CSUN | COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

ECE 524L – Advanced FPGA Design
Fall 2024

Lab 8

Guessing Game

11/17/2024
Daniel Beltran

Table of Contents

1. Introduction

The goal of this lab was to use 3 PMODs to implement a Hot/Cold guessing game. The Seven Segment Display, Keypad controller, and UART module were brought over from the previous Lab. All these PMODs work the same as they did in the previous lab. In this lab, the keypad inputs will be used as the guessed inputs, these guesses will be displayed on the Seven Segment Display and will be output through the UART module. The user will select a random target number between 0x0 and 0xF by pressing button 1, this will begin the game. After the number is selected, the user will try to guess the number using the key pads, the RGB LED will be red if the user is close to guessing the number and cold if they are farther from the number. If the number is guessed correctly then the RGB LED will blink green 10 times.

2. Pre-lab Assignment

1. What is a Hot/Cold Game?

The Hot/Cold game is a game where a random hiding spot or in the case a number is picked. The seeker does not know the value of the number but is forced to try and guess the number. Once they make a guess, a response would be given on whether the seeker is hot(close) or cold(far). The initial guess will always be hot. Based on the initial guess the hot and cold prompts will be given to the seeker; from this information they can extrapolate what the number is and eventually guess the correct number.

2. What is your strategy to show two digits on the SSD?

The strategy is the same as previous labs using the Seven Segment Display. First, to display both digits, they must be toggled at above 50 Hz. I found that 83.3 Hz worked the best to reduce any jitter in the display. There is a separate component which is used to decode the keypad presses and relay an integer to the Seven Segment Display Controller. The digits are held in an integer array inside the SSD Controller. The right digit is determined by the input of the keypad. If the new keypress is not equal to the data in the right digit, then the right digit is updated and the previous right digit is moved to the left digit.

3. What were some of your challenges on previous labs to get the keypad and UART and SSD working together?

Initially I could not get the UART output working correctly, nothing would echo back from the Zybo board when input was given on my PC. I found that my issue was that my RX and TX pins in the constraints file were mix-matched. After fixing the constraints file the UART echo worked as intended. Another challenge I had was with shifting the digits after new inputs were given through the keypad. When a new key was pressed both digits would update instead of shifting. To solve this, I added the condition that the new incoming key data cannot be equal to the data in the right digit before updating the digit values as described in the previous question. Lastly, the keypad would not respond to any inputs. It took me a while to realize that the keypad used an active low reset while the rest of the design used an active high reset so the keypad was held in reset the whole time. I changed the design given by Digikey in order to solve this issue.

3. Design/Implementation

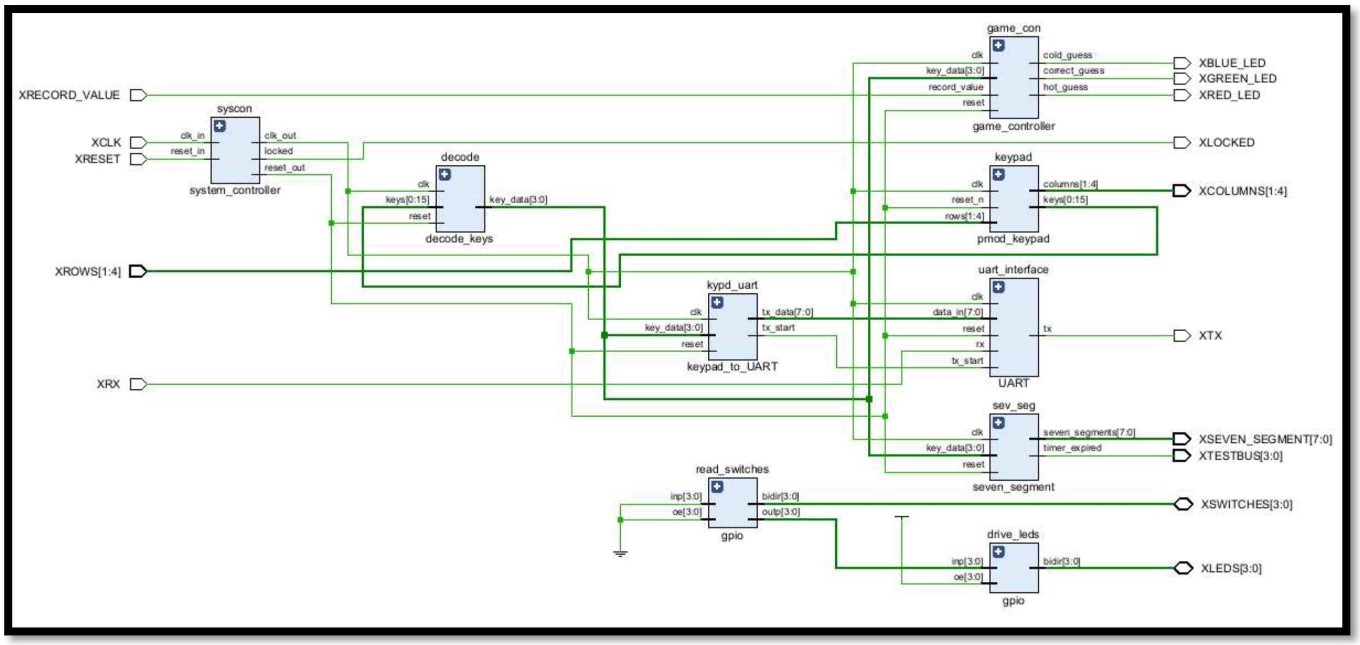

Fig. 1 Design Schematic

Key Decoder

In this lab, I moved the key decoder to be modular. In the previous lab the keys were decoded inside the SSD controller and keypad to UART controller. The key decoder uses the same process as the previous lab it is just now in its own module. Now, all key presses are converted into integers using the decoder module which is utilized by the SSD controller keypad to UART and Game Controller module.

Game Controller

The game controller, uses a state machine to control the hot and cold game, it also uses a counter to randomly choose a target number. There are 4 states, IDLE, HOT, COLD, and CORRECT. The state machine begins in the IDLE state, it leaves this state as soon as button 1 is pressed. When this button is pressed, the target number is set from the current value of the counter and the state is then moved to the HOT state. In the HOT state, the red LED is lit up and the distance from the target to the guessed value is calculated. The distance is then compared to the distance of the previous guess, if the current distance is less than the previous distance then the state remains HOT i.e. the guess is closer to the target value. If the current distance is more than the previous distance then the state is now the COLD state. In the COLD state, the blue LED is lit and the distance from the next guess is calculated. The same comparison to determine hot and cold is used in the COLD state. In both the HOT and COLD state, there is a condition where the target is equal to the guessed value, this condition leads to the CORRECT state. In the CORRECT state, the green LED is blinked 10 times. To blink the LED, a counter is used which counts up once every clock cycle until half of the clock frequency is reached. The clock frequency is determined using a generic. If the generic is set to the clock frequency of the board, the LED will turn on for half of a second and then off for half of a second.

4. Verification

In this lab, the only new module which had not been tested was the game controller so that was the only module which was tested. To test the module, a button was simulated by setting the record_value signal high for 1 clock cycle. After calculating the timing of the target counter, the target number in simulation was found to be 5. As seen in Fig. 2, the initial keypad data is 0, and the red LED is high which means that the data is hot. Next, the value is set to 11 which is farther from 5 than 0, so 1 clock cycle later the blue LED is high which means the guess is cold. The key data is then set to 1, 14 and 7 in this order. The corresponding hot and cold status can be seen in the waveform and it operates as expected. Lastly, the keypad value is set to 5 which is the correct value, at this point the green LED flashes 10 times. One thing in the test bench that differs from the version of the component uploaded to the board is the clock frequency generic. In the test bench, the generic is set to 10, this means that the LED will blink every 10 clock cycles. If this change was not added then the blinking would not be visible in the waveform of the testbench. The blinking waveform can be seen in Fig. 3.

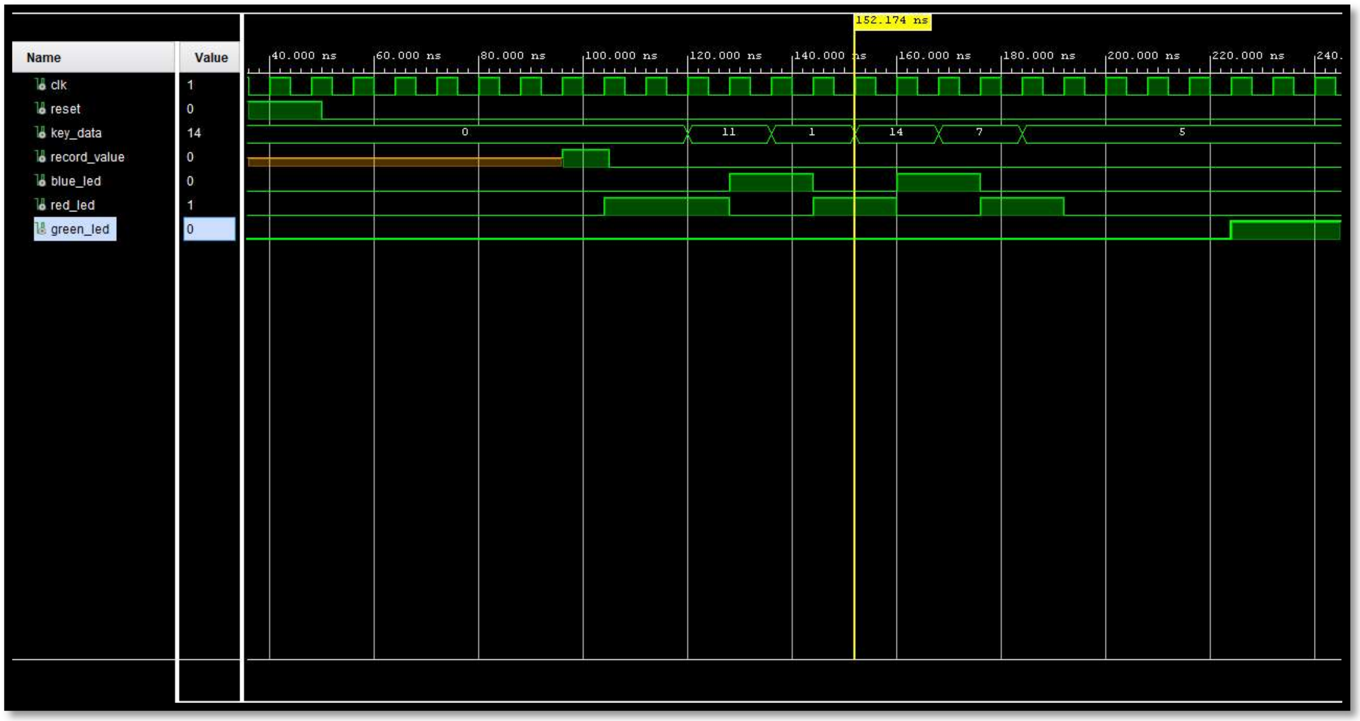

Fig. 2 Proper Hot and Cold LEDs

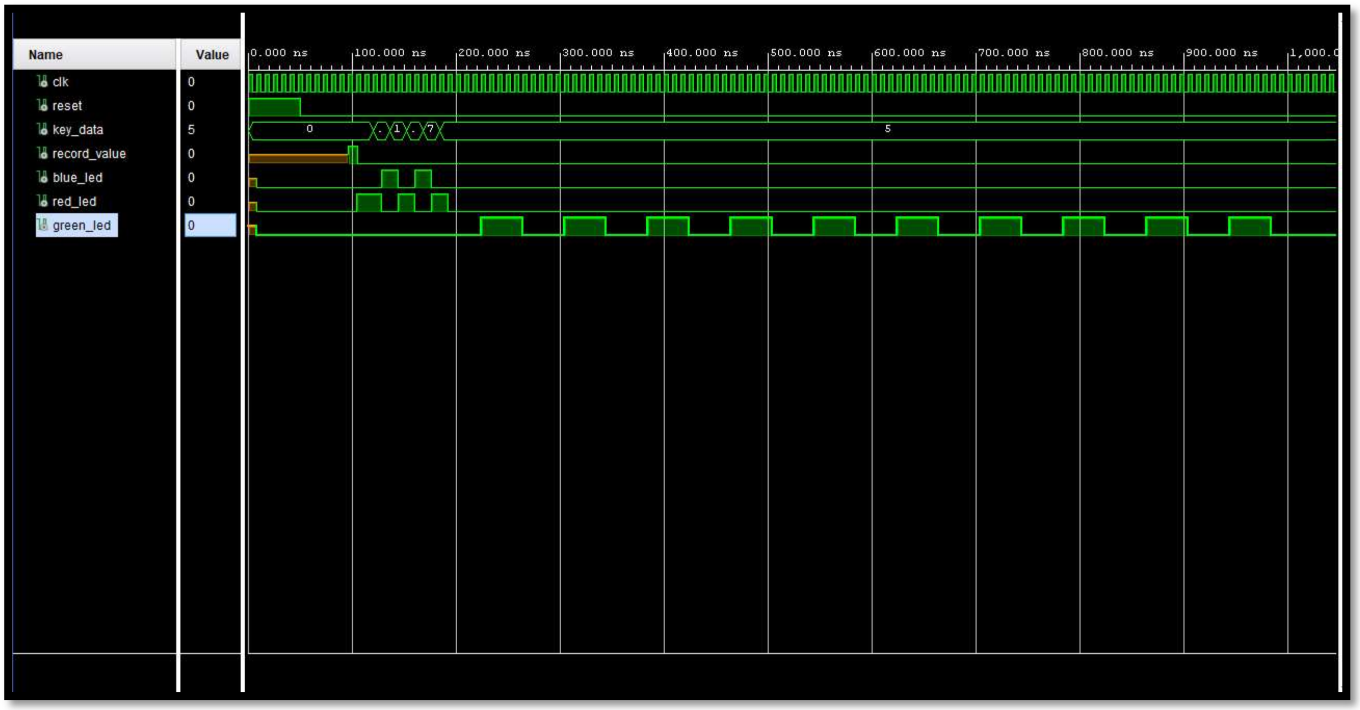

Fig. 3 Green LED Blinks 10 times

5. Conclusions

In conclusion, this was a successful lab and simple considering most of the lab was completed in the previous lab. I was successfully able to get my key decoder module working. I wanted to get this working in the previous lab but was unsuccessful in doing so. This module was the first thing I got operating in this lab since key decoding would have been used in three of the modules and it would be more optimal to have a key decoder instead of rewriting the same code multiple times. Next, the game controller module did not give me much trouble. I originally had some timing issues since I only had one state to control the HOT and COLD LEDs. I then moved this to two states and my timing issues were solved.

6. Appendix

1. Project Video Demonstration: https://youtu.be/cwGH-QW32pA