EC311 Alexander Salmi
Lab #4 U04201274

Lab 4 Report

Objective:

The objective of this lab was to continue learning about sequential logic design and it's implementation in Verilog, as well as to design a 7-segment display driver for the FPGA board. The driver is meant to display the results of the counter from the previous lab on the FPGA board's 7 segment display.

Methodology:

This lab has a lot of individual components that need to be made before consolidating them all into the top level 7-segment display module. These components are a 7-segment decoder, a 16-bit counter, a display control, two clock dividers, and a debouncer module. The debouncer module was simple, since I was just able to reuse the debouncer module I designed for the previous lab.

After getting the debouncer from my previous lab to use in this one, I decided to continue with making the 16-bit counter. I chose to start with this because it is very similar to the counter module from the previous lab, except that it needs to count up to 16 bits instead of just 8. This was very simple to update, I just copied my counter module from the last lab, and changed the counter output from an 8-bit output to a 16-bit output. This was enough to ensure that the 16-bit counter would work correctly, counting from 0 to 65,536. The reset functionality still works as it did in the last lab, resetting the count every time reset is 1 at a positive clock edge.

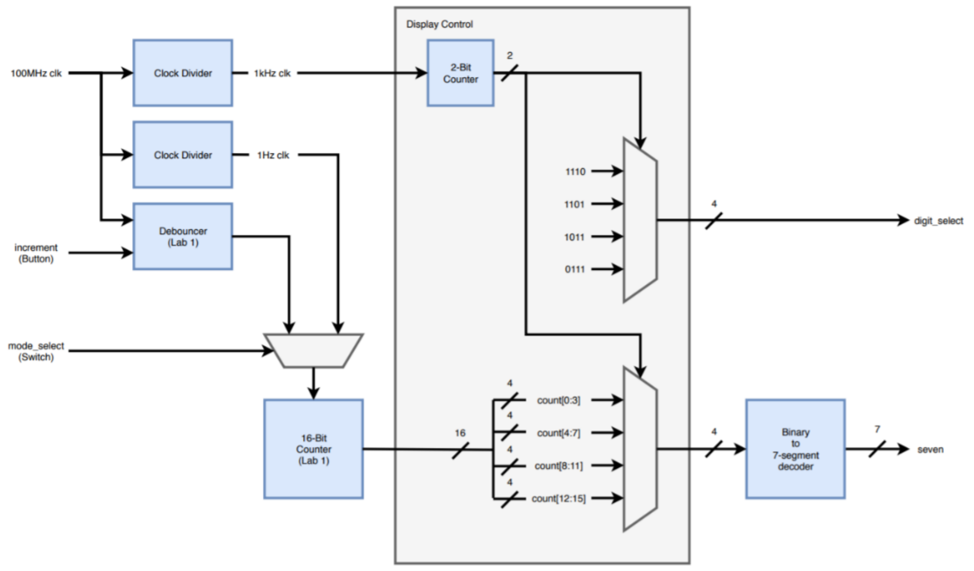
After updating the counter, I moved on to the new components. I decided to start on the two clock dividers next. We needed to create to separate clock dividers for the 7-segment display driver to function correctly, one divider that will output a 1 kHz clock, and one that will output a 1Hz clock. Since the display driver should be able to count wither automatically or manually, the 1Hz clock is used to help implement the automatic counting function. It is used to slow down the count to a pace that is readable to the human eye, approximately 1 increment per second. This module was fairly simple to implement. It included a clk input, a clk_1Hz output, and a integer variable 'count'. Since the FPGA's clock is a 100MHz clock, I had to divide this clock cycle by 100000000 to get a 1Hz clock. To do this, I simply had the count variable increment by 1 every positive edge of the input clock, and when the counter hit 100000000, the output clock (1Hz) would be set to 1. I would then reset the output clock back to 0 on the next input clock cycle, as the 1Hz output should only increment the counter by 1 every cycle. If it were to be left at 1 for 100000000 input clock cycles, the counter would increment much more than intended.

The 1 kHz clock divider was to be used for updating the actual 7-segment displays on the FPGA at a pace that would make the displays look constantly turned on to the human eye. This clock divider is almost identical to the 1Hz one, except that the count only counts up to 100000, since we do not have to divide the clock by as much. This clock was also allowed to last in the 1 position for the full 100000 input clock cycles, as the 7-segment displays need an even clock cycle to display properly.

