F28HS Coursework-2

Members: Behla Mustansir Patrawala and Mehak Shameer

Course: Hardware Software interface

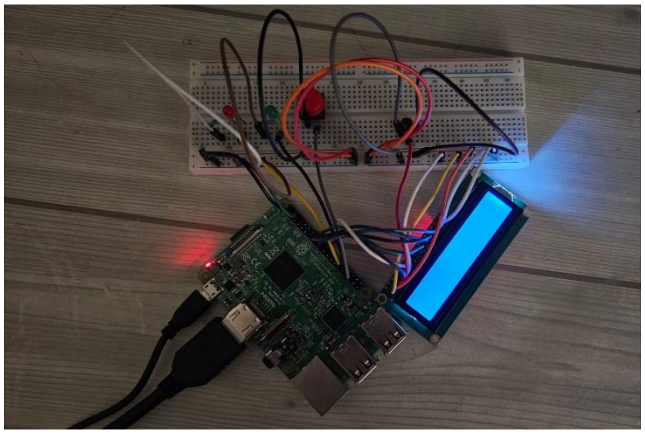
1. Problem Specification

The aim of this coursework is to create a simple guessing game: MasterMind using C and ARM assembler as implementation languages on RPi with attached devices as shown in section 2.

The task involves implementing a simplified version of the MasterMind board game using C and ARM assembler on a Raspberry Pi. The game comprises two players: a code keeper and a codebreaker. The code keeper selects a sequence of colored pegs of length N from C available colors and conceals it. The codebreaker's objective is to deduce this hidden sequence through a series of guesses. After each guess, the code keeper provides feedback on the correctness of the guessed sequence, indicating the number of pegs of the correct color in the correct position and the number of pegs of the correct color but in the wrong position. The game continues until the codebreaker successfully guesses the sequence or reaches a predefined number of turns.

2. Hardware Specification

This coursework requires us to execute the application on a Raspberry Pi 3 connected to multiple devices via a breadboard. This course's devices include two LEDs (One green (for data) connected to GPIO pin 13 and one red (for control information) connected to GPIO pin 5), a button (Used as an input device and connected to GPIO pin 19), three resistors, an LCD display (Connected to GPIO pins 23, 10, 27, and 22), and a potentiometer (LCD is connected to a potentiometer for adjusting the contrast of the display). All are connected to the RPi 3's GPIO pins via the breadboard using either the jumper cables or directly connecting it to the GPIO pins. Wiring is done according to the Fritzing diagram in the class instructions.



3. Code Structure

initSeq:

Functionality: Initializes the secret sequence of N length (3) and C values (1/2/3) for the codebreaker to guess.
Description: This function initializes a random sequence of LEDs using the srand() function to generate a random seed. The sequence is stored in an array with length N (3) and each element represents a color value (1/2/3).

showSeq:

Functionality: Displays the secret sequence on the terminal window.
Description: This function takes a pointer to the sequence array as input and prints the color values to the terminal. It's primarily used for debugging purposes to verify the secret sequence.

countMatches:

Functionality: Counts the number of matches between two sequences.
Description: The function takes two sequences seq1 and seq2 as input and
counts the number of exact matches and approximate matches between them. This function iterates through both sequences simultaneously, comparing the characters. If the character is equal, it increments the exact count. If it is not equal, it updates the secret array and guess array to track the occurrences of each character in both sequences. The function calculates the approximate matches by taking the minimum value of each character in secret and guess array then sums up the minimum values to get result. This function returns pointer to an integer array that stores the exact and approximate matches. The first element of the array represents the number of exact matches, and the second element represents the number of approximate matches.

showMatches:

Functionality: Displays the exact and approximate matches between two sequences.
Description: This function checks if any inputs are invalid or if the length of the sequence is less than or equal to zero. If the exact match is equal to the sequence length, it prints an "Exact match!" message along with the number of approximate matches. Otherwise, it prints the number of exact and approximate matches.

