University of Houston

Department of Electrical and Computer Engineering

**Final Project: Memory Game**

Lil-Engineers Team Members:

Sinan Kekic (Team Lead)

Faiza Badarpura

Sayra Rodriguez

Mohammed Kassif

Andrew Davis

ECE 5440 - Advanced Digital Design

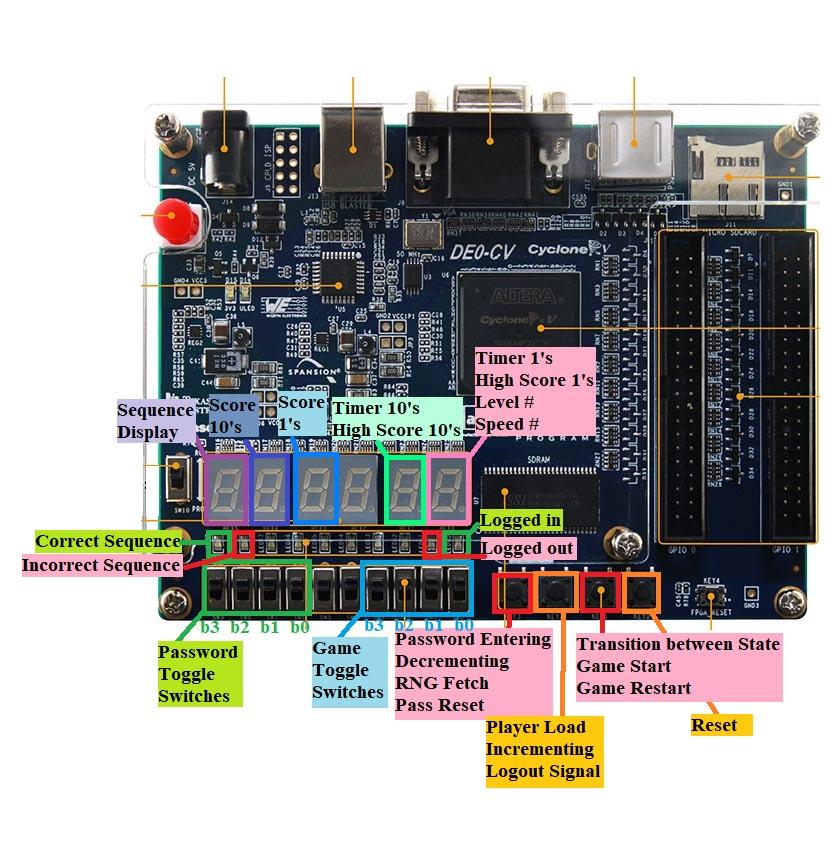
Dr. Yuhua Chen

Spring 2022

**9.2 Introduction**

The FPGA-based Memory Game is a quick logic game that allows players to sharpen their binary knowledge with a single user game.

The game requires one player, the player will have four toggle switches which represent four binary bits that will be used to enter their binary sequence. This will be achieved by using the random generator button that will essentially display a random sequence for the player to see and memorize before using the toggle switches and player load button to lock in their sequence.



*Figure 1. Labeled FPGA Board*

The game is accessible to 6 players with the use of a user ID and password and to a guest with a different set of rules. In order to get the game started the user will use one of the following IDs and corresponding password.

|  |  |  |
| --- | --- | --- |
| **User** | **User ID** | **Password** |
| Andrew | 8-9-2-6 | 1-2-3-4-5-6 |
| Faiza | 4-4-5-3 | A-B-C-D-E-F |
| Sayra | 7-6-0-7 | E-F-1-2-1-2 |
| Mohammed | 6-0-9-2 | 8-9-A-B-C-D |
| Sinan | 2-8-1-6 | 1-2-A-B-C-D |

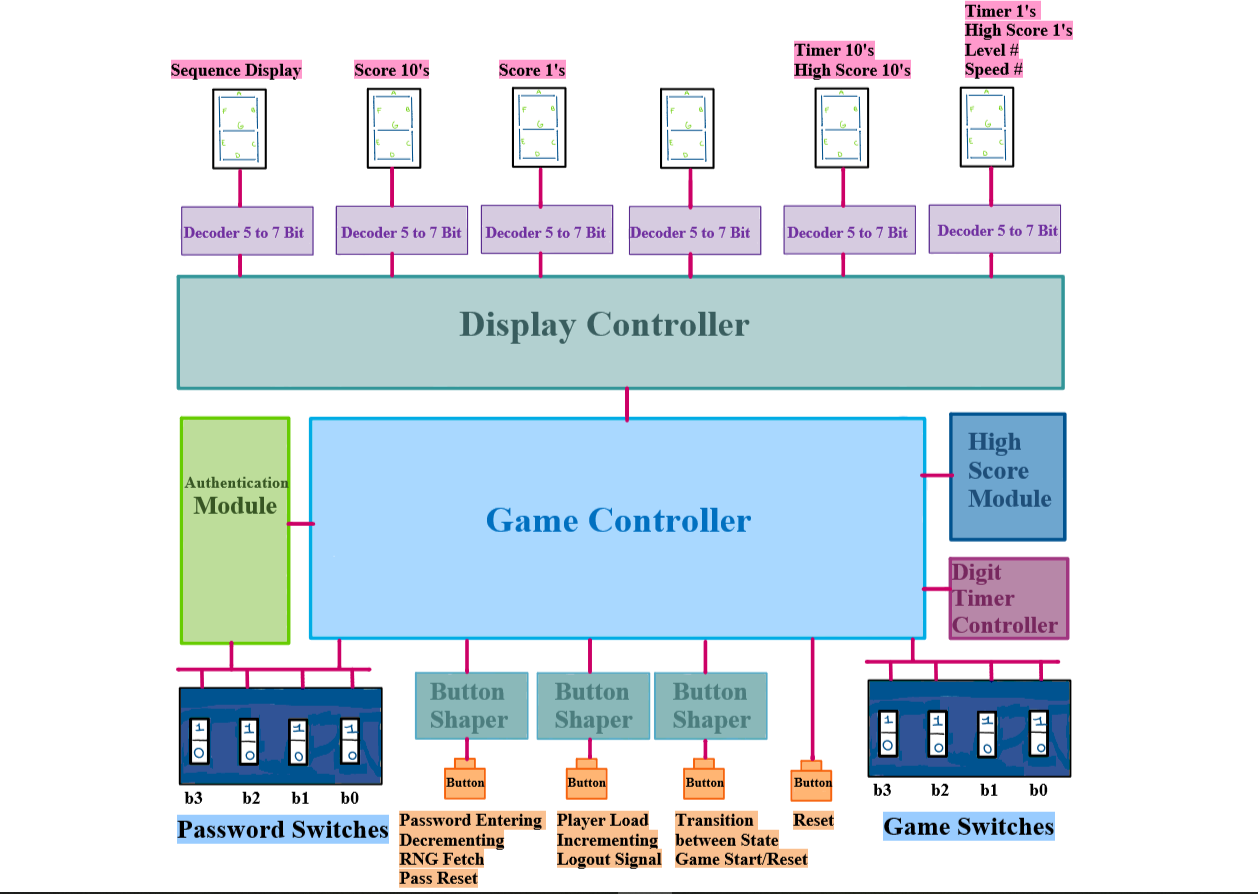
The 4 digit user ID and 6 digit password is entered value by value using the password toggle switches and password load button. If the passcode is entered correctly the logged-off LED will turn off and the logged-in LED will turn on.’

Once this initial step is completed the user is ready to begin configuring their game. The player will configure the time each round will last, the speed at which the sequence will be displayed and the level they want to play in. This will be done in the same way for all three settings. The user will use the incrementing or decrementing button to change the timer by increments of 10. The configuration will be locked in using the transition between state button. This will trigger the next configuration to be displayed and the following 2 configurations will also be set using the incrementing, decrementing and the transition between state button.

Once the configuration is set up the player will be put on standby mode that so that the user can start the game when they are ready with the press of the game start button. The player begins the game by pressing the game start button, this will trigger the game to start. The user will then press the RNG button to begin loading and displaying the random sequence on the display screen. The user will then use their toggle switches and player load button to set each value of the sequence they just saw and memorized. An LED will let the user know if they entered the correct number. This will then give the user the ability to load another sequence if their timer hasn’t run out.Once the timer runs out of time the game will enter the game over state which will display the score for the round played and the global or high score on the last screen.

**9.3 System Architecture Design**

The memory game is made up of a game controller, 3 instances of the button shaper module, a display controller with 6 instances of the 5 but to 7 bit decoder module, a high score module, a digit timer module, and an authentication module. These modules are able to create a level of security before the player can begin their game. Once the player is logged in with their user ID and password the modules are able to connect the game toggle switches and buttons to begin the game. This will begin by configuring the timer, speed, and level with the toggle switches and load button. This will then put the player on standby until they are ready to start the game. When the game starts the player will load the random number sequence with the designated buttons, and then use their toggle switches and load button to enter the sequence they just saw. You can learn more about the function of each module and see a visualization of the top-level system architecture below.

****

*Figure 2. Top-Level System Architecture*

**Game Controller Module**

The following game controller module is made up of the following input and output signals:

o **Input Signal: rst, clk, loggedInSignalInput, passInput, playInput, passBeginAddress, timeout, Button1\_Wire, Button2\_Wire, Button3\_Wire**

§ The input signal rst is connected directly to the 1-bit binary reset button. This will be used to initiate the device and reset values back to zero.

§ The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency. This signal is primarily responsible for transitioning states and updating signals on the rising edge of the clock cycle.

* + The loggedInSignalInput is fed from the authentication module. Before the user logs in, the game controller stays in a “LoggedOut” state until the user successfully logs into the device. During this time, the user will not be able to play the game. Once the user successfully logs in, the device receives a high input from the loggedInSignalInput and allows the user to begin playing the game.
  + The passBeginAddress is also fed from the authentication module. If the user chooses to reset their password, the module takes in the beginning address of the user’s password and writes a new password using this address.
  + The passInput signal is a 4-bit value corresponding to the four password toggle switches. These switches are used to write values when resetting the password.
  + the playInput signal is a 4-bit value corresponding to the four player toggle switches. While the game is active, the user will use these switches to input the sequence back into the device.
  + The timeout signal is an input fed from the Digit Timer Module. When the timer counts down to 0, this signal goes to high to indicate that the game is over. This transitions the game from the Game Active state to Game Over.
  + The Button1\_Wire is fed from the Button Shaper module corresponding to the first button. Depending on the State of the game controller, this button serves different purposes.
    - Reconfig Time: Pressing Button 1 in this state pulses the output signal decrementTime10. See outputs for a detailed explanation.
    - Reconfig Speed: Pressing Button 1 in this state decrements the Speed of the Game by 1 (assuming the Speed is at 2 or 3)
    - Reconfig Level: Pressing Button 1 in this state decrements the Level of the Game by 1 (assuming the Level is at 2 or 3)
    - Standby/Game Over: Pressing Button 1 in these States allows the user to enter the Pass Reset State, which allows the user to reset their password.
    - Pass Reset: When the user selects a password using the password toggle switches, pressing this button allows them to load that value. In the pass reset state, they will have to press this button 6 times in order to load all 6 password values.
    - Game Active: When the user presses this button, a randomly generated number sequence is generated and displayed to the 7-segment display.
  + The Button2\_Wire is fed from the Button Shaper module corresponding to the second button. Depending on the State of the game controller, this button serves different purposed.
    - Reconfig Time: Pressing Button 2 in this state pulses the output signal incrementTime10. See outputs for a detailed explanation.
    - Reconfig Speed: Pressing Button 2 in this state increments the Speed of the Game by 1 (assuming the Speed is at 1 or 2)
    - Reconfig Level: Pressing Button 2 in this state increments the Level of the Game by 1 (assuming the Level is at 1 or 2)
    - Standby/Game Over: Pressing Button 2 in these States allows the user to log out. This will reset the game controller state back to Logged Out. In order to play the game, they will need to log back in with a user ID and password.
    - Game Active: After the user generates a sequence with the RNG button, the user will be able to load in their player toggle switches using this button. They will have to press this button for each digit.
  + The Button3\_Wire is fed from the Button Shaper module corresponding to the third button. Depending on the State of the game controller, this button serves different purposed.
    - Reconfig States: During these States, the user can press the third button in order to lock in their desired values during the Reconfiguring states.
    - Standby State: Pressing this button in the Standby state allows the game to begin and transitions the game to the Game Active State.
    - Game Over: When the user has finished their round of the game, pressing this button allows the user to restart the game by transitioning to the Reconfig Time state.

o **Output Signal: numToFlash, logOutSignalOutput, decrementTime10, incrementTime10, displayState, speedNumber, levelNumber, noNumToFlash, score10s, score1s, timerEnable, ReConfig, mux, correctLED, incorrectLED**

§ The numToFlash signal is a 4-bit output fed to the Display Controller. This value will correspond to the RNG sequence that needs to be flashed while the game is active. This display is controlled by another signal noNumToFlash. When a value should not be displayed, noNumToFlash goes high, which results in the 7-segment display for the RNG sequence turning off (displays “-“).

§ the logOutSignalOutput is fed to the Authentication module. When the user decides to log out, this signal goes high, which tells the Authentication module to go back to its initial state and check for the user ID input.

§ The decrementTime10 and incrementTime10 are signals fed to the Digit Timer module. These values pulse and decrement or increment the timer by 10 seconds respectively.

§ The displayState signal is fed to the Display Controller module. This signal is responsible for telling the display module what state the game controller is in, and consequently what values / letters should be displayed to the 7-segment displays.

§ The signals speedNumber and levelNumber correspond to the values in the Reconfig Speed and Level states respectively. These outputs tells the Display Control module what value to display for the speed and the level of the game.

