F28HS – CW2 Mastermind Game Report

Group 19

Mohammed Faiz Iqbal – H00****
Mahak Bachani – H00****

Table of Contents

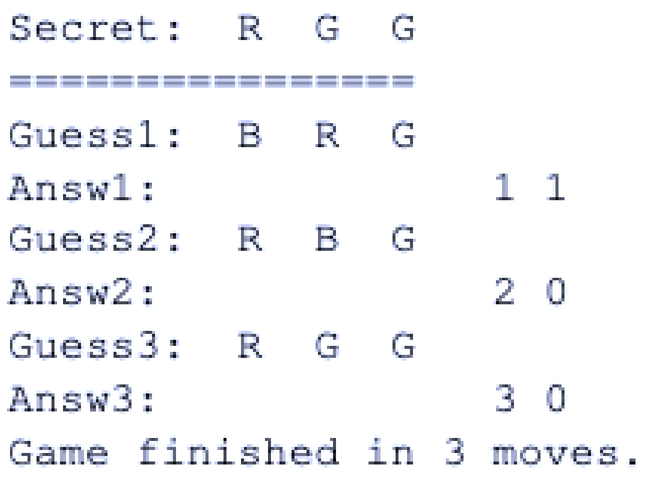
Abstract

Introduction

This project is based on the Mastermind board-game and uses C and ARM Assembler as its implementation language. This program can run on a Raspberry Pi 2 or a Raspberry Pi 3 with peripheral devices such as buttons and LED lights.

About the Game

The mastermind game involves two players, one person is a code keeper, and one person is the codebreaker. Before the game, a sequence length of N and a number of C colors for an arbitrary number of pegs are fixed. Then the code keeper selects N colored pegs and places them into a sequence of N slots. This (hidden) sequence comprises the code that should be broken. In turns, the codebreaker tries to guess the hidden sequence, by composing a sequence of N colored pegs, choosing from the C colors. In each turn the code keeper answers by stating how many pegs in the guess sequence are both of the right color and at the right position, and how many pegs are of the right color but not in the right position. The codebreaker uses this information to refine their guess in the next round. The game is over when the codebreaker successfully guesses the code, or if a fixed number of turns has been played. Below is a sample sequence of the game. (About the game adapted from the CW specification)



```
Secret:  R  G  G
================
Guess1:  B  R  G
Answ1:             1 1
Guess2:  R  B  G
Answ2:             2 0
Guess3:  R  G  G
Answ3:             3 0
Game finished in 3 moves.
```

Hardware Specification/Platform

As mentioned above, this program can run on a Raspberry Pi 2 or a Raspberry Pi 3 and requires the following peripheral devices:

- LEDS x2
- Resistors x3
- Button x1
- LCD x1
- Potentiometer x1

The setup and wiring are shown below in a Fritzing diagram, which shows the connections between the raspberry pi and the peripheral devices.

