ECE 272 Lab 5

System Verilog

Benjamin Anderson II

Date:
Nov 15, 2022

1. Introduction

The purpose of this lab is to create a circuit for a stopwatch that resets every 24 hours. The stopwatch is to be created using SystemVerilog, and made to display on the 6 seven segment displays (7-segs) on a DE10-Lite. The SystemVerilog code that makes the stopwatch will be uploaded to the DE10-Lite through Quartus Prime v18.0, and simulated in ModelSim

2. Design

Name	Active High/Low	Location	Name	Active High/Low	Location
Clock_50MHz	HIGH	PIN_P11	seg3_a	LOW	PIN_F21
reset_n	HIGH	PIN_B8	seg3_b	LOW	PIN_E22
seg0_a	LOW	PIN_C14	seg3_c	LOW	PIN_E21
seg0_b	LOW	PIN_E15	seg3_d	LOW	PIN_C19
seg0_c	LOW	PIN_C15	seg3_e	LOW	PIN_C20
seg0_d	LOW	PIN_C16	seg3_f	LOW	PIN_D19
seg0_e	LOW	PIN_E16	seg3_g	LOW	PIN_E17
seg0_f	LOW	PIN_D17	seg4_a	LOW	PIN_F18
seg0_g	LOW	PIN_C17	seg4_b	LOW	PIN_E20
seg1_a	LOW	PIN_C18	seg4_c	LOW	PIN_E19
seg1_b	LOW	PIN_D18	seg4_d	LOW	PIN_J18
seg1_c	LOW	PIN_E18	seg4_e	LOW	PIN_H19
seg1_d	LOW	PIN_B16	seg4_f	LOW	PIN_F19
seg1_e	LOW	PIN_A17	seg4_g	LOW	PIN_F20
seg1_f	LOW	PIN_A18	seg5_a	LOW	PIN_J20
seg1_g	LOW	PIN_B17	seg5_b	LOW	PIN_K20
seg2_a	LOW	PIN_B20	seg5_c	LOW	PIN_L18
seg2_b	LOW	PIN_A20	seg5_d	LOW	PIN_N18
seg2_c	LOW	PIN_B19	seg5_e	LOW	PIN_M20
seg2_d	LOW	PIN_A21	seg5_f	LOW	PIN_N19
seg2_e	LOW	PIN_B21	seg5_g	LOW	PIN_N20
seg2_f	LOW	PIN_C22			
seg2_g	LOW	PIN_B22			

