Lab 4
Simon Carballo
1618309
11/13/2020
Section D

Write Up

- **Description:** In this lab we were asked to design and implement D flip-flop counters for our sequential circuit. The counters are used to create a 16-bit counter that can increment, decrement and load values. These values are operated with buttons, and switches that. The outputs for the design will be LEDs to detect UTC, DTC, and load values as well as the four segment displays.

- **Design:** For this design we have the Top Level that calls to 7 subsections which includes: Clock, Edge_Detector(Up, Dw), 16-bit Counter, 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 so I do not know how to go into depth on every detail. Basically this module is 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).

    - Edge_Detector:
      The Edge Detector is used to input Up and Dw into the 16-bit counter. The Edge Detector will generate a high value for one clock cycle if the past two inputs consist of a 0 followed by a 1. Basically, if the BtnU or BtnD is pressed, it will output one high value which feeds into the counter.

    - 16-bit Counter:
      Assuming that we know how adders work, we will be using 4 counters of 4 bits to create the model for a 16-bit counter.
        - 4-bit counter:
          The four bit counter is created using 4 D flipflops which are connected together using a boolean expression found by creating a truth table:

	Present State				Next State | Up = 1 Dw = 0				Next State | Up = 0 Dw = 1			
Val	Q3	Q2	Q1	Q0	D3	D2	D1	D0	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	1	1	1	1

1	0	0	0	1	0	0	1	0	0	0	0	0
2	0	0	1	0	0	0	1	1	0	0	0	1
3	0	0	1	1	0	1	0	0	0	0	1	0
4	0	1	0	0	0	1	0	1	0	0	1	1
5	0	1	0	1	0	1	1	0	0	1	0	0
6	0	1	1	0	0	1	1	1	0	1	0	1
7	0	1	1	1	1	0	0	0	0	1	1	0
8	1	0	0	0	1	0	0	1	0	1	1	1
9	1	0	0	1	1	0	1	0	1	0	0	0
10	1	0	1	0	1	0	1	1	1	0	0	1
11	1	0	1	1	1	1	0	0	1	0	1	0
12	1	1	0	0	1	1	0	1	1	0	1	1
13	1	1	0	1	1	1	1	0	1	1	0	0
14	1	1	1	0	1	1	1	1	1	1	0	1
15	1	1	1	1	0	0	0	0	1	1	1	0

- With this truth table I used Kmaps to find the boolean expression for the next state(D).
    - Example: Down State

D3			
1	0	1	0
0	0	1	1
0	0	1	1
0	0	1	1

- D3 = Q1Q2+Q1Q4+Q1Q3+~Q1~Q2~Q3~Q4

D2			
1	0	0	1

0	1	1	0
0	1	1	0
0	1	1	0

- D2 = ~Q2~Q4~Q3+Q4Q2+Q3Q2

D1			
1	1	1	1
0	0	0	0
1	1	1	1
0	0	0	0

- D1 = ~Q4~Q3+Q4Q3

D0			
1	1	1	1
0	0	0	0
0	0	0	0
1	1	1	1

- D0 = ~Q4

- By evaluating these maps we can now grasp a pattern to minimize these boolean expressions using the following formula:
  - Qn^(ΣQn-1)
- Therefore we can make these boolean expressions:
  - d0 = ((q0^(Up|Dw))&~LD)|LD&Din[0];
  - d1 = ((q1^((Up&q0)|(Dw&~q0)))&~LD)|LD&Din[1];
  - d2 = ((q2^((Up&q0&q1)|(Dw&~q0&~q1)))&~LD)|LD&Din[2];
  - d3 = ((q3^((Up&q0&q1&q2)|(Dw&~q0&~q1&~q2)))&~LD)|LD&Din[3];
- On top of the formula we covered, you also see the integration of LD, Din, Up and Dw. You could integrate this into your truth table, but I found it easier to add the inputs after the fact to better understand the expression.
- The LD will decide whether or not the counters will be loaded with values from the SW input (Din).

        - Finally to make this an official adder we need the sum and the cout which is expressed as Q[3:0] for sum and UTC, DTC for cout. The UTC and DTC is made by anding the sum or nanding the sums.

    - In order to make this an official 16-bit adder we call 4 4-bit counting modules and connect them together using wires. Using the UTC and DTC as inputs for the next 4-bit module

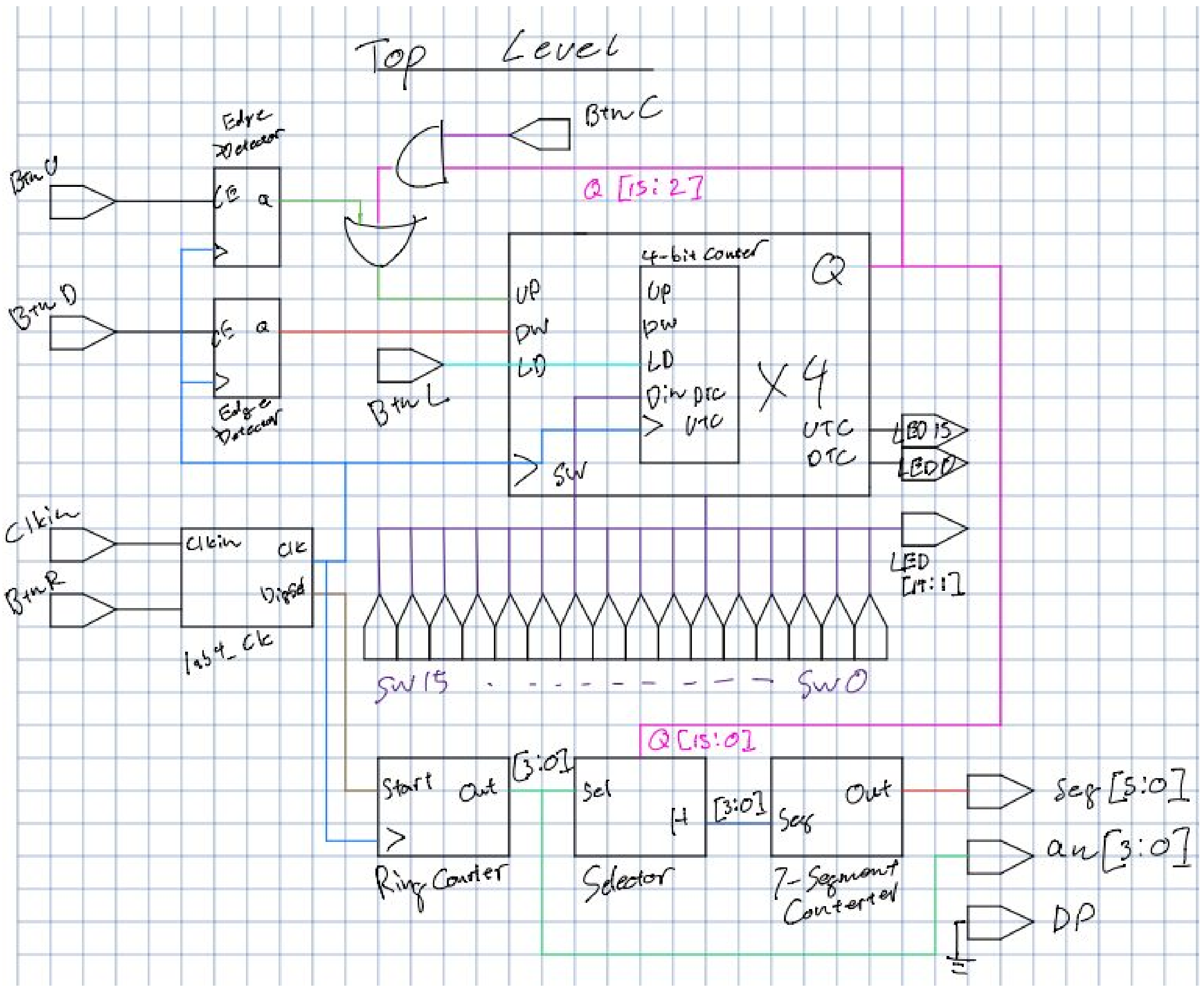
- <u>Ring Counter:</u>
  The ring counter is used as encode to create a one-hot/single active in 4-bit bus. This is used to set values for the selector. The module is created using a start(CE) and the clock(clk) which iterates through 4 flip flops as one state. In order to do this we connect the 4 flip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

- <u>Selector:</u>
  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])}});

- <u>Seven Segment Converter:</u>
  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.

- <u>Top Level:</u>



- In this Diagram you can see how each module is connected with each other to make the sequential circuit work. It may seem like there is only one segment display, and that is because the ring counter is also connected to the 4 segments(an) which displays the proper values according to the state of the counter.

- **Testing & Simulation:**
  This was a very stressful lab as it is the first use of clocks. This meant that there were timing loop errors to account for. Timing loops are accidentally created by creating a pathway that can loop back into itself. This error is common, and could cause many hours of stressful digging to find the problem. My design had a system loop which prevented me from running simulations until the weekend when I figured out the cause. Simulations were run with the provided test fill, and making the necessary adjustments to test other functions. One of the TAs was a great help in explaining how to debug specific wires with the simulation which really helped figure out the small undetectable problems in my design.

- **Results:**
    1. By holding down btnR the display an[3:1] goes dark. This is basically caused by resetting all of the counters. Meaning that the clk will stay at a constant low leaving the ring counter to only display a single display. I only think this is the reason because the output of the ring counter is the power to the displays.
    2. When switching from clk to the fastclk, the displays an[3:1] are dimmer, this is most likely because of the speed it is oscillating. This can be proven by hitting btnC, which counts to the Max FFFC immediately. On top of that the Up and Dw sometimes skips a number because of how fast the clock is oscillating creating a glitch in the edge detector.

- **Conclusion:**
  In conclusion I learned how to design a sequential circuit of D Flip Flop Counters, that can be used to store information and display through multiple seven segments. I better understand how to create a truth table for the counters and the benefits to having counters. I did feel a lot of frustration with the amount of errors I had to deal with during the designing process so I should learn to better organize my code for ease of future troubleshooting.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 02:06:48 AM
// 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, btnR, btnC, btnU, btnD, btnL,
    input [15:0] sw,
    output [15:0] led,
    output [6:0] seg,
    output [3:0] an,
    output dp
    );

    wire clk;
    wire Digwire;
    wire [15:0] Q;
    wire t1, t2, t3;
    wire up, dw;
    wire utc, dtc;
    wire [3:0] c;
    wire [3:0] r;
    wire fclk;

    assign an = ~r;
//    assign UTC = utc;
//    assign DTC = dtc;

    assign dp = ~((utc|an[3])&(dtc|an[0]));
    assign led[14:1] = sw[14:1];
    assign led[15]=dtc;
    assign led[0]=utc;

    //Clock
    lab4_clks clock(.clkin(clkin), .greset(btnR), .clk(clk), .fastclk(fclk),
.digsel(Digwire));

    ClockCase ClockButton(.Btn(btnC), .clk(clk), .Q(Q), .out(t1));
    Edge_Detector UpButton(.Btn(btnU), .clk(clk), .out(t2));
    assign up = (t2 | btnC &~(&Q[15:2]));
    Edge_Detector DownButton(.Btn(btnD), .clk(clk), .out(t3));
    assign dw = t3;

    counterUD16L Counter(.clk(fclk), .Up(up), .Dw(dw), .LD(btnL), .din(sw), .Q(Q),
.UTC(utc), .DTC(dtc));

    Ring_Counter Ring(.start(Digwire), .clk(clk), .out(r));
    Selector Select(.sel(r), .N(Q), .H(c));
    Segment_Display Display(.n(c), .sego(seg));


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 02:04:10 AM
// Design Name:
// Module Name: counterUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module counterUD16L(
    input clk, Up, Dw, LD,
    input [15:0] din,
    output UTC, DTC,
    output [15:0] 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), .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), .LD(LD), .Din(din[7:4]),
.Q(Q[7:4]), .UTC(u2), .DTC(d2));
    assign Up2 = u1&u2&Up&~Dw;
    assign Dw2 = d1&d2&~Up&Dw;
    countUD4L counter3(.clk(clk), .Up(Up2), .Dw(Dw2), .LD(LD), .Din(din[11:8]),
.Q(Q[11:8]), .UTC(u3), .DTC(d3));
    assign Up3 = u1&u2&u3&Up&~Dw;
    assign Dw3 = d1&d2&d3&~Up&Dw;
    countUD4L counter4(.clk(clk), .Up(Up3), .Dw(Dw3), .LD(LD), .Din(din[15:12]), 
.Q(Q[15:12]), .UTC(u4), .DTC(d4));

    assign UTC = u1&u2&u3&u4;
    assign DTC = d1&d2&d3&d4;

