Asteroid Dodge Game with Multi-User ROM-based Authentication, RAM-based Score Tracking, Difficulty Levels, and Adjustable Timer

Team Q4

Design Document

Nhat Nguyen PSID: \*\*\*4700

Christopher Andrew PSID: 1885918

Gantabya Kadel PSID: \*\*\*8522

Matthew Andersen PSID: 1812071

*ECE 5440 – Advanced Digital Design*

*Spring 2022*

Contents

[Introduction 4](#_Toc103072088)

[About 4](#_Toc103072089)

[Gameplay 4](#_Toc103072090)

[Game Setup Instructions 4](#_Toc103072091)

[Game Login Instructions 5](#_Toc103072092)

[Playing the game 5](#_Toc103072093)

[System Architecture Design 6](#_Toc103072094)

[AccessController module 8](#_Toc103072095)

[MultiUserAuthentication 9](#_Toc103072096)

[OneSecondTimer Module 13](#_Toc103072097)

[GameController Module 15](#_Toc103072098)

[AccessController Simulation 17](#_Toc103072099)

[LED Bar Timer Module 23](#_Toc103072100)

[Object Shifter module 24](#_Toc103072101)

[Object Shifter Simulation 26](#_Toc103072102)

[RNG\_LFSR\_3b module 27](#_Toc103072103)

[Decoder\_3bTo7bShape module 27](#_Toc103072104)

[Decoder\_3bTo7bShape simulation 28](#_Toc103072105)

[Crash Detector module 28](#_Toc103072106)

[CrashDetector Simulation 30](#_Toc103072107)

[LED\_Animator module 31](#_Toc103072108)

[PlayerPosition module 32](#_Toc103072109)

[PlayerPosition Simulation 33](#_Toc103072110)

[Score\_Control\_Unit 33](#_Toc103072111)

[Score\_Ram 35](#_Toc103072112)

[Score\_Track 36](#_Toc103072113)

[Score\_Tracker Simulation 39](#_Toc103072114)

[Digit\_2\_Count\_Up 41](#_Toc103072115)

[Digit\_Count\_Up 42](#_Toc103072116)

[Best\_Score\_Display 43](#_Toc103072117)

[Player\_Score\_Display 44](#_Toc103072118)

[Decoder\_7SegAnode 45](#_Toc103072119)

[Decoder\_7SegAnode Simulation 46](#_Toc103072120)

[Seg\_7\_Control 46](#_Toc103072121)

[Variable\_Timer 47](#_Toc103072122)

[Varible\_Timer Simulation 48](#_Toc103072123)

[ButtonShaper module 48](#_Toc103072124)

[ButtonShaper simulation 49](#_Toc103072125)

[Timer\_1ms 50](#_Toc103072126)

[Timer\_100ms 51](#_Toc103072127)

[FPGA Board Testing Results 52](#_Toc103072128)

[Video Demo Links 55](#_Toc103072129)

[Conclusion 56](#_Toc103072130)

# Introduction

A close-up of a circuit board

Description automatically generated with medium confidence

Power Switch

LED Timer countdown

Game Display

Player Display

ID/Password input

Choose level length

Enter ID/Password

Log Out

Game Start/Toggle Player position

Reset

Fig. 1. Diagram of User Interface for Asteroid Dodge Game.

## About

This project details a side-scrolling, asteroid-dodging space game, where the goal is to continually travel right and avoid the debris flying at you for as long as possible. Score is tracked across multiple users, and the global best score is also tracked and displayed. Additional features include difficulty increment, a user-adjustable game timer, and LED display animations.

## Gameplay

### Game Setup Instructions

1. Ensure the programming switch located on the left side of the board is switched to the run position
2. Ensure that the DC power Jack is plugged into the board and plugged into a powered American standard 120 [V] wall outlet
   1. Alternatively the device can be powered by the USB blaster port
3. If there are no LEDs lit, press the red power switch on the upper right side of the board
4. If nothing happens, check that the power cable is plugged in at both ends.
5. If there is a single blue LED lit and the 7 segment displays show the word “ID” the game is loaded correctly
6. Press the right most push button switch, KEY0, to restart the game and allow a player to log into the game

### Game Login Instructions

1. To log into the game we must first enter our desired username, a 4 digit number.
2. Each digit is submitted by selecting the username digit in binary using the leftmost 4 slide switches.
3. The username is then submitted to the game using the password/enter ID button KEY2
4. Valid usernames are
   1. 8522
   2. 4700
   3. 5928
   4. 2071
   5. FFFF
5. Once you have entered a valid ID you are then taken to the passcode screen and the 7 segment displays will display the word “PSCO”
6. Next you must enter the 6 digit passcode associated with the previously entered username
7. Password digits are selected using the leftmost 4 slide switches
8. Password digits are submitted using the password/enter ID button KEY2
9. Valid passwords, password are associated with the ID that shares the same position in the list
   1. A54E32
   2. EEE420
   3. F24630
   4. AAB431
   5. FFFFFF
10. If you successfully log into the game you will be greeted with the global score display then the player score display a few seconds latter
11. You will then be able to play the game

### Playing the game

1. First you must select the size of your fuel gage using the right most set of 4 slide switches in binary
   1. Each point of fuel represents 5 seconds of level time. Default is 10 lights.
   2. Longer levels are more difficult
2. In order to play a game after logging in the game start/toggle player position button KEY1 must be pressed
3. In order to score a point the player must dodge oncoming objects on the 7 segment displays until the fuel gage runs out
4. If the player succeeds in dodging all asteroids they are awarded 1 point and their score will be displayed
   1. The LED’s in the countdown timer will blink if the player beats their personal best
   2. The LED’s in the countdown timer will fill up if the player beats the global best
5. If they player fails at completing a level they will see the global score screen for a few seconds before returning to the personal score screen
6. The player can log out of the game at any time game play is not happening.

# System Architecture Design

The system architecture uses a total of eight slide switches and four buttons as input, and all six 7-segment displays and the full 10 LED light bar for output. The top-level modules were developed in four distinct areas: Multi-user access and game control, scoring, gameplay, and timing/LED display. The full architecture is shown in Fig. 2.

The AccessController module interfaces with ROM and controls user authentication, gameplay state, and display states. The Score Control Unit controls scoring and interfaces with RAM to keep track of high scores. The LED Bar Timer and Variable timer control the variable length and speed of the levels, and the Object Shifter and Crash Detector control the objects and interactions when the gameplay is enabled. All of these modules interface together for a full game experience, and the connections are detailed in the module descriptions that follow.

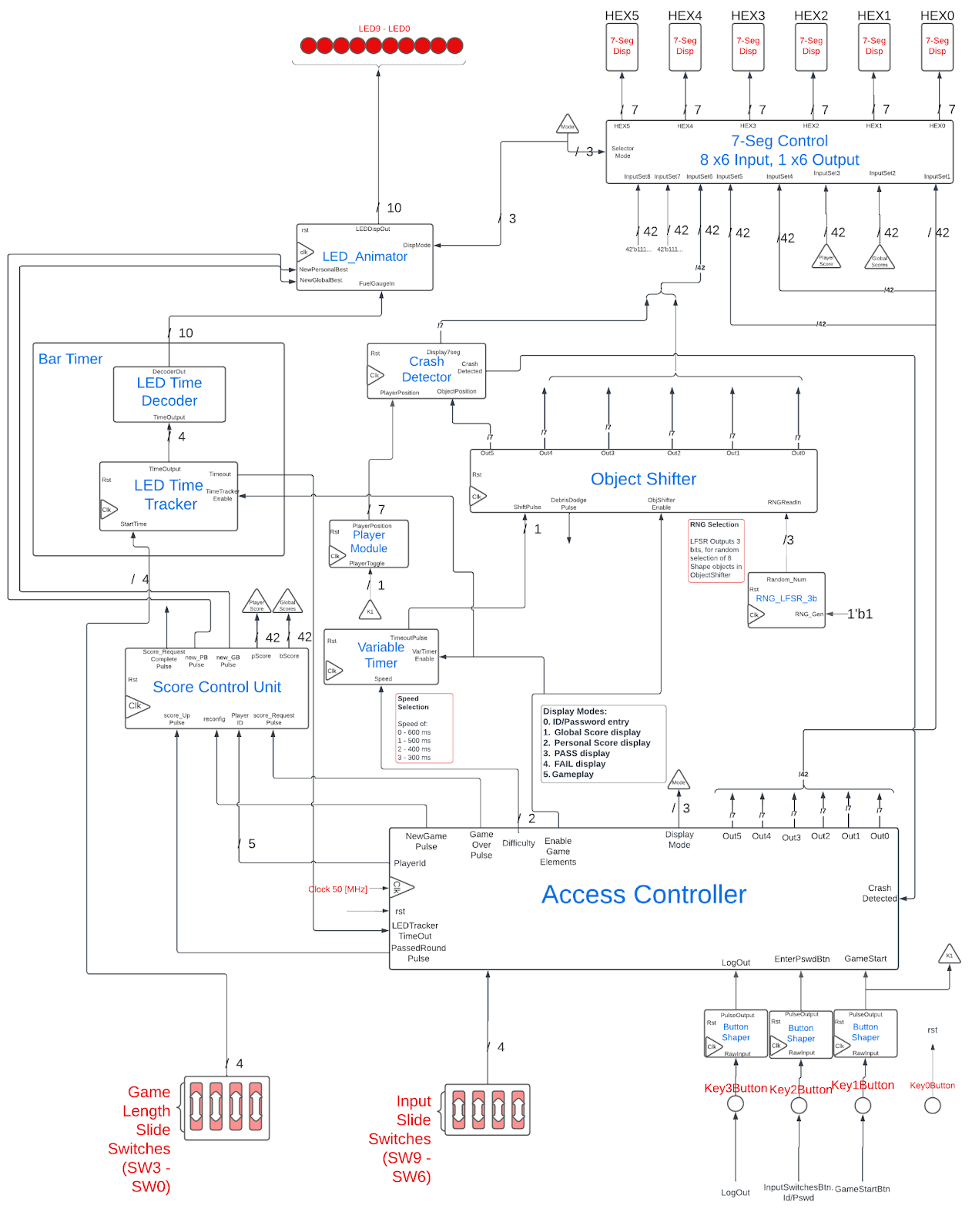


Fig. 2. Top-level module diagram with user inputs and outputs in red.

The signals “clk” and “rst” are system-wide and described as follows:  
clk: System clock signal at 50 [MHz]. 50% duty cycle square wave.  
rst: System reset button signal. Expected to be HIGH until the button is pressed which results in a long LOW signal.

## AccessController module

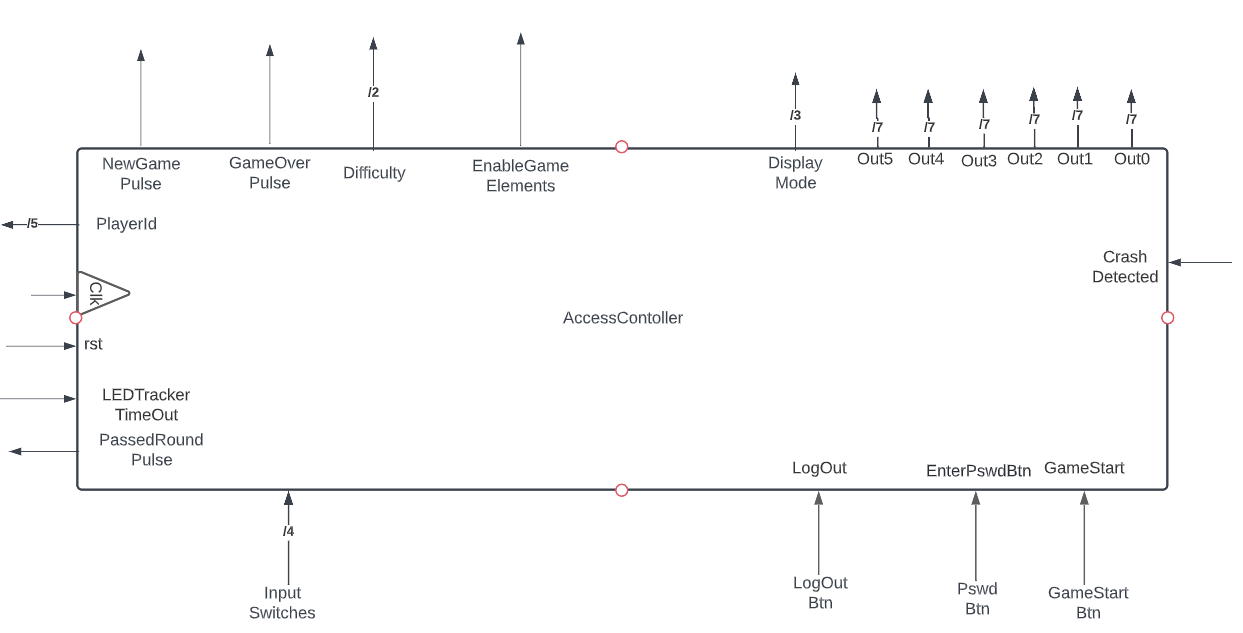
The AccessController, as the name suggests, controls access to the system. This module is responsible for authenticating the user and then allows the game to be played after GameStartBtn is pressed. This module instantiates two “daughter-board” modules, MultiUserAuthentication and GameController, alongside the OneSecondTimer module. The main internal modules are MultiUserAuthentication and GameController modules, whereas the OneSecondTimer is simply used as a “time tracker” for a component of GameController module. The MultiUserAuthentication is responsible for authenticating the user, while GameController is responsible for controlling/enabling in game functions. Fig. 3 illustrates the AccessController module is all the input and output signals defined. Signals without a width indication are 1-bit. The discussion of the inputs and outputs of this module is done immediately after Fig. 3. Finally, immediately following this module are the discussion of the modules instantiated by the AccessController, and in turn module instantiated by the inner module of AccessController. Fig. 3 shows this module with all of its input and output signals defined.

Fig. 3. AccessController module with Input and Output signals defined.

The inputs and outputs are as follows:

**Inputs:**  
clk, rst: system inputs described above.  
InputSwitches: On Board FPGA Slide Switches. 4 slide switches are joined together to make a 6-bit bus signal for the AccessController.   
LogOutBtn: 1-bit Shaped signal input of Key 3’s input. Users press this button to “logoff” the system.  
PswdBtn: 1-bit Shaped signal input of Key 2’s input. Users press this button to enter their Id and password.  
GameStartBtn: 1-bit Shaped signal input of Key 1’s input. Users press this button to start the game.  
LEDTrackerTimeOut: 1-bit signal from the LEDTimeTracker module. Used to indicate the completion of a level.  
CrashDetected: 1-bit signal from the CrashDetector module. Used to indicate player’s collison and game end.

**Outputs**:  
PlayerId: 5-bit output bus signal. This is the ROM address of matched ID/Password sequence.  
NewGamePulse: 1-bit single cycle active high signal. Used to indicate a game has started for the first time.  
GameOverPulse: 1-bit single cycle active high signal. Used to indicate that a game has finished.  
Difficulty: 2-bit output signal. The difficulty length for the game.  
EnableGameElements: 1-bit steady active high signal. Used to enable game elements.  
DisplayMode: 3-bit bus signal to signify the current mode/state of the GameController module, instantiated by the AccessController module, to be used elsewhere in the system.  
Out0 – Out5: 7-bit decoded signal for the six seven segment displays on board the FPGA, respectively.  
PassedRoundPulse: 1-bit active high single cycle pulse. Used to indicate that a level has been passed.

### MultiUserAuthentication

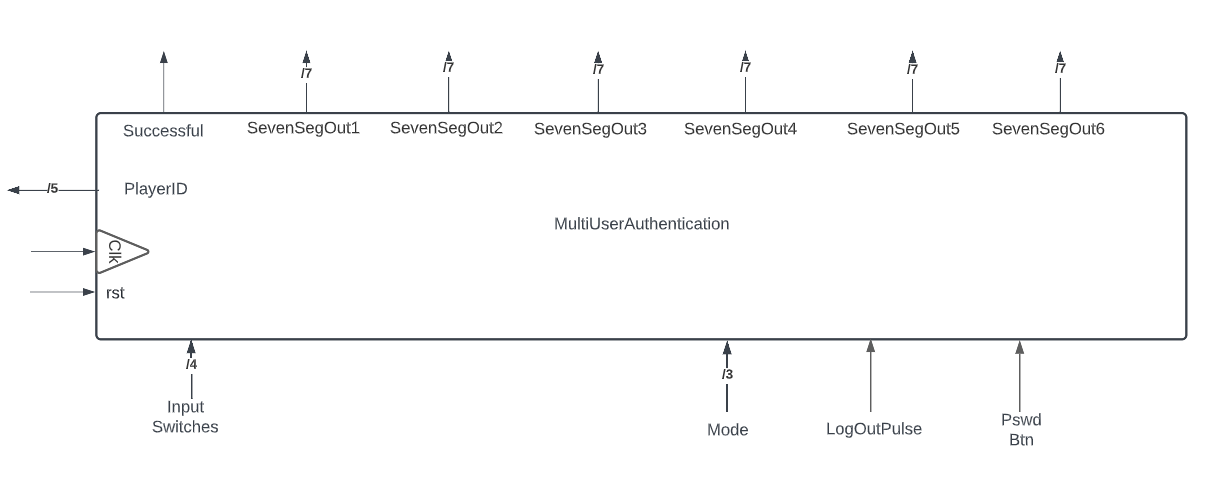
This module is one of the two “daughter-board” modules of the top-level module, AccessController module. And it is solely responsible for validating user entered id and password combination to authenticate the user into the system. Fig X illustrates this module with all the inputs and output signals defined. Signals without a width indication are 1-bit wide. This module instantiates three modules, CheckID, CheckPswd, Display. CheckID module is used to verify user’s ID input, and CheckPswd module is used to verify user’s password input. Lastly, Display module is used to output decoded seven segment display signals that are generated by CheckID, CheckPswd signals. Fig. 4 shows this module with all of its input and output signals defined.****

Fig. 4. The Symbol for MultiUserAuthentication Module with all inputs and outputs defined.

The inputs and outputs are as follows:

**Inputs:**

clk, rst: sytem inputs described above.  
InputSwitches: On Board FPGA Slide Switches. 4 slide switches are joined together to make a 6-bit bus signal. Passed to this module by its parent module AccessController.  
Mode: 3-bit bus signal. Display Mode of the FSM of GameController module. This module passes it to an internal module it instantiates, Display module.  
LogOutPulse: 1-bit active high single cycle signal. Used to log out.  
PswdBtn: 1-bit active high single cycle signal. Used to enter-in the user entered values via switches. Button shaped signal.

**Outputs:**

PlayerId: 5-bit output bus signal. This is the ROM address of matched ID/Password sequence.  
Successful: 1-bit active high steady signal. Used to indicate successful authentication.  
SevenSegOut1 – SevenSegOut6: 7-bit decoded signal for the six seven segment displays on board the FPGA, respectively.

#### CheckID Module

This module is one of the three modules instantiated by the MultiUserAuthentication module. This module is responsible for validating user entered ID sequence and then providing a reference ROM address (InternalID/PlayerID) for the CheckPswd Module to match against. This is one of the two modules used to authenticate the user into the system. Fig. 5 shows this module with all of its input and output signals defined. Finally, the illustratrion of the CheckID module’s Finite State Machine (FSM) can be found in Fig. 6.

Diagram

Description automatically generated

Fig. 5. CheckID module with Input and Output signals defined.

The inputs and outputs are as follows:

**Inputs:**

clk, rst: system inputs described above.  
InputSwitches: On Board FPGA Slide Switches. 4 slide switches are joined together to make a 6-bit bus signal.   
PswdBtn: 1-bit button shaped signal used to pass the user input to internal system.  
LogOutPulse: 1-bit active high single cycle pulse to indicate logout.

**Outputs:**

InternalID: 5-bit but signal. It’s the ROM address of matched user inputted ID sequence.  
IDOK: 1-bit active high steady signal. Used to indicate an ID has been matched. Fed to CheckPswd module to begin checking for password.

