Mastermind

F28HS CW2

Mohammed Yaseen Jamal (myj2, H00268891)
Amaan Akram (aa322, H00272130)

Problem Specification

This assignment involves an implementation of the Mastermind board-game for the Raspberry Pi, developed using peripherals consisting of two LEDs, a button for input, and an LCD for output. The game was developed using C and ARM Assembler.

The game involves two players to compete with each other; the ‘code maker’, and the ‘code breaker’. The code maker starts the game and creates a code:
The code is a sequence of 3 numbers with 3 different possibilities
e.g.

```
111
123
223
```

Are all valid codes. Once the code is set the code breaker must begin to guess and ‘break’ the code. If the code breaker guessing wrong he/she is given the number of times they almost got a correct value: If they guess the right number but in the wrong location.

e.g.

```
Code = 123      Guess=213       Almost= 2       Correct = 1

Code = 332      Guess=231       Almost= 1       Correct = 1

Code = 322      Guess=321       Almost= 0       Correct = 2
```

The code breaker keeps guessing until Correct = 3.

Our solution also contains a “debug” mode, where the program prints the secret sequence at the beginning created by the code maker, so that the answers given may be checked, and each entered sequence (guessed by code breaker) with its corresponding answer.

To produce a single executable program, the two source files can be linked and compiled using:

```
gcc gameplay.c peripherals.c -o mastermind
```

To run this normally:

```
sudo ./mastermind
```

(See Debug section for running in debug mode).

Assumptions

We assumed that the only values that a user can input is: 1,2 and 3.

If either the code breaker or the code maker does not enter a value for an input, then we assume the value is 1

If either the code breaker or the code maker enters a value greater than 3 for an input, then we assume that the value is 3

Hardware Setup

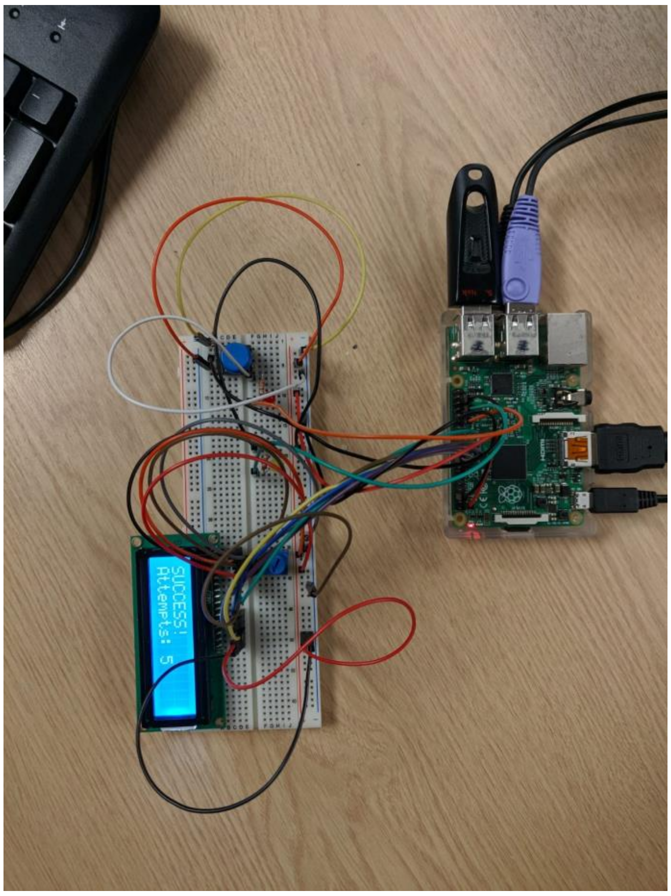
Peripherals used in the game consist of two LEDs (red and yellow), a button, and a Hitachi LCD (with attached potentiometer) – these were laid out onto an external breadboard, connected to the Raspberry Pi through GPIO connections.

Peripherals used in the game were laid out onto an external breadboard and connected to the Raspberry Pi through GPIO connections. A detailed description of the devices is given below:

Two LEDs were used as output devices: a yellow LED for data and a red LED for control information (e.g. to separate parts of the input and to start a new round). The red LED was connected to GPIO pin 5, and the yellow LED to GPIO pin 13.

A button was also connected as an input device, connected to GPIO pin 19.

Finally, a Hitachi HD44780U LCD was configured as an output device, with its data pins connected to GPIO pins 23, 10, 27 and 22.



Code Structure

As mentioned previously, solutions to the specification were developed using C and ARM assembler.

The implementation of setting up peripherals along with their functionalities are located in `peripherals.c`, coupled with the game controls found in `gameplay.c`

gameplay.c

Peripheral setup was performed by including setup() at the start of the gameplay, allowing the hardware functions specified in `peripherals.c` to be used.

User inputs (secret and guess sequences) are retrieved through `getInput()` and `getGuess()`.

Note that we also had to use `fflush(stdout)` at the beginning of the program because there was a problem with printing to the command line for debug mode.

peripherals.c

There is no main function in this file – instead, setup() is called, which sets up the peripherals for use in gameplay.c.

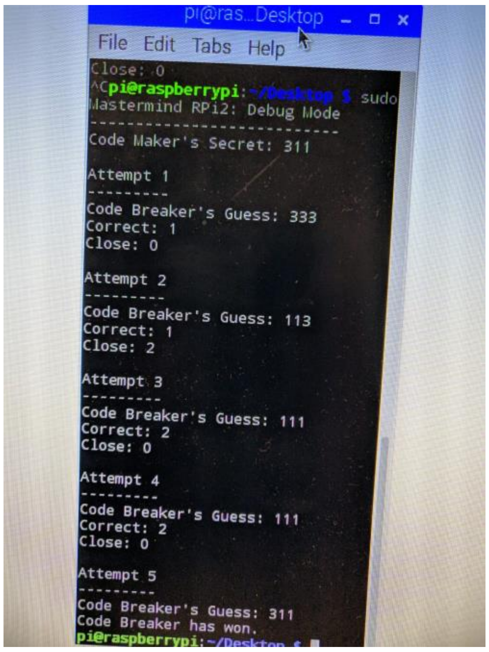
Functions Directly Accessing the Hardware

- `sendDataCmd (const struct lcdDataStruct *lcd, unsigned char data)`
- `lcdPutCommand (const struct lcdDataStuct *lcd, unsigned char command)`
- `lcdPut4Command (const struct lcdDataStruct *lcd, unsigned char command)`
- `lcdHome (struct lcdDataStruct *lcd)`
- `lcdClear (struct lcdDataStruct *lcd)`
- `lcdPosition (struct lcdDataStruct *lcd, int x, int y)`
- `lcdDisplay (struct lcdDayaStruct *lcd, int state)`
- `lcdCursor (struct lcdDayaStuct *lcd, int state)`
- `lcdCursorBlink (struct lcdDataStruct *lcd, int state)`
- `lcdPutchar (struct lcdDataStuct *lcd, unsigned char date)`
- `lcdPuts (struct lcdDataStruct *lcd, const char *string)`
- `digitalWrite (volatile unit32_t *gpio, int pin, int value)` *uses in-line assembler to "write" 1/0 to the pin*
- `say (char line1[16], char line2[16])`
- `wink (int LED)`
- `blink()`
- `pinMode (volatile unit32_t *gpio, int pin, int value) –` *uses in-line assembler to set the mode of the GPIO PIN*
- `button() –` *uses in-line assembler to get a click register*
- `clickCount()`

Debug Mode

To run in debug mode:

```
sudo ./mastermind -d
```



```
Close: 0
^Cpi@raspberrypi:~/Desktop $ sudo
Mastermind RPi2: Debug Mode
---------------------------
Code Maker's Secret: 311

Attempt 1
---------
Code Breaker's Guess: 333
Correct: 1
Close: 0

Attempt 2
---------
Code Breaker's Guess: 113
Correct: 1
Close: 2

Attempt 3
---------
Code Breaker's Guess: 111
Correct: 2
Close: 0

Attempt 4
---------
Code Breaker's Guess: 111
Correct: 2
Close: 0

Attempt 5
---------
Code Breaker's Guess: 311
Code Breaker has won.
pi@raspberrypi:~/Desktop $
```

Summary

Overall this coursework was very challenging for us. We did however fully meet all the criteria of the specification. In particular the LCD functions seemed most challenging and took up most of our time and effort.

Overall the game works as intended however one aspect we could improve on is inputting the values with the button. Sometimes the intended input's do not match the actual inputs, for example if you press the button too quick it will not register. If you press the button for too long it will count as multiple inputs.

From this coursework we have leveraged our experience in low-level C and ARM Assembler programming and now have a greater understanding of low level programming as a whole.

References

Peripheral.c adapted from Professor Wolfgang's sample sources:

http://www.macs.hw.ac.uk/~hwloidl/Courses/F28HS/srcs/index.html