|  |
| --- |
| Heriot Watt University |
| Mastermind Application |
| F28HS – Coursework 2 |

|  |
| --- |
| Callum Hayden & Paul Cockburn  cmh1@hw.ac.uk & psc4@hw.ac.uk |

# Problem specification:

Mastermind is a 70s board game which consists of a player creating a code and a player trying to break the code. This project consists of building mastermind on the Raspberry Pi, where a sequence of colourful pegs would instead be a sequence of numbers. The Raspberry Pi would act as the “code master”, making a random code at the start of each game.

The game progresses in rounds of the player or “code breaker” guessing the code. The code is guessed by pressing a button. After each input the raspberry pi confirms the users input through flashing LEDs.

At the end of each round, the code master communicates through LEDs the amount of exact or approximate matches. It must also communicate the start of a new round. The game continues in this routine of rounds until the code breaker correctly breaks the code.

This project must also include a debug mode which displays the secret code, the inputs and the outputs on the command line.

# Hardware specification and wiring used:

The hardware used includes a Raspberry Pi 2 model B, a red LED, a yellow LED (originally specified to use a green LED, but due to not having a green LED a yellow one was used instead), a button, three resistors, a bread board, five male/female connector cables and and four male/male connector cables. The five male/female connector cables are used to connect the Raspberry Pi with the breadboard. The red LED is connected to the GPIO pin 5 of the Raspberry Pi, the yellow LED is connected to the GPIO pin 6 of the Raspberry Pi and the button is connected to the GPIO pin 19 of the Raspberry Pi. The last two male/female connector cables are connected to the Raspberry Pi’s power pin and ground pin.

On the breadboard, the LEDs and button is connected to the corresponding cables coming from the Raspberry Pi. Each LED and the button also has a resistor connected to it through to breadboard.

See appendix for images.

# Code structure:

Here is a list of the functions we created to help us to make this project work, along with what they do. Functions not included here are in the next segment, as these are functions which use C and don’t interact directly with hardware.

* **getGPIO()** this function takes care of the memory mapping of GPIO
* **pinFlash()** this function is used by **redFlash()** and **yellowFlash()**. This takes care of the amount of times the LED flashes and makes them flash by calling the **digitalWrite()** function.
* **redFlash()** this function calls **pinFlash()** to make the red LED flash.
* **yellowFlash()** this function calls **pinFlash()** to make the yellow LED flash.
* **ledInputRecieved()** this function controls the confirmation of the users input, making the red LED flash once and the yellow LED flash the same amount of times as the users input.
* **ledShowResult()** this function displays the result after each guess, flashing the yellow LED for the amount of exact and approximate matches, with a red LED flash in-between, and three red LED flashes to show the end of the guesses results.
* **ledSuccess()** this function makes the red LED stay on while the yellow LED flashes three times to signify a correct guess by making use of the **digitalWrite()** function to turn the red LED on and off, and the **yellowFlash()** function to make the yellow LED flash.
* **getButtonInput()** this function calls the **readPin()** function to read the number of times the button was pressed. This also has a timer to allow sufficient time for the user to enter their input.
* **initialiseMastermindIO()** this function makes calls to **pinMode()** to set the function of the yellow and red LED pins, and **digitalWrite()** to ensure the LEDs are off.
* **checkGuess()** this function checks the guess input by the user against the secret code, and returns the number of exact and approximate matches.
* **getGuess()** this function returns the users guess as a list of integers.
* **showResult()** this function displays the number of exact and approximate guesses by calling the **ledShowResult()** function. It also checks if the program is running in debug mode and, if it is, prints the number of exact and approximate matches.
* **generateAnswer()** this function creates and returns a random sequence of numbers to use as the code the player is trying to break. This function also prints out this secret code, if the program is in debug mode.
* **main()** this function is the first function to be called when the program starts. It makes calls to every other function and controls the flow and order of the program.

# Functions that interact with the hardware in assembly:

Here is a list of functions that interact directly with hardware and use assembly.

* **pinMode()** this function sets the function of a BCM pin.
* **digitalWrite()** this function sets a BCM pin to high or low, allowing to turn on or off a piece of hardware. It is called multiple times throughout the program to turn an LED on or off.
* **readPin()** this function reads the state of a pin. This is used to check if the button has been pressed.

# Debug mode:

Sample execution of the program in debug mode.

