CS M152A

Lab 3 Report

Puvali Chatterjee
Lab Partner: Vivian Ha
TA: Samuel Lee

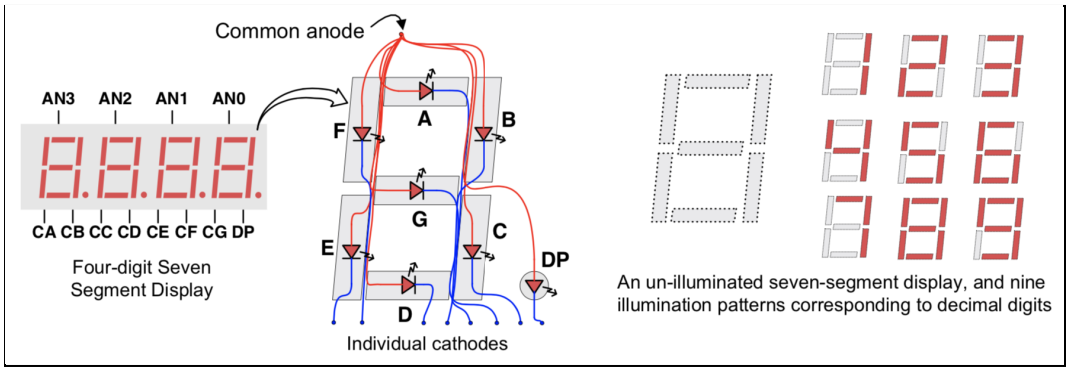
Introduction and Requirements

In this lab, we designed a stopwatch circuit and implemented it on the Nexys 3 Spartan-6 FPGA board. The inputs of the stopwatch are buttons and slider switches and the digits of the stopwatch are displayed on the on-board seven segment display.

The stopwatch starts as a basic clock which counts seconds and minutes at 1 Hz. The two leftmost digits on the seven-segment display are used to display the minutes and the two rightmost digits display the seconds. The mode of the stopwatch can be changed by the two input switches SEL and ADJ. ADJ puts the timer in adjustment mode, where it freezes either the minutes or seconds while incrementing the other at 2 Hz. The SEL switch selects which time unit to increment in adjustment mode: when SEL = 0, seconds are frozen and minutes are incremented and when SEL = 1, minutes are frozen and seconds are incremented. There are also two input pushbuttons which alter the timer behavior: RESET forces all counters to the initial state 00:00 and PAUSE pauses the counter and resumes it when pressed again.

To eliminate noise in the pushbutton inputs, we debounced the PAUSE and RESET buttons. We did this by generating a single pulse with the period of the fast clock whenever a pushbutton is pressed and released. This is done with three debouncing flip-flops that are driven by the fast clock[1].

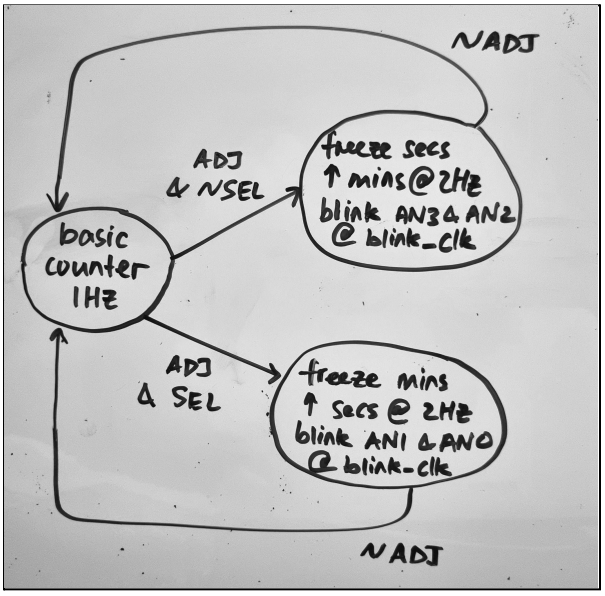
To understand how to use the seven-segment display, we studied the reference manual of the board. The seven-segment display consists of a four-digit common anode and separate LED cathodes. Each of the four digits is composed of seven segments arranged in a “figure 8” pattern, with an LED embedded in each segment. The cathodes are connected into seven circuit nodes labeled CA through CG . These seven cathode signals are available as inputs to the 4-digit display [2].



