Objective

The purpose of this lab was to expand on the students familiarity of sequential circuits by exposing them to the inner workings of a binary counter. The lab manual guided the students through desgining a binary up-counter using Verilog. Once designing was complete, the design will be synthesized onto the FPGA board using a student edited UCF file. Finally, the lab will conclude with two important use cases for binary counters, namely clock frequency division and I/O debouncing.

Design

Experiment 1

In **Experiment 1**, we described a synchronous sequential circuit and loaded it onto the FBGA board. We also, gained experienced with counters and clock frequency division.

In this first block of code, we designed a clock divider which would divide the clock into different cycles. This design was synthesized and implemented onto the FBGA board.

Code Block 1: Clock Divider

```
1  `timescale 1ns / 1ps
2  `default_nettype none
3
4  module clock_divider (ClkOut, ClkIn);
5    // Output port needs to be a reg because we will drive it with
6    // a behavioral statement
7    output wire [3:0] ClkOut;
8    input wire ClkIn; // wires can drive regs
9
10   // This is is a parameter that
11   // can be changed at compile time.
12   parameter n = 26;
13
14   // Count bit width is based on n
```

```
15    reg [n-1:0] Count;
16
17    // Simple behavioral construct to describe a counter
18    always @ ( posedge ClkIn ) begin
19      Count <= Count + 1;
20    end
21
22    // Now we need to wire up our ClkOut which is a 4-bit wire
23    // Wire up to most signficant bit
24    assign ClkOut[3:0] = Count[n-1:n-4];
25
26  endmodule // clock_divider
```

Experiment 2

Next, we designed a simple half adder that would take in two inputs and outputs a Sum and Cout.

Code Block 2: Half Adder

```
1  `timescale 1ns / 1ps
2  `default_nettype none
3
4  module half_adder (S,Cout, A,B);
5    // Instantiate the wires
6    output wire S, Cout;
7    input wire A, B;
8
9    // Simple Dataflow logic for Half Adder
10   assign S = A^B;
11   assign Cout = A&B;
12
13 endmodule // half_adder
```

The next design was an up counter. The design implemented and utilized four half adders that were just created. The half adders were used to incrementally add the bits together. The reset would reset the count back to zero. The counter would only increment if the enable was on high.

Code Block 3: Up Counter

```
1  `timescale 1ns / 1ps
2
3  module up_counter (Count, Carry3, En, Clk, Rst);
4          // Instantiate Wires and Reg
5          output reg [3:0] Count;
6          output wire Carry3;
7          input wire En, Clk, Rst;
8
9          //intermediate nets
10         wire [3:0] Carry, Sum;
11
12         // Instantiate and wire up 4 half-adders
13         half_adder ha0(Sum[0], Carry[0], En, Count[0]);
14         half_adder ha1(Sum[1], Carry[1], Carry[0], Count[1]);
15         half_adder ha2(Sum[2], Carry[2], Carry[1], Count[2]);
16         half_adder ha3(Sum[3], Carry[3], Carry[2], Count[3]);
17
18         // Wire up carry 3
19         assign Carry3 = Carry[3];
20         // Positve edge triggered flip-flops for count
21         always @ ( posedge Clk or posedge Rst ) begin
22                 if (Rst) begin //if RST == 1'b1
23                         Count <= 0;
24                 end else begin //otherwise latch sum
25                         Count <= Sum;
```

```
26                      end
27              end
28  endmodule
```

In the next two blocks of code, a Top Level Verilog design was created as well as an accompanying UCF file. The Verilog design first described a MUX that would select a clock signal based on the values of the switch. The clock signals were divided by our clock divider module. We also instantiated four separate up counter modules in order to count up four bits that would output to four separate LEDs.

The UCF file took the Verilog file and synthesized in a way such that the FBGA board would understand. The outputs of the up counter were connected to the LEDs on the FBGA board. The switches were connected to two of the switches on the board. The reset was connected to the South button and enable was wired to North.

