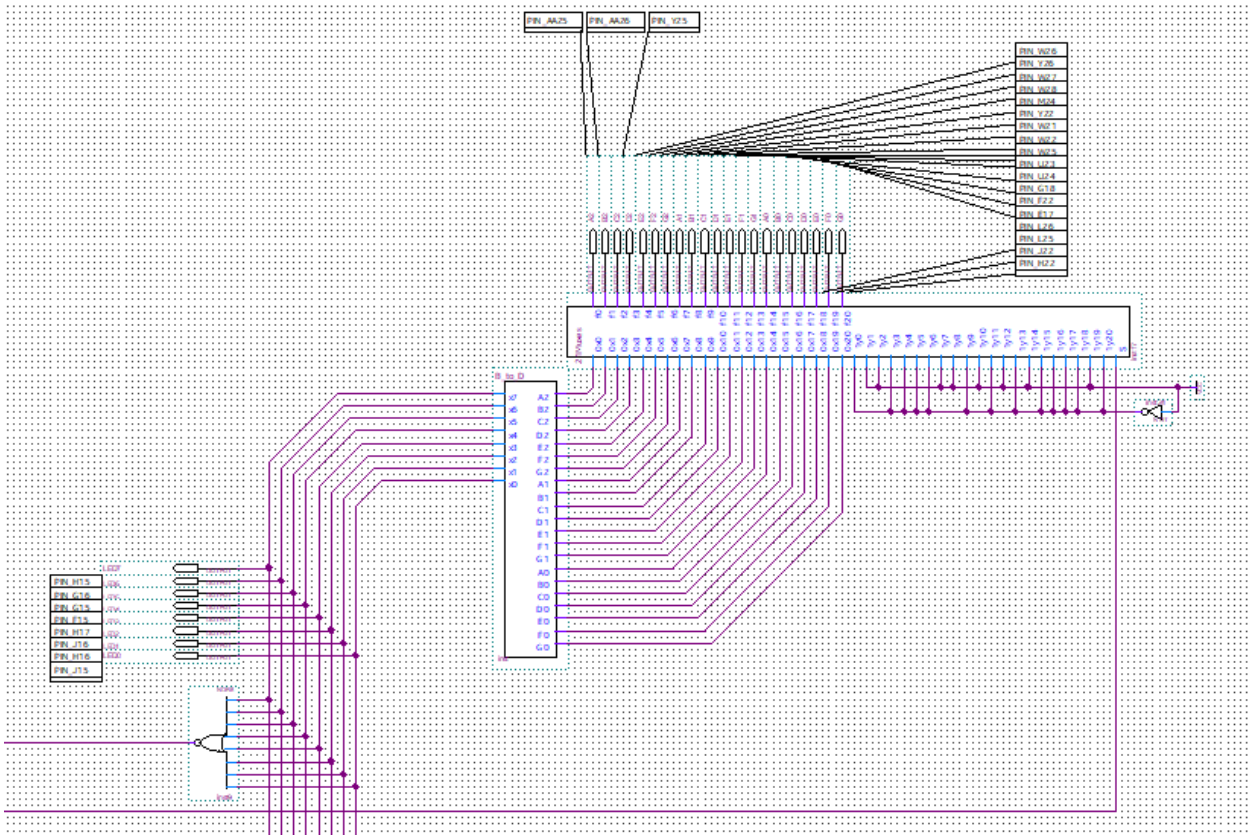
The 8-bit Adder is an array of eight full adders which adds two 8-bit binary numbers and outputs the result and carry bit.

