Lab 6
Simon Carballo
1618309
11/29/2020
Section D

Lab6 Write Up

- **Description:** In this lab we were tasked to design a sequential circuit that detects when a turkey goes left or right between two IR sensors and adds/subtracts a count depending on the direction the turkey goes. In this case we are using buttons as a replacement to the IR sensors. This time the challenge of the lab is to use our past knowledge to design this system.

- **Design:** For this design we have the Top Level that calls for multiple subsections which includes: Clock(Provided), StateMachine, 8-bit Counter, Two's Complement Converter, 2to1x8MUX, Ring Counter, Selector, and the Segment_Display Converter. As we go over each subsection we will look more into depth of how each section is designed.
    - Clock:
      This code is provided by the lab manual. This module intakes the basys3 clkin value and outputs the global clock and reset for all of the counters in the Top_Level. It takes in the inputs of clkin from the constraint file, and manipulates it to become the net clk for the sequential design. We also use this to get Dig_sel for the ring counter (Explained in Ring Counter Module). In this lab we also use the qsec, which outputs the clk signal which is high for one clk cycle every ¼ of a second making it possible to be used as a different form of clk.

    - State Machine;
      The entire lab is operated using a state machine that functions with D Flip-Flops and control logic. In this lab I built to control logic using 7 different states: IDLE, L, L_B, LR, R, R_B, RL. The states were controlled with two inputs: Left, and Right. These states were used to determine the location of the turkey and whether or not to count up, down, or not at all. My control logic was built using the following boolean algebra using one-hot encoding:

        - IDLE:

PS	L(Left)	R(Right)
Q0	0	0
Q1	0	0
Q3	0	0

Q4	0	0
Q6	0	0

D0 = ~L*~R*(Q0+Q3+Q4+Q6)

- L:

PS	L(Left)	R(Right)
Q0	1	0
Q1	1	0
Q2	1	0

D1 = L*~R*(Q0+Q1+Q2)

- L_B:

PS	L(Left)	R(Right)
Q1	1	1
Q2	1	1
Q3	1	1

D2 = L*R*(Q1+Q2+Q3)

- LR:

PS	L(Left)	R(Right)
Q2	0	1
Q3	0	1

D3 = ~L*R*(Q2+Q3)

- R

PS	L(Left)	R(Right)
Q0	0	1
Q4	0	1
Q5	0	1

D4 = ~L*R*(Q0+Q4+Q5)

- R_B:

PS	L(Left)	R(Right)
Q4	1	1

Q5	1	1
Q6	1	1

D5 = L*R*(Q4+Q5+Q6)

- RL:

PS	L(Left)	R(Right)
Q5	1	0
Q6	1	0

D6 = L*~R*(Q5+Q6)

Outputs:

- Reset = IDLE
- CountUp = LR*~Right*~Left
- CountDw = RL*~Left*~Right
- L_LED = ~Left
- R_LED = ~Right
- On = Left+Right

- In my design I mixed Mealy and Moore, because I added more outputs after designing my model.
- With all of these boolean expressions we can combine them into one module to create a State Machine that acts as the brain of the game.

- 8-bit Counter(Converted from 16-bit counter from previous labs):
  In our previous lab write-up we examined how the U/D 16-bit counter was made and how it works. Since we only require 8 bits maximum in this lab I added a reset to the counter and set it to reset when the most significant bit is high(This function is only used for the timer counter). Finally for the Up/Dw Counter I added an output that detects when the values have gone negative or not.

- Two Complement Converter:
  Since we will be using negative numbers in this design we need to account for two's complement to prevent the counter from going to FF when one is subtracted from 00. This can be done by taking the NOT values of the counter and adding one to the value. I created this module by just creating an 8-bit adder(From lab2) and adding one to it.

- 2to1x8 MUX:

In order to determine whether I should display positive values or negative values I used a two to one 8bit mux that will select the right values depending on whether the values are negative or not.

