

MICRO PERIPHERALS AND INTERFACES

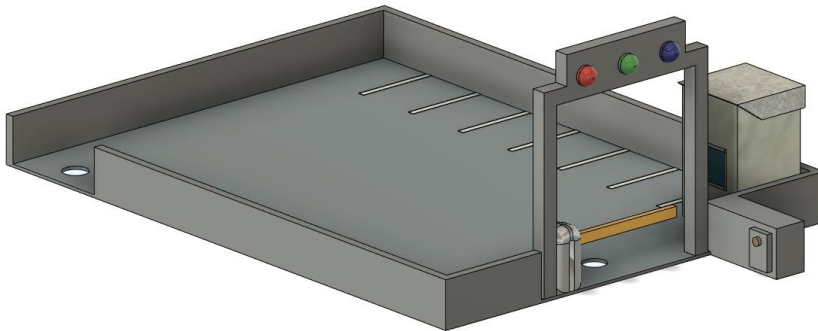
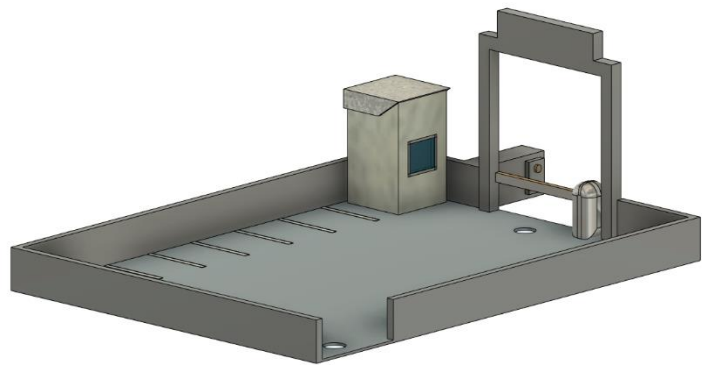


TABLE OF CONTENTS

Introduction.....	3
Task brief	4
Flowcharts	5
ENTRANCE	5
EXIT	7
operator interface	8
Digital prototype.....	11
B.O.M (bill of materials).....	12
Coding.....	13
3D prototype	18
Microcontroller build (Segments328 OrangePip).....	18
Blink Test (Arduino IDE)	20
Interface PCB	21
3D Models.....	26
Finised Prototype.....	28
Conclusion	32
Bibliography.....	33

INTRODUCTION

For Computing assignment 1, I was tasked with creating a microcontroller-based car park with a single entrance, controlled via light indicators and a barrier. The brief also states that the exit is free flow.

What is a Microcontroller? – “You can define a microcontroller as a simplified computer - one that is generally designed to run a single basic programme repeatedly. By definition, microcontrollers are typically intended to perform a single automated task, as pre-programmed by the user, in a single device. They are designed to do this one job repeatedly (or, as is also common, on a timed loop)”. (Subbaroybhat, 2021)

This assignment document will cover my chronological planning from a flowchart design for the code, all of the way through to the finalised physical prototype of a generic automated car park that includes an operator interface.

TASK BRIEF

The car park operation brief goes as follows:

1. An operator sets the maximum available number of parking spaces (in the range 10 to 255).
2. Cars arriving at the entrance are initially prevented from entry by a barrier.
A red light will also be illuminated
3. The driver of the car may request entry by pressing a push button.
4. If the car park is full then a full sign is illuminated and entry will be denied
5. If the car park is not full then the barrier will raise. When it has risen the red light will extinguish and a green lamp will become illuminated.
6. When the car has passed the barrier as detected by the car's presence at sensor 1 then the green lamp will extinguish, the red lamp will be illuminated and the barrier will lower. (note there is an assumption that vehicles are clear of the barrier at this time).
7. Cars may leave the car park at any time. In doing so they activate sensor 2.
(Billing is handled by a separate number plate reading system)
8. The operator has additional controls and indications namely :-
 - a. The operator can view how many cars are parked at any time
 - b. The operator can see the state of the barrier, red lamp, green lamp, Full lamp.
 - c. The operator can cause the barrier to raise or lower
 - d. The operator can cause the indicators (Red lamp, Green lamp, Full Lamp) to be ON or OFF.
 - e. The operator can prevent more cars from entering the car park regardless of how many cars are actually in the car park.

FLOWCHARTS

In order to write the operational code for the car park, I needed to have a clear idea of how I wanted it to run. To do this, I used an online flowchart maker.

I decided to split the car parks functionality into 3 sections:

- **Entrance** – how and when the barrier would raise, LED indicators, entry sensor and adding the number of cars and the surrounding variables.
- **Exit** – exit sensor, subtracting the car count and again, the surrounding variables.
- **Operator interface** – Complying with the brief requirements in a logical manner.

ENTRANCE

To begin with, I focused on the entrance section. This starts with the red “stop” light being illuminated, followed by an if statement with a variable for the car count.

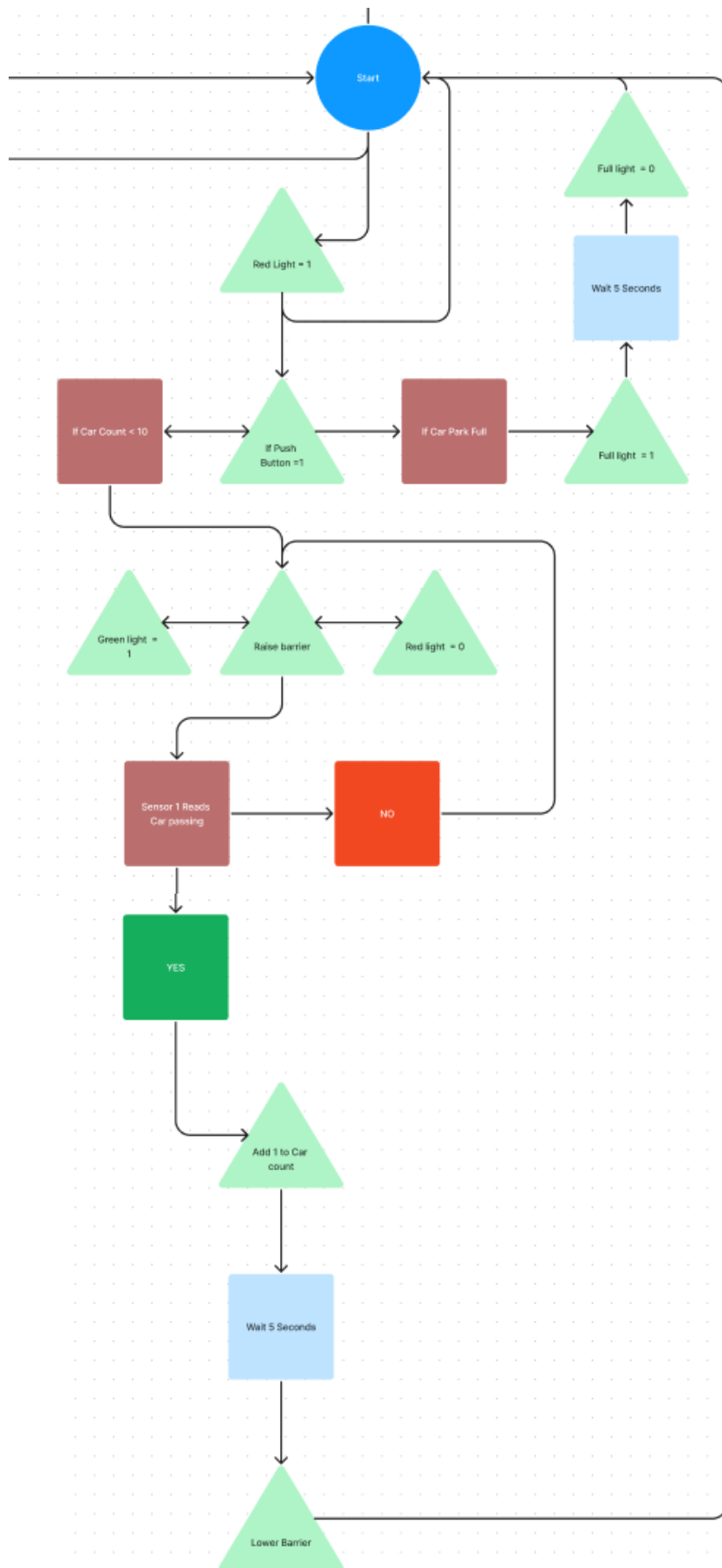
Both need to be true for the code to continue, we can see that if the carpark is full, the blue “full” light is illuminated, and the code returns to “start”.

When both statements are true, the code moves onto raising the barrier, whilst simultaneously turning off the red light and turning on the green “go” light.

In the next stage, the entrance sensor must detect the car passing through. This is important for the safety of the driver and to avoid damage to the vehicle due to the barrier closing on the car. We can see that if the sensor doesn't detect the car, the barrier will remain raised indefinitely.

Once the sensor detects the car, the code will add 1 to the total car count, wait 5 seconds for the car to be a safe distance from the barrier, then lower.

The code then returns to its starting point.



EXIT

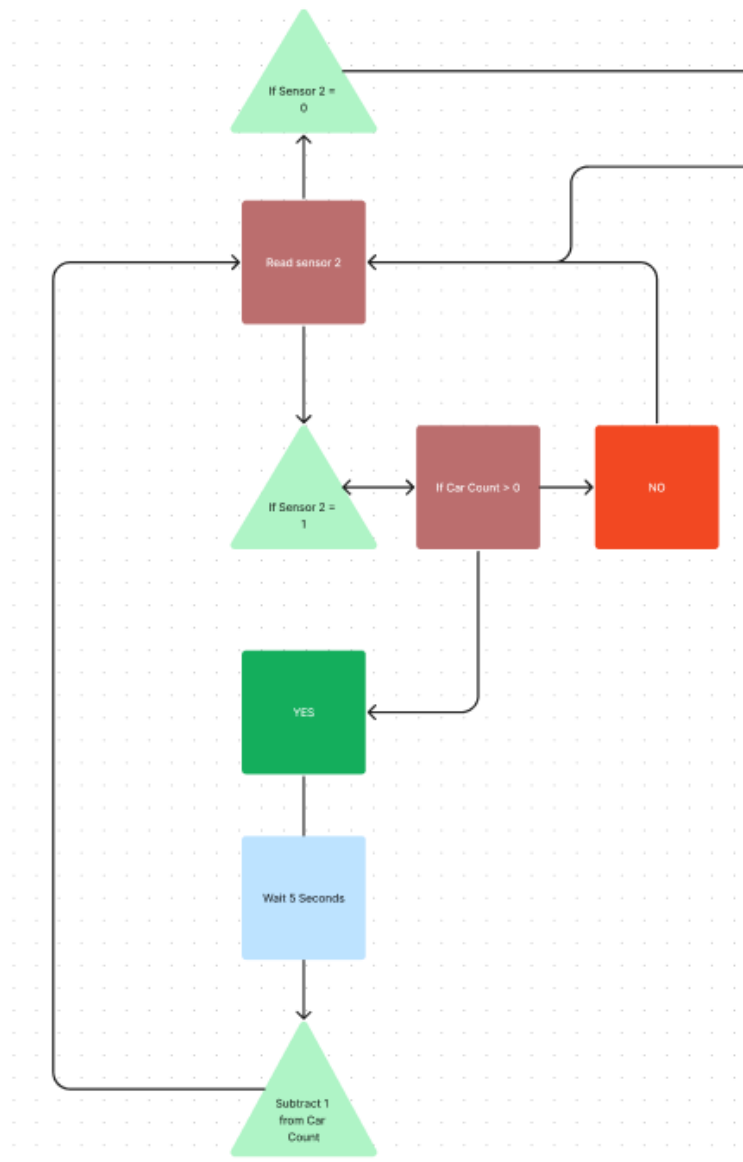
The exit section of the flowchart starts with the exit sensor being read. If the sensor is low, it will continuously loop back to the start as there is nothing that needs to happen at this point.