8 MUXes



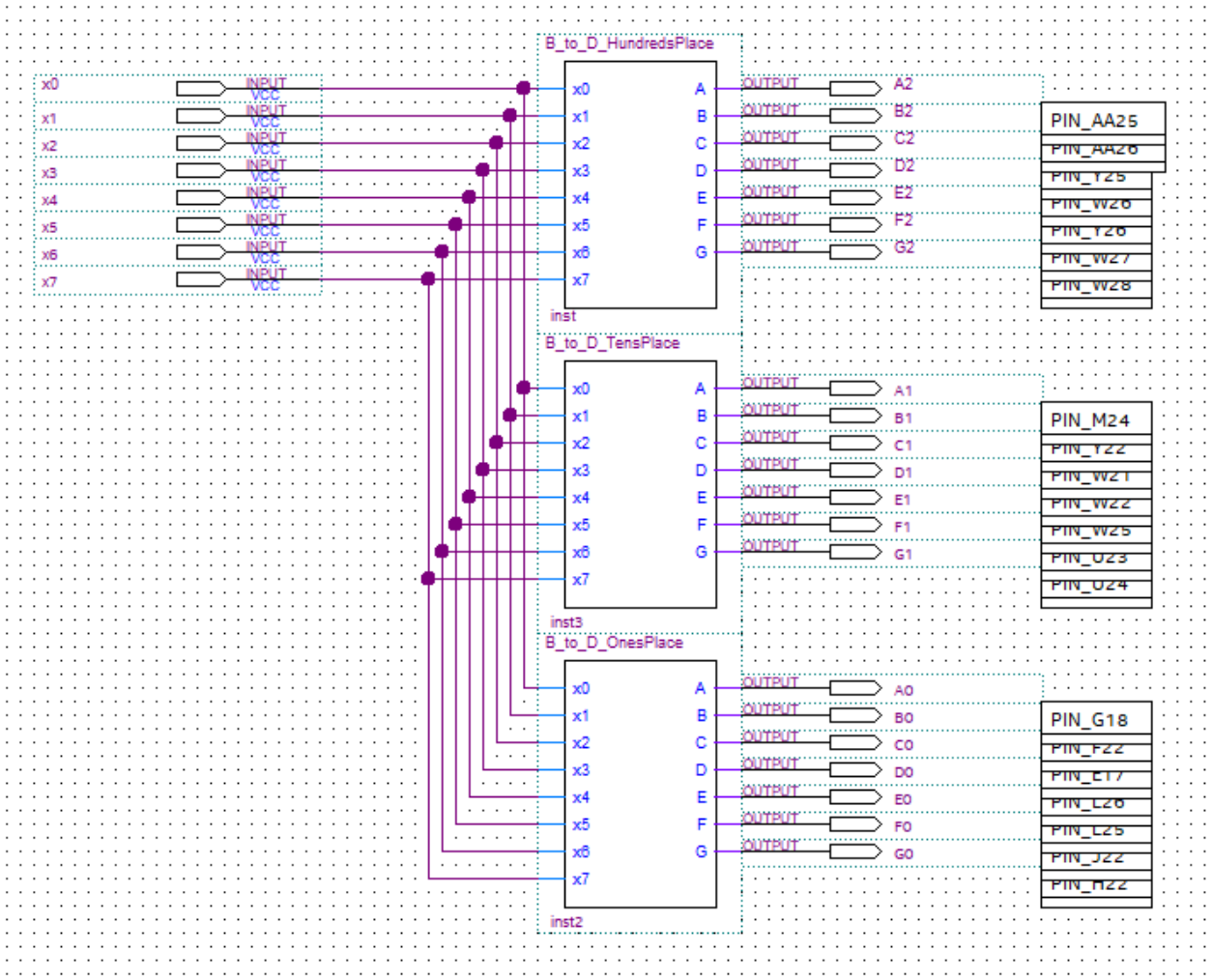
“8 MUXes” is a series of eight 2-to-1 multiplexers which all share a single select line and each has its own set of inputs and outputs.

Section 3



Section 3 contains the output of the timer as well as the binary-to-decimal conversion. The output is displayed using a series of 21 2-to-1 multiplexers all sharing the same select line in order to display either the current time remaining or the end result “End”. This section also has the outputs for the LEDs that show the timer’s number in binary.

Binary to Decimal



This part of the Binary to Decimal converter takes an 8-bit input and computes the digits for the one's place, tens place, and hundreds place separately. Originally, all three digits were included in one Verilog file and it didn’t work, so we separated them.

```
module B_to_D_HundredsPlace(x0,x1,x2,x3,x4,x5,x6,x7,A,B,C,D,E,F,G);

    input x0,x1,x2,x3,x4,x5,x6,x7;
    output A,B,C,D,E,F,G;
    reg A,B,C,D,E,F,G;

    always@(x7,x6,x5,x4,x3,x2,x1,x0)
    begin
       case({x7,x6,x5,x4,x3,x2,x1,x0})
          8'b00000000:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00000001:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00000010:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00000011:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00000100:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00000101:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00000110:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00000111:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00001000:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00001001:{A,B,C,D,E,F,G}=7'b0000001;

          8'b00001010:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00001011:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00001100:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00001101:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00001110:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00001111:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00010000:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00010001:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00010010:{A,B,C,D,E,F,G}=7'b0000001;
          8'b00010011:{A,B,C,D,E,F,G}=7'b0000001;
```

.