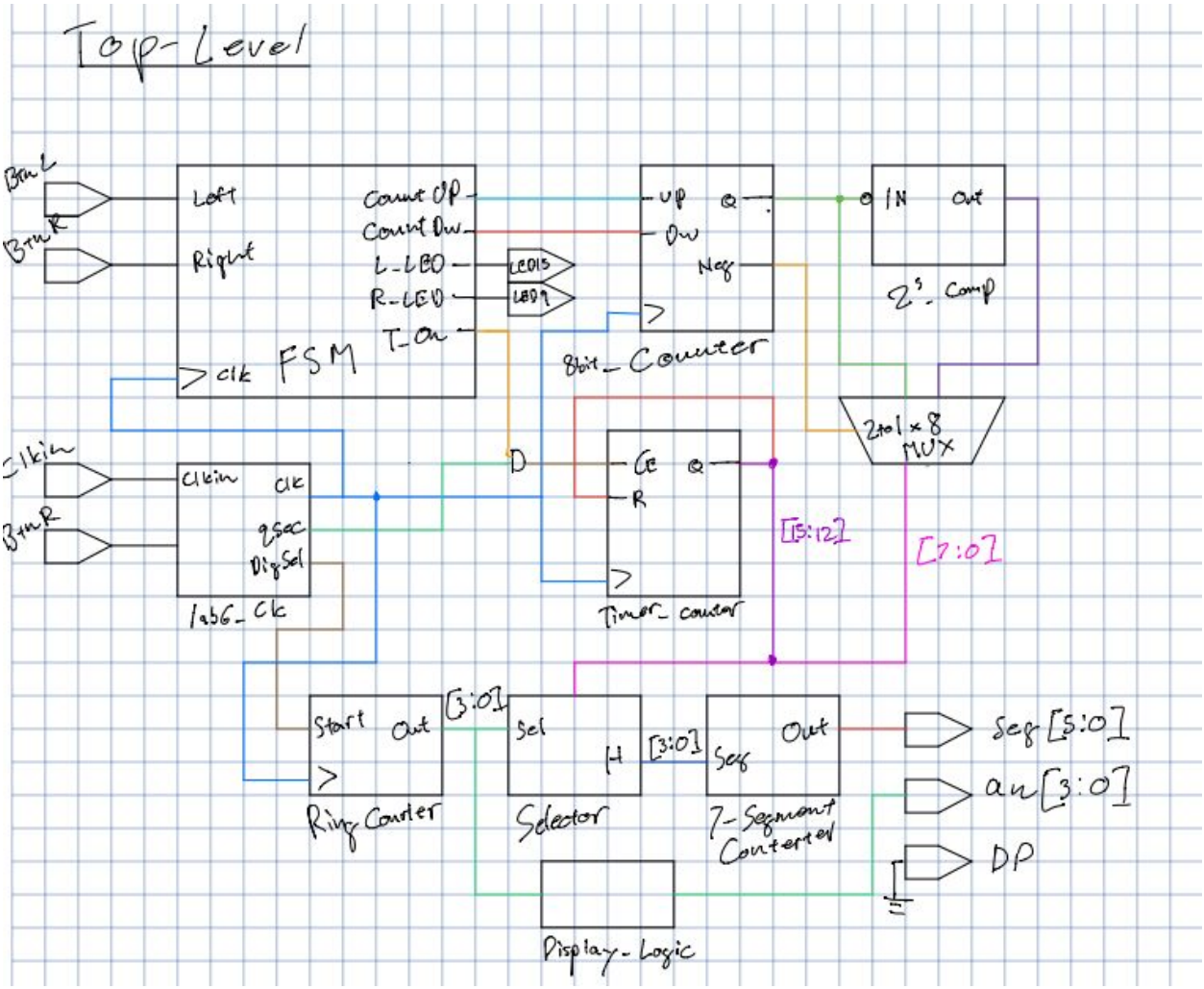
- Ring Counter:
  The ring counter is used as encode to create a one-hot/single active in 4-bit bus. This is used to set values for the selector. The module is created using a start(CE) and the clock(clk) which iterates through 4 flip flops as one state. In order to do this we connect the 4 flip flops in a complete loop and initialize one of the flip flops. This way we would get the outputs 0001, 0010, 0100, 1000, repeat.

- Selector:
  The selector is used to select a segment of it's inputs and output it with it's respective weight. This module is made with just boolean expressions with the inputs being the ring counter and the 16-bit counter and the output being the input to a seven_segment converter. As we review above the ring counter will feed a set of 4-bits with one active bit therefore giving it a state value for a certain part of the 16-bit you want to pass through. In this case, we want to pass through every 4-bits of the counter starting from 0. In pseudo code, it looks like this:
  - H is N[15:12] when sel=(1000)
  - H is N[11:8] when sel=(0100)
  - H is N[7:4] when sel=(0010)
  - H is N[3:0] when sel=(0001)

  When we put it in verilog it looks like this:
  - H = (N[15:12] & {4{(sel[3] & ~sel[2] & ~sel[1] & ~sel[0])}})|
    (N[11:8] & {4{(~sel[3] & sel[2] & ~sel[1] & ~sel[0])}})|
    (N[7:4] & {4{(~sel[3] & ~sel[2] & sel[1] & ~sel[0])}})|
    (N[3:0] & {4{(~sel[3] & ~sel[2] & ~sel[1] & sel[0])}});

- Seven Segment Converter:
  Finally we have the Seven Segment Converter. This module has been reviewed in previous lab reports and it works exactly the same. It takes in 4 inputs and uses boolean expressions to output the proper values to its respective segments. Unlike previous labs, we also needed to account for a negative sign. In order to do this, I made a case for when the ring counter for the specific display and negative are TRUE, every segment except for G(6) will be off/1.

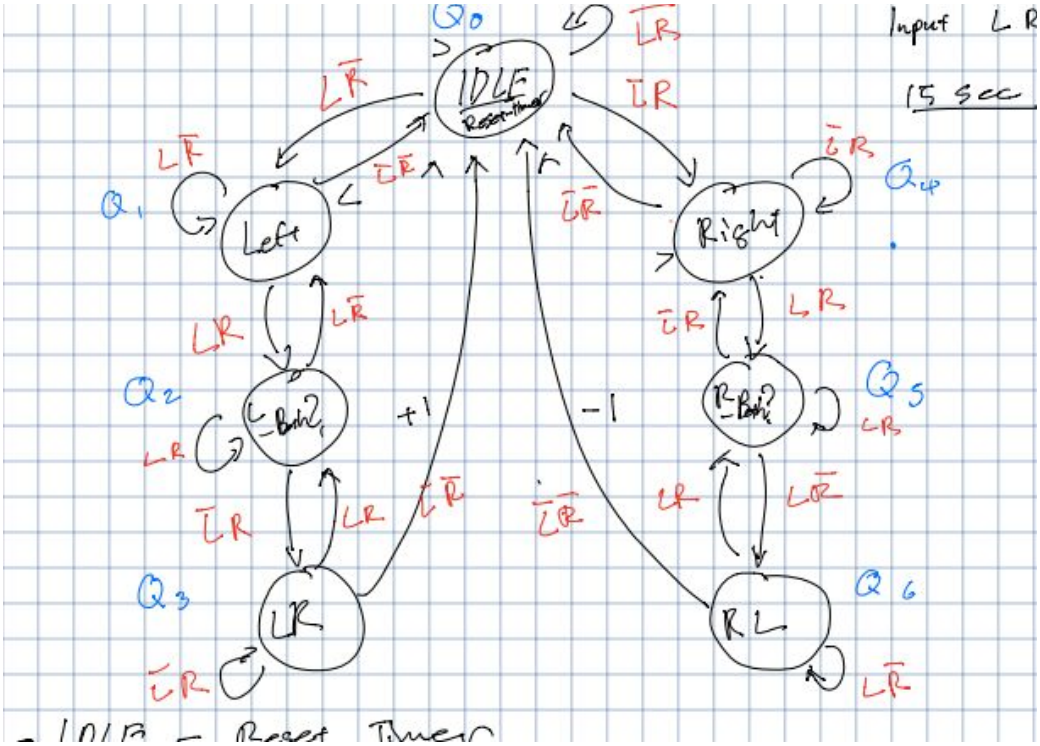
- Top Level:



- In this diagram we see all the modules that were used in this lab. I have compacted many of the logic for the wires to fit everything into the page.

