Project 1

Synchronous Design

Counter with Display

Umar Khan

014477331

CECS 360

09/21/2017

Introduction:

This project shows the synchronous design with the counter on the display. The switches are used for incrementing and decrementing the counter. The incrementing and decrementing depends on the switch on being up or down, however they are set in the counter. The up-switch increments and down-switch decrements. The project contains multiple modules in it i.e. AISO, Debounce, Pulse Maker, Counter, and Display Controller where all of them were instantiated in the top level.

Major Block Description:

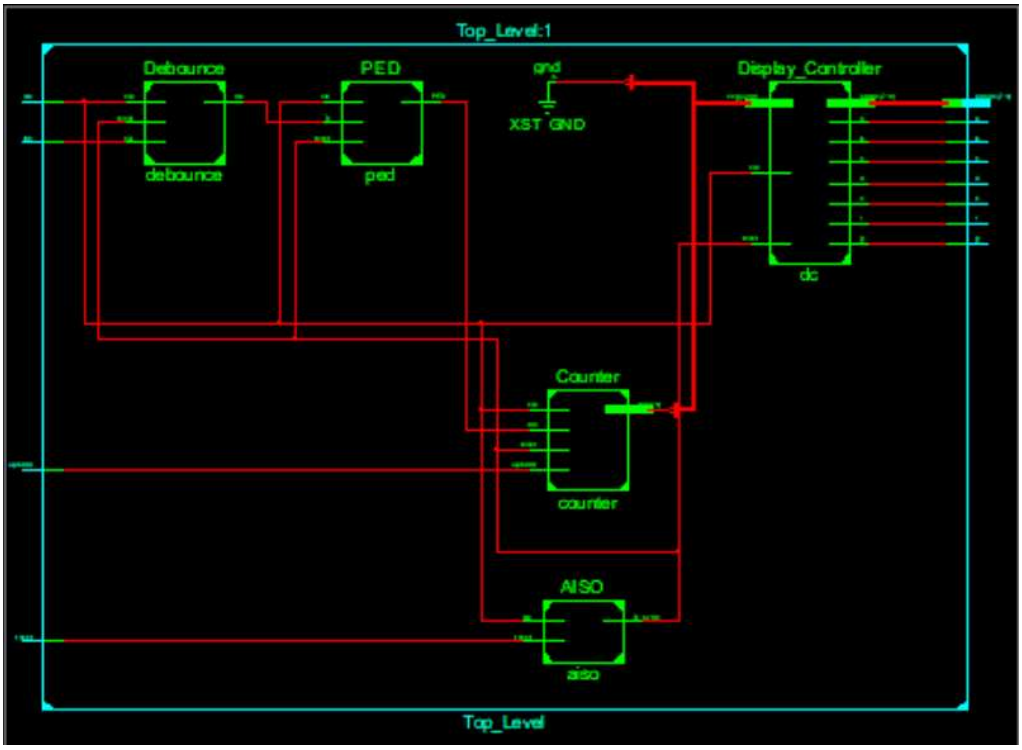
- Debounce: This block of the project stabilizes the output instead of showing too many transitions. Pulse maker is instantiated within the debounce.
    - Pulse Maker: This module generates a tick every 10ms.
- AISO: This module helps with the violation of timing constraints. This module defines the Asynchronous In and Synchronous out idea.
- PED: This module detects the positive edge of the clock. This is done since the output only changes on the positive edge of the clock.
- Counter: This is the module where incrementing and decrementing is done based on the condition of the switch. The switch is set as “Up High Down Low”.
- Display Controller: This is the module where the 7-segment module is instantiated and controlled using led clock, led controller, ad mux, and hex to 7 segment.
    - Led Clock: This is the module where frequency of clk_out is determined by integer count, count = (Incoming Freq/(Outgoing Freq))/2. Clk_out will alternate once every time clk_in alternates 'count' number of times. To achieve a refresh rate of 60Hz with 8 pixels, the output frequency must be 480hz. [Outgoing Freq = (Refresh Rate * Number of Pixels)] To produce an output clock of 480hz, we divide the clock every 104167 cycles. We assume that clk_in from the Nexys 4 FPGA is 100mhz.
    - Led Controller: This is the module that clock as an input. At every posedge clock, Moore state machines will increment the states regardless of the clock value (which always remains 1). Every state will output corresponding to the bit value of the a.

