

Simulation Exercise 7
Digital Stopwatch
Input Debouncing & Synchronization Counters
ELEE 2640

Benyamain YACOOB

March 30, 2024

Date Performed: March 19, 2024

Partners: Ara OLADIPO
Andre PRICE

Instructor: Professor PAULIK



Contents

1 Objectives	3
2 Problem Statement	3
3 Materials	3
4 Multidigit Displays	3
4.1 Task #1	3
4.2 Task #2	5
4.3 Results and Analysis Discussion	6
5 Digital Stopwatch	7
5.1 Debouncer	7
5.2 Stopwatch	8
5.3 Results and Analysis Discussion	10
6 Results and Conclusion	11
7 Group Contributions	11

1 Objectives

First Objective

Explore the development of counters, frequency dividers, and input debouncers.

Second Objective

Implement clock dividers, debouncing, and single pulse circuits in SystemVerilog.

Third

Simulate a four-digit stopwatch circuit with Vivado.

2 Problem Statement

For this simulation exercise, you will have an opportunity to test your familiarity with SystemVerilog by converting and implementing a tutorial from the web and then using the resulting code to develop a four-digit stopwatch system. Along the way, you will explore input synchronization and debouncing, and counter implementation.

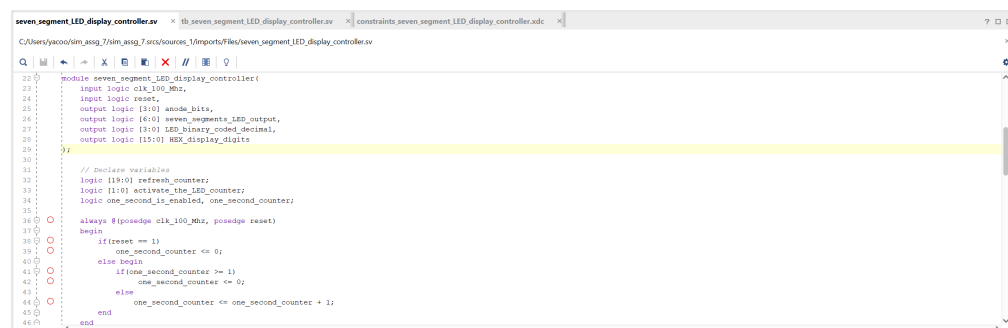
3 Materials

Xilinx integrated synthesis environment (ISE)

4 Multidigit Displays

4.1 Task #1

Using the code from the provided tutorial, we were able to create the seven segment display controller. See below.



```
22 module seven_segment_LED_display_controller(
23     input logic clk_100_MHz,
24     input logic reset,
25     output logic [3:0] anode_bits,
26     output logic [4:0] seven_segments_LED_output,
27     output logic [3:0] LED_binary_coded_decimal,
28     output logic [15:0] HEX_display_digits
29 )
30
31 // Declare variables
32 logic [19:0] refresh_counter;
33 logic [1:0] activate_the_LED_counter;
34 logic one_second_is_enabled, one_second_counter;
35
36 always @(posedge clk_100_MHz, posedge reset)
37 begin
38     if(reset == 1)
39         one_second_counter <= 0;
40     else begin
41         if(one_second_counter >= 1)
42             one_second_counter <= 0;
43         else
44             one_second_counter <= one_second_counter + 1;
45     end
46 end
```