- **Testing & Simulation:**
  In this lab the logic was mostly straightforward and simple. Unfortunately due to the many modules that were involved in this lab unlike the others, there was a higher chance for errors, therefore leading to more time searching for the cause of these errors. I bumped into a timing loop early on in my design and could not run any tests until I located the error. This process took a very long time, but after I found the issue I was able to visually test the sequential circuit using a visual inspection of the basys3 board.

- **Results:**
  1. State Diagram (Equations Explained in Module)



- IDLE: Is a state when no inputs are triggered
- Left: Is a state when the left trigger is triggered
- Right: Is a state when the right trigger is triggered
- L_Both: Is a state when both triggers are triggered after Left was initially triggered
- R_Both: Is a state when both triggers are triggered after Right was initially triggered
- LR: Is a state when Left is no longer triggered after state L_Both.
- RL: Is a state when Right is no longer triggered after state R_Both.

- Note: Since I did the opposite counts for the direction Turkey is going when designing the circuit the L and R are swapped on the Top_Level.

- **Conclusion:**
  This lab I got to experience designing my sequential circuit from scratch. Since I took CE12 my freshman year, I had forgotten how to do two's complement, but after getting help I was smoothly sailing through this lab. I'm glad my design is working after making a few bug fixes. If I had more time I would definitely go over the state diagram and simplify it, as there were many states that had similar functionality.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/18/2020 07:43:26 PM
// Design Name:
// Module Name: Top_Level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Top_Level(
    input clkin, btnU, btnL, btnR,
    output [15:0] led,
    output [3:0] an,
    output [6:0] seg,
    output dp
    );

    // Timing Wires
    wire clk;
    wire digsel;
    wire qsec;

    lab6_clks slowit(.clkin(clkin), .greset(btnU), .clk(clk), .digsel(digsel),
.qsec(qsec));

    // Edge Detector Wires
    wire Left;
    wire Right;

    //Edge_Detector Go_Edge(.clk(clk), .Btn(btnL), .out(Left));
    //Edge_Detector Stop_Edge(.clk(clk), .Btn(btnR), .out(Right));
    assign Left = btnR;
    assign Right = btnL;
```

```
    // Test
    //wire [1:0] test;
    //assign led[1] = Left;
    //assign led[0] = Right;
    //assign test[0] = Left;
    //assign test[1] = Right;

    // State Machine Wires
    wire Up;
    wire Dw;
    wire R;
    wire t1;

    State_Machine FSM(.clk(clk), .btnL(Left), .btnR(Right), .Up(Up), .Dw(Dw),
.L_LED(led[15]), .R_LED(led[9]), .R(R), .T_On(t1));

    // Counter Wires
    wire [7:0] P;
    wire [7:0] N;
    wire [7:0] Q;
    wire Neg;
    //wire [7:0] Unused1;

    //8bit counter
    Counter8UDL TurkeyCount8(.clk(clk), .Up(Up), .Dw(Dw), .din(8'b0), .Q(P),
.Z_out(Neg));
    Bit8_Adder TwosComp(.In(~P), .Out(N));
    MUX2_1x8 NegSwitch(.in0(P), .in1(N), .sel(Neg), .out(Q));

    // Timer Wires
    wire [7:0] Timer;
    //wire [15:4] Unused2;
    //4bit counter
//    wire [3:0] timer;
//    assign timer = {Timer[3:0]};
    Counter8UDL Timer_Display8(.clk(clk), .Up(qsec&t1&(~Timer[3:0])), .R(R),
.Q(Timer));

    // Display Wires
    wire [3:0] ring;
    wire [3:0] Neg_D;
    wire [3:0] Inputs;
    assign Neg_D = 4'b0;

    wire [15:0] main;
```

```
    assign main = {Timer[3:0], Neg_D[3:0], Q[7:0]};
    //assign seg[6] = Neg;

    Ring_Counter Ring(.start(digsel), .clk(clk), .out(ring));
    Selector Select(.sel(ring), .N(main), .H(Inputs));
    Segment_Display Display(.n(Inputs), .sego(seg), .neg(Neg&ring[2]),
.state(ring[2]));

    assign an[0] = ~(ring[0]);
    assign an[1] = ~(ring[1]);
    assign an[2] = ~(ring[2]&Neg);
    assign an[3] = ~(ring[3]&t1);
    assign dp = 1'b1;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/12/2020 03:47:40 PM
// Design Name:
// Module Name: State_Machine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module State_Machine(
    input clk, btnL, btnR,
    output Up, Dw, L_LED, R_LED, R, T_On
    );

    wire [6:0] PS;
    wire [6:0] NS;

    Control_Logic Control_Logic_FSM(.PS(PS), .NS(NS), .Left(btnL), .Right(btnR),
.CountUp(Up), .CountDw(Dw), .Reset(R), .L_LED(L_LED), .R_LED(R_LED), .On(T_On));

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(NS[0]), .Q(PS[0]));
    FDRE #(.INIT(1'b0)) Q123_FF[6:1] (.C({6{clk}}), .CE({6{1'b1}}), .D(NS[6:1]),
.Q(PS[6:1]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/12/2020 03:57:20 PM
// Design Name:
// Module Name: Control_Logic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Control_Logic(
    input Left, Right,
    input [6:0] PS,
    output [6:0] NS,
    output CountUp, CountDw, L_LED, R_LED, Reset, On
    );

    wire IDLE, L, L_B, LR, R, R_B, RL;
    wire Next_IDLE, Next_L, Next_L_B, Next_LR, Next_R, Next_R_B, Next_RL;