.

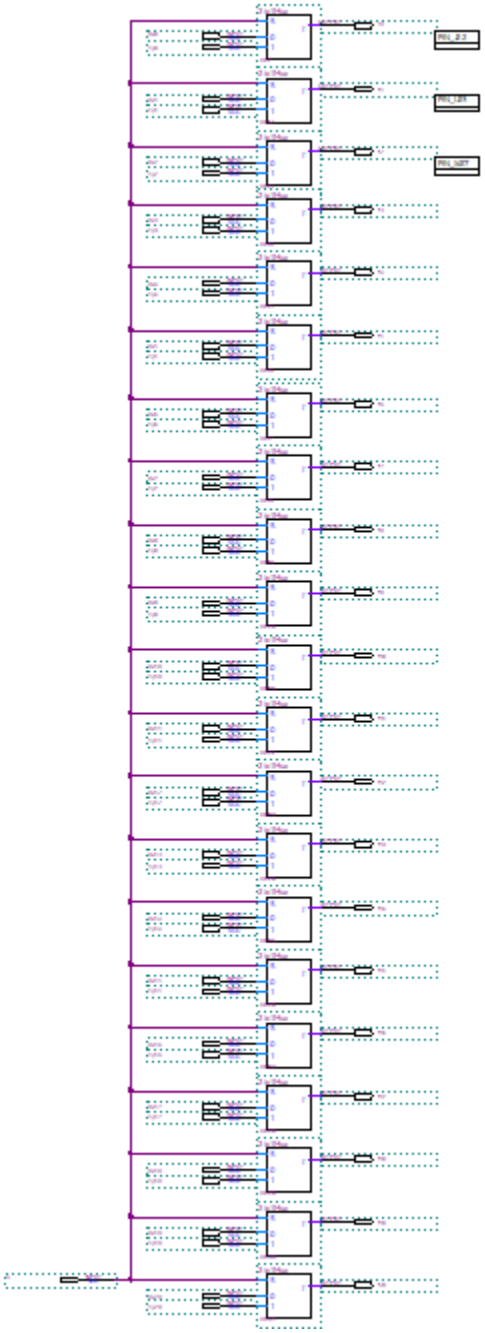
.

```
          8'b11111010:{A,B,C,D,E,F,G}=7'b0010010;
          8'b11111011:{A,B,C,D,E,F,G}=7'b0010010;
          8'b11111100:{A,B,C,D,E,F,G}=7'b0010010;
          8'b11111101:{A,B,C,D,E,F,G}=7'b0010010;
          8'b11111110:{A,B,C,D,E,F,G}=7'b0010010;
          8'b11111111:{A,B,C,D,E,F,G}=7'b0010010;
       endcase
    end
endmodule
```

(Not the entire file, also this is only the hundreds place file).

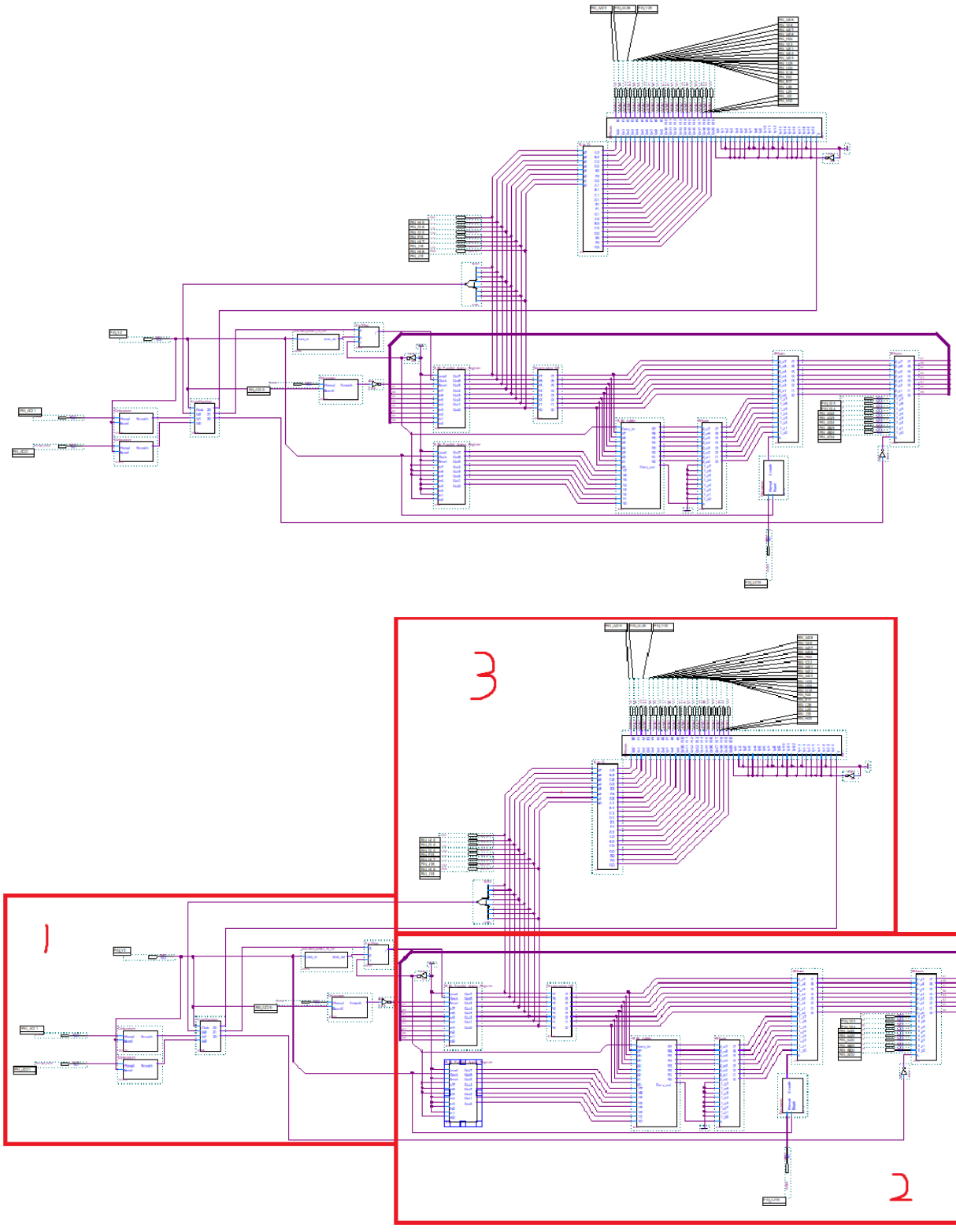
The separate Verilog files have a case for each possible 8-bit input and they output seven bits depending on whether that section of the corresponding seven-segment display should be lit up or not.

21 MUXes



“21 MUXes” is a series of 21 2-to-1 multiplexers which all share a single select line and each has its own set of inputs and outputs.

Putting It All Together



Section 1 Explanation

Section 1 takes input from section 3 and switches on the board, and outputs to sections 2 and 3. Section 1 contains the control logic for sections 2 and 3. Section 1 also contains the logic for pausing the timer. The finite state machine's outputs go into sections 2 and 3. One of the outputs remains in section 1 where it goes into a select line of a multiplexer that determines whether or not to stop the clock, in turn pausing the timer. Another output of the state machine goes into the select line of eight multiplexers that determines whether to take input from the user or count down. And the last output of the state machine goes into the select line of 21 multiplexers in section 3 where it determines whether or not to display numbers, or "End". Section 1 also contains the pin for the board clock and the 50 MHz to 1 Hz clock divider.

Section 2 Explanation

Section 2 contains the logic for the timer itself. Input is taken from the user and the cycle starts. The 8-bits at the right side of section 2 wrap back around to the front where they go into an 8-bit parallel access register with a 1-second clock. Everytime the register updates, the next value of the timer is calculated. The next value is calculated using the decrementer and possibly the 8-bit adder depending on whether or not the user is adding 30 seconds. This cycle continues until the timer reaches zero.

Section 3 Explanation

Section 3 takes the 8-bit number from section 2 and displays it to the seven-segment displays. Section 3 uses the binary to decimal converter to calculate which pins to light up. It also contains 21 multiplexers that determine whether or not to display a number or "End". Also, section 3 has output pins taken from the 8-bit number that light up the LEDs accordingly above the slide switches.

Conclusion

Overall, we learned a lot about digital logic design and had a good time working on this project. Some of the problems we ran into included getting the clock cycle down to exactly 1 Hz and making a down counter. We solved these problems by doing research and ditching the idea of a down counter and instead using a decrementer. In the future we hope to expand upon the knowledge we learned while doing this project and use it in the real world.