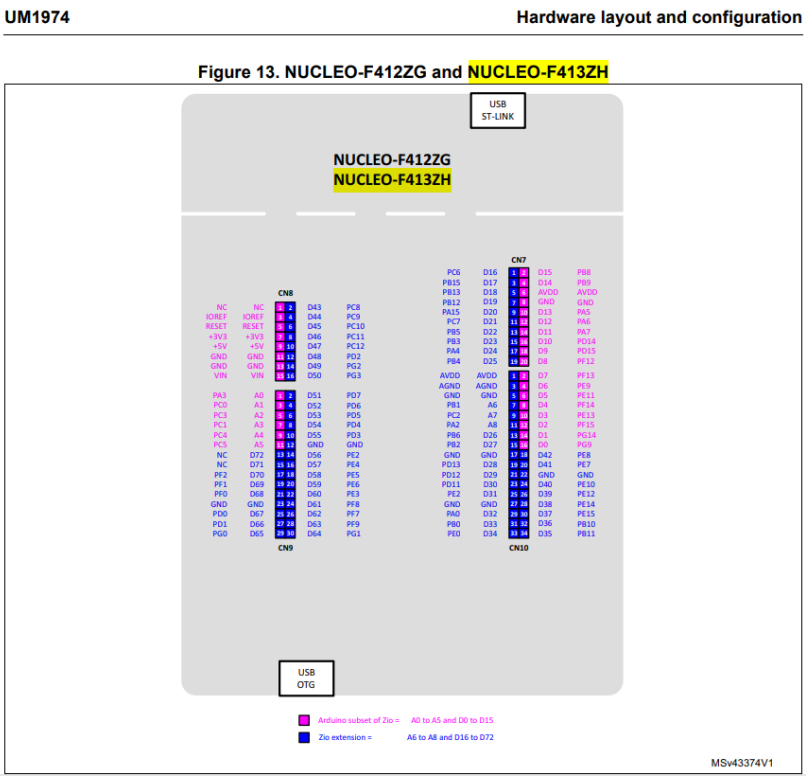
01 | Project Documentation

Aim: Develop a thermocouple sensor array which will use approximately 256 sensors to measure the temperature of a liquid throughout a container. These measurements will then be analysed and presented using visualisation software which will show how the temperature moves throughout the fluid. The overall aim is to help understand and model the heating effects of a temperature pad on a non-Newtonian fluid such as molasses so a deeper understanding can be had of how it responds to heat and therefore movement in an IBC.

1. STM32 Board

The STM board pin layout is shown below.



Code was uploaded onto the MCU (microcontroller) to allow the board to be coded with micro python.

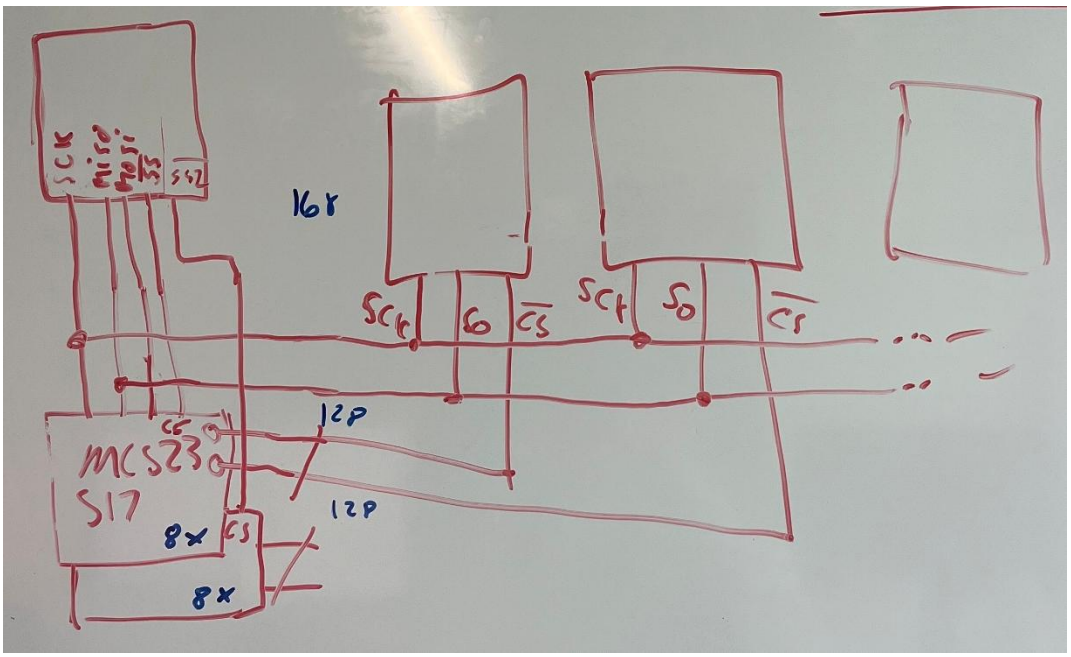
Then with Thonny IDE the code that is shown below allows the LED on the STM board to blink.

```python
import machine
import pyb
import time

def main():
    # Blinking code
    while(1):
        pyb.LED(1).on()
        time.sleep_ms(500)
        pyb.LED(1).off()
        time.sleep_ms(500)

        pyb.LED(3).on()
        time.sleep_ms(500)
        pyb.LED(3).off()
        time.sleep_ms(500)

        pyb.LED(2).on()
        time.sleep_ms(500)
        pyb.LED(2).off()
        time.sleep_ms(500)

main()
```

time_sleep_ms --> delay command in milliseconds

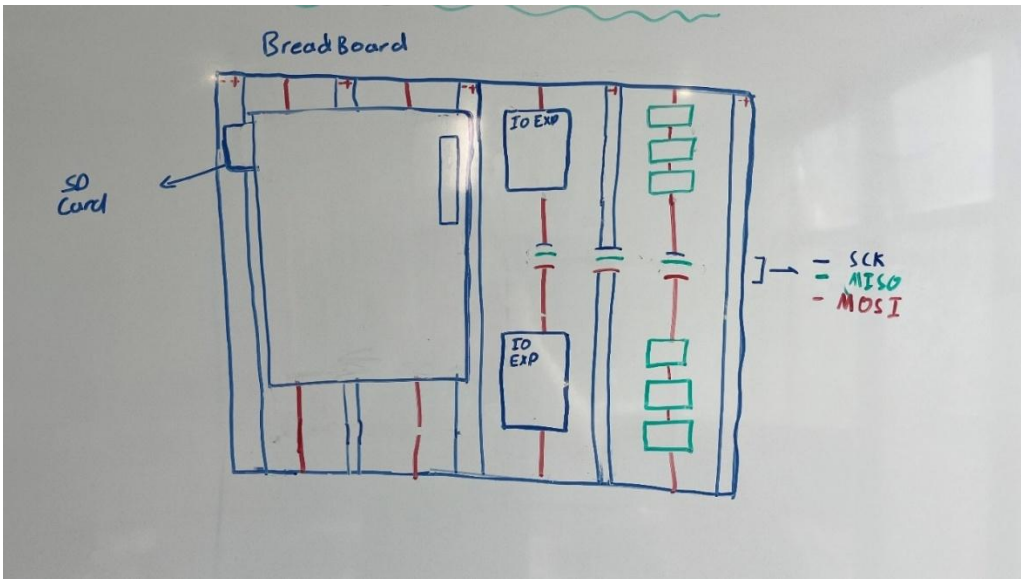
pyb.LED.on(X), pyb.LED.off(X) --> Turns desired led on and off on the STM board

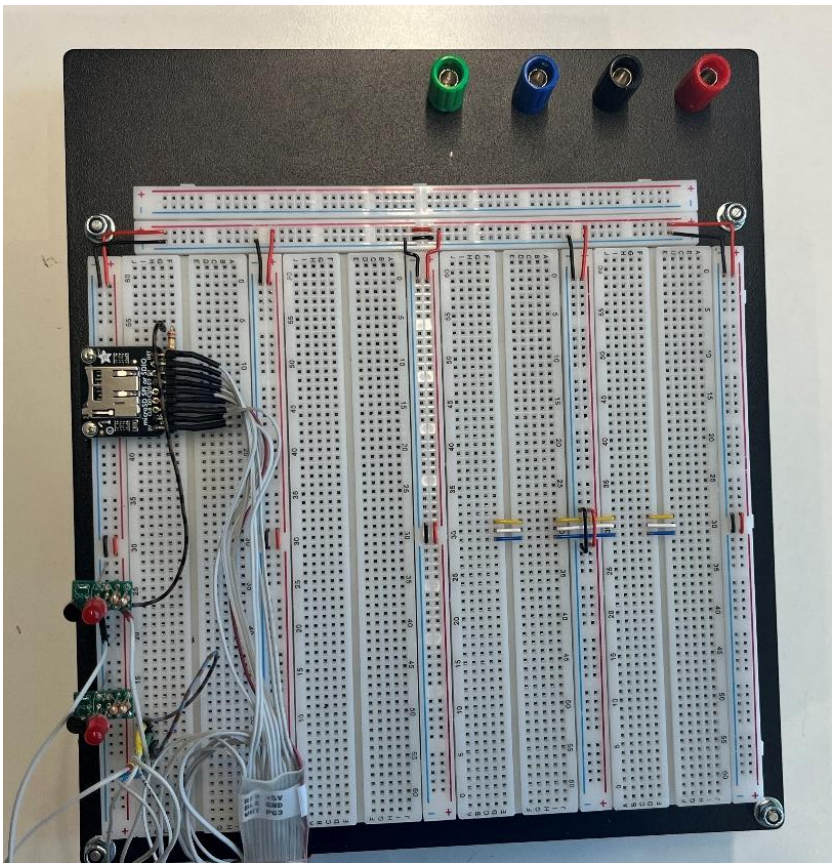
This code is implemented to verify the functionality of the STM Nucleo board.



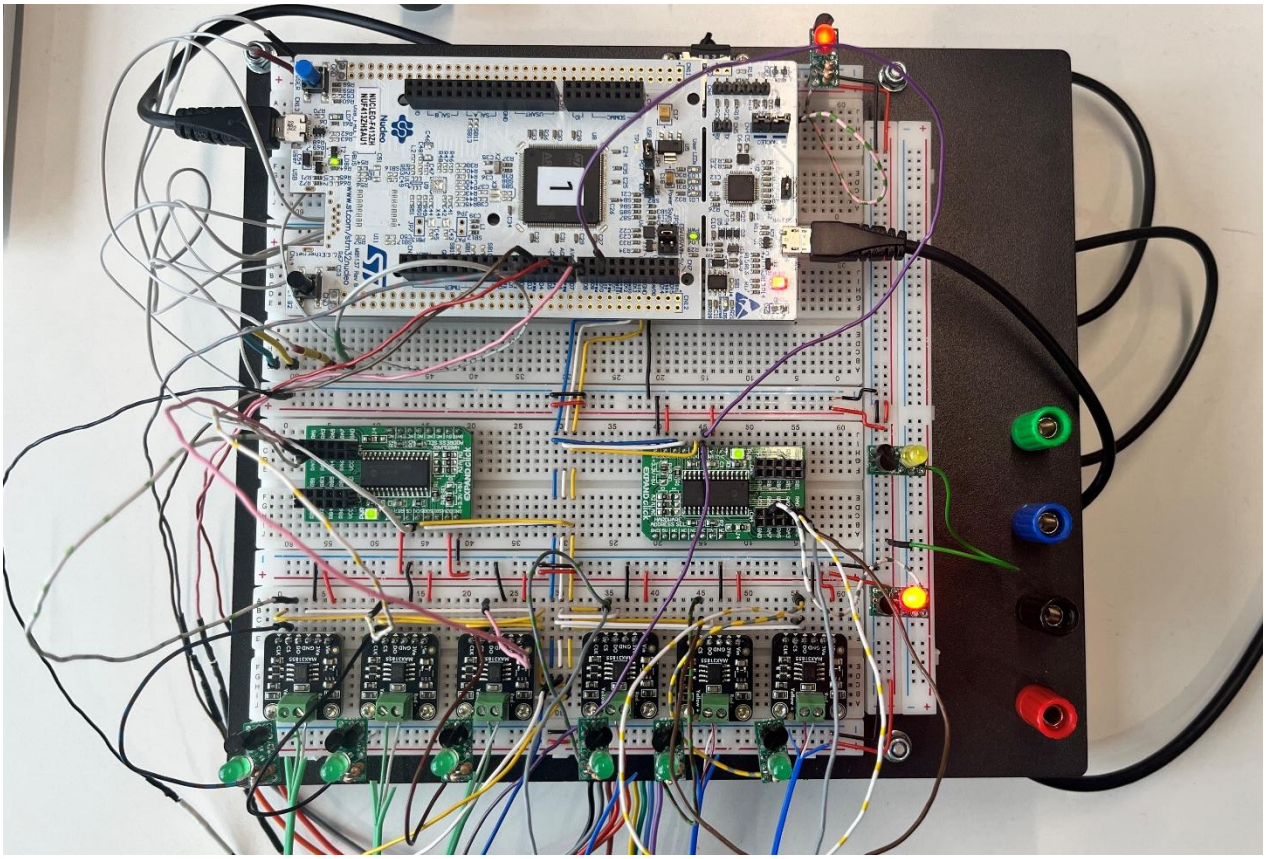
The planned breadboard layout is shown below, where the STM board is placed on the left of the board, then 2 IO expander (MIKROE-951) board are positioned as seen in the figure. Then 3 thermocouples are going to be connected to each IO expander board. This layout will allow us to test how chip select will work with the IO expanders and with each of the thermocouples.

Below shows an image of the planned breadboard layout, with developments being conducted in the image below it.





Below showcases a fully wired board with 2 I/O Expanders and 6 Thermocouples. Readings from all thermocouples are able to be obtained and work as expected.



Code links that provides guides on micro python:

machine — functions related to the hardware — MicroPython latest documentation--> “import machine” codes

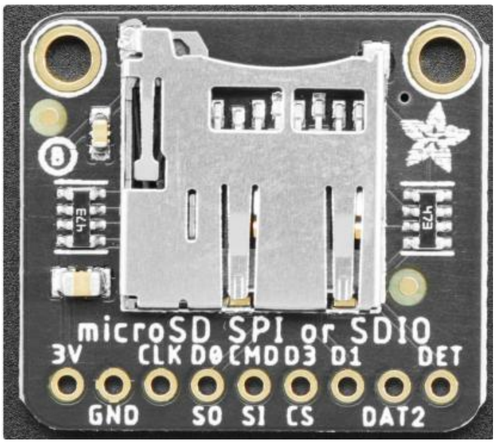
Mastering MicroPython with IO Expanders — pythontutorials.net--> python tutorials

Mastering MicroPython with IO Expanders — pythontutorials.net – SPI control with IO expander

Documentation of MCU board: NUCLEO-F413ZH | Product - STMicroelectronics

2. Micro SD board

Has SDIO and SPI controls, SDIO is 200% faster than SPI controls. SDIO has 4 bidirectional pins for communication between the microcontroller and the SD card. (This allows for bulk data transfer,



and thus faster data transfer).

Documentation: adafruit-microsd-spi-sdio.pdf

Details regarding what pins do are available on the pdf.

3. Risk Assessment

Analysed the possible risks throughout the project by first identifying the task, then the possible hazards involved, before giving a likelihood and consequence score to each one. Then after considering ways to decrease the risk another evaluation occurs.

An example of the risk assessment is shown below:

PROJECT	WDT UP15 Bench Terminator 551PE	OWNER	J Jury	APPROVER	D White
OPERATION	Wire Terminating	DATE	24-Nov-23	DATE	

Warning Matrix

	1	2	3	4
4	ALERT	ALARM	ALARM	ALARM
3	ALERT	ALERT	ALARM	ALARM
2	CAUTION	CAUTION	ALERT	ALERT
1			CAUTION	CAUTION

Likelihood (L)
- combination of probablity and frequency
1 - Unlikely - Little or no chance of occurring.
2 - Likely - 25% to 50% chance of occuring.
3- Probable - 50% to 75% chance of occurring
4 - Almost Certain - >75% likelihood to occur.

Severity or Consequence (S)
1 - Minor - First Aid non-recordable injury or No injury, little to no environmental impact, no/minor property damage.
2 - Serious - Medical Treatment/Recordable Injury, minor damage rapid onsite clean-up.
3 - Significant - Serious LT injury, short term damage within facility requiring work suspension to respond.
4 - Severe or Devastating - Permanent disability/fatality, major contamination, property damage & work suspension.

TASK(1) Include Tools, Chemicals etc	HAZARD (2)	AT RISK(3)	INITIAL RISK (4) L	INITIAL RISK (4) S	INITIAL RISK (4) R	WORKPLACE PRECAUTIONS (5) Evidence required that the hiearchy of control has been used.	RESIDUAL RISK (6) L	RESIDUAL RISK (6) S	RESIDUAL RISK (6) R
Solder	Fumes	**People**	**1**	3	X at SEVERITY 3 / LIKELIHOOD 1	Eliminate - No Substitute - No Isolate - No Engineer - Extraction system Admin - Procedure to open windows before soldering PPE - N/A	**1**	3	X at SEVERITY 3 / LIKELIHOOD 1

R (initial and residual risk):

SEVERITY	1	2	3	4
4				
3	**X**			
2				
1				
	LIKELIHOOD			

Risks were then analysed for these tasks:

- Soldering
- Stripping wires
- Electrical Assembly
- Testing electrical equipment

- Setup of testing environment
- Designing PCB
- Heat Shrinking

4. Expander board

MCP23S17 Chip – 16-Bit I/O Expander with Serial Interface

MIKROE – 951 allows for SPI control using the MCP23S17 chip. With 2 expander boards it will allow us to access and read data from 256 thermocouples.

Input voltage can be either 3.3V or 5V, but if 5V input is desired then a joint will be required to be soldered.

MIKROE-951 digikey documentation: MIKROE-951 MikroElektronika | Development Boards, Kits, Programmers | DigiKey

MCP23S17 Chip Datasheet: MCP23S17 | Microchip Technology

5. Linear regulator

Linear regulators are used to smooth out the DC voltage after rectification. Both the I/O expander board and the thermocouple boards have built in linear regulars as the chips require a clean 3.3 volts to operate successfully. This means the board will take the 5V signal and step it down to ensure a smooth and constant 3.3V for both the I/O expander and thermocouple chip.

The MCU board takes in 5V from the laptop and then also uses a voltage regulator to drop down

the voltage to 3.3V and smooths the voltage out and releases excess energy as heat.

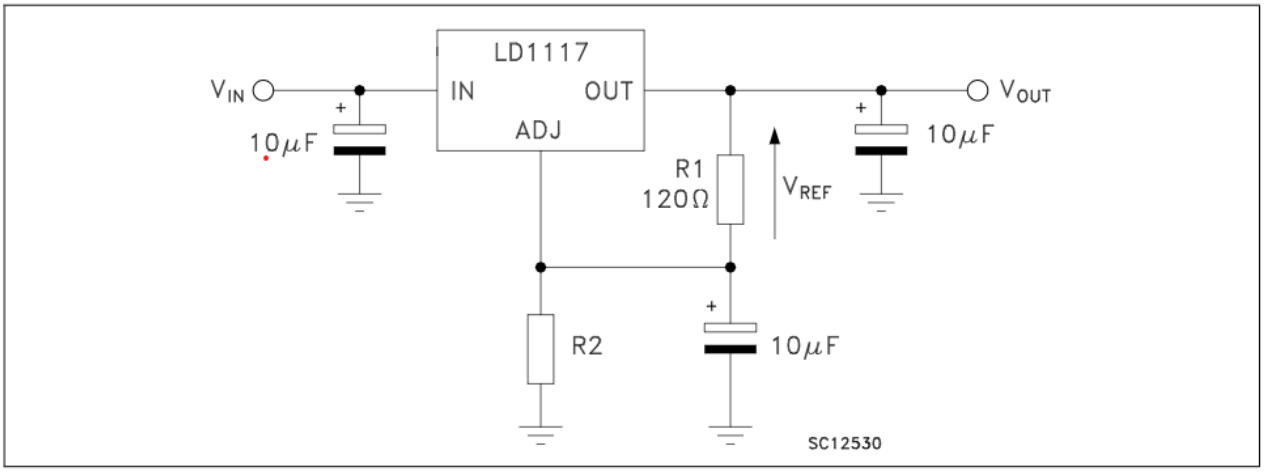
This 3.3V from the microcontroller could be used for the boards, although it would contain too much noise and thus may affect operations due multiple devices operating from the same power supply line. Therefore, it is best to supply the boards with 5V and allow their built-in linear regulators to smooth the voltage.

The linear regulator we chose to use was the LD1117STR.

Below shows a figure of the output voltage application and this was how we based our calculations to allow for an Vin of 5V R2 being 200 ohms to get Vout to be 3.3V.

Equation utilised is VOUT = VREF (1 + R2 / R1).