However, when the sensor goes high, it is again matched with a variable.

This variable is also key to keeping track of the car count. "if car count > 0" means that it will only allow the rest of the code to run if the car count is more than 0, as we don't want the code to begin subtracting into the negative values.

This would lead to inaccurate values for the number of cars in the car park.

When both are true, it waits 5 seconds for the car to be away from the sensor, this will aid in avoiding subtracting multiple times for the same car.



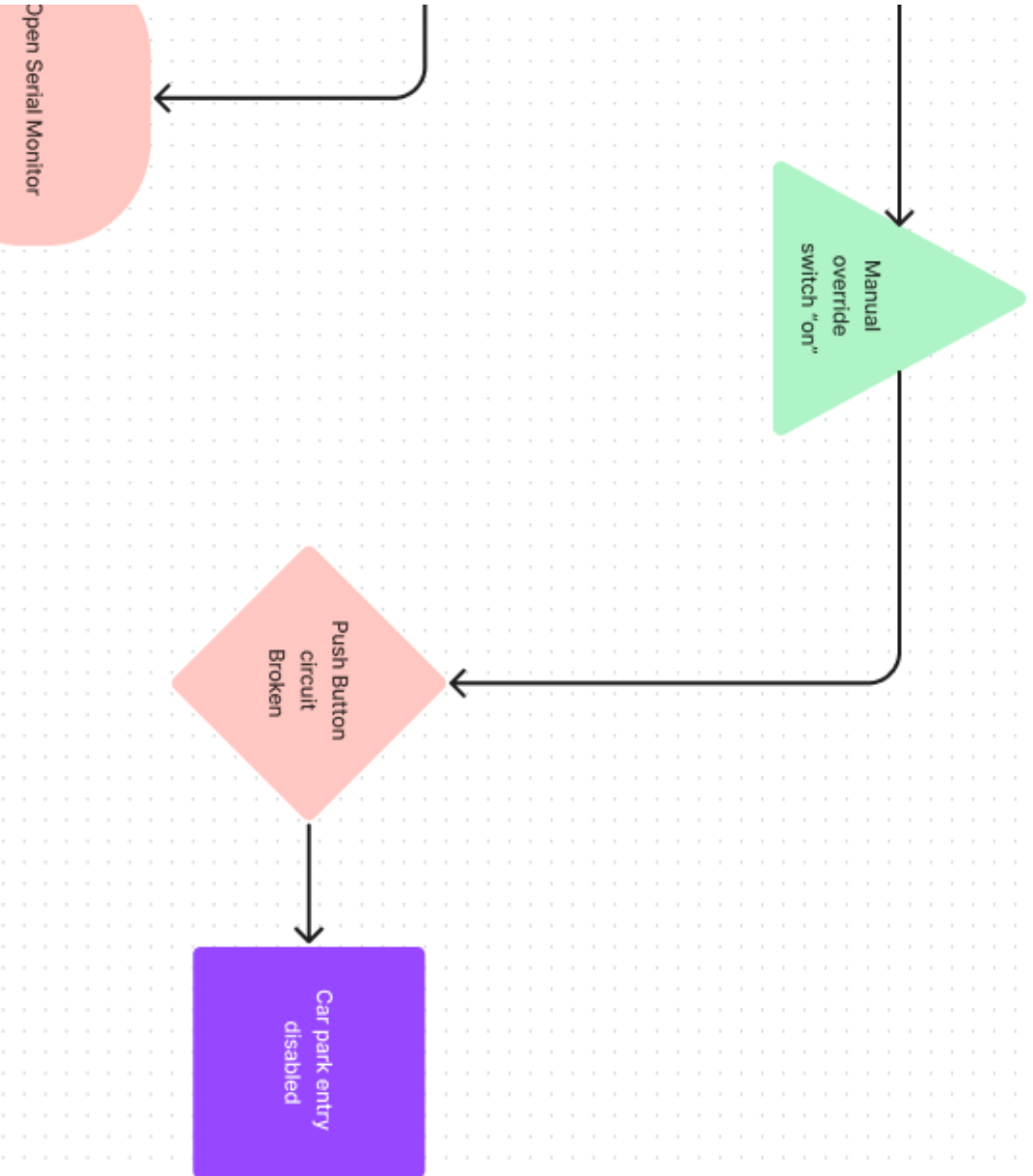
OPERATOR INTERFACE

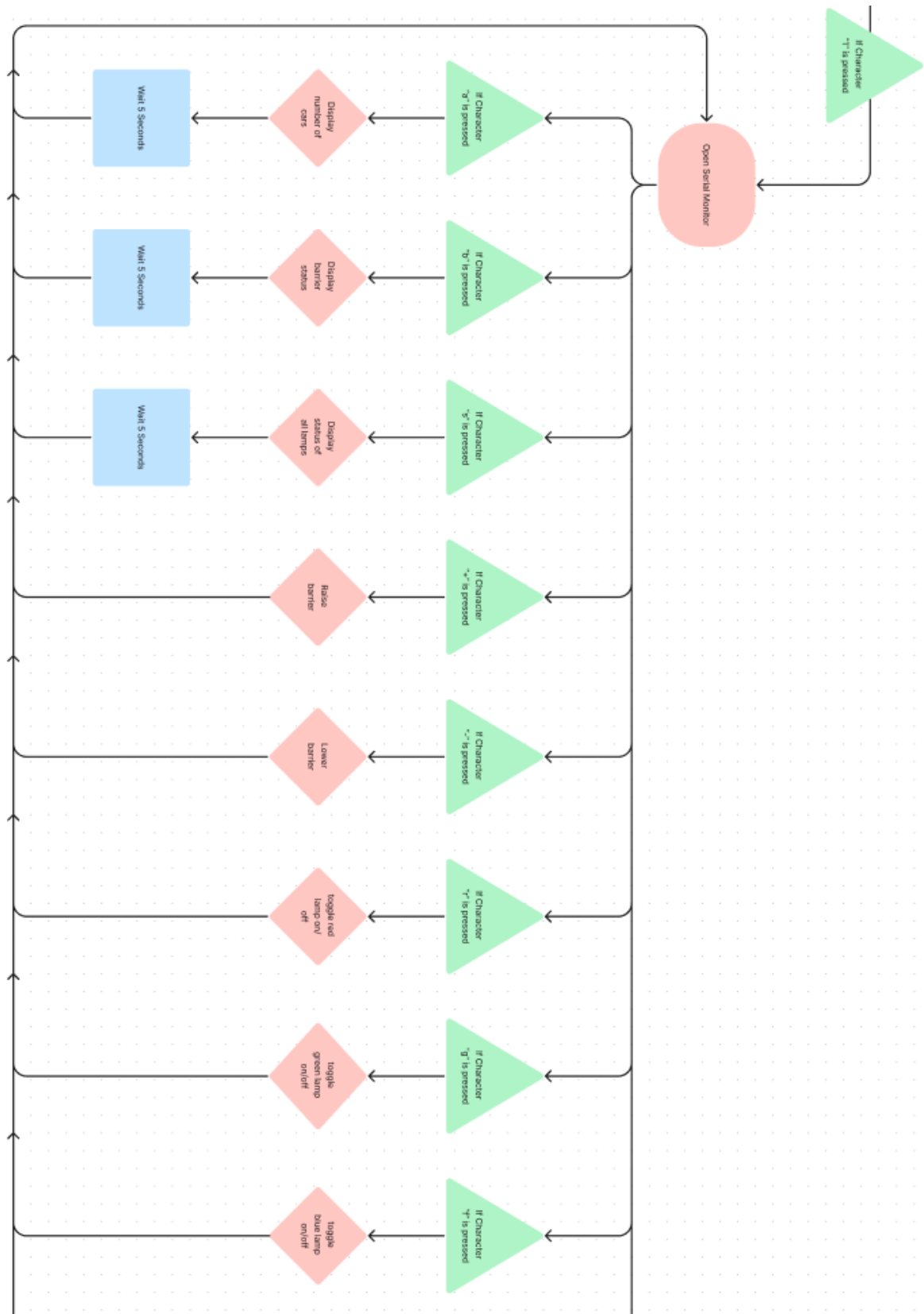
The operator interface has specific functions, set out by the brief. It is accessible at any given time by pressing "1", which opens the serial monitor menu.

Once in the serial monitor, there are numerous options that have specific functions:

- "a" will display the number of cars, wait 5 seconds to give the operator a chance to read the information, then loop back to the main menu of the terminal.
- "b" will display the barrier status and follows the same structure.
- "s" will display the status of all lamp indicators, again, following the same method.
- "+" gives the operator the option to raise the barrier manually.
- "-" gives the operator the option to lower the barrier manually.
- "r" allows the operator to toggle the red lamp on/off.
- "g" allows the operator to toggle the green lamp on/off.
- "f" allows the operator to toggle the blue (full) lamp on/off.
- To disable any entry to the car park, I opted for a manual override switch (SPDT). This switch has been incorporated into the push button circuit for the barrier. When the switch is set to "break" the circuit, the push button no longer gets read as an input by pin 2.

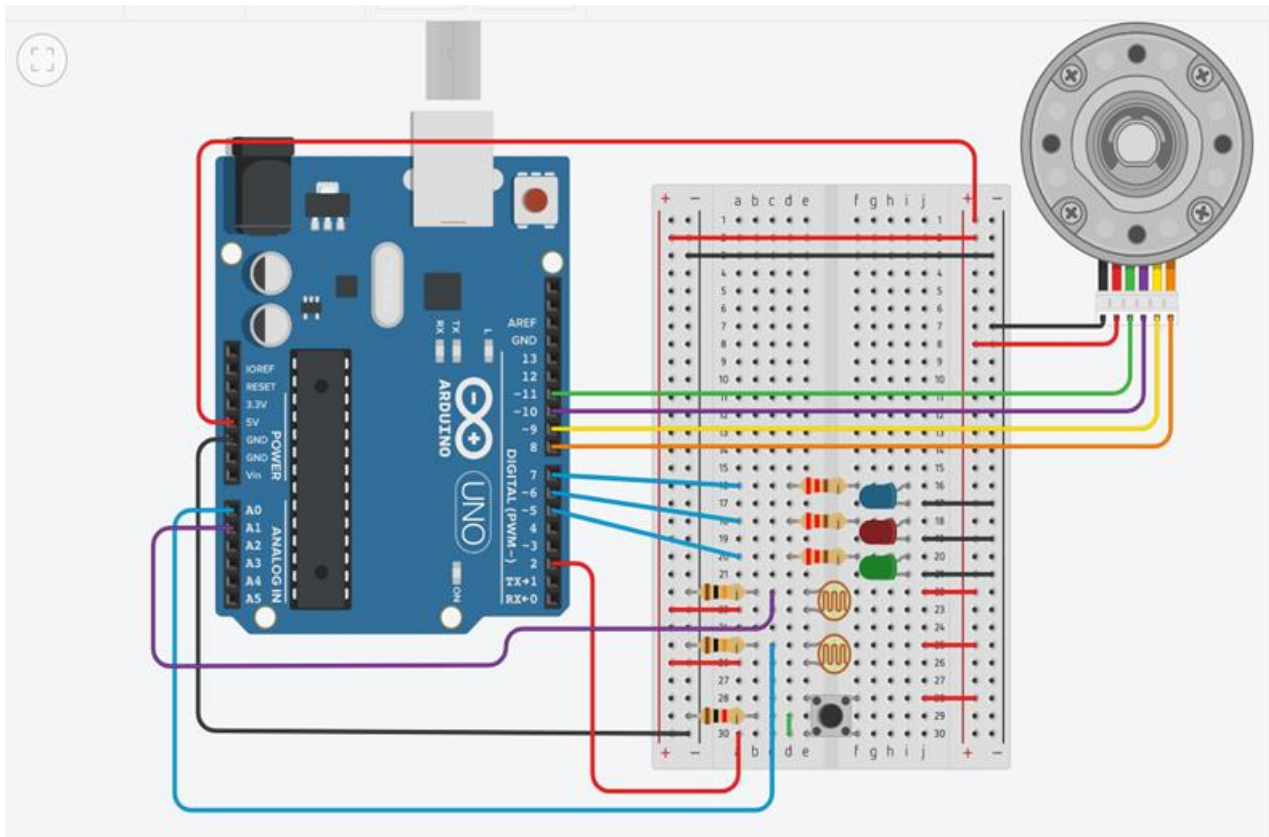
Note, how from characters "+" to "f", the wait functions are no longer present. This is because a state change is only available at the operator's discretion.





DIGITAL PROTOTYPE

After creating the flowchart, I Designed the circuit on TinkerCAD. This enabled me to prototype without using materials and creating un-needed waste.



Here we can see how the components interface to the Arduino as per the brief:

- Red LED – Pin 6
- Green LED – Pin 5
- Blue (Full) LED – Pin 7
- Stepper motor – Pins 8, 9, 10, 11
- Push Button – Pin 2

As I decided to Use LDR's for the sensors, Pins 3 & 4 were not suitable due to them being digital I/O's. Therefore, I opted for analogue pins A0 & A1.

B.O.M (BILL OF MATERIALS)

Once I had completed the first stage of prototyping, I created a B.O.M.

This element is important, as it ensures that all the components are obtained prior to building the physical prototype and means that it is easy to replicate if needed.

Component	Value	Quantity
Resistor	220Ω	3
Resistor	1kΩ	1
Resistor	10kΩ	2
LDR	/	2
LED-BLUE	5mm	1
LED-RED	5mm	1
LED-GREEN	5mm	1
Microcontroller	Segments328	1
Stepper Motor	28BYJ-48	1
Motor Driver	ULN2003	1
Push Button	Tactile Switch 6x6mm	1
Red Link Wire	1/0.6 single core	N/A
Black Link Wire	1/0.6 single core	N/A
Green Link Wire	1/0.6 single core	N/A

CODING

Below is the code created, each element has been explained with the use of comments, within the Arduino IDE:

```
#include <Stepper.h>

const int red = 6;           // setting each pin name
const int green = 5;         // ^
const int full_led = 7;      // ^
const int button = 2;        // ^
const int entry_ldr = A0;    // ^
const int exit_ldr = A1;     // ^

const int stepsPerRevolution = 512; // setting the stepper motors rotation
value

int redHigh = HIGH; // setting initial LED states that can be called later
on
int greenHigh = LOW; // ^
int fullHigh = LOW; // ^

int barrier; // barrier variable to call later on
int cars = 0; // car quantity variable to add or subtract from
int x = 0; // analogue entry sensor variable
int outsense = 0; // analogue exit sensor variable

unsigned int fullpark = 10; // maximum car value integer

Stepper myStepper (stepsPerRevolution, 8, 10, 9, 11); // initialising the I/O
pins for the stepper motor **note the pin order**

void setup() {
  myStepper.setSpeed(60); // setting the stepper motor speed
  Serial.begin(9600); // initialising the serial monitor

  pinMode(red, OUTPUT); // setting each pin as an input or output
  pinMode(green, OUTPUT); // ^
  pinMode(full_led, OUTPUT); // ^
  pinMode(button, INPUT); // ^
  pinMode(entry_ldr, INPUT); // ^
  pinMode(exit_ldr, INPUT); // ^

  digitalWrite(red, redHigh); // writing the states of each LED for
when the circuit is initially powered on, using the variables created above
  digitalWrite(green, greenHigh); // ^
  digitalWrite(full_led, fullHigh); // ^
```

```

}

void loop() {

    if (cars < fullpark){ // initial "if" statement, if the value of cars is less
than the carpark maximum...

        if(digitalRead(button)==1) // if the entrance button has been pressed...
        {
            myStepper.step(-stepsPerRevolution); // raise the barrier
            digitalWrite(red, LOW); // turn the red LED off
            digitalWrite(green, HIGH); // turn the green LED on

            do {
                delay(500);
                x = analogRead(entry_ldr); // using the variable "x" as the entry LDR
value
            }
            while (x < 900); // whilst the sensor value is less then 900 (a car is
now above the sensor)...

            delay(1000); // wait 1 second

            myStepper.step(stepsPerRevolution); // lower the barrier
            digitalWrite(green, LOW); // turn the green LED off
            digitalWrite(red, HIGH); // turn the red LED on
            cars++; // add 1 to the car count variable
            delay(1000); // wait 1 second
            barrier == 0; // barrier state is now 0 (lowered)
        }
    }

    {
        outsense = analogRead(exit_ldr); // using the variable "outsense" as
the exit LDR value

        if(outsense <= 400 && cars >=1) // boolean expression, if the exit LDR
value is less than or equals to 400 AND the car count variable is more than or
equals to 1...
        {
            delay(2000); // wait 2 seconds
            cars--; // subtract 1 from the car count variable
            digitalWrite(full_led, LOW); // keep the blue LED (full led) LOW
        }
    }
}

```

```

if (cars == fullpark){ // if the amount of cars equals the maximum capacity
(full park)...
    delay(500); // wait 500ms for the system to stabilise...
    digitalWrite(full_led, HIGH); // turn the blue LED on
    digitalWrite(red, HIGH); // turn the red LED on
    delay(2000); // wait 2 seconds for the system to stabilise

    delay(300);}

    if (Serial.available() > 0){ // if the serial monitor COM port is
available...
    char c = Serial.read(); // setup for reading keyboard inputs
    if (c == '1') { // if 1 is pressed...
        Serial.println("Operator Menu, please select from the following options:-
"); // print the operator menu welcome line, with the following
options...
        Serial.println(" Press r to toggle the red light
on/off."); // ^
        Serial.println(" Press g to toggle the green light
on/off."); // ^
        Serial.println(" Press f to toggle the full sign
on/off."); // ^
        Serial.println(" Press a to view how many cars are in the
garage."); // ^
        Serial.println(" Press + to raise
barrier."); // ^
        Serial.println(" Press - to lower
barrier."); // ^
        Serial.println(" Press s to view the status of the
LED's."); // ^
        Serial.println(" Press b to check the status of the
barrier."); // ^
    }

// Toggle the red LED on/off
    if (c == 'r'){
        if(redHigh == HIGH){
            redHigh = LOW;
            digitalWrite(red, redHigh);
            Serial.println("Red Light OFF");
        } else {
            redHigh = HIGH;
            digitalWrite(red, redHigh);
            Serial.println("Red Light ON");
        }
    }
}

```

```

// Toggle the green LED on/off
    if (c == 'g'){
        if(greenHigh == HIGH){
            greenHigh = LOW;
            digitalWrite(green, greenHigh);
            Serial.println("Green Light OFF");
        } else {
            greenHigh = HIGH;
            digitalWrite(green, greenHigh);
            Serial.println("Green Light ON");
        }
    }

// Toggle the blue LED (full sign) on/off
    if (c == 'f'){
        if(fullHigh == HIGH){
            fullHigh = LOW;
            digitalWrite(full_led, fullHigh);
            Serial.println("Full sign OFF ");
        } else {
            digitalWrite(button, LOW);
            fullHigh = HIGH;
            digitalWrite(full_led, fullHigh);

            Serial.println("Full sign ON ");
        }
    }

// Check the car quantity at any given time
    if (c == 'a'){
        Serial.print("The current number of cars in the garage is ");
        Serial.println(cars);
    }

// Raise the barrier manually
    if (c == '+'){
        myStepper.step(-stepsPerRevolution);
        Serial.println("Raising barrier");
        digitalWrite(red, LOW);
        digitalWrite(green, HIGH);
        barrier = 1;
    }

// Lower the barrier manually
    if (c == '-'){
        myStepper.step(stepsPerRevolution);
        Serial.println("Lowering barrier");
        digitalWrite(red, HIGH);
    }

```

```

        digitalWrite(green, LOW);
        barrier = 0;
    }

    // check the status of the barrier at any given time
    if (c == 'b'){
        Serial.println("checking barrier status.... ");

        if (digitalRead(red) == HIGH && barrier == 0)
        {
            Serial.println("lowered");}
        else if (digitalRead(red) == LOW and barrier == 1) {
            Serial.println("raised");
        }
        else { Serial.println("lowered");
        }
    }

    // check the status of all LED's at any given time
    if (c == 's'){
        if (digitalRead(red) == HIGH){
            Serial.println("Red LED is ON ");}
        else if (digitalRead(red) == LOW)
        {
            Serial.println("Red LED is OFF");
        }
        if (digitalRead(green) == HIGH){
            Serial.println("Green LED is ON ");}
        else if (digitalRead(green) == LOW)
        {
            Serial.println("Green LED is OFF");
        }
        if (digitalRead(full_led) == HIGH){
            Serial.println("Blue LED is ON ");}
        else if (digitalRead(full_led) == LOW)
        {
            Serial.println("Blue LED is OFF");
        }
    }
}
}

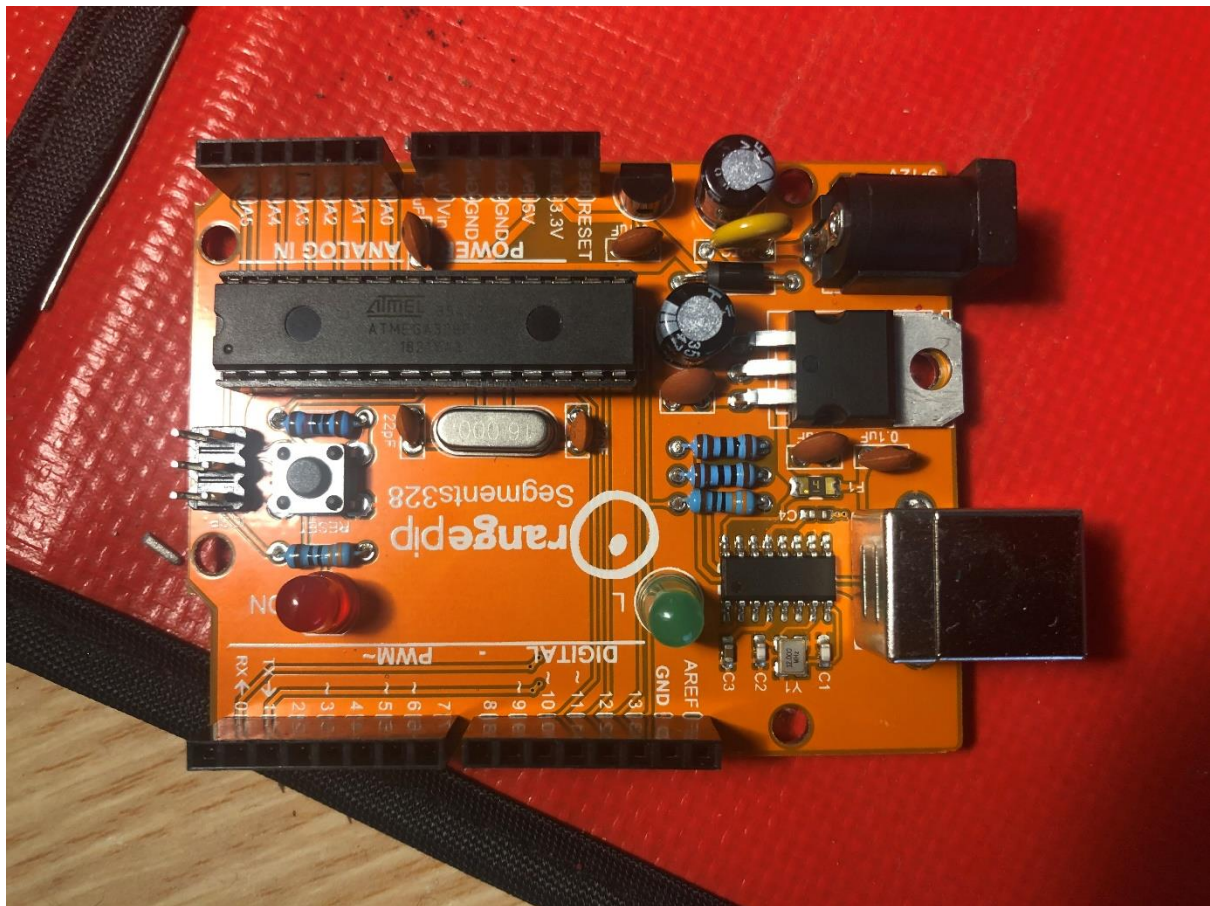
```

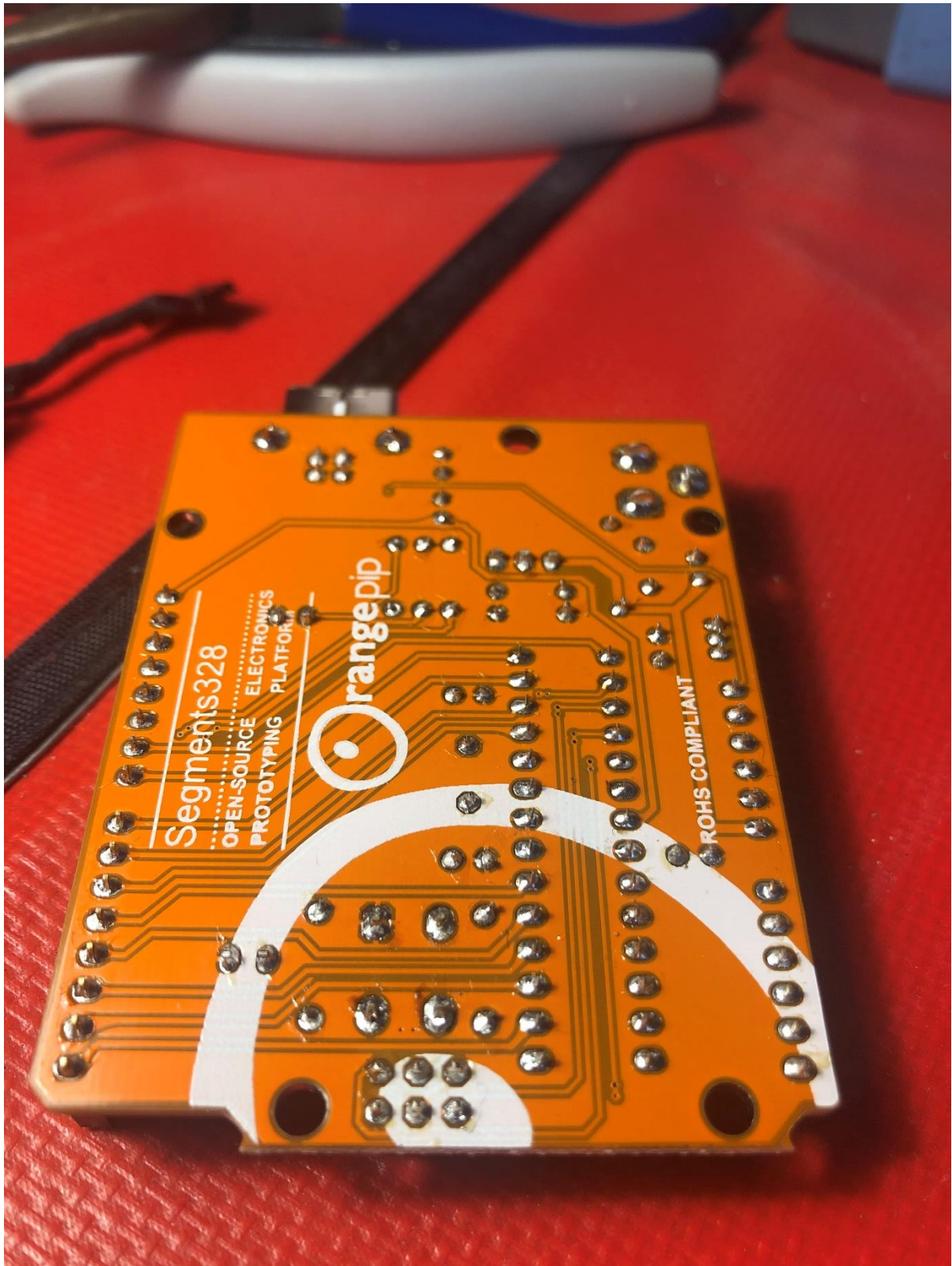
3D PROTOTYPE

MICROCONTROLLER BUILD (SEGMENTS328 ORANGEPIP)

The first step I took to create the prototype was building the OrangePip, This would then allow me to connect the peripherals and other elements together.

To do this, I followed the build Instructions Provided by www.rapidonline.com .





BLINK TEST (ARDUINO IDE)

The BLINK test is commonly used to test Arduino-type microcontrollers for basic functionality. It is pre-written code that can be found by going to:

File>Examples>Basic>Blink.

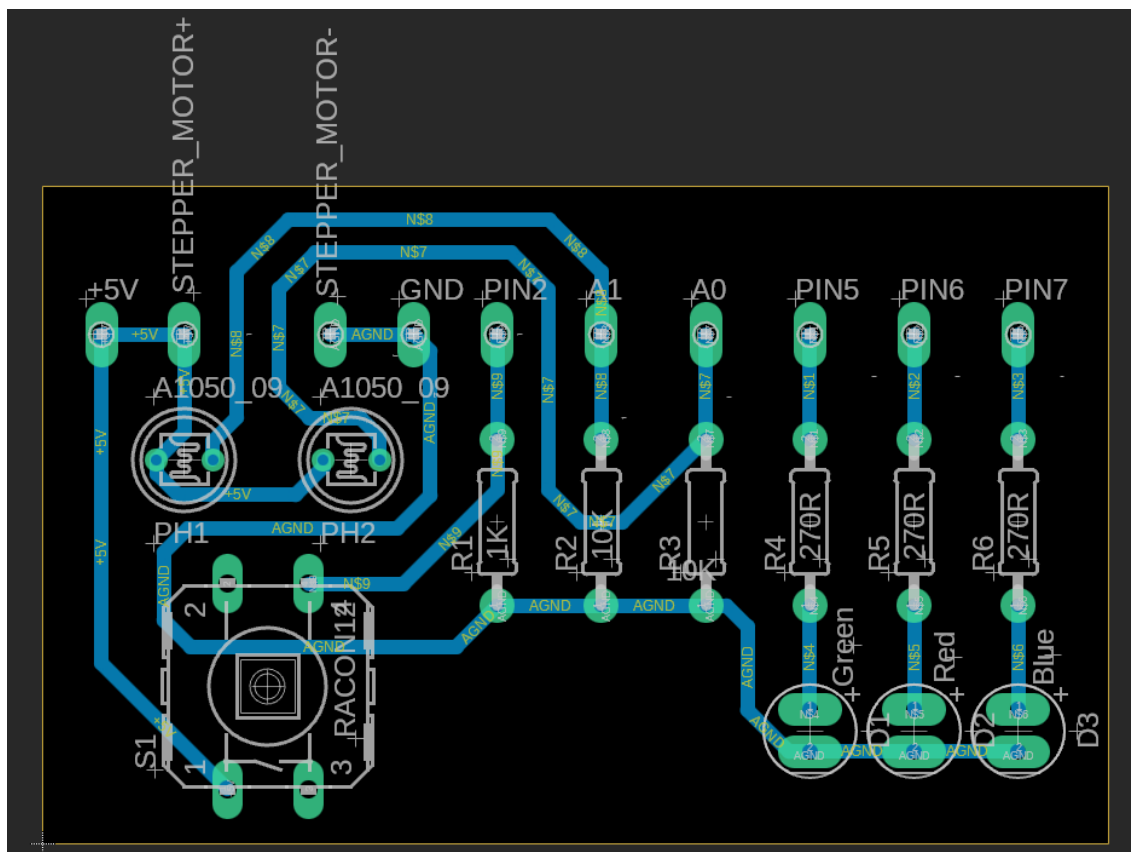
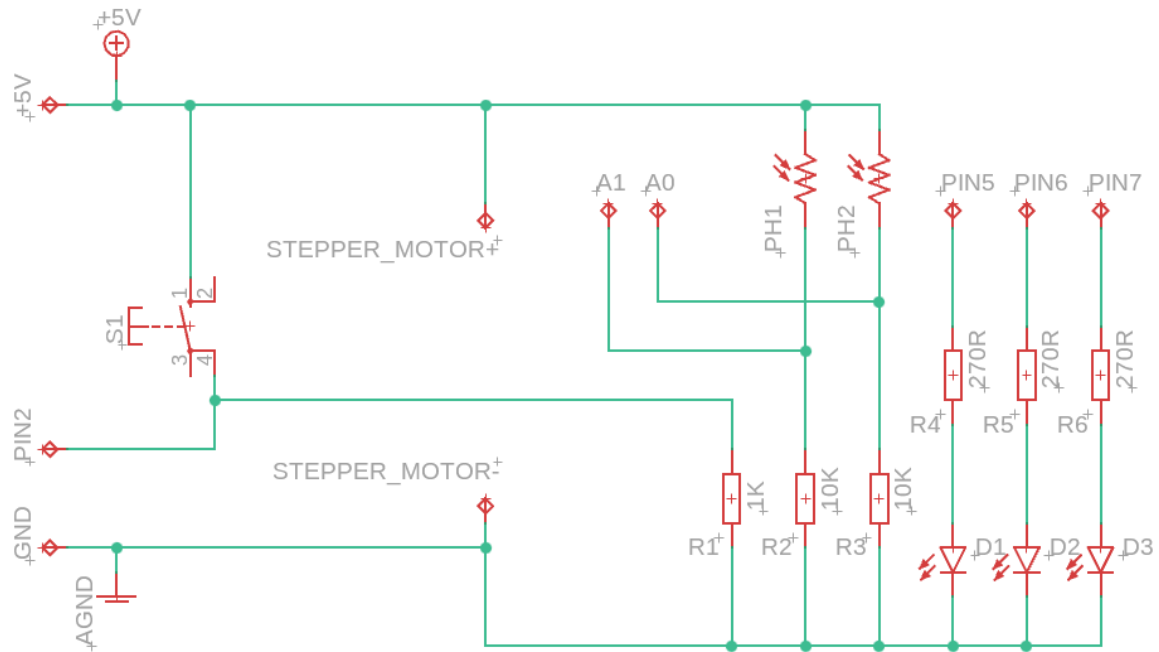
Once the code is uploaded, the onboard LED (connected to digital I/O pin 13) will flash. This is achieved by using the `digitalWrite` function and `delay()`, along with other standard structures to turn the LED on/off for any desired periods of time.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

