F28HS 2022-2023 CourseWork 2:

MasterMind

Mufadddal Abizar Ezzi (H00360993)

Muhammad Imaad Muhammad Ismail (H00362645)

Rohan Gautam Advani (H00376821)

Group 26

Link for Demo Video on OneDrive: F28HSI CW2 Group 26.mp4

Introduction

The purpose of this coursework is to create a straightforward systems-level program that runs on a Raspberry Pi with connected devices in C and ARM Assembler.

Two players compete in the game MasterMind as a codebreaker and a codekeeper. For any number of pegs, a sequence length of N and a set number of C colours are established before the game. The codekeeper then chooses N coloured pegs and inserts them into N slots in order. The code that must be cracked is included in this (hidden) sequence. The codebreaker takes it in turns to try to decipher the concealed sequence by creating a string of N coloured pegs using C colours.

The codekeeper responds to each turn by indicating how many pegs in the guess sequence are both the correct colour and in the proper position, as well as how many are the correct colour but not in the proper position. This knowledge helps the codebreaker improve their estimate in the subsequent round. When the code is correctly decoded or after a predetermined number of turns, the game is done.

Here is a what a sample output will look like:

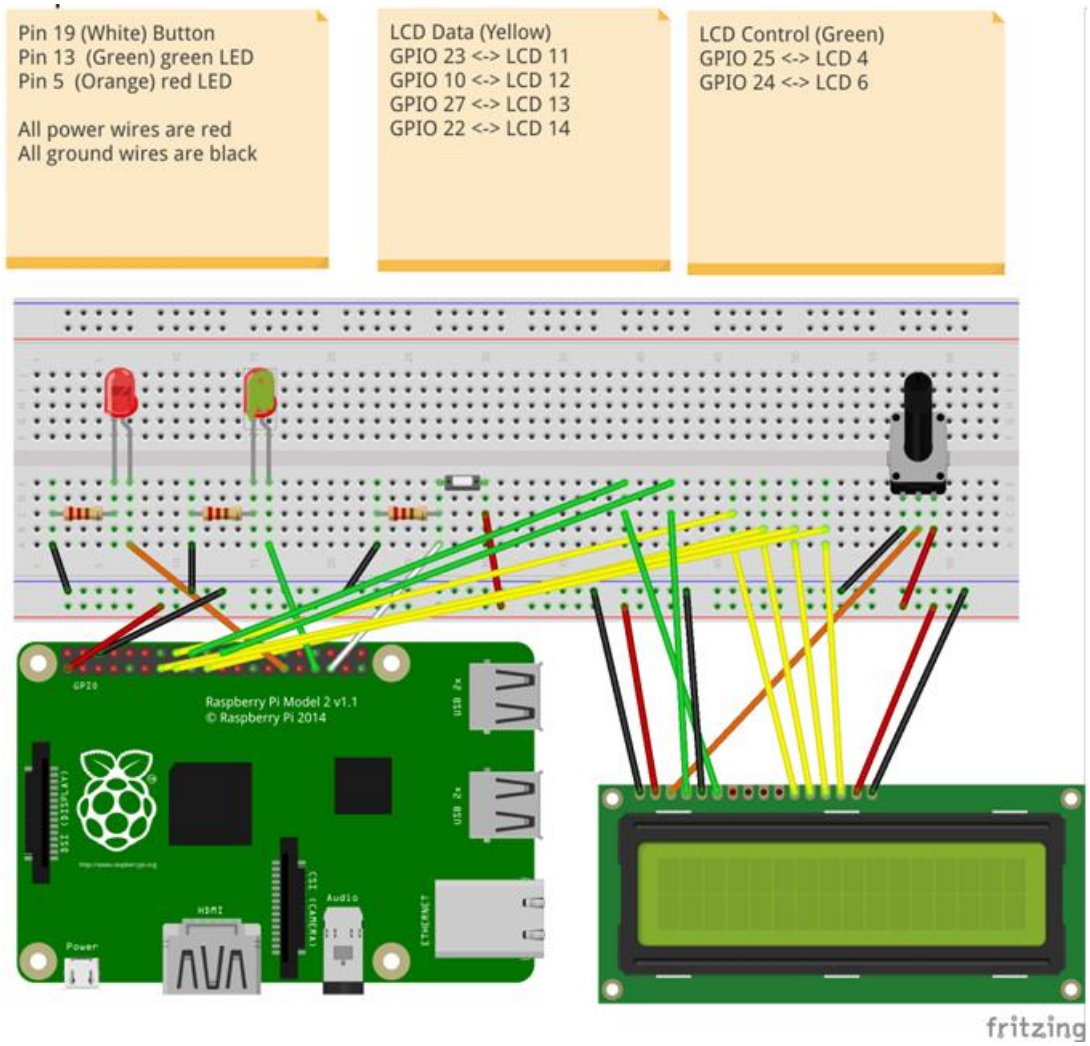
```
Secret:  R  G  G
===============
Guess1:  B  R  G
Answ1:           1 1
Guess2:  R  B  G
Answ2:           2 0
Guess3:  R  G  G
Answ3:           3 0
Game finished in 3 moves.
```

Hardware Specification

The hardware platform used is a Raspberry Pi 3 with the starter kit which consists of: -

- LCD
- LEDs
- Potentiometer
- Button

In terms of wiring, the following wiring and connections were implemented from the coursework specifications:



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.