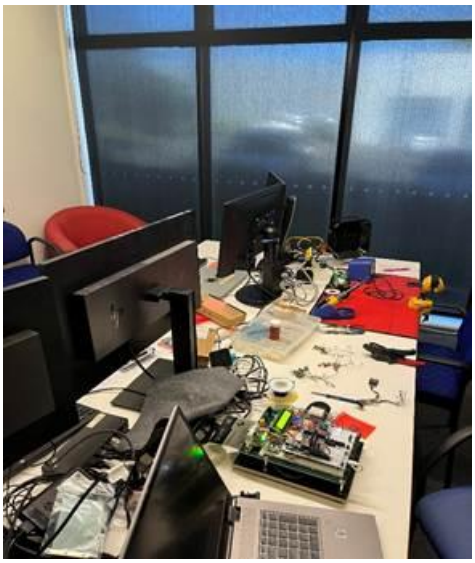
Figure 11. Adjustable output voltage application with improved ripple rejection

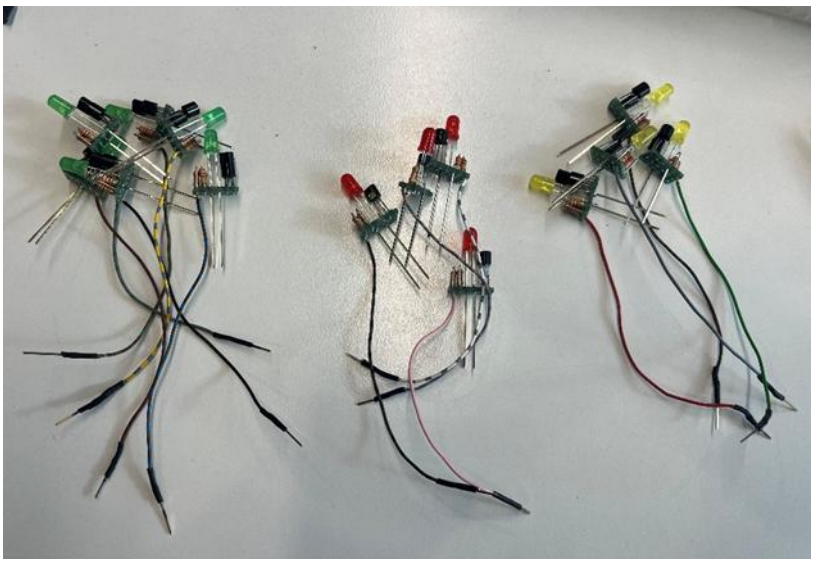


6. Debugging Devices

Image of debugging devices are shown below:

- Red – General debuggers: They are active high.
- Green – Inverted debugging devices for !CS for the thermocouples: They are active low,
- Yellow – Inverted debugging device for !CS for the IO expander board: They are active low.



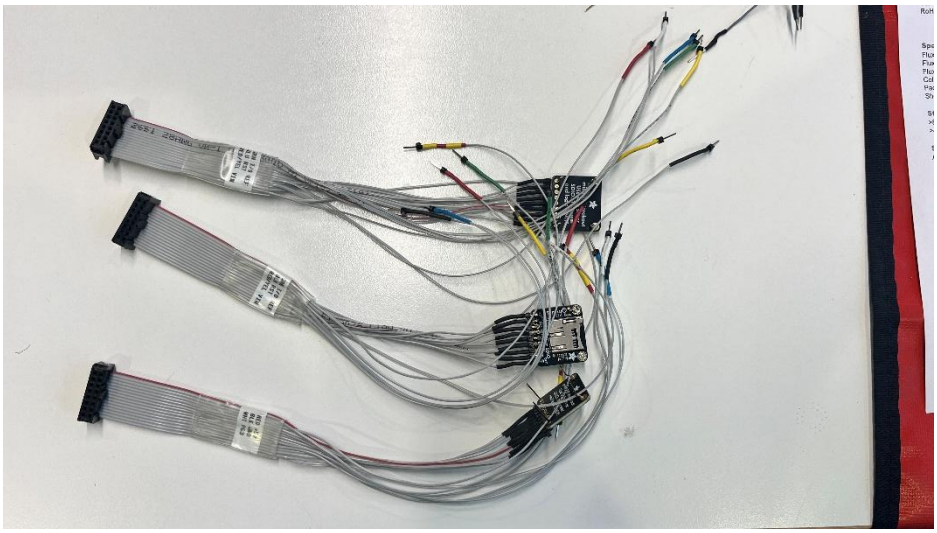


Steps to follow to design the debugging devices are as follows:

1. Solder resistors onto PCB board. 10k ohms for input to base of BJT and depending on which LED is required another resistor is soldered on the collector side. 2 or 3 resistors are required depending on if creating a non-inverting or inverting LED debugging device.
2. Solder BJT onto the PCB board. (2 BJT's are required for non
3. Solder LED onto the PCB board.
4. Solder 2 long pins for Vcc and GND connections and a jumper wire for the input of the base to BJT.
5. Test the components.

7. SD card connectors

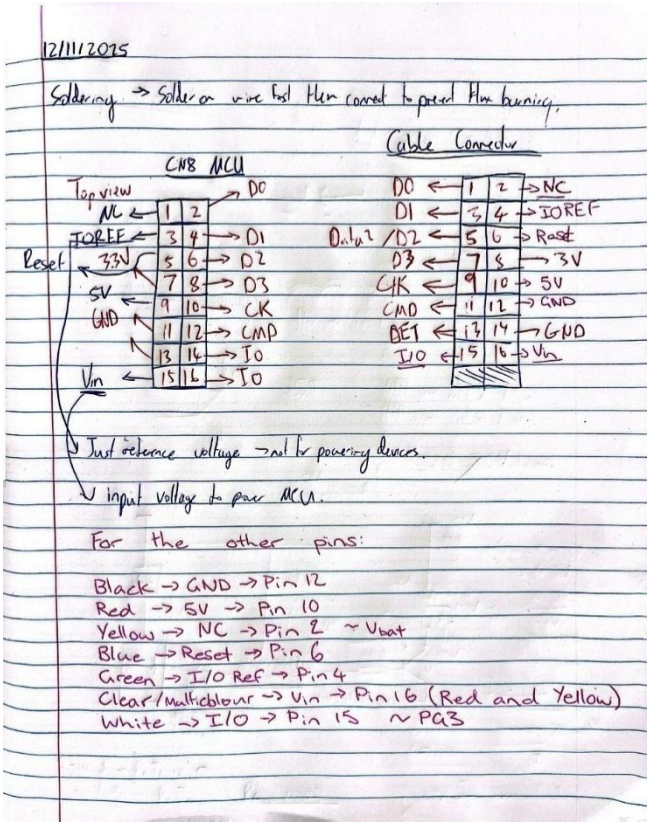
The SD card connectors are shown below, these connectors will allow us to write and upload code via a SD card. This is important since the internal flash on the MCU only has a limited amount of "writes" approximately 10000.



Pins were soldered onto the breakout board and then a 16-pin insulation-displacement contact was crimped onto the other end.

Firstly, for each connector the pins were identified into either going to be connected onto the breakout board or was an extra that we could potentially use later on.

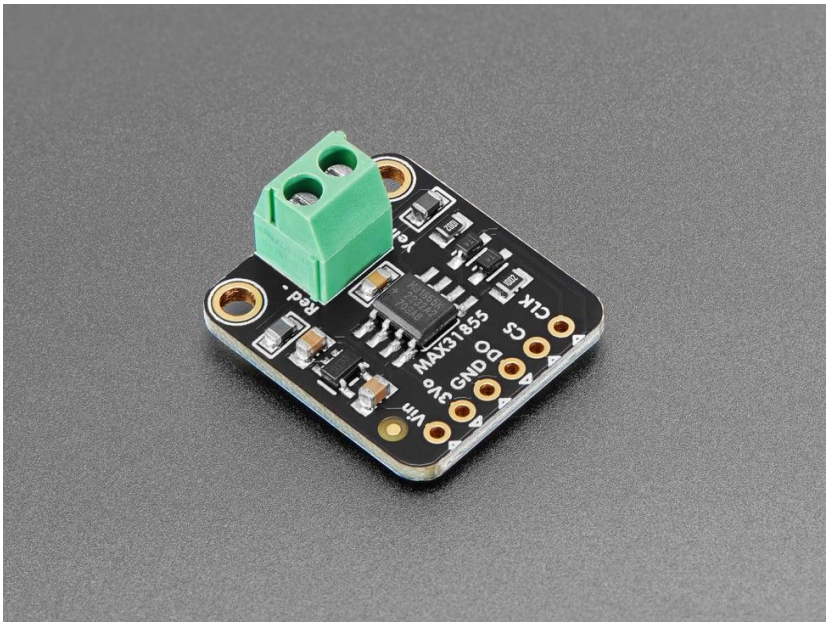
The pin identification for the cable/pin connection of CN8 section of the MCU and the cable connector can be seen below:



As the IDC connector is placed on the bottom of the microcontroller instead of the top, the pin connections will be mirrored. Therefore pin 2 (D0) on the MCU (left image) is connected to pin 1 (D0) on the IDC connector (right image).

8. Thermocouple boards – MAX31855

The thermocouple board require a 5V input, the 5V is stepped down to 3.3V via a linear regulator since the maximum input voltage for the MAX31855 chip is 3.6V. This will ensure that the voltage supplied to the chip is sufficient for smooth and continuous operation.



The pins were soldered onto the max chip and then connected to the breadboard with jumper wires connecting to the corresponding SPI pins in the microcontroller. Once temperatures were being successfully read, two more max chips were soldered and connected to the bread board using more permanent wiring. The chip select inputs are still using jumper wires so they can be altered and tested accordingly.

Thermocouple Amplifier MAX31855 breakout board (MAX6675 upgrade) : ID 269 : Adafruit Industries, Unique & fun DIY electronics and kits

9. Coding

Below illustrates code to initialise the SPI bus on the MCU as well as the chip select pins. Chip select pins are just general IO pins that can be pulled high or low.

```
""" SPI and Chip seclect initalisation """

spi_signal = pyb.SPI(1)
spi_signal.init(pyb.SPI.CONTROLLER, baudrate = 1000000, polarity = 0, phase = 0, bits = 8, firstbit = pyb.SPI.MSB)


cs1 = pyb.Pin('PD14', pyb.Pin.OUT)
cs2 = pyb.Pin('PF12', pyb.Pin.OUT)
cs3 = pyb.Pin('PG1', pyb.Pin.OUT)

print("SPI initilization")
```

To read data from the thermocouple sensor, the chip select (CS) pin for the desired thermocouple must first be pulled low to initiate communication. Then once the data transfer is complete, the CS pin is pulled high again to signal the end of the transaction and acknowledge that the data bytes has been received.

```
def main():

    while(1):

        #print("while loop active")
        time.sleep_ms(500)
        cs1.high()
        cs3.high()
        read_thermocouple(cs2)




""" Function for reading the SPI data signal from the thermocouple."""
def read_thermocouple(cs):
        cs.low()
        data = spi_signal.recv(4)
        cs.high()

        print(data)
        convert_temp(data)
```

To convert the raw data into Celsius for both the reference temperature and the probe temperature, the following steps are performed:

1. Convert the received bytes into an integer.
2. Extract the thermocouple probe temperature:
    - The probe data occupies bits 31:18.
    - Shift the integer right by 18 bits to isolate this field.
3. Extract the reference temperature:
    - The reference data occupies bits [15:4].

- Shift the integer right by 4 bits, then apply a mask to keep only bits [15:4].

4. Apply resolution factors:
    - Multiply the probe value by 0.25 °C per unit.
    - Multiply the reference value by 0.0625 °C per unit.

These scaling factors come from the sensor's datasheet and represent the resolution of each measurement.

```
""" Function for converting thermocouple data reading temps to celcius and prints the data outt."""
def convert_temp(data):
      value = int.from_bytes(data, 'big')
      print(value)

      #Shifting value by 18 since thermocouple data is only bits [31:18]
      tc_data = value >> 18

      #Bit masking for the reference temperature data since [15:4] is the reference temp data
      ref_data = (value >> 4) & 0xFFF

      #print(ref_data, "Ref data")
      temp_c = tc_data * 0.25

      temp_c_ref = ref_data * 0.0625
      print(f'Temperature in celcius: {temp_c}')
      print(f'Ref Temperature in celcius: {temp_c_ref}\n')
```

Interrupts:

To ensure that the microcontroller still has time and properly perform all the tasks that are required of it, the temperature readings should be coded on timers instead of a constant while loop with delays. This way the readings will occur on an ISR and not keep the microcontroller busy when it is needed to perform other tasks in the meantime. This also allows the information from the interrupted task to be stored on the stack so no information is lost, and it can be continued after the readings.

Below shows the readings of each the green thermocouples for each MAX chip.

	MAX Chip 1	MAX Chip 2	MAX Chip 3
Thermocouple 1 – 1 band	Thermocouple temp: **24.0** Ref temp: **25.3125**	Thermocouple temp: **24.75** Ref temp: **25.375**	Thermocouple temp: **24.0** Ref temp: **25.375**
Thermocouple 2	Thermocouple	Thermocouple temp: **24.5**	Thermocouple temp:

– 2 band	temp: 24.0 Ref temp: 25.25	Ref temp: 25.25	24.0 Ref temp: 25.3125
Thermocouple 3 – 3 band	Thermocouple temp: 24.75 Ref temp: 25.375	Thermocouple temp: 23.5 Ref temp: 24.875	Thermocouple temp: 24.25 Ref temp: 24.875

Below shows the readings of each the blue thermocouples for each MAX chip.

	MAX Chip 1	MAX Chip 2	MAX Chip 3
Thermocouple 1 – 1 band	Thermocouple temp: 24.5 Ref temp: 25.375	Thermocouple temp: 23.75 Ref temp: 25.375	Thermocouple temp: 24.5 Ref temp: 25.5
Thermocouple 2 – 2 band	Thermocouple temp: 24.5 Ref temp: 25.4375	Thermocouple temp: 24.5 Ref temp: 25.3125	Thermocouple temp: 23.5 Ref temp: 25.375
Thermocouple 3 – 3 band	Thermocouple temp: 23.5 Ref temp: 25.0	Thermocouple temp: 24.0 Ref temp: 25.0625	Thermocouple temp: 23.75 Ref temp: 24.875

Classes were created for the IO expander and the thermocouple chip to improve code organisation, readability and to encapsulate related functionality with one another.

The I/O expander class includes an initialisation function and write register functions to write values to ports.

```python
# Class for the I/O expander
class MCP23S17:
    # Addressing for IO expander
    # Using IOCON.BANK = 0 (Default)

    IOCON   = 0x05  #Address of the configuration register
    IODIRA  = 0x00  #Address of the register that defines the direction of the port bits for port A
    IODIRB  = 0x01  #Address of the register that defines the direction of the port bits for port B
    GPIOA   = 0x12  #Address of the register that controls the GPIO pins for port A
    GPIOB   = 0x13  #Address of the register that controls the GPIO pins for port B
    OLATA   = 0x14  #Address of the register that controls the output latch for port A
    OLATB   = 0x15  #Address of the register that controls the output latch for port B

    WRITE_OPCODE    = 0x40          #Write opcode
    READ_OPCODE     = 0x41          #Read opcode
    write_buffer    = bytearray(3)  #Creates a three byte array for writing
    read_buffer     = bytearray(3)  #Creates a three byte array for reading

    #0x00 defines IODIRX as all outputs and 0xFF defines IODIRX as all inputs
    IODIR_OUTPUT_CONFIG = 0x00  #IODIRX Configuration
    ALL_PINS_HIGH       = 0xFF #All pins are high


    def __init__(self, cs_pin_name, spi_bus):
        self.spi_bus = spi_bus  #Spi bus creation
        self.cs = machine.Pin(cs_pin_name, machine.Pin.OUT) #Sets own chip select pin register for the I/O expander to be an output
        self.cs.high()  #Pulls up CS to ensure no communication is active

        self.write_register(self.IODIRA, self.IODIR_OUTPUT_CONFIG)  #Writing port A as outputs
        self.write_register(self.IODIRB, self.IODIR_OUTPUT_CONFIG)  #Writing port B as outputs
        self.write_register(self.OLATA, self.ALL_PINS_HIGH)         #Sets all pins high for the OLATA port


    def write_register(self, reg, value):
        # Function to write a value to a register, 0x40 is the write opcode, reg is the desired register to write to, and value is the value to write to the register
        self.cs.low()   #Pulls down chip select signal for 1st IO expander to initiate communication

        self.write_buffer[0] = self.WRITE_OPCODE    #Writes OPCODE
        self.write_buffer[1] = reg                  #Writes desired register value to write to
        self.write_buffer[2] = value                #Writes desired value to register
        self.spi_bus.write(self.write_buffer)       #Writes 3 bytes on the SPI bus
        self.cs.high()                              #Pulls up chip select signal for 1st IO expander to end communication

    def write_OLATA(self, value):
        self.write_register(self.OLATA, value)      #Writes value to OLATA register

    def write_OLATB(self, value):
        self.write_register(self.OLATB, value)      #Writes value to OLATB register
```

For the MAX31855 chip, the class provides an initialisation function, a “read_thermocouple” function to read thermocouple values, a “convert_temp” function to convert raw readings to temperatures, and a “fill_array_tc_probe” function that fills the temperature buffer and calculates the average using a running sum algorithm.

```python
class MAX31855:

    size = 8                        #Array size variable
    #Initiliase the thermocouple
    def __init__(self, cs_pin, spi_bus):
        self.cs_pin = cs_pin        #Chip select pin for the IO expander
        self.spi_bus = spi_bus      #Spi bus lane
        self.raw_tc_data = 0        #Raw data from the thermocouple
        self.raw_tc_integer = 0     #Raw data from the thermocouple as an integer
        self.tc_data = 0            #Thermocouple probe data in binary
        self.tc_ref_data = 0        #Thermocouple reference data in binary
        self.tc_c = 0               #Thermocouple probe temperature in celcius
        self.cj_c = 0               #Thermocouple reference temperature in celcius
        self.tc_buf = [0] * self.size   #Thermocouple probe temperature array

        self.tc_total = 0 #Thermocouple probe total temperature value
        self.tc_avg = 0 #Thermocouple probe temperature average

        self.counter = 0            #Array counter used to fill temperature array

    def read_thermocouple(self):
        self.raw_tc_data = self.spi_bus.read(4)    #Read the thermocouple value from spi bus

    #Converts temperature of data recived from thermocouple into degrees
    def convert_temp(self):
        self.raw_tc_integer = int.from_bytes(self.raw_tc_data, 'big') #Converts raw thermocouple data from hex using big indiannes to an integer

        self.tc_data = self.raw_tc_integer >> 18        #Shifting value by 18 since thermocouple data is only bits [31:18]
        self.tc_ref_data = (self.raw_tc_integer >> 4) & 0xFFF   #Bit masking for the reference temperature data since [15:4] is the reference temp data

        self.tc_c = self.tc_data >> 2       #Each bit represents 0.25 degrees celcius of probe temperature
        self.cj_c = self.tc_ref_data >> 4   #Each bit represents 0.0625 degrees celcius of reference temperature
        self.fill_array_tc_probe()              #Calls function to fill up thermocouple probe array

    def fill_array_tc_probe(self):
        #When counter is 0 initally populates buffer with first temperature value from probe else just populate one index
        if(self.counter == 0):
            # Fills buffer up with first value obtained from temperature probe
            for i in range(self.size):
                self.tc_buf[i] = self.tc_c
            self.tc_total = self.tc_c * self.size   #Obtains total of tc_buf
            self.counter = self.counter + 1         #Exits initialisation
        else:
            old_value = self.tc_buf[self.counter - 1]               #Obtains "old" value of array
            self.tc_buf[self.counter - 1] = self.tc_c               #Populates index counter - 1 with temperature probe value
            self.tc_total = self.tc_total + (self.tc_c - old_value) #Adds difference between new reading and old reading
            self.tc_avg = self.tc_total >> 3                        #Bit shift by 3 to divide by 8
            self.counter = (self.counter % 8) + 1                   #Increments counter by 1 and resets at counter = 8

    def tc_info(self):
        print(f'Thermocouple temp: {self.tc_c}')    #Prints current probe temp
        print(f'Ref temp: {self.cj_c}\n')   #Prints current ref temp

    def tc_average_info(self):
        print(f'Average thermocouple temperature: {self.tc_avg}')
```

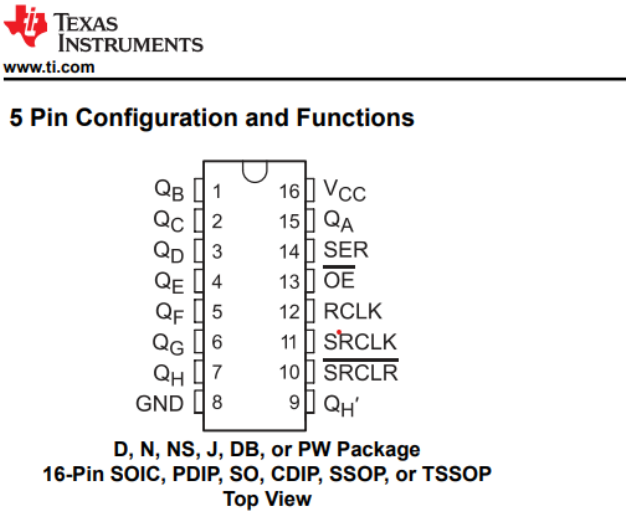
A running sum algorithm was implemented to calculate the average of the thermocouple readings. When the first thermocouple reading is obtained, this value is used to populate the entire buffer array.

