Logo

Description automatically generated

**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: 3](#_Toc103205871)

[Interface: 3](#_Toc103205872)

[Counter Module: 3](#_Toc103205873)

[Test bench: 5](#_Toc103205874)

[Top module: 7](#_Toc103205875)

[Assertion output: 8](#_Toc103205876)

[Output: 9](#_Toc103205877)

# 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

                // sig.rst\_l <= 1;

                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 reset\_signals;

      @(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)) ##[150:250] GAMEOVER == 1;

    endproperty

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        Asserions

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    assert\_winner: assert property(winner)$display("@ cycle [%0t] Assertion GameOver passed", $time / 2);

    assert\_reset\_signals: assert property (reset\_signals) $display("@ cycle [%0t] Assertion Reseting signals passed", $time / 2);

endprogram

## 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

## Assertion output:

A close-up of a document

Description automatically generated with medium confidence

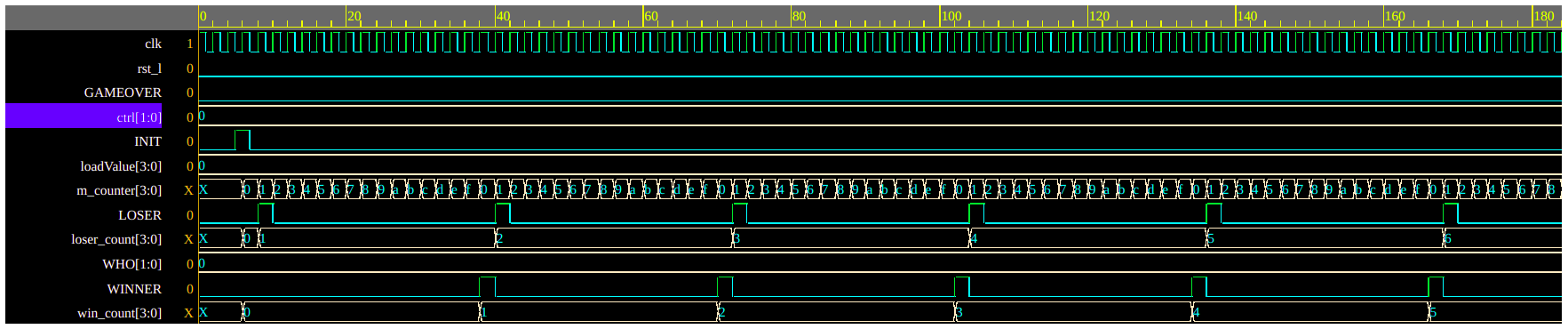
Text

Description automatically generated

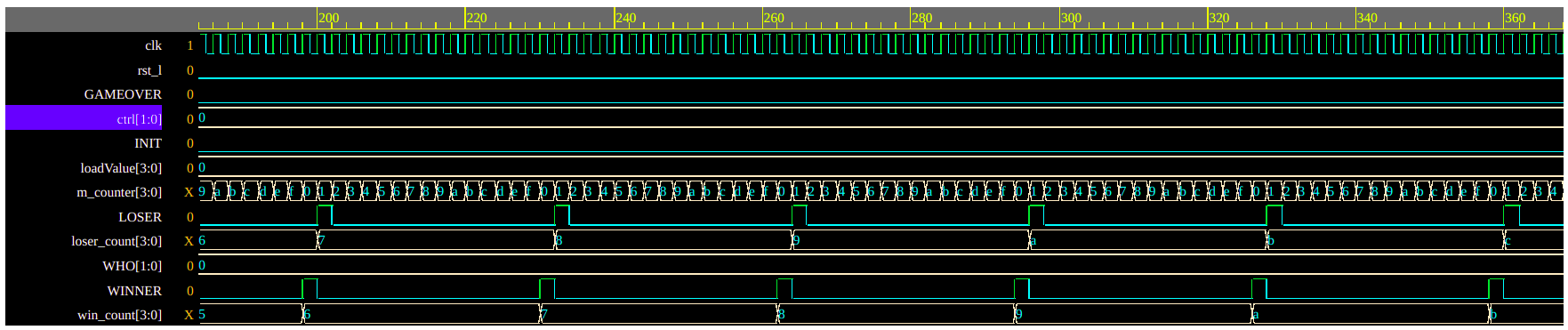
# Output:

Control signal = 2’b00 (count up by 1)