    - Ad Mux: This is the module where the 8 to 1 bit mux with 3 bit select is created and the output is determined by the 3 bit select.
    - Hex to 7 Segment: This module uses input hex and decodes the value to its equivalent representation for the NEXYS 4 seven segment display.
- TopLevel_UCF: This is the module where all the blocks are linked to the NEXYS 4 board.

Hierarchy:

- Top_Level
    - Debounce
        - Pulse_Maker
    - AISO
    - PED
    - Counter
    - Display Controller
        - Led_Clk
        - Led_Controller
        - Ad_Mux
        - Hex_to_7seg
    - Toplevel_ucf

RTL Schematic:



Conclusion:

In conclusion, using AISO module allowed us not to violate timing constraints and successfully make the Counter which decrements when the switch is set to low and vice versa. Also, Debounce stabilizes the output of the clock. The debounce was built referred to Pong Chu’s style. The counter is displayed on the board using the display controller. The counter can display 8-bit values or 16 bit values on the board.

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                                                                              //
4   //  Class: CECS 360                                                             //
5   //  Project name: Project1_CECS360                                              //
6   //  File name: Top_Level.v                                                      //
7   //                                                                              //
8   //  Created by Umar Khan 09/19/2017                                             //
9   //                                                                              //
10  //  Abstract: Instantiation of all the modules.                                 //
11  //                                                                              //
12  //////////////////////////////////////////////////////////////////////////////////
13  module Top_Level (clk, reset, inc, uphdnl, anode, a, b, c, d, e, f, g);
14
15      input  clk, reset, inc, uphdnl;
16
17      output [7:0] anode;
18      output a, b, c, d, e, f, g;
19
20      wire   [31:0] q;
21      wire   PED;
22      wire   reset_out;
23      wire   deb;
24
25
26      Debounce
27        debounce(clk, reset_out, inc, deb);
28
29      AISO
30        aiso(clk, reset, reset_out);
31
32      PED
33        ped(clk, reset_out, deb, PED);
34
35      Counter
36        counter(clk, reset_out, PED, uphdnl, q);
37
38      Display_Controller
39        dc(clk, reset_out, anode, q, a, b ,c ,d ,e, f,g);
40
41  endmodule
42
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////
3   //  Class: CECS 360                                             //
4   //  Project name: Project1_CECS360                              //
5   //  File name: Debounce.v                                       //
6   //                                                              //
7   //  Created by Umar Khan on 09/16/2017                          //
8   //                                                              //
9   //                                                              //
10  //  Abstract: Stabilizes the output instead of showing too many //
11  //  transitions.                                                //
12  //  Reference to Pong Chu's Debounce                            //
13  //                                                              //
14  //////////////////////////////////////////////////////////////////
15  module Debounce(clk, reset, sw, db);
16
17  input wire clk, reset;
18  input wire sw;
19  output reg db;
20
21  // symbolic state declaration
22  pulse_maker pulse_maker(clk,reset, pulse);
23
24  //Pulse_Maker     P1 (clk, reset, pulse);
25     localparam [2:0]
26              zero    = 3'b000,
27              wait1_1 = 3'b001,
28              wait1_2 = 3'b010,
29              wait1_3 = 3'b011,
30              one     = 3'b100,
31              wait0_1 = 3'b101,
32              wait0_2 = 3'b110,
33              wait0_3 = 3'b111;
34
35  // number of counter bits (2"N * 2Ons = lOms tick)
36
37   localparam N = 20;
38
39  // signal declaration
40
41        reg [N-1:0] q_reg;
42        wire [N-1 : 0] q_next ;
43        wire pulse;
44        reg [2:0] state_reg , state_next ;
45
46  // body
47  // counter to generate 10 ms tick
48
49     always @ (posedge clk, posedge reset)
50        if (reset)
51           q_reg <= 0;
52        else
53           q_reg <= q_next;
54
55  // next-state logic
56
57     assign q_next = q_reg + 1;
```