Code Block 4: Top Level (Verilog)

```
1  `timescale 1ns / 1ps
2
3  module top_level (LEDs, SWs, North, South, FastClk);
4    // All ports will be wires
5          output wire [4:0] LEDs;
6          input wire FastClk, North, South;
7          input wire [1:0] SWs;
8
9          // Intermediate Nets
10   wire [3:0] Clocks;
11   reg SlowClk; // will use an always blcok for mux
12
13   // behavioral description of a mux which selects
14   // between the four available clock signals
15   always @ ( * ) begin
16     case (SWs) // SWs is a 2-bit bus
17                          // Combinational logic
```

```
18                              2'b00: SlowClk = Clocks[0];
19                              2'b01: SlowClk = Clocks[1];
20                              2'b10: SlowClk = Clocks[2];
21                              2'b11: SlowClk = Clocks[3];
22      endcase
23    end
24
25    // Instantiate Up Counter
26          up_counter upCount (LEDs[3:0], LEDs[4], North,
27                      SlowClk, South);
28
29    // Instantiate the clock divider
30    clock_divider clk_div0(
31         .ClkOut(Clocks),
32         .ClkIn(FastClk)
33       );
34  endmodule // top_level
```

Code Block 5: Top Level (UCF)

```
1  #Switches
2  NET "SWs[0]" LOC = "L13" | IOSTANDARD = LVTTL; #SW0
3  NET "SWs[1]" LOC = "L14" | IOSTANDARD = LVTTL; #SW1
4
5  #Push-buttons
6  NET "NORTH" LOC = "V4" | IOSTANDARD = LVttL | PULLDOWN; #North
7  NET "SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN; #South
8
9  # LEDs
10 NET "LEDs[0]" LOC = "F12" | IOSTANDARD = LVTTL; #LD0
11 NET "LEDs[1]" LOC = "E12" | IOSTANDARD = LVTTL; #LD1
12 NET "LEDs[2]" LOC = "E11" | IOSTANDARD = LVTTL; #LD2
```

```
13 NET "LEDs[3]" LOC = "F11" | IOSTANDARD = LVTTL; #LD3
14 NET "LEDs[4]" LOC = "C11" | IOSTANDARD = LVTTL; #LD0
15
16 NET "FastClk" LOC = "C9" | IOSTANDARD = LVTTL;
17
18 # Define clock period for 50 MHz oscillator
19 NET "FastClk" PERIOD = 20.0ns HIGH 40%;
```

Experiment 3

For the switch bounce and no denounce tests, the code was given and tested.

The following is the design given for with debounce. The input wire goes into two flip flops in order to synchronize the signal and eliminate electrical chatter. Then the signal goes into a binary counter. The LEDs correctly incremented based on the number of times the button was pressed.

Code Block 6: With Debounce

```
1  `timescale 1ns / 1ps
2  `default_nettype none
3
4  module withDebounce(LEDs, Center, Clk);
5
6    // Instantiate 8 LEDs
7    output reg [7:0] LEDs;
8
9    // Instantiate Center (Big Knob) and Clk wire
10   input wire Center, Clk;
11
12   /*-this is a keyword we have not seen yet!*
13    *-as the name implies, it is a parameter *
14    * that can be changed at compile time... */
15   parameter n = 18;
16
```

```
17    // Instantiate intermediate wires
18    wire notMsb, Rst, En, Debounced;
19    reg Synchronizer0, Synchronized;
20
21    // Instantiate n wires for Count
22    reg [n-1:0] Count;
23
24    reg edge_detect0;
25    wire rising_edge;
26
27    /*****************************************/
28    /* Debounce circuitry !!!                */
29    /*****************************************/
30
31    // Pass signal through Two Flip Flops
32    always@(posedge Clk)
33      begin
34        Synchronizer0 <= Center;
35        Synchronized <= Synchronizer0;
36      end
37
38    // Binary Counter
39    always@(posedge Clk)
40        if(Rst)
41            Count <= 0;
42        else if(En)
43            Count <= Count + 1;
44
45    // notMsb is NOT of most significant bit of Count
46    assign notMsb = ~Count[n-1];
47    // Enable is notMsb AND Synchronized
```

```
48    assign En = notMsb & Synchronized;
49    // Reset is NOT Synchronized
50    assign Rst = ~Synchronized;
51    // Debounced is most significant bit of Count
52    assign Debounced = Count[n-1];
53
54    /*****************************************/
55    /* End of Debounce circuitry!!!          */
56    /*****************************************/
57
58    always@(posedge Clk)
59        edge_detect0 <= Debounced;
60
61    // Determine if it is  a rising edge
62    assign rising_edge = ~edge_detect0 & Debounced;
63
64    // If rising edgs, light up the next LEDs
65    always@(posedge Clk)
66        if(rising_edge)
67            LEDs <= LEDs + 1;
68
69  endmodule
```

Results

Experiment 1

This is the results for the clock divider design displayed on the Logic Analyzer provided in lab. It displays the different frequencys of the output COUNT, and the results of clock division.

