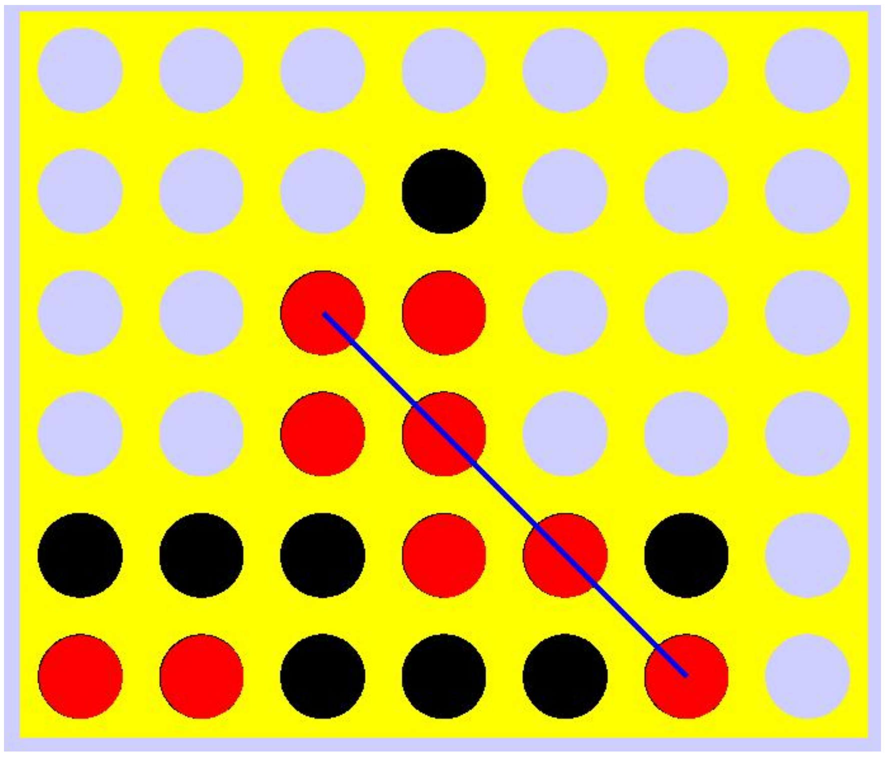
Design of Logic Systems II – Verilog HDL

Final Project Report

Connect Four Game (via VGA)



Muaz Rahman
Jose Hernandez

ECE 3135 – Design of Logic Systems II

25 November 2013

I. Evaluation of Submitted Proposal

Our submitted proposal back on 30 Sept 2013 was the following:

"In this project, we will build a videogame which will be controlled using the Basys2 FPGA board. The game to be built is called connect-4. The game consists of a 6-by-6 board where two players place circles until one of them has 4 circles connected in a row. The circles can be aligned vertically, horizontally or diagonally. The circles are placed on the board by dropping them to the bottom of the grid and taking turns between the two players. Once one of the players has reached a connection of 4 circles in a row, that player wins the game."

Our finished product came about to be exactly what was submitted on our proposal except we used a 6x7 board rather than a 6x6 one simply because of the fact that the logic of the game would be hindered if we did not resize to 6x7. Besides this minor adjustment, our finished product is the same as the one we proposed.

II. Detailed Description of Project Goal

The goal of this connect-4 game is to basically create an interface wherein there are two players, and they both take turns dropping pins/blocks into a grid. Each player has their own color block that is specific to them. The one who wins is the first player that can get at least four of their blocks connected in a row. This connection can either be aligned vertically, horizontally, or diagonally. This is pretty much the overall goal and structure of this game.

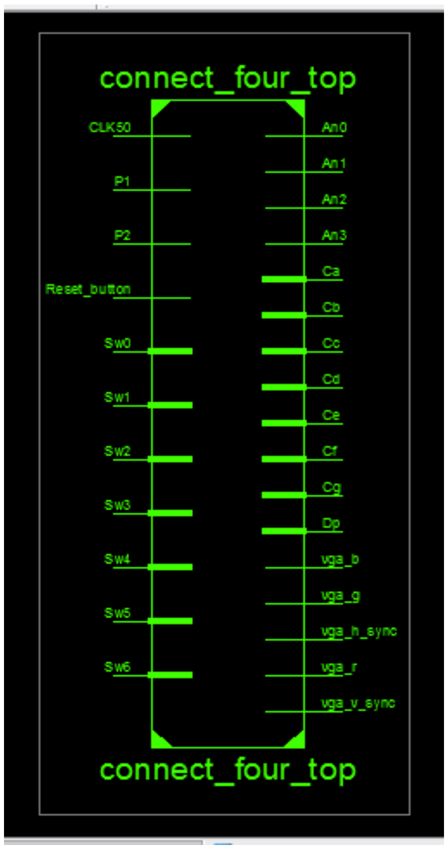
Now in terms of our implementation of this game described above using a Basys2 Board and HDL programming language, we had to create several modules that would implement the necessary functionalities. One of the essential modules that we created was the VGA module. This was one of the most critical aspects simply because this actually made the game visible to our user/audience. Without this visual feature, our game would be rendered pointless. The visual aspects would include the game board itself, and the blocks for each of the two players.

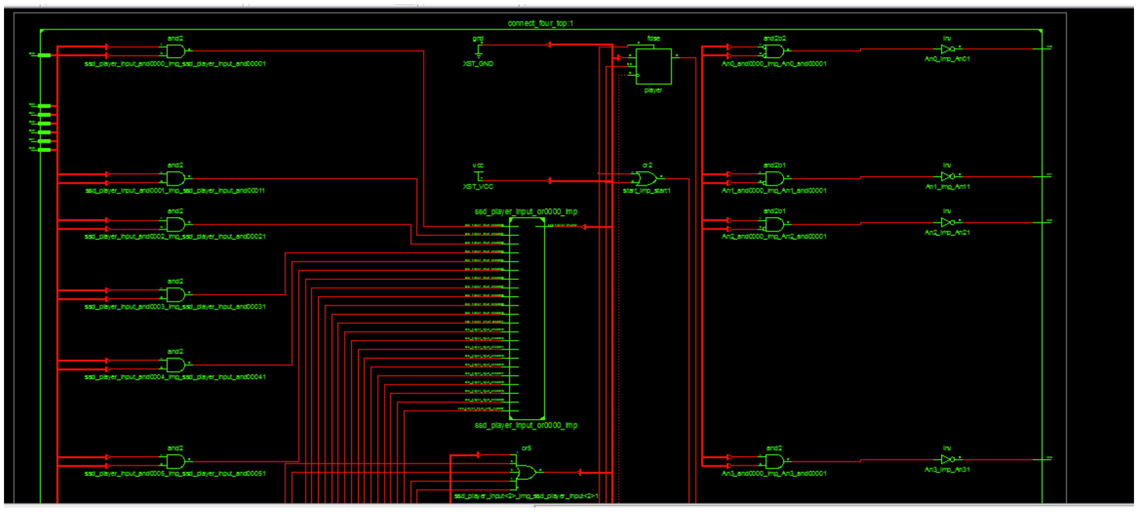
Aside from the VGA module, the next important module was trying to implement the logic behind this game, and essentially we had to create an extremely detailed state machine that would take us through every possible scenario that can occur during this game. We had to make sure each player has their successive turns and had to account for their every possible move on the game itself. Using this state machine module, we would then also have to generate an output when there is a four way connection between the blocks of the same player, deeming them to be the winner of the game.

