David Zhang

2213233

Lab 2: Game access control on fpga

ECE 6370

# Introduction

This mental binary math game is a two-player game, where one player tries to set a number from 0 to A for the other player to match, using the number’s binary representation on 4 slide switches. The objective of the other player is match the first players number such that the addition of their two numbers equals the maximum 4-bit hex number, A. Additionally, this lab implements a feature to prevent peaking at the other players inputs—previously, since numbers were displayed instantaneously as soon as the switches were set, the second player could look at the first players switches and set theirs to the complement, ensuring a win. This is prevented through the additions of separate load registers for each player, which will load in the number to be displayed when each players push button is depressed. As well, a master control module has been added that allows players to log in using a set password that will prevent player inputs until verified.

# System Architecture Design

Diagram, schematic

Description automatically generated

Figure 1: Overall system architecture

The master drawing for the system architecture is shown above in Figure 1. While complex, there are only 5 unique modules that exist while the rest remains duplicates: Adder, 7-segment Display, Access Controller, ButtonShaper, and LOAD.

Adder (represented above symbolically by Adder) is a combinational module that takes in two 4-bit inputs (p1In, p2In) and adds them together to create another 4-bit output (AdderOut). This module cannot do carryover sums, so if the resultant sum of the inputs creates an overflow the sum will simply be rolled over.

7-Segment Display (represented as 7SD) is a combinational module that takes in a 4-bit input (decoderin) and creates a 7-bit output (decoderout) that represents the bits necessary to light up a 7-segment display as the original input’s decimal representation.

Access Controller (represented as Access Controller) is a sequential and combinational module that takes in three 1-bit signals (p1loadIn, p2loadIn, passButton), one 4-bit signal (passwordIn), and outputs four 1-bit signals (p1loadOut, p2loadOut, logIn, logOut). This module uses a finite state machine to create the log-in process for players. This FSM is shown below in Figure 2.

Diagram

Description automatically generated

Figure 2: Access controller fsm

To summarize what happens in the finite state machine, the inputs to the LOAD registers are blocked by the Access Controller until it is in the OPERATIONAL state. The only way to reach the OPERATIONAL state is to input the correct sequence of numbers on player 2’s switches followed by pushing the password button each time. The password itself is arbitrary, but for simplicity the last 4 digits of the student’s ID is used—in this case, being 3-2-3-3. If all 4 digits are correct, the internal signal passRight remains as 1, and allows the VERIFY state to move to the OPERATIONAL state, after which the game can be played normally. In addition, another push of the password button logs out the player, moving the FSM back to its original state where it waits for the correct password input.

ButtonShaper (represented in Figure 1 as Button Shaper) is a sequential module that takes in a 1-bit input (D) and outputs a 1-bit, one clock cycle pulse (Q). Since the actual push button’s on the FPGA register a signal low when depressed, the ButtonShaper module corrects this into a waveform with signal high instead. In addition, the module creates a finite state machine that ensures that any input, no matter how long, can only create a 1-cycle output until a new input is detected. The state machine as well as conceptual timing diagram for this module is shown below in Figure 3.

Diagram

Description automatically generated

Figure 3: Architecture of buttonshaper

LOAD (represented in Figure 1 by LOAD R1/2) is a sequential module that takes in a 1-bit input (p1loadOut), a 4-bit input (slide switches from players) and outputs a 4-bit signal (decoderin). This module is responsible for loading the inputs of each player’s switches to their respective display when the button is pushed. However, the controlling input (p1loadOut) is only enabled when the Access Controller has verified a correct log in sequence. This way, players cannot play the game before logging in.

As a corollary, all sequential modules have two additional 1-bit inputs: reset and clock. However, as these are signals universal to any sequential module they were not deemed as unique inputs that needed to be mentioned for each one.

# Simulation Results

For the sake of organization, all resultant waveforms from testbenches have been included below rather than listed in their respective subsections above to keep visual clarity. These waveforms are shown in Figure 4, 5 and 6.

A screenshot of a computer

Description automatically generated

Figure 4: Output waveform of buttonshaper

Figure 4 details the results of the ButtonShaper module when two different inputs are tested. From figure 3, we should expect the module to convert any signal low input of varying length into a single clock cycle pulse of signal high at the output. We can see that both input signals of length ~4 clock cycles and ~2 cycles, respectively, produce a correct output of only one clock cycle. Thus, this module works as intended.

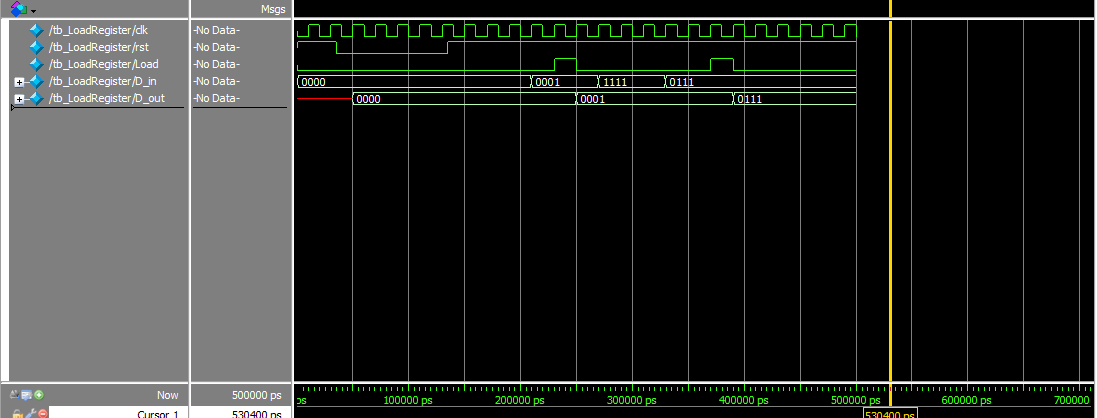


Figure 5: output waveform of LOAD

Figure 5 shows the resultant waveform of the LOAD module with the testing of three inputs. In the first input of decimal 1, the load button is pressed and the input goes to the output, as expected. In the second input, the button is not pressed and hence an input of decimal 15 is not passed to the output, and the output stays the same. In the last input, the button is pressed with a new input of decimal 7 and the new output becomes 7, as expected. Thus, this module works as intended and only updates the output when the load button is pressed.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 6: output waveform of access controller