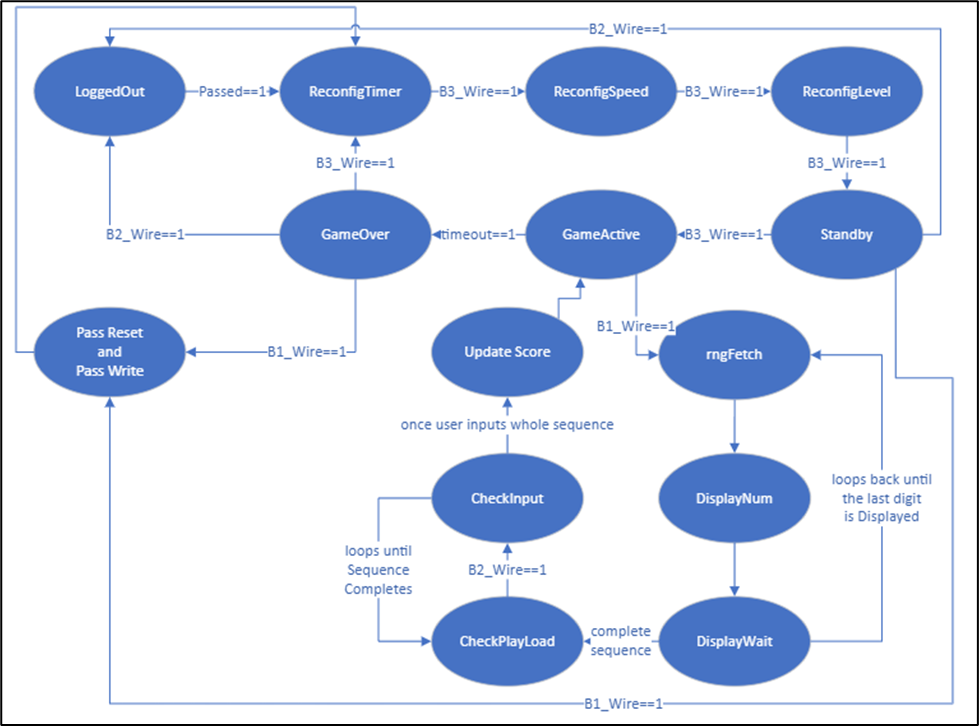
§ The two 4-bit signals score10s and score1s are used to keep track of the user’s score during the Game Active state. When the user inputs a correct sequence, these values are incremented according to the score multiplier (as decided by the level of the game). When the game ends, these values are displayed on the 2nd and 3rd to left displays to show the user’s recent score.

§ The timerEnable and ReConfig signals are fed to the Digit Timer Module. While adjusting the timer of the game, the ReConfig signal is set to high to allow the timer to be incremented or decremented. The timerEnable signal is set to high while the game is active. While it is high, the timer counts down to 0, at which point the timeout signal goes high and ends the game.

§ The mux signal is fed to the authentication module when the user chooses to reset their password. Upon pressing the Pass Reset button, the mux signal is set to high, which causes the Authentication Module to read from RAM (reset password values) rather than ROM.

§ The correctLED and incorrectLED signals are fed to the corresponding LEDs for inputting the correct or incorrect sequence. One of these values go high once the user inputs back a sequence to the device and back to low when a new sequence is fetched.

The module serves the purpose of controlling most aspects of the design, such as reconfiguring the timer, speed, and level of the game, allowing the user to play the game, and keeping track of the user’s current score. This module also allows the user to reset the password as well as log out of the current user. A finite state machine can be seen in the following figure:



*Figure 3. Finite State Machine of Game Controller*

From the above Figure, the individual states can be explained in depth:

o Logged Out: The point of this state is to ensure that the player cannot play the game until they either log in with their user ID and password or continue as a guest. Once the user successfully logs in or continues as a guest, the Authentication module passes a high signal for the “Passed” signal (loggedInSignalInput) and transitions to the Reconfig Timer state.

o Reconfig Timer: During this State, the user can increment or decrement the Timer by 10 seconds by utilizing the first two buttons. Once the user has selected their desired Time, they can use the Third button to transition to the Reconfig Speed State.

o Reconfig Speed: During this state, the user can increment or decrement the speed of the game from 1 to 3. The speed of the game is responsible for how long each digit of the sequence will be flashed to the screen during the state “Display Num”. The values can be described as:

|  |  |
| --- | --- |
| **Speed Number** | **Seconds / Flash** |
| 1 | 1.5 |
| 2 | 1.0 |
| 3 | 0.5 |

The user can also transition out of this state and to the Reconfig Level by pressing Button 3.

o Reconfig Level: During this state, the user can increment or decrement the level of the game from 1 to 3. The level of the game impacts the length of each sequence and the score multiplier the user receives for inputting the correct sequence. These values are as follows:

|  |  |  |
| --- | --- | --- |
| **Level Number** | **Sequence Length** | **Score Multiplier** |
| 1 | 3 digits | 1x |
| 2 | 4 digits | 3x |
| 3 | 5 digits | 5x |

The user can also transition out of this state and to the Standby State by pressing Button 3.

o Standby: The Standby state occurs just after the user finishes reconfiguring the desired values for the game. This state is meant to allow the user to Start the game by pressing the 3rd Button (Game Start button). During this state, the user can also choose to either reset their password (using Button 1) or log out (using Button 2).

o Game Active: The majority of the game will be controlled by this state. While in the Game Active State, the user may press the first button to generate a random sequence (to the length selected by the Level Number). This will transition the State to rngFetch. (Note: If the timer runs out during this state, the game moves onto the Game Over State.)

o rngFetch + DisplayNum + DisplayWait: The primary objective of these states is to produce and display the randomly generated sequence. These states begin at rngFetch, which receives a RNG digit from the RNG module. This value is written to a RAM module corresponding to the sequence (to be used for checking the user’s input in later States). Once this is complete, the State transitions to the DisplayNum state. During this state, the specific digit is flashed to the 7-segment display for the duration of the time set by the Speed. When this time is complete, the state transitions to DisplayWait, which turns off the RNG display for 0.5 seconds. These states will continue to loop until the sequence reaches its last digit, at which point the state transitions to Check Play Load.

o Check Play Load + Check Input: In these states, the user will attempt to input back the sequence that has been flashed to the 7-segment display. The user will first set the player’s toggle switches to the value in which they wish to input. They will then press the Player Load button (Button 2), in which the state transitions to Check Input, which compares the user’s input to the values stored in the RAM for the Sequence. These states will continue to loop until a complete sequence has been inputted in, at which point the state transitions to the Update Score State.

o Update Score: In this state, if the user successfully inputted the correct sequence, then their score will increment by the value specified by the score multiplier and the LED corresponding to the Correct Sequence will turn on. Otherwise the LED corresponding to the Incorrect Sequence will turn on. After the respective LED is turned on, the state goes back to the Game Active state, at which point the user can move onto the next sequence.

o Game Over: When the timer eventually runs out, the State transitions to the Game Over state, which will display the user’s score. During this state, the user’s high score or the global high score (default for guests) will also be displayed. This will be further explained in the High Score module. Similarly to the Standby State, the user can also choose to log out or reset their password. Otherwise, the user can press the 3rd button to restart the game and go back to reconfiguring the game.

o Pass Reset and Pass Write: If the user opts to reset their password, they may do so by toggling the password switches and inputting that value with the password entering button (button 1). Once they complete this, the state goes right back to the ReconfigTime state.

**High Score Module**

The following \_ module is made up of the following input and output signals:

o **Input Signal: rst, clk, userAddress, score10s, score1s, timeout and timerEnable**

§ The input signal rst is connected directly to the 1-bit binary reset button.

§ The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency

§ The userAddress signal is fed from the authentication module. When the user logs in, this address signal is set to their corresponding address, which will be used for updating their individual score.

§ The score10s and score1s inputs are fed directly from the game controller module. When the game completes, these inputs are used to compare their high scores. These values are also compared to the global high score. If the user’s recent scores are greater than that of the one stored in their high score, their high score will be updated to reflect these values.

§ The timeout and timerEnable inputs are primarily used for keeping track of states within the high score module. Rather than continuously comparing their scores, even when the game is not being played, these signals ensure that the user’s scores are only compared once the game has ended.

o **Output Signal: globalHighSignal, display10s, and display1s**

§ The globalHighSignal is fed to the display controller. This signal controls whether the letter “H” (for high score) or “G” is displayed during the Game Over state.

§ The display10s and display1s signals are also fed to the display controller. These values will reflect the high/global score values needed to be displayed in the Game Over state.

This module primarily serves the purpose of keeping track of the user’s high score values. When the game has ended, this module will display either the old high score (if the user fails to beat it) or the global high score.

**Authentication Module**

The Authentication module is made up of the following input and output signals:

* + **Input Signal: rst, clk, PasswordEnter, PasswordDigit, LogoutSignal, mux**
    - The input signal rst is connected directly to the 1-bit binary reset button.
    - The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency
    - The PasswordEnter is a 1-bit pulse input that sends a single pulse to confirm that a step is completed, such as entering a password digit.
    - PasswordDigit is a 4-bit input that takes the binaries of corresponding switches to give a number from 0-15.
    - LogoutSignal is a 1-bit pulse signal that logs out the current user if this signal has been raised to 1 after having someone logged in.
    - Mux is a 1-bit input signal that switches between ROM and RAM. When mux is high, this means that the game will be looking at RAM for the updated passwords and compare it with the PasswordDigit in order to log in a user. Otherwise, when mux is low, the game will read the base passwords set for each player from the ROM.
  + **Output Signal: Addr, Passed, LoggedIn, LoggedOut**
    - Addr is the 6-bit Address outputted by the authentication module when the user is logged in. This address is consistent with all Users, therefore, the device will know exactly who is logged in the game at any point in time.
    - Passed is a 1-bit output signal that is high whenever a correct combination of the UserID and Passcode has been entered.
    - LoggedIn is a 1-bit output signal that is high whenever a user has successfully entered their login UserID and Passcode. Also used for the visual LED showing that a user is currently logged in.
    - LoggedOut is a 1-bit output signal that is high whenever a user has not successfully logged in with a correct UserID and Passcode. Also used for the visual LED showing that a user is currently not logged in.

The module serves the purpose of authenticating each user who plays the game. It also serves as the controller that keeps track of the current person logged in and sends this information to the game controller so that scores are kept correctly.

**ROM\_Pass & ROM\_UserID Module**

The ROM\_Pass module is made up of the following input and output signals:

* + **Input Signal: address, clock**
    - The signal address is a 6-bit input that tells the ROM the location of the number we currently want to display or change.
    - The clock input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency
  + **Output Signal: q**
    - The q signal is a 4-bit output that displays the number currently saved at a specific address of the ROM.

The module serves the purpose of storing 6 bit passwords and 4 bit user id’s of each of the 6 users including a guest user. The ROM for the passwords stores 6 bit passwords consisting of numbers and letters. The ROM for the user id’s stores 6 different user id’s which consists of the last 4 digits of each of our id’s and a created id for the guest user. Each user id and corresponding password is stored in the same address.

**RAM\_Pass Module**

The RAM\_Pass module is made up of the following input and output signals:

* + **Input Signal: address, clock, data, wren**
    - The signal address is a 6-bit input that tells the RAM the location of the number we currently want to display or change.
    - The clock input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency
    - The data signal is a 4-bit input that is only used when the read/write (wren) variable is set to high. Only then will the q output at the current address be changed.
    - The wren signal is a 1-bit input that determines whether to write or read to the current address on the RAM. When wren is set to high, the module will write whatever number is placed in data and save it to the current address. Whereas, when wren is set to low, the module will not change the data currently saved at the specific address.
  + **Output Signal: q**
    - The q signal is a 4-bit output that displays the number currently saved at a specific address of the RAM.

The module serves the purpose of storing a RAM file that is used by the player to change their password. The password is stored in the address corresponding to where it is stored in the ROM file for passwords, depending on which player is changing their password. The guest user cannot change their password.

**RAM for scores Module (Sayra)**

The following RAM for scores module is made up of the following input signals:

* + **Input Signal: rst, clk, score, timeout**
    - The input signal rst is connected directly to the 1-bit binary reset button.
    - The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency
    - The score signal is a 4-bit binary that the user has earned
    - The address signal is a 5-bit binary that lets the module know where to store the address.

The module only receives input signals because it serves the purpose of saving the scores received by a specified user to the RAM, so there is no need to output a signal. This module works in conjunction with the RAM\_Score module that is generated by Quartus in order to pass these values through this module to effectively save the scores.

**7seg Display Controller Module**

The following 7 segment display controller module is made up of the following input and output signals:

* + **Input Signal: rst, clk, displayState, levelNumber, speedNumber, numToFlash, noNumToFlash,GorH, score10s, score1s,timer10s, timer1s,display10s, display1s**
    - The input signal rst is connected directly to the 1-bit binary reset button.
    - The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency
    - The displayState input signal is a 4-bit binary number that is sent from the game controller that lets the display controller what state of the game they are in
    - The levelNumber input signal is a 2-bit binary number that gets set directly by the user
    - The speed number input signal is a 2-bit binary number that gets set directly by the user
    - The numToFlash input signal is a 4-bit binary that is triggered when the game starts displaying the random numbers of the sequence
    - The noNumToFlash input signal is a 1-bit binary number that let’s the display controller know if it is on stand by if there is no number to flash
    - The GorH input signal is a 1-bit binary connected to the high score controller to let the display controller to display a G or H depending on whether you are displaying a global high score or simply the high score of the current game
    - The score10s input signal is a 4-bit binary that display’s the 10’s value of the score
    - The score1s input signal is a 4-bit binary that display’s the 1’s value of the score
    - The timer10s input signal is a 4-bit binary that display’s the 10’s value of the timer
    - The timer1s input signal is a 4-bit binary that display’s the 1’s value of the timer
    - The display10s input signal is a 4-bit binary that display’s the 10’s value of the high score
    - The display1s input signal is a 4-bit binary that display’s the 1’s value of the high score
  + **Output Signal: disp0,disp1,disp2,disp3,disp4,disp5**
    - The disp0 - disp 5 output returns a 5-bit binary number, after it has been assigned it’s value based on the state it is in and this value will be taken through the 7 bit decoder to be displayed on its designated display

The module serves the purpose of controlling the 5 different display states: logged out, reconfig time, reconfig speed, reconfig speed, standby/gamestart, and game over. The module will display the corresponding word, random number, score, or timer value based on what state the game controller assigns to the display controller.

**Timer Module(Mohammed)**

The following \_ module is made up of the following input and output signals:

* + **Input Signal: rst, clk, ReConfig, enable, increment, decrement**
    - The input signal rst is connected directly to the 1-bit binary reset button.
    - The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency.
    - The ReConfig signal is used to reset the output.
    - The signal starts the timer.
    - The Increment pulse adds 10 seconds to the timer for every button press.
    - The decrement pulse subtracts 10 seconds from the timer for every button press.
  + **Output Signal: Tens\_BorrowUp, Ones\_NoBorrowDn, Ones\_Digit, Tens\_Digit**
    - The Tens\_BorrowUp signals to borrow from the next digit (not used in here as there is not hundreds place).
    - The Ones\_NoBorrowDn signal indicates when the timer is finished.
    - The Ones\_Digit will output the ones digit of the timer.
    - The Tens\_Digit will output the tens digit of the timer.

The module serves the purpose of adjustable count-down timer for the game. It contains two digit-timer modules and the time scaling module to simplify the top-level module. It allows the user to increment/decrement the tens place of the timer, and when the game starts it acts as a count-down timer.

**Time Scaling Module (Mohammed)**

The following Time Scaling module is made up of the following input and output signals:

* + **Input Signal: rst, clk, increment, decrement, Num** 
    - The input signal rst is connected directly to the 1-bit binary reset button.
    - The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency
    - The Increment pulse adds 10 seconds to the timer for every button press.
    - The decrement pulse subtracts 10 seconds from the timer for every button press.
  + **Output Signal: Num**
    - Num is a 4-bit binary number between 0 and 9. Goes to the tens place of the timer module. It will be increment or decremented by one using the input buttons.