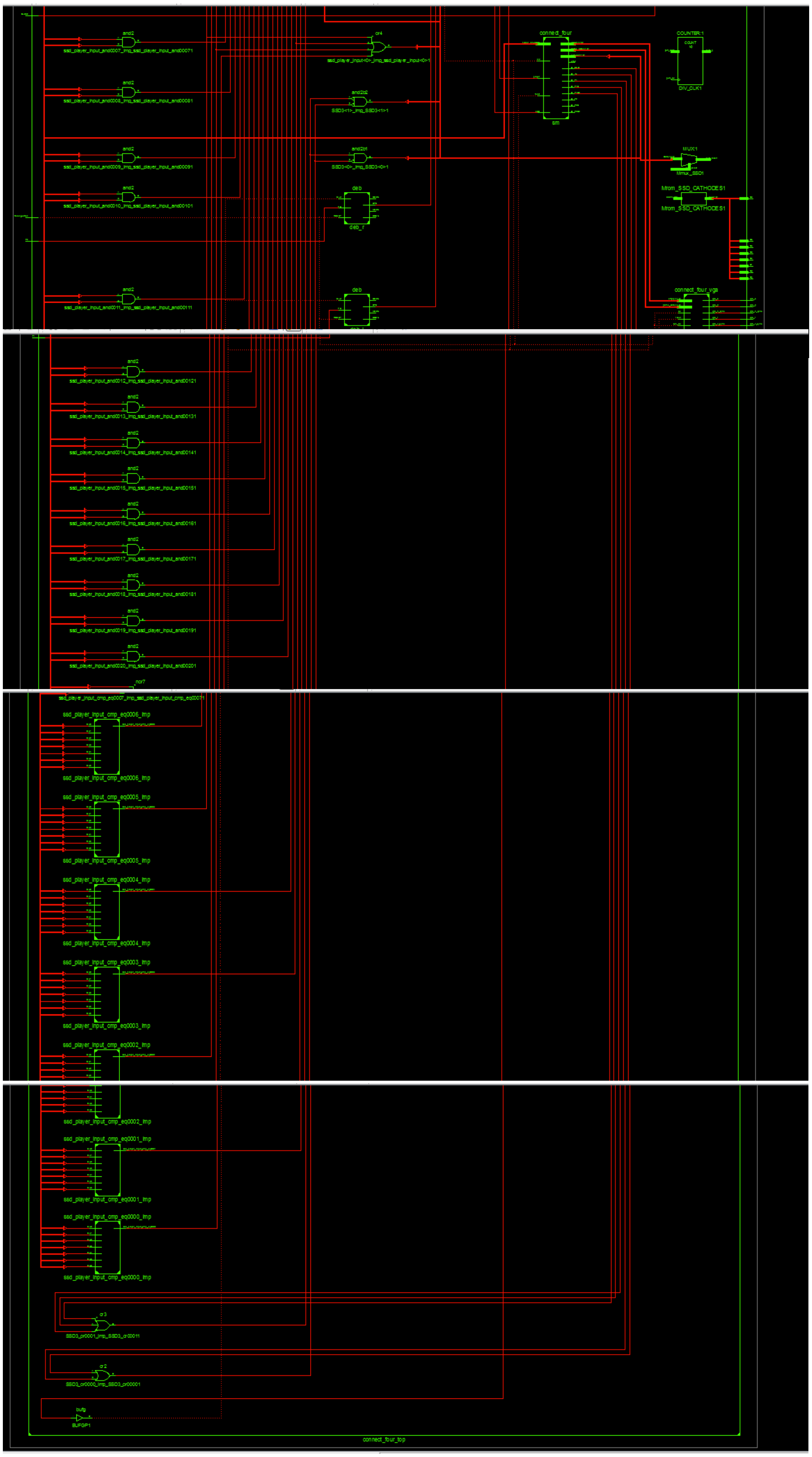
The other modules included our top module where we highlighted the major inputs and outputs of our program, as well as passing along our variables to the other sub-modules. One of the sub-modules included a debouncing module that was for the buttons that we would need to use in order for the user to communicate to the vga module. This module took care of the buttons being pressed and to make sure that the players would not drop twice. On our board itself, we had to implement both the buttons to select block drop, as well as the switches to select where the user would like to drop their block.

III. Design and Implementation

A. High Level Schematics



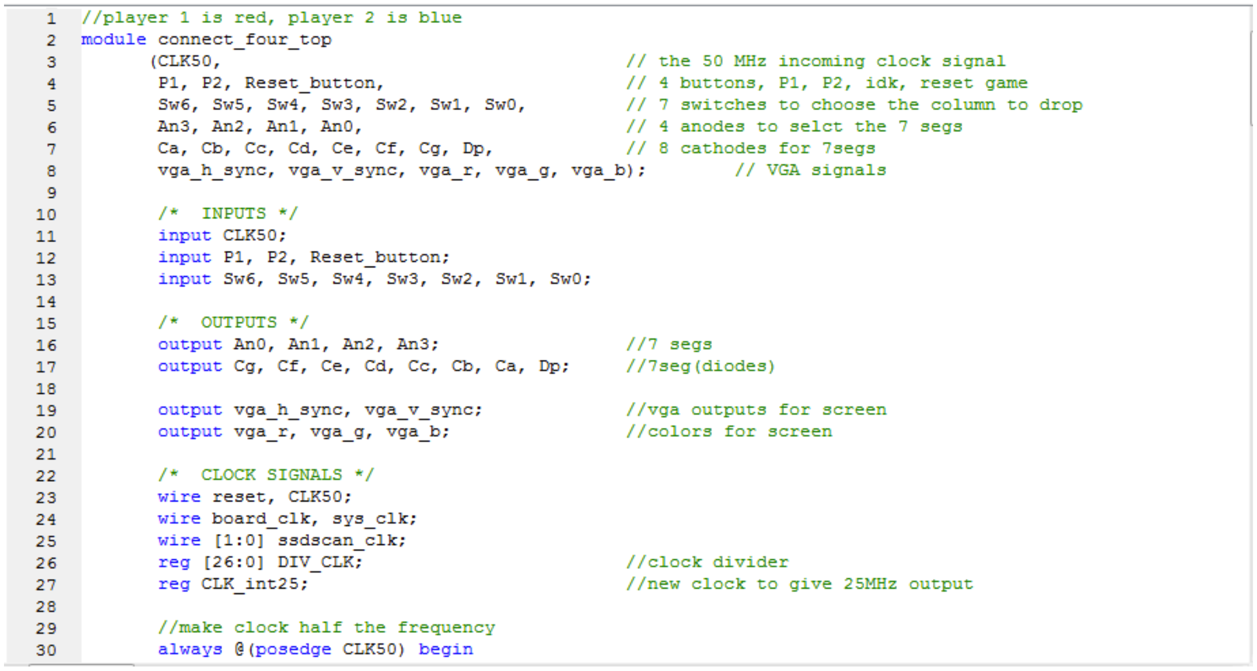




and2
ssd_player_input_and0007_imp_ssd_player_input_and00071
and2
ssd_player_input_and0008_imp_ssd_player_input_and00081
and2
ssd_player_input_and0009_imp_ssd_player_input_and00091
and2
ssd_player_input_and0010_imp_ssd_player_input_and00101
and2
ssd_player_input_and0011_imp_ssd_player_input_and00111
or4
ssd_player_input<0>_imp_ssd_player_input<0>1
and2b2
SSD3<1>_imp_SSD3<1>1
and2b1
SSD3<0>_imp_SSD3<0>1
deb
deb_r
deb
connect_four
sm
COUNTER:1
DIV_CLK1
MUX:1
Mmux_SSD1
Mrom_SSD_CATHODES1
Mrom_SSD_CATHODES1
connect_four_vga
and2
ssd_player_input_and0012_imp_ssd_player_input_and00121
and2
ssd_player_input_and0013_imp_ssd_player_input_and00131
and2
ssd_player_input_and0014_imp_ssd_player_input_and00141
and2
ssd_player_input_and0015_imp_ssd_player_input_and00151
and2
ssd_player_input_and0016_imp_ssd_player_input_and00161
and2
ssd_player_input_and0017_imp_ssd_player_input_and00171
and2
ssd_player_input_and0018_imp_ssd_player_input_and00181
and2
ssd_player_input_and0019_imp_ssd_player_input_and00191
and2
ssd_player_input_and0020_imp_ssd_player_input_and00201
nor7
ssd_player_input_cmp_eq0007_imp_ssd_player_input_cmp_eq00071
ssd_player_input_cmp_eq0006_imp
ssd_player_input_cmp_eq0006_imp
ssd_player_input_cmp_eq0005_imp
ssd_player_input_cmp_eq0005_imp
ssd_player_input_cmp_eq0004_imp
ssd_player_input_cmp_eq0004_imp
ssd_player_input_cmp_eq0003_imp
ssd_player_input_cmp_eq0003_imp
ssd_player_input_cmp_eq0002_imp
ssd_player_input_cmp_eq0002_imp
ssd_player_input_cmp_eq0001_imp
ssd_player_input_cmp_eq0001_imp
ssd_player_input_cmp_eq0000_imp
ssd_player_input_cmp_eq0000_imp
or3
SSD3_or0001_imp_SSD3_or00011
or2
SSD3_or0000_imp_SSD3_or00001
bufg
BUFGP1
connect_four_top