Figure 0: Interface Definition

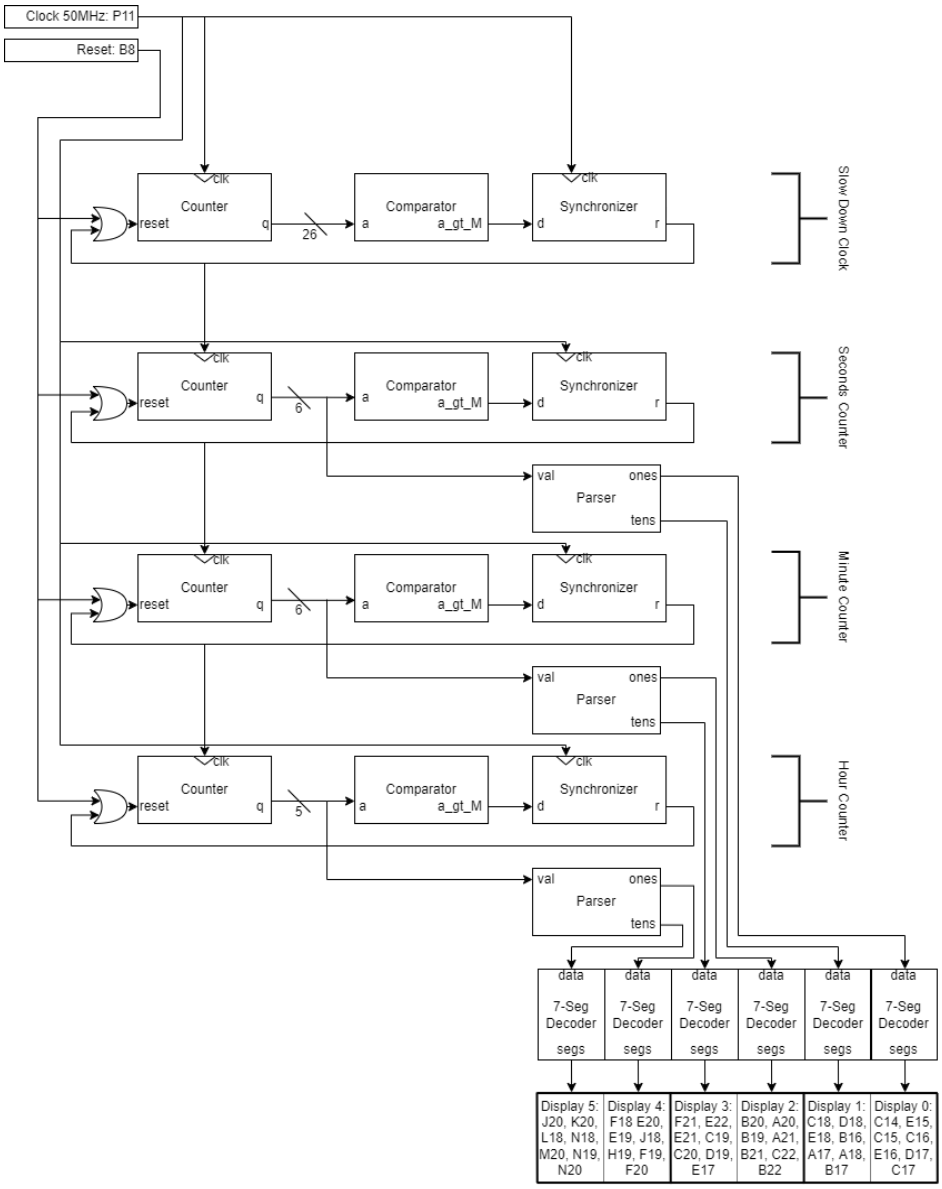

Figure 1: Clock Block Diagram

Figure 1, shown above, describes the logic used in the circuit as a block diagram. To follow the logic, clock and reset are the only input signals, and the clock is the only one that the logic itself is based off of. The clock signal goes in at 50MHz and is converted to 1Hz, or 1 second. This then counts up to 60, displaying its current value to the display. After reaching 60 the seconds reset to 0, and the minutes increment by one. The same logic then follows with minutes and hours, with the hours resetting at 24 rather than 60. The way these values are able to reset at 50 million, 60, 24 are through the use of comparators which make sure the number hasn't gotten too big, and sends a reset signal when they do. This reset signal then syncs up with the 50MHz clock and tells the counter to reset while telling the next counter in the line to increment.