The running sum algorithm tracks the previous thermocouple reading, calculates the difference between the new and previous values, updates the total with this difference, and then performs a right bit shift by 3 to compute the average.

Garbage Collection:

There is a built in garbage collection system on the microcontroller that runs automatically when the RAM on the microcontroller is getting full. It can also be run manually by calling the gc.collect() function. This goes through the heap and marks all the variables that are being used before sweeping out anything unnecessary to create more space.

10. Shift Register



QA – QH are the output registers of the shift register.

SER is the serial data input which is connected to the MOSI line.

!OE is output enable pin and is active low.

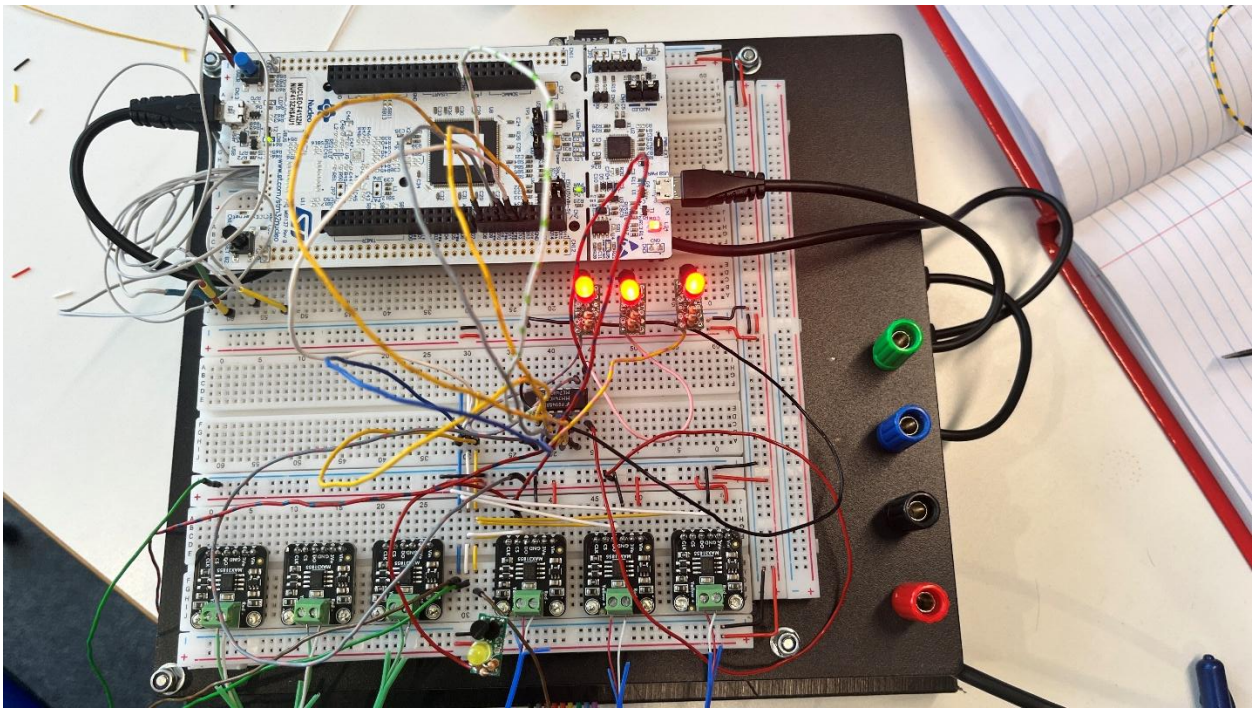
RCLK is the register clock or commonly known as the latch clock – when this is pulsed high then low the contents of the shift register are applied onto the output registers (QA-QH).

SRCLK – is shift register clock line

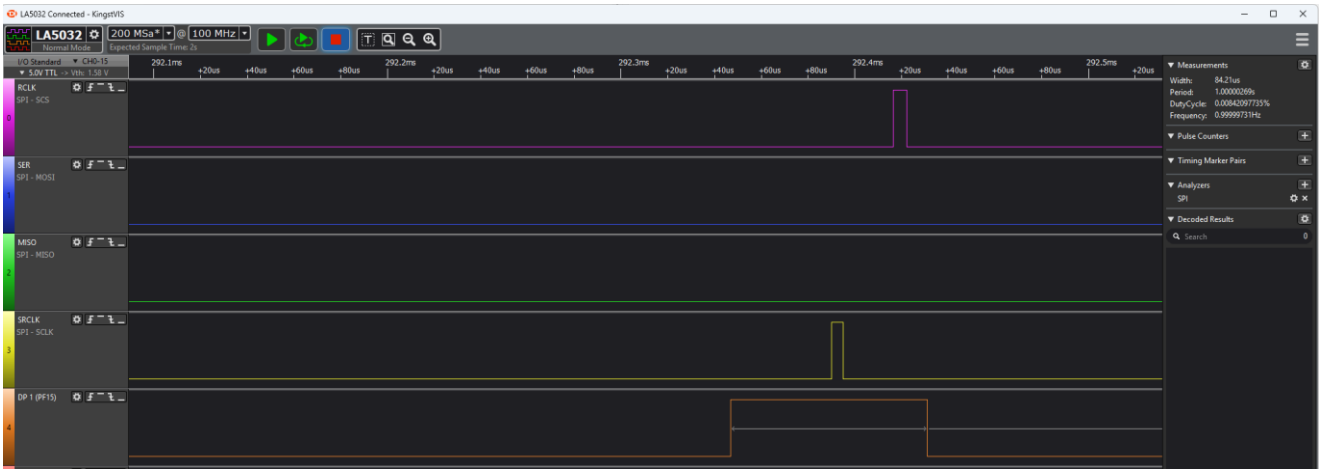
!SRCLR – is shift register clear and clears the contents of the output registers.

QH' allows cascading of multiple shift registers together.

Using this shift register (sn74hc595) we were able to light one LED off and then on at a time. Initial testing was conducted to turn one led off at a time.



Using bit banging, time taken for 1 shift is approximately 84.21us. This was just to shift a 1 throughout the shift register and each shift of that 1 took that long.



Using the shift register and bit-banging of the shift registers the time it took to read a thermocouple value i.e. send a 4 byte pattern over the MISO line is ~100us which is 3x faster than the time compared to using the I/O expander to read the data from the thermocouple.

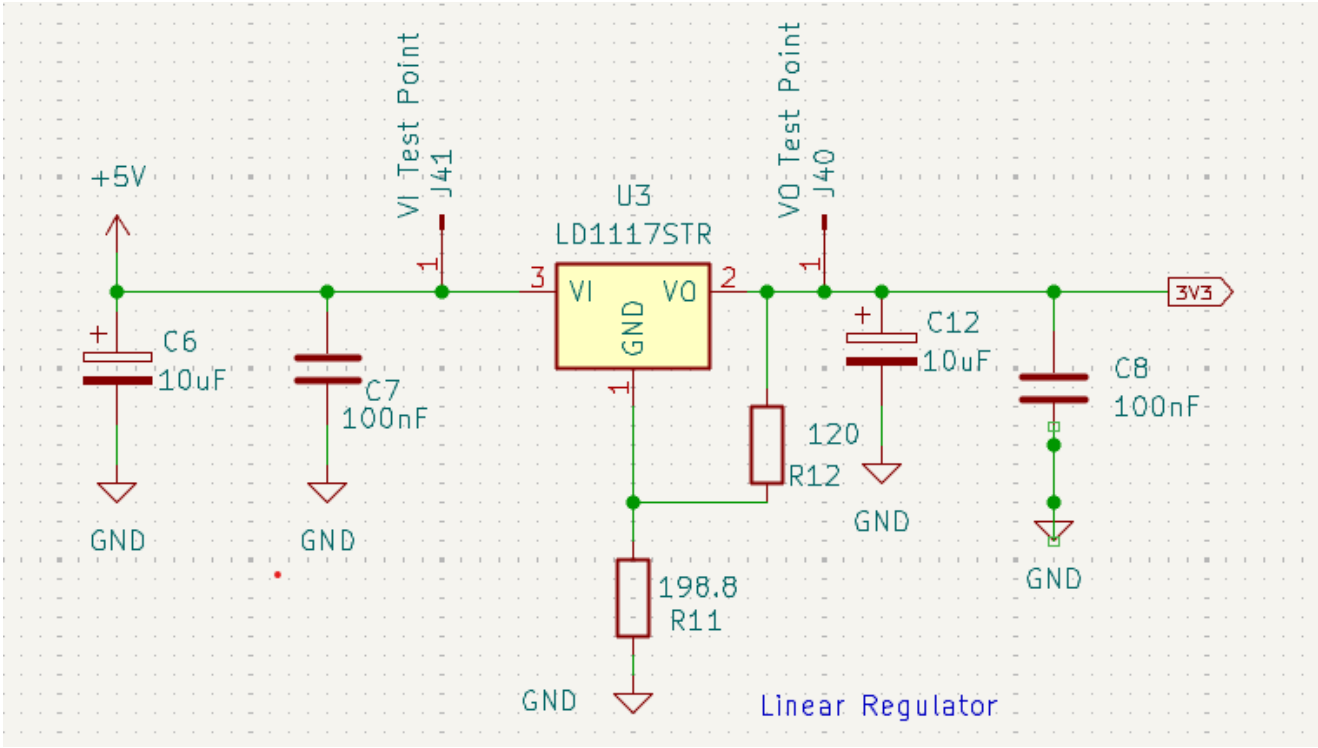
Code used as reference for the shift register can be viewed in this link: [micropython-74hc595/src/sr74hc595_spi.py at master • mcauser/micropython-74hc595 • GitHub]

11. Schematic

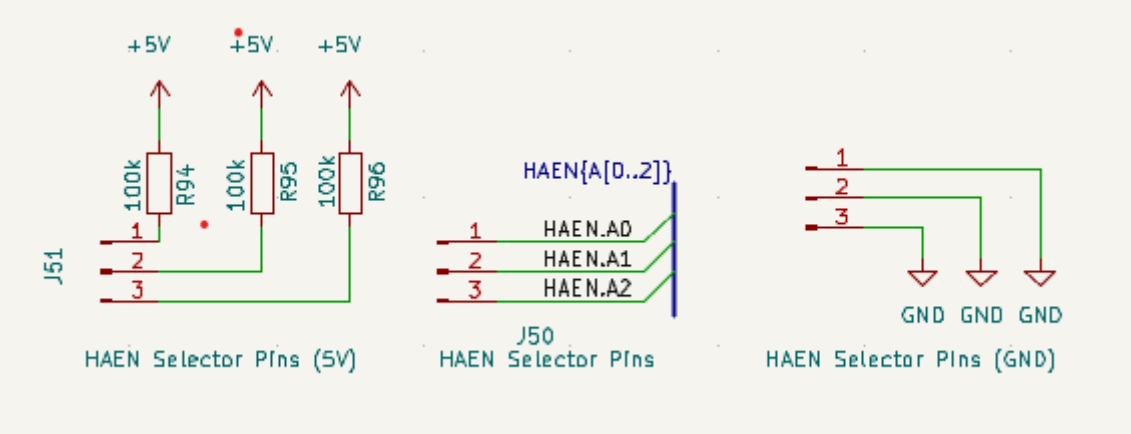
A schematic outlining the function of a PCB for the thermocouple sensors was created in KiCad based off the testing done with the breadboard mock up and our knowledge of the devices being used based off of the datasheets and other relevant documents. Each section of the schematic has a specific function and work together to unsure the temperatures measured by the thermocouples are accurately and effectively moved into the microcontroller and SD card so they can be used later with the visualisation software.

Linear Regulator:

To ensure a constant supply of 3.3 volts is being supplied to each of the max chips, a linear regulator LD1117STR was included. This can be seen in the snippet of the schematic included below. There are two capacitors and a test point on both the input and output side of the linear regulator. The 10uF capacitors are for energy storage to ensure a smooth voltage is being fed through. The 100nF capacitors are filtering out high frequency noise signals and decoupling them from the circuit. And the test points allow us to probe to help debug potential problems with the PCB when it arrives. The resistor values R11 and R12 were calculated to ensure a constant output voltage of 3.3 volts.

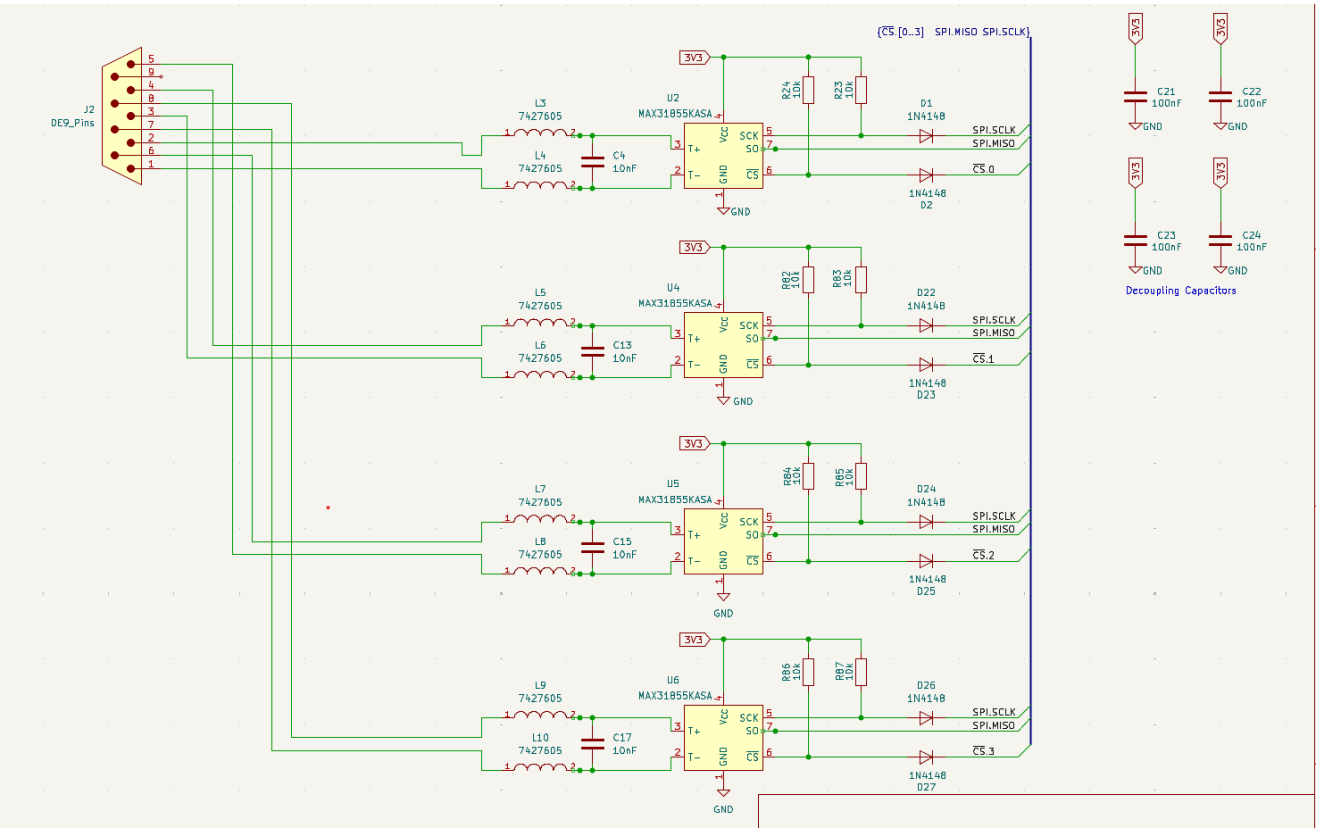


To enable HAEN for the IO expander, selector pins are utilised to enable 8 I/O expanders to be utilised on 1 chip select pin. They are initially not biased and will be needed to have a selector pin to either set the HAEN pins to high or low.



There are four thermocouple temperature sensors connected by wires 2-3 meter long wires to the other end of the 9 pin D sub connector. These will measure a voltage which is then sent through the D sub to the two ends of the max chip and a temperature value can be determined from here. The circuitry around the max chip is largely similar to that on the Adafruit break out board that was used on the breadboard during testing. The capacitors and inductors at the input of the max chip are to filter out any unwanted noise from the signal to ensure an accurate temperature value. The two resistors on SCK and CS line are for biasing the pins high. (No diodes anymore, change

picture).



Layout rules: 2.54mm grid/1.27mm grid

Removed outer ring of the silkscreen for diodes as the diodes are actually only ~2mm in width but silkscreen was ~2.6mm.

Changed inductor to beads so inductor takes up less space on the PCB board

Added line drivers to MOSI, $\overline{CS}$, SRCLK, RCLK, $\overline{SRCLR}$, $\overline{RESET}$, $\overline{OE}$.

Transistor layout for LED design was replaced with line drivers to reduce amount of components needed to solder.

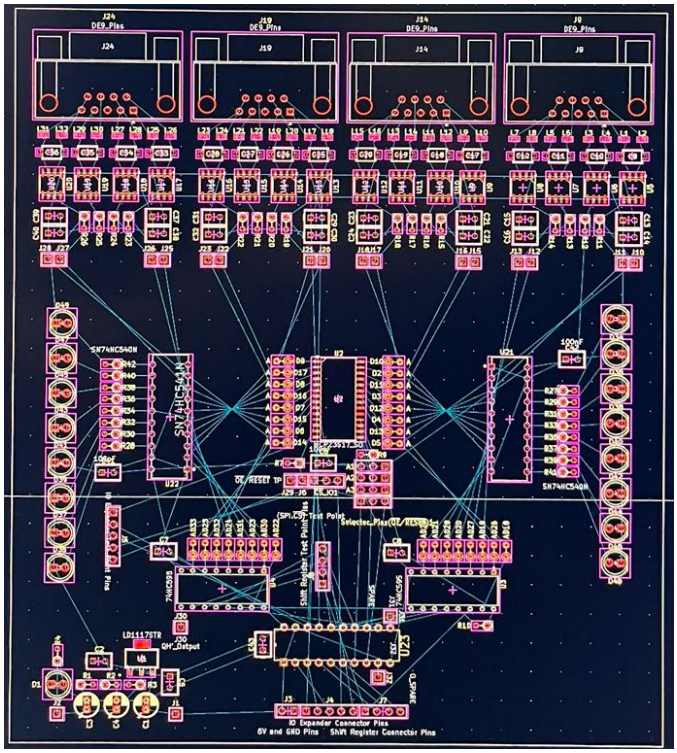
Made new footprint from existing KiCAD footprints to allow the style of the footprint and how it is laid out to be changed.

Added resistors to MISO line for all thermocouple chips as a problem was discovered that if two thermocouples would want to drive MISO line at same time a short circuit may occur.

12. PCB Design

Layout:

We needed to make sure that we could make the PCB as small as possible to save on cost but also not so compact that it wouldn't be possible to solder all the parts onto it. Especially the surface mount components. They are a lot smaller and had the potential to be difficult to solder correctly so we gave both the max chips and the ferrite beads enough clearance. We also knew we were going to solder them first so there would be more room around them.

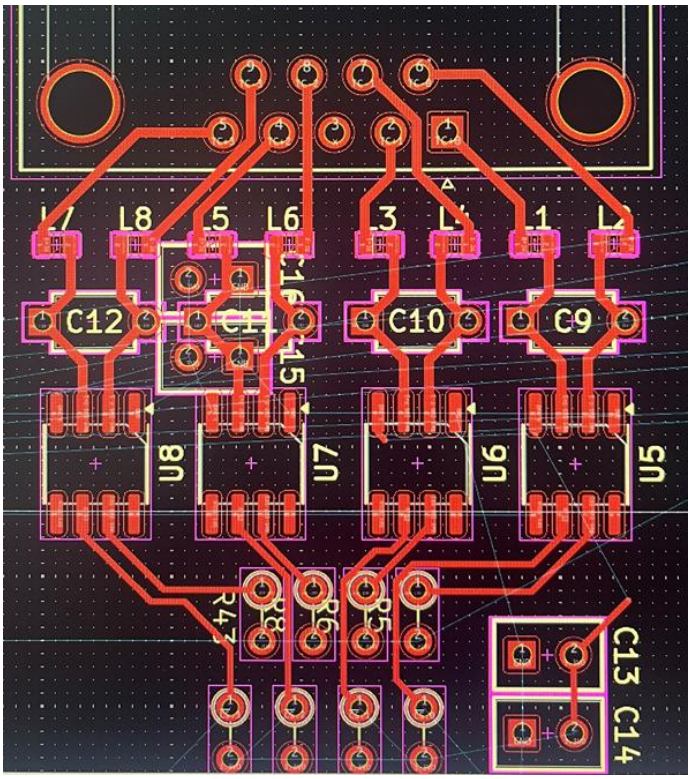


This was our first draft of placing the components, things changed a little bit after we started laying traces but it gives a general idea. For each of the D-Subs, all the max chips were placed next to them and all the relevant circuitry is there. The decoupling caps were next to all the main chips, the I/O expander was in the center, with two shift registers either side just below it. The line drivers for the LED's were on either side at this stage, and the linear regulator and all the power circuitry was down the bottom in its own section.