The module serves the purpose of incrementing or decrementing the tens place of the timer depending on the user's input.

**RNG LFSR Module**

The following rngLFSR module is made up of the following input and output signals:

* + **Input Signal: rst, clk, rngFetchSignal**
    - The input signal rst is connected directly to the 1-bit binary reset button.
    - The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency
    - The rngFetchSignal is fed from the game controller. When the game is active, pressing button 1 sends several pulses to the rng module to produce several RNG values to be displayed.
  + **Output Signal: rngDigit**
    - The rngDigit is a 4-bit value that is passed to an upper level module for the game controller. This value is both displayed and saved to RAM for the sequence that will later be checked when the user inputs the sequence from memory.

This module is a submodule for the game controller. Its primary function is to generate an RNG value for each digit of the Sequence being displayed.

**Digit Timer Module**

The following Digit Timer module is made up of the following input and output signals:

* + **Input Signal: borrowDown,noBorrowUp,reconfig, clk,rst, Num**
    - Num is a 4-bit input from the Time Scaling module. It is the number the timer will start counting down from.
    - The borrowDown signal is a 1-bit signal that is connected to the access controller and the 10s digit timer to indicate whether there is more to count down.
    - The no borrowUp signal is a 1-bit signal connected to the access controller that indicates that time is up and there is nothing left to borrow
    - The reconfig signal is a 1-bit signal that will reset the values of the digit timer back to 9 based on user input
    - The input signal rst is connected directly to the 1-bit binary reset button.
    - The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency
  + **Output Signal: noBorrowDown,borrowUp,digit**
    - The noBorrowDown signal is a 1-bit signal connected from the 10s place digit timer that indicates there is nothing left to borrow.
    - The borrowUp signal is a 1-bit signal that is connected to the 10s place digit timer to indicate there is more to be borrowed.
    - The digit signal is a 4-bit signal that represents the values 9 to 0 when the timer is counting down

The module serves the purpose of counting down from 9 to 0, to create a timer for the user to play their game. In our case, we created a timer of 99 seconds so that required 2 instances of the digit timer, to be connected to each other in order to accurately count down.

**Button Shaper Module**

The following Button Shaper module is made up of the following input and output signals:

**Input Signals: B\_in, rst, clk**

* The input signal B\_in receives the 1-bit binary number given by the button connected when it is pressed or unpressed.
  + B\_in = 1’b0 : signifies the button being pressed
  + B\_in = 1’b1 : signifies the button being unpressed
* The input signal rst is connected directly to the 1 -bit binary reset button.
* The clk input signal is connected to the clock signal of the FPGA board that sets the 50 MHz frequency.

**Output Signal: B\_out**

* The output signal B\_out returns a 1-bit binary number that is able to load a single clock cycle into the input of the access controller.

The following Button Shaper module serves the role of connecting the button’s output to the access controller. The module is the bridge between the two because the button shaper is able to load a single clock cycle into the load of the access controller.

**5-bit to 7-bit Decoder Module (sayra)**

The following Decoder module is made up of the following input and output signals:

**Input Signal: Decoder\_in**

* The decoder input receives a 5-bit binary number.

**Output Signal: Decoder\_out**

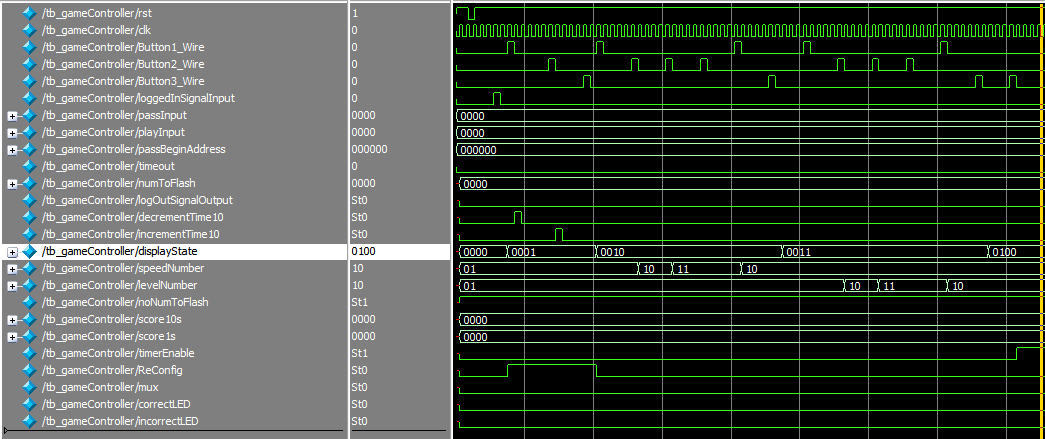
* The decoder output returns a 7-bit binary number.

The module serves the important role of decoding a player’s 5-bit binary number from their toggle switches to a 7-bit binary number so that it can be displayed on the seven-segment display. It is also used to decode the 5-bit sum of both of the player’s values so that it can be displayed on its own seven-segment display. The table below shows all the conversation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **5 bit to 7 bit Decoder** | | | | | |
| **Decimal** | **Binary** | **Hex** | **Decimal** | **Binary** | **Conversion** |
| 0 | 00000 | 0 | 16 | 10000 | G |
| 1 | 00001 | 1 | 17 | 10001 | H |
| 2 | 00010 | 2 | 18 | 10010 | I |
| 3 | 00011 | 3 | 19 | 10011 | L |
| 4 | 00100 | 4 | 20 | 10100 | M |
| 5 | 00101 | 5 | 21 | 10101 | P |
| 6 | 00110 | 6 | 22 | 10110 | S |
| 7 | 00111 | 7 | 23 | 10111 | T |
| 8 | 01000 | 8 | 24 | 11000 | V |
| 9 | 01001 | 9 | 25 | 11001 | - |
| 10 | 01010 | A | 26 | 11010 |  |
| 11 | 01011 | B | 27 | 11011 |  |
| 12 | 01100 | C | 28 | 11100 |  |
| 13 | 01101 | D | 29 | 11101 |  |
| 14 | 01110 | E | 30 | 11110 |  |
| 15 | 01111 | F | 31 | 11111 |  |

**9.4 Simulation Results**

**Game Controller**

****

*Figure 4.Game Controller Testbench Waveform*

The intention of this simulation is to show the game controller actively reconfiguring the game settings. Per the design of this module, the module should be in the “Logged Out” State until the user successfully logs in. The user logging in can be simulated by pulsing the input “loggedInSignalInput”. One clock cycle after this pulse, the state enters the Reconfiguring Timer State (indicated by displayState==0001).

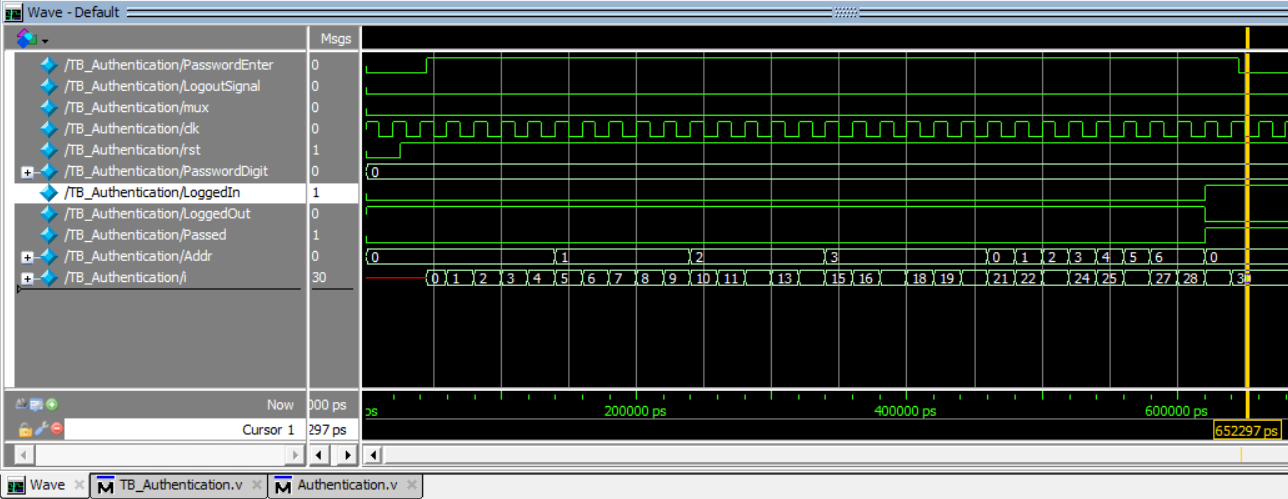
In the Reconfiguring Timer State, the user will be able to decrement or increment the time by 10 seconds using Button 1 and 2 respectively. Following a pulse of Button1\_Wire is a pulse of the signal “decrementTime10”. Similarly, pressing Button 2 pulses a signal for “incrementTime10”. When the user finishes reconfiguring the time, they can press Button 3 to enter the Reconfigure Speed State.

In the Reconfiguring Speed State (displayState==0010), the speed of the game (speedNumber) is initialized at 01, which is the minimum. To ensure that the value of this cannot fall below 1, Button 1 is pressed and the value of speedNumber does not change. This value is then incremented up to the max value of 3 using Button 2. And to show off the decrementing value of the Speed Number, Button 2 is pressed once more, resulting in Speed being set to 2.

Once this value for the Speed is decided, Button 3 is pressed once more to lock it in. The state moves onto Reconfiguring the Level of the Game (displaySate==0011). The same decrementation and incrementation is repeated for the level, and then locked in.

In the next state (displayState==0100), the game is in the Standby. When the user presses the third button once more, the game begins as indicated by the timerEnable signal going high.

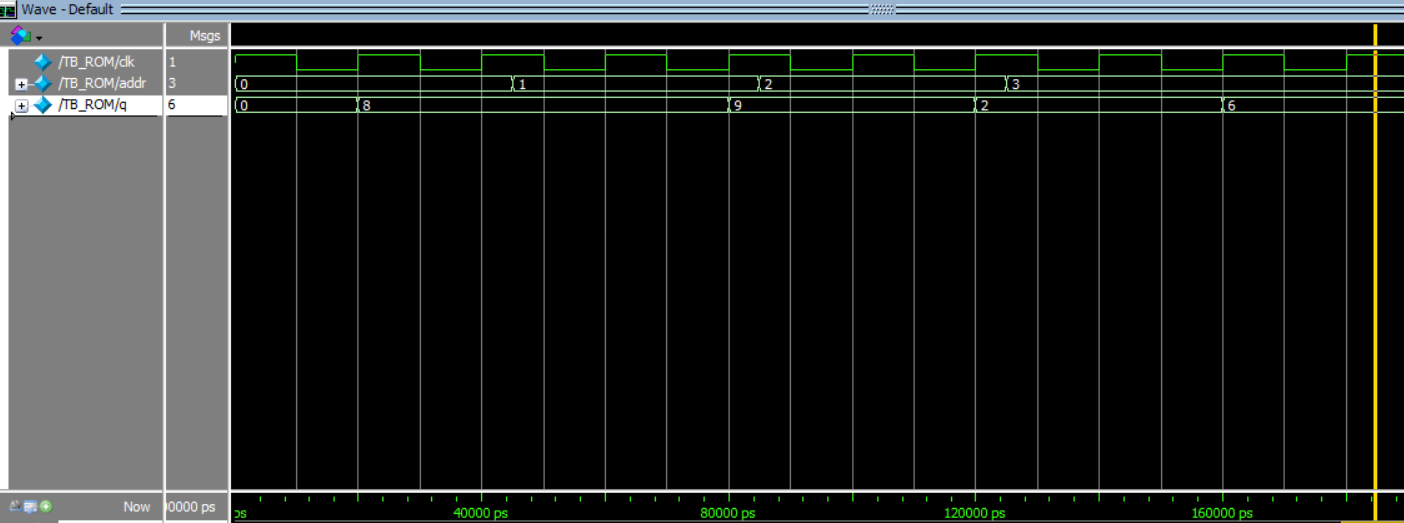
**Authentication Module**

****

*Figure 5. Authentication Testbench Waveform*

The intention of this test bench is to showcase the login feature. We had an account with a UserID of 0000 and a Pass of 000000. After entering all of the digits to account for the UserID and Pass, we are resulted with a change in signals from the LoggedIn and LoggedOut outputs. LoggedIn switches from 0 to 1, while LoggedOut switches from 1 to 0. This proves that our Authentication module is performing correctly.

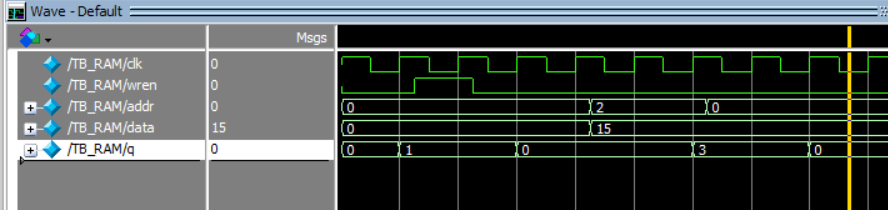
**ROM\_Pass & ROM\_UserID**

****

*Figure 6. ROM for Password and UserID Testbench Waveform*

The intention of the ROM for the passwords and user id is that it will store each of our passwords and user ids. The passwords will be 6 bit values consisting of numbers and letters. The user ids will be 4 bit values that are each of our last 4 digits of our ids. These ROMs also store the password and id of the guest user. The id being FFFF and password being FFFFFF. In this test bench, we show the first user id as seen in our ROM\_UserID.hex file (8926). This proves that our ROM file is successfully being read into the system.

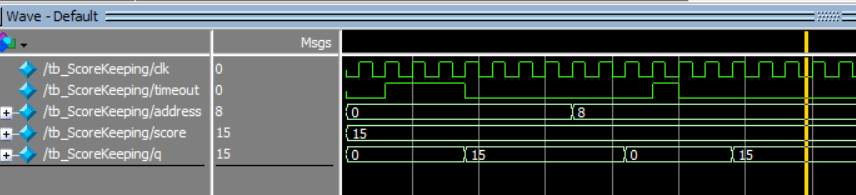
**RAM\_Pass**

****

*Figure 7. RAM for Password Testbench Waveform*

The intention of the RAM for the passwords is that it will store the original passwords for each of the 6 users, including the guest user. This will allow the players to change their password. When a player changes their password, it will be changed here. The password will be stored corresponding to which player is playing, by looking at the corresponding addresses. In this test bench, we show the password of the first user (originally 123456) as seen in our RAM\_Pass.hex file. We start by setting data to 0, address to 0, read/write (wren) to write (1), and change the first number of our original passcode (1) to data. We then set read/write back to read (0) and data to 15. We change the address to 2, which will give us the 3rd number in our passcode (3). Lastly we change address to 0, going back to our first number in the passcode that we had previously changed. It should still display 0, which it does, even though we still have data as 15.

