Lab 3 Report

Objective:

The objective of this lab was to familiarize ourselves with sequential logic design and its implementation in Verilog, as well as to learn how to design a simple 8-bit counter and implement it on the FPGA.

Methodology:

To create the 8-bit counter, we first had to go through and create all the different components of the counter. This meant creating the actual counter module, which outputs the updated count based on the current number and the increment input, as well as creating a debouncer, which is a module used to clean up the input signal from pushing the button on the FPGA. Without the debouncer, pressing the button could cause several active inputs for each button press, so the debouncer fixes this by making sure the counter is only incremented once per button push.

I decided to start out with the debouncer. At first I was confused how to design it, as I didn't completely understand the purpose of it, but once I figured it out it was pretty simple. The debouncer has a clk, reset, and button_in input, and a button_out output. I also included an integer register called count, which would count from 0 to 1000000 while the button is being pushed in. Every positive edge of the clock, the debouncer checks for if button_in has a positive input. If the button_in input is 1, meaning that the button is being pushed down, the counter increments by 1. When the counter hits 1000000, the debouncer will output 1 to button_out for one clock cycle, and then remain at 0 until the button is released. If the button_in input becomes 0 at any positive edge of the clock, the count resets back to 0. Also, if the reset input is 1 at the positive edge of the clock, the count also resets to 0. The count register allows for the board to ignore any erroneous input signals that occur when pressing the button, only outputting once per button press.

I then moved on to the counter itself, which was a much simpler module to design. The module also has a clk, reset, and increment input, and an 8-bit count output. The count output is initialized as 0. Then, every positive clock edge, if the increment input is 1, the output gets incremented by 1. Once the output hits its maximum value, and the increment input is 1, it resets back to 0. Furthermore, if the reset input is 1 at any positive clock edge, the output also resets to 0.

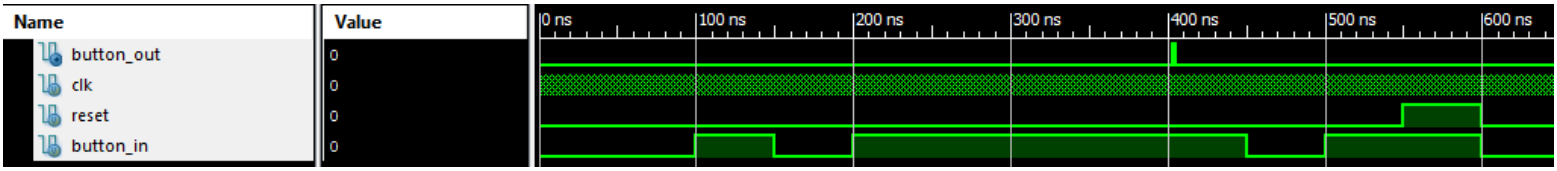
After designing the two internal components of the 8-bit counter, I moved on to combining them to create the full final module. The inputs of the module has a clk, reset, and increment input, and an 8-bit count output. The clk and reset inputs are connected straight to the clk and reset inputs of the debouncer and counter modules. The increment counter is connected to the button_in input of the debouncer. The button_out output of the debouncer is

then connected to the increment input of the counter, and the 8-bit count output of the counter is the same 8-bit output of the overall 8-bit counter module.