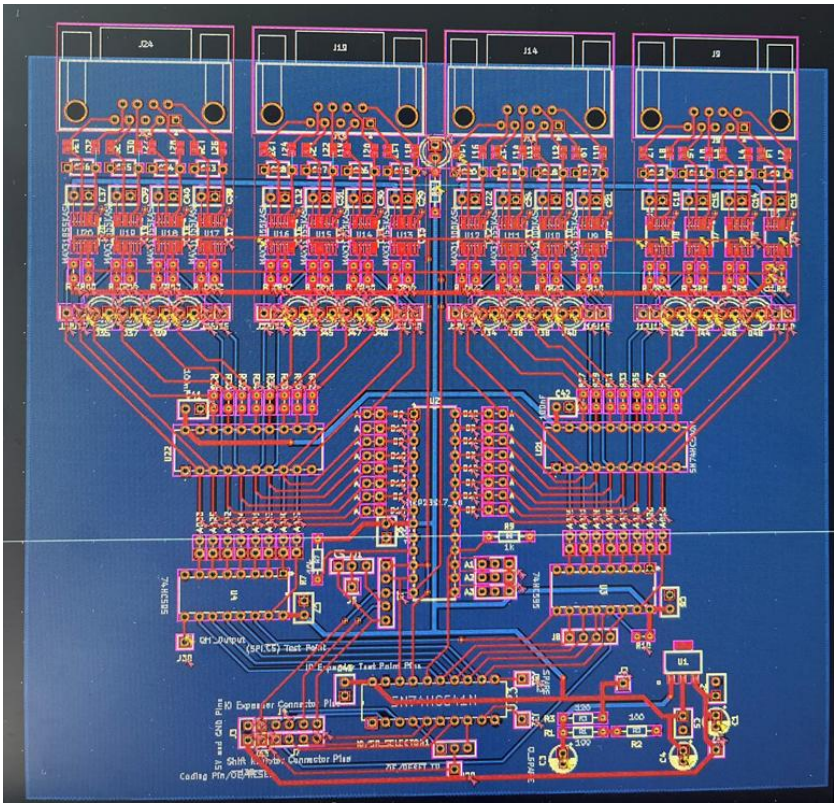
We tried to keep things symmetrical just to keep it easier to understand the layout. We had quite a few learnings from working with KiCad, mostly about grid size and placing components on the grid. The grid was initially on 2.54mm spacing to keep everything consistent with standard measurements. However, the D-Sub pins were on a different grid size so we had to make a custom grid and switch between them when placing things.

Traces:

We traced the D-Sub pin areas first, connecting all relevant parts together in a first draft that looked like this.



This was then continued throughout all the D-sub section and the rest of the PCB was traced into first draft with a flooded ground plane that can be seen below.

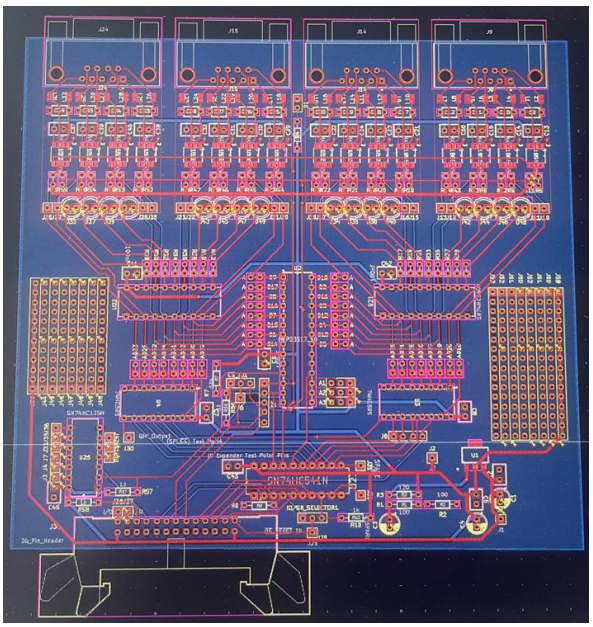


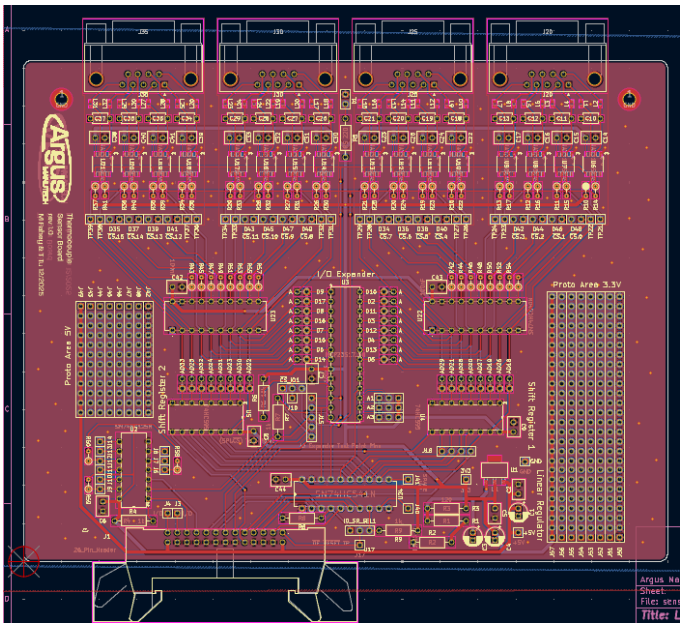
Finishing Touches…

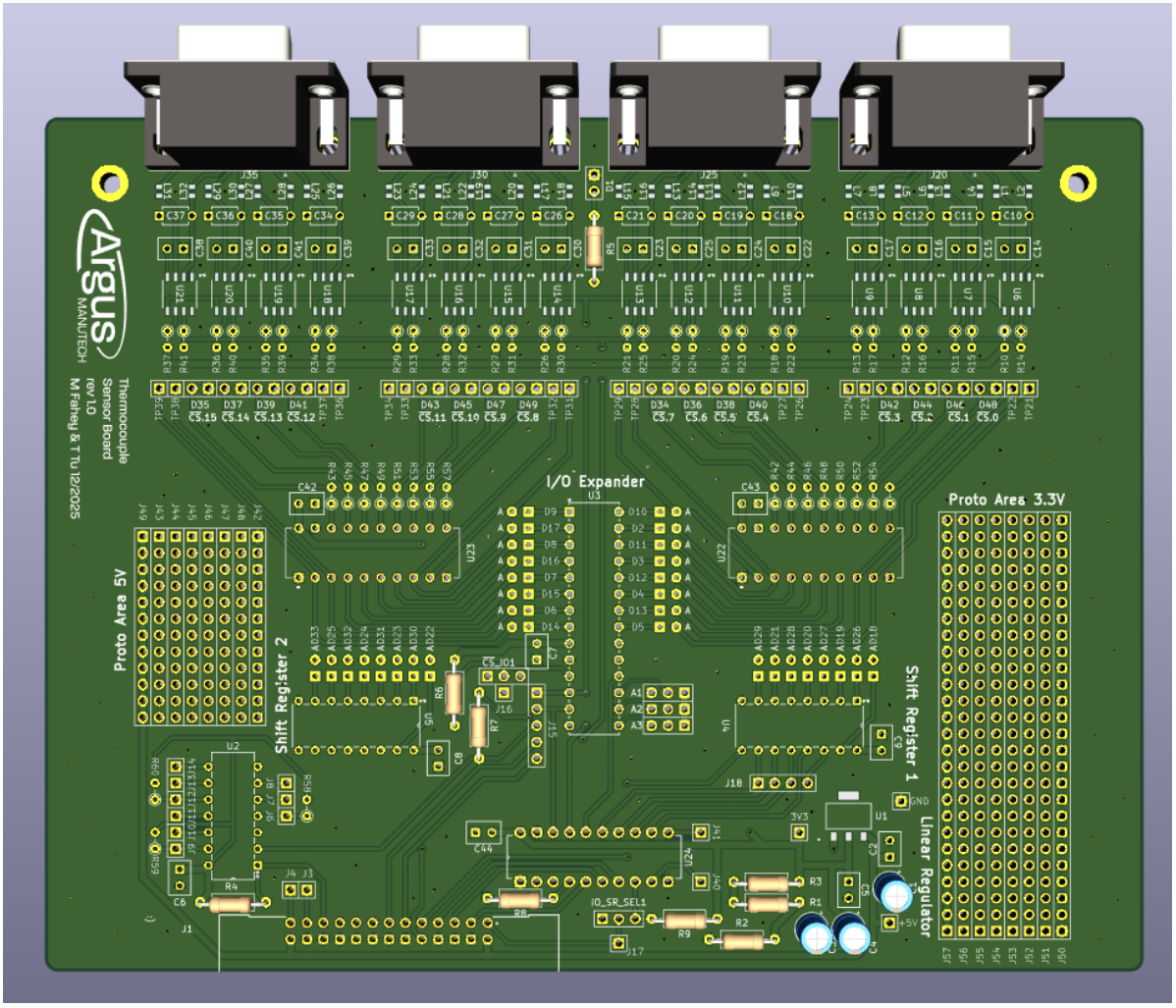
Below is a list of all the updates that happened to the PCB in the days that followed as well as

some more pictures of the PCB as it developed over the following week.

- Fix all errors found from running the design rule checks.
- Created custom footprints for all the pin headers
- Changed the back connections to a 26 pin right angled IDC connector.
- Added an extra line driver for the SPI MISO line.
- Created both a 3.3V and a 5V prototyping area on either side of the PCB.
- Created a solution for connecting the Q' pins for the shift registers.
- Make the PCB wider in dimension so there is enough clearance on either side to slide into the housing.
- Change all the LED footprints to pin headers.
- Smoothed the edges of the PCB
- Add mounting holes to attach the PCB to the front plate of the housing.
- Extend the prototyping area on the right side.
- Flood both the top and bottom copper plates to fill all the ground connections.
- Increase the number of vias to create connections between the two ground plates.
- Add labels to all the test points and sections of the PCB.
- Add the Argus logo and PCB information to the silkscreen layer.
- Rerun the DRC and fix all the new errors created from the updates.







The PCB was completed and ordered before the Christmas deadline as planned.

13. Bill of Materials

A Bill of Materials was made with all the parts needed for populating the PCB board. This was made originally by exporting the information from KiCad and then reviewing the list to establish which of those parts needed to be purchased. There are also parts needed for the system that weren't on the PCB such as the D-sub and IDC connectors, ribbon cables and housing bolts which were added manually also. A multiplier box was also added to the excel sheet that would multiply parts by however many PCB's needed to be populated. Those numbers were then used for the Digikey order to ensure we have the correct amount of parts.

72	Summary! Yay slay					
73	PCB Mounted Parts					
74	Part	Qty	Part Number	Digikey Number	Total for A	Total to order
75	1 Pin	33			33	
76	2 Pin	17			17	
77	3 Pin	5			5	
78	4 Pin	1			1	
79	5 Pin	1			1	
80	Pin Headers	91(~100)			200?	
81	26 Pin Right Angle Connection Header 2.54mm	1	61202622121	732-2694-ND	5	5
82	10uF Electrolytic Capacitor	3	ESH106M035AC3AA	399-ESH106M035AC3AA-ND	15	15
83	100nF Ceramic Capacitor (0.1uF)	25	C324C104M5U5TA	399-9819-ND	125	125
84	10nF Ceramic Capacitor (10000pF)	16	C317C103K1R5TA7301	399-13924-1-ND	80	80
85	LED's	17			85	
86	1N4148 Diodes	32	1N4148	1655-1N4148CT-ND	160	160
87	9 Pin D-Sub Right Angle Plug Connector	4	DLS1XP5AK40X	626-1953-ND	20	20
88	Ferrite Bead 120 ohm 0603 1LN	32	MH1608-121Y	MH1608-121YCT-ND	160	160
89	100 ? Resistor	2			10	
90	120 ? Resistor	1			5	
91	1K ? Resistor	7			35	
92	220 ? Resistor	17			85	
93	10K ? Resistor	17			85	
94	47 ? Resistor	16			80	
95	Linear Regulator (Surface Mount)	1	LD1117STR	497-1244-1-ND	5	5
96	Buffer, Non-inverting	1	SN74HC125N	296-1572-5-ND	5	5
97	I/O Expander/I2C and SPI version	1	MCP23S17-E/SP	MCP23S17-E/SP-ND	5	5
98	Shift Register	2	SN74HC595N	296-1600-5-ND	10	Do we need to order?
99	Thermocouple Max Chip (Surface Mount)	16	MAX31855KASA+	MAX31855KASA+-ND	80	80
100	Line Driver Inverting	2	SN74HC540N	296-8334-5-ND	10	10
101	Line Driver Non-Inverting	1	SN74HC541N	296-1594-5-ND	5	5
102	14 Pin DIP (Buffer, Noniverting)	1	2485264-2	17-2485264-2-ND	5	5
103	28 Pin DIP (I/O Expander)	1	2485264-7	17-2485264-7-ND	5	5
104	16 Pin DIP (Shift Register)	2	2485264-3	17-2485264-3-ND	10	10
105	20 Pin DIP (Line Drivers)	3	2485264-6	17-2485264-6-ND	15	15
106						
107	Breakaway Headers to Create Custom Pins					
108	Short Breakaway Pin Header (11.39mm)	1	PRPC040SAAN-RC	S1011EC-40-ND		5
109	Medium breakaway pin header (16.17mm)	1	PRPC040SACN-RC	S1131EC-40-ND		5
110	Long Breakaway Pin Header (29.21)	1	PEC36SAHN	S1082E-36-ND		5
111	40+40+36 = 116 should give us good options for making the pin connectors					
112						
113	Connectors and Wiring (To connect to PCB)					
114	Female D-Sub Connector 9 Pin	4	DE09S064TLF	609-1525-ND	20	20
115	26 Pin IDC Connector	1	61202623021	732-2105-ND	6	6
116	26 Line Ribbon Cable (10 Total)	2	3365/26	19-3365/26-DS-ND	10	10
117	D Sub Back Shell	4	40-9709H	116-40-9709H-ND	20	20
118						
119	Note for Ivo: micro usb connectors					

14. CAD model

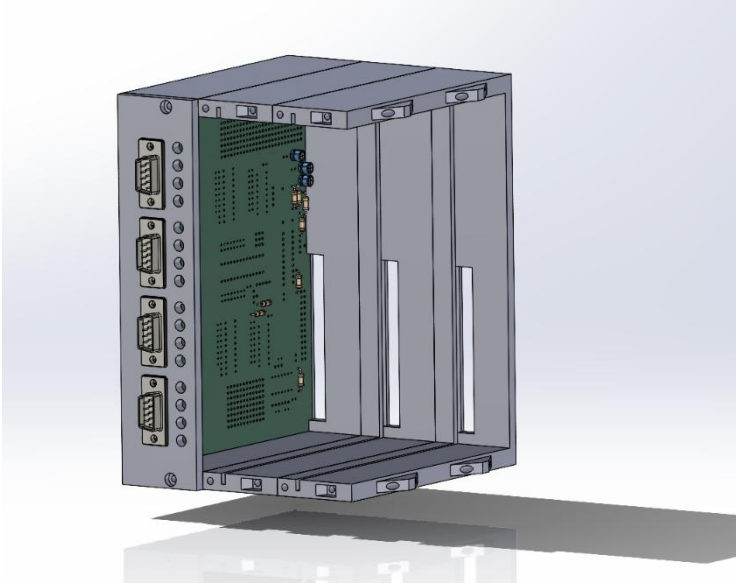
Once the PCB design was well underway, we began to brainstorm ideas for the housing, how we wanted to store the PCBs in a way that would protect them, how we wanted to manufacture the

storage box. Initially there was an idea that there would be a metal box of some description, potentially with card holders inside that the PCBs could slide in and out of. The metal was initially thought of to keep the electrical systems inside the box protected. However, when it came to deciding how big the box would be there were some problems. Due to the variable nature of the project, we were unsure exactly how many PCBs were going to be populated, or if it was to work, and how many boxes they would need on a larger scale. Hence it was decided on a system that was entirely modular and expandable depending on the exact number of PCBs being used. 3D printing with PLA was the best manufacturing option here, for the initial development, as it allowed us to develop a system with exact dimensions required. Rather than wasting more expensive material developing concepts that may be unnecessary.

The housing is made up of two main parts. A modular housing and a front plate. The housing is a u-shaped part printed with tabs that let them interlock and grooves for the PCB to slot into. There is a hole at the back for the 26-pin D-sub to connect to, and along the back of the system will run a ribbon cable connecting all the PCBs together. This will connect them up to the microcontroller and computer for visualisation.

The front plate is where all the thermocouples will attach to the PCBs with 4 D-sub connectors. There is 4 thermocouples on each connector and a corresponding LED light for the chip select of the thermocouple max chip, so we can see exactly which thermocouple is being read from or communicated with. The PCB will be attached to the front plate, so that the LEDs and dubs don't have to be disconnected for the board to be accessed. The idea is that they will move in and out of the housing as one.

Below shows a first design of the CAD model for the PCB housing. The idea is to attach the PCB to the front plate and thus the PCB will be able to be dragged out of the housing alongside the front plate.

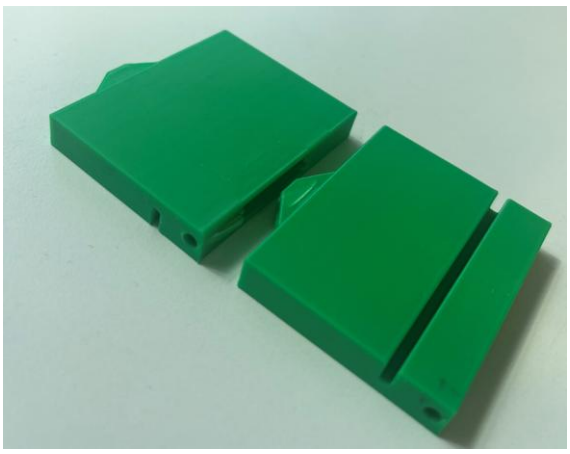


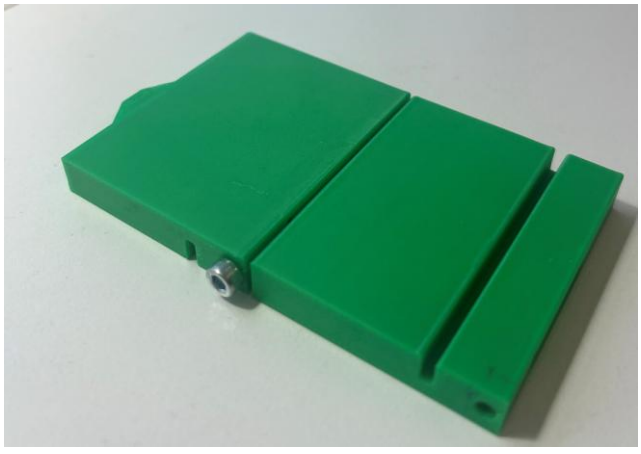
15. Housing Iterations – Before PCB

When designing the housing it was important that it was a modular design that could be easily extended if the scale of the project was to increase, or if they needed to house multiple more PCBs. The design should be able auto-locate into the correct position, with movement in the x, y and z directions constrained, and we wanted to be able to access any of the PCBs inside the housing at any point in time.

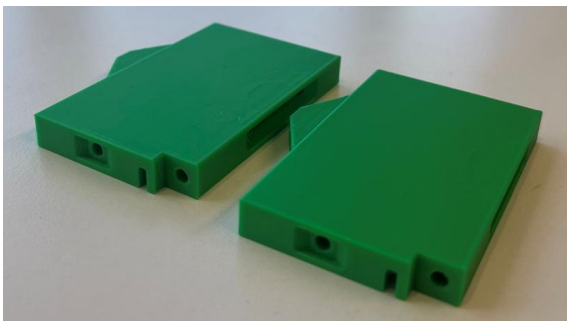
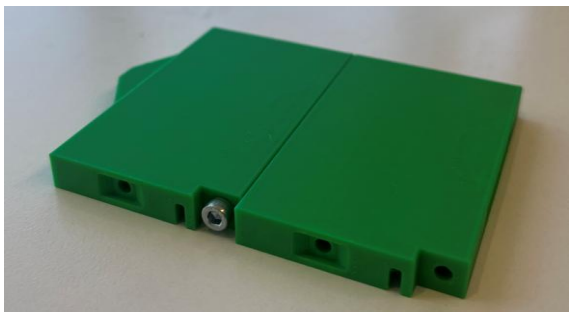
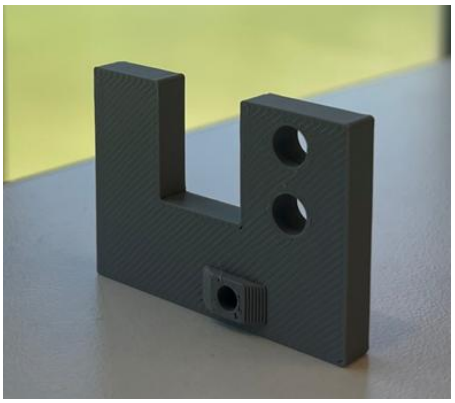
After we were happy with the CAD design and wanted to test the system for both modularity and the front plate, we began test printing using the Bambu Lab 3D printers.