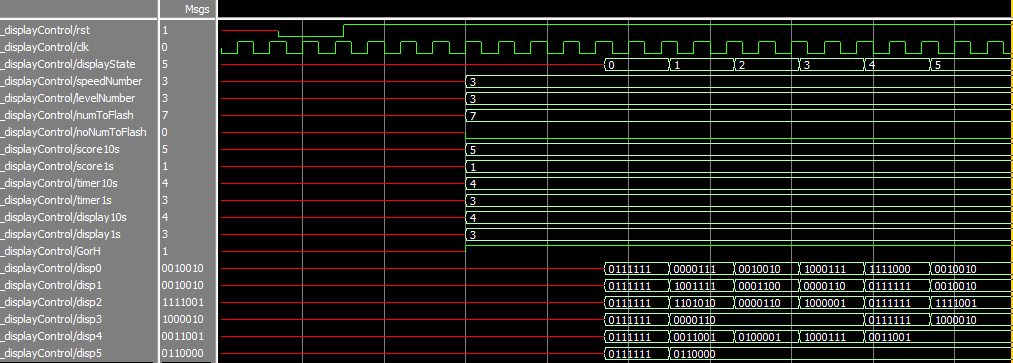
**RAM for scores**

****

*Figure 8. RAM for Scores Testbench Waveform*

The intention of the RAM for score module is that it will save the user’s high score to the RAM, so that it can be compared for different gameplay and update the high score as the user plays the game. The following testbench was created to show that as the timeout occurs the high score will be displayed for a given address. The following test bench shows that the high score is displayed for 2 different addresses.

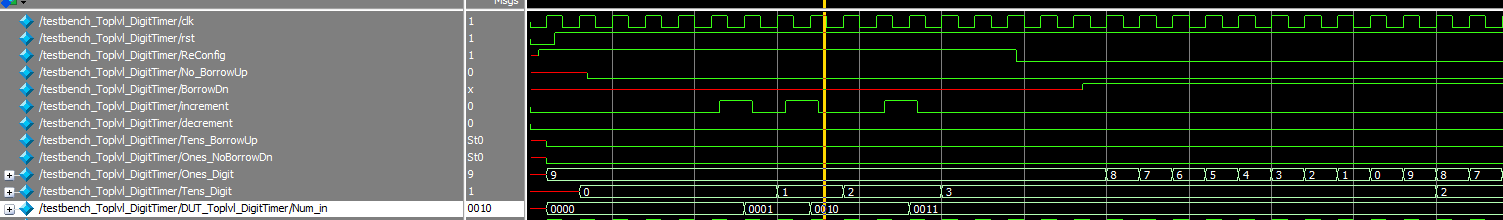
**7seg Display Control**



*Figure 9. 7 segment Display Controller Testbench Waveform*

The intention of the 7 segment Display Controller is to display a specified word, value, or score based on the state that the game is in. The following testbench shows the value of each display screen for the given state, which has already been decoded to a 7-bit binary number.

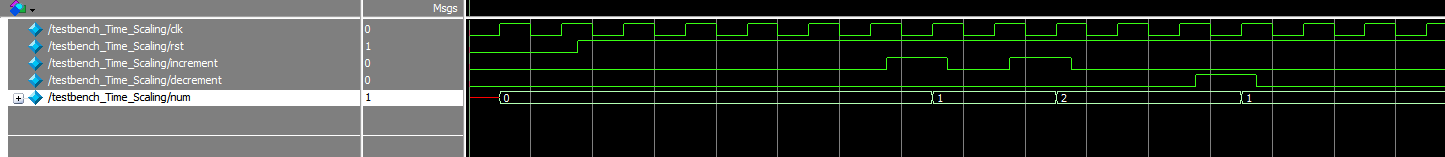
**Timer**



*Figure 10. Timer Testbench Waveform*

The intention of the timer module is work as a countdown timer. Before it starts the user will have the option to increment/decrement the tens place. As seen in figure X above, the tens place is incremented three times and then the ReConfig signal is lowered to stop updating the timer. Then the timer starts counting down from the chosen value once the BorrowDn pulses start.

**Time Scaling Module (Mohammed)**

*Figure 11. Time Scaling Module Testbench Waveform*

The intention of the Time scaling module is to increment/decrement the digit timer this module is connected to. As seen in the figure above, the testbench shows the increments/decrement pulse a few times. And the resulting number is shown in the num output.

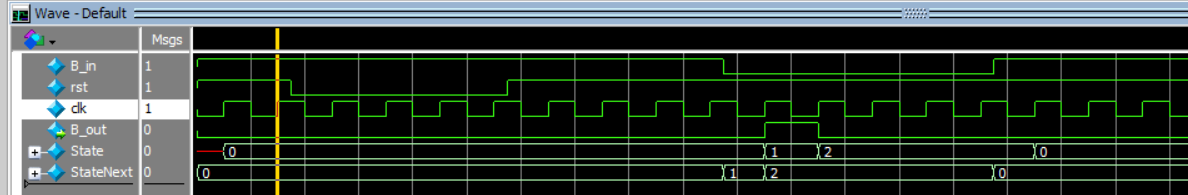
**Digit Timer**

****

*Figure 12. Digit Timer Testbench Waveform*

The intention of the digit timer is that it will count down from a given number to indicate when a player’s term is up. In our case, we are creating a two-digit timer that starts at 99 and counts down. This two-digit timer is achieved by connecting two Digit timer modules together. In our waveform above we represent the individual digit count down, so a user can see the number will count down from 9 to 0 and once it is connected to a second digit timer module in the top-level it will count down from 99 to 0.

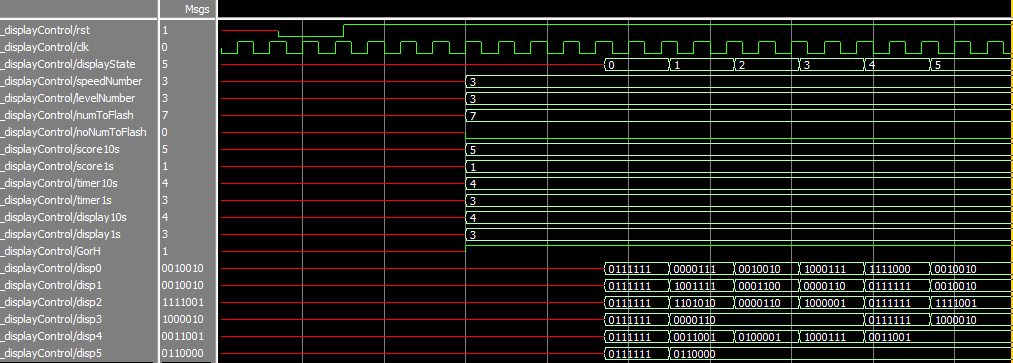
**Button Shaper Module**



*Figure 13. Button Shaper Testbench Waveform*

The button shaper testbench module served the purpose of verifying the button shaper’s actions based on its clock cycle. This test bench was different than the previous test benches we have worked with because of the clock cycles. Before you create your testbench you should understand the function of your module so that you are able to verify your state machine design with the waveforms. In our case, the clock wave will be at a constant pace so that you can observe the reset, input, and output waveforms.

**5-bit to 7-bit Decoder Module**



*Figure 14. Top Level display Controller Testbench Waveform*

The decoder module served the purpose of converting a 5 -bit binary number into a 7 -bit binary number. The testbench shows the result of a couple of the conversions for display 0 - display 5 based on the state they are in. This shows that the digit controller is able to accurately decode the value passed in by the game controller. The waveform above shows the change in display values based on the state they are in.

**10.5 FPGA Board Testing Resul**t**s**

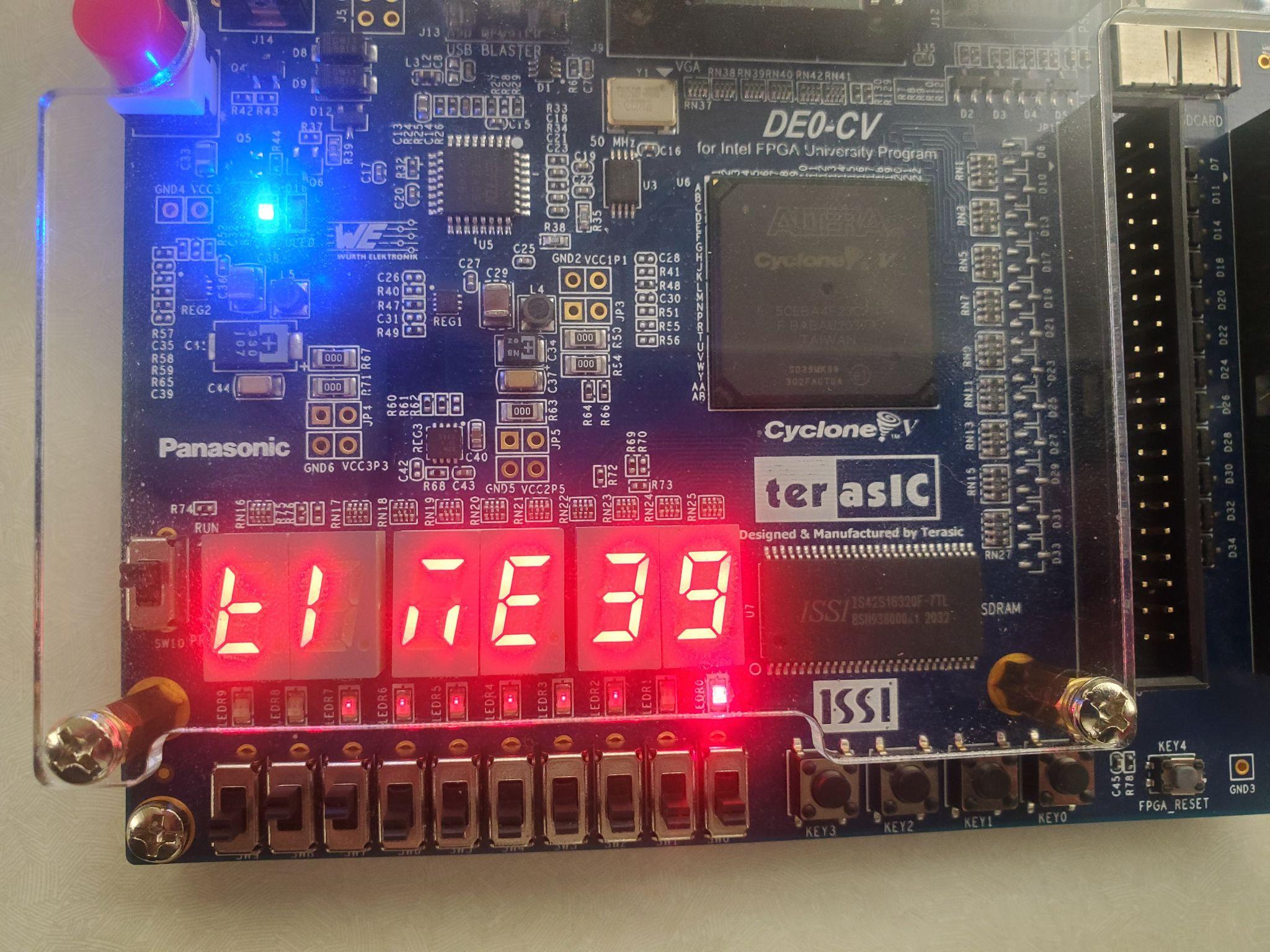
The functions that we want to present are

**The Logged out state displays (-) for each display**



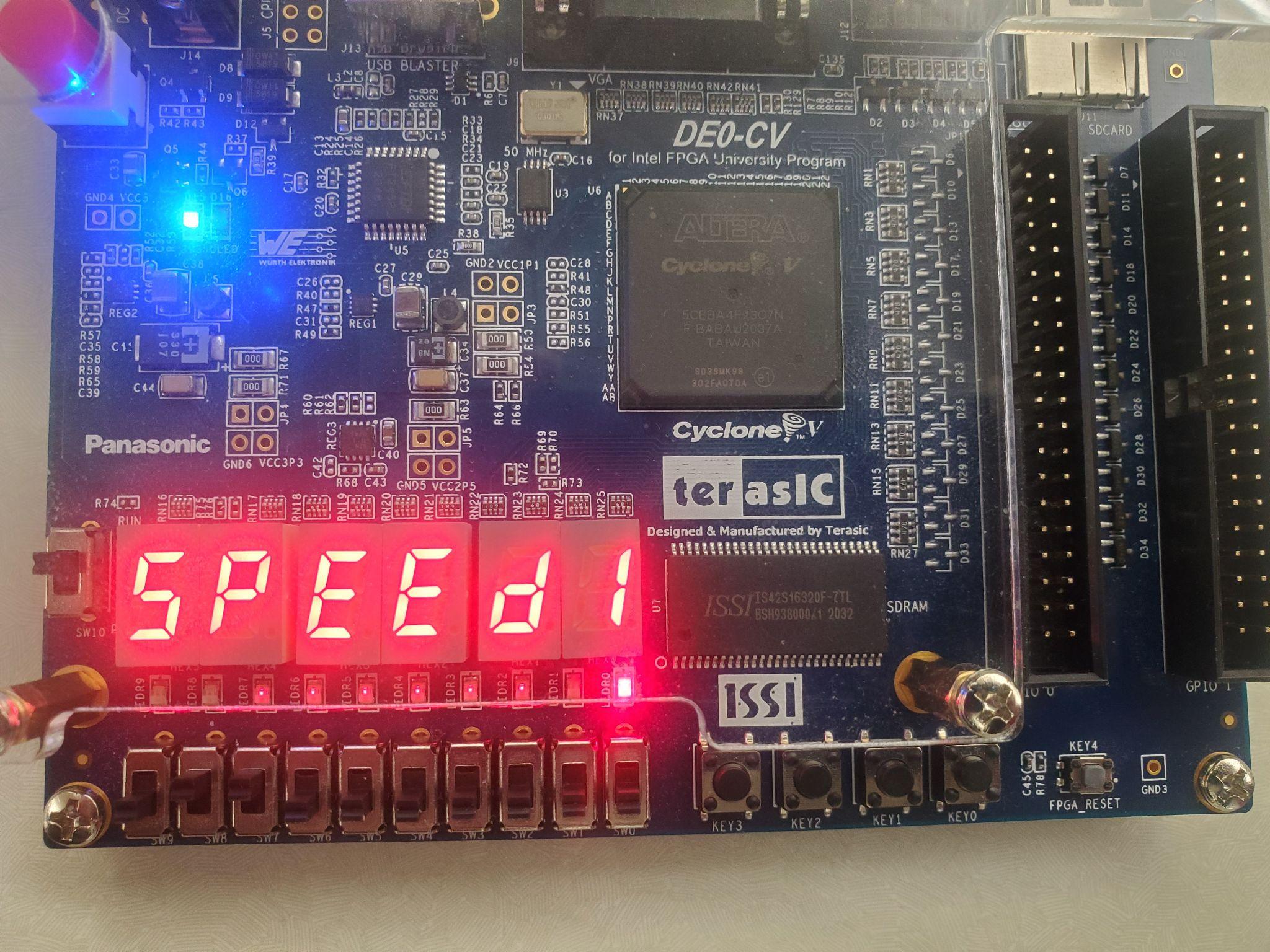
*Figure 15. Logged Out - Values are not Displayed on the FPGA Board*

**Reconfigure State - Configure Timer: display the word TIME**



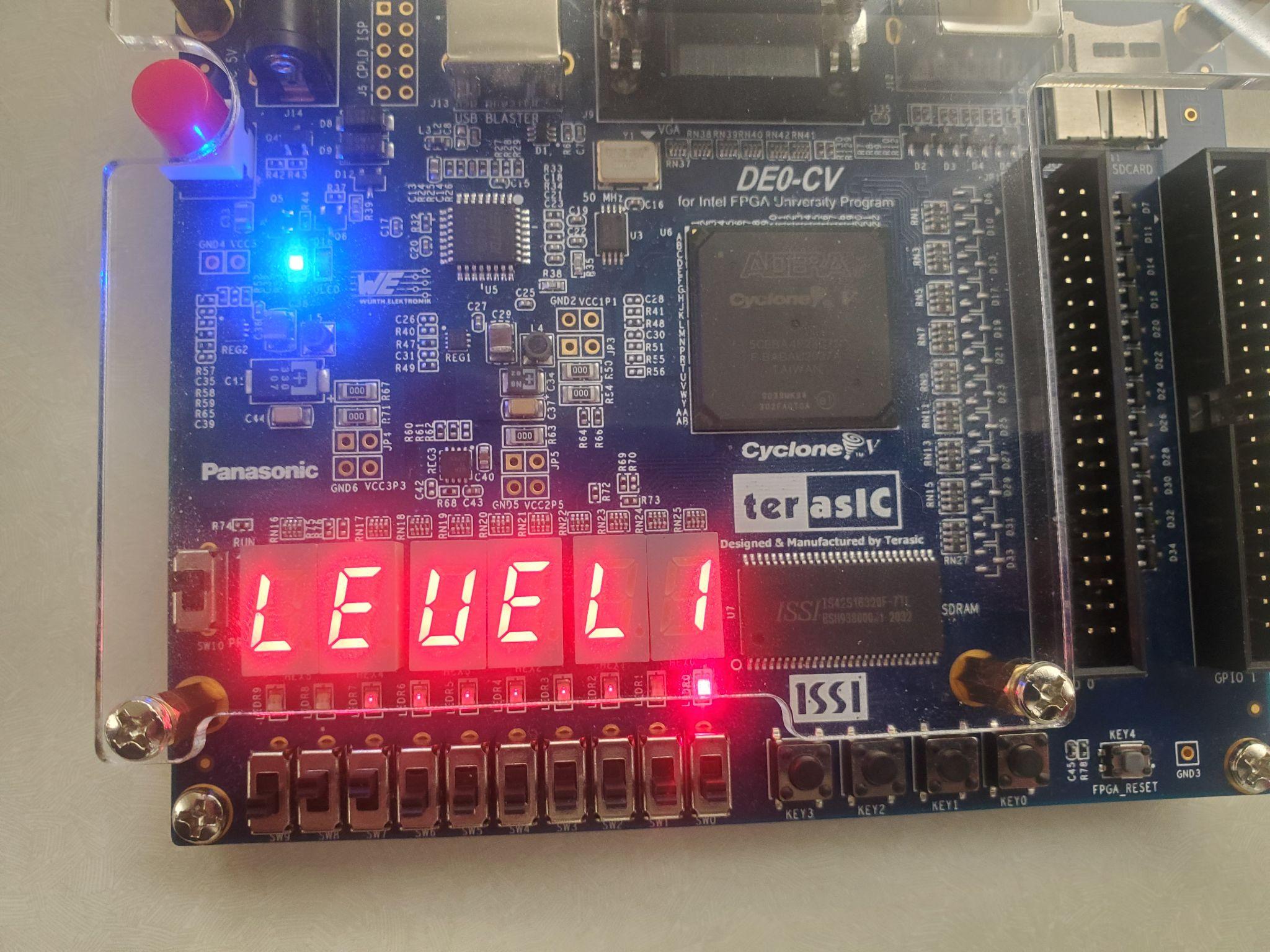
*Figure 16. Reconfigure State - display the word TIME on the FPGA Board*

**Reconfigure State - Configure Speed of Sequence: display the word SPEED**



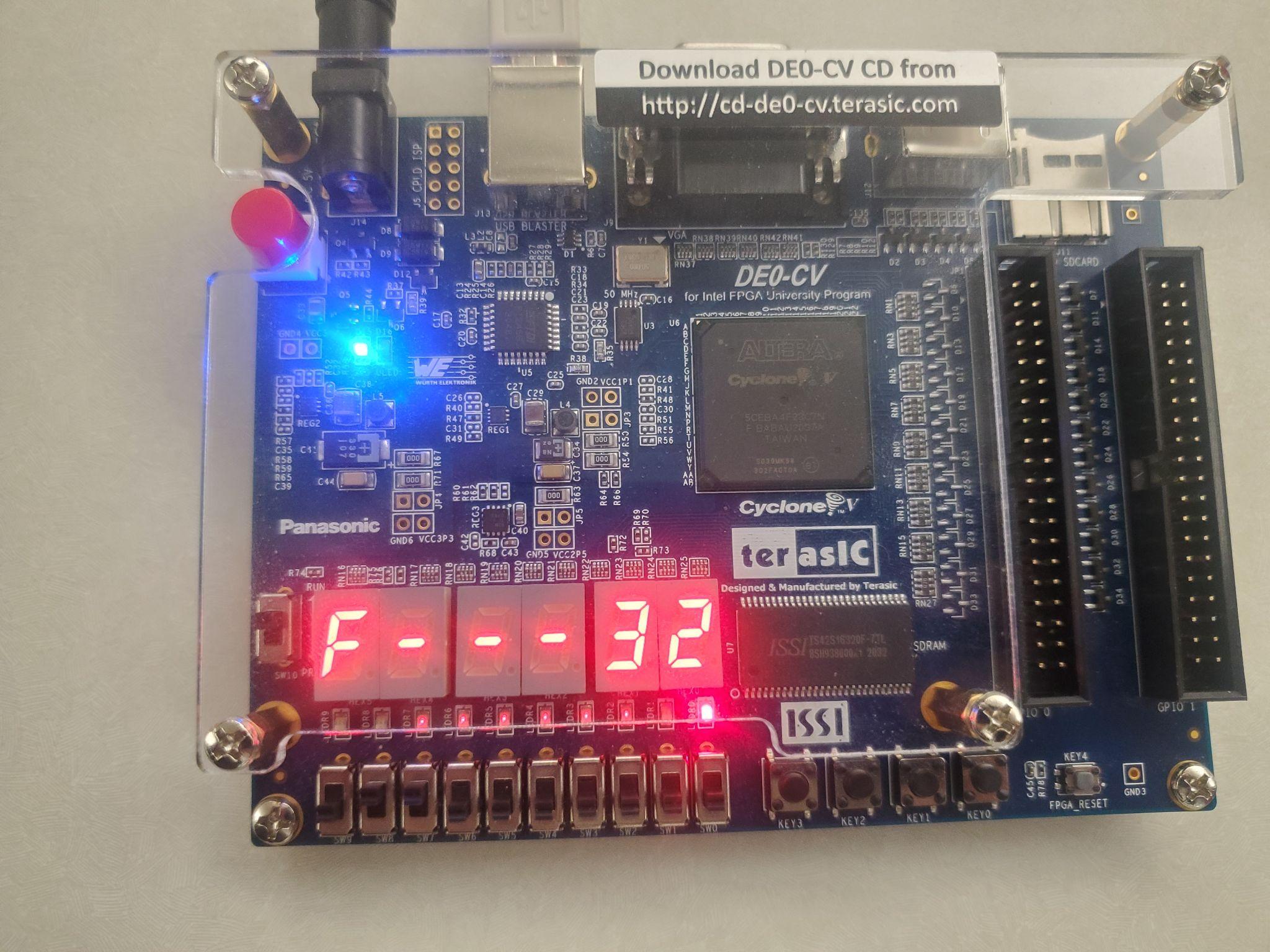
*Figure 17. Reconfigure State - display the word SPEED on the FPGA Board*

**Reconfigure State - Configure Level: display the word LEVEL**



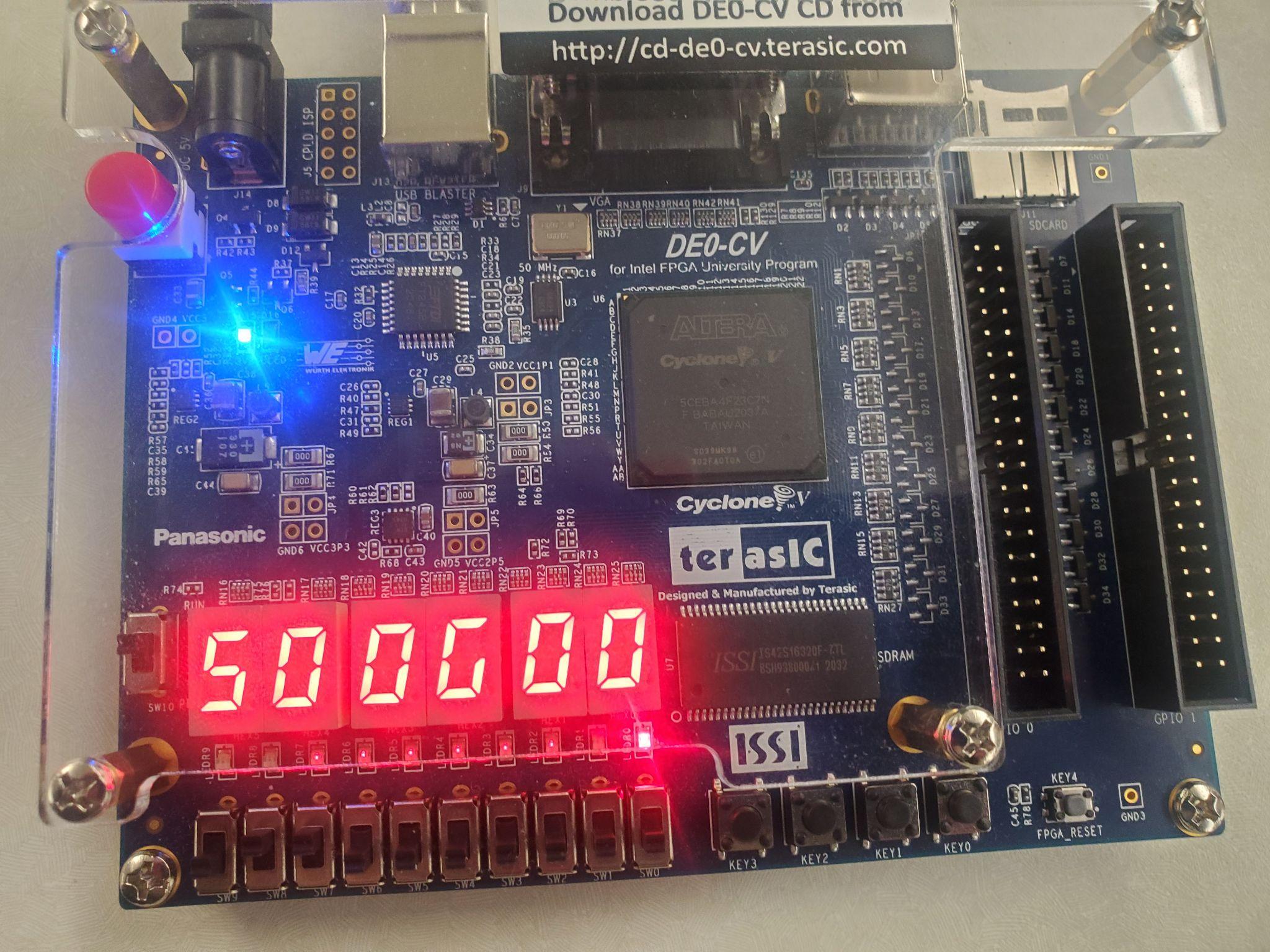
*Figure 18. Reconfigure State - display the word LEVEL on the FPGA Board*

**Game Start - Random number (left) displayed and timer countdown (right)**



*Figure 19. Game Start State - Random number and timer countdown displayed*

**Game Over State - Display Score (left) and Global High Score (right)**



*Figure 20. Game Over State - Display Score and Global High Score displayed*

**9.6 Video Demo**

Below you will find a link to 4 demos that explain the functionalities of the board and the instructions for the Memory Game.

