CECS 460 Fall 2019 Project 1

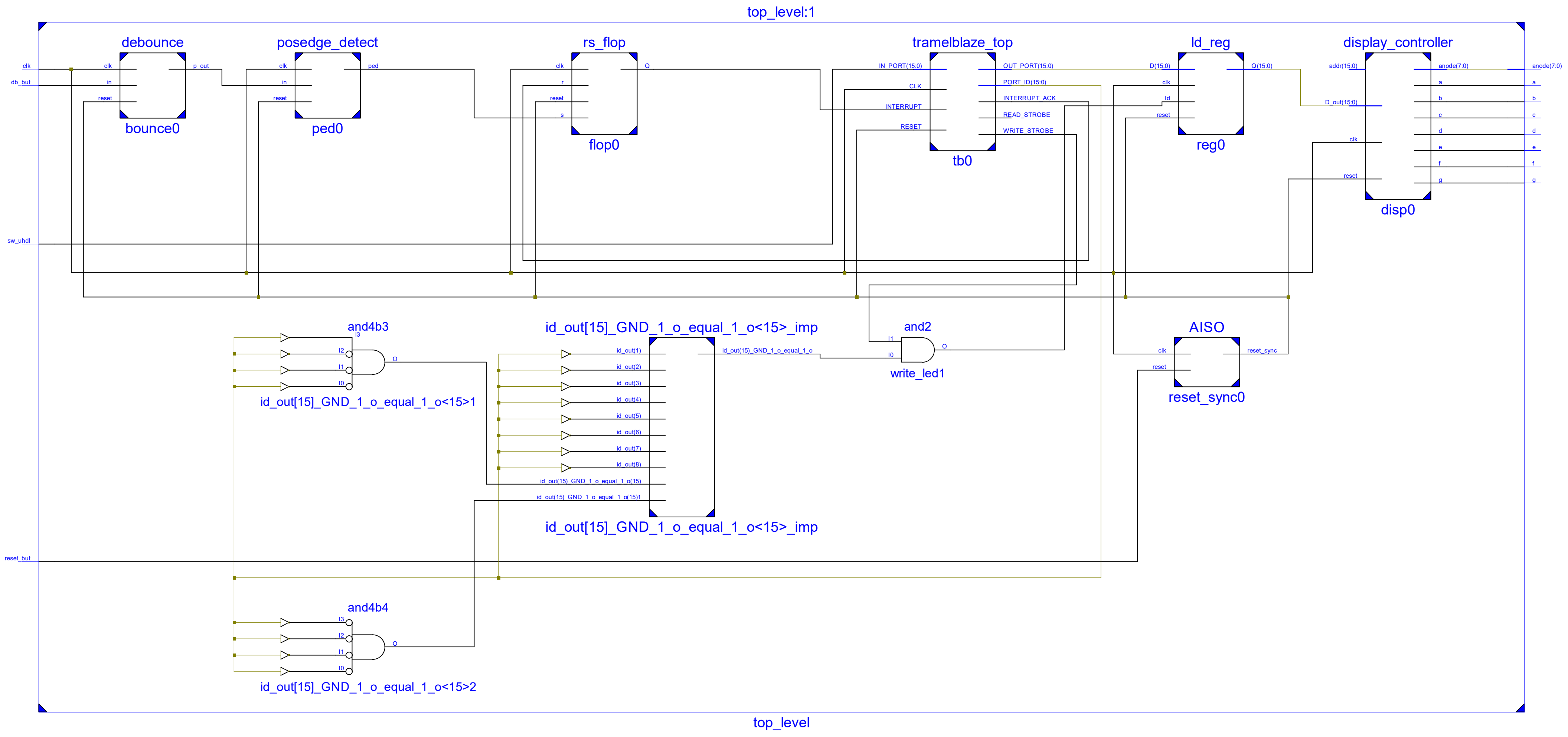
John Tramel

Zachery Takkesh -015656509

Purpose: The project is a review of the topics covered in 201, 301, and 361 while introducing the TramelBlaze.

