3/29/2017

Coursework 2 Report

F28HS: Hardware-Software Interface

Mark Gordon & Calum McLean
(H00228757) (H00211376)

Problem Specification

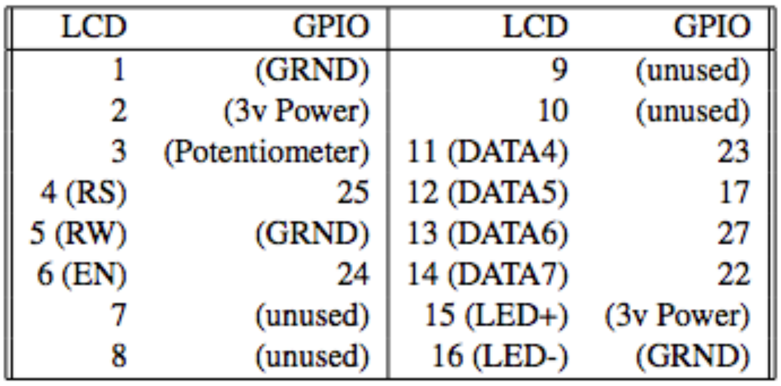
As stated by the coursework assignment:

> The aim of this coursework is to develop a simple, systems-level application of the MasterMind board game in C and ARM assembler, running on a Raspberry Pi2 with attached devices. MasterMind is a two player game between a codemaker and a codebreaker. Before the game, a sequence length of N and a number of C colours for an arbitrary number of pegs are fixed. Then the codemaker selects N pegs and places them into a sequence of N slots. This (hidden) sequence comprises the code that should be broken. In turns, the codebreaker tries to guess the hidden sequence, by composing a sequence of N coloured pegs, choosing from the C colours. In each turn the codemaker answers by stating how many pegs in the guess sequence are both of the right colour and at the right position, and how many pegs are of the right colour but not in the right position. The codebreaker uses this information in order to refine his guess in the next round. The game is over when the codebreaker successfully guesses the code, or if a fixed number of turns has been played.

Hardware Specification and Wiring

As stated by the coursework assignment:

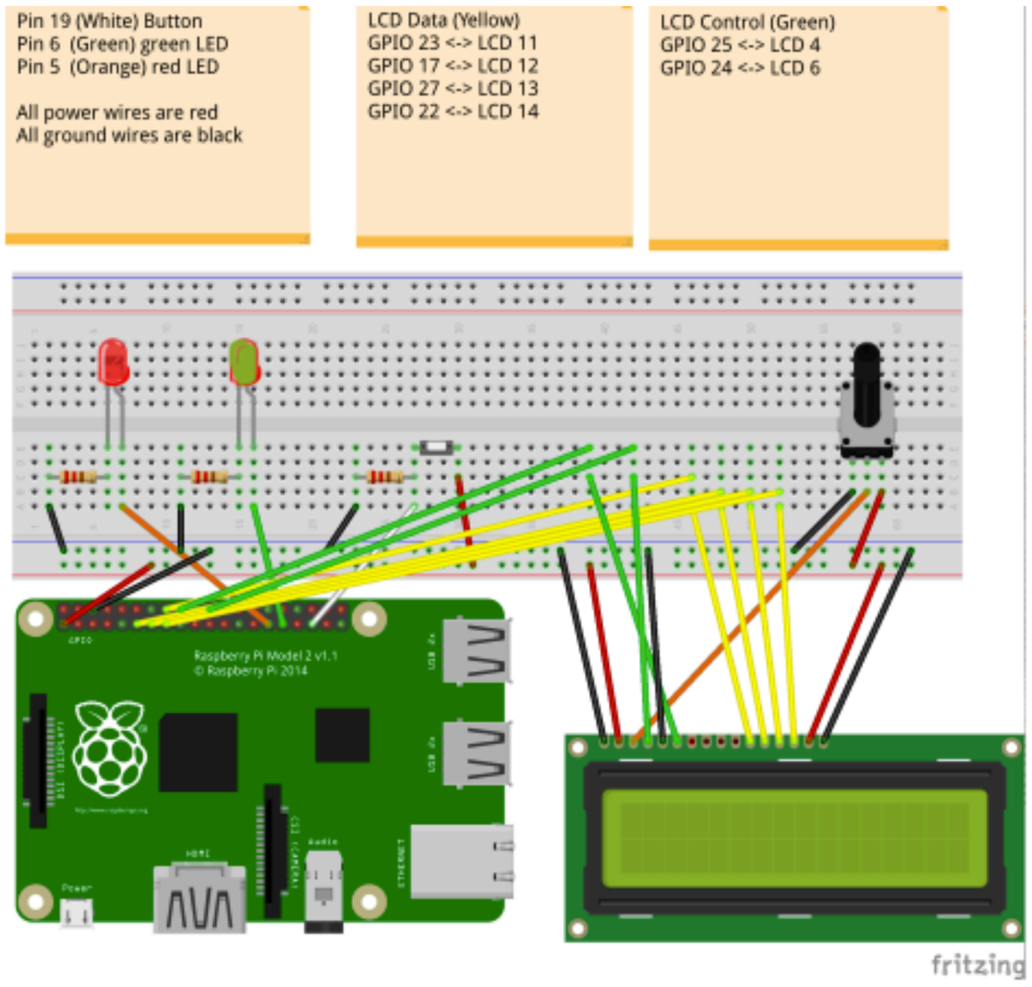
> Two LEDs should be used as output devices: one (green) LED for data, and another (red) LED for control information (e.g. to separate parts of the input and to start a new round). The green data LED (right) should be connected to the RPi2 using GPIO pin 6. The red control LED (left) should be connected to the RPi2 using GPIO pin 5. A button should be used as input device. It should be connected to GPIO pin 19. An LCD should be used as an additional output device. It should be connected as follows:



LCD	GPIO	LCD	GPIO
1	(GRND)	9	(unused)
2	(3v Power)	10	(unused)
3	(Potentiometer)	11 (DATA4)	23
4 (RS)	25	12 (DATA5)	17
5 (RW)	(GRND)	13 (DATA6)	27
6 (EN)	24	14 (DATA7)	22
7	(unused)	15 (LED+)	(3v Power)
8	(unused)	16 (LED-)	(GRND)

> This means that the 4 data connections to the LCD display are connected to these 4 GPIO pins on the RPi2: 23, 17, 27, 22. All devices should be connected to the RPi2 using a breadboard. This is a Fritzing diagram that visualises the entire wiring. Note

that the middle pin of the potentiometer needs to be wired to LCD 3, and the other 2 legs to ground and power as shown in this version. Also, use the 3.3V and not the 5V power GPIO pin from the RPi2 to be on the safe side



The Functions

void mmapfunc()

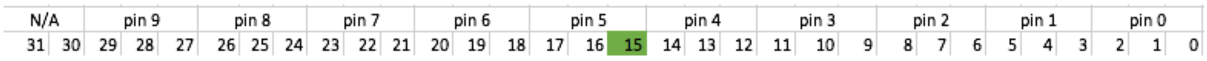
The purpose of this function is to memory map the devices memory

void toggleLED(int pin, int mode)

The purpose of this function is to toggle an LED connected to pin on or off (decided by the mode)

- We start by loading the gpio address into a register (R1). We then need to control pins 5 and 6 for the LED. So we can just add 0 to the base address for register GPFSEL0 which controls pins 0-9. We then load the bit string of the address in R1 into register R2
- Move #1 into R3 and perform a left shift on it by a shift variable given by (pin *3). This is because each pin takes up 3 bits in the GPFSEL0 register, so pin 5 is located in bits 15, 16 and 17 (pin 5 * 3 = 15)
- We now perform an ORR operation on R2 and R3, storing the result in R2 so we now have the original address bit string, but with the shifted 1 value put into place. This result is then stored into R2 to set up the required pin