Figure 1: Seven Segment Display Controller Code Implementation

```

seven_segment_LED_display_controller.av constraints_seven_segment_LED_display_controller.xdc tb_seven_segment_LED_display_controller.av
C:/Users/yacoo/sim_essg_7/sim_essg_7/srcs/constrs_1/imports/files/constraints_seven_segment_LED_display_controller.xdc

1: # Clock signal
2: set_property PACKAGE_PIN W5 [get_ports clk_100_Mhz]
3: set_property IOSTANDARD LVCMOS33 [get_ports clk_100_Mhz]
4: create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk_100_Mhz]
5:
6: # Reset signal
7: set_property PACKAGE_PIN R2 [get_ports reset]
8: set_property IOSTANDARD LVCMOS33 [get_ports reset]
9:
10: # Seven segment LED outputs
11: set_property PACKAGE_PIN W7 [get_ports (seven_segments_LED_output{6})]
12: set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output{6})]
13: set_property PACKAGE_PIN W6 [get_ports (seven_segments_LED_output{5})]
14: set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output{5})]
15: set_property PACKAGE_PIN U8 [get_ports (seven_segments_LED_output{4})]
16: set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output{4})]
17: set_property PACKAGE_PIN V8 [get_ports (seven_segments_LED_output{3})]
18: set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output{3})]
19: set_property PACKAGE_PIN U5 [get_ports (seven_segments_LED_output{2})]
20: set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output{2})]
21: set_property PACKAGE_PIN V5 [get_ports (seven_segments_LED_output{1})]
22: set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output{1})]
23: set_property PACKAGE_PIN U7 [get_ports (seven_segments_LED_output{0})]
24: set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output{0})]
25:
26: # Anode bits