Project one introduces the TramelBlaze processor designed by Professor John Tramel. The TramelBlaze is a 16-bit processor core designed to emulate the Xilinx PicoBlaze for use on the Nexys 4 DDR Artix-7 FPGA. Tramelblaze written in Verilog and is comprised of multiple functional blocks; the first being general-purpose registers. There are 16, 16-bit wide registers. There are three memory blocks within the TramelBlaze architecture: scratch RAM, stack RAM, and instruction ROM. The instruction ROM is 4096x16 and it provides a storage for instructions for the assembler and is loaded with the COE file compiled from the Trambler python script. The scratchpad ram is 512x16 and it stores copies of the contents to registers and fetches copies content of the RAM to registers; the stack RAM is 128x16 and its purpose is to track return address for subroutines and interrupt handling. For Project 1, the assembly code is modified according to Tramel's youtube video on the TramelBlaze. When an interrupt occurs, the processor will complete the instructions, fetch the instructions at 0x0FFE and then execute it.

Many modules are re-used from Project 1 of 361: aiso, debounce, ticker, ped(positive edge detect), and display controller. Two new modules are the RS flop and a loadable register. The RS flop intakes two 1-bit wide R and S values and they determine which value to pass into Q. When both are 0's, Q retains its value; when S is 1 and R is 0, Q gets 1, when S is 0 and R is 1 then Q gets 0; . Loadable register accepts in a load signal and that 1-bit wide signal allows Q to get inputted data or retain the current data. The outputted data then is displayed on the seven-segment display. For the demonstration, when the switch 0 is up, pressing the debounce button(P18) increments and the 7-segment displays a 16-bit wide number counting upwards. When the switch is down, pressing the debounce button decrements the value displayed.


top_level:1
debounce
bounce0
posedge_detect
ped0
rs_flop
flop0
tramelblaze_top
tb0
ld_reg
reg0
display_controller
disp0
clk
db_but
sw_uhdl
reset_but
clk
in
reset
p_out
clk
in
reset
ped
clk
r
reset
s
Q
IN_PORT(15:0)
CLK
INTERRUPT
RESET
OUT_PORT(15:0)
PORT_ID(15:0)
INTERRUPT_ACK
READ_STROBE
WRITE_STROBE
D(15:0)
clk
ld
reset
Q(15:0)
addr(15:0)
D_out(15:0)
clk
reset
anode(7:0)
a
b
c
d
e
f
g
anode(7:0)
a
b
c
d
e
f
g
and4b3
I3
I2
I1
I0
O
id_out[15]_GND_1_o_equal_1_o<15>1
id_out[15]_GND_1_o_equal_1_o<15>_imp
id_out(1)
id_out(2)
id_out(3)
id_out(4)
id_out(5)
id_out(6)
id_out(7)
id_out(8)
id_out(15)_GND_1_o_equal_1_o(15)
id_out(15)_GND_1_o_equal_1_o(15)1
id_out(15)_GND_1_o_equal_1_o
id_out[15]_GND_1_o_equal_1_o<15>_imp
and2
I1
I0
O
write_led1
AISO
clk
reset
reset_sync
reset_sync0
and4b4
I3
I2
I1
I0
O
id_out[15]_GND_1_o_equal_1_o<15>2
top_level

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: top_level.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: This is the top level module to the counter that utilizes the
//          tramelblze for its functionality. For project 1, we are using the
//          tramel blaze to instantiate a counter. When sw is up, the counter
//          increments. When the switch is down, the counter decrements.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
/////////////////////////////////////////////////////////////////////////////////

module top_level(clk, reset_but, db_but, anode, sw_uhdl,
                 a, b, c, d, e, f, g);

   // Input and Output declarations
   input        clk, reset_but, db_but, sw_uhdl;
   output [7:0] anode;
   output       a, b, c, d, e, f, g;