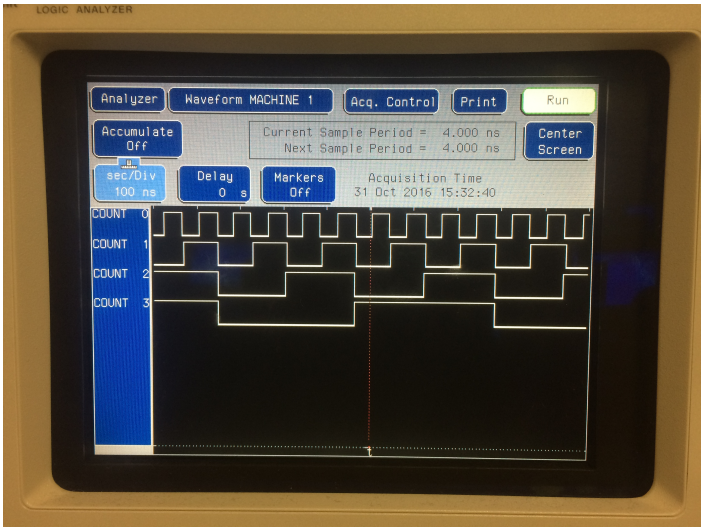

Figure 1: *Clock Divider on Logic Analyzer*

In the next figure, the period for each clock cycle was calculated by determining the difference between the green and yellow. In this case, COUNT3 clock cycle was calculated.

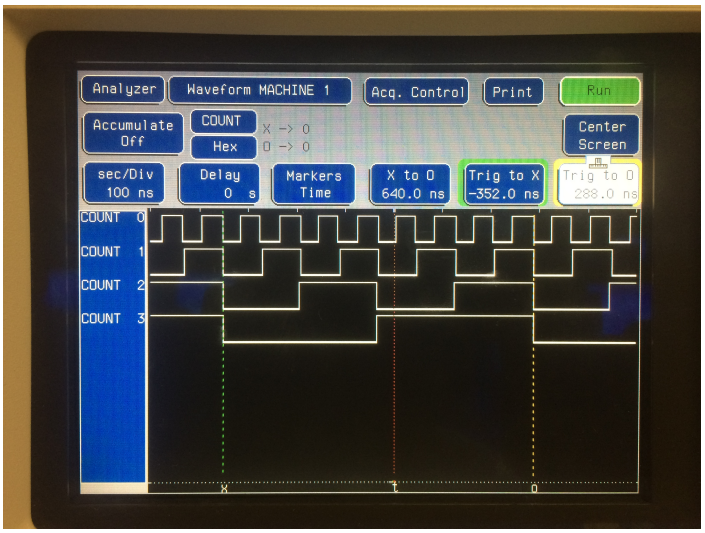

Figure 2: *Clock Divider with Distance Analysis*

This figure, we modified the settings on the Logic Analyzer in order to display the hexadecimal values of the COUNT ALL.

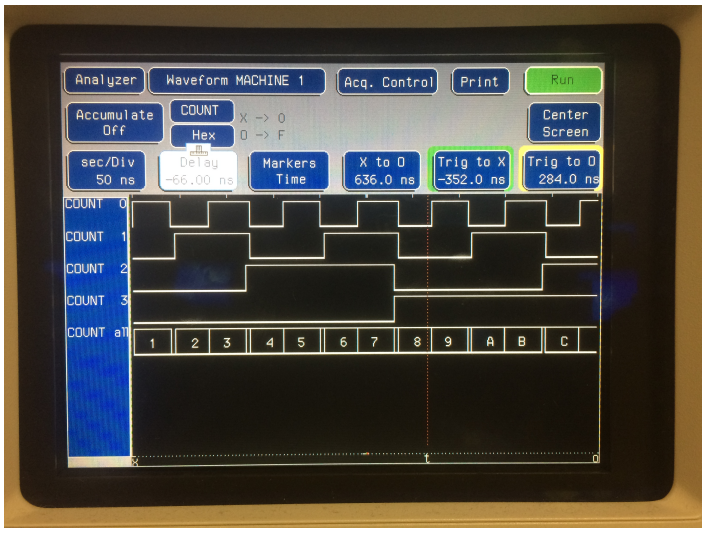

Figure 3: *Clock Divider with Count All Shown with Delay*

The following is the result of the up counter being displayed in hexadecimal form.