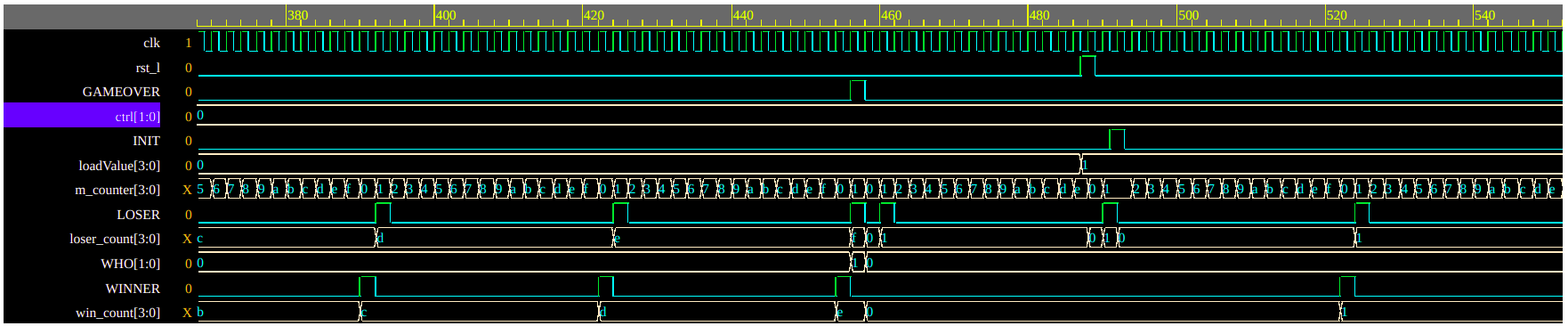
Load value = 4’b0000



*Figure 1*



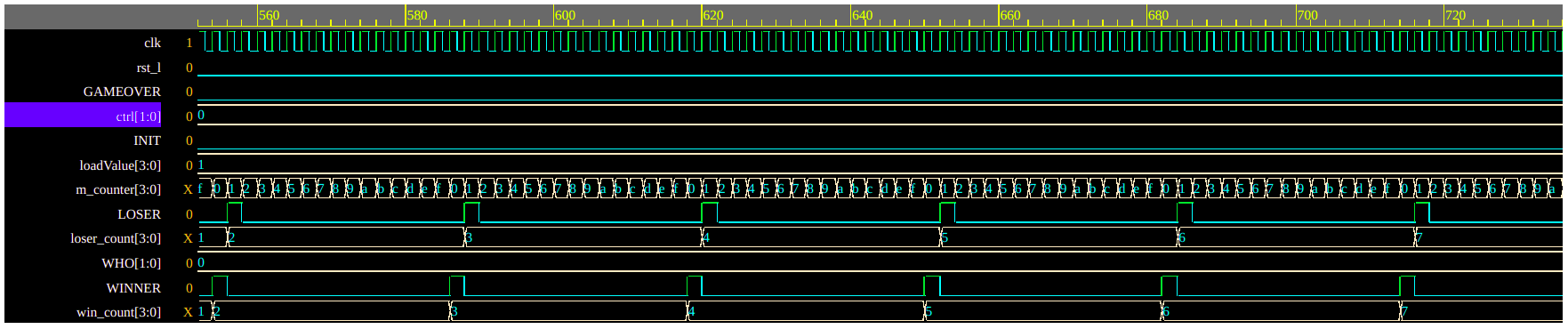
*Figure 3*



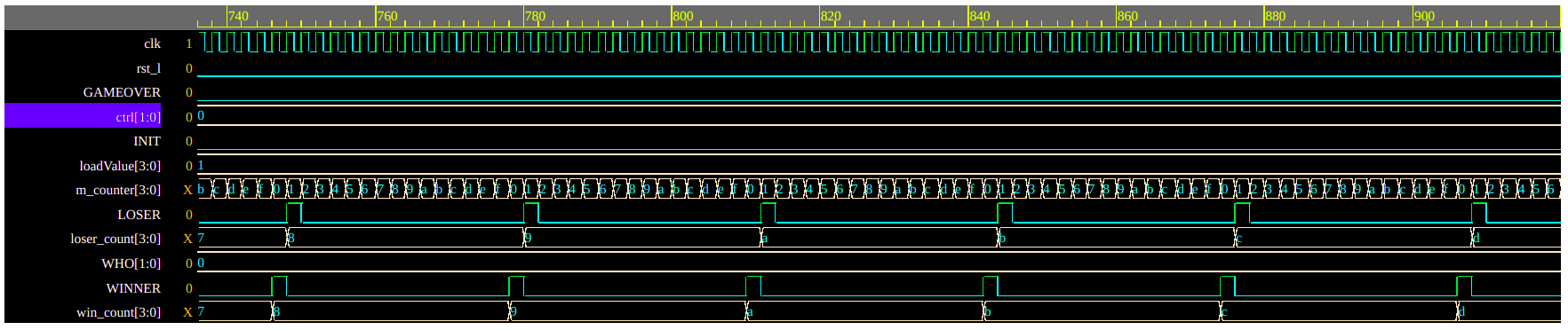
*Figure 4*

Control signal = 2’b00 (count up by 1)