First Iteration - Proved concept successful and sturdy, positioning system worked well however was broken with force. Needed a space for the bolt to sit to create a flat front face.

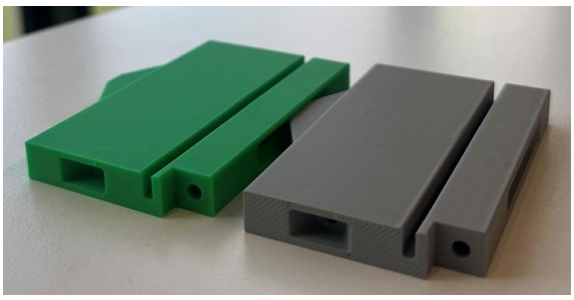
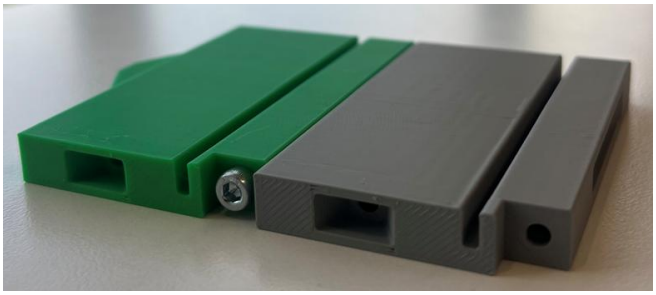
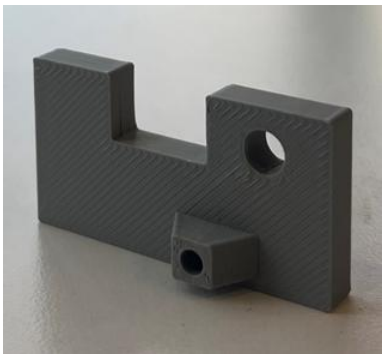




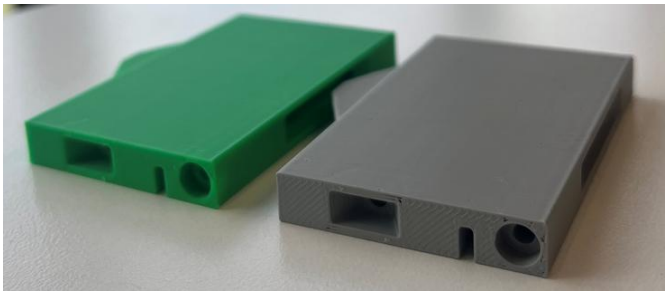
Second Iteration - Modular box tabs were larger to increase strength and had a cut-out space for the bolt to sit. Also introduced a smaller tab system to help position the front plate. Although effective, it needed to be deeper for the positioning to be as stable.

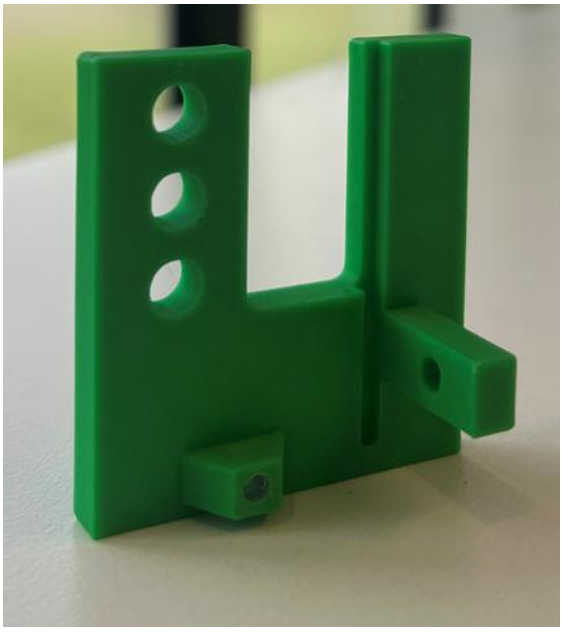




Third Iteration - The tab for positioning the front plate was made deeper for stability which worked well. We decided we wanted to be able to attach the front plate to the PCB so needed to adjust the design slightly.

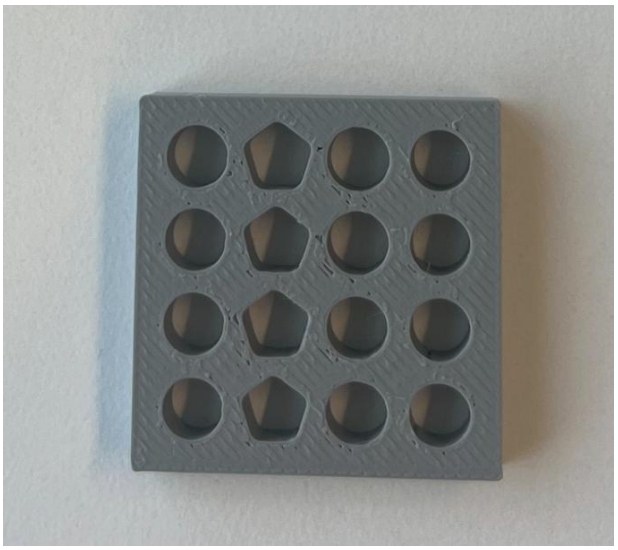




Fourth Iteration - The bolt cut out was made circular for a more seamless effect. A grove for the PCB was made in the front plate, and tabs were added for mounting the PCB to the front plate.





LED Hole Test – We also did a test print to confirm which sized hole would be the best fit for the LED to be held securely without extra support. Many shapes were tried but a simple circle with a diameter of 5.1mm was perfect and was the used in the final print out of the housing.



After all the iterations to confirm the design that would work best the final housing was printed. This can be reevaluated when the PCB arrives and any further adjustments that are necessary can be made.

16. Populating the PCB

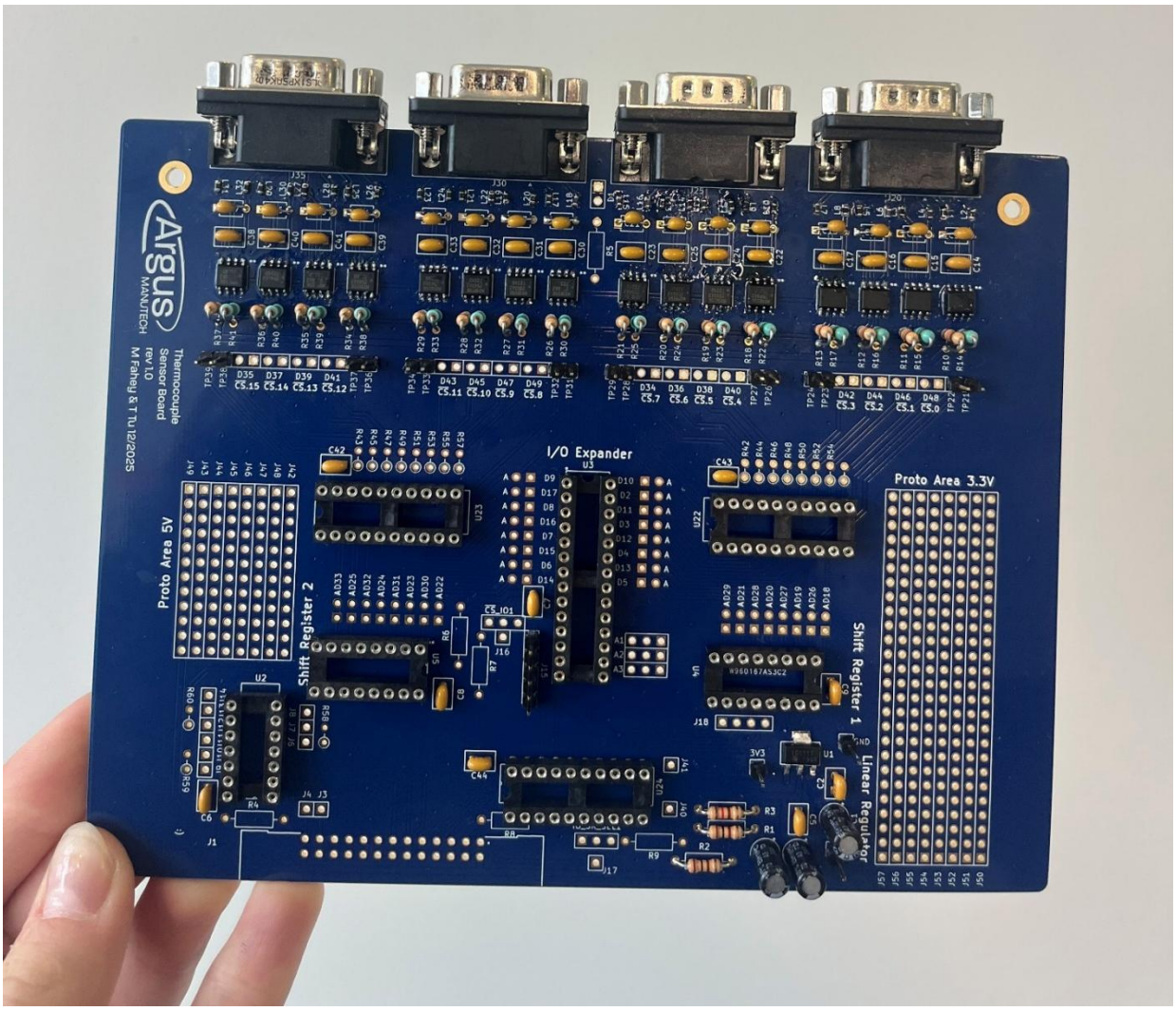
When the PCB arrived we needed to populate it section at a time ensure no faults and made debugging easier. As the max chips are very sensitive to voltage, it was important that the linear regulator was functioning first so that the max chips weren't accidentally being fried if these was a fault in the power circuitry.

Once this was working, the first max chip and its relevant circuitry was populated. After this worked, the next three channels were also populated, however they didn't work as seamlessly as hoped. We knew it was a soldering issue as the circuitry was identical to the first chip which was

working. With the help of James and the 6x microscope (AKA the best thing ever) We realised that there were a lot of faults with the surface mount soldering of both the max chips and the ferrite beads. The microscope allowed us to diagnose these problems easily and see where the solder wasn't fully attached to the pad, so that they could be easily fixed. After the soldering adjustments had been made, all 4 chip worked well.

For the next 12 channels, we wanted to avoid having to fix surface mount soldering when the channel was fully populated because it was much more difficult to solder with other parts in the way. Therefore, All the surface mount soldering was completed and inspected under the 6x microscope before any other parts were added, to ensure these were no issues we couldn't see. This worked great, and all channels worked perfectly on the first test after this system was implemented.

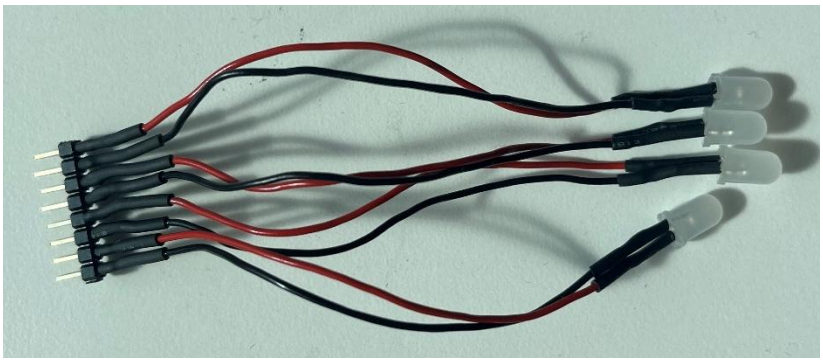
All the sockets and decoupling capacitors for the chips were then added

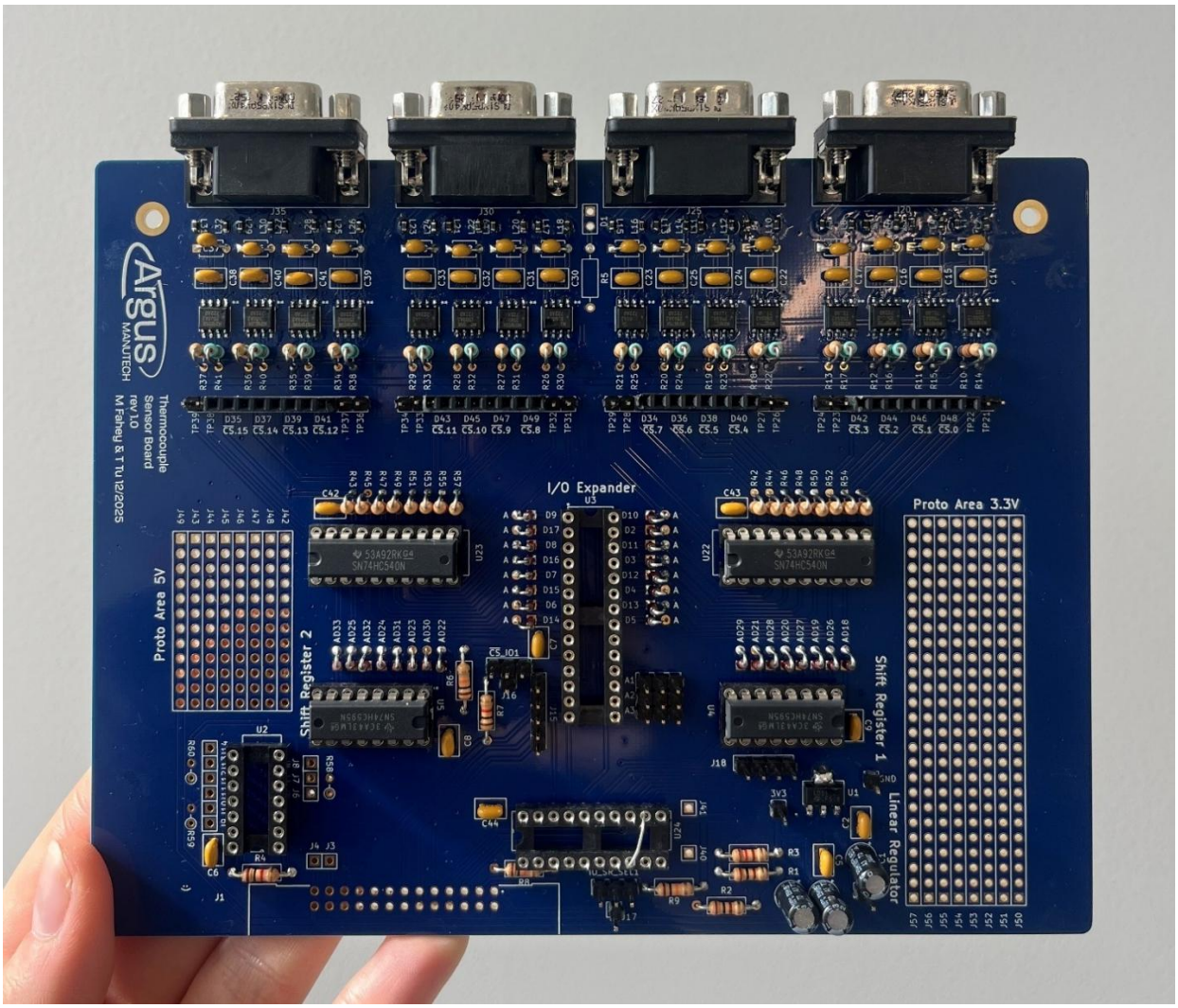


After this the pull up and pull-down resistors, as well and relevant test point pins and diodes for

the shift registers were added, and the shift register was tested for functionality. This worked well, and the rest of the PCB was populated. Lastly the bus drivers for the LEDs were tested and worked successfully, and the whole system was operating well as hoped with the code (Yay!).

In order to see the LEDs from the front plate, they needed to be extended using wires so they could fit into the front plate instead of just being soldered into the PCB. The positive end of the LED was soldered onto a red wire to signal that it was the power side, and the negative end was soldered onto a black wire to represent the GND side. The connections were heat shrunk to protect the connections, before the other ends were soldered onto an 8 pin header alternating between power and GND and heat shrunk also.





It was decided that a second PCB would be populated using the same processes and procedures, a list of which are outlined below.

PCB Populating initial plan – Hopefully:

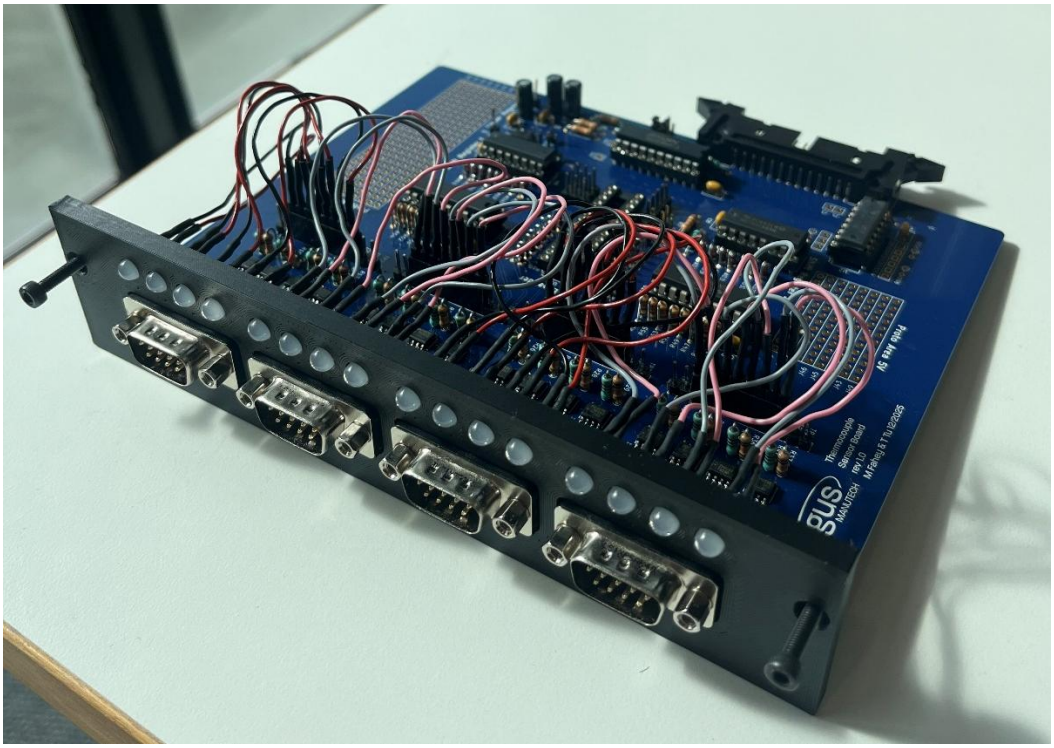
- Linear Regulator – Ensure 5V input turns into 3.3V output, measure with multimeter and reference to GND.
- Solder all ferrite beads and inspect under 6x microscope, making further adjustments as needed.
- Solder all MAX chips and inspect under 6x microscope, making further adjustments as needed.
- Populate each channel in sets of 4, testing for temp readings after every set.
- Solder in sockets and decoupling caps
- Solder in pull up and pull down resistors (flat resistors)

- Solder in all pin headers (test pins ect)
- Solder in diodes, bending into verticle position, Anode up, cathode down (cathode has black strip.
- Place shift register in socket and test for temperatures
- Add in veriticle resistors and place line drivers in socket.
- Create LED strips for chip selects and test
- Attach right angle 26 pin header and test through ribbon cable.
- Mount to front plate and PCB is complete (ensure selector connection is made for the shift register)

17. Housing Adjustments – After PCB

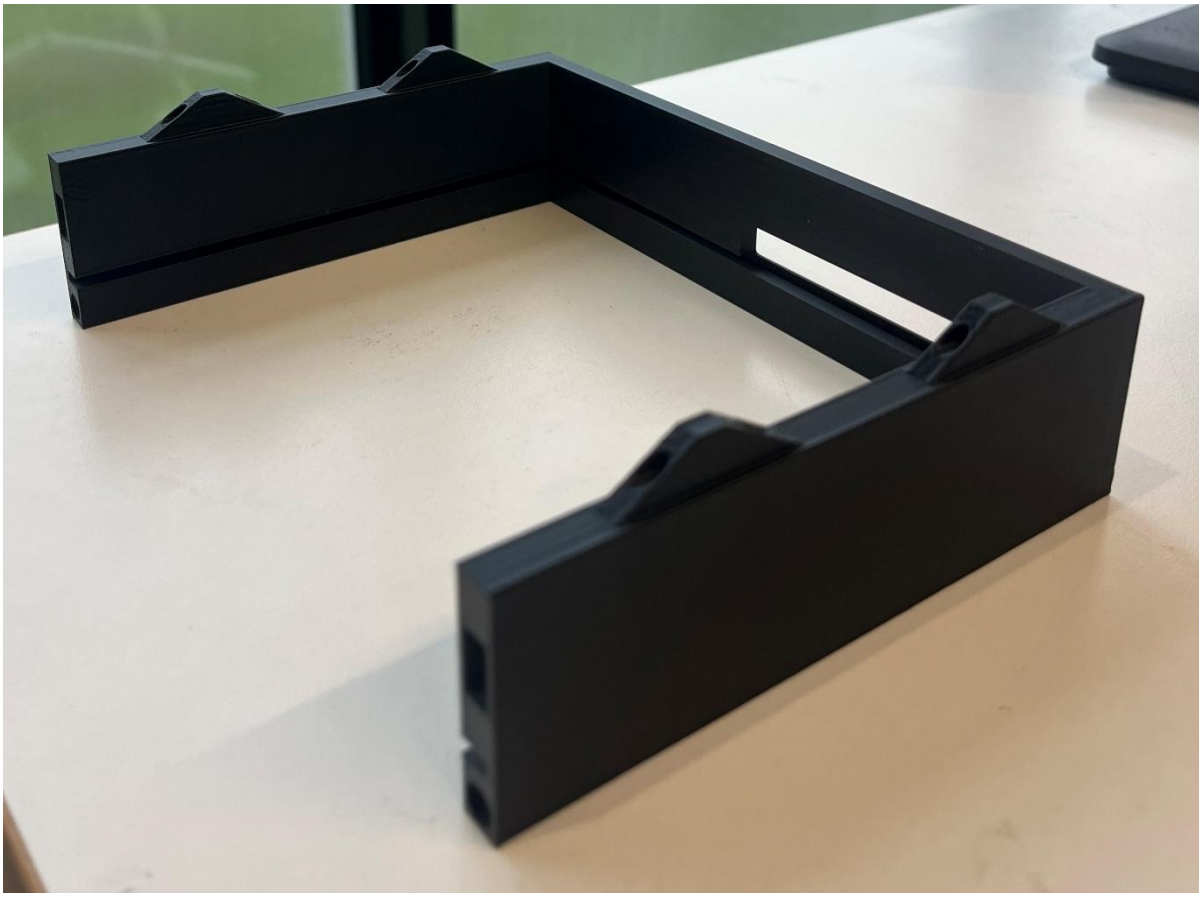
PCB Housing:

Once the PCB arrived, we put it into the housing to make sure it all fit as expected. The front plate worked perfectly, with the correct fit for all D-subs and the mounting holes also. This was then printed in PETG for a higher temperature rating, and the populated PCB was mounted to it with LEDs connected also.