# Summary:

This project managed to achieve almost everything requested. The downfall has been the LCD screen that we were unable to implement this part of the project. With more time we believe this would be achievable. Apart from not being able to complete the task perfectly, we feel that we have managed to learn a lot about C, assembly, and the raspberry pi in general. It has sparked our interest in the Raspberry Pi, and we have since searched online for prices of the new Raspberry Pi 3 model B and are considering buying it.

The hardest part was the assembly. If it wasn’t for being able to get help from fellow students, it would have been impossible. If we were able to borrow the raspberry pi for longer, then I’m sure we would love to work out how to use the LCD screen over summer.

# Appendix:

Figures used to help explain this project.

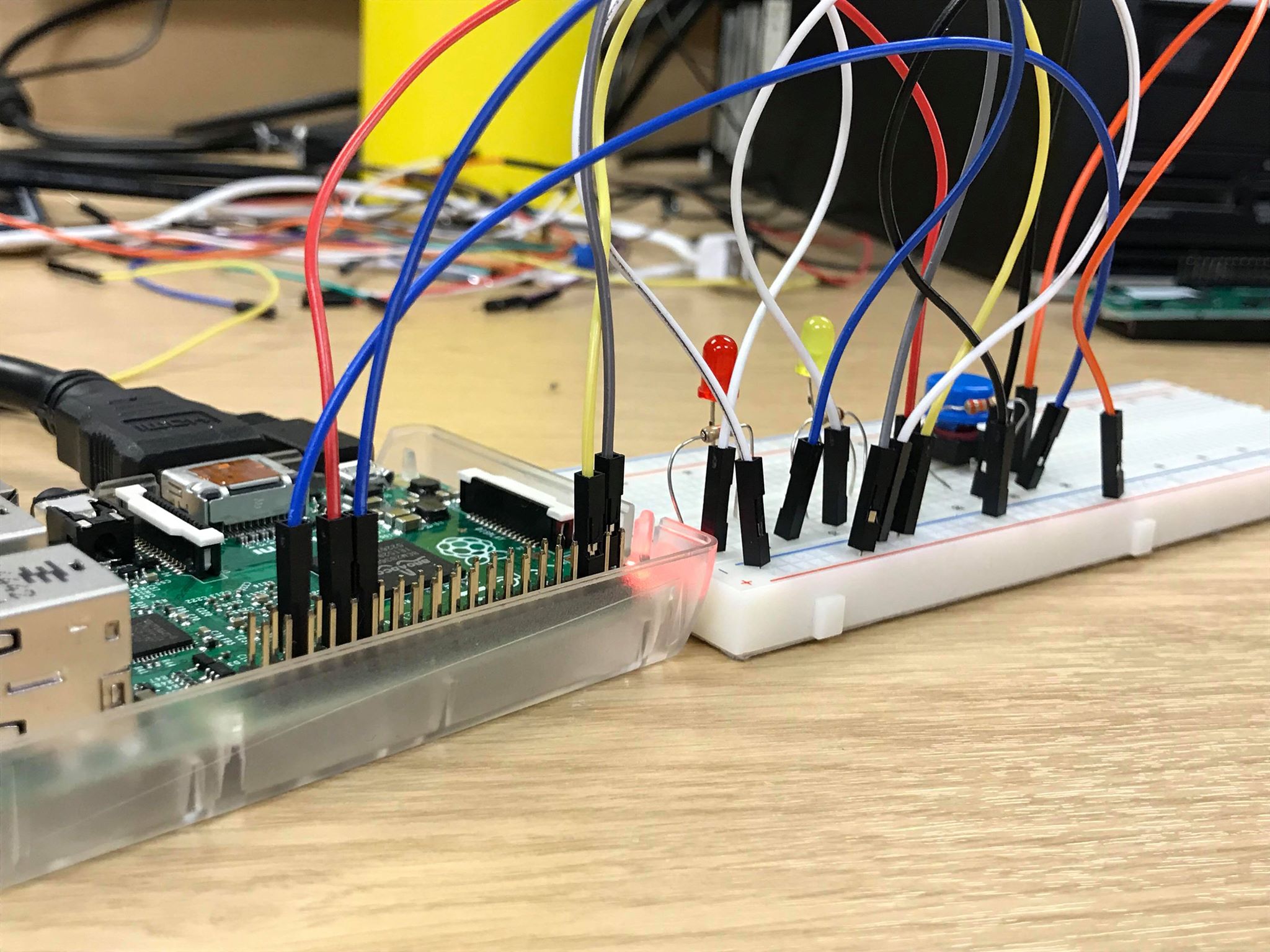


Figure 1: Side view of wiring

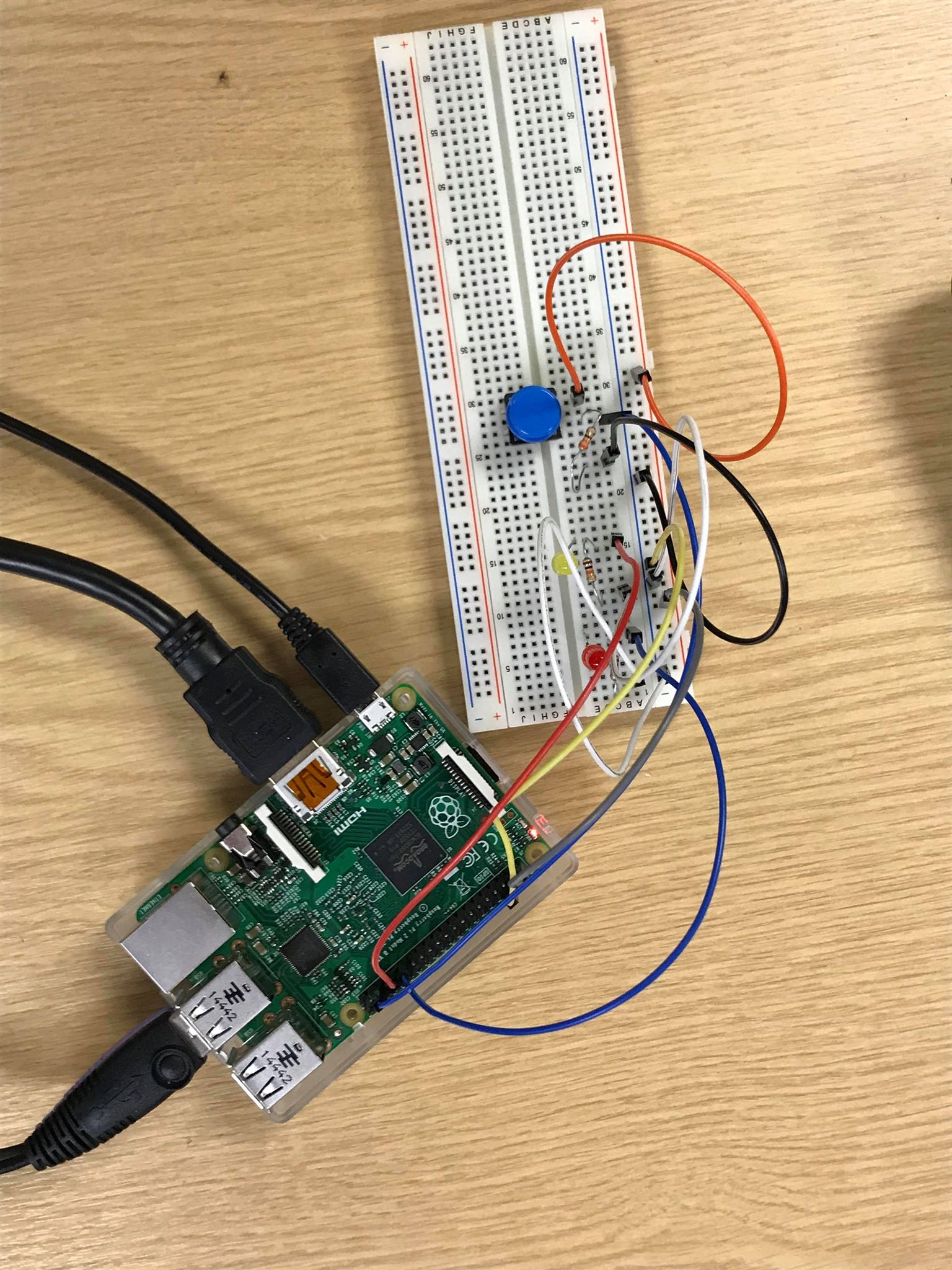


Figure 2: top down view of wiring