Game Application Report

Problem Specification:

The game application is a code-breaking game where the Raspberry-Pi acts as a code-keeper, generating a random, hidden sequence. The user acts as a codebreaker, entering the guess sequences. Game has several rounds, and, in each round, the player enters a sequence of numbers using a button. Timers are used to separate inputs. The green led acknowledges the input of a number by blinking once. After acknowledgment, the green LED blinks a number of times equal to the value of input. After the player has inputted all the colors of a guess, the red LED blinks twice to indicate the end of an input. The application calculates the number of exact (same color same location) and approximate matches (same color different location) between the guessed and hidden sequence. The green LED first blinks the number of exact matches, then the red LED blinks once as a separator and then green blinks again with the number of approximate matches. The red LED blinks thrice to indicate the start of a new round. If the hidden sequence is guessed correctly, the green LED blinks thrice while the red LED is turned ON, otherwise the game is preceded to the next round. On successful completion of the game, the LCD displays a message “SUCCESS” and the number of attempts taken by the player.

Hardware Specification and Wiring:

LEDs are connected to GPIO pins to control their illumination. The button is connected for input. LCD display is connected to GPIO pins for data and control signals.

Wiring:

Green LED: GPIO PIN 13

Red LED: GPIO PIN 5

Button: One leg of button connected to GPIO PIN 19 and the other leg to GND.

A pull-up resistor is used between the GPIO pin and a 3.3V.

LCD Display:

Connect pin 1 to GND.

Connect pin 2 to 5V.

Connect pin 4 to GPIO pin 25.

Connect pin 5 to GND.

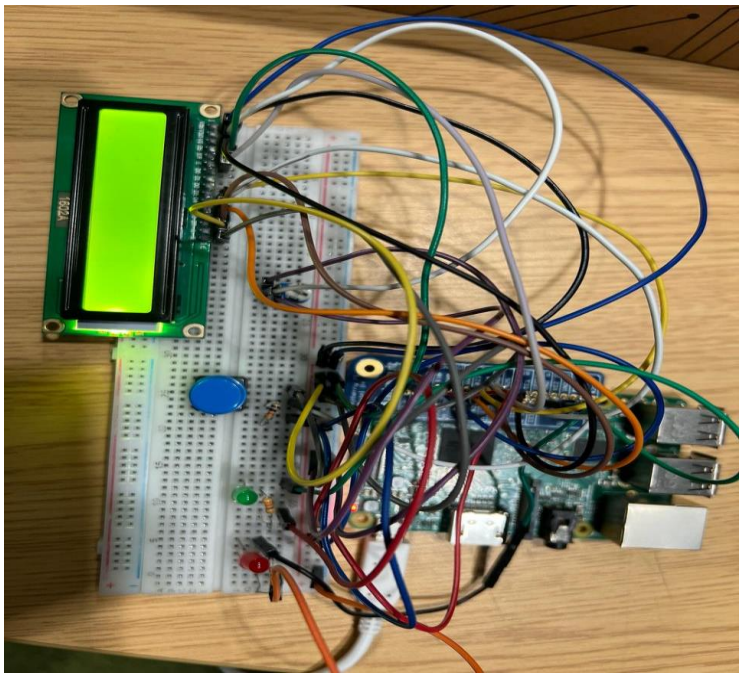
Connect pin 6 to GPIO pin 24.

Connect pins 11-14 to GPIO pins 23,10,27,22 respectively.

Connect pin 15 to 5V.

Connect pin 16 to GND.