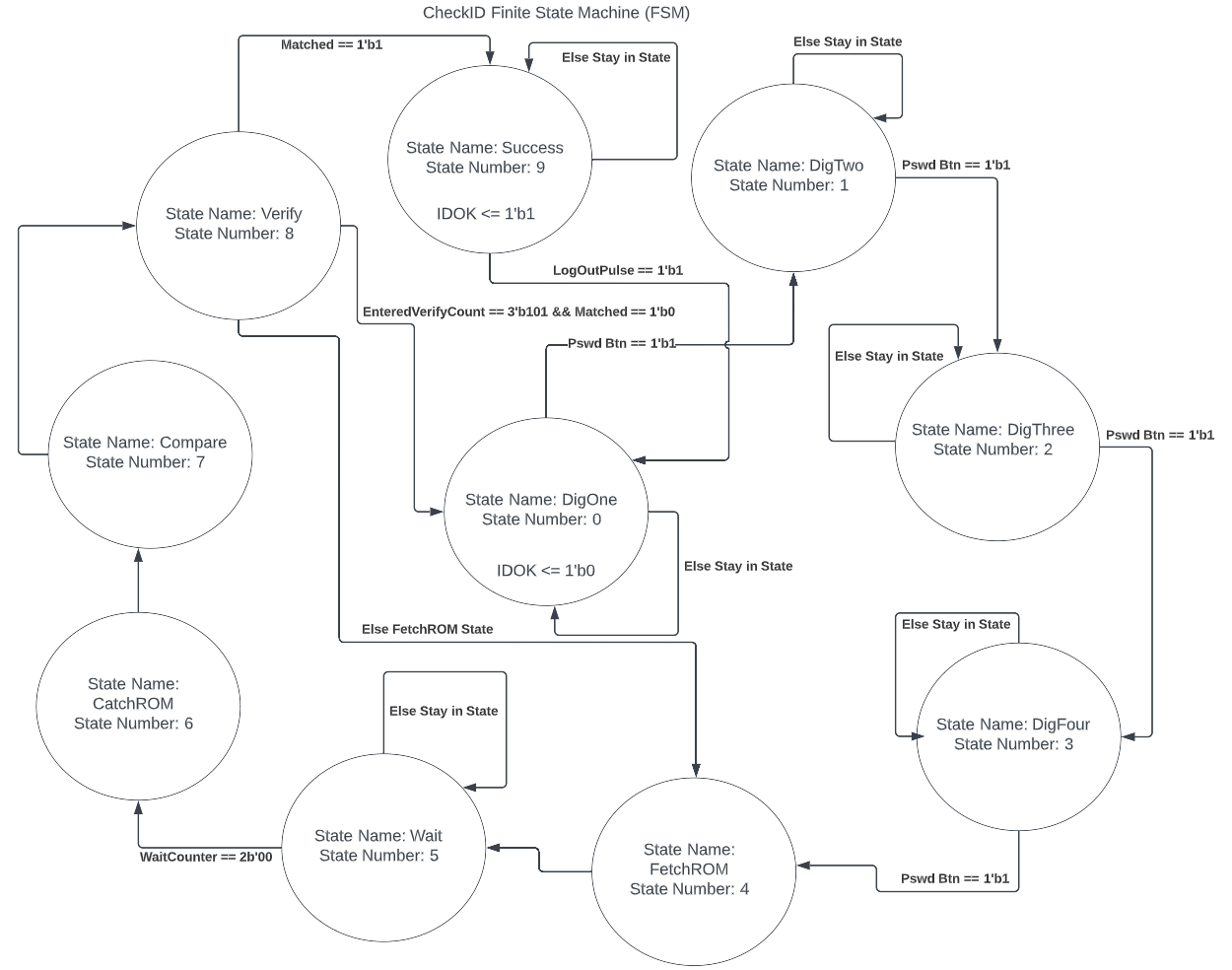


Fig. 6. Finite State Machine (FSM) of the CheckID module.

The CheckID module FSM takes all four ID numbers into internal registers before cycling through the ROM entries to find a match. If no match is found, the ID state still displays on the 7-segment display and the player can try again.

#### CheckPswd Module

This module is one of the three modules instantiated by the MultiUserAuthentication module. It’s the second, and last module used to verify user entered ID/Password combination and log the user in. After successful loggin, provides a 1-bit signal to the Gamecontroller module to begin the game processes. Fig. 7 shows this module with all of its input and output signals defined. Finally, the illustratrion of the CheckPswd module’s Finite State Machine (FSM) can be found in Fig. 8.

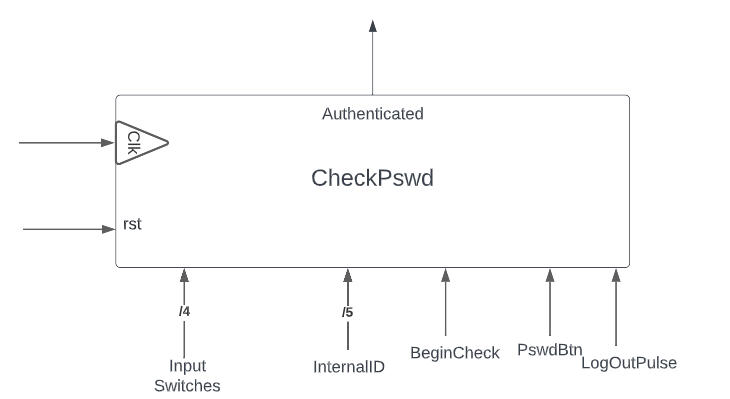


Fig. 7. CheckPswd module with Input and Output signals defined.

The inputs and outputs are as follows:

**Inputs:**

clk, rst: system inputs described above.  
InputSwitches: On Board FPGA Slide Switches. 4 slide switches are joined together to make a 6-bit bus signal.   
BeginCheck: 1-bit active high steady signal. Provided by the CheckID module to indicate that the user entered ID has matched and to, now, begin the password matching process.  
PswdBtn: 1-bit button shaped signal used to pass the user input to internal system.  
LogOutPulse: 1-bit active high single cycle pulse to indicate logout.  
InternalID: 5-bit but signal. It’s the ROM address of matched user inputted ID sequence. Used to verify is matched password belongs to the address of matched ID.

**Output:**

Authenticated: 1-bit activr high steady signal that activates the Game Controller.

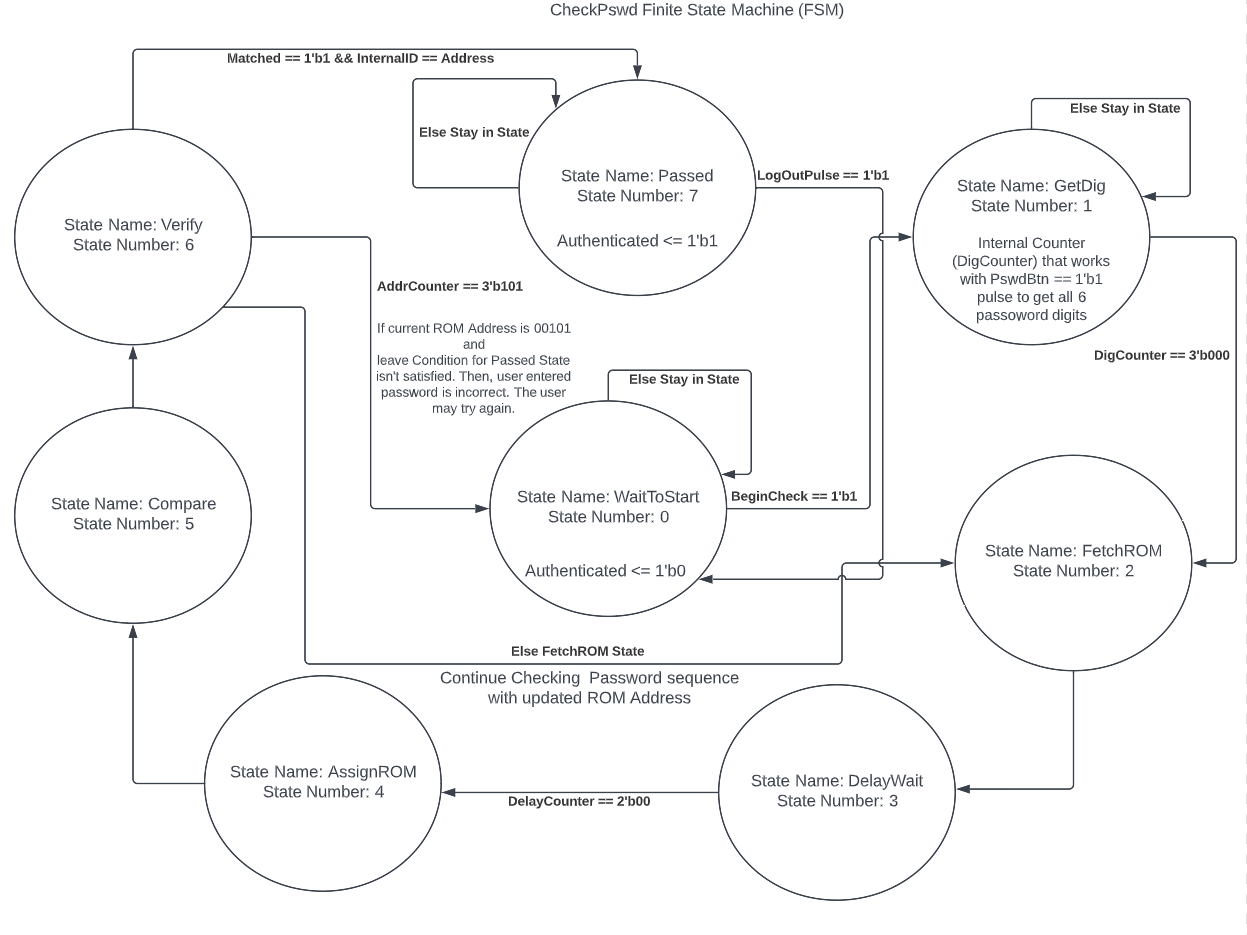


Fig. 8. Finite State Machine (FSM) for the CheckPswd module.

The finite state machine for the CheckPswd module is similar to the CheckID module, but with 6 states for the user to enter their six hexadecimal digits.

### OneSecondTimer Module

This module produces a single cycle pulse every 1 second, or 50,000,000 clock cycles. An enable pin is sent to the module to enable its internal mechanism, allowing the system to start the timer whenever it is necessary. The symbol is shown in Fig. 9.

Diagram

Description automatically generated

Fig. 9. OneSecondTimer module with Input and Output signals defined.

The inputs and outputs are as follows:

**Inputs:**

Enable: 1-bit signal. Enables the module to start counting when signal is HIGH, or 1.

**Outputs:**

TimeoutPulse: Produces a single cycle pulse from LOW to HIGH to LOW every 1 second on a 50 MHz clock.

#### *Display Module*

This module is responsible for displaying “ID,” “PSCO,” “SUCC,” and “FAIL” via the six onboard seven segment displays. And it is one of three module instansiated by the MultiUserAuthentication module. This the only combinational logic module instatiated inside the AcessController module. Fig. 10 shows this module with all of its input and output signals defined.

Timeline

Description automatically generated

Fig. 10. Display module with Input and Output signals defined.

The inputs and outputs are as follows:

**Inputs:**

clk, rst: system inputs described above.  
IdOk: 1-bit signal. Used to indicate id validation status so that they display can switch between “ID” and “PSCO.”  
PswdOk: 1-bit signal. Used to indicate password validation status can switch from “PSCO” to other displays.  
Mode: 3-bit bus signal. Used to display “SUCC” or “FAIL” depeniding on the mode.

**Outputs:**

Hex1Out – Hext6Out: 7-bit decoded signal for the six seven segment displays on board the FPGA, respectively.

#### LetterSevenSegDecoder Module

This module is instansiated by the Display module and is explicitly used as a decoder by said module. It is a custom decoder for the project that decodes A-Z (barring some letters) and 1 – 9 digits. This is also a combinational logic module. Fig. 11 shows this module with all of its input and output signals defined.

Diagram

Description automatically generated

Fig. 11. LetterSevenSegDecoder module with Input and Output signals defined.

The inputs and outputs are as follows:

**Inputs:**

DecoderInput: 5-bit bus signal. Used to determine one of the possbile 30 ouputs, include default.

**Outputs:**

DecoderOutput: Decoded 7-bit bus signal for the Seven Segment Display.

### GameController Module

One of the two gatekeeping module for the game. It is responsible for enabling game feature access after authentication and user presses to button to start the game. This module is instantiated by the AccessController Module. And, game access can only begin if a Authenticated signal is provided to it but the MultiUserAuthentication module. Nothing will occur before that, even if the game button is being pressed. The module also provides display mode outputs, to the rest of the system for various displays seetings. A unique feature is that, it uses a one second timer pulse to transition between 2 display states when the user logs in as well as when the user passes a level. Fig. 12 shows this module with all of its input and output signals defined. Finally, the illustratrion of the GameController module’s Finite State Machine (FSM) can be found in Fig. 13.

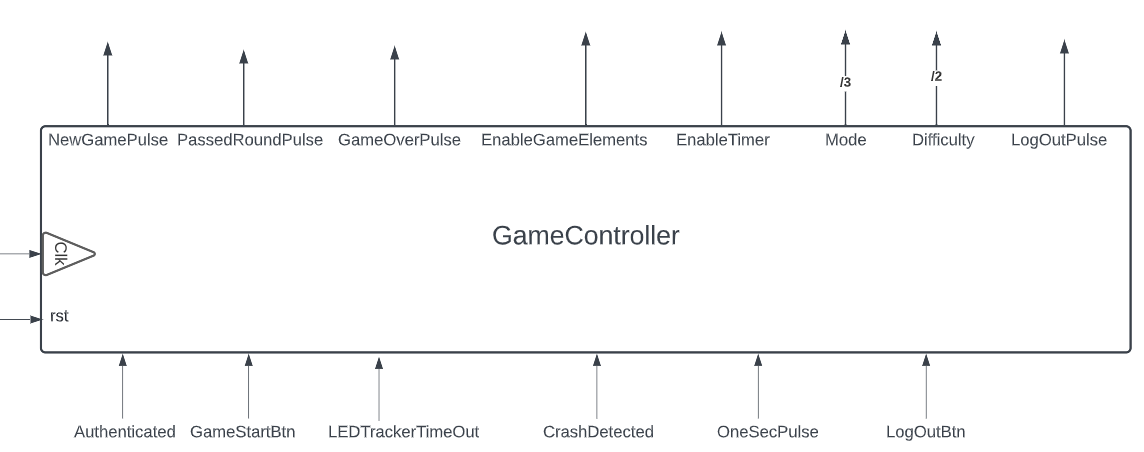


Fig. 12. GameController module with Input and Output signals defined.

The inputs and outputs are as follows:

**Inputs:**

clk, rst: system inputs described above  
Authenticated: 1-bit steady active high signal. Used to indicate authentication status of the user.  
GameStartBtn: 1-bit Shaped signal input of Key 1’s input. Users press this button to start the game.  
LEDTrackerTimeOut: 1-bit signal from the LEDTimeTracker module. Used to indicate the completion of a level.  
CrashDetected: 1-bit signal from the CrashDetector module. Used to indicate player’s collison and game end.  
OneSecPulse: 1-bit single cycle active high signal high. Used to indicate the passing of a second.  
LogOutBtn: 1-bit Shaped signal input of Key 3’s input. Users press this button to “logoff” the system.

**Outputs:**

NewGamePulse: 1-bit single cycle active high signal. Used to indicate a game has started for the first time.  
PassedRoundPulse: 1-bit active high single cycle pulse. Used to indicate that a level has been passed.  
GameOverPulse: 1-bit single cycle active high signal. Used to indicate that a game has finished.  
EnableGameElements: 1-bit steady active high signal. Used to enable game elements.  
EnableTimer: 1-bit steady active high signal. Used to enable the timer, instantiated by the AccessController module, the parent module.  
Mode: 3-bit bus signal to signify the current mode/state of FSM to be used elsewhere in the system.  
Difficulty: 2-bit output signal. The difficulty length for the game.  
LogOutPulse: 1-bit active high single cycle pulse outputted after the user has pressed the logout button.

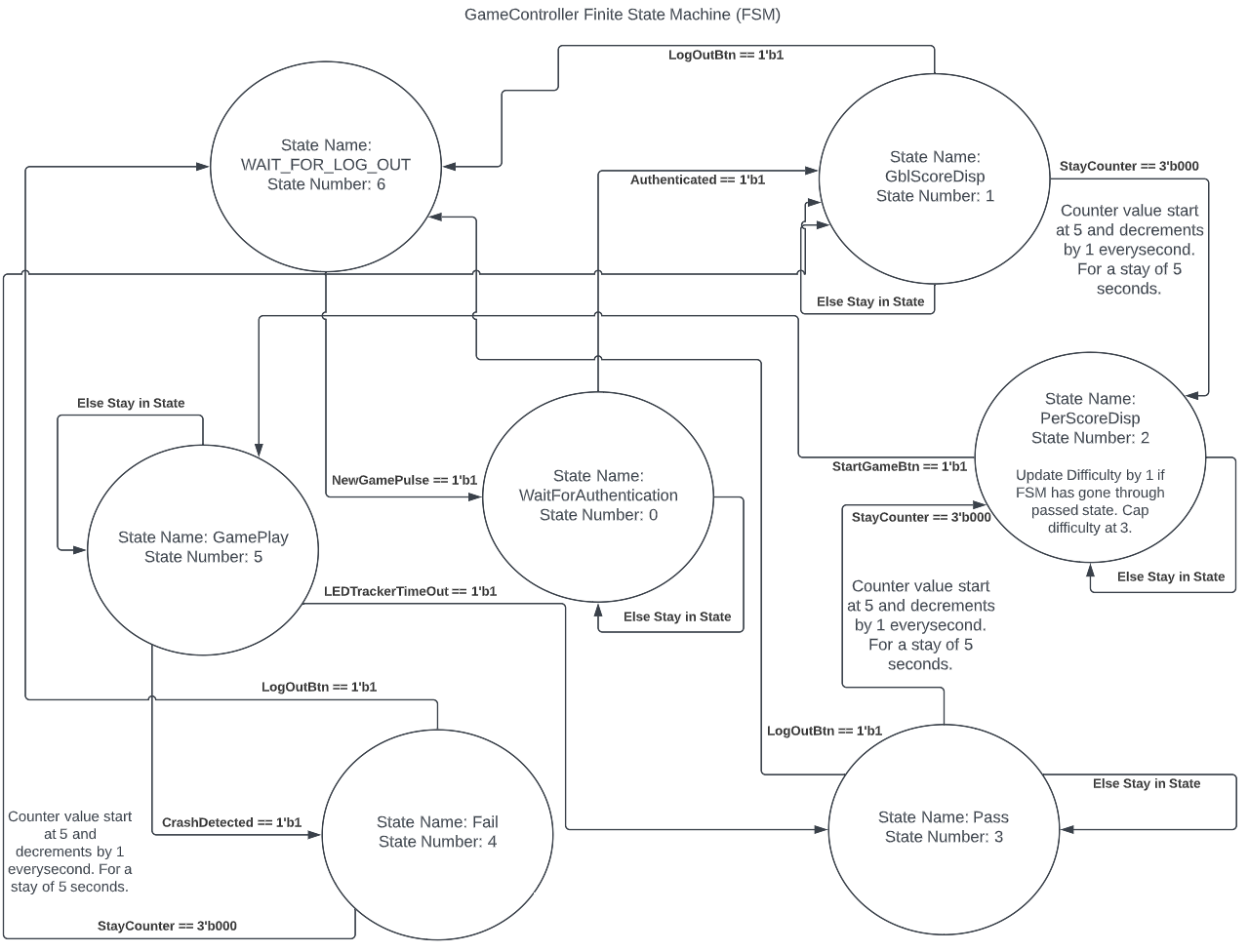


Fig. 13. Finite State Machine (FSM) for the GameController module.

### AccessController Simulation

Since the AccessController is a top-level module, of the control category, its simulation results are comprised of the simulations of modules CheckID, CheckPswd, and GameController. As a point to note further, CheckID and CheckPswd are instantiated by the MultiUserAuthentication module. Essentially, AccessController simulation is the simulation of child modules of MultiUserAuthentication and the stand-alone GameController module. *See respective sections for further information and simulation.*

#### CheckID Simulation

The CheckID module simulation features 3 test conditions, Failed Attempt, Successful Second Attempt, and Logout. Moreover, as this module requires ROM files, the waveform was generated with the option of “All items in region,” so that only signals of interest are generated.

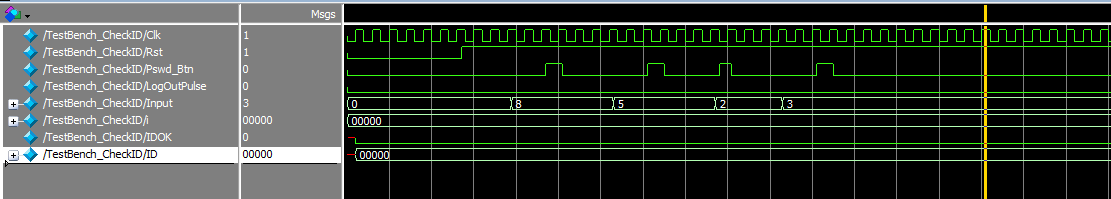
Failed Attempt:

Fig. 14. Test condition 1 of 3 for CheckID module simulation.

The simulation shown in Fig. 14, is of the 1st test condition –Failed Attempt. The “IDOK” signal is the signal of interest. It is an active high steady signal that will be 1 after the user inputted id has been validated. As shown in the simulation above, it remains 0 after all the user input has been factored. This is so because, the user input id “8523” is not a valid id. This id number is not present in the ROM. Thus, the system stays in the first step of authentication. Waiting for user to enter a valid id. Finally, the method of entering user digit inputs remains the same as it has been from lab 1 onwards. Although, this can’t be explicitly seen in the simulation above due to the fact that internal signals are not shown. The user input dig values are only passed into the system when the user pressed the password button to do so.

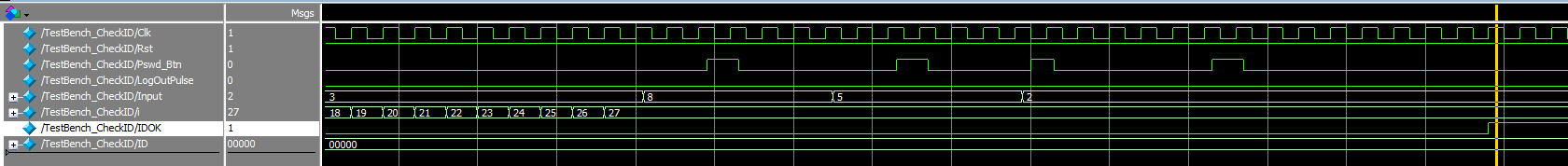
Successful Second Attempt:

Fig. 15. Test condition 2 of 3 for CheckID module simulation.

The simulation shown in Fig. 15, is of the 2nd test condition –Successful Second Attempt. Much like with the first test condition, “IDOK” is the signal of interest. As a side note, the “Input” signal is 3 prior to it being the correct id sequence of “8522.” This is shown here to, authenticate the case that the simulation shown above is indeed of the second attempt. Much like in the first case, user inputs are only passed when the Pswd\_Btn is pressed, noted by the single cycle active-high pulse it generates. And, after the correct sequence has been entered, the “IDOK” signal is pulled up to 1 and it remains there. Finally, the “ID” signal remains, 00000, initial value, because the id of 8522 is for the 1st address of ROM file. If the id of address of 00001 was entered and validated, the “ID” signal would be updated to said address.