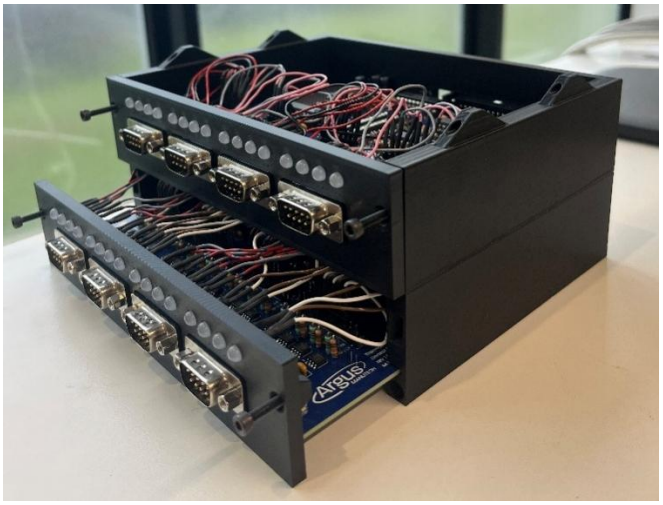


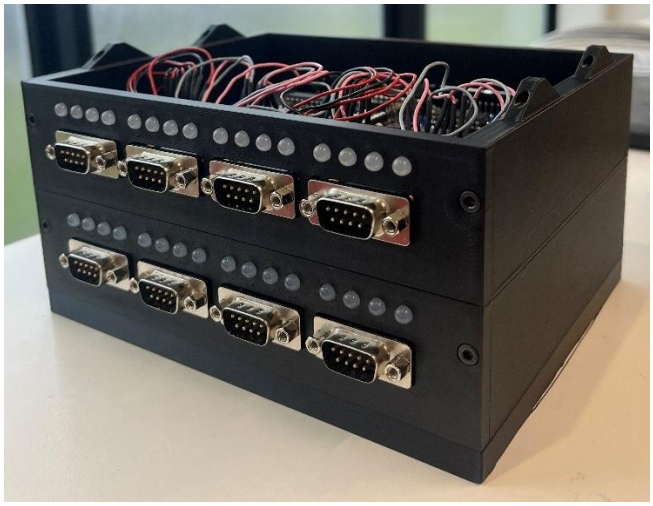
There were a few minor fixes needed for the main housing, there had been no tolerances with the size, so while it fit in, it was very rigid and a tight fit, not allowing the PCB to slide in and out as easily as we had hoped, a tolerance of a few mms was added to the grove where the PCB sat to fix this problem for the final PETG print. Another small fix was that the 26 pin right angled header was slightly out of alignment, this just needed to be moved along slightly in the next iteration as well.

After that the PCB housing was printed in PETG for a high temperature rating also.



The front plate was connected to the PCB which was then slotted into the housing for testing and operation. They were all secured together using M3 screws.

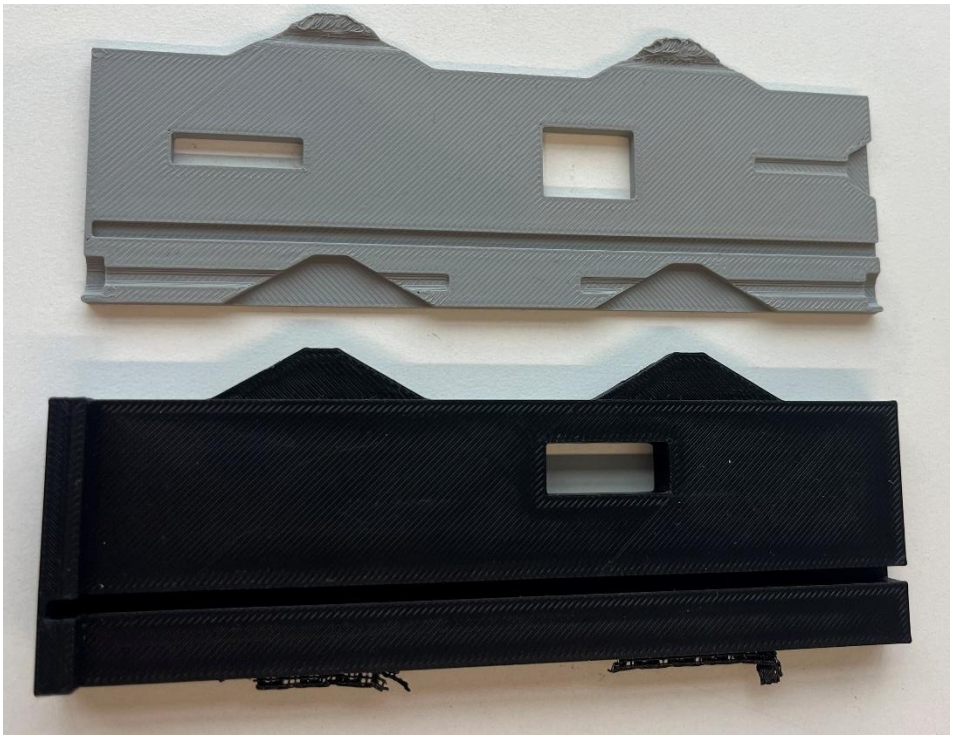




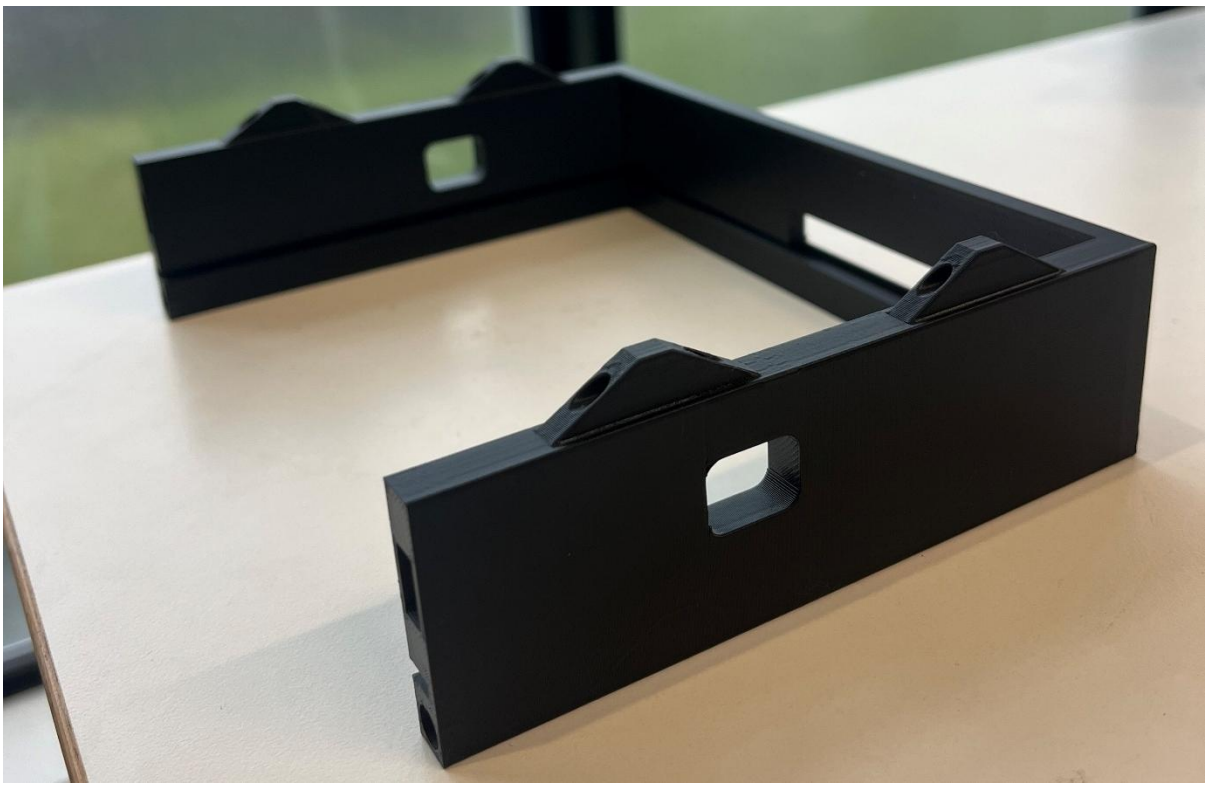
Microcontroller Housing:

The housing for the microcontroller was similar in function to that of the PCB however there were some adjustments needed for the access points that were different from the PCB.

Firstly, the microcontroller needed two cut outs for the micro-USB cable to connect into the Nucleo board, one was added on either side so that the computer could connect easily to the housing. These positions were measurement with a calliper, and some test prints were done using PLA.

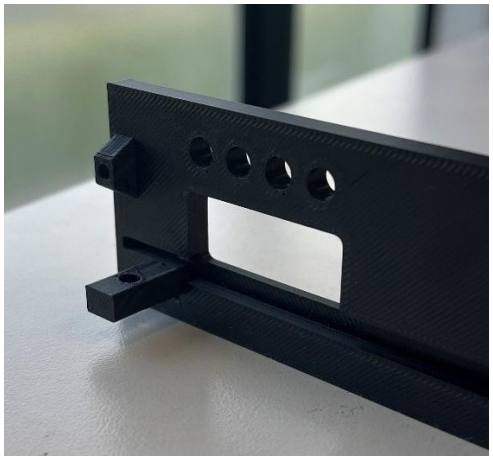


Once the holes aligned well with the microcontroller board and the micro USB connection, a proper housing was printed in PETG.

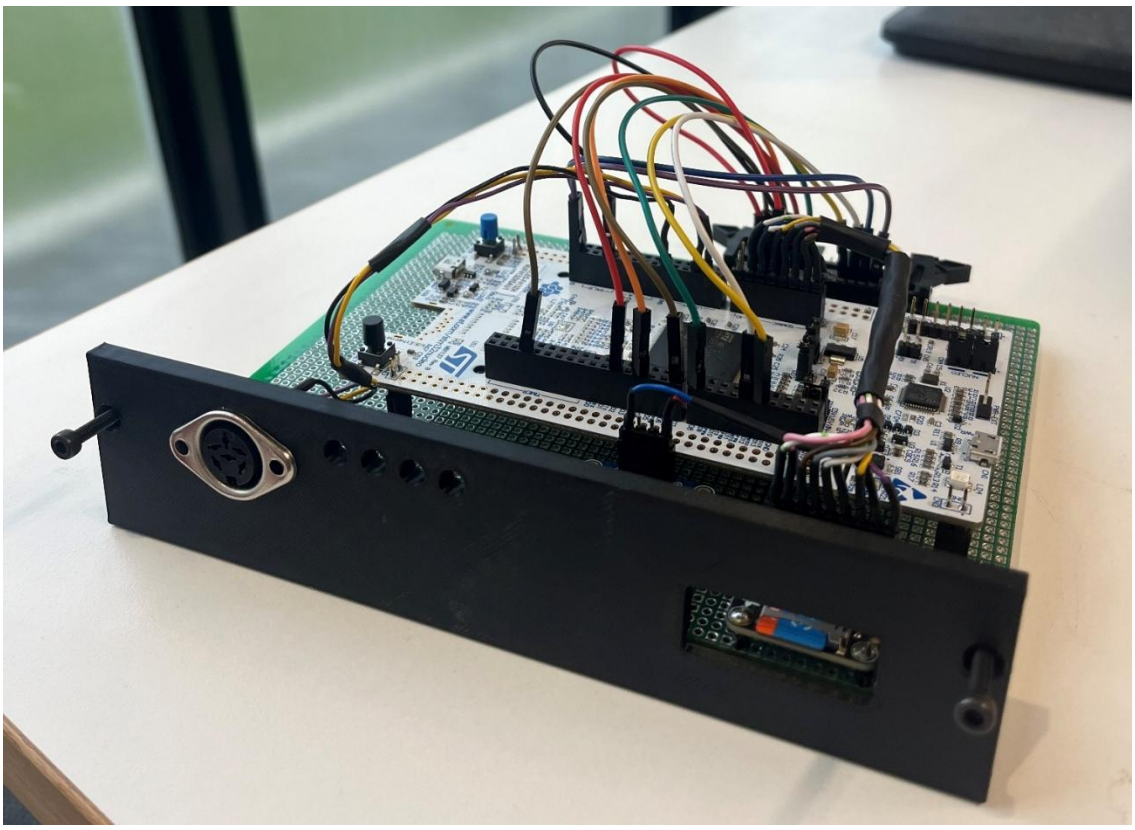


Secondly, the MCU front plate didn't need all the cut outs for the D-Sub pins. Instead, there was a slot for the SD-Card so that it could be inserted and ejected easily. As the SD card holder was soldered near the edge there needed to be more clearance underneath for the solder as well as the board. There were four LED spots for testing and debugging LEDs to be connected to if necessary, and a circular gap and mounting holes for the DIN connector which allows serial communication with the laptop via UART.

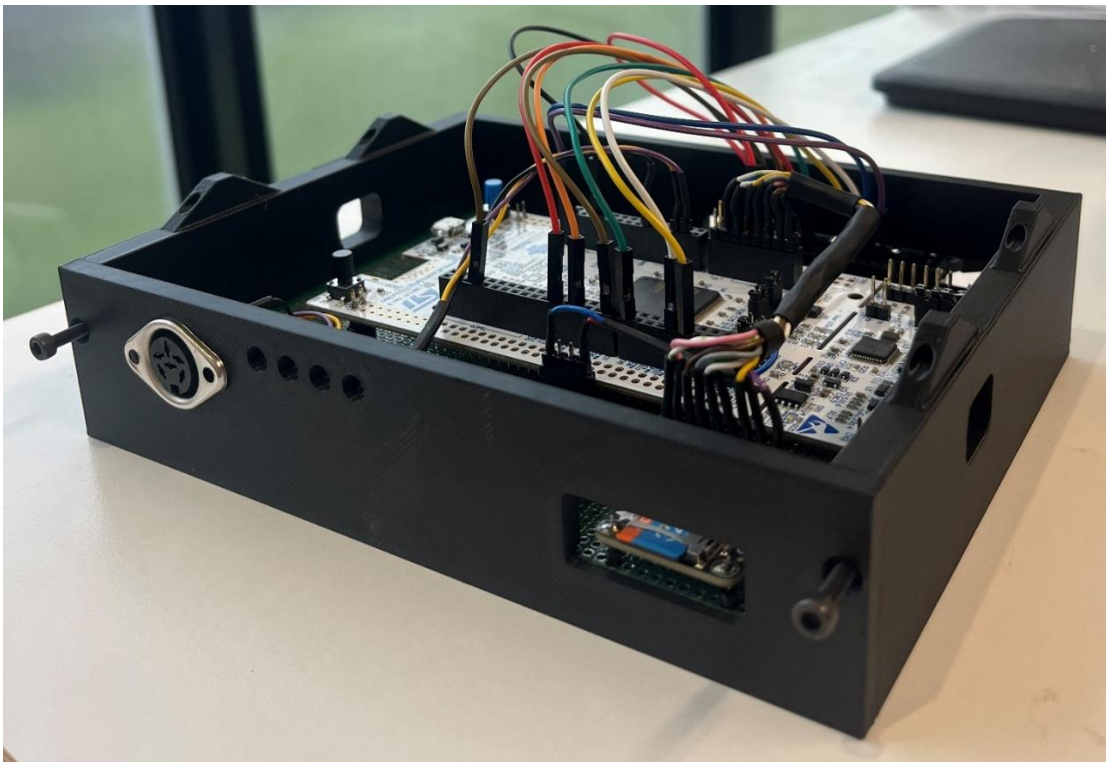
A test print was done to ensure the connect dimensioning for the DIN connector and enough clearance for the SD-Card holder.



The only adjustment after this was to shift the LEDs along so they had more room to be jumped from the microcontroller without disrupting access to the SD-card. The final PETG version was then printed.

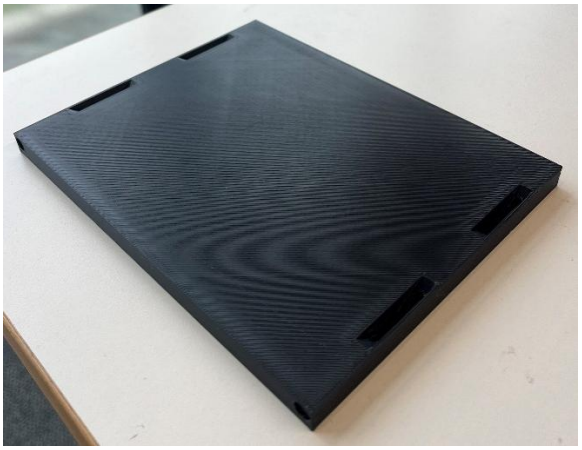


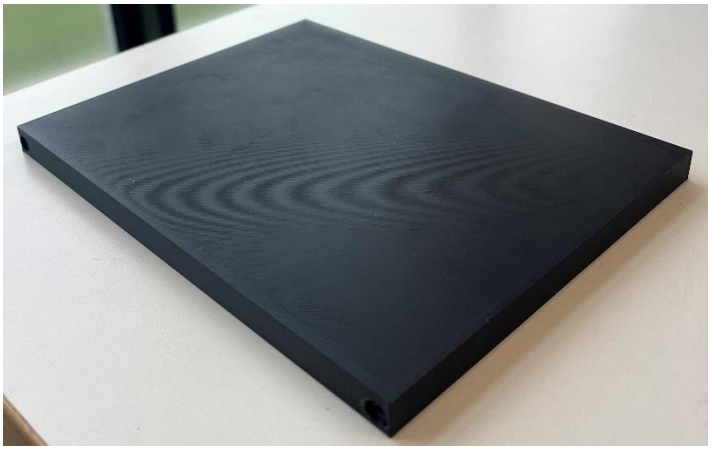
These can then be connected to protect the microcontroller during testing.



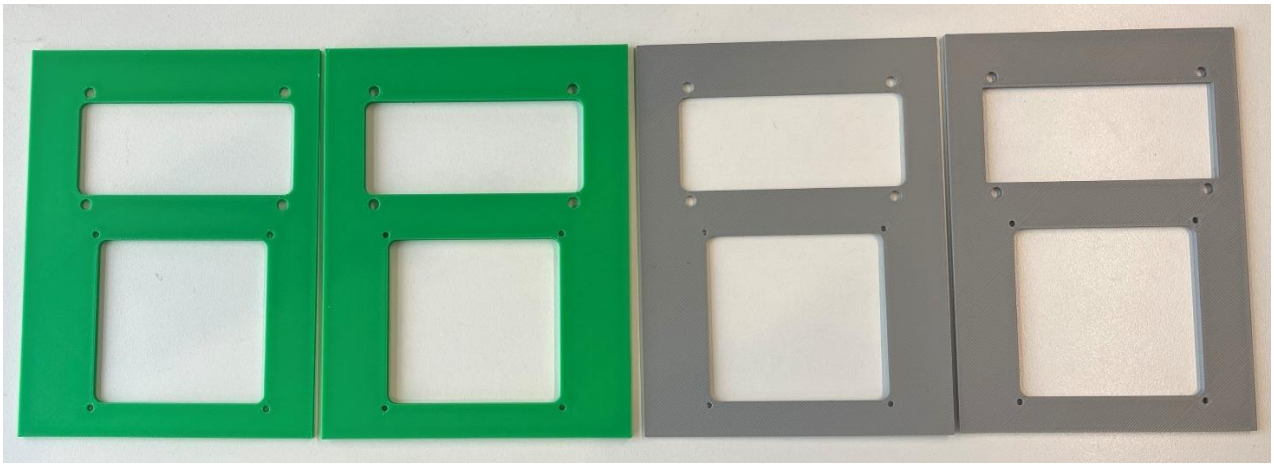
Top Plate:

The front plate was initially going to be a flat plate of a similar design to the base plat, to secure to the top of the structure and keep everything contained. A print of this was done in PETG, although further developments were suggested after this.



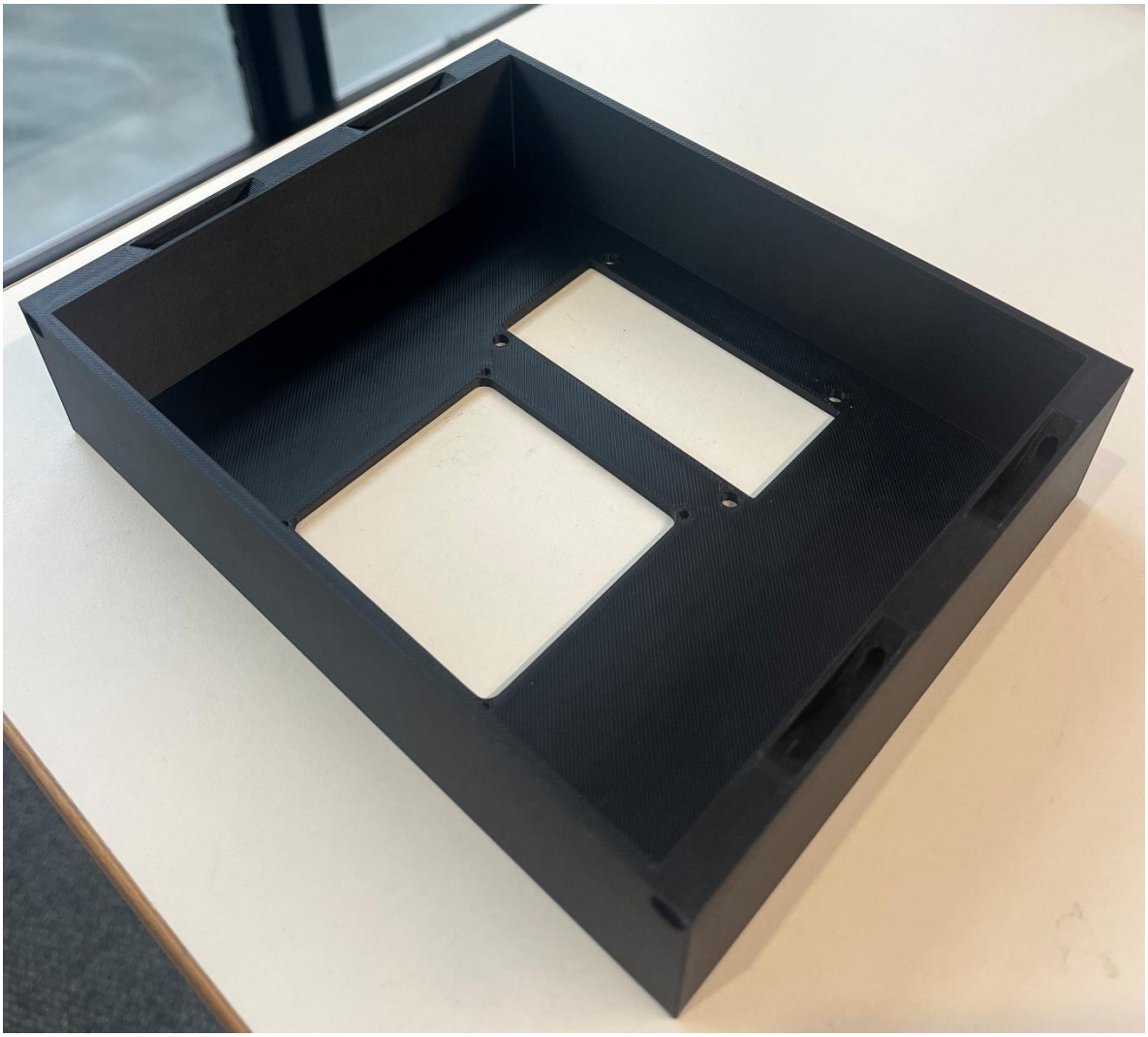


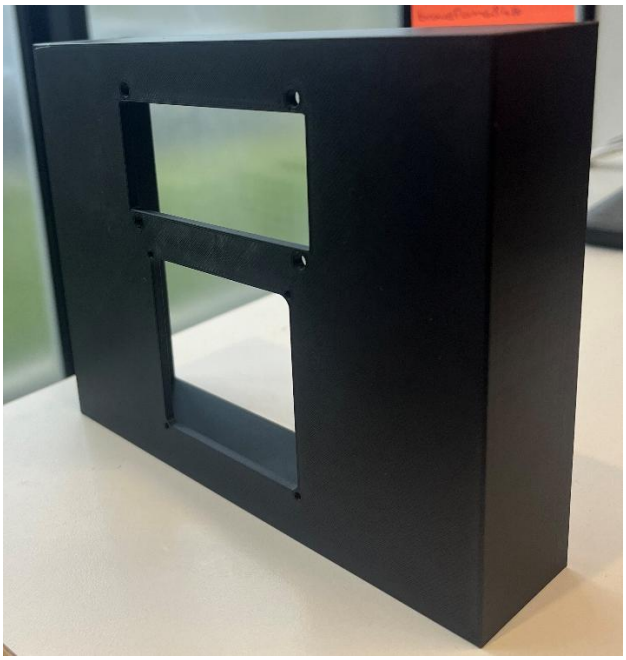
It was then decided that as well as the user interface that was operating on the laptop, it would be useful for Callan to be able to set positions of the thermocouples using a keypad and small screen display on the housing. We began test printing for these cut outs because the keypad chosen didn't have any dimensioned drawings. We tried to measure and estimate as best we could and adjust based on the test prints to ensure the most optimal fit to the keypad and mounting holes.



We also decided that the addition of the screen and keypad as well as the jumper wires for the microcontroller meant there needed to be more space in the housing, so the front plate was extruded by 40mm to allow for a spacious fit of everything inside. Another test print was done to ensure the keypad and screen fit inside the walls of the housing properly.

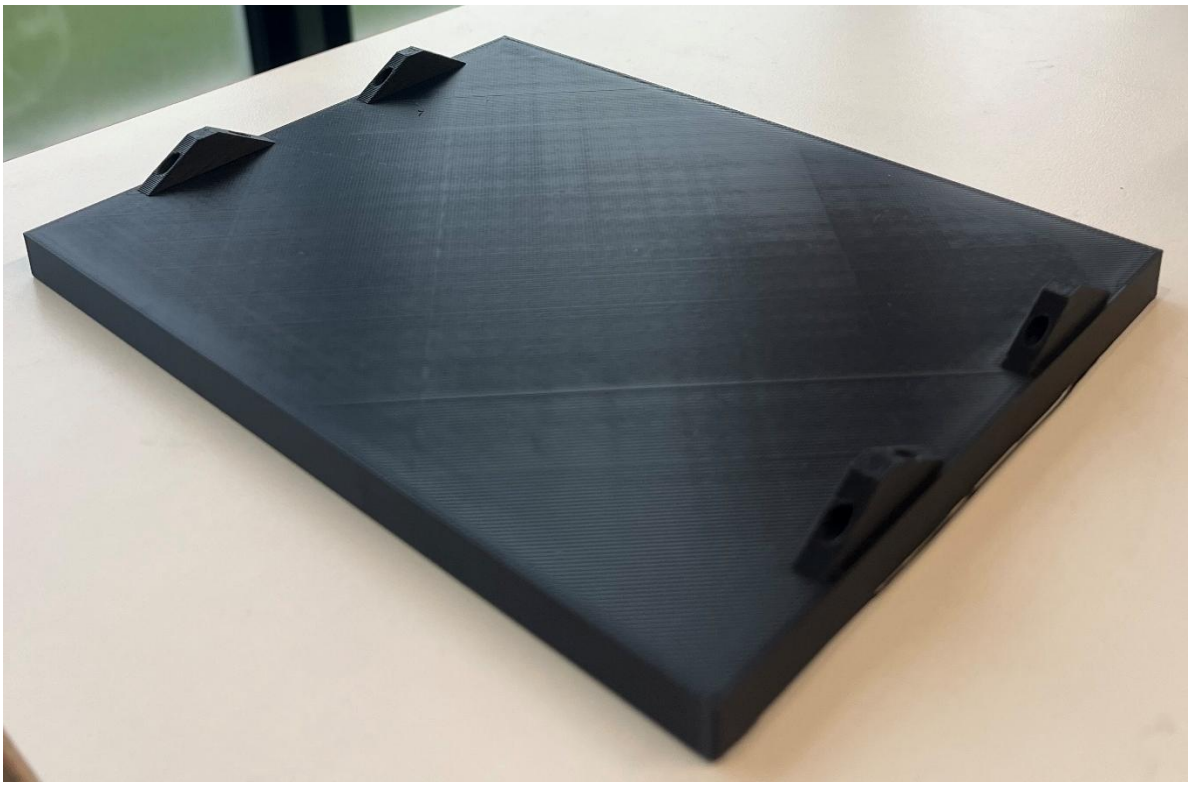
Once everything was aligned and working well a full-scale top plate was printed in PETG.





Base plate:

The bottom of the box was printed with the same tabs that allowed modularity of the housing. It was made 8mm thick to ensure the tabs had enough room to be secured.



All of These elements can be combined with the modular tabs and secured with 35mm M3 screws into one compact potable structure.

18. Thermocouple D-Subs

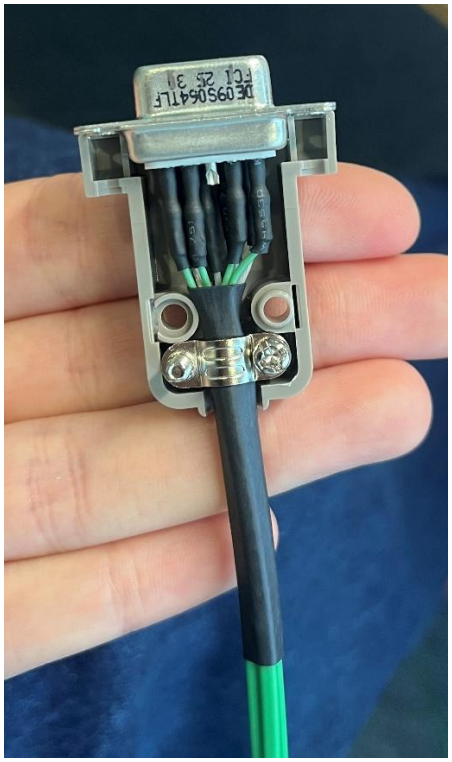
Creating the D-Sub pins initially began with an attempt to solder the thermocouple wire into the D-Sub pin hole connections like initially planned. However, this resulted in faulty and unstable connection as the material wasn't a good match for the solder and connecting them this way wasn't going to work.

From here it was decided that the ends of the thermocouples were going to be crimped into pins, and these pin connections were then soldered successfully into the D-subs.

Green D-Sub used the K-13-1 crimping tool with a setting of 1, the green end is T+ which goes on the row with 5 pins, and the white ends are T- which goes on the rows with 4 pins.

Blue D-Sub used the K-42 crimping tool with a setting of 3. The red end is T+ which goes on the row with 5 pins, and the white ends are T- which goes on the rows with 4 pins.

After this, each pin of the D-sub was covered with heat shrink, and all 4 thermocouple wires were heat shrunk together as well to allow them to fit easily into the D-Sub back shell casings.



For each D-sub, the thermocouples were sealed using glue lined heat shrink to protect them from liquids and to prevent deterioration. The thermocouples were also colour coded using heat shrink, details outlines here:

Red – Thermocouple 1 – D Sub pins 1 and 6

Yellow – Thermocouple 2 – D Sub pins 2 and 7

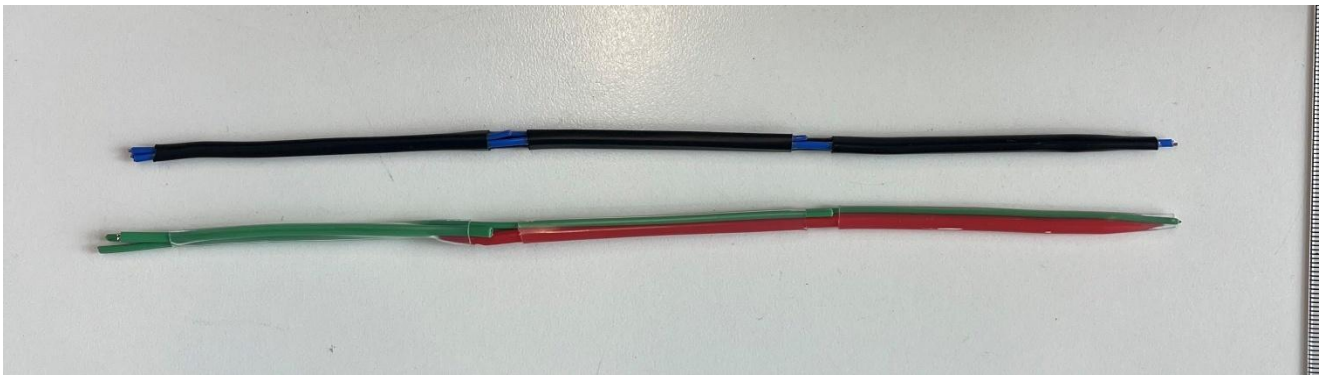
Green – Thermocouple 3 – D Sub pins 4 and 8

Blue – Thermocouple 4 – D Sub pins 5 and 9

It is also in rainbow order to make the colouring system more intuitive. The colour coding was placed around 2 inches from the end of the thermocouple, as Callan had specified that this would be the most helpful for when he was placing them in the tank.

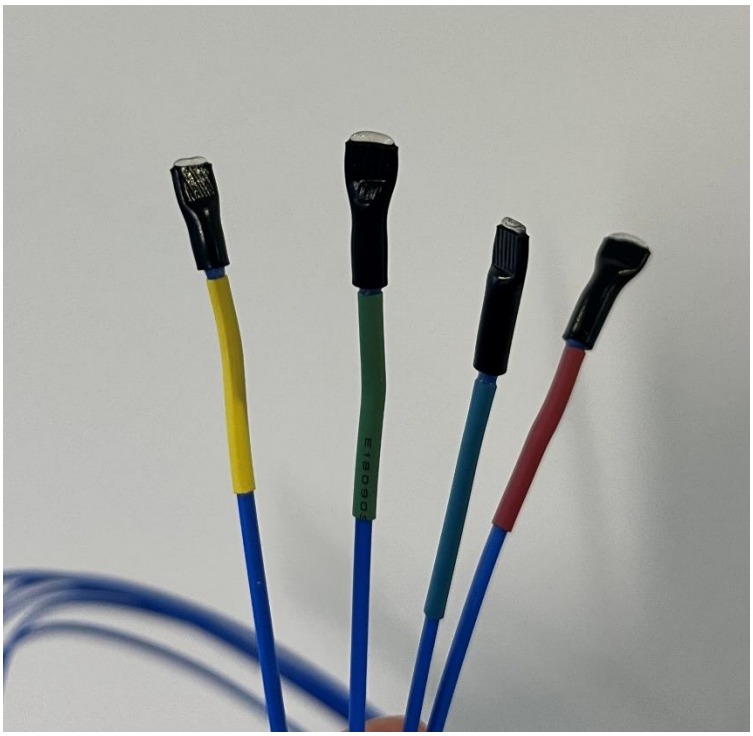
19. Thermocouple Measurement Array

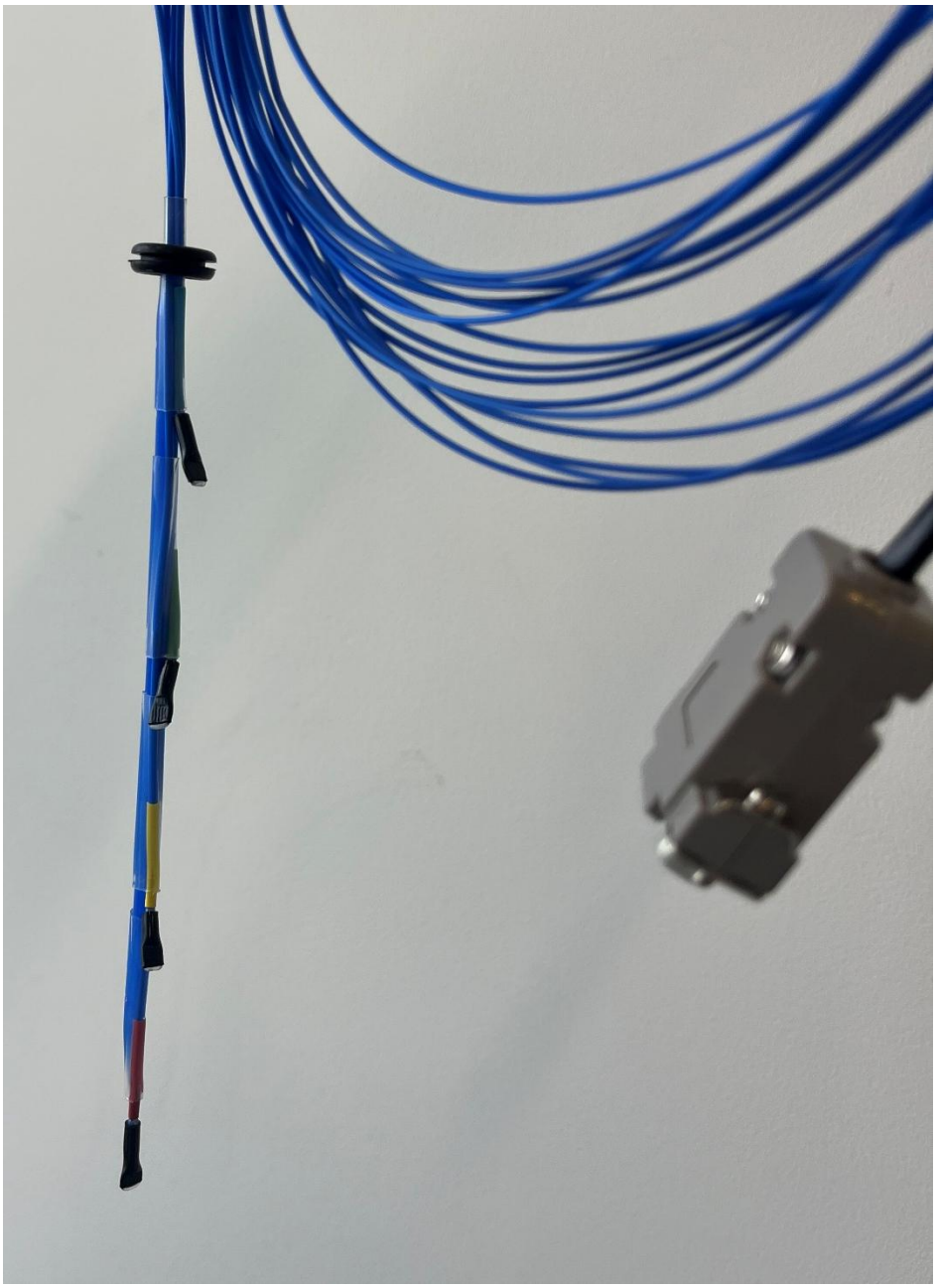
Before making the thermocouple array, some tests were done using thermocouple wire and heat shrink, along with pick up sticks, to see how we could make a rigid stick that could be threaded through a grommet in the lid of the tank.



The first test was done using a pickup stick and black heat shrink, which worked really well and proved the rigidity of the structure. However, issues arose when it came to establishing which thermocouple was which. Because the heat shrink was black and therefore identifying the thermocouple was not possible.