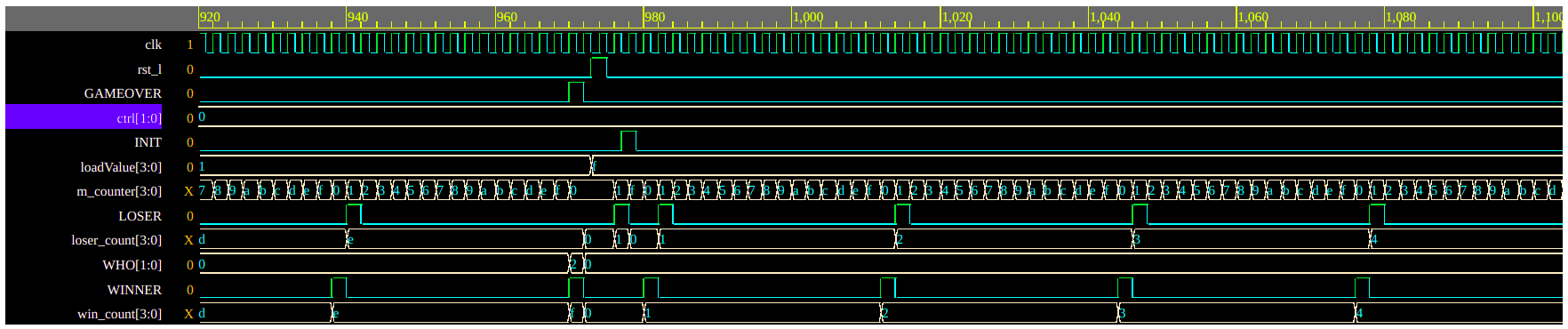
Load value = 4’b0001



*Figure 5*



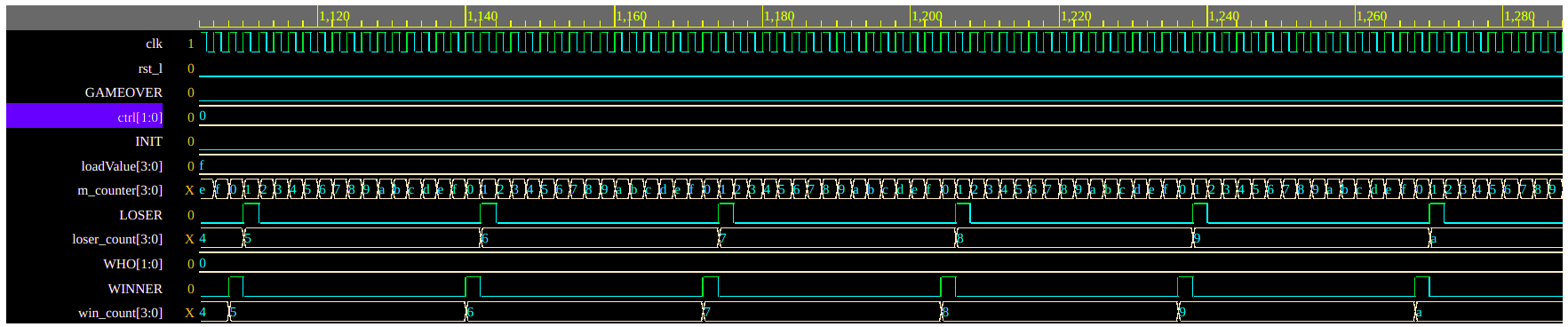
*Figure 6*



*Figure 7*

Control signal = 2’b00 (count up by 1)

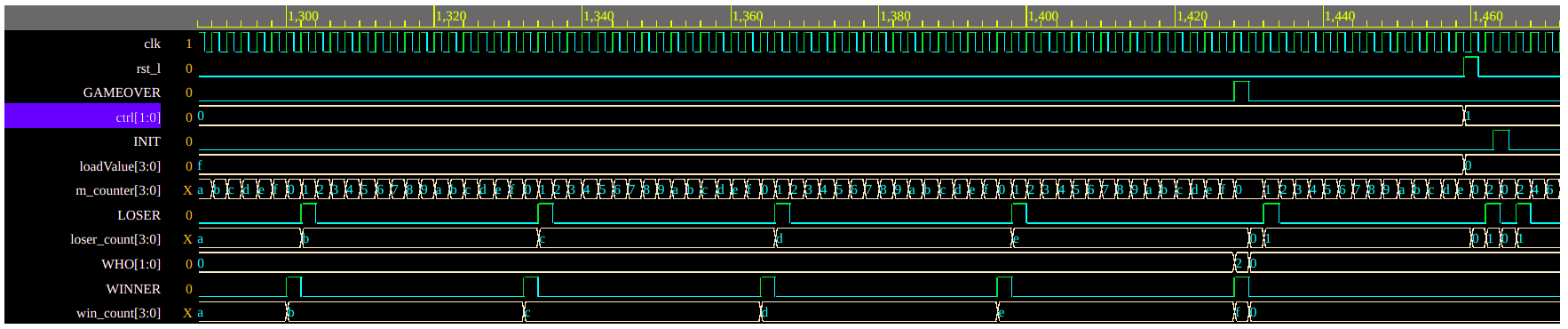
Load value = 4’b1111



*Figure 8*

Control signal = 2’b00 (count up by 1)

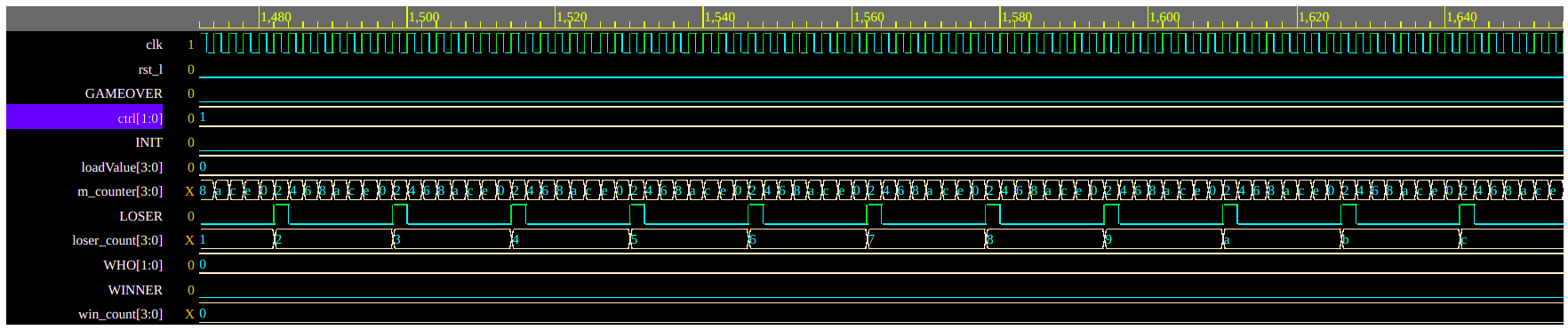
Load value = 4’b1111



*Figure 9*

Control signal = 2’b01 (count up by 2)