readSeq(length):

Functionality: Parses an integer value as a list of digits and stores them in an array.
Description: This function takes an integer value and parses it as a list of digits, storing them in the seq array. It iterates over the integer value, using modular arithmetic to extract each digit and storing them in the seq array in reverse order. Finally, it reverses the order of the digits in the array to put them back in the correct order. This function is primarily used for processing command-line arguments with options -s or -u.

timeInMicroseconds:

Functionality: Returns the current time in microseconds.
Description: This function fetches the current time in microseconds using system-provided timing functions or hardware timers.

timer_handler:

Functionality: Updates a timed_out variable to indicate that the timer has expired.
Description: This function is invoked when the timer interrupt signal is triggered. It updates a timed_out variable to indicate that the timer has expired, which can be used in the main program to indicate the end of the timer.

initITimer:

Functionality: Initializes a non-recurring timer based on the timeout provided as an argument.
Description: This function initializes a timer using system or hardware timers, configuring it as a non-recurring timer. It's primarily used in the main program to set up timed events or intervals.

delay & delayMicroseconds:

Functionality: Introduces a delay in the execution of the program for a specified amount of time.
Description: These functions halt the execution of the program for the specified duration using system-provided timing functions or loops. They are used in the main function to create delays, such as in displaying outputs or blinking LEDs.

blinkN:

Functionality: Blinks an LED a specified number of times.

Description: This function toggles the state of the LED connected to the specified pin on and off for the specified number of times, with each blink lasting for the specified duration. It uses the delay() function to control the timing of the blinks.

Main:

The main function is the program's entry point and initiates all other game-related functions. This function also initializes variables, handles command-line parameters, and establishes the secret sequence.

4. Design Choices Applied

Use of Arrays:

We opted to employ arrays for storing data such as the sequence. This decision was motivated by the efficiency offered by arrays in memory utilization. By storing data in contiguous blocks of memory, arrays facilitate quicker access to data and minimize memory fragmentation. Consequently, our program benefits from faster data retrieval without incurring excessive memory overhead.

Optimized Loops:

We crafted our loops with optimization in mind, aiming to minimize their impact on resource consumption. For instance, in the initSeq() function, we employed a for loop to efficiently iterate over the sequence and generate random values between 1 and 3. Similarly, a while loop was utilized in the readSeq() function for reading user input. By ensuring that our loops are well-structured and efficient, we mitigate unnecessary resource usage while maintaining code clarity and functionality.

Utilization of Pointers:

Leveraging pointers in our code facilitates efficient memory management and data access. By dynamically allocating memory at runtime and reducing memory requirements per operation, pointers contribute to resource optimization. Moreover, they enhance program speed by enabling direct memory access.

Integration of Timer:

Incorporating delay functions via timers introduces a delay between inputs, thereby enhancing efficiency and accuracy so that at no time in process execution user can enter two inputs at a time. By allowing for controlled timing between operations, we optimize resource utilization by preventing unnecessary processing overhead. This ensures that our program operates smoothly and reliably without undue strain on system resources.

Inline Assembly:

Using inline assembly simplifies access to the hardware's GPIO pins, which include buttons and LEDs. This decision is based on a desire to improve program speed and accuracy by allowing direct control of hardware registers from within our code. Inline assembly improves program responsiveness and efficiency by skipping abstract layers and connecting directly with hardware.

5. Functions directly accessing the hardware

Below is a list of functions that directly accesses the hardware components (LEDs, button, and LCD display) along with the parts of each function that use assembler and which parts use C:

- digitalWrite:

  This function manages the state of a pin, controlling LEDs, LCD, and buttons. While the C code handles parameter passing and return values, the inline assembly code undertakes low-level register operations.

- pinMode:

  This function assigns the input/output mode to a specific pin based on the provided argument. While the C code handles setting certain values, the inline assembly code manages the mode itself.