```
58
59    // output tick
60
61       assign pulse = (q_reg==0) ? 1'b1 : 1'b0;
62
63    // debouncing FSM
64    // state register
65
66       always @ ( posedge clk , posedge reset)
67          if (reset)
68             state_reg <= zero;
69          else
70             state_reg <= state_next ;
71
72       // next-state logic and output logic
73
74       always @*
75          begin
76             state_next = state_reg; // default state: the same
77             db = 1'b0; // default output: 0
78       case (state_reg)
79
80    zero :
81          if (sw)
82             state_next = wait1_1 ;
83             wait1_1 :
84          if (~sw)
85             state_next = zero;
86          else
87
88          if (pulse)
89             state_next = wait1_2 ;
90    wait1_2 :
91          if (~sw)
92             state_next = zero;
93          else
94          if (pulse)
95             state_next = wait1_3;
96    wait1_3 :
97          if (~sw)
98             state_next = zero;
99          else
100         if (pulse)
101            state_next = one;
102   one :
103         begin
104            db = 1'b1;
105         if (~sw)
106            state_next = wait0_1;
107         end
108   wait0_1 :
109         begin
110            db = 1'b1;
111         if (sw)
112            state_next = one;
113         else
114         if (pulse)
```

```
115             state_next = wait0_2;
116          end
117    wait0_2 :
118          begin
119             db = 1'b1;
120          if (sw)
121             state_next = one;
122          else
123          if (pulse)
124             state_next = wait0_3;
125          end
126    wait0_3 :
127          begin
128             db = 1'b1;
129          if (sw)
130             state_next = one;
131          else
132          if (pulse)
133             state_next = zero;
134          end
135
136          default : state_next = zero;
137
138          endcase
139
140       end
141
142    endmodule
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //  Class: CECS 360                                                             //
4   //  Project name: Project1_CECS360                                              //
5   //  File name: pulse_maker.v                                                      //
6   //                                                                              //
7   //  Created by Umar Khan on 09/16/2017                                          //
8   //                                                                              //
9   //                                                                              //
10  //  Abstract: Generates a tick every 10ms.                                      //
11  //                                                                              //
12  //////////////////////////////////////////////////////////////////////////////////
13
14  module pulse_maker(clk, reset, pulse);
15
16     input clk, reset;
17     output pulse;
18     reg [19:0] count;
19
20     assign pulse = (count == 999999);
21
22     always @ (posedge clk, posedge reset)
23
24        if(reset)   count <= 20'b0;
25     else
26        if(pulse) count <= 20'b0;
27     else
28                   count<= count + 20'b1;
29
30  endmodule
31
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                                                                              //
4   //  Class: CECS 360                                                             //
5   //  Project name: Project1_CECS360                                              //
6   //  File name: AISO.v                                                           //
7   //                                                                              //
8   //  Created by Umar Khan on 09/16/2017                                          //
9   //                                                                              //
10  //  Abstract: Helps with the violation of timing constraints.                   //
11  //                                                                              //
12  //////////////////////////////////////////////////////////////////////////////////
13
14  module AISO(clk,reset, R_Sync);
15
16     input clk, reset;
17     output R_Sync;
18     wire R_sync;
19     reg F1, F2;
20
21     always @ (posedge clk, posedge reset) begin
22
23        if (reset)
24           {F1,F2} <= 2'B00;
25        else
26           {F1,F2} <= {1'b1,F1};
27        end
28
29        assign R_Sync = !F2;
30
31  endmodule
32
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                                                                              //
4   //  Class: CECS 360                                                             //
5   //  Project name: Project1_CECS360                                              //
6   //  File name: Counter.v                                                        //
7   //                                                                              //
8   //  Created by Umar Khan 09/19/2017       .                                     //
9   //                                                                              //
10  //  Abstract: Increments or decrements based on the switch UPHDWL.              //
11  //                                                                              //
12  //////////////////////////////////////////////////////////////////////////////////
13  module Counter(clk, reset, inc, uphdnl, q);
14
15     input clk, reset;
16     input uphdnl, inc;
17
18     output [15:0] q;
19     reg [15:0] q;
20
21     always @(posedge clk, posedge reset)