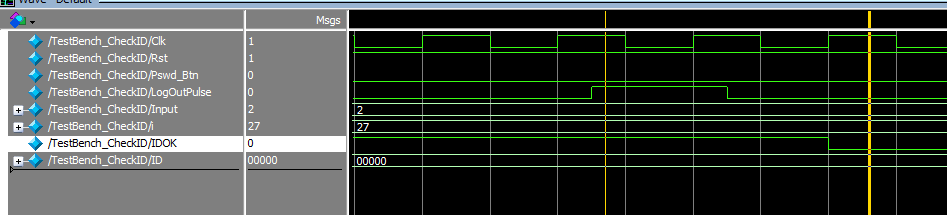
Log Out:

Fig. 16. Test condition 3 of 3 for CheckID module simulation.

The simulation shown in Fig. 16, is of the 3rd and final test condition –Logout. There are two signals of interest in this test case. “IDOK” and “LogOutPulse.” As a side note, to be able to logout, the user must have been authenticated, otherwise it makes no logical sense to logout of something one’s not logged into. For the CheckID module, the IDOK signal must be high when logout button is pressed and the “LogOutPulse” is an active high single cycle pulse. From the simulation above, it is indeed the case. The “IDOK” signal is a level/steady 1 when the logout button is pressed, noted by the generation of a “LogOutPulse.” A further distinction must be made here. When the “LogOutPulse” is generated, it takes exactly 1 cycle before IDOK signal is pulled down to 0 and system goes back to the initial state and begins the id validation process.

#### CheckPswd Simulation

The CheckPswd module simulation features 3 test conditions, Wrong Id – Right Password, Right Id – Wrong Password, and Verified & Log out. A distinction must be made before further discussion is continued. The “wrong” and “right” is in reference to the tester’s choice of id and password. The tester chose the ID 00000 and password A54E32 combination. Thus, if either one of the values are inconsistent it’s considered incorrect. Lastly, the ID isn’t chosen by user perse. The ID is, instead, passed to this module by the CheckID module upon successful ID validation. And, the fed id is simply the ROM address where the user entered sequence matches the ROM sequence. Thus, the CheckPswd module needs to not only check if entered password is one of the present ROM sequences but also determine if the user entered ID matches the right password. This is done by determining if the matched ROM password sequence has the same ROM address as the one for ID. If they’re incorrect, as shown in the first test case, the user is not authenticated.

Wrong Id – Right Password:

A screenshot of a computer

Description automatically generated

Fig. 17. Test condition 1 of 3 for CheckPswd module simulation.

The simulation shown in Fig. 17, is of the 1st test condition –Wrong Id – Right Password. The correct id password combination as stated above should be ID = 00000, Password = A54E32. The signals of interest for the above simulation are the “Begin,” “Internal\_ID,” “Input,” and “Verified” signals. The Begin signal is driven to this module by the CheckID module. And, it “tells” the CheckPswd module to begin password authentication process as the user entered ID has been validated. Like the Begin signal, the Internal\_ID signal is also fed in by the CheckID module upon successful id validation. The finite state machine (FSM) will not move forward if Begin signal is 0. This can be seen in the simulation above. It is regrettable that internal signals aren’t shown in fig x, as they would make this condition clear. However, introducing internal signals will generate many unnecessary signals to be generated. For the first test case, the Internal\_ID is 00001 and user input, denoted by Input signal, is a54e32. This combination is incorrect, as a54e32 is the password for the Internal\_ID of 00000. Thus, the Verified signal remains low at 0. As a final note, “a54e32” is equivalent to “A54E32.” Selecting Hex as the radix form on simulation options shows the “A” as “a”, they are equivalent.

Right Id – Wrong Password: Graphical user interface

Description automatically generated

Fig. 18. Test condition 2 of 3 for CheckPswd module simulation.

The simulation shown in Fig. 18, is of the 2nd test condition –Right Id – Wrong Password. The correct id password combination, stated earlier, is D = 00000, Password = A54E32. The signals of interest for the above simulation are the “Begin,” “Internal\_ID,” “Input,” and “Verified” signals. The Begin signal is driven to this module by the CheckID module. And, it “tells” the CheckPswd module to begin password authentication process as the user entered ID has been validated. Like the Begin signal, the Internal\_ID signal is also fed in by the CheckID module upon successful id validation. After a wait of 42 cycles, denoted by “i” signal, the “Internal\_ID” is changed to 00000 and user inputs the password again. The “Input” signal is now a54e33. The second 3, in the figure above, is the continued signal bus with value of 3. As stated earlier for the “Internal\_ID” of 00000, the correct password sequence is A54E32. Thus, the “Verified” signal remains low, as the entered sequence is incorrect. This CheckPswed FSM will attempt to match with all available rom values before returning to initial state to begin password authentication process once more.

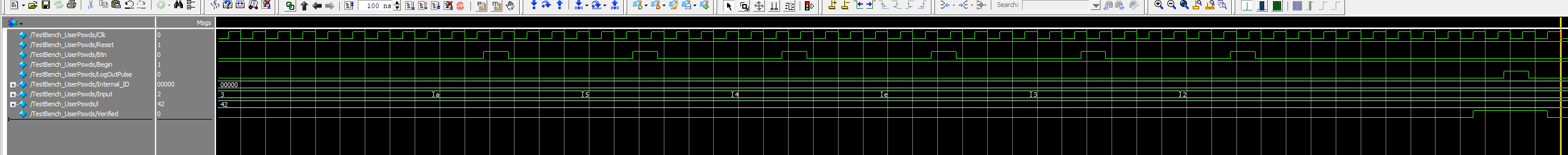
Verified & Logout:**

Fig. 19. Test condition 3 of 3 for CheckPswd module simulation.

The simulation shown in Fig. 19, is of the 3rd test condition – Verified & Logout. The figure is a little harder to read as compared to the other simulation screen captures. However, this an unavoidable as a screen width larger than for previous figures is required to show both Verified and logout. The signals of interest for this test are “Internal\_ID,” “Input,” “Verified,” and “LogOutPulse.” In the figure above, it can be seen that the “Internal\_ID” is 00000 and that the “Input” sequence is a54e32. Since this is the correct id-password combination, the “Verified” signal is pulled up to 1 and stays there. The signal stays as 1 until the “LogOutPulse,” an active high single cycle pulse is generated. This pulse is generated when the Log out button is pressed. A 1 cycle delay after said pulse is generated, the “Verified” signal is driven back to 0 once more. The user must now start the authentication process once again.

#### GameController Simulation

The GameController module simulation features 4 test cases, Initial Gameplay, Pass State, Game Restart After Pass, Fail & Logout. The waveform for these simulations were generated with the option to include internal signals as well. This module doesn’t require ROM files so only the internal signals used by the modules directly will be generated and shown.

Initial Gameplay:A picture containing graphical user interface

Description automatically generated

Fig. 20. Test condition 1 of 4 for GameController module simulation.

The simulation shown in Fig. 20, is of the 1st test condition –Initial Gameplay. The signals of particular interest for this testcase are “Authenticated,” “GameStartBtn,” and “State.” The reader may pay attention to either internal “DUT” signals or the top-level test bench signals for the first 2 signals, as the latter signal is only available to be observed amongst the DUT signals. The “Authenticated” signal is fed to the GameController module by the MultiUserAuthentication module, who, in turn, generates this signal from the CheckPswd module. It signifies that both the Id and Password combination has been validated and that access to game is now allowed. Thus, when “Authenticated” signal is 1, the state transitions from state 0 to state 1 and then to state 2, the wait for start state and personal score display state. State 1 is the global score display. In state 2, the FSM waits for the user to press the game start button on the FPGA board so that the game may begin. As is observable, when the “GameStartBtn” is an active high single cycle pulse, upon pressing the game start button, the FSM state transitions to state 5, the Gameplay state. Other signals may be ignored, as they are not of particular importance for this test case.

Pass State:Graphical user interface, diagram

Description automatically generated

Fig. 21. Test condition 2 of 4 for GameController module simulation.

The simulation shown in Fig. 21, is of the 2nd test condition –Pass State. The signals of particular interest for this testcase are “State,” “LEDTimeOut,” and “PassedRoundPulse” signals. The LEDTimeOut signal is an active signal cycle pulse outputted by the LEDTimeTracker module. The signal indicates that the user has passed the current level and can chose to play again at a higher level, capped at 3, or quit. The passed state is State 3. This can be seen in fig x where, upon detecting the LEDTimeOut pulse at rising edge the State value changes for 5, Gameplay state, to 3, the Pass state. Lastly, the PassedRounPulse also becomes an active high single cycle pulse at this point.

Game Restart After Pass: 

Fig. 22. Test condition 3 of 4 for GameController module simulation.

The simulation shown in Fig. 22, is of the 3rd test condition –Game Restart After Pass. The signals of particular interest for this testcase are “State,” “GameStartBtn,” and “Difficulty.” The point of this test is to check if the user is able to start the game again after they have successfully passed a level. As can be seen from the “State” signal that there is a transition of 2 to 5. State 2 is the Pass state and State 5 is the gameplay state. And this transition occurs when the “GameStartBtn” is 1. This signal is representative of the button shaped input of the game start button. Finally, it can be seen that the “Difficulty” signal is 00 at State 2 but when FSM enters State 5 once again, the Difficulty is 1. Thus, it can be confirmed that the user is able to paly the game once again at a higher difficulty after successfully passing a level.

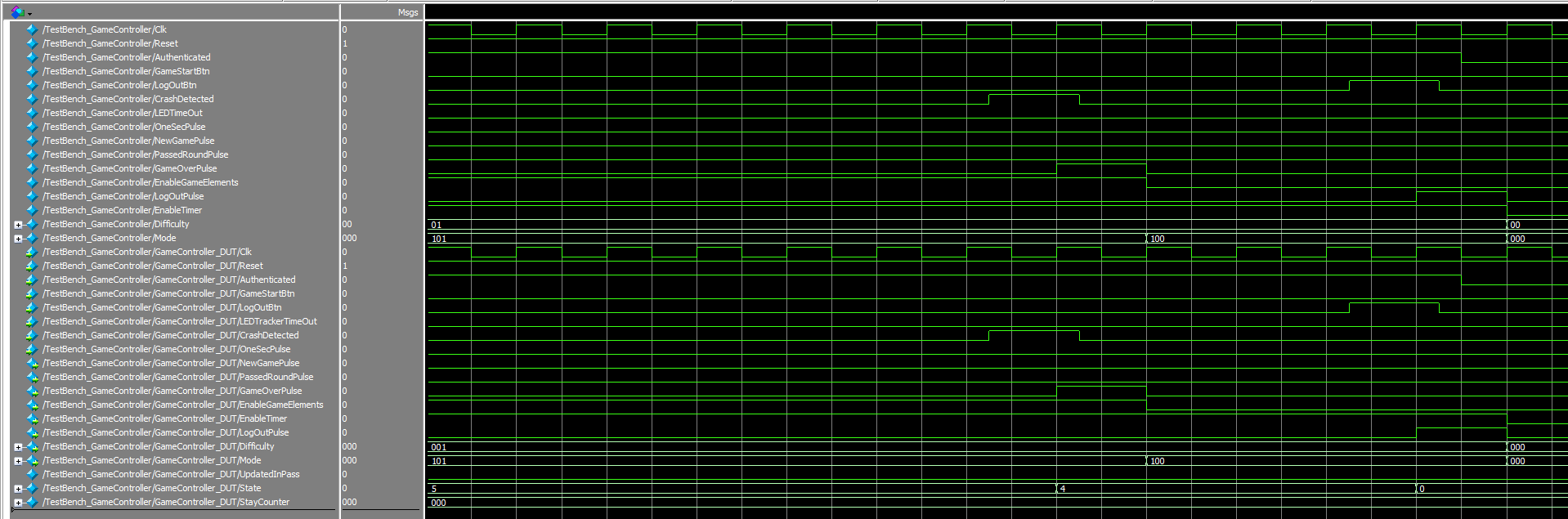
Fail & Logout: 

Fig. 23. Test condition 4 of 4 for GameController module simulation.

The simulation shown in Fig. 23 is of the 4th test condition –Fail & Logout. This test is done to essentially verify that the gamecontroller FSM enters the Fail state after the user crash has been detected. Afterwards, if the user pressed the logout button, then the user would be logged off. This is, of course, not the only time a user can logout from the game. The signals of particular importance for this test are “State,” “LogOutBtn,” “LogOutPulse,” “CrashDetected,” and “Authenticated.” As can be noted from the figure above, when CrashDetected signal is 1 the State transitions from 5 (Gameplay state) to 4 (Fail state). While on the fail state, it can be seen that the Authenticated signal remains a steady 1, signaling that the user is still logged in. However, once the user presses the logout button, the LogOutBtn becomes an active high single cycle pulse, which then triggers the LogOutPulse to do the same. And, after a cycle delay, the State transitions from 4 to State 0. The state doesn’t change after that as the Authenticated signal is 0. This signal is used to determine if a user is logged in or not. If user is not logged in, gameplay is prohibited. This is the case here.

## LED Bar Timer Module

This module provides a simple interface between other modules and the 10 LEDs on the FPGA hardware. It creates a countdown timer that acts as a “fuel gauge” for each level. The symbol is shown in Fig. 24.

Diagram

Description automatically generated

Fig. 24. LED bar module with Input and Output signals defined.

The inputs and outputs are as follows:

**Inputs:**

StartTime: 4-bit signal. Tracker will load this value into its internal register at the end of every clock cycle when TimeTrackerEnable is LOW.

TimeTrackerEnable: Enable signal, starts a countdown on its internal register, which is loaded form StartTime when the signal was LOW.

**Outputs:**

DecoderOut: Produces a single cycle pulse from LOW to HIGH to LOW every 1 second on a 50 MHz clock.

Timeout: When the internal counter tracking fuel gauge reaches 0, this signal will be a constant HIGH. Otherwise, it is a constant LOW.

## Object Shifter module

The Object Shifter module takes a random 3-bit number from the continuously running LFSR random number generator. It has an internal decoder that takes the three-bit number and returns a 7-bit wire representing a shape to be displayed on the 7-segment display. The shape starts at Out0, then is shifted to the left with each Shift Pulse input. As each pulse is received, the Object Shifter module alternates between sending a random shape and a blank space (since the 7segs use inverted logic, 7’b111 1111 is a blank shape). When a non-blank shape is pushed out of Out5, the Debris Dodge pulse is sent, which is used to count how many shapes are dodged. A separate Crash Detector module determines if the player position and object position conflict. When the Object Shifter is reset, or ObjShifterEnable bit is LOW, all outputs are set to 7’b111 1111 and Debris Dodge is set to 1’b0. The diagram representing the Object Shifter module is shown in Fig. 25

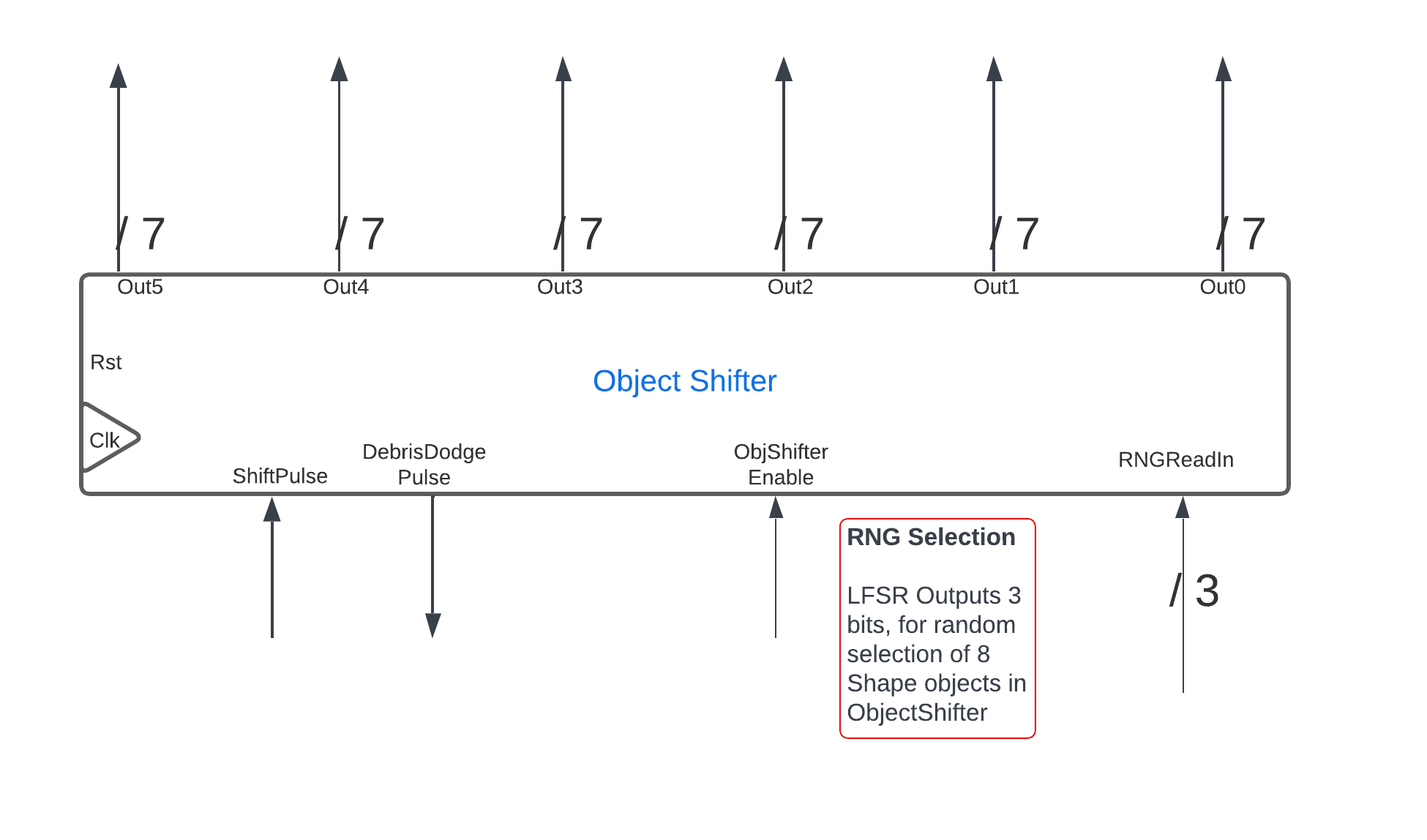


Fig. 25. The symbol for the Object Shifter module with all inputs and outputs labeled.

The inputs and outputs are as follows:

**Inputs:**clk, rst: system inputs described above.  
ObjShifterEnable: Enables ObjectShifter module and allows ShiftPulse to shift objects. When LOW, all Out0 – Out5 are reset to 7’b111 1111.  
RNGReadIn: An input of 3 bits from the LFSR random number generator that is constantly running in the background.  
ShiftPulse: Expects a single HIGH pulse that shifts all 7-bit outputs to the left. This also triggers the creation of an alternating random shape or 7’b111 1111 (blank shape) into Out0.

**Outputs:**  
Out0 – Out5: 7-bit outputs intended for the 7-seg displays. A shape is generated at Out0 and shifts toward Out5 with every ShiftPulse.  
DebrisDodge: A HIGH pulse when a non-blank shape object is shifted out of Out5. This can be used for counting how many shapes pass the player for scoring.

The Object Shifter module uses a one-procedure Finite State Machine (FSM) to get the random shape, then shift all shapes to from Out0 toward Out5. The FSM diagram is shown in Fig. 26

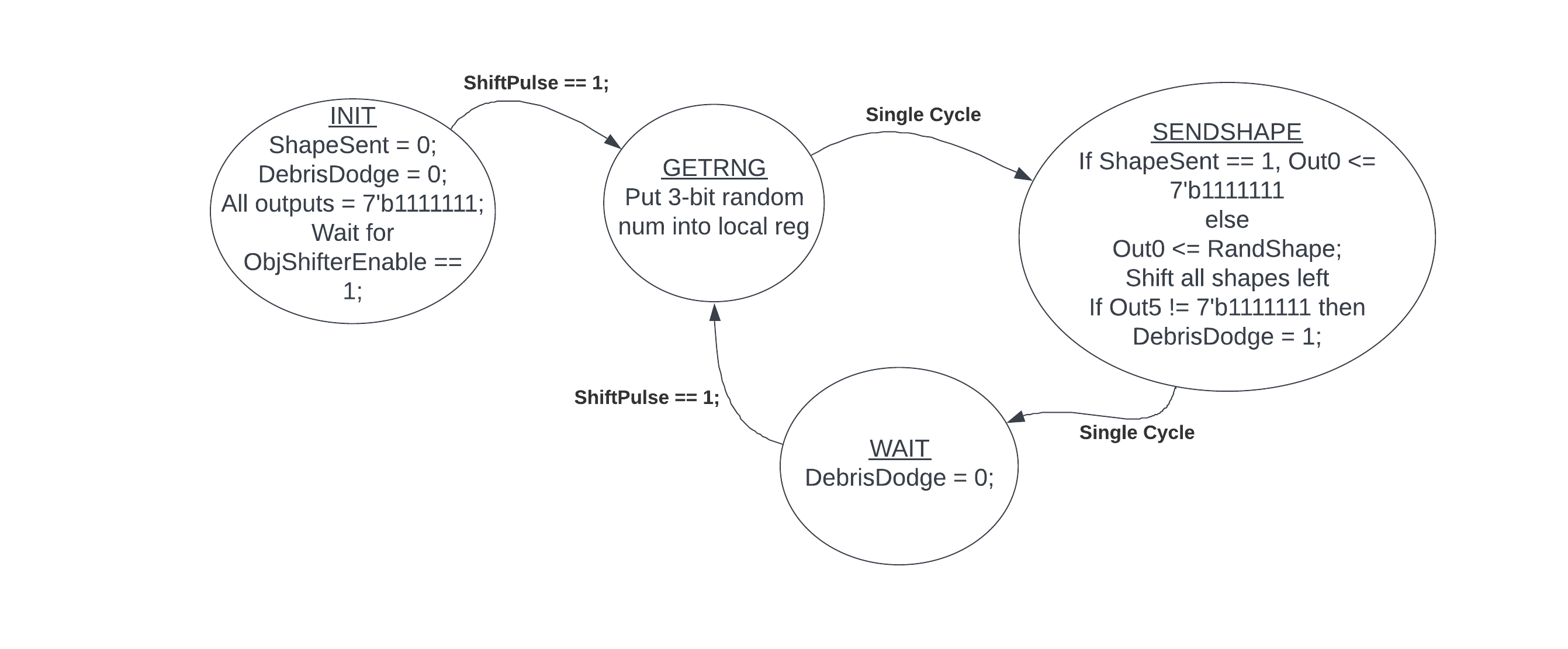


Fig. 26. Finite State Machine for ObjectShifter module

The Object Shifter stays in the INIT state any time that ObjShifterEnable == 1’b0. Once ObjShifterEnable == 1’b1, it waits for a ShiftPulse signal. When the signal is received, GETRNG takes the 3-bit random number from the RNG\_LFSR module and puts it into an internal register. The 3-bit random number is decoded by an instance of the combinational logic Decoder\_3bTo7bShape module into a 7-bit shape. SENDSHAPE shifts all Out0-Out5 registers to the left, alternatively placing the random shape or the blank shape (7’b111 1111) into the Out0 register, and if the Out5 register is a non-blank shape, DebrisDodge sends a HIGH pulse indicating that a shape was passed out to the left. The WAIT state sets DebrisDodge back to 1’b0 and waits for the next ShiftPulse signal.