```

Figure 2: Constraints File

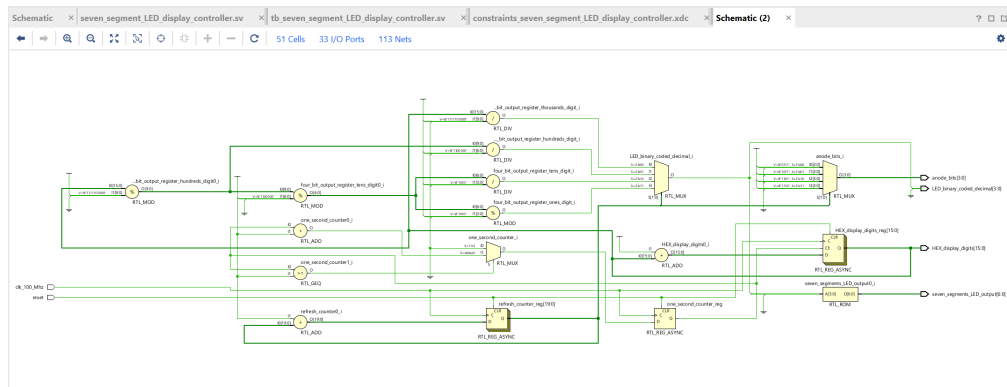


Figure 3: Seven Segment Display Controller Elaborated Design Schematic

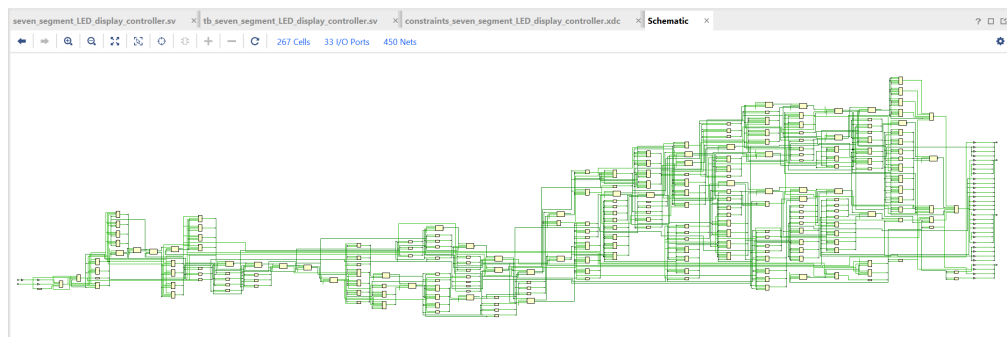


Figure 4: Seven Segment Display Controller Synthesis Design Schematic

4.2 Task #2

```

seven_segment_LED_display_controller.av | tb_seven_segment_LED_display_controller.av | constraints.seven_segment_LED_display_controller.adc | Untitled 6
C:\Users\yaco\sim_avg_7\sim_avg_7\reports\file\tb_seven_segment_LED_display_controller.av

19 //
20 ///////////////////////////////////////////////////////////////////
21
22 module tb_seven_segment_LED_display_controller()
23
24 // Declare testbench signals
25 logic clk;
26 logic reset;
27 logic [3:0] anode_bits;
28 logic [6:0] seven_segments;
29 logic [3:0] ones_digit, tens_digit, hundreds_digit, thousands_digit;
30 logic [3:0] LED_binary_coded_decimal;
31 logic [15:0] HEX_display_digits;
32
33 // Instantiate the seven_segment_LED_display_controller module
34 seven_segment_LED_display_controller dut (
35     .clk_100_MHz(clk),
36     .reset(reset),
37     .anode_bits(anode_bits),
38     .seven_segments_LED_output(seven_segments),
39     .LED_binary_coded_decimal(LED_binary_coded_decimal),
40     .HEX_display_digits(HEX_display_digits)
41 );
42
43 assign ones_digit = HEX_display_digits[3:0];

```

Figure 5: Seven Segment Display Controller Testbench Code

Tcl Console	Messages	Log
<pre> Time: 8370000, Ones Digit: 2, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1101, Segment: 1001111 Time: 8410000, Ones Digit: 3, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1110, Segment: 1001111 Time: 8450000, Ones Digit: 4, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 0000001 Time: 8490000, Ones Digit: 5, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1011, Segment: 1001111 Time: 8530000, Ones Digit: 6, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1101, Segment: 1001111 Time: 8570000, Ones Digit: 7, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1110, Segment: 1001111 Time: 8610000, Ones Digit: 8, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 0000001 Time: 8650000, Ones Digit: 9, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1011, Segment: 0000001 Time: 8690000, Ones Digit: 10, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1101, Segment: 0000001 Time: 8730000, Ones Digit: 11, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1110, Segment: 1001111 Time: 8770000, Ones Digit: 12, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 0000001 Time: 8810000, Ones Digit: 13, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1011, Segment: 0000001 Time: 8850000, Ones Digit: 14, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1101, Segment: 0000001 Time: 8890000, Ones Digit: 15, Tens Digit: 12, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1110, Segment: 1001111 Time: 8930000, Ones Digit: 0, Tens Digit: 13, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 0000001 Time: 8970000, Ones Digit: 1, Tens Digit: 13, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1011, Segment: 0000001 </pre>		

Figure 6: Seven Segment Display Controller Testbench TCL Console

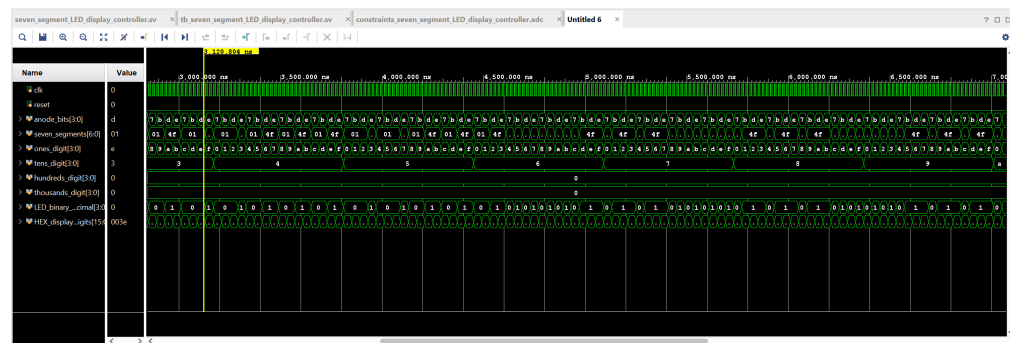


Figure 7: Seven Segment Display Controller Testbench Waveform

4.3 Results and Analysis Discussion

The purpose of this section was to learn about the development of counters, frequency dividers, input debouncers, and implement clock dividers within SystemVerilog, utilizing Vivado and testbenches.