    // Present State
    assign IDLE     = PS[0];
    assign L        = PS[1];
    assign L_B      = PS[2];
    assign LR       = PS[3];
    assign R        = PS[4];
    assign R_B      = PS[5];
    assign RL       = PS[6];

    // Next State
    assign NS[0] = Next_IDLE;
    assign NS[1] = Next_L;
    assign NS[2] = Next_L_B;
    assign NS[3] = Next_LR;
```

```
    assign NS[4] = Next_R;
    assign NS[5] = Next_R_B;
    assign NS[6] = Next_RL;

    //Enter Logic
    assign Next_IDLE    = (~Left&~Right&(IDLE|L|LR|R|RL));
    assign Next_L       = (Left&~Right&(IDLE|L|L_B));
    assign Next_L_B     = (Left&Right&(L|L_B|LR));
    assign Next_LR      = (~Left&Right&(L_B|LR));
    assign Next_R       = (~Left&Right&(IDLE|R|R_B));
    assign Next_R_B     = (Left&Right&(R|R_B|RL));
    assign Next_RL      = (Left&~Right&(R_B|RL));

    // Outputs
    assign Reset    = IDLE;
    assign CountUp  = LR&~Right&~Left;
    assign CountDw  = RL&~Left&~Right;
    assign L_LED    = ~Left;
    assign R_LED    = ~Right;
    assign On       = Left|Right;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/07/2017 11:19:41 AM
// Design Name:
// Module Name: lab4_clks
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module lab6_clks(
    input clkin,
    input greset,  //btnU
    output clk,
    output digsel,
    output qsec,
    output fastclk);

       wire clk_int;
       assign fastclk = clk_int;
       clk_wiz_0 my_clk_inst (.clk_out1(clk_int), .reset(greset), .locked(),
.clk_in1(clkin));
       clkcntrl4 slowclk (.clk_int(clk_int), .seldig(digsel), .clk_out(clk),
.qsec(qsec));

       STARTUPE2 #(.PROG_USR("FALSE"), // Activate program event security feature.
Requires encrypted bitstreams.
                   .SIM_CCLK_FREQ(0.0)   // Set the Configuration Clock
Frequency(ns) for simulation.
                                      )
             STARTUPE2_inst (.CFGCLK(),  // 1-bit output: Configuration main clock
output
                             .CFGMCLK(), // 1-bit output: Configuration internal
```

```
oscillator clock output
                              .EOS(),     // 1-bit output: Active high output signal
indicating the End Of Startup.
                              .PREQ(),// 1-bit output: PROGRAM request to fabric
output
                              .CLK(),  // 1-bit input: User start-up clock input
                              .GSR(greset),  // 1-bit input: Global Set/Reset input
(GSR cannot be used for the port name)
                              .GTS(),  // 1-bit input: Global 3-state input (GTS
cannot be used for the port name)
                              .KEYCLEARB(), // 1-bit input: Clear AES Decrypter Key
input from Battery-Backed RAM (BBRAM)
                              .PACK(), // 1-bit input: PROGRAM acknowledge input
                              .USRCCLKO(), // 1-bit input: User CCLK input
                              .USRCCLKTS(), // 1-bit input: User CCLK 3-state enable
input
                               .USRDONEO(), // 1-bit input: User DONE pin output
control
                               .USRDONETS() // 1-bit input: User DONE 3-state enable
output
                             );  // End of STARTUPE2_inst instantiation

endmodule

module clk_wiz_0

 (// Clock in ports
  // Clock out ports
  output        clk_out1,
  // Status and control signals
  input         reset,
  output        locked,
  input         clk_in1
 );
  // Input buffering
  //------------------------------------
wire clk_in1_clk_wiz_0;
wire clk_in2_clk_wiz_0;
  IBUF clkin1_ibufg
   (.O (clk_in1_clk_wiz_0),
    .I (clk_in1));


  // Clocking PRIMITIVE
  //------------------------------------
```

```
// Instantiation of the MMCM PRIMITIVE
//    * Unused inputs are tied off
//    * Unused outputs are labeled unused

wire        clk_out1_clk_wiz_0;
wire        clk_out2_clk_wiz_0;
wire        clk_out3_clk_wiz_0;
wire        clk_out4_clk_wiz_0;
wire        clk_out5_clk_wiz_0;
wire        clk_out6_clk_wiz_0;
wire        clk_out7_clk_wiz_0;

wire [15:0] do_unused;
wire        drdy_unused;
wire        psdone_unused;
wire        locked_int;
wire        clkfbout_clk_wiz_0;
wire        clkfbout_buf_clk_wiz_0;
wire        clkfboutb_unused;
  wire clkout0b_unused;
 wire clkout1_unused;
 wire clkout1b_unused;
 wire clkout2_unused;
 wire clkout2b_unused;
 wire clkout3_unused;
 wire clkout3b_unused;
 wire clkout4_unused;
wire        clkout5_unused;
wire        clkout6_unused;
wire        clkfbstopped_unused;
wire        clkinstopped_unused;
wire        reset_high;

MMCME2_ADV
#(.BANDWIDTH            ("OPTIMIZED"),
  .CLKOUT4_CASCADE      ("FALSE"),
  .COMPENSATION         ("ZHOLD"),
  .STARTUP_WAIT         ("FALSE"),
  .DIVCLK_DIVIDE        (1),
  .CLKFBOUT_MULT_F      (9.125),
  .CLKFBOUT_PHASE       (0.000),
  .CLKFBOUT_USE_FINE_PS ("FALSE"),
  .CLKOUT0_DIVIDE_F     (36.500),
  .CLKOUT0_PHASE        (0.000),
  .CLKOUT0_DUTY_CYCLE   (0.500),
  .CLKOUT0_USE_FINE_PS  ("FALSE"),
```

```
    .CLKIN1_PERIOD        (10.0))
  mmcm_adv_inst
    // Output clocks
   (
    .CLKFBOUT            (clkfbout_clk_wiz_0),
    .CLKFBOUTB           (clkfboutb_unused),
    .CLKOUT0             (clk_out1_clk_wiz_0),
    .CLKOUT0B            (clkout0b_unused),
    .CLKOUT1             (clkout1_unused),
    .CLKOUT1B            (clkout1b_unused),
    .CLKOUT2             (clkout2_unused),
    .CLKOUT2B            (clkout2b_unused),
    .CLKOUT3             (clkout3_unused),
    .CLKOUT3B            (clkout3b_unused),
    .CLKOUT4             (clkout4_unused),
    .CLKOUT5             (clkout5_unused),
    .CLKOUT6             (clkout6_unused),
     // Input clock control
    .CLKFBIN             (clkfbout_buf_clk_wiz_0),
    .CLKIN1              (clk_in1_clk_wiz_0),
    .CLKIN2              (1'b0),
     // Tied to always select the primary input clock
    .CLKINSEL            (1'b1),
    // Ports for dynamic reconfiguration
    .DADDR               (7'h0),
    .DCLK                (1'b0),
    .DEN                 (1'b0),
    .DI                  (16'h0),
    .DO                  (do_unused),
    .DRDY                (drdy_unused),
    .DWE                 (1'b0),
    // Ports for dynamic phase shift
    .PSCLK               (1'b0),
    .PSEN                (1'b0),
    .PSINCDEC            (1'b0),
    .PSDONE              (psdone_unused),
    // Other control and status signals
    .LOCKED              (locked_int),
    .CLKINSTOPPED        (clkinstopped_unused),
    .CLKFBSTOPPED        (clkfbstopped_unused),
    .PWRDWN              (1'b0),
    .RST                 (reset_high));
  assign reset_high = reset;

  assign locked = locked_int;
// Clock Monitor clock assigning
```

```
//-------------------------------------
 // Output buffering
  //-----------------------------------

  BUFG clkf_buf
   (.O (clkfbout_buf_clk_wiz_0),
    .I (clkfbout_clk_wiz_0));



  BUFG clkout1_buf
   (.O   (clk_out1),
    .I   (clk_out1_clk_wiz_0));




endmodule

module clkcntrl4(
     input clk_int,
     output seldig,
     output clk_out,
     output qsec);

   //wire XLXN_38;
   //wire XLXN_39;
   wire XLXN_44;
   wire XLXN_47;
   wire XLXN_70;
   wire XLXN_71;
   wire XLXN_72;
   wire XLXN_73;
   wire XLXN_74;
   wire XLXN_75;
   wire XLXN_76;
   wire XLXN_77;
   wire XLXN_79;
   wire clkb2_DUMMY;

   GND  XLXI_24 (.G(XLXN_44));

   (* HU_SET = "XLXI_37_73" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_37 (.C(clkb2_DUMMY),
                                     .CE(XLXN_73),
                                     .CLR(XLXN_76),
```

```
                                .CEO(XLXN_72),
                                .Q0(),
                                .Q1(),
                                .Q2(XLXN_74),
                                .Q3(),
                                .TC());
(* HU_SET = "XLXI_38_74" *)
CB4CE_MXILINX_clkcntrl4  XLXI_38 (.C(clkb2_DUMMY),
                                .CE(XLXN_72),
                                .CLR(XLXN_76),
                                .CEO(XLXN_70),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC());
(* HU_SET = "XLXI_39_75" *)
CB4CE_MXILINX_clkcntrl4  XLXI_39 (.C(clkb2_DUMMY),
                                .CE(XLXN_70),
                                .CLR(XLXN_76),
                                .CEO(XLXN_71),
                               .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(XLXN_77),
                                .TC());
//(* HU_SET = "XLXI_40_76" *)
CB4CE_MXILINX_clkcntrl4  XLXI_40 (.C(clk_out),
                                .CE(XLXN_73),
                                .CLR(XLXN_76),
                                .CEO(XLXN_78),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC(XLXN_75));

CB4CE_MXILINX_clkcntrl4  XLXI_45 (.C(clk_out),
                                .CE(XLXN_78),
                                .CLR(XLXN_76),
                                .CEO(XLXN_79),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC());
```

```
   CB4CE_MXILINX_clkcntrl4  XLXI_44 (.C(clk_out),
                                    .CE(XLXN_79),
                                    .CLR(XLXN_76),
                                    .CEO(),
                                    .Q0(qsec0),
                                    .Q1(qsec2),
                                    .Q2(),
                                    .Q3(),
                                    .TC());

   AND3  I_12222 (.I0(qsec0),
                  .I1(qsec2),
                  .I2(XLXN_79),
                  //.I3(),
                  .O(qsec3));


   VCC  XLXI_41 (.P(XLXN_73));
   GND  XLXI_43 (.G(XLXN_76));
   BUF  XLXI_328 (.I(clk_int),
                  .O(clkb2_DUMMY));
`ifdef XILINX_SIMULATOR
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   BUF  XLXI_398 (.I(XLXN_75),.O(qsec));
   BUFG  XLXI_399 (.I(clk_int),.O(clk_out));
   //BUFG  XLXI_401 (.I(XLXN_77),.O(clk_out));
`else
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   BUF  XLXI_398 (.I(qsec3),.O(qsec));
   BUFG  XLXI_401 (.I(XLXN_77),.O(clk_out));
`endif

endmodule


module FTCE_MXILINX_clkcntrl4(C,
                              CE,
                              CLR,
                              T,
                              Q);

   parameter INIT = 1'b0;

    input C;
    input CE;
```

```
    input CLR;
    input T;
   output Q;

   wire TQ;
   wire Q_DUMMY;

   assign Q = Q_DUMMY;
   XOR2  I_36_32 (.I0(T),
                  .I1(Q_DUMMY),
                  .O(TQ));
   ///(* RLOC = "X0Y0" *)
   FDCE  I_36_35 (.C(C),
                  .CE(CE),
                  .CLR(CLR),
                  .D(TQ),
                  .Q(Q_DUMMY));
endmodule
`timescale 1ns / 1ps

module CB4CE_MXILINX_clkcntrl4(C,
                               CE,
                               CLR,
                               CEO,
                               Q0,
                               Q1,
                               Q2,
                               Q3,
                               TC);

    input C;
    input CE;
    input CLR;
   output CEO;
   output Q0;
   output Q1;
   output Q2;
   output Q3;
   output TC;

   wire T2;
   wire T3;
   wire XLXN_1;
   wire Q0_DUMMY;
   wire Q1_DUMMY;
   wire Q2_DUMMY;
```