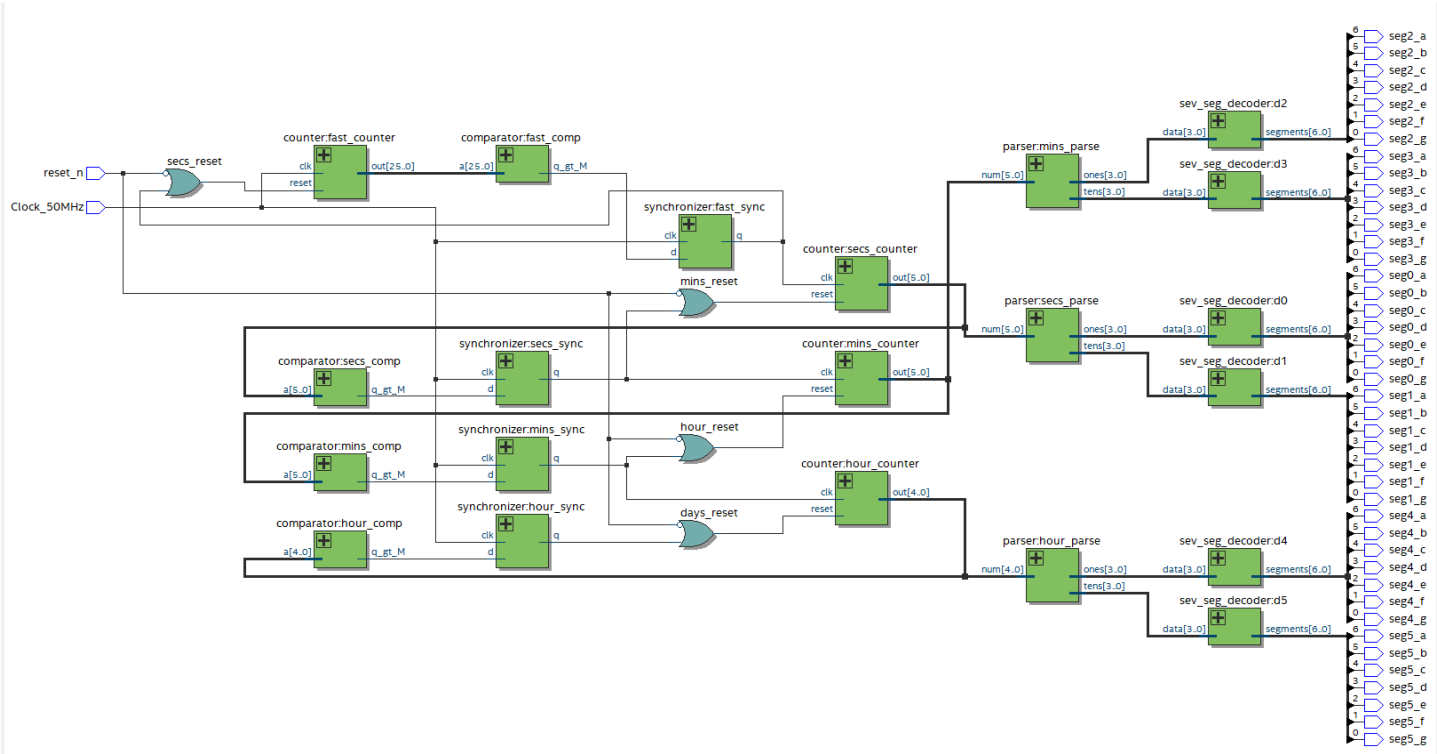

Figure 2: Computer Generated Clock Schematic

This next section goes through the computer generated schematics of the different modules used to create the counter. The first of these is Figure 2, which shows the top-level view of the entire circuit, and is almost identical in structure to the block diagram shown in Figure 1. The reason for this, and the rest of the modules, being computer generated, rather than handmade is because they were all coded in SystemVerilog, rather than drawn out using Quartus.

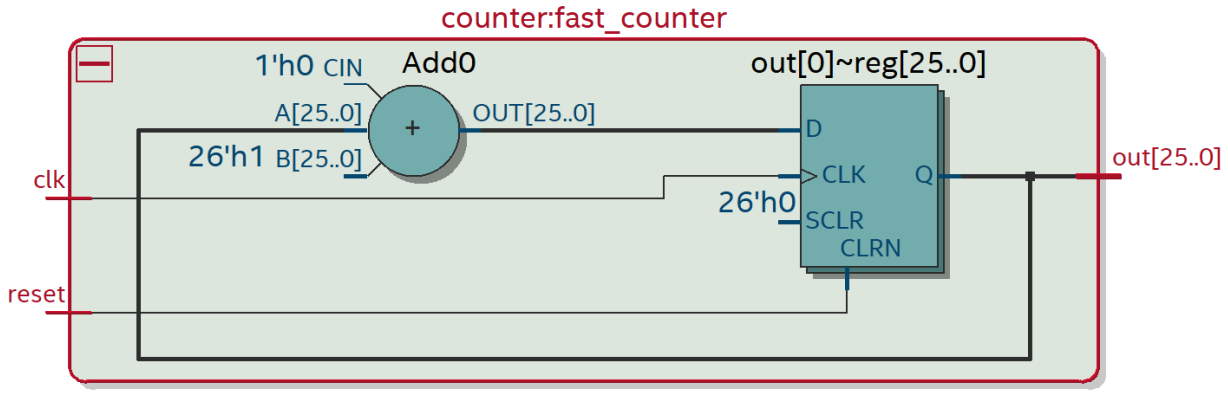

Figure 3: Computer Generated Counter Schematic

Figure 3 above shows a very basic counter module that increments the output by 1 on the clock edge, specifically one with 26 possible output bits. The reason for this being the fact that 50 million is between 2^{25} and 2^{26}.

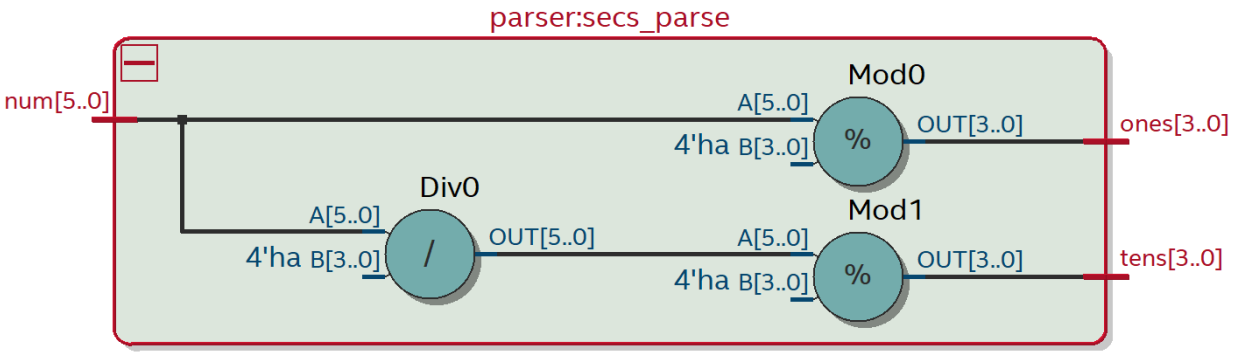

Figure 4: Computer Generated Parser Schematic

The parser schematic shown in Figure 4 is capable of outputting 0-9 in both the 1s and 10s place. This is performed by taking the number modulus 10, and outputting it to the 1s, meaning the *remainder* of the number divided by 10 is displayed in the 1s. The 10s works similarly, but it is first divided by 10 to truncate the 1s place.

