Digital Verification Project

Contents

1. Project Brief:

Multi-mode counter that can count up, down, by ones and by twos. There is a two-bit control bus input indicating which one of the four modes are active.
There's an Initial value input and a control signal called INIT. When INIT is logic 1, the initial value will be parallelly loaded into the multi-mode counter.

Whenever the count is equal to all zeros, a signal called LOSER is high.
When the count is all ones, a signal called WINNER is high.
In either case, the signal will remain high for only one cycle.

When the number of times WINNER and LOSER goes high 15 times an output called GAMEOVER will be set high.
If the game is over because WINNER got to 15 first, WHO is set to 2'b10, and if LOSER got to 15 first then WHO is set to 2'b01. WHO starts at 2'b00 and returns to it after each game over.

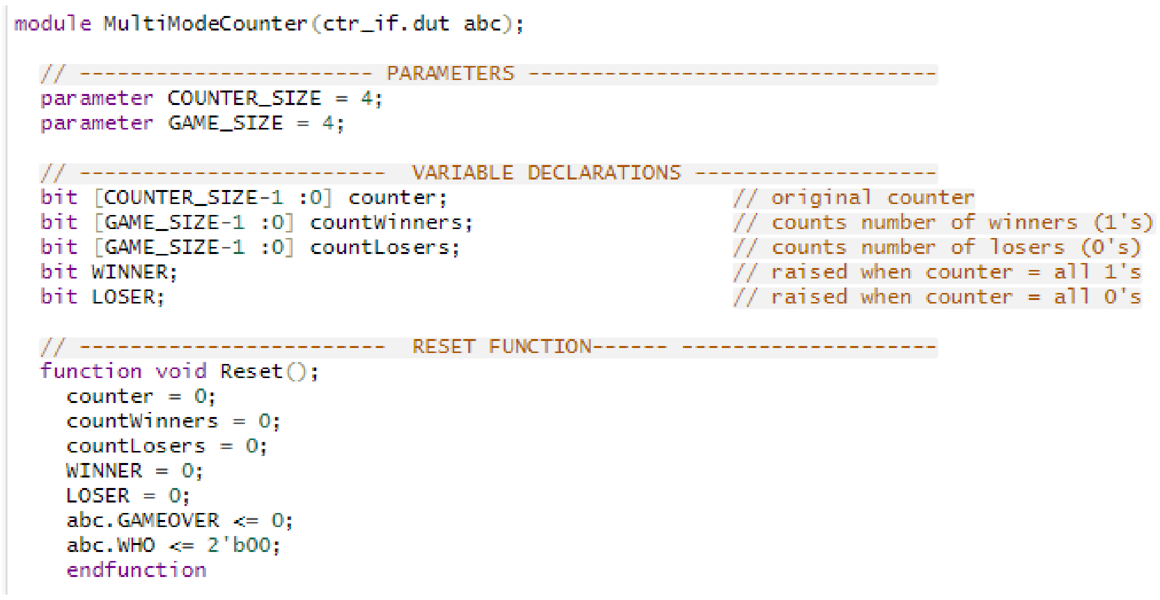
All the counters are synchronously cleared and then the game starts over.

Using the following:
1. An interface with the appropriate set of modports, and clocking blocks.
2. A testbench (implemented as a program not a module).
3. A top module.
4. Assertions to ensure that some illegal scenarios can't be generated.