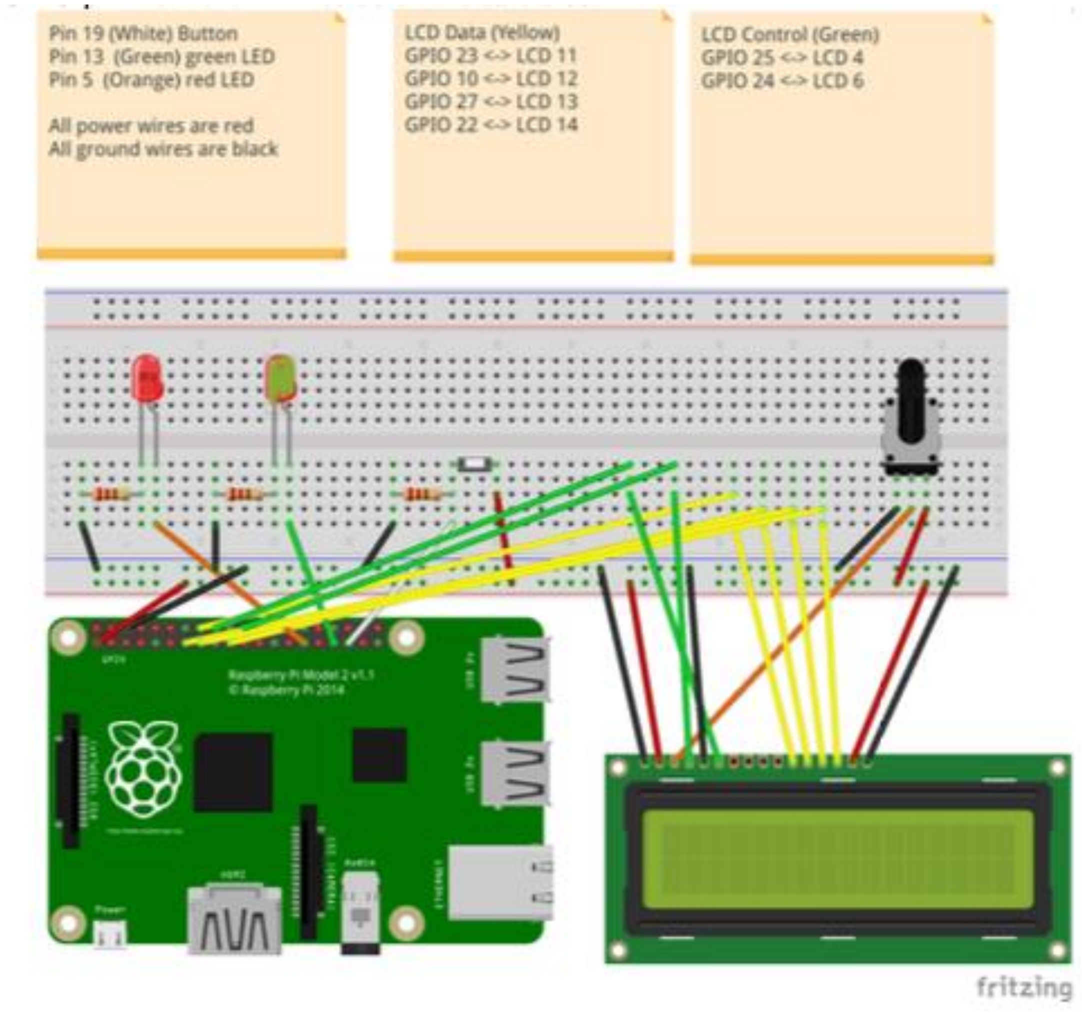


Fig1. Fritzing diagram adapted from CW specification.

LED lights

This project makes use of 2 LED lights. A red LED for Indication, which blinks to acknowledge an input and indicate successful entry and a green LED which is a Data LED that displays the number of times a user presses a button. The LED lights are programmed to turn on or off by changing mode of GPIO pins 5(Red LED) and 13(Green LED).

Resistors

Resistors are used here to limit the flow of current and avoid burning LEDs or other components. There are 3 resistors being used, 1 connected to the red LED, 1 to the green LED and 1 to the Button.

Button

The button is used to take input from the user (codebreaker). The codebreaker uses it to enter either R (1 press), G (2 presses) or B (3 presses). The button is connected using GPIO pin 19 on the raspberry pi.

LCD Display

The LCD display connected to the Raspberry Pi and is used to display the Guesses the codebreaker makes. The display uses a 5V power supply from the Raspberry Pi and is connected to several GPIO pins for LCD Control and LCD Data, mentioned in detail on the fritzing diagram (Fig1).

Potentiometer

The potentiometer is used to change the current that's supplied to the LCD, hence allowing us to change the LCDs contrast.

Code Structure

Hardware Interface section:

digitalWrite(): The digitalWrite function, which is defined in this code, uses the mapped GPIO base address to send a value (LOW or HIGH) to a particular pin number. It carries out the action using inline assembly code.
pinMode(): This function changes a GPIO pin's mode from OUTPUT to INPUT by using inline assembly code. By shifting the bit to select the mode, it determines the register to be used based on the pin number. After that, the register's changed value is saved.
writeLed(): In order to execute bitwise operations and memory addressing, this function uses inline assembly code to write a digital value (LOW or HIGH) to a particular pin on a Raspberry Pi's GPIO.
readButton(): Using inline assembly code, this method performs a bitwise AND operation on the GPIO memory address and the pin number, returning the state as an integer. The function reads the status (HIGH or LOW) of a button linked to a certain pin. It uses inline assembly code.
waitForButton(): This function uses the readButton() function to repeatedly check whether the button is pressed while waiting for a button input on a particular GPIO pin. When the button is pressed or after a certain number of checks, the loop ends.