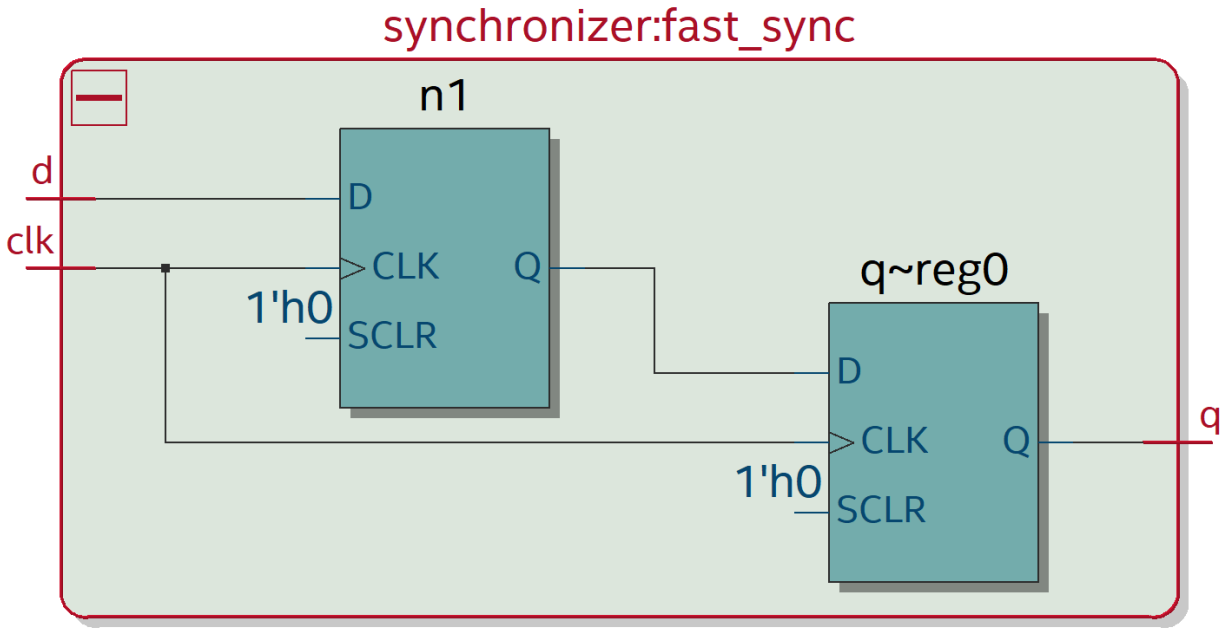

Figure 5: Computer Generated Synchronizer Schematic

The logic used for figure 5 was found in the textbook *Digital Design and Computer Architecture, Second Edition by David and Sarah Harris*. It describes a circuit in which the data is output in line with a clock 2 cycles after the input is received.