- writeLED:

  This function essentially mirrors digitalWrite, hence it solely invokes digitalWrite in C.

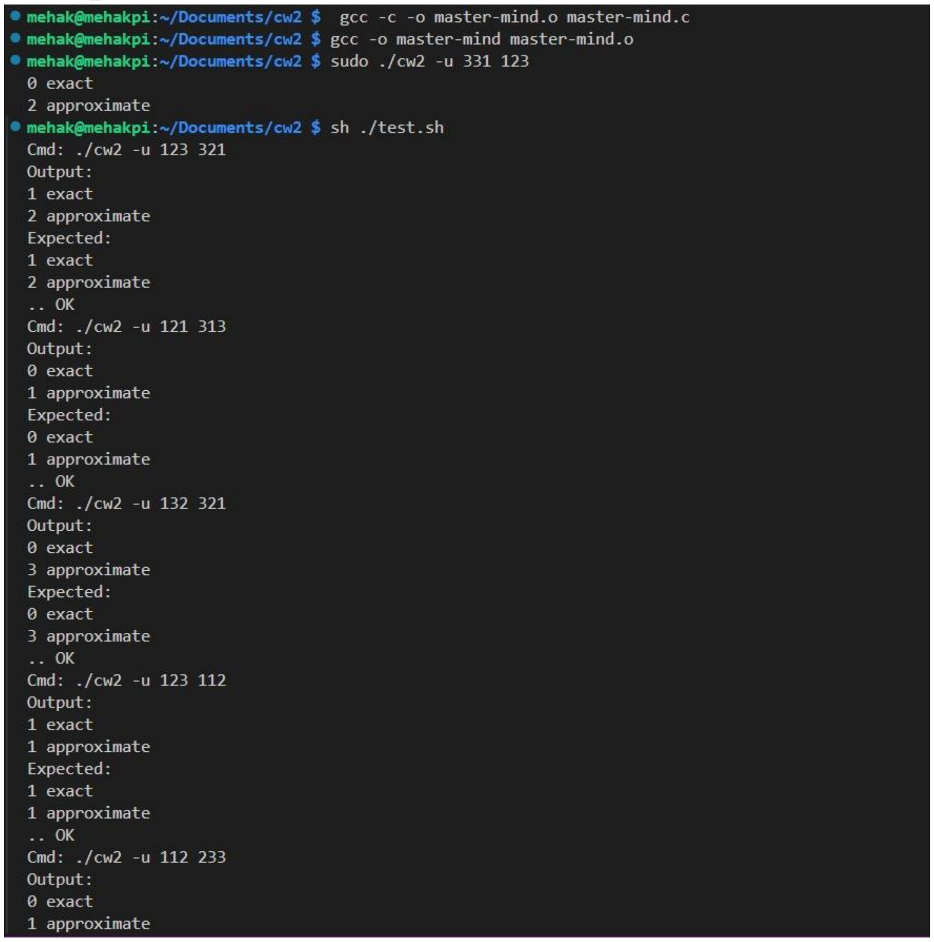
- readButton:

  Responsible for reading the current state of the button. Implemented using a combination of C and Arm assembly language.

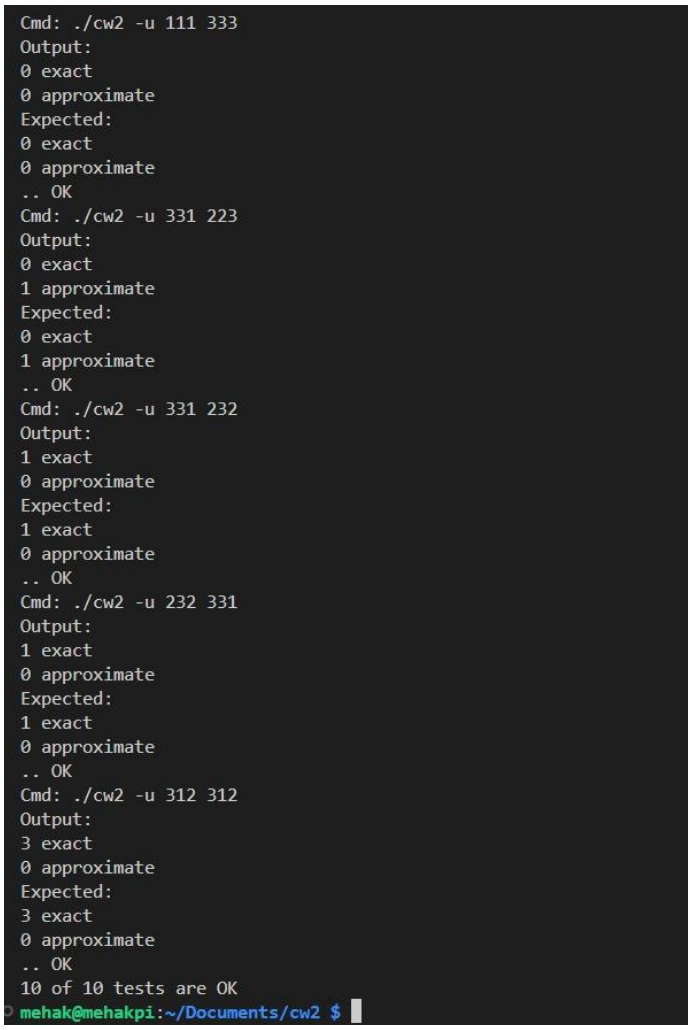
- waitForButton:

This function halts execution until the user presses the button. Implemented solely in C.