Nexys 3 Seven-Segment Display

The above diagram shows the illumination patterns required to light up a specific digit. The cathode signals are common to all digits but they can only illuminate the segments of the digit whose corresponding anode signal is asserted. For example, to display ‘7’ in the one’s place of the minutes, AN2 is asserted (logic level low) while setting the cathode signals to 7'b0001111. Only one anode signal can be asserted at a time so the display cycles between the four digits by updating at a rate that is faster than the human eye can detect so that it looks like all four digits are being displayed at once. We set the update rate to 500 Hz driven by the fast clock.

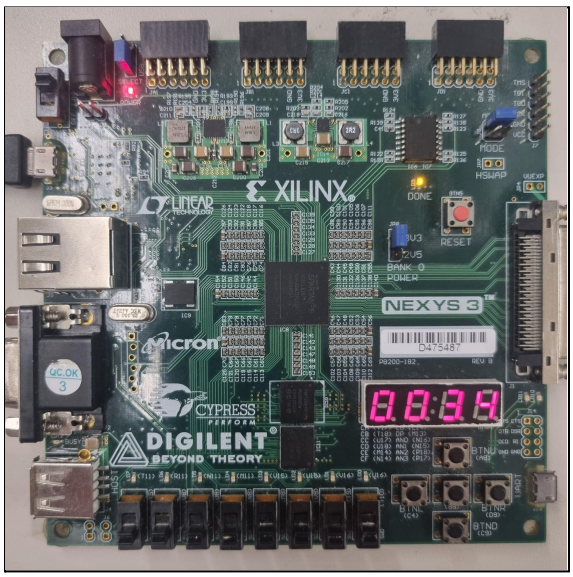
Design Description



FSM Diagram of Stopwatch

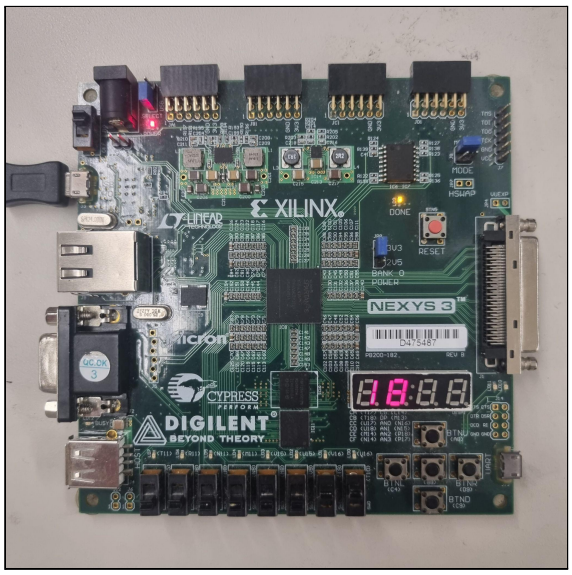
The above finite state machine diagram shows the 3 states of the stopwatch: basic counter and the 2 adjusting mode states for SEL = 0 and SEL = 1. RESET and PAUSE are not considered states since they do not result in mode change i.e. resetting or pausing the stopwatch does not change the counter from basic counter mode to adjustment mode, or vice versa.

- Basic counter: In this state, the stopwatch acts as a basic clock that counts seconds and minutes at 1 Hz. At 59:59 (59 minutes and 59 seconds), the counter resets to 00:00. The following photo shows the counter when it has counted up to 0 minutes and 34 seconds.