2. Code Explanation:

2.1 Design

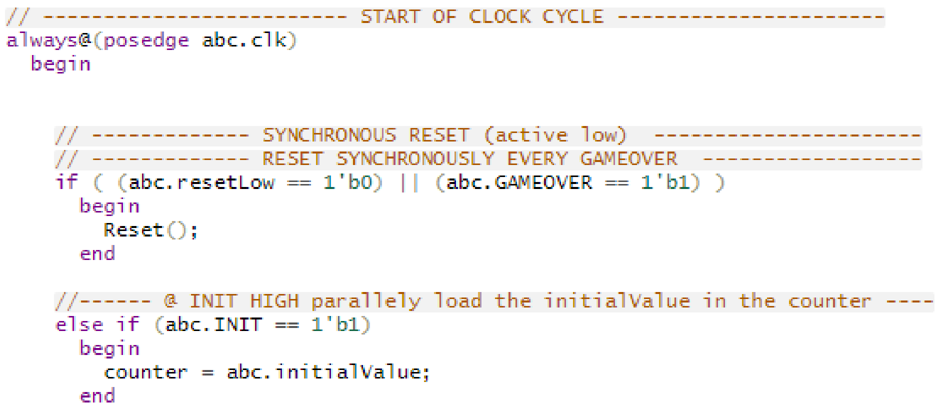
First, we begin with defining the parameters then by declaring all the variables that will be used inside the module, and a function for Reset, (Interface is used for input and output signals)



```
module MultiModeCounter(ctr_if.dut abc);

  // ----------------------- PARAMETERS --------------------------------
  parameter COUNTER_SIZE = 4;
  parameter GAME_SIZE = 4;

  // ------------------------  VARIABLE DECLARATIONS --------------------
  bit [COUNTER_SIZE-1 :0] counter;                    // original counter
  bit [GAME_SIZE-1 :0] countWinners;                  // counts number of winners (1's)
  bit [GAME_SIZE-1 :0] countLosers;                   // counts number of losers (0's)
  bit WINNER;                                         // raised when counter = all 1's
  bit LOSER;                                          // raised when counter = all 0's

  // ------------------------  RESET FUNCTION------ --------------------
  function void Reset();
    counter = 0;
    countWinners = 0;
    countLosers = 0;
    WINNER = 0;
    LOSER = 0;
    abc.GAMEOVER <= 0;
    abc.WHO <= 2'b00;
    endfunction
```

2- Inside the always block which is triggered by positive edge of the clock performs the following actions:

1. Check reset (active low), if triggered reset all flags and counters. (Synchronous Reset)
2. Check the INIT control signal, if high load the initial value in the multi-mode counter.
3. Check if GAMEOVER is high, if high then reset the game.



```
// ------------------------ START OF CLOCK CYCLE ---------------------
always@(posedge abc.clk)
  begin


    // ------------- SYNCHRONOUS RESET (active low)  ----------------------
    // ------------- RESET SYNCHRONOUSLY EVERY GAMEOVER  ------------------
    if ( (abc.resetLow == 1'b0) || (abc.GAMEOVER == 1'b1) )
      begin
        Reset();
      end

    //------ @ INIT HIGH parallely load the initialValue in the counter ----
    else if (abc.INIT == 1'b1)
      begin
        counter = abc.initialValue;
      end
```

3- Else it performs the main body of the operations

1. If WINNER or LOSER signals are high, reset both signals.

2. Check control value to specify which mode to use to perform the count operation.
3. If the counter is equal to all 1's, then increment the "winnerCounter" and raise the WINNER flag.
    - If the counter is equal to all 0's, then increment the "loserCounter" and raise the LOSER flag.
    - If "winnerCounter" or "loserCounter" reaches 15, then raise the "GAMEOVER" signal and specify which counter using the "WHO" signal.

```
  // ------------------------  MAIN BODY  ------------------------------
  else
    begin

      // --------- RESET EACH TIME A WIN/LOSE CONDITION OCCURS  --------------
      if (WINNER || LOSER)
        begin
          WINNER = 0;    // CAN BE HIGH FOR ONE CYCLE ONLY
          LOSER = 0;     // CAN BE HIGH FOR ONE CYCLE ONLY
      end

      // controlValue determines counter mode
      case(abc.controlValue)
        2'b00: counter = counter + 1;
        2'b01: counter = counter + 2;
        2'b10: counter = counter - 1;
        2'b11: counter = counter - 2;
      endcase

  // ------------------------ WINNER OR LOSER?  ------------------------

      // check for winners ( all 1's set)
      if (counter == '1)
        begin
          // set winner signal
          countWinners = countWinners +1;
          WINNER = 1;
        end

      // check for losers ( all 1's set)
      else if (counter == 0)
        begin
          // set loser signal
          countLosers = countLosers +1;
          LOSER = 1;
        end
  // ------------------------ GAMEOVER  ---------------------------

      // GAMEOVER DUE TO WINNERS
      if (countWinners == '1)
        begin
          abc.GAMEOVER <= 1;
          //Game is over because WINNER got to 15 first
          abc.WHO <= 2'b10;
        end

       // GAMEOVER DUE TO LOSERS
       else if (countLosers == '1)
        begin
          abc.GAMEOVER <= 1;
          //Game is over because LOSER got to 15 first
          abc.WHO <= 2'b01;
        end
    end // end of "else"

end // end of "always"
```

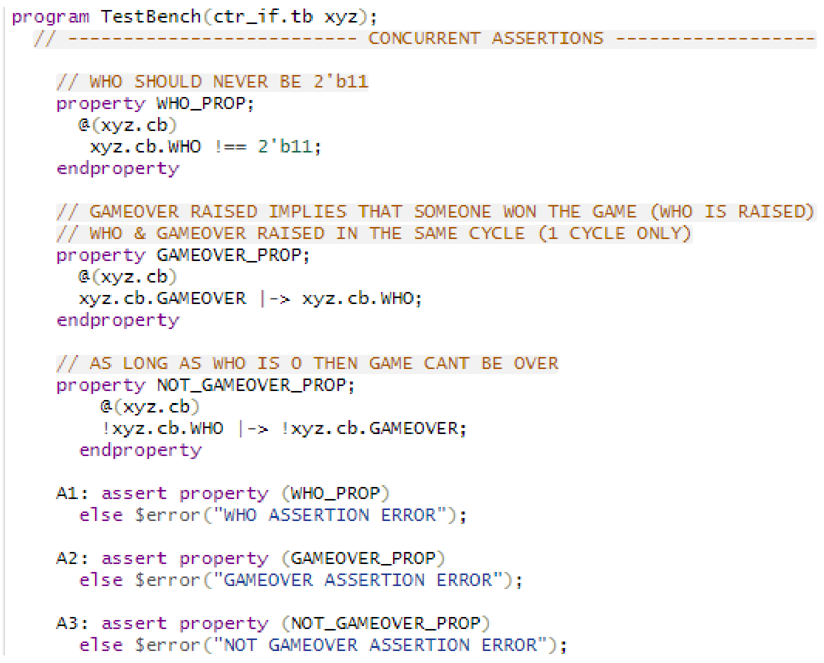
2.2 TOP Module

```
  // ----------------------------------------------------------------------
  // -------------------------- TOP  -------------------------------------
  // ----------------------------------------------------------------------

module top(output logic clk);

  // ---------------------------- CLOCK ------------------------------
  initial clk = 0;
  initial forever #5 clk=~clk;

  // ---------------------------- CONNECTION -------------------------
  ctr_if i1(clk);
  MultiModeCounter c1(i1.dut);
  TestBench test(i1.tb);

  initial begin
    $dumpfile("waves.vcd");
    $dumpvars;
  end
endmodule
```

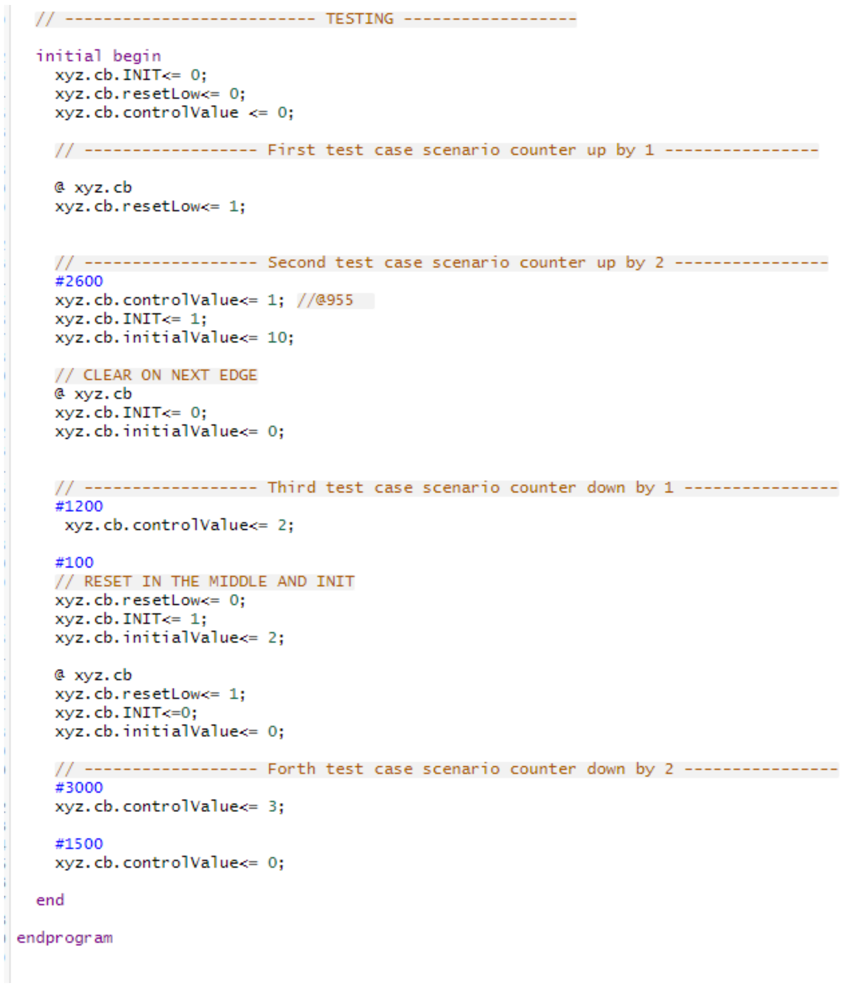
2.3 Interface Module

```
  // -------------------------------------------------------------------
  // ------------------------- INTERFACE--------------------------------
  // -------------------------------------------------------------------

interface ctr_if(input logic clk);
  // ----------------------- PARAMETERS ------------------------------
  parameter COUNTER_SIZE = 4;
  parameter GAME_SIZE = 4;

  // ------------------------- INPUTS IN DESIGN -------------------------
  logic [1:0] controlValue = 0;
  logic [COUNTER_SIZE-1 :0] initialValue = 0; // initialValue is loaded into the counter
  logic INIT= 0, resetLow = 0;

  // ------------------------- OUTPUTS IN DESIGN ----------------------
  logic GAMEOVER = 0;
  logic [1:0] WHO = 2'b00;

  // -------------------------- CLOCKING SYNCH -------------------------
  clocking cb @(posedge clk);
    //default input #1ns;// output #1ns;
    output resetLow, controlValue, initialValue, INIT;
    input  GAMEOVER, WHO;
  endclocking

  // ----------------------------- PORTING -----------------------------

  modport dut(input clk, resetLow, controlValue, initialValue, INIT
             ,output  GAMEOVER, WHO);

  // -------------------------- SYNCHRONOUS TB PORTING -----------------
  modport tb(clocking cb);

endinterface
```

2.4 Interface Module



```
program TestBench(ctr_if.tb xyz);
  // -------------------------- CONCURRENT ASSERTIONS ------------------

    // WHO SHOULD NEVER BE 2'b11
    property WHO_PROP;
      @(xyz.cb)
       xyz.cb.WHO !== 2'b11;
    endproperty

    // GAMEOVER RAISED IMPLIES THAT SOMEONE WON THE GAME (WHO IS RAISED)
    // WHO & GAMEOVER RAISED IN THE SAME CYCLE (1 CYCLE ONLY)
    property GAMEOVER_PROP;
      @(xyz.cb)
      xyz.cb.GAMEOVER |-> xyz.cb.WHO;
    endproperty

    // AS LONG AS WHO IS 0 THEN GAME CANT BE OVER
    property NOT_GAMEOVER_PROP;
        @(xyz.cb)
        !xyz.cb.WHO |-> !xyz.cb.GAMEOVER;
      endproperty

    A1: assert property (WHO_PROP)
      else $error("WHO ASSERTION ERROR");

    A2: assert property (GAMEOVER_PROP)
      else $error("GAMEOVER ASSERTION ERROR");

    A3: assert property (NOT_GAMEOVER_PROP)
      else $error("NOT GAMEOVER ASSERTION ERROR");
```



```
  // ------------------------- TESTING -----------------

  initial begin
    xyz.cb.INIT<= 0;
    xyz.cb.resetLow<= 0;
    xyz.cb.controlValue <= 0;

    // ----------------- First test case scenario counter up by 1 ---------------

    @ xyz.cb
    xyz.cb.resetLow<= 1;


    // ----------------- Second test case scenario counter up by 2 ---------------
    #2600
    xyz.cb.controlValue<= 1; //@955
    xyz.cb.INIT<= 1;
    xyz.cb.initialValue<= 10;

    // CLEAR ON NEXT EDGE
    @ xyz.cb
    xyz.cb.INIT<= 0;
    xyz.cb.initialValue<= 0;


    // ----------------- Third test case scenario counter down by 1 ---------------
    #1200
     xyz.cb.controlValue<= 2;

    #100
    // RESET IN THE MIDDLE AND INIT
    xyz.cb.resetLow<= 0;
    xyz.cb.INIT<= 1;
    xyz.cb.initialValue<= 2;

    @ xyz.cb
    xyz.cb.resetLow<= 1;
    xyz.cb.INIT<=0;
    xyz.cb.initialValue<= 0;

    // ----------------- Forth test case scenario counter down by 2 ---------------
    #3000
    xyz.cb.controlValue<= 3;

    #1500
    xyz.cb.controlValue<= 0;

  end

endprogram
```

3. Design Choice:

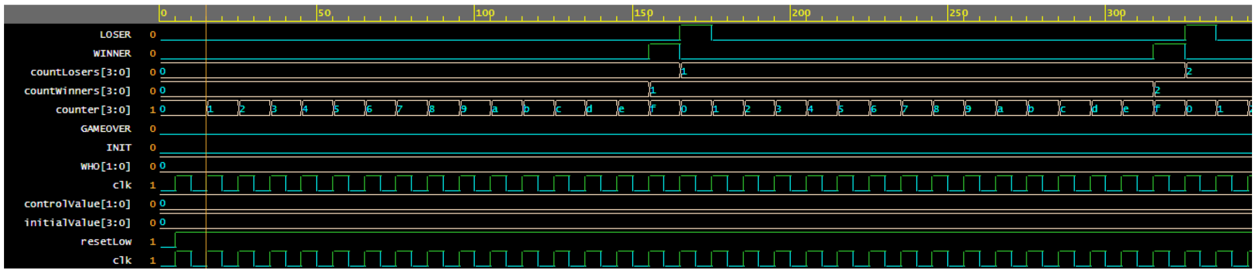
The clock is used in the design to trigger the always block which contains the main body of the code including the count operation.
Counter counts based on a positive edge of an input signal which is the "clk" signal (clock).
"resetLow" is an input signal which is used to reset the game at any time, so the user can reset the game at any time. The reset being synchronous or asynchronous won't matter much in this design, here it is synchronous.

4. Test Bench Scenarios

- Counter Up by 1 (NO INIT)
- Counter Up by 2 (INIT)
- Counter down by 1 (Reset in the middle & INIT signal raised high)
- Counter down by 2 (Reset in the middle & NO INIT)

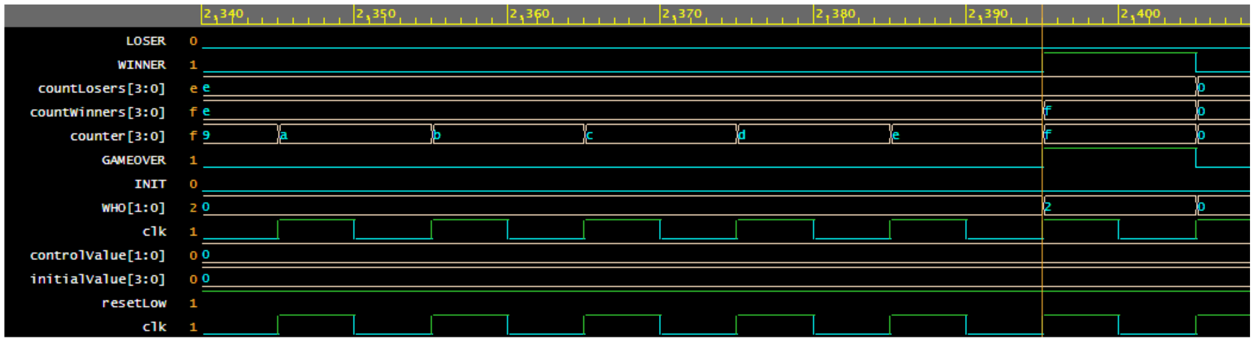
4.1. Counter Up by 1 (INIT)

Counter starts counting from 0 by 1



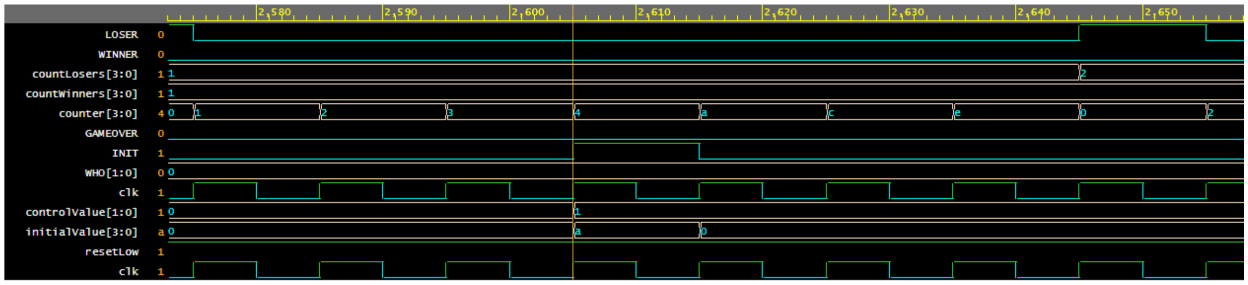
GAMEOVER since the "counterWinners" reached 15 (0xF), so the WINNER, GAMEOVER, WHO signals are raised for one cycle, WHO is set to 0b10 (0x02) since "counterWinners" reached 15 first.

GAMEOVER (reset all the counters and signals).



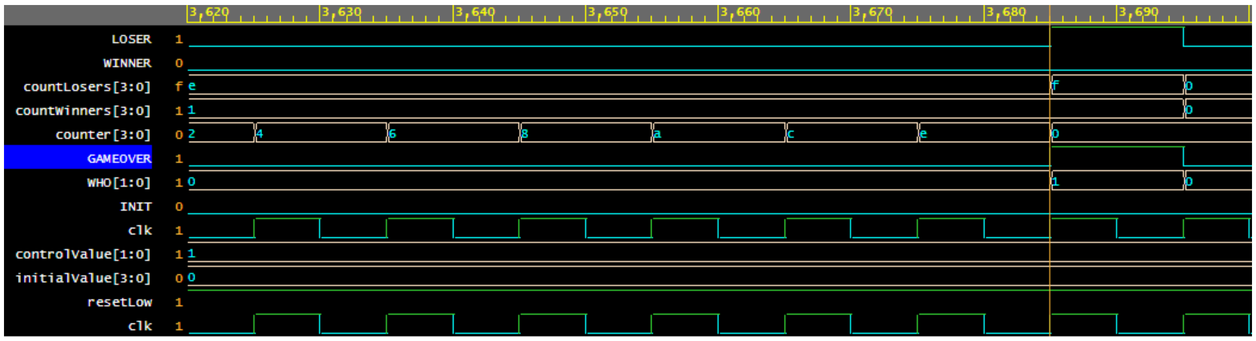
4.2. Counter Up by 2 (INIT)

"controlValue" is set to 1 which indicates the second mode (Counter Up By 2) so at the next positive edge of the clock cycle the counter starts incrementing by 2 and initial value is loaded since INIT is logically high.



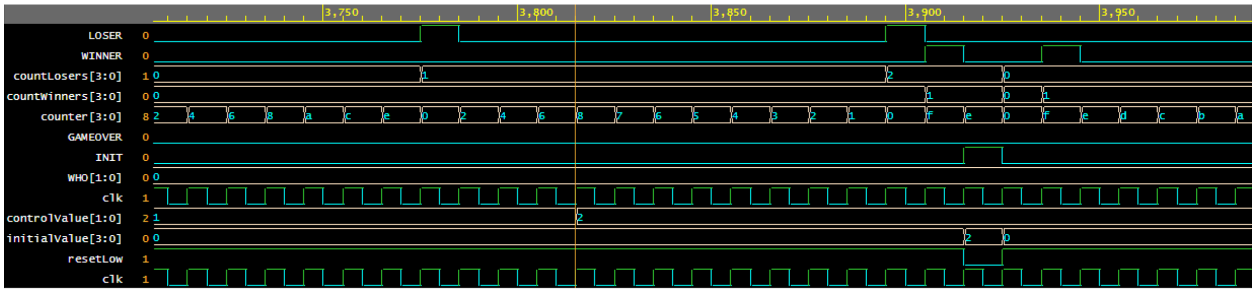
GAMEOVER since the "counterLosers" reached 15 (0xF), so the WINNER, GAMEOVER, WHO signals are raised for one cycle, WHO is set to 0b01 (0x01) since "counterLosers" reached 15 first.

GAMEOVER (reset all the counters and signals).

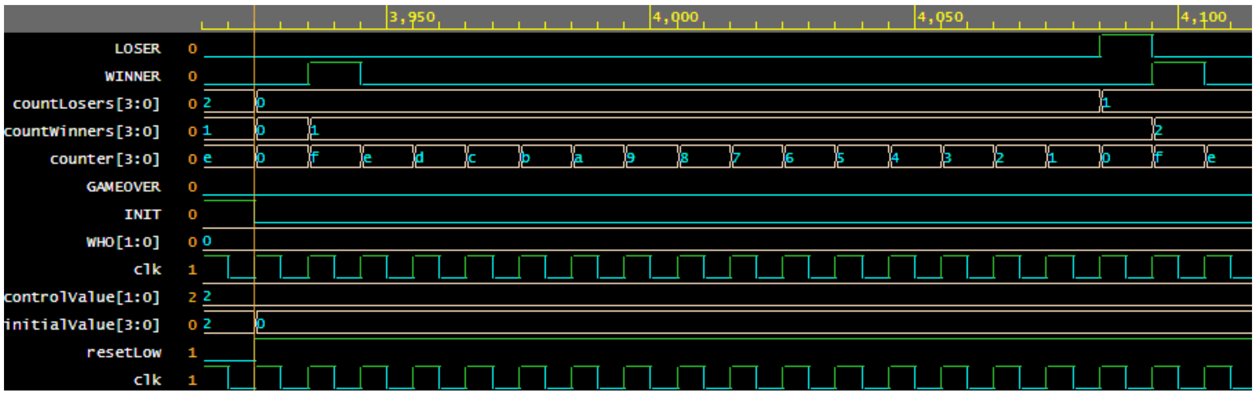


4.3. Counter Down by 1 (Reset in the middle & INIT)

“controlValue” is set to 2 which indicates the third mode (counter down by 1) so at the next positive edge of the clock cycle the counter starts decrementing by 1.

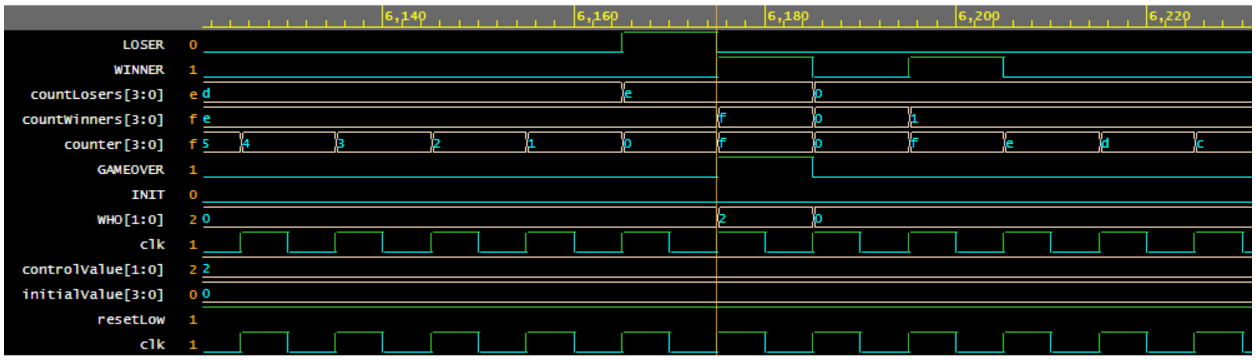


@3925 ns “resetLow” is set to low (active low reset) so the game is reset and starts on the 3rd mode since “controlValue” is set to 2 and initial value is loaded since INIT is logically high.