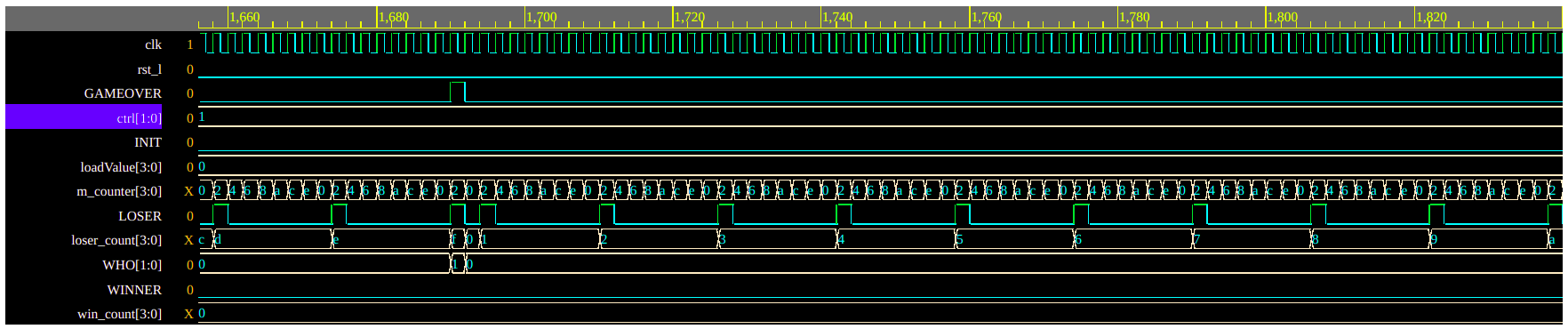
Load value = 4’b0000



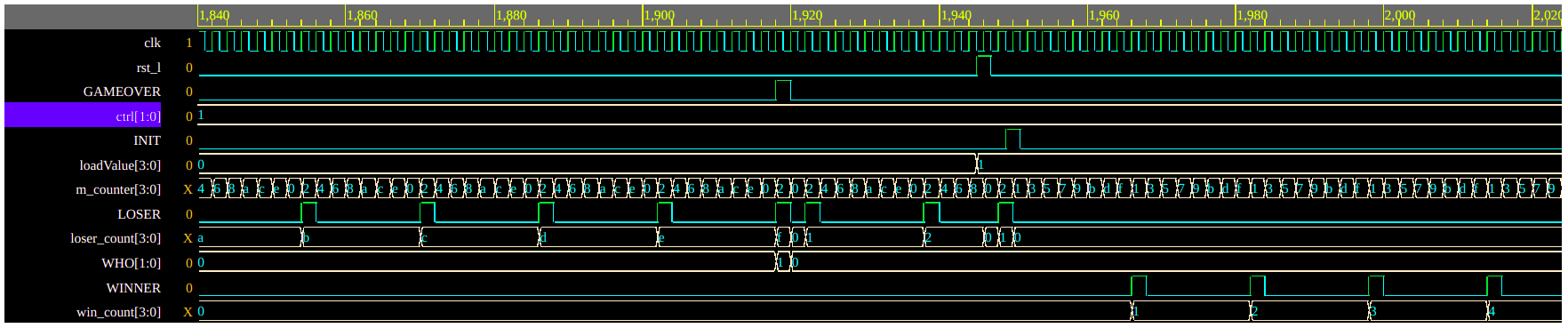
*Figure 10*

Control signal = 2’b01 (count up by 2)

Load value = 4’b0000



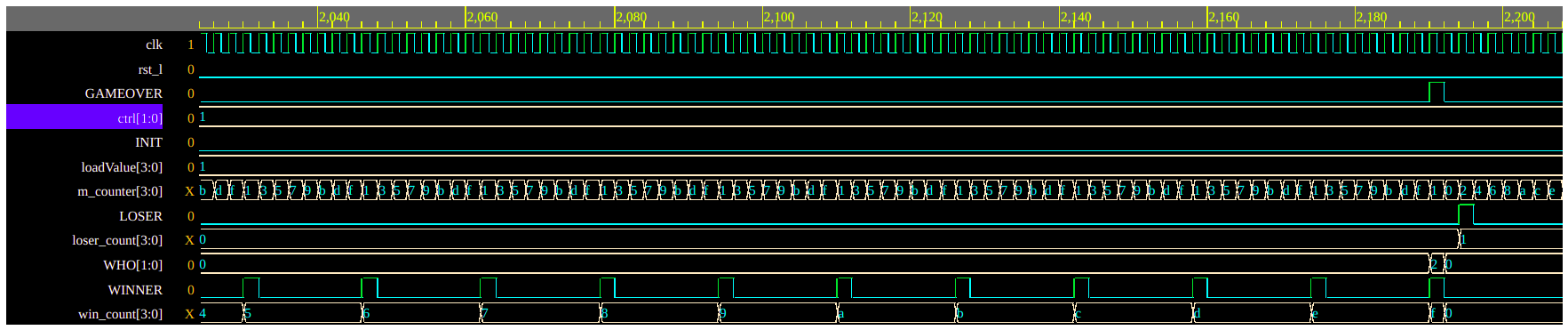
*Figure 11*



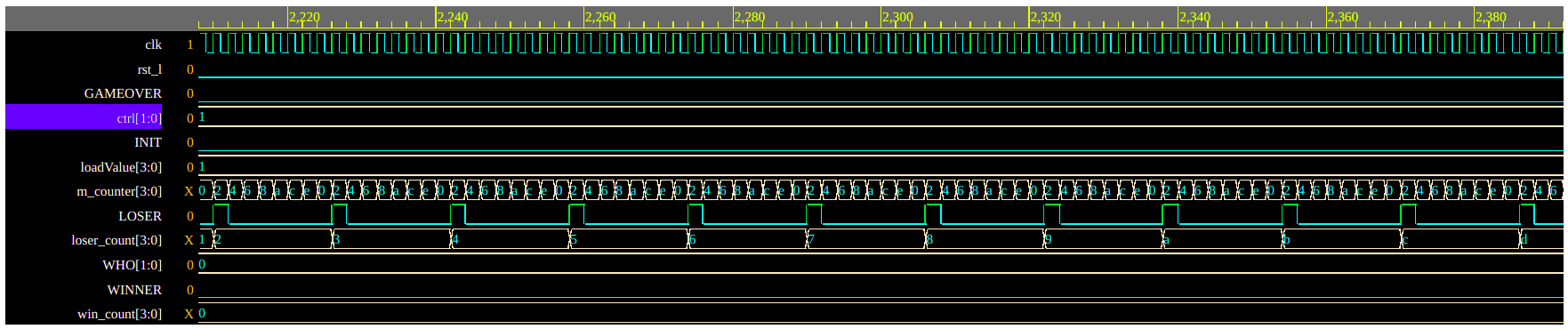
*Figure 12*

Control signal = 2’b01 (count up by 2)