This section had its abundant share of complexity, made obvious as the group made efforts to finish Simulation Exercise #7. The main positive, and the only real positive, displayed in this section was the background information. The tutorial supplemented to the group helped provide a simple foundation: the essence of what we were attempting required a similar function to a carry adder, seeing the values propagate in conjunction with the seven segment display. The problems with the tutorial, however, were understanding the usage of *LED_BCE* and the circuitry implementation. So, although the tutorial was useful in introducing the topic, it was not a practical tool or resource that could be utilized. In order to circumvent some-

what harrowing circumstances, the group met with the teacher assistants (TAs) to obtain guidance and receive clarity on the issues we were having. Through this, we were able to separate and modularize the circuit, allowing us to easier understand the output, and see the changes from the refresh counter and anode bit representations being reflected on the resulting segment outputs. After understanding this, we were able to see how all the internal mechanisms were functioning together. To validate this and also follow extreme programming (XP) practices, we used a testbench to see the output of components. We see that our outputs respond to the positive edge of the 100 *Mhz* clock and reset signals accordingly. Lastly, we increased the simulation time so that we could see the prolonged simulation of the circuit. This allowed us to better verify the functioning of our circuits, and when combined with the testbench TCL console outputs, we were able to conclude that we had a fully functional circuit.

5 Digital Stopwatch

5.1 Debouncer

```
constraints_stopwatch.xdc | debounce.v | stopwatch.v | tb_stopwatch.v
C:/Users/yacoo/sim_essg_7/sim_essg_7/src/sources_1/imports/Files/debounce.v

22
23 module debounce(
24     input logic clk,
25     input logic negated_reset,
26     input logic btn_input,
27     output logic debounce_output
28 );
29
30 // Declare register state and next state
31 logic [10:0] register_state;
32 logic [10:0] next_state;
33 logic first_d_flip_flop, second_d_flip_flop;
34 logic add_state, reset_state;
35
36 // Calculate reset and add states
37 assign reset_state = (first_d_flip_flop ^ second_d_flip_flop);
38 assign add_state = ~(register_state[10]);
39
40 // State transition logic
41 always @(reset_state, add_state, register_state) begin
42     case((reset_state, add_state))
43         2'b00:
44             next_state <= register_state;
45     endcase
46 end
```