This video explains how to login:

[LINK](https://drive.google.com/file/d/183oYs6wvRJHrXFqsl-pv1XZC62LYKjNA/view?usp=sharing)

This video explains how to reconfigure the timer, speed and level of the game once the player has been logged in:

[LINK](https://drive.google.com/file/d/18B5ly4UPYyhyBqT7nXJk-eYuHca4TXwH/view?usp=sharing)

This video explains how the memory game works after the timer,speed and level has been configured:

[LINK](https://drive.google.com/file/d/18DP9SJ4nstJToJgkWfx88LBKvURbdzt8/view?usp=sharing)

This video explains how what happenes when the game is over:

[LINK](https://drive.google.com/file/d/18E98HCfjjrFFWX7WULWNlF4BO7RqvTHp/view?usp=sharing)

This video explains a complete demonstration of all the features of the game:

[LINK](https://drive.google.com/file/d/1LbjBf5zfO78zabMbMA91yem_MZ_SKtCv/view?usp=sharing)

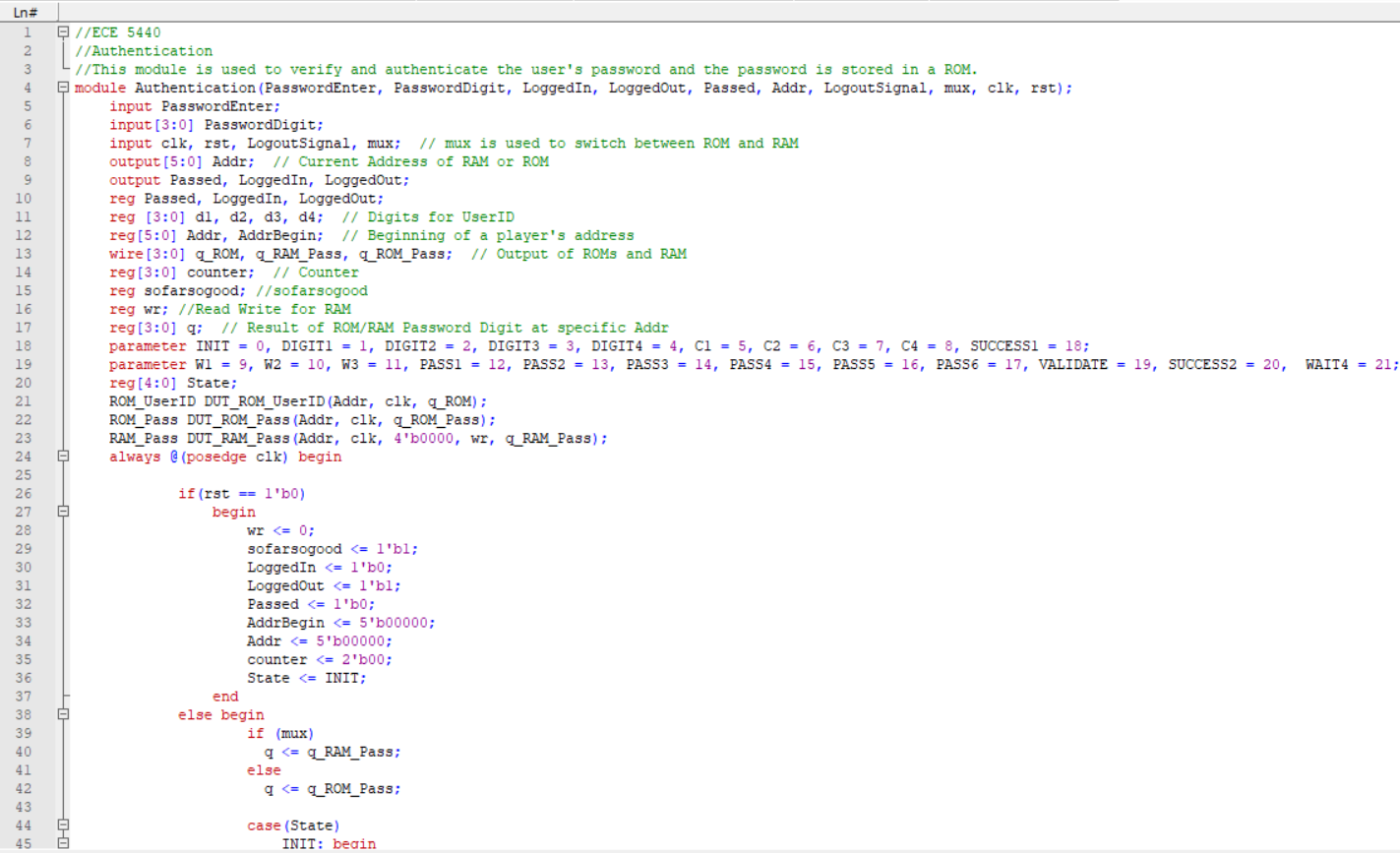
**9.7 Conclusion**

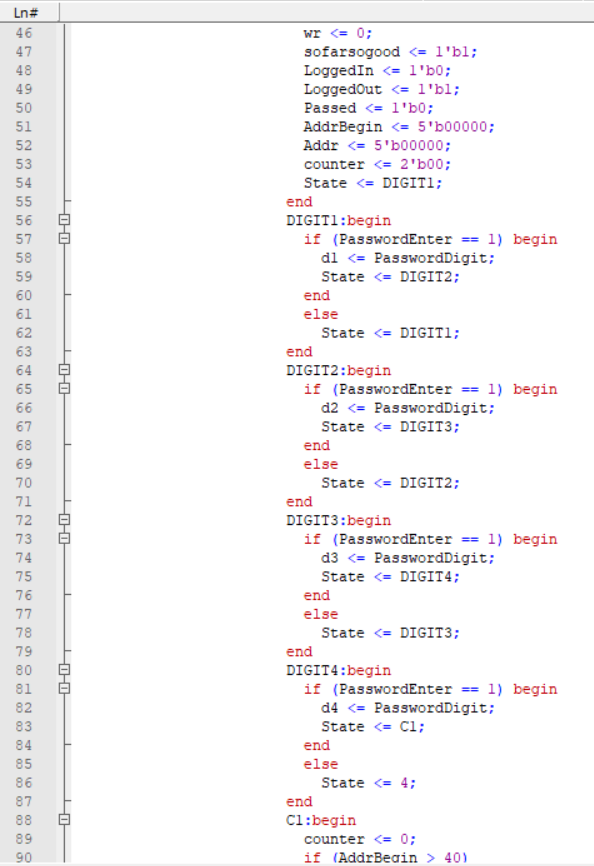
The memory game system is made up of …

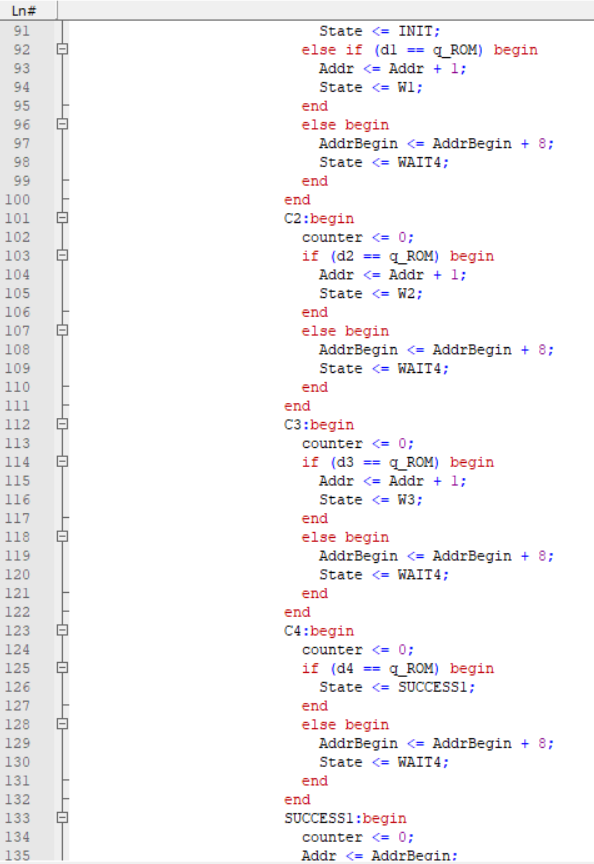
**9.8 Appendix**

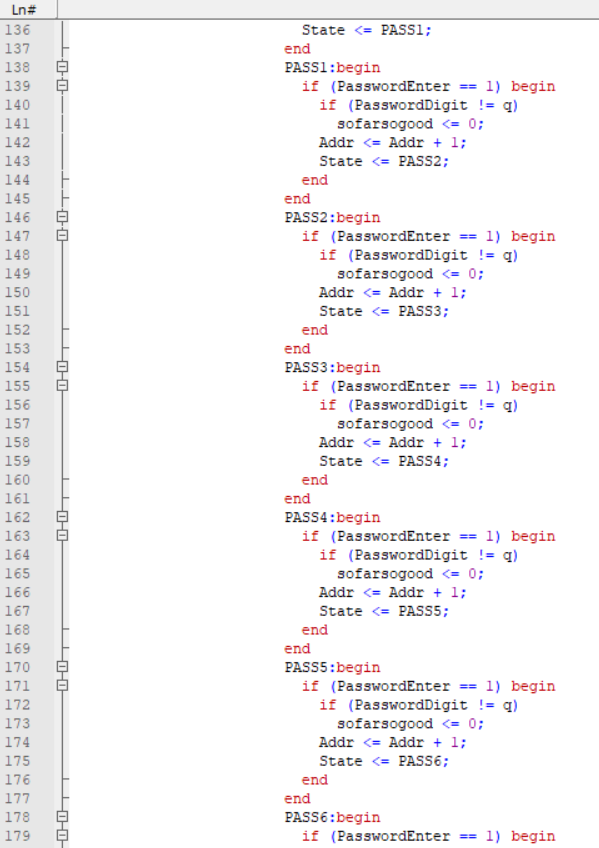
We want to present all the modules and their respective test benches that were used to create the memory game.