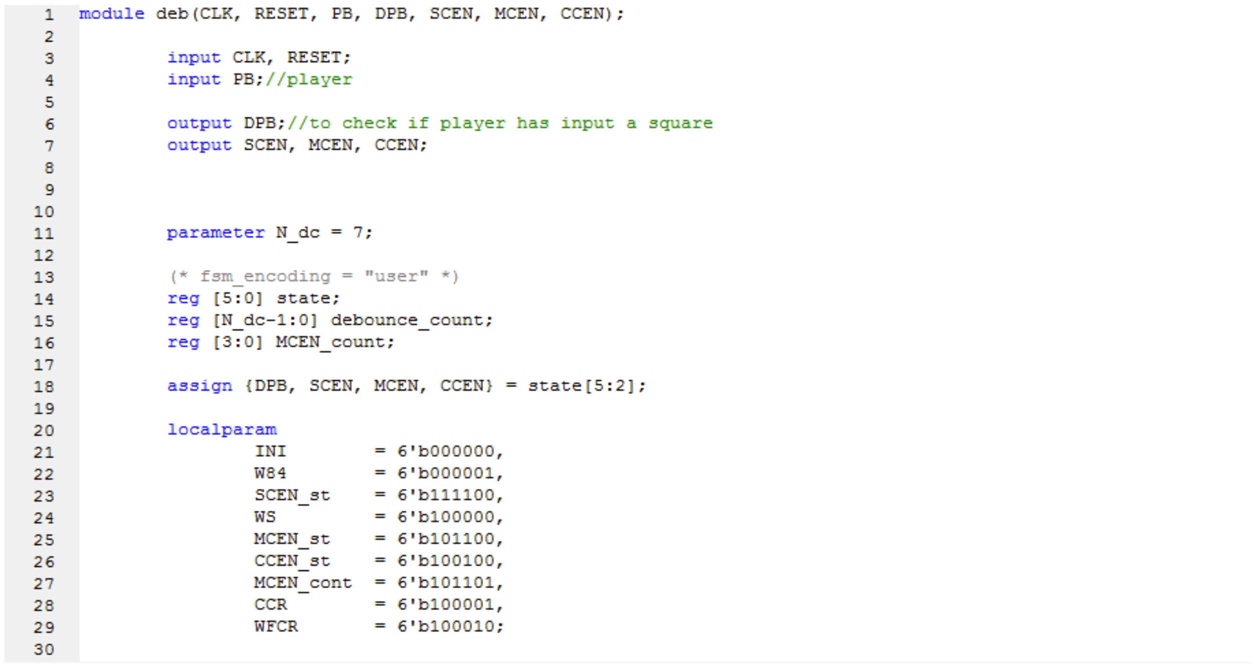
B. *RTL Level Code Snippets*

Top Module:



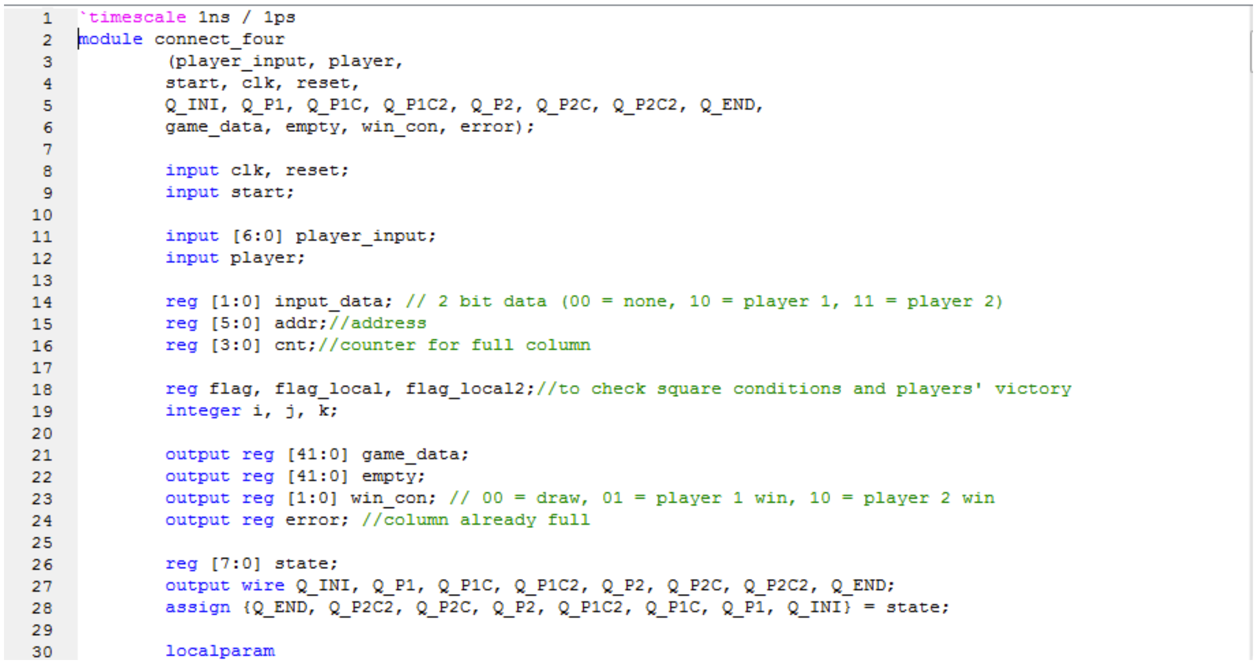
```verilog
//player 1 is red, player 2 is blue
module connect_four_top
      (CLK50,                                     // the 50 MHz incoming clock signal
       P1, P2, Reset_button,                      // 4 buttons, P1, P2, idk, reset game
       Sw6, Sw5, Sw4, Sw3, Sw2, Sw1, Sw0,         // 7 switches to choose the column to drop
       An3, An2, An1, An0,                        // 4 anodes to selct the 7 segs
       Ca, Cb, Cc, Cd, Ce, Cf, Cg, Dp,            // 8 cathodes for 7segs
       vga_h_sync, vga_v_sync, vga_r, vga_g, vga_b);          // VGA signals

       /*  INPUTS */
       input CLK50;
       input P1, P2, Reset_button;
       input Sw6, Sw5, Sw4, Sw3, Sw2, Sw1, Sw0;

       /*  OUTPUTS */
       output An0, An1, An2, An3;                 //7 segs
       output Cg, Cf, Ce, Cd, Cc, Cb, Ca, Dp;     //7seg(diodes)

       output vga_h_sync, vga_v_sync;             //vga outputs for screen
       output vga_r, vga_g, vga_b;                //colors for screen

       /*  CLOCK SIGNALS */
       wire reset, CLK50;
       wire board_clk, sys_clk;
       wire [1:0] ssdscan_clk;
       reg [26:0] DIV_CLK;                        //clock divider
       reg CLK_int25;                             //new clock to give 25MHz output

       //make clock half the frequency
       always @(posedge CLK50) begin
```

Debouncing Module:



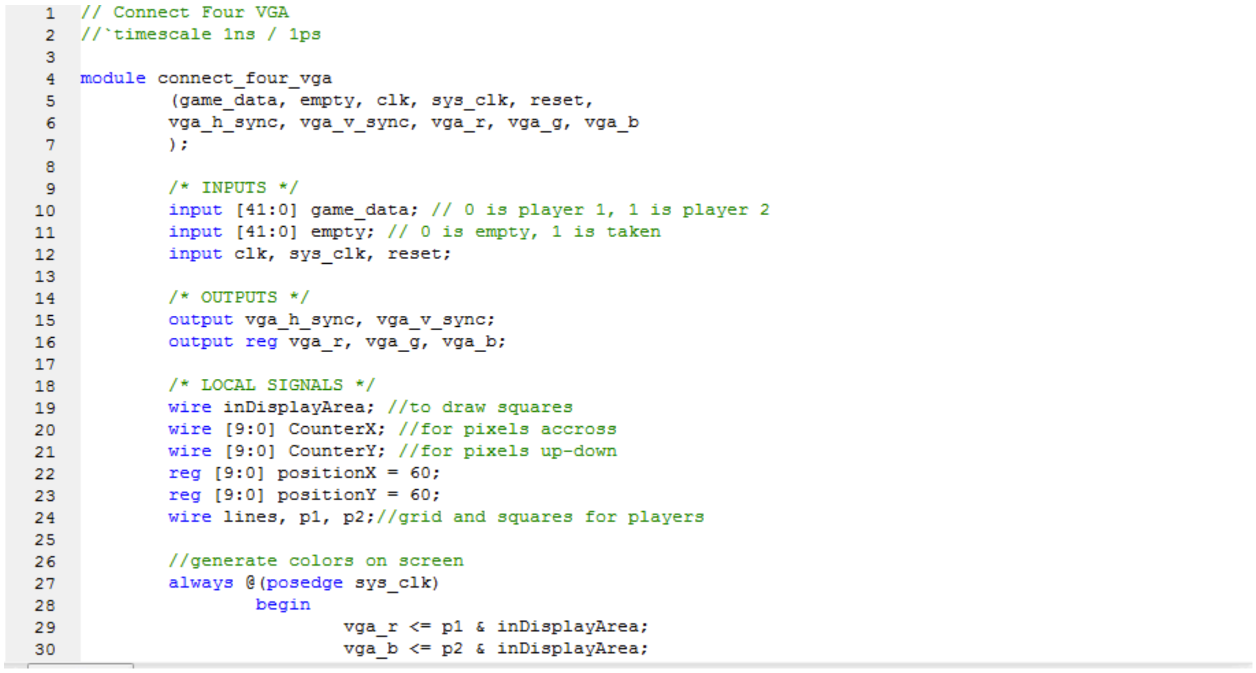
```verilog
module deb(CLK, RESET, PB, DPB, SCEN, MCEN, CCEN);

        input CLK, RESET;
        input PB;//player

        output DPB;//to check if player has input a square
        output SCEN, MCEN, CCEN;



        parameter N_dc = 7;

        (* fsm_encoding = "user" *)
        reg [5:0] state;
        reg [N_dc-1:0] debounce_count;
        reg [3:0] MCEN_count;

        assign {DPB, SCEN, MCEN, CCEN} = state[5:2];

        localparam
                INI        = 6'b000000,
                W84        = 6'b000001,
                SCEN_st    = 6'b111100,
                WS         = 6'b100000,
                MCEN_st    = 6'b101100,
                CCEN_st    = 6'b100100,
                MCEN_cont  = 6'b101101,
                CCR        = 6'b100001,
                WFCR       = 6'b100010;