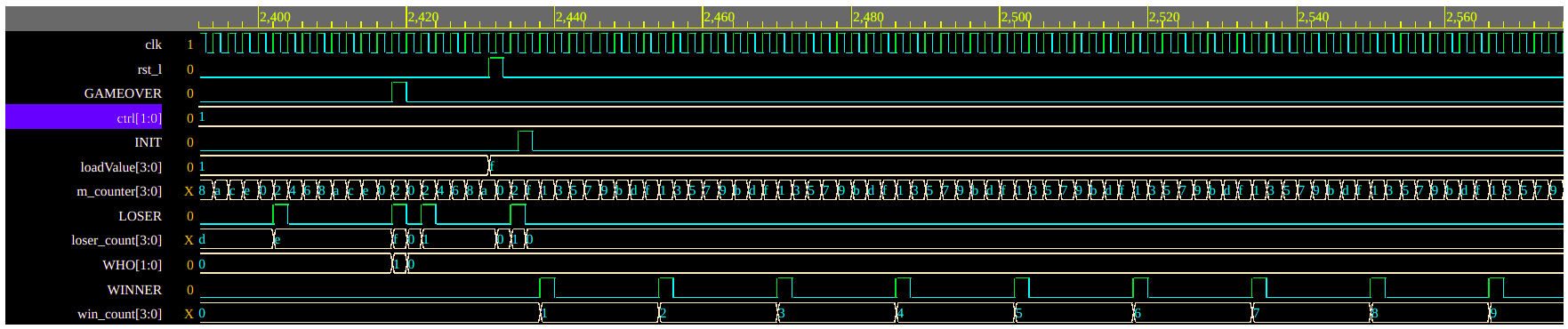
Load value = 4’b0001



*Figure 13*



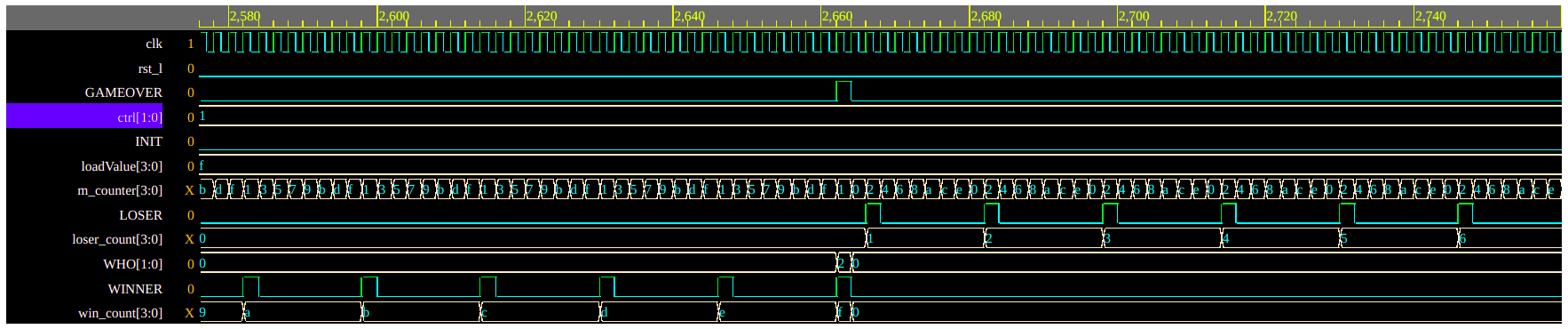
*Figure 14*



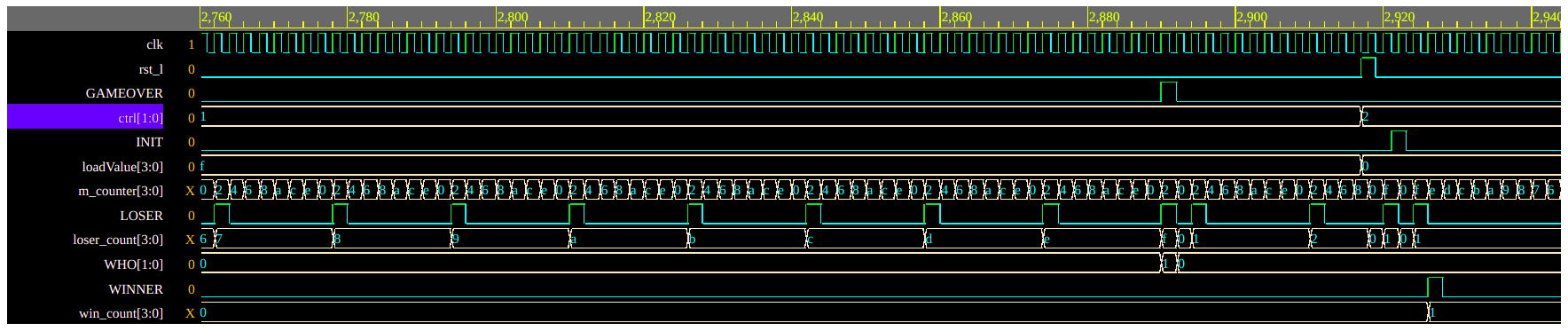
*Figure 15*

Control signal = 2’b01 (count up by 2)