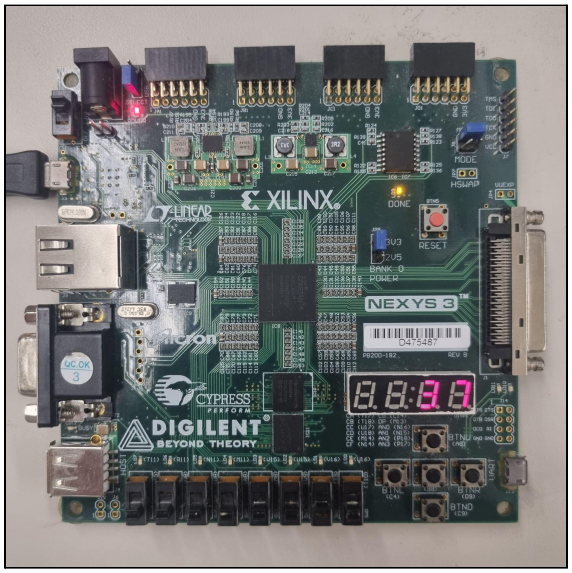
Stopwatch in basic counter state

- Adjusting minutes: The stopwatch enters the minutes adjustment mode when the ADJ switch is on and the SEL switch is off. The stopwatch freezes the seconds and starts incrementing only the minutes at 2 Hz while blinking the minutes' digits (the two leftmost digits) at 1.25 Hz. Following is a photo of the stopwatch mid-blink while incrementing minutes.



Stopwatch in minutes adjustment mode

- Adjusting seconds: The stopwatch enters the seconds adjustment mode when the ADJ switch is on and the SEL switch is on. The stopwatch freezes the minutes and starts incrementing only the seconds at 2 Hz while blinking the seconds' digits (the two rightmost digits) at 1.25 Hz. Following is a photo of the stopwatch mid-blink while incrementing seconds.



Stopwatch in seconds adjustment mode

In the code, we implemented several modules to build the overall stopwatch:

1. **module stopwatch (input clk,**
   input SEL,
   input ADJ,
   input RESET,
   input PAUSE,
   output reg [3:0] anode,
   output reg [6:0] cathode);

   The *stopwatch* module is the top-level module which controls the overall logic of the program.

   ➔ First, it instantiates the clock module four times to get the divided clocks from the input *clk* (100 MHz internal clock of board).

   ➔ Then it debounces the *RESET* and *PAUSE* inputs by instantiating the debouncer module twice.

   ➔ Then we declare three parameters for the 3 states: *basic, adj_min* and *adj_sec* corresponding with 0, 1 and 2. An always block sets the *state* register depending on the values of the ADJ and SEL inputs. This block also sets the value of a *blinking* register to 0, 1 or 2 for no blinking, blink minutes or blink seconds respectively. This is shown below:

```
//3 states: basic clock, adjusting minutes and adjusting seconds
parameter basic = 4'd0;
parameter adj_min = 4'd1;
parameter adj_sec = 4'd2;

reg [2:0] state;

//0 for no blink, 1 for blink min, 2 for blink sec
reg [2:0] blinking;

always @(*) begin
        if (ADJ & ~SEL) begin
                state = adj_min;
                blinking = 1;
        end
        else if (ADJ & SEL) begin
                state = adj_sec;
                blinking = 2;
        end
        else if (~ADJ) begin
                state = basic;
                blinking = 0;
        end

end
```

   ➔ Another always block (triggered by posedge of the debounced pause pushbutton) sets a register *ispaused.*

   ➔ Then, the *counter* module is instantiated and it returns values for the counted *minutes* and *seconds* so far, and these numbers are separated into their tens' and ones' place values as follows:

```
//minutes and seconds
wire [5:0] minutes;
wire [5:0] seconds;

wire [3:0] min_tens;
wire [3:0] min_ones;
wire [3:0] sec_tens;
wire [3:0] sec_ones;

//instantiate counter module
counter ctr(RESET, ispaused, state, clk_1hz, clk_2hz, minutes, seconds);

assign min_tens = minutes/10;
assign min_ones = minutes - (min_tens * 10);
assign sec_tens = seconds/10;
assign sec_ones = seconds - (sec_tens * 10);
```

➔ Then the *segments* module is instantiated to obtain the cathode signal patterns for the digits to be displayed for the current time stored in the stopwatch.