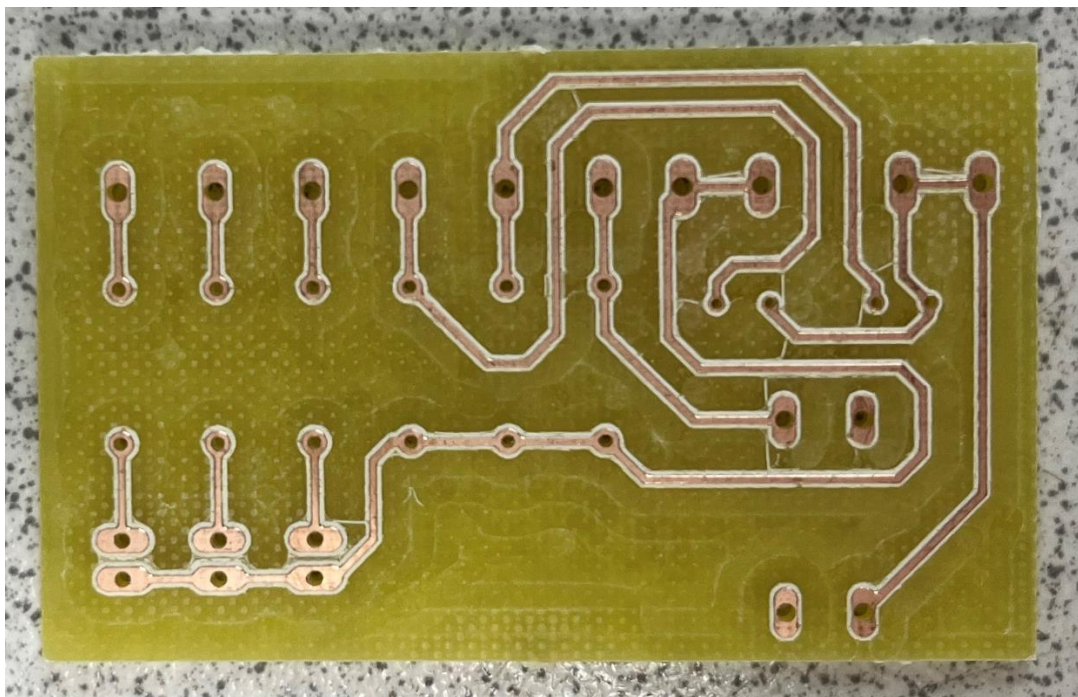
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage
level)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage
LOW
  delay(1000);                     // wait for a second
}
```

Verification of the OrangePip's functionality was now completed.

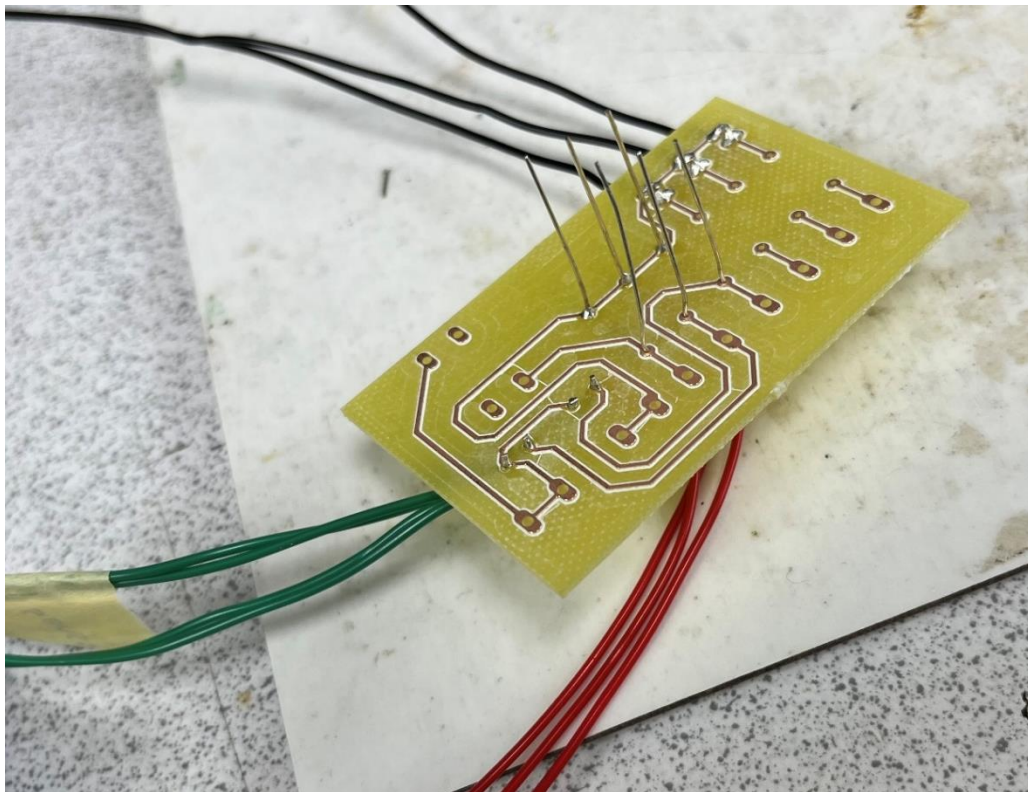
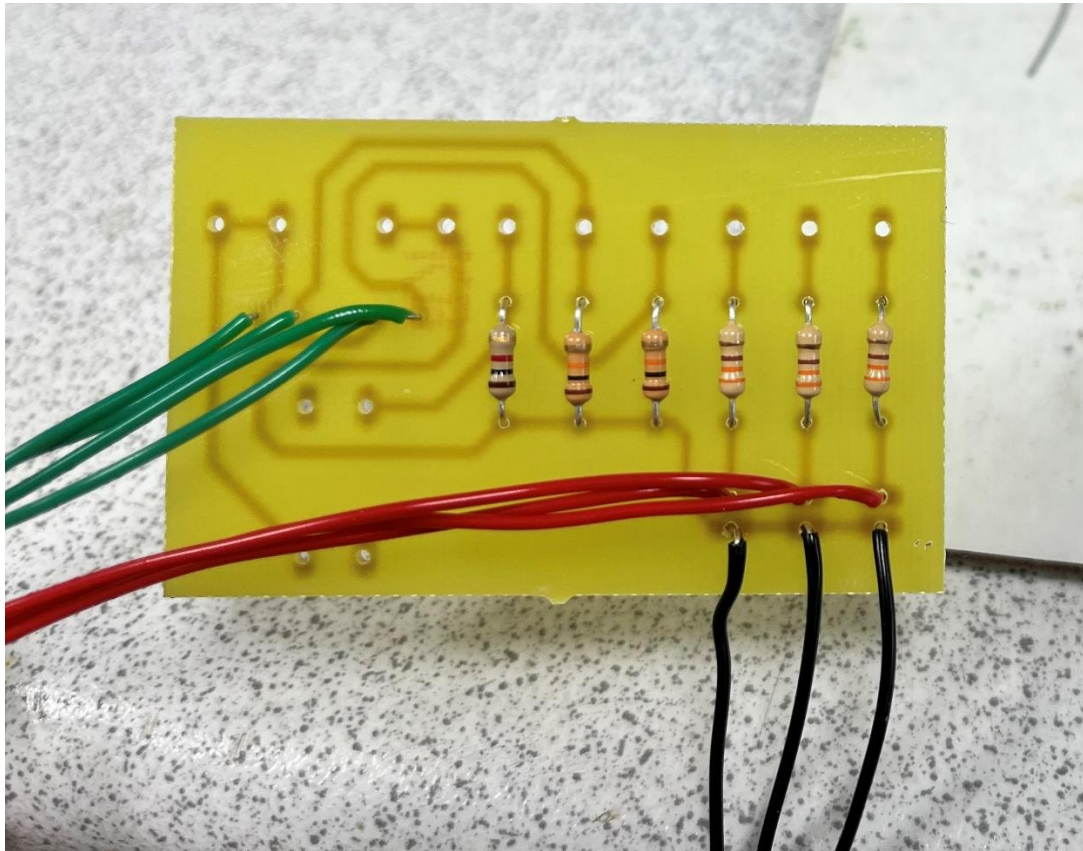
I then converted the breadboard design created on TinkerCAD into a PCB (this was done using Fusion 360):

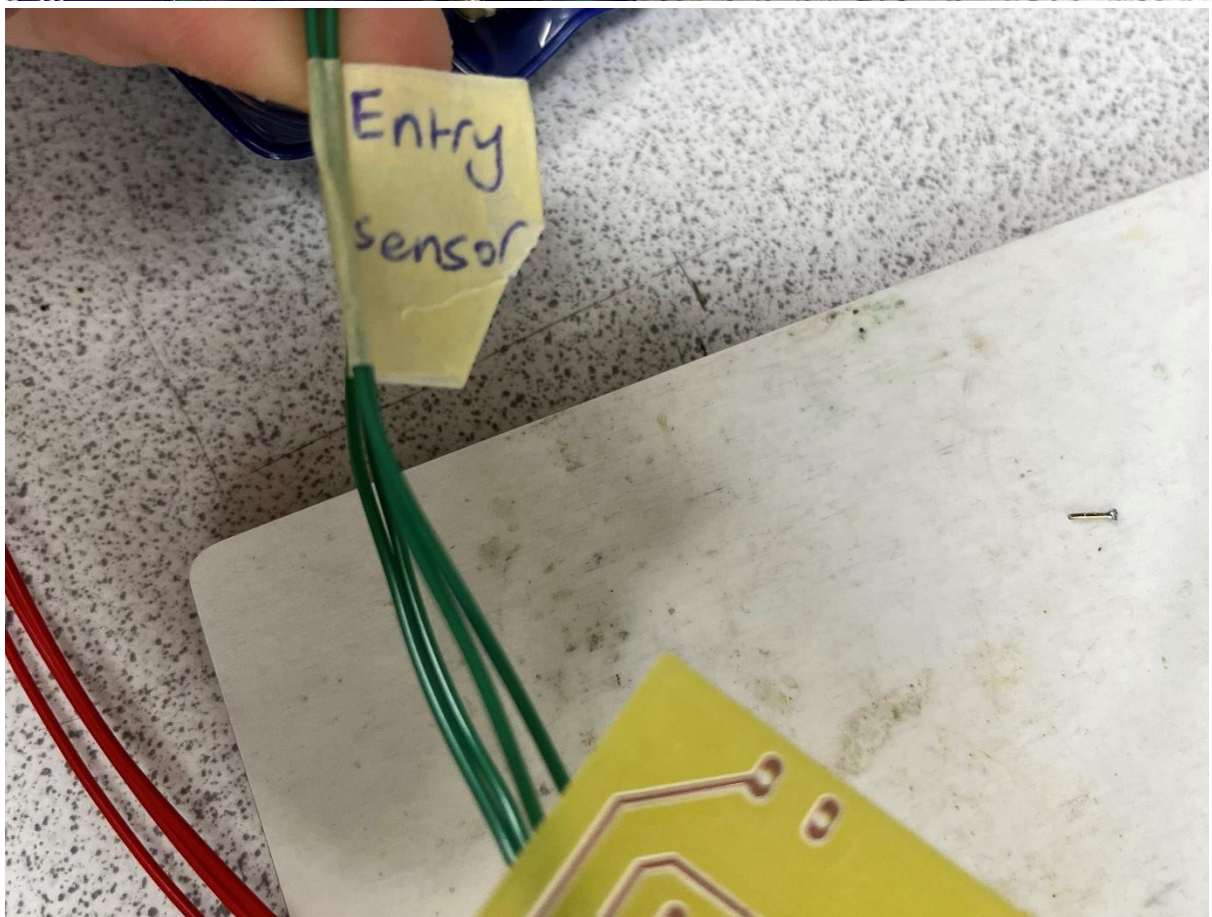
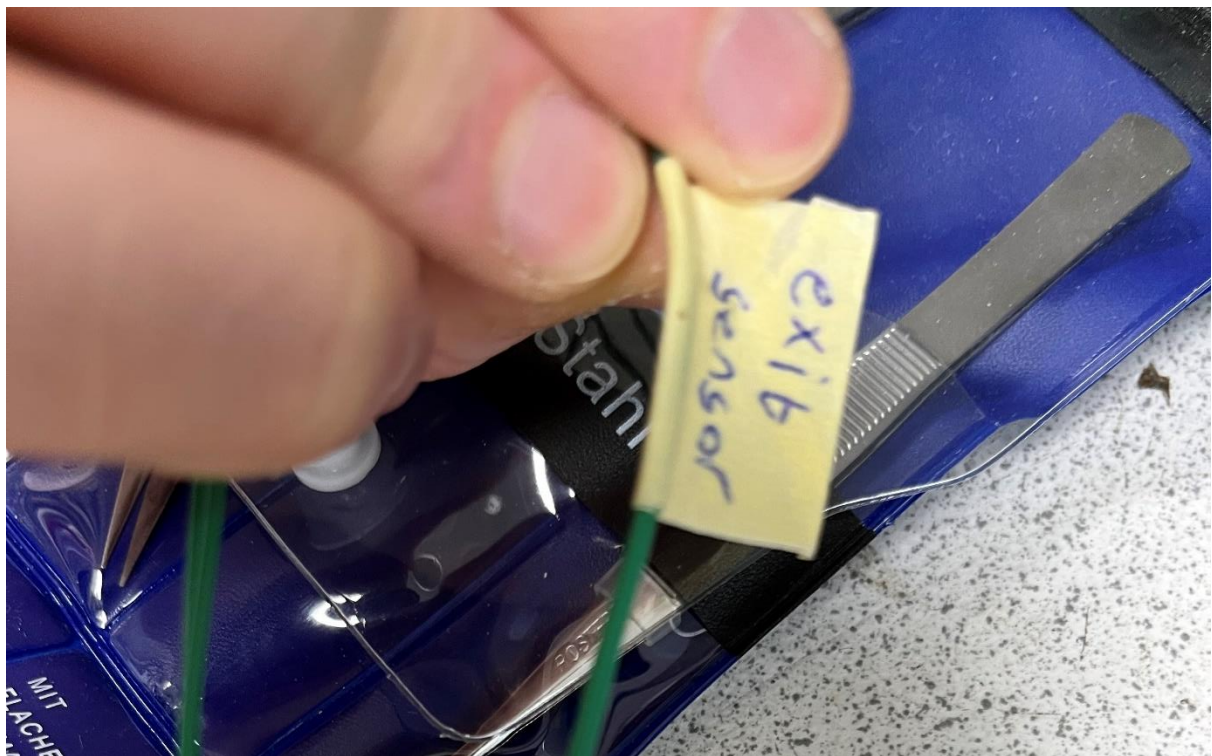


Next, I used a PCB milling machine to create the physical PCB. The milling machine used was an LPKF Promat S63, with FR4 Single sided copper material:



After the PCB was milled, I populated it with the components specified in the B.O.M and labelled the cables where necessary:





Once the PCB was fully populated, I ensured that all functions still worked correctly. This included calibrating the LDR's - to do this I used a short block of code, separate to the main code, that allowed me to view the live readings from the sensor, within the IDE's serial monitor.

Noting those values down, I was able to accurately input my required values into the main code.

```
int ldr;
void setup() {
  // put your setup code here, to run once:

  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  ldr = analogRead(A1);
  Serial.println(ldr);
  delay(300);
}
```

Not covered:

742
741
741
741
741
741

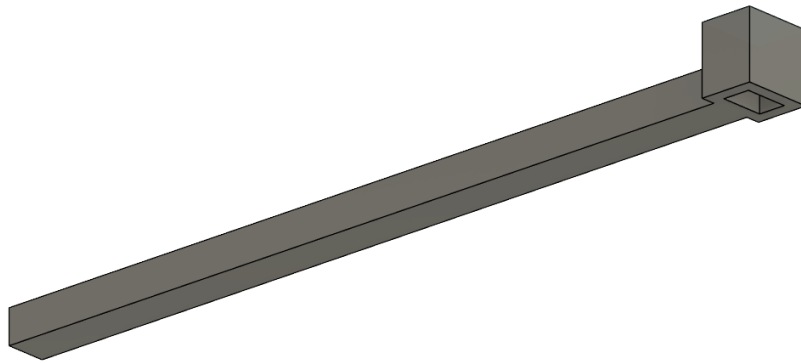
Covered:

64
69
71
71
73
74
65

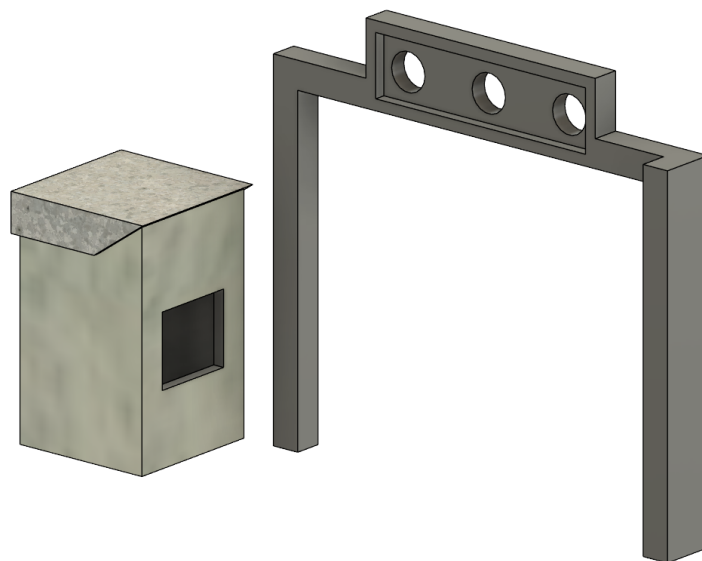
3D MODELS

I then began 3D printing the CAD models. To design the models, I again used Fusion 360. The Models were printed using PLA materials with the Flash forge Adventurer 3.

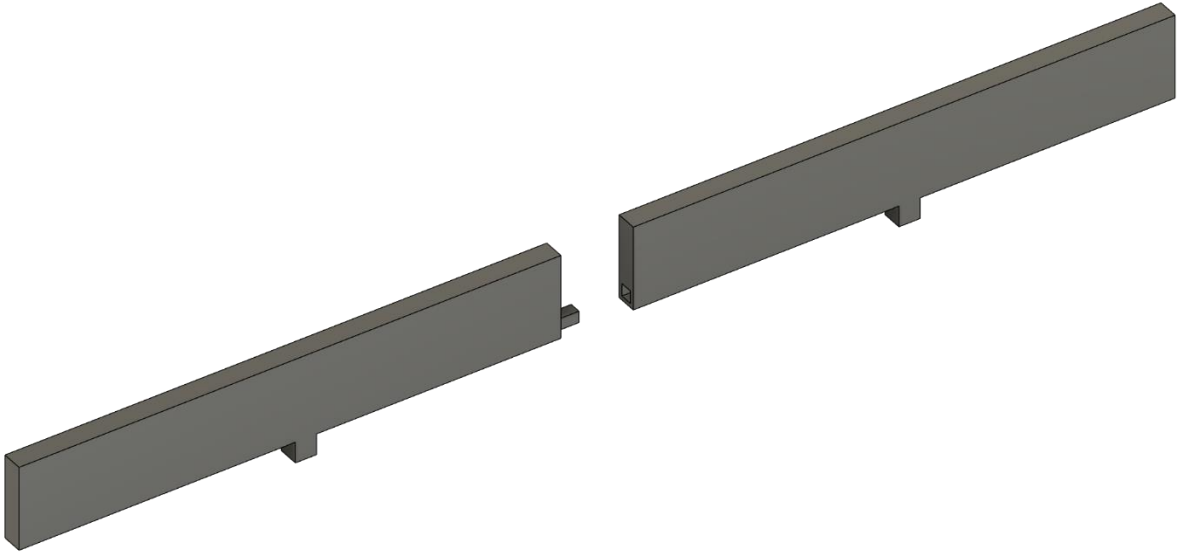
Barrier



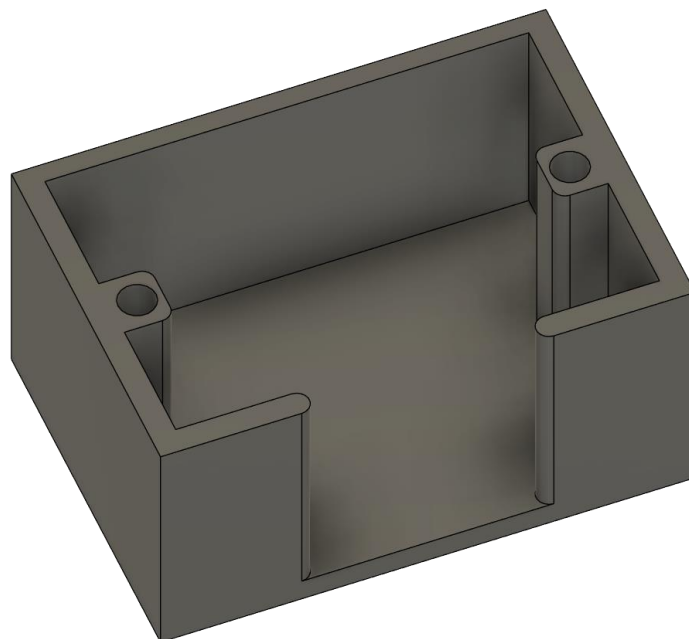
Operator's Hut & Overhead Indicator Mount



Car Park Walls

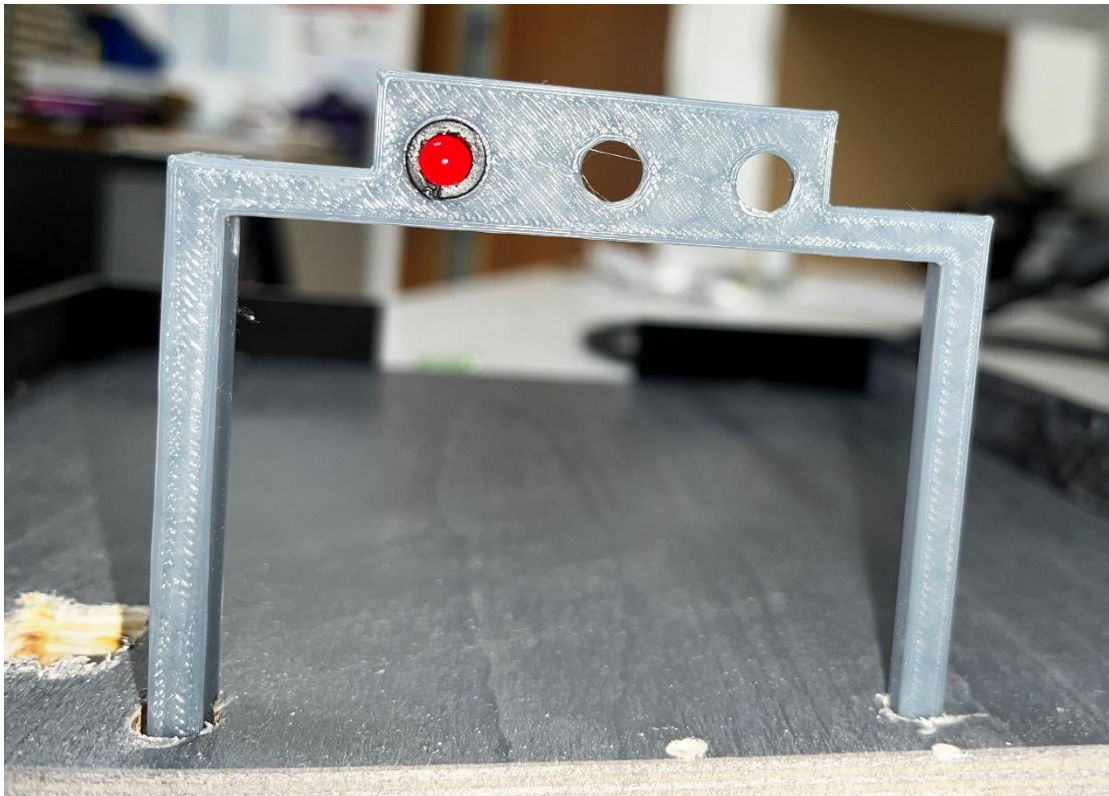
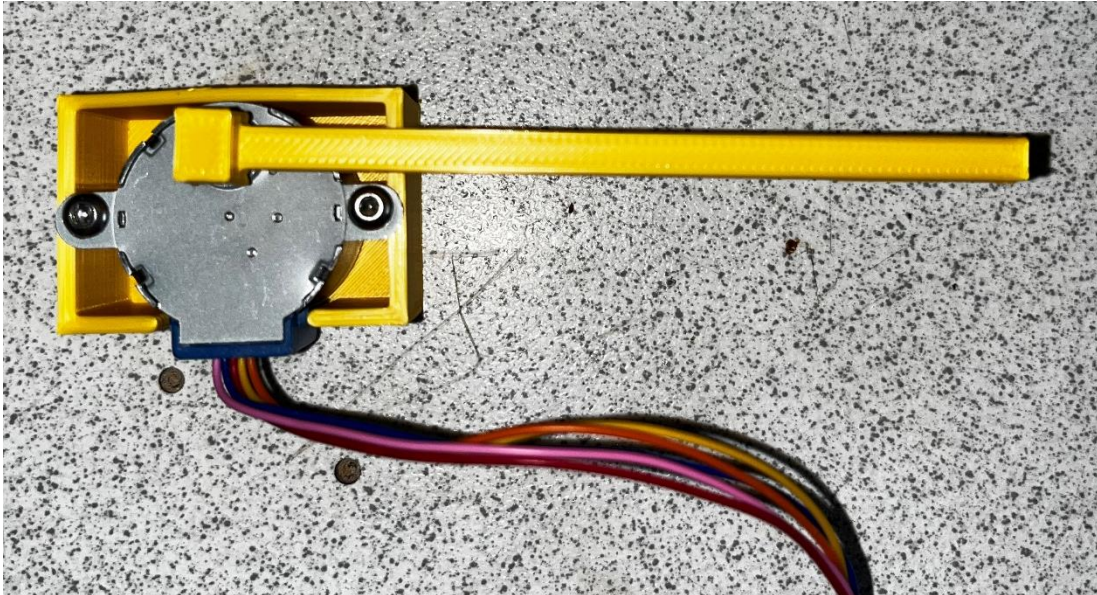


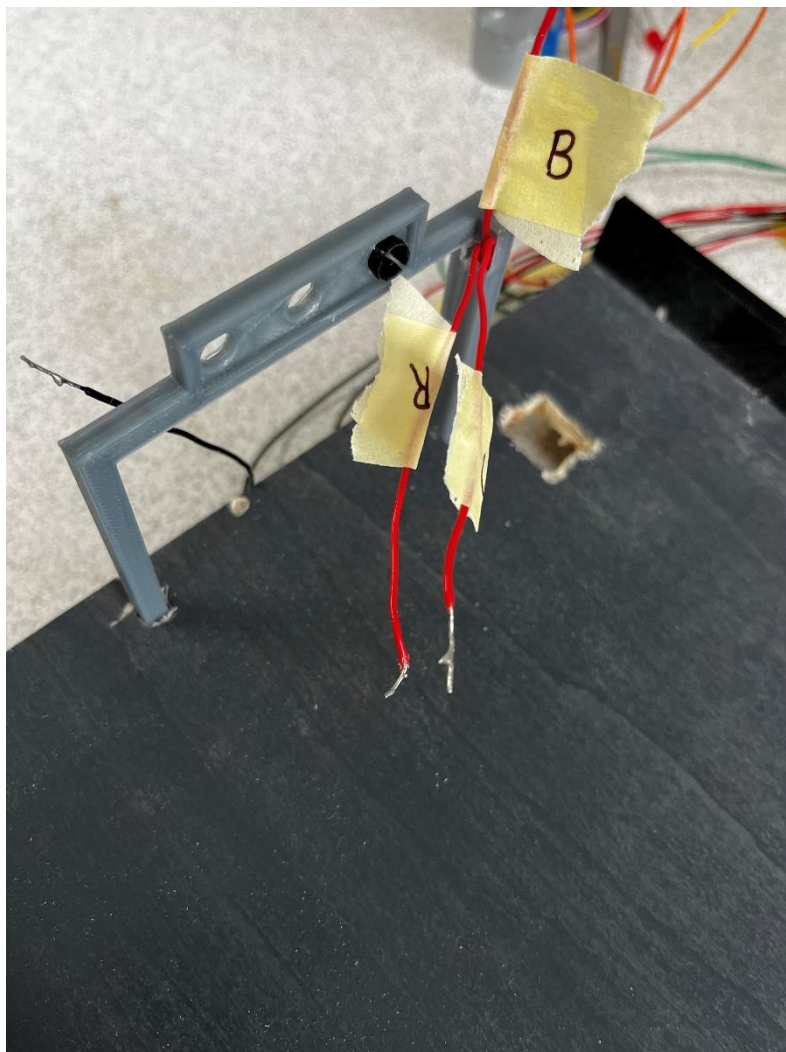
Stepper Motor Mounting box

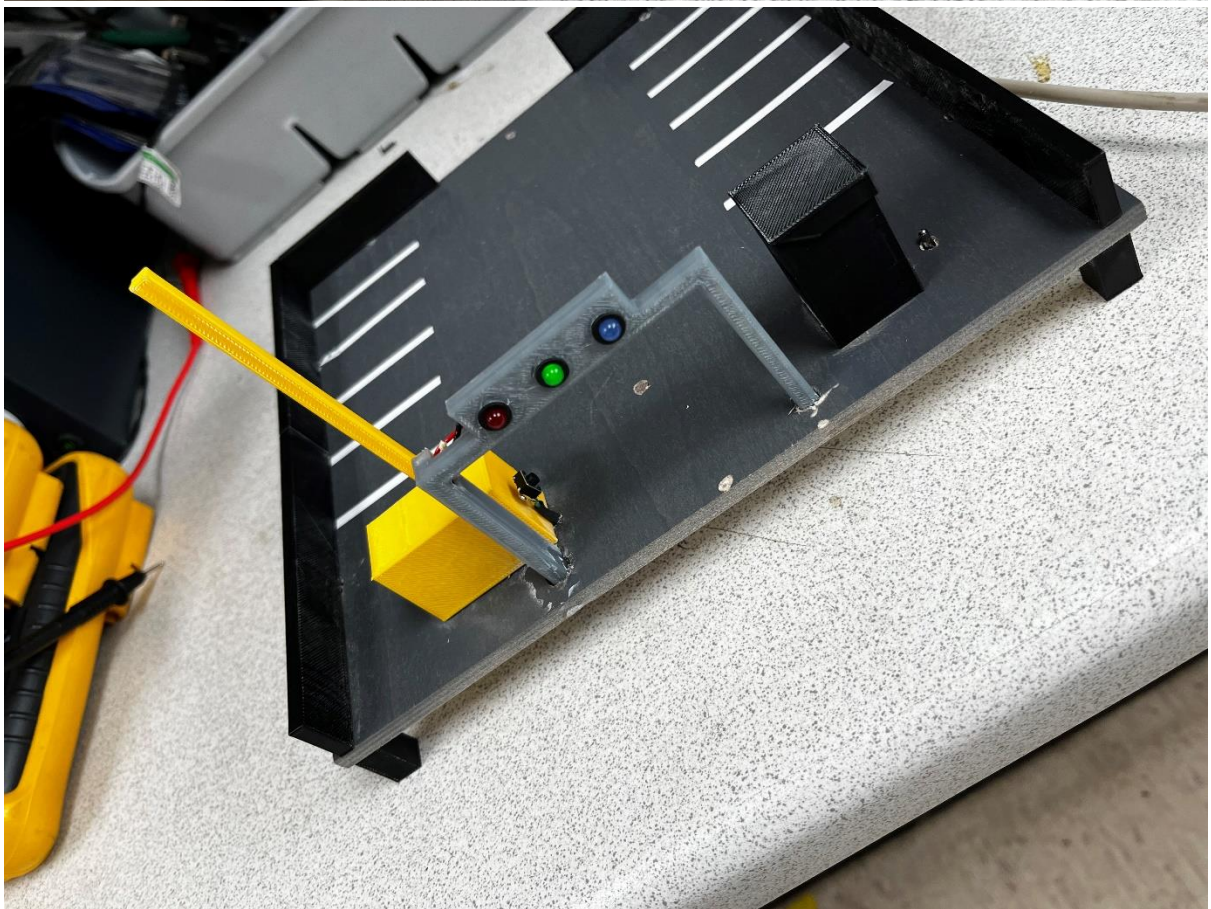
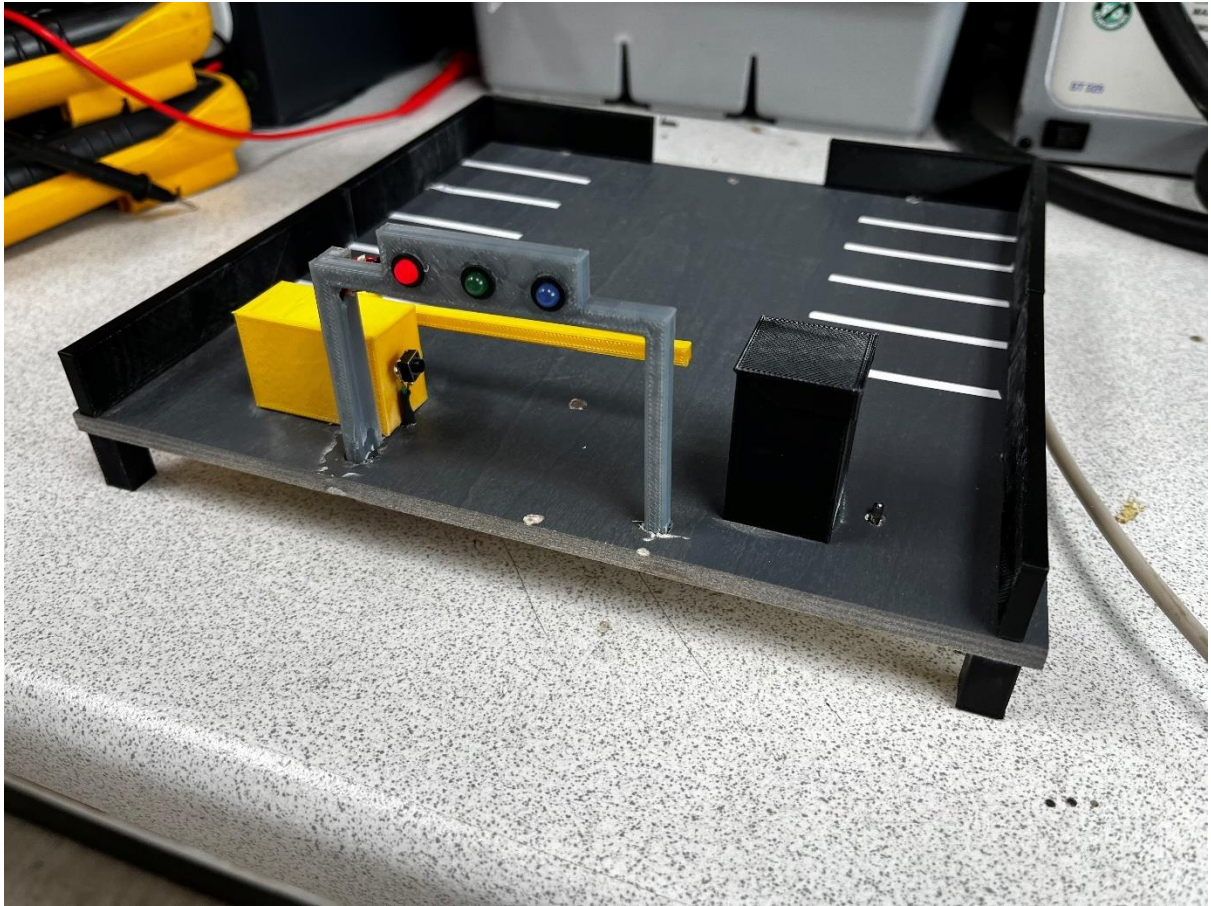


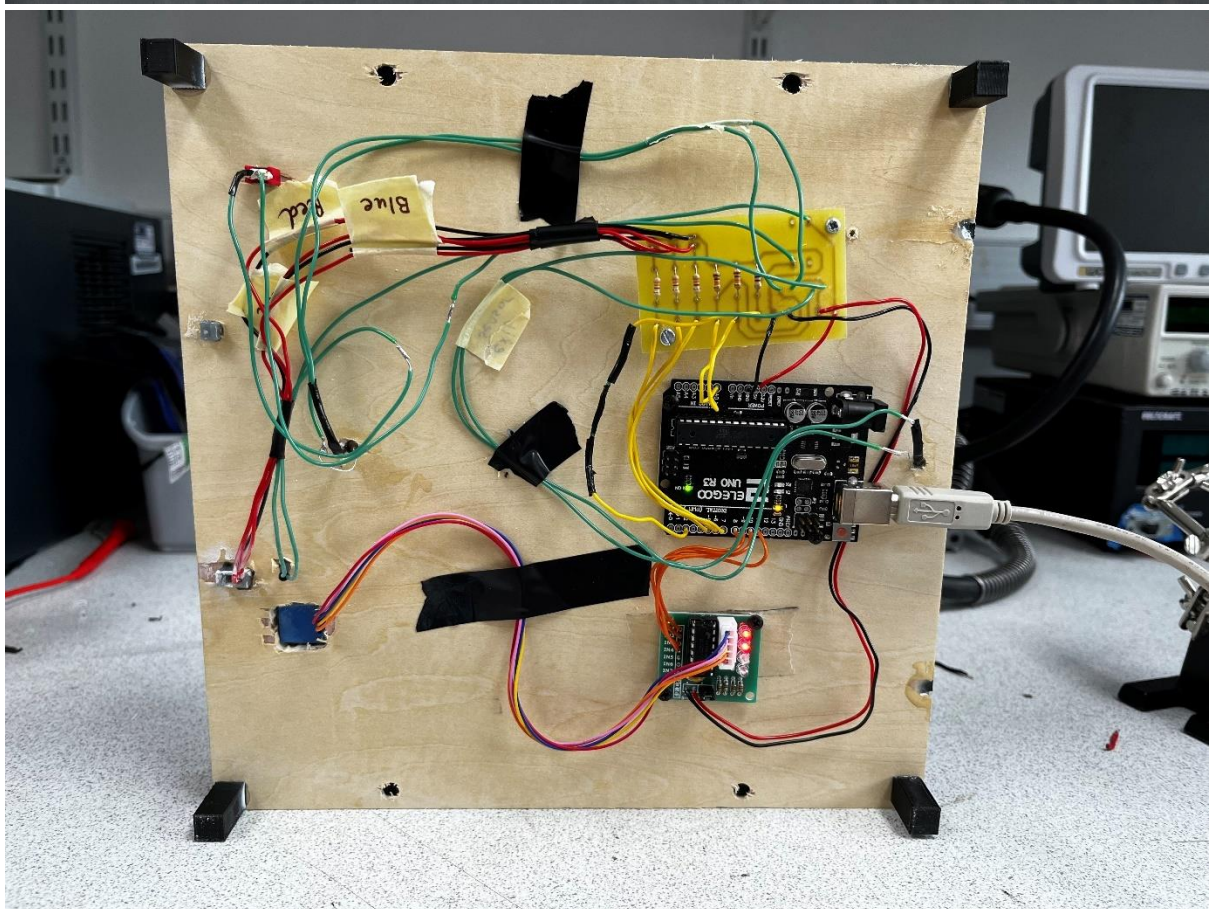
FINISHED PROTOTYPE

Once the printing had finished, I began mounting and connecting all elements together:











CONCLUSION

In conclusion, the car park operated in line with the brief entirely. The functions operate correctly with repetition which is of high importance due to its nature.

Creating the PCB and soldering all of the elements together aided in the projects repeatability, rather than relying on loose male to female connections.

However, I would have preferred to code the override function in, rather than use a physical switch. I feel like this would have created a cleaner result.

Tidying up the runs of cable underneath would have been my next step, giving the prototype a higher level of serviceability.

Lastly, I was having issues with uploading to the OrangePip, Hence why another variation of the Arduino Uno was used for the final prototype.

BIBLIOGRAPHY

Subbaroybhat, M., 2021. *What is a Microcontroller?*. [Online]

Available at: <https://uk.rs-online.com/web/content/discovery/ideas-and-advice/microcontrollers-guide>
[Accessed 23 03 2024].