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,
    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];

    //assign d0 = ((~Q[0]&Up&~Dw)|(~Q[4]&~Up&Dw))&~LD | LD&Din[0];
    //assign d1 = (((Q[1]^Q[0])&Up&~Dw)|(((~Q[4]&Q[3])|(Q[4]&Q[3]))&~Up&Dw))&~LD |
LD&Din[1];
    //assign d2 =
((((Q[2]&~Q[0])|(Q[2]&~Q[1])|(~Q[2]&Q[1]&Q[0]))&Up&~Dw)|(((~Q[2]&~Q[3]&~Q[4])|(Q[4]&
Q[2])|(Q[3]&Q[2]))&~Up&Dw))&~LD | LD&Din[2];
    //assign d3 =
((((Q[3]&~Q[2])|(Q[3]&~Q[0])|(Q[3]&~Q[1])|(~Q[3]&Q[2]&Q[1]&Q[0]))&Up&~Dw)|((((Q[1]&Q
[2])|(Q[1]&Q[4])|(Q[1]&Q[3])|(~Q[1]&~Q[2]&~Q[3]&~Q[4]))&~Up&Dw)))&~LD | LD&Din[3];

    //assign d1 = ((Q[0]^((Up&~Dw)|(~Up&Dw))))&~LD | LD&Din[0];
    //assign d2 = ((Q[1]^((Up&~Dw&Q[0])|(~Up&Dw&~Q[0]))))&~LD | LD&Din[1];
    //assign d3 = ((Q[2]^((Up&~Dw&Q[0]&Q[1])|(~Up&Dw&~Q[0]&~Q[1]))))&~LD | LD&Din[2];
    //assign d4 = ((Q[3]^((Up&~Dw&Q[0]&Q[1]&Q[2])|(~Up&Dw&~Q[0]&~Q[1]&~Q[2]))))&~LD
| LD&Din[3];

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(1'b0), .CE(t1), .D(d0), .Q(q0));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(t1), .D(d1), .Q(q1));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(t1), .D(d2), .Q(q2));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .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: 01/25/2018 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 lab4_clks(
    input clkin,
    input greset,  //btnR
    output clk,   // system clock for Lab 4
    output digsel, // signal to advance ring counter for display
    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));

       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);

   //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 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(),
                                     .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(),
                                     .Q0(),
                                     .Q1(),
                                     .Q2(),
                                     .Q3(),
                                     .TC(XLXN_75));
   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));
   BUFG  XLXI_399 (.I(clk_int),.O(clk_out));
`else
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   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
```

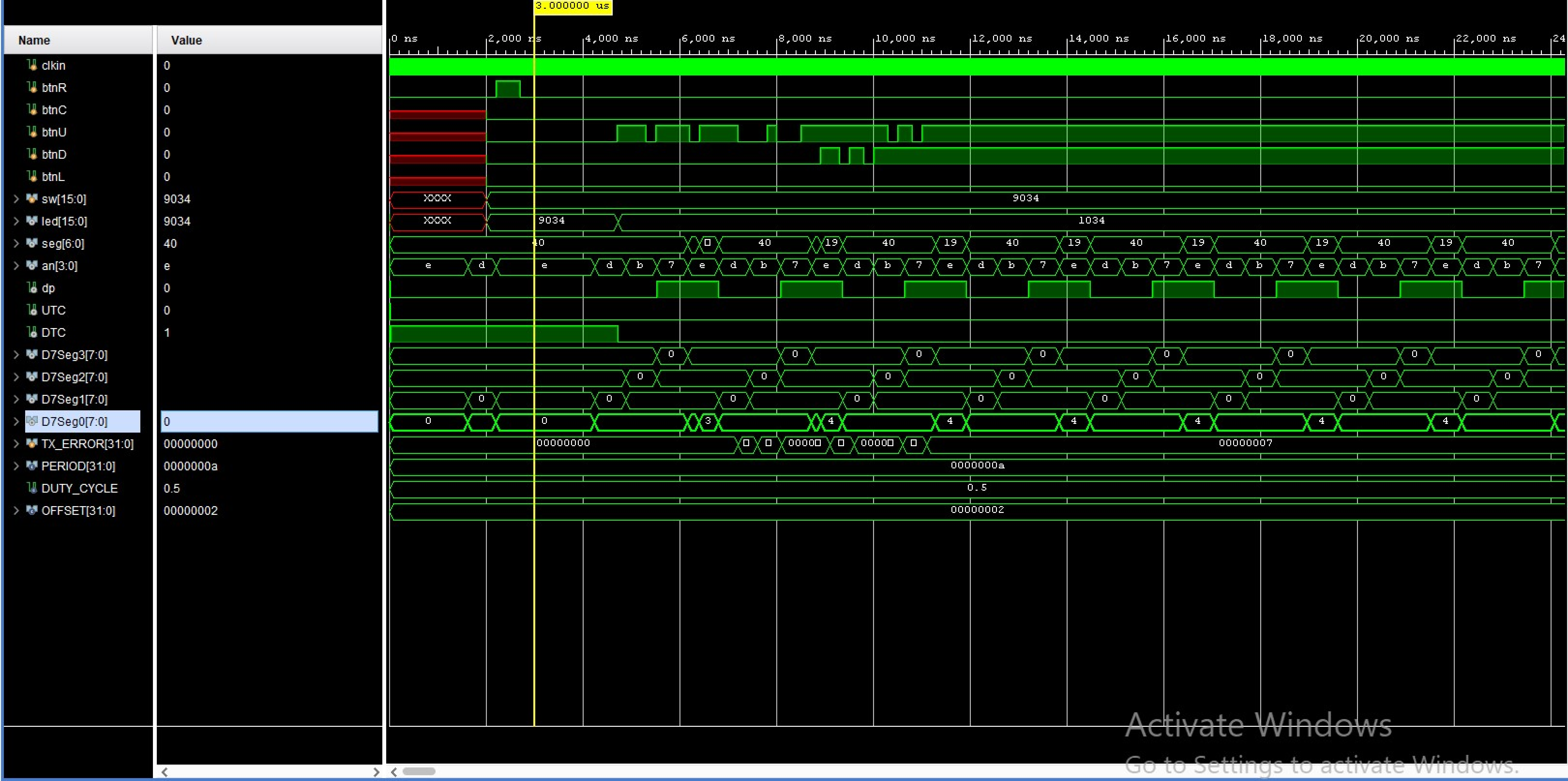
```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/03/2020 02:06:48 AM
// Design Name:
// Module Name: Edge_Detector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Edge_Detector(
    input Btn, clk,
    output out
    );

    wire t1;

    FDRE #(.INIT(1'b0)) Edge (.C(clk), .R(1'b0), .CE(1'b1), .D(Btn), .Q(t1));

    assign out = Btn&~t1;

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: 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
```

```verilog
`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,
    output [6:0] sego
    );

    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(sego[0]));
    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(sego[1]));
    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(sego[2]));
    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(sego[3]));
    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(sego[4]));
    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(sego[5]));
    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(sego[6]));

endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2020 01:27:56 AM
// Design Name:
// Module Name: MUX8_1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies: _!
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module MUX8_1(
    input [7:0] in,
    input [2:0] sel,
    output out
    );

    assign out = (in[0] & ~sel[2] & ~sel[1] & ~sel[0])|
    (in[1] & ~sel[2] & ~sel[1] & sel[0])|
    (in[2] & ~sel[2] & sel[1] & ~sel[0])|
    (in[3] & ~sel[2] & sel[1] & sel[0])|
    (in[4] & sel[2] & ~sel[1] & ~sel[0])|
    (in[5] & sel[2] & ~sel[1] & sel[0])|
    (in[6] & sel[2] & sel[1] & ~sel[0])|
    (in[7] & sel[2] & sel[1] & sel[0]);

endmodule
```



Name	Value
clkin	0
btnR	0
btnC	0
btnU	0
btnD	0
btnL	0
sw[15:0]	9034
led[15:0]	9034
seg[6:0]	40
an[3:0]	e
dp	0
UTC	0
DTC	1
D7Seg3[7:0]	
D7Seg2[7:0]	
D7Seg1[7:0]	
D7Seg0[7:0]	0
TX_ERROR[31:0]	00000000
PERIOD[31:0]	0000000a
DUTY_CYCLE	0.5
OFFSET[31:0]	00000002