### Object Shifter Simulation

The Object Shifter module was tested in ModelSim using a testbench file. For the test, the module was first reset, then periodic ShiftPulse signals were sent with the ObjShifterEnable HIGH. The waveform is shown in Fig. 27.

A screenshot of a computer

Description automatically generated

Fig. 27. Waveform of ObjectShifter testbench

As the first red arrow indicates, the 7-bit shape signal can be seen shifting from Out0 to Out5 with each pulse from the ShiftPulse signal. The next red box shows the DebrisDodge pulse when the first object is shifted out of Out5, indicating that the shape was dodged. The input from the LFSR can be seen as RNGReadIn, and it changes nearly every cycle except when it randomly chooses the same number (since the values are 0-7, this happens often). Further in the simulation, the ObjShifterEnable was set to 1’b0, and all of the outputs went to 7’b111 1111 as expected.

## RNG\_LFSR\_3b module

The diagram in Fig. 28 shows the RNG\_LFSR\_3b module, with all the inputs and outputs labeled.

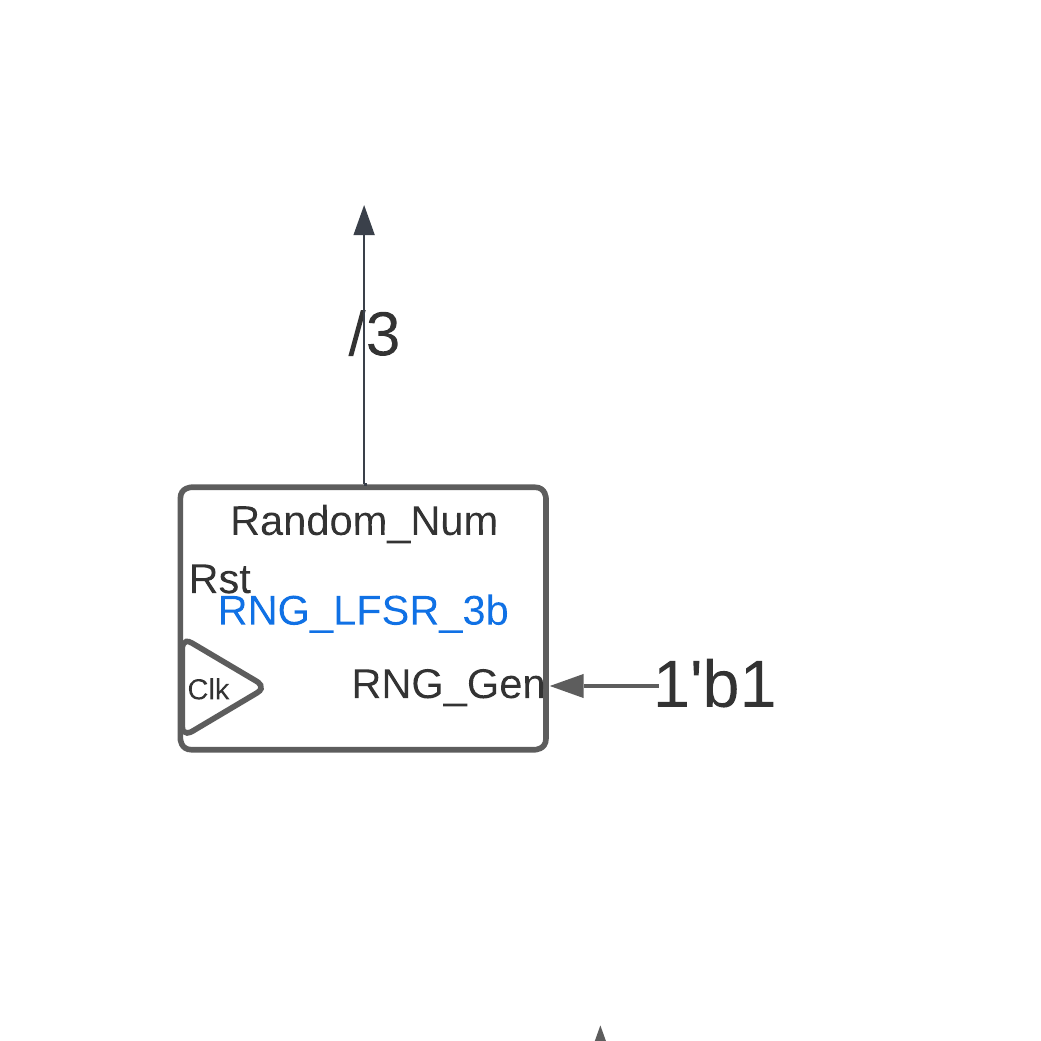


Fig. 28. Symbol for RNG\_LFSR\_3b with inputs and outputs.

**Inputs:**clk, rst: The standard definitions of clock and reset signals.  
RNG\_Gen: For this application, the shift register runs in the background every clock cycle, so RNG\_Gen is HIGH.

**Output:**  
Random\_Num: A 3-bit pseudo-random number output derived from 3 bits of the 16-bit LFSR counter.

The random number generator is meant to create a pseudo random 3-bit number, which is based on the shift register changing every cycle as the clock runs. This module was derived from a previous LFSR module, so did not require a separate testbench, however its operation can be seen in the Object Shifter simulation above. The 3 bits change almost every cycle, but since there are only 8 possible outputs, there are cycles where the number is repeated at random.

## Decoder\_3bTo7bShape module

The symbol for the Decoder\_3bTo7bShape is shown in Fig. 29 with its input and output labeled. This is customized decoder module that accepts a 3-bit number, and outputs 7 bits representing one of 8 random shapes for the 7-segment display.

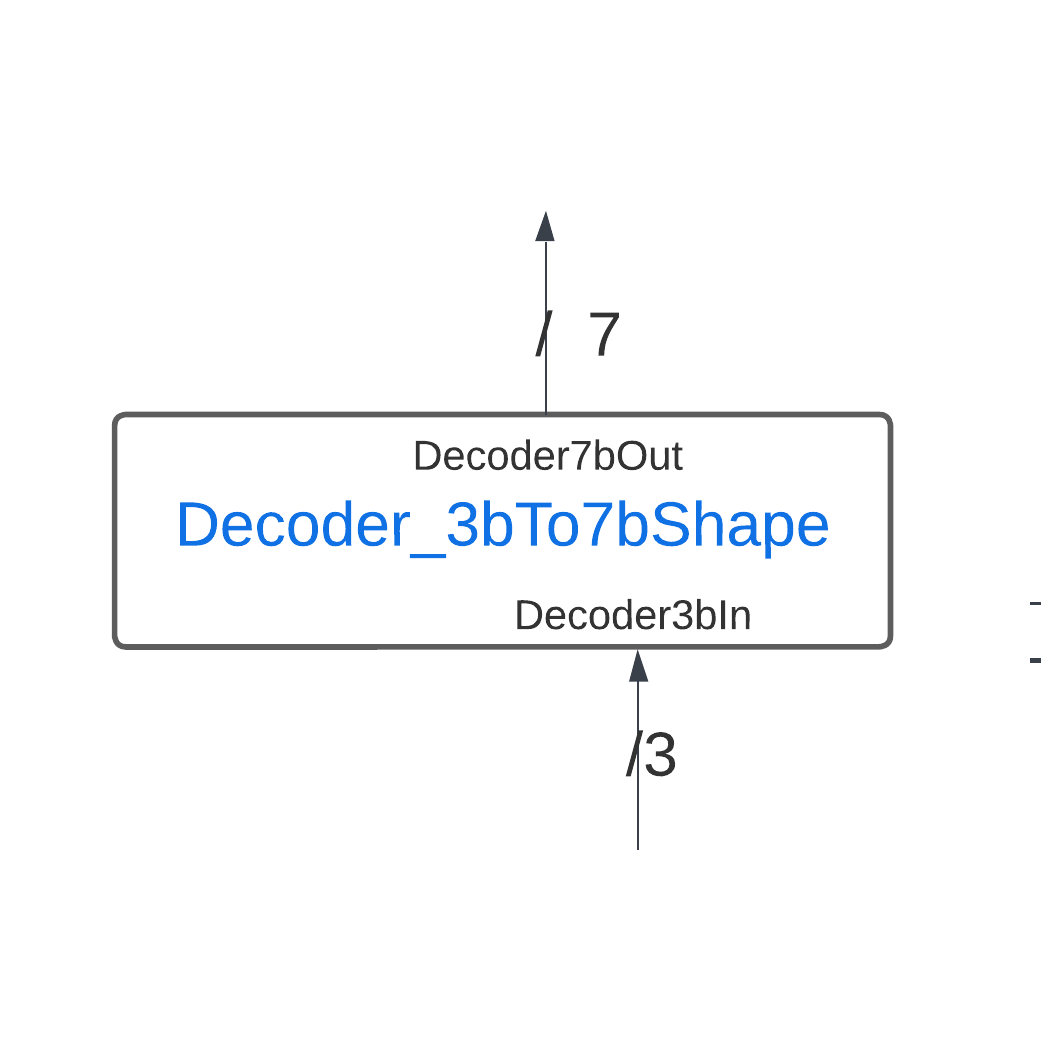


Fig. 29. Symbol for Decoder\_3bTo7bShape module with input and output labeled.

The Decoder\_3bTo7bShape module has the following inputs and outputs.

**Input:**  
Decoder3bIn: A 3-bit input which comes from the random number generator.

**Output:**  
Decoder7bOut: A 7-bit signal that will display one of 8 shapes(4 top shapes and 4 bottom shapes) when connected to a 7-segment display. A ‘0’ means that segment is lit, and a ‘1’ means that segment is unlit.

The decoder uses combinational logic, so any change in input is immediately reflected in the output signal.

### Decoder\_3bTo7bShape simulation

The Decoder\_3bTo7bShape module was tested using testbench\_ Decoder\_3bTo7bShape in ModelSim. All eight input signals were tested from ‘000’ to ‘111’. A portion of the waveform is shown in Fig. 30.

Text

Description automatically generated

Fig. 30. Simulation waveform of testbench\_Decoder\_3bTo7bShape from ModelSim.

The screenshot shows the first five outputs for the top shapes. The next 4 outputs are the bottom shapes.

## Crash Detector module

The Crash Detector module takes two inputs representing an object and a player object on a 7-segment display. The output is a combination the two inputs, and if they conflict, a Crash Detected pulse is sent to tell the game controller that the player has failed to avoid the obstacles in that round. The symbol for the module is shown in Fig. 31.

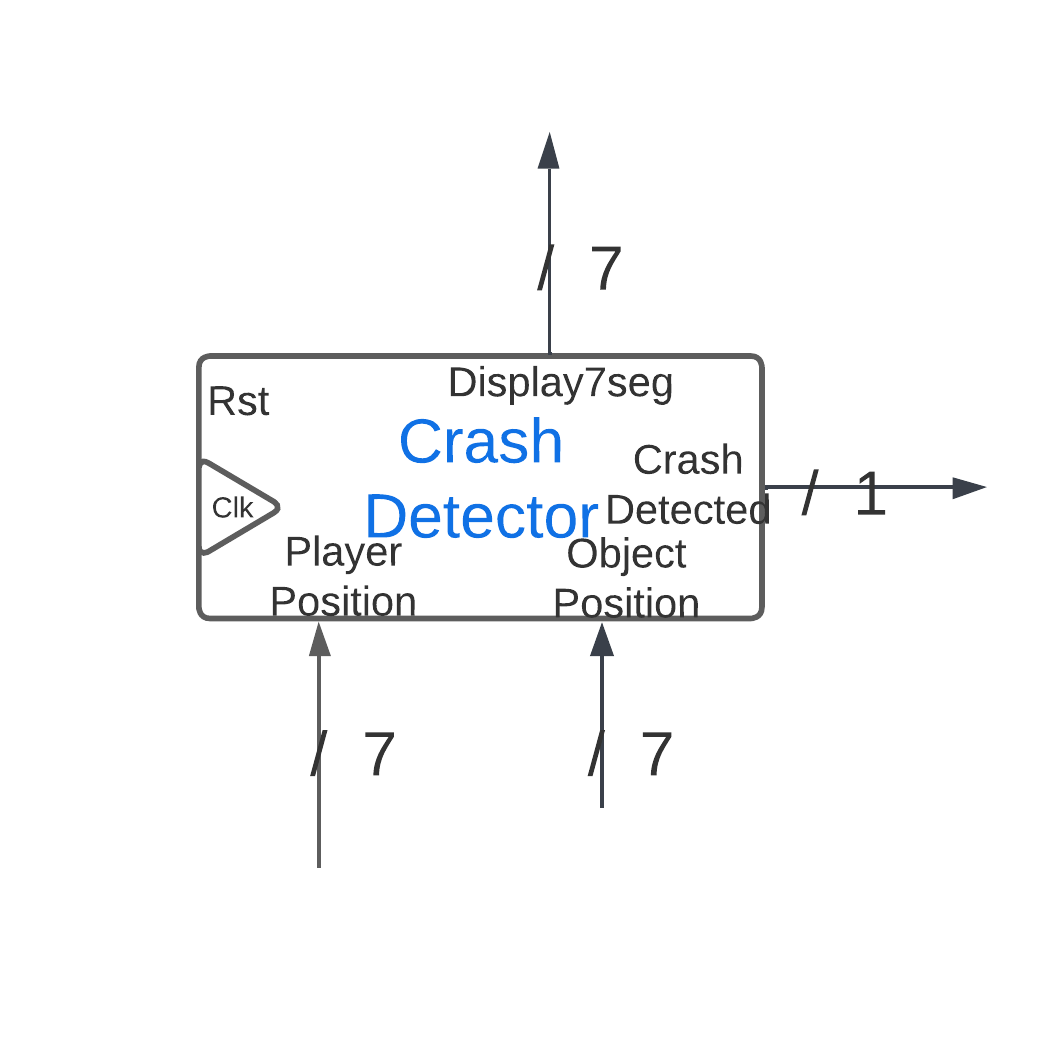


Fig. 31. Symbol for Crash Detector module with inputs and outputs labeled.

The Crash Detector module has the following inputs and outputs.

**Input:**clk, rst: The standard definitions of clock and reset signals.  
PlayerPosition: A 7-bit input representing the player position in a single 7-segment display.  
ObjectPosition: A 7-bit input representing the object’s position in a sing 7-segment display

**Output:**  
Display7seg: A 7-bit signal that will display the combination of the two input positions.  
CrashDetected: A single HIGH pulse when the two inputs conflict.

The logic allowing for combination of the two shapes and detection of conflict is deceptively simple. Since the 7-segment displays use inverted logic (‘0’ lights the segment, ‘1’ dims the segment), the crash detection needs to detect when two corresponding bits both have a ‘0’ in the input. The center segment (the most significant bit) is allowed to conflict because the player has to use it to make a complete shape, so the crash detector only checks the lower 6 bits. This was done by using a bitwise OR, so that any conflicts show as a 0, and if they don’t conflict the OR operation results in 6’b11 1111. To combine the shape and position for the Display output, a bitwise AND operation is used.

The Crash Detector uses a one-procedure FSM to send the CrashDetected pulse, shown in Fig. 32.

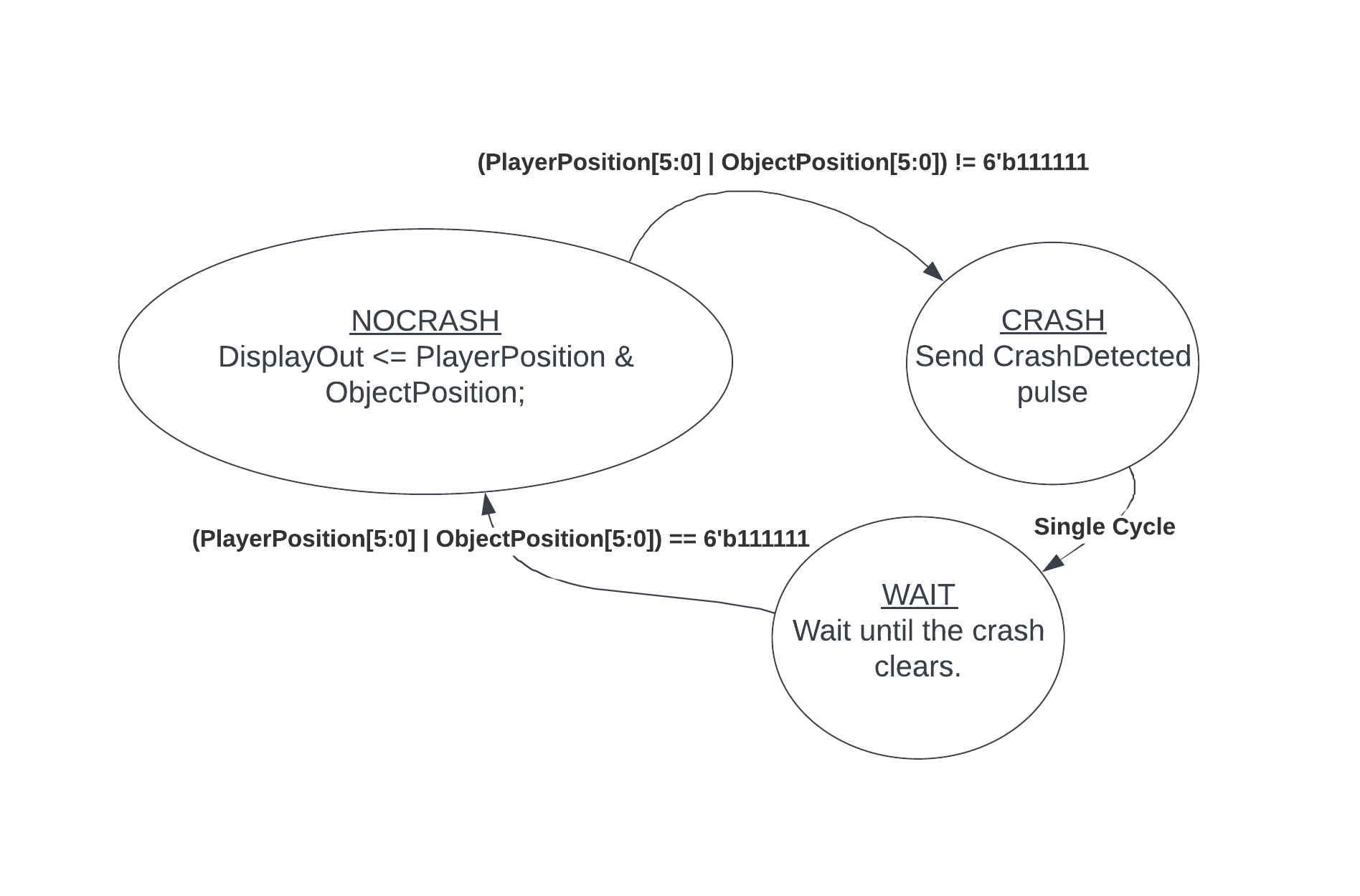


Fig. 32. Crash Detector finite state machine(FSM).

The finite state machine ensures that once a crash is detected, only a single pulse is sent until that crash is cleared.

### CrashDetector Simulation

The CrashDetector module was tested using testbench\_CrashDetector on ModelSim. The waveform is shown in Fig. 33.

A screenshot of a computer

Description automatically generated with medium confidence  
A screenshot of a computer

Description automatically generated with medium confidence

Case 4 No Crashes

Case 3 Four Crashes

Case 2 Four Crashes

Case 1 No Crashes

Fig. 33. Waveform of testbench\_CrashDetected simulation.

In the first case, the player is in the “top” position, and the 4 shapes are “bottom” shapes. Between each shape signal, a 7’b111 1111 shape is sent to clear the potential crash state, but since the player is on top and the shape is on the bottom, there are no CrashDetected pulses sent. The next 4 shapes are the “top” shapes, so all of them should conflict with the player in the top position, and 4 CrashDetected pulses are sent. The next case shows the bottom shapes with the bottom player position, resulting in 4 crashes, and then the top shapes with the bottom player position, resulting in no crashes.