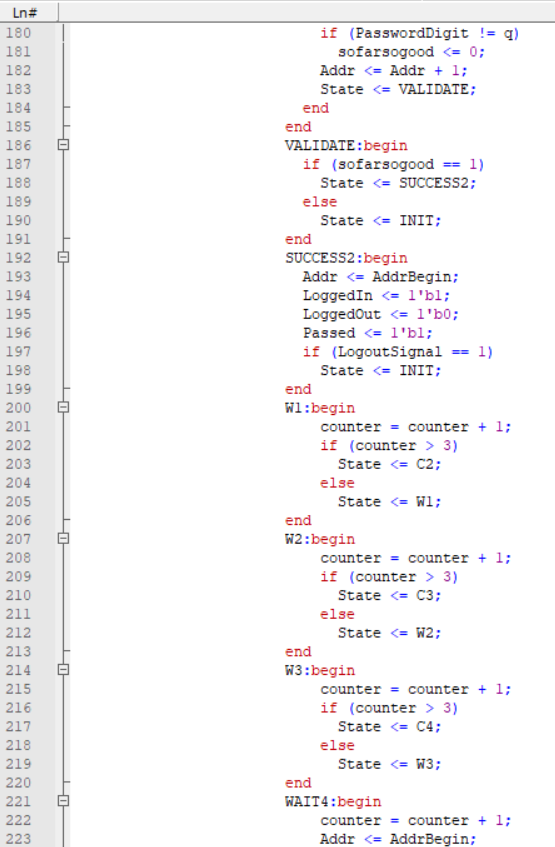
**Authentication Module**

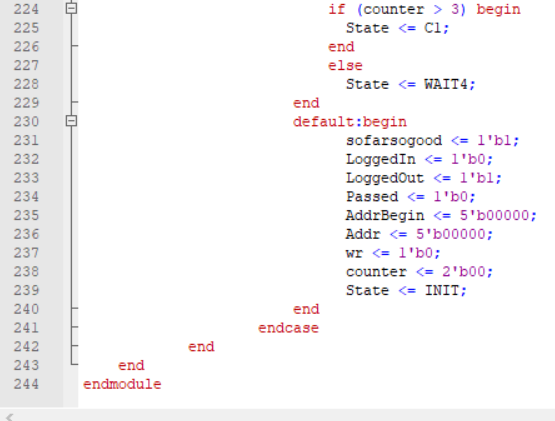
****

****

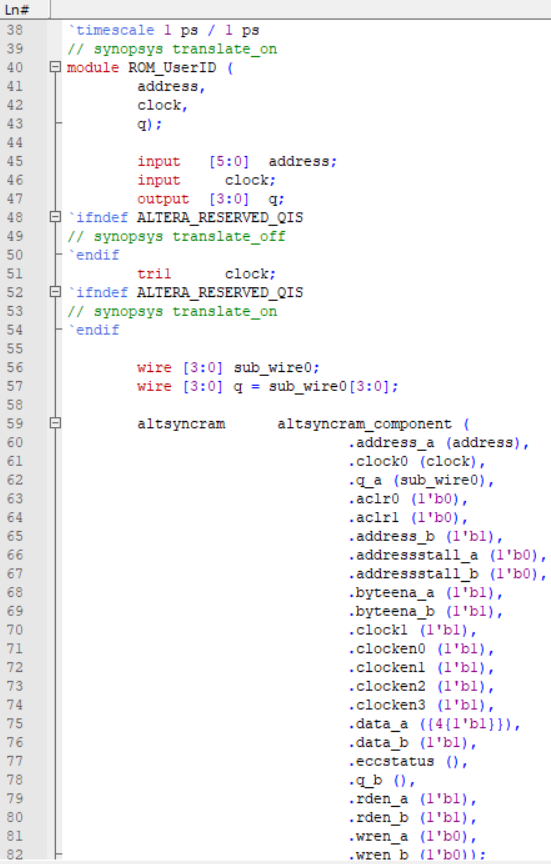
****

****

****

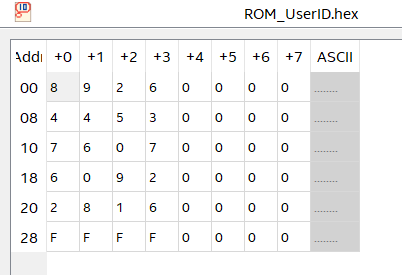
****

**ROM\_UserID Module**

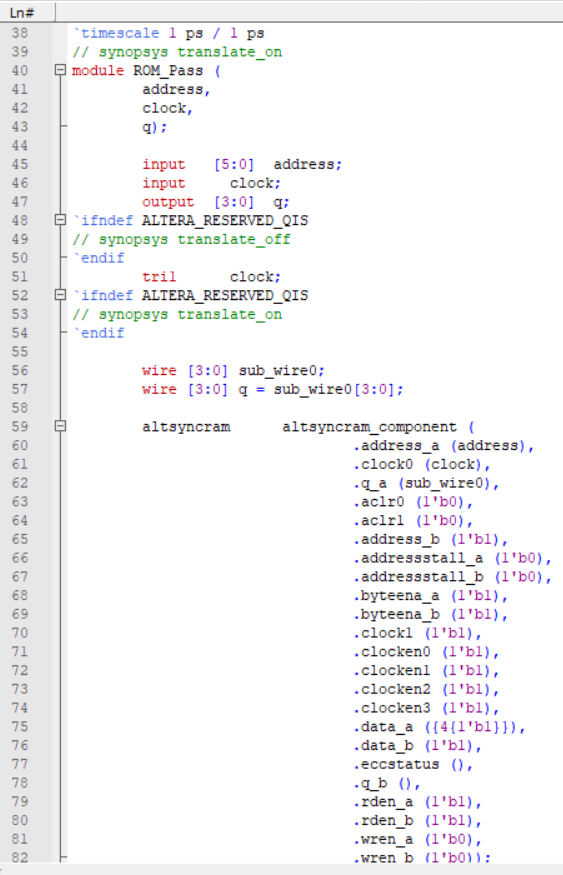
****

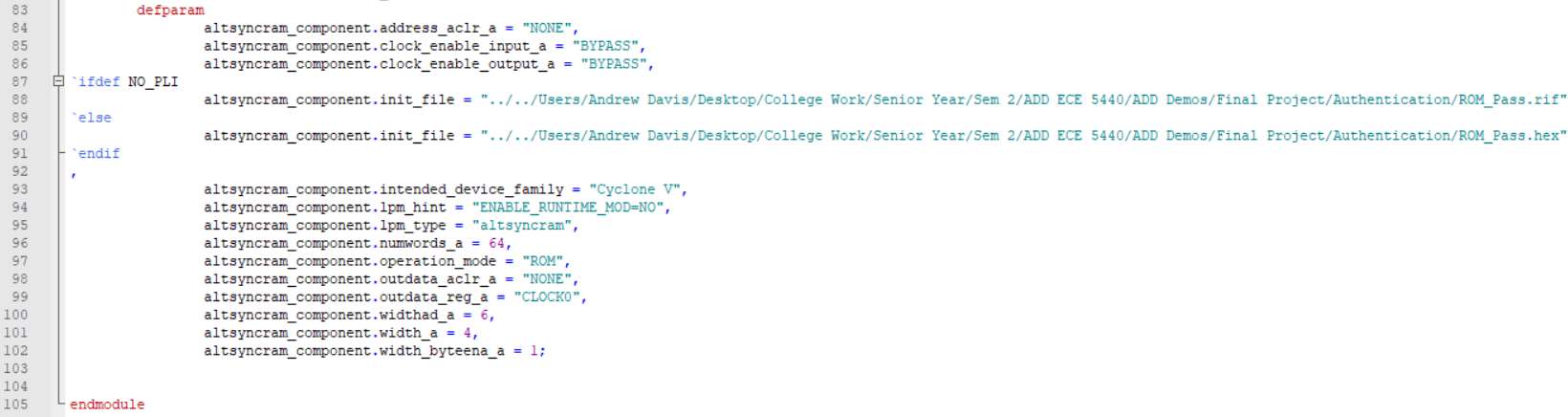
****

**ROM\_UserID.hex**

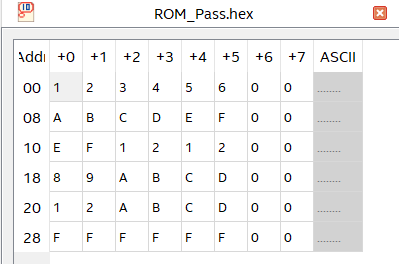
****

**ROM\_Pass Module**

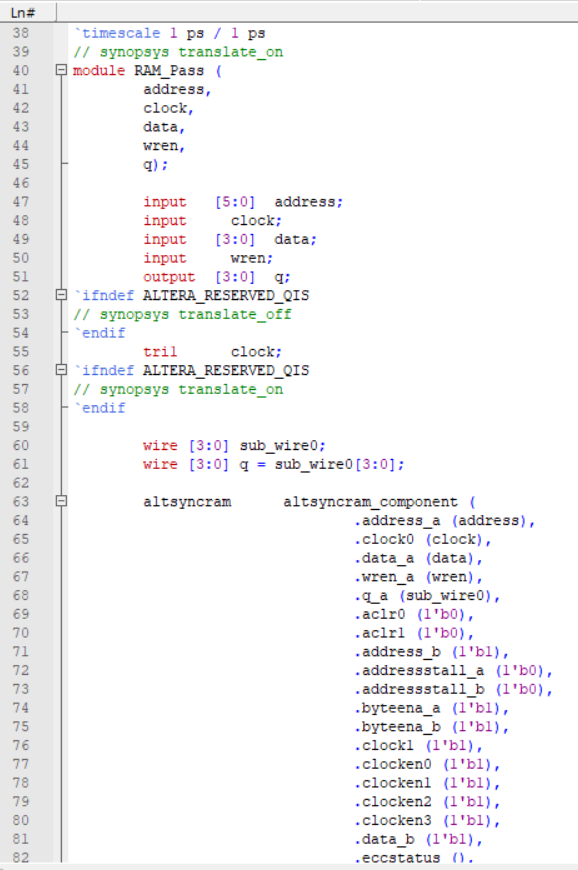
****

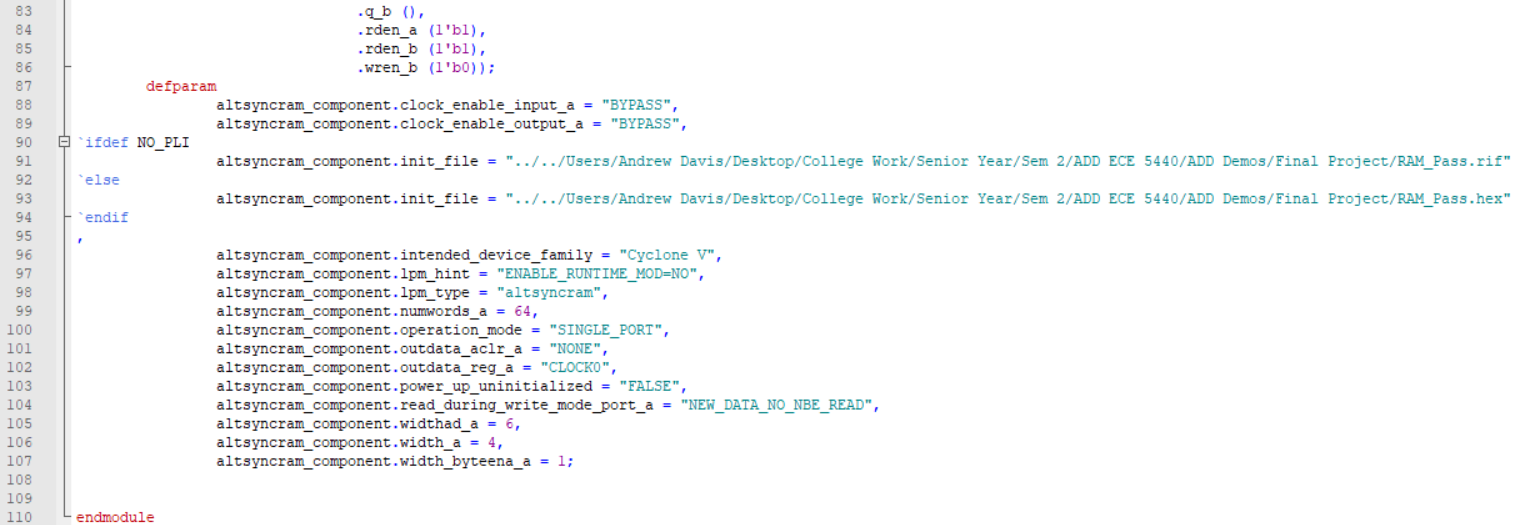
****

**ROM\_Pass.hex**

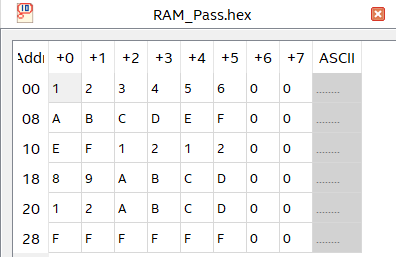
****

**RAM\_Pass Module**

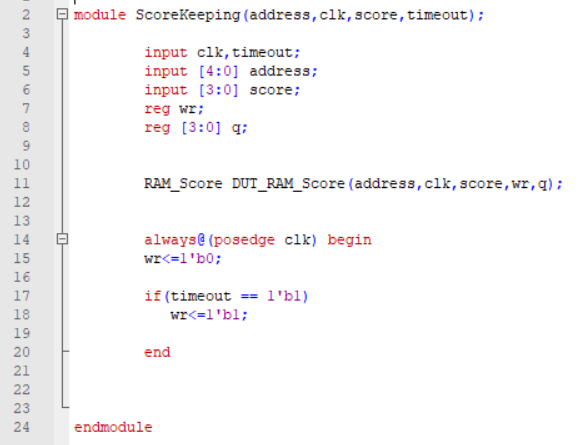
****

****

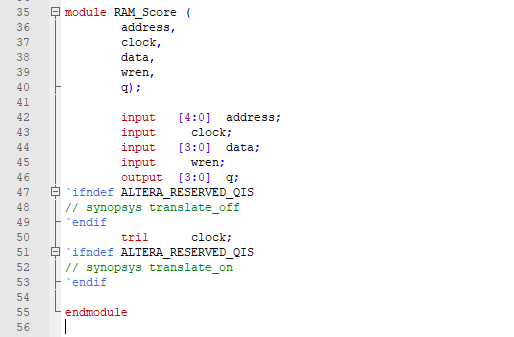
**RAM\_Pass.hex**

****

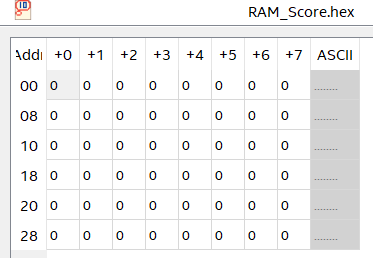
**ScoreKeeping Module**

****

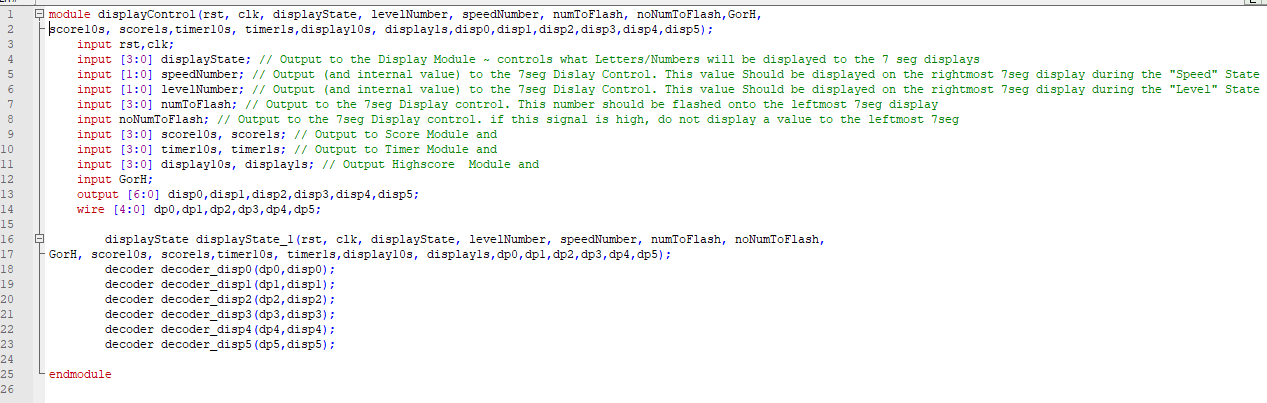
**RAM\_Score Module**

****

**RAM\_Score.hex**

****

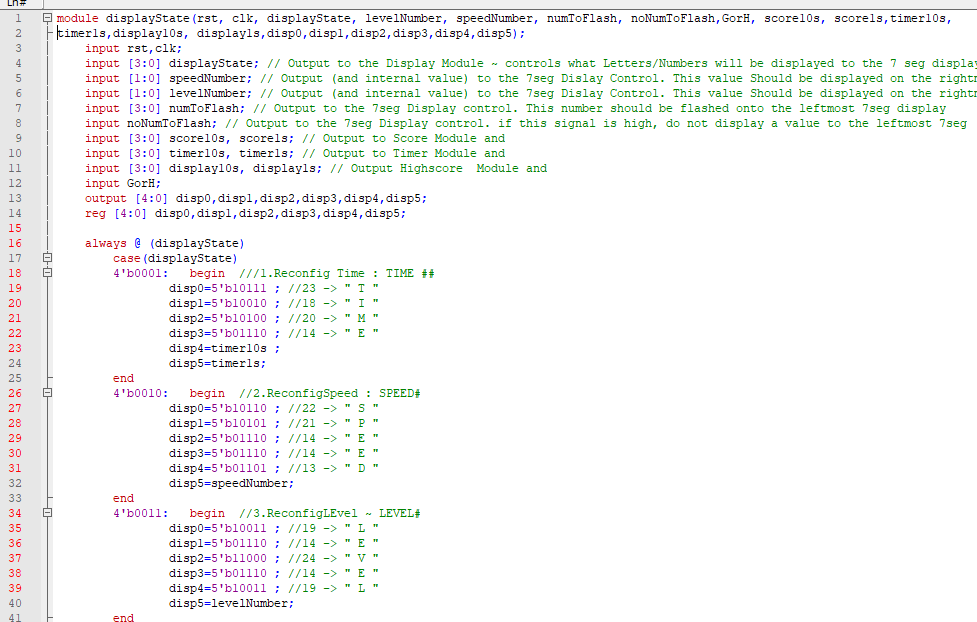
**Display Controller: Top-Level Module**

