数字逻辑设计 2019/2020：Final Project

王跃明老师

洪奇军老师

Game Design: Finger Dancer

2020-01-20

Anna Tang 3180300155

Joshua Malmberg 3180300189

Chen Yi Hui 3180300684

Justin Choi 3180300160

1. Abstract

The following report presents the final project for the course “Digital Logic Design”, taken in the fall and winter semester of 2019-2020. We chose to create a finger dancing game, where the player needs to recreate the pattern generated by the circuit in a given time frame. The game is implemented on the SWORD board, and only requires the four component seven-segment display, switches, a button and the LED lights below the display. To reflect on what we have learned from the course, we have used a variety of different modules such as registers, counters, and frequency dividers. Proper memory and register access have also been applied here. Modules from previous labs such as the multiplexers, and full adder have been recycled into this project. We have also constructed our own input, display and compare modules from scratch. This report will summarize the game behavior, the design process, simulations and debugging process.

1. Introduction
2. Background

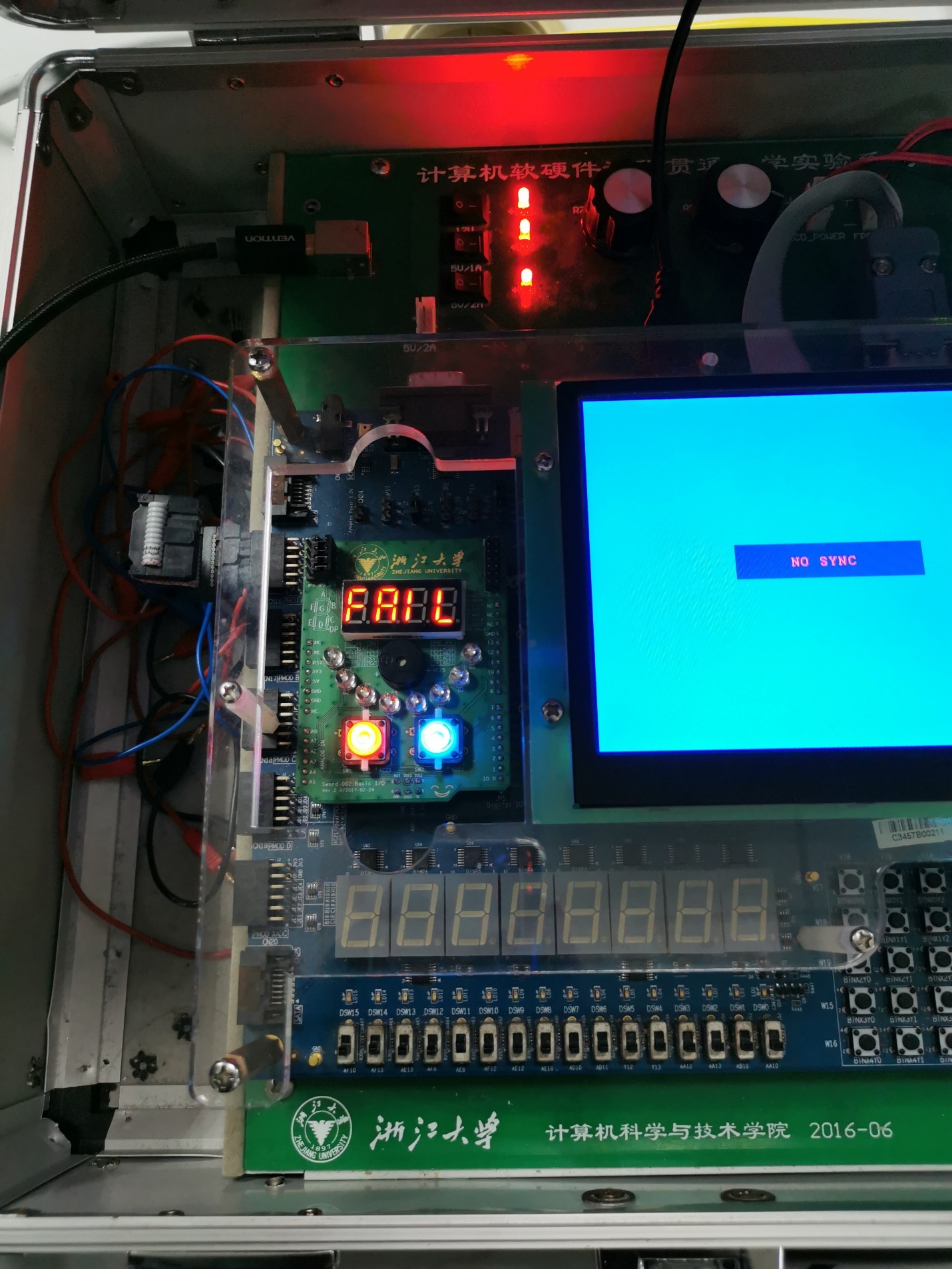
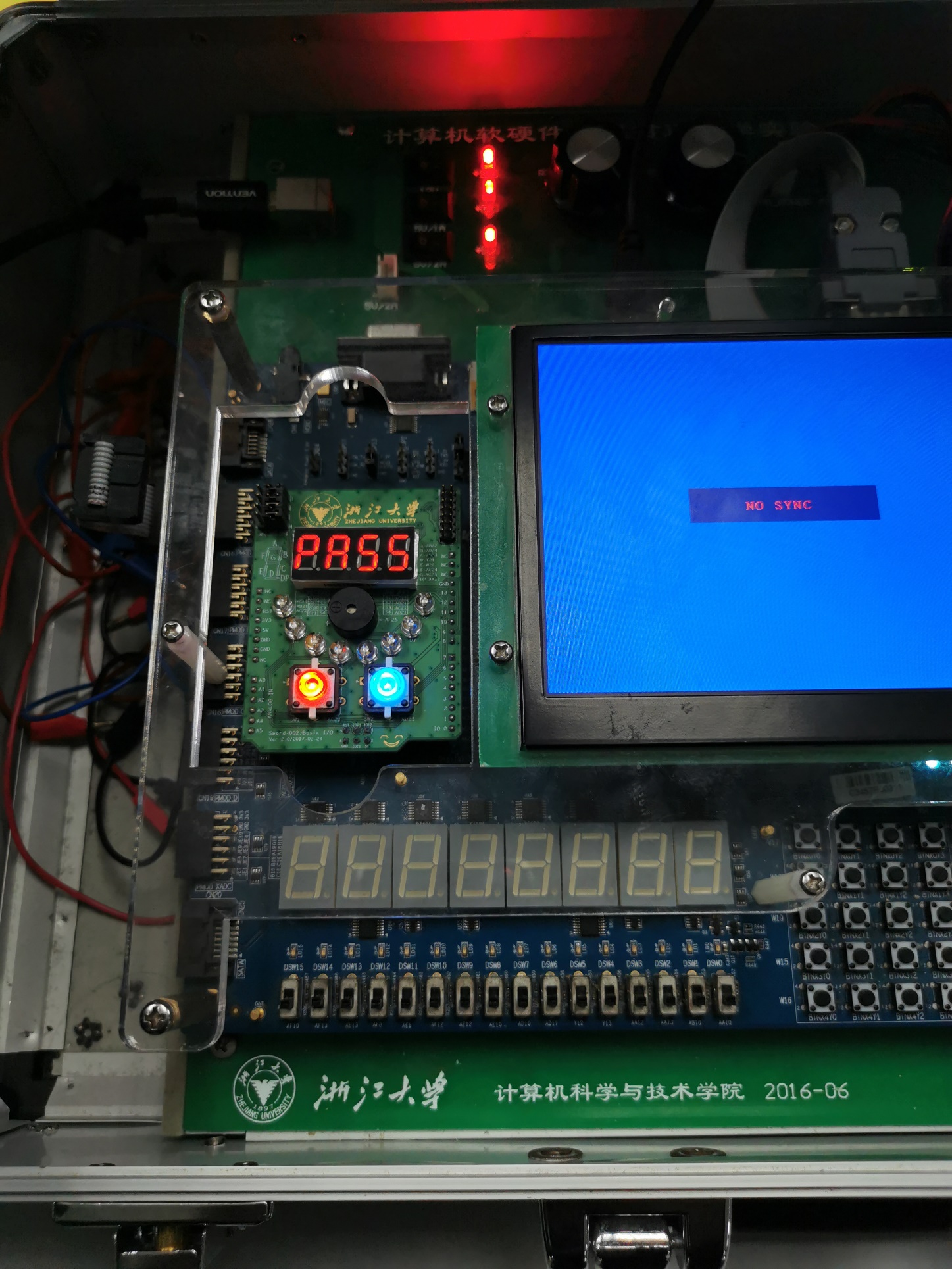
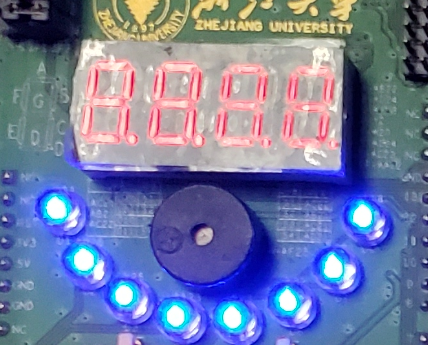
“Finger Dancer” is a simple game where the player is given a pattern of lights and must match it using the corresponding switches before the time runs out. If the player successfully completes the round, “PASS” will be displayed on the seven-segment display and the score increases. In the case that the player does match the pattern, “FAIL” will be displayed and automatically ends the game. The player has five seconds to match the pattern using the switches of the SWORD box. Failure to do so will result in an automatic defeat and reset of the game.

1. Purpose

The purpose of this final assignment is to make use of all the various skills and tools learned throughout this course and apply it to construct this game. This allows our team to practice working on the SWORD board and get a sense of what it is like to design practical circuits.

1. Instruments and Materials
2. PC with Xilinx ISE 14.7
3. SWORD Board with Kintex7 Chip
4. User Manual: How to Play Finger Dancer

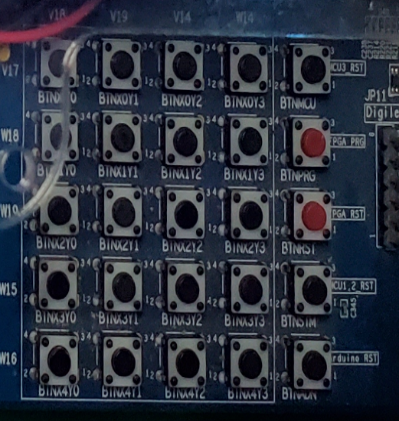
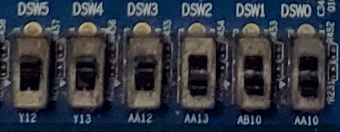
To start the game, press the button and the game will immediately begin. The basic idea of “Finger Dancer” is to match the pattern indicated on the LEDs below the seven-segment display, using the switches. Each round is timed for five seconds. At the end of a round, the circuit evaluates if the player has inputted the correct pattern. If the player successfully does so, “PASS” is indicated on the seven-segment display, and their score is increased. The game continues until the player fails to match the switches with the LEDs. Consequently, the game ends and “FAIL” is displayed. The score is reset, and the player may try again.

Pattern display

Score display

Figure – FAIL displayed Figure 2 – PASS displayed Figure 3 – Game UI

Input switches

Initialization button

Figure 4 – INIT button Figure 5 – Game controls

1. Implementation and Design
2. Circuit Design - Top

The circuit for “Finger Dancer” comprises of eight different modules, as depicted in the diagram. pbdebounce is an anti-jitter module for the button, which is an external input used to jumpstart the circuit. Note that INIT is not an input, but a signal used to initialize and reset the state of the game. userInput and compare are respectively responsible for taking in input from the switches and comparing them to see if they match the pattern generated by the circuit. patternUp generates the pattern, and its connected register patternReg stores it for display. The scoreUp module either increments or resets the score, and the connected register scoreReg stores it for display. The timing module provides the initialization to the pattern and score modules, right when the player presses the button. sec\_clk is the circuit’s clock, which provides clock pulses with one-second periods. Lastly, the display module outputs the generated pattern to the LEDs, and the score onto the seven-segment display.

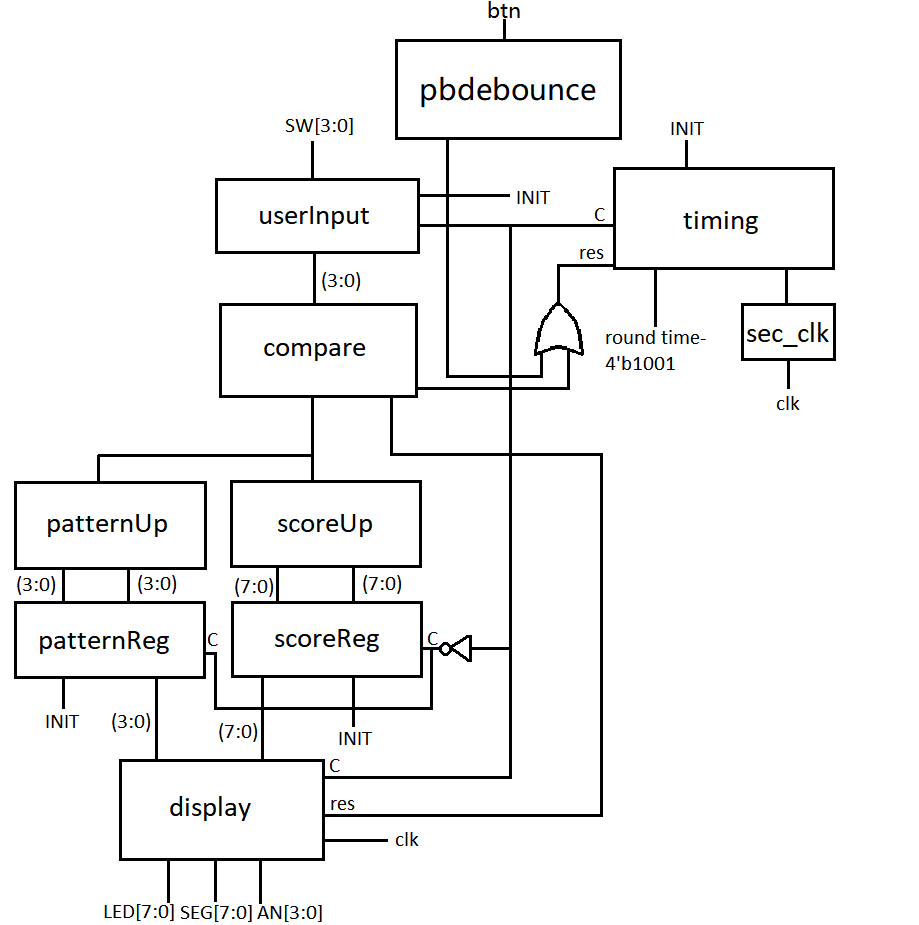


Figure 6 – Diagram of top module

1. Taking in User Input – userInput

The userInput module takes in the logical on/off values from the switches (I[3:0]) and returns it to the output O[3:0] for further processing in the compare4bit module. It inputs a clock cycle from the timing module (C), and an INIT signal from the pressed button. This module is comprised of a single 4-bit register.

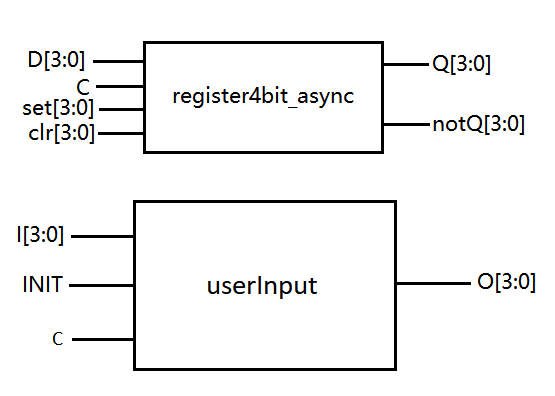


Figure 7 – Input module

1. Comparing Input – compare4bit

The compare4bit module takes in two 1-bit binary numbers as inputs; one from the input module and one from the state module. To check if these numbers are equal, they are compared by using a complemented XOR gate. If the two inputs are equal, the output “equal” is set to return 1. Otherwise, difference inputs set the output to 0. This output is used as an enable signal for the other modules.

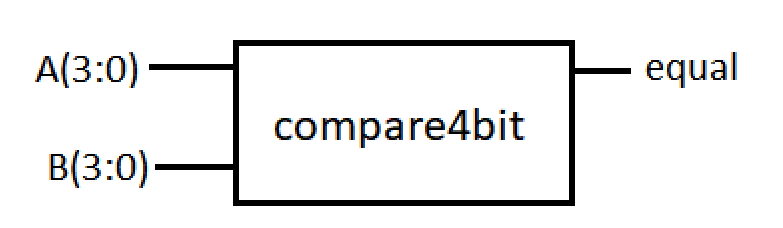


Figure 8 – Compare module

1. Updating the Score – ScoreUp

The ScoreUp module consists of an 8-bit full-adder, and an 8-bit 2-1 multiplexer. Inputs S1A and S1B are respectively the current score and the score increment. C and S are respectively the carry-in value for the full-adder, and the select value. S0 is the selected input to the MUX when S is 0. Outputs Co and O are respectively the carry-out value, and the next score to be displayed.

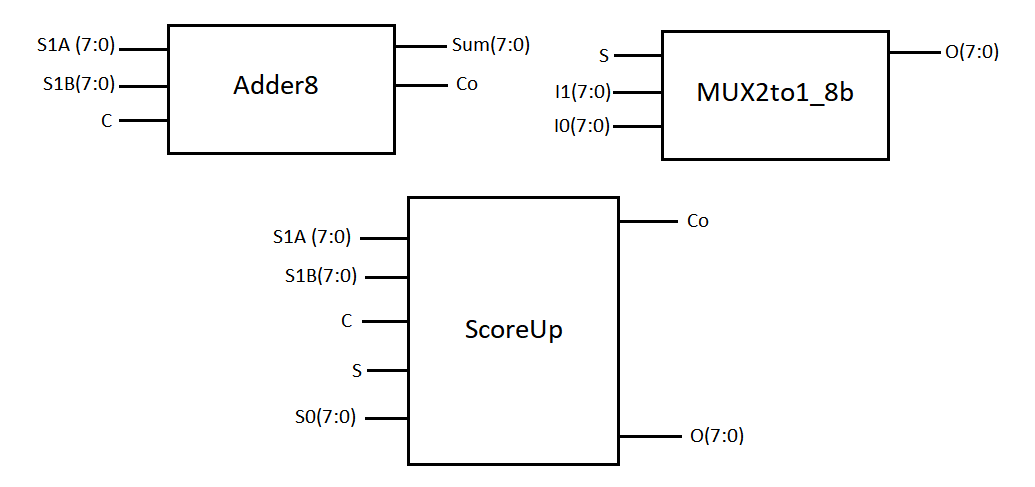


Figure 9 – Module that generates the next score

* The 8-bit full-adder takes the current score as input and increments it, then returns it as output to the MUX.
* The 8-bit 2-1 MUX will do either of the two things depending on the select input S; if S is 1 then it takes the output of the full-adder as output O. Otherwise, it returns 0000 to the output.

1. Changing the Pattern – patternUp

The patternUp module generates the pattern that is to be displayed by the LEDs, and the player needs to successfully copy this sequence in order to pass the round. It takes the current pattern from its input currentPattern[3:0] and updates it using a 4-bit full-adder. The module returns the next pattern to be displayed in output nextPattern[3:0], which is connected to its corresponding register.

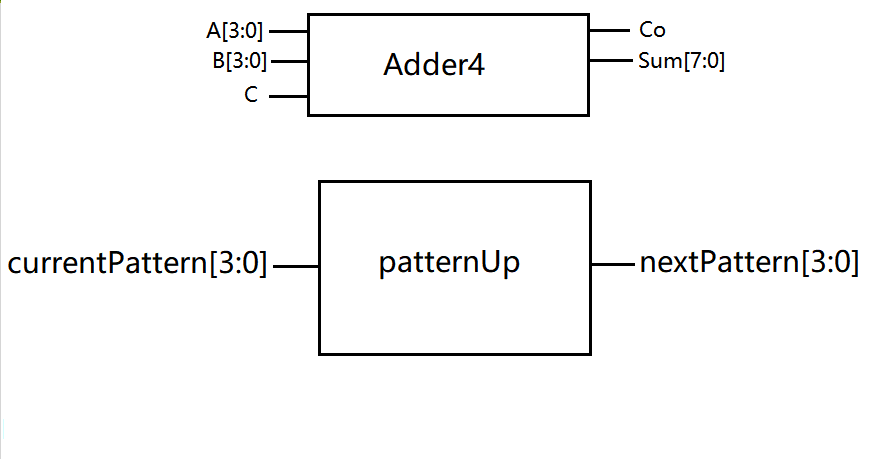


Figure 10 – Module that generates the next pattern

1. Score and Pattern Registers – register4bit\_async

The scoreUp and patternUp are both connected to their own registers, scoreReg and patternReg respectively. Both registers are made up of 4-bit registers, with asynchronous D flip flops. The score register has two 4-bit registers, and the pattern register has one. Both registers are initialized after the button is pressed (hence the INIT input) and provides the corresponding output signals to the display module. scoreReg takes the next score to be displayed (generated by scoreUp) as input, as well as an enable signal from the timing module. patternReg is similar, but instead takes the next pattern to be displayed as input. Thus, both registers store the proceeding attributes and passes it on to the display module.

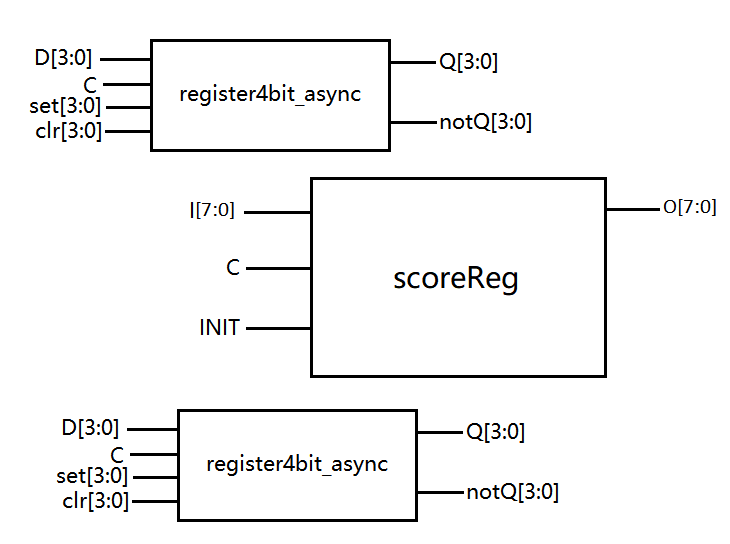
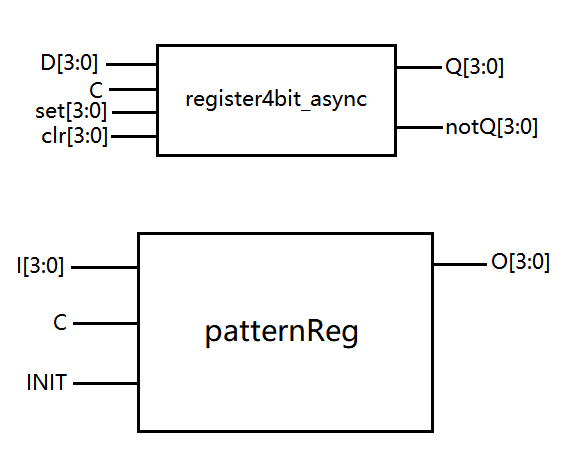


Figure 11 – Modules that stores next pattern and score to be displayed

1. Initializing the Modules – Timing

The timing module consists of an asynchronous D flip-flop, a XOR gate, an AND gate and two frequency dividers. Its purpose is to prepare and initialize the modules in the circuit to update the score, and pattern of each round in the game, and provide rest time. It takes in a clock cycle through input clk, and the two frequency dividers are triggered when clk is at a positive edge. gameState acts as the enabling input signal for this module, and roundTime(3:0) provides the frequency for processing, which is then returned in output cout.

* The frequency divider module takes in a input signal of a frequency through freq[3:0] and generates a output signal of cout, when triggered by a positive-edge input of a clock cycle (clk). The count input acts as an enabling signal for the module. There are two frequency dividers in this module, one to provide timing of each round, and one to provide a rest period for reset and generating the parameters of the next round.
* The asynchronous D flip-flop resets and initializes the two frequency dividers, by controlling their enable inputs.

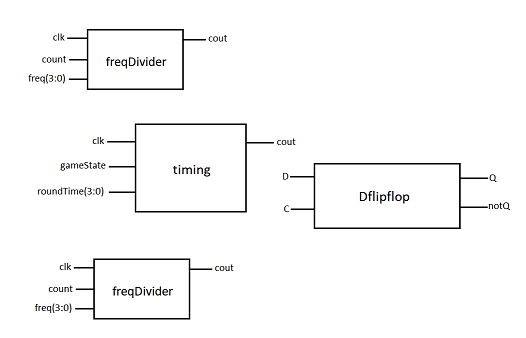


Figure 1 – Module that controls timing of other modules

1. Anti-Jitter Module – pbdebounce

The anti-jitter module used in this project is similar to the one used in class. The purpose of pbdebounce is to reduce the level of jitter in a regular pulse signal, from when the button is pressed.

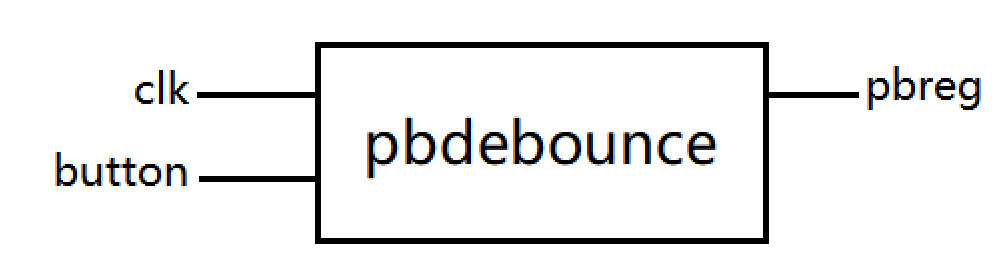


Figure 1 – Anti-jitter module

1. One-Second Clock – sec\_clk

The sec\_clk module provides clock pulses of one-second periods. It outputs these clock pulses through cout, and sends it to the timing module.

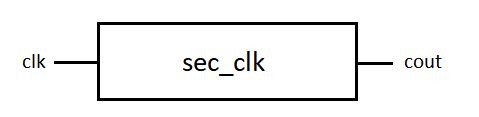


Figure 1 – One-second clock module

1. Display

The display module consists of a binary-to-BCD converter, an 8-bit 4-1 multiplexer, a 16-bit 4-1 multiplexer, and an altered version of DispNum (which was made in lab 7). Input C is a control signal from the built-in timing module, and input clk takes in a clock signal from the machine’s internal clock. The current score and pattern and pattern are also part of the machine’s inputs. SEG, AN, and LED provide the output to the seven-segment display and the LEDS, test provides the output of the 16-bit 4-1 MUX and test0 provides the result of the binary-to-BCD converter. This module is responsible for displaying the score, generating and displaying the pattern.

* The binary-to-BCD converter takes the input from score(7:0) and performs a conversion, which is then returned to the 16-bit 4-1 MUX as its input.
* The 16-bit 4-1 MUX is comprised of four 4-bit 4-1 MUXs, and a decoder. The purpose of this segment is to select the proper digits to be outputted for when displaying the score. The output of this module is the input of DispNum.
* The 8-bit 4-1 MUX is made up of two 4-bit MUXs, and a decoder. The purpose of this MUX is to produce the pattern to be displayed and output the signals to the LEDs.
* DispNum synchronously outputs the digits selected by the multiplexers. There is also a decoder which determines whether to output PASS or FAIL, from input C.

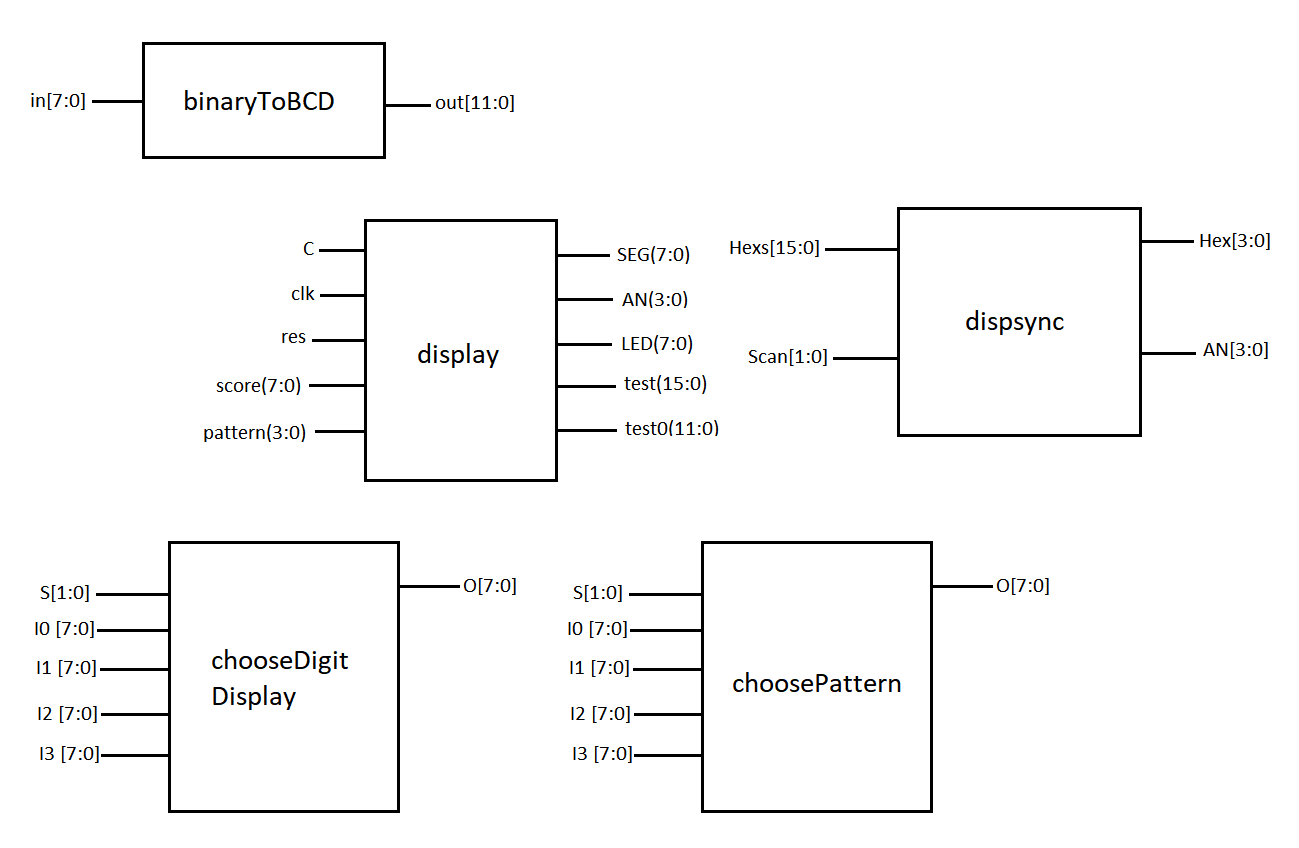


Figure 1 – Module that controls the 7-segment display

1. Program Flowchart

A screenshot of a cell phone

Description automatically generated

Figure 1 – Flowchart of simulated working circuit

1. Simulations

*All simulation files can be found in top module folder.*

1. Top (All Modules Combined)
2. userInput
3. compare4bit

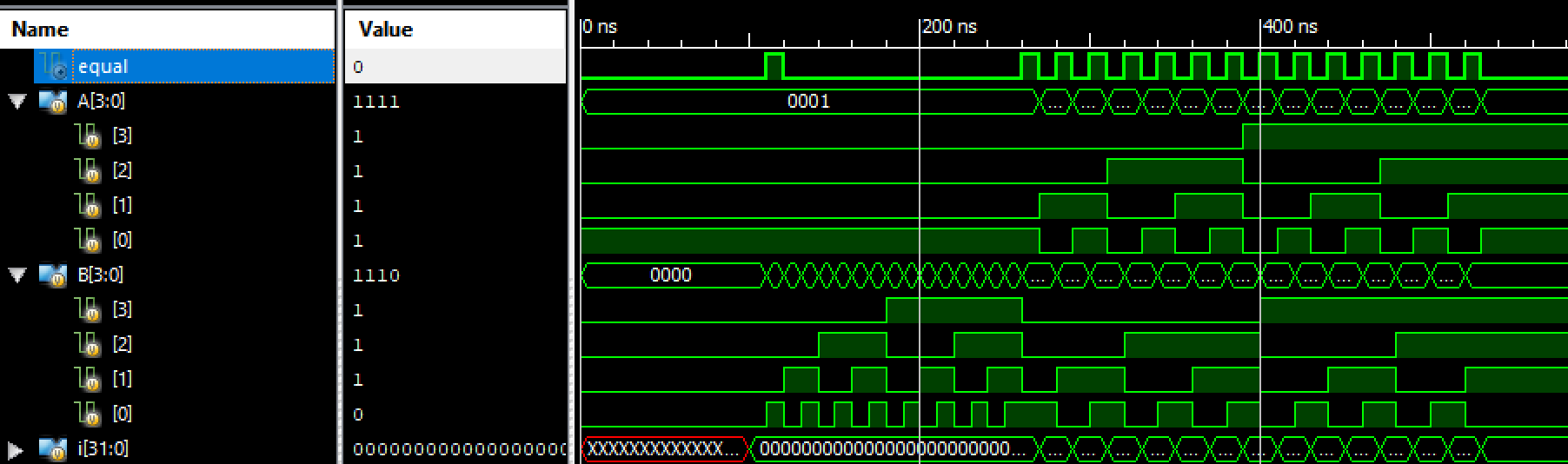
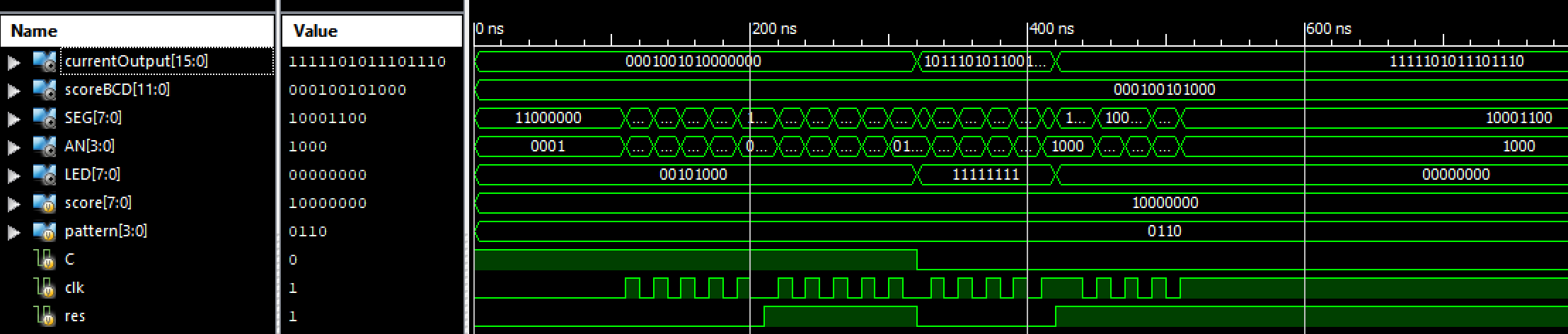


Figure 19 – Simulation of compare module

1. Display

  
Figure 20 – Simulation of display module

1. sec\_clk

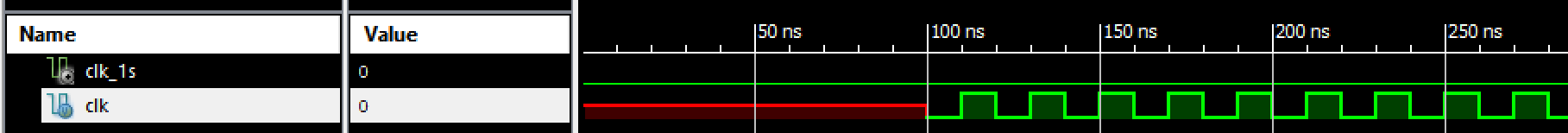


Figure 21 – Simulation of one-second clock module

1. timing

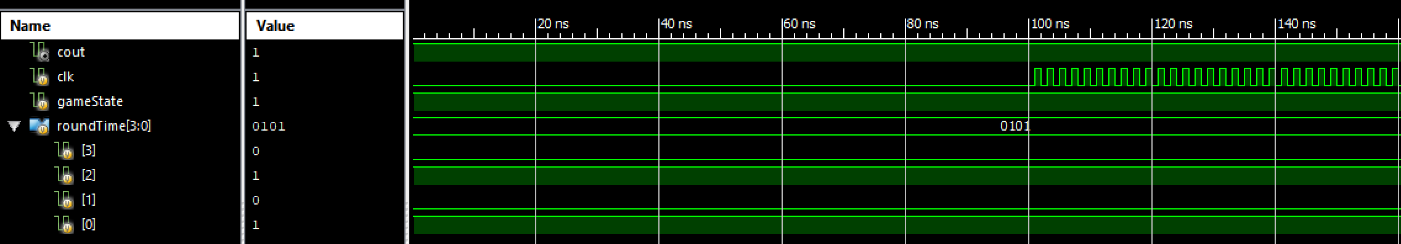


Figure 22 – Simulation of timing module

1. scoreUp

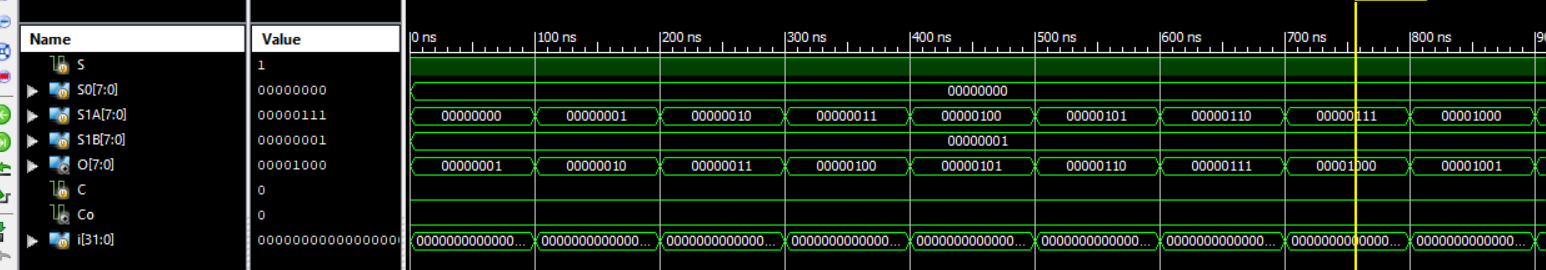


Figure 23 – Simulation of score update module

1. patternUp

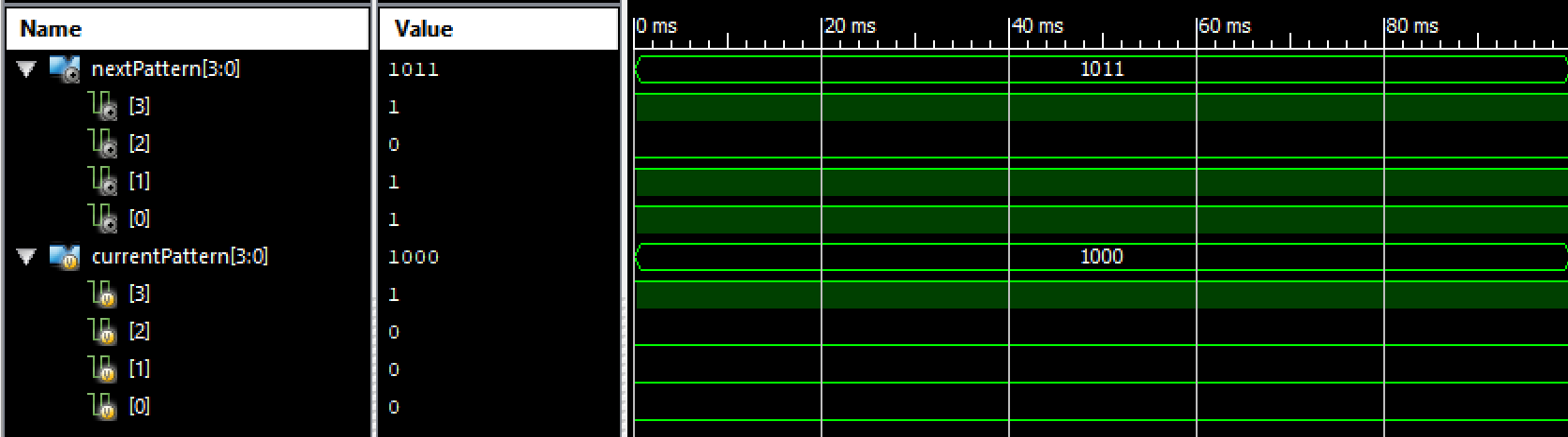


Figure 24 – Simulation of pattern update module

1. patternReg

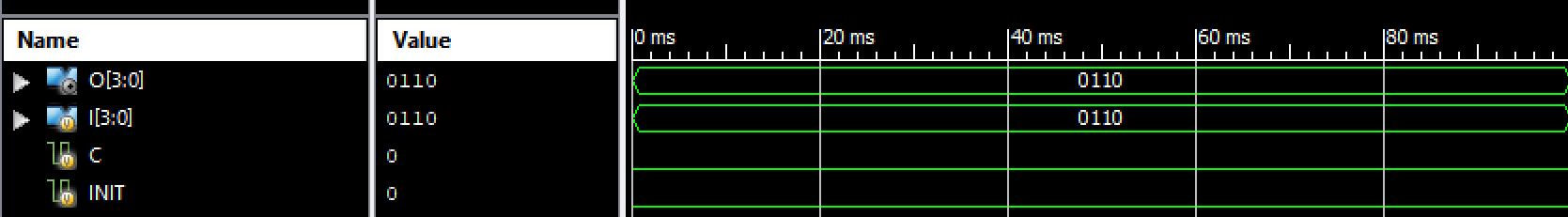


Figure 25 – Simulation of pattern register module

1. scoreReg

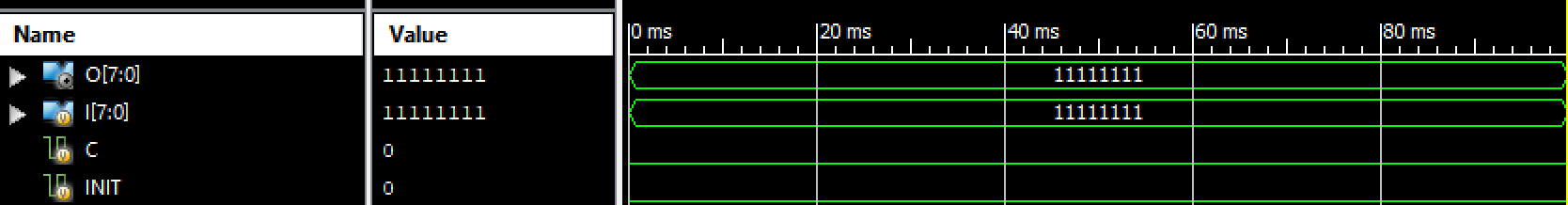


Figure 26 – Simulation of score register module

1. Debugging
2. Final Comments

Our group ran into many difficulties while working on this project. Firstly, we underestimated the level of difficulty and the amount of time it would take to create a working circuit. As a result, we were unable to perform proper debugging and test out our experiment. Many of the modules that we made in class (i.e. DispNum, MUX4to1b, DispSync, clkdiv…) were reused in this project, and it also forced us create our own components such as the registers. Although the idea and implementation seemed simple at first, many changes had to be made and simplified to get the project done within a reasonable timeframe. Our group originally wanted to implement a timeUpdate module where the duration of each round got gradually shorter but was unable to do so due to the lack of time and extra difficulty it could potentially bring. Another hurdle that we faced was that half of our group was separated in two different countries near the deadline, making it hard to communicate with each other and evenly distribute work. Nonetheless, we all found the project to be beneficial towards reinforcing what we have learned in class and in labs. We were able to continue practicing what we have learned from lectures, as well as learn new concepts beyond the scope of this course by ourselves.

9- Work Distribution

Anna Tang (Group Leader): **35%**

* Controlled and managed project repository on Github
* Performed simulations and drew all diagrams on report
* Wrote the entire report
* Provided modules from past experiments
* Made the UCF

Joshua Malmberg: **40%**

* Did all circuit/module drafts and draw-ups
* Constructed most modules in the circuit
* Compiled all modules into a top module
* Provided Verilog testing modules for simulations
* Debugged all modules

Chen Yi Hui: **20%**

* Constructed scoreUp module and its components
* Tested some modules on SWORD board
* Contributed to small jobs

Justin Choi: **5%**

* Attended and contributed to group meetings

1. References
2. Sutherland, S. (2002). *Verilog-2001: a guide to the new features of the Verilog hardware description language*. Boston: Kluwer Academic.
3. Mano, M. M., Kime, C. R., & Martin, T. (2016). *Logic and Computer Design Fundamentals*. Harlow, Essex: Pearson Education Limited.

11- Source Code and Pinouts

Please see the following files in the submitted folder. The top module folder contains all files (including simulations used in this project).

* Top.v
* Timing.v
* Compare4bit.v
* updateScore.v
* uppdatePattern.v
* patternRegister.v
* scoreRegister.v
* inputRegister.v
* display.v
* clk\_1ms.v
* pbdebounce.v