Here's our circuit with the connections mentioned above along with the test cases tested.



```
asmalaiba@asma:~/Documents/f28hs-2023-24-cwk2-sys $ ./testm
Running tests of matches function with 10 pairs of random input sequences ...
Matches (encoded) (in C):   11
Matches (encoded) (in Asm): 11
1 exact
1 approximate
1 exact
1 approximate
__ result OK
Matches (encoded) (in C):   11
Matches (encoded) (in Asm): 11
1 exact
1 approximate
1 exact
1 approximate
__ result OK
Matches (encoded) (in C):   20
Matches (encoded) (in Asm): 20
2 exact
0 approximate
2 exact
0 approximate
__ result OK
Matches (encoded) (in C):   10
Matches (encoded) (in Asm): 10
1 exact
0 approximate
1 exact
0 approximate
__ result OK
Matches (encoded) (in C):   11
Matches (encoded) (in Asm): 11
1 exact
1 approximate
1 exact
1 approximate
__ result OK
Matches (encoded) (in C):   10
Matches (encoded) (in Asm): 10
1 exact
0 approximate
1 exact
0 approximate
__ result OK
Matches (encoded) (in C):   20
Matches (encoded) (in Asm): 20
2 exact
```

```
0 approximate
2 exact
0 approximate
__ result OK
Matches (encoded) (in C):   10
Matches (encoded) (in Asm): 10
1 exact
0 approximate
1 exact
0 approximate
__ result OK
Matches (encoded) (in C):   20
Matches (encoded) (in Asm): 20
2 exact
0 approximate
2 exact
0 approximate
__ result OK
Matches (encoded) (in C):   12
Matches (encoded) (in Asm): 12
1 exact
2 approximate
1 exact
2 approximate
__ result OK
10 out of 10 tests OK
asmalaiba@asma:~/Documents/f28hs-
```

Code Structure:

These are functions implemented to control GPIO pins:

- **'digitalWrite':** This function writes a digital value (ON or OFF) to a specific GPIO pin. If the value if OFF, it clears the bit corresponding to the pin in the GPIO output set register (offset 10 in GPIO memory map). If the value is ON, it sets the bit corresponding to the pin in the GPIO output set register (offset 7 in the GPIO memory map).
- **'pinMode':** This function sets the mode (INPUT or OUTPUT) of a specific GPIO pin.I uses bitwise operation to manipulate the bits in the GPIO function select register. If the modeis OUTPUT, it sets the mode for bits to 001 else 000 (INPUT).

- **'readButton':** This function is used to read the state of the button. It checks the bit corresponding to the pin in the GPIO input level register (offset 13 in the GPIO memory map). If the bit is not set (button not pressed), it returns OFF. If the bit is set (button pressed), it returns ON.
- **'waitForButton':** This function waits for a button to press on a specific GPIO pin. It continuously reads the button state using readButton() in a loop. If the button state is ON, indicating a button press, it returns 1 to indicate success. Otherwise, it waits for a short period using nanosleep() before checking the button state again.

These are functions implemented for the game sequence:

- **'initSeq':** This function initializes the game sequence by generating a random sequence of numbers of length SEQL.
- **'showSeq':** This function displays the game sequence on the terminal window.
- **'countMatches':** This function compares seq1 and seq2 and counts the number of exact matches and approximate matches. It returns a single integer where the upper 4 bits represent the exact matches, and the lower 4 bits represent the approximate matches.
- **'showMatches':** This function takes the result from countMatches( ) and prints exact and approximate matches separately.
- **'readSeq':** this function parses an integer value into an array seq of digits. It reads the digits from right to left and stores them in the array.
- **'readNum':** This function reads a number from stdin (standard input) and returns it. It prompts the user to enter a number between 0 and max and reads the input from the console.
- **'timeInMicroseconds , timer_handler, initITimer':** All these functions are integral to managing timing and signals in the game. timeInMicroseconds() retrieves the current time in microseconds, timer_handler() is a signal handler for when the timer expires, and initITimer() sets up and starts a virtual timer with a specified timeout value

There are many functions implemented that involve controlling a character LCD display and managing timing for game events. The failure() handles error reporting and program termination, waitForEnter() pauses the game until the user presses Enter, delay() and delayMicroseconds() provide timing delays, strobe() manages the strobe signal for the LCD, sendDataCmd() sends data to the LCD, lcdPutCommand() sends commands to the LCD, lcdHome() resets the LCD cursor position, lcdClear() clears the LCD screen, lcdPosition() sets the cursor position on the LCD, lcdDisplay() controls the LCD display state, lcdCursorBlink() controls the blinking of the cursor on the LCD, lcdPutchar() displays a character on the LCD, lcdPuts() displays a string on the LCD, and blinkN() blinks an LED a specified number of times. These functions together manage the display and timing aspects of the game.