The following GPIO Connections are for LCD Power:

1. LCD 2 – 5V Pin
2. LCD 15 – 5V Pin

The following GPIO Connections are for LCD Control:

1. LCD 4 – GPIO 25
2. LCD 6 – GPIO 24

The following GPIO Connections are for LCD Data:

1. LCD 11 – GPIO 23
2. LCD 12 – GPIO 10
3. LCD 13 – GPIO 27

4. LCD 14 – GPIO 22

LEDs

GPIO Pins are used to connect the two LEDs to the Raspberry Pi so that information and data output may be controlled. To regulate the current flow, a resistor is used in each LED. The current state of the pin is directly set by Inline Assembly code for turning on or off an LED. Both red and green LEDs are directly powered through the GPIO pins 5 (for Red LED) and 13 (for Green LED).

Potentiometer

The LCD Display, which is connected to LCD 3, has a potentiometer attached to it to regulate and manipulate contrast and visibility.

Button

User input is collected and transferred to the RPI using a button that is connected using the GPIO pin 19. It utilizes a 5V power supply pin as well as resistors to control the level of current that flows into it.

Code Structure

The code is broken up into various functions that each carry out a certain purpose. Each function makes use of either C or Assembly or both. Majority of the implementation is done using C.

The main functions are as follows:

initSeq: creates a random integer array with length seqlen named theSequence with random values between and not excluding 1 and 3.

showSeq: function to output the randomly generated secret sequence that the application initiated. This is executed in in debug mode.

countMatches: counts how many entries in seq2 match entries in seq1 and returns exact and approximate matches, either both encoded in one value, or as a pointer to a pair of values.

showMatches: This function calls the countMatches function and then prints the exact and approximate matches in the terminal.

readSeq: parse an integer value as a list of digits, and put them into @seq@, needed for processing command-line with options -s or -u.

blinkN: blinks a given LED specific number of times. Makes the LED blink with a certain specified delay.

main: When the code is executed, this is the primary program that starts. It contains the whole blueprint for the sophisticated mind game. Additionally, it contains the code for the various modes and flags that the user may select to execute the application in.

Design Choices:

Pointers:

Throughout the program, a pointer array is utilized to hold the precise and approximative matches. Each match is stored in a reference that is mallocated at runtime with a size of 2. Since indexing may be utilized in this fashion, no additional ceremonies are needed to access each of the values independently.

Use of Timer:

The user's feedback is requested using a 5-second timer. Only when the user hits the button for the first time while entering one number does the timer start. In this manner, the user's subsequent inputs may be read, increasing accuracy and efficiency.

Assembly Implementation:

To access the hardware, including the button and the led, the lcdBinary file uses inline assembly. Since the registers are directly controlled by the code when assembly is used, accuracy is increased while program speed is also increased.

Functions Accessing Hardware

The following operations have direct access to hardware:

pinMode: Depending on the argument supplied, this method either changes the GPIO pin to input or output mode. Mainly uses C.

writeLED: The purpose of this code is to control the on/off state of a specific LED. The function writes a particular mode to either the set or clear register, depending on the value of the variable, using C. By using inline assembly, the code modifies the set/clear register to switch the LED on or off.

readButton: A code snippet is used to detect when a user clicks a button. The programming language C is utilized to identify the specific input/output pin that the button is connected to, by accessing the gplev0 register. The inline assembly code then reads the value from the identified register and outputs it.

waitForButton: Function that uses the readButton method and waits for the user to press the button.

Sample Execution of the Program in Debug Mode

```
heisenberg@raspberrypi:~/f28hs-2022-23-cwk2-sys $ sudo ./master-mind -d
The sequence is: B B G
Raspberry Pi LCD driver, for a 16x2 display (4-bit wiring)
==============================

           Debug Mode
==============================

Only uses command-line interface
for inputting the guess sequence

 Primarily to check Game Logic

Attempts #1 RBB
1 exact
1 approximate

Attempts #2 BBR
2 exact
0 approximate

Attempts #3 GGB
0 exact
2 approximate

Attempts #4 BBG
3 exact
0 approximate

Good job bro! You figured out the code in 4 attempts!
```

Summary

We were able to create a straightforward systems-level program that runs on a Raspberry Pi with connected devices in C and ARM Assembler where a human player competes in the game MasterMind as a codebreaker against an automatically generated codekeeper.

The effective development of the Mastermind application was made possible by all the program's features.

Unfortunately, we were not able to correctly implement ARM in place of C in a few areas. Everything else has been done to the best of our abilities and our program complies with all necessary standards.

In conclusion, the code was successful in controlling the operation of an LED, button, and LCD display. C and ARM Assembler were used to implement the functions that have direct access to the hardware. Overall, working on this project gave us the chance to learn more about ARM-based microcontrollers and how to program them in C and ARM Assembler.