```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // File name: topLevelMod.v                                                     //
4   //                                                                              //
5   // Created by     Paul Valenzuela on 10/16/19.                                  //
6   // Copyright (C) 2018   Paul Valenzuela. All rights reserved.                   //
7   //                                                                              //
8   //                                                                              //
9   //    In submitting this file for class work at CSULB                           //
10  //    I am confirming that this is my work and the work                         //
11  //    of no one else. In submitting this code I acknowledge that                //
12  //    plagiarism in student project work is subject to dismissal                //
13  //    from the class                                                            //
14  //////////////////////////////////////////////////////////////////////////////////
15  //////////////////////////////////////////////////////////////////////////////////
16  //                            topLevelMod.v                                     //
17  //                                                                              //
18  //    The topLevelMod block instantiates all the other blocks and connects      //
19  //    them in a manner so that they all contribute to the top level design.     //
20  //    Each component is essential and is instantiated at a specific moment in   //
21  //    the execution process                                                     //
22  //                                                                              //
23  //    @input   clk, rst, switches[11:0]                                         //
24  //    @output  [3:0] vga_r, [3:0] vga_g, [3:0] vga_b, hSync, vSync              //
25  //                                                                              //
26  //////////////////////////////////////////////////////////////////////////////////
27  module topLevelModule(clk,rst,switches,hSync,vSync, vga_b, vga_r, vga_g, topButton,
    bottomButton);
28
29      // Input From Nexys A7
30      input wire clk, rst, topButton, bottomButton;
31      input wire [11:0] switches;
32
33      // Output That Designates Which Color To Output
34      output wire [3:0] vga_r, vga_g, vga_b;
35
36      // Ouput That Tells Pixels When To Update
37      output wire hSync, vSync;
38
39      // Wires That Connect Instantiations
40      wire mRst, pulse, video_on;
41      wire [9:0] pixel_x, pixel_y;
42
43      // Synchronized Reset
44      AISO mReset(.clk(clk), .rst(rst), .mRst(mRst));
45
46      // 25MHz Clock Signal
47      pixelPulse pulser (.clk(clk), .rst(mRst), .pulse(pulse));
48
49      // VSync Block Used To Synchronize Pixels
50      vgaLevelMod vgaConnect(.clk(clk), .rst(mRst), .pixel_x(pixel_x[9:0]),
51         .pixel_y(pixel_y[9:0]), .pulse(pulse), .hSync(hSync), .vSync(vSync),
52         .video_on(video_on));
53
54      // PixelGenerator Block That Designates Color Assignment
55      pixelGenerator generateVGA(.clk(clk), .rst(mRst), .pulse(pulse),
56         .pixel_x(pixel_x[9:0]), .pixel_y(pixel_y[9:0]),
```

```
57          .switches(switches[11:0]), .video_on(video_on),
58        .rgb({vga_r[3:0],vga_g[3:0],vga_b[3:0]}), .topButton(topButton), .bottomButton(
      bottomButton));
59
60    endmodule
61
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // File name: pixelPulse.v                                                      //
4   //                                                                              //
5   // Created by     Paul Valenzuela on 10/16/19.                                  //
6   // Copyright (C) 2018   Paul Valenzuela. All rights reserved.                   //
7   //                                                                              //
8   //                                                                              //
9   //    In submitting this file for class work at CSULB                           //
10  //    I am confirming that this is my work and the work                         //
11  //    of no one else. In submitting this code I acknowledge that                //
12  //    plagiarism in student project work is subject to dismissal                //
13  //    from the class                                                            //
14  //////////////////////////////////////////////////////////////////////////////////
15  //////////////////////////////////////////////////////////////////////////////////
16  //                            pixelPulse.v                                      //
17  //                                                                              //
18  //    The pulse block creates a specific 25MHz wide clock pulse signal.         //
19  //    It accomplishes this by counting up every 1ns all the way to              //
20  //    3. Once the counter reaches that number, 1 pulse signal is sent out.      //
21  //                                                                              //
22  //    @input   clk, rst                                                         //
23  //    @output  pulse                                                            //
24  //                                                                              //
25  //////////////////////////////////////////////////////////////////////////////////
26  module pixelPulse(clk,rst,pulse);
27     // Inputs
28     input clk,rst;
29
30     // 25MHz Clock Signal Output
31     output wire pulse;
32
33     // Registers That Hold Count
34     reg [1:0] count, nCount;
35
36
37     // Flop That Changes Count At Each Rising Edge of Clock
38     always@(posedge clk, posedge rst)
39        if(rst)
40           count <= 2'b0;
41        else
42           count <= nCount;
43
44     // Pulse Output 1 When Pulse == 3
45     assign pulse = (count == 2'b11);
46
47     // nCount Resets to 0 When Pulse is On
48     always @(*)
49        nCount = (pulse) ? 2'b0 : count + 2'b1;
50
51     ///100MHz / 25MHz = 4
52
53  endmodule
54
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // File name: AISO.v                                                            //
4   //                                                                              //
5   // Created by     Paul Valenzuela on 10/16/19.                                  //
6   // Copyright (C) 2018   Paul Valenzuela. All rights reserved.                   //
7   //                                                                              //
8   //                                                                              //
9   //    In submitting this file for class work at CSULB                           //
10  //    I am confirming that this is my work and the work                         //
11  //    of no one else. In submitting this code I acknowledge that                //
12  //    plagiarism in student project work is subject to dismissal                //
13  //    from the class                                                            //
14  //////////////////////////////////////////////////////////////////////////////////
15  //////////////////////////////////////////////////////////////////////////////////
16  //                            AISO.v                                            //
17  //                                                                              //
18  //    The AISO block ensures that all the blocks within the design share a      //
19  //    synchronous reset input. The AISO block works by converting the input     //
20  //    of reset into two flops. This makes sure that reset is stable and         //
21  //    helps prevent metastability                                               //
22  //                                                                              //
23  //    @input   clk, rst,                                                        //
24  //    @output  sRst                                                             //
25  //                                                                              //
26  //////////////////////////////////////////////////////////////////////////////////
27  module AISO(clk,rst,mRst);
28
29     // Inputs
30     input clk, rst;
31
32     // Synchonized Reset Output
33     output wire mRst;
34
35     // Registers That Hold Flop Values
36     reg q1,q2;
37
38     // Flop
39     always@(posedge clk, posedge rst)
40        if(rst)
41           q1 <= 1'b0;
42        else
43           {q1,q2} <= {1'b1,q1};
44
45     // Output Wire With Inverter
46     assign mRst = ~q2;
47
48
49  endmodule
50
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // File name: vgaLevelMod.v                                                     //
4   //                                                                              //
5   // Created by     Paul Valenzuela on 10/16/19.                                  //
6   // Copyright (C) 2018   Paul Valenzuela. All rights reserved.                   //
7   //                                                                              //
8   //                                                                              //
9   //    In submitting this file for class work at CSULB                           //
10  //    I am confirming that this is my work and the work                         //
11  //    of no one else. In submitting this code I acknowledge that                //
12  //    plagiarism in student project work is subject to dismissal                //
13  //    from the class                                                            //
14  //////////////////////////////////////////////////////////////////////////////////
15  //////////////////////////////////////////////////////////////////////////////////
16  //                            vgeLevelMod.v                                     //
17  //                                                                              //
18  //    The vgaLevelMod block instantiates the hSync and vSync blocks             //
19  //    in a manner so that they all contribute to the vga level design.          //
20  //    They help to establish a precise hSync and vSync signal. This module      //
21  //    also  establishes a video_on signal that tells rgb when to output
    //
22  //                                                                              //
23  //    @input   clk, rst, pulse                                                  //
24  //    @output  [9:0] pixel_x, [9:0] pixel_y, hSync, vSync, video_on             //
25  //                                                                              //
26  //////////////////////////////////////////////////////////////////////////////////
27  module vgaLevelMod(clk,rst,pixel_x, pixel_y, hSync, vSync,pulse,video_on);
28
29     // Inputs from Top Level
30     input wire clk, rst,pulse;
31
32     // Wires Between Instantiations
33     wire hMax, vMax, vOn, hOn;
34
35     // 10-Bit Output
36     output wire [9:0] pixel_x, pixel_y;
37
38     // Outputs That Are Sent to Top Module To Refresh Pixels and Turn on RGB
39     output wire hSync, vSync, video_on;
40
41
42
43     // hSync Instantiation
44     hSync horizontal (.clk(clk), .rst(rst), .count(pixel_x[9:0]), .hMax(hMax),
45        .pixelPulse(pulse), .hSync(hSync), .hOn(hOn));
46
47     // vSync Instantiation
48     vSync vertical (.clk(clk), .rst(rst) , .pixelPulse(pulse), .hMax(hMax),
49        .count(pixel_y[9:0]), .vMax(vMax), .vSync(vSync), .vOn(vOn));
50
51
52     // Video On Depends On hOn and vOn Outputs Being Both On
53     assign video_on = (hOn && vOn) ;
54
55
56
```

```
57    endmodule
58
```

```
1   `timescale 1ns / 1ps
2   //**********************************************************************************//
3   //                      File name: hSync.v                                          //
4   //                                                                                  //
5   // Created by     Paul Valenzuela on 10/11/19.                                      //
6   // Copyright (C) 2018   Paul Valenzuela. All rights reserved.                       //
7   //                                                                                  //
8   //                                                                                  //
9   //    In submitting this file for class work at CSULB                               //
10  //    I am confirming that this is my work and the work                             //
11  //    of no one else. In submitting this code I acknowledge that                    //
12  //    plagiarism in student project work is subject to dismissal                    //
13  //    from the class                                                                //
14  //**********************************************************************************//
15  //**********************************************************************************//
16  //                            hSync.v                                               //
17  //                                                                                  //
18  //    The hSync block uses a counter that increments at a 25MHz rate.               //
19  //    This module will output hSync, hMax, and hOn depending on the value of        //
20  //    the registered counter                                                        //
21  //                                                                                  //
22  //    @input   clk, rst, pixelPulse                                                 //
23  //    @output  [9:0] count, hSync, hMax, hOn                                        //
24  //                                                                                  //
25  //**********************************************************************************//
26  module hSync(clk,rst,pixelPulse,count,hMax,hSync, hOn);
27
28     // Inputs
29     input wire clk,rst, pixelPulse;
30
31     // 10-Bit Output Counter Value
32     output reg [9:0] count;
33
34     // Ouputs That Depend on the Value of Counter
35     output wire hMax, hSync, hOn;
36
37     // Counter Used in Combinational Block
38     reg [9:0] nCount;
39
40     // hMax Is On When 800 Reached
41     assign hMax = (count == 10'd799);
42
43     // hSync is On When Count Is Not 656-751
44     assign hSync = ~((count >= (10'd656)) && (count <= (10'd751)));
45
46     // hOn is On When Count Is Less Than 640
47     assign hOn = (count < (10'd640));
48
49     // Combinational Block That Decides When To Increment nCount
50     always@(*)
51        case({hMax,pixelPulse})
52           // When 00 nCount Remains The Same
53           2'b00: nCount = count;
54           // When pixelPulse, nCount Increments by 1
55           2'b01: nCount = count + 10'b1;
56           // When hMax && !pixelPulse, nCount Remains The Same
57           2'b10: nCount = count;
```

```
58            // When hMax && pixelPulse, nCount Resets to 0
59            2'b11: nCount = 10'b0;
60         endcase
61
62      //Flop That Increments Count at Rising Clock Edge
63      always@(posedge clk, posedge rst)
64         if(rst)
65            {count} <= 10'b0;
66         else
67            {count} <= nCount;
68
69
70   endmodule
71
```

```
1   `timescale 1ns / 1ps
2   //**********************************************************************************//
3   //                      File name: vSync.v                                          //
4   //                                                                                  //
5   // Created by     Paul Valenzuela on 10/11/19.                                      //
6   // Copyright (C) 2018   Paul Valenzuela. All rights reserved.                       //
7   //                                                                                  //
8   //                                                                                  //
9   //    In submitting this file for class work at CSULB                               //
10  //    I am confirming that this is my work and the work                             //
11  //    of no one else. In submitting this code I acknowledge that                    //
12  //    plagiarism in student project work is subject to dismissal                    //
13  //    from the class                                                                //
14  //**********************************************************************************//
15  //**********************************************************************************//
16  //                             vSync.v                                              //
17  //                                                                                  //
18  //    The vSync block uses a counter that increments at a 25MHz rate.               //
19  //    This module will output vSync, vMax, and vOn depending on the value of       //
20  //    the registered counter                                                        //
21  //                                                                                  //
22  //    @input   clk, rst, pixelPulse                                                 //
23  //    @output  [9:0] count, vSync, vMax, vOn                                        //
24  //                                                                                  //
25  //**********************************************************************************//
26  module vSync(clk,rst,hMax,count, pixelPulse, vMax,vSync, vOn);
27
28     // Inputs
29     input clk, rst, hMax, pixelPulse;
30
31     // 10-Bit Output, Counter Value
32     output reg [9:0] count;
33
34     // Ouputs That Depend on the Value of Counter
35     output wire vMax, vSync, vOn;
36
37     // Counter Used in Combinational Block
38     reg [9:0] nCount;
39
40     //Flop That Increments Count at Rising Clock Edge
41     always@(posedge clk, posedge rst)
42        if(rst)
43           count <= 10'b0;
44        else
45           count <= nCount;
46
47     // vMax is On When 525 is Reached
48     assign vMax = (count == 10'd524);
49
50     // vSync Is On When Count Is Not 490 or 491
51     assign vSync = ~((count == (10'd490)) || (count == (10'd491)));
52
53     // vOn Is On When Count Is Less Than 480
54     assign vOn = (count < (10'd480));
55
56     // Combinational Block That Decides When To Increment nCount
57     always@(*)
```

```
58          case({vMax, (hMax && pixelPulse)})
59             // When 00 nCount Remains The Same
60             2'b00: nCount = count;
61             // When hMax && pixelPulse, nCount Increments by 1
62             2'b01: nCount = count + 10'b1;
63             // When !(hMax && pixelPulse), nCount Remains the Same
64             2'b10: nCount = count;
65             // When hMax && pixelPulse && vMax, nCount Resets
66             2'b11: nCount = 10'b0;
67             default: nCount = 10'b0;
68          endcase
69
70
71    endmodule
72
```