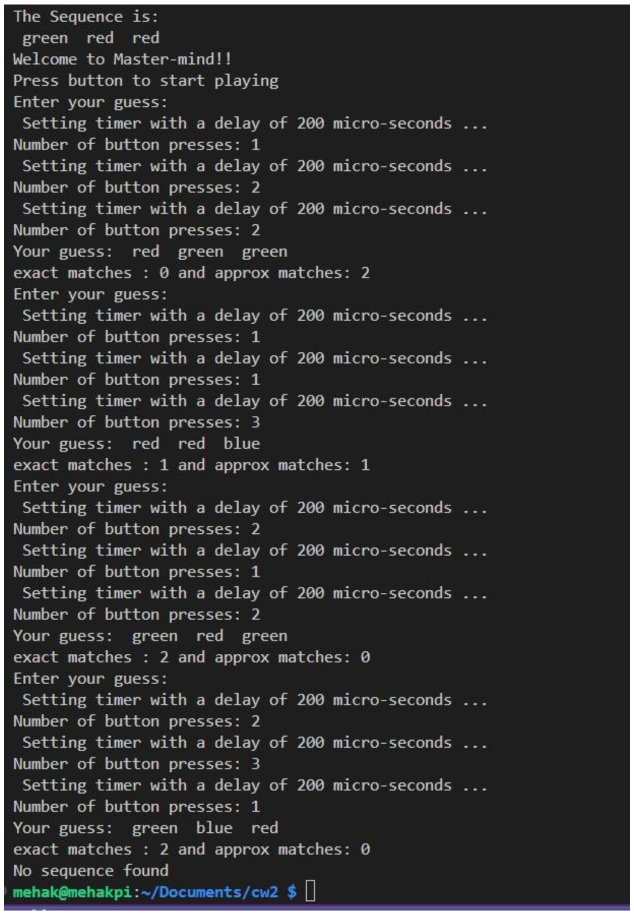
6. Sample execution



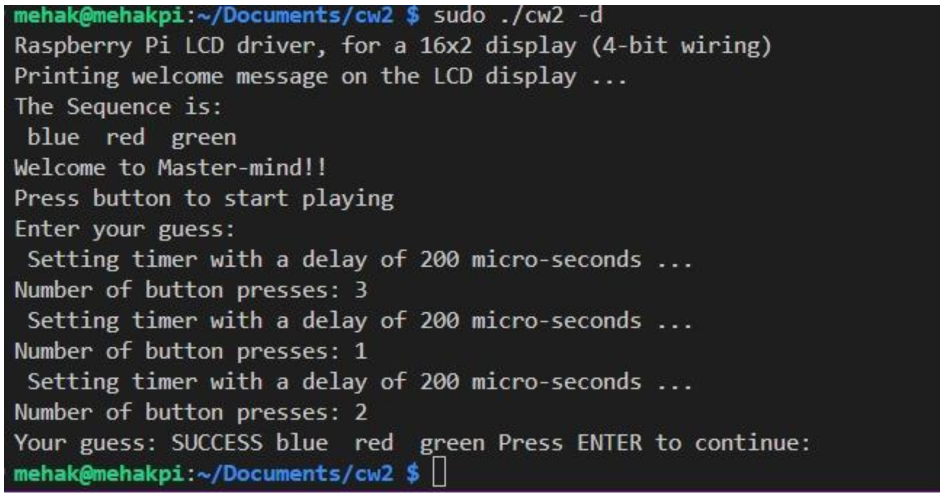
```
mehak@mehakpi:~/Documents/cw2 $  gcc -c -o master-mind.o master-mind.c
mehak@mehakpi:~/Documents/cw2 $ gcc -o master-mind master-mind.o
mehak@mehakpi:~/Documents/cw2 $ sudo ./cw2 -u 331 123
0 exact
2 approximate
mehak@mehakpi:~/Documents/cw2 $ sh ./test.sh
Cmd: ./cw2 -u 123 321
Output:
1 exact
2 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
Cmd: ./cw2 -u 112 233
Output:
0 exact
1 approximate
```



```
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
0 approximate
Expected:
1 exact
0 approximate
.. OK
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
10 of 10 tests are OK
mehak@mehakpi:~/Documents/cw2 $
```



```
The Sequence is:
 green  red  red
Welcome to Master-mind!!
Press button to start playing
Enter your guess:
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 1
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 2
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 2
Your guess:  red  green  green
exact matches : 0 and approx matches: 2
Enter your guess:
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 1
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 1
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 3
Your guess:  red  red  blue
exact matches : 1 and approx matches: 1
Enter your guess:
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 2
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 1
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 2
Your guess:  green  red  green
exact matches : 2 and approx matches: 0
Enter your guess:
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 2
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 3
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 1
Your guess:  green  blue  red
exact matches : 2 and approx matches: 0
No sequence found
mehak@mehakpi:~/Documents/cw2 $ 
```



```
mehak@mehakpi:~/Documents/cw2 $ sudo ./cw2 -d
Raspberry Pi LCD driver, for a 16x2 display (4-bit wiring)
Printing welcome message on the LCD display ...
The Sequence is:
 blue  red  green
Welcome to Master-mind!!
Press button to start playing
Enter your guess:
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 3
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 1
 Setting timer with a delay of 200 micro-seconds ...
Number of button presses: 2
Your guess: SUCCESS blue  red  green Press ENTER to continue:
mehak@mehakpi:~/Documents/cw2 $ 
```

7. Summary

In our coursework project, we created a user-friendly interface for controlling an LED and a button on a Raspberry Pi using a combination of C code and inline assembly language designed specifically for the ARM architecture. We used the mmap() function to read GPIO registers and efficiently control pin modes and values. We built methods that allowed us to set pin modes, write pin values, read button inputs, and generate blinking LED sequences with exact timing using delay functions. We successfully tested the Raspberry Pi and observed the desired LED blinking and button input responses. We highlighted potential modifications, such as incorporating error handling and input validation, to protect the code from crashes and incorrect results when encountering invalid inputs or execution failures. This project provided valuable insights into low-level programming methods for interfacing with hardware components, and deepened our comprehension of computer systems, encompassing CPU functionality, memory management, and input/output mechanisms.