Figure 8: Debouncer Code Implementation

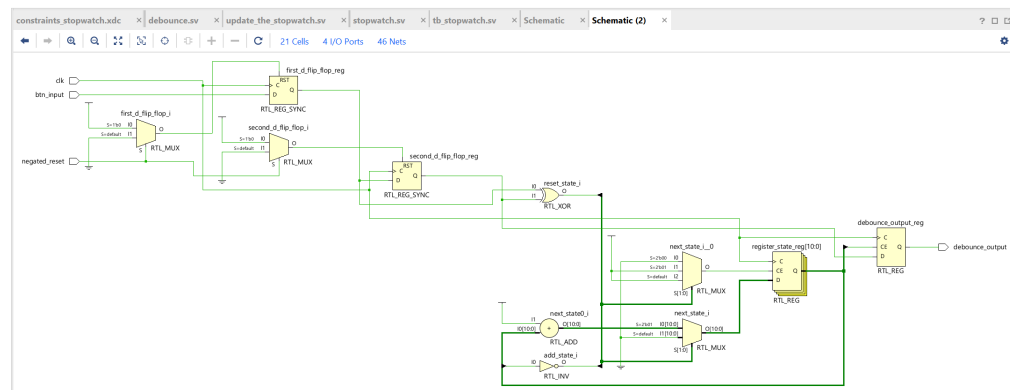


Figure 9: Debouncer Elaborated Design Schematic

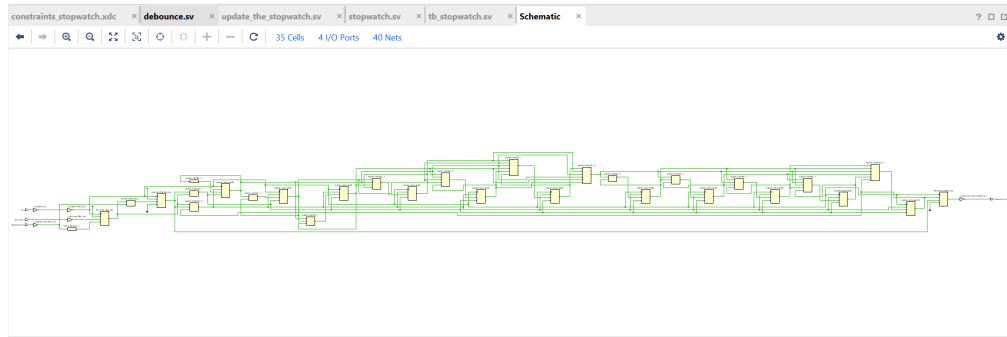


Figure 10: Debouncer Synthesis Design Schematic

5.2 Stopwatch

```
constraints_stopwatch.xdc | debounce.v | stopwatch.v | tb_stopwatch.v
C:/Users/yacoo/sim_assg_7/sim_assg_7/srcs/constes_1/imports/Files/constraints_stopwatchxdc

1 // Clock signal
2 set_property PACKAGE_PIN M5 [get_ports clk]
3 set_property IOSTANDARD LVCMOS33 [get_ports clk]
4 create_clock -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
5
6 // Switches
7 set_property PACKAGE_PIN V17 [get_ports (start)]
8 set_property IOSTANDARD LVCMOS33 [get_ports (start)]
9 set_property PACKAGE_PIN V16 [get_ports (reset)]
10 set_property IOSTANDARD LVCMOS33 [get_ports (reset)]
11
12 //7 segment display
13 set_property PACKAGE_PIN W7 [get_ports (seven_segments_LED_output[6])]
14 set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output[6])]
15 set_property PACKAGE_PIN W6 [get_ports (seven_segments_LED_output[5])]
16 set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output[5])]
17 set_property PACKAGE_PIN U8 [get_ports (seven_segments_LED_output[4])]
18 set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output[4])]
19 set_property PACKAGE_PIN V8 [get_ports (seven_segments_LED_output[3])]
20 set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output[3])]
21 set_property PACKAGE_PIN U5 [get_ports (seven_segments_LED_output[2])]
22 set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output[2])]
23 set_property PACKAGE_PIN V5 [get_ports (seven_segments_LED_output[1])]
24 set_property IOSTANDARD LVCMOS33 [get_ports (seven_segments_LED_output[1])]
```

Figure 11: Constraints File

```
constraints_stopwatch.xdc | debounce.v | stopwatch.v | tb_stopwatch.v
C:/Users/yacoo/sim_assg_7/sim_assg_7/srcs/sources_1/imports/Files/stopwatch.v

22 module stopwatch(
23     input logic clk,
24     input logic reset,
25     input logic start,
26     output logic [3:0] anode_bits,
27     output logic [7:0] seven_segments_LED_output,
28     output logic [3:0] LED_binary_coded_decimal,
29     output logic [15:0] HEX_display_digits
30 );
31
32     seven_segment_LED_display_controller seven_segment_display(
33         .clk_100_mhz(clk),
34         .reset(reset),
35         .anode_bits(anode_bits),
36         .seven_segments_LED_output(seven_segments_LED_output),
37         .LED_binary_coded_decimal(LED_binary_coded_decimal),
38         .HEX_display_digits(HEX_display_digits));
39
40     // Instantiate debounce modules for start and reset buttons
41     debounce start_btn(
42         .clk(clk),
43         .negated_reset(!start),
44         .btn_input(start),
45         .debounced_output(start_debounced));
46
47     debounce reset_btn(
48         .clk(clk),
49         .negated_reset(!reset),
50         .btn_input(reset),
51         .debounced_output(reset_debounced));
52
53     logic [3:0] anode_bits_reg;
54     logic [7:0] seven_segments_LED_output_reg;
55     logic [3:0] LED_binary_coded_decimal_reg;
56     logic [15:0] HEX_display_digits_reg;
57
58     anode_bits_reg <= anode_bits;
59     seven_segments_LED_output_reg <= seven_segments_LED_output;
60     LED_binary_coded_decimal_reg <= LED_binary_coded_decimal;
61     HEX_display_digits_reg <= HEX_display_digits;
62
63     logic [3:0] anode_bits_out;
64     logic [7:0] seven_segments_LED_output_out;
65     logic [3:0] LED_binary_coded_decimal_out;
66     logic [15:0] HEX_display_digits_out;
67
68     anode_bits_out <= anode_bits_reg;
69     seven_segments_LED_output_out <= seven_segments_LED_output_reg;
70     LED_binary_coded_decimal_out <= LED_binary_coded_decimal_reg;
71     HEX_display_digits_out <= HEX_display_digits_reg;
72
73     anode_bits <= anode_bits_out;
74     seven_segments_LED_output <= seven_segments_LED_output_out;
75     LED_binary_coded_decimal <= LED_binary_coded_decimal_out;
76     HEX_display_digits <= HEX_display_digits_out;
77
78 endmodule
```