Load value = 4’b1111



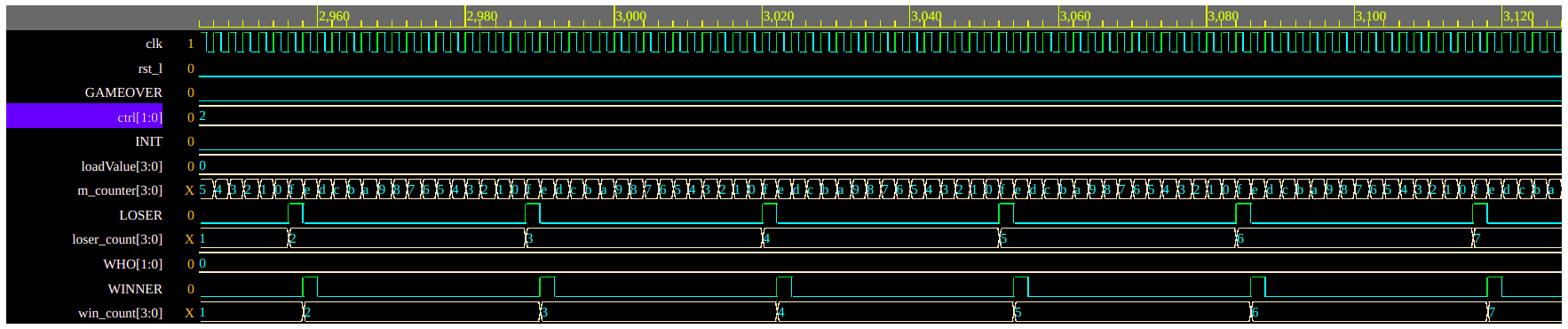
*Figure 16*



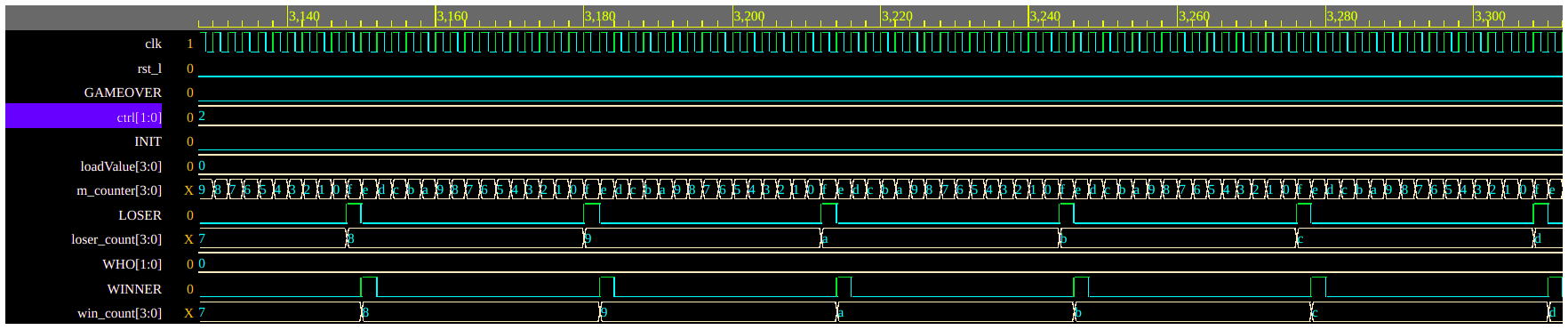
*Figure 17*

Control signal = 2’b10 (count down by 1)