```

State Machine Module:



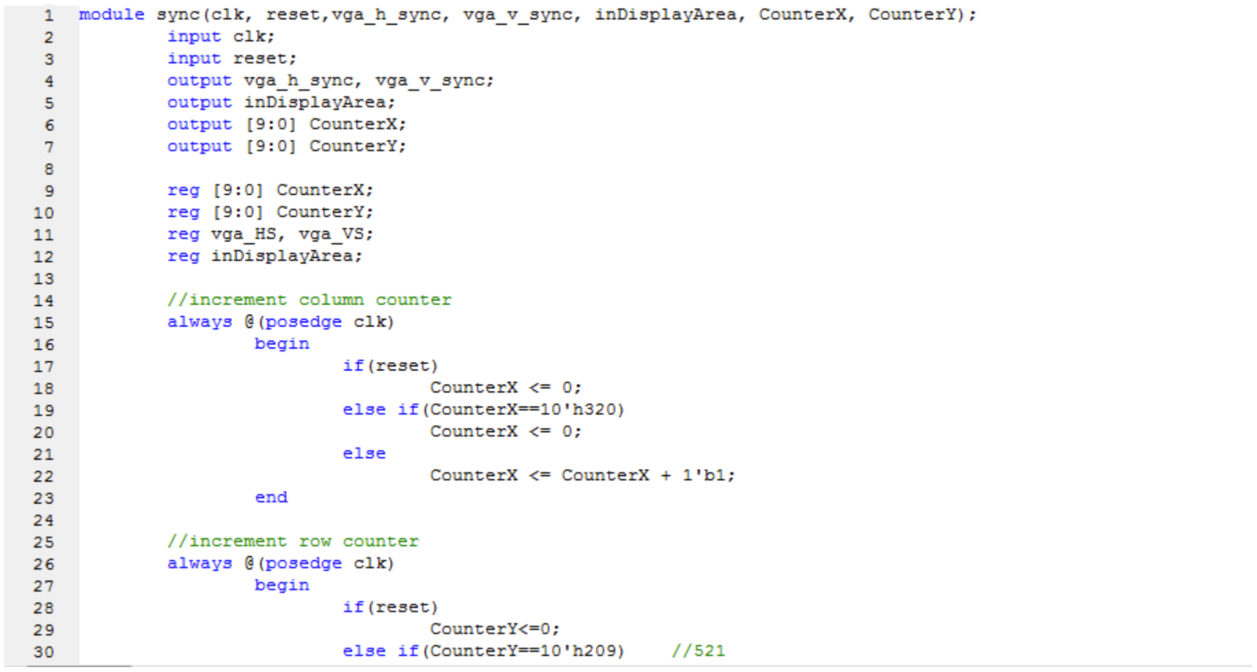
```verilog
`timescale 1ns / 1ps
module connect_four
        (player_input, player,
        start, clk, reset,
        Q_INI, Q_P1, Q_P1C, Q_P1C2, Q_P2, Q_P2C, Q_P2C2, Q_END,
        game_data, empty, win_con, error);

        input clk, reset;
        input start;

        input [6:0] player_input;
        input player;

        reg [1:0] input_data; // 2 bit data (00 = none, 10 = player 1, 11 = player 2)
        reg [5:0] addr;//address
        reg [3:0] cnt;//counter for full column

        reg flag, flag_local, flag_local2;//to check square conditions and players' victory
        integer i, j, k;

        output reg [41:0] game_data;
        output reg [41:0] empty;
        output reg [1:0] win_con; // 00 = draw, 01 = player 1 win, 10 = player 2 win
        output reg error; //column already full

        reg [7:0] state;
        output wire Q_INI, Q_P1, Q_P1C, Q_P1C2, Q_P2, Q_P2C, Q_P2C2, Q_END;
        assign {Q_END, Q_P2C2, Q_P2C, Q_P2, Q_P1C2, Q_P1C, Q_P1, Q_INI} = state;

        localparam
```

VGA Module:



```verilog
// Connect Four VGA
//`timescale 1ns / 1ps

module connect_four_vga
        (game_data, empty, clk, sys_clk, reset,
        vga_h_sync, vga_v_sync, vga_r, vga_g, vga_b
        );

        /* INPUTS */
        input [41:0] game_data; // 0 is player 1, 1 is player 2
        input [41:0] empty; // 0 is empty, 1 is taken
        input clk, sys_clk, reset;

        /* OUTPUTS */
        output vga_h_sync, vga_v_sync;
        output reg vga_r, vga_g, vga_b;

        /* LOCAL SIGNALS */
        wire inDisplayArea; //to draw squares
        wire [9:0] CounterX; //for pixels accross
        wire [9:0] CounterY; //for pixels up-down
        reg [9:0] positionX = 60;
        reg [9:0] positionY = 60;
        wire lines, p1, p2;//grid and squares for players

        //generate colors on screen
        always @(posedge sys_clk)
                begin
                        vga_r <= p1 & inDisplayArea;
                        vga_b <= p2 & inDisplayArea;
```

Sync Module:



```
module sync(clk, reset,vga_h_sync, vga_v_sync, inDisplayArea, CounterX, CounterY);
        input clk;
        input reset;
        output vga_h_sync, vga_v_sync;
        output inDisplayArea;
        output [9:0] CounterX;
        output [9:0] CounterY;

        reg [9:0] CounterX;
        reg [9:0] CounterY;
        reg vga_HS, vga_VS;
        reg inDisplayArea;

        //increment column counter
        always @(posedge clk)
                begin
                        if(reset)
                                CounterX <= 0;
                        else if(CounterX==10'h320)
                                CounterX <= 0;
                        else
                                CounterX <= CounterX + 1'b1;
                end

        //increment row counter
        always @(posedge clk)
                begin
                        if(reset)
                                CounterY<=0;
                        else if(CounterY==10'h209)    //521
```

C. *Explanation of RTL Level Code*

Top Module:

The top module initializes all the necessary major inputs and outputs for this project. This would include aspects such as the buttons, switches, seven segment displays, vga components, the clock, and various others. One of the important tasks this top module does is send out various signals to sub-modules, which basically allows us to divide up the various workloads that would be included in this project. Aside from this , the top module begins by implementing various clock signals, which would allow us to implement these clock signals for a varitey of functions. The purpose of creating different clocks is to use a different clock for each module without having the modules to interfere whit each other. However each clock is just a copy of the main clock with same frequency. The original 50Mhz clock originated from the board was converted to a 25 Mhz clock to be able to sync with our VGA display. We then go on to convert our input signals to wires and create a variety of new wires in order to pass the information to the sub-modules. As stated earlier, the major task of this top module is to take the input signals and assign them to the correct modules to be implemented properly. This module also takes care of displaying the correct player number, column number and winner using the seven segment displays. This is done using the techniques learned during lab. The seven segment displays are control using case statement s where we control which 7-seg and which part of the 7-seg is turned on.

Debouncing Module:

The idea behind this debouncing module is to simply make sure that the boards functionality and push-button inputs do not become misaligned with the vga module. Thus this module has to keep in check what the user has input so far and then re-sync the vga and input button to align both sections properly for the users next input. This module thus has a variety of states that it needs to account for. Each state has a different count that is assigned to it, which is a 6-bit signature. As each of these states increment the next state will be outputted to the function. Each state checks the specific debounce count and generates the necessary state. This makes sure that when the player presses the button, the block is not dropped more than once. Two identical modules were used for debouncing. One was used for player one and another for player two.

State Machine Module:

The state machine module is the most critical module of this project. It keeps track of what is happening and knows what logic is being inputted and outputted to and from the system. Based on these factors, it goes to the next state and does the proper computations to assign the proper values to the necessary variables. The state machine keeps track of the logic of the game and the system as a whole. We start off this module by assigning the various states that will be necessary. These states cover progress from the beginning of the game to the end of the game where the user has won by getting four blocks of their own in a row. In order to keep track of which input is selected by the user where, we also need to select and initialize the 7-bit variable to represent the 7 columns. We then begin our always method which is selected and used based on the positive edge of the clock cycle. It first checks if the user presses the reset button. If this is the case then the board will reset itself and all of its data, and will go to the first state. Moving along, it next begins a case statement with all of the possible states. If we are still in the initial state, it gives it initial values to get the board started and moves on to the next state. The next check will be for which player’s turn at that moment in time. We have two separate sets of states to represent the different possibilities of each player. Essentially the states have the same functionality but they depend on each player. After determing which player's turn it is, we then go on to see which input they have inserted their block into. And depending on this input, we store in a register what the board itself looks like and which spaces have been taken up so far. Every time a player inputs a block our logic checks the entire board again to find the blocks that have been filled and to check if there is any winner. We have three check conditions, one for veritical, one for horizontal, and one for diagonal. We loop through each column and row of the board using for loops, and we check every cell to see what the values are and we store this in memory. If a match of four in a row is found, we assign a flag to go high. At the end, if this flag is still high, we deep that specific player the winner, and we go to our end state. Thus the game is over.