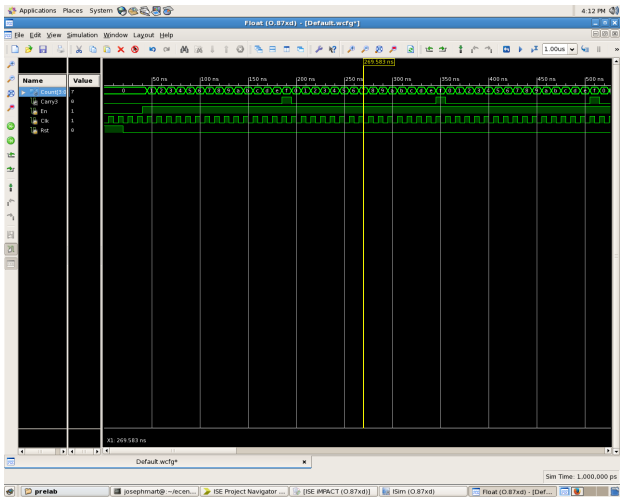

Figure 4: *Up Counter*

The following was the zoomed in version of the previous image in order to determine the period for the maximum value of the up counter (195 ns)

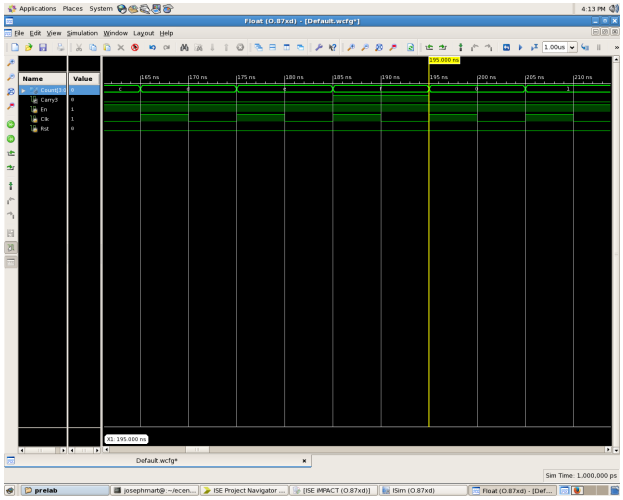

Figure 5: *Roll Over*

In this image, the Count bit width was changed from 5 bits to 26 bits.

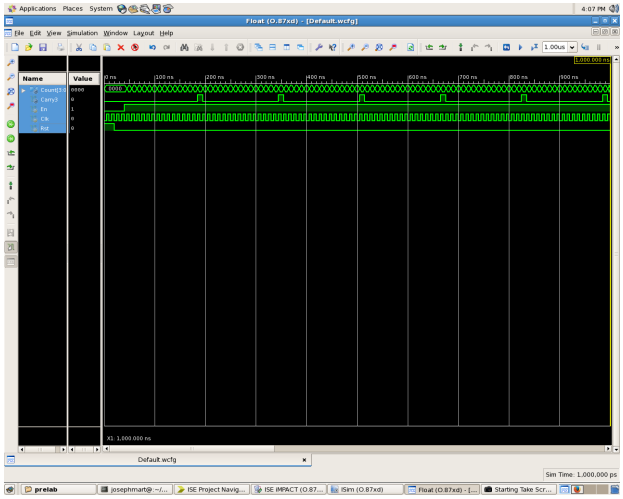

Figure 6: *Up Counter with 26 Bits*

Electrical chatter of the Switch Bounce is visible in this final figure.