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //                      File name: pixelGenerator                               //
4   //                                                                              //
5   // Created by     Paul Valenzuela on 10/17/19.                                  //
6   // Copyright (C) 2018   Paul Valenzuela. All rights reserved.                   //
7   //                                                                              //
8   //                                                                              //
9   //    In submitting this file for class work at CSULB                           //
10  //    I am confirming that this is my work and the work                         //
11  //    of no one else. In submitting this code I acknowledge that                //
12  //    plagiarism in student project work is subject to dismissal                //
13  //    from the class                                                            //
14  //////////////////////////////////////////////////////////////////////////////////
15  //////////////////////////////////////////////////////////////////////////////////
16  //                            pixelGenerator.v                                  //
17  //                                                                              //
18  //    The pixelGenerator block assigns rgb values depending on values of        //
19  //    12 switches, the location of the pixels, and video_on.                    //
20  //    More specifically, this module creates the shapes of a pong game.         //
21  //    It also animates the pieces needed to operate a pong game.                //
22  //                                                                              //
23  //                                                                              //
24  //    @input   clk, rst, pulse, pixel_x, pixel_y, switches, video_on            //
25  //    @output  [11:0] rgb                                                       //
26  //                                                                              //
27  //////////////////////////////////////////////////////////////////////////////////
28  module pixelGenerator(clk,rst,pulse, pixel_x,pixel_y,switches, video_on,rgb, topButton
    , bottomButton);
29
30      // Inputs
31      input wire clk, rst, pulse, video_on, topButton, bottomButton;
32
33      // 10-Bit Inputs Indicating Pixel Coordinate
34      input [9:0] pixel_x, pixel_y;
35
36      // 12-Bit Switch Input
37      input [11:0] switches;
38
39      // 12-Bit RGB Output
40      output reg [11:0] rgb;
41
42      // Registered nRGB Value
43      reg [11:0] nRgb;
44
45      // Register to Detect Where Paddle Is
46      reg [9:0] topOfPaddle, nTopOfPaddle;
47
48      // Debounced Buttons
49      wire upDetected, downDetected;
50
51      // Signal That Tells Pixels When To Update Location
52      wire refesh;
53
54      // The Farthest The Paddle Can Go On Screen
55      localparam paddleBottomEdge = 475;
56      localparam paddleTopEdge = 9;
```