```
wire Q3_DUMMY;
wire TC_DUMMY;

assign Q0 = Q0_DUMMY;
assign Q1 = Q1_DUMMY;
assign Q2 = Q2_DUMMY;
assign Q3 = Q3_DUMMY;
assign TC = TC_DUMMY;
(* HU_SET = "I_Q0_69" *)
FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q0 (.C(C),
                              .CE(CE),
                              .CLR(CLR),
                              .T(XLXN_1),
                              .Q(Q0_DUMMY));
(* HU_SET = "I_Q1_70" *)
FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q1 (.C(C),
                              .CE(CE),
                              .CLR(CLR),
                              .T(Q0_DUMMY),
                              .Q(Q1_DUMMY));
(* HU_SET = "I_Q2_71" *)
FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q2 (.C(C),
                              .CE(CE),
                              .CLR(CLR),
                              .T(T2),
                              .Q(Q2_DUMMY));
(* HU_SET = "I_Q3_72" *)
FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q3 (.C(C),
                              .CE(CE),
                              .CLR(CLR),
                              .T(T3),
                              .Q(Q3_DUMMY));
AND4  I_36_31 (.I0(Q3_DUMMY),
               .I1(Q2_DUMMY),
               .I2(Q1_DUMMY),
               .I3(Q0_DUMMY),
               .O(TC_DUMMY));
AND3  I_36_32 (.I0(Q2_DUMMY),
               .I1(Q1_DUMMY),
               .I2(Q0_DUMMY),
               .O(T3));
AND2  I_36_33 (.I0(Q1_DUMMY),
               .I1(Q0_DUMMY),
               .O(T2));
VCC  I_36_58 (.P(XLXN_1));
AND2  I_36_67 (.I0(CE),
```

```
                .I1(TC_DUMMY),
                .O(CEO));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2020 02:40:39 PM
// Design Name:
// Module Name: Segment_Display
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Segment_Display(
    input [3:0] n, neg, state,
    output [6:0] sego
    );

    wire s0, s1, s2, s3, s4, s5, s6;

//    assign sego[0] = (s0&(~state|~neg))|(state|neg);
//    assign sego[1] = (s1&(~state|~neg))|(state|neg);
//    assign sego[2] = (s2&(~state|~neg))|(state|neg);
//    assign sego[3] = (s3&(~state|~neg))|(state|neg);
//    assign sego[4] = (s4&(~state|~neg))|(state|neg);
//    assign sego[5] = (s5&(~state|~neg))|(state|neg);
//    assign sego[6] = (s6&(~state|~neg));

    assign sego[0] = s0|neg;
    assign sego[1] = s1|neg;
    assign sego[2] = s2|neg;
    assign sego[3] = s3|neg;
    assign sego[4] = s4|neg;
    assign sego[5] = s5|neg;
    assign sego[6] = s6&~neg;

    MUX8_1 SegA(.in({1'b0, n[0], n[0], 1'b0, 1'b0, ~n[0], 1'b0, n[0]}),
```

```
.sel(n[3:1]), .out(s0));
    MUX8_1 SegB(.in({1'b1, ~n[0], n[0], 1'b0, ~n[0], n[0], 1'b0, 1'b0}),
.sel(n[3:1]), .out(s1));
    MUX8_1 SegC(.in({1'b1, ~n[0], 1'b0, 1'b0, 1'b0, 1'b0, ~n[0], 1'b0}),
.sel(n[3:1]), .out(s2));
    MUX8_1 SegD(.in({n[0], 1'b0, ~n[0], n[0], n[0], ~n[0], 1'b0, n[0]}),
.sel(n[3:1]), .out(s3));
    MUX8_1 SegE(.in({1'b0, 1'b0, 1'b0, n[0], n[0], 1'b1, n[0], n[0]}), .sel(n[3:1]),
.out(s4));
    MUX8_1 SegF(.in({1'b0, n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b1, n[0]}), .sel(n[3:1]),
.out(s5));
    MUX8_1 SegG(.in({1'b0, ~n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b0, 1'b1}),
.sel(n[3:1]), .out(s6));

    //assign sego[6] = neg;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 03:54:47 PM
// Design Name:
// Module Name: Selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Selector(
    input [3:0] sel,
    input [15:0] N,
    output [3:0] H
    );


    assign H =  (N[15:12] & {4{( sel[3] & ~sel[2] & ~sel[1] & ~sel[0])}})|
    (N[11:8] & {4{(~sel[3] & sel[2] & ~sel[1] & ~sel[0])}})|
    (N[7:4] & {4{(~sel[3] & ~sel[2] & sel[1] & ~sel[0])}})|
    (N[3:0] & {4{(~sel[3] & ~sel[2] & ~sel[1] & sel[0])}});



    //H is N[15:12] when sel=(1000);
    //H is N[11:8] when sel=(0100);
    //H is N[7:4] when sel=(0010);
    //H is N[3:0] when sel=(0001);


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/13/2020 06:08:33 PM
// Design Name:
// Module Name: Full_Adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Full_Adder(
    input A,
    input B,
    input Cin,
    output S,
    output Cout
    );

    assign S = A ^ (B ^ Cin);
    assign Cout = (B & Cin)|(A & Cin)|(A & B);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/19/2020 04:24:08 PM
// Design Name:
// Module Name: 8Bit_Adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Bit8_Adder(
    input [7:0] In,
    output [7:0] Out
    );

    wire [7:0] C;

