Ain Shams University

Faculty of Engineering

Computer and Systems Engineering Department

CSE 412: Digital Verification

4th Year CSE

2nd Semester 2021/2022

Assignment 2

Submitted by:

Sarah Mohamed Ahmed Ahmed

Code: 1700593

Section: 2

Submitted to:

Dr. Ayman Wahba

Github link: https://github.com/Sarah-56/Digital-Verification/tree/main/latest

Table of Contents

Design:

Interface:

```
interface Counter_Interface #(
 parameter COUNTER_SIZE = 4
)(
   input bit clk
   );
    bit [1:0] ctrl, WHO;
   bit INIT, LOSER, WINNER, GAMEOVER, rst_l;
   bit [COUNTER_SIZE - 1:0] loadValue;

   clocking cb @(posedge clk);
     default input #0ns output #1ns;
     output rst_l, ctrl, INIT, loadValue;
     input WHO, LOSER, WINNER, GAMEOVER;
   endclocking

   modport dut(
       output GAMEOVER, WHO, LOSER, WINNER,
       input clk, rst_l, ctrl, INIT, loadValue
       );

   modport tb
   (
     clocking cb,
     output rst_l

   );
endinterface
```

Counter Module:

```
module counter #(
   parameter COUNTER_SIZE = 4          // number of bits in counter
   )(
   Counter_Interface.dut sig
   );
   /*****************************************************************
       PARAMETERS
   *****************************************************************/
   parameter cycle = 2;          // clock cycle = 2 msec.
   parameter whoValue = 2'b00;  //start value
   parameter upOne = 2'b00;     //00 count up by 1
   parameter upTwo = 2'b01;     //01 count up by 2
   parameter downOne = 2'b10;   //10 count down by 1
   parameter downTwo = 2'b11;   //11 count down by 2
    /*****************************************************************
       REGISTERS & WIRES
   *****************************************************************/
   reg LOSER;
   reg WINNER;
   reg GAMEOVER;
   reg [1:0] WHO;
   reg [1:0] ctrl;
   reg [COUNTER_SIZE - 1:0] m_counter;
    reg [COUNTER_SIZE - 1:0] loser_count;
    reg [COUNTER_SIZE - 1:0] win_count;
    wire [COUNTER_SIZE - 1:0] loadValue;
```

```
 /*********************************************************************
    ALWAYS BLOCK
*********************************************************************/
always @(posedge sig.clk) begin
    if (sig.INIT) begin
        m_counter = sig.loadValue;
        sig.WHO = whoValue;
        loser_count = 0;
        win_count = 0;
        sig.LOSER = 0;
        sig.WINNER = 0;
        sig.GAMEOVER = 0;
    end
    else begin
        /************************
        SYNCHRONOUS RESET
        ************************/
        if (sig.rst_l || sig.GAMEOVER) begin
        m_counter <= 4'b0000;
        sig.LOSER <= 0;
        sig.WINNER <= 0;
        sig.WHO <= 2'b00;
        loser_count <= 0;
        win_count <= 0;
        sig.GAMEOVER <= 0;
        end
        /************************
        INITIALIZATION
        ************************/
        else if (sig.INIT) begin
        m_counter <= sig.loadValue;
        loser_count <= 0;
        win_count <= 0;
        sig.WHO <= 2'b00;
        sig.WINNER <= 0;
        sig.LOSER <= 0;
        sig.GAMEOVER <= 0;

        end
        else begin
        case (sig.ctrl)
            upOne:  m_counter <= m_counter + 1;
            upTwo:  m_counter <= m_counter + 2;
            downOne:  m_counter <= m_counter - 1;
            downTwo:  m_counter <= m_counter - 2;
        endcase
        /*************************************************************************
            set LOSER signal to 1 for 1 clock cycle then clear it and increase
            loser counter by 1 if counter reaches 0
        *************************************************************************/
          if(m_counter == 0) begin
              sig.LOSER <= 1;
              sig.WINNER <= 0;
              loser_count = loser_count + 1;
          end
         /*********************************************************
            set WINNER signal to 1 for 1 clock cycle then clear
             it and increase
            winner counter by 1 if counter reaches 15
        *********************************************************/
```

```
            else if(m_counter == 15) begin
                sig.WINNER <= 1;
                sig.LOSER <= 0;
                win_count = win_count + 1;
            end
            else begin
                sig.LOSER <= 0;
                sig.WINNER <= 0;
            end
            // raise gameover signal if loser or winner counter reaches 15
            if(loser_count == 15 || win_count == 15) begin
                sig.GAMEOVER <= 1;
                if(loser_count == 15) sig.WHO <= 2'b01;
                else sig.WHO <= 2'b10;
            end
            end
        end
    end
endmodule
```

Test bench:

```
program tb_counter(Counter_Interface.tb sig);
    /*****************************************************************************
        PARAMETERS
    *****************************************************************************/
    parameter cycle = 2;
    parameter COUNTER_SIZE = 4;
    /*****************************************************************************
        INITIAL BLOCK
    *****************************************************************************/
    initial begin
        sig.cb.loadValue <= 4'b0000;
        sig.cb.ctrl <= 2'b00;
        sig.cb.rst_l <= 0;
        sig.cb.INIT <= 1;
        for (int ctrl_c = 0; ctrl_c <= 3; ctrl_c = ctrl_c + 1) begin
            for(int loadValue_c = 0; loadValue_c < 3; loadValue_c = loadValue_c + 1)
begin
                assertion_1: assert (sig.cb.WINNER == 0)
                    $display("WINNER = %d asserted correctly", sig.cb.WINNER);
                else
                    $fatal("WINNER = %d not asserted correctly", sig.cb.WINNER);
                sig.cb.ctrl <= ctrl_c;
                if(loadValue_c == 2) sig.cb.loadValue <= {COUNTER_SIZE{1'b1}};
                else sig.cb.loadValue <= loadValue_c;
                sig.cb.INIT <= 0;
                #2
                sig.cb.rst_l <= 0;
                #2
                sig.cb.INIT <= 1;
                #1
                sig.cb.INIT <= 0;
                #481
                sig.cb.rst_l <= 1;
            end
        end
    end
    /***************************
        Assign BLOCK
    ***************************/
```

```
   assign WHO = sig.cb.WHO;
   assign LOSER = sig.cb.LOSER;
   assign WINNER = sig.cb.WINNER;
   assign GAMEOVER = sig.cb.GAMEOVER;
   /***************************
       Properties
   ***************************/
   property signals_cleared;
     @(sig.cb) disable iff(!($fell(sig.rst_l) )) (WHO ==0 || LOSER == 0 ||
      GAMEOVER == 0 || WINNER ==0);
   endproperty

   property winner;
     @(sig.cb)
     if($fell(sig.rst_l)) ##[100:200] GAMEOVER ==1;
   endproperty

   /***************************
       Asserions
   ***************************/
   assert_winner: assert property(winner)$display("[%0t] ----- Assertion GameOver
passed", $time);
   assert_signals_cleared: assert property (signals_cleared) $display("[%0t] -----
Assertion Reseting_signals passed", $time);
endprogram
```

Assertion output:

```
WINNER = 0 asserted correctly
WINNER = 0 asserted correctly
[490] ----- Assertion Reseting_signals passed
"testbench.sv", 65: top.t0.assert_winner: started at 490ns failed at 890ns
        Offending '(GAMEOVER == 1)'
WINNER = 0 asserted correctly
[976] ----- Assertion Reseting_signals passed
"testbench.sv", 65: top.t0.assert_winner: started at 976ns failed at 1376ns
        Offending '(GAMEOVER == 1)'
WINNER = 0 asserted correctly
[1462] ----- Assertion Reseting_signals passed
[1692] ----- Assertion GameOver passed
WINNER = 0 asserted correctly
[1948] ----- Assertion Reseting_signals passed
[2192] ----- Assertion GameOver passed
WINNER = 0 asserted correctly
[2434] ----- Assertion Reseting_signals passed
[2664] ----- Assertion GameOver passed
WINNER = 0 asserted correctly
[2920] ----- Assertion Reseting_signals passed
"testbench.sv", 65: top.t0.assert_winner: started at 2920ns failed at 3320ns
        Offending '(GAMEOVER == 1)'
WINNER = 0 asserted correctly
[3406] ----- Assertion Reseting_signals passed
"testbench.sv", 65: top.t0.assert_winner: started at 3406ns failed at 3806ns
        Offending '(GAMEOVER == 1)'
WINNER = 0 asserted correctly
[3892] ----- Assertion Reseting_signals passed
"testbench.sv", 65: top.t0.assert_winner: started at 3892ns failed at 4292ns
```

```
        Offending '(GAMEOVER == 1)'
WINNER = 0 asserted correctly
[3406] ----- Assertion Reseting_signals passed
"testbench.sv", 65: top.t0.assert_winner: started at 3406ns failed at 3806ns
        Offending '(GAMEOVER == 1)'
WINNER = 0 asserted correctly
[3892] ----- Assertion Reseting_signals passed
"testbench.sv", 65: top.t0.assert_winner: started at 3892ns failed at 4292ns
        Offending '(GAMEOVER == 1)'
WINNER = 0 asserted correctly
[4378] ----- Assertion Reseting_signals passed
[4608] ----- Assertion GameOver passed
WINNER = 0 asserted correctly
[4864] ----- Assertion Reseting_signals passed
[5096] ----- Assertion GameOver passed
WINNER = 0 asserted correctly
[5350] ----- Assertion Reseting_signals passed
[5580] ----- Assertion GameOver passed
$finish at simulation time                 5832
```

Top module:

```
module top (output bit clk);
   initial clk = 1;
   initial forever #1 clk = ~clk;
   Counter_Interface iface(clk);
   tb_counter t0(iface.tb);
   counter G0(iface.dut);
   /*****************************
       DUMP VARIABLES
   *****************************/
   initial begin
       $dumpfile("wave.vcd");
       $dumpvars;
   end

endmodule
```

Output:

Control signal = 2’b00 (count up by 1)
Load value = 4’b0000

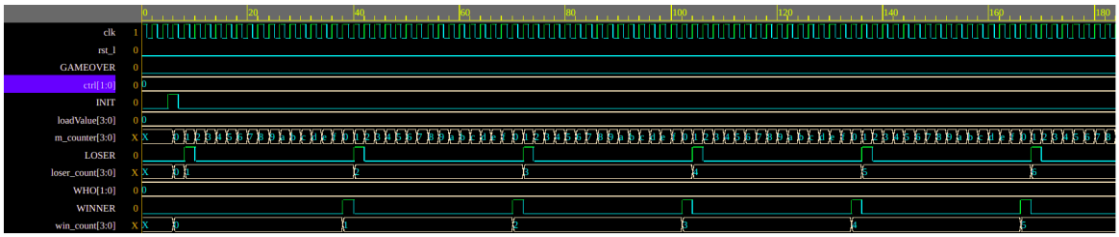

Figure 1

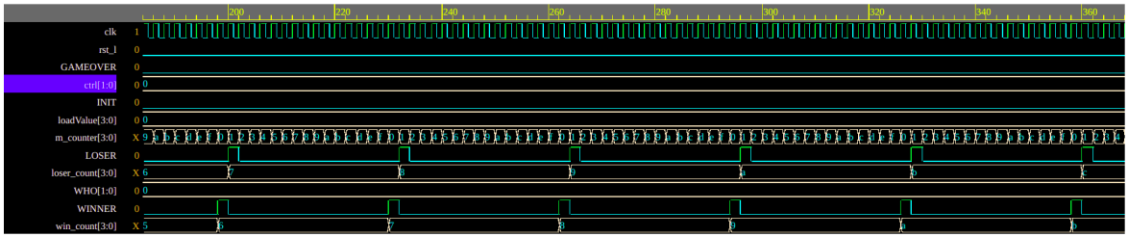

Figure 3

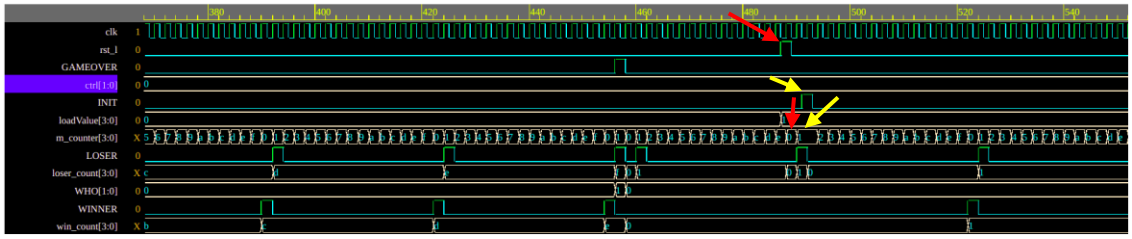

Figure 4

Control signal = 2’b00 (count up by 1)
Load value = 4’b0001

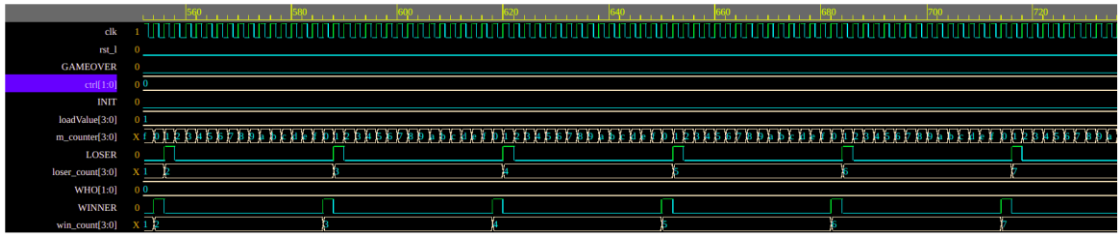

Figure 5

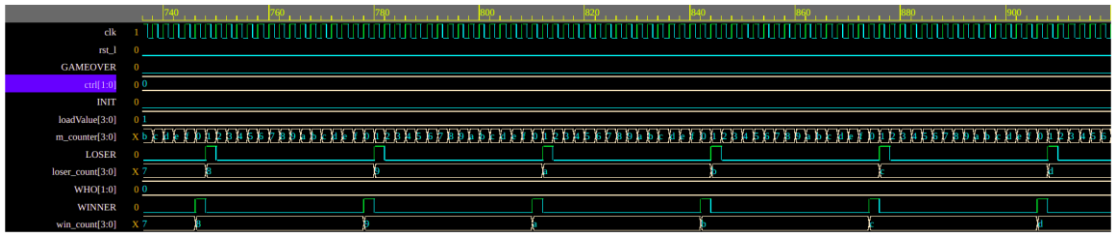

Figure 6

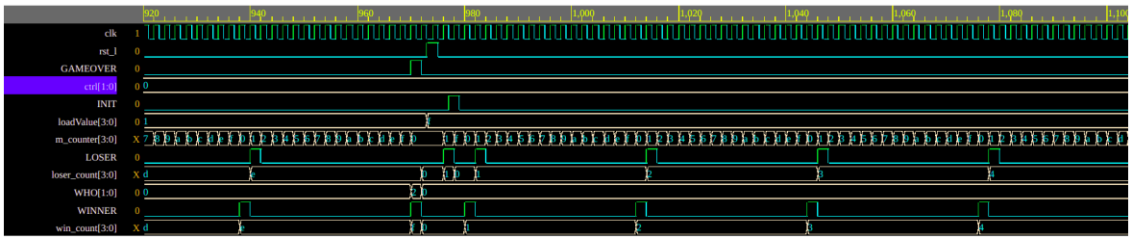

Figure 7

Control signal = 2'b00 (count up by 1)
Load value = 4'b1111

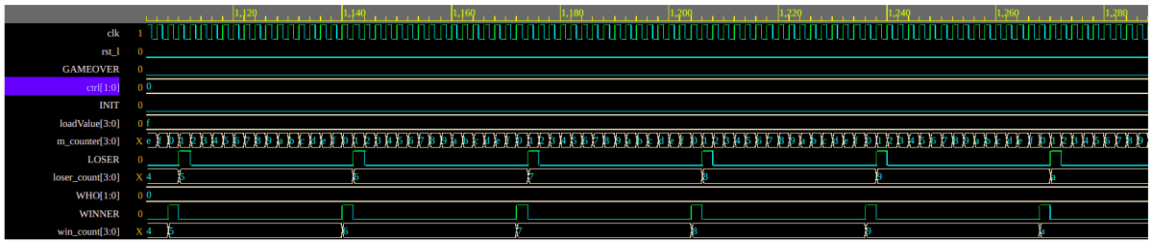

Figure 8

Control signal = 2'b00 (count up by 1)
Load value = 4'b1111

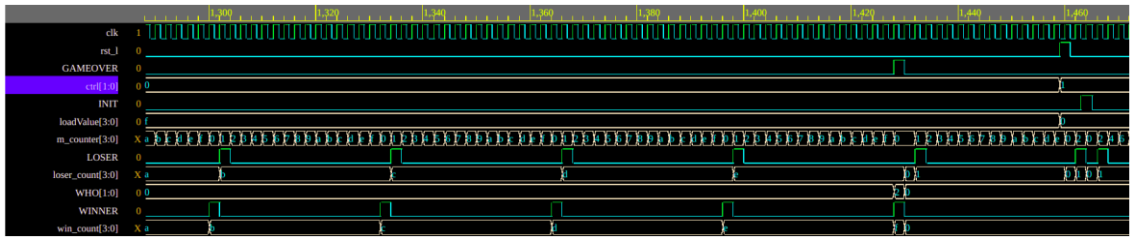

Figure 9

Control signal = 2’b01 (count up by 2)
Load value = 4’b0000

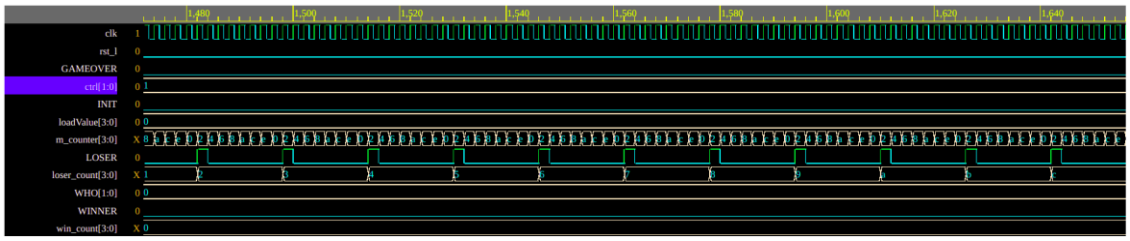

Figure 10

Control signal = 2’b01 (count up by 2)
Load value = 4’b0000

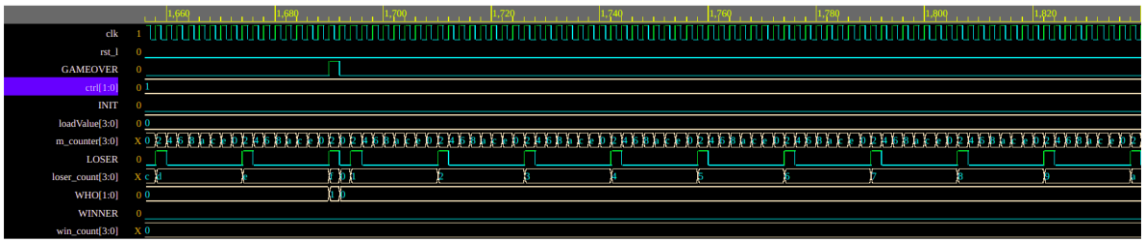

Figure 11

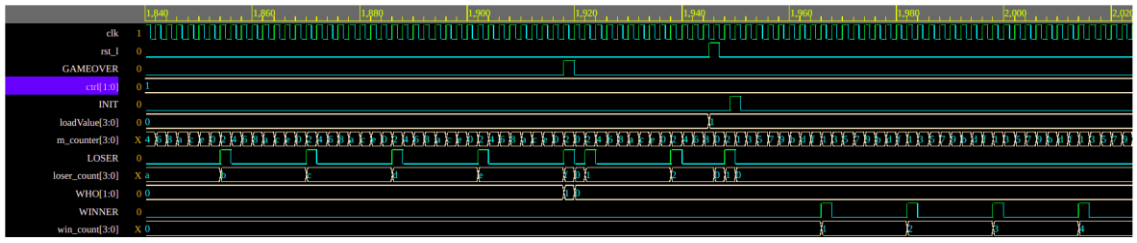

Figure 12

Control signal = 2’b01 (count up by 2)
Load value = 4’b0001

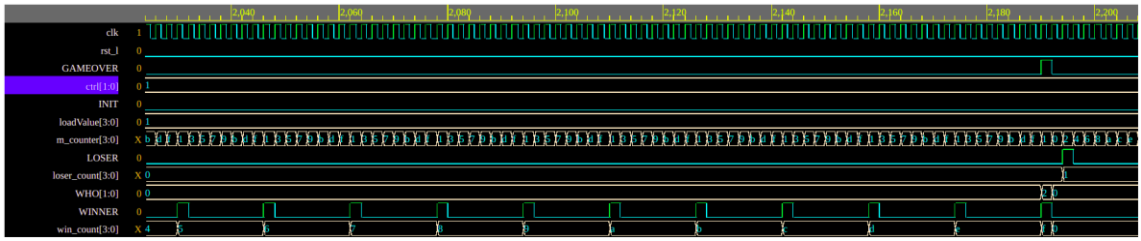

Figure 13

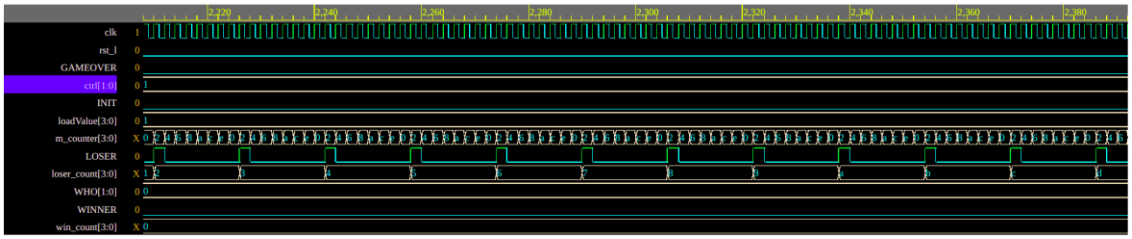

Figure 14

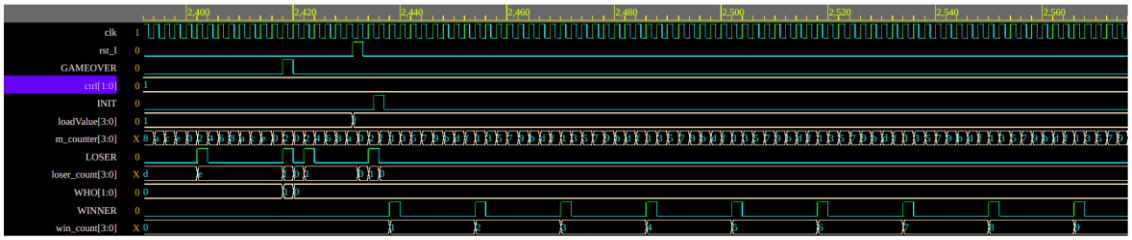

Figure 15

Control signal = 2'b01 (count up by 2)
Load value = 4'b1111

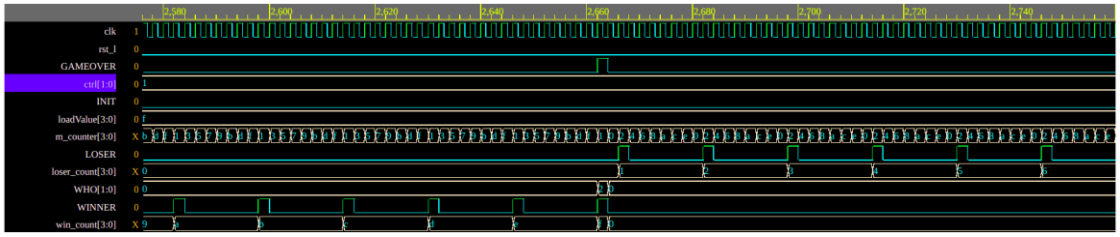

Figure 16

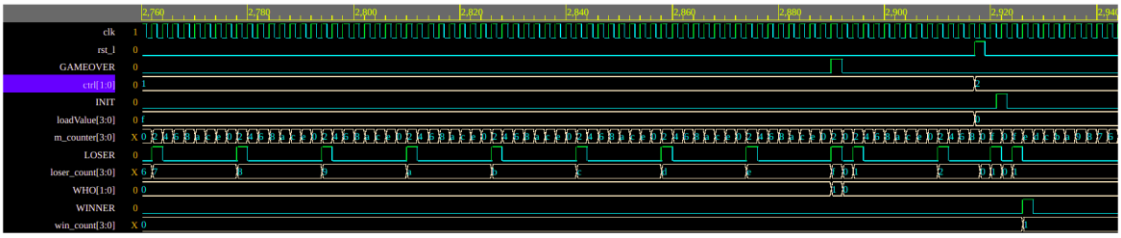

Figure 17

Control signal = 2’b10 (count down by 1)
Load value = 4’b0000

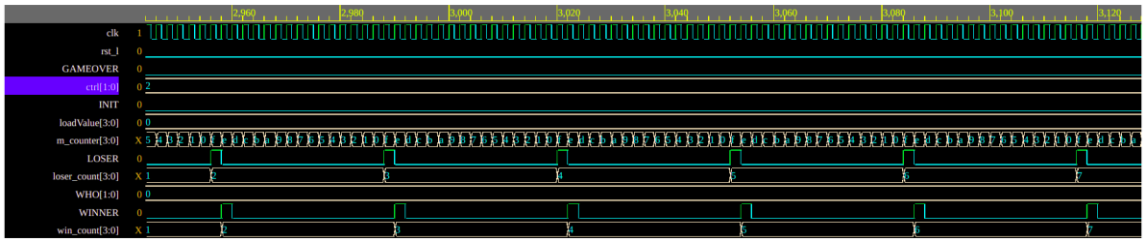

Figure 18

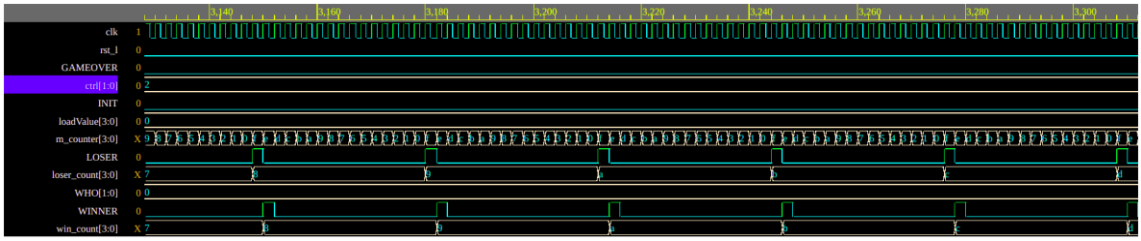

Figure 19

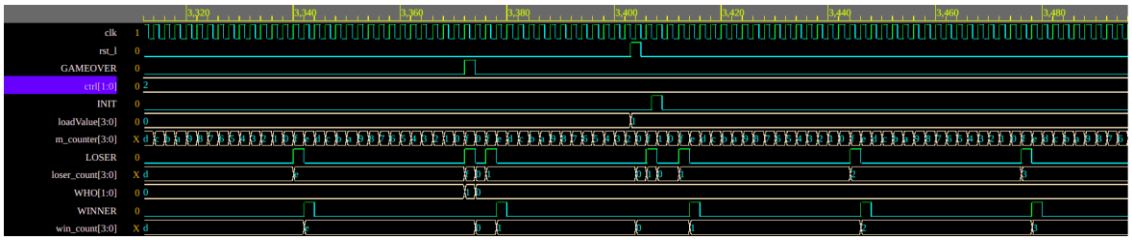

Figure 20

Control signal = 2’b10 (count down by 1)
Load value = 4’b0001

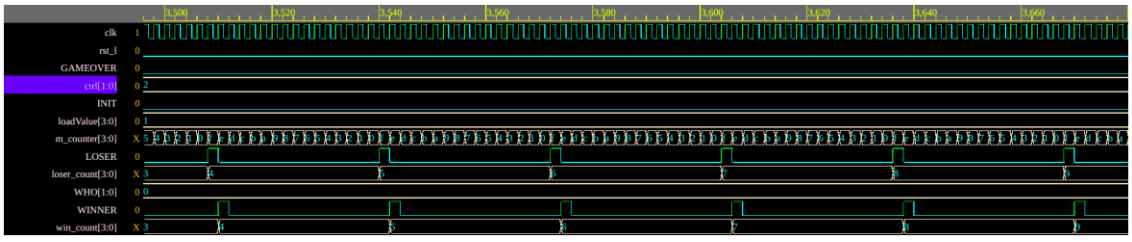

Figure 21

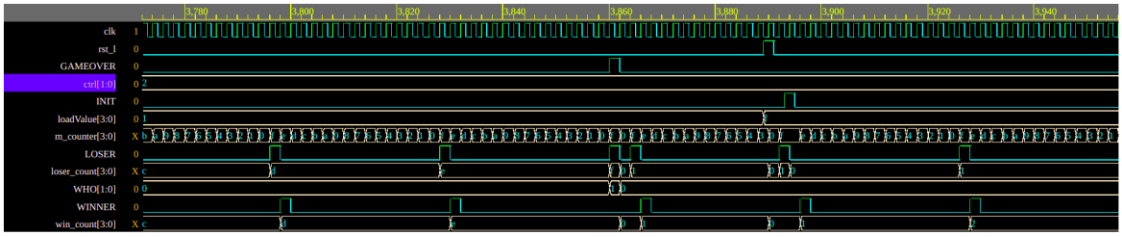

Figure 22

Control signal = 2'b10 (count down by 1)
Load value = 4'b1111

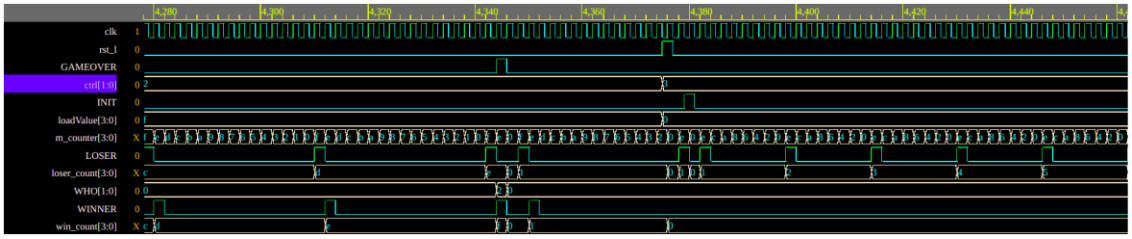

Figure 23

Control signal = 2'b11 (count down by 2)
Load value = 4'b0000

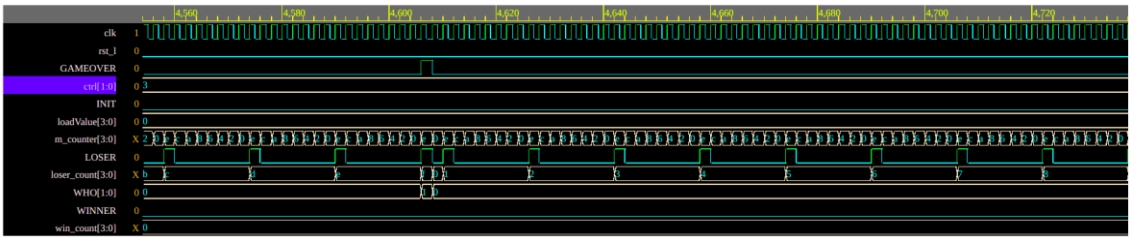

Figure 24

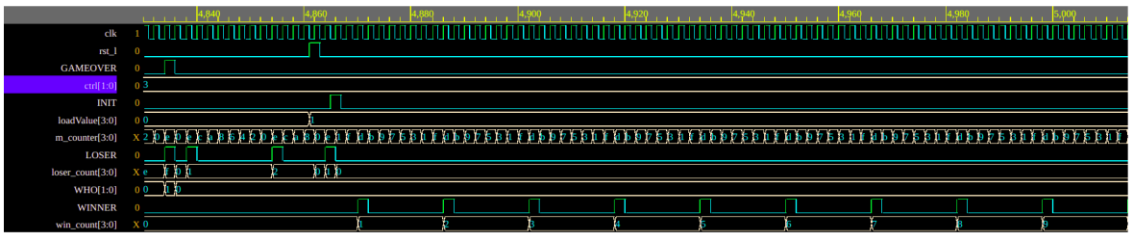

Figure 25

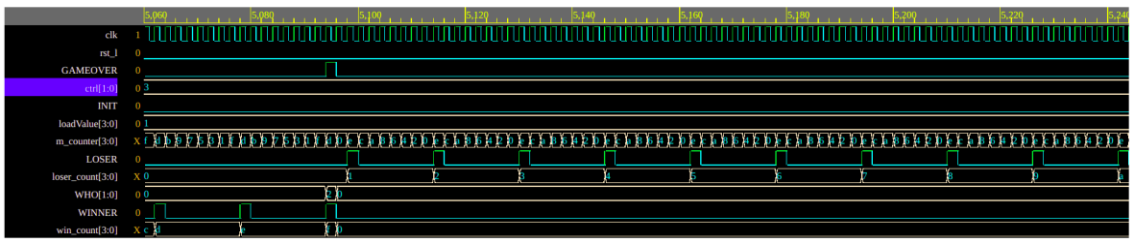

Figure 26

Control signal = 2’b11 (count down by 2)
Load value = 4’b0001

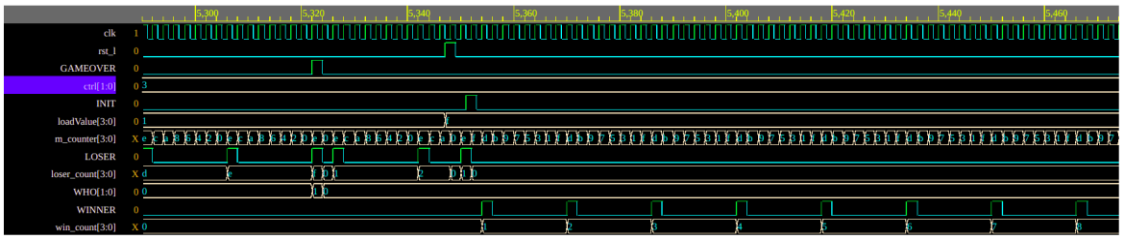

Figure 27

Control signal = 2’b11 (count down by21)
Load value = 4’b1111

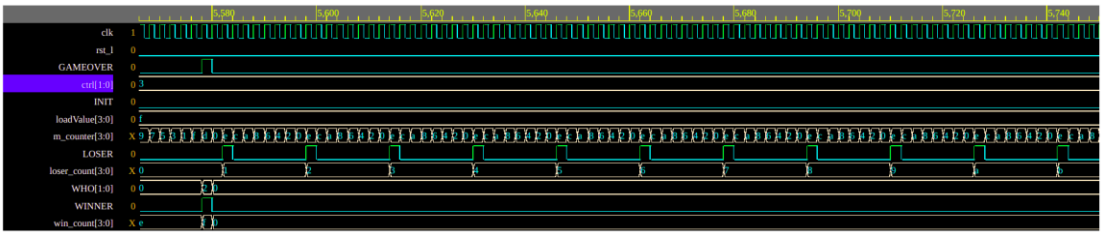

Figure 28