```
57
58      // Height of Paddle
59      localparam paddleHeight = 40;
60
61      // Sides of Paddle
62      localparam paddleLeftSide = 10;
63      localparam paddleRightSide = 16;
64
65      // Top Wall Borders
66      localparam topWallUpperBorder = 3;
67      localparam topWallLowerBorder = 9;
68
69      // Bottom Wall Borders
70      localparam bottomWallUpperBorder = 475;
71      localparam bottomWallLowerBorder = 481;
72
73      // Right Wall Borders
74      localparam rightWallLeftBorder = 624;
75      localparam rightWallRightBorder = 630;
76
77      // Borders That Will Indicate Position of Ball
78      wire [9:0] ballLeftBorder ;
79      wire [9:0] ballRightBorder ;
80      wire [9:0] ballUpperBorder ;
81      wire [9:0] ballLowerBorder ;
82
83      // Dimensions Of Ball
84      localparam ballHeight = 5 ;
85      localparam ballWidth  = 5;
86
87      // Register Values to Hold Location of Ball
88      reg [9:0] ballUpperReg;
89      wire [9:0] ballUpperNext;
90      reg [9:0] ballRightReg;
91      wire [9:0] ballRightNext;
92
93      // Horizontal Velocity of Ball
94      reg [9:0] ballXDeltaReg;
95      reg [9:0] ballXDeltaNext;
96
97      // Vertical Velocity of Ball
98      reg [9:0] ballYDeltaReg;
99      reg [9:0] ballYDeltaNext;
100
101     // Pre-Defined Velocities
102     localparam ballVelocityLeft = -1;
103     localparam ballVelocityRight = 1;
104     localparam ballVelocityUp = -1;
105     localparam ballVelocityDown = 1;
106
107     // Wire That Indicates When The Pixel is On The Ball Pixel Values
108     wire ballOn;
109
110     // The Ball Lower Border Should Be The Upper Border + The Height
111     assign ballLowerBorder = ballUpperBorder + ballHeight ;
112
113     // The Pixels Have Reached The Ball Pixels When They Are Within The Ball Borders
```