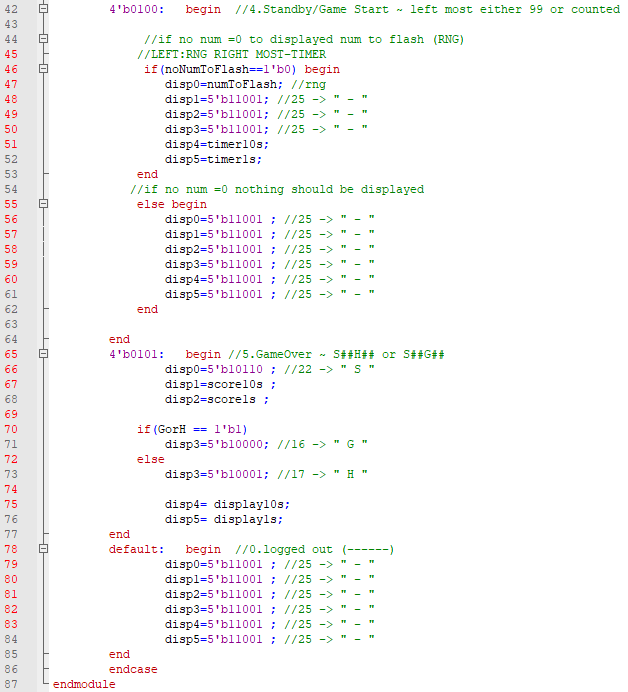
****

**Display Controller: 5 bit to 7 bit Decoder**

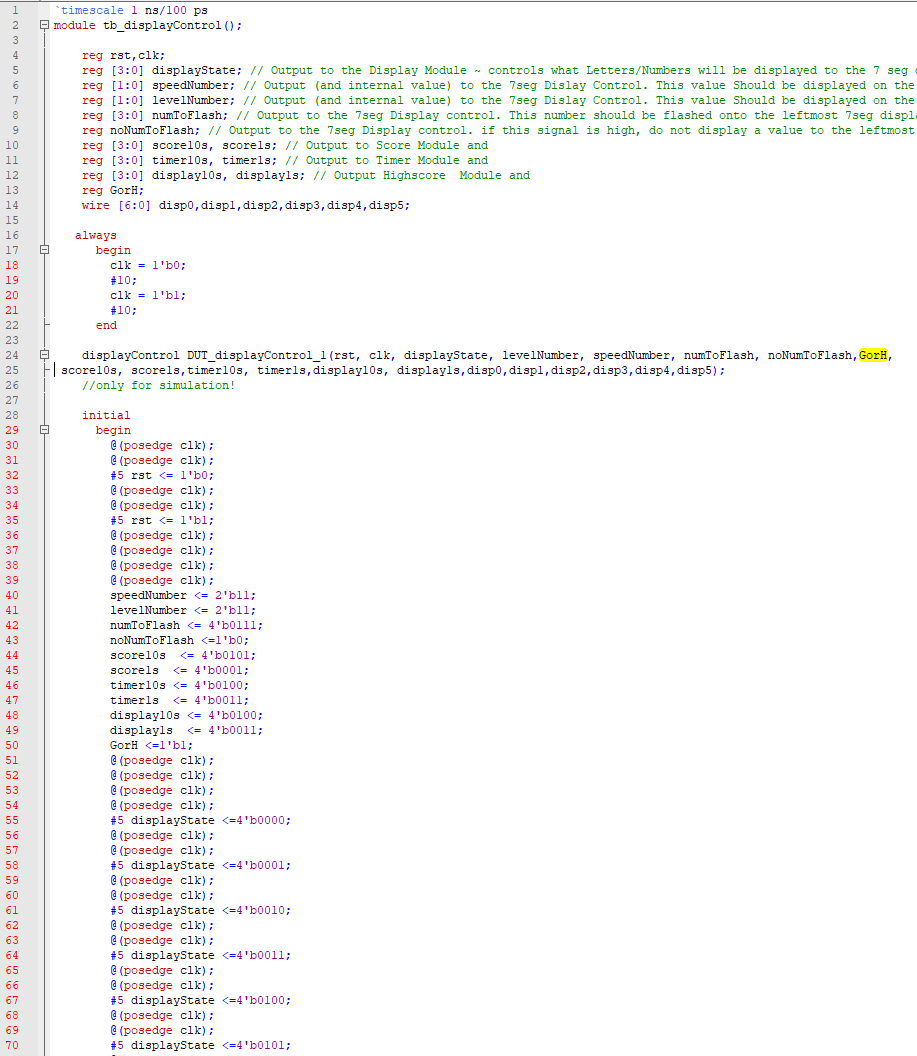
****

**Display Controller: State Controller**

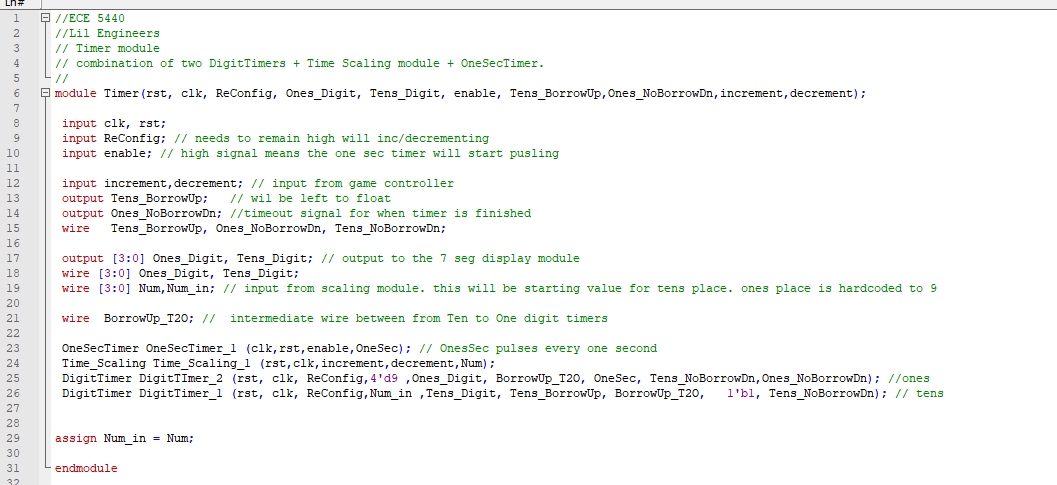
****

****

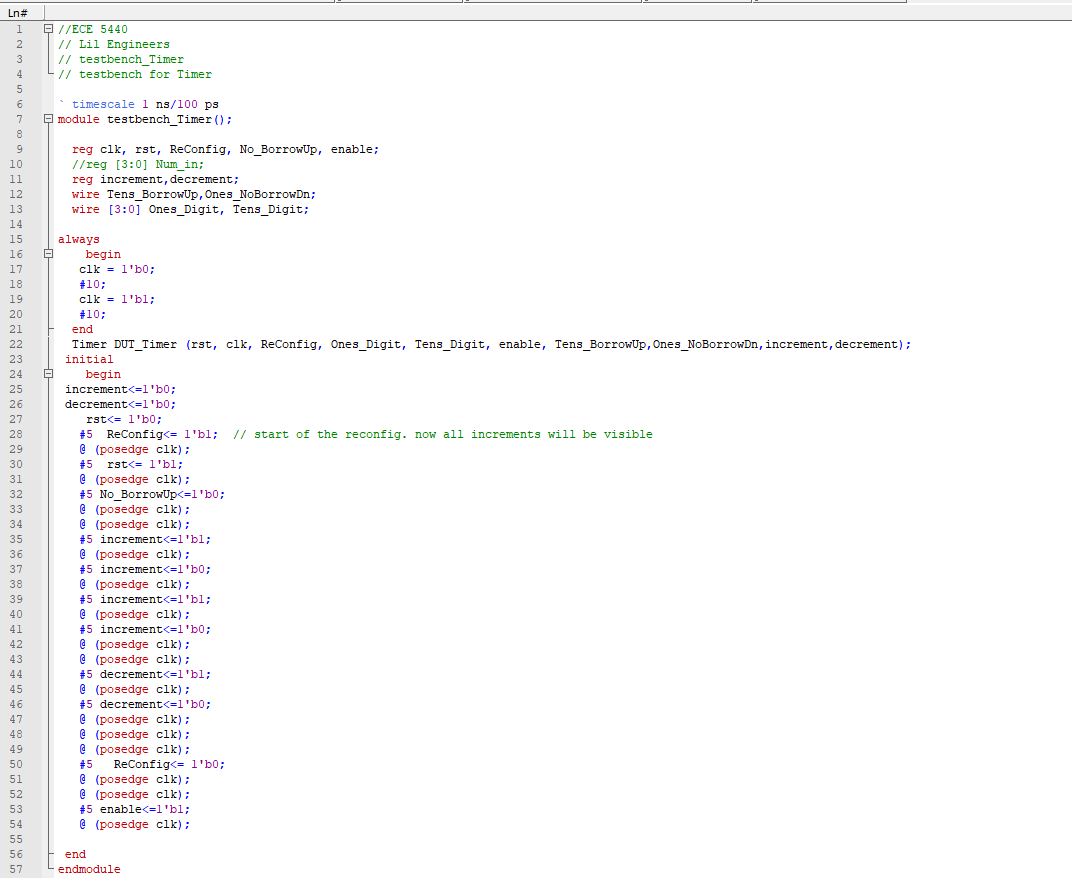
**Display Controller: Testbench**

****

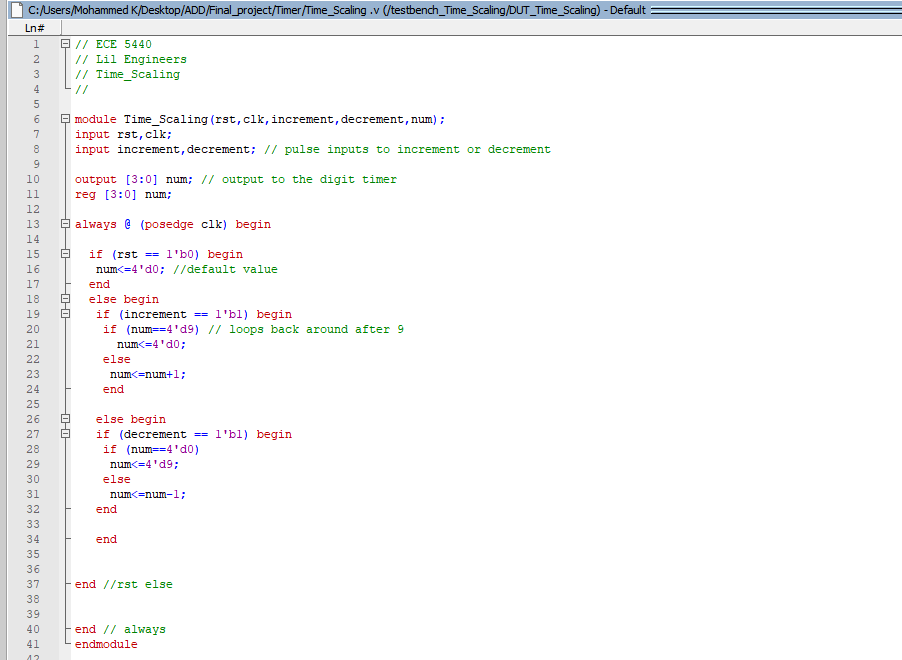
**Timer**

****

**Timer Testbench**

****

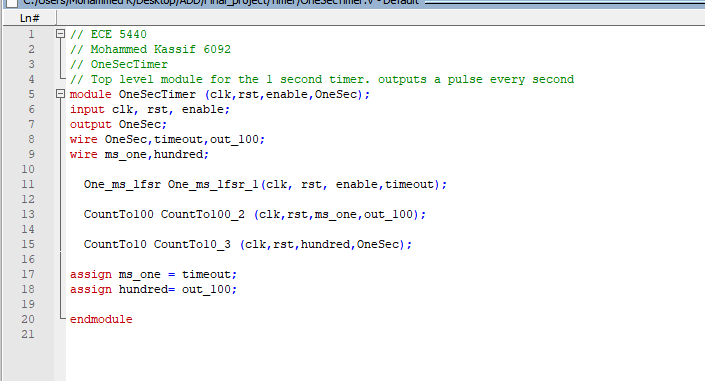
**Time Scaling**

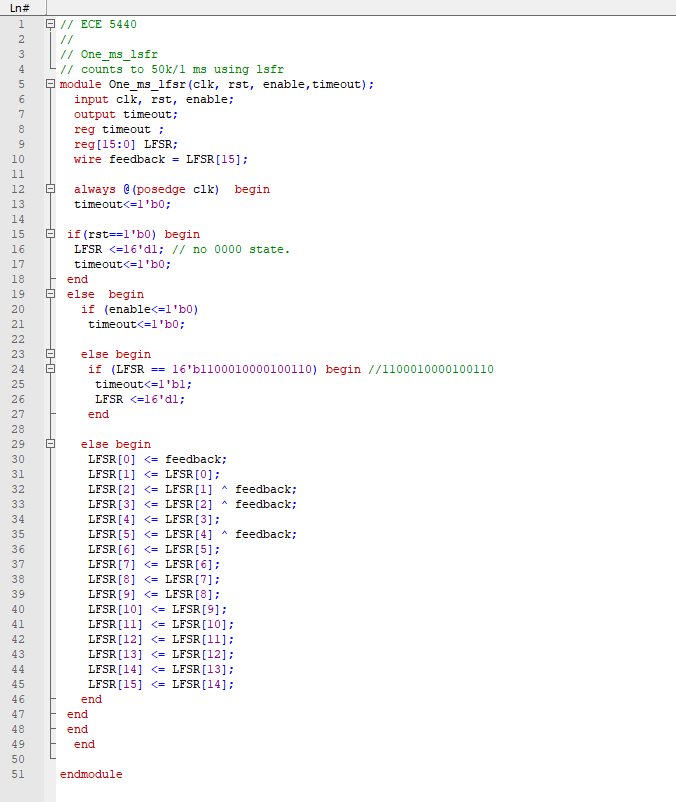
****

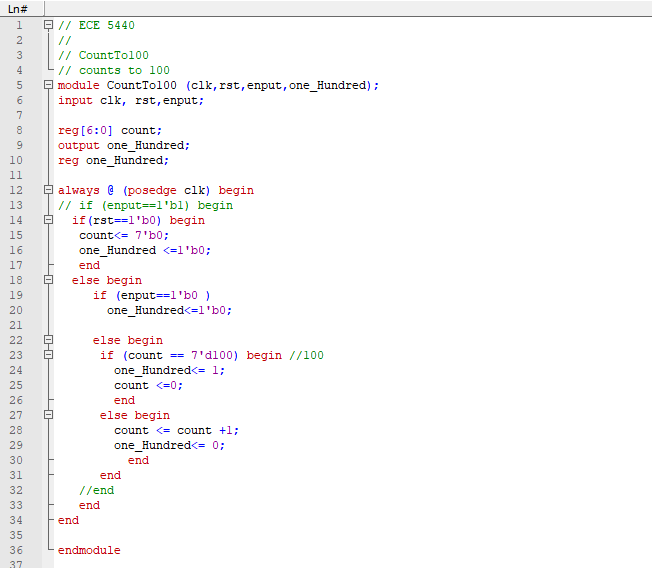
**Time Scaling testbench**

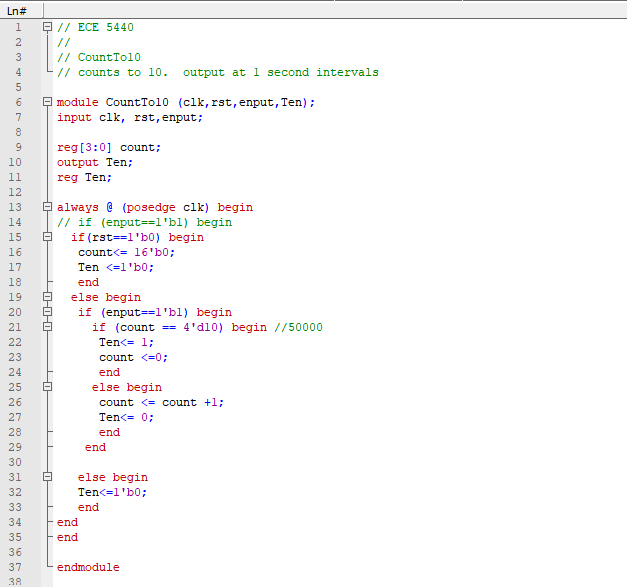
****

**One sec timer**

****

****

****

****

**Score Controller**

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

A picture containing timeline

Description automatically generated

Text, timeline

Description automatically generated

Text

Description automatically generated

A picture containing timeline

Description automatically generated

Text

Description automatically generated

A picture containing timeline

Description automatically generated

Text

Description automatically generated

Timeline

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated with low confidence

**Timer for Flashing the Sequence**

Text

Description automatically generated

**Timer Multiplier**

Timeline

Description automatically generated

**Count to 500**

Timeline

Description automatically generated

**Timer for Turning off the Sequence**

Text, letter

Description automatically generated

**RNG Module**

A picture containing table

Description automatically generated

**High Score Module**

Text

Description automatically generated with medium confidence

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

A picture containing chart

Description automatically generated

**Top Level Module for the whole Game**

Graphical user interface, text, application, email

Description automatically generated