Game Logic section:

initSeq(): This code creates a secret sequence by allocating memory for an array and using the rand() method to fill it with random numbers between 1 and 3.
showSeq(): This code uses the format described in the program specification to display the secret sequence as a series of colors (R for red, G for green, and B for blue).
countMatches(): This code checks each number in the two sequences to determine how many exact and approximate matches there are between the secret sequence and the codebreaker's guess sequence. To keep track of the matches, it makes use of a flag array.
showMatches(): This code shows how many times the secret sequence and the codebreaker's sequence match each other. It creates an array of digits from the

concatenated integer code, extracts the exact and approximative match counts, and outputs them.

readSeq(): The function takes an integer value and converts it into an array of digits, storing them in the seq array.

readNum(): The function reads an integer from stdin, transforms it to an array of digits, puts the digits in the array in reverse order, and returns a pointer to the array. The function allocates memory for an array of integers of length seqlen.

Timer section:

This code uses a collection of functions to build a time-out mechanism. While the timer_handler() function serves as a callback function triggered by an interval timer, the timeInMicroseconds() function returns the current time in microseconds. The start and stop timestamps are used to calculate the time-out interval. Together, these make it possible to create a time-out mechanism.

Main function sections:

The main function serves as the starting point of the game. There are different modes initialized (can be seen in testing section of this report).

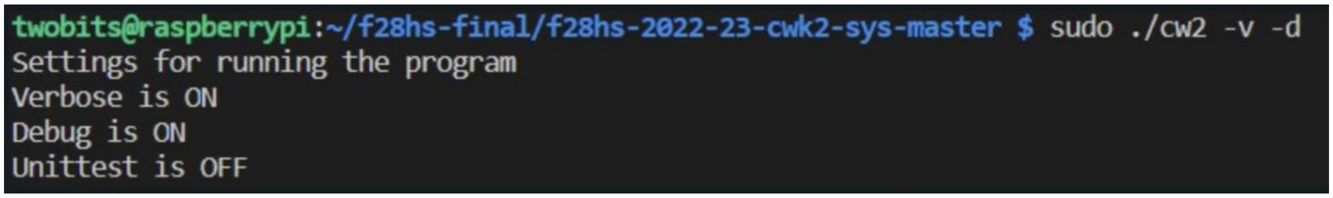
Testing

The application also provides a command line interface which can be used in combinations to test the games functionality automatically, the command line interface comes with a set of modes as explained below:

```
./cw2 [-v] [-d] [-u <seq1> <seq2>] [-s <secret sequence>]
```

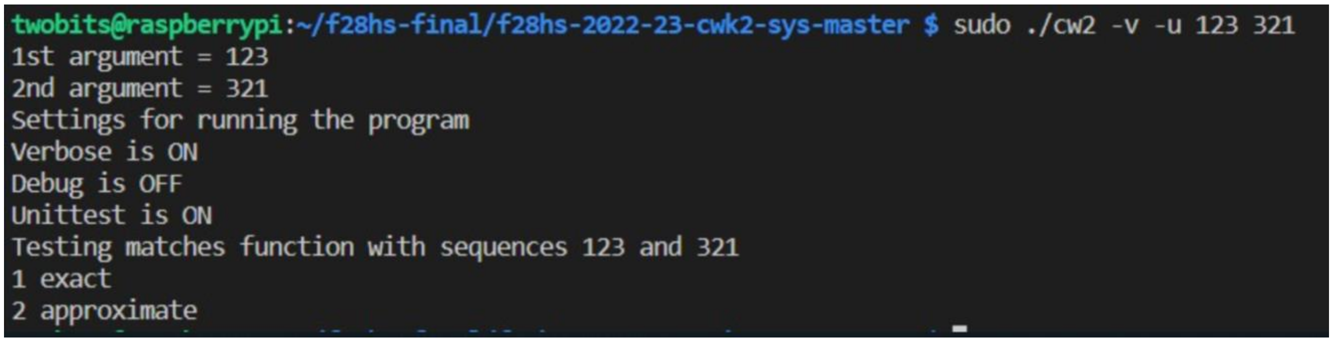
[-V] – Verbose Mode:

The verbose mode allows the user to run the program with additional details about the execution of the program. Example as shown below:



```
twobits@raspberrypi:~/f28hs-final/f28hs-2022-23-cwk2-sys-master $ sudo ./cw2 -v -d
Settings for running the program
Verbose is ON
Debug is ON
Unittest is OFF
```

Fig 2. Program running in Verbose + Debug.



```
twobits@raspberrypi:~/f28hs-final/f28hs-2022-23-cwk2-sys-master $ sudo ./cw2 -v -u 123 321
1st argument = 123
2nd argument = 321
Settings for running the program
Verbose is ON
Debug is OFF
Unittest is ON
Testing matches function with sequences 123 and 321
1 exact
2 approximate
```

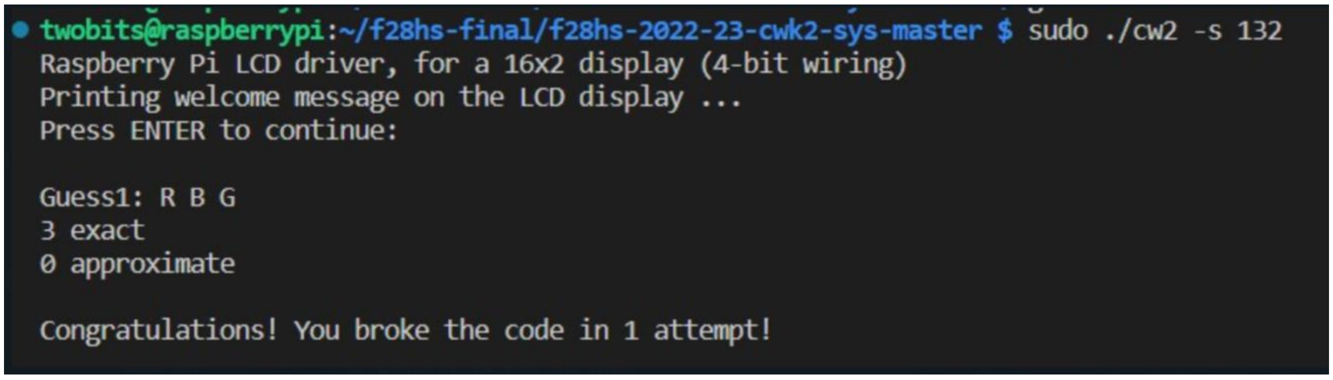
Fig 3. Program running with Verbose & Unit test.

[-d] – Debug Mode:

Debug mode allows the user to run the program showing the secret sequence. The guessed sequence and the answer.

[-s] – Secret mode:

Secret mode allows the user to act as a code keeper and set the Code themselves instead of it being randomly generated. This allows the user to test success cases of their program. Example:

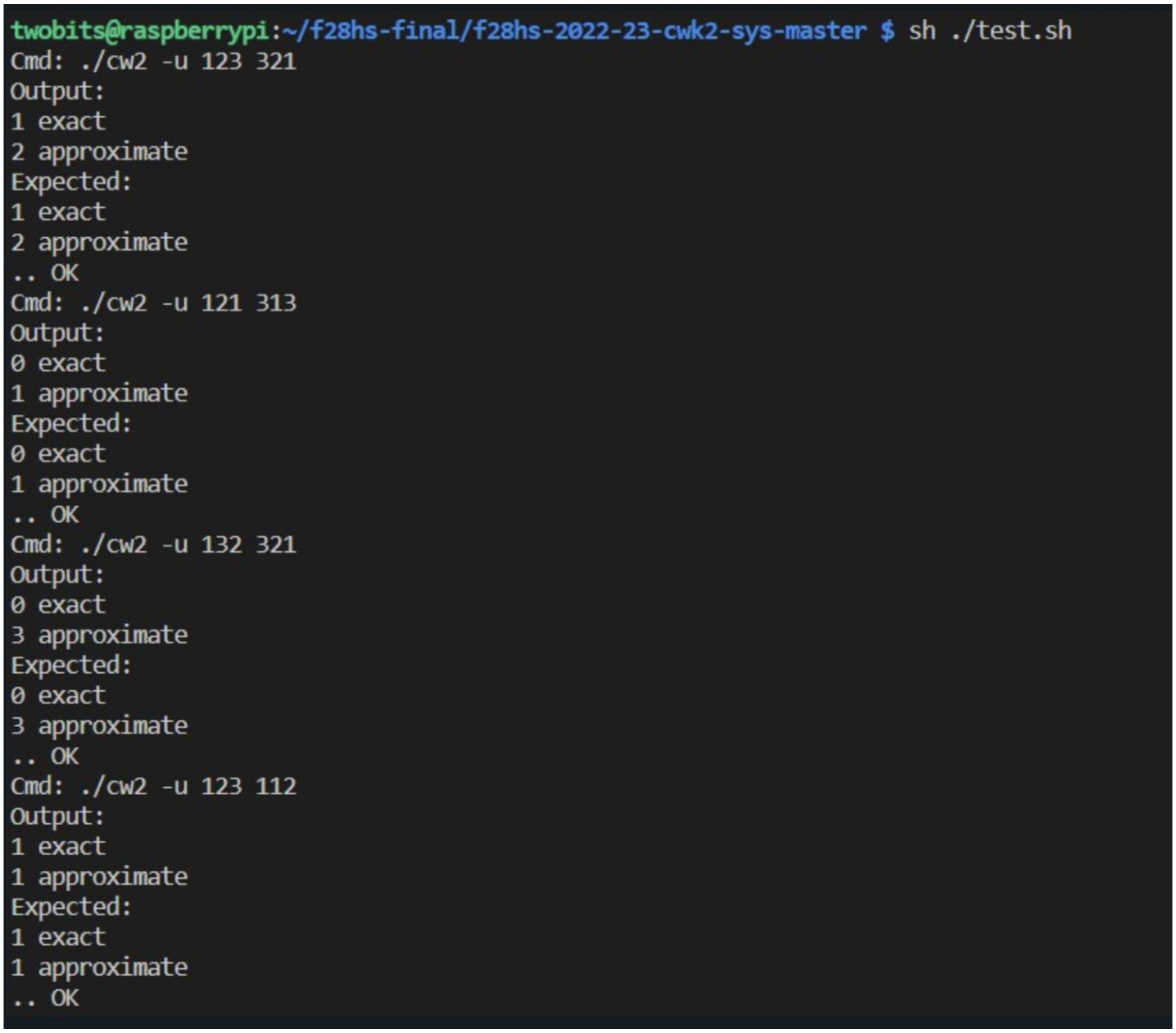


```
twobits@raspberrypi:~/f28hs-final/f28hs-2022-23-cwk2-sys-master $ sudo ./cw2 -s 132
Raspberry Pi LCD driver, for a 16x2 display (4-bit wiring)
Printing welcome message on the LCD display ...
Press ENTER to continue:

Guess1: R B G
3 exact
0 approximate

Congratulations! You broke the code in 1 attempt!
```

Fig 4. Program running in Secret mode.

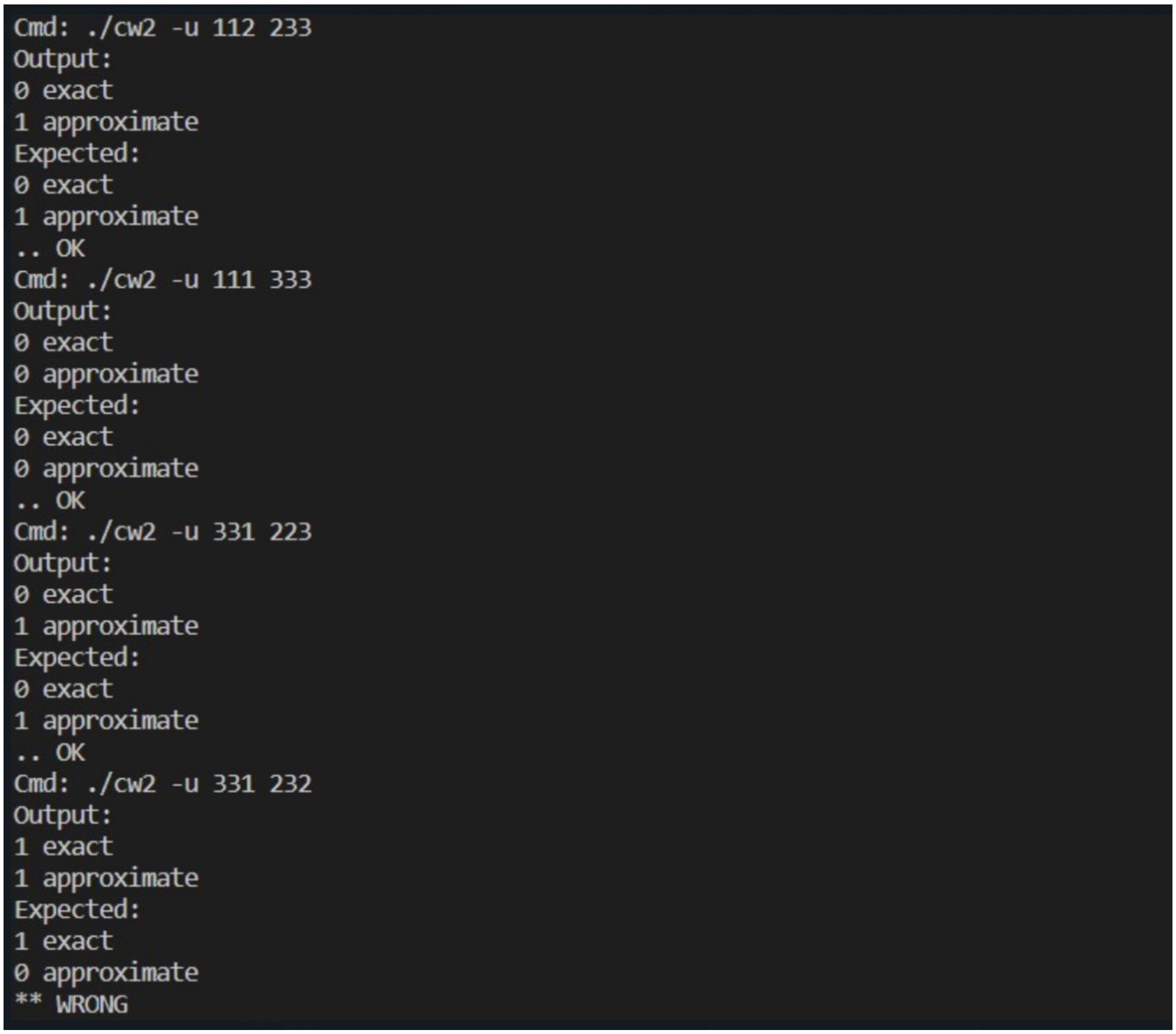
[-u] – Unit Testing

Unit testing allows the user to run a unit test on 2 input sequences and print the number of exact and approximate matches. Example:

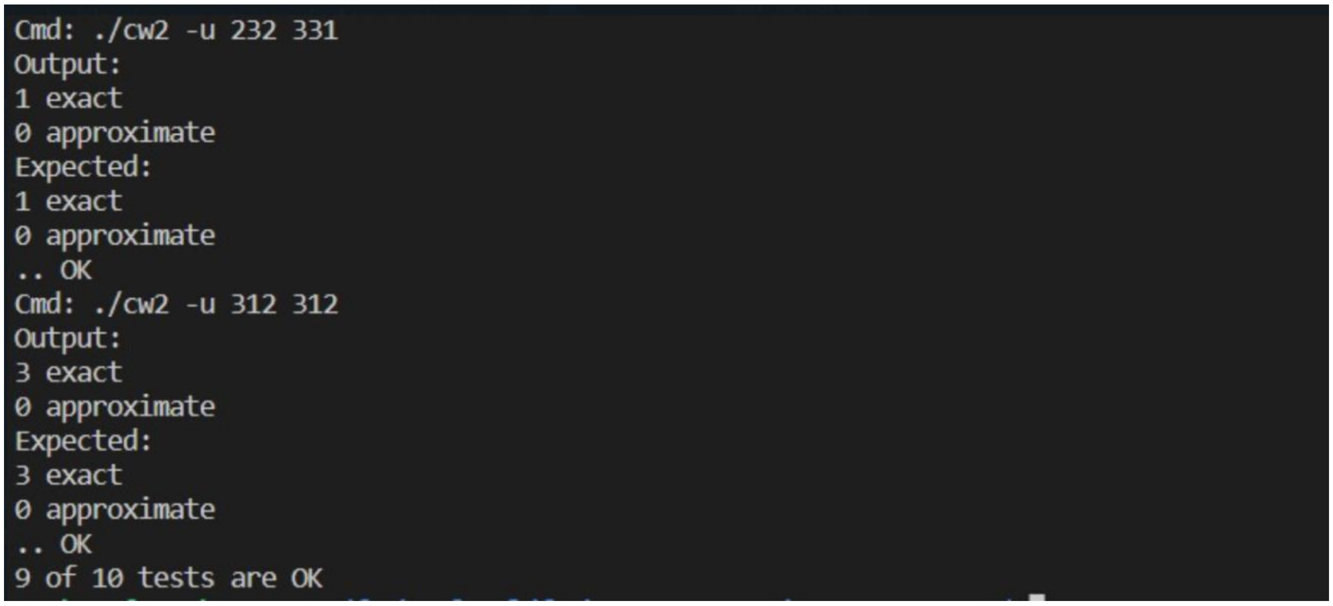


```
twobits@raspberrypi:~/f28hs-final/f28hs-2022-23-cwk2-sys-master $ sh ./test.sh
Cmd: ./cw2 -u 123 321
Output:
1 exact
2 approximate
Expected:
1 exact
2 approximate
.. OK
Cmd: ./cw2 -u 121 313
Output:
0 exact
1 approximate
Expected:
0 exact
1 approximate
.. OK
Cmd: ./cw2 -u 132 321
Output:
0 exact
3 approximate
Expected:
0 exact
3 approximate
.. OK
Cmd: ./cw2 -u 123 112
Output:
1 exact
1 approximate
Expected:
1 exact
1 approximate
.. OK
```

Fig. 5 Unit tests run on program.



```
Cmd: ./cw2 -u 112 233
Output:
0 exact
1 approximate
Expected:
0 exact
1 approximate
.. OK
Cmd: ./cw2 -u 111 333
Output:
0 exact
0 approximate
Expected:
0 exact
0 approximate
.. OK
Cmd: ./cw2 -u 331 223
Output:
0 exact
1 approximate
Expected:
0 exact
1 approximate
.. OK
Cmd: ./cw2 -u 331 232
Output:
1 exact
1 approximate
Expected:
1 exact
0 approximate
** WRONG
```



```
Cmd: ./cw2 -u 232 331
Output:
1 exact
0 approximate
Expected:
1 exact
0 approximate
.. OK
Cmd: ./cw2 -u 312 312
Output:
3 exact
0 approximate
Expected:
3 exact
0 approximate
.. OK
9 of 10 tests are OK
```

Peer assessment

Mahak Bachani
☆☆☆☆☆
Mohammed Faiz Iqbal
☆☆☆☆☆

Video Demo

Our video demo can be found on both Microsoft Stream and YouTube:
YouTube:
https://youtu.be/6qSMBq6Uids
Microsoft Stream:
https://heriotwatt-my.sharepoint.com/:v:/g/personal/mm2046_hw_ac_uk/EZk7Hk86OO9BnfT6gOOjM6oBlGobUIYt5iL6PSk_1VysyA?e=YUEDa9

Summary & Conclusion

We were able to successfully implement the mastermind game using C and ARM Assembly.
This Mastermind coursework helped us gain detailed understanding of the interaction between embedded hardware and external devices. The skills we gained from this coursework were resource management and time sensitive operations and identifying performance implications. And lastly this coursework helped build on our team working abilities.