Multi-Mode Counter Game Digital Design and Verification

Using SystemVerilog - Synopsys VCS - Modelsim

Author: Ahmed Abd-Elmotagly

Contents

Abstract

This project presents the design and verification of a multi-mode digital counter system using SystemVerilog. The counter supports four operational modes—counting up or down by 1s or 2s—based on a 2-bit control input. Additional functionality includes initialization with a user-defined value, detection of special conditions such as WINNER and LOSER states, and automatic game state management. The design includes mechanisms for generating single-cycle pulses upon specific counter values and tracking how many times each special condition is reached. Upon reaching a terminal condition, the system resets and restarts automatically. Additional specifications were added to the design: Asynchronous reset and clock input. A modular SystemVerilog testbench, built using interfaces and clocking blocks, was developed to verify functionality through simulation in Synopsys VCS. This report outlines the design architecture, verification methodology, test scenarios, and key results, demonstrating correct behavior across all functional modes.

Introduction

Digital counters are fundamental components in many digital systems, ranging from timers and event counters to finite-state machines and embedded control logic. This project focuses on the design and verification of a multi-mode counter system with gamelike behavior, implemented using SystemVerilog.

The counter is capable of operating in four modes: counting up or down by 1 or 2, depending on a 2-bit control signal. It features an initialization mechanism to load a starting value, and produces special output signals—WINNER or LOSER—when the counter reaches all ones or all zeros, respectively. These outputs are used to track gamelike win/loss events. Once a win or loss condition has occurred 15 times, a GAMEOVER signal is triggered and a 2-bit status output (WHO) indicates which side reached the terminal condition first. The system then resets and begins a new game cycle automatically.

To verify this design, a structured testbench was created, incorporating modular components such as a clock generator, interface, and clocking blocks for controlled signal timing. Simulation was performed using Synopsys VCS, and functional correctness was evaluated using waveform inspection and scenario-based testing.

This report documents the system specification, design architecture, verification strategy, simulation results.

1 Design Specifications

A multi-mode counter with configurable counting behavior and game-state logic. The counter supports four distinct modes determined by a 2-bit control input and includes functionality to initialize the counter, detect special counter values, and manage a simple game state based on WINNER and LOSER conditions. Figure [1] shows a block diagram illustrating the design.

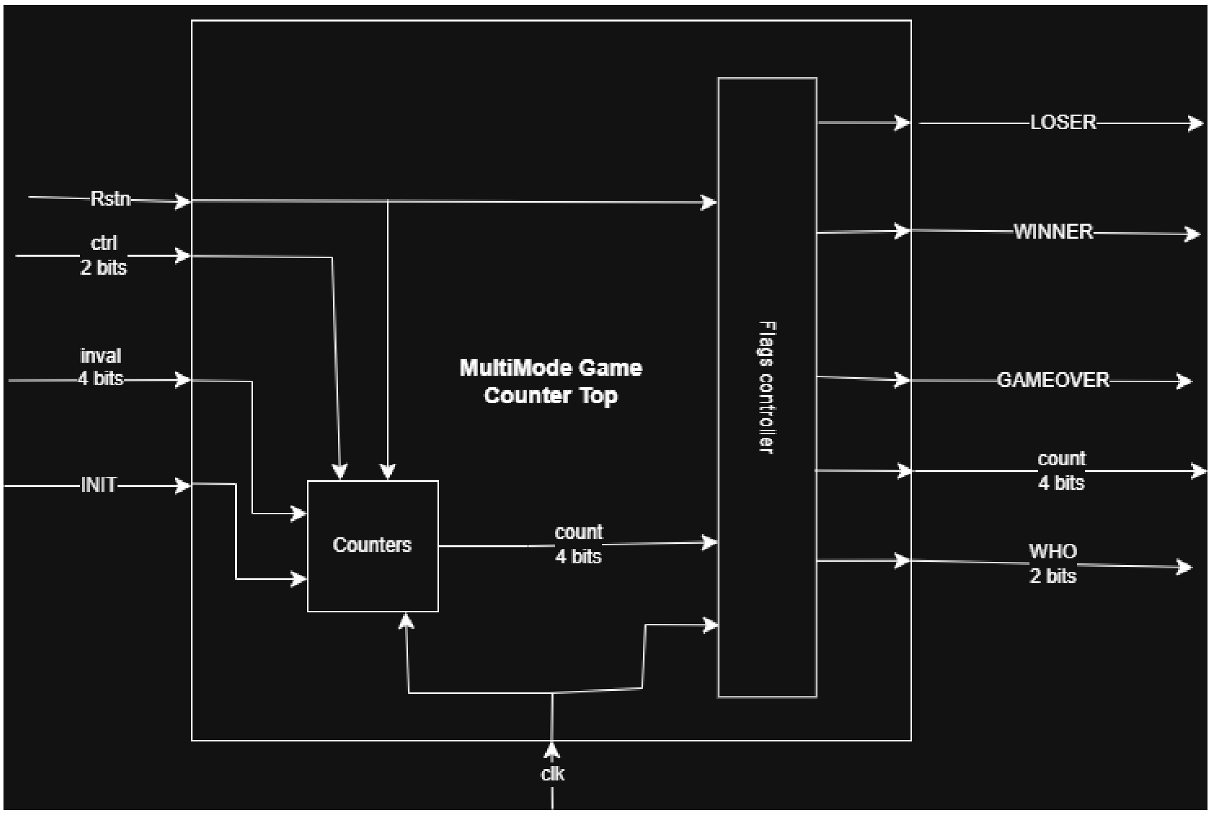


Figure 1: Design block diagram

2 Functional Requirements

- The counter supports four counting modes, selected via a 2-bit mode input:
  - 2’b00: Count up by 1
  - 2’b01: Count up by 2
  - 2’b10: Count down by 1
  - 2’b11: Count down by 2
- When INIT = 1, the counter loads the initial value from init val.
- When the counter reaches 0, the output LOSER is asserted high for exactly one clock cycle.
- When the counter reaches 4’b1111, the output WINNER is asserted high for exactly one clock cycle.
- WINNER and LOSER events are tracked using separate internal 4-bit counters.
- When either event counter reaches a value of 15:
  - GAMEOVER is asserted high.
  - WHO = 2’b01 if LOSER reached 15 first.
  - WHO = 2’b10 if WINNER reached 15 first.
- After a GAMEOVER event, the system resets all counters and restarts the game automatically.

2.1 Design Constraints and Assumptions

- All behavior is synchronous to the rising edge of clk.
- Asynchronous reset is added to help control the design.
- The reset value is assumed to be 7 (an average number between 0 and 15)
- Pulse outputs (WINNER, LOSER) must remain high for exactly one cycle.
- The system automatically resets and restarts after GAMEOVER.
- Counter is assumed to be just like event counters where all are 4-bit wide.

2.2 Inputs and Outputs

The following table summarize the interface signals of the design

Signal	Direction	Width	Description
clk	Input	1 bit	System clock. All logic is synchronous to this signal.
reset	Input	1 bit	Reset signal is selected to be asynchronous.
init _val	Input	4 bits	Initial value to load into the counter on INIT.
INIT	Input	1 bit	Load signal; when high, loads init val.
mode	Input	2 bits	Selects counting mode (up/down by 1 or 2).
count	Output	4 bits	Current value of the counter.
WINNER	Output	1 bit	High for one clock cycle when count equals 4'b1111.
LOSER	Output	1 bit	High for one clock cycle when count equals 0.
GAMEOVER	Output	1 bit	High when WINNER or LOSER event reaches count of 15.
WHO	Output	2 bits	Indicates result: 2'b01 = LOSER won, 2'b10 = WINNER won.

Table 1: Signal Descriptions

3 Verification Plan

The verification environment for this multi-mode counter design focuses on verifying the features that are implemented and operate with correct functionality. The Table below illustrates the verification plan.

Features		Checkers list	Stimulus	Priority
1.	Count	Count functions correctly at the four cases [UP 1, UP 2, DOWN 1, DOWN 2]	- ctrl = 2'b00 for 5 clk cycles - ctrl = 2'b10 for 5 clk cycles - ctrl = 2'b01 for 5 clk cycles - ctrl = 2'b11 for 5 clk cycles	High
2.	Load	Count = input value; after 1 cycle of INIT = 1	- inval = 4'b1010	High
3.	WINNER	=1 @ count = 4'b1111 for 1 clk	- inval = 4'b1111	Medium
4.	LOSER	=1 @ count = 4'b0000 for 1 clk	- inval = 4'b0000	Medium
5.	Game over	Whenever Loser or winner count reaches 15 GAMEOVER signal gets asserted for one cycle	- ctrl = 2'b11 for 90 cycles or inval = 4'b1111 & INIT =1 for 16 cycles - ctrl = 2'b01 for 90 cycles or inval = 4'b0000 & INIT = 1 for 16 cycles	Medium
6.	WHO	- 2'b10 @ GAMEOVER = 1 for winner count 15 - 2'b01 @GAMEOVER= 1 for loser count 15	Same stimuli for GAMEOVER	Medium
7.	Restart	- Count restarts from reset value (assumed to be 7) after reaching loser, winner or GAMEOVER - Loser and winner counts return to zero after GAMEOVER - WHO = 00	Same stimuli for GAMEOVER	High
8.	Reset	- Count = 7 - All flags are reset to zero - All internal counters are reset to zero	arstn = 0	High

4 Simulation Results

The verification process is divided into multiple test benches defined as programs.
The features are tested in different test benches for easier access to tests.
Verification environment implemented interfaces, clocking block, program block, mod-ports and assertions.
The simulation is debugged on both Modelsim and Synopsys VCS.

4.1 Testbench 1 [arst_load_tb]

This testbench program checked on the features of:

- Asynchronous active low reset.
- Load.
- Winner condition.
- Loser condition.

A concurrent assertion checked that winner and loser flags are never asserted at the same time.

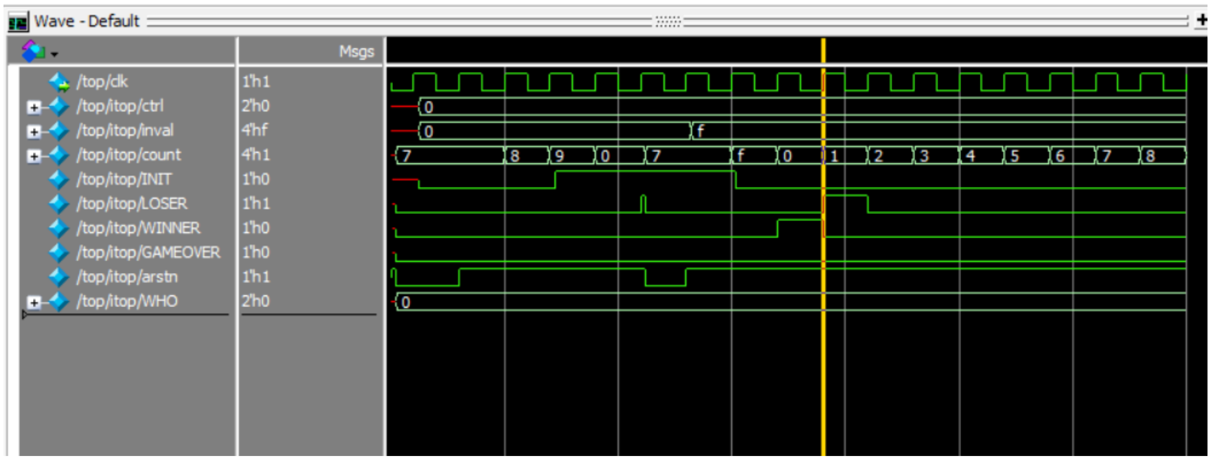
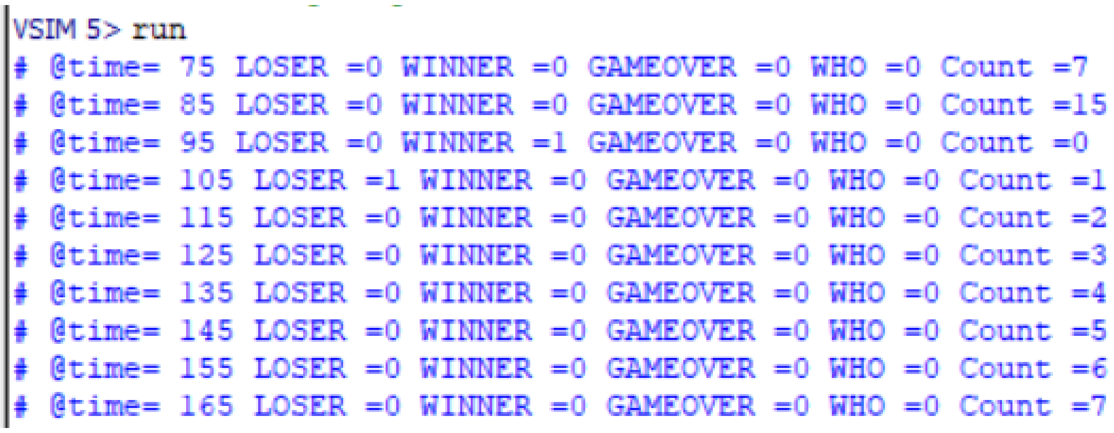


Figure 2: Simulation waves on modelsim



```
VSIM 5> run
# @time= 75 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =7
# @time= 85 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =15
# @time= 95 LOSER =0 WINNER =1 GAMEOVER =0 WHO =0 Count =0
# @time= 105 LOSER =1 WINNER =0 GAMEOVER =0 WHO =0 Count =1
# @time= 115 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =2
# @time= 125 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =3
# @time= 135 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =4
# @time= 145 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =5
# @time= 155 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =6
# @time= 165 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =7
```

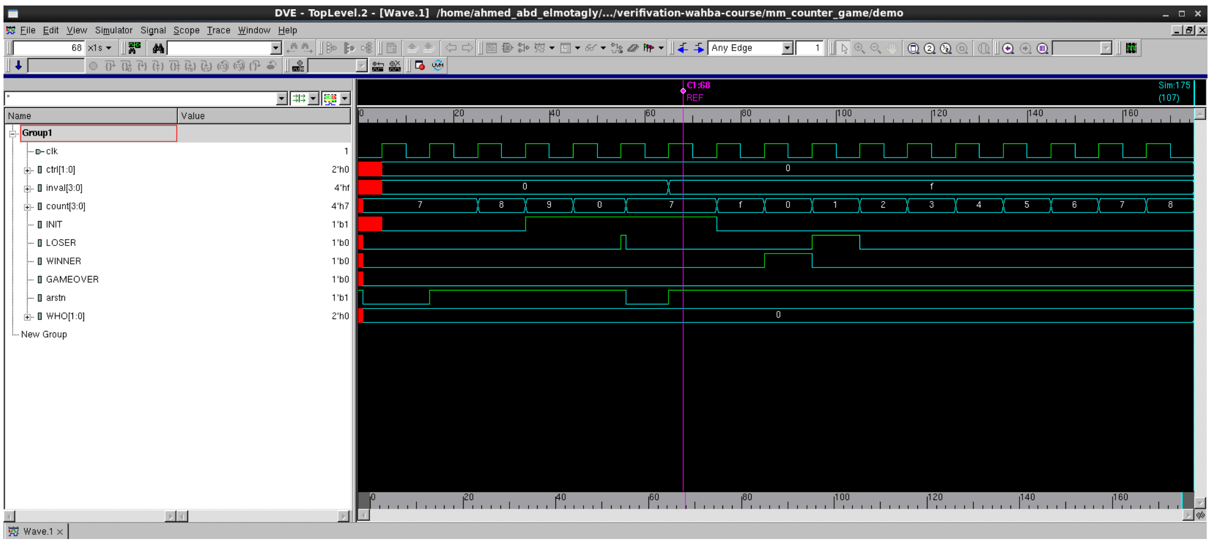


Figure 3: Simulation waves on Synopsys VCS (viewed in DVE)

```
The design has assertions or cover properties.
The assertion browser can be used to view them. Click on the assertion toolbar button or use the menu 'Window->Panes->Assertion' to open it.
The file '/home/ahmed_abd_elmotagly/ahmed_a_a/verifivation-wahba-course/mm_counter_game/inter.vpd' was opened successfully.
@time= 75 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =7
@time= 85 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =15
@time= 95 LOSER =0 WINNER =1 GAMEOVER =0 WHO =0 Count =0
@time= 105 LOSER =1 WINNER =0 GAMEOVER =0 WHO =0 Count =1
@time= 115 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =2
@time= 125 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =3
@time= 135 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =4
@time= 145 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =5
@time= 155 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =6
@time= 165 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =7
@time= 175 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =8
$finish at simulation time                175
Simulation complete, time is 175.
           V C S   S i m u l a t i o n   R e p o r t
```

4.2 Testbench 2 [count_tb]

This testbench program checks all modes of counting are functioning correctly:

- Count up by 1
- Count up by 2
- Count down by 1
- Count down by 2

Some immediate assertions are added on the testbench to check on the counting values.
The important signals are monitored to double check values.
Simulation debugged on both modelsim and VCS.

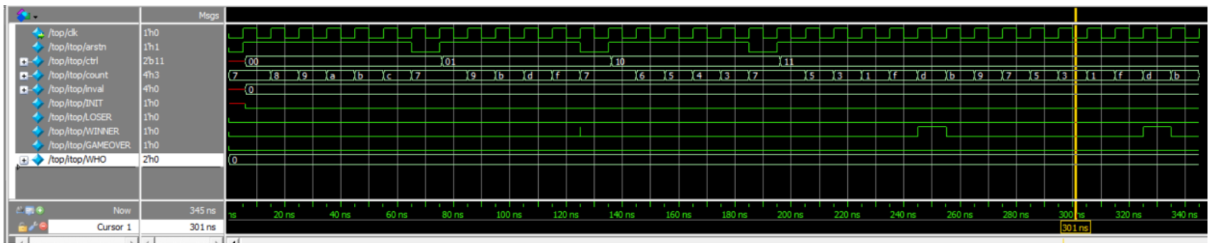
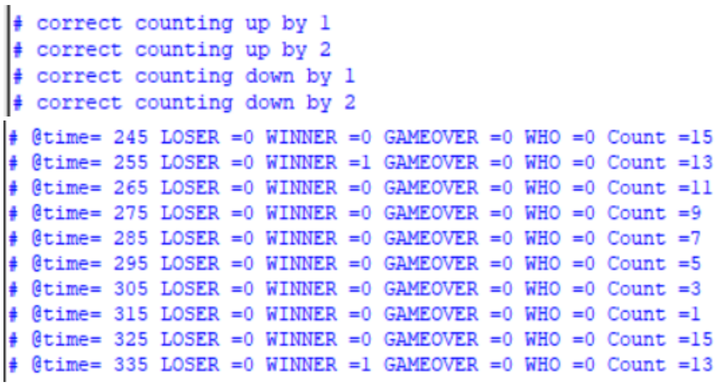
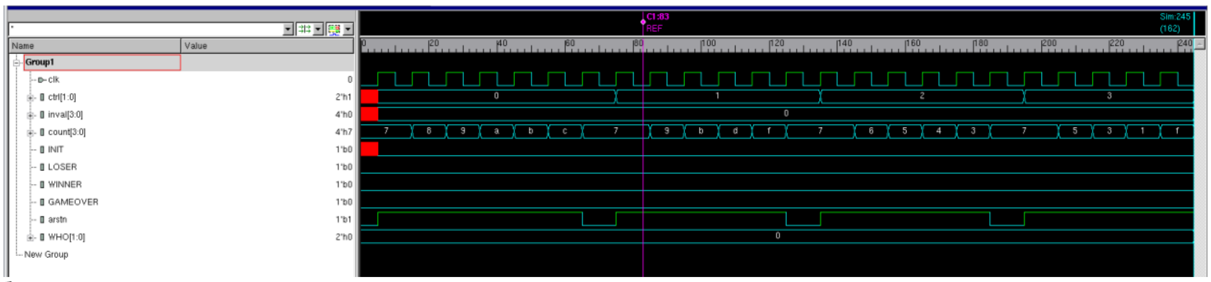
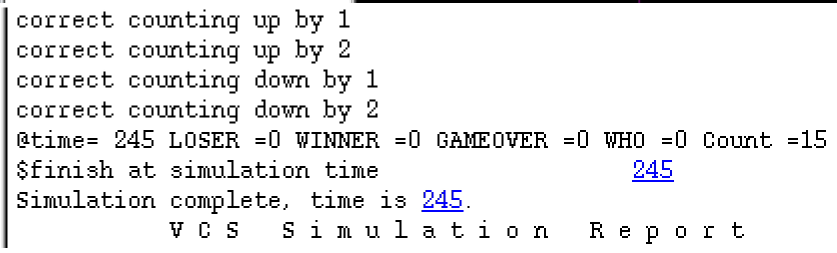


Figure 4: checking all counting modes are working



```
# correct counting up by 1
# correct counting up by 2
# correct counting down by 1
# correct counting down by 2
# @time= 245 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =15
# @time= 255 LOSER =0 WINNER =1 GAMEOVER =0 WHO =0 Count =13
# @time= 265 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =11
# @time= 275 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =9
# @time= 285 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =7
# @time= 295 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =5
# @time= 305 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =3
# @time= 315 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =1
# @time= 325 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =15
# @time= 335 LOSER =0 WINNER =1 GAMEOVER =0 WHO =0 Count =13
```



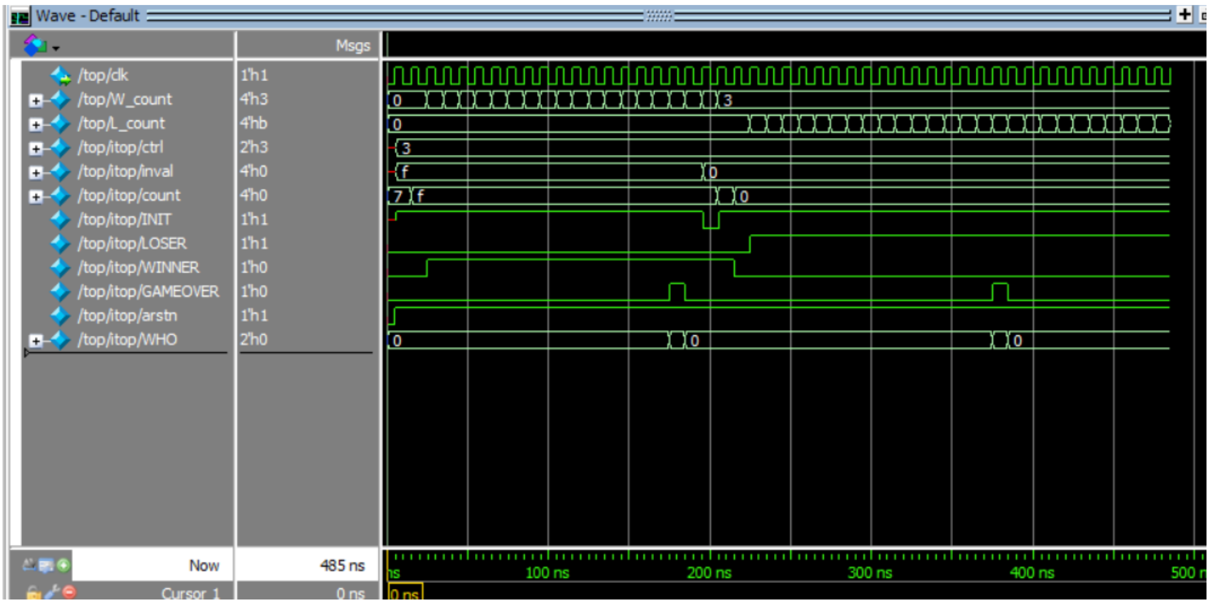


```
correct counting up by 1
correct counting up by 2
correct counting down by 1
correct counting down by 2
@time= 245 LOSER =0 WINNER =0 GAMEOVER =0 WHO =0 Count =15
$finish at simulation time                    245
Simulation complete, time is 245.
           V C S   S i m u l a t i o n   R e p o r t
```

Note: $monitor does not function as intended in VCS , instead always@ $display can be used

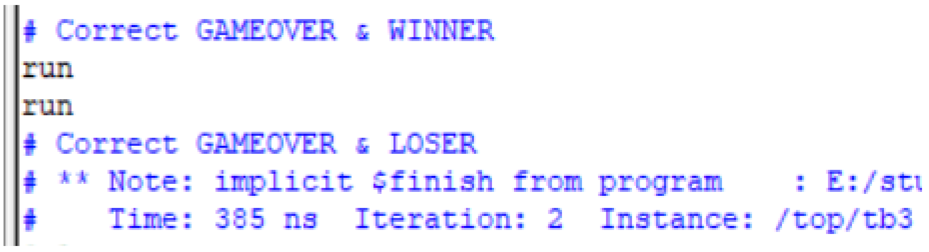
4.3 Testbench 3 [mm_end_tb]

Tested the end of the game and the restart at the cases of Game over with winner and loser
Used immediate assertions to ensure on the results.

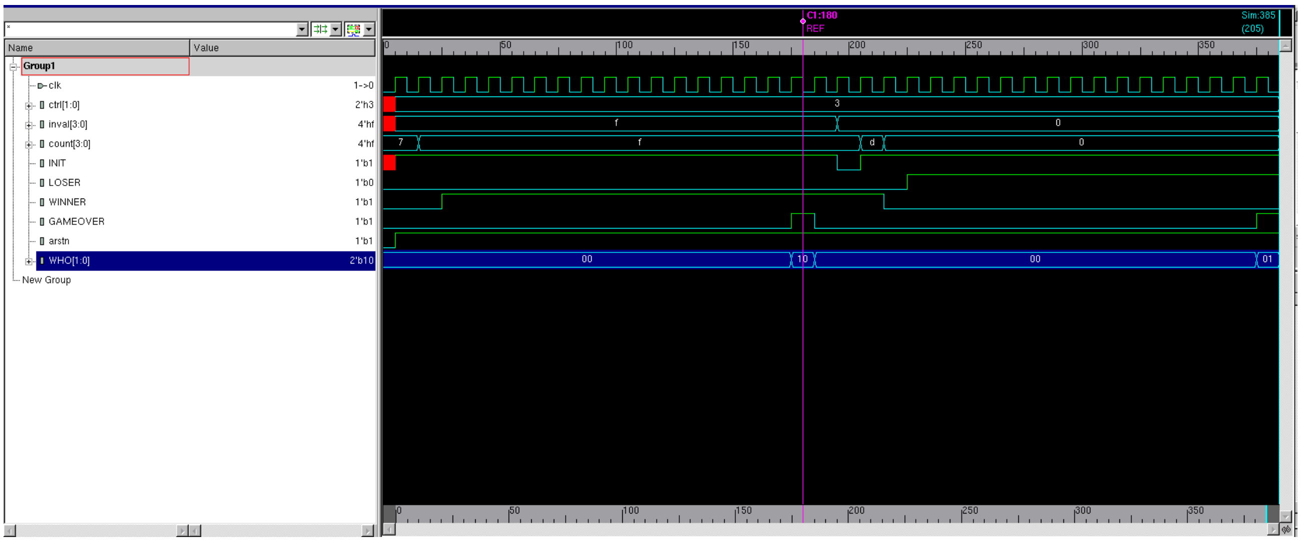


```
Wave - Default
                      Msgs
/top/clk              1'h1
/top/W_count          4'h3
/top/L_count          4'hb
/top/itop/ctrl        2'h3
/top/itop/inval       4'h0
/top/itop/count       4'h0
/top/itop/INIT        1'h1
/top/itop/LOSER       1'h1
/top/itop/WINNER      1'h0
/top/itop/GAMEOVER    1'h0
/top/itop/arstn       1'h1
/top/itop/WHO         2'h0
Now                   485 ns
Cursor 1              0 ns
```

Figure 5: Results wave forms on Modelsim



```
# Correct GAMEOVER & WINNER
run
run
# Correct GAMEOVER & LOSER
# ** Note: implicit $finish from program    : E:/stu
#    Time: 385 ns  Iteration: 2  Instance: /top/tb3
```



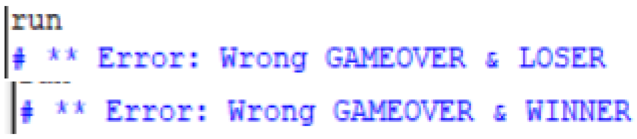
```
Name            Value
Group1
  clk           1->0
  ctrl[1:0]     2'h3
  inval[3:0]    4'hf
  count[3:0]    4'hf
  INIT          1'b1
  LOSER         1'b0
  WINNER        1'b1
  GAMEOVER      1'b1
  arstn         1'b1
  WHO[1:0]      2'b10
New Group
```

Figure 6: Results in Synopsys vcs

5 Notes

In this project the RTL designer is the same person as the verification engineer, so while verifying, when any bugs appeared in the functionality I returned to modify the design again; here I demonstrate one of the bugs as an example:

In both cases of Game over the Game over and Who flags were wrongly functioning as they never change.
The assertions did their job and caught this:



```
run
# ** Error: Wrong GAMEOVER & LOSER
# ** Error: Wrong GAMEOVER & WINNER
```

The problem was the design had a race condition in the flags controller were the signals were written on to procedural statements at the same time.