Figure 12: Stopwatch Code Implementation

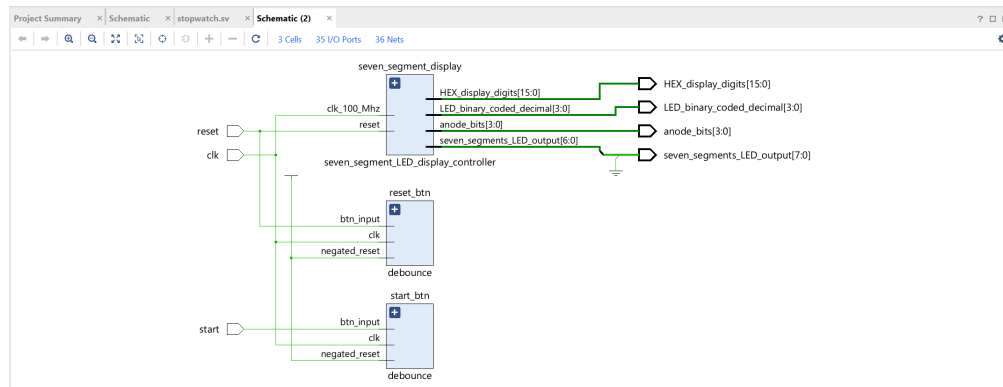


Figure 13: Stopwatch Elaborated Design Schematic

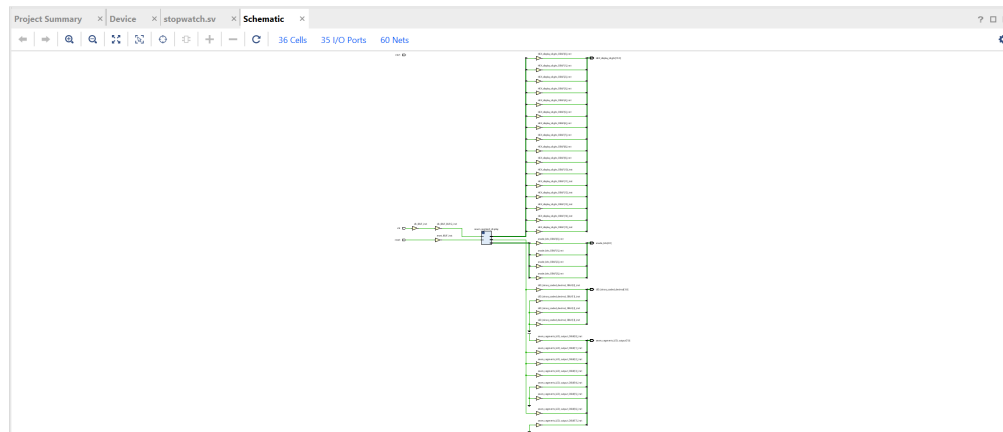


Figure 14: Stopwatch Synthesis Design Schematic

```

constraints_stopwatch.xdc | debounce.v | stopwatch.v | tb_stopwatch.v
C:/Users/yacoo/sim_assg_7/sim_assg_7/srcs/sim_1/imports/Files/tb_stopwatch.v

22: module tb_stopwatch();
23:
24: // Declare testbench signals
25: logic clk;
26: logic reset, start;
27: logic [3:0] anode_bits;
28: logic [6:0] seven_segments;
29: logic [3:0] ones_digit, tens_digit, hundreds_digit, thousands_digit;
30: logic [3:0] LED_binary_coded_decimal;
31: logic [15:0] HEX_display_digits;
32:
33: // Instantiate the stopwatch module
34: stopwatch dut (
35:     .clk(clk),
36:     .reset(reset),
37:     .start(start),
38:     .anode_bits(anode_bits),
39:     .seven_segments_LED_output(seven_segments),
40:     .LED_binary_coded_decimal(LED_binary_coded_decimal),
41:     .HEX_display_digits(HEX_display_digits));
42:
43: assign ones_digit = HEX_display_digits[3:0];
44: assign tens_digit = HEX_display_digits[7:4];
45: assign hundreds_digit = HEX_display_digits[11:8];
46: assign thousands_digit = HEX_display_digits[15:12];
47:
48: endmodule

```

Figure 15: Stopwatch Testbench Code

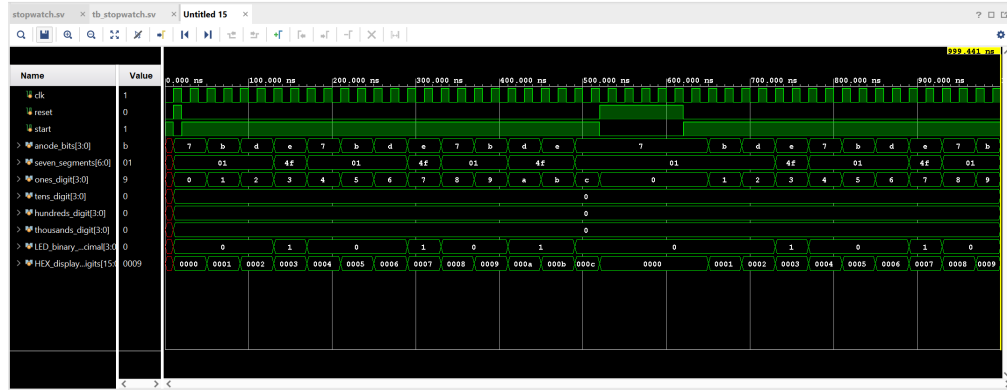


Figure 16: Stopwatch Testbench Waveform

```
Tcl Console  Messages  Log
[Icons]
Time: 0, Start: 1, Reset: 0, Ones Digit: x, Tens Digit: x, Hundreds Digit: x, Thousands Digit: x, Anode Bits: xxxx, Segment: xxxxxxxx
Time: 10000, Start: 0, Reset: 1, Ones Digit: 0, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 00000001
Time: 20000, Start: 1, Reset: 0, Ones Digit: 0, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 00000001
Time: 50000, Start: 1, Reset: 0, Ones Digit: 1, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1011, Segment: 00000001
Time: 90000, Start: 1, Reset: 0, Ones Digit: 2, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1101, Segment: 00000001
Time: 130000, Start: 1, Reset: 0, Ones Digit: 3, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1110, Segment: 10011111
Time: 170000, Start: 1, Reset: 0, Ones Digit: 4, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 00000001
Time: 210000, Start: 1, Reset: 0, Ones Digit: 5, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1011, Segment: 00000001
Time: 250000, Start: 1, Reset: 0, Ones Digit: 6, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1101, Segment: 00000001
Time: 290000, Start: 1, Reset: 0, Ones Digit: 7, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1110, Segment: 10011111
Time: 330000, Start: 1, Reset: 0, Ones Digit: 8, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 00000001
Time: 370000, Start: 1, Reset: 0, Ones Digit: 9, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1011, Segment: 00000001
Time: 410000, Start: 1, Reset: 0, Ones Digit: 10, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1101, Segment: 10011111
Time: 450000, Start: 1, Reset: 0, Ones Digit: 11, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 1110, Segment: 10011111
Time: 490000, Start: 1, Reset: 0, Ones Digit: 12, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 00000001
Time: 520000, Start: 0, Reset: 1, Ones Digit: 0, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 00000001
Time: 620000, Start: 1, Reset: 0, Ones Digit: 0, Tens Digit: 0, Hundreds Digit: 0, Thousands Digit: 0, Anode Bits: 0111, Segment: 00000001
```

