F28HS

Hardware-Software Interface CW2

Systems Programming

Dominic Calina H00235891
Matthew Reilly H00219672

Game Specification

In this coursework, we had to build the classic game Mastermind. The users interact with the system by entering numbers with a button. There are two LEDs and an 16x2 LCD display that give feedback to the user to show them the amount of exact and approximate entries they have given. The game continues until the codebreaker guessing the exact code the codemaker created at the start of the program.

Hardware Specification

- Raspberry Pi 2 with Raspbian 7 installed on SD
- Green LED(data) -**GPIO 6**
- Red LED(control) -**GPIO 5**
- Button (input)- **GPIO 19**
- 2x16 LCD display with 4 GPIO pins on the RPi2: **23, 17, 27, 22.**

There are 29 wires. All power wires are red and all ground wires are black.

There are 3 resistors and a potentiometer. These are used to reduce/control current flow.

Code Structure

Main program: Mastermind.c

This program has the code for the mastermind game. It uses header files to call the functions in other c programs that control the hardware e.g. button input or sending a string to the LCD screen.

The main functions of the program are:

yblink(int n) – This function is calls the external function blinkyellow() n number of times.

rblink(int n) – This function is calls the external function blinkyellow() n number of times.

inputsecret() – This function takes in the secret code from the codemaster by them clicking on the button a certain number of times.

inputguess() – This function takes in the guess code from the codebreaker that has been entered using the button.

exactFun() – This function finds all the exact matches between the guess code and the secret code.

approximateFun() – This function finds all the approximate matches between the guess code and the secret code.

success() – This function is called when the secret code has been found by the codebreaker.

<u>Other programs (accessed via header files)</u>

1. blinkred.c
2. blinkyellow.c
3. buttoninput.c
4. buttonwait.c
5. lcd.c
6. red.c

Hardware Functions

blinkred() from blinkred.c: This function was used to make the red LED turn on for 400ms and then turn back off. This is all done in **C**.

blinkyellow() from blinkyellow.c: This function was used to make the green(yellow) LED turn on for 400ms then turn off. This is all done in **C**.

redon() from red.c: This function turns on the red LED. This is all done in **C**.

redoff() from red.c: This function turns off the red LED. This is all done in **C**.

buttoninput(*'time'*) from buttoninput.c: This function will run for time '*time*', during that time it will record the number of button presses there were, also every time the button is pressed the green(yellow) will turn on for 400ms then turn back off. The mode of the button and LED are both set in **assembler**. The green/yellow button is turned on with **inline assembler**. Registering button press is done in **C**.

waitforpress() from buttonwait.c: When this function is call it will pause the program until the button is pressed. The mode is set in **inline assembler**. Registering button press in done in C.

lcd(*'String','String'*) from lcd.c: This function will take in 2 strings and output each one onto a line on the LCD screen. This is all done in **C**.

Clear() from lcd.c: This function when called will clear all data from the LCD screen. This is all done in **C**.

Sample Execution- Debug mode

```
gcc -0 -o cw3 Mastermind.c buttoninput.c lcd.c lcdBinary.o blinkyellow.c blinkred.c red.c buttonwait.c
```

```
OUT:  <Red Blink 3>
Codemaker input starts here
IN:   <Press 2> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 2>
IN:   <Press 1> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 1>
IN:   <Press 2> <Pause>
```

```
OUT:  <Red Blink 1>
OUT:  <Green Blink 2>
OUT:                              Secret: 2 1 2
OUT:  <Red Blink 2>
Codemaker input stops here

Codebreaker input starts here
OUT:  <Red Blink 3>
IN:   <Press 3> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 3>
IN:   <Press 2> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 2>
IN:   <Press 1> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 1>
OUT:                               Guess: 3 2 1
OUT:  <Red Blink 2>
OUT:  <Green Blink 0>
OUT:  <Red Blink 1>
OUT:  <Green Blink 2>
OUT:  Exact: 0
OUT:  Approx: 2
OUT:  <Red Blink 3>
IN:   <Press 2> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 2>
IN:   <Press 3> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 3>
IN:   <Press 3> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 3>
OUT:                               Guess: 2 3 3
OUT:  <Red Blink 2>
OUT:  <Green Blink 1>
OUT:  <Red Blink 1>
OUT:  <Green Blink 0>
OUT:  Exact: 1
OUT:  Approx: 0
OUT:  <Red Blink 3>
IN:   <Press 2> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 2>
IN:   <Press 1> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 1>
IN:   <Press 3> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 3>
OUT:                               Guess: 2 1 3
OUT:  <Red Blink 2>
OUT:  <Green Blink 2>
OUT:  <Red Blink 1>
OUT:  <Green Blink 0>
OUT:  Exact: 2
OUT:  Approx: 0
OUT:  <Red Blink 3>
IN:   <Press 2> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 2>
IN:   <Press 1> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 1>
IN:   <Press 2> <Pause>
OUT:  <Red Blink 1>
OUT:  <Green Blink 2>
OUT:                               Guess: 2 1 2
OUT:  <Red Blink 2>
OUT:  <Green Blink 3>
OUT:  <Red Blink 1>
Codebreaker input stops here
OUT:  SUCCESS
OUT:  Attempts: 4
OUT:  <Red On>
```

```
OUT:  <Green Blink 3>
OUT:  <Red Off>
```

Summary

What we achieved

We were able to complete the full functionality of the game using all the hardware (2 LEDs, button and LCD screen) specified. The game can be played without any interaction with the console, you only need to interact with the pi once the program is run.

What wasn't achieved

We failed to get all interaction with the hardware done in inline assembler. We had all the coding involving LED pins done in assembler but some of them stopped working. We therefore returned to using C for those LED interactions. Furthermore, we had to use the lcdBinary.o you provided on your website to get the LCD screen working. We tried to write our own digitalWrite function but we couldn't get it to work

Outstanding features

- More inline assembly
- LCD screen working independently of the object file

What we have learned

- How to use header files
- How to build a basic game using hardware rather than just the console
- Importance of reusable code
- Teamwork when doing different tasks
- Using C and inline assembler to control to control hardware