Example of writing 1 into pin 5 of GPFSEL0 (Green represents 1, rest are 0)



N/A		pin 9			pin 8			pin 7			pin 6			pin 5			pin 4			pin 3			pin 2			pin 1			pin 0		
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- Now we have to toggle the LED on or off so we move the gpio address back into R1 and then we add the #28 (gpset0) or #40 (gpclr0) to turn the LED on or off. This is decided by a c variable passed into the function 'mode'.
- Again we move #1 into R3 but this time we do a LSL just by the pin value that was passed into the function Storing in R3. An ORR is then performed on R2 and R3 with the result then stored to toggle the LED

  unlike GPFSEL0, GPSET0 is one bit per pin, so pin 5 is bit 5

void checkButton():

The purpose of this method is to read the level of the pin connected to the button to see if the button is being pressed or not.

- We begin by moving the gpio address into R1 and then adding #52 in order to get the GPLEV0 address. This is responsible for reading the pin level low or high. We load R2 with the bit string of R1 and move R1 into a C variable buttonCurrentValue
- Then we perform an ORR operation on R2 and #1<<19 (left shift 19 places). This moves 1 into the 19th place of R2 which is where the button is located (pin 19). This is then read into a C variable buttonOnValue as this is the bit address in GPLEV0 when the button is on.
- An equality check is then done on both the C variables buttonOnValue and buttonCurrentValue and returned. 0 for off, 1 for on.

void setUpTimer ():

This method simply sets up a itimer, upon SIGALRM, getButtonInput() is called. Timer is set via an interval (in milliseconds) which is passed into the method.

void delay (unsigned int howLong):

Simple method to make program sleep. howLong variable is passed into the program and represents the milliseconds the program should sleep for.

Void flashLED(int pin, int noOfFlashes):

Simple method that will flash the pin passed in for noOfFlashes. This is done with use of a simple for loop and toggling the LED on and off with a delay in between.

int checkCorrect(int codecopy[]):

This method checks how many of the user's inputs are correct and in the right position returning the result

- Simple for loop checking the users guess against the exact position in the code and if so increments the correct variable, replacing the code and users guess with negative values so they don't get counted again when checking the approximate matches
- Result is returned

int checkApproximate(int codecopy[]):

This method checks how many of the user's inputs are correct but NOT in the right position and returns the result

- This is done with use of a simple loop and 3 if statements checking each 3 positions of the code against the current indexed guess. If any of them match, then approximate variable is incremented and the code and guess replaced with negative values so they don't match the next loop through.
- Result is returned

void getUserInput():

this method handles getting the user input for guessing the code combination.

- This is done with a for loop, looping 3 times to get 3 inputs.
- A message is outputted to the user informing them to enter the input of the current index and then a timer is set for 0.5 seconds by use of the setUpTimer method and then pause() is called, calling the getButtonInput() method which gets the input from the user. If the user enters a valid input (between 1 and 3) then LED 5 is flashed once to signify the input being accepted, and then LED 6 flashes for the number of times the button was pressed to signal what the user inputted. If the user enters an invalid input, then the loop will re-loop with an invalid input message displayed to the user.

void correctCombination():

Purpose of this method is to inform user they have worked out the correct combination by displaying a message and turning on LED 5 while flashing LED 6 3 times.

void userFeedback(int correct, int approximate):

This method gives feedback to the user for many many correct and approximate inputs they have

- Displays message to user informing of how many in the correct position before flashing LED 6 for the number of times they have correct
- LED 5 is flashed as a separator
- Then a message to user informing of how many approximate matches before flashing LED 6 for the number of times.

void endOfRound():

This method handles the end of the round by displaying a message to the user, flashing LED 5 twice, and informing them of their inputs.

void getButtonInput(void):

Purpose of this method is to allow the user to enter input via the button to get their guess code

- We start by stopping the timer by calling setUpTimer and passing 0, and setting the buttonPresses of the index buttonTurn to 0.
- We then loop 100 times with a 30ms delay (3 seconds total) in which we call checkButton() to see if the button is being pressed or not and add the result to the current index of buttonPresses. To avoid the issue of a button press being counted for multiple times, there is a while loop that only exits when checkButton() != 1
- After exiting the loop, the users input is displayed and the buttonTurn is incremented for the next calling of the method

int main(int argc, const char* argv[])

- Calls mmapfunc()
- Generates a random code of length 3 by using the numbers 1-3
- Enters an infinite while loop that can only be broken when the users guesses the correct combination for the code
- Displays message to user that the round as started and delays for 2 seconds
- Calls getUserInput() method
- Calls endOfRound() method

- Creates a copy of the randomly generated code
- Calls checkCorrect(codecopy) method passing the copied code in and stores the result into variable ‘correct’ and then proceeds to see if check equals 3. If so the correctCombination() method is called the infinite loop is broken out of.
- Else a message is displayed to user that the combination was incorrect and the checkApproximate(codecopy) is called passing in the copied code. The result is stored into a variable ‘approximate’.
- The userFeedBack variable is then called passing in the variables ‘correct’ and ‘approximate’
- Finally, preparation is done for the next round by resetting the variables and flashing LED 5 3 times to inform the user of the start of the next round.

List of Functions Directly Accessing Hardware

- void toggleLED(int pin, int mode) – All interaction is in assembly, C is used to hold certain variables such as pin, mode, shift and gpio
- int checkButton() – All direct interaction is in assembly, C is used to hold variables and check the equality between buttonOnValue and buttonCurrentValue, with the result returned

Sample Execution

(RANDOM CODE GENERATED: 1   2   1)

output:

```
==ROUND START== (command line)

Enter input 1…  (command line)
Button pressed twice (action)
Input: 2 (command line)
LED 5 FLASHES
LED 6 FLASHES TWICE

Enter input 2… (command line)
Button pressed once (action)
Input: 1 (command line)
LED 5 FLASHES
LED 6 FLASHES ONCE

Enter input 3… (command line)
Button pressed 3 times (action)
Input: 3 (command line)
LED 5 FLASHES
LED 6 FLASHES THREE TIMES

==END OF ROUND== (command line)
Inputs: 2, 1, 3 (command line)
```

LED 5 FLASHES TWICE

0 inputs had the correct input and position (command line)
LED 5 FLASHES
2 inputs had the correct input but the wrong position (command line)
LED 6 FLASHES TWICE
LED 5 FLASHES THREE TIMES

==ROUND START== (command line)

Enter input 1… (command line)
Button pressed once (action)
Input: 1 (command line)
LED 5 FLASHES
LED 6 FLASHES ONCE

Enter input 2… (command line)
Button pressed twice (action)
Input: 2 (command line)
LED 5 FLASHES
LED 6 FLASHES TWICE

Enter input 3… (command line)
Button pressed 1 times (action)
Input: 1 (command line)
LED 5 FLASHES
LED 6 FLASHES ONCE

==END OF ROUND== (command line)
Inputs: 1, 2, 1 (command line)
LED 5 FLASHES TWICE

Well done! You have entered the correct combination! (command line)
LED 5 TOGGLED ON
LED 6 FLASHES THREE TIMES
LED 5 TURNED OFF

Summary

Overall in this coursework, we managed to implement most of the key features (12/14 functional requirements). However, we had difficulties finishing the LCD time in time. We believe we have a reasonable understanding of how the LCD works theoretically, just when we came to code it we couldn't quite manage. From this coursework we gained an understanding of how code interacts with the hardware and the interaction between embedded hardware and external devices. This will be very useful knowledge going forward as we believe its incredibly useful to know how the code you are writing is interacting with the underlying system.