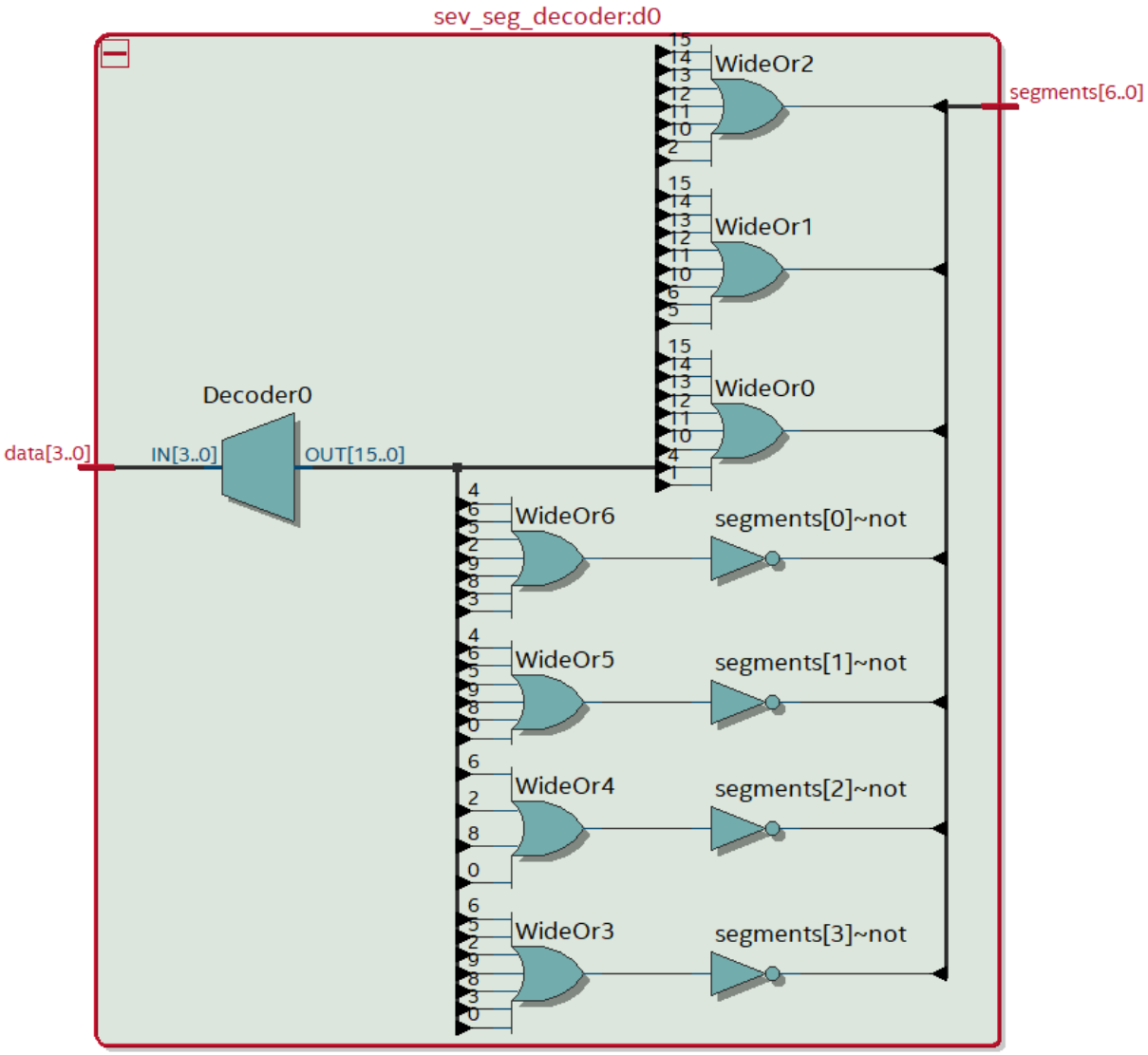

Figure 6: Computer Generated 7-seg Decoder Schematic

Above (Figure 6) is the 7-seg decoder used in the Stopwatch circuit. The extremely keen will notice that it does not account for the full hexadecimal range of outputs, and instead outputs nothing when the input exceeds decimal 9.