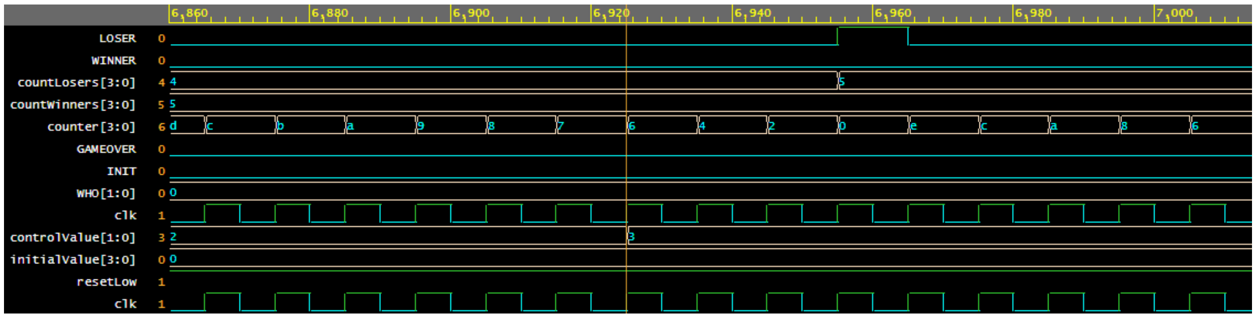
GAMEOVER since the “counterWinners” reached 15 (0xF), so the WINNER, GAMEOVER, WHO signals are raised for one cycle, WHO is set to 0b10 (0x02) since “counterWinners” reached 15 first.

GAMEOVER (reset all the counters and signals).



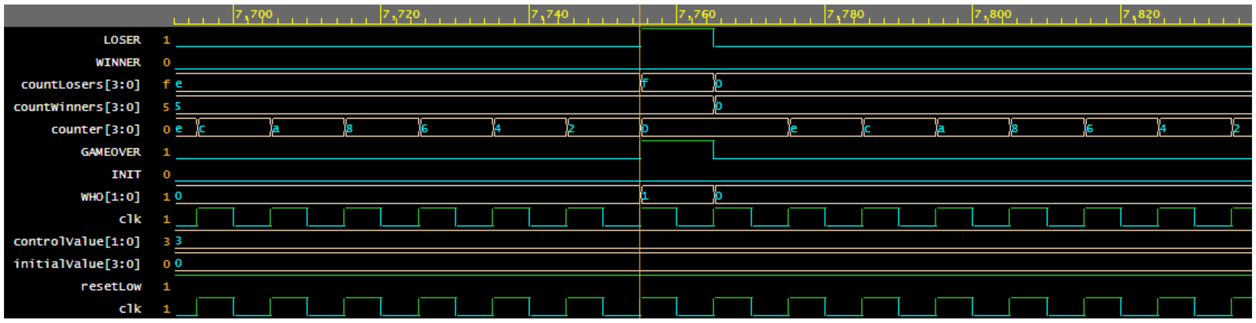
4.4. Counter Down by 2

@ 2450 ns “controlValue” is set to 3 which indicates the fourth mode (counter down by 2) so at the next positive edge of the clock cycle the counter starts decrementing by 2.



GAMEOVER since the “counterLosers” reached 15 (0xF), so the LOSER, GAMEOVER, WHO signals are raised for one cycle, WHO is set to 0b01 (0x01) since “counterLosers” reached 15 first.

GAMEOVER (reset all the counters and signals).


LOSER 1
WINNER 0
countLosers[3:0] f
countWinners[3:0] 5
counter[3:0] 0
GAMEOVER 1
INIT 0
WHO[1:0] 1
clk 1
controlValue[1:0] 3
initialValue[3:0] 0
resetLow 1
clk 1