Zhang, David J

2213233

Lab 4: rom-based game access control on FPGA

ECE 6370

# Introduction

This lab introduces three new mechanisms on top of the previous iterations of the FPGA based mental binary game. The log-in feature using the second player’s switches still functions, and now the **log-out feature can be done by pressing the player’s LOAD button whenever the timer is not running**. The load feature, which only loads a player’s number when their pushbutton is pressed, is retained however since there is no longer a second player their load button functionality has been removed. This iteration also introduces a 99 second timer that will count down at the start of the game and will lock out the player once it reaches zero. In addition, a scoring feature is added that will display at the end of the game (timer reaches zero) indicating how many correct matches the player had (combinations of RNG and player input that equate to F). Finally, if a player wishes to change their password to some other 4-digit combination, at any point when the game timer is not running they can press the RNG button to log out of the game and begin setting a new 4-digit password using the password entry system.

# System Architecture

Diagram

Description automatically generated

Figure : Overall System Architecture

The above figure depicts the top-level system architecture for the game. Along with all the modules of previous games, there are 4 new modules: Mux, RNG, 1secTimer, and digitTimer. In addition, password management is now handled by the ROM/RAM instead of the Access Controller.

Mux is a simple module that acts as a two-to-one multiplexer. It takes in two separate 7-bit inputs and outputs a 7-bit output to one of the 7-segment LED’s. The select signal comes from the Access Controller and is a single bit that is 0 for the left input and 1 for the right input. This module is needed to share resources—there are not enough 7-segment LED’s to display everything needed, so the SUM display is shared with the ones digit of the final score.

Graphical user interface

Description automatically generated with medium confidence

RNG is a module that takes in an unshaped 1-bit signal from a push button and outputs a random 4-bit number from 0 to F. As with every sequential logic module, it also takes in a global clock and reset signal. This module functions by **using a Linear Feedback Shift Register**. It inputs and shifts values every clock cycle it detects that rng\_In is high, so for this an unshaped button signal is needed to ensure that the number of clock cycles read is random, as humans cannot press buttons fast enough to consistently get certain outputs.

Diagram

Description automatically generated

Figure : Module Depiction of RNG

Figure : Module Depiction of OneSecondTimer

1secTimer is a module that consists of 2 nested modules: countTo100 and countTo10. However, all of these modules function similarly and are only nested for speed of operation. 1secTimer is functionally the same as countTo100 and countTo10, except that it counts to 50,000 instead. This is done to ensure that in total, a count of 50,000,000 is reached (50,000 x 100 x 10). At a clock speed of 50 MHz, counting to 50,000,000 clock cycles effectively counts for 1 real second. Like the RNG module, the highest module that counts to 50,000 is **implemented using an LFSR to save resources.**

1secTimer (and by extension, countTo10 and countTo100) takes in a long active high signal called enable that puts the module in operational mode. In this mode, it takes in a 1-bit input that will increment an internal counter until it reaches the specified maximum (10, 100 or 50,000) at which point it will output a pulsed 1-bit signal high. Importantly, the most nested module takes in an input directly from the global clock. Its output is then fed as an input to its upper-nested version. As an example, since countTo10 is nested inside countTo100, then countTo100 will only increment when countTo10 has reached its stated maximum (10). This way, when countTo100 outputs, it has actually counted 100x10 clock cycles, not 100.

Diagram

Description automatically generated with medium confidence

Figure : Module Depiction of Digit Timer

digitTimer is a module that has 2 signals that come and leave externally and 4 signals that it uses to interface with other iterations of itself. This module is non-digits place specific, meaning that it can be used for a tens digit, a ones digit, a hundreds digit and so on non-specifically. The 2 “external” signals are a 1-bit active high signal called reconfig and a 4-bit binary number output, num\_out. The reconfig signal comes from the access controller and is only sent upon Game Start. The 4-bit num\_out represents the number that the digitTimer is holding. Importantly, since these modules are for a timer, the 4-bit maximum is defined in the module to be decimal 9, not the true 4-bit maximum of F as that would be unwieldy and awkward for a timer.

The 4 “interface” signals are 1-bit active high signals that are used to interface with other iterations of itself. DecrementD and DecrementU are pulsed signals that come from Downstream or go Upstream to tell the receiver to decrement its internal register. noBorrowD and noBorrowU are long signals that similarly come from Downstream or go Upstream to tell the receiver that it cannot “lend” a 1 to its messenger. The logic of this module is modeled after how we as humans perceive subtraction—for example, for 22 – 8, a human will look at the ones place and see that 8 cannot be subtracted from 2, so we look to the tens digit, see that we can “lend” a digit to the ones place and do so to create a mathematical operation that makes sense: 12-8.

By default, the noBorrowD signals (outputs) are set long high when reconfig is pressed—in other words, when a 9 is loaded to the internal register the module tells its downstream neighbor that it has numbers to lend. When the internal register counts down to 0 as a result of DecrementD signals, it checks if noBorrowU (inputs) is low—the signal from its upstream neighbor indicating that it CAN borrow—and if it is, it sends a DecrementU (outputs) signal and resets its own internal counter to 9. If it is not, then it holds its register at 0 and outputs noBorrowD high—indicating to its downstream neighbor that there is no higher order digits that it can borrow from.

Importantly, in Figure 1 the leftmost Digit Timer does not have its DecrementU or noBorrowD signal drawn. As this is the most significant digit of the timer, the tens spot, noBorrowD is set to always 1 to ensure that it does not try to lend from a fictional hundreds spot that doesn’t exist. DecrementU is not used anywhere then, so where it is mapped is not important.

**For the Lab 4 bonus features,** RAM and ROM represent the onboard memory that stores the values of the password for log in entry. Initially, the 4-digits are stored in ROM but if desired a new password can be written into RAM which the system will use as the new password in subsequent log ins. Each memories respective dataBus signal is a 4-bit signal that represents the value of the digit from 0 to F, and each address signal represents the value of the address for each number. For a 4-digit signal, only 2 bits are needed for addressing but since the smallest size of ROM/RAM to instantiate is 32 addresses the top 3 bits are zero-padded. RAMWR is a necessary signal for the RAM only to tell it whether the databus contains data to be written or read into RAM, respectively.

Other modules not listed are legacy code from the previous iteration, but in short:

Button Shaper outputs a single clock cycle active high signal whenever it detects that its input has gone low. It will only send out another pulsed signal after input has gone back to high, then low again.

Load Register passes its 4-bit input to 4-bit output whenever it receives a 1-bit load signal from the Access Controller. This load signal is shaped from Button Shaper and is in reality the push of the Player’s LOAD button.

Diagram, schematic

Description automatically generated

Figure : Finite State Machine for Split Access Controller

Depicted in Figure 5 is the modified FSM for the Split Access Controller. **The responsibilities of the Access Controller have been split into Authentication and Game Controller.** Authentication handles the log in and password change features, while Game Controller handles all game functions. **Signals that cross the dotted line do not represent state change conditions, but signals that are passed between modules**. The addition of a new WAIT state for the Game Controller is necessary to handle initialization, since now both modules FSM’s are technically independent from each other.

In TIMER RECONFIG, all player inputs are blocked, the reconfig signal is set high and immediately moves to PRE-GAME START. For implementation of the bonus feature, the internal signal score\_enable is set to 0 during TIMER RECONFIG for correct looping behavior. During PRE-GAME START, reconfig is set back low to ensure that it was only high for one clock cycle. In addition, it waits until Button Push is high—the shaped signal coming from the Game Start button—before setting timer\_enable to high and moving to GAME IN PROGRESS.

During GAME IN PROGRESS, player inputs are finally unblocked and all inputs are passed to outputs. Once the timeOut signal has gone high—from the timer reaching 0 seconds—the system moves to the last state, GAME OVER. In GAME OVER, timer\_enable is set low to turn off the function and all player inputs are blocked again. Importantly, in this stage the bonus feature must be implemented: an internal signal named score\_enable is set to 1, changing the output of the muxes to the score counter instead of the sum display. The system remains in GAME OVER until it detects that the Game Start button has been pressed again, after which it moves back to TIMER RECONFIG and the cycle begins anew.

For the implementation of the previous bonus feature, the score counting mechanism, an internal counter records how many correct matches there were during GAME IN PROGRESS and divides that number by 10 to get the tens digit and computes the modulo of the number to obtain the ones digit. These two numbers are then passed into separate 7-segment decoders to obtain a 7-segment display version of them. These decoded scores are sent to the mux, which only shifts values when select is switched high—only possible in the GAME OVER state. When GAME OVER is exited, the mux switches back to the normal sum display, as depicted in Figure 2.

**For the new bonus features**, logout and password reset, 2 new 1-bit control signals (load\_button and rng\_button) are needed to exit out of any state where the timer is not currently running. These control signals directly come from the push button they are named after and move the current state back to wait while also setting loggedout to 1 or passReset to 1, respectively. Loggedout is an internal trigger for Authentication that simply moves the system back to DIGIT and waits for the correct password. Passreset is a signal that triggers a special state of DIGIT where new password values can be entered to RAM through the password switches. After 4 numbers have been input, a special flag is raised that reads subsequent password values from the RAM instead of the ROM.

# Simulation Results

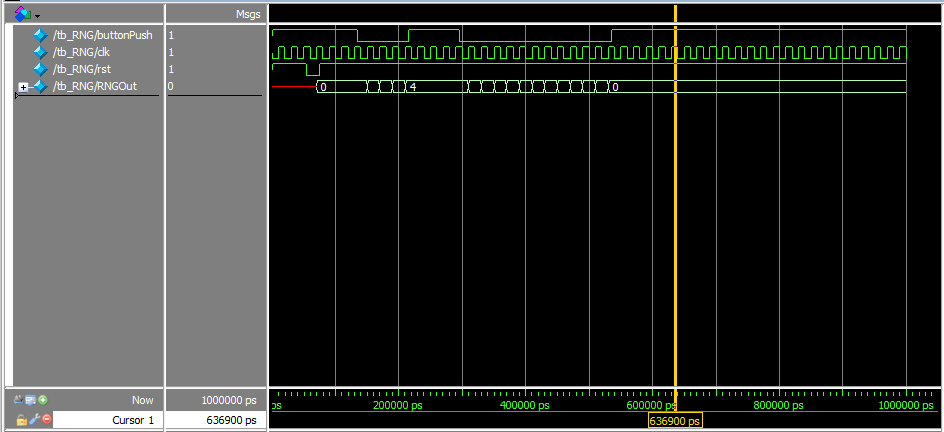


Figure : Simulation Results of RNG Module

Pictured above in Figure 6 is the output waveform of the RNG module. When buttonPush is low (normal operation for a pushbutton) for 4 clock cycles, the output is incremented to 4 one at a time. After an arbitrary amount of time, buttonPush is set low again for 12 clock cycles and the output increases again one at a time until it reaches F, at which point it will overflow and roll back to 0. This behavior is desired, as this will be how we ensure that the output is always a number between 0 and F.

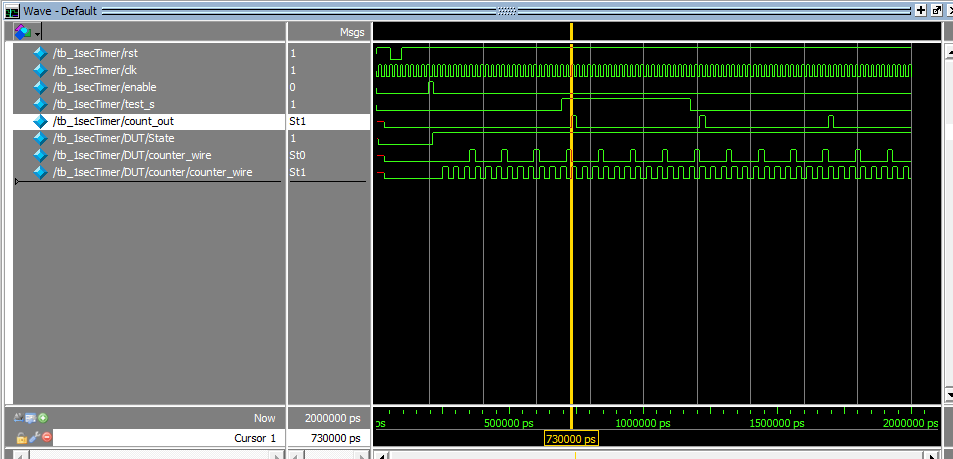


Figure : Output Waveform of Modified 1secTimer, First Output

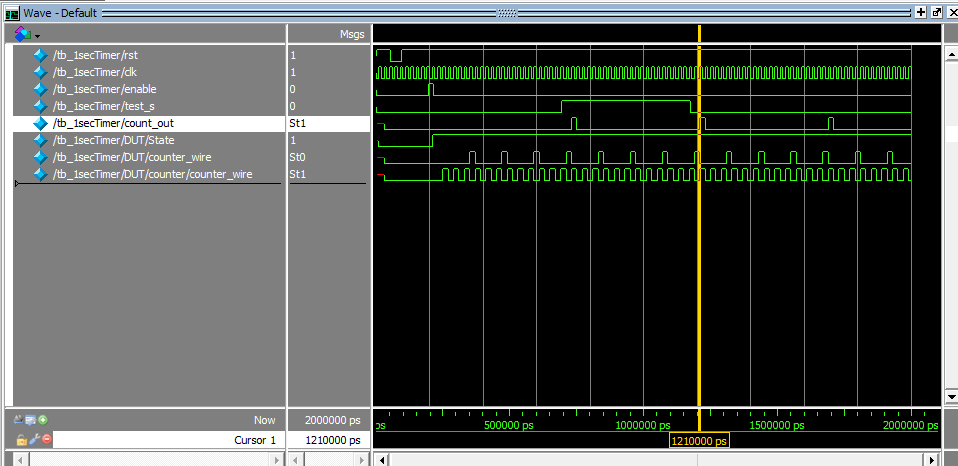


Figure : Output Waveform of Modified 1secTimer, Second Output

Depicted above in Figures 7 and 8 are the waveforms of the modified 1secTimer. While the real 1secTimer needs to count to 50,000,000, this would be unfeasible for testing purposes so the final count was modified to be 24 (4x3x2) as a proof of concept. This means that all outputs should be separated by 24 clock cycles. The signal highlighted in the above figures, count\_out, is the output of the highest level module, i.e. the one that needs to be separated by 24 clock cycles. Counter\_wire and counter/counter\_wire represent the nested signals from one module to another.

As can be seen, counter/counter\_wire is high every 2 clock cycles, and counter\_wire is high for every 3 counter/counter\_wire highs. Counter\_out is then only high for every 4 counter\_wire highs. As a quick proof of function, we can look at the marker in yellow—the first total output occurs at 730 ns and the second one at 1210 ns, a difference of 480 ns. In our testbench (Appendix C) the clock cycle is defined to be 20 ns long, indicating that there are 24 clock cycles between outputs: a perfect match.

Chart, treemap chart

Description automatically generated

Figure : Waveform of OneSecondTimer Done with LFSR

Due to the way that LFSR work, a termination value of 50,000 must still be set in a 16-bit LFSR to get correct looping behavior. However, since LFSR do not count linearly from 0 to 50,000 but input and shift values, one easy way to decide the new termination value is simply to compare the output of the LFSR timer with that of the binary timer. By seeing which number corresponds to 50,000 in the binary timer, the termination value can be found quite easily. Figure 9 shows the behavior of the LFSR timer when this correct termination value is used and as can be seen, it perfectly replicates the old binary timer.

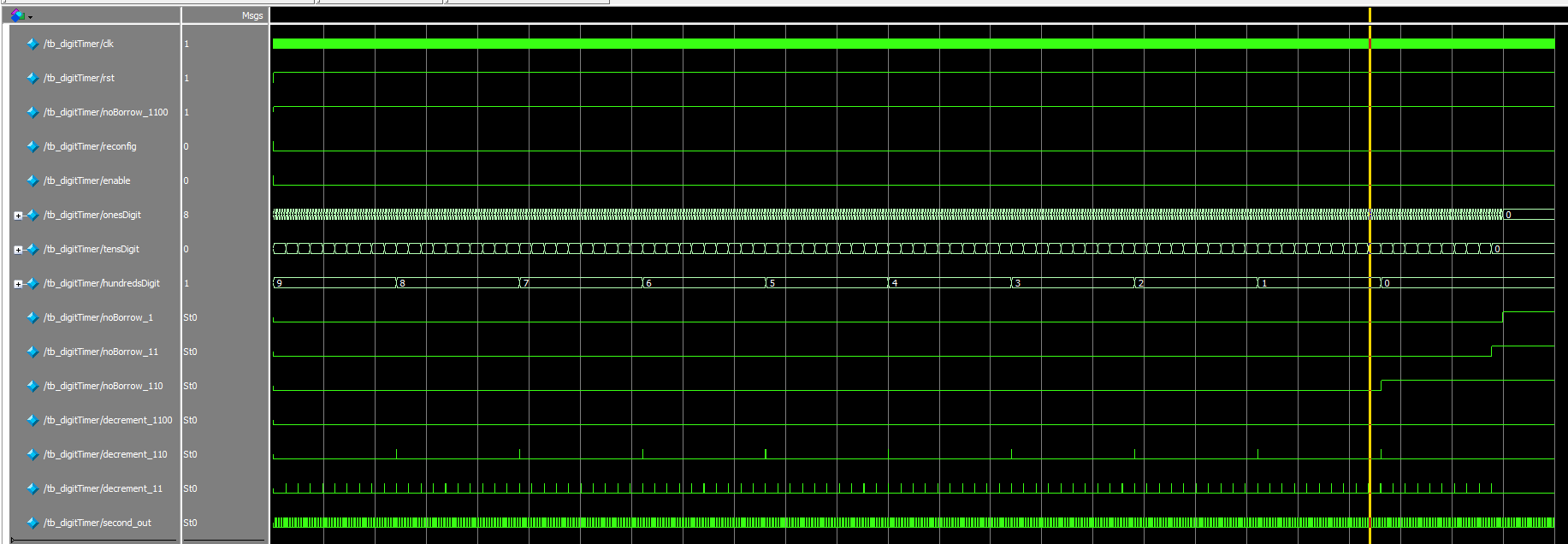


Figure : Output Waveform of 3 digitTimer Modules

Depicted above in Figure 10 is the output waveform of 3 instances of digitTimer counting down from 999 to 0. While it is impossible to read what is happening in onesDigit, what is important is to notice the operation of hundredsDigit and tensDigit. From these, the operation of tensDigit to onesDigit can be inferred since these modules are all instances of each other. It takes 10 changes of tensDigit to decrement hundredsDigit by one, indicating that tensDigit is correctly counting down from 9 to 0, and then looping back to 9. At the yellow marker, when the hundredsDigit is just about to turn to 0, we can see a little further down that the instant that hundredsDigit turns to 0 it detects that noBorrow\_1100 (the borrow signal from a theoretical thousands digit) is 1, changing its output noBorrow\_110 from zero to one indicating to tensDigit that it cannot borrow any more from it. Ten digit changes later in tensDigit it sees that noBorrow\_110 is high and changes its own noBorrow\_11 from zero to high indicating to onesDigit that it cannot borrow more. onesDigit replicates this behavior and after 10 of its own digit changes, it changes its output noBorrow\_1.

# FPGA Board Testing Results

Depicted below are pictures of the FPGA in various, key stages of operation. The captions for each picture explain what step is depicted.

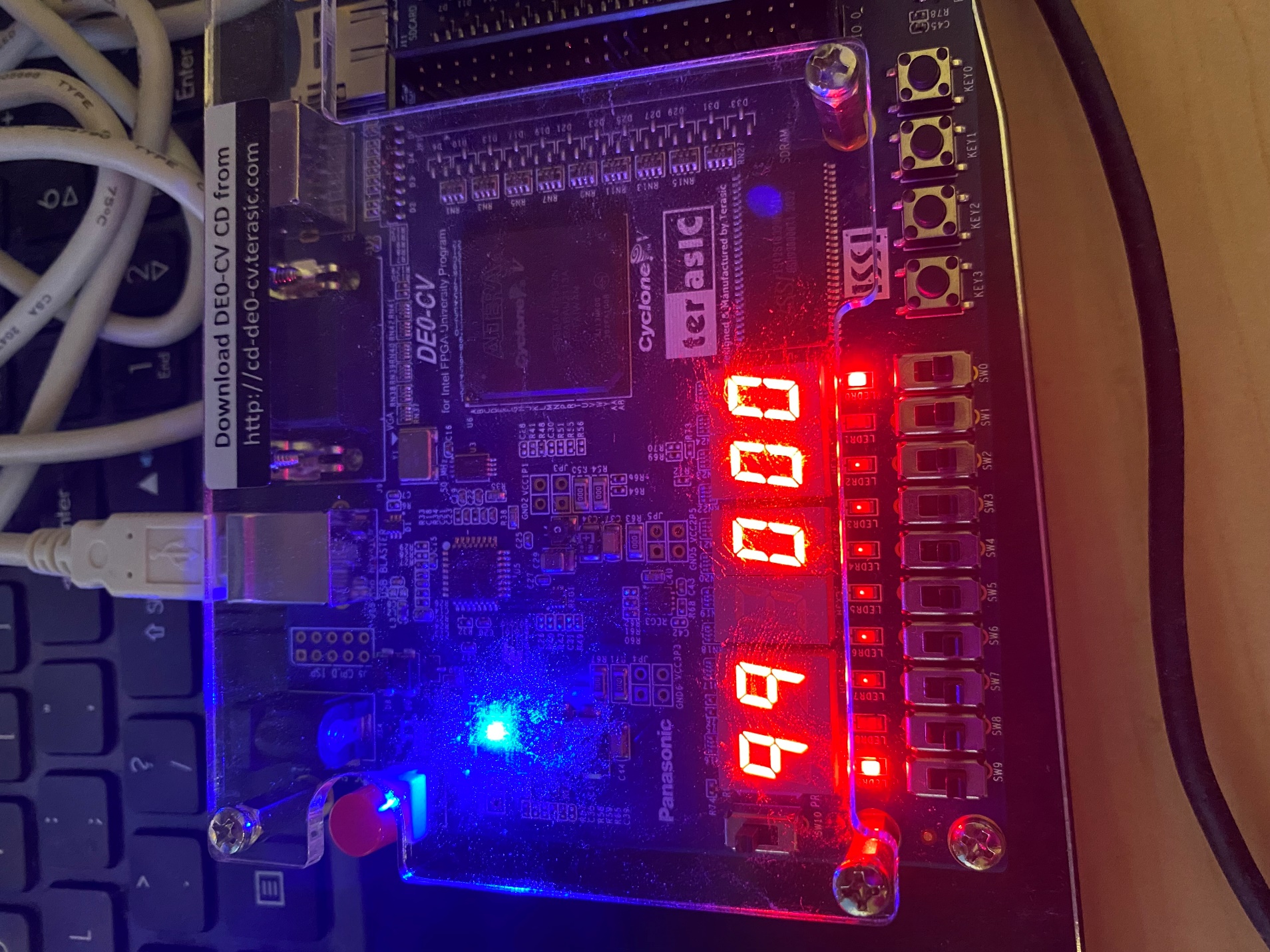


Figure : FPGA After Logging In: Pre-Game Start

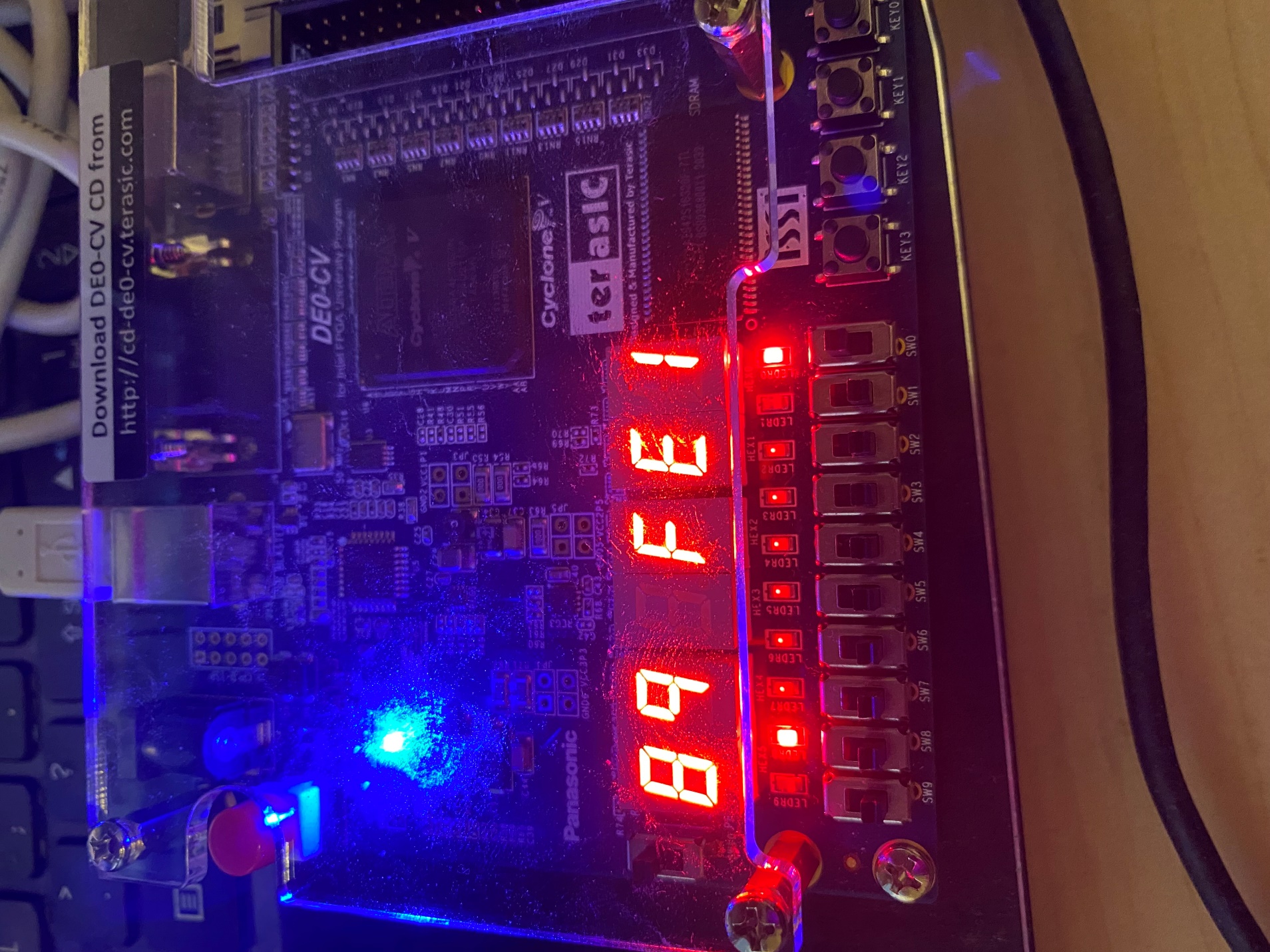


Figure : Game in Progress

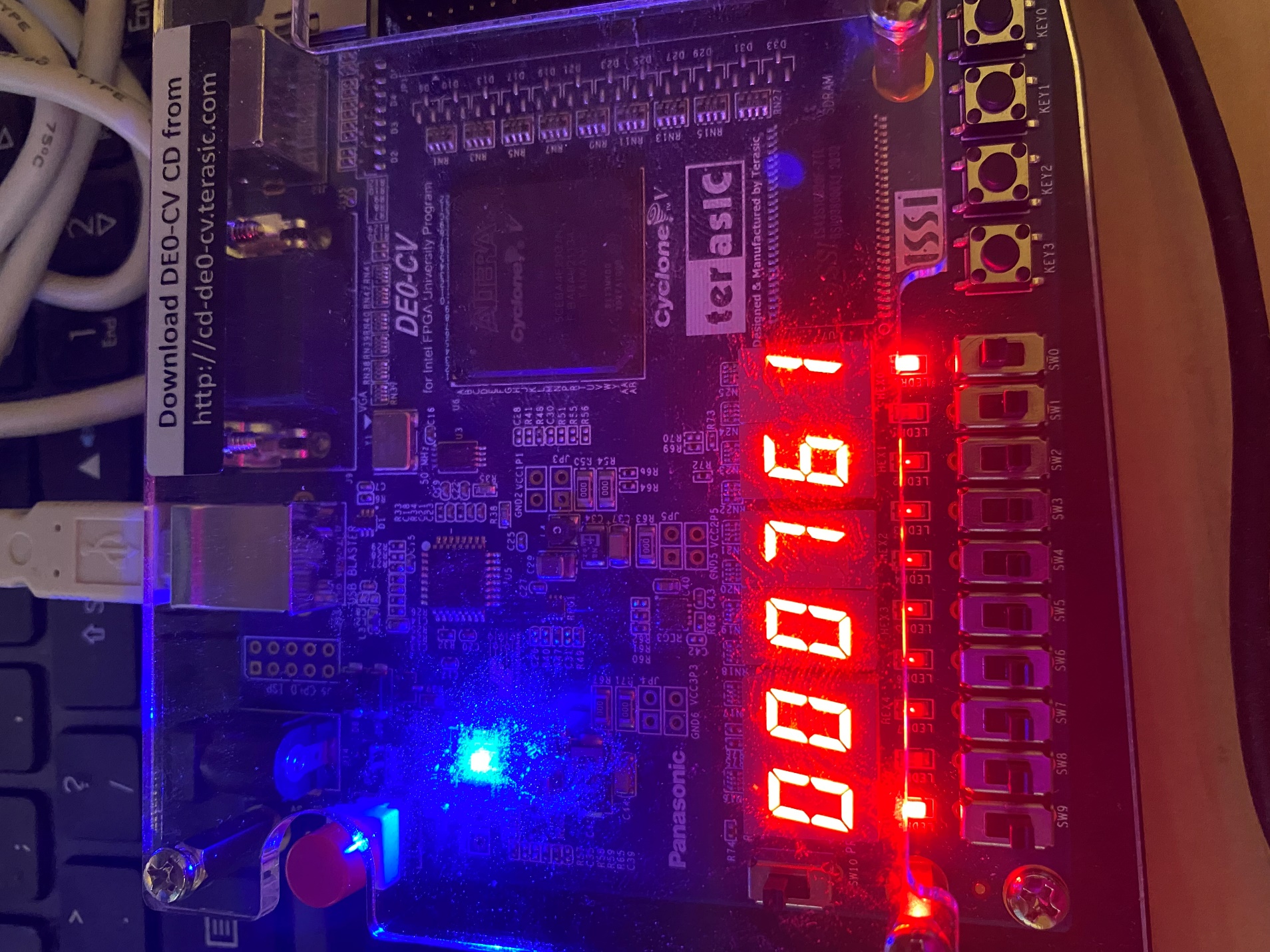


Figure : Game Over with Score Depicted

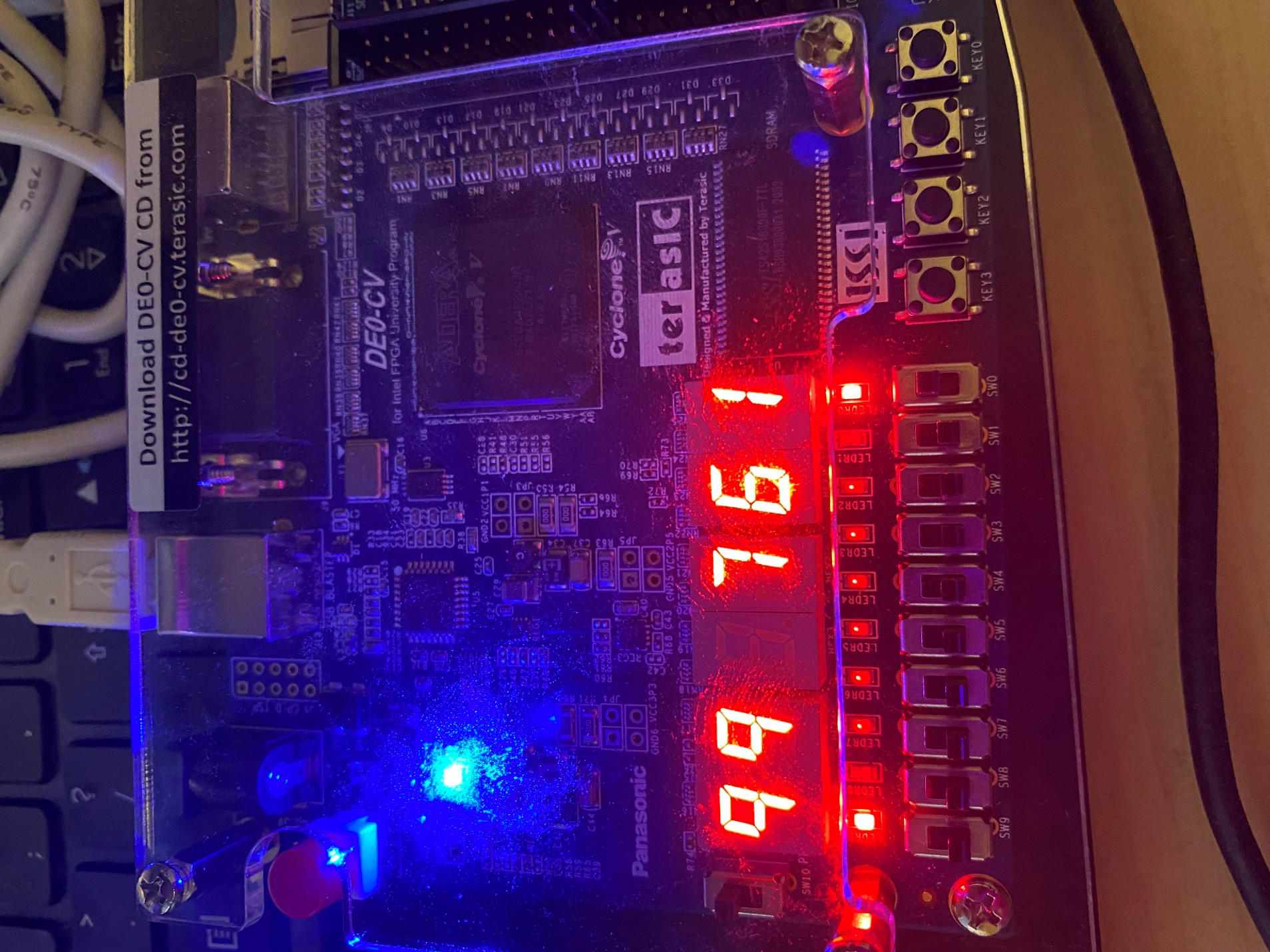


Figure : Pre-Game Start of Game 2

It is important to note that in Figure 13 while it may look like the score depicted is actually the sum of RNG and Player 1, it is not. The correct score LED is off, indicating that the game is over as well as the 3rd display showing 0 instead of being off.

Several video demos are shown below illustrating key functions of the new game.

<https://drive.google.com/file/d/1EGNSd8g9HJ9wQEqASp-rZ16MK3skOBR-/view?usp=share_link>

The video demo above depicts what the board should look like after correctly logging in. A 99 should be displayed on the timer portion and all player inputs should be locked out. When the Game Start button is pressed, the timer begins counting down from 99 and the game has begun.

<https://drive.google.com/file/d/10DozI-mu0b0nq8crwDgMqS1due3Z9xwN/view?usp=share_link>

The video above demonstrates the functionality of the RNG button and an example round being played. The player will hold the RNG button for as long as they wish to generate a number, and then attempt to match that number to 15 using their slide switches. If the sum is correct, the matching LED will turn on.

<https://drive.google.com/file/d/1yrBwHdfnqTjIFGQVVAFpznf_sbw16nY6/view?usp=share_link>

The video above shows the functionality of the score counter. After the timer has reached 0 seconds, the middle two displays will change to the score counter display instead of just sum. In the scenario depicted in the demo, 4 correct matches have been done.

<https://drive.google.com/file/d/1Mcj7-ZlvStBLn5D8iwJH94jFhoO-n1UE/view?usp=share_link>

The video demo above depicts the Game Over state and the beginning of a second iteration of gameplay. In the Game Over state, all the player inputs will be locked out with the only way to progress being pushing the Game Start button. Once this is done, 99 is loaded back into the timer and the middle two displays switch from displaying score to displaying sum again. After this, a second push of Game Start counts the timer down again—signifying a new round of the game.

<https://drive.google.com/file/d/1nlhqI6b15eV04f9KkGcBECbihv353-9_/view?usp=share_link>

The video demo above shows the new logout feature. After logging in, a player’s LOAD button can be pressed to log out. Entering the correct password will subsequently log the player back in.

# Conclusion

This new iteration of the game shifts the mental-binary game from a 2-player game to a single player game, increasing replayability and skill level required. The implementation of the bonus feature was actually the hardest feature to create—far more difficult than the timer and digit displays. To do so, a shaped signal from the RNG pushbutton needed to be created for timing reasons.

Detection of a single correct answer is done whenever RNG is pressed, not when player 1’s Load button is pressed, as there is no way to check the next output of the Adder module at the initial rising edge. To remedy this, the Adder output is checked at every RNG push. Logically, a player will either keep trying to match an RNG number or move on to a different one. In both cases, at every RNG push the Adder output will be incorrect and the internal register doesn’t increment. A player will only be correct if they match the RNG number and then press the RNG button to move on to a new number. In this situation, at the rising edge of RNG pushbutton, the adder output is checked and found to be correct, and the internal register is incremented.

The unintended side effect of coding the score counter like this is that it relies on players to play as fast as they can, since correct answers are only counted when moving on to a new number. If a player correctly matches a number right as the timer hits 0, unfortunately that isn’t counted towards the score total as RNG has not yet been pressed.

**The new bonus features** logout and an LFSR RNG module have also been implemented. The updated LFSR RNG code can be found in Appendix B. Since password values are no longer hard coded into the FSM of the Access Controller but read off the ROM, the various DIGIT states can all be condensed down into one DIGIT state that iterates a set number of times. This change can be seen in Appendix F. Password reset proved difficult to implement, as in the end there was something wrong with how values are being written into the RAM—the correct password is not the entered values but 0000. The address is being iterated through correctly and it works in simulation, but something about the RAM is messing with the write timing. Since there is no way to see this, this feature was left broken.

# Appendix A: Verilog Code for Lab 4

Text

Description automatically generatedText

Description automatically generated

Figure : Module Definition for Lab 4 in Quartus

# Appendix B: Verilog Code for LFSR RNG

Text

Description automatically generated

Figure : Module Definition for RNG

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated with low confidence

Figure : Testbench Definition for RNG

# Appendix C: Verilog Code for 1 second Timer

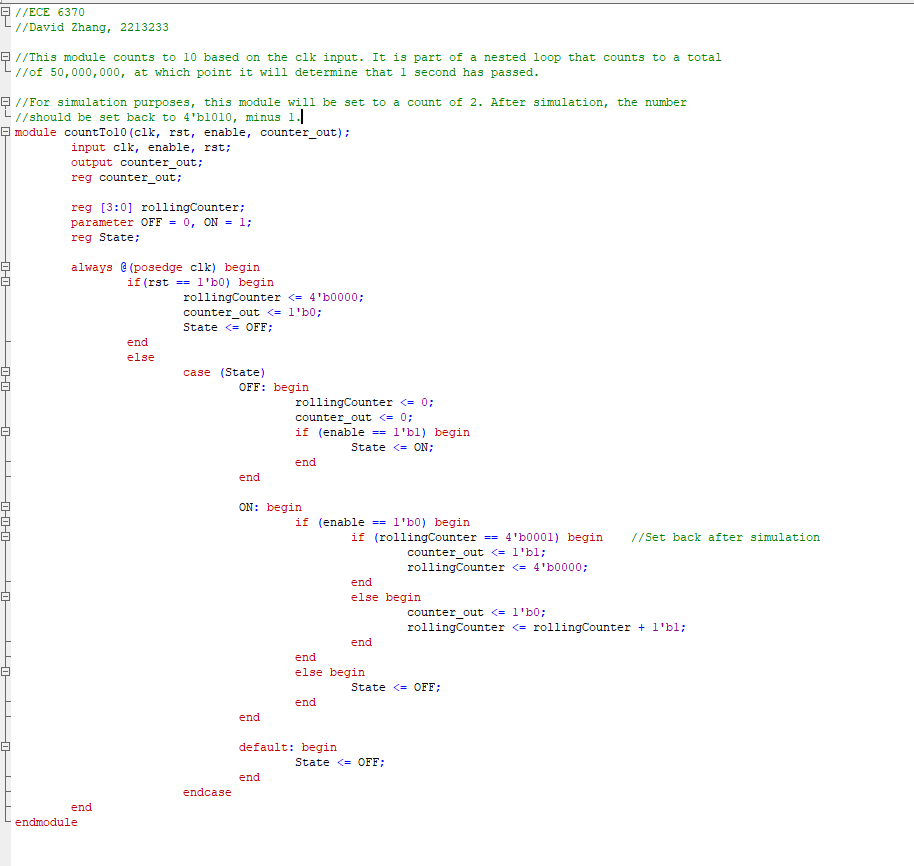


Figure : Module Definition for countTo10, the 3rd nested module.

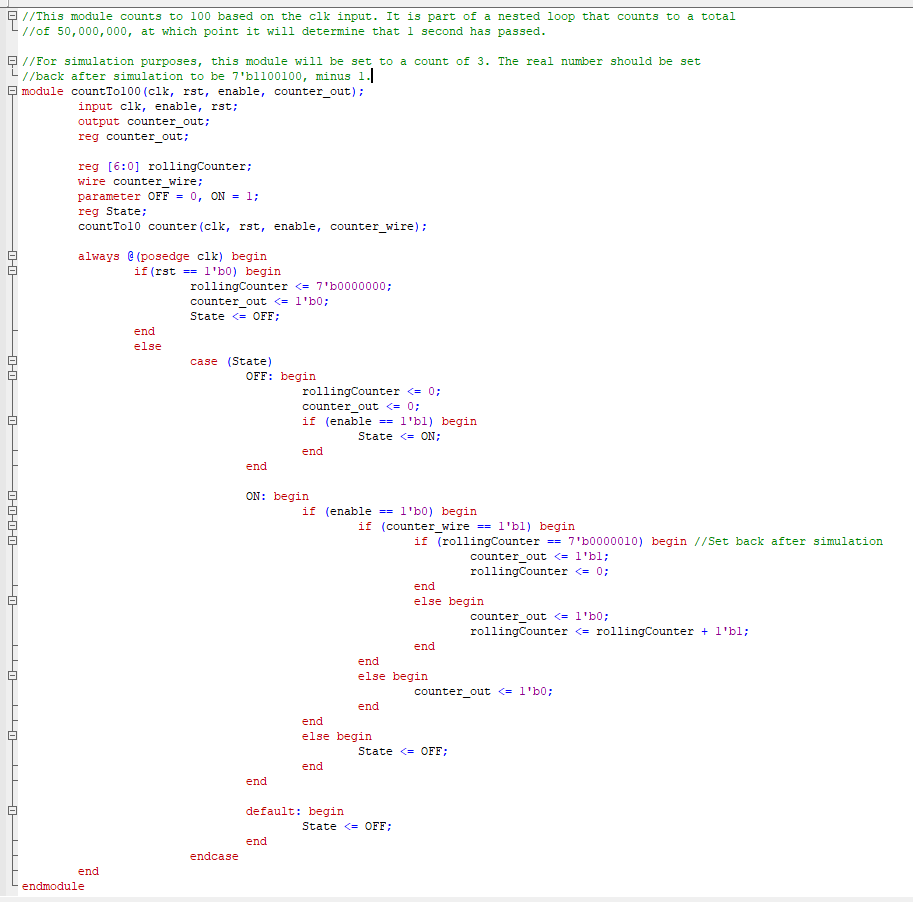


Figure : Module Definition for countTo100, the 2nd nested module

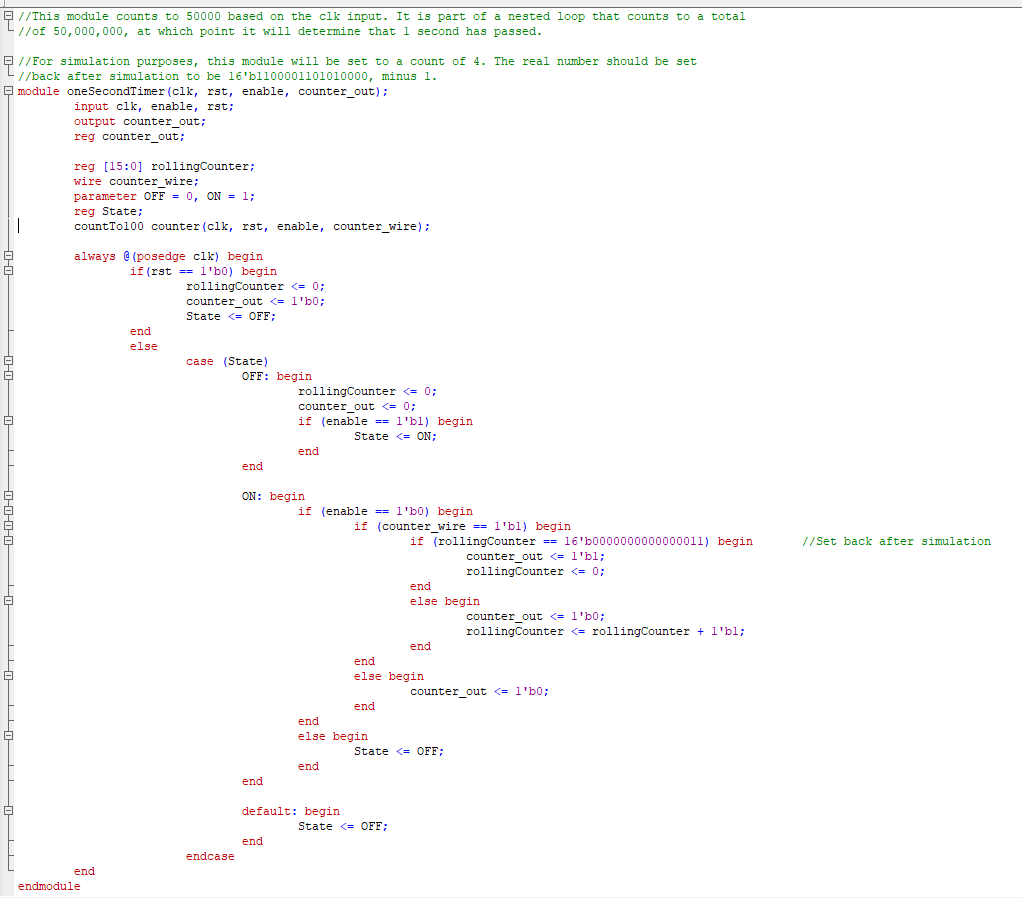


Figure : Module Definition for 1 second Timer, the highest module

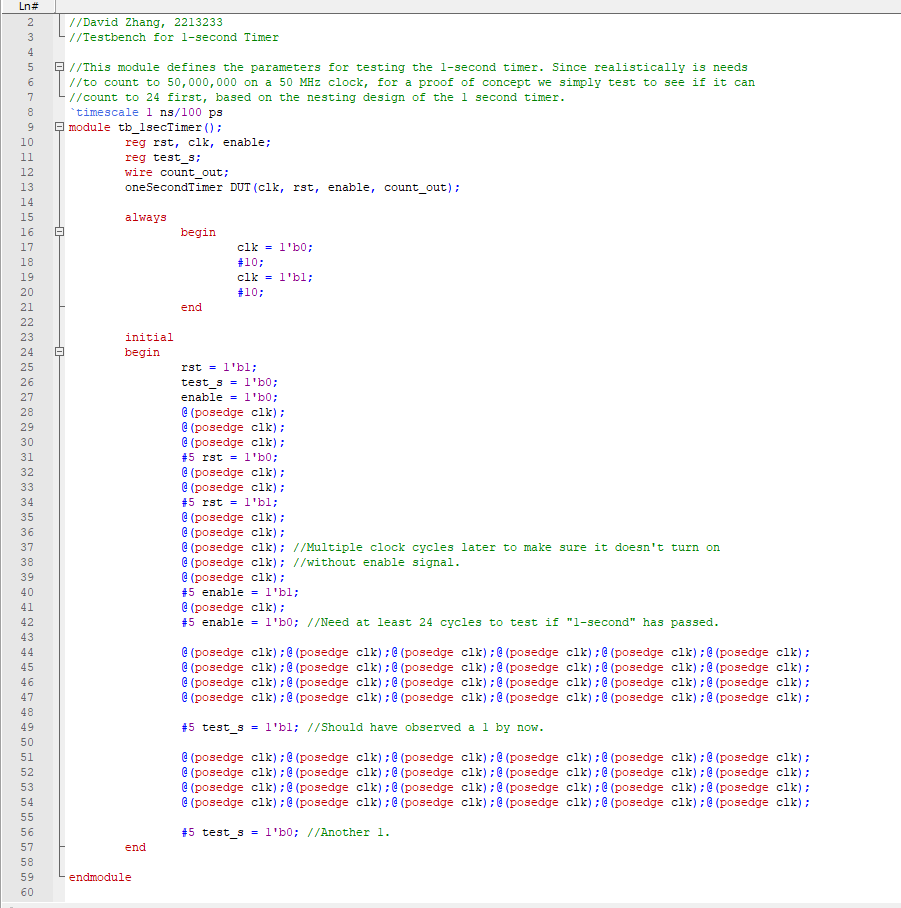


Figure : Testbench Definition for the Modified 1 second Timer

# Appendix D: Verilog Code for Digit Timer

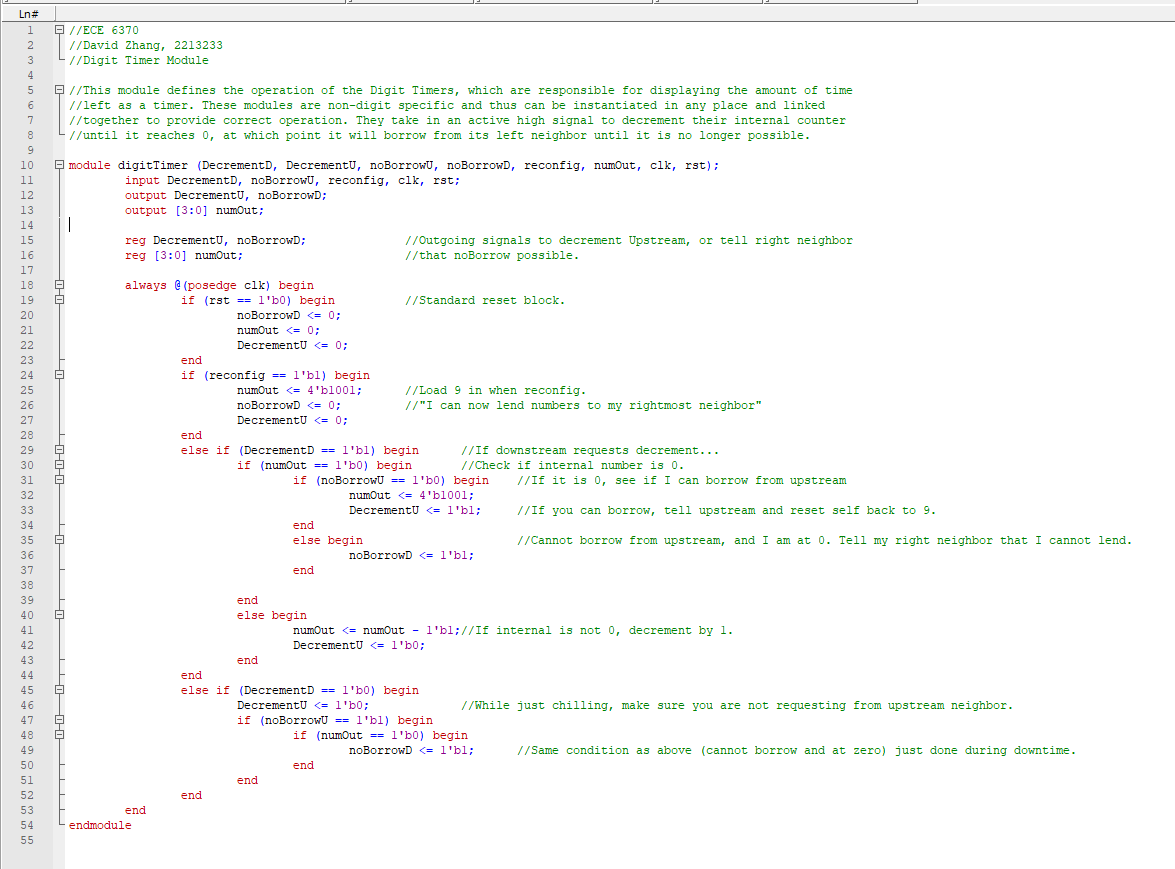


Figure : Module Definition for Digit Timer

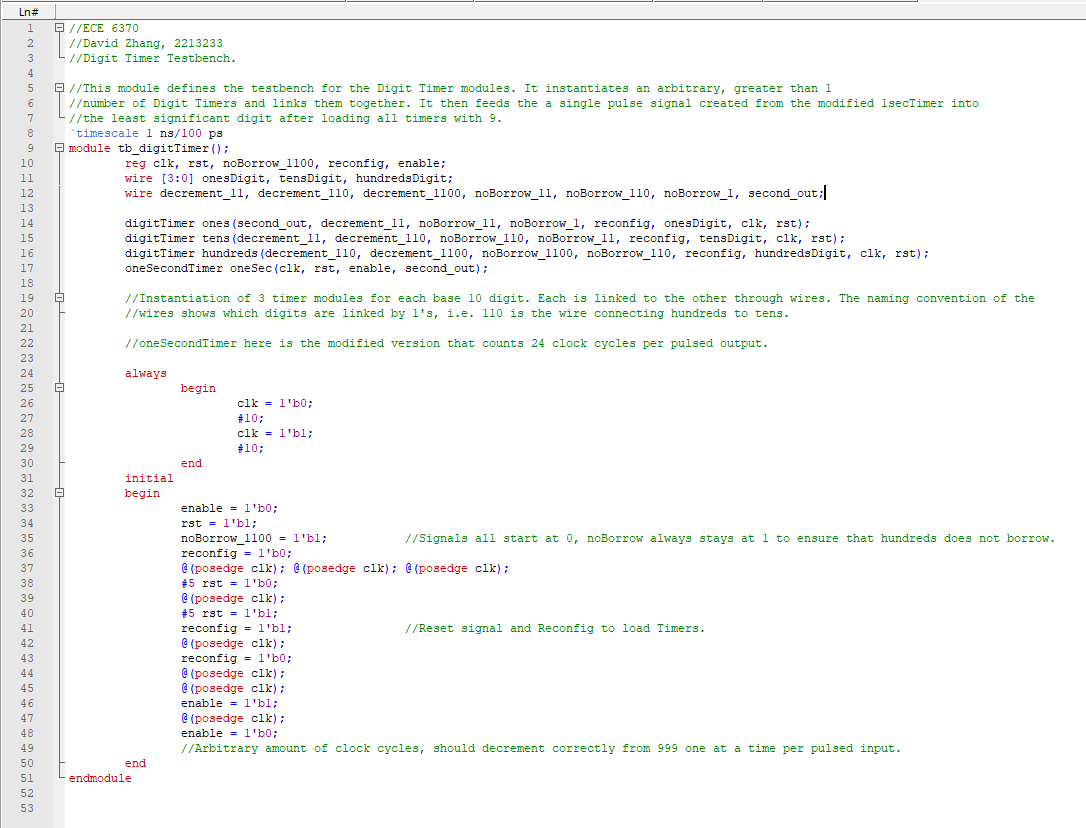


Figure : Testbench Definition for 3-instance Digit Timer

# Appendix E: Verilog Code for 2-to-1 Multiplexer

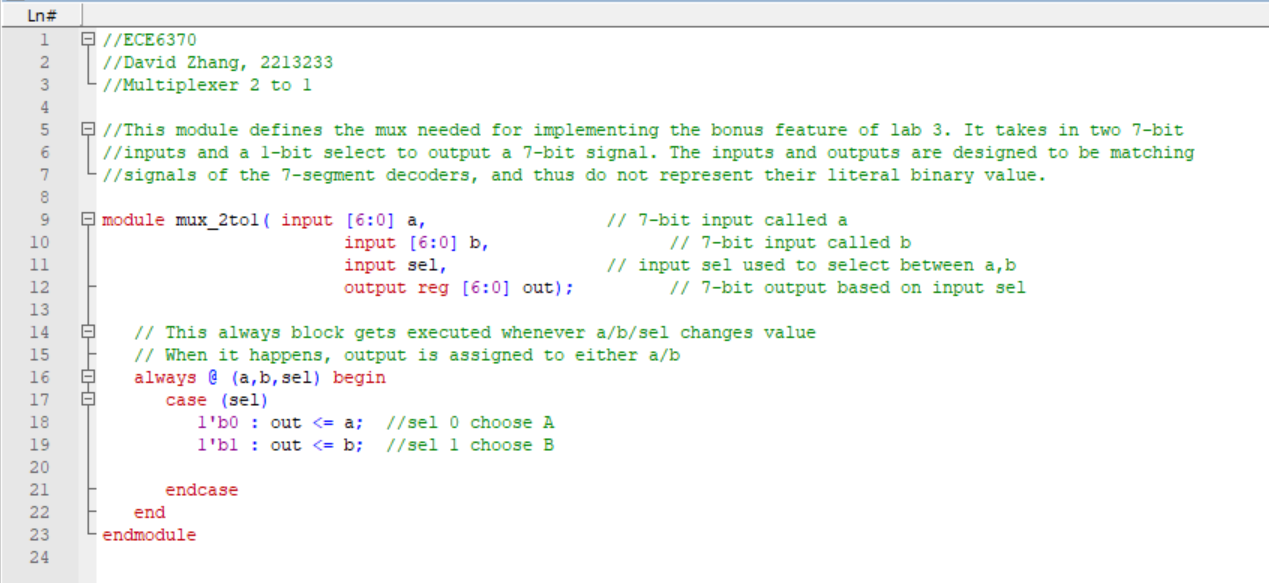


Figure : Module Definition for the 2-to-1 7-bit Multiplexer

# Appendix F: Verilog Code for Modified Access Controller

Text

Description automatically generated

Figure : Module Definition for Modified Access Controller

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated

Figure : Module Definition for Authentication Half of Access Controller

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated with low confidence

Figure : Game Controller Module half of Access Controller

# Appendix G: Legacy Verilog Code

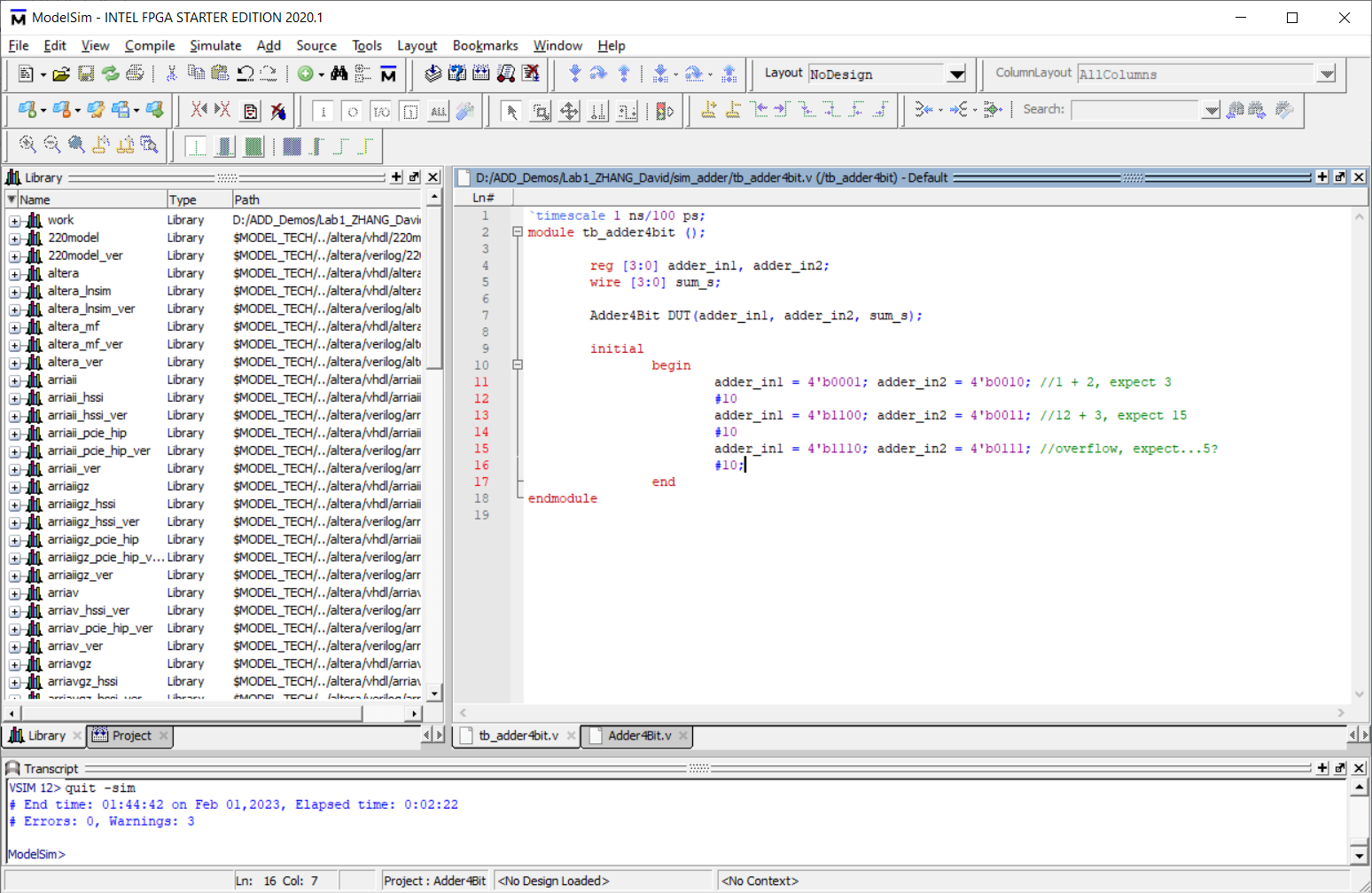


Figure : Module Definition for Adder

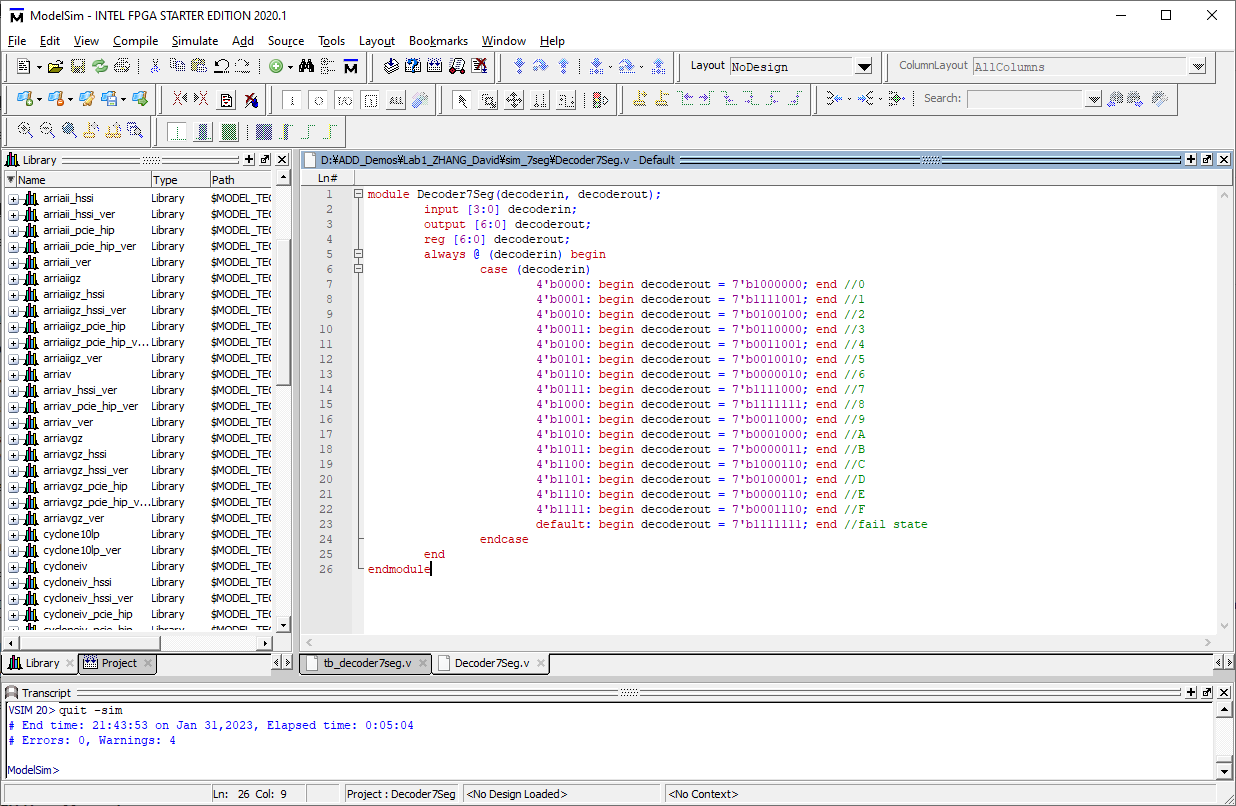


Figure : Module Definition for 7-Segment Decoder

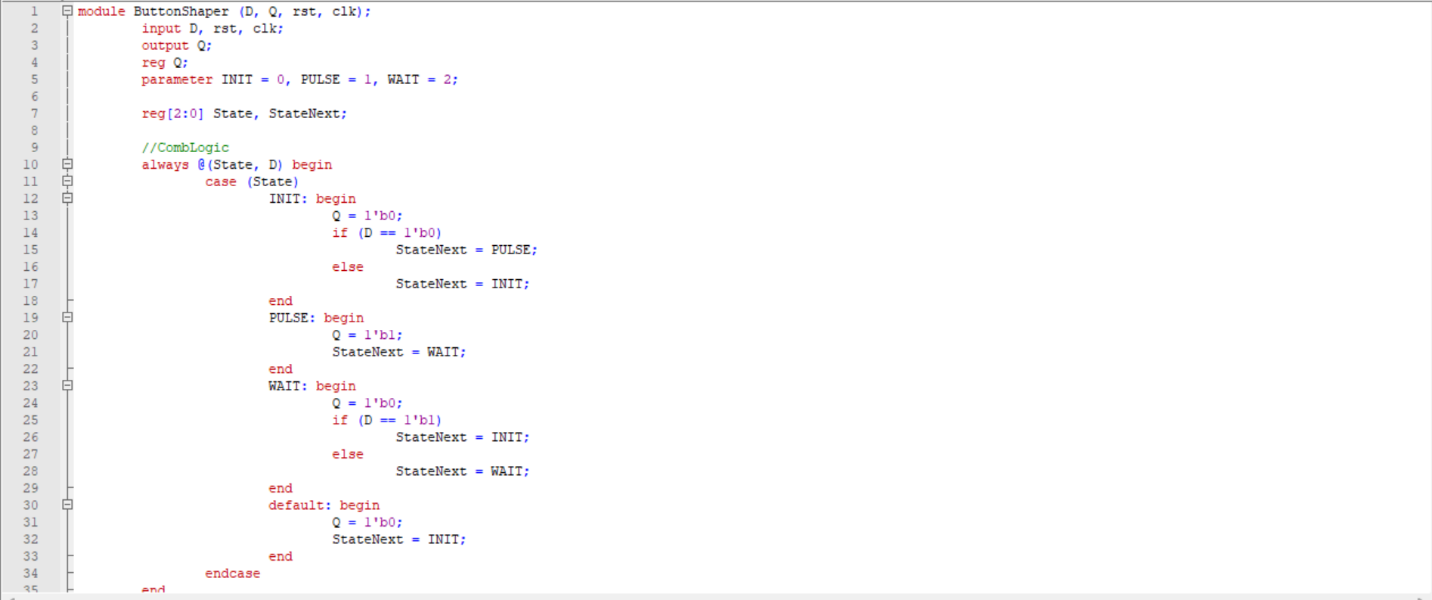


Figure : Module Definition for Button Shaper

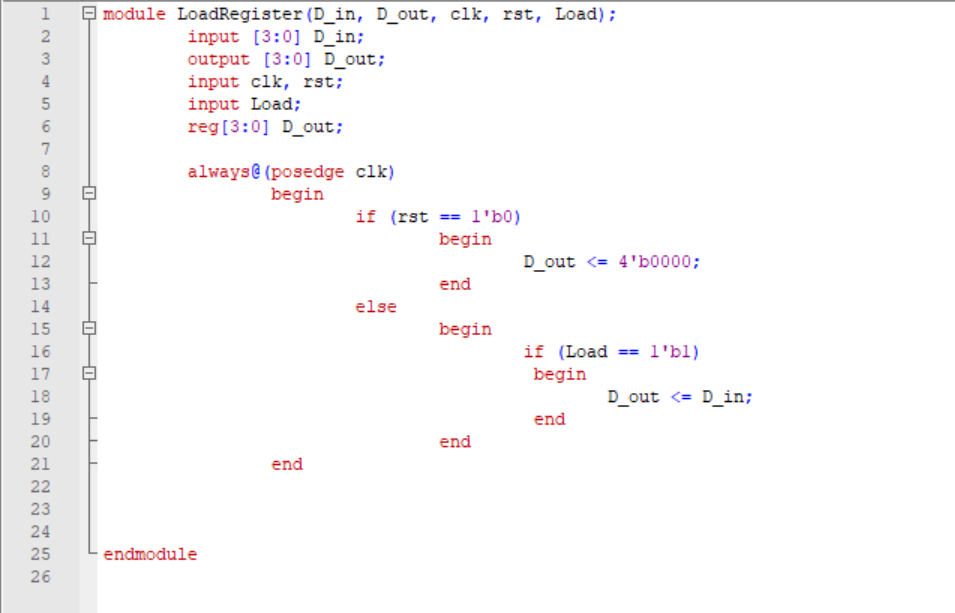


Figure : Module Definition for Load Register