    Full_Adder bit0(.A(In[0]), .B(1'b0), .Cin(1'b1), .S(Out[0]), .Cout(C[0]));
    Full_Adder bit1(.A(In[1]), .B(1'b0), .Cin(C[0]), .S(Out[1]), .Cout(C[1]));
    Full_Adder bit2(.A(In[2]), .B(1'b0), .Cin(C[1]), .S(Out[2]), .Cout(C[2]));
    Full_Adder bit3(.A(In[3]), .B(1'b0), .Cin(C[2]), .S(Out[3]), .Cout(C[3]));
    Full_Adder bit4(.A(In[4]), .B(1'b0), .Cin(C[3]), .S(Out[4]), .Cout(C[4]));
    Full_Adder bit5(.A(In[5]), .B(1'b0), .Cin(C[4]), .S(Out[5]), .Cout(C[5]));
    Full_Adder bit6(.A(In[6]), .B(1'b0), .Cin(C[6]), .S(Out[6]), .Cout(Out[7]));
    //Full_Adder bit7(.A(In[7]), .B(1'b0), .Cin(C[7]), .S(Out[0]), .Cout(C[0]));


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 04:00:44 PM
// Design Name:
// Module Name: Ring_Counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Ring_Counter(
    input start, clk,
    output [3:0] out
    );

    //wire [1:0] Q;  // comment this line if you want Q as an output
    wire d0,d1,d2,d3;

    FDRE #(.INIT(1'b1)) Ringcounter1 (.C(clk), .R(1'b0), .CE(start), .D(d3), .Q(d0));
    FDRE #(.INIT(1'b0)) Ringcounter2 (.C(clk), .R(1'b0), .CE(start), .D(d0), .Q(d1));
    FDRE #(.INIT(1'b0)) Ringcounter3 (.C(clk), .R(1'b0), .CE(start), .D(d1), .Q(d2));
    FDRE #(.INIT(1'b0)) Ringcounter4 (.C(clk), .R(1'b0), .CE(start), .D(d2), .Q(d3));

    assign out = {d3, d2, d1, d0};

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2020 01:52:49 AM
// Design Name:
// Module Name: MUX2_1x8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module MUX2_1x8(
    input [7:0] in0,
    input [7:0] in1,
    input sel,
    output [7:0] out
    );

    assign out = (in0 & {8{~sel}})|(in1[7:0] & {8{sel}});

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/02/2020 10:48:48 PM
// Design Name:
// Module Name: countUD4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD4L(
    input clk, Up, Dw, LD, R,
    input [3:0] Din,
    output [3:0] Q, // uncomment this line if you want Q as an output
    output UTC, DTC
    );

    //wire [1:0] Q;  // comment this line if you want Q as an output
    wire d0, d1, d2, d3;
    wire q0, q1, q2, q3;
    wire t1;

    assign t1 = (LD|Up^Dw);
    assign Q = {q3, q2, q1, q0};

    assign d0 = ((q0^(Up|Dw))&~LD)|LD&Din[0];
    assign d1 = ((q1^((Up&q0)|(Dw&~q0)))&~LD)|LD&Din[1];
    assign d2 = ((q2^((Up&q0&q1)|(Dw&~q0&~q1)))&~LD)|LD&Din[2];
    assign d3 = ((q3^((Up&q0&q1&q2)|(Dw&~q0&~q1&~q2)))&~LD)|LD&Din[3];

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(R), .CE(t1), .D(d0), .Q(q0));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(R), .CE(t1), .D(d1), .Q(q1));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(R), .CE(t1), .D(d2), .Q(q2));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(R), .CE(t1), .D(d3), .Q(q3));
```

```
    assign UTC = Q[3] & Q[2] & Q[1] & Q[0];
    assign DTC = ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/22/2020 06:44:00 PM
// Design Name:
// Module Name: Counter8UDL
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Counter8UDL(
    input clk, Up, Dw, LD, R,
    input [7:0] din,
    output UTC, DTC, Z_out,
    output [7:0] Q
    );

    wire u1, u2, u3, u4;
    wire d1, d2, d3, d4;
    wire Up1, Up2, Up3;
    wire Dw1, Dw2, Dw3;

    countUD4L counter1(.clk(clk), .Up(Up), .Dw(Dw), .R(R), .LD(LD), .Din(din[3:0]),
.Q(Q[3:0]), .UTC(u1), .DTC(d1));
    assign Up1 = u1&Up&~Dw;
    assign Dw1 = d1&~Up&Dw;
    countUD4L counter2(.clk(clk), .Up(Up1), .Dw(Dw1), .R(R), .LD(LD),
.Din(din[7:4]), .Q(Q[7:4]));

    assign Z_out = Q[7];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/22/2020 07:26:27 PM
// Design Name:
// Module Name: Test
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Test();
  reg clkin, btnR, btnL, btnU;
  wire [15:0] led;
  wire [6:0] seg;
  wire [3:0] an;
  wire dp;
  wire [7:0] D7Seg3,D7Seg2,D7Seg1,D7Seg0;

  Top_Level
   UUT (
      .clkin(clkin),
      .btnR(btnR),
      .btnL(btnL),
      .btnU(btnU),
      .seg(seg),
      .dp(dp),
      .led(led),
      .an(an)
      );
  show_7segDisplay  showit (.seg(seg),.dp(dp),.an(an),
              .D7Seg0(D7Seg0),.D7Seg1(D7Seg1),.D7Seg2(D7Seg2),.D7Seg3(D7Seg3));


// Run this simulation for 3ms. If correct TX_ERROR should be 0 at the end.
```

```
// UTC should be high and then go low at 2,705us and go low at 2,706.3us.