After the clock dividers, I moved on to the display control module. The purpose of this module was to take in the current count to be displayed, and output the four corresponding hexadecimal values for each of the four 7-segment displays. This module had a clk input (the 1kHz clock), a 16-bit num input, and 4-bit 'digit_select' and 'seven' outputs, as well as a 2-bit counter register. The counter would increment once every positive edge of the input clock cycle, looping around when it hits 4. This counter would then be used as the input of two multiplexers, one for choosing the output for 'digit_select', and one for 'seven'. Since the select input for the multiplexers are the same, they change at the same time, ensuring that the correct hexadecimal value will output to 'seven' at the same time as its corresponding 'digit_select' value. Digit_select can take on four different values (1110, 1101, 1011, 0111) corresponding to each of the four 7-segment FPGA displays. This ensures that the correct bits from the input number will be sent to the correct displays. (The most significant bit of the hexadecimal number will be shown on the righter most display, etc.)

After completing the display control, I moved on to the final module, the seven segment decoder. The module has a 4-bit input, and a 7-bit output. Since each display has seven segments, they have to be encoded to properly display the correct digit. To do this, you have to feed it a 7-bit number, with each bit corresponding to each segment of the display. If the bit is 0, the segment lights up, if it is 1, it does not. To do this, I simply had a switch case based on the input, outputting the correct 7-bit number that will display the hex digit corresponding to the 4-bit input.

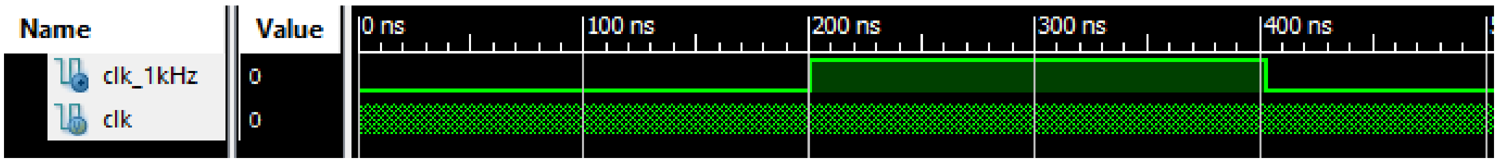
Finally, I had to combine all of the modules into the overall 7-segment display driver module. The modules were connected according to the diagram below.

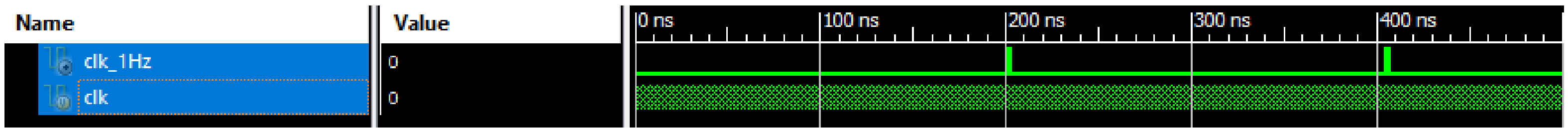


After finishing the design, I uploaded it to the board. The mode_select input was set to one of the board's switches, the clock input to the board's built in clock, the increment and reset inputs to buttons, and the count itself to the four 7-segment displays.