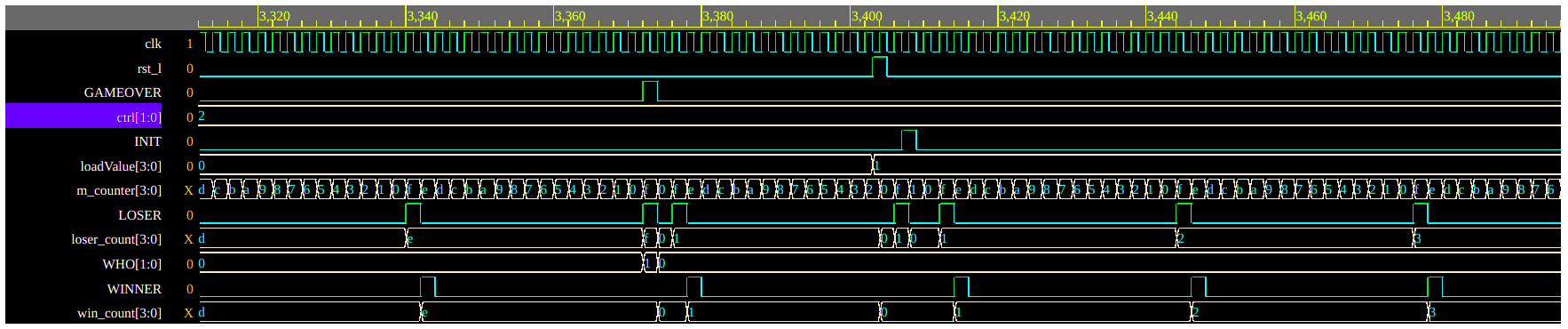
Load value = 4’b0000



*Figure 18*



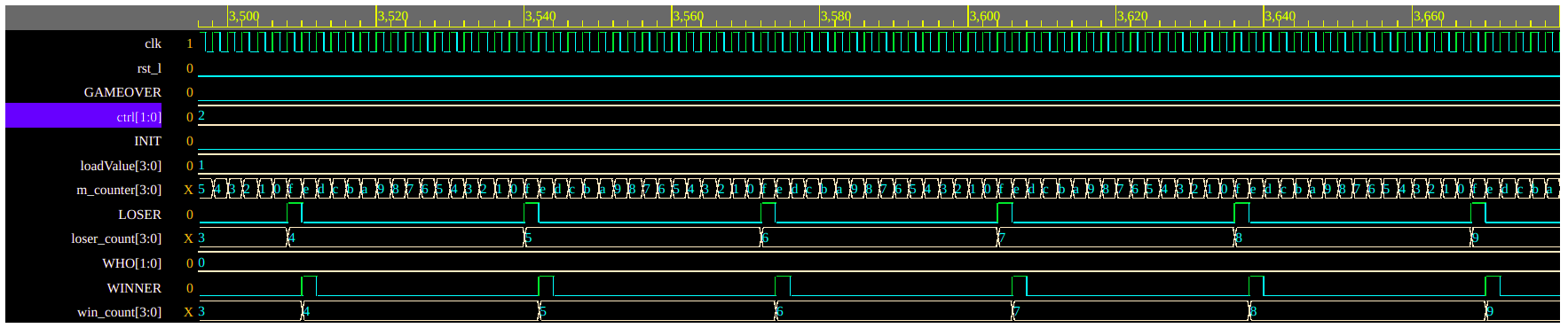
*Figure 19*



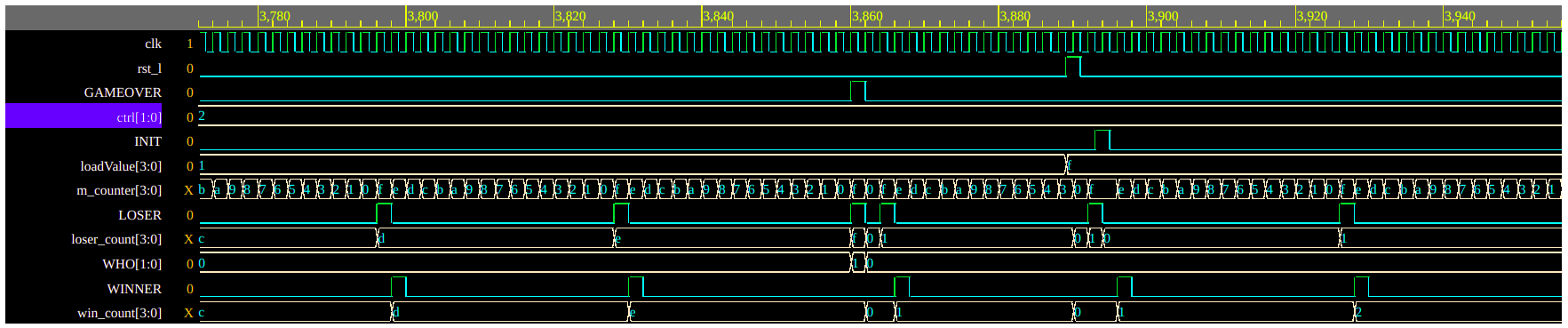
*Figure 20*

Control signal = 2’b10 (count down by 1)

Load value = 4’b0001



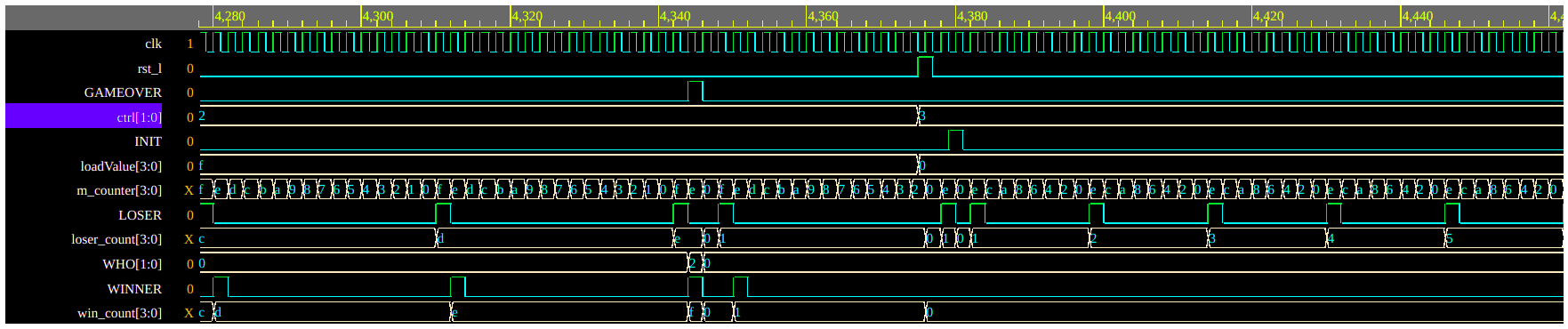
*Figure 21*



*Figure 22*

Control signal = 2’b10 (count down by 1)

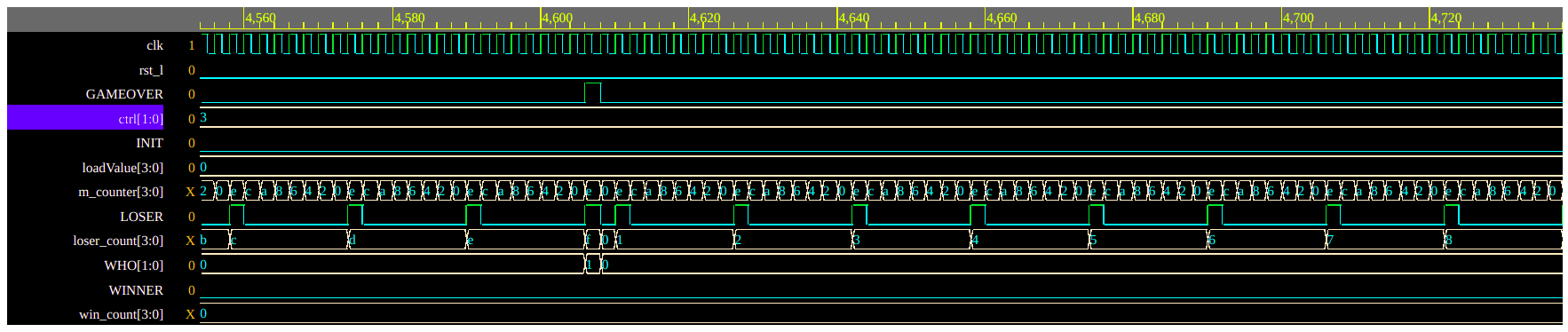
Load value = 4’b1111



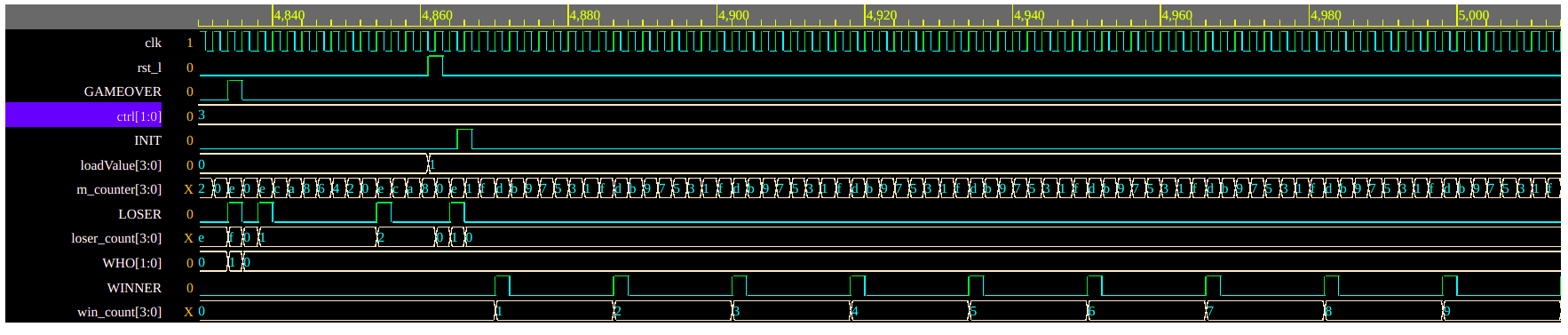
*Figure 23*

Control signal = 2’b11 (count down by 2)

