Lab 2: Game Access Control on FPGA

Stefan Bucur 3153

ECE5440

Contents

[**1.** Introduction 3](#_Toc32854910)

[**2.** System Architecture and Design 4](#_Toc32854911)

[2.A: Button Shaper (bshaper) 5](#_Toc32854912)

[2.B: Load Register (loadreg) 6](#_Toc32854913)

[2.C: Access Control (access) 7](#_Toc32854914)

[2.D: Addition (adder) 8](#_Toc32854915)

[2.E: 4-to-7 Decoder (decoder7) 8](#_Toc32854916)

[2.F: Check Module (check) 8](#_Toc32854917)

[**3.** Simulations 8](#_Toc32854918)

# **1.** Introduction

Lab 2 is a math memorization game that tests players’ ability to add and convert between binary, decimal and hexadecimal in their minds. The digital logic described in this design was made for the Altera DE2-115 prototyping board. The development in this lab focuses on implementing a login system to the game and a more robust player input system. When the device is first powered on, the number displays and indication lights are visible, but the player(s) cannot immediately begin playing the game; they must first enter the correct sequence of four-bit numbers one by one into the login system. Once the correct sequence is read, the system will indicate the game can be played and players can begin entering numbers with their designated button and sets of switches. Players enter numbers into the system by toggling switches that represent a binary number and pressing their button. If the two numbers add to 0xF (15 in decimal), the FPGA will change the indicator LEDs to verify the correct result. Players record score using their own means and can take turns being the first player to enter a number each round. If the players would like to leave the board for any reason and not want others to alter the inputs, they can log out of the game and the system will preserve the present values should the players log in again. At any point during operation the reset button can be pressed, reverting the system back to its initial state as if it was powered on for the first time. In the modules with a state machine, the transition upon activating RESET is assumed to be its default state.

# **2.** System Architecture and Design

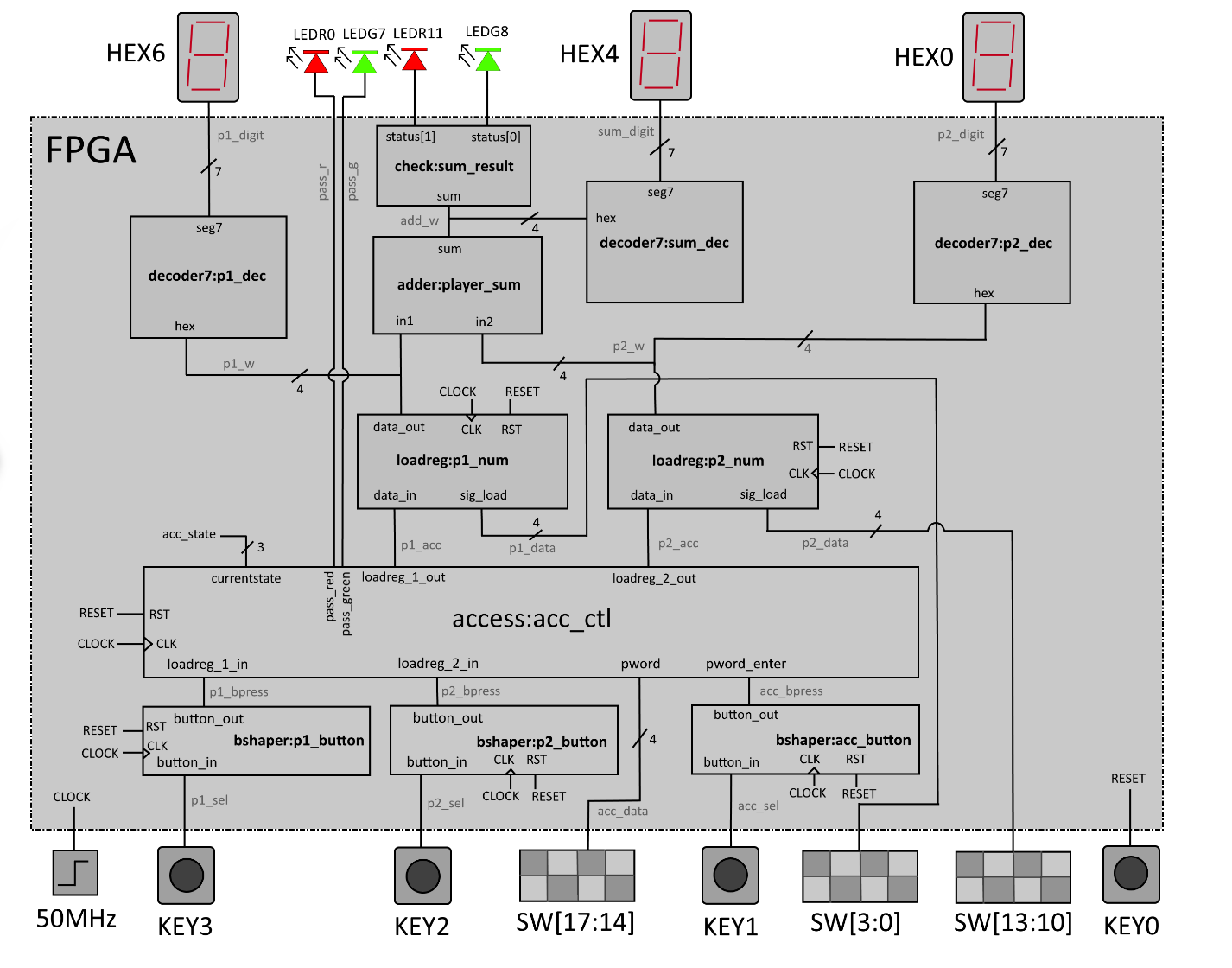


Figure : FPGA system architecture for Lab 2. KEY inputs pin out to the 4 onboard buttons. SW switches pin out to the onboard switches. A 50MHz clock is wired into the FPGA from an onboard oscillator.

|  |  |
| --- | --- |
| Input/output | Description |
| 50MHz | Onboard 50MHz clock from the DE2-115 board. Clock signal is synchronized with a PLL. |
| SW[17:14] | The four input switches for player 1 to enter a 4-bit value into the FPGA. |
| SW[13:10] | The four input switches for player 2 to enter a 4-bit value into the FPGA. |
| SW[3:0] | The four input switches to input a 4-bit value into the FPGA’s access control module. |
| KEY3 | Push button to instruct the system to load player 1’s value into their load register. |
| KEY2 | Push button to instruct the system to load player 2’s value into their load register. |
| KEY1 | Push button to input a password digit from SW[3:0] into the FPGA’s access control module. |
| KEY0 | Push button to reset all system modules to their default state. |
| HEX6 | Seven-segment display for player 1’s number within the FPGA. |
| HEX4 | Seven-segment display for player 2’s number within the FPGA. |
| HEX0 | Seven-segment display for the sum of the two player’s values within the FPGA. |
| LEDR0 | Red LED indicating the system is in logout mode and requires a password. |
| LEDG7 | Green LED indicating the system is in login mode and will accept player input. |
| LEDR11 | Red LED indicating the sum of the two player’s numbers in the system does not equal 15 (0xF). |
| LEDG8 | Green LED indicating the sum of the two player’s numbers in the system equals 15 (0xF). |

Table 1: short description of the I/O connected to the FPGA on the DE2-115 board.

The FPGA’s sequential logic modules use the DE2-115’s 50MHz internal clock signal to update on its rising edge signal. Apart from RESET, the other three input button signals are first passed through their own button shaper module. This is due to the internal state machines of both the Access Control and Load Register modules treating a button activation signal as a new button press upon each new clock cycle. In order to reduce the complexity of the state machine, the button shaper converts the button’s debounced active low signal into a positive logic pulse that lasts one clock cycle. RESET is not shaped because the sequential logic modules should ideally be subject to multiple subsequent reset signals to ensure the entire system is set to the same initial state. The acc\_state wire within the FPGA is left for debugging purposes in simulations. Explained further below is a detailed explanation of the functionality of each of the modules in Figure 1. Signal names within module borders are defined within the Verilog module. Wire names are defined in the top-level module.

## 2.A: Button Shaper (bshaper)

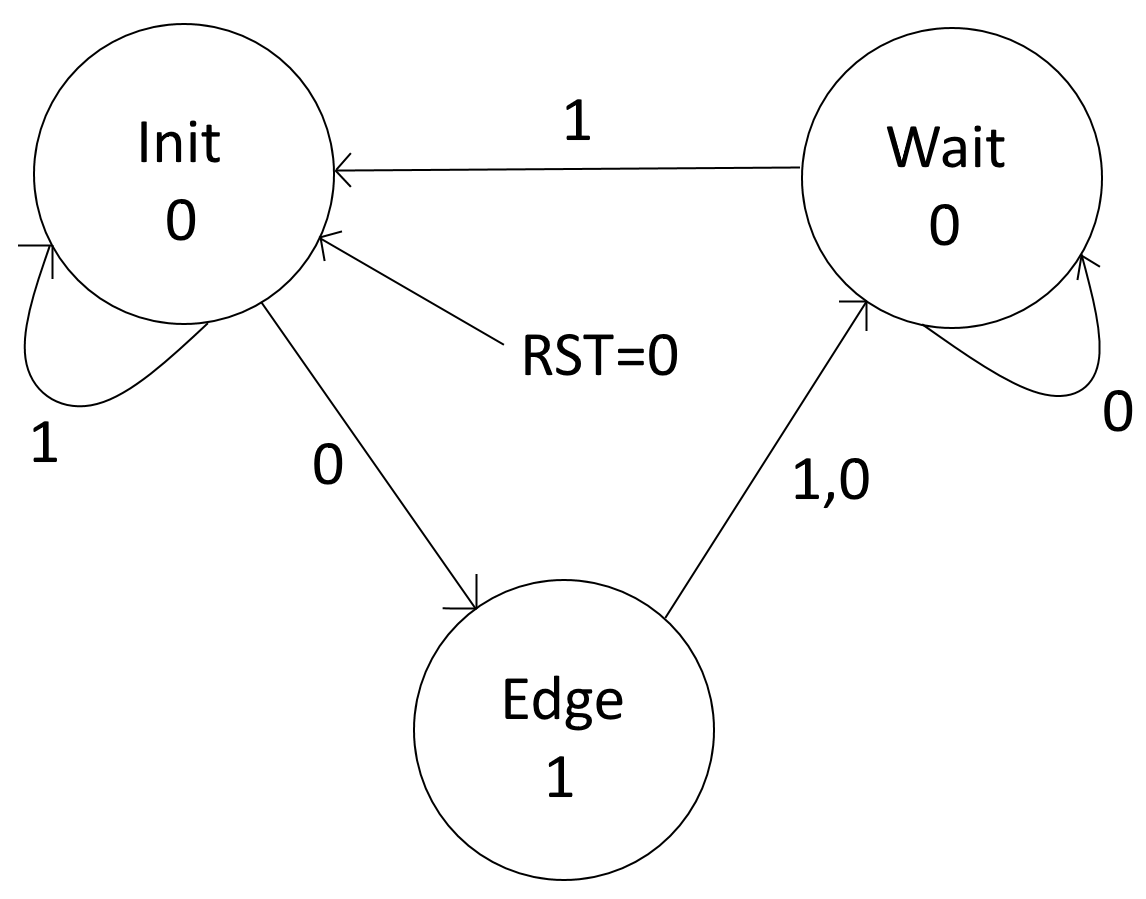


Figure : Finite state machine (FSM) for the button shaper. State values correspond to the button\_out signal. Transition arrow values correspond to the button\_in signal. RST = 1 does not affect the FSM and is therefore omitted.

The purpose of using the button shaper is to convert button inputs into signal events that are easier to process in sequential logic. By making the button press last one clock cycle at the start, the modules connected at the button shaper’s output will trigger an event at the next clock cycle without triggering another erroneously at the next cycle.

The button shaper module is a two-procedure finite state machine that converts an active low input signal lasting multiple clock cycles and converts it into an active high square pulse lasting only one clock cycle. The Init state is the default state, where it waits for the button active signal to change at which point it will transition to the edge state within that clock cycle. At the edge state, the output signal is set high and the state machine transitions to a wait state. Here it waits for the input signal to revert to its inactive value at which point the button shaper returns to the Init state. If RST is active when the button is held down, the state machine will ignore any button input until the reset is deactivated.

## 2.B: Load Register (loadreg)

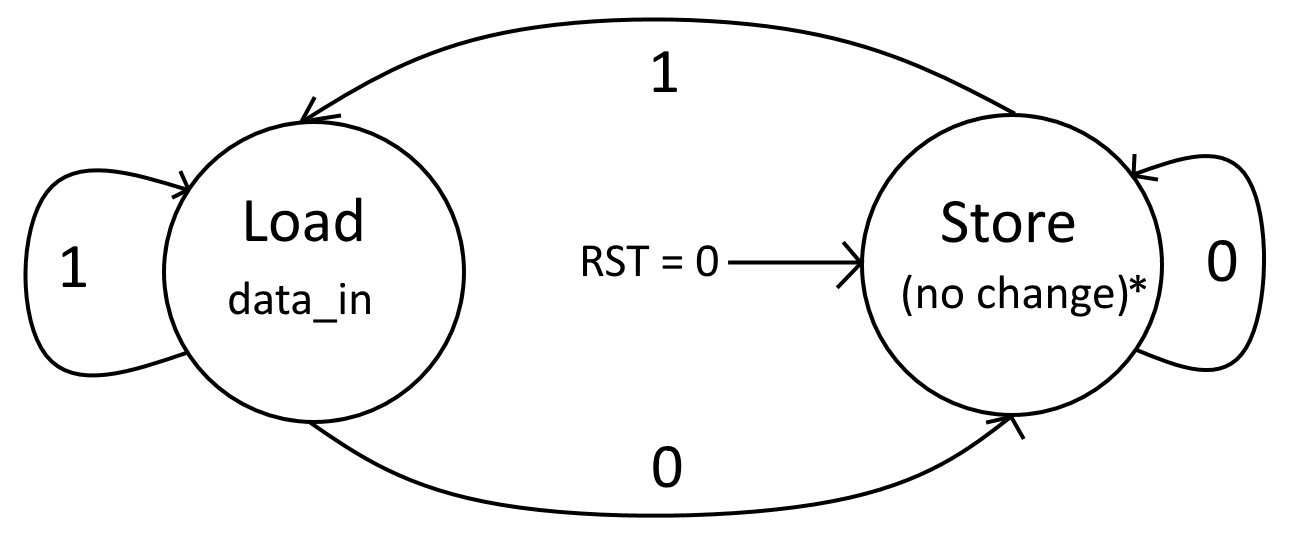


Figure 3: Finite state machine (FSM) for the load register module. State values correspond to the data\_out signal. Transition arrow values correspond to the sig\_load signal. RST = 1 does not affect the FSM and is therefore omitted.

The load register, while not being a requirement for proper operation, is implemented using a one-procedure finite state machine. There are only two states, where Store will keep the same output value unless it is reset, and upon receiving a high pulse from the button shaper will read from the player’s switch input (internally labelled data\_in) in the Load state. Note in Figure 1 that sig\_load is blocked by the access module and will only receive a signal if the access module passes it from the assigned button shaper as explained in section 2.C. Resetting the module with the RST signal will force the state machine to the Store state and overwrite the output to 0.

## 2.C: Access Control (access)

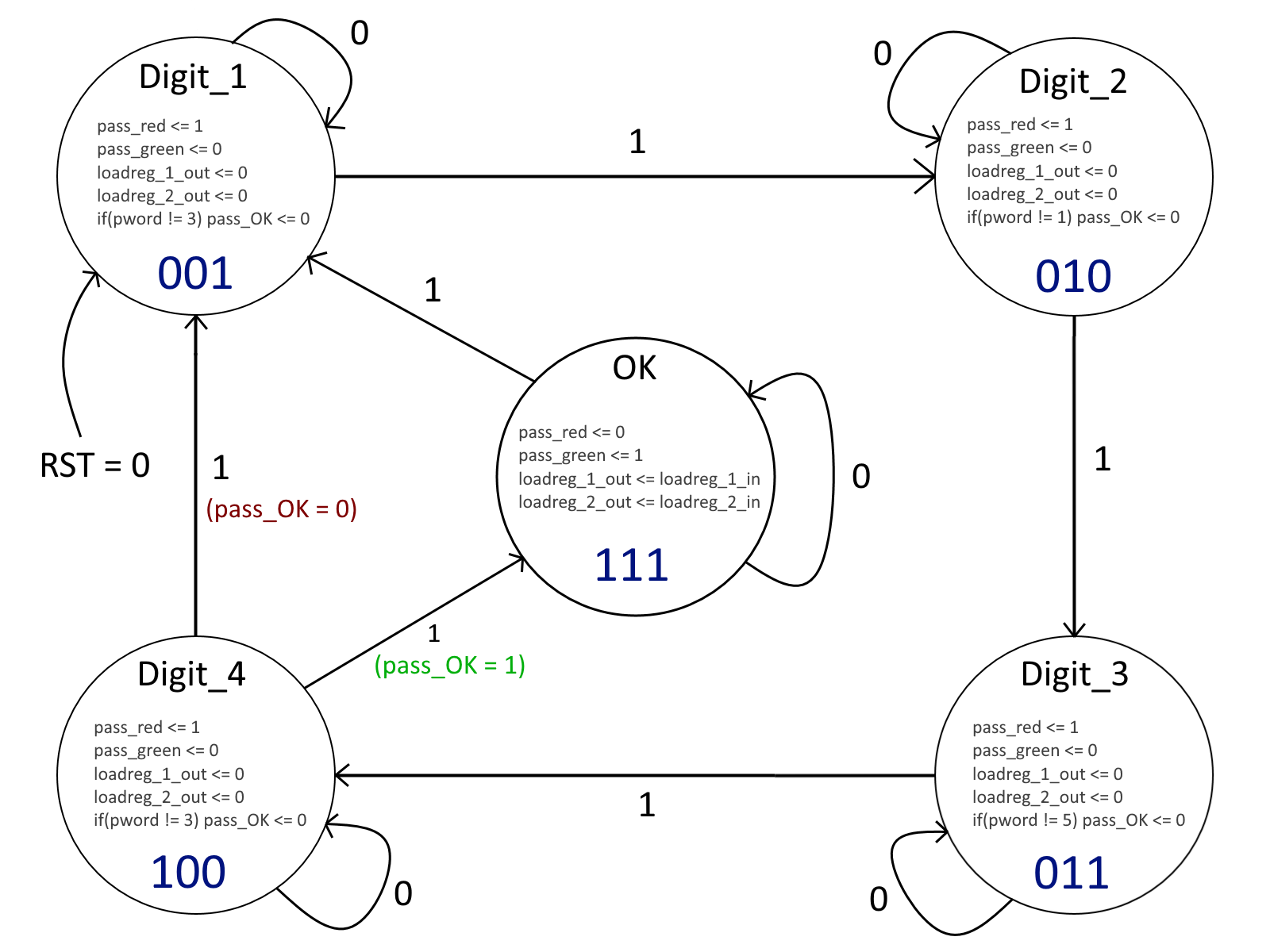


Figure 4: Finite state machine (FSM) for the access control module. Transition arrow values correspond to the pword\_enter signal. RST = 1 does not affect the FSM and is therefore omitted. The if statement within each state does not evaluate until the FSM changes the current state.

The access module is responsible for ensuring players can send button signals into the load registers only after the correct sequence is entered into the module via the pword 4-bit input and the pword\_enter signal which is first passed through a button shaper. For this design, the sequence is **3 1 5 3**. The state names represent the number that the access module is asking for at that time. The 3-bit binary number at the bottom of each state is the numerical value corresponding to that state within the FSM and is stored as the variable currentstate. The module is configured to allow currentstate to be wired to an output pin, but for this design it is intentionally left out, as there already exists a pair of indicator lights (pass\_r for a red LED and pass\_g for green) that show when the player(s) are logged out (only red on) or logged in (only green on). The module still has it available as an output for either future use in other designs or debugging within simulations.

When pword\_enter is 1, the if statement in each state circle will check if the correct number is present within pword. If incorrect, the 1-bit value pass\_OK acts as a flag for the state machine to indicate that a number within the sequence is incorrect. The module then transitions to the next Digit state. Digit\_4 not only evaluates the last digit, but the next state it transitions to depends on the final value of pass\_OK. If pass\_OK has not been altered from its default value of 1, the state machine is set to an OK state where the button signals from the player buttons are allowed through the module and carried into the load register’s sig\_load input each clock cycle.

It is important to note that if an incorrect number is entered, the state machine does not return to Digit\_1 and carries on asking for the next digit in the sequence. Meaning if the 2nd digit is incorrect and the player tries to enter the sequence again from its current state, the access module will not go to the OK state. The player can either press the KEY1 button until the module returns to the Digit\_1 state or press the RESET button.

The access module features a log out system so that players can prevent unwanted changes to the system and preserve the current load register values during that time. When in the OK state, pressing KEY1 again will return to the Digit\_1 state. Pressing RESET does the same action from any state, however it will reset other sequential modules to their respective default values and should only be used if any of the saved value.

## 2.D: Addition (adder)

The system uses a 4-bit adder to sum up the two player numbers and output the result. In this system the in1 and in2 values are wired to the output of the load register modules and update as soon as one of the input signals change. The sum is wired to a seven-segment display decoder to present it to the user.

## 2.E: 4-to-7 Decoder (decoder7)

In order to display numerical values in a clear and readable manner on the DE2-115 board, the FPGA must first convert the number into a 7-bit value to output to one of the seven-segment displays available on the board. The decoder7 module converts a 4-bit value and interprets it as a hexadecimal digit and output the matching 7-bit value that displays the digit in standard hex format.

## 2.F: Check Module (check)

The system will turn on a red or green LED if the sum of the two player’s numbers is or is not 15, respectively. The check module takes the output of the sum module and checks if the output is equal to 0xF and turns on the proper LED upon receiving the sum value.

# **3.** Simulations

Described below are the system modules running in a simulation testbench. Note that some modules are not tested for every possible value, however edge cases were carefully chosen to reflect real behavior. All simulations of the system modules are done in Altera ModelSim.

## 3.A: Button Shaper (bshaper)

## 3.B: Load Register (loadreg)

## 3.C: Access Control (access)

## 3.D: Addition (adder)

## 3.E: 4-to-7 Decoder (decoder7)

## 3.F: Check Module (check)