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

Code Block 1: Clock Divider

```
1  `timescale 1ns / 1ps
2  `default_nettype none
3
4
5  module clock_divider (ClkOut, ClkIn);
6    /* output port needs to be a reg because we will drive it with *
7    a behavioral statement */
8    output wire [3:0] ClkOut;
9    input wire ClkIn; // wires can drive regs
10
11   /*-this is a keyword we ahve not seen yet!*
12   as the name implies, it is a parameter that
13   can be changed at compile time...*/
14   parameter n = 26; // make count 5-bits for now
15
16   reg [n-1:0] Count; //count bit width is based on n! how cool is that!
17
18   //simple behavioral construct to describe a counter
19   always @ ( posedge ClkIn ) begin
```

```
20      Count <= Count + 1;
21    end
22
23    // Now we need to wire up our ClkOut which is a 4-bit wire
24    // Wire up to most signficant bit
25    assign ClkOut[3:0] = Count[n-1:n-4];
26
27  endmodule // clock_divider
```

Code Block 2: Half Adder

```
1  `timescale 1ns / 1ps
2  `default_nettype none
3
4  module half_adder (S,Cout, A,B);
5
6    output wire S, Cout;
7    input wire A, B;
8
9    assign S = A^B;
10   assign Cout = A&B;
11
12 endmodule // half_adder
```

Code Block 3: Up Counter

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:     15:54:59 10/31/2016
7  // Design Name:
```

```
8  // Module Name:     up_counter
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module up_counter (Count, Carry3, En, Clk, Rst);
22         // Count output needs to be a reg
23         output reg [3:0] Count;
24         output wire Carry3;
25         input wire En, Clk, Rst;
26         //intermediate nets
27         wire [3:0] Carry, Sum;
28
29         // Instantiate and wire up half-adders here
30         // module half_adder (output S, output Cout, input A, input B);
31         half_adder ha0(Sum[0], Carry[0], En, Count[0]);
32         half_adder ha1(Sum[1], Carry[1], Carry[0], Count[1]);
33         half_adder ha2(Sum[2], Carry[2], Carry[1], Count[2]);
34         half_adder ha3(Sum[3], Carry[3], Carry[2], Count[3]);
35
36         // Wire up carry 3
37         assign Carry3 = Carry[3];
38         //Describe positve edge triggered flip-flops for count
```

```
39          // Including ``posedge Rst" in the sensitivty list
40          // Implies an asynchronous reset!
41          always @ ( posedge Clk or posedge Rst ) begin
42                  if (Rst) begin //if RST == 1'b1
43                          Count <= 0;
44                  end else begin //otherwise latch sum
45                          Count <= Sum;
46                  end
47          end
48  endmodule
```

Code Block 4: Top Level (Verilog)

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:     16:15:32 10/31/2016
7   // Design Name:
8   // Module Name:     top_level
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
```

```
20 ////////////////////////////////////////////////////////////////////////////////
21 module top_level (LEDs, SWs, North, South, FastClk);
22
23   // All ports will be wires
24         output wire [4:0] LEDs;
25         input wire FastClk, North, South;
26         input wire [1:0] SWs;
27
28   wire [3:0] Clocks;
29   reg SlowClk; // will use an always blcok for mux
30
31   // behavioral description of a mux which selects
32   // between the four available clock signals
33   always @ ( * ) begin
34     case (SWs) // SWs is a 2-bit bus
35       2'b00: SlowClk = Clocks[0]; // use blcoking statemnt for
36                                   // combinational logic
37                 2'b01: SlowClk = Clocks[1];
38                 2'b10: SlowClk = Clocks[2];
39                 2'b11: SlowClk = Clocks[3];
40       // finish this up
41
42       // default: ;
43     endcase
44   end
45
46   // instantiate your up_counter here
47   // Hint: if you want to wire a port to just the first 4
48   // bits of a bus, you can do something like this: LEDs[3:0]
49         up_counter upCount (LEDs[3:0], LEDs[4], North, SlowClk, South);
50
```

```
51    // instantiate the clock divider
52    clock_divider clk_div0(
53        .ClkOut(Clocks),
54        .ClkIn(FastClk)
55      );
56
57  endmodule // top_level
```