Figure 17: Stopwatch Testbench TCL Console

5.3 Results and Analysis Discussion

This section encompassed simulating a 4-digit stopwatch circuit in SystemVerilog with Vivado. This entailed creating a testbench to exercise the circuits we created and verify that it was correct.

Similar to the statements and conclusions reached in the first section, this section also laid out its share of difficulties and complexities. When coding, we had to first consider implementing the synchronization and debouncer and test that module before moving on to the stopwatch. Once we got that to work, we worked on the stopwatch module that integrated the components said above, and implemented buttons that serve to start, stop, or reset the stopwatch timer. Eventually, to get this to work, we exhausted its functionality with a testbench, but the process for creating the testbench module was not clear. We had to go through all forms of trial and error to figure out how you can simulate a testbench for a stopwatch, especially considering the timer buttons. The general notion of the buttons was also not clear, because the team did not know how buttons can be used in simulation exercises in the same way we might understand them when looking at the software side of

it (implementing a button UI display for it and the functionality for it works as it should) as this exercise approaches the design of the stopwatch from a hardware perspective.

Question: *Explain how a 10 Hz clock will allow us to measure 0.100 seconds.*

Answer: *A 10 Hz clock has a period of 0.100 seconds, which means that one complete cycle of the clock signal takes 0.100 seconds. Since the period is the amount of time required for one full cycle, a 10 Hz clock will allow us to measure intervals of 0.100 seconds by counting the number of clock cycles. Each cycle represents 0.100 seconds, so by counting the cycles, we can accurately measure time in multiples of 0.100 seconds.*

6 Results and Conclusion

This section discusses the results of the Vivado simulation exercise. The discussion provides in-depth explanations of the results and any discrepancies between the results and theoretical expectations. It also explores potential sources of error and discusses the accuracy of the tests, the debugging process, and what was learned from these experiences.

Simulation Exercise #7 has succeeded in requiring the group to put in strenuous work just to complete the code. With the tutorial not being a stable resource in completing the simulation exercise. There were debates and discussions about the content and the formatting of the respective content in this document because of the complexity of the material provided within this project. Although we were able to finish the exercise, it was very difficult to get a working code and implementation that met the expectations of the report. On the contrary, the exercise did not particularly triumph in its purpose of finding its identity as a resource for study material. To put it simply, the simulation exercise was too complex.

Through hard work and perseverance, the group was able to code the requirement elements needed and type a well-written report. However, the rigor displayed in this simulation exercise left much to be desired. We hope that for future reference, more readable tutorials are provided to ensure a final project that is of high quality to all parties involved, whether that be the professor who reads the reports, the teacher assistants who give input, or group members who do the work to the best of their ability.

7 Group Contributions

The work for this simulation assignment was divided among three individuals: Benyamain Yacoob, Andre Price Jr, and Ara Oladipo. Ara and Benyamain collaborated using Vivado software and writing in SystemVerilog. It should be noted, however, that this was a group effort and all members were present to answer any questions

related to the circuits. Andre led the effort in organizing the document, formatting it correctly, and writing the analysis for this report.