```
114      assign ballOn = (pixel_x >= ballLeftBorder && pixel_x <= ballRightBorder) && (
         pixel_y <= ballLowerBorder && pixel_y >= ballUpperBorder);
115
116      // The Right Border Will Always Be The Horizontal Reg Value
117      assign ballRightBorder = ballRightReg;
118      assign ballLeftBorder = ballRightReg - ballWidth;
119
120      // The Upper Border Will Always Be The Vertical Reg Value
121      assign ballUpperBorder = ballUpperReg;
122
123      // The Location Refreshes Everytime the Monitor Refreshes
124      assign refresh = (pixel_y == 481) && (pixel_x == 0);
125
126      // Debounce the Buttons on The Board
127      debounce up (.clk(clk),.rst(rst),.pulse(pulse),.button(topButton), .yes(upDetected
         ));
128      debounce down ( .clk(clk),.rst(rst),.pulse(pulse),.button(bottomButton), .yes(
         downDetected));
129
130      // Flop That Updates rgb on Rising Clock Edge
131      always @(posedge clk , posedge rst)
132         if (rst)
133            nRgb <= 12'b0;
134         else
135            nRgb <= switches;
136
137      // The Paddle Will Get Its Next Value Unless Reset is Active
138      always @(posedge clk , posedge rst)
139         if (rst)
140            topOfPaddle <= 10'd249;
141         else
142            topOfPaddle <= nTopOfPaddle;
143
144      // Combinational Block Telling Paddle Where to Move
145      always@(*)
146         case({upDetected,downDetected,refresh})
147            // If Refresh Inactive, Paddle Stays In Position
148            3'b000: nTopOfPaddle = topOfPaddle;
149            3'b001: nTopOfPaddle = topOfPaddle;
150            3'b010: nTopOfPaddle = topOfPaddle;
151            // If Refresh Active and Down Pressed, Paddle Goes Down
152            3'b011:
153               begin
154               nTopOfPaddle = topOfPaddle + 10'd2;
155               //Paddle Cannot Go Lower Than Bottom Wall
156               if(topOfPaddle + paddleHeight >= paddleBottomEdge) nTopOfPaddle =
                  paddleBottomEdge - paddleHeight;
157               end
158            // If Refresh Inactive, Paddle Stays In Position
159            3'b100: nTopOfPaddle = topOfPaddle;
160            // If Refresh Active and Up Pressed, Paddle Goes Up
161            3'b101:
162               begin
163                nTopOfPaddle = topOfPaddle - 10'd2;
164                // Paddle Cannot Go Higher Than Top Wall
165                if (topOfPaddle <= paddleTopEdge) nTopOfPaddle = paddleTopEdge;
166               end
```

```
167            // If Refresh Inactive, Paddle Stays In Position
168            3'b110: nTopOfPaddle = topOfPaddle;
169            // If Both Buttons Pressed, Paddle Does Not Move
170            3'b111: nTopOfPaddle = topOfPaddle;
171            default: nTopOfPaddle = topOfPaddle;
172         endcase
173
174      // Flop That Updates The Registers At Every Rising Clock Edge
175      always @(posedge clk, posedge rst)
176         // If Reset or The Ball Has Passed the Paddle, Reset Position and Velocity of
       Ball
177         if((rst) || (ballLeftBorder < paddleLeftSide))
178            begin
179               ballRightReg <= 317;
180               ballUpperReg <= 237;
181               ballXDeltaReg <= -1;
182               ballYDeltaReg <= 1;
183            end
184         // Otherwise Continue
185         else
186            begin
187               ballRightReg <= ballRightNext;
188               ballUpperReg <= ballUpperNext;
189               ballXDeltaReg <= ballXDeltaNext;
190               ballYDeltaReg <= ballYDeltaNext;
191            end
192
193      // The Next Horizontal Value of Ball Will Be The Previous Position Plus Horizontal
       Velocity
194      assign ballRightNext = (refresh) ? ballRightReg + ballXDeltaReg : ballRightReg;
195
196      // The Next Vertical Value of Ball Will Be The Previous Position Plus Vertical
       Velocity
197      assign ballUpperNext = (refresh) ? ballUpperReg + ballYDeltaReg : ballUpperReg;
198
199      // This Combinational Block Tells The Delta Reg When To Change Velocity
200      always@(*)
201         begin
202            ballXDeltaNext = ballXDeltaReg;
203            ballYDeltaNext = ballYDeltaReg;
204
205            // If Ball Hits Top Wall, It Goes Down
206            if(ballUpperBorder <= topWallLowerBorder)
207               ballYDeltaNext = ballVelocityDown;
208
209            // If Ball Hits Lower Wall, It Goes Up
210            else if(ballLowerBorder >= bottomWallUpperBorder)
211               ballYDeltaNext = ballVelocityUp;
212
213            // If Ball Hits Right Wall, It Goes Left
214            else if(ballRightBorder >= rightWallLeftBorder)
215               ballXDeltaNext = ballVelocityLeft;
216
217            // If Ball Hits Paddle, It Switches Horizontal Velocity
218            else if(((ballLeftBorder) <= paddleRightSide) && (ballLeftBorder >=
       paddleLeftSide) && ((ballUpperBorder)>= topOfPaddle) && ((ballUpperBorder <=
       topOfPaddle + paddleHeight)))
```

```
219              begin
220              ballXDeltaNext = ballVelocityRight;
221              end
222
223         end
224
225
226       always@(*)
227
228         // Top Wall
229         if((pixel_x >=2 && pixel_x <= rightWallRightBorder) && (pixel_y >=
    topWallUpperBorder && pixel_y <= topWallLowerBorder) && video_on) rgb = 12'hF0F;
230
231         // Bottom Wall
232         else if((pixel_x >=2 && pixel_x <= rightWallRightBorder) && (pixel_y >=
    bottomWallUpperBorder && pixel_y <= bottomWallLowerBorder) && video_on) rgb = 12'hF0f;
233
234         // Left Paddle
235         else if((pixel_x >= paddleLeftSide && pixel_x <= paddleRightSide) && (pixel_y >=
     topOfPaddle) && (pixel_y <= topOfPaddle + paddleHeight) && video_on) rgb = nRgb;
236
237         // Right Wall
238         else if((pixel_x >= rightWallLeftBorder && pixel_x <= rightWallRightBorder)  &&
    (pixel_y >= topWallUpperBorder && pixel_y <= bottomWallLowerBorder) && video_on) rgb =
     12'hF0F;
239
240         // Ball
241         else if((ballOn) && video_on) rgb = 12'hFFF;
242
243         // Black Background
244         else rgb = 12'b0;
245
246
247
248
249   endmodule
250
```

```
1   `timescale 1ns / 1ps
2   //**********************************************************************************//
3   // File name: debounceStateMachine.v                                                //
4   //                                                                                  //
5   // Created by     Paul Valenzuela on 10/1/19.                                       //
6   // Copyright (C) 2018   Paul Valenzuela. All rights reserved.                       //
7   //                                                                                  //
8   //                                                                                  //
9   //    In submitting this file for class work at CSULB                               //
10  //    I am confirming that this is my work and the work                             //
11  //    of no one else. In submitting this code I acknowledge that                    //
12  //    plagiarism in student project work is subject to dismissal                    //
13  //    from the class                                                                //
14  //**********************************************************************************//
15  //**********************************************************************************//
16  //                          debounceStateMachine.v                                  //
17  //                                                                                  //
18  //    The debounceStateMachine is a state machine that registers the output         //
19  //    using a Modified-Moore state machine implementation. It ensures that         //
20  //    input has been stable for at least 30ms before the output is one              //
21  //                                                                                  //
22  //    @input   clk, rst, pulse, button                                              //
23  //    @output  yes                                                                  //
24  //                                                                                  //
25  //**********************************************************************************//
26  module debounce(clk,rst,pulse,button,yes);
27     input clk,rst,pulse,button;
28     output reg yes;
29     reg [2:0] state, nState;
30     reg nYes;
31
32     always@(posedge clk, posedge rst)
33        if(rst)
34           {state,yes} <= 4'b0;
35        else
36           {state,yes} <= {nState, nYes};
37
38     always@(*)
39        case({state,pulse,button})
40           5'b000_00: {nState, nYes} = 4'b000_0;
41           5'b000_01: {nState, nYes} = 4'b001_0;
42           5'b000_10: {nState, nYes} = 4'b000_0;
43           5'b000_11: {nState, nYes} = 4'b001_0;
44           5'b001_00: {nState, nYes} = 4'b000_0;
45           5'b001_01: {nState, nYes} = 4'b001_0;
46           5'b001_10: {nState, nYes} = 4'b000_0;
47           5'b001_11: {nState, nYes} = 4'b010_0;
48           5'b010_00: {nState, nYes} = 4'b000_0;
49           5'b010_01: {nState, nYes} = 4'b010_0;
50           5'b010_10: {nState, nYes} = 4'b000_0;
51           5'b010_11: {nState, nYes} = 4'b011_0;
52           5'b011_00: {nState, nYes} = 4'b000_0;
53           5'b011_01: {nState, nYes} = 4'b011_0;
54           5'b011_10: {nState, nYes} = 4'b000_0;
55           5'b011_11: {nState, nYes} = 4'b100_1;
56           5'b100_00: {nState, nYes} = 4'b101_1;
57           5'b100_01: {nState, nYes} = 4'b100_1;
```

```
58            5'b100_10: {nState, nYes} = 4'b101_1;
59            5'b100_11: {nState, nYes} = 4'b100_1;
60            5'b101_00: {nState, nYes} = 4'b101_1;
61            5'b101_01: {nState, nYes} = 4'b100_1;
62            5'b101_10: {nState, nYes} = 4'b110_1;
63            5'b101_11: {nState, nYes} = 4'b100_1;
64            5'b110_00: {nState, nYes} = 4'b110_1;
65            5'b110_01: {nState, nYes} = 4'b100_1;
66            5'b110_10: {nState, nYes} = 4'b111_1;
67            5'b110_11: {nState, nYes} = 4'b100_1;
68            5'b111_00: {nState, nYes} = 4'b111_1;
69            5'b111_01: {nState, nYes} = 4'b100_1;
70            5'b111_10: {nState, nYes} = 4'b000_0;
71            5'b111_11: {nState, nYes} = 4'b100_1;
72            default: {nState, nYes} = 4'b000_0;
73        endcase
74
75
76
77
78
79
80  endmodule
81
```

```
## This file is a general .ucf for the Nexys4 DDR Rev C board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project

## Clock signal
NET "clk"   LOC = "E3"  | IOSTANDARD = "LVCMOS33";             #Bank = 35, Pin name =
#IO_L12P_T1_MRCC_35,              Sch name = clk100mhz
#NET "clk100mhz" TNM_NET = sys_clk_pin;
#TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100 MHz HIGH 50%;


## Switches
NET "switches<0>"          LOC=J15 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_RS0_15
NET "switches<1>"          LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "switches<2>"          LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "switches<3>"          LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
NET "switches<4>"          LOC=R17 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_14
NET "switches<5>"          LOC=T18 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_D10_14
NET "switches<6>"          LOC=U18 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A13_D29_14
NET "switches<7>"          LOC=R13 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_D07_14
NET "switches<8>"          LOC=T8 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_34
NET "switches<9>"          LOC=U8 | IOSTANDARD=LVCMOS33; #IO_25_34
NET "switches<10>"         LOC=R16 | IOSTANDARD=LVCMOS33; #IO_L15P_T2_DQS_RDWR_B_14
NET "switches<11>"         LOC=T13 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_A03_D19_14
#NET "sw<12>"          LOC=H6 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_35
#NET "sw<13>"          LOC=U12 | IOSTANDARD=LVCMOS33; #IO_L20P_T3_A08_D24_14
#NET "sw<14>"          LOC=U11 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A09_D25_VREF_14
#NET "sw<15>"          LOC=V10 | IOSTANDARD=LVCMOS33; #IO_L21P_T3_DQS_14


## Buttons
#NET "cpu_resetn"     LOC=C12 | IOSTANDARD=LVCMOS33; #IO_L3P_T0_DQS_AD1P_15

#NET "btnc"           LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14
NET "rst"           LOC=P18 | IOSTANDARD=LVCMOS33; #IO_L9N_T1_DQS_D13_14
#NET "btnl"           LOC=P17 | IOSTANDARD=LVCMOS33; #IO_L12P_T1_MRCC_14
#NET "btnr"           LOC=M17 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_D15_14
#NET "button"           LOC=M18 | IOSTANDARD=LVCMOS33; #IO_L4N_T0_D05_14


## LEDs
#NET "led<0>"          LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
#NET "led<1>"          LOC=K15 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_RS1_15
#NET "led<2>"          LOC=J13 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A25_15
#NET "led<3>"          LOC=N14 | IOSTANDARD=LVCMOS33; #IO_L8P_T1_D11_14
#NET "led<4>"          LOC=R18 | IOSTANDARD=LVCMOS33; #IO_L7P_T1_D09_14
#NET "led<5>"          LOC=V17 | IOSTANDARD=LVCMOS33; #IO_L18N_T2_A11_D27_14
#NET "led<6>"          LOC=U17 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A14_D30_14
#NET "led<7>"          LOC=U16 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A12_D28_14
#NET "led<8>"          LOC=V16 | IOSTANDARD=LVCMOS33; #IO_L16N_T2_A15_D31_14
#NET "led<9>"          LOC=T15 | IOSTANDARD=LVCMOS33; #IO_L14N_T2_SRCC_14
#NET "led<10>"         LOC=U14 | IOSTANDARD=LVCMOS33; #IO_L22P_T3_A05_D21_14
#NET "led<11>"         LOC=T16 | IOSTANDARD=LVCMOS33; #IO_L15N_T2_DQS_DOUT_CSO_B_14
#NET "led<12>"         LOC=V15 | IOSTANDARD=LVCMOS33; #IO_L16P_T2_CSI_B_14
#NET "led<13>"         LOC=V14 | IOSTANDARD=LVCMOS33; #IO_L22N_T3_A04_D20_14
#NET "led<14>"         LOC=V12 | IOSTANDARD=LVCMOS33; #IO_L20N_T3_A07_D23_14
```

```
57  #NET "led<15>"        LOC=V11 | IOSTANDARD=LVCMOS33; #IO_L21N_T3_DQS_A06_D22_14
58
59
60  ##LEDs_RGB
61  #NET "led16_b"        LOC=R12 | IOSTANDARD=LVCMOS33; #IO_L5P_T0_D06_14
62  #NET "led16_g"        LOC=M16 | IOSTANDARD=LVCMOS33; #IO_L10P_T1_D14_14
63  #NET "led16_r"        LOC=N15 | IOSTANDARD=LVCMOS33; #IO_L11P_T1_SRCC_14
64  #NET "led17_b"        LOC=G14 | IOSTANDARD=LVCMOS33; #IO_L15N_T2_DQS_ADV_B_15
65  #NET "led17_g"        LOC=R11 | IOSTANDARD=LVCMOS33; #IO_0_14
66  #NET "led17_r"        LOC=N16 | IOSTANDARD=LVCMOS33; #IO_L11N_T1_SRCC_14
67
68
69  ## 7 segment display
70  #NET "cathode[0]"            LOC=T10 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_A00_D16_14
71  #NET "cathode[1]"            LOC=R10 | IOSTANDARD=LVCMOS33; #IO_25_14
72  #NET "cathode[2]"            LOC=K16 | IOSTANDARD=LVCMOS33; #IO_25_15
73  #NET "cathode[3]"            LOC=K13 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A26_15
74  #NET "cathode[4]"            LOC=P15 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_14
75  #NET "cathode[5]"            LOC=T11 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A10_D26_14
76  #NET "cathode[6]"            LOC=L18 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_D04_14
77  #NET "cathode[7]"            LOC=H15 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_A21_VREF_15
78
79  #NET "anode[0]"          LOC=J17 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_FOE_B_15
80  #NET "anode[1]"          LOC=J18 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_FWE_B_15
81  #NET "anode[2]"          LOC=T9 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_A01_D17_14
82  #NET "anode[3]"          LOC=J14 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_A22_15
83  #NET "anode[4]"          LOC=P14 | IOSTANDARD=LVCMOS33; #IO_L8N_T1_D12_14
84  #NET "anode[5]"          LOC=T14 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_14
85  #NET "anode[6]"          LOC=K2 | IOSTANDARD=LVCMOS33; #IO_L23P_T3_35
86  #NET "anode[7]"          LOC=U13 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_A02_D18_14
87
88
89  ## Pmod Header JA
90  #NET "ja<1>"          LOC=C17 | IOSTANDARD=LVCMOS33; #IO_L20N_T3_A19_15
91  #NET "ja<2>"          LOC=D18 | IOSTANDARD=LVCMOS33; #IO_L21N_T3_DQS_A18_15
92  #NET "ja<3>"          LOC=E18 | IOSTANDARD=LVCMOS33; #IO_L21P_T3_DQS_15
93  #NET "ja<4>"          LOC=G17 | IOSTANDARD=LVCMOS33; #IO_L18N_T2_A23_15
94  #NET "ja<7>"          LOC=D17 | IOSTANDARD=LVCMOS33; #IO_L16N_T2_A27_15
95  #NET "ja<8>"          LOC=E17 | IOSTANDARD=LVCMOS33; #IO_L16P_T2_A28_15
96  #NET "ja<9>"          LOC=F18 | IOSTANDARD=LVCMOS33; #IO_L22N_T3_A16_15
97  #NET "ja<10>"         LOC=G18 | IOSTANDARD=LVCMOS33; #IO_L22P_T3_A17_15
98
99  ## Pmod Header JB
100 #NET "jb<1>"          LOC=D14 | IOSTANDARD=LVCMOS33; #IO_L1P_T0_AD0P_15
101 #NET "jb<2>"          LOC=F16 | IOSTANDARD=LVCMOS33; #IO_L14N_T2_SRCC_15
102 #NET "jb<3>"          LOC=G16 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_15
103 #NET "jb<4>"          LOC=H14 | IOSTANDARD=LVCMOS33; #IO_L15P_T2_DQS_15
104 #NET "jb<7>"          LOC=E16 | IOSTANDARD=LVCMOS33; #IO_L11N_T1_SRCC_15
105 #NET "jb<8>"          LOC=F13 | IOSTANDARD=LVCMOS33; #IO_L5P_T0_AD9P_15
106 #NET "jb<9>"          LOC=G13 | IOSTANDARD=LVCMOS33; #IO_0_15
107 #NET "jb<10>"         LOC=H16 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_15
108
109 ## Pmod Header JC
110 #NET "jc<1>"          LOC=K1 | IOSTANDARD=LVCMOS33; #IO_L23N_T3_35
111 #NET "jc<2>"          LOC=F6 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_VREF_35
112 #NET "jc<3>"          LOC=J2 | IOSTANDARD=LVCMOS33; #IO_L22N_T3_35
113 #NET "jc<4>"          LOC=G6 | IOSTANDARD=LVCMOS33; #IO_L19P_T3_35
```

```
114   #NET "jc<7>"          LOC=E7 | IOSTANDARD=LVCMOS33; #IO_L6P_T0_35
115   #NET "jc<8>"          LOC=J3 | IOSTANDARD=LVCMOS33; #IO_L22P_T3_35
116   #NET "jc<9>"          LOC=J4 | IOSTANDARD=LVCMOS33; #IO_L21P_T3_DQS_35
117   #NET "jc<10>"         LOC=E6 | IOSTANDARD=LVCMOS33; #IO_L5P_T0_AD13P_35
118
119   ## Pmod Header JD
120   #NET "jd<1>"          LOC=H4 | IOSTANDARD=LVCMOS33; #IO_L21N_T3_DQS_35
121   #NET "jd<2>"          LOC=H1 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_35
122   #NET "jd<3>"          LOC=G1 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_35
123   #NET "jd<4>"          LOC=G3 | IOSTANDARD=LVCMOS33; #IO_L20N_T3_35
124   #NET "jd<7>"          LOC=H2 | IOSTANDARD=LVCMOS33; #IO_L15P_T2_DQS_35
125   #NET "jd<8>"          LOC=G4 | IOSTANDARD=LVCMOS33; #IO_L20P_T3_35
126   #NET "jd<9>"          LOC=G2 | IOSTANDARD=LVCMOS33; #IO_L15N_T2_DQS_35
127   #NET "jd<10>"         LOC=F3 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_35
128
129   ##Pmod Header JXADC
130   #NET "xa_n<1>"        LOC=A14 | IOSTANDARD=LVDS; #IO_L9N_T1_DQS_AD3N_15
131   #NET "xa_p<1>"        LOC=A13 | IOSTANDARD=LVDS; #IO_L9P_T1_DQS_AD3P_15
132   #NET "xa_n<2>"        LOC=A16 | IOSTANDARD=LVDS; #IO_L8N_T1_AD10N_15
133   #NET "xa_p<2>"        LOC=A15 | IOSTANDARD=LVDS; #IO_L8P_T1_AD10P_15
134   #NET "xa_n<3>"        LOC=B17 | IOSTANDARD=LVDS; #IO_L7N_T1_AD2N_15
135   #NET "xa_p<3>"        LOC=B16 | IOSTANDARD=LVDS; #IO_L7P_T1_AD2P_15
136   #NET "xa_n<4>"        LOC=A18 | IOSTANDARD=LVDS; #IO_L10N_T1_AD11N_15
137   #NET "xa_p<4>"        LOC=B18 | IOSTANDARD=LVDS; #IO_L10P_T1_AD11P_15
138
139
140   ##VGA Connector
141   NET "vga_r<0>"       LOC=A3 | IOSTANDARD=LVCMOS33; #IO_L8N_T1_AD14N_35
142   NET "vga_r<1>"       LOC=B4 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_AD6N_35
143   NET "vga_r<2>"       LOC=C5 | IOSTANDARD=LVCMOS33; #IO_L1N_T0_AD4N_35
144   NET "vga_r<3>"       LOC=A4 | IOSTANDARD=LVCMOS33; #IO_L8P_T1_AD14P_35
145
146   NET "vga_g<0>"       LOC=C6 | IOSTANDARD=LVCMOS33; #IO_L1P_T0_AD4P_35
147   NET "vga_g<1>"       LOC=A5 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_AD5N_35
148   NET "vga_g<2>"       LOC=B6 | IOSTANDARD=LVCMOS33; #IO_L2N_T0_AD12N_35
149   NET "vga_g<3>"       LOC=A6 | IOSTANDARD=LVCMOS33; #IO_L3P_T0_DQS_AD5P_35
150
151   NET "vga_b<0>"       LOC=B7 | IOSTANDARD=LVCMOS33; #IO_L2P_T0_AD12P_35
152   NET "vga_b<1>"       LOC=C7 | IOSTANDARD=LVCMOS33; #IO_L4N_T0_35
153   NET "vga_b<2>"       LOC=D7 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_VREF_35
154   NET "vga_b<3>"       LOC=D8 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_35
155
156   NET "hSync"         LOC=B11 | IOSTANDARD=LVCMOS33; #IO_L4P_T0_15
157   NET "vSync"         LOC=B12 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_AD1N_15
158
159
160   ##Micro SD Connector
161   #NET "sd_sck"         LOC=B1 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_AD7P_35
162   #NET "sd_reset"       LOC=E2 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_35
163   #NET "sd_cd"          LOC=A1 | IOSTANDARD=LVCMOS33; #IO_L9N_T1_DQS_AD7N_35
164   #NET "sd_cmd"         LOC=C1 | IOSTANDARD=LVCMOS33; #IO_L16N_T2_35
165   #NET "sd_dat<0>"      LOC=C2 | IOSTANDARD=LVCMOS33; #IO_L16P_T2_35
166   #NET "sd_dat<1>"      LOC=E1 | IOSTANDARD=LVCMOS33; #IO_L18N_T2_35
167   #NET "sd_dat<2>"      LOC=F1 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_35
168   #NET "sd_dat<3>"      LOC=D2 | IOSTANDARD=LVCMOS33; #IO_L14N_T2_SRCC_35
169
170
```

```
171   ##PWM Audio Amplifier
172   #NET "aud_pwm"          LOC=A11 | IOSTANDARD=LVCMOS33; #IO_L4N_T0_15
173   #NET "aud_sd"           LOC=D12 | IOSTANDARD=LVCMOS33; #IO_L6P_T0_15
174
175
176   ##Accelerometer
177   #NET "acl_miso"         LOC=E15 | IOSTANDARD=LVCMOS33; #IO_L11P_T1_SRCC_15
178   #NET "acl_mosi"         LOC=F14 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_AD9N_15
179   #NET "acl_sclk"         LOC=F15 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_15
180   #NET "acl_csn"          LOC=D15 | IOSTANDARD=LVCMOS33; #IO_L12P_T1_MRCC_15
181   #NET "acl_int<1>"       LOC=B13 | IOSTANDARD=LVCMOS33; #IO_L2P_T0_AD8P_15
182   #NET "acl_int<2>"       LOC=C16 | IOSTANDARD=LVCMOS33; #IO_L20P_T3_A20_15
183
184
185   ##Temperature Sensor
186   #NET "tmp_ct"           LOC=B14 | IOSTANDARD=LVCMOS33; #IO_L2N_T0_AD8N_15
187   #NET "tmp_int"          LOC=D13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_VREF_15
188   #NET "tmp_scl"          LOC=C14 | IOSTANDARD=LVCMOS33; #IO_L1N_T0_AD0N_15
189   #NET "tmp_sda"          LOC=C15 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_15
190
191
192   ##USB-RS232 Interface
193   #NET "uart_cts"         LOC=D3 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_35
194   #NET "uart_rts"         LOC=E5 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_AD13N_35
195   #NET "uart_rxd_out"     LOC=D4 | IOSTANDARD=LVCMOS33; #IO_L11N_T1_SRCC_35
196   #NET "uart_txd_in"      LOC=C4 | IOSTANDARD=LVCMOS33; #IO_L7P_T1_AD6P_35
197
198
199   ##Omnidirectional Microphone
200   #NET "m_clk"            LOC=J5 | IOSTANDARD=LVCMOS33; #IO_25_35
201   #NET "m_data"           LOC=H5 | IOSTANDARD=LVCMOS33; #IO_L24N_T3_35
202   #NET "m_lrsel"          LOC=F5 | IOSTANDARD=LVCMOS33; #IO_0_35
203
204
205   ##USB HID (PS/2)
206   #NET "ps2_clk"          LOC=F4 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_35
207   #NET "ps2_data"         LOC=B2 | IOSTANDARD=LVCMOS33; #IO_L10N_T1_AD15N_35
208
209
210   ##Quad SPI Flash
211   #NET "qspi_csn"         LOC=L13 | IOSTANDARD=LVCMOS33; #IO_L6P_T0_FCS_B_14
212   #NET "qspi_dq<0>"       LOC=K17 | IOSTANDARD=LVCMOS33; #IO_L1P_T0_D00_MOSI_14
213   #NET "qspi_dq<1>"       LOC=K18 | IOSTANDARD=LVCMOS33; #IO_L1N_T0_D01_DIN_14
214   #NET "qspi_dq<2>"       LOC=L14 | IOSTANDARD=LVCMOS33; #IO_L2P_T0_D02_14
215   #NET "qspi_dq<3>"       LOC=M14 | IOSTANDARD=LVCMOS33; #IO_L2N_T0_D03_14
216
217
218   ##SMSC Ethernet PHY
219   #NET "eth_rxd<0>"       LOC=C11 | IOSTANDARD=LVCMOS33; #IO_L13P_T2_MRCC_16
220   #NET "eth_rxd<1>"       LOC=D10 | IOSTANDARD=LVCMOS33; #IO_L19N_T3_VREF_16
221   #NET "eth_txd<0>"       LOC=A10 | IOSTANDARD=LVCMOS33; #IO_L14P_T2_SRCC_16
222   #NET "eth_txd<1>"       LOC=A8 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_16
223   #NET "eth_crsdv"        LOC=D9 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_VREF_16
224   #NET "eth_intn"         LOC=B8 | IOSTANDARD=LVCMOS33; #IO_L12P_T1_MRCC_16
225   #NET "eth_mdc"          LOC=C9 | IOSTANDARD=LVCMOS33; #IO_L11P_T1_SRCC_16
226   #NET "eth_mdio"         LOC=A9 | IOSTANDARD=LVCMOS33; #IO_L14N_T2_SRCC_16
227   #NET "eth_refclk"       LOC=D5 | IOSTANDARD=LVCMOS33; #IO_L11P_T1_SRCC_35
```

```
228   #NET "eth_rstn"        LOC=B3 | IOSTANDARD=LVCMOS33; #IO_L10P_T1_AD15P_35
229   #NET "eth_txen"        LOC=B9 | IOSTANDARD=LVCMOS33; #IO_L11N_T1_SRCC_16
230   #NET "eth_rxerr"       LOC=C10 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_16
231
```