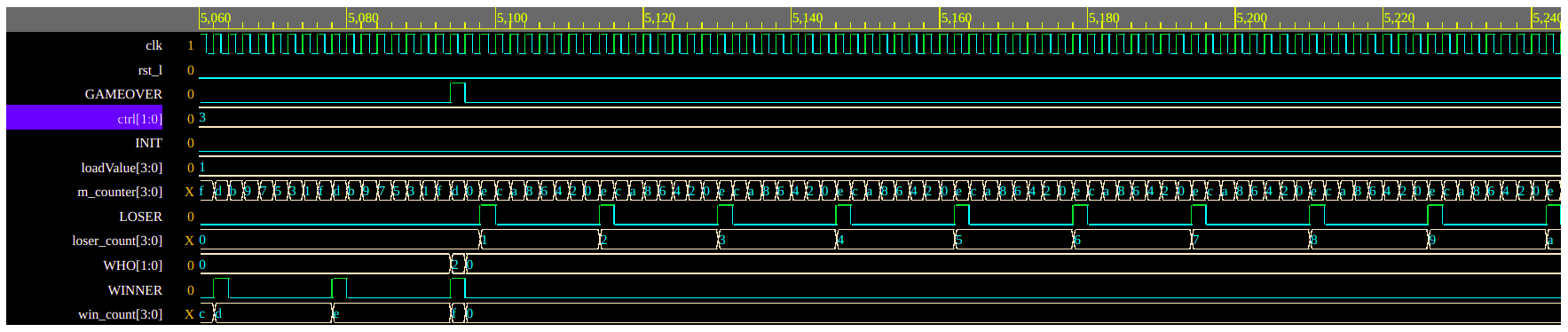
Load value = 4’b0000



*Figure 24*



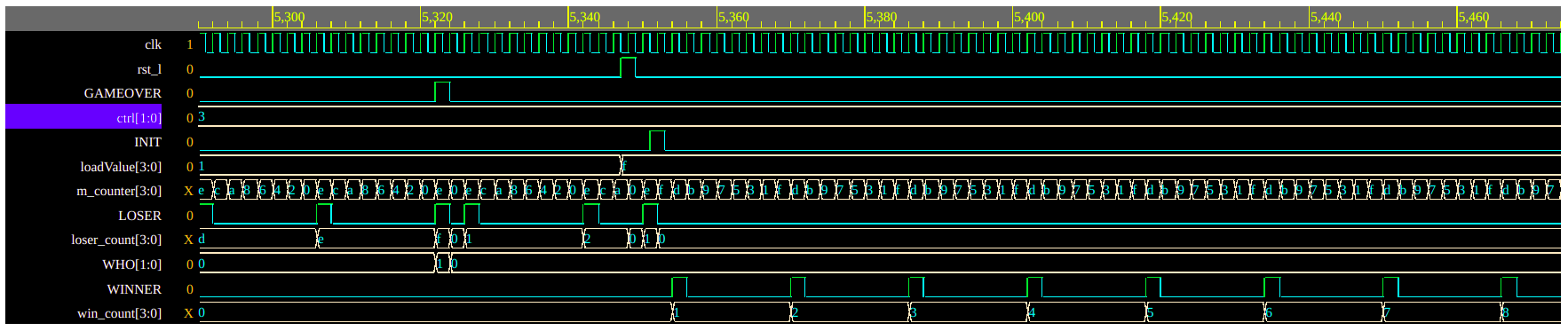
*Figure 25*



*Figure 26*

Control signal = 2’b11 (count down by 2)

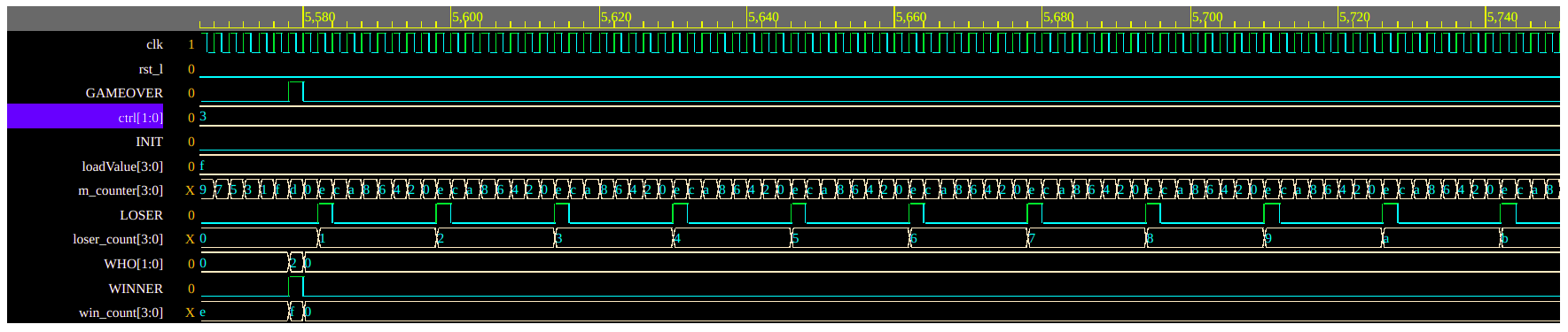
Load value = 4’b0001



*Figure 27*

Control signal = 2’b11 (count down by21)

Load value = 4’b1111



*Figure 28*