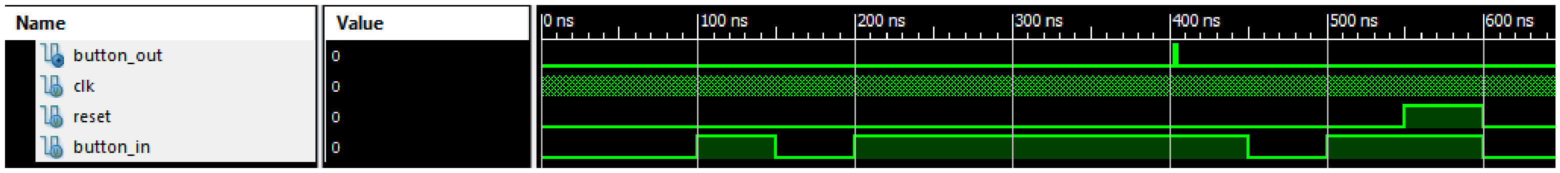
Observation

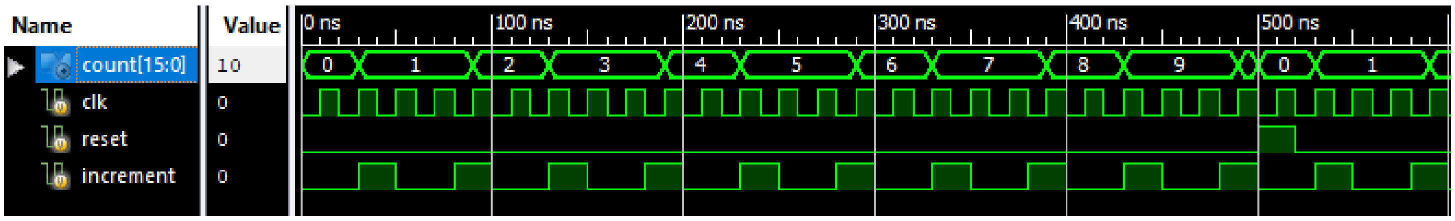
Overall, everything worked as it should have. The two clock dividers correct divided the clock into their correct frequencies, as shown in the timing diagrams below. The 1 kHz clock is an even clock cycle, and the 1Hz clock cycle only briefly outputs a 1, as intended



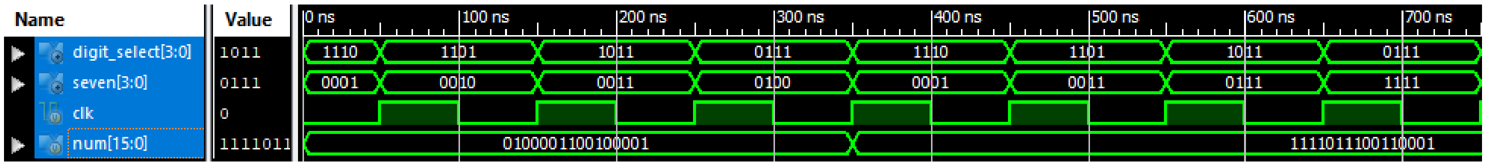


The debouncer and counter16 modules also worked correctly, as they had in the previous lab as well. This is shown in the timing diagrams below.

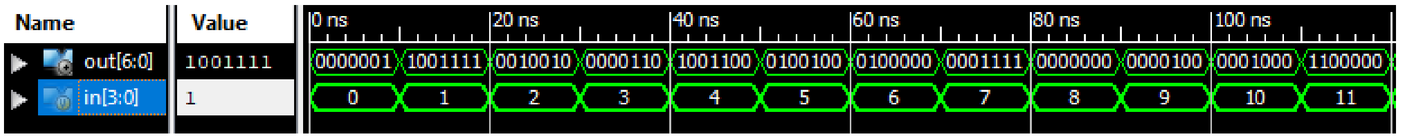




The display control was able to correctly output the 4-bit sequence corresponding the correct 4-bit display digit select number, switching between them at the correct times. This is shown in the timing diagram below.



Finally, the seven segment decoder also worked correctly, outputting the correct sequence for the hexadecimal characters based on the 4-bit input, as shown in the timing diagram below.



Once I uploaded the design to the FPGA, it worked very well. It counted automatically when the switch was up, and the manual counting worked as it did in the previous lab when the switch was down.

Conclusion

Overall, this lab went very well. I learned how to design more complicated designs that require a lot more modules than I am used to. I also learned the use of clock dividers, and how to implement them, as well as how to simulate an automatic counter, all very interesting and useful skills.

Alexander Salmi
U04201274

Code

Debouncer Module:

```
module debouncer(
   input clk,
   input reset,
   input button_in,
   output reg button_out = 0
   );
reg output_exists, count_start;
        integer count;

        always@(posedge clk or posedge reset)begin
                if(reset == 1)begin
                                count = 0;
                                output_exists = 0;
                                count_start = 0;
                                button_out <= 0;
                end else begin
                        if(button_in == 0)begin
                                count = 0;
                                output_exists = 0;
                                count_start = 0;
                        end else begin
                                if(output_exists == 1)begin
                                        button_out <= 0;
                                end else begin
                                        if(count_start == 0)begin
                                                count_start = 1;
                                        end else begin
                                                if(count == 1000000)begin
                                                        button_out <= 1;
                                                        count_start = 0;
                                                        count = 0;
                                                        output_exists = 1;
                                                end else begin
                                                        count = count + 1;
                                                end
                                        end
                                end
                        end
                end
        end //always
endmodule
```

Alexander Salmi
U04201274

Debouncer Testbench:

```
module debouncer_TB;

	// Inputs
	reg clk;
	reg reset;
	reg button_in;

	// Outputs
	wire button_out;

	// Instantiate the Unit Under Test (UUT)
	debouncer uut (
		.clk(clk),
		.reset(reset),
		.button_in(button_in),
		.button_out(button_out)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 0;
		button_in = 0;

		// Wait 100 ns for global reset to finish
		#100 button_in = 1;
		#50 button_in = 0;
		#50 button_in = 1;
		#250 button_in = 0;
		#50 button_in = 1;
		#50 reset = 1;
		#50 reset = 0; button_in = 0;
		#100 button_in = 1;
		#100 button_in = 0;

		// Add stimulus here

	end

	always begin
		#1 clk = ~clk;
	end
endmodule
```

EC311 Alexander Salmi
Lab #4 U04201274

Counter16 Module:

```
module counter16(
   input clk,
   input reset,
   input increment,
   output reg [15:0] count = 16'b0000000000000000
   );

           always@(posedge clk or posedge reset)begin
                      if(reset == 1) begin
                                 count <= 16'b0000000000000000;
                      end else begin
                                 if(increment == 1)begin
                                            count <= count + 16'b0000000000000001;
                                 end
                      end
            end //always
endmodule
```

Alexander Salmi
U04201274

<u>Counter16 Testbench:</u>

```
module counter16_TB;
	// Inputs
	reg clk;
	reg reset;
	reg increment;

	// Outputs
	wire [15:0] count;

	// Instantiate the Unit Under Test (UUT)
	counter16 uut (
		.clk(clk),
		.reset(reset),
		.increment(increment),
		.count(count)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 0;
		increment = 0;

		// Wait 100 ns for global reset to finish
		#500 reset = 1;
		#20 reset = 0;
	end

	always begin
		#10 clk = ~clk;
		#10 clk = ~clk;
		#10 clk = ~clk; increment = ~increment;
		#10 clk = ~clk;
		#10 clk = ~clk; increment = ~increment;
	end
endmodule
```

EC311 Alexander Salmi
Lab #4 U04201274

Clock Divider 1kHz Module:

```
module clock_divider_1kHz(
   input clk,
   output reg clk_1kHz = 0
   );
	integer count = 0;

	always@(posedge clk)begin
		if(count == 100000)begin
			clk_1kHz = ~clk_1kHz;
			count = 0;
		end else begin
			count = count + 1;
		end
	end
endmodule
```

Clock Divider 1kHz Testbench:

```
module clock_divider_1kHz_TB;
	// Inputs
	reg clk;

	// Outputs
	wire clk_1kHz;

	// Instantiate the Unit Under Test (UUT)
	clock_divider_1kHz uut (
		.clk(clk),
		.clk_1kHz(clk_1kHz)
	);

	initial begin
		// Initialize Inputs
		clk = 0;

		// Wait 100 ns for global reset to finish
		#100;
	end

	always begin
		#1 clk = ~clk;
	end

endmodule
```

Clock Divider 1Hz Module:
```
module clock_divider_1Hz(
   input clk,
   output reg clk_1Hz = 0
   );
        integer count = 0;

         always@(posedge clk)begin
                if(clk_1Hz == 1)begin
                        clk_1Hz = 0;
                end else begin
                        if(count == 100000000)begin
                                clk_1Hz = 1;
                                count = 0;
                        end else begin
                                count = count + 1;
                        end
                end
         end
endmodule
```

Clock Divider 1Hz Testbench:
```
module clock_divider_1Hz_TB;
        // Inputs
        reg clk;
        // Outputs
        wire clk_1Hz;

        // Instantiate the Unit Under Test (UUT)
        clock_divider_1Hz uut (
                .clk(clk),
                .clk_1Hz(clk_1Hz)
        );
        initial begin
                // Initialize Inputs
                clk = 0;

                // Wait 100 ns for global reset to finish
                #100;
        end
        always begin
                #1 clk = ~clk;
        end
endmodule
```

Alexander Salmi
U04201274

Display Control Module

```
module display_control(
   input clk,
   input [15:0] num,
   output reg [3:0] digit_select,
   output reg [3:0] seven
   );
	reg[1:0] counter = 2'b00;


	always@(posedge clk)begin
		if(counter == 2'b11)begin
			counter = 2'b00;
		end else begin
			counter = counter + 2'b01;
		end
	end

	always@(*)begin
		case(counter)
			2'b00:begin
				digit_select = 4'b1110;
				seven = num[3:0];
			end
			2'b01:begin
				digit_select = 4'b1101;
				seven = num[7:4];
			end
			2'b10:begin
				digit_select = 4'b1011;
				seven = num[11:8];
			end
			2'b11:begin
				digit_select = 4'b0111;
				seven = num[15:12];
			end
		endcase
	end
endmodule
```