3. Results

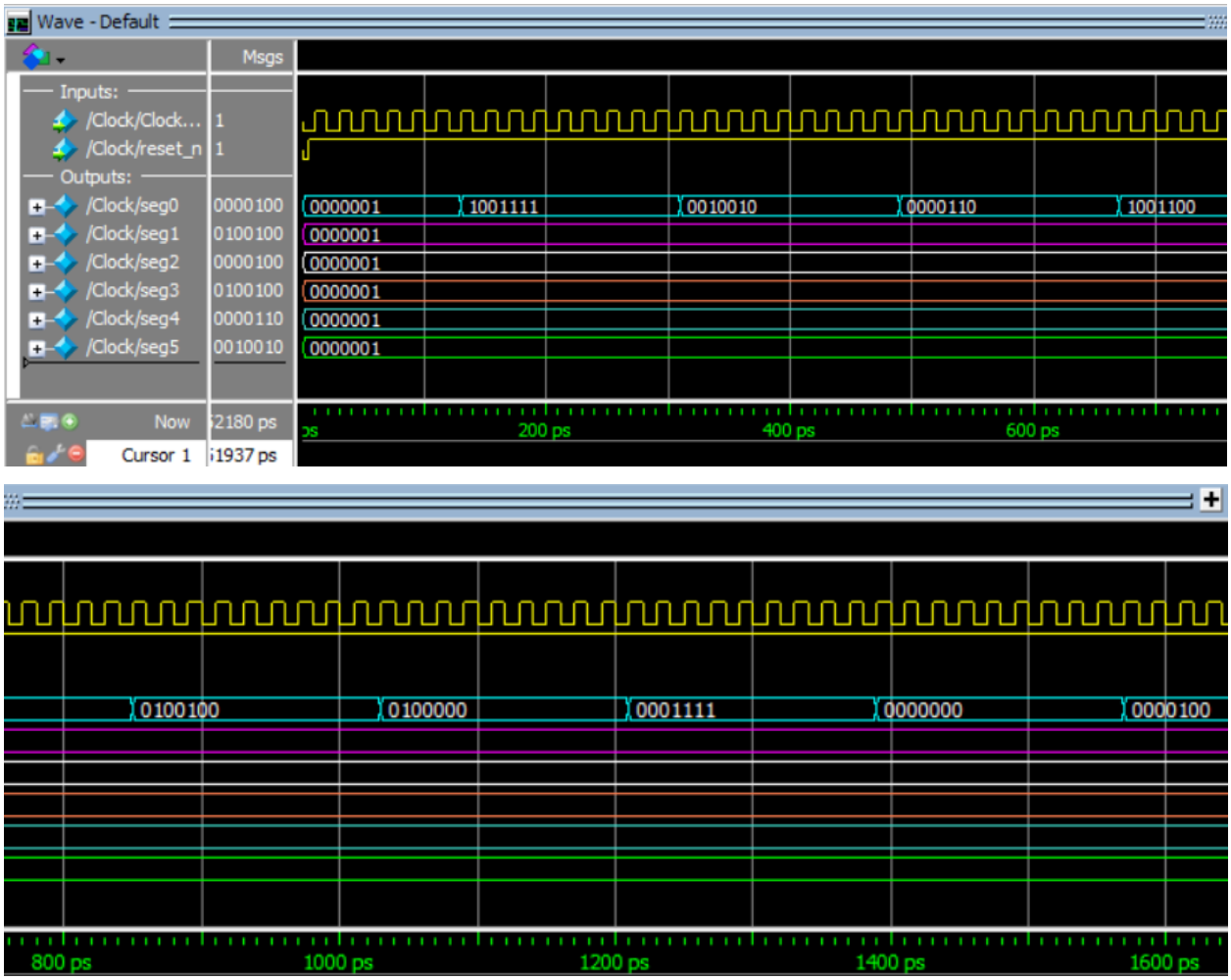

Figure 7: Counter Incrementing the Rightmost 7-seg

Figure 7 has been split into two different images to help the reader better see the counter increment.

To describe what is happening above, every 10 iterations of the top clock signal seg0 (the rightmost 7-seg display) changes. When decoded in active low onto a 7-seg display will show the number increasing from 0 to 9.

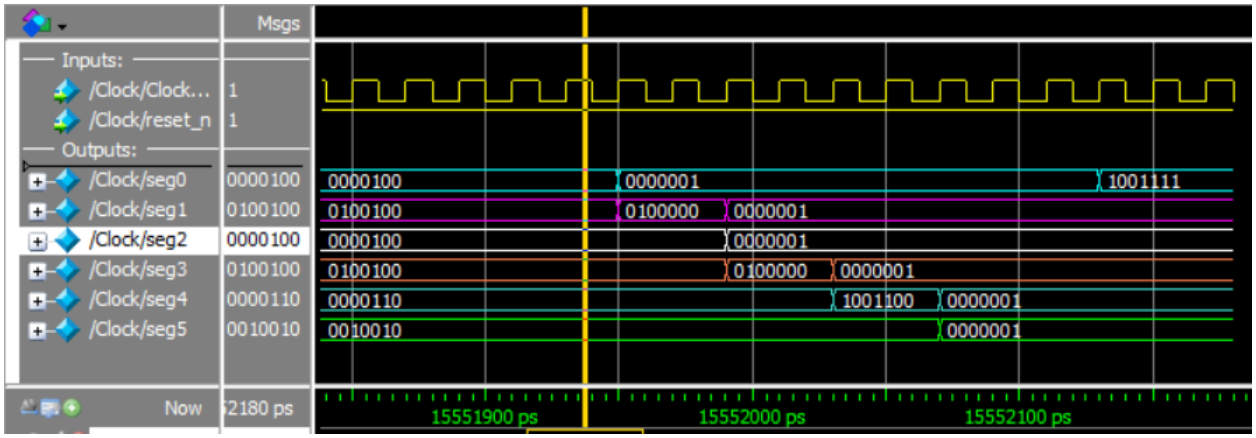

Figure 8: All Segments Resetting to 0

Figure 8 shows all of the comparators and parsers working at the same time. The cascading resets are due to the synchronizer putting the next counter off by 2 clock cycles (1/25,000,000 seconds) every time it is reset. However, roughly 24.5 million seconds (~9 months) would have to pass for the seconds to be off by 1, so it isn't much of a concern when used short-term.

4. Experiment Notes

Being the Computer Science major that I am, I challenged myself to do this entire lab in System Verilog, and to be honest, I don't think I'll switch back to making schematics. As for how the assignment went, I made the block diagram the night before the lab so that I would be able to have something to work off of for the lab itself, which really helped the debugging that I did later. Writing up the code for the modules wasn't too difficult, the issues arose more in the connecting of modules. I found on more than one occasion that I simply didn't state that the wire being used was supposed to be a bus, so it wasn't passing the right values to the modules. All of these were issues that were not present in the Block diagram I had made, so it was more an issue of figuring out what the discrepancy was between the plan I had made and the implementation of the plan I was using. After that I modeled the clock in ModelSim, which took a bit of fenangling, but eventually worked out, and I learned how to properly make a .do file in the process.