  parameter PERIOD = 10;
  parameter real DUTY_CYCLE = 0.5;
  parameter OFFSET = 2;


    initial    // Clock process for clkin
    begin
      btnU = 1'b0;

    #OFFSET
        clkin = 1'b1;
    forever
    begin
      #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
    end
    end

    initial
    begin
    // Going Left to Right
    #100;
    btnL=1'b1;
    btnR=1'b0;
    #200;
    btnL=1'b1;
    btnR=1'b1;
    #300;
    btnL=1'b0;
    btnR=1'b1;
    #400;
    btnL=1'b0;
    btnR=1'b0;
    // Going Right to Left
    #500;
    btnL=1'b0;
    btnR=1'b1;
    #600;
    btnL=1'b1;
    btnR=1'b1;
    #700;
    btnL=1'b1;
    btnR=1'b0;
    #800;
    btnL=1'b0;
```

```
btnR=1'b0;
// Enter Left and retreating Left
#900;
btnL=1'b1;
btnR=1'b0;
#1000;
btnL=1'b1;
btnR=1'b1;
#1100;
btnL=1'b1;
btnR=1'b0;
#1200;
btnL=1'b0;
btnR=1'b0;
// Enter Left, Wander, and going Right
#1300;
btnL=1'b1;
btnR=1'b0;
#1400;
btnL=1'b1;
btnR=1'b1;
#1500;
btnL=1'b0;
btnR=1'b1;
#1600;
btnL=1'b1;
btnR=1'b1;
#1700;
btnL=1'b0;
btnR=1'b1;
#1800;
btnL=1'b0;
btnR=1'b0;
// Enter Right and retreating Right
#1900;
btnL=1'b0;
btnR=1'b1;
#2000;
btnL=1'b1;
btnR=1'b1;
#2100;
btnL=1'b1;
btnR=1'b0;
#2200;
btnL=1'b1;
btnR=1'b1;
```

```
#2300;
btnL=1'b0;
btnR=1'b1;
#2400;
btnL=1'b0;
btnR=1'b0;
// Enter Right, Wander Leave Left
#2500;
btnL=1'b0;
btnR=1'b1;
#2600;
btnL=1'b1;
btnR=1'b1;
#2700;
btnL=1'b1;
btnR=1'b0;
#2800;
btnL=1'b1;
btnR=1'b1;
#2900;
btnL=1'b1;
btnR=1'b0;
#3000;
btnL=1'b0;
btnR=1'b0;
// Wander for 15 Sec
#3100;
btnL=1'b0;
btnR=1'b1;
#3200;
btnL=1'b1;
btnR=1'b1;
#3300;
btnL=1'b1;
btnR=1'b0;
#3400;
btnL=1'b1;
btnR=1'b1;
#3500;
btnL=1'b1;
btnR=1'b1;
#3600;
btnL=1'b1;
btnR=1'b0;
#3700;
btnL=1'b1;
```

```
    btnR=1'b1;
    #3800;
    btnL=1'b1;
    btnR=1'b0;
    #3900;
    btnL=1'b1;
    btnR=1'b1;
    #4000;
    btnL=1'b1;
    btnR=1'b0;
    #4100;
    btnL=1'b1;
    btnR=1'b1;
    #4200;
    btnL=1'b1;
    btnR=1'b0;
    #4300;
    btnL=1'b1;
    btnR=1'b1;
    #4400;
    btnL=1'b1;
    btnR=1'b0;
    #4500;
    btnL=1'b1;
    btnR=1'b0;
    #4600;
    end

endmodule


module show_7segDisplay (
 input [6:0] seg,
 input dp,
 input [3:0] an,
 output reg [7:0] D7Seg0, D7Seg1, D7Seg2,D7Seg3);

 reg [7:0] val;

 wire AN0, AN1, AN2, AN3;
 assign AN0=an[0];
 assign AN1=an[1];
 assign AN2=an[2];
 assign AN3=an[3];

  always @(AN0 or val)
```

```
begin
  if (AN0 == 0) D7Seg0 <= val;
  else if (AN0 == 1) D7Seg0 <= " ";
  else D7Seg0 <= 8'bX;   //  non-blocking assignment
end

always @(AN1 or val)
begin
  if (AN1 == 0) D7Seg1 <= val;
  else if (AN1 == 1) D7Seg1 <= " ";
  else D7Seg1 <= 8'bX;   //  non-blocking assignment
end

always @(AN2 or val)
begin
  if (AN2 == 0) D7Seg2 <= val;
  else if (AN2 == 1) D7Seg2 <= " ";
  else D7Seg2 <= 8'bX;   //  non-blocking assignment
end

always @(AN3 or val)
begin
  if (AN3 == 0) D7Seg3 <= val;
  else if (AN3 == 1) D7Seg3 <= " ";
  else D7Seg3 <= 8'bX;   //  non-blocking assignment
end

always @(seg)
case (seg)
7'b0111111:
     val = "-";
7'b1111111:
     val = " ";
7'b1000000:
     val = "0";
7'b1111001:
     val = "1";
7'b0100100:
     val = "2";
7'b0110000:
     val = "3";
7'b0011001:
     val = "4";
7'b0010010:
     val = "5";
7'b0000010:
```

```
      val = "6";
  7'b1111000:
      val = "7";
  7'b0000000:
      val = "8";
  7'b0011000:
      val = "9";
  7'b0001000:
      val = "A";
  7'b0000011:
      val = "B";
  7'b1000110:
      val = "C";
  7'b0100001:
      val = "D";
  7'b0000110:
      val = "E";
  7'b0001110:
      val = "F";
  default:
      val = 8'bX;
  endcase

endmodule
```


7.800000 us
0 ns
5,000 ns
10,000 ns
15,000 ns
20,000 ns
25,000 ns
30,000 ns
35,000 ns
40,000 ns
45,000 ns
50,000 ns
55,000 ns
60,000 ns
65,00
0000000a
0.5
00000002
Activate Windows