VGA Module:

The purpose of this module is basically to give the correct values for our display monitor and make sure the blocks for each player as well as the grid itself is in correct order and fashion. We have to make sure we assign the correct coordinates for each of these three variables and there after we pass on all of these to our secondary module named "sync" which basically is in charge of v_sync and h_sync functionalities, that will actually display what is necessary. This VGA module is simply in charge of making sure the correct pixels are displayed at the correct place at the correct time. We assign the correct values for Players1 block, which is a Red Square. Players2 block is a Blue Square. And the grid itself is composed of green lines 6x7. We then pass on our necessary values to our sync module, explained in the next section.

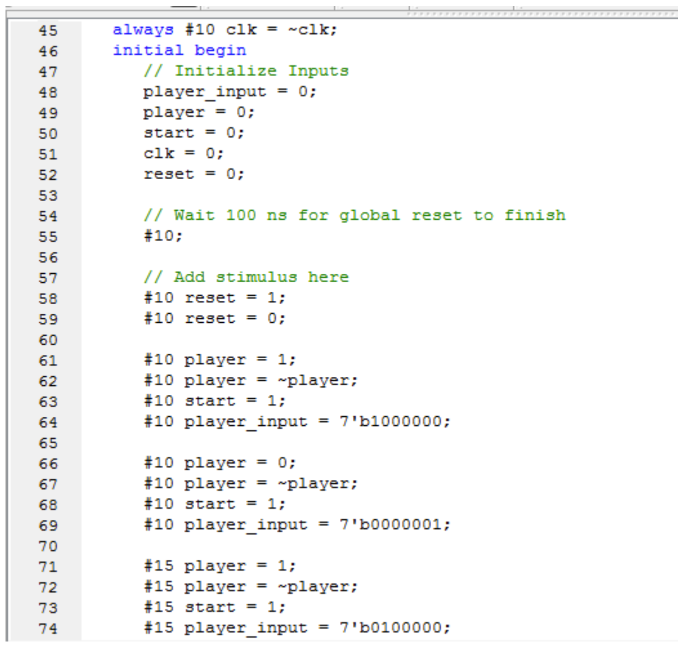
Sync Module:

This sync module is basically what is used to actually display a picture on the screen via the VGA cable. It takes as the major input the h_sync and v_sync and based on the positive clock cycle, we have it sending signals to the monitor. We use two variables CounterX and CounterY to specify which lines and what direction should these two sync variables display. We also have to make sure to increment each row counter on every positive clock edge as well. Using the counter variables we generate synchronization signals for the v_sync and h_sync. We also specify in this module the Display Area itself, and how much of the screen it should cover. This is basically the entire functionality of this module.

IV. Testing Methods

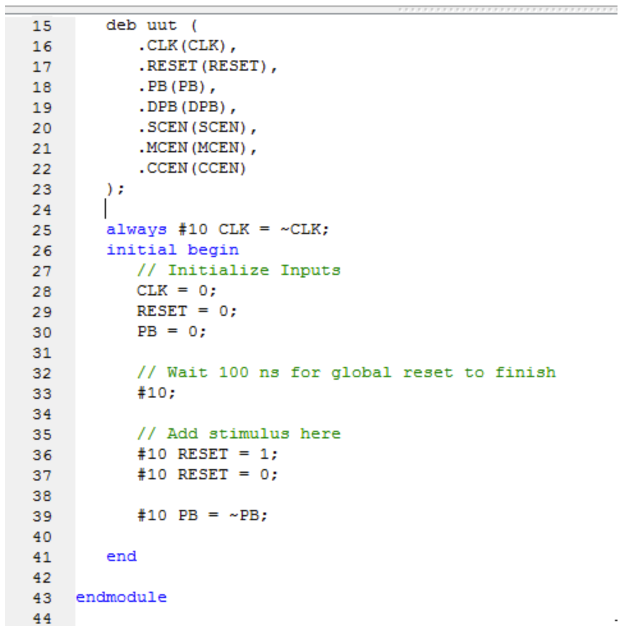
A. Testbench Code File Snippets

State Machine Module:



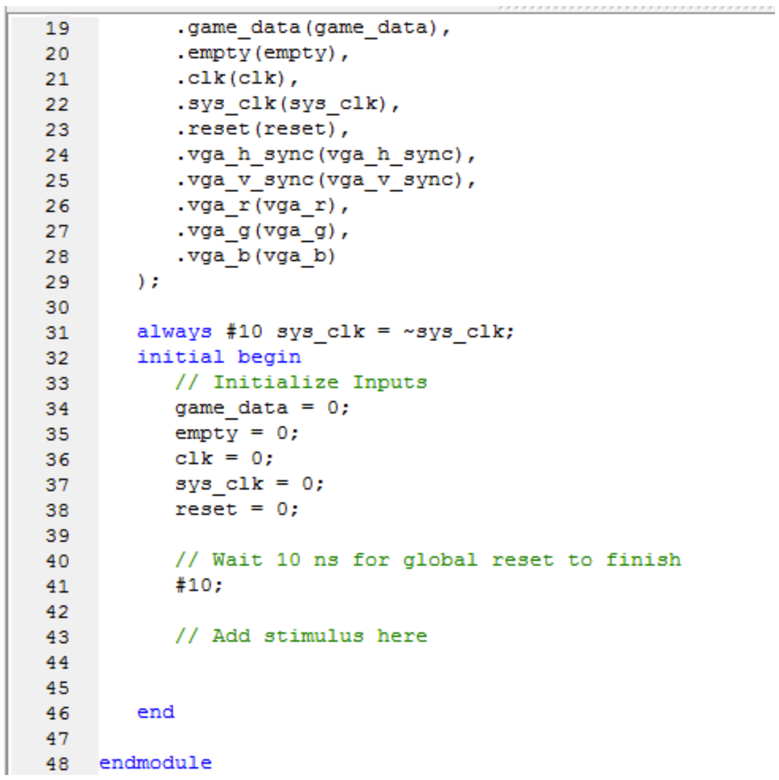
```
always #10 clk = ~clk;
initial begin
   // Initialize Inputs
   player_input = 0;
   player = 0;
   start = 0;
   clk = 0;
   reset = 0;

   // Wait 100 ns for global reset to finish
   #10;

   // Add stimulus here
   #10 reset = 1;
   #10 reset = 0;

   #10 player = 1;
   #10 player = ~player;
   #10 start = 1;
   #10 player_input = 7'b1000000;

   #10 player = 0;
   #10 player = ~player;
   #10 start = 1;
   #10 player_input = 7'b0000001;

   #15 player = 1;
   #15 player = ~player;
   #15 start = 1;
   #15 player_input = 7'b0100000;
```

Debouncing Module:



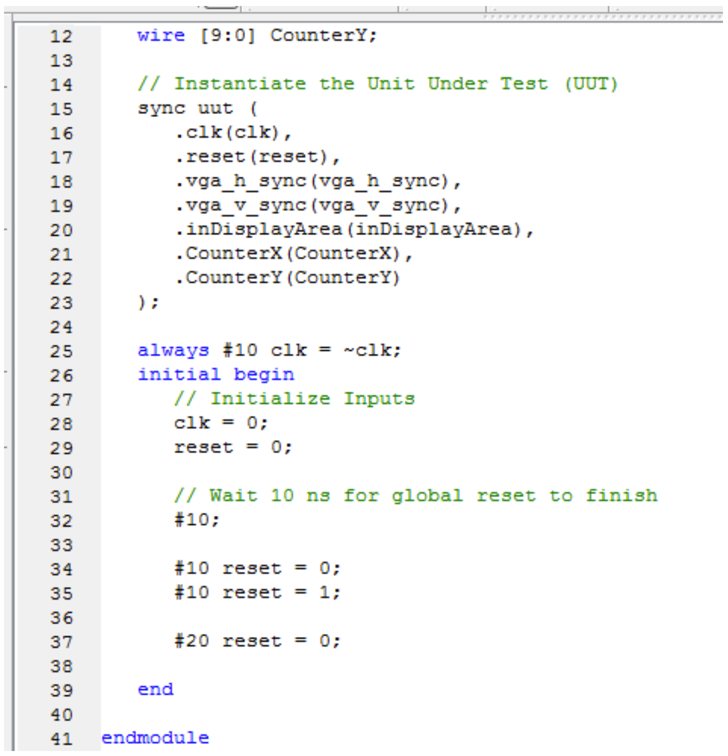
```
deb uut (
   .CLK(CLK),
   .RESET(RESET),
   .PB(PB),
   .DPB(DPB),
   .SCEN(SCEN),
   .MCEN(MCEN),
   .CCEN(CCEN)
);

always #10 CLK = ~CLK;
initial begin
   // Initialize Inputs
   CLK = 0;
   RESET = 0;
   PB = 0;

   // Wait 100 ns for global reset to finish
   #10;

   // Add stimulus here
   #10 RESET = 1;
   #10 RESET = 0;

   #10 PB = ~PB;

end

endmodule
```