5. Study Questions

1) How off is your clock? (how much does it differ from a minute on a stopwatch/phone)

I know my FPGA is off by 1/25,000,000 seconds every minute, hour and day that passes. Of course, this isn't readable by people, so when I tested this I got that my FPGA was exactly on time (within the margin of error of human reflexes), so I would say that for anything less than around 24.5 million seconds the clock is accurate to the second, after that it's noticeably off by around a second.

6. Appendix

TOP LEVEL FILE

```
module Clock
    (input logic Clock_50MHz,
    input  logic reset_n,

    output logic seg0_a,
    output logic seg0_b,
    output logic seg0_c,
    output logic seg0_d,
    output logic seg0_e,
    output logic seg0_f,
    output logic seg0_g,

    output logic seg1_a,
    output logic seg1_b,
    output logic seg1_c,
    output logic seg1_d,
    output logic seg1_e,
    output logic seg1_f,
    output logic seg1_g,

    output logic seg2_a,
    output logic seg2_b,
    output logic seg2_c,
    output logic seg2_d,
    output logic seg2_e,
    output logic seg2_f,
    output logic seg2_g,

    output logic seg3_a,
    output logic seg3_b,
    output logic seg3_c,
    output logic seg3_d,
    output logic seg3_e,
    output logic seg3_f,
    output logic seg3_g,

    output logic seg4_a,
    output logic seg4_b,
    output logic seg4_c,
    output logic seg4_d,
    output logic seg4_e,
    output logic seg4_f,
    output logic seg4_g,

    output logic seg5_a,
    output logic seg5_b,
    output logic seg5_c,
    output logic seg5_d,
    output logic seg5_e,
    output logic seg5_f,
    output logic seg5_g);
```

```
52
53      //resets (named after what they reset on)
54      logic secs_reset;
55      logic mins_reset;
56      logic hour_reset;
57      logic days_reset;
58
59      //counter to comparator
60      logic[25:0] FifM_to_1_comp;
61      logic[5:0] sec_comp;
62      logic[5:0] min_comp;
63      logic[4:0] hrs_comp;
64
65      //comparator to synchronizer
66      logic a_gt_B;
67      logic a_gt_D;
68      logic a_gt_F;
69      logic a_gt_H;
70
71      //post parser
72      logic[3:0] sec_ones;
73      logic[3:0] min_ones;
74      logic[3:0] hrs_ones;
75      logic[3:0] sec_tens;
76      logic[3:0] min_tens;
77      logic[3:0] hrs_tens;
78
79      //clock signals
80      logic secs_clock;
81      logic mins_clock;
82      logic hour_clock;
83      logic days_clock;
84
85      //sev_seg_bus
86      logic[6:0] seg0;
87      logic[6:0] seg1;
88      logic[6:0] seg2;
89      logic[6:0] seg3;
90      logic[6:0] seg4;
91      logic[6:0] seg5;
92
93      //assign reseters
94      assign secs_reset = secs_clock | ~reset_n;
95      assign mins_reset = mins_clock | ~reset_n;
96      assign hour_reset = hour_clock | ~reset_n;
97      assign days_reset = days_clock | ~reset_n;
98
99      //assign outputs
100     assign seg0_a = seg0[6];
101     assign seg0_b = seg0[5];
102     assign seg0_c = seg0[4];
103     assign seg0_d = seg0[3];
104     assign seg0_e = seg0[2];
105     assign seg0_f = seg0[1];
106     assign seg0_g = seg0[0];
```

```
108     assign seg1_a = seg1[6];
109     assign seg1_b = seg1[5];
110     assign seg1_c = seg1[4];
111     assign seg1_d = seg1[3];
112     assign seg1_e = seg1[2];
113     assign seg1_f = seg1[1];
114     assign seg1_g = seg1[0];
115
116     assign seg2_a = seg2[6];
117     assign seg2_b = seg2[5];
118     assign seg2_c = seg2[4];
119     assign seg2_d = seg2[3];
120     assign seg2_e = seg2[2];
121     assign seg2_f = seg2[1];
122     assign seg2_g = seg2[0];
123
124     assign seg3_a = seg3[6];
125     assign seg3_b = seg3[5];
126     assign seg3_c = seg3[4];
127     assign seg3_d = seg3[3];
128     assign seg3_e = seg3[2];
129     assign seg3_f = seg3[1];
130     assign seg3_g = seg3[0];
131
132     assign seg4_a = seg4[6];
133     assign seg4_b = seg4[5];
134     assign seg4_c = seg4[4];
135     assign seg4_d = seg4[3];
136     assign seg4_e = seg4[2];
137     assign seg4_f = seg4[1];
138     assign seg4_g = seg4[0];
139
140     assign seg5_a = seg5[6];
141     assign seg5_b = seg5[5];
142     assign seg5_c = seg5[4];
143     assign seg5_d = seg5[3];
144     assign seg5_e = seg5[2];
145     assign seg5_f = seg5[1];
146     assign seg5_g = seg5[0];
147
148     //counters (clk, reset,  out)
149     counter #(26) fast_counter(Clock_50MHz, secs_reset, FifM_to_1_comp[25:0]);
150     counter #(6)  secs_counter(secs_clock, mins_reset, sec_comp[5:0]);
151     counter #(6)  mins_counter(mins_clock, hour_reset, min_comp[5:0]);
152     counter #(5)  hour_counter(hour_clock, days_reset, hrs_comp[4:0]);
153
154     //comparators (params: N, M .. logic: a[N-1], a_gt_M)
155     comparator #(26, 5/*0000000*/) fast_comp(FifM_to_1_comp[25:0], a_gt_B);
156     comparator #(6, 60)        secs_comp(sec_comp[5:0], a_gt_D);
157     comparator #(6, 60)        mins_comp(min_comp[5:0], a_gt_F);
158     comparator #(5, 24)        hour_comp(hrs_comp[4:0], a_gt_H);
```

```
159
160     //synchronizers (clk, d, q)
161     synchronizer fast_sync(Clock_50MHz, a_gt_B, secs_clock);
162     synchronizer secs_sync(Clock_50MHz, a_gt_D, mins_clock);
163     synchronizer mins_sync(Clock_50MHz, a_gt_F, hour_clock);
164     synchronizer hour_sync(Clock_50MHz, a_gt_H, days_clock);
165
166     //parsers (params: N .. logic: num, ones, tens)
167     parser #(6) secs_parse(sec_comp[5:0], sec_ones[3:0], sec_tens[3:0]);
168     parser #(6) mins_parse(min_comp[5:0], min_ones[3:0], min_tens[3:0]);
169     parser #(5) hour_parse(hrs_comp[4:0], hrs_ones[3:0], hrs_tens[3:0]);
170
171     //seven segment decoders (in, out)
172     sev_seg_decoder d0(sec_ones[3:0], seg0[6:0]);
173     sev_seg_decoder d1(sec_tens[3:0], seg1[6:0]);
174     sev_seg_decoder d2(min_ones[3:0], seg2[6:0]);
175     sev_seg_decoder d3(min_tens[3:0], seg3[6:0]);
176     sev_seg_decoder d4(hrs_ones[3:0], seg4[6:0]);
177     sev_seg_decoder d5(hrs_tens[3:0], seg5[6:0]);
178
179 endmodule
180
```

