CS/EE120B Final Project: Simon

Jeremy O'Neill - jonei005@ucr.edu

Collaborators: Gabriel Cortez, Benjamin Li

Summer 2017

Overview

For the final project, my group chose to work on recreating the game Simon. It follows the same logic as the original Simon game: the game displays a random sequence of increasing length, and the player must input the sequence in correct order to win.

Design

Our code consists of two concurrent state machines: one state machine handles all of the game logic, such as waiting for player input and determining wins/losses, and the other state machine handles output to the peripherals, such as blinking the LEDs or playing a note from the speaker.

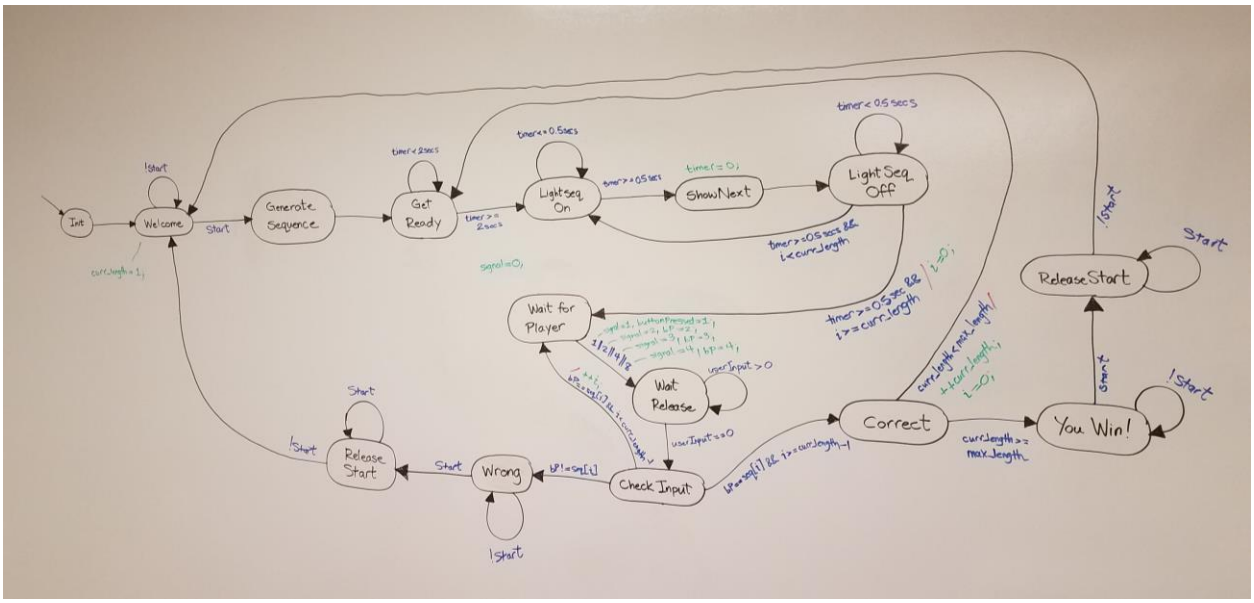
For the first milestone, we wanted to create a simple state machine that would wait for user input to start the game, wait a short amount of time (to allow the user to get ready), and then blink a light. This way we knew we had our microcontroller and peripherals set up correctly, and could get user input.

The second milestone was generating a random sequence and blinking it in order, first blinking only 1 light, then 2, until blinking all 9 lights in the sequence.

The third milestone was reading user input from buttons and determining whether that input was correct or incorrect, by comparing each button press to the corresponding number in the sequence.

The fourth milestone was determining what to do with correct or incorrect button presses. If the user pressed wrong input, they were immediately told that they lost, and were told to press start to replay the game. If the user correctly guessed the sequence of length 9, then they were told that they won, and to press start to replay the game. The tough part here was to generate a new random sequence each time they replay the game. We were able to do this with the C library functions srand() and rand().

The final GameLogic state machine looked like this:



I do not have a picture of our Signal state machine, but its purpose was only to read variable "signal" and based on that value, go to a specific state which would light up one of four lights and output a sound unique to that light. The "signals" variable was assigned in the GameLogic state machine. For example, if the user pressed button 1, then the GameLogic state machine set "signals" to 1. The Signal state machine would then go from the Off state to Signal1 state, which would light up LED 1 and output a G4 note on the speaker. It also had states to control the LEDs and speaker in the event of a win or a loss, where all LEDs would blink or light up at once.

Components

In addition to our ATmega1284 microcontroller, we used many peripherals for this project:

- LCD display (connected to PORT D7…D6 & PORT C7…C0)
- 4 LEDs (connected to PORT B3…B0)
- 5 buttons
    - 1 start button (connected to PORT A7)
    - 4 game buttons, corresponding to an adjacent LED (connected to PORT A3…A0)
- Speaker (connected to PORT B6)
- On-board timer
- On-board pulse width modulator

Additional Information

Because we used the LCD display, we needed to include 2 files that were given to us in Lab 6, in which we first used the LCD display. Those files were “io.c” and “io.h”, which contained relevant functions to control the LCD, including clearing and writing data to the screen. These files are located in the GitHub repo, where the Readme.md file will describe the filepath.

In addition to those files, our “main.c” file also included functions that were provided to us in labs to control the on-board timer and pulse width modulator.

Here are relevant project links:

GitHub Repo: https://github.com/jonei005/simon

Demo Video: https://youtu.be/zdUH_CReJn4