22
23        if (reset) q <= 8'b0;
24        else
25        if (inc) q <= (uphdnl)? q + 16'b1 : q - 16'b1;
26
27  endmodule
28
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                                                                              //
4   //  Class: CECS 360                                                             //
5   //  Project name: Project1_CECS360                                              //
6   //  File name: PED.v                                                            //
7   //                                                                              //
8   //  Created by Umar Khan 09/19/2017                                             //
9   //                                                                              //
10  //  Abstract: Detects the positive edge of the clock  .                         //
11  //            It detects the PED as the output will only                        //
12  //            change on the PED                                                 //
13  //                                                                              //
14  //////////////////////////////////////////////////////////////////////////////////
15  module PED(clk, reset, D, PED);
16
17     input clk, reset, D;
18
19     output PED;
20     wire PED;
21
22     reg F1;
23
24     always @ (posedge clk, posedge reset) begin
25        if (reset)
26           F1 <= 1'b00;
27        else
28           F1 <= D;
29        end
30
31     assign PED = {~F1 & D};
32
33  endmodule
34
35
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                                                                              //
4   //  Class: CECS 360                                                             //
5   //  Project name: Project1_CECS360                                              //
6   //  File name: Display_Controller.v                                             //
7   //                                                                              //
8   //  Created by Umar Khan 09/19/2017       .                                     //
9   //                                                                              //
10  //  Abstract: This is the 7 segement display module consisting of               //
11  //            led_clk, led_controller, ad_mux, and hex_to_7seg                  //
12  //                                                                              //
13  //////////////////////////////////////////////////////////////////////////////////
14  module Display_Controller  (clk, reset, anode, seg, a, b, c, d, e, f, g);
15     input              clk, reset;
16     input      [31:0] seg;
17
18     output      [7:0] anode;
19     output            a, b, c, d, e, f, g;
20
21     wire              led_cout;
22     wire        [2:0] sel;
23     wire        [3:0] y;
24
25     led_clk           a1(clk, reset, led_cout);
26
27
28     led_controller    a2(led_cout, reset, anode, sel);
29
30
31     ad_mux            a3(sel, seg[31:28], seg[27:24], seg[23:20],
32                       seg[19:16], seg[15:12], seg[11:8], seg[7:4],
33                       seg[3:0], y);
34
35     hex_to_7seg       a4(y, a, b, c, d, e, f, g);
36
37  endmodule
38
39
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                                                                              //
4   //  Class: CECS 360                                                             //
5   //  Project name: Project1_CECS360                                              //
6   //  File name: hex_to_7seg.v                                                    //
7   //                                                                              //
8   //  Created by Umar Khan 09/19/2017       .                                     //
9   //                                                                              //
10  //  Abstract: Module hex_to_7segment uses input hex and decodes                 //
11  //            the value to its equivalent representation for the                //
12  //            NEXYS 4 seven segment display. This is outputed to                //
13  //            the wires a, b, c, d, e, f, g accordingly.                        //
14  //                                                                              //
15  //////////////////////////////////////////////////////////////////////////////////
16
17  module hex_to_7seg(hex, a, b, c, d, e, f, g);
18
19     input    [3:0] hex;
20     output         a, b, c, d, e, f, g;
21     reg            a, b, c, d, e, f, g;
22
23     always @ (hex) begin
24        case (hex)
25           4'b0000: {a, b, c, d, e, f, g} = 7'b0000001;
26           4'b0001: {a, b, c, d, e, f, g} = 7'b1001111;
27           4'b0010: {a, b, c, d, e, f, g} = 7'b0010010;
28           4'b0011: {a, b, c, d, e, f, g} = 7'b0000110;
29           4'b0100: {a, b, c, d, e, f, g} = 7'b1001100;
30           4'b0101: {a, b, c, d, e, f, g} = 7'b0100100;
31           4'b0110: {a, b, c, d, e, f, g} = 7'b0100000;
32           4'b0111: {a, b, c, d, e, f, g} = 7'b0001111;
33           4'b1000: {a, b, c, d, e, f, g} = 7'b0000000;
34           4'b1001: {a, b, c, d, e, f, g} = 7'b0000100;
35           4'b1010: {a, b, c, d, e, f, g} = 7'b0001000;
36           4'b1011: {a, b, c, d, e, f, g} = 7'b1100000;
37           4'b1100: {a, b, c, d, e, f, g} = 7'b0110001;
38           4'b1101: {a, b, c, d, e, f, g} = 7'b1000010;
39           4'b1110: {a, b, c, d, e, f, g} = 7'b0110000;
40           4'b1111: {a, b, c, d, e, f, g} = 7'b0111000;