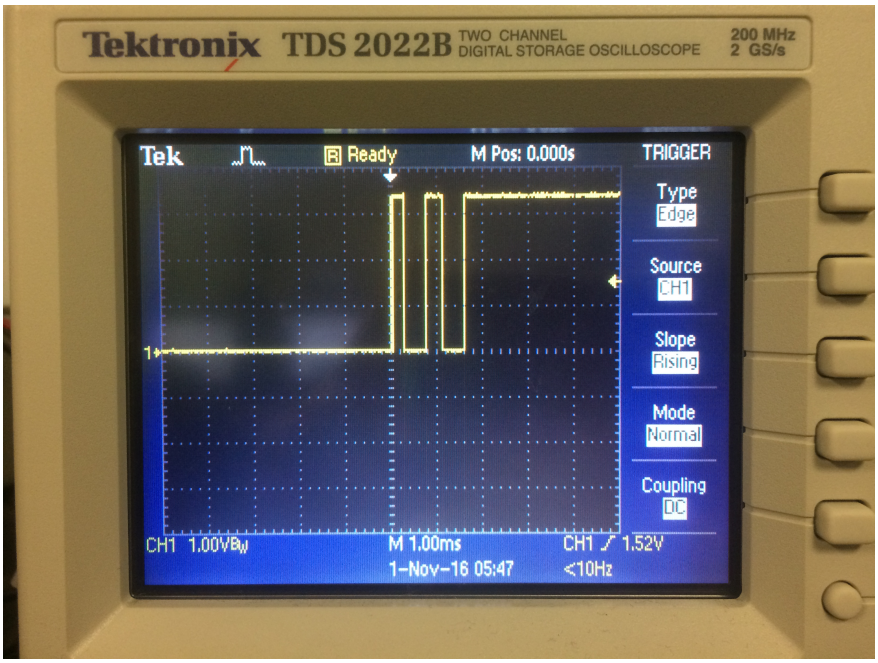

Figure 7: *Switch Bounce*

Conclusion

In conclusion, this lab allowed us to expand on our familiarity of sequential circuits allowing us to design a half adder, up counter, top level, and a clock divider. We also were able to make a UCF file in order to synthesise Verilog onto the FBGA board for the first time. Finally, the last part of the lab had us analyze the effects of electrical chatter and how adding a synthesizer can reduce the interference.

Questions

1. **Include the source code with comments for all modules in lab. You do not have to include test bench code. Code without comments will not be accepted!**

   In the report.

2. **Include any UCFs that you wrote or modified.**

   In the report.

3. **Include screenshots of all waveforms captured during simulation in addition to the test bench console output for each test bench simulation.**

   In the report. **The following are the questions and answers listed in the lab report.**

4. **Measure and record the period of each clock signal using the green and yellow markers. Based on you measurements, what frequency do you think the input clock is running at?**

   $$COUNT0 - 80ns$$

   $$COUNT1 - 160ns$$

   $$COUNT2 - 320ns$$

   $$COUNT3 - 640ns$$

   ClockIn should be $40ns$ resulting in a frequency of 25 MHz.

5. **Open up the test bench file and try to understand what is going on. You should see that the test bench produces a Clk signal. What is the frequency of that signal?**

   In the test bench, the clock signal is held for 5 ns high and then 5 ns low which would result in a 10 ns cycle. The frequency would then be 100 MHz

6. **You should also see that the test bench holds the counter in reset for a specific interval of time. How long is that interval?**

   The interval is 20 ns

7. **After reset is de-asserted, the test bench holds the enable LOW for some amount of time before allowing the counter to run. How long is this time period?**

   This time period is 20 ns.

8. **What is this maximum count value and what signal in the waveform could we use to know exactly when the counter is going to roll over?**

   The maximum count value is f in hexadecimal or 15 in decimal. Whenever Carry3 was HIGH, Count was a max, on the downward edge of Carry3, the Count would reset/roll over

9. **If we use a 50MHz clock to drive our frequency divider, what rate will the most significant bit of the divider oscillate at?**

   $f_2 = \frac{f_1}{2^n} = \frac{50mHz}{2^{26}} = 0.745Hz$

   The switches slowed down the rate at which the LEDs would count up. The fastest counting up would occur when none of the switches were toggled. The slowest would occur when both switches were toggled. This is due to the how the switches were interpreted into binary. There were two switches S_0 and S_1 where S_0 was of least importance. As the binary value of the switches went up, the speed of the LEDs would slow down.

10. **Make a note in you lab write up of what what is described in these (Switch Bounce) files.**

    It connects J10 to the B4 input on the board. It also connects center to V16 (big center button) on the board.

11. **NoDebounce. Does the design work as intended?**

    The LEDs are a little glitchy because of multiple rising edges due to electrical chatter.

12. **Explain the operation of the circuit described in withDebounce.v**

    The input wire goes into two flip flops in order to synchronize the signal and eliminate electrical chatter. Then the signal goes into a binary counter. The LEDs correctly incremented based on the number of times the button was pressed.

Student Feedback

1. **What did you like most about the lab assignment and why? What did you like least about it and why?**

   In this lab, working with a partner proved to be very beneficial. Our ground circuit on the Logic Analyzer was faulty.

2. **Were there any section of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving the clarity?**

   The lab manual was clear in this lab contrary to previous labs.

3. **What suggestions do you have to improve the overall lab assignment?**

   Better equipment.