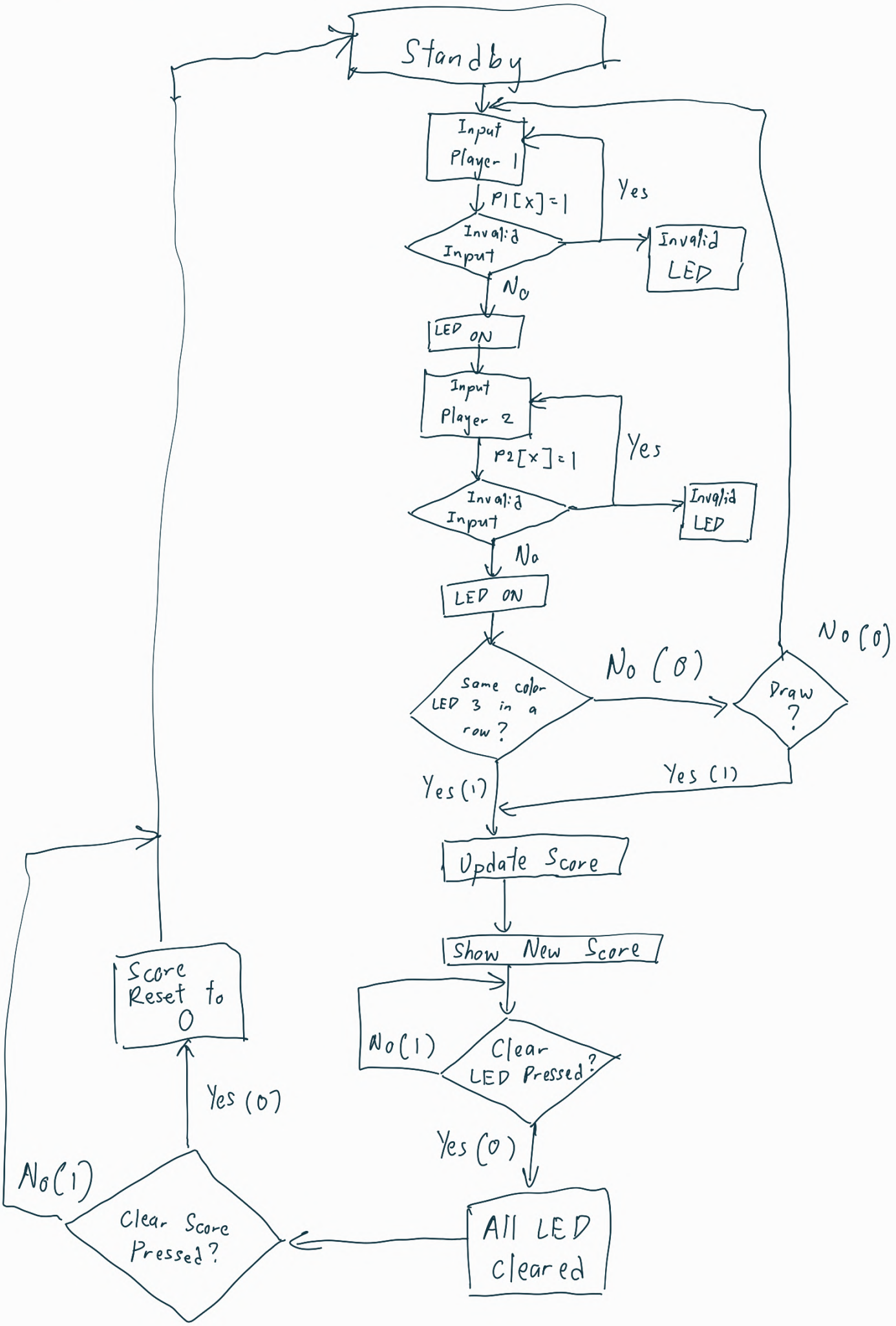
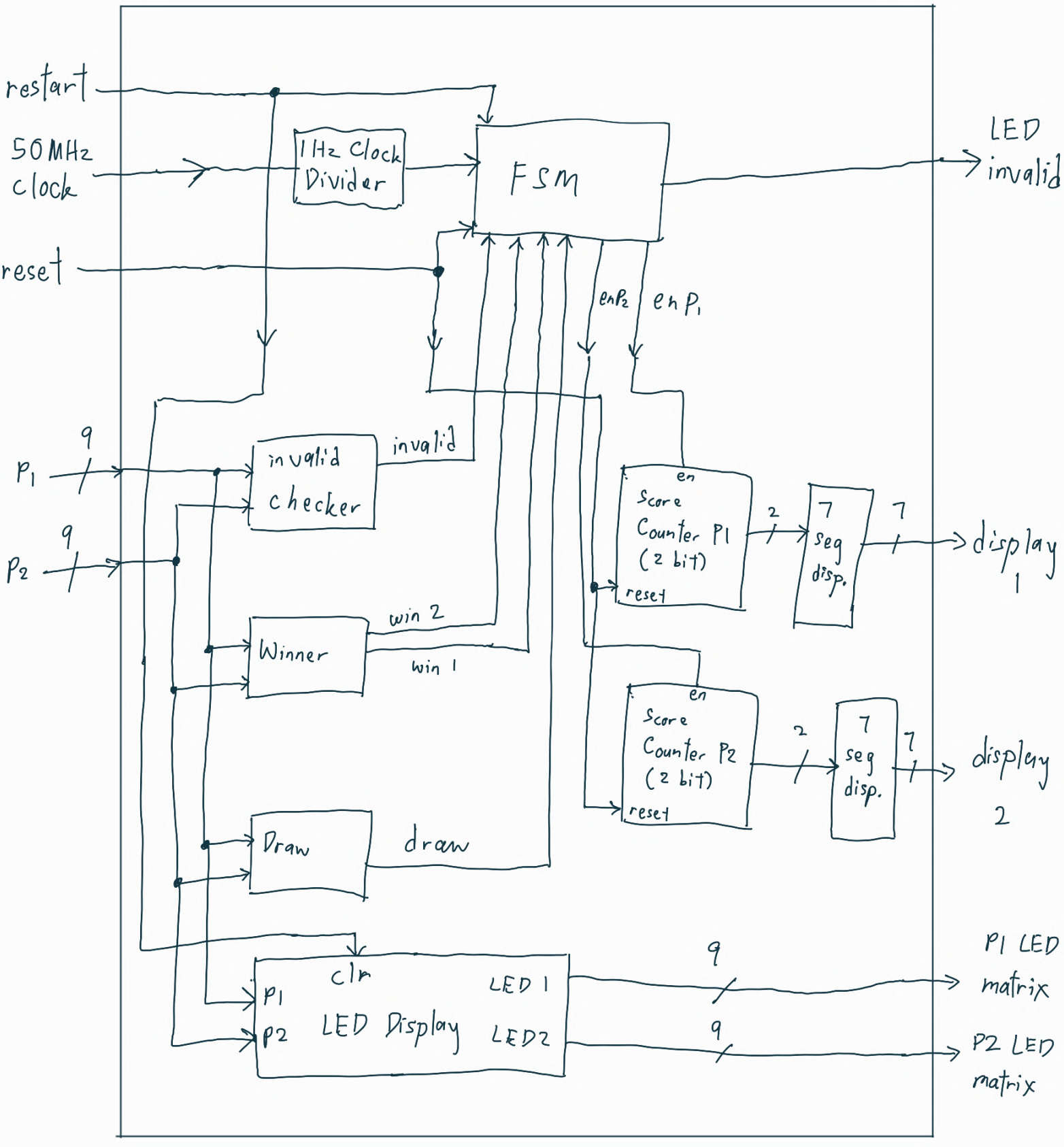
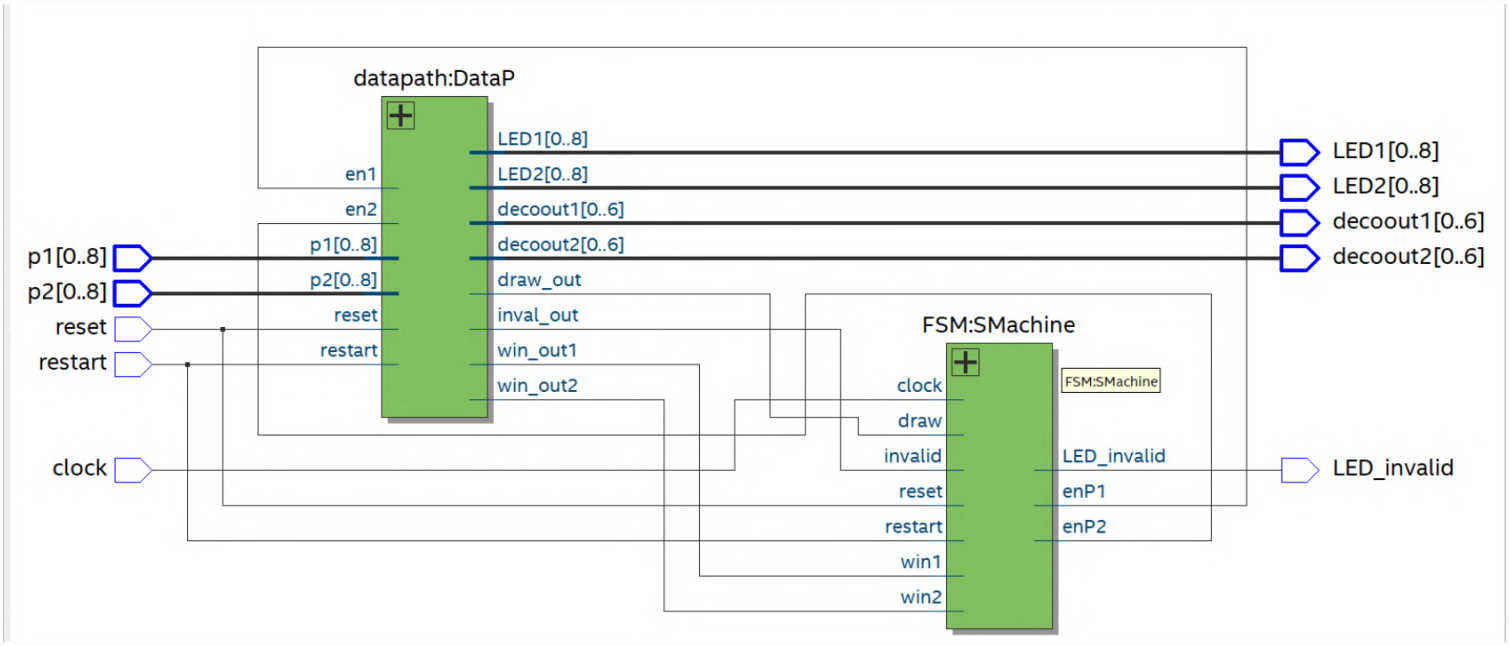
3) Flow Chart


Standby
Input Player 1
P1[x]=1
Yes
Invalid Input
Invalid LED
No
LED ON
Input Player 2
P2[x]=1
Yes
Invalid Input
Invalid LED
No
LED ON
Same color LED 3 in a row?
No (0)
Draw ?
No (0)
Yes (1)
Yes (1)
Update Score
Show New Score
No(1)
Clear LED Pressed?
Yes (0)
All LED cleared
Clear Score Pressed?
No(1)
Yes (0)
Score Reset to 0