Code Block 5: Top Level UCF

```
1  #Switches
2  NET "SWs[0]" LOC = "L13" | IOSTANDARD = LVTTL; #SW0
3  # fill in for SWs[1]
4  NET "SWs[1]" LOC = "L14" | IOSTANDARD = LVTTL; #SW1
5
6  #Push-buttons
7  NET "NORTH" LOC = "V4" | IOSTANDARD = LVttL | PULLDOWN; #North
8
9  #fill in for South
10 NET "SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN; #South
11
12 # LEDs
13 NET "LEDs[0]" LOC = "F12" | IOSTANDARD = LVTTL; #LD0
14 #somethingiss missing here
15 NET "LEDs[1]" LOC = "E12" | IOSTANDARD = LVTTL; #LD1
16 NET "LEDs[2]" LOC = "E11" | IOSTANDARD = LVTTL; #LD2
17 NET "LEDs[3]" LOC = "F11" | IOSTANDARD = LVTTL; #LD3
18 NET "LEDs[4]" LOC = "C11" | IOSTANDARD = LVTTL; #LD0
19
20 NET "FastClk" LOC = "C9" | IOSTANDARD = LVTTL;
21
22 # Define clock period for 50 MHz oscillator
```

```
23 NET "FastClk" PERIOD = 20.0ns HIGH 40%;
```

Code Block 6: Switch Bounce

```
1  `timescale 1ns / 1ps
2
3  module switch_bounce(
4      input wire Center,
5      output wire J1_0
6      );
7
8      // Assign the output from FBGA board to an input Center wire
9      assign J1_0 = Center;
10
11 endmodule
```

Code Block 7: No Debounce

```
1  `timescale 1ns / 1ps
2  `default_nettype none
3
4  /*This module describes a counter that is triggered off of*
5   *the rising-edge of an signal that has not been debounced*
6   *in order to demonstrate the effects of switch bounce    */
7  module noDebounce(LEDs, Center, Clk);
8
9      /*The output LEDs are of type reg because they are*
10      *modified using behavioral Verilog                */
11     output reg [7:0] LEDs;
12     input wire Center, Clk;
13
14     /*intermediate nets*/
15     reg edge_detect0, edge_detect1;
```

```
16      wire rising_edge;//asserted when an edge is detected
17
18      /*describe an edge-detector circuit which detects*
19       *a rising-edge of an asynchronous signal. The   *
20       *usage of two flip-flops may seem redundant but *
21       *is necessary for synchronization purposes!     */
22      always@(posedge Clk)
23        begin
24          edge_detect0 <= Center;//input signal
25          edge_detect1 <= edge_detect0;
26        end
27      /*when older value is 0 and new value is 1, a   *
28       *rising edge has occurred                       */
29      assign rising_edge = ~edge_detect1 & edge_detect0;
30
31      /*describe a counter that increments each time a*
32       *rising-edge of input signal is detected       */
33      always@(posedge Clk)
34          if(rising_edge)
35              LEDs <= LEDs + 1;
36
37  endmodule
```

Code Block 8: 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

Measure and record the period of each clock signal using the green and yellow markers. Based on your measurements, what frequency do you think the input (Experiment 1.4.b) 2. 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? (Exp 2.2.b) 3. You should also see that the test bench holds the counter in reset for a specific interval of time. How long is that interval? (Exp 2.2.c) 4. 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? (Exp 2.2.d) 5. 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? (Exp 2.2.f) 6. If we use a 50MHz clock to drive our frequency divider, what rate will the most significant bit of the divider oscillate at? (Exp 2.3.a) 7. Copy the waveform on the scope into your lab write-up. (Exp 3.1.j) 8. Does the design work as intended? Why or why not? (Exp 3.2.f)

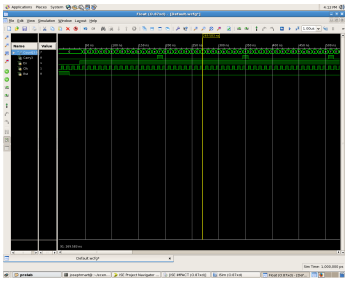

Figure 1: *2-Bit 2:1 MUX Plots*

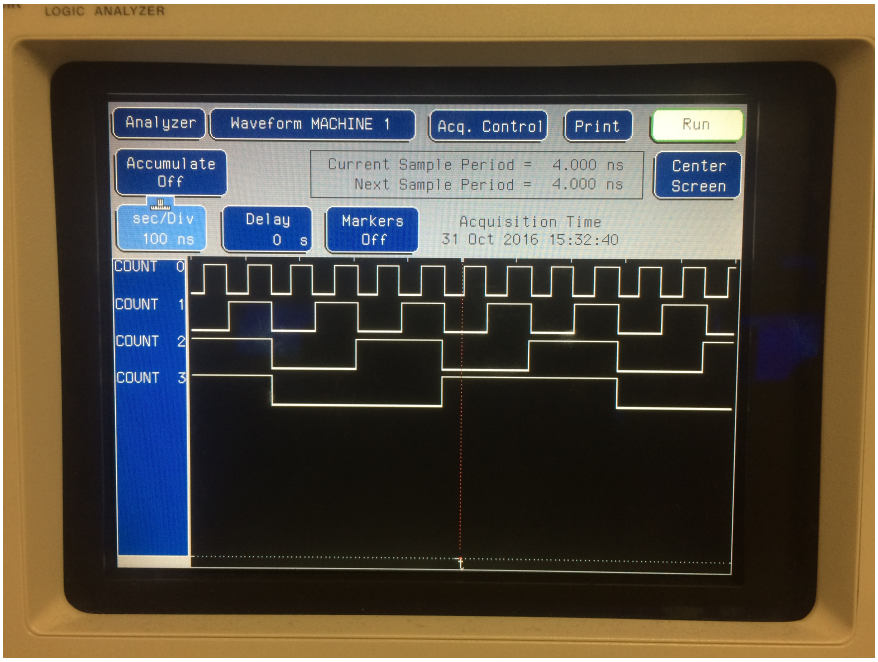

Figure 2: *2-Bit 2:1 MUX Plots*

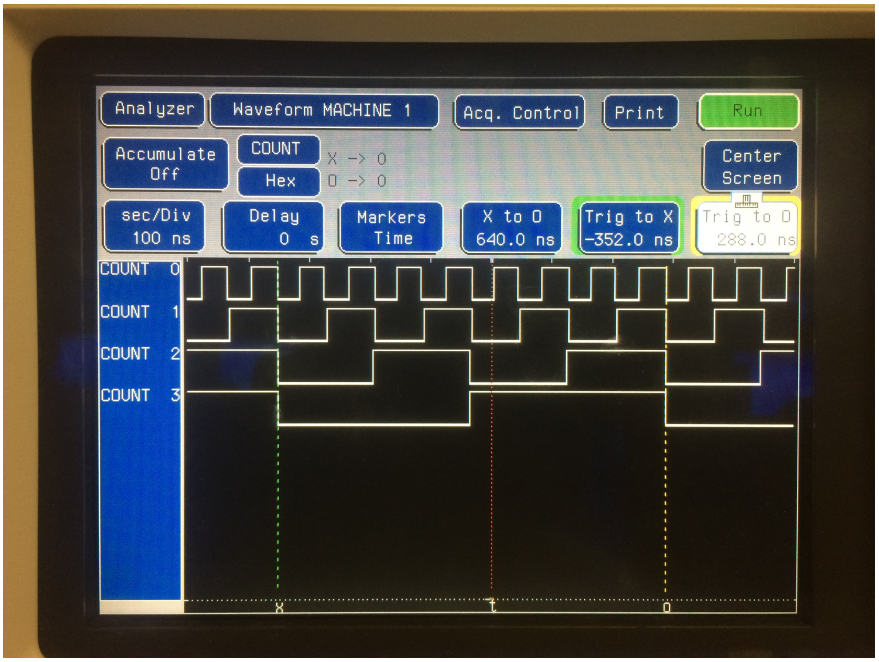

Figure 3: *2-Bit 2:1 MUX Plots*

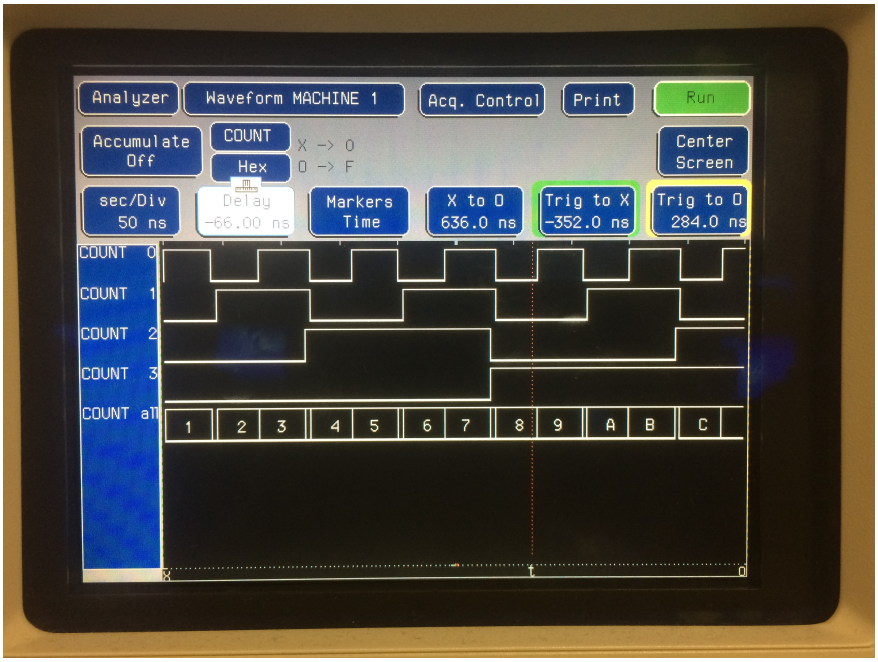

Figure 4: *2-Bit 2:1 MUX Plots*

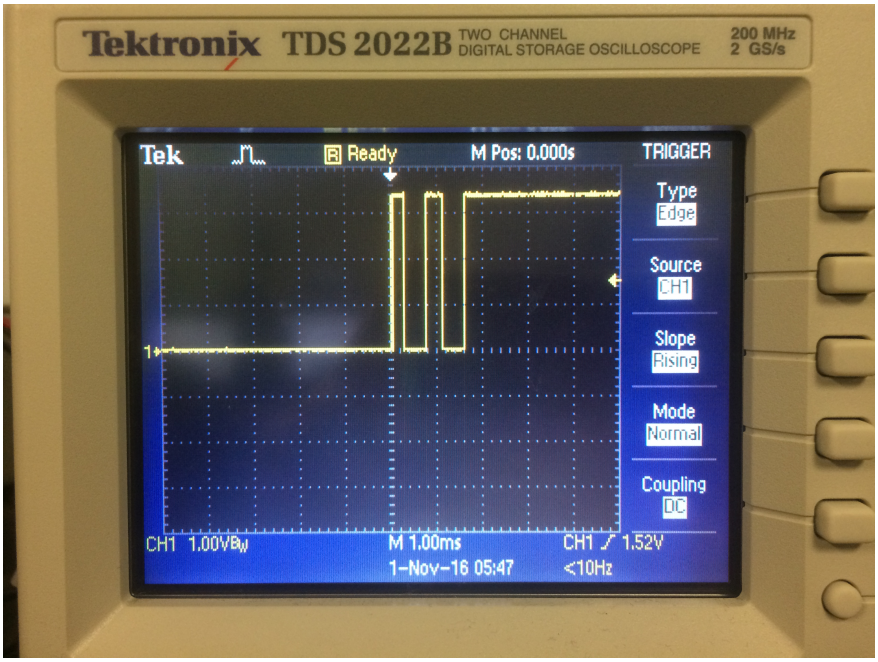

Figure 5: *2-Bit 2:1 MUX Plots*

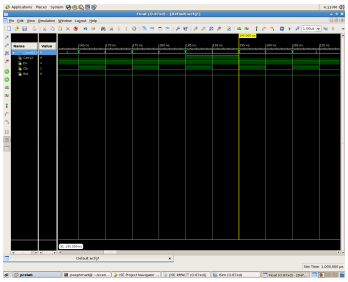

Figure 6: *2-Bit 2:1 MUX Plots*

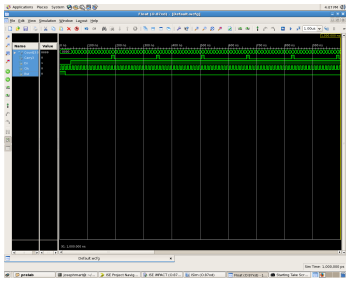

Figure 7: *2-Bit 2:1 MUX Plots*

Conclusions

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.

4. **Answer all questions throughout the lab manual.**

5. **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 - 80 ns

   COUNT1 - 160 ns

   COUNT2 - 320 ns

COUNT3 - 640 ns

CLock in should be 40 resulting in a frequency of 25 MHz

6. **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

7. **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

8. **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.

9. **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

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

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

11. **Make note in you lab write-up how the switches affected the LED outputs.**

    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.

12. **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.

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

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

14. **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.