EC311 Alexander Salmi
Lab #4 U04201274

Display Control Testbench

```
module display_control_TB;

	// Inputs
	reg clk;
	reg [15:0] num;

	// Outputs
	wire [3:0] digit_select;
	wire [3:0] seven;

	// Instantiate the Unit Under Test (UUT)
	display_control uut (
		.clk(clk),
		.num(num),
		.digit_select(digit_select),
		.seven(seven)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		num = 16'b0100001100100001;

		// Wait 100 ns for global reset to finish
		#350 num = 16'b1111011100110001;

		// Add stimulus here

	end

	always begin
		#50 clk = ~clk;
	end

endmodule
```

EC311 Alexander Salmi
Lab #4 U04201274

Seven Segment Decoder Module:

```
module seven_segment_decoder(
   input [3:0] in,
   output reg [6:0] out = 7'b1111111
   );

            always@(*)begin
                        case(in)
                                    4'b0000: out = 7'b0000001;
                                    4'b0001: out = 7'b1001111;
                                    4'b0010: out = 7'b0010010;
                                    4'b0011: out = 7'b0000110;
                                    4'b0100: out = 7'b1001100;
                                    4'b0101: out = 7'b0100100;
                                    4'b0110: out = 7'b0100000;
                                    4'b0111: out = 7'b0001111;
                                    4'b1000: out = 7'b0000000;
                                    4'b1001: out = 7'b0000100;
                                    4'b1010: out = 7'b0001000;
                                    4'b1011: out = 7'b1100000;
                                    4'b1100: out = 7'b0110001;
                                    4'b1101: out = 7'b1000010;
                                    4'b1110: out = 7'b0110000;
                                    4'b1111: out = 7'b0111000;
                        endcase
            end
endmodule
```

Seven Segment Display Testbench:

```
module seven_segment_decoder_TB;

	// Inputs
	reg [3:0] in;

	// Outputs
	wire [6:0] out;

	// Instantiate the Unit Under Test (UUT)
	seven_segment_decoder uut (
		.in(in),
		.out(out)
	);

	initial begin
		// Initialize Inputs
		in = 4'b0000;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here

	end

	always begin
		#10 in = in + 4'b0001;
	end

endmodule
```

Alexander Salmi
U04201274

UCF File

```
NET "clk" LOC = v10;
NET "increment" LOC = D9;
NET "reset" LOC = C4;
NET "mode_select" LOC = T10;
NET "digit_select[0]" LOC = N16;
NET "digit_select[1]" LOC = N15;
NET "digit_select[2]" LOC = P18;
NET "digit_select[3]" LOC = P17;
NET "seven[6]" LOC = T17;
NET "seven[5]" LOC = T18;
NET "seven[4]" LOC = U17;
NET "seven[3]" LOC = U18;
NET "seven[2]" LOC = M14;
NET "seven[1]" LOC = N14;
NET "seven[0]" LOC = L14;
```

EC311 Alexander Salmi
Lab #4 U04201274

Lab4 Module:
```
module lab4(
   input clk,
   input increment,
   input mode_select,
   input reset,
   output [3:0] digit_select,
   output [6:0] seven
   );

        wire clk_1kHz, clk_1Hz, debounce_out, counter16_in;
        wire [3:0] seven_seg_in;
        wire [15:0] counter16_out;
        reg increment16;

        clock_divider_1kHz g1(.clk(clk), .clk_1kHz(clk_1kHz));
        clock_divider_1Hz g2(.clk(clk), .clk_1Hz(clk_1Hz));
        debouncer g3(.clk(clk), .reset(reset), .button_in(increment), .button_out(debounce_out));
        counter16 g4(.clk(clk), .reset(reset), .increment(increment16), .count(counter16_out));
        display_control g5(.clk(clk_1kHz), .num(counter16_out), .digit_select(digit_select),
.seven(seven_seg_in));
        seven_segment_decoder g6(.in(seven_seg_in), .out(seven));

        always@(*)begin
                case(mode_select)
                        0:increment16 = debounce_out;
                        1:increment16 = clk_1Hz;
                endcase
        end
endmodule
```