Overall Circuit Diagram:



RTL View Overall Circuit Diagram



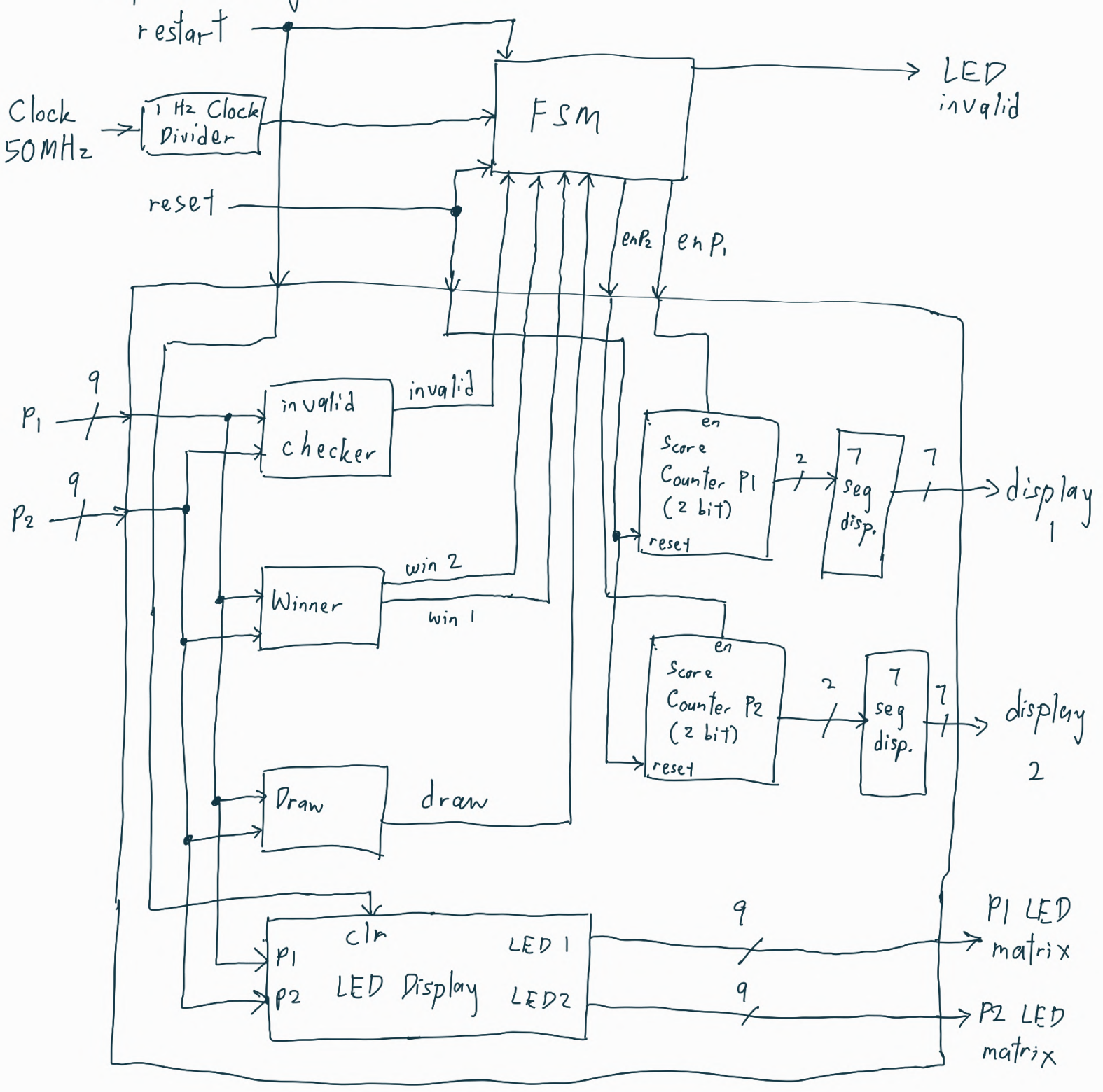
Overall coding:

```
1  module TTC(LED1, LED2, decoout1, decoout2, LED_invalid, p1, p2, restart, reset, clock);
2  input [0:8] p1, p2;
3  input restart, reset, clock;
4  output [0:8] LED1, LED2;
5  output [0:6] decoout1, decoout2;
6  output LED_invalid;
7  wire win_out1, win_out2, inval_out, draw_out, enP1, enP2;
8
9  datapath DataP (LED1, LED2, decoout1, decoout2, win_out1, win_out2, inval_out, draw_out, p1, p2, enP1, enP2, reset, restart);
10 FSM SMachine (clock, restart, reset, inval_out, win_out1, win_out2, draw_out, enP1, enP2, LED_invalid);
11
12 endmodule
```

Clock Divider Submodule Coding:

```
1  module Clk_Div(clk_out,clk);
2  input clk;
3  output reg clk_out;
4
5  reg[24:0] count;
6
7  always @(posedge clk) begin
8
9     if (count == 25000000) begin
10       clk_out <= ~clk_out;
11       count <= 0;
12    end else
13       count <= count+1;
14
15 end
16
17 endmodule
```

datapath diagram:



Verilog Coding for Datapath Top Module:

```v
module datapath (LED1, LED2, decoout1, decoout2, win_out1, win_out2, inval_out, draw_out, p1, p2, en1, en2, reset, restart);
input [0:8] p1, p2;
input en1, en2, reset, restart;
output [0:8] LED1, LED2;
output [0:6] decoout1, decoout2;
output inval_out, draw_out, win_out1, win_out2;
wire [1:0] counter_out1, counter_out2;

invalid_check invalchecker (p1, p2, inval_out);
winner_case win (p1, p2, win_out1, win_out2);
Draw drawcheck (draw_out, p1, p2);
Score_Counter SCount1 (counter_out1, en1, reset);
Score_Counter SCount2 (counter_out2, en2, reset);
decoder2to7 dec27_1 (decoout1, counter_out1);
decoder2to7 dec27_2 (decoout2, counter_out2);
LED_Disp LED_OP (p1, p2, restart, LED1, LED2);

endmodule
```

invalid checker submodule coding

```v
module invalid_check(p1,p2,s);
    input [0:8] p1,p2;
    output reg s;

    always @* begin
        if(p1[0]==1'b1 && p2[0]==1'b1) s = 1'b1; //invalid input in position [0]
        else if(p1[1]==1'b1 && p2[1]==1'b1) s = 1'b1; //invalid input in position [1]
        else if(p1[2]==1'b1 && p2[2]==1'b1) s = 1'b1; //invalid input in position [2]
        else if(p1[3]==1'b1 && p2[3]==1'b1) s = 1'b1; //invalid input in position [3]
        else if(p1[4]==1'b1 && p2[4]==1'b1) s = 1'b1; //invalid input in position [4]
        else if(p1[5]==1'b1 && p2[5]==1'b1) s = 1'b1; //invalid input in position [5]
        else if(p1[6]==1'b1 && p2[6]==1'b1) s = 1'b1; //invalid input in position [6]
        else if(p1[7]==1'b1 && p2[7]==1'b1) s = 1'b1; //invalid input in position [7]
        else if(p1[8]==1'b1 && p2[8]==1'b1) s = 1'b1; //invalid input in position [8]
        else s = 1'b0; //for p1[x] != p2[x] & p1[x]=0 and p2[x]=0
   end

endmodule
```

Winner submodule Coding:

```v
module winner_case(p1,p2,s1,s2);//p1=player1, p2=player2, s=result
        input[0:8] p1,p2;//9 position, 0-2 from top left to right, 3-5 from middle left to right, 6-8 from bottom left to right
        output reg s1,s2;

        always @(*)
        begin

        //start with 00, so winner not decided before the game

        casex(p1)//case for player 1
              9'b111xxxxxx: s1 = 1'b1;//rows
              9'bxxx111xxx: s1 = 1'b1;//rows
              9'bxxxxxx111: s1 = 1'b1;//rows
              9'bxx1xx1xx1: s1 = 1'b1;//columns
              9'bx1xx1xx1x: s1 = 1'b1;//columns
              9'b1xx1xx1xx: s1 = 1'b1;//columns
              9'bxx1x1x1xx: s1 = 1'b1;//diagonals
              9'b1xxx1xxx1: s1 = 1'b1;//diagonals
              default: s1 = 1'b0;
        endcase

        casex(p2)//case for player 2
              9'b111xxxxxx: s2 = 1'b1;//rows
              9'bxxx111xxx: s2 = 1'b1;//rows
              9'bxxxxxx111: s2 = 1'b1;//rows
              9'bxx1xx1xx1: s2 = 1'b1;//columns
              9'bx1xx1xx1x: s2 = 1'b1;//columns
              9'b1xx1xx1xx: s2 = 1'b1;//columns
              9'bxx1x1x1xx: s2 = 1'b1;//diagonals
              9'b1xxx1xxx1: s2 = 1'b1;//diagonals
              default: s2 = 1'b0;
        endcase
      end

endmodule
```

Verilog Coding for Draw Submodule:

Draw.v | Compilation Report - Draw

```
module Draw (draw,p1,p2);
input [0:8] p1, p2;
output reg draw;

always @(draw)
   if (
         (p1[0] ^ p2[0]) &
         (p1[1] ^ p2[1]) &
         (p1[2] ^ p2[2]) &
         (p1[3] ^ p2[3]) &
         (p1[4] ^ p2[4]) &
         (p1[5] ^ p2[5]) &
         (p1[6] ^ p2[6]) &
         (p1[7] ^ p2[7]) &
         (p1[8] ^ p2[8])
      )
      draw <= 1;
   else
      draw <=0;

endmodule
```

Score Counter Submodule Coding:

Score_Counter.v* | Compilation Report - Score_Counter

```
module Score_Counter (score,en,reset);
input en, reset;
output reg [1:0] score;

always@(negedge reset or posedge en)

   if (reset==0)
      score <= 2'b00;
   else if (en==1)
      score <= score + 2'b01;

endmodule
```

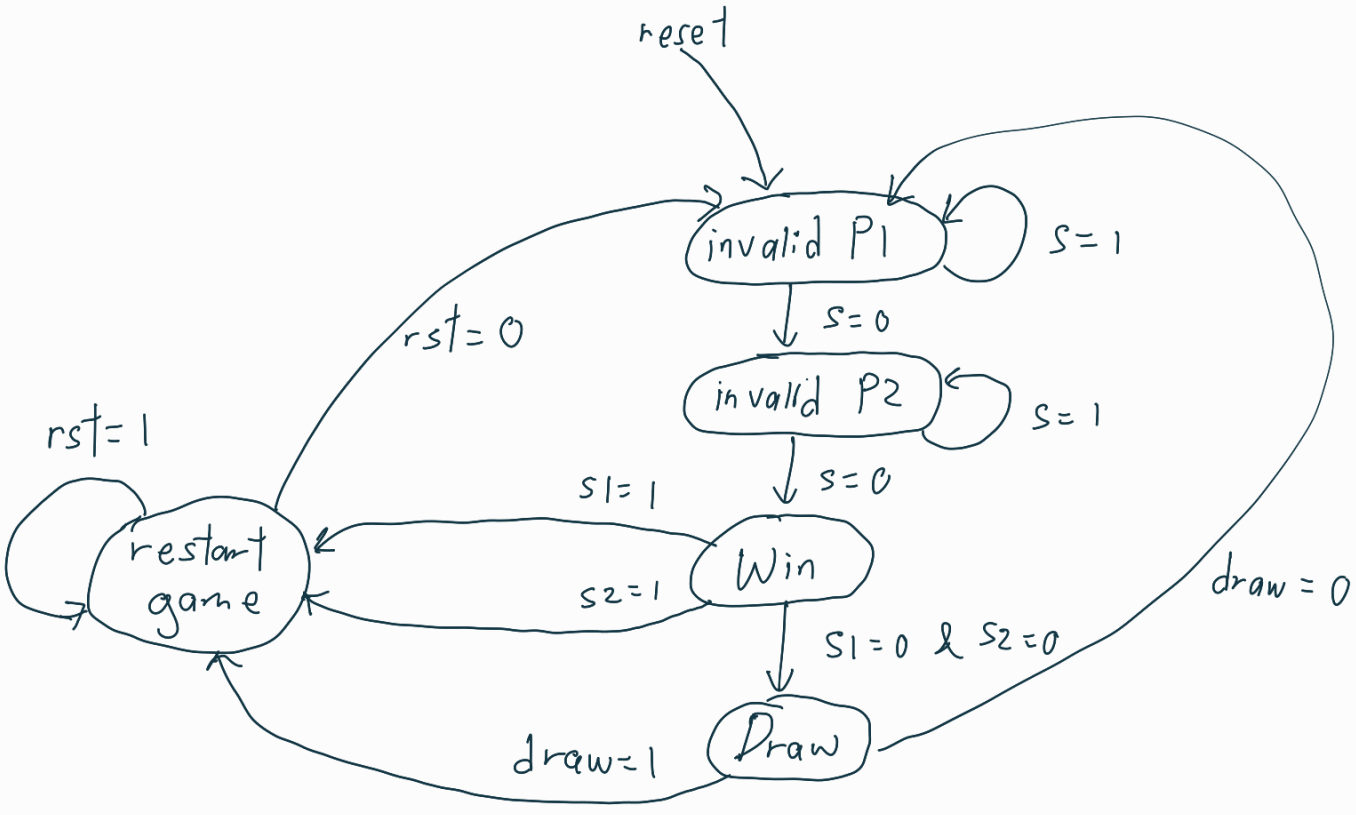
7 Seg Decoder Sub module Coding:

```
module decoder2to7 (d0,m);//d0=display, m=input
input[1:0] m;
output reg[0:6] d0;

   always @ *

      case (m)
         2'b00: d0=7'b0000001;//0
         2'b01: d0=7'b1001111;//1
         2'b10: d0=7'b0010010;//2
         2'b11: d0=7'b0000110;//3
      endcase

endmodule
```

LED Display Submodule Coding:

```
1   module LED_Disp (P1, P2, clr, LED1, LED2);
2   input [0:8] P1, P2;
3   input clr;
4   output [0:8] LED1, LED2;
5
6   /* LED Position:
7   [0] [1] [2]
8   [3] [4] [5]
9   [6] [7] [8]*/
10
11  assign LED1 = (clr == 0) ? 9'b0 : P1;
12  assign LED2 = (clr == 0) ? 9'b0 : P2;
13
14  endmodule
```

FSM State Diagram :



Next State Table:

Present State	invalid	win 1	win 2	draw	restart	Next State	LED invalid	enP1	enP2
invalid P1	1	0	0	0	0	invalid P1	1	0	0
	0	0	0	0	0	invalid P2	0	0	0
invalid P2	1	0	0	0	0	invalid P2	1	0	0
	0	0	0	0	0	Win	0	0	0
Win	0	1	0	0	0	restart game	0	1	0
	0	0	1	0	0		0	0	1
	0	0	0	0	0	Draw	0	0	0
Draw	0	0	0	1	0	restart game	0	0	0
	0	0	0	0	0	invalid P1	0	0	0
restart game	0	0	0	0	1	restart game	0	0	0
	0	0	0	0	0	invalid P1	0	0	0

State Encoding:

Present State	invalid	win 1	win 2	draw	restart	Next State	LED invalid	enP1	enP2
000	1	0	0	0	0	000	1	0	0
	0	0	0	0	0	001	0	0	0
001	1	0	0	0	0	001	1	0	0
	0	0	0	0	0	010	0	0	0
010	0	1	0	0	0	100	0	1	0
	0	0	1	0	0		0	0	1
	0	0	0	0	0	011	0	0	0
011	0	0	0	1	0	100	0	0	0
	0	0	0	0	0	000	0	0	0
100	0	0	0	0	1	100	0	0	0
	0	0	0	0	0	000	0	0	0

invalid P1: 000

invalid P2: 001

Win : 010

Draw : 011

Restart game: 100

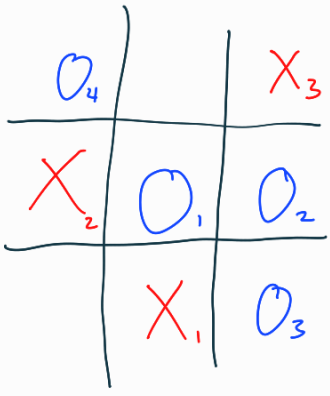
* FSM is sequential circuit as output depends on current input & past states. Clock input is required

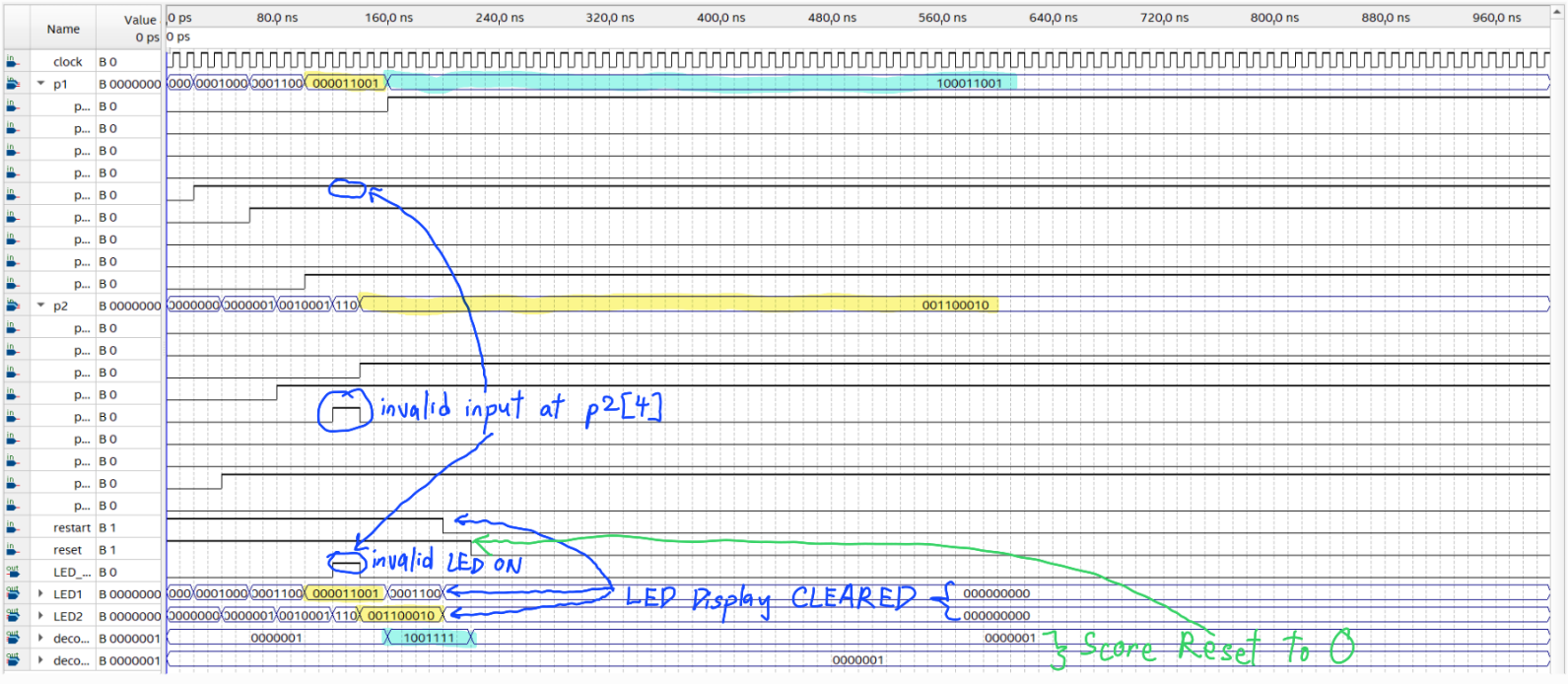
FSM Coding:

```
1  module FSM(clock, restart, reset, invalid, win1, win2, draw, enP1, enP2, LED_invalid);
2  reg [2:0] now_state,next_state;
3  input clock, restart, reset, invalid, win1, win2, draw;
4  output reg enP1, enP2, LED_invalid;
5  wire clk_out;
6
7  parameter invalid_P1 = 3'b000;
8  parameter invalid_P2 = 3'b001;
9  parameter Win = 3'b010;
10 parameter Draw = 3'b011;
11 parameter Restart_game = 3'b100;
12
13 Clk_Div Adjusted_clock (clk_out,clock);
14
15 always@(posedge clk_out or negedge reset)
16    if (reset == 0) begin
17       now_state <= invalid_P1;
18    end else
19       now_state <= next_state;
20
21 always@ (now_state, invalid, win1, win2, draw, restart)
22    case (now_state)
23    //INVALID P1
24       invalid_P1 : begin
25          if ( (invalid == 1) && (restart == 0) )
26             LED_invalid <= 1'b0;
27          else if ( (invalid == 1) && (restart == 1) ) begin
28             next_state <= invalid_P1;
29             LED_invalid <= 1'b1;
30          end else begin
31             next_state <= invalid_P2;
32             LED_invalid <= 1'b0;
33          end
34       end
35
36    //INVALID P2
37       invalid_P2 : begin
38          if ( (invalid == 1) && (restart == 0) )
39             LED_invalid <= 1'b0;
40          else if ( (invalid == 1) && (restart == 1) )begin
41             next_state <= invalid_P2;
42             LED_invalid <= 1'b1;
43          end else
44             next_state <= Win;
45             LED_invalid <= 1'b0;
46       end
47
48    //Win
49       Win: begin
50          if (win1 == 1) begin
51             next_state <= Restart_game;
52             enP1 <= 1'b1;
53          end else if (win2 == 1) begin
54             next_state <= Restart_game;
55             enP2 <= 1'b1;
56          end else begin
57             next_state <= Draw;
58             enP1 <= 1'b0;
59             enP2 <= 1'b0;
60          end
61       end
62
63    //DRAW
64       Draw: begin
65          if (draw == 1)
66             next_state <= Restart_game;
67          else
68             next_state <= invalid_P1;
69       end
70
71    //Restart game
72       Restart_game: begin
73          if (restart == 0) begin
74             next_state <= invalid_P1;
75             enP1 <= 1'b0;
76             enP2 <= 1'b0;
77          end
78          else
79             next_state <= Restart_game;
80       end
81
82       default: next_state <= invalid_P1;
83    endcase
84
85 endmodule
```

Overall System Simulation Waveform:

* Our simulation inputs is based on this game:

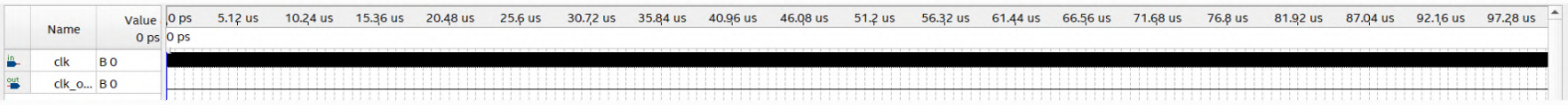




* Yellow Highlighted part show that LED Display lights up according to input by players

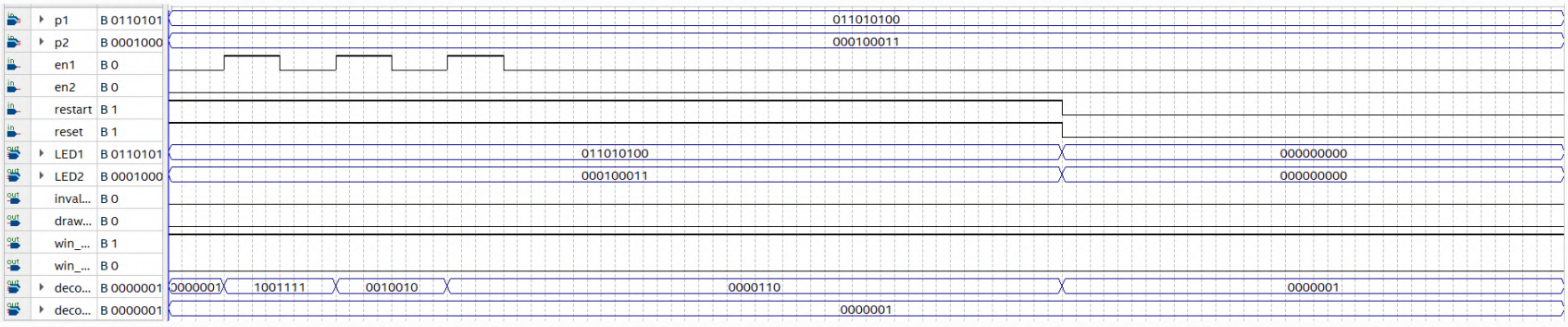
* Blue Highlighted part show that Player 1 Score increase from 0 to 1 when Player 1 Win

Clock Divider Submodule Simulation Waveform



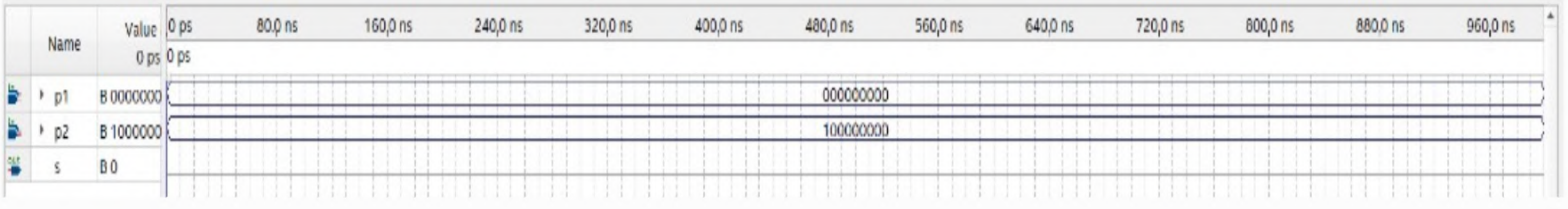
* ModelSim cannot produce period for 50MHz

Datapath Simulation Results of Tic Tac Toe Game when P1 win:

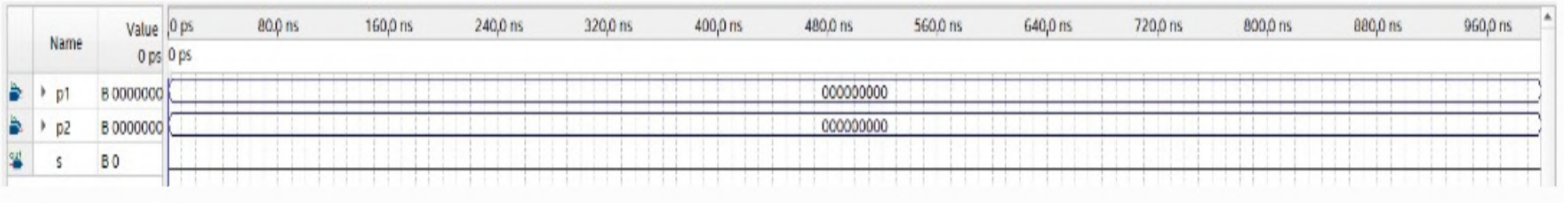


Invalid checker submodule Simulation Results:

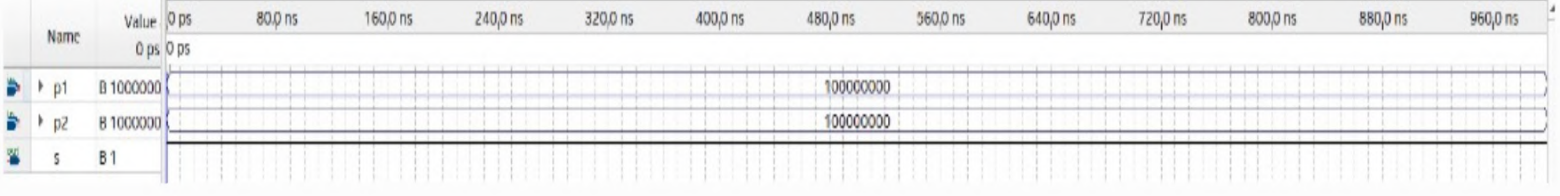
Non invalid case:



All 0 case:

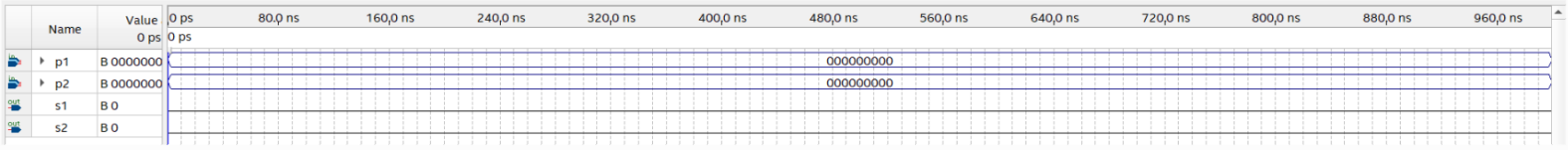


Invalid case:

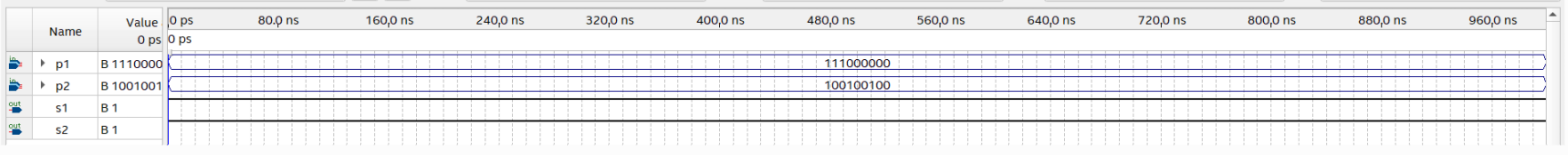


Winner Submodule Simulation Results:

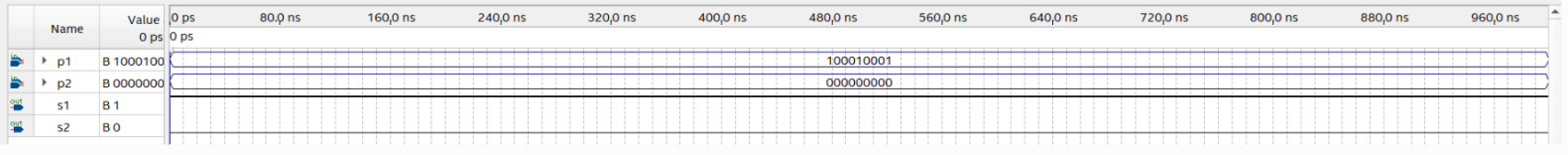
Game not started case:



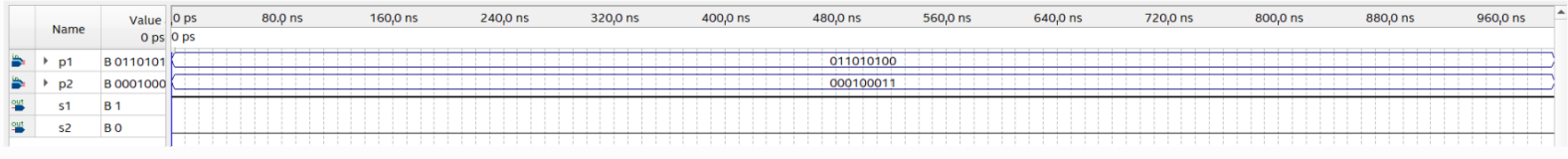
Horizontal win case (P1) & Vertical win case (P2):



Diagonal win case:

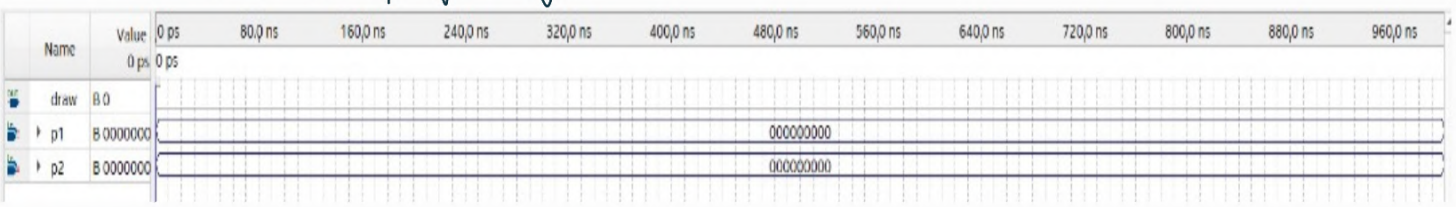


Simulation of Real Tic Tac Toe Game:

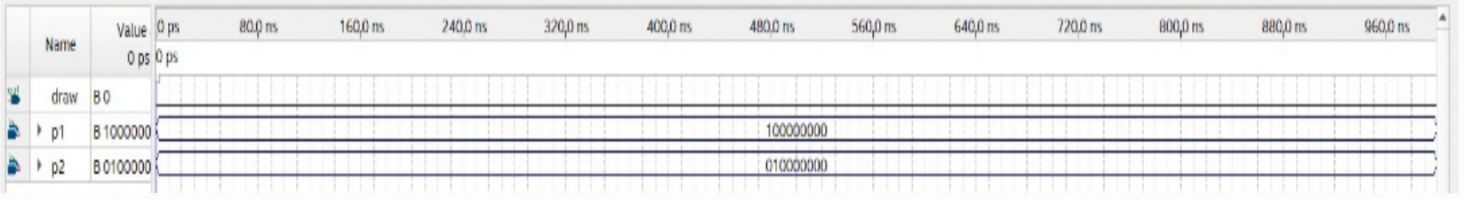


Draw Submodule Simulation Results:

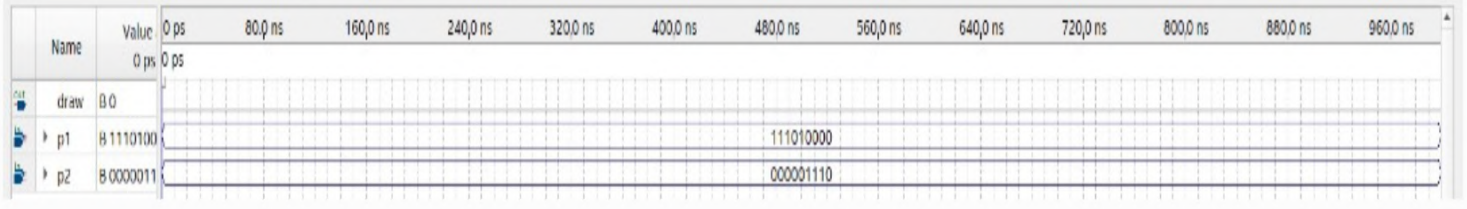
When Game not played yet:



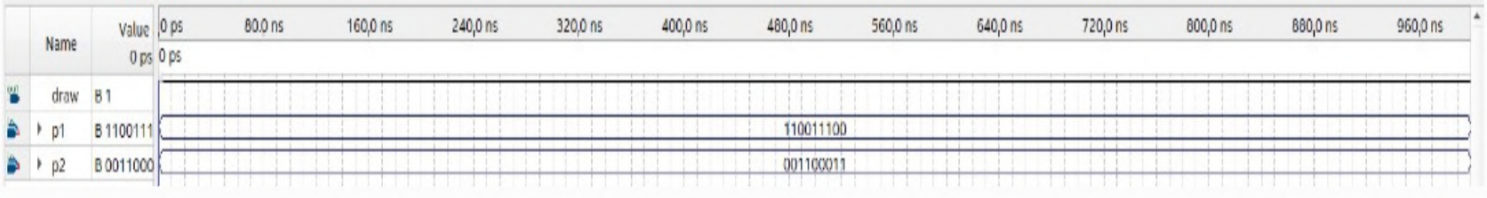
When Game not ended yet:



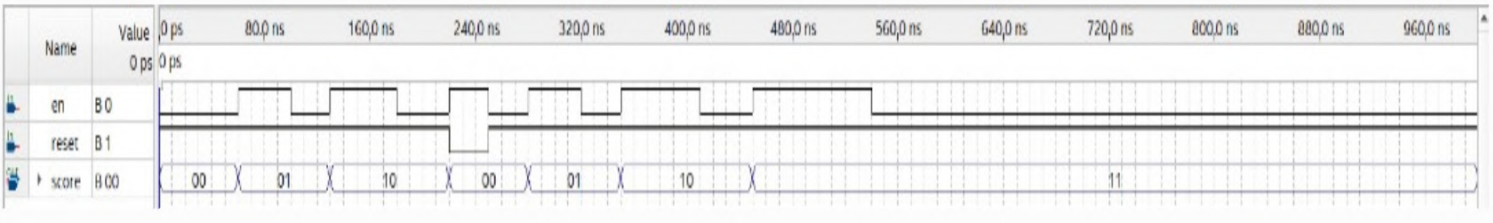
When one player (p1) wins:



When game ends in draw:

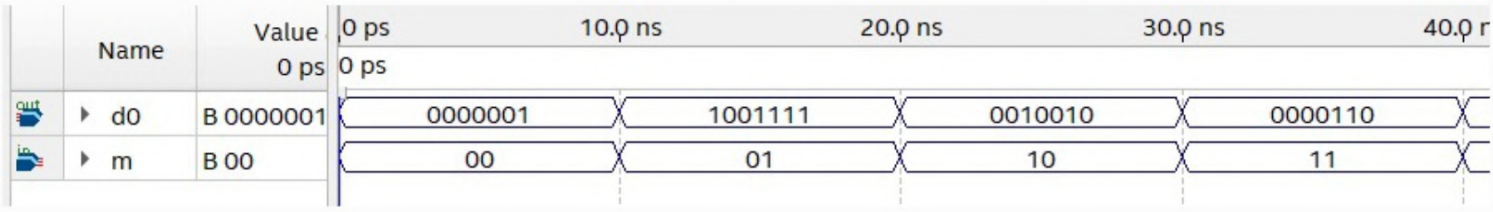


Score Counter Submodule Simulation Results:



Counter count upward from 0 when en = HIGH, resets counting back to 0 when reset = LOW.

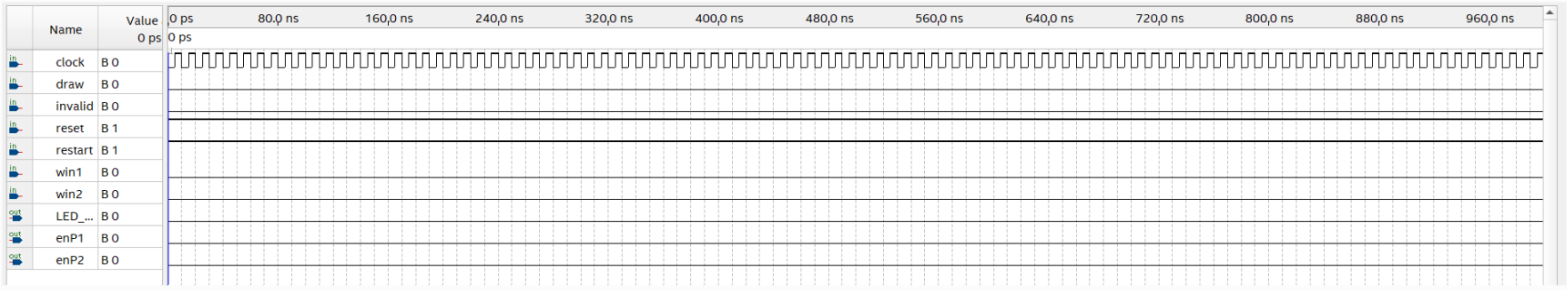
7 Seg Decoder Submodule Simulation Results:



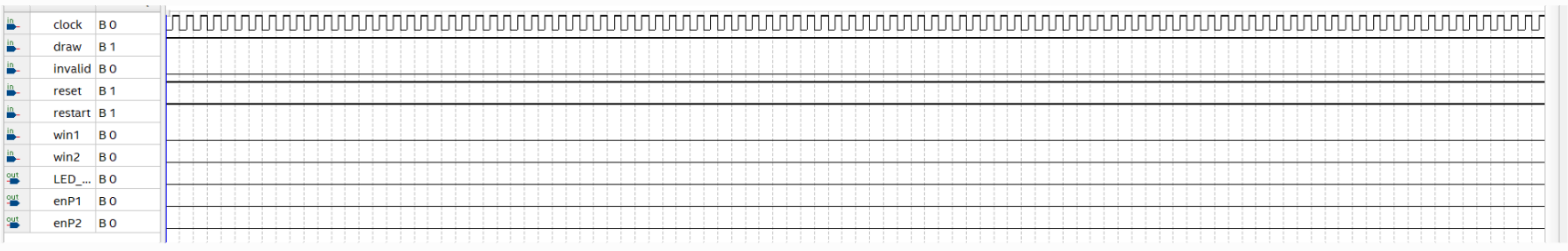
LED Display Submodule Simulation Results:

FSM Simulation Waveform:

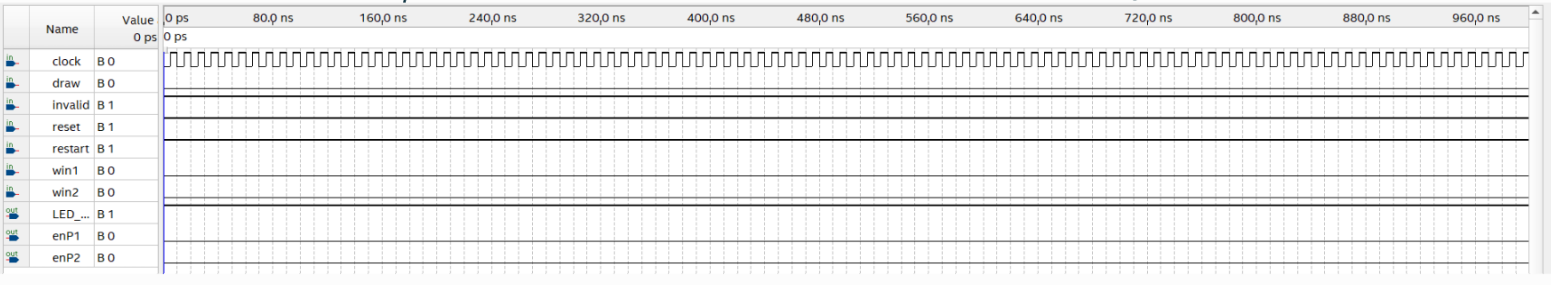
Idle State:



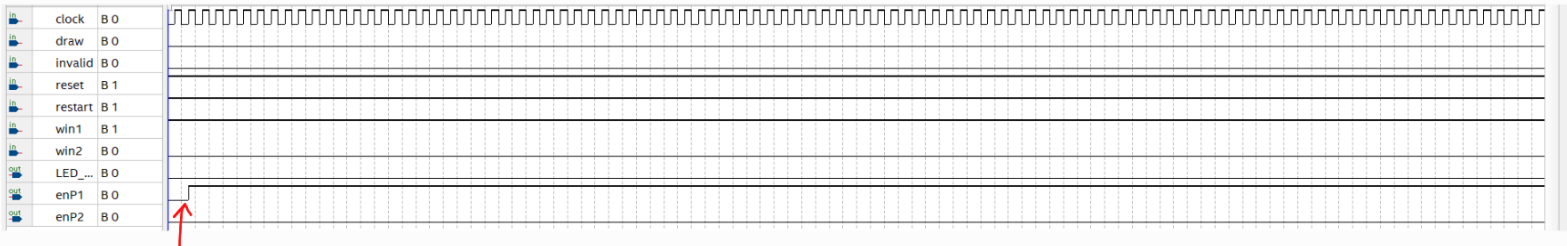
Draw State:



Invalid Input P1 / P2: (LED_invalid light up)

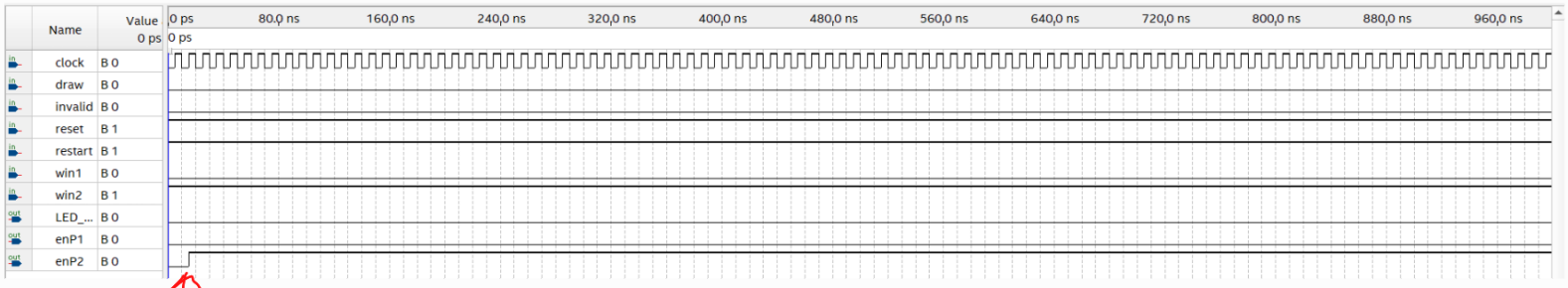


P1 Win State: (enP1 is HIGH)



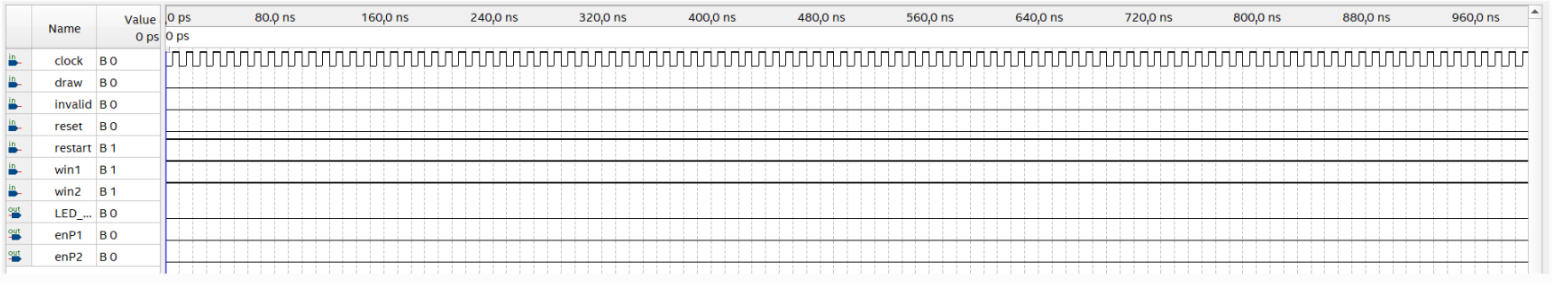
Small delay before enP1 = 1 because FSM is Mealy machine (output depend on input & present state) & is prone to false output generally found in Mealy machine although non-blocking assignment "<=" is used.

P2 Win State: (enP2 is HIGH)

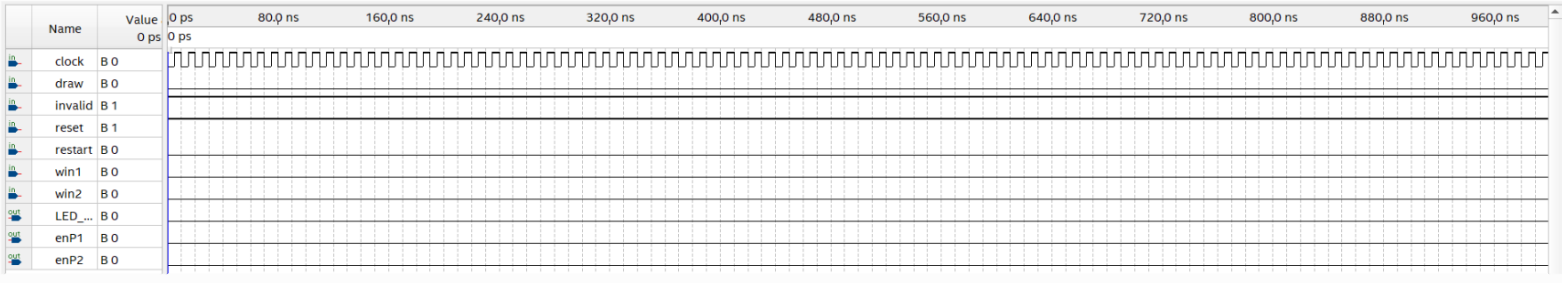


Same explanation as P1 Win State

Reset = 0 (activated) State & Overwrite Win1 & Win 2 input to Reset Score:

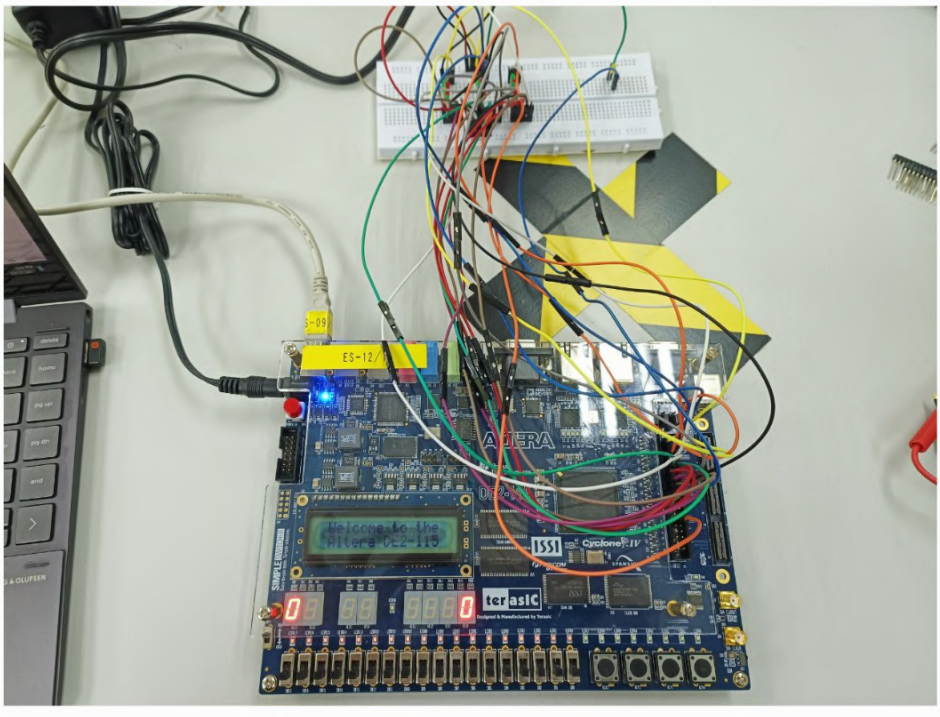


Restart = 0 (activated) & Overwrite invalid input since all LED turn off:

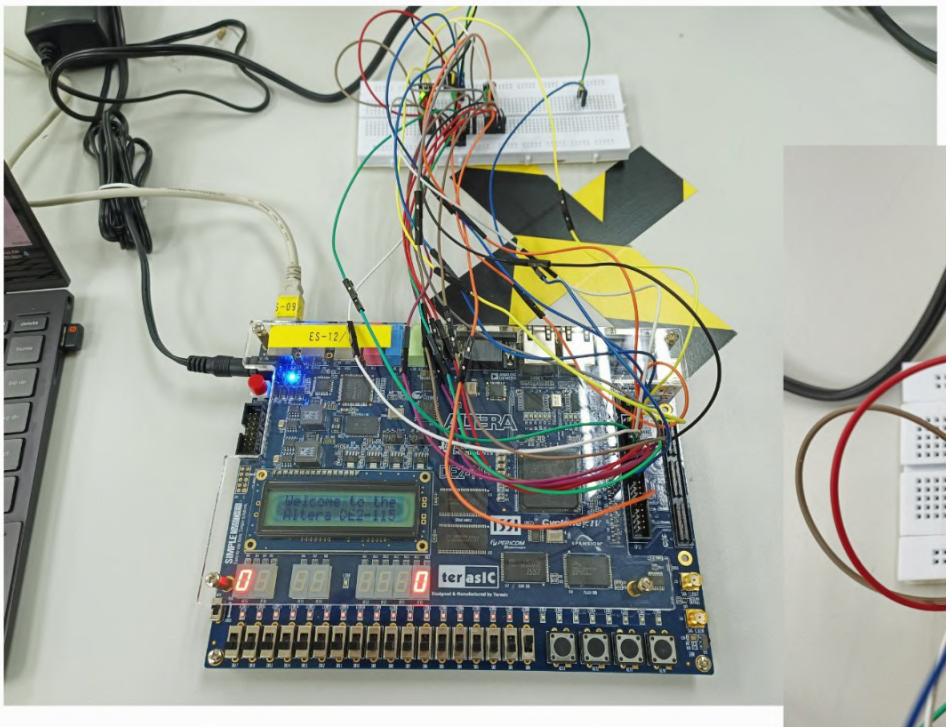


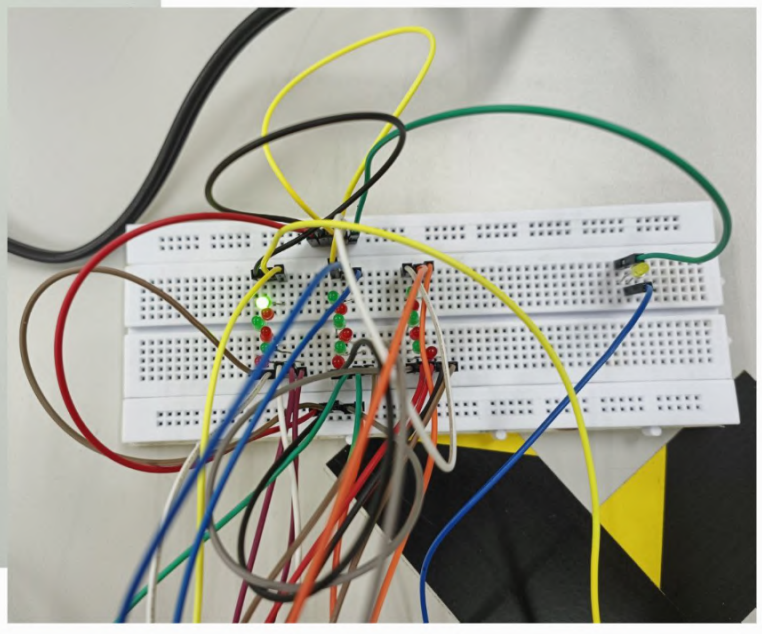
FPGA Results:

Idle State:

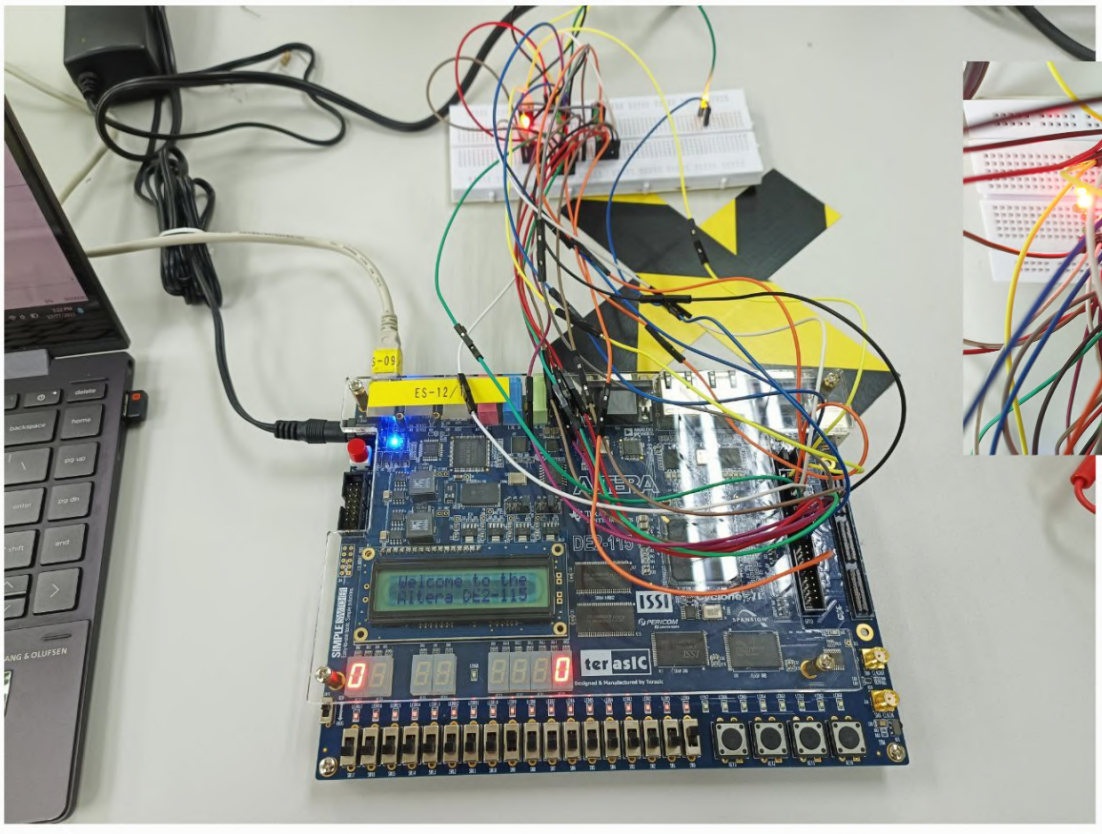


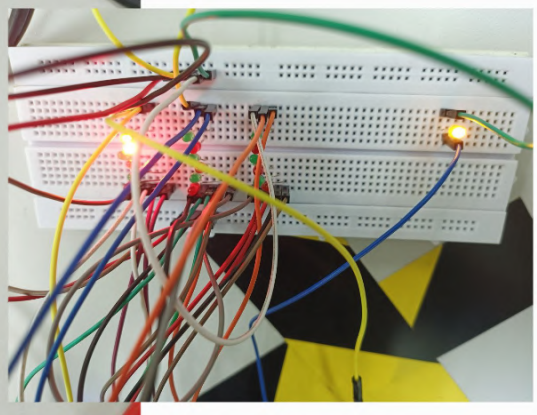
When Input P1 is entered, LED turn on:



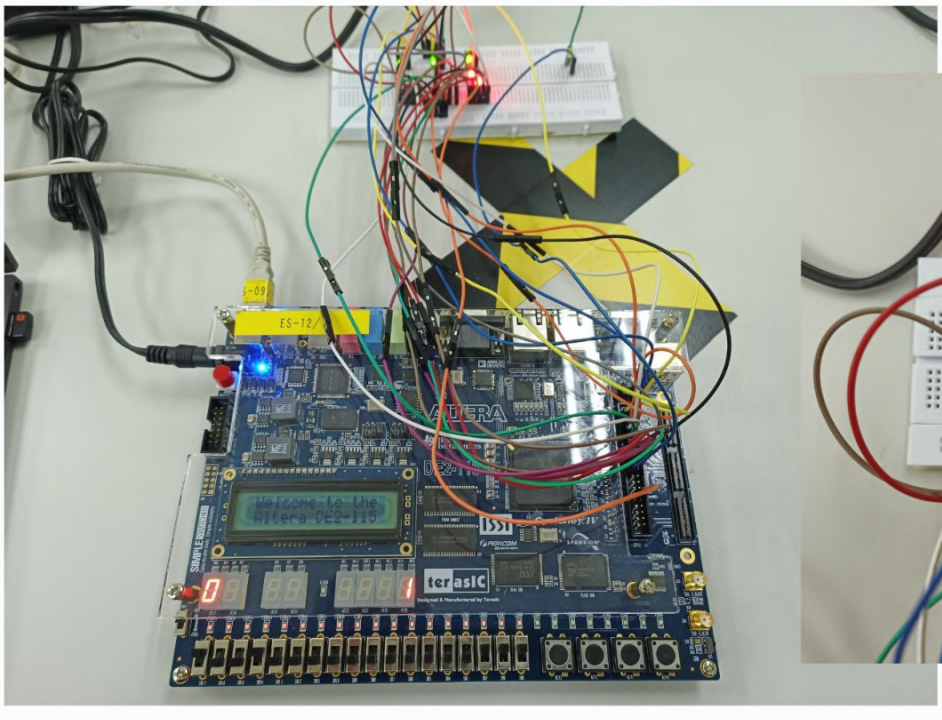


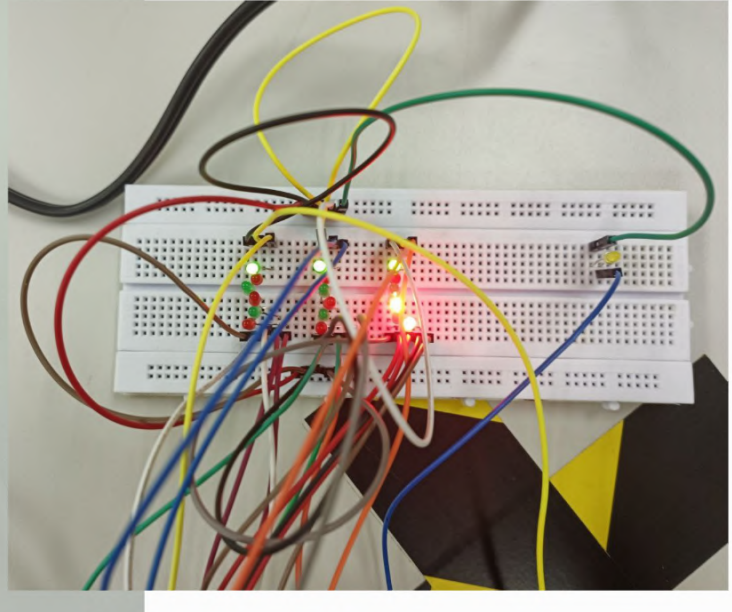
Invalid Input:



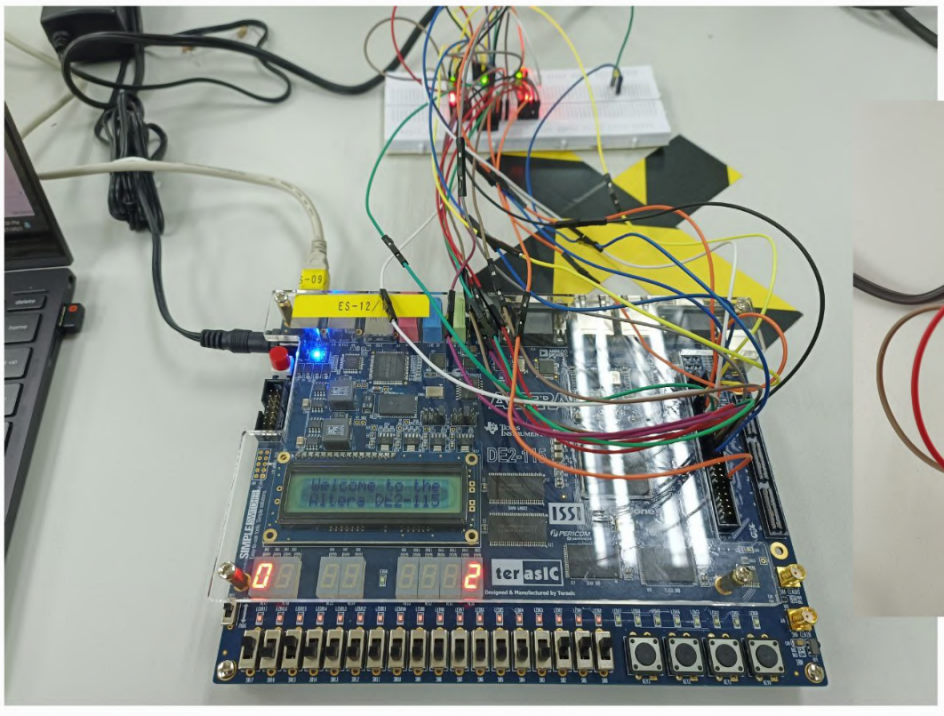


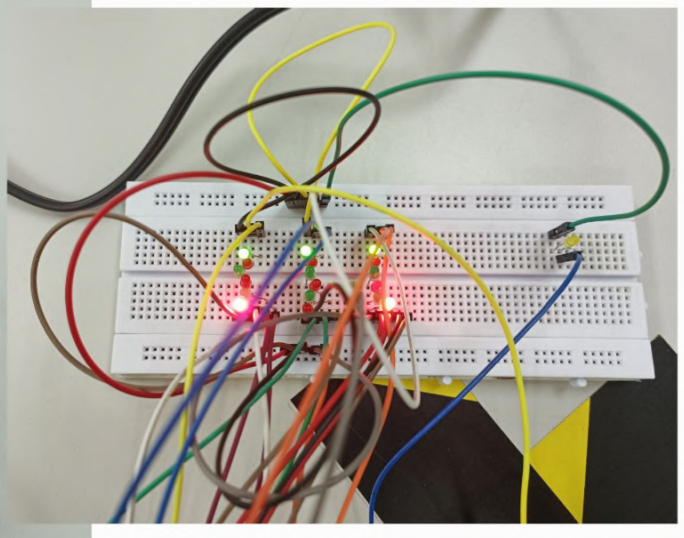
When P1 Win 1st Time:



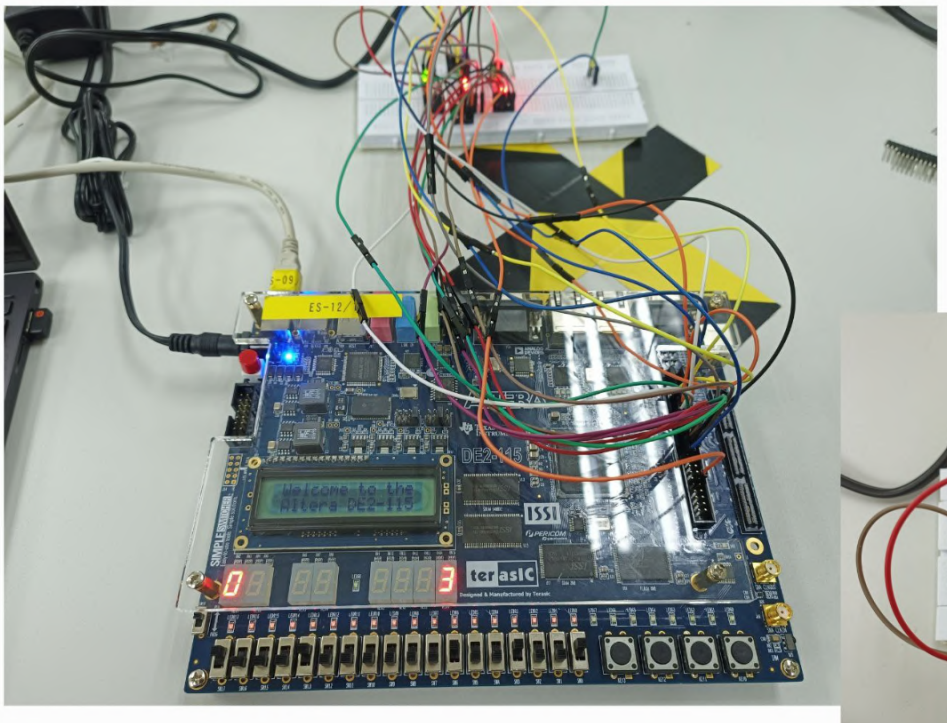


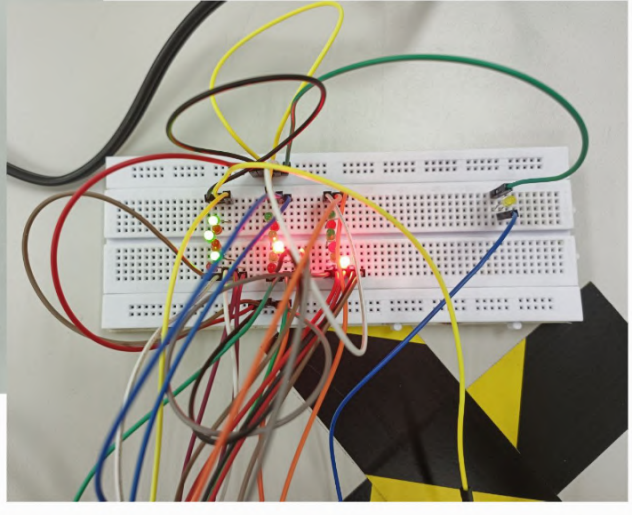
P1 Win 2nd Time:



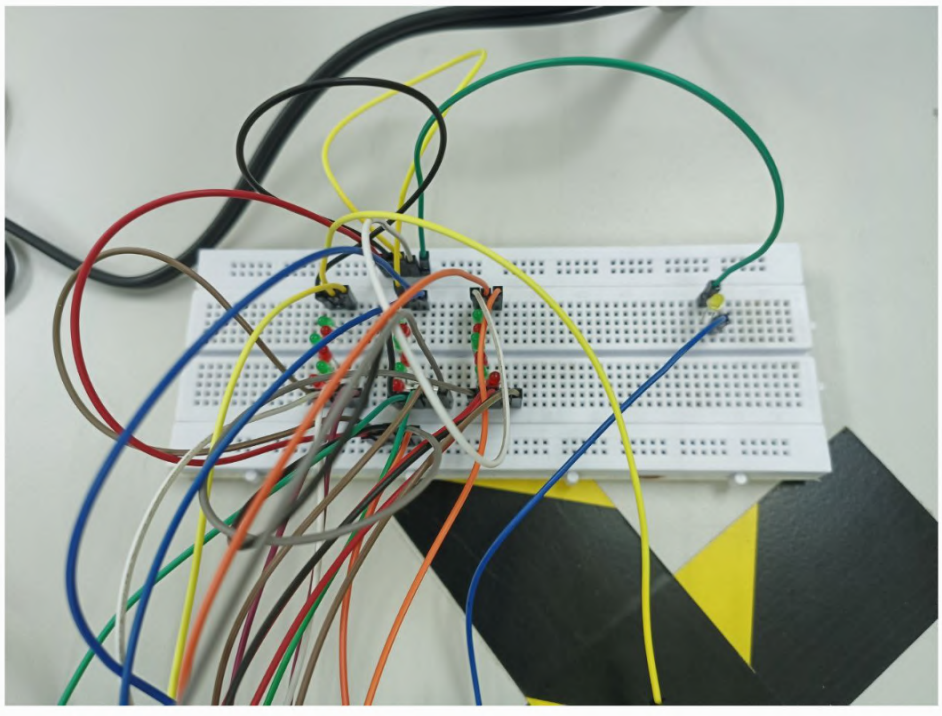


When P1 Win 3rd Time:

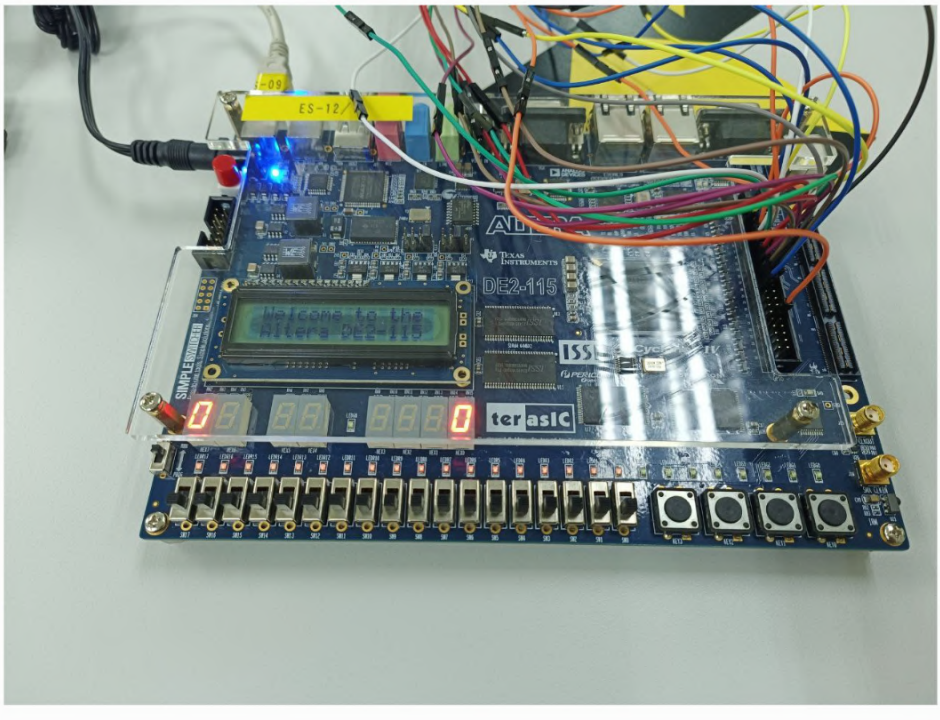




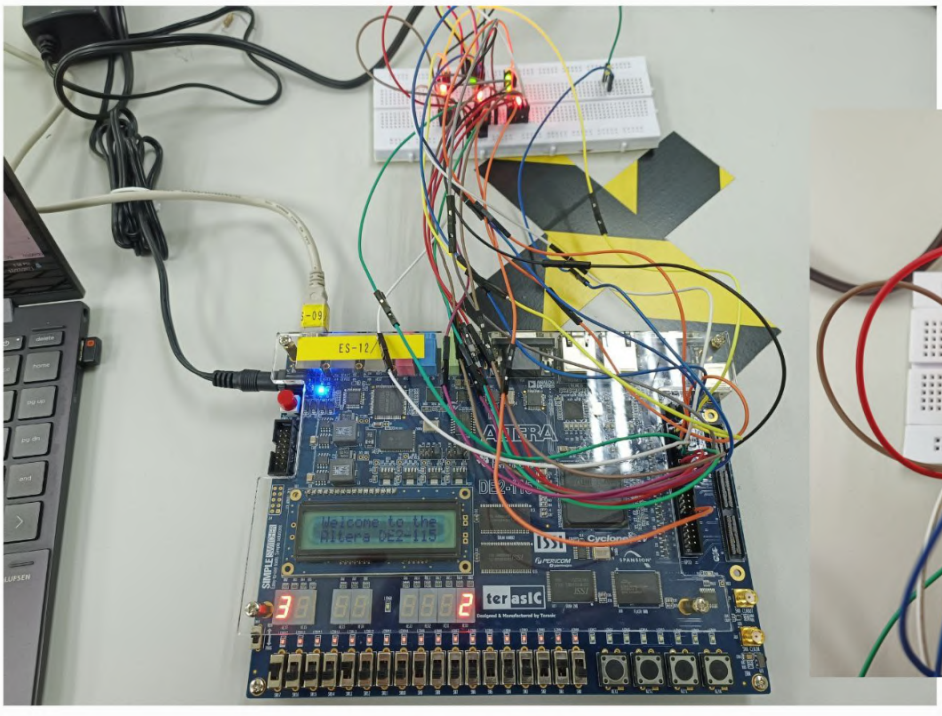
When Restart Button Pressed:

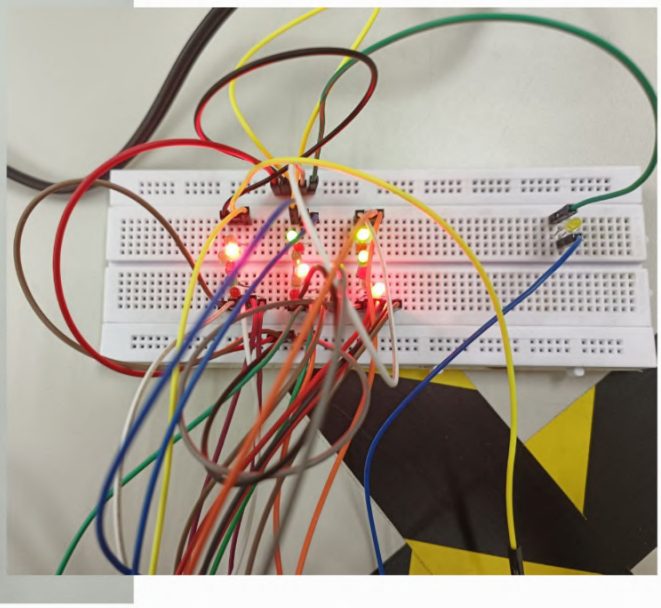


When Reset Button Pressed:



Best of 3 Game where P2 Win:





Draw: