F28HS CW2 Report

Rebekah Mandelbaum H00427974

Osaja Jafri H00443659

Group 21

Problem Specification

The MasterMind game on the RaspberryPi is a systems-level application implemented in C and ARM Assembler. In this implementation of MasterMind, the RasberryPi acts as the codekeeper. The codekeeper is the player who selects a sequence of coloured pegs. The RaspberryPi achieves this by generating a secret random sequence of coloured pegs. The player interacting with the system is known as the codebreaker. The codebreaker tries to guess the sequence through multiple rounds. The codekeeper processes each guess and outputs how many pegs are in the right position and the right colour, along with how many pegs are the right colour but in the wrong position after each guess. The game continues until the codebreaker successfully guesses the sequence or they have reached the maximum number of guesses allowed. The codebreaker enters their guesses using a button and the output of the game is displayed through LEDs and an LCD.

Hardware Specification

The RasberryPi 3 kit was used with the following hardware components:

1. LCD
2. Red LED
3. Green LED
4. 3 resistors
5. Potentiometer
6. Button

The wiring used was based on the fritzing diagram below.

LCD: The Raspberry Pi is connected to a 16x2 LCD display using a 5V power supply for LCD Data and LCD Control each and GPIO pins.

GPIO Connections for LCD Power:

1. LCD 2: 5V Pin
2. LCD 15: 5V Pin

GPIO Connections for LCD Control:

1. LCD 4: GPIO 25
2. LCD 6: GPIO 24

GPIO for LCD Data:

1. LCD 11: GPIO 23
2. LCD 12: GPIO 10
3. LCD 13: GPIO 27
4. LCD 14: GPIO 22

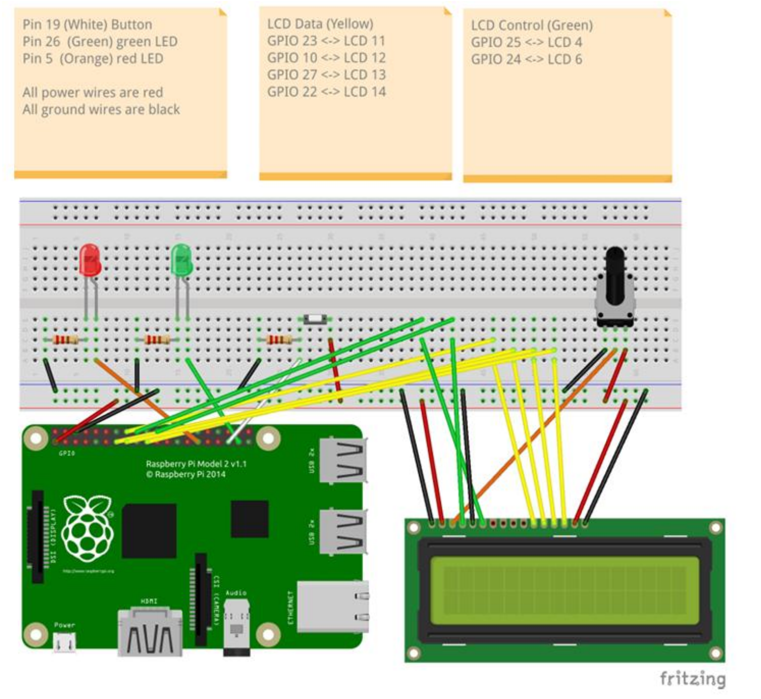
<u>LEDs:</u> Two LEDs are connected to the RaspberryPi through corresponding GPIO pins. A resistor is connected to each LED to control current flow. The current state of the pin is directly set by Inline Assembly code for turning on or off an LED.

GPIO pins used:

1. Red LED: GPIO pin 5
2. Green LED: GPIO pin 13

<u>Potentiometer:</u> The LCD Display is connected to LCD 3 and has a potentiometer attached to it which is used to adjust and control the contrast and visibility of the LCD display.

<u>Button:</u> User input is collected via a button that is connected using the GPIO pin 19. It utilizes a 5V power supply pin as well as a resistor which controls the level of current that flows into it.



Code Structure

The code gains structure from being split into specific functions. These functions are implemented using C, ARM Assembler or a combination of both.

- initSeq: Generates a random integer array called *theSequence* with a length of *seqlen*, containing random values between 1 and 3 (exclusive).
- showSeq: Displays the randomly generated secret sequence created by the application, executed in debug mode.
- countMatches: Compares *seq2* against *seq1* to count the exact and approximate matches, returning these values either as a single encoded result or as a pair of values via a pointer.
- showMatches: Calls the countMatches function and prints the exact and approximate match counts in the terminal.
- readSeq: Parses an integer input as a list of digits and stores them in *seq*, used for processing command-line options such as -s or -u.
- blinkN: Blinks a given LED a specified number of times with a set delay.
- main: This is the entry point of the application, where the game logic is initiated. It contains the structure of the game and handles various modes and flags that the user may select to execute the application.

Design decisions

Use of Timer:
A 5-second timer is used when retrieving user input. The timer starts only when the user presses the button for the first time while entering a number, ensuring that subsequent inputs can be read within the time limit. This approach enhances both the accuracy and efficiency of user input handling.

Pointers:
In the program, an array of pointers is used to store the exact and approximate matches. Each match is allocated dynamically at runtime with a size of 2. This approach allows for efficient access to each match by referencing the values directly, eliminating the need for additional overhead when retrieving each value independently.

Assembly Implementation:
Inline assembly is used in the *lcdBinary* file to interact with hardware components, such as the button and LEDs. By directly controlling the registers through assembly, the program achieves greater accuracy and improved performance, as it minimizes overhead and speeds up the execution of hardware-related tasks.

Functions Directly Accessing Hardware

pinMode: This method changes the GPIO pin to input mode or output mode depending on the given argument. Implemented mainly using C.

writeLED: This method turns the LED on or off. The function either sets or clears the register based on the variable value, this is implemented using C. The set or clear code is modified to turn the LED on or off respectively through inline ARM Assembler.

readButton: A code snippet detects when a user clicks a button. By accessing the gplev0 register, the specific input/output pin is determined through the implementation of C. The inline assembly code is used to then read the value from the identified register and output it.

waitForButton: This method implements the readButton function and allows time for the user to press the button.

Matching Function

The function is named matches and is implemented using ARM Assembly. It compares the generated secret random sequence, and the user entered guess sequence.

Inputs:

- R0: A pointer to the secret random sequence (an integer array).
- R1: A pointer to the guess sequence (an integer array)

Outputs:

R0: The result of the matching function, returned as a two-digit decimal number:

- The tens place of the number represents the number of exact matches (pegs of the right colour and in the correct position).
- The ones place represents the number of approximate matches (pegs of the correct colour but in the wrong position).

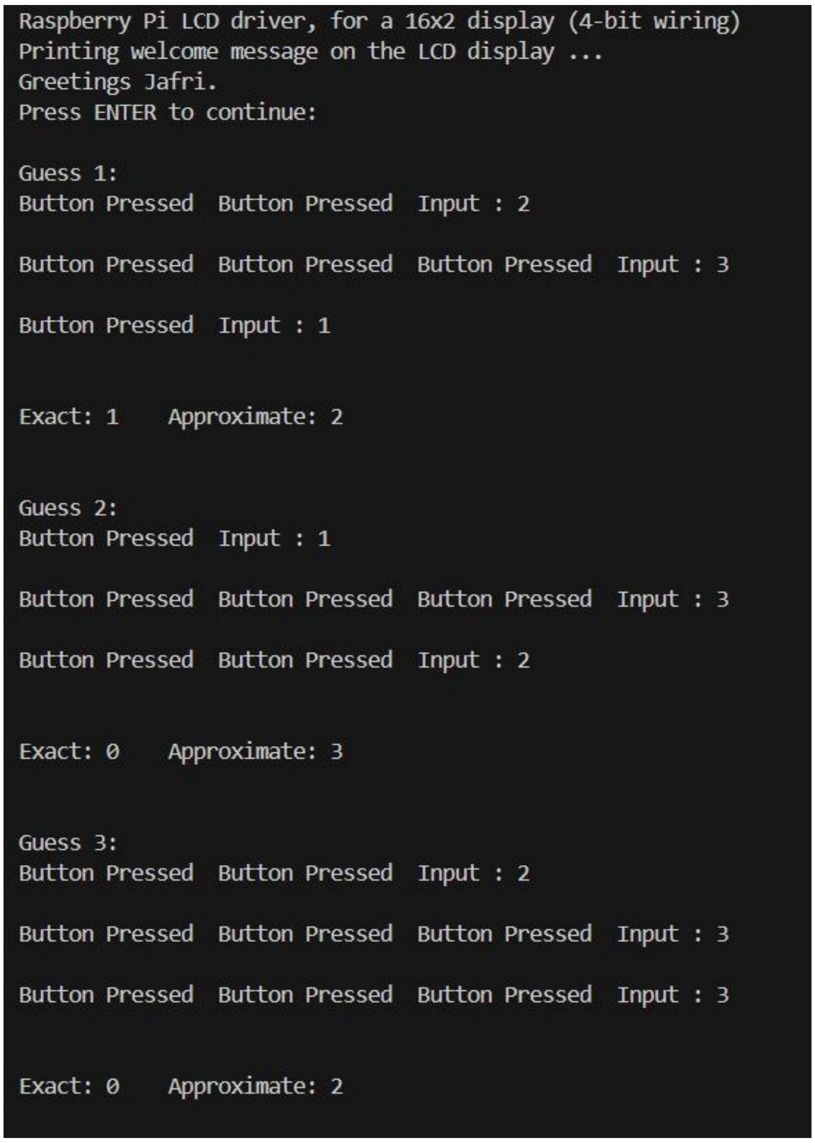
Example:

If the secret sequence is [1, 2, 1] and the guess sequence is [3, 1, 3], the output will be:
Exact: 0

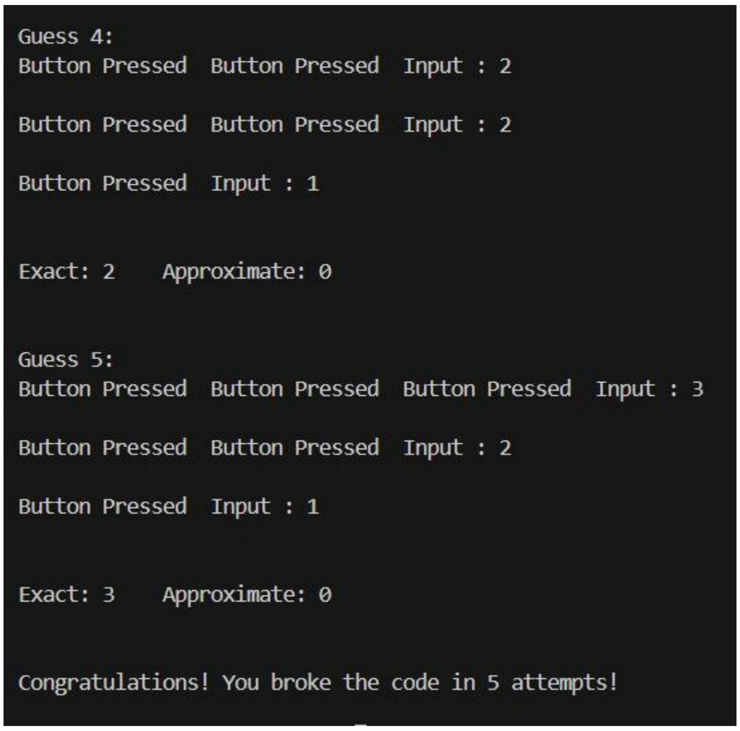
Approximate: 1

Example Set of Output

Successful guess:

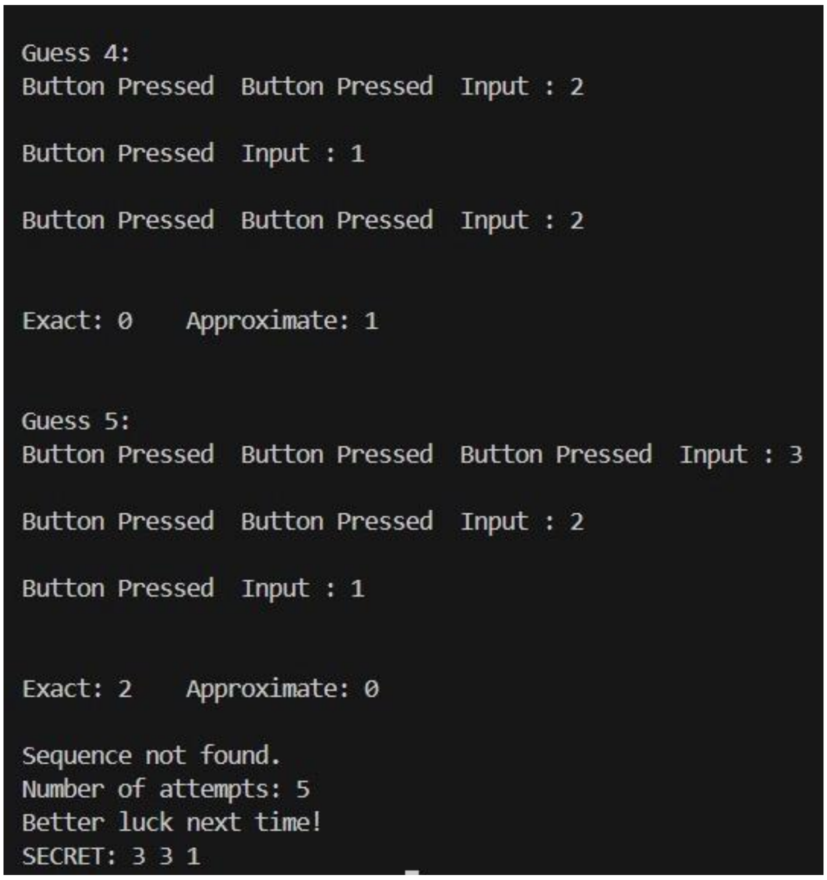


```
Raspberry Pi LCD driver, for a 16x2 display (4-bit wiring)
Printing welcome message on the LCD display ...
Greetings Jafri.
Press ENTER to continue:

Guess 1:
Button Pressed  Button Pressed  Input : 2

Button Pressed  Button Pressed  Button Pressed  Input : 3

Button Pressed  Input : 1


Exact: 1    Approximate: 2


Guess 2:
Button Pressed  Input : 1

Button Pressed  Button Pressed  Button Pressed  Input : 3

Button Pressed  Button Pressed  Input : 2


Exact: 0    Approximate: 3


Guess 3:
Button Pressed  Button Pressed  Input : 2

Button Pressed  Button Pressed  Button Pressed  Input : 3

Button Pressed  Button Pressed  Button Pressed  Input : 3


Exact: 0    Approximate: 2
```



```
Guess 4:
Button Pressed  Button Pressed  Input : 2

Button Pressed  Button Pressed  Input : 2

Button Pressed  Input : 1


Exact: 2    Approximate: 0


Guess 5:
Button Pressed  Button Pressed  Button Pressed  Input : 3

Button Pressed  Button Pressed  Input : 2

Button Pressed  Input : 1


Exact: 3    Approximate: 0


Congratulations! You broke the code in 5 attempts!
```

Incorrect guess:



```
Guess 4:
Button Pressed  Button Pressed  Input : 2

Button Pressed  Input : 1

Button Pressed  Button Pressed  Input : 2


Exact: 0    Approximate: 1


Guess 5:
Button Pressed  Button Pressed  Button Pressed  Input : 3

Button Pressed  Button Pressed  Input : 2

Button Pressed  Input : 1


Exact: 2    Approximate: 0

Sequence not found.
Number of attempts: 5
Better luck next time!
SECRET: 3 3 1
```

Summary

The MasterMind game was completed using both C and ARM Assembler. All required functionality was achieved and coded successfully. A user can play the game with the correct and fully functioning game logic, give input through the button and receive output through the LEDs and LCD.

In conclusion, we gained an understanding of low-level code used to control the interaction between embedded hardware and external devices. We also learnt the importance of testing after coding each function.