VGA Module:



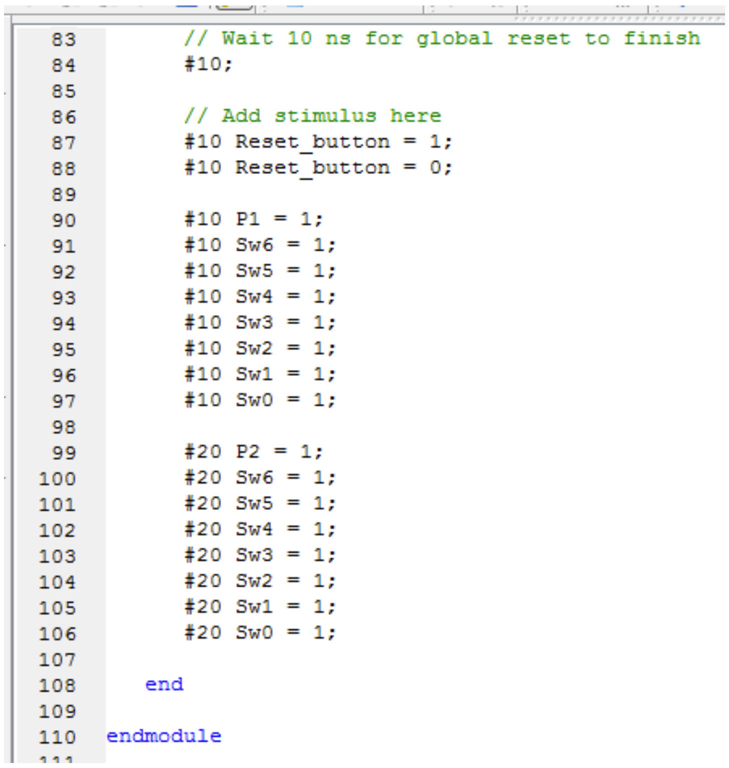
```
        .game_data(game_data),
        .empty(empty),
        .clk(clk),
        .sys_clk(sys_clk),
        .reset(reset),
        .vga_h_sync(vga_h_sync),
        .vga_v_sync(vga_v_sync),
        .vga_r(vga_r),
        .vga_g(vga_g),
        .vga_b(vga_b)
    );

    always #10 sys_clk = ~sys_clk;
    initial begin
        // Initialize Inputs
        game_data = 0;
        empty = 0;
        clk = 0;
        sys_clk = 0;
        reset = 0;

        // Wait 10 ns for global reset to finish
        #10;

        // Add stimulus here


    end

endmodule
```

Sync Module:



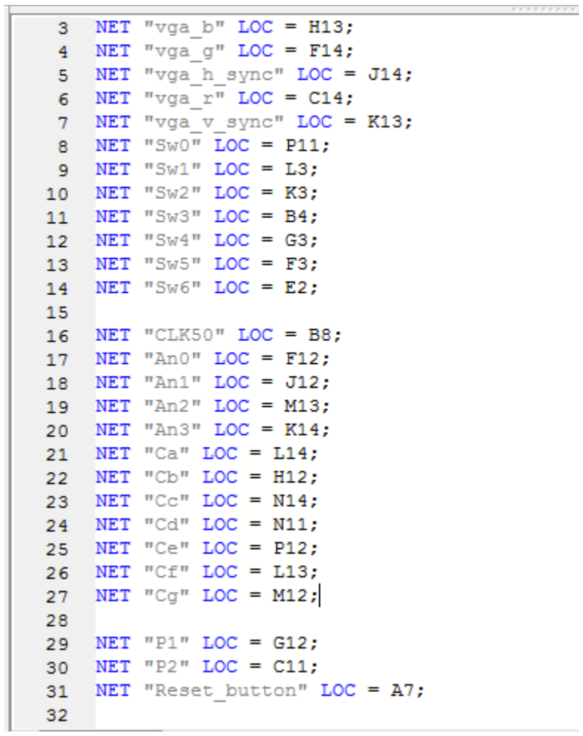
```
    wire [9:0] CounterY;

    // Instantiate the Unit Under Test (UUT)
    sync uut (
        .clk(clk),
        .reset(reset),
        .vga_h_sync(vga_h_sync),
        .vga_v_sync(vga_v_sync),
        .inDisplayArea(inDisplayArea),
        .CounterX(CounterX),
        .CounterY(CounterY)
    );

    always #10 clk = ~clk;
    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 0;

        // Wait 10 ns for global reset to finish
        #10;

        #10 reset = 0;
        #10 reset = 1;

        #20 reset = 0;

    end

endmodule
```

Top Module:



```
        // Wait 10 ns for global reset to finish
        #10;

        // Add stimulus here
        #10 Reset_button = 1;
        #10 Reset_button = 0;

        #10 P1 = 1;
        #10 Sw6 = 1;
        #10 Sw5 = 1;
        #10 Sw4 = 1;
        #10 Sw3 = 1;
        #10 Sw2 = 1;
        #10 Sw1 = 1;
        #10 Sw0 = 1;

        #20 P2 = 1;
        #20 Sw6 = 1;
        #20 Sw5 = 1;
        #20 Sw4 = 1;
        #20 Sw3 = 1;
        #20 Sw2 = 1;
        #20 Sw1 = 1;
        #20 Sw0 = 1;

    end

endmodule
```

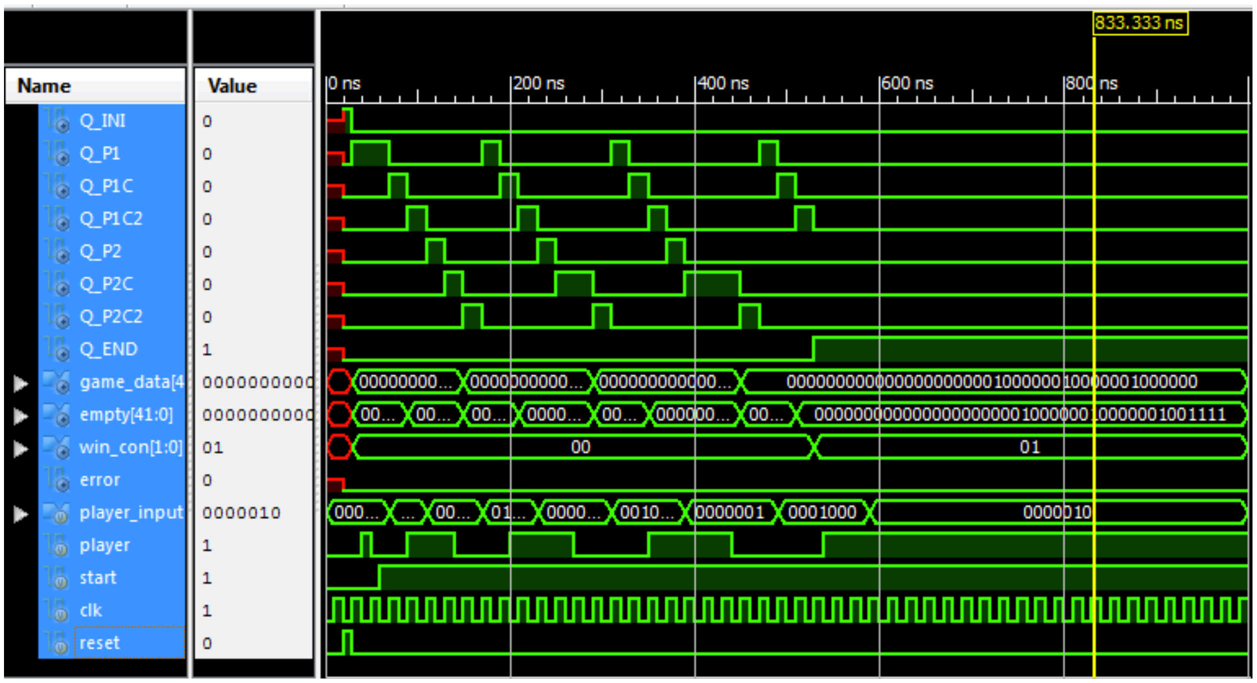
UCF:



```
NET "vga_b" LOC = H13;
NET "vga_g" LOC = F14;
NET "vga_h_sync" LOC = J14;
NET "vga_r" LOC = C14;
NET "vga_v_sync" LOC = K13;
NET "Sw0" LOC = P11;
NET "Sw1" LOC = L3;
NET "Sw2" LOC = K3;
NET "Sw3" LOC = B4;
NET "Sw4" LOC = G3;
NET "Sw5" LOC = F3;
NET "Sw6" LOC = E2;

NET "CLK50" LOC = B8;
NET "An0" LOC = F12;
NET "An1" LOC = J12;
NET "An2" LOC = M13;
NET "An3" LOC = K14;
NET "Ca" LOC = L14;
NET "Cb" LOC = H12;
NET "Cc" LOC = N14;
NET "Cd" LOC = N11;
NET "Ce" LOC = P12;
NET "Cf" LOC = L13;
NET "Cg" LOC = M12;

NET "P1" LOC = G12;
NET "P2" LOC = C11;
NET "Reset_button" LOC = A7;
```

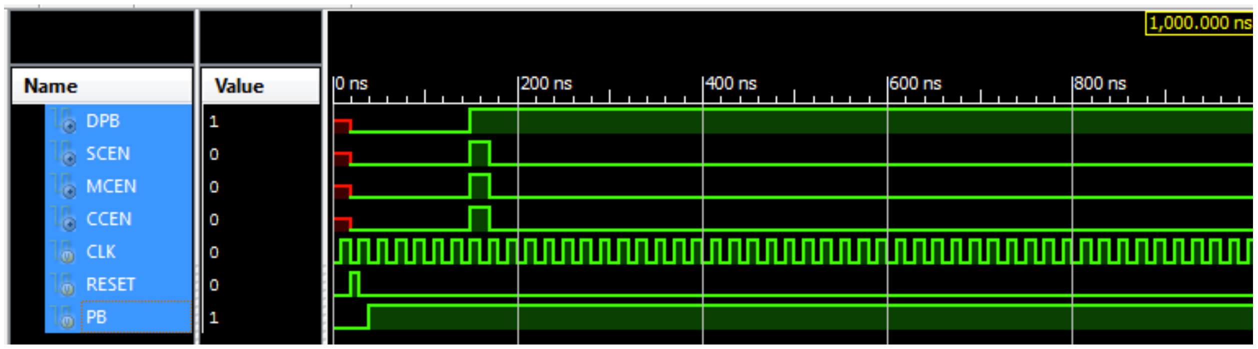
B. Testbench Diagrams

State Machine Module:



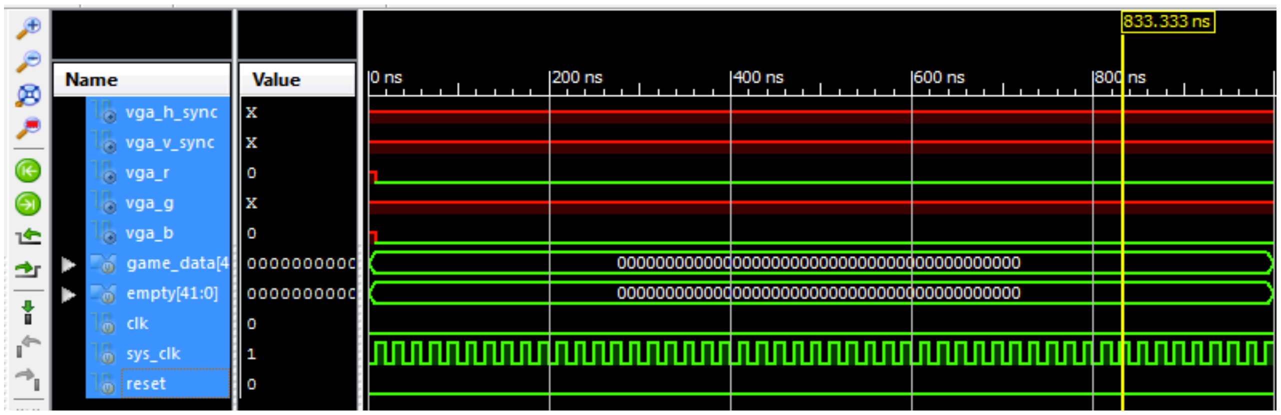
Name	Value
Q_INI	0
Q_P1	0
Q_P1C	0
Q_P1C2	0
Q_P2	0
Q_P2C	0
Q_P2C2	0
Q_END	1
game_data[4	0000000000
empty[41:0]	0000000000
win_con[1:0]	01
error	0
player_input	0000010
player	1
start	1
clk	1
reset	0

Debouncing Module:



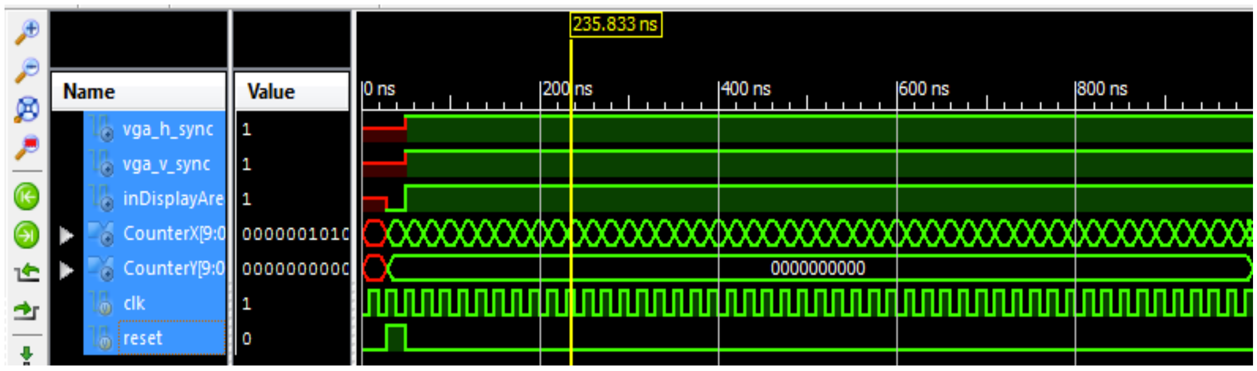
Name	Value
DPB	1
SCEN	0
MCEN	0
CCEN	0
CLK	0
RESET	0
PB	1

VGA Module:

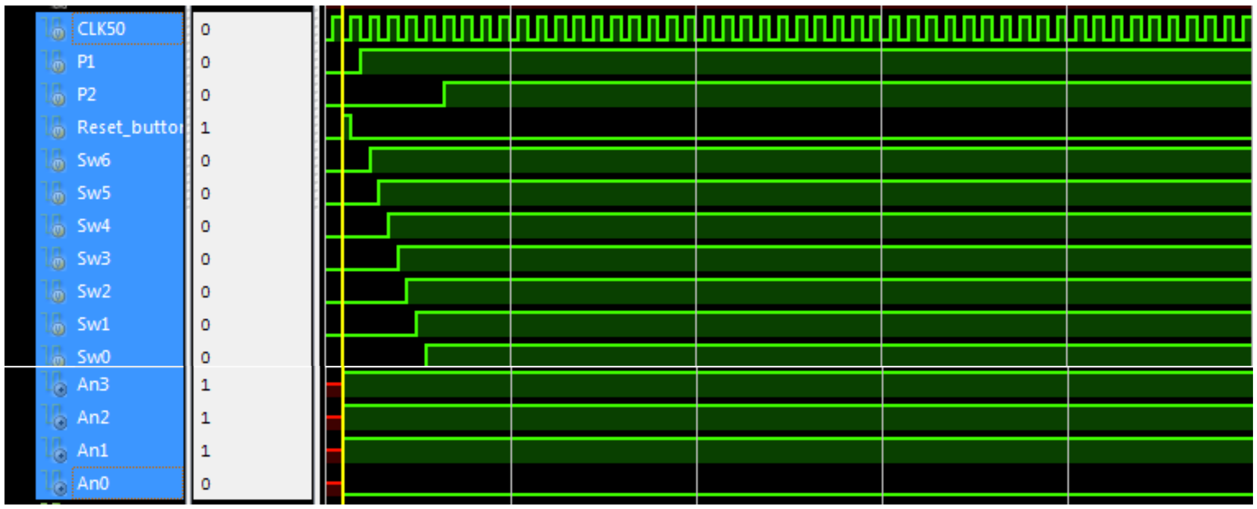


Name	Value
vga_h_sync	X
vga_v_sync	X
vga_r	0
vga_g	X
vga_b	0
game_data[4	0000000000
empty[41:0]	0000000000
clk	0
sys_clk	1
reset	0

Sync Module:



Top Module:



C. *Testbench Diagram Explanation*

State Machine Module:

Initially we reset all the values by setting the Reset to high and then to low to make sure everything is in order. Then the game begins – we are in the Initial State. Then the state that begins is the Player1 State. In this state the player chooses their input value. The input is inserted and the game data variable is updated with the inserted value. Then the next state checks if this in fact is a correct value. If so it moves to the next state which checks the state of the board to see if this move makes this player the winner. If no winner is found, then we move on the states dealing with Player2. Input for Player2 is inserted and the same checks and flags that were for Player1 apply for Player2. We continue to loop through these states and update the game data as necessary until the game data logic finds that the board is in a state in which the last player that made a move was one that deems them a winner (meaning they were able to get Four in a Row). When this happens, the last state begins which is the End state in which the game stops, signaling a winner. The game is now over. You can restart the game again by setting reset to high.

Debouncing Module:

The purpose of this module is basically to make sure that all the signals are synchronized at the same time and are in order. When looking at the waveform, you can see that when the input signal is high the concurring signals only turn high after a certain period of time. Thus the idea of debouncing. It makes sure that there is a slight delay in the output variable in order to make sure that that specific input is actually set to high. This module applies to our push button switches.

VGA Module:

The purpose of this module is to make sure the colors and color signals are being displayed properly. The color signals in this case only apply to the actual colors themselves and not the V_sync or H_sync signals. If you take a closer look at the module itself you can see that what we do in this module is just assign the necessary values to all the color lines and player inputs to make sure they will be displayed correctly, such as the red and blue player blocks. This waveform confirms the fact that these values will be displayed properly.

Sync Module:

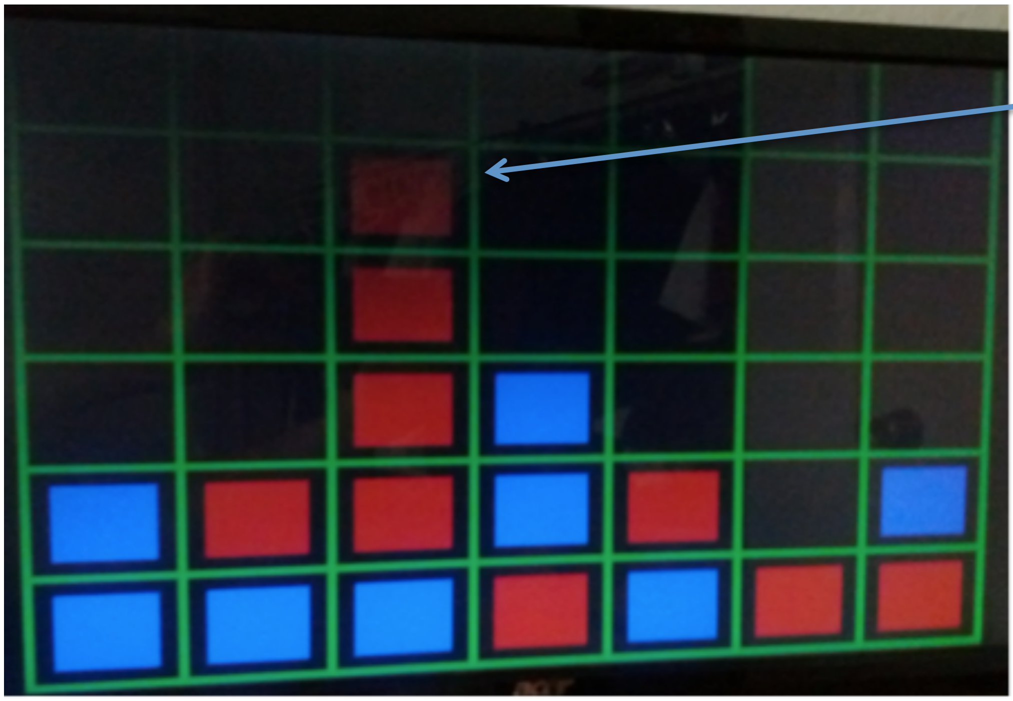
The sync module is where the major components of the VGA are being initialized and worked out. You can find in this module that the V_sync and H_sync are working here, along with the Display output which is the actual grid that is being displayed on the screen at the correct positions and times. We also make sure that the correct clock signal is being implemented and passed along to the VGA output.

Top Module:

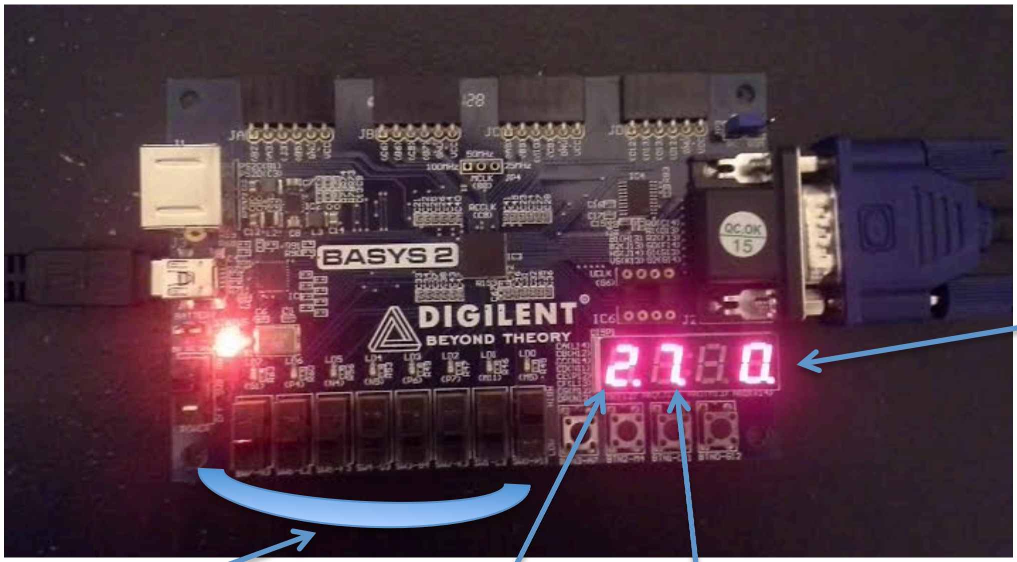
In this module we make sure that the switches and seven segments are working in order and when the switches are switched up (meaning it's high – as you can see on the waveforms). The idea is that when the user makes a switch turn high, the corresponding anodes or seven segments should also turn on and display its correct value. From the waveform shown in the previous section, you can tell that when the switches are high the corresponding anodes also turn high.

V. Results

A. Screenshots



Game over and won by Player Red because he was able to get Four Blocks in a Row.

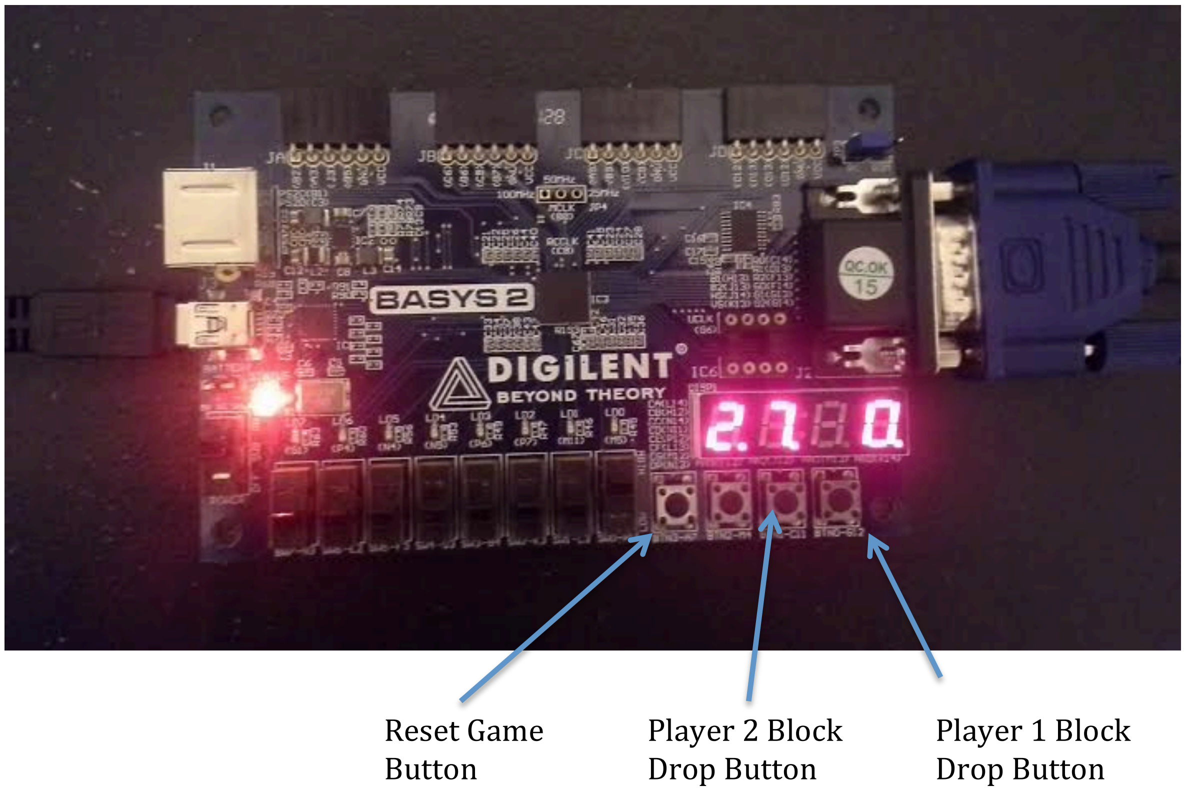


Shows which player has won the game

Switches – (column selector for user. 0 thru 6

Shows which player's turn it is

Shows which column the player is choosing (depending on the switch)



B. Video/Recordings

Video recordings will be presented in class during final project presentations and will also be attached to this project report as an additional document (if sent via electronically).