The Access Controller has a much more complicated testbench, however the most important part is at the marker depicted in yellow. As seen in Figure 2, this module functions as a log-in procedure where the players cannot input numbers until the correct 4-digit string of numbers is passed in, after which the operational state is verified by the signals loggedIn going to 1 and loggedOut going to 0. The testbench above was written to test first an incorrect sequence of digits, followed by the correct sequence of digits. This can be visually seen by each time passButton\_s goes high, indicating that a digit has been received by the module. The marker in yellow shows the time that the 8th digit is received, or in other words the last digit of the correct password. 2 clock cycles after the marker, the system enters the OPERATIONAL state, which is as expected as it first needs to pass through the VERIFY state.

The module can be confirmed to be working as intended through observing the p2loadOut\_s signal and p2loadIn\_s signal. As can be seen, p2loadIn\_s is tested once when the module is not in the OPERATIONAL state and once again when the module is. Only when the system is OPERATIONAL does p2loadOut\_s go high immediately after p1loadIn\_s, indicating that the output is correctly blocked until the system is logged in.

# FPGA Board Testing Results

Captures of the FPGA working in various important steps are shown below. However, as many of these steps are dynamic and unable to be captured through a static medium like photos, the video demo will clarify in more detail how each step should look like. The pictures below are simply to provide a guideline for what the board should look like at various steps.

A close up of a keyboard

Description automatically generated with low confidence

Figure : Initial board state upon powering on

A close up of a keyboard

Description automatically generated with low confidence

Figure : Board state after correct password has been entered

A picture containing text

Description automatically generated

Figure : Example correct game

A close up of a keyboard

Description automatically generated with medium confidence

Figure : Board state upon logging out

The figures above represent a typical operation of the game. Figure 7 and 8 show what should happen between the initial power on state and correct login, where the loggedIn LED should be lit. Figure 9 and 10 show a typical game played between two players, and the implementation of the logout feature where the logout LED is on and players cannot input new numbers.

<https://drive.google.com/file/d/1tyE6avN37Vs4MWAF24cnOQ-sHxfam0mj/view?usp=sharing>

The video demo above demonstrates the correct log in process. 3-2-3-3 is input on player 2’s switches followed by successive password button presses until the loggedIn LED is lit.

<https://drive.google.com/file/d/1XAfMtYzItR-fPPnDs9SL2p8j5ROSeX13/view?usp=sharing>

The video demo above showcases the utilization of player load buttons for inputs. Numbers are not displayed on each players display until their load button has been pressed, at which time it will display the current binary number set.

<https://drive.google.com/file/d/1QKFH3ar99c66onecK7tQskuv46jzDr2K/view?usp=sharing>

The video demo above demonstrates the logout feature. After any point in time after logging in, players can press the password button again to logout, shown visually as the loggedout LED being lit. Now, any further inputs from either player are blocked and not shown on the display until the correct password has been re-entered from player 2’s input switches.

# Conclusion

This mental binary based math game represents a marked improvement over the previous iteration that now includes a feature to prevent cheating as well as an early prototype for logging in that could be built upon for future features. One problem right now, however, is that when entering the password to log in the 7-segment displays do not show the number being input, meaning that players must already know the binary representation for the correct decimal digit. This is a problem that cannot be easily circumvented, as the way the architecture is designed there is no direct path from input switches to decoders—if there were, there would be no way to load them to registers, or at least no point.

Players can press the password entry button at any time during their game to log out and restart the login process, however as it is the displays keep their current entries without being erased back to 0. This bug may be an intended feature however, as now it is possible to record entries after logging out for the purpose of keeping score, for example.

As it is now, the lab has met all requirements and functions as intended. The two bugs mentioned above are expected and will most likely have to be addressed in later issues of this game.

# Appendix A: Module Code for ButtonShaper

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Figure 11: Module definition for buttonshaper

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application, Word

Description automatically generated

Figure 12: testbench of buttonshaper

# Appendix B: Module Code for LOAD

A picture containing text

Description automatically generated

Figure 13: Module definition for load

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Figure 14: testbench definition for load

# Appendix C: Module Code for Access Controller

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Figure 15: Module definition for access controller

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Figure 16: Testbench definition for Access controller

# Appendix D: Module Code for Lab 2

Text

Description automatically generated

Figure : Module definition for lab 2

# Appendix E: Module Code for 7-Segment Display and Adder

Graphical user interface, text, application

Description automatically generated

Figure 18: module definition for adder

Graphical user interface, text, application

Description automatically generated

Figure 19: Testbench definition for adder

Graphical user interface, text, application

Description automatically generated

Figure 20: module definition for decoder

Graphical user interface, text, application

Description automatically generated

Figure 21: testbench definition for decoder