Ryan Hagler

CMPE 311, Ivan

Homework 5

System Report

1.0 User Manual

1.1 Playing LockPick Simulator

For the user to play the game, an LED and 360-degree servo are required to fully experience the game. The LED is connected to the AVR Butterfly using PORTB1 and GND pins. The servo is connected using PORTB2 as well as VCC and GND pins. Figure 1.1 illustrate where the user can plug these components in.

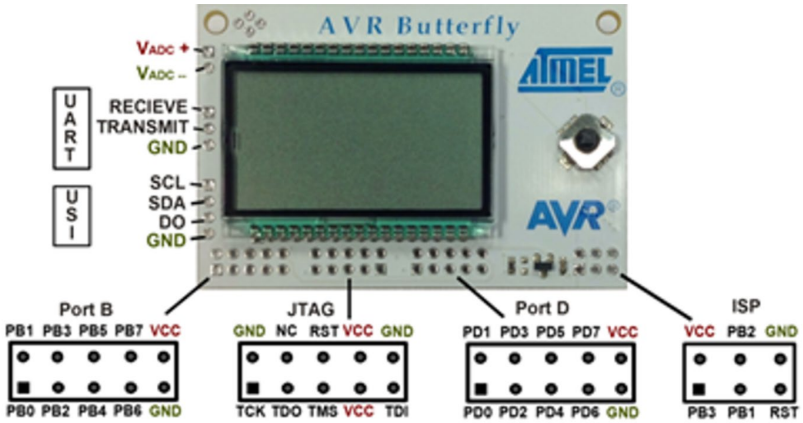

Figure 1.1

Once these connections have been made, the user only has to interact with the D-pad integrated into the Butterfly. Strictly due to the limitations of the hardware, the user must “initiate” each button once before the software can recognize it.

The user is greeted with a welcome splash screen before the main menu is scrolled across the display, at which point the user may click center to go into lock selection mode. From here, the user will use the left and right buttons to select the lock they would like to attempt to crack. This menu is a carousel rotation so that “lock 1” goes to “lock 3” from the left and vice versa. The user may press the center button to start that game mode. Once the game has started, the user will see visual and audio clues for how well they are doing. If their move made them closer to the lock, the buzzer will sound at an according pitch. If the user goes over the secret position, the led will quickly flash to give them an approximate idea of where the lockpick location is.

At any point during the game, the user can again click the center button to go to the selection page. If the user completes the game, however, the LED will shine for a longer time and the LCD will display ”WIN” to the user. The lock selection screen will then be loaded for the user to try again.

2.0 Scope of Work

2.1 Behavior Description

Project 5 is at quite a stable state with all three major aspects working. The d-pad, LCD, LED, and buzzer are all working completely to my design. The only limitation I have found is the state of my servo. Because I have a 360-degree servo, fine motor movement is challenging, and the lack of PWM due to the buzzer and buttons being used makes it more difficult. I believe that if I had a 180-degree servo instead, the integration of the servo movement would be more functional. While the servo itself works and at times can be operational, it is not at the point that feels 100% implemented.

It is noted that the servo timings have been tuned since the taping of the demo video, and the stability of each direction has improved on my servo.

I implemented a global variable to track the position of the servo, as this allowed my to never cross the 90- or 270-degree mark. The servo also rotates back to its original position both when the user wins a game or hits ‘center’ to go back. I also set interrupt pins for the Center, Left, and Right button, while Up and Down were isolated enough in their function to deem interrupts unnecessary. The center button acts like a state machine, as it sets the value for what mode the program is in. The Left-Right interrupts use this value to know whether or not the lock selection can be change.

2.2 Testing and Debugging

A good majority of the time put into this project was solely for testing and debugging, I faced many issues, one of the largest ones being that the buzzer function would call on an out-of-bounds array index which would lock up the entire AVR. This was due to my equation used to map the pitch of the buzzer with the position of the servo compared to the secret position.

By far the hardest bug to track down was that the program would seemingly freeze after only two of three rounds of lock-picking. I had many iterations of the project where I thought I solved to issue, only to have the freezing take place somewhere else. The final solution to this bug was to issue ‘return’ statements throughout my functions in conjunction with the center-button interrupt. While I initially tried to use a variable to track where the program was based on the center button presses, it seems as though the program got to a place in memory that it didn’t know escape. By using ‘return’ commands, this issue appears to be completely resolved.

2.3 State of Completeness

As stated above, the project is fully complete, with the only exception being the shortcoming of the 360-degree motor. While the movements are not accurate all of the time, my testing showed that the delays chosen moved the servo in the correct direction more times than not. This is an improvement over project 4 where I was only able to get the servo moving one way. The other challenge to this project is that the movement windows are incredibly small, so there was a tight balance between proper movements and rotating the key slow enough to where the player could actually find the secret angle.

3.0 Project Overview

3.1 Design Decisions

As I have done in prior projects, I used a lot of modular coding practices in order to test and debug different parts of the project before combining everything together. This allowed me to tune the servo, the LCD display, and the buzzer without having to change parameters of the others. I also implemented 3 .c files for this project. There are two driver files, the provided LCD_Display driver as well as a buzzer_driver file which borrows from my MP3 player project 2. Although the buzzer driver only has one function, I found that the separate file made it easier to distinguish. The game logic for this program is all in main.c, with main() only serving as a "jumping off point" for the other functions. There were four main functions used. These were the Welcome screen, the Main Menu screen, the Lock Selection screen, and the Play Lock game itself.

I decided to use a state machine approach to this project because there aren't many deviations from the main functionality. Using the documentation, I knew that the Welcome screen and Main Menu screen would only be seen once, and the Lock Selection and Play Lock modes were the only things left. I utilize the AVR's built in timer0 to track approximately when the user hasn't touched a button on the lock selection screen within 15 seconds. I also use the AVR's timer to count during Play Lock, both from scrolling text across the screen but also when deciding the timing for moving the key in the lock.

This was a very important design choice, as letting the key move too fast created poor gameplay that was borderline un-winnable, to too slow of lock movement where the user would feel bored and the servo would tend to lock up. I found that using timer1 and its overflow flag to be a good mix between these two. Additionally, I incremented the servo position by 5 degrees each time instead of 1, as this felt more natural and made lining the key up to be slightly less tedious.