Lab 3: Hardware‐based True Random Number Generator and Timer Control on FPGA

Thomas Vo

2179861

ECE 5440

# Introduction

This game is a binary arithmetic game where a player toggles switches to reach a target sum of 1111 (binary 15). A hardware-based Random Number Generator (RNG) determines the second number.

A circuit board with numbers and red arrows

AI-generated content may be incorrect.

Figure 1. Diagram for Controls and Displays

As shown in Figure 1, the 7-segment displays show the number entered by the player, the randomly generated number, and their sum. LED indicators show whether the entered numbers match 1111, as well as the player’s login status. The 4 right-most switches are for the player’s input, while a push button is used to generate a random number. The 4 left-most switches are used to enter a password at the start of the game.  
The game begins with the password entry phase. If the correct password is entered, the system unlocks, and the player can start the game. If the password is incorrect, the system remains locked, preventing any inputs.  
Once logged in, the player selects a number using the RNG button. The player generates a second number, which determines the sum. If the sum matches 1111, the player earns a point. Otherwise, the round continues with new numbers.

A game timer (default: 99 seconds) controls gameplay, counting down while the player tries to complete as many successful rounds as possible. The timer remains at 00 until the password is verified, after which pressing the GAME button starts the countdown. Once time runs out, no further inputs are accepted. Pressing the GAME button again resets the timer to 99 seconds, allowing for a new session.

At the end of the game, a bonus scoring feature displays the total number of successful rounds on two 7-segment displays (in base-10). The scoring display is only active after the game ends.

If the user wishes to reset the game at any time, they can press the reset button, which clears all progress and returns the system to the login phase.

# System Architecture Design

Shown below in Figure 2 is the system architectural design, which shows the modules, input signals, and output signals.

A computer screen shot of a computer screen

AI-generated content may be incorrect.

Figure 2. Top-level System Architectural Design

The top-level system module connects the adder, decoder, verification, button shaper, load register, and access controller modules to perform the game’s functions. The input signals are player 1 and 2’s inputs, which are 4-bit signals, and player 1’s, player 2’s, password/ log out’s button, and reset button, which are 1-bit signals. The 7-bit output signals are the displays of player 1, player 2, the sum. The 1-bit output signals are the matching, non-matching, logged-in, and logged-out LEDs.

# Digit Timer

A diagram of a diagram

AI-generated content may be incorrect.

Figure 3. Digit Timer

The digitTimer module implements a two-digit countdown timer using two dt submodules to represent the ones and tens places. It operates based on a one-second timer pulse generated by the oneSecTimer module. The timer starts decrementing when timer\_enable is asserted, counting down from 99 to 00. The dt module handles individual digit decrements, borrowing from the next digit when necessary. If the countdown reaches zero, the timeout signal is asserted. The timer can be reset via rst or reconfigured to restart at 99 using reconfig. The input signals include clk, rst, timer\_enable, and reconfig, while the output signals are ones, tens, and timeout.

# RNG and RNG Latch

A diagram of a computer

AI-generated content may be incorrect.

Figure 4. RNG and RNG Latch

The rng module generates a 4-bit random number using a counter-based approach. The random number updates when rng\_gen is toggled, with an inverted signal (inv\_in) controlling the counter module. The rng\_latch module ensures stability by capturing and storing the random number only when the rng\_button is released (falling edge detection). This prevents rapid changes and ensures a consistent output. The rst signal resets the stored value to 0000. The input signals for the system include rng\_gen, rng\_button, clk, rst, and rng\_value, while the output signal is stable\_rng\_value, representing the latched random number.

# MUX 3 to 2

# A diagram of a graph AI-generated content may be incorrect.

Figure 5. MUX 3 to 2

The mux3to2 module is a multiplexer that selects between two sets of 4-bit values based on the gameover\_s signal. If gameover\_s is 1 (game over), the module outputs score\_ones and score\_tens, representing the final score. Otherwise, it outputs sum and sets tens\_out to 0. This allows dynamic switching between the current game sum and the final stored score. The module takes sum, score\_ones, score\_tens, and gameover\_s as inputs and produces out and tens\_out as outputs.

# Score

A graph with a rectangular object and text

AI-generated content may be incorrect.

Figure 6. Score

The score module extracts the ones and tens digits from a 7-bit win count (ranging from 0 to 99). It computes the tens digit by performing integer division by 10 and the ones digit using the modulus operation. This allows the win\_count value to be split into two separate 4-bit outputs, making it easier to display or process. The module takes win\_count as an input and outputs ones and tens.

# One Second Timer

A diagram of a computer program

AI-generated content may be incorrect.

Figure 7. One Second Timer

The oneSecTimer module generates a one-second timeout pulse by cascading three counters: timer\_1ms, count\_to\_100, and count\_to\_10. The timer\_1ms module produces a 1 ms pulse by counting 50,000 clock cycles (assuming a 50 MHz clock). This pulse feeds into the count\_to\_100 module, which counts 100 pulses to generate a 100 ms timeout signal. Finally, the count\_to\_10 module accumulates ten 100 ms pulses to produce a 1-second timeout pulse. The module resets when rst is asserted and operates when enable is active. The input signals include clk, rst, and enable, while the output signal is one\_sec\_timeout, indicating the completion of a one-second interval.

# Access Controller

A diagram of a computer

AI-generated content may be incorrect.

Figure 8. Access Controller

A diagram of a diagram

AI-generated content may be incorrect.

Figure 9. Finite State Machine for Access Controller

The Access module manages user authentication and system control by processing a four-digit passcode and handling various states for gameplay and reconfiguration. It transitions through multiple states to verify the entered passcode, granting access if the correct sequence (9-8-6-1) is entered. Once verified, the module enables system reconfiguration, gameplay activation, and win tracking. It controls various system components, including a random number generator (rng\_gen), a timer (timer\_enable), and indicators (loginLED, logoutLED, gameover). If an incorrect passcode is entered, the module resets to the initial state. Additionally, it monitors gameplay, detecting timeouts and tracking the number of wins (win\_count). The input signals are PassDigit, PassEnter, Load\_P1\_In, rng\_button, timeout, clk, rst, and win, while the output signals include Load\_P1\_Out, rng\_gen, timer\_enable, reconfig, logoutLED, loginLED, and gameover.

# Button Shaper

A diagram of a button

AI-generated content may be incorrect.

Figure 10. Button Shaper

A drawing of a castle

AI-generated content may be incorrect.

Figure 11. Expected Waveform for Button Shaper

A diagram of a rabbit

AI-generated content may be incorrect.

Figure 12. Finite State Machine drawing for Button Shaper

The buttonShaper module converts a button input into a predictable digital signal. It uses a finite state machine with three states: init, pulse, and wait\_s. When the button is pressed, the module outputs a clean pulse signal on b\_out. The input signals are b\_in, clk, and rst, and the output signal is b\_out.

# Load Register

A diagram of a computer

AI-generated content may be incorrect.

Figure 13. Load Register

The LoadRegister module is a 4-bit register that stores data from the input D\_in until a button press. It updates its output D\_out on the rising edge of the clock when the Load signal is high. If the reset (rst) is low, the register is cleared to 0000. The input signals are D\_in, clk, rst, and Load, and the output signal is D\_out.

# Adder

A diagram of a rectangular object with arrows

AI-generated content may be incorrect.

Figure 14. Adder Module

The adder module takes two 4-bit inputs and performs a logical AND, outputting a 4-bit output. The adder module also outputs two 1-bit outputs for the matching and non-matching LEDs. The signal names for the inputs are: “num1” and “num2”, and the output names are: “out”, “led\_match\_on”, and “led\_match\_off”, respectively.

# Seven Segment Decoder

A rectangular rectangle with arrows and letters on a grid

AI-generated content may be incorrect.

Figure 15. Seven Segment Decoder

The sevenSegDecoder\_X module takes a 4-bit input signal and decodes it to be able to be read by the seven-segment display in the form of a 7-bit output signal. The input signal is “decode\_in” and the output signal is “decode\_out”.

# Is15 Verification Module

A diagram of a game

AI-generated content may be incorrect.

Figure 16. Is15 Verification Module

The Is15 module is a verification module that takes a 4-bit input from the Adder module and determines if it is 15 or not. It outputs two signals , “led\_match\_on” and “led\_match\_off”, which are 1-bit signals to turn on/off the LEDs.

# Simulation Results

A screen shot of a computer

AI-generated content may be incorrect.

Figure 17. Simulation for Access Controller

The testbench verifies the Access Controller by testing correct and faulty password entries, login, and logout functions. It begins with a reset, then tests a correct 9-8-6-1 password entry, confirming successful login by asserting Load\_P1\_In. Logging out is simulated with LogOut\_Input, followed by Load\_P2\_In to test another login attempt. After a reset, a second test introduces an incorrect second digit to confirm access denial. The testbench ensures correct password handling, proper logout behavior, and system reset functionality.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 18. Simulation for Button Shaper

The testbench initializes the ButtonShaper module by setting up the clock, reset, and button input signals. It begins with a reset sequence to ensure a known initial state. The first test simulates a long button press by setting b\_in low for multiple clock cycles before releasing it back to high. After a delay, the button is pressed and released again to verify consistent behavior. This test ensures that the ButtonShaper correctly processes and stabilizes button signals, effectively handling debounce effects.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 19. Load Register Simulation

The testbench initializes the LoadRegister module, generating a clock signal and setting up the reset and load controls. It begins by asserting a reset to ensure a known initial state before releasing it. The first test sets D\_in to 0001 and enables the load signal to store the value in the register, then disables load to hold the value. Next, D\_in is updated to 1111 without loading to confirm that the register retains its previous value. Finally, D\_in is set to 0111, and after enabling load, the new value is stored in the register. This verifies that the register correctly loads and holds data as expected.

A screen shot of a computer

Description automatically generated

Figure 20. Adder Simulation

The testbench tested two matching cases, 1 non-matching case, and 1 overflow case (non-matching). I tested 10+5 & 15 + 0, these both resulted in 1111 (15). I also tested 2 + 3, which resulted in 0101 (5). Lastly, I tested 15 + 1, which resulted in 0000 (0).   
In cases where there was a match (1111), the matching LED signal was set to ON and the non-matching LED was set to OFF. When there was not a match, the matching LED signal was set to OFF and the non-matching LED was set to ON.These results were expected and correct.

For Figure 4 below, the simulation for the sevenSegDecoder is shown.

A screen shot of a computer

Description automatically generated

Figure 21. Simulation for sevenSegDecoder

The testbench was simple and tested for each input and output its corresponding 7-segment signal. All outputs were expected and correct.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 22. Simulation for RNG

The testbench initializes the rng module, generating a clock signal and controlling the reset and in signals. It starts by asserting a reset to initialize the counter. The first test sets in = 1, preventing the counter from incrementing. Then, in = 0 allows the counter to increment for two clock cycles, after which in is set back to 1, stopping the counter. Finally, in is set to 0 again, and the counter resumes incrementing. This verifies the counter’s behavior based on the in signal and the reset functionality.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 23. Simulation for Digit Timer

The testbench initializes the digitTimer module by generating a clock signal and controlling the reset, timer enable, and reconfig signals. Initially, the reset (rst = 1) is asserted, ensuring a known state. After 20 ns, the reset is deasserted (rst = 0) to allow the timer to function. Then, the reset is asserted again for 30 ns before being deasserted once more. After 50 ns, the timer\_enable signal is set to 1, activating the timer. The testbench runs for 5000 ns to observe the behavior of the timer and the output signals (ones, tens, and timeout). The ones and tens digits should decrement over time, and the timeout signal will be set based on the conditions defined in the digitTimer and dt modules.

This simulation verifies the operation of the digitTimer module, including proper digit counting, timeout behavior, and response to reset and timer enable signals.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 24. 1 Second Timer Simulation

The testbench initializes the oneSecTimer module, generating a clock signal and controlling the reset (rst) and enable signals. Initially, the reset (rst = 1) is asserted to ensure a known state. After 20 ns, the reset is deasserted (rst = 0), and after another 20 ns, it is asserted again. Then, a 10 ns delay is followed by enabling the timer (enable = 1) for 24000 ns, allowing the timer to run and generate the one\_sec\_timeout signal.

The oneSecTimer module generates a 1-second timeout using a combination of smaller timers:

1. timer\_1ms generates a 1-millisecond timeout using a counter that counts up to 3 (instead of 49999 in the original version) for easier simulation.
2. count\_to\_100 counts 100 milliseconds, now using a counter that counts up to 3 (instead of 99) to simplify the simulation.
3. count\_to\_10 counts 1 second, using a counter that counts up to 3 (instead of 9) to make the simulation easier to observe.

The testbench verifies that the one\_sec\_timeout signal is correctly asserted after 1 second when the enable signal is active. This setup ensures that the smaller timers work together to generate the 1-second timeout in a simplified manner for simulation.

This simulation checks the behavior of the oneSecTimer module and confirms that it produces a 1-second timeout after the enable signal is asserted, with the threshold values lowered for more manageable simulation times.

# FPGA Board Testing Results

A close up of a circuit board

AI-generated content may be incorrect.

Figure 25. Default State / Reset State

This shows the board when it is first powered on and when it is reset.

A circuit board with red and blue lights

AI-generated content may be incorrect.

Figure 26. Logged In State

This shows the board whenever the player successfully logs in, as shown by the LED on the right turning on.

A close up of a circuit board

AI-generated content may be incorrect.

Figure 27. Matching Case

Player 1 inputs a binary 10 (A) and player 2 inputs a binary 5. This adds to 15 (1111 or F) and the matching LED lights up. This functions as expected.

A close up of a circuit board

AI-generated content may be incorrect.

Figure 28. Non-matching Case

Player 1 inputs a binary 1 and player 2 inputs a binary 1. This adds to 2 and the non-matching LED lights up. This functions as expected.

A close up of a circuit board

AI-generated content may be incorrect.

Figure 29. Game over showing score

Shows state of FPGA after game has ended and number of wins.

# Video Demo

[gameplay.mov](https://uofh-my.sharepoint.com/:v:/g/personal/thvo20_cougarnet_uh_edu/EabZ4nVSha1OmAFX2NwxCd8Bxau_BI1leKZ7ALCw-B13nQ?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=xpr2kZ)

This shows gameplay after being logged in and demonstration of the load registers.

[reset.MOV](https://uofh-my.sharepoint.com/:v:/g/personal/thvo20_cougarnet_uh_edu/EUvVf7o0ni9IiyHP5n7GoT8BxTeFGC1t44zxyIBsIVeaDQ?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=dciY70)  
This shows how the reset button operates.

[password entry to log in.MOV](https://uofh-my.sharepoint.com/:v:/g/personal/thvo20_cougarnet_uh_edu/ESohESCMr_hNjkWBNqz40pkBXSHt5wXHmWcWX4r2jtxDcA?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=yeFhCU)

This shows logging in for the first time to operate the game.

[reconfigure.MOV](https://uofh-my.sharepoint.com/:v:/g/personal/thvo20_cougarnet_uh_edu/EQcqxK2qs5lBlKhZv6U9iVoBxehUA0iBX3TVI9LfRYoYHQ?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=PZJpzy)

This shows the reconfigure feature

# Conclusion

I have successfully built and completed lab 3, which was implementing a mental binary math game on an FPGA with load registers, button shapers, and access controller modules. The bonus features implemented were the matching, non-matching LEDs, and the automatic scoring.

# Appendix

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 30. Top Level Module Code

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 31. Access Controller Code 1/3

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 32. Access Controller Code 2/3

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 33. Access Controller Code 3/3

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 34. Button Shaper Code

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 35. Button Shaper Testbench Code

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 36. Load Register Code

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 37. Load Register Testbench Code 1/2

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 38. Load Register Testbench Code 2/2

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 39. Is15 Verification Module Code

A screen shot of a computer program

Description automatically generated

Figure 40. Adder Module Code

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 41. Adder Testbench Code

A computer screen shot of a program

Description automatically generated

Figure 42. Code for sevenSegDecoder Module

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 43. Seven Segment Decoder Testbench Code

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 44. oneSecTimer, sub Module Code, and testbench code

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 45. RNG, sub module, and testbench code

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 46. digitTimer, sub Module, and testbench code

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 47. Score Module Code

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 48. Mux 3 to 2 Code