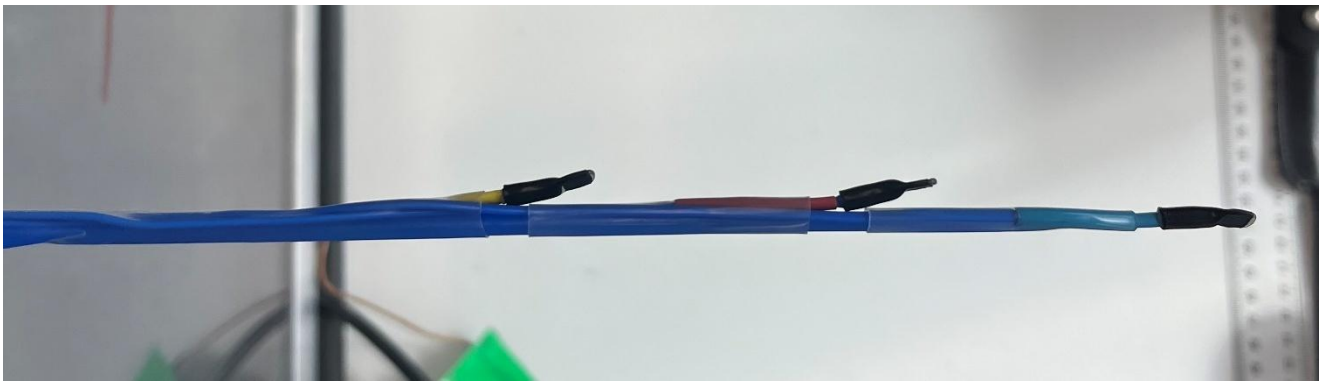
A second test was done, using clear heat shrink. This worked well also, however there were a few things to sort out. Firstly, while the pickup stick worked well for the structure, there was minor concern that the colour of the pickup stick might get mixed with the thermocouple colour coding and cause confusion. The location of the colour coded heat shrink was also a problem. When speaking to Callan, before the hardware was designed properly, he said the colour coding would be best around 2 inched from the end of the thermocouple, so this is how they were made. However, in this design, that would mean the colour for each thermocouple would sit next to a different thermocouple, because they were being placed around 2 inches apart from each other. To solve this, a new set of 16 thermocouples were made for the small tank demo, so that the ones created for Callan's testing remained how he wanted them. The new thermocouples had the heat shrink right next to the end of the thermocouple and were easily identifiable under the clear heat shrink as the rods were made with a blue pick up stick to match the colour of the thermocouple wire.





The first four thermocouples were all attached together, 5cm apart and attached to the first D-sub pin which worked well. However, the design that was chosen had all the other thermocouples placed in rows of three. This meant that 4 rows were feeding into three D-sub, so the wiring was done very carefully to ensure that there were no mix ups, and we could identify which thermocouple was which. The process went as follows.

Four thermocouple wires were threaded through usually two different grommets in the lid of the tank (Non thermocouple end first, threaded from the underside of the lid). These four wires were then threaded through heat shrink and soldered onto the associated pins of the D-sub. The same was done for the other 3 grommets on the lid. Soldering each D-Sub after four were threaded, to ensure the colours weren't being mixed up or sent to different D-Subs.



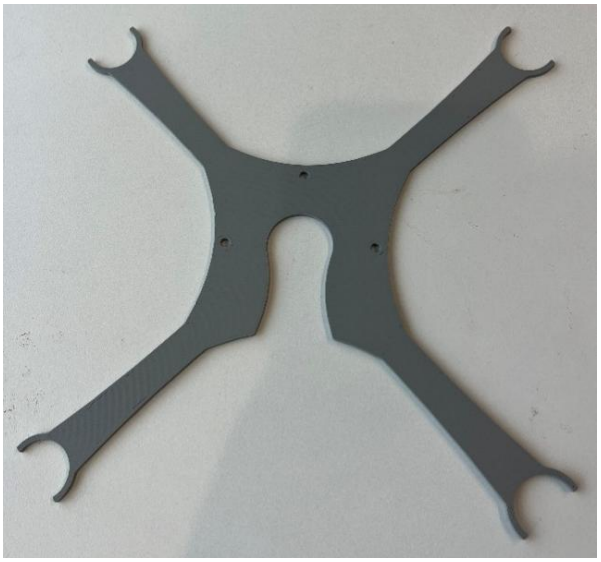
After this, the thermocouple rods were made with the three thermocouples on the underside of the grommets. They were spaced 5cm apart from each other and had two blue pickup sticks for stability. Two clear heat shrink pieces of 4cm were placed between the bottom two thermocouples and a longer one to secure them all the way to the top of the take was used above it. All three heat shrink pieces needed to be threaded on before then being shrink one at a time from the bottom. This also needed to happen very gradually as to not melt the plastic of the pickup stick in the process. This process was done for all the thermocouples, and they were then threaded back through the grommets slightly until they were sitting roughly in the correct position.

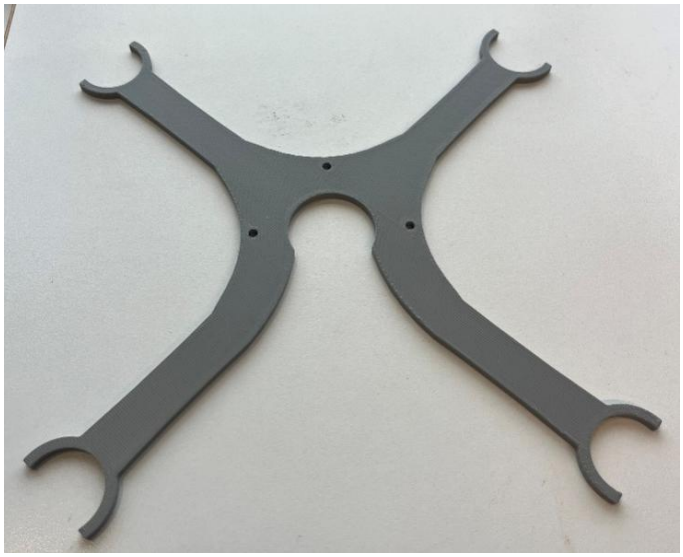
While a successful process in general, the angle of the wires into the tank meant that the thermocouples were not as stable as anticipated, therefore a design solution to keep the thermocouples straight needed to be designed.

To solve this, we decided that there needed to be some sort of structure attached to the thermocouples above the lid of the tank to force them to sit fully vertical, ensuring a straight line of

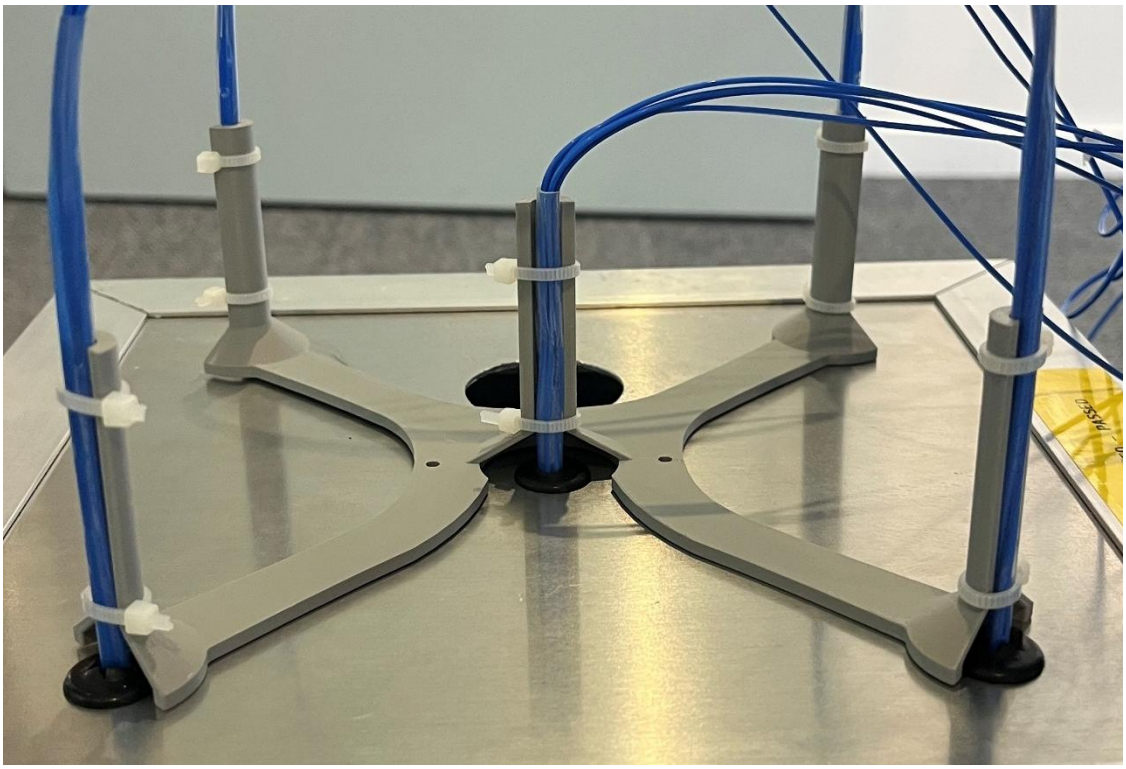
thermocouples and uniform measurements. The grommets on the top of the tank meant that there needed to be some space before the extrusion to secure the thermocouples too. A test print of the fit around the grommets was done as well as some extrusions to ensure the thermocouples would sit correctly inside them.

Next, the base of the structure with a 150x150mm grid spacing between grommets was printed. Although, the fit didn't quite align perfectly, so the hole diameter was increased from 20mm to 22mm to allow for more space and a better fit of the structure.





A final PLA structure was printed and fit exactly as anticipated, so was then attached to the thermocouples using zip ties and the array was ready for testing.



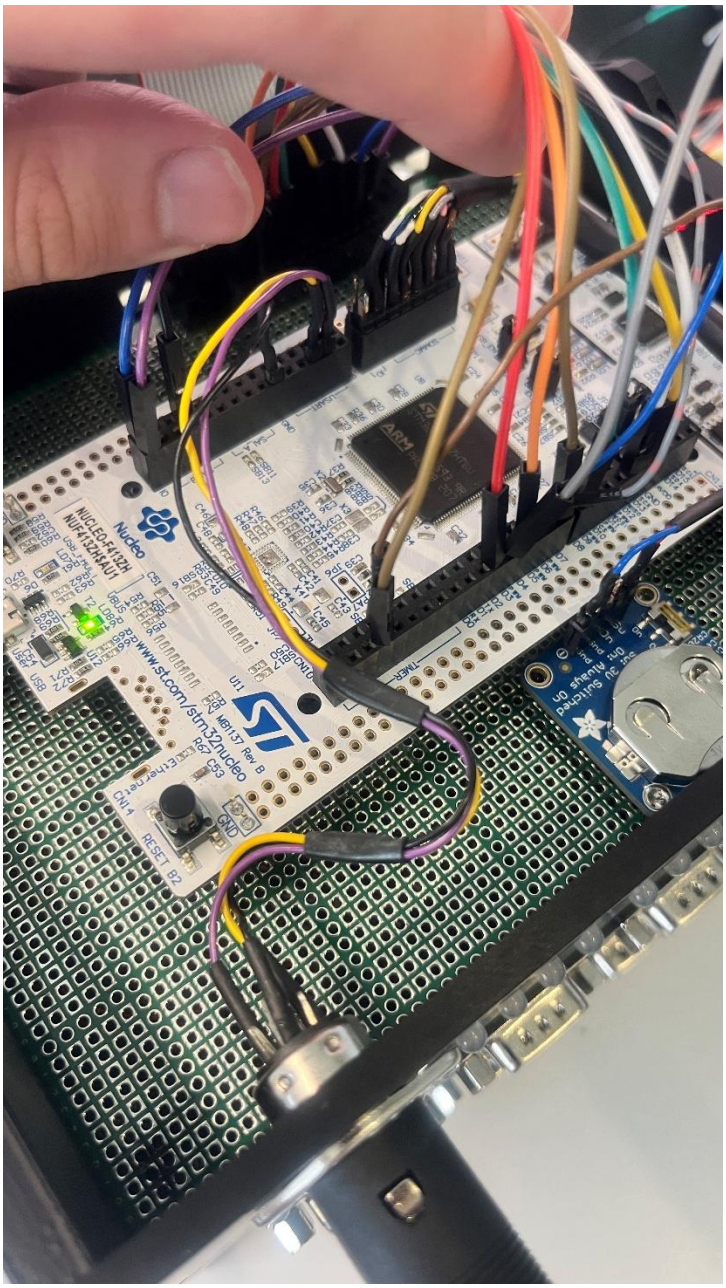
20. UART Connections

Purple line is TTL's TX which means it's connected to PD6 (USART_B_RX) and the yellow from the TTL is RX which means it is connected to PD5 (USART_B_TX).

Purple is TX

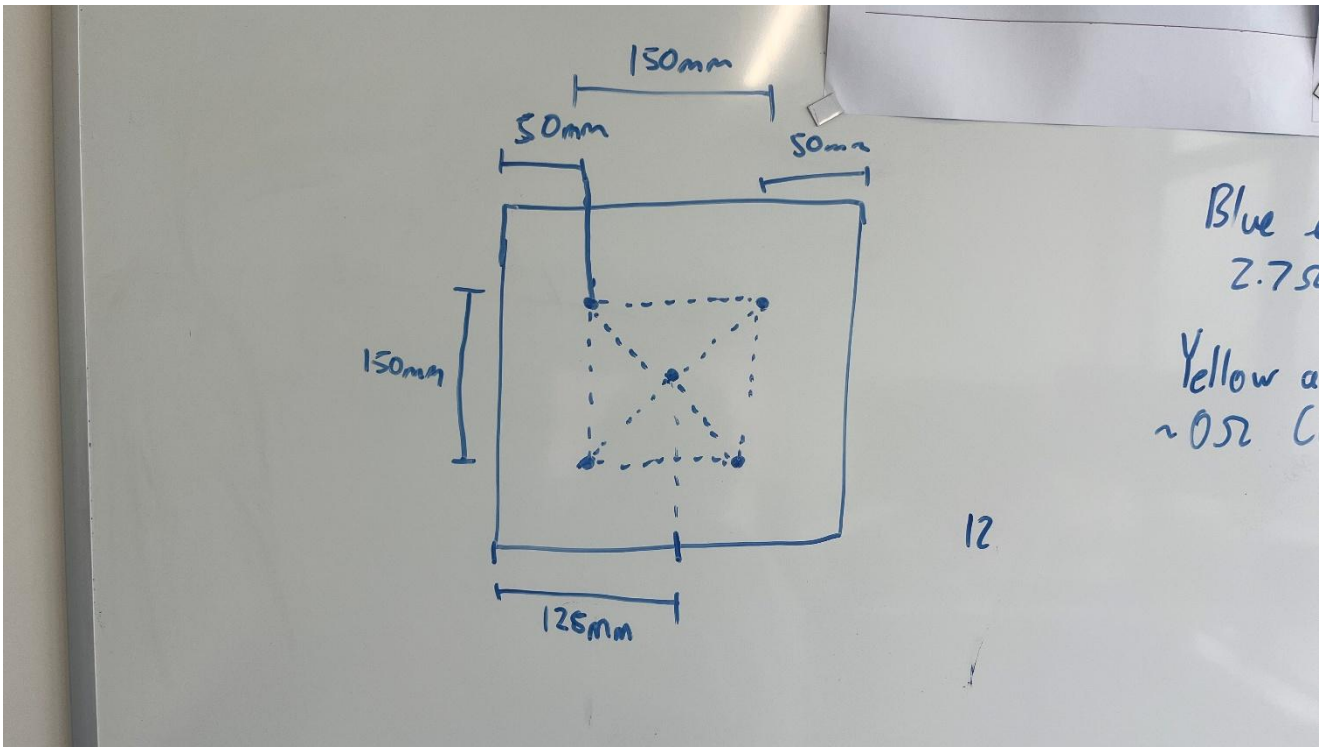
Yellow is RX

Black is GND

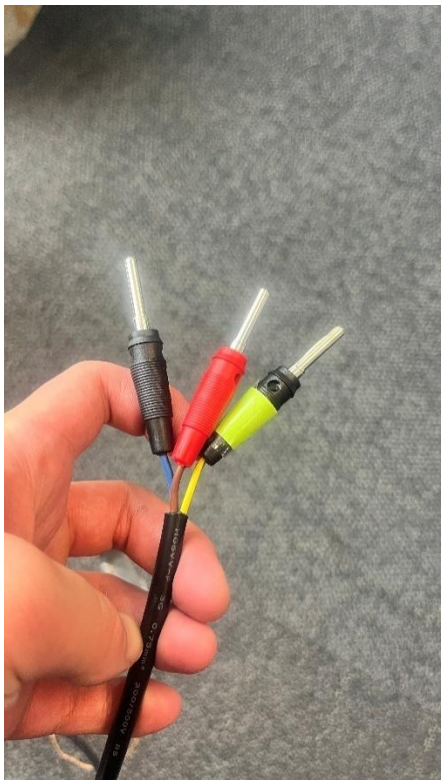


21. Water Tank

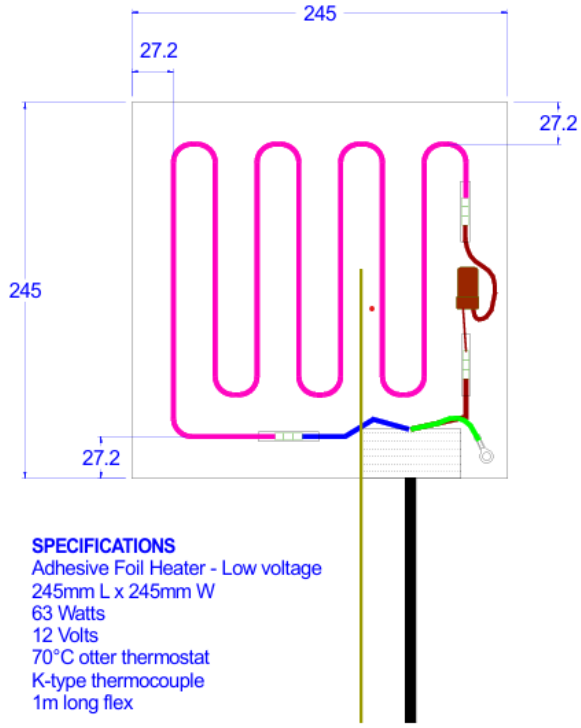
Dimensions planned where the thermocouples are going to enter through the lid are shown below.



The power supply for the tank is shown below, where red is VCC +12V black is GND and green is protective earth. There is a ferrule surrounding the banana cable to protect the copper wiring.



The heater underneath the tank has specifics shown below.



22. Presentation!

Intro:

- For our summer intern project, we have been developing a low-cost thermocouple sensor measurement array and visualisation software. This project is just a proof of concept, and we have developed the mechanical structure, PCB and visualisation software for this concept.

Why is our project important for you?

Company needs to understand behaviour of material of clients, high value material honey chocolate and molasses.

Difficult in industry to heat products as fast as possible and not damage those products.

Small tank to see behaviour

Heating element takes hours and so to show principle of operation we poured hot water into cold water.

What is the problem that we are trying to solve?

The problem that it is difficult to understand the dynamics of non-Newtonian fluids when heated, So, by understanding the dynamics of Non-Newtonian Fluids Argus would be able to expand into new markets to heat expensive products in bulk like chocolates, molasses and AMF.

Current Problems

- Expensive product can get damaged or wasted,
- Product is burnt, packaging melts and product is lost, product isn't fully melted.
- A lot of gaps due to not understanding behaviour and lack of data

Current system to address this issue is using PICO loggers and they have

- Extremely high sampling rate which we don't require as heating products can take days. (Can read all connected thermocouples in ~0.1 seconds)
- Expensive: costing ~$130 per sensor.
- Limited in scale (Callen is only placing 8 sensors on the heaters and 5 in the fluid itself)

The system that we have developed

- Our system is cheap: costing $40 per sensor at scale.
- The system is scalable, as it's capable of acquiring data from up to 256 sensors.
- The system can sample 8 times per second and has a resolution of 0.25 degrees Celsius.
- By building our own system we will own the IP.

The other benefits of the system we have developed is:

- that we can visualise the data points in a 3D view – this cannot be achieved with the current PICO logger.
- Set the positions of the measurement points to required locations
- Playback the data points and visualise the spread of heat throughout material

This is the playback of a test we conducted in water.

So what does this mean?

- Proof of concet
- Verify FEA
- Small scale testing of products

It proves that is it possible to get data from sensors in a tank, and is scalable, so we can get as many data points as we want. It can verfily FEA fluid dynamics of known products like molasis, and give us new data for small samples of products that way have dynamics that we don't understand

Sensors can also be used in small scale tests. If a small amount of product can be tested and the data So, by understanding the dynamics of Non-Newtonian Fluids Argus would be able to expand into new markets to heat expensive products in bulk like chocolates, molasses and AMF.

Allows us to prove simulation against reality and if it's accurate then we can work with a small amount of material and scale that experiment up verify the large scale.

Why is our system better?

Show system

23. Testing

The first test was conducted by heating a 12.5L tank of cold water from 20 degrees Celsius with a 63W heater. This test didn't go so well since there were some data corruptions therefore, we could not replay the data back. Code changes and improvements were made and further tests conducted did not have those issues appear again. The error could have been to due these following reasons:

