Hardware-Software Interface

F28HS

Mastermind RPi Project – Coursework 2

3rd April 2024

David Farid - dgf2001@hw.ac.uk

Aditya Shibu - as2397@hw.ac.uk

Dubai Campus

Problem Specification:

This project aims to recreate the Mastermind logic game using a Raspberry Pi with an LCD interface in conjunction with a button for user input and an LED to display the appropriate signals indicating the current state of the game.

The program should generate a secret sequence, check for what the user inputs, use the LED to display the correctness of the user's guess (via EXACT and APPROXIMATE matches), and loop until the guess matches the sequence.

Hardware Specification:

- Raspberry Pi 3 Model B
- 16x2 I2C LCD Display Blue Backlit
- Red LED x1, Green LED x1
- 3 level Potentiometer
- Mechanical Button
- 330Ω resistor x2, 10 kΩ resistor x1
- 16 Male-Female, 4 Male-Male, 6 Female-Female wire

Code Structure:

The code is structured by first defining all prerequisite functions and variables, followed by writing the necessary main game functions which will be described below shortly.

- **initSeq()** – Initiates a secret sequence of 3 digits using a loop and the random function from stdlib.h.
- **showSeq()** – Takes in an argument which is a pointer to a sequence and prints it out into the terminal using a loop to iterate over the elements.
- **countMatches(int* seq1, int* seq2)** – Takes in two arguments namely seq1 and seq2 and compares seq2 with seq1 using two main loops, one to check for exact matches and one to check for approximate matches and returns it as an encoded code which is (exact * 10) + approx.
- **showMatches(code)** – Code is the value you get when two sequences are passed to countMatches(), showMatches decodes the code and prints out the exact and approximate matches.
- **readSeq(int* seq, int val)** – Used to read the value such as 1 2 3 as a list and then put them into the seq, this is done using simple math to get the first, second and third digits that the user inputs.
- **readNum(int max)** – Used to parse a guess sequence and then store it in an array called inputs using a loop to iterate over input digits and push them into the array.

Performance Optimizations:

Dynamically Allocated Memory:

```
theSeq = (int *)malloc(SEQL * sizeof(int));
seq = (int *)malloc(SEQL * sizeof(int));
seq1 = (int *)malloc(seqlen * sizeof(int));
seq2 = (int *)malloc(seqlen * sizeof(int));
attSeq = (int *)malloc(seqlen * sizeof(int));
cpy1 = (int *)malloc(seqlen * sizeof(int));
cpy2 = (int *)malloc(seqlen * sizeof(int));
lcd = (struct lcdDataStruct *)malloc(sizeof(struct lcdDataStruct));
```

Freeing Allocated Memory:

```
  free(theSeq);
  free(seq1);
  free(seq2);
  free(cpy1);
  free(cpy2);
  free(attSeq);
  free(lcd);
```

Hardware Functions:

- **digitalWrite(uint32_t *gpio, int pin, int value) –** digitalWrite takes 3 arguments, the mapped gpio, the pin to send the value to and the value i.e., HIGH, or LOW. This calculates the pin value and then sends the value to the given value.
    - **Calculating offset, mask and gpio register**: C
    - **Sending value to pin, setting, and clearing pin**: Inline Assembly
- **pinMode(uint32_t *gpio, int pin, int mode) –** pinMode uses the mapped gpio, pin and mode i.e., INPUT, or OUTPUT to configure the given pin as the given mode.
    - **Calculating offset, mask and reg**: C
    - **Setting pin to specified mode**: Inline Assembly
- **writeLED(uint32_t *gpio, int led, int value) –** writeLED uses digitalWrite and pinMode to set the given led pin to be configured as OUTPUT (1) and send the specified value i.e., HIGH (on) or LOW (off) to the given led pin.
    - **Uses existing implementations of pinMode and digitalWrite**
- **readButton(uint32_t *gpio, int button) –** readButton uses the mapped gpio and reads when the button has been pressed (i.e., set the value to HIGH), returns 1 when the button is pressed.
    - **Calculating offset and register**: C
    - **Setting pin for button to output is done by using pinMode**
    - **Reading pin state**: Inline Assembly
- **waitForButton(uint32_t *gpio, int button) –** waitForButton uses readButton and runs an infinite loop to keep checking if the button has been pressed.

Matching function in ARM (mm-matches.s):

Inputs:

- **Secret Sequence:** Pointer to the memory location containing the secret sequence.
- **Guess Sequence:** Pointer to the memory location containing the guessed sequence.

Outputs:

- **Exact Matches:** Calculates the digits that are the right number and in the right position in the sequence.
- **Approximate Matches:** Calculates the digits that are the right number but in the wrong position in the sequence.

Sub-Routine Breakdown:

- **Matches:** This is the countMatches function defined in ARM assembly and callable from mm-matches.s.
- **Inputs:** Two pointers to sequences (secret and guess).
- **Outputs:** Returns two numbers encoded within one register (R0) representing the exact matches and the approximate matches.

Example Output:

Running master-mind with secret sequence given in as: 3 1 3

(Ignoring all the LCD Put commands such as lcdPutCommand: digitalWrite(25,0) and sendDataCmd(25835032,2)) etc.

```
pi@coolraspberrypi:~/Desktop/masterMind_Final $ please ./master-mind -s 313

Raspberry Pi LCD driver, for a 16x2 display (4-bit wiring)

Printing welcome message on the LCD display ...

Press ENTER to continue:

Round: 1

Turn: 1

Enter a sequence of 3 numbers

Button pressed

Button pressed

Button pressed
Button pressed 3 times

Turn: 2

Enter a sequence of 3 numbers

Button pressed

Button pressed 1 times

Turn: 3

Enter a sequence of 3 numbers

Button pressed

Button pressed

Button pressed

Button pressed 3 times

3 exact

0 approximate

SUCCESS
```

Summary:

What was achieved:

A successfully operational embedded system with full Mastermind functionality and efficient usage of resources via low-level programming.

Outstanding Features:

Efficient hardware-software connectivity, optimized wiring, user ease of access via LCD feedback, simple program usage via the button.

What was learned:

How to manage and operate an embedded system on a large scale with low level code, how to use peripherals such as buttons, LEDs, and LCDs, and how to effectively manage electrical circuitry using potentiometers and resistors, and the essence of teamwork.

Contributions:

David Farid: Worked on game logic and part of main flow, half the report and participated in the demo

Aditya Shibu: Worked on assembly logic/aux functions and part of main flow, half the report and participated in the demo