41           default: {a, b, c, d, e, f, g} = 7'b1111111;
42        endcase
43     end
44
45  endmodule
46
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                                                                              //
4   //  Class: CECS 360                                                             //
5   //  Project name: Project1_CECS360                                              //
6   //  File name: led_clk.v                                                  //
7   //                                                                              //
8   //  Created by Umar Khan 09/19/2017       .                                     //
9   //  Abstract:                                                                   //
10  //          Frequency of clk_out is determined by integer count,                //
11  //          count = (Incoming Freq/(Outgoing Freq))/2.                          //
12  //          clk_out will alternate once every time clk_in                       //
13  //          alternates 'count' number of times. To achieve a refresh            //
14  //          rate of 60Hz with 8 pixels, the output frequency must be            //
15  //          480hz.[Outgoing Freq = (Refresh Rate * Number of Pixels)]           //
16  //          To produce an output clock of 480hz, we divide the clock            //
17  //          every 104167 cycles. We assume that clk_in from the Nexys 4         //
18  //          FPGA is 100mhz.                                                     //
19  //                                                                              //
20  //////////////////////////////////////////////////////////////////////////////////
21  ////////////////////////////////////////////////////////////////////////////////////
22  module led_clk(clk, reset, led_clk);
23     input  clk, reset;
24     output led_clk;
25     reg    led_clk;
26     integer clk_ticks;
27
28     always @(posedge clk, posedge reset) begin
29        if(reset == 1'b1) begin
30           clk_ticks = 0;
31           led_clk   = 0;
32        end
33        //got a clock, so increment the counter and
34        //test to see if half a period has elapsed
35        else begin
36           clk_ticks = clk_ticks + 1;
37           //104166 is used to create 480Hz
38           if (clk_ticks >= 104166) begin
39              led_clk = ~led_clk;
40              clk_ticks = 0;
41           end
42        end
43     end
44
45
46  endmodule
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                                                                              //
4   //  Class: CECS 360                                                             //
5   //  Project name: Project1_CECS360                                              //
6   //  File name: led_controller.v                                                   //
7   //                                                                              //
8   //  Created by Umar Khan 09/19/2017       .                                     //
9   //  Abstract:                                                                   //
10  //        This shift module uses clk as an input. At every posedge clk, the     //
11  //        Moore state machine will increment states regardless of the clk       //
12  //        value (which will always be 1). Every state will output the           //
13  //        corresponding analog pins {a7,a6,a5,a4,a3,a2,a1,a0} as well as         //
14  //        the present state {seg_sel}. For example the present state 000        //
15  //        will turn on the rightmost segment display, as well as selecting      //
16  //        the correct value to display. This moore state machine will           //
17  //        activate one pixel at a time from the rightmost display to the        //
18  //        leftmost display and then return to the rightmost display. This       //
19  //        will cycle indefinately.                                              //
20  //                                                                              //
21  //////////////////////////////////////////////////////////////////////////////////
22
23
24
25  module led_controller(clk, reset, a, seq_sel);
26     input                clk, reset;
27     output  reg    [7:0] a;
28     output  reg    [2:0] seq_sel;
29
30
31  ////////////////////////////////////////////////////////////////////////////////
32  //                               state register and
33  //                              next_state variables
34  ////////////////////////////////////////////////////////////////////////////////
35     reg            [2:0] Q;     //present state
36     reg            [2:0] D;     //next state
37
38  ////////////////////////////////////////////////////////////////////////////////
39  //                          Next State Combinational Logic
40  //   (next state values can change anytime but will only be "clocked" below)
41  ////////////////////////////////////////////////////////////////////////////////
42     always @(Q) begin
43        case(Q)
44           3'b000:     D = 3'b001;
45           3'b001:     D = 3'b010;
46           3'b010:     D = 3'b011;
47           3'b011:     D = 3'b100;
48           3'b100:     D = 3'b101;
49           3'b101:     D = 3'b110;
50           3'b110:     D = 3'b111;
51           3'b111:     D = 3'b000;
52           default     D = 3'b000;
53        endcase
54     end
55
56  ////////////////////////////////////////////////////////////////////////////////
57  //                     State Register Logic (Sequential Logic)
```

