Summary and Analysis of my Custom Project

In my final project for the CS120B class, an introduction to embedded systems, I created an embedded system using the ATMEGA 1284 chip to create a system that mimics the game Simon Says. If you are unfamiliar with the rules of the game, it is simply a game where users must replicate audio and visual patterns which increases in difficulty as the game progresses.

The criteria for this game was to have four LEDs to show the player what the sequence to follow is, and four buttons for the player to replicate said sequence. In my project, I did this by generating the nine "values" that determine what LED will light into an array. This array is filled and generated the moment the program is initialized, before the game even starts. On victory or defeat though, the array is regenerated with different random values, thus keeping the random factor of the game intact. A global variable named level keeps track of the progress of the game, so that users can see his or her progress as well as inhibiting users from entering more input values than required. For example, if the game shows four LEDs light up in a sequence, level prevents the user from entering more than four inputs, thus avoiding corruption and keeping the game intact.

In terms of reading input, button de-bouncing is placed inside the function that reads input, so that one user input will translate to only array input. This means that even if the user holds down the button, only one input will be read and for more inputs to be read you will need to release and repress. All user inputs are stored in a separate array that overrides itself with new user inputs every time the level changes or repeats. I then use a simply comparison function that compares the array and if one value is wrong at least then the level resets and a life is lost. Each game starts with three lives and if the life count reaches 0 then the game automatically is lost and the game is over. (The game after victory or lost will automatically restart after showing the respective screen for 10 seconds)

An issue found while dealing with this game is the generation of a truly random sequence. In the end, due to the limitations of the ATMEGA 1284 microcontroller only pseudorandom arrays could be generated. This means that for a game to be truly random, the user must program again. A possible software solution would have been to seed the random program by time between the first user input and when the program is first initialized. However, this would not be truly random as the time between said input and initialization is not truly random as we desire, so we left it as pseudorandom.

The LCD screen was the simplest part of the program. Following the wiring configuration from a previous lab (Lab 7), I could get it successfully wired with no issues. After initializing the ports C and D which controls the data lines and configuration of the LCD, I could utilize the LCD for my final project. I had the LCD update the score and the user life count This display is also written after a user loses a life or if a user enters a sequence correctly, thus increasing the level and difficulty. I created my program so that the LCD constantly shows one of six things: the score and life count, the screen showing that the player has entered the correct sequence, the screen showing that the player has entered the incorrect sequence, the victory screen, the defeat screen, and the introduction sequence. Each of the possible LCD scenarios has

a custom LCD message ready and will automatically move once a certain period has passed (usually one second or two). The only LCD display that breaks from this trend is the introduction, which due to the length of the text, had to be broken into four segments that display one after the other (I have one out of the four LEDs light up each in sequence to show what stage of the introduction the LCD is currently displaying). At the end of the introduction, a user input is required (Any button would work) and the game would then officially start.

The PWM (Pulse width modulation) of the code is easy to implement since I had the previous lab on the PWM noise maker (Lab 9) to use. Once inside the program, to utilize it, I simply called on the PWM every time a LED is lit or a button is pressed and mapped certain frequencies to its respected button or LED.