➔ Finally, an edge-triggered always block cycles through the four digits and displays them on the seven-segment display at the update rate driven by *fast_clk.* This is shown below:

```
reg [1:0] digit_switch;

always @ (posedge fast_clk, posedge RESET) begin
        if (RESET)
                digit_switch <= 0;


        //Minute tens' place
        else if (digit_switch == 0) begin
                anode <= 4'b0111;
                cathode <= cathode3;

                if (blinking == 1 && blink_clk)
                        anode <= 4'b1111;

                digit_switch <= digit_switch + 1;


        //Minute ones' place
        end else if (digit_switch == 1) begin
                anode <= 4'b1011;
                cathode <= cathode2;

                if (blinking == 1 && blink_clk)
                        anode <= 4'b1111;

                digit_switch <= digit_switch + 1;


        //Second tens' place
        end else if (digit_switch == 2) begin
                anode <= 4'b1101;
                cathode <= cathode1;

                if (blinking == 2 && blink_clk)
                        anode <= 4'b1111;

                digit_switch <= digit_switch + 1;


        //Second ones' place
        end else if (digit_switch == 3) begin
                anode <= 4'b1110;
                cathode <= cathode0;

                if (blinking == 2 && blink_clk)
                        anode <= 4'b1111;

                digit_switch <= 0;
        end
end
```

2. **module clock(input RESET,**
   input clk,
   output reg clk_2hz,
   output reg clk_1hz,
   output reg fast_clk,
   output reg blink_clk);

The *clock* module takes in *RESET* and the 100 MHz internal clock of the board and returns four divided clocks: *clk_2hz, clk_1hz, fast_clk* and *blink_clk* with frequencies of 2 Hz, 1 Hz, 500 Hz and 1.25 Hz respectively. The 1 Hz clock is used for incrementing *seconds* and *minutes* in the basic counter, the 2 Hz clock is used for incrementing the non-frozen time unit in adjustment mode, the fast clock is used for cycling through the four digits of the seven-segment display to make the display appear continuous to the human eye, and the blink clock is used to blink the incrementing digits in adjustment mode. The clock dividers are implemented using counters, divisor parameters and edge-triggered always blocks. *RESET* sets all clocks to 0.

```
//2 Hz clock
reg [25:0] ctr2 = 0;
parameter dv2 = 26'd25_000_000;
//parameter dv2 = 27'd25;


//1 Hz clock
reg [25:0] ctr1 = 0;
parameter dv1 = 26'd50_000_000;
//parameter dv1 = 27'd50;


//Fast clock
reg [25:0] ctrf = 0;
parameter dvf = 26'd100_000;
//parameter dvf = 27'd2;


//Blinking clock
reg [25:0] ctrb = 0;
parameter dvb = 26'd40_000_000;
```

Divisor parameters

```
//100 MHz -> 1 Hz
always @(posedge clk, posedge RESET) begin
        if (RESET) begin
                ctr1 <= 0;
                clk_1hz <= 0;
        end else if (ctr1 == dv1 - 1) begin
                ctr1 <= 0;
                clk_1hz <= ~clk_1hz;
        end else begin
                ctr1 <= ctr1 + 1;
                clk_1hz <= clk_1hz;
        end
end
```

1 Hz clock divider logic

3. **module counter(input RESET,**
   input ispaused,
   input [2:0] state,
   input clk_1hz,
   input clk_2hz,
   output reg [5:0] minutes,
   output reg [5:0] seconds);

The *counter* module counts the *seconds* and *minutes* in all states of the stopwatch. First, an always block is used to set a *state_clk* register to 1 Hz or 2 Hz depending on whether the stopwatch is in basic counter mode or adjustment mode. Then, an always block triggered by the posedge of *state_clk* or posedge of *RESET* assigns values to *minutes* and *seconds.* If *RESET* is high, *minutes* and *seconds* are both set to 0. If *ispaused* is true (= 1), *minutes* and *seconds* remain

unchanged. If *state* is *basic (= 0)*, *seconds* and *minutes* are incremented normally and both are reset to 0 when expected. If *state* is *adj_min (= 1)*, *seconds* is frozen and *minutes* is incremented. If *state* is *adj_sec (= 2), minutes* is *frozen* and *seconds* is incremented.