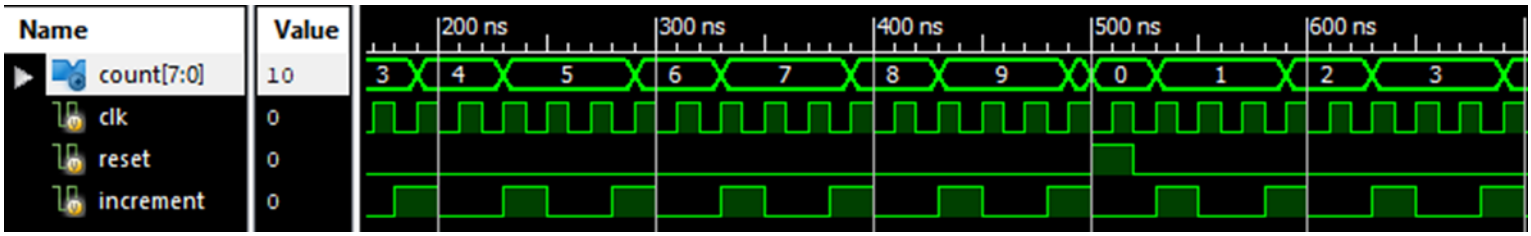
Finally, I had to connect the inputs and outputs to buttons and LED's on the FPGA board. To do this, I simply had the FPGA's clock act as the clk input to the module, and two of the push buttons on the board act as the increment and reset buttons. The 8-bit count output was connected to the eight LED's on the board, each LED representing one bit of the output.

Observations:

Overall, the functions of the 8-bit counter worked as intended. The debouncer successfully ignored button inputs that were shorter than 1000000 clock cycles, and only outputted a 1 once per button push. The reset also correctly resets everything back to its initial state. This is all shown in the timing diagram below. Note: The max count was set to 100 for the purposes of the test bench, to be able to correctly demonstrate all functions of the debouncer, in the actual program, the count counts up to 1000000.



The counter module also worked as intended. It incremented the output once every time the increment input was a 1 at a positive clock edge, looping back to 0 after hitting its maximum value. The reset button also successfully reset the count back to 0 as it is supposed to do. This is all shown in the timing diagram below. The timing diagram does not show every value for the count, as that would take a lot of clock cycles, but enough is shown to demonstrate the correct function of the counter.



The overall 8-bit counter module also worked correctly, as expected. While I do not have a timing diagram for it, I made sure it worked correctly by testing it out on the FPGA board. Every time I pressed the increment button, the count being displayed on the LED's would increment. The count would only increment once every time I pressed the button, showing me that the debouncer was still working correctly. When I pressed the reset button, all of the LED's turned off, meaning that the count reset back to 0 as it is supposed to.

Alexander Salmi
U04201274

Conclusion

Overall, this was a very successful lab. I became a lot more familiar with writing sequential logic in Verilog, and how to implement a clock cycle in my modules. I also learned about the use of a debouncer, and how it can help when designing modules for the FPGA board. Working with the FPGA board was very interesting, as it showed me how what we've been learning in class can be implemented in real world projects and products, which is very useful for my future in engineering.

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
```

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
```

Counter Module:

```
module counter8(
   input clk,
   input reset,
   input increment,
   output reg [7:0] count = 8'b00000000
   );

        always@(posedge clk or posedge reset)begin
                if(reset == 1) begin
                        count = 8'b00000000;
                end else begin
                        if(increment == 1)begin
                                count = count + 8'b00000001;
                        end
                end
        end //always
endmodule
```

Counter Testbench:

```
module counter8_TB;

	// Inputs
	reg clk;
	reg reset;
	reg increment;

	// Outputs
	wire [7:0] count;

	// Instantiate the Unit Under Test (UUT)
	counter8 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
```

Alexander Salmi
U04201274

UCF File

```
NET "increment" LOC = D9;
NET "reset" LOC = C4;
NET "clk" LOC = v10;
NET "count[7]" LOC = T11;
NET "count[6]" LOC = R11;
NET "count[5]" LOC = N11;
NET "count[4]" LOC = M11;
NET "count[3]" LOC = V15;
NET "count[2]" LOC = U15;
NET "count[1]" LOC = V16;
NET "count[0]" LOC = U16;
```

Lab3 Module:

```
module Lab3(
   input clk,
   input reset,
   input increment,
   output [7:0] count
   );
         wire w1;

         debouncer d1(clk, reset, increment, w1);
         counter8 c1(clk, reset, w1, count);

endmodule
```