The only note I have about the Top-Level is on line 155. I forgot to uncomment out the 0s from when I was simulating the code, so just know that the number is supposed to be 50,000,000, not 5.

COUNTER

```
module counter
    #(parameter N)
    (input logic clk,
    input  logic reset,
    output logic[N-1:0] out);

    always_ff @(posedge clk, posedge reset)
        if(reset) out <= 0;
        else out <= out + 1;
endmodule

```

COMPARATOR

```
module comparator
    #(parameter N,
    parameter   M)
    (input logic[N-1:0] a,
    output logic q_gt_M);

    assign q_gt_M = (a >= M);
endmodule

```

SYNCHRONIZER

```
module synchronizer
    (input logic clk,
    input  logic d,
    output logic q);

    logic n1;
    always_ff @(posedge clk) begin
        n1 <= d;
        q <= n1;
    end
endmodule
```

PARSER

```
module parser
    #(parameter N)
    (input logic[N-1:0] num,
    output logic[3:0] ones,
    output logic[3:0] tens);

    assign ones = num % 10;
    assign tens = (num / 10) % 10;
endmodule
```

SEVEN SEGMENT DECODER

```
module sev_seg_decoder(input  logic[3:0] data,
                       output logic[6:0] segments);
    always_comb
        case(data)
            0:       segments<=7'b0000_001;
            1:       segments<=7'b1001_111;
            2:       segments<=7'b0010_010;
            3:       segments<=7'b0000_110;
            4:       segments<=7'b1001_100;
            5:       segments<=7'b0100_100;
            6:       segments<=7'b0100_000;
            7:       segments<=7'b0001_111;
            8:       segments<=7'b0000_000;
            9:       segments<=7'b0000_100;
            default: segments<=7'b1111_111;
    endcase
endmodule
```