4. **module segments(input [3:0] digit, output reg [6:0] cathode);**

The *segments* module takes in a 4-bit *digit* ranging from 0-9 which is to be displayed and returns the cathode signals pattern for it. This is done using an always block as follows:

```
always @* begin
        case (digit)              //ABCDEFG
                4'd0: cathode = 7'b0000001;
                4'd1: cathode = 7'b1001111;
                4'd2: cathode = 7'b0010010;
                4'd3: cathode = 7'b0000110;
                4'd4: cathode = 7'b1001100;
                4'd5: cathode = 7'b0100100;
                4'd6: cathode = 7'b0100000;
                4'd7: cathode = 7'b0001111;
                4'd8: cathode = 7'b0000000;
                4'd9: cathode = 7'b0000100;
                default: cathode = 7'b1111111;
        endcase
        end
```

The 7-bit cathode pattern encodes the seven circuit nodes A-B of each digit.

5. **module debouncer(input clk,**
   input fast_clk,
   input button,
   output bounce_state);

The *debouncer* module debounces the pushbutton input *button* and returns a single debounced pulse of period *fast_clk* every time the pushbutton is pressed. This eliminates issues caused by noise in the button press. This module instantiates the *dff* module thrice. The debouncing is done with multiple flip-flops for metastability.

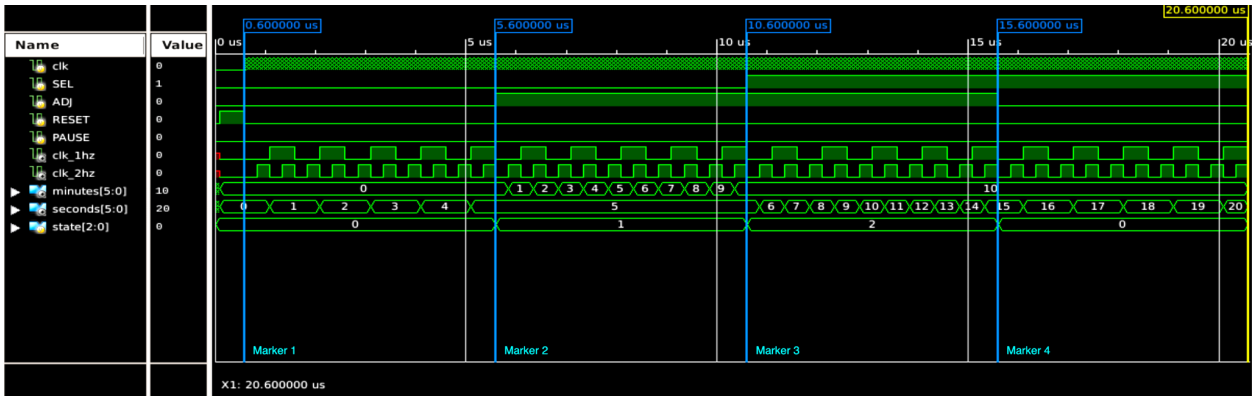
```
module debouncer(
        input clk,
        input fast_clk,
        input button, //input to be debounced
        output bounce_state //debounced switch/button
        );

        wire Q0, Q1, Q2;

        dff d0(fast_clk, button, Q0);
        dff d1(fast_clk, Q0, Q1);
        dff d2(fast_clk, Q1, Q2);

        assign bounce_state = Q1 & ~Q2;

endmodule



module dff(input fast_clk,
                        input D,
                        output reg Q);
always @ (posedge fast_clk) begin
        Q <= D;
end
endmodule
```

6. **module dff(input fast_clk, input D, output reg Q);**

This is a simple D flip-flop module used in debouncing as shown above.