```
58  ////////////////////////////////////////////////////////////////////////////////
59     always @ (posedge clk, posedge reset) begin
60        if(reset == 1'b1)
61           Q <= 3'b000;
62        else
63           Q <= D;
64     end
65
66  ////////////////////////////////////////////////////////////////////////////////
67  //                            Output Combinational Logic
68  //             (outputs will only change when present state changes)
69  ////////////////////////////////////////////////////////////////////////////////
70     always @(Q) begin
71        case(Q)
72           3'b000:     {a, seq_sel} = 11'b01111111_000;
73           3'b001:     {a, seq_sel} = 11'b10111111_001;
74           3'b010:     {a, seq_sel} = 11'b11011111_010;
75           3'b011:     {a, seq_sel} = 11'b11101111_011;
76           3'b100:     {a, seq_sel} = 11'b11110111_100;
77           3'b101:     {a, seq_sel} = 11'b11111011_101;
78           3'b110:     {a, seq_sel} = 11'b11111101_110;
79           3'b111:     {a, seq_sel} = 11'b11111110_111;
80           default:    {a, seq_sel} = 11'b11111111_000;
81        endcase
82     end
83
84
85  endmodule
86
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////
3   //                                                              //
4   //  Class: CECS 360                                             //
5   //  Project name: Project1_CECS360                              //
6   //  File name: hex_to_7seg.v                                    //
7   //                                                              //
8   //  Created by Umar Khan 09/19/2017       .                     //
9   //                                                              //
10  //  Abstract: 8-to-1 4bit mux with 3 bit select(Sel). Output (y)
11  //            is always set to only one specific 4 bit input    //
12  //            (d7,d6,d5,d4,d3,d2,d1,d0)                         //
13  //            that is determined by the 3 bit select(seq_sel).  //
14  //////////////////////////////////////////////////////////////////
15  ////////////////////////////////////////////////////////////////////
16  module ad_mux(seq_sel, d0, d1, d2, d3, d4, d5, d6, d7, y);
17
18     input    [2:0]    seq_sel;
19     input    [15:12]  d0;
20     input    [11:8]   d1;
21     input    [7:4]    d2;
22     input    [3:0]    d3;
23     input    [15:12]  d4;
24     input    [11:8]   d5;
25     input    [7:4]    d6;
26     input    [3:0]    d7;
27
28     output   [3:0]    y;     //output to the hex_to_7seg
29
30     reg      [3:0]    y;
31
32     always @(*) begin
33        case(seq_sel)
34           3'b000:     y = d0;
35           3'b001:     y = d1;
36           3'b010:     y = d2;
37           3'b011:     y = d3;
38           3'b100:     y = d4;
39           3'b101:     y = d5;
40           3'b110:     y = d6;
41           3'b111:     y = d7;
42        endcase
43     end
44
45  endmodule
46
```