The main function implements a MasterMind game using an LCD display, LEDs, and a button. It initializes the game settings and the hardware GPIO pins, then enters a loop where the player guesses a sequence of numbers using the button. The game provides feedback on the LCD display and LEDs to indicate the correctness of the guess. The loop continues until the player either guesses the correct sequence or reaches a maximum number of attempts. If the correct sequence is guessed, the game displays a success message and the number of attempts. Otherwise, it displays a failure message.

Performance-Relevant Design Decisions:

1. The program minimizes GPIO operations by configuring GPIO pins only once at the beginning of the program and reusing them throughout the game. This reduces the overhead of setting up and resetting GPIO pins for each operation, improving overall performance.

2. an increased delay between LCD blinking to maintain the pace of the game. This can be done to give players enough time to observe the blinking before the next action or update occurs.

Functions Accessing Hardware:

1. blinkN()
2. lcdPuts()
3. lcdPutChar()
4. delay()
5. waitForButton()
6. readButton()
7. pinMode()
8. digitalWrite()
9. writeLED()

All these functions are used to directly access the hardware. The readButton(), digitalWrite(), pinMode() and writeLED() functions were implemented in C language first and then they were translated to assembly language. Along with that the countMatches() function was first implemented in C in master-mind.c and then in assembly language in mm-matches.s. Matches function was then called in master-mind.c and countMatches() was commented.

ARM Assembler Implementation:

Function Name: matches

Inputs:

r0: Pointer to the first sequence

r1: Pointer to the second sequence

Output:

Returns two integers encoded within one number. The first integer represents the number of exact matches. The second integer represents the number of approximate matches.

Description:

The function compares each element of seq1 with the elements of seq2 to find exact matches. If an exact match is found, it increments the exact match counter. After checking for exact matches, the function enters a loop to find approximate matches. For each element in seq1, it checks if that element appears anywhere in seq2 but not at the same position. If found, it increments the approximate match counter. The function then returns the total count of exact and approximate matches encoded within a single number. The function uses loop structures to iterate over the sequences and compare their elements, efficiently counting the exact and approximate matches.

Example Usage:

For the provided example sequences:

secret: 1 2 1

guess: 3 1 3

The expected output is 0 for exact matches and 1 for approximate matches, as the sequence '1' in secret matches the '1' in guess, but not in the correct position.

Output:

./cw2 -u 121 313

0 exact

1 approximate

Example Set of Output:

Consider the following output example:

The LCD display prompts the user to enter their guesses one by one for a secret sequence, which is known to be "1 2 1" in this case. Each input is indicated by the user pressing a button. Upon entering a guess, the system provides visual feedback using LEDs as follows:

Red LED Blinking: After each button press, the red LED blinks once as a separator.

Green LED Blinking: The green LED blinks a number of times corresponding to the button presses for that guess.

End of Round Indicator: After entering all three guesses, the red LED blinks twice to indicate the end of the round.

Feedback for Matches:

After each round, the green LED blinks to indicate the number of exact matches. The red LED then blinks as a separator and then green LED blinks again to indicate the number of approximate matches. If the correct sequence is entered, the red LED remains on while the green LED blinks thrice and a success message is displayed on the LCD. The LCD displays the number of attempts made by the user.

The system continues this process until the correct sequence is entered. However, for testing purposes, the number of attempts can be limited to a predefined number.

Summary:

Achievements:

We Implemented a matching function in ARM Assembly to compare a secret sequence with a user's guess, counting exact and approximate matches. Along with that, we integrated the matching function with C code for controlling LEDs and receiving user inputs via buttons. We used inline assembly in C functions to control GPIO pins for LED output and button input.

Additional Features:

- We added difficulty levels with varying sequence lengths options to make the game more engaging.

Learnings:

- Understanding of ARM Assembly language and its integration with C code for embedded systems.
- Experience in low-level hardware manipulation, such as controlling GPIO pins for input and output.

Testing the Logic:

https://1drv.ms/u/s!AokH3_4gcrx50SG2FD1FQrFzMkuK?e=ximHol

This is the link to the code.log file which contains the logic of the game in debug mode testing with the sequence 121.

Contributions:

1. Asma Sathar: lcdBinary.c & main function
2. Laiba Shehzad: mm-matches.s, timer functions & report