## LED\_Animator module

The LED Animator module controls the display bar of 10 LEDs below the 7-segment displays. It has a 10-bit input that is the user-selected fuel gauge (timer), which displays when logged in to the game, and animation states for when a new personal best is achieved and for when a global best is achieved. The symbol for the module is shown in Fig. 34 with inputs and outputs labeled.

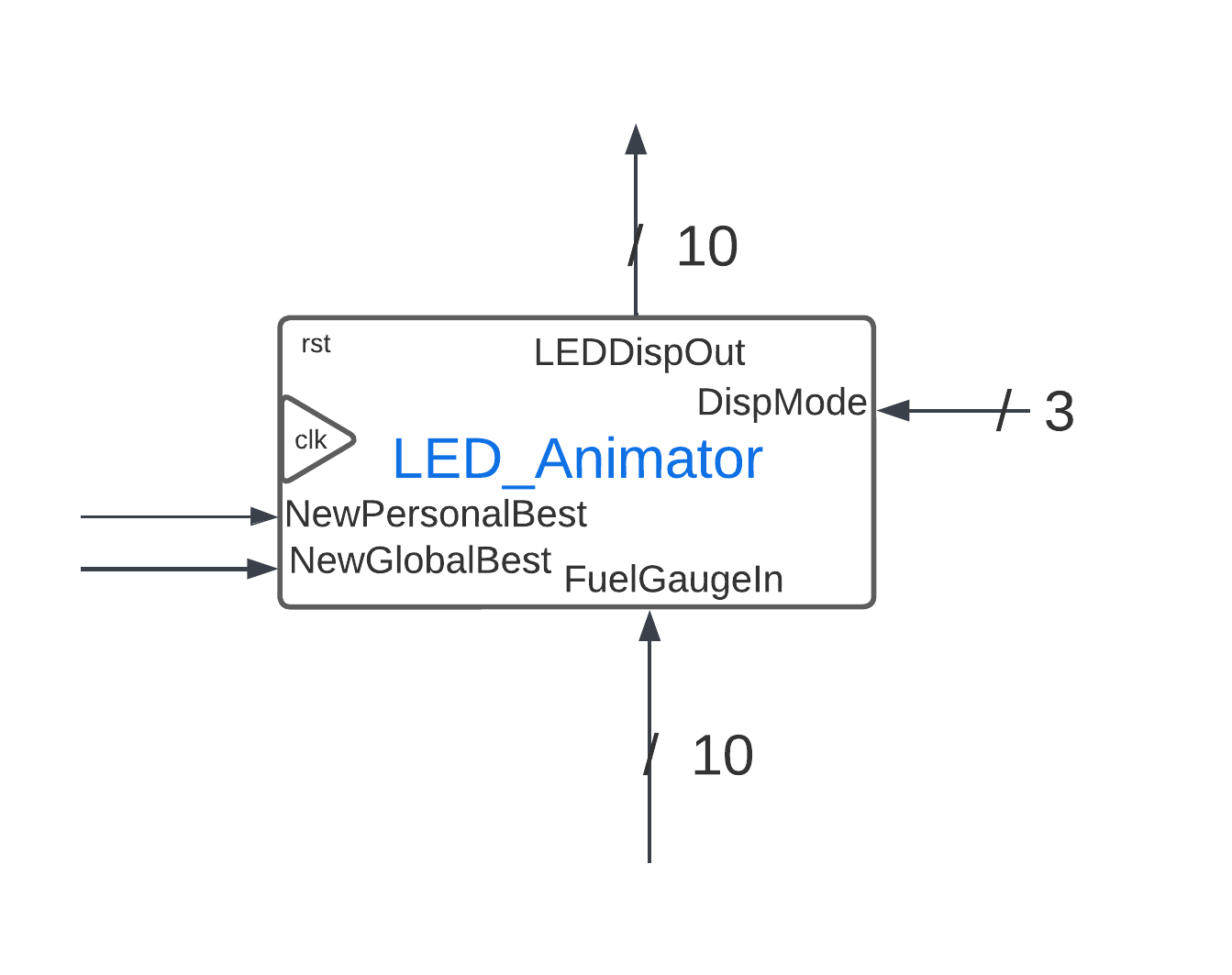


Fig. 34. LED\_Animator module with all inputs and outputs.

The inputs and outputs are as follows:  
**Input:**clk, rst: The standard definitions of clock and reset signals.  
NewPersonalBest: A single HIGH pulse is expected when the scorer detects a new personal high score.  
NewGlobalBest: A single HIGH pulse is expected when the scorer detects a new global high score.  
FuelGaugeIn: A 10-bit input representing the user selected level length display.  
DispMode: Tells whether the AccessController is in the authenticator vs. gameplay states.

**Output:**  
LEDDispOut: A 10-bit signal that drives the 10 LED display.

The module is driven by a one-procedure finite state machine, shown in Fig. 35.

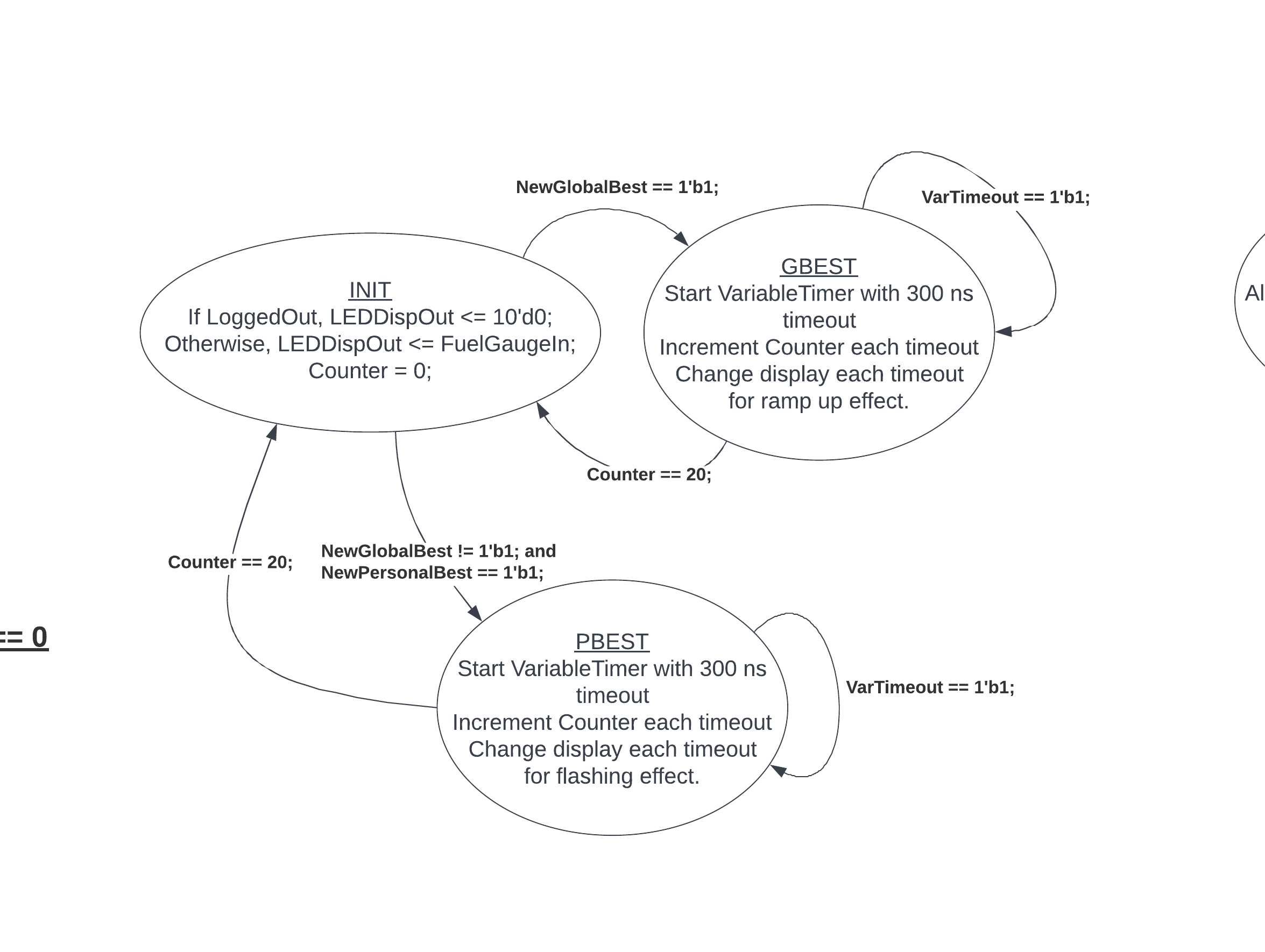


Fig. 35. LED\_Animator finite state machine.

When logged out, the LED display is blank, but when in a gameplay or scorer display mode, the LEDs will show the fuel gauge the user selected. If a NewGlobalBest pulse is detected, the display will do a “ramping up” animation to celebrate the new global high score. If a global high score is not detected, but a personal high score is detected, the LED display does a flashing animation. Both of these states use the Variable\_Timer module and an internal counter to create the animation. When the animation finishes, the fuel gauge displays again.

This module was added to the project after all other modules were completed, and was tested directly on the hardware.

## PlayerPosition module

The PlayerPosition module is responsible for toggling the player’s position from top to bottom when the player presses the Game Start/ Toggle Player button. The symbol for the Player module is shown in Fig. 36.

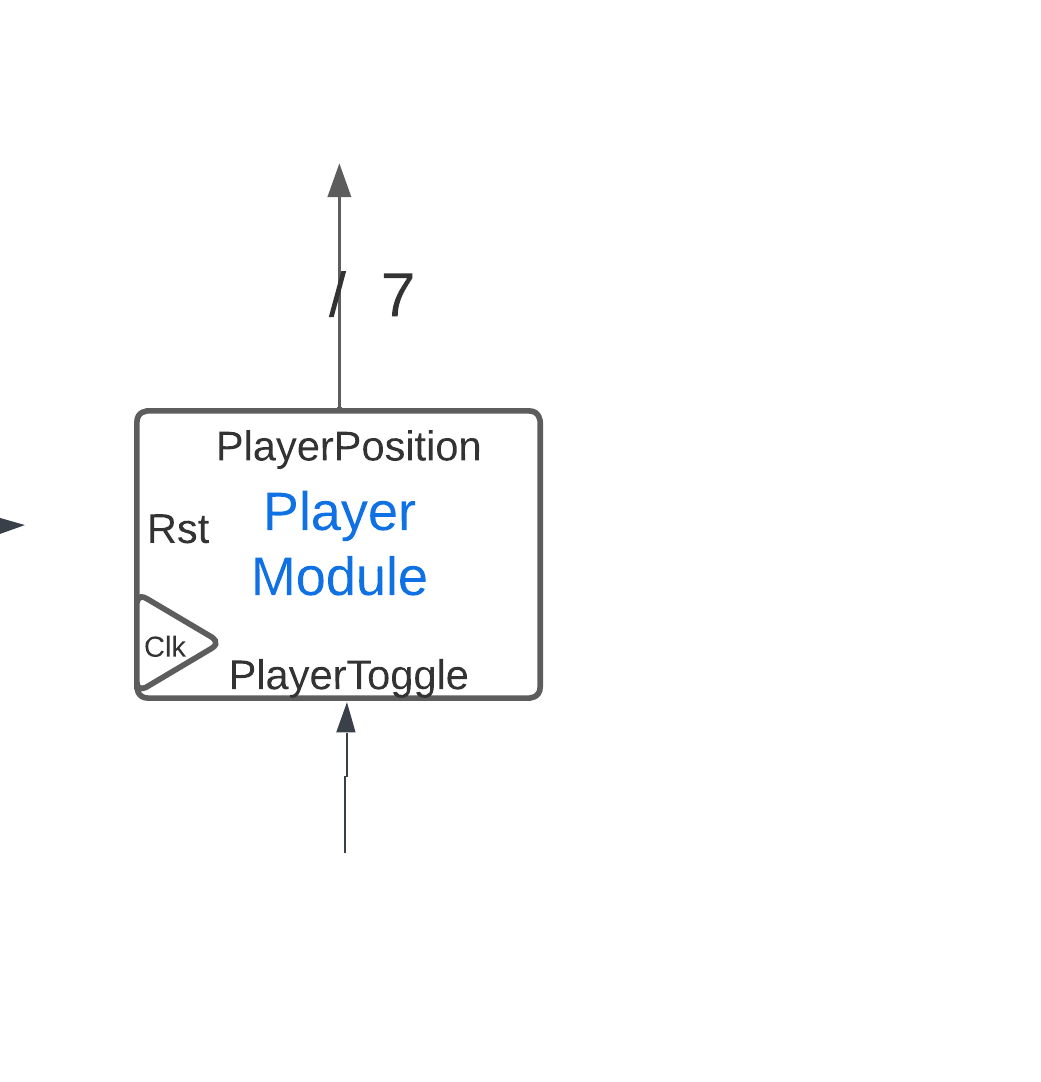


Fig. 36. Player module with input and output labeled.

The inputs and outputs for the Player module are as follows:  
**Input:**clk, rst: The standard definitions of clock and reset signals.  
PlayerToggle: A single HIGH pulse is expected when the player presses the GameStart button.

**Output:**  
PlayerPosition: A 7-bit signal that represents the player’s shape on a 7-seg display.

When a single pulse is received, the 7-bit output toggles between a TOP and BOTTOM state that looks like a circle on the 7segment display.

### PlayerPosition Simulation

The PlayerPosition module was tested with ModelSim to ensure that the input/output behavior was as expected. The waveform is shown in Fig. 37.

Timeline

Description automatically generated with medium confidence

Fig. 37. Waveform of PlayerPosition simulation.

When the reset button is pressed, the output is a blank 7-segment signal (7’b111 1111), then the position defaults to the TOP, and is toggled with each successive button press to the BOTTOM position.

## Score\_Control\_Unit

The Purpose of this module is to keep track of the current game score, the personal best of each player, the overall best scoring player, and to display your score, personal best, and global best score. In addition, this module can select new personal and global bests when it is requested to. The individual functions of this module will be detailed in the sub modules as this module is just a daughter board for these sub modules.

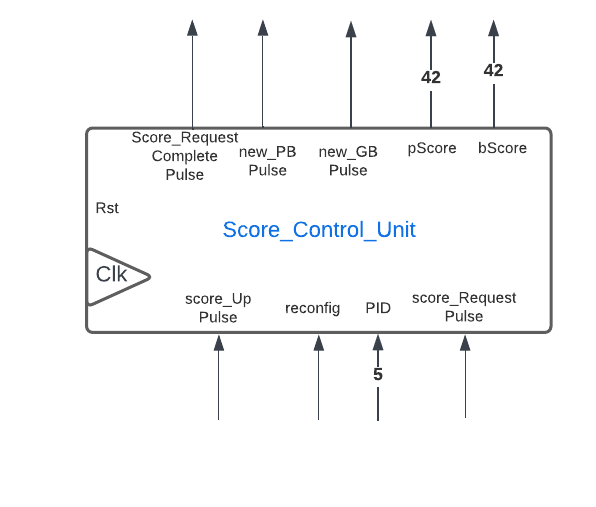


Fig. 38. module symbol for the Score\_Control\_Unit

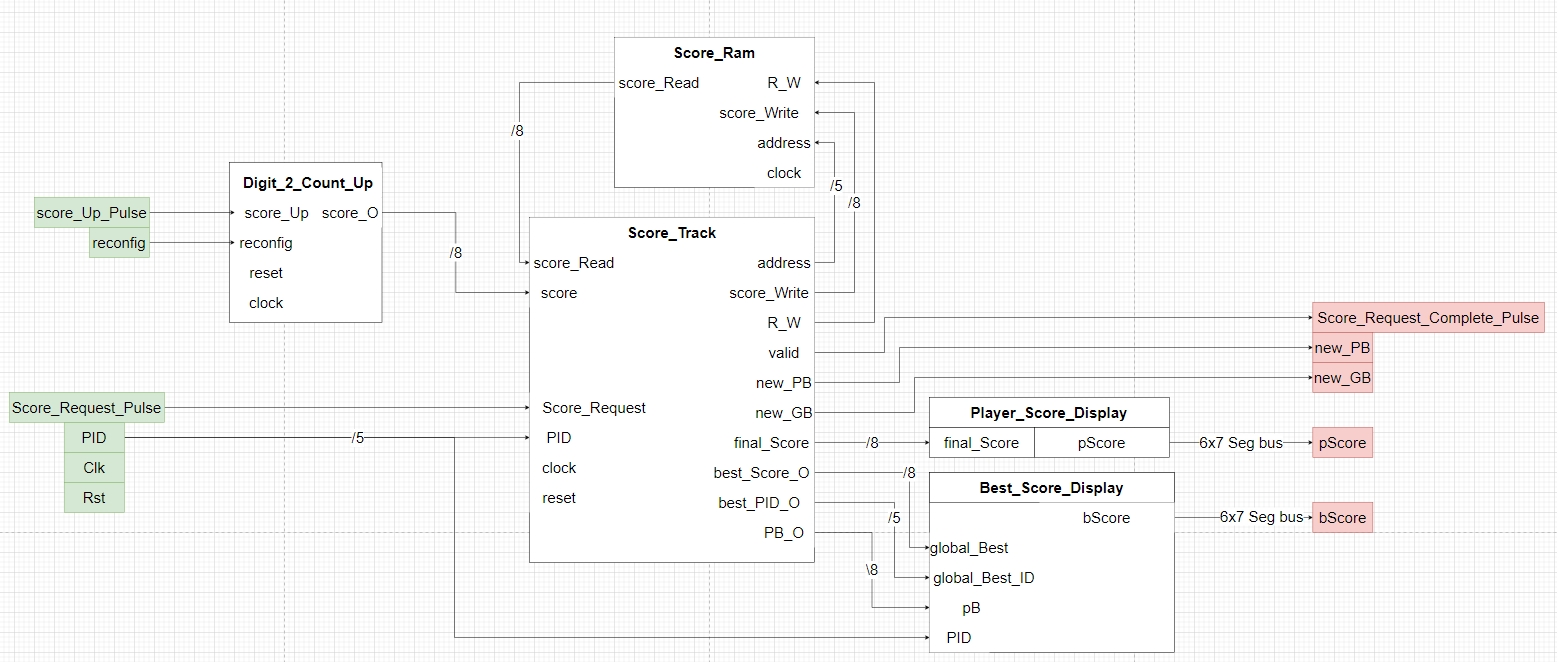


Fig. 39. Internal modules of Score\_Control\_Unit.

* Clk – 1 bit - input
  + 50 [Mhz] Synchronization pulse
* Rst – 1 bit - input
  + Resets the device and initializes the score memory
  + Active low
  + Once pulsed takes 32 clock cycles to complete
* Score\_Up\_Pulse – 1 bit - input
  + Signals the module to increment the score
  + 1 clock length pulse expected
* Reconfig – 1 bit - input
  + Signals the module to reset the currently stored score
* PID – 5 bit – input
  + The player ID of the currently logged in player
* Score\_Request\_Pulse – 1 bit - input
  + Signals the module to process the players score and compare it to their personal best and global best replacing them if they are higher
  + Takes approximately 10 clock cycles to complete
* Score\_Request\_Complete\_Pulse – 1 bit - output
  + Signals other modules that the score processing request was completed
* New\_PB\_Pulse – 1 bit – output
  + Signals other modules that a new personal best has been achieved
* New\_GB\_Pulse – 1 bit – output
  + Signals other modules that a new global best has Been achieved
* pScore – 42 bit – output
  + contains the completed display frame to display the players score that round
  + [6:0] hex0
  + [13:7] hex1
  + [20:14] hex2
  + [27:21] hex3
  + [34:28] hex4
  + [41:35] hex5
* bScore – 42 bit – output
  + contains the completed display frame to display the personal and global best scores
  + [6:0] hex0
  + [13:7] hex1
  + [20:14] hex2
  + [27:21] hex3
  + [34:28] hex4
  + [41:35] hex5

## Score\_Ram

The purpose of this module is to store up to 32 players personal best scores. Data takes 2 clock cycles after address change to become viable.

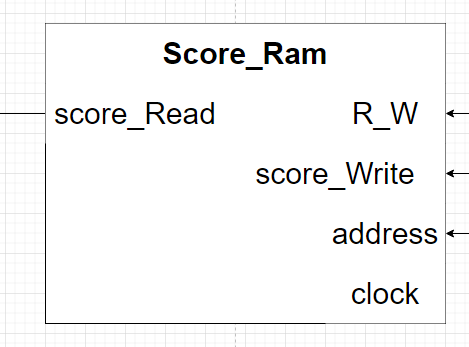


Fig. 40. Score\_Ram Symbol

* R\_W – 1 bit – input
  + Determines if data should be read from the current address
  + Read = 0
  + Write = 1
* Score\_Write – 8 bit – input
  + The score value to be written to the RAM
* Address – 5 bit – input
  + The location that data should be read/written from/to
* Clock – 1 bit – input
  + Synchronization signal
* Score\_Read – 8 bit – output
  + A score that is being read from RAM
  + Takes 2 clock cycles to update on address change

## Score\_Track

The purpose of this module is to keep track of the personal best for each player, the overall best scoring player, the final score, and if a new personal/global best is achieved.