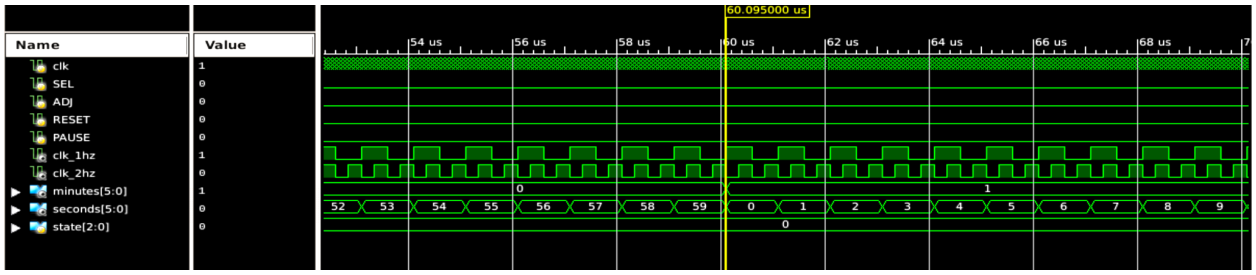
Simulation Documentations

Since the frequencies of the divided clocks are too slow for simulation, I used faster versions of the divided clocks in simulation. Thus, in the following waveform, clk_1hz and clk_2hz actually have frequencies 1 MHz and 2 MHz (instead of 1 Hz and 2 Hz).



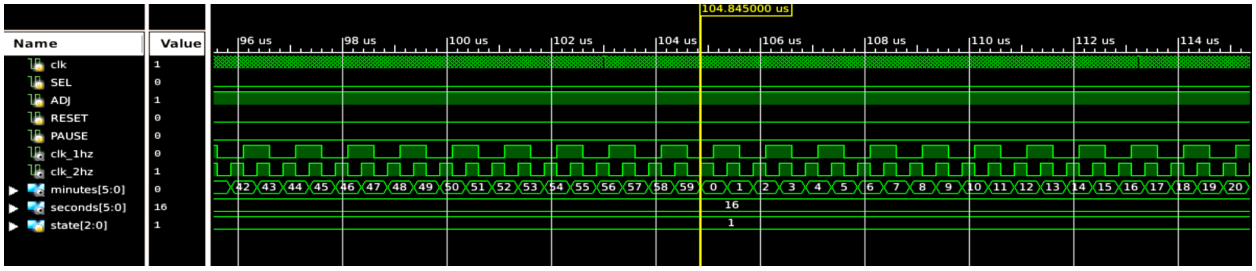
Waveform showing all states

- Marker 1: RESET turns off and clk, clk_1hz and clk_2hz start incrementing. By default, the stopwatch is in the *basic* (= 0) state so when RESET is turned off, seconds and minutes start incrementing normally.
- Marker 2: Here ADJ is toggled on and SEL stays off so the *state* changes to *adj_min* (= 1). At this point, *seconds* is frozen and *minutes* is incremented at 2 Hz.
- Marker 3: ADJ remains on and SEL is toggled on so the *state* changes to *adj_sec* (= 2). Now *minutes* is frozen and *seconds* is incremented at 2 Hz.
- Marker 4: ADJ and SEL are both off so the stopwatch returns to the *basic* state where it continues incrementing the stored *seconds* and *minutes* normally.



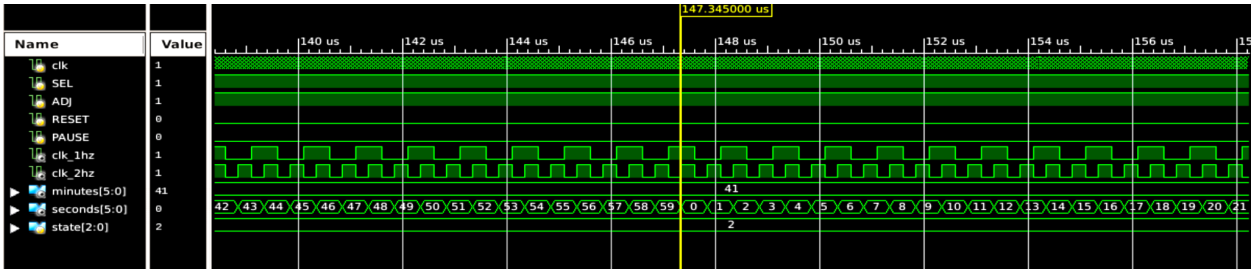
Waveform 1 of stopwatch in basic state (state = 0)

In the above waveform, *seconds* is incrementing at 1 Hz; when it hits 59, it is reset to 0 and *minutes* is incremented to 1 at the next posedge of *clk_1hz.*



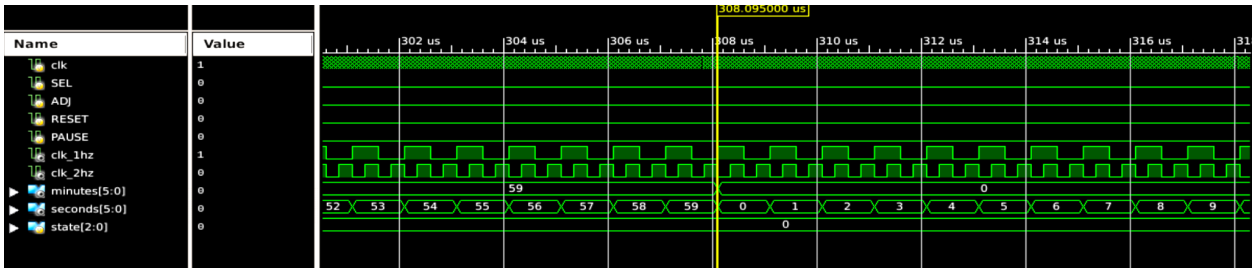
Waveform of stopwatch in adj_min state (state = 1)

In the above waveform, *seconds* is frozen (at 16 seconds) and *minutes* is incrementing at 2 Hz, which is the expected behavior in minutes adjustment mode. When *minutes* reaches 59, it is reset to 0 at the next posedge of *clk_2hz.*



Waveform of stopwatch in adj_sec state (state = 2)

In the above waveform, *minutes* is frozen (at 2 minutes) and *seconds* is incremented at 2 Hz. When *seconds* reaches 59, it is reset to 0 at the next posedge of *clk_2hz.*



Waveform 2 of stopwatch in basic state (state = 0)

In the above waveform, both *seconds* and *minutes* hit 59 i.e. the counted time is 59:59, in the *basic* state. The stopwatch shows expected behavior and both *minutes* and *seconds* are reset to 0 on the next edge of *clk_1hz*.

Conclusion

Our design implements a finite state machine with three states to create a stopwatch which can act as a basic seconds+minutes counter as well as in adjustment mode where it can freeze either the minutes or seconds counted so far and increment the other. The stopwatch can also be reset so that minutes and seconds are returned to 0, or paused so that the values stored in the minutes and seconds registers remain unchanged until the pause pushbutton is pressed again. The stopwatch demonstrates correct behavior in all 3 states as well as correct reset/pause functionality.

We ran into a few issues while working on this lab. First, when we wrote a debouncer module in a file debouncer.v and added it to the project on Xilinx ISE, for some reason we ran into "unknown module" errors when instantiating the debouncer module and running the simulation. We did not have the same issue on the board. Another issue we had was in the counter, where I found that using non-blocking statements in the edge-triggered always block sometimes led to the counter not resetting and counting past 60 in some cases. This may have also been due to some other minor errors in logic, but when I removed the non-blocking statements and changed the logic slightly, the counter started working properly. Finally, we found that using case statements in the counter module for the different counter functionalities in each state allowed our code to compile with no errors and even worked in simulation but did not work on the board. We replaced case statements with if-else-if statements and that solved most of the problems.

My lab partner Vivian and I either worked on this lab together in class or I worked on it alone at home. My contribution was the stopwatch, clock, counter and segments modules and Vivian's contribution was the debouncing modules. We both worked on board debugging together in lab.

References

1. https://www.fpga4student.com/2017/04/simple-debouncing-verilog-code-for.html
2. https://digilent.com/reference/programmable-logic/nexys-3/reference-manual