   // Wires declarations to interconnect the instantiated modules
   wire        reset,  pulse,  ped, INT_ACK, rs_out,write_strobe,write_led;
   wire [15:0] tb_out, id_out, load_out;


   assign write_led = (id_out == 16'h0001) && write_strobe;

   // Instantiate Asynchronous In Synchrnous Out module
   // to synchronize the release of reset
   AISO     reset_sync0 (.clk(clk),
                         .reset(reset_but),
                         .reset_sync(reset));

   // Instantiate debounce module to generate a stable
   // signal. Any transition less than 20ms is ignored
   debounce bounce0      (.clk(clk),
                          .reset(reset),
                          .in(db_but),
                          .p_out(pulse));

   // Instantiate posedge_detect module to detect the
   // positive edge of the clock
   posedge_detect ped0   (.clk(clk),
                          .reset(reset),
                          .in(pulse),
                          .ped(ped));

```

```
    // Instantiate the rs_flop to allow Q outputs to be
    // either 1 or 0
    rs_flop  flop0        (.clk(clk),
                           .reset(reset),
                           .s(ped),
                           .r(INT_ACK),
                           .Q(rs_out));

    // Instanatiate the TramelBlaze as a processor
    tramelblaze_top  tb0     (.CLK(clk),
                              .RESET(reset),
                              .IN_PORT({15'b0,sw_uhdl}),
                              .INTERRUPT(rs_out),
                              .INTERRUPT_ACK(INT_ACK),
                              .READ_STROBE(),
                              .OUT_PORT(tb_out),
                              .PORT_ID(id_out),
                              .WRITE_STROBE(write_strobe));

    // Instantiate the loadable registers to load data
    // to the output
    ld_reg         reg0      (.clk(clk),
                              .reset(reset),
                              .ld(write_led),
                              .D(tb_out),
                              .Q(load_out));

    // Instantiate the display controller module
    // to display counter Q onto the 7seg display
    display_controller disp0 (.clk(clk),
                              .reset(reset),
                              .addr(16'b0),
                              .D_out(load_out),
                              .anode(anode),
                              .a(a), .b(b), .c(c), .d(d),
                              .e(e), .f(f), .g(g));

endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// This document contains information proprietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: AISO.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: This is an Asynchronous In Synchronous Out module. The purpose of
//          this module is to synchronize the release of reset. Once the reset button
//          is released, reset is turned off synchronously.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
////////////////////////////////////////////////////////////////////////////////

module AISO ( clk , reset , reset_sync ) ;
   // Input declarations
   input      clk       , reset;

   // Output declaration
   output   wire  reset_sync;

   // Wire declarations to interconnect
   reg Q1 , Q2 ;

   // Sequential block to instantiate two d-flops
   // As soon as reset go to 1, Q goes to 0, and
   // get inverted to output a 1
   always @ (posedge clk, posedge reset)
      if (reset)
         {Q1, Q2} <= 2'b0;
      else
         {Q1, Q2} <= {1'b1, Q1};

   // Assign statement to invert the output to be a 1
   assign reset_sync = ~Q2;

endmodule //end of AISO

```

```verilog
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: debounce.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: The debounce module is a FSM (finite state machine) that waits for
//          the input as well as a tick module instantiated within
//          itself to determine its next state and output. This module's
//          purpose is to stablize incoming signals.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
///////////////////////////////////////////////////////////////////////////////

module debounce( clk , reset , in , p_out ) ;

   // Input declarations
   input  clk, reset , in ;

   // Output declaration
   output reg p_out ;

   // Register declarations for next state and present state
   reg [2:0]  n_state , p_state ;
   reg        n_out ;

   // Wire declaration for tick which is a determine factor
   // for the FSM
   wire       tick  ;


   // Instantiation of the ticker to generate a 10ms pulse
   ticker tick0 (.clk(clk),
                 .reset(reset),
                 .tick(tick));

   // Sequential block for state register. When reset is on,
   // present state and output is 0. When it's off, present
   // state and outputnget next state and output
   always @( posedge clk , posedge reset )
      if (reset)
         {p_state , p_out} <= {3'b000, 1'b0};

      else
         {p_state , p_out} <= {n_state, n_out};

   // Next state and Output logic block. A case statement based on
   // the present state, in, and tick to determine where the FSM
   // should go for its next state and output.
```

```
    always @ (*)
       begin
       casex({ p_state , in , tick })

          // First four states of the FSM. The FSM moves forward when
          // input is 1 and tick is 1. When input is 0, it will go back
          // to state 0. When input is 1 and tick is 0, it will loop to
          // itself
          {3'b000 , 1'b1 , 1'bx} : {n_state, n_out} = {3'b001, 1'b0};
          {3'b001 , 1'b1 , 1'b1} : {n_state, n_out} = {3'b010, 1'b0};
          {3'b010 , 1'b1 , 1'b1} : {n_state, n_out} = {3'b011, 1'b0};
          {3'b011 , 1'b1 , 1'b1} : {n_state, n_out} = {3'b100, 1'b0};

          // Last four states of the FSM. The FSM moves forward when
          // input is 0 and tick is 1, When the input is 1, it will
          // go back to state 4. When input is 0 and tick is 0, it will
          // loop back to itself
          {3'b100 , 1'b0 , 1'bx} : {n_state, n_out} = {3'b101, 1'b1};
          {3'b101 , 1'b0 , 1'b1} : {n_state, n_out} = {3'b110, 1'b1};
          {3'b110 , 1'b0 , 1'b1} : {n_state, n_out} = {3'b111, 1'b1};
          {3'b111 , 1'b0 , 1'b1} : {n_state, n_out} = {3'b000, 1'b1};

          // Default cases
          {3'b0xx , 1'b0 , 1'bx} : {n_state, n_out} = {3'b000, 1'b0};
          {3'b1xx , 1'b1 , 1'bx} : {n_state, n_out} = {3'b100, 1'b1};
          default: {n_state, n_out} = {p_state, p_out};

          endcase
       end

endmodule // end of debounce

```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: ticker.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: The ticker intakes in the default clock frequency and generate a
//          10 ms clock.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
////////////////////////////////////////////////////////////////////////////////
module ticker( clk , reset , tick ) ;
   // Input declarations
   input       clk   , reset ;

   // Output declarations
   output      tick  ;

   // Register to hold values for D and count
   reg  [19:0] count , D ;

   // Tick goes high if count counts up to 999,999 in decimal
   assign tick = (count == 20'd999_999);

   // If tick goes high, reset D. If tick is not yet high,
   // continue to increment D
   always @ (*)
      if (tick)
         D = 20'b0;
      else
         D = count + 20'b1;

   // If reset goes high, count goes 0. Else count gets D for
   // incrementing
   always @ (posedge clk, posedge reset)
      if (reset)
         count <= 20'b0;
      else
         count <= D;

endmodule // end of ticker

```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: posedge_detect.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: The positive edge detect module is designed to detect a positive edge
//          input and then returns a one-shot pulse output. If the input is HIGH
//          for first and second clock period, it will output a high one
//          clock period.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
//////////////////////////////////////////////////////////////////////////////////
module posedge_detect(clk , reset , in , ped ) ;

   // Input declarations
   input  clk , reset ;
   input  in  ;

   // Output declarations
   output wire ped ;

   // Reg declarations
   reg Q1, Q2;

   // if reset asserted clear Q1/2 registers
   always @ (posedge clk, posedge reset)
      if(reset)
         {Q1, Q2} <= 2'b0;
      else
         {Q1, Q2} <= {in, Q1};


   assign ped = Q1 & ~Q2;

endmodule // end of positive edge detect

```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: rs_flop.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: The RS flop module has R and S as inputs to determine the Q output.
//          Different values for R and S allow the flop to output different
//          values for Q. When both signals are on, a don't-care is used so it
//          doesn't matter what value is passed to Q.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
////////////////////////////////////////////////////////////////////////////////

module rs_flop(clk, reset, r, s, Q);

   // Input and output declarations
   input      clk, reset, r,s;
   output reg Q;

   // Sequential block to determine value for Q output
   always @ (posedge clk, posedge reset)
      if(reset)
         Q <= 1'b0;
      else begin

      // Case statement for S and R inputs
         casex({s,r})
            2'b00 : Q <= Q;
            2'b01 : Q <= 1'b0;
            2'b10 : Q <= 1'b1;
            2'b11 : Q <= 1'bx;
            default: Q <= 1'b0;
         endcase
      end
endmodule // end of R_S module
```

```verilog
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: ld_reg.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: The loadable register module produces a one-bit load signal. It is a
//          high-active signal. When asserted, D datat gets loaded into Q. When
//          deasserted, Q retains the same data.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
////////////////////////////////////////////////////////////////////////////////
module ld_reg(clk, reset, D, Q, ld);

   // Input and Output declarations
   input             clk, reset, ld;
   input      [15:0] D;
   output reg [15:0] Q;

   // Sequential block for to read the load signal
   // When load is active, Q gets D, or else Q gets Q
   always @(posedge clk, posedge reset)
      if(reset)
         Q <= 16'b0;
      else begin
         if(ld)
            Q <= D;
         else
            Q <= Q;
      end
endmodule

```

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: display_controller.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: This is the display controller module used to tie all the features of
//          the display controller in Lab 4 together. This module implements and
//          instantiates the pixel_controller, ad_mux(8-1-Mux), and
//          Hex_to_7seg module.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
//////////////////////////////////////////////////////////////////////////////

module display_controller(clk, reset,addr, D_out, anode, a, b, c, d, e, f, g);

    // Input declarations
    input         clk , reset ;
    input   [15:0] addr ;
    input   [15:0] D_out ;

    // Output declarations
    output [7:0] anode;
    output        a , b , c , d , e , f , g ;

    // Temp wires to pass binary data from module to modules
    wire    [2:0] S;
    wire    [3:0] mux_wire;

   // Instantiate Pixel Controller
   // A Finite State Machine(FSM) counting up states
   // and outputs a select variable(S) that sends
   // 3-bits of data to the 8-1 mux module to perform data selection on
   // which inputs are to be selected in order to be represented on the display
    pixel_controller controller0 (.clk(clk) ,
                                  .reset(reset) ,
                                  .S(S) ,
                                  .anode(anode)) ;

    // Instantiates an address mux that sends 4-bit value to the 7 segment
    // module. This 8-1 mux sends a nibble(4-bits) to the
    // Hex_to_7seg module to choose the correct segments to display on the board
    ad_mux           mux0        (.addr(addr) ,
                                  .S(S) ,
                                  .D_out(D_out) ,
                                  .ad_out(mux_wire)) ;


    // Instantiate the hex to 7 segment to display the values onto the board
    hex_to_7seg      hex7        (.hex(mux_wire),
                                  .a(a) , .b(b) , .c(c) , .d(d) ,
                                  .e(e) , .f(f) , .g(g));

endmodule // end of display controller module
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: pixel_controller.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: This module is a FSM which has 3 parts:
//          the next state output, state register, and the output
//          combo. Since this is an Moore FSM, it will go
//          from S0 to S7 with its corresponding outputs for anode and S (select).
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
//////////////////////////////////////////////////////////////////////////////

module pixel_controller(clk, reset, S, anode );

        // Inputs, outputs, and register declarations
        input             clk , reset ;
        output reg [7:0] anode;

        // S value selects a value which will be assign to Y
        output reg [2:0] S;

        // Present state and next state
        reg        [2:0] pState, nState;

        // Clock wire to divide down the clock
         wire clk_out;

        // Instantiate pixel clock to divide the default clock of 100MHz to 480Hz
        // The reason for using a divided clock is to refresh the 7Segments on the
        // FPGA board
        pix_ticker        clk0       (.clk(clk),
                                      .reset(reset),
                                      .tick(clk_out));

        /********NEXT STATE COMBINATIONAL LOGIC********
        * This FSM sets a new state based on what the
        * present state is.
        ***********************************************/
        always@ (pState) begin
            case ({pState,clk_out})
                4'b000_0:  nState = 3'b000;
                4'b000_1:  nState = 3'b001;

                4'b001_0:  nState = 3'b001;
                4'b001_1:  nState = 3'b010;

                4'b010_0:  nState = 3'b010;
```

```
                 4'b010_1:  nState = 3'b011;

                 4'b011_0:  nState = 3'b011;
                 4'b011_1:  nState = 3'b100;

                 4'b100_0:  nState = 3'b100;
                 4'b100_1:  nState = 3'b101;

                 4'b101_0:  nState = 3'b101;
                 4'b101_1:  nState = 3'b110;

                 4'b110_0:  nState = 3'b110;
                 4'b110_1:  nState = 3'b111;

                 4'b111_0:  nState = 3'b111;
                 4'b111_1:  nState = 3'b000;
                 default: nState = 3'b000; // default next state is 0
             endcase // end of case statement

         end // end of next state combo block

         /******************STATE REGISTER******************
         * The state register is a simple D_ff. When the reset
         * is 1, the present state returns to its initial state.
         * Otherwise, present state is set equal to next state.
         *****************************************************/
         always@(posedge clk or posedge reset) begin
             if (reset == 1'b1)
                 pState <= 3'b000;
             else
                 pState <= nState;

         end // end of state register block

        /************OUTPUT COMBINATIONAL LOGIC************
         * When the present state is at a specific state the
         * anodes and S will have an output. anode has 8 bits
         * and S has 3 bits which is totalled up to be 11 bits.
         **************************************************/
         always@ (pState) begin
             case (pState)
                 3'b000:  {anode, S} = 11'b11111110_000; // anode[0]
                 3'b001:  {anode, S} = 11'b11111101_001; // anode[1]
                 3'b010:  {anode, S} = 11'b11111011_010; // anode[2]
                 3'b011:  {anode, S} = 11'b11110111_011; // anode[3]
                 3'b100:  {anode, S} = 11'b11101111_100; // anode[4]
                 3'b101:  {anode, S} = 11'b11011111_101; // anode[5]
                 3'b110:  {anode, S} = 11'b10111111_110; // anode[6]
                 3'b111:  {anode, S} = 11'b01111111_111; // anode[7]
                 default: {anode, S} = 11'b00000000_000; // default all anode on, S is 0
             endcase // end of case statement

         end // end of output combo logic

     endmodule // end of pixel_controller
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: ticker.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: The ticker intakes in the default clock frequency and generate a
// 480 Hz clock
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
//////////////////////////////////////////////////////////////////////////////
module pix_ticker(clk, reset, tick);

   // Input declarations
   input clk, reset;

   // Output declarations
   output wire tick;

   // Reg to hold values for count and D
   reg [17:0] count, D;

   // Tick goes high if count counts up to 208_334 in decimal
   assign tick = count == 18'd208334;

   // If tick goes high, reset D. If tick is not yet high,
   // continue to increment D
   always @(*)
      if (tick)
         D = 18'b0;
      else
         D = count + 18'b1;

   // If reset goes high, count goes 0. Else count gets D for
   // increment
   always @ ( posedge clk, posedge reset)
      if (reset)
         count <= 18'b0;
      else
         count <= D;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: ad_mux.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: This mux will select what D_out will be, based
//          on the select signal which comes from the pixel controller
//          to display a hex value from 0 to F. The S is controlled
//          by what anode is active at the time.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
////////////////////////////////////////////////////////////////////////////////

module ad_mux(addr, S, D_out, ad_out);

   // Select chooses which input will be assigned to the output
   input      [2:0]  S;

   // Address input holds 4 sets of 4 bits
   input      [15:0]  addr;

   // The 16-bit input has 4 bits for every data
   input      [15:0] D_out;

   // The output carries the addr based on the value of S
   output reg [3:0]  ad_out;

   // Always block is executed based on any changes from S, D_out
   // and/or addr. The case statement is solely based on the value
   // of S.
   always @ (S, D_out, addr)
      begin
         case (S)

            // anode[0], lowest nibble of data out of the RAM
            3'b000: ad_out = D_out [3:0] ;

            // anode[1], the next nibble of data out of RAM
            3'b001: ad_out = D_out [7:4] ;

            // anode[2], the next nibggle of data out of RAM
            3'b010: ad_out = D_out [11:8] ;

            // anode[3], the highest nibble of data out of RAM
            3'b011: ad_out = D_out [15:12] ;

            // anode[4], the lowest nibble of the address location
            3'b100: ad_out = addr  [3:0] ;
```

```

             // anode[5], the next nibble of the address location
             3'b101: ad_out = addr  [7:4] ;

             // anode[6], the next nibble of the address location
             3'b110: ad_out = addr  [11:8];

             // anode [7], the higest nibble of the address location
             3'b111: ad_out = addr  [15:12];

             // default case for hex to be 0
             default: ad_out = 4'h0;

          endcase // end of case statement

       end // end of always block

endmodule // end of 8-1 mux
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: hex_to_7seg.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: This hex_to_7Seg module essentially takes in a four-bit(nibble) input,
//          converts the input information by selecting the correct corresponding
//          segments (a-g) on the seven segment LED display in order to represent
//          that four-bit input integer. This method was completed by using
//          case-statements for all of the appropriate four-bit input options
//          from 0-F and assigning the correct values of a-g to later display.
//          A 0 is on and a 1 is off for the LED segments.
//
// In submitting this file for class work at CSULB
// I am confirming that this is my work and the work of no one else.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
/////////////////////////////////////////////////////////////////////////////////

module hex_to_7seg(hex, a, b, c, d, e, f, g);

   // Input and output declarations
   input       [3:0] hex ;
   output reg        a , b , c , d , e , f , g ;

   always @ (hex)

      /************************************************
      * The four-bit hex input from 0-F is specified
      * by 4'h0 to 4'hF. The case statement is used to
      * decode each hex value in order to choose the
      * correct LED is on for that specific hex value.
      ************************************************/
      case(hex)

          // 7-segment display for 0-3
          4'h0: {a, b, c, d, e, f, g} = 7'b0000001; // 0
          4'h1: {a, b, c, d, e, f, g} = 7'b1001111; // 1
          4'h2: {a, b, c, d, e, f, g} = 7'b0010010; // 2
          4'h3: {a, b, c, d, e, f, g} = 7'b0000110; // 3

          // 7-segment display for 4-7
          4'h4: {a, b, c, d, e, f, g} = 7'b1001100; // 4
          4'h5: {a, b, c, d, e, f, g} = 7'b0100100; // 5
          4'h6: {a, b, c, d, e, f, g} = 7'b0100000; // 6
          4'h7: {a, b, c, d, e, f, g} = 7'b0001111; // 7

          // 7-segment display for 8-B
          4'h8: {a, b, c, d, e, f, g} = 7'b0000000; // 8
          4'h9: {a, b, c, d, e, f, g} = 7'b0000100; // 9
          4'hA: {a, b, c, d, e, f, g} = 7'b0001000; // A
```

```
            4'hB: {a, b, c, d, e, f, g} = 7'b1100000; // B

            // 7-segment display for C-F
            4'hC: {a, b, c, d, e, f, g} = 7'b0110001; // C
            4'hD: {a, b, c, d, e, f, g} = 7'b1000010; // D
            4'hE: {a, b, c, d, e, f, g} = 7'b0110000; // E
            4'hF: {a, b, c, d, e, f, g} = 7'b0111000; // F

            // default display if other cases do not apply
            default: {a, b, c, d, e, f, g} = 7'b0110111;

        endcase // endcase

endmodule // end of hex_to7seg module
```

```verilog
`timescale 1ns / 1ps

module tramelblaze_top_tb;

   // Inputs
   reg CLK;
   reg RESET;
   reg [15:0] IN_PORT;
   reg INTERRUPT;

   // Outputs
   wire [15:0] OUT_PORT;
   wire [15:0] PORT_ID;
   wire READ_STROBE;
   wire WRITE_STROBE;
   wire INTERRUPT_ACK;

   // Instantiate the Unit Under Test (UUT)
   tramelblaze_top uut (
      .CLK(CLK),
      .RESET(RESET),
      .IN_PORT(IN_PORT),
      .INTERRUPT(INTERRUPT),
      .OUT_PORT(OUT_PORT),
      .PORT_ID(PORT_ID),
      .READ_STROBE(READ_STROBE),
      .WRITE_STROBE(WRITE_STROBE),
      .INTERRUPT_ACK(INTERRUPT_ACK)
   );

   always #5 CLK = ~CLK;

   initial begin
      // Initialize Inputs
      CLK = 0;
      RESET = 1;
      IN_PORT = 0;
      INTERRUPT = 0;

      // Wait 100 ns for global reset to finish
      #100 RESET = 0;
      repeat(10)
         begin
         #100
         @(posedge CLK)
         INTERRUPT = 1;
         @(posedge INTERRUPT_ACK)
         INTERRUPT = 0;
         end

      // Add stimulus here

   end

endmodule

```

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////////////////
// This document contains information prorietary to the CSULB student that created
// the file -  any reuse without adequate approval and documentation is prohibited
//
// File Name: top_level_tb.v
// Project: Lab 1
// Created by <Zachery Takkesh> on <September 23, 2019>
// Copright @ 2019 <Zachery Takkesh>. All rights reserved
//
// Purpose: This is the top level module testbench. It provides the top level module
// with a stimulus; clk, reset, db_but, sw_uhdl and wait for the TramelBlaze
// to do the increment/decrement of data.
//
// In submitting this file for class work at CSULB
// I am confirming that this is Tramel's testbench represented on his instruction
// video.
//
// In submitting this code I acknowledge that plagiarism in student project
// work is subject to dismissal from the class.
////////////////////////////////////////////////////////////////////////////////
module top_level_tb;

   // Inputs
   reg clk;
   reg reset_but;
   reg db_but;
   reg sw_uhdl;

   // Outputs
   wire [7:0] anode;
   wire a;
   wire b;
   wire c;
   wire d;
   wire e;
   wire f;
   wire g;

   // Instantiate the Unit Under Test (UUT)
   top_level uut (
      .clk(clk),
      .reset_but(reset_but),
      .db_but(db_but),
      .anode(anode),
      .sw_uhdl(sw_uhdl),
      .a(a),
      .b(b),
      .c(c),
      .d(d),
      .e(e),
      .f(f),
      .g(g)
   );
   always #5 clk = ~clk;
   initial begin
      // Initialize Inputs
```

```
        clk = 0;
        reset_but = 1;
        db_but = 0;
        sw_uhdl = 0; // This value depends since the TramelBlaze
                 // take quite long to execute an instruction.
                 // Set as 0 to start decrement or 1 to start increment

        // Wait 100 ns for global reset to finish
        #100 reset_but = 0;

        // Wait 1000ns to turn on debounce
        #1000
        db_but  = 1;
        repeat(10) begin // Repeat this 10 times
        db_but = 1;
        #40_000_000 db_but = 0;
        #40_000_000 db_but = 1;

        end

        #1000 // Wait 1000ns to turn db off
        db_but = 0;
        sw_uhdl = 1;
        repeat(10) begin // Repeat this 10 times
        db_but = 1;
        #40_000_000 db_but = 0;
        #40_000_000 db_but = 1;
        end

    end

endmodule

```