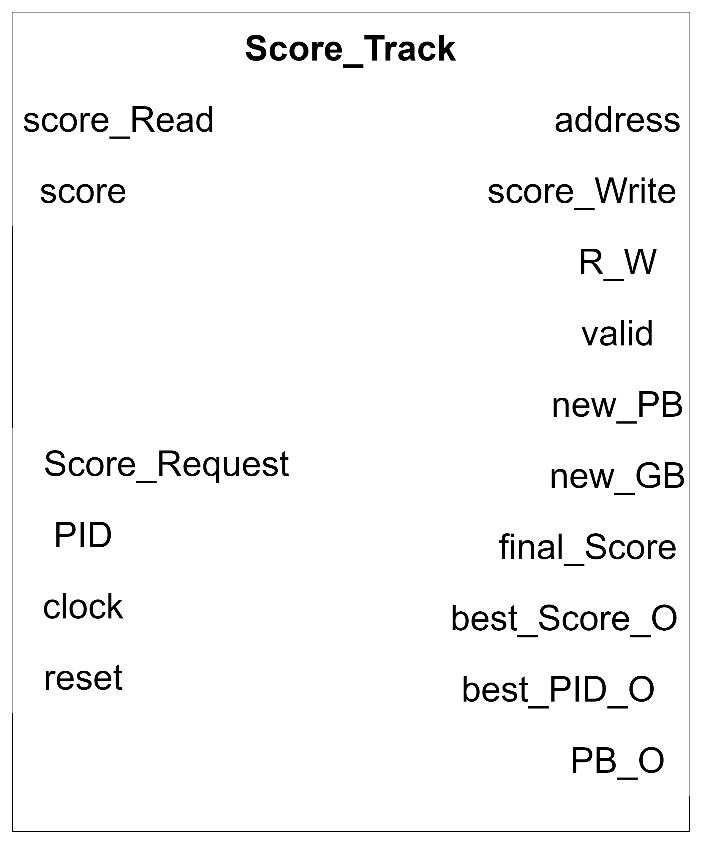


Fig. 41. Score\_Track Symbol

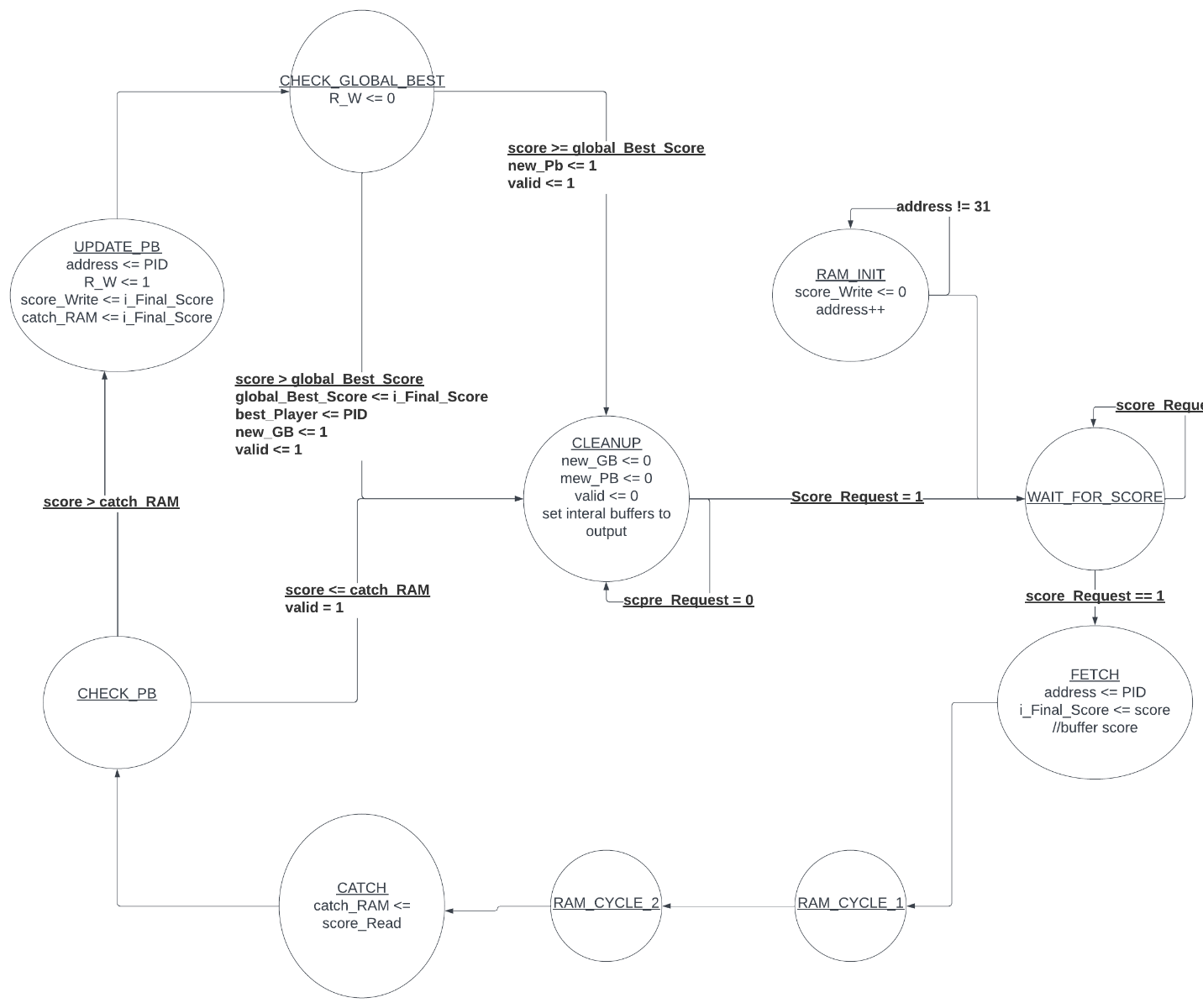


Fig. 42. state diagram for Score\_Track

* Score\_Read – 8 bit – input
  + The personal best score from RAM
* Score – 8 bit – input
  + The current games score
* Score\_Request – 1 bit – input
  + Pulse to trigger comparisons against personal and global best and update displayed scores
* PID – 5 bit – input
  + The current players ID number
* Clock – 1 bit - input
  + Synchronization signal
* Reset – 1 bit - input
  + Active low signal to reset the module and perform ram initialization
  + Takes 32 clock cycle to initialize the RAM on start up
* Address – 5 bit – output
  + What location in ram the module what to read or write to
* Score\_Write – 8 bit – output
  + The value to be written to RAM
* R\_W – 1 bit – output
  + Controls whether the RAM should be written to or read from
* Valid – 1 bit – output
  + Pulse to let other modules know that score processing has completed
* New\_PB – 1 bit – output
  + Pulse to let other modules know that a new personal best has been achieved
* New\_GB – 1 bit – output
  + Pulse to let other modules know that a new global best has been achieved
* Final\_Score – 8 bit – output
  + The final score that the player achieved this round after score comparison
* Best\_Score\_O – 8 bit – output
  + The best score overall that has been achieved
* Best\_PID\_O – 5 bit – output
  + The best players ID
* PB\_O – 8 bit – output
  + The current players personal best after score\_Request

### Score\_Tracker Simulation

To simulate the module, we created a series of test cases. The first test is to ensure that the reset initializes the RAM correctly.

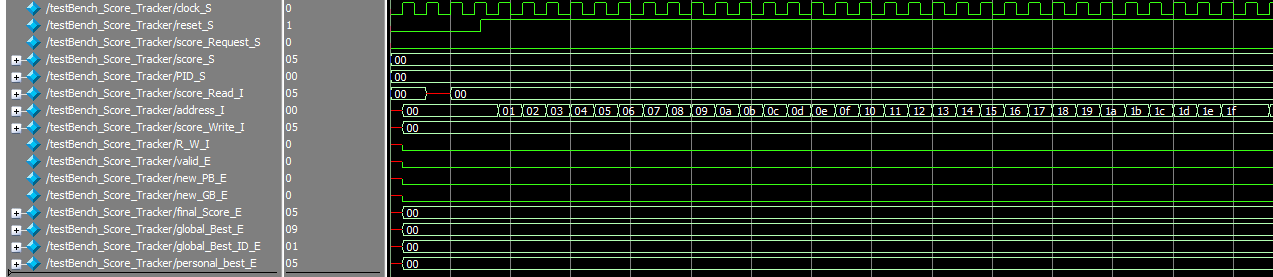


Fig. 43. RAM initialization

The next test case would be to submit a score and check that global/personal/final scores update correctly.

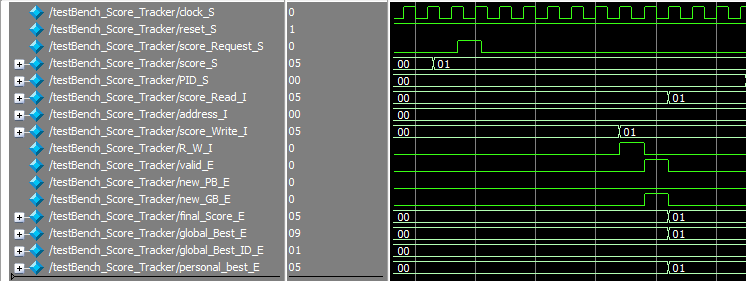


Fig. 44. New global best

The next test case is to submit a new number to ensure that those global/personal/final scores update correctly.

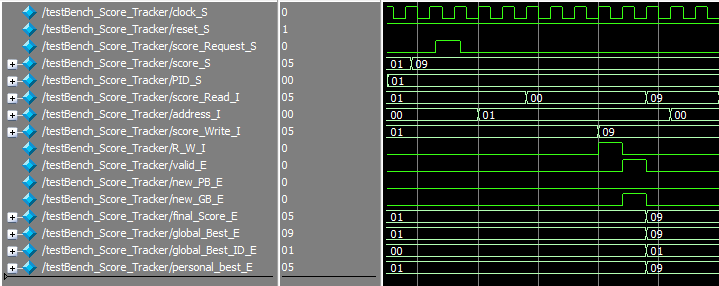


Fig. 45. Another new global best

The final test case is to submit a new number lower than the current global best to test that a personal best is updated correctly.

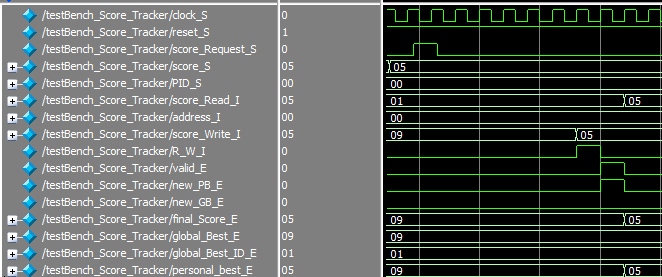


Fig. 46. New personal Best

## Digit\_2\_Count\_Up

The purpose of this module is to keep track of the game score up to 99 and to reset the score when instructed to.

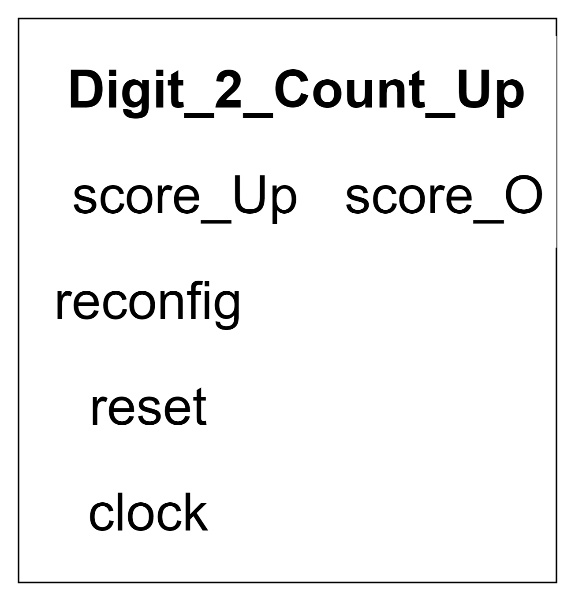


Fig. 47. Digit\_Count\_Up Symbol

* clock – 1 bit
  + synchronization signal
  + 50 [Mhz]
* reset 1 - bit
  + resets the module to its default state
  + active low
* count\_Up – 1 bit
  + triggers the module count up
  + expects a 1 clock pulse
* digit\_0 – 4 bit
  + the ones digit of the counter
  + base 10
* digit\_1 – 4 bit
  + the tens digit of the counter
  + base 10
* reconfig – 1 bit
  + resets the counter to 0

## Digit\_Count\_Up

The purpose of this module is to count from 0 to 9 including 0 and 9 then trigger a time out on reaching 9 and resetting to 0. Sub module of 2 digit up counter.

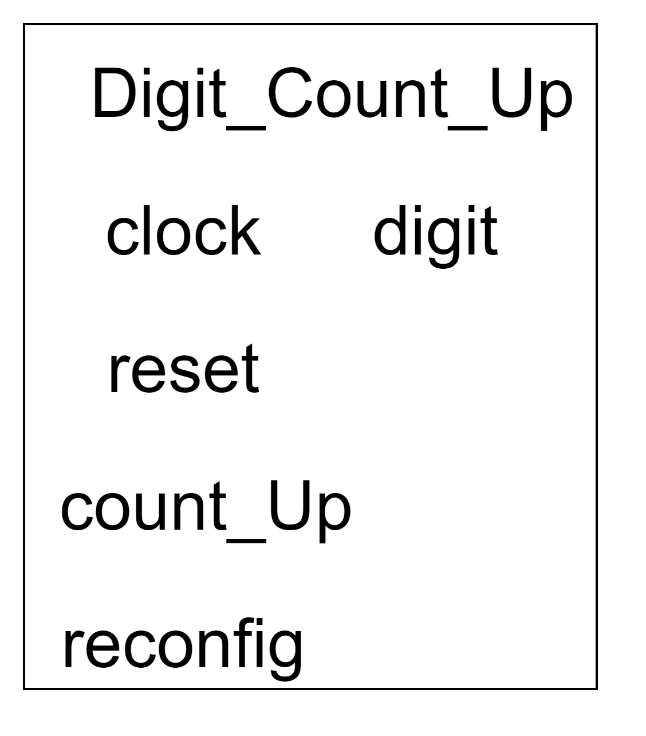


Fig. 48. Digit\_Count\_Up Module

* clock – 1 bit
  + synchronization signal
  + 50 [Mhz]
* reset 1 - bit
  + resets the module to its default state
  + active low
* count\_Up – 1 bit
  + triggers the module count up
  + expects a 1 clock pulse
* carry – 1 bit
  + outputs a 1 when the timer resets from 9 to 0
  + 1 clock pulse length output
* digit – 4 bit
  + the current count of the counter
* reconfig – 1 bit
  + resets the counter to 0

## Best\_Score\_Display

The purpose of this module is to display the best score and best scoring player, and the personal best score and their ID. This display is done on 6 7 segment displays and the precise format is shown below.

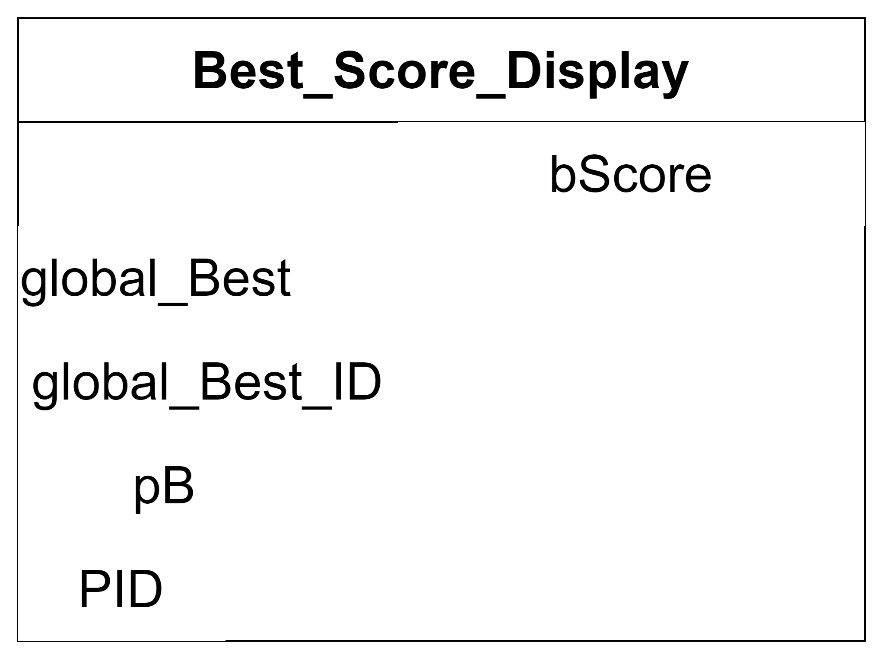


Fig. 49. Best\_Score\_Display symbol

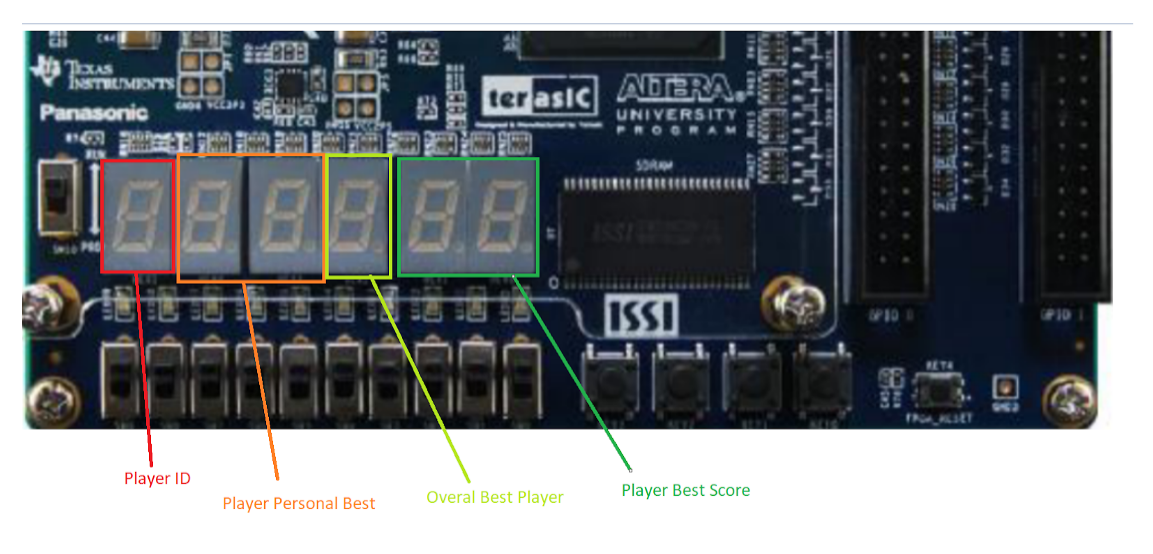


Fig. 50. Display Format

* global\_Best – 8 bit – input
  + the global best score, 2 digits decimal
  + [7:4] digit 1
  + [3:0] digit 0
* global\_Best\_ID – 5 bit – input
  + the ID of the best player
* pB – 8 bit – input
  + the personal best score of the logged in player 2 digits decimal
  + [7:4] digit 1
  + [3:0] digit 0
* PID – 5 bit – input
  + The ID of the currently logged in player
* bScore – 42 bit - output
  + the combined display frame for personal/global best score
  + [6:0] hex0
  + [13:7] hex1
  + [20:14] hex2
  + [27:21] hex3
  + [34:28] hex4
  + [41:35] hex5

## Player\_Score\_Display

The purpose of this module is to format the players final score in a way that can easily be interpreted by the player.

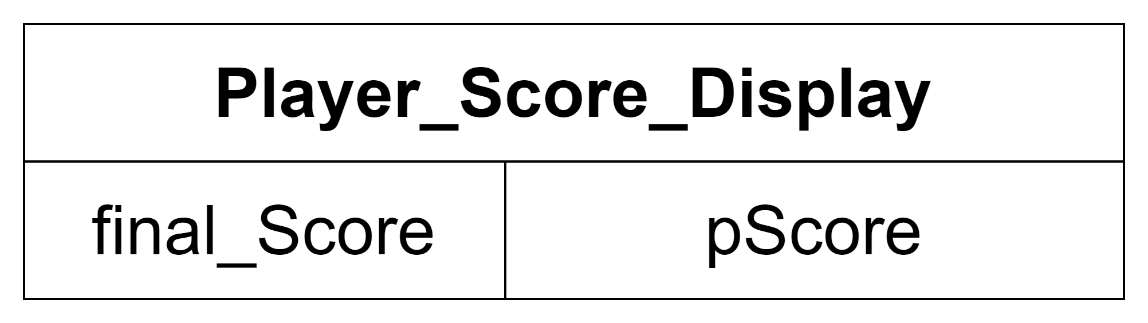


Fig. 51. Player\_Score\_Display Symbol

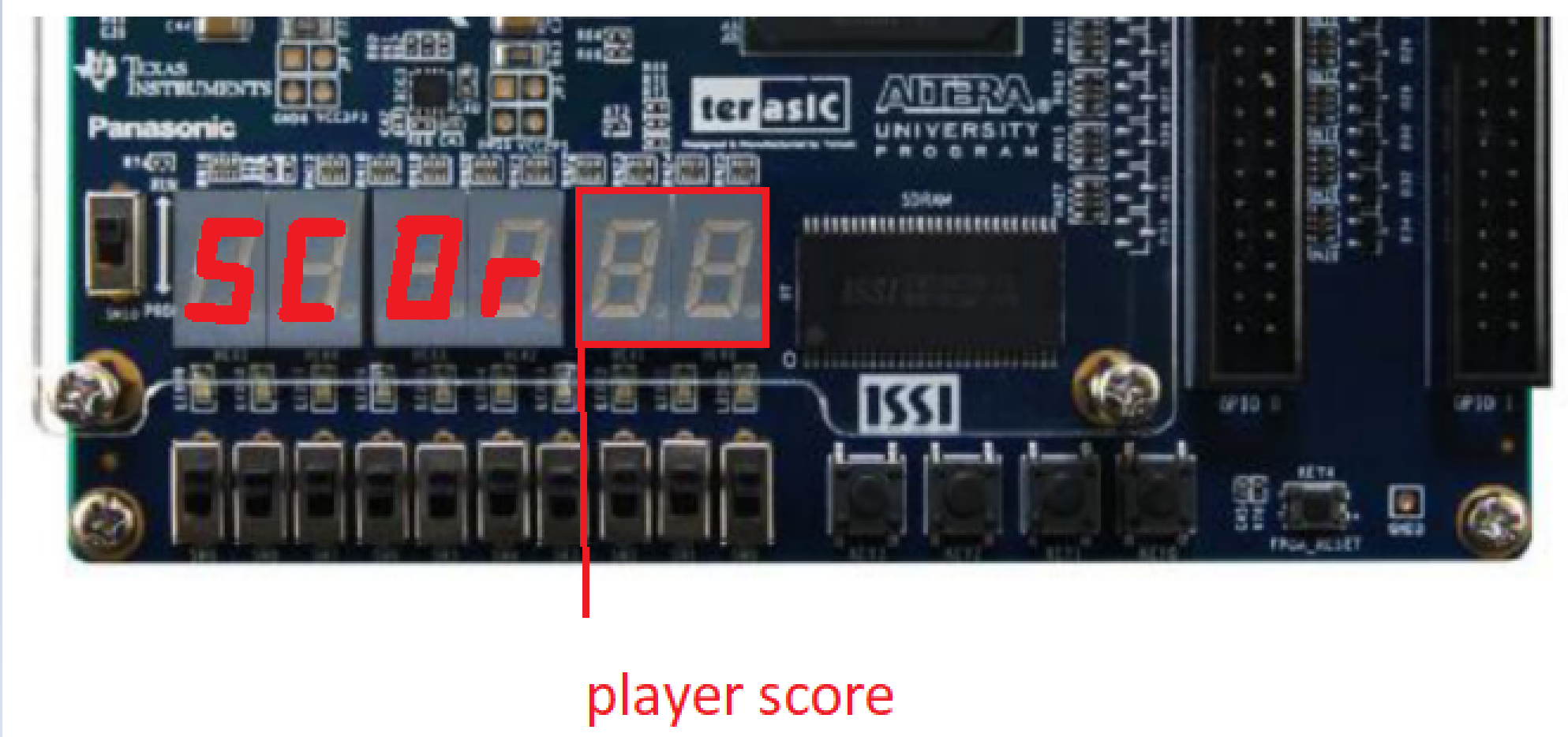


Fig. 52. display format

* Final\_Score – 8 bit – input
  + The 2-digit decimal score of the player
* pScore – 42 bit - output
  + the combined display frame for the players score that round
  + [6:0] hex0
  + [13:7] hex1
  + [20:14] hex2
  + [27:21] hex3
  + [34:28] hex4
  + [41:35] hex5

## Decoder\_7SegAnode

The goal of this module is to take in a 4 bit number and decode that number into a form that can be easily used to drive a 7 segment display.

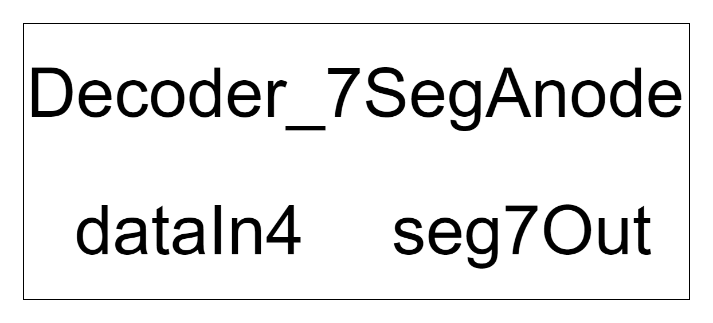


Fig. 53. Decoder\_7SegAnode Symbol

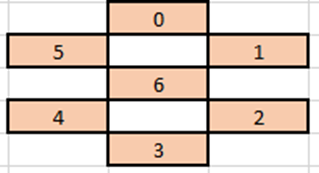


Fig. 54. 7 segment display bit map

* dataIn4 – 4 bit
  + number to be decoded and displayed
* seg7Out – 7 bit
  + corresponds to the 7 segments in the 7 segment display via the bit map

### Decoder\_7SegAnode Simulation

This module was tested by providing every combination of its 4-bit input signal. The resulting output was then applied to the 7-segment common anode display as well as checking against the required table, figure 5 for signal map, to double check that they produce the right output. All test cases are important however, the test case to display “F” is the most critical as it is required to produce an absolute minimum viable product as described in the lab 1 requirements. The other cases are helpful but are not required for the final scoring indicator and the user inputs can be extracted from looking at the inputs.

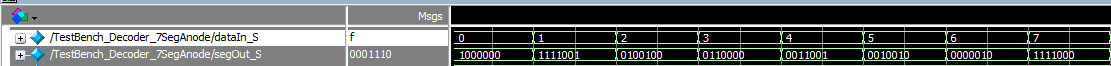


Fig. 55. Decoder simulation digits 0 – 7



Fig. 56. Decoder simulation 8 – F

## Seg\_7\_Control

The purpose of this module is to multiplex all the possible display modes together to allow another module to choose display mode.

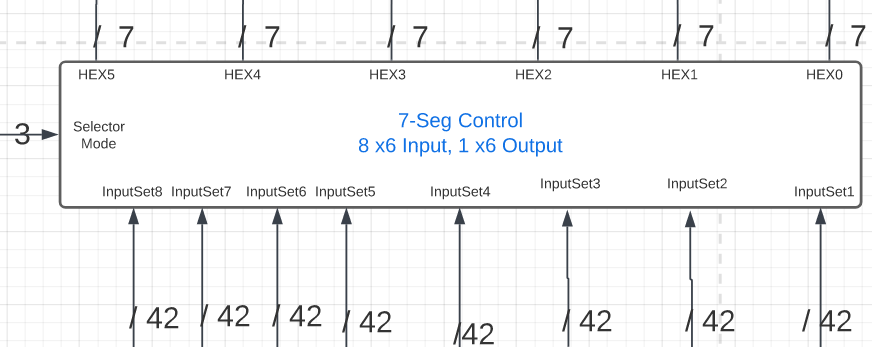


Fig. 57. Seg\_7\_Control Symbol

* SelectorMode – 3 bit – input
  + Selects what input set should be passed to the output
* InputSet1 – 42 bit – input
  + 1 set of 6 7 segment displays to be displayed
* InputSet2 – 42 bit – input
  + 1 set of 6 7 segment displays to be displayed
* InputSet3 – 42 bit – input
  + 1 set of 6 7 segment displays to be displayed
* InputSet4 – 42 bit – input
  + 1 set of 6 7 segment displays to be displayed
* InputSet5 – 42 bit – input
  + 1 set of 6 7 segment displays to be displayed
* InputSet6 – 42 bit – input
  + 1 set of 6 7 segment displays to be displayed
* InputSet7 – 42 bit – input
  + 1 set of 6 7 segment displays to be displayed
* InputSet8 – 42 bit – input
  + 1 set of 6 7 segment displays to be displayed
* HEX0 – 7 bit – output
  + The driving signals for a 7 segment display
* HEX1 – 7 bit – output
  + The driving signals for a 7 segment display
* HEX2 – 7 bit – output
  + The driving signals for a 7 segment display
* HEX3 – 7 bit – output
  + The driving signals for a 7 segment display
* HEX4 – 7 bit – output
  + The driving signals for a 7 segment display
* HEX5 – 7 bit – output
  + The driving signals for a 7 segment display

## Variable\_Timer

The purpose of this module is to create a variable delay in multiples of 100 ms following the formula of 6 - Speed = time\* 100 [ms].

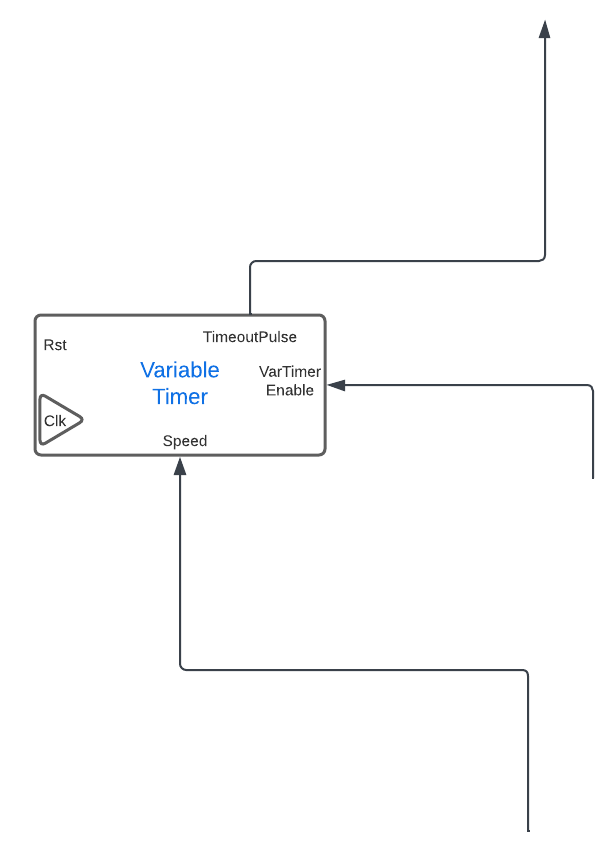


Fig. 58. Variable Timer Symbol

* Rst – 1 bit – input
  + Active low reset number
* Clk – 1 bit – input
  + Synchronization signal
* Speed – 2 bit – input
  + Input signal to decrease the time out of the timer
* VarTimerEnable – 1 bit – input
  + Enables the timer on constant high
  + Resets the timer to 0 on low
* TimeOutPulse – 1 bit - output
  + A 1 clock length pulse
  + Output on timer timeout

### Varible\_Timer Simulation

The goal of the simulation is to show that the timer will count according to 6 - Speed = time\* 100 [ms].

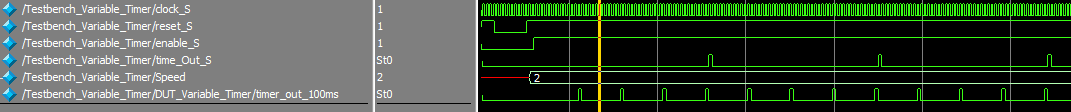


Fig. 59. Variable Timer set to timeout at 400 ns

## ButtonShaper module

Fig. 60 shows the symbol for the ButtonShaper module, with its inputs and outputs labeled.

Diagram, schematic

Description automatically generated

Fig. 60. ButtonShaper module with inputs and outputs labeled.

The inputs and outputs are as follows:

**Inputs:**  
clk, rst – system inputs described above.  
B\_in – Debounced button signal from hardware button. Default signal is HIGH, then becomes a long LOW signal as the button is pressed, and HIGH again when released.

**Output:**  
B\_Out – Default LOW signal that will output a single clock cycle HIGH when activated.

The purpose of the ButtonShaper module is to take an input from a hardware button which start HIGH and go to a long LOW signal when pressed, and transform it into a single clock cycle LOW HIGH LOW pulse. The expected waveforms are shown in Fig. 61.

Diagram

Description automatically generated

Fig. 61. Expected waveforms for ButtonShaper module.

This behavior is achieved using a finite state machine as shown in Fig. 62.

Diagram

Description automatically generated

rst

Fig. 62. FSM for ButtonShaper module.

B\_Out is 1’b0 until B\_In changes from 1’b1 to 1’b0 (the hardware button is pressed). When this happens, B\_Out goes to 1’b1 for a single cycle in the PULSE state, then goes back to 1’b0 to wait until the button is released.

### ButtonShaper simulation

The ButtonShaper module was tested with ModelSim using a testbench file (see appendix for code). The testbench tests two different button press durations: 12 cycles and 9 cycles. The button presses will be magnitudes longer in practice because the 50 MHz clock is so fast, but for simulation this is sufficient. The first button press is shown in Fig. 63.

Graphical user interface

Description automatically generated

Fig. 63 - Simulated button press lasting 12 cycles.

The first signal change other than the clock is the rst signal, just to ensure that all signals are initialized properly. The long red box shows the button press (HIGH to LOW) for 12 cycles, and the B\_out signal is in the small red box above it. It is a single LOW HIGH LOW pulse as expected. A second test case with only 9 cycle is shown in Fig. 64.

Graphical user interface

Description automatically generated

Fig. 64 - Second button press simulation.

The second test shows a similar result to the first. A single pulse output for the long button press input. It is interesting to note the last two signals, State and StateNext. State echoes StateNext because this was created using a two-procedure FSM vs. the more common one-procedure FSM used in the AccessController module. The two-procedure FSM has a StateNext signal that changes asynchronously to the clk signal, vs. all signals changing with the rising edge of clk in a one procedure FSM and elimination of the StateNext signal. This can be seen in the appendix by examining the always@ block of each modules’ code.

## Timer\_1ms

The goal of this module is to output a 1 clock pulse width pulse every millisecond if the enable pin is held high allowing it to act as a timer. This timer uses a LFSR based counter to vastly reduce the module cost compared to simple counters.

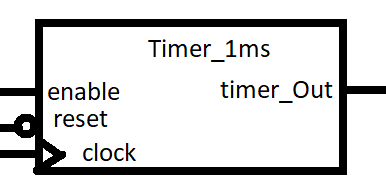


Fig. 65. Timer\_1ms module with inputs and outputs labeled.

* clock – 1 bit
  + Synchronization signal
  + Built with the assumption of a 50 [Mhz] clock
* Reset – 1 bit
  + Active low signal to reset module to default state
* Enable - 1 bit
  + Signal that controls if the timer should continue counting or not
* Time\_Out – 1 bit
  + Signal that indicates if the timer has reached its time out
  + 1 clock pulse width pulse

## Timer\_100ms

The goal of this module is to output a 1 clock pulse width pulse every 100 milliseconds if the enable pin is held high allowing it to act as a timer.

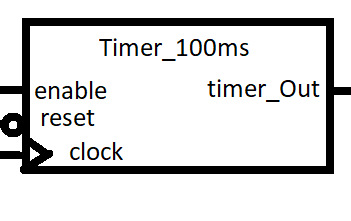


Fig. 66. Timer\_100ms module with inputs and outputs labeled.

* clock – 1 bit
  + Synchronization signal
  + Built with the assumption of a 50 [Mhz] clock
* Reset – 1 bit
  + Active low signal to reset module to default state
* Enable - 1 bit
  + Signal that controls if the timer should continue counting or not
* Time\_Out – 1 bit
  + Signal that indicates if the timer has reached its time out
  + 1 clock pulse width pulse

# FPGA Board Testing Results

The first section of the assembled game to be tested is the log in and log out feature. To test the log portion we must first enter 1 of the 5 possible 4 digit username’s while on the ‘ID’ screen using the leftmost set of slide switches and the password enter button KEY2.

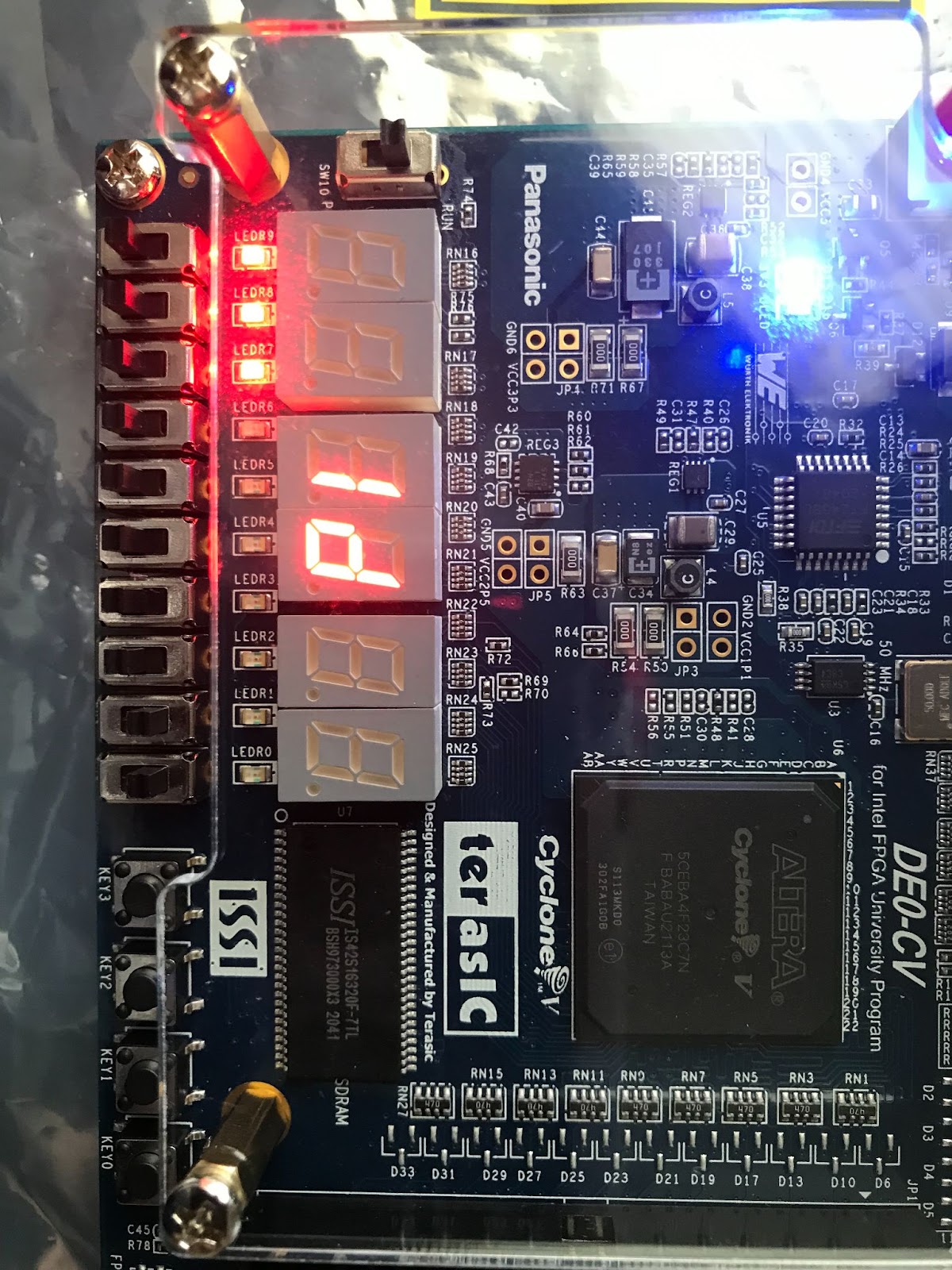


Fig. 67. Entering user ID.

To test the password checking portion we must enter the 6 digit password associated with the previously entered username. This password is entered on the ‘PSCO’ screen in the same manner that the username was entered.

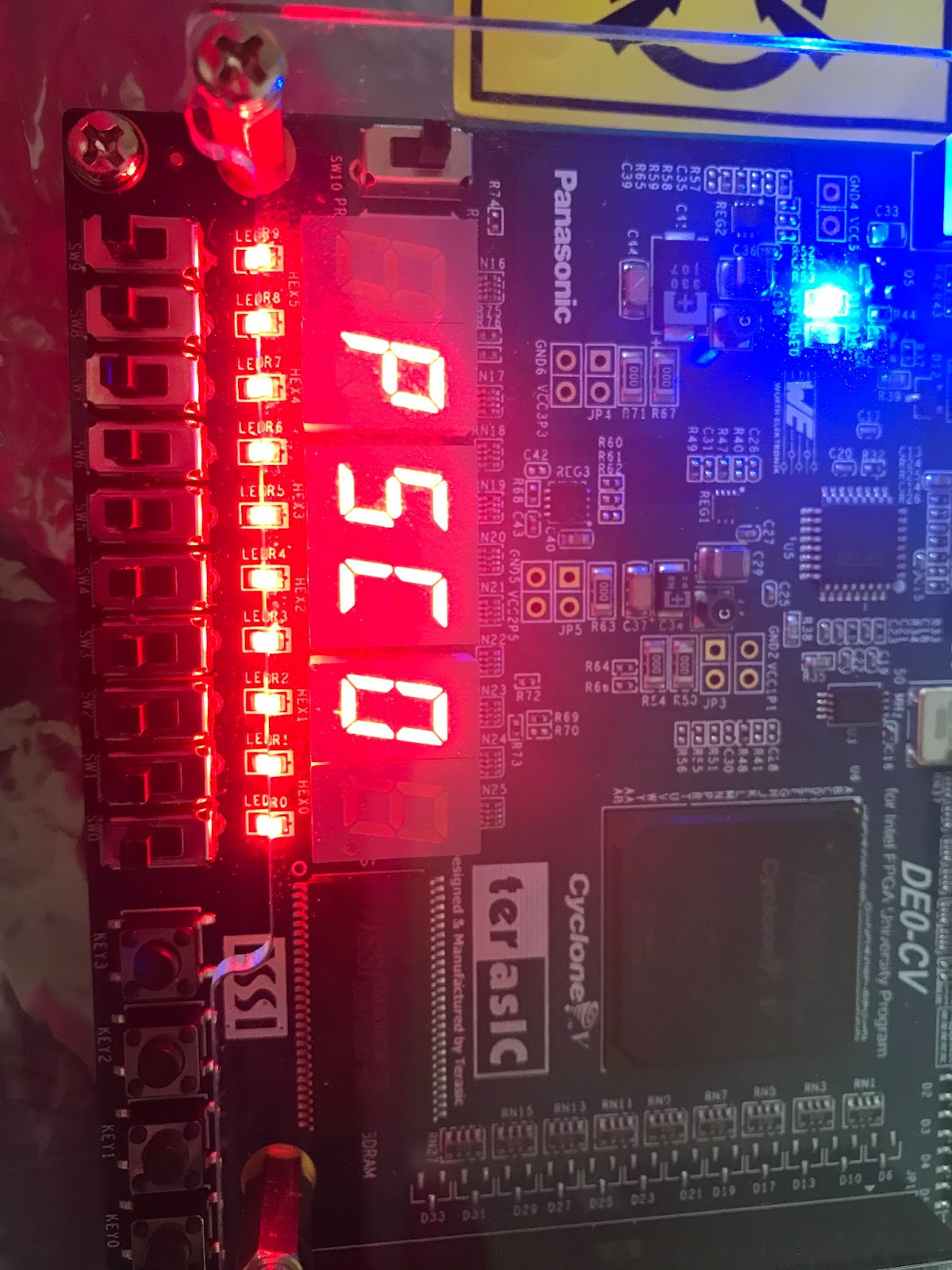


Fig. 68. Entering the user passcode.

In order to test the level length, fuel gauge, the right most set of slide switches can be adjusted and their setting will be reflected by adjusting the number of LEDs are lit under the 7 segment displays.

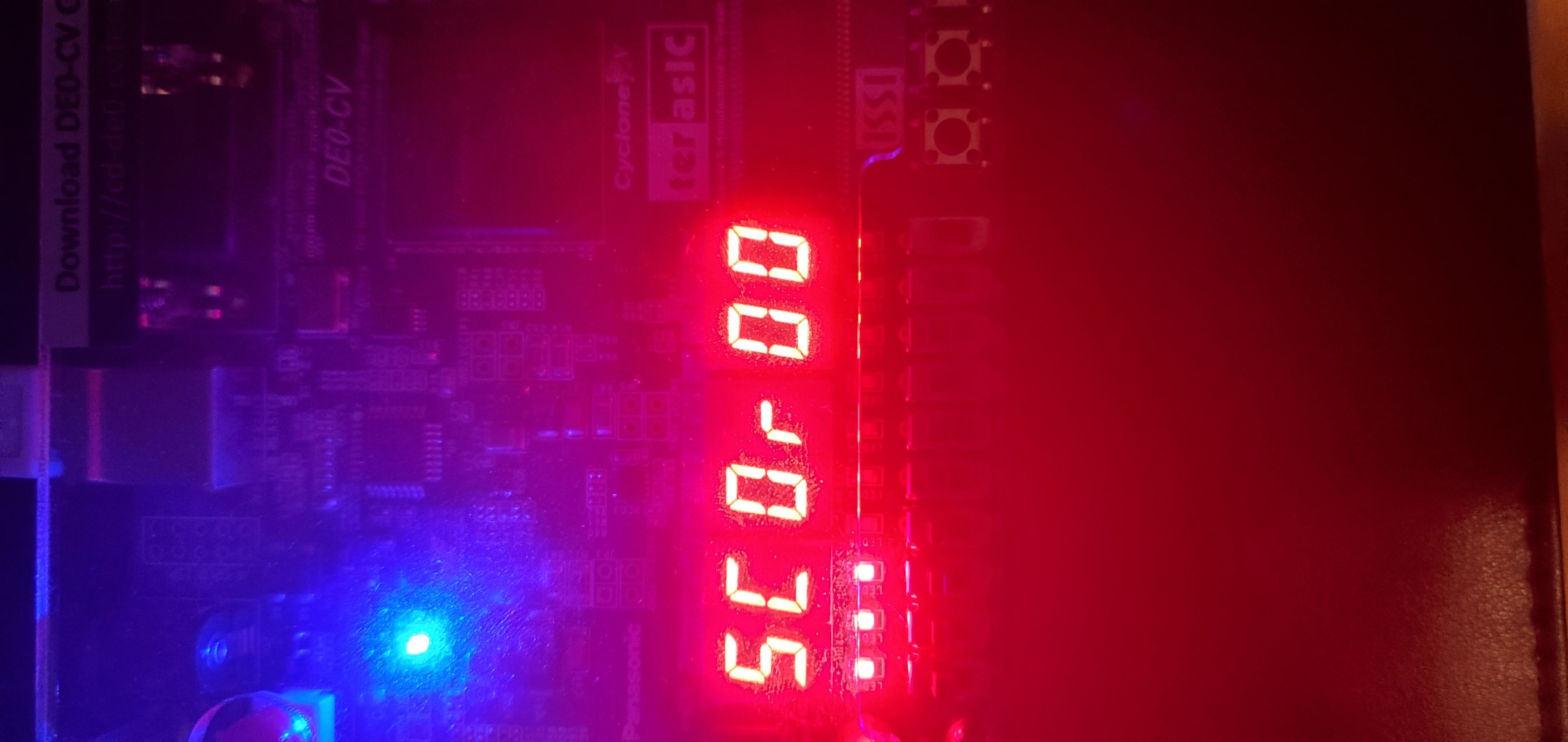


Fig. 69. Personal current score display, with "fuel gauge" timer set to 3.

In order to test the basic gameplay loop we must press the start game/toggle position button KEY1 on the personal score screen. The fuel gauge slide switches should be set to 1 to make scoring easier to facilitate testing. The tester should then attempt to dodge the oncoming asteroids.

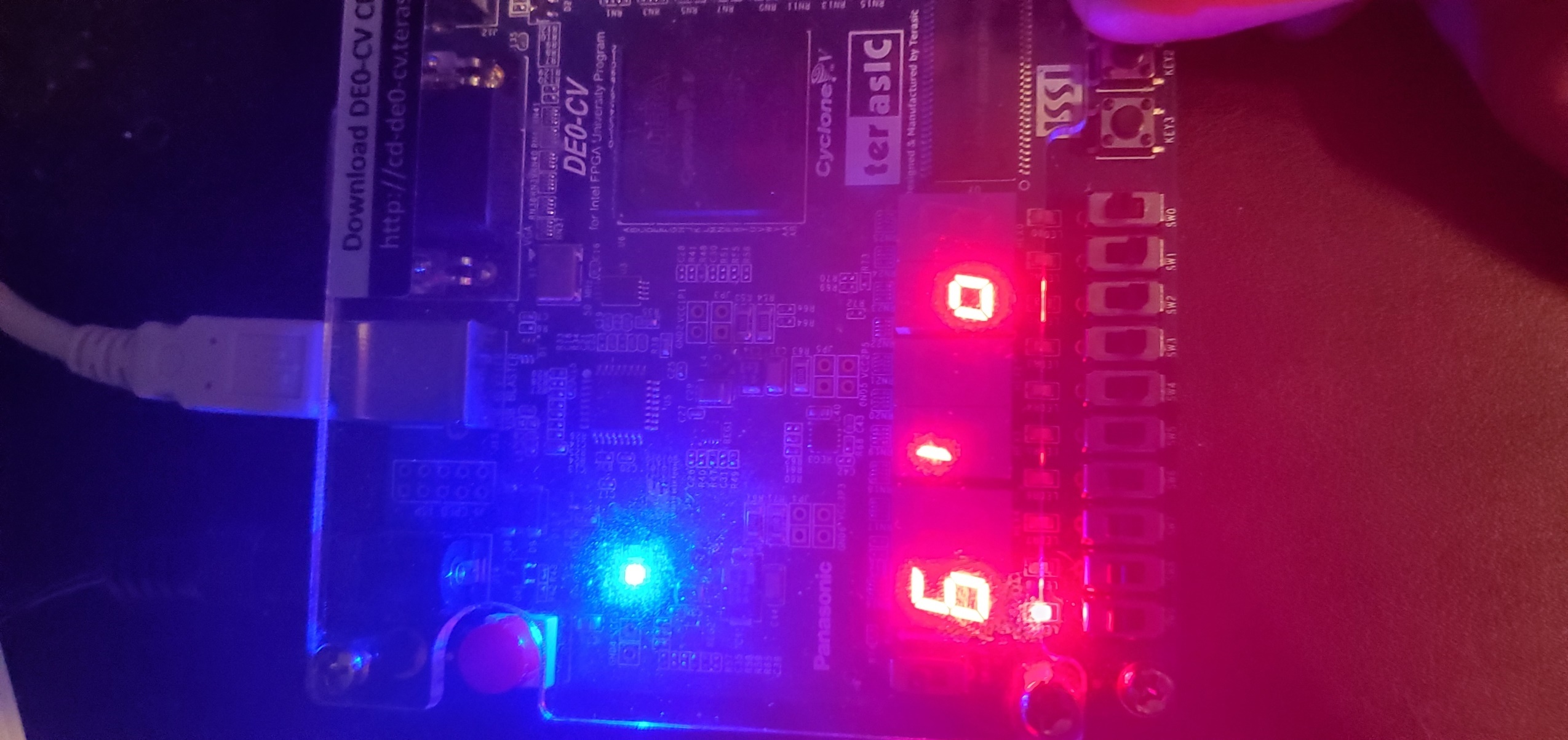


Fig. 70. Gameplay as user avoids obstacles.

On a successful round test tester should see the ‘SUCC’ screen, followed a few seconds later by the player score screen. If the tester achieved a personal best the LEDs should flash. If the tester achieved a global best the LED bar will ramp up. After this the player can start a new round to further increment their score and the asteroids will be slightly faster.

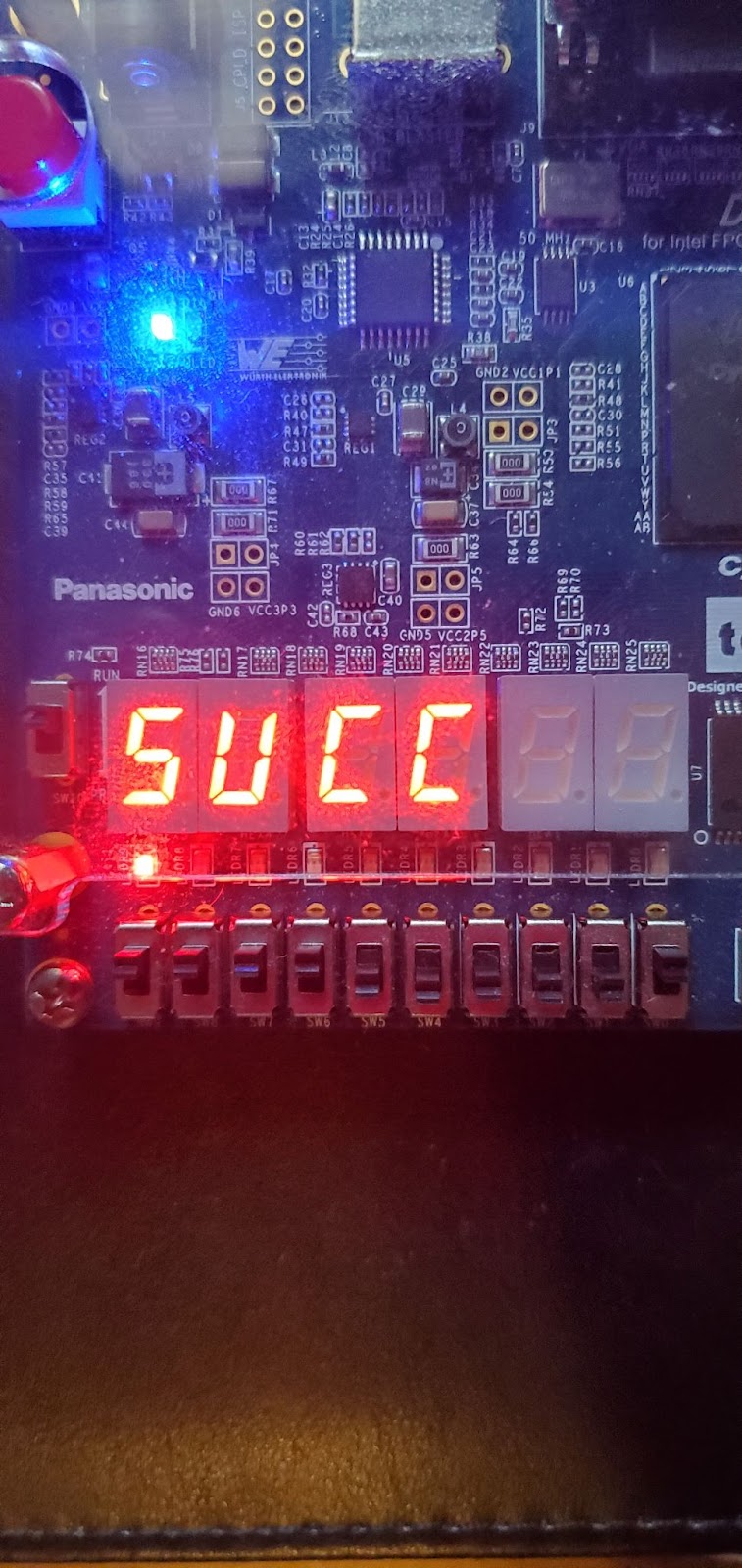


Fig. 71. User has passed a level and reached the SUCCESS display.

If the player failed the round they will see the ‘FAIL’ screen, no personal or global best LED animations will be played. They will be presented by the global best score screen, then a few seconds later the player score screen.

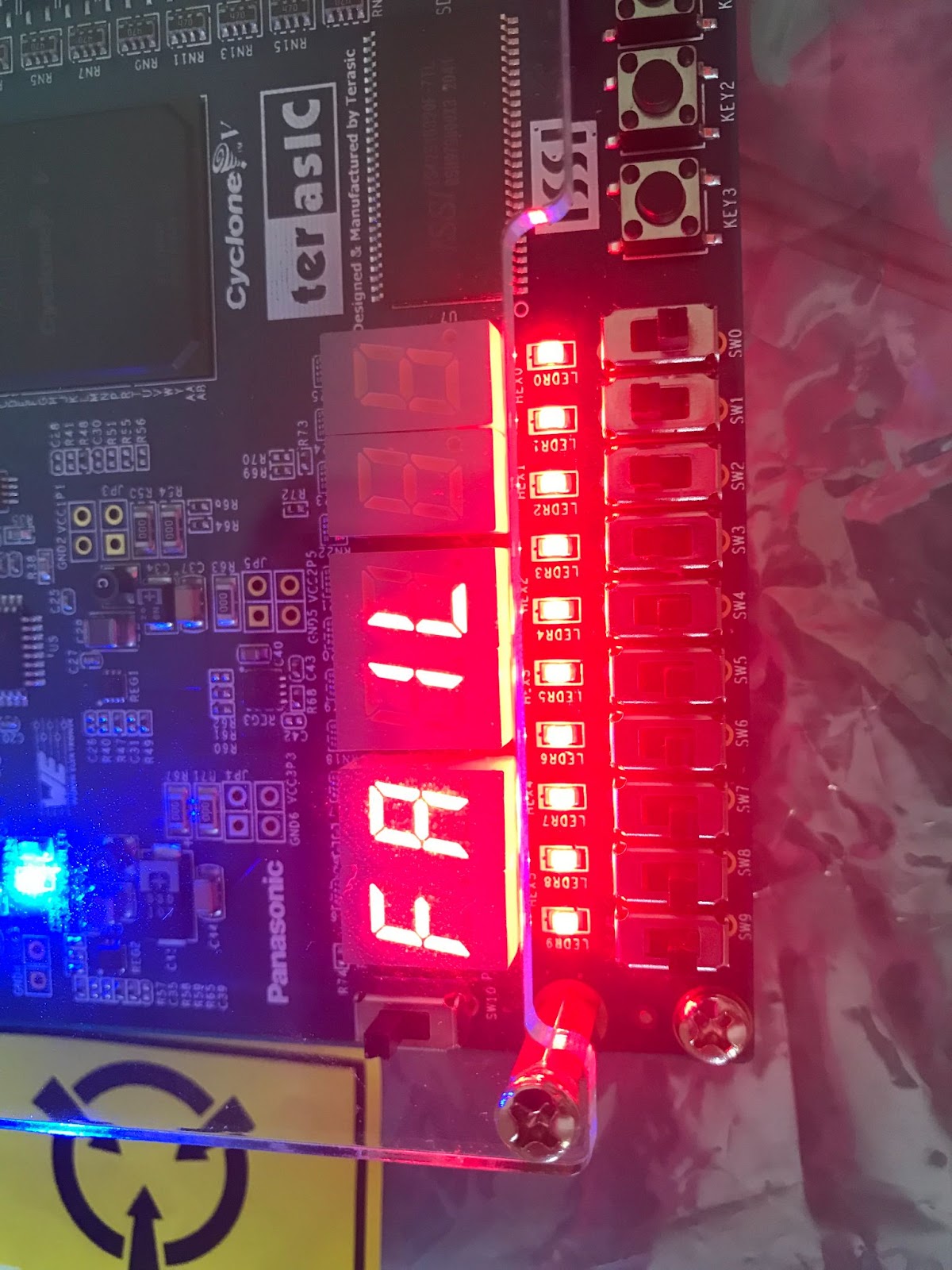


Fig. 72. User has crashed into an object and reached the FAIL screen.

To test the log out functionality all the tester has to do is to press the logout button, KEY3, on either the global display screen or the personal display screen. The player should not be able to log out in the middle of a game.

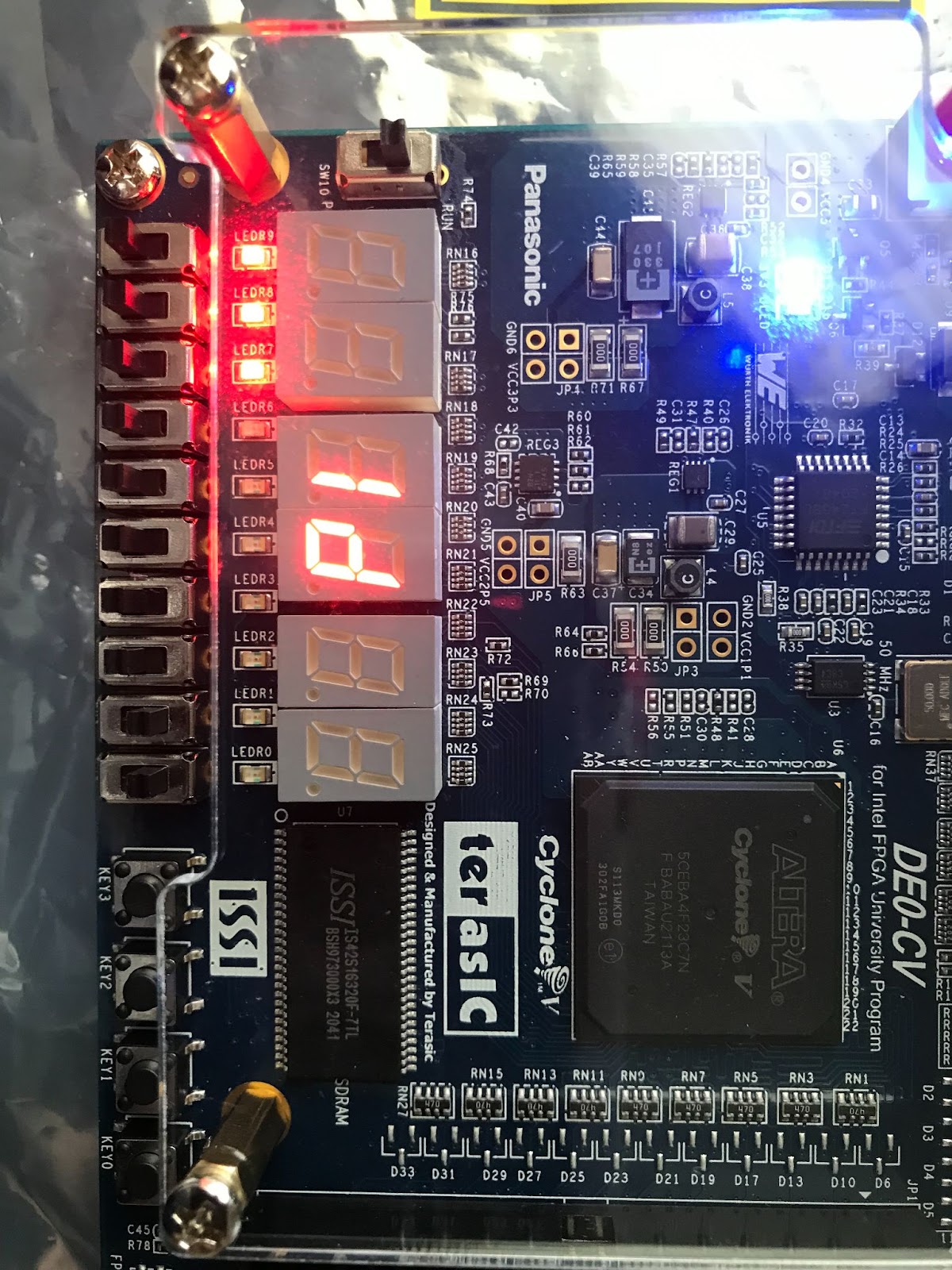


Fig. 73. Screen after player has logged out.

## Video Demo Links

The following video clips show the major features of the game.

**Clip 1**: Gameplay and log out demo.  
User starts game, changes fuel gauge/level timer, earns personal best, global best, and logs out.

[Gameplay and logout demo Team Q4](https://drive.google.com/file/d/1yREnGMH9rBzOy6Fyk6hytyyZSeqHbM-Z/view)

**Clip 2**: Game overview presentation

[Team Q4 Presentation](https://uofh.sharepoint.com/:v:/s/ADDFinalProject-DodgeDuckDipDiveandDodge/Eb3g4q1aTahJsWcWZG-RDOUBa8Sjy_eIz3-87S-MxkUfSQ?e=k7Rpxg)

**Clip 3**: Multi-User login.  
Player 1 logs in, plays a short game getting global best, logs out. Guest account logs in, plays a short game earning personal best, logs out.

[Multi-User Login Team Q4](https://share.icloud.com/photos/0a0kHasEZFyKeTjaLGgK1htmg)

# Conclusion

In this project we set our to build a side-scrolling asteroid dodging game. In the pursuit of this goal we ended up creating many different subsystems to support this including score tracking, log in/log out, adjustable difficulty, and interesting but limited graphics. The largest difficulty in this project came from intergrating our slightly different versions on how different subsystems would interact. From this issue we learned the value of being explicit about how different sub systems should interact before creating them. What we would do in the future to improve this project is adjust how scoring is handled to give more points per round at higher difficulties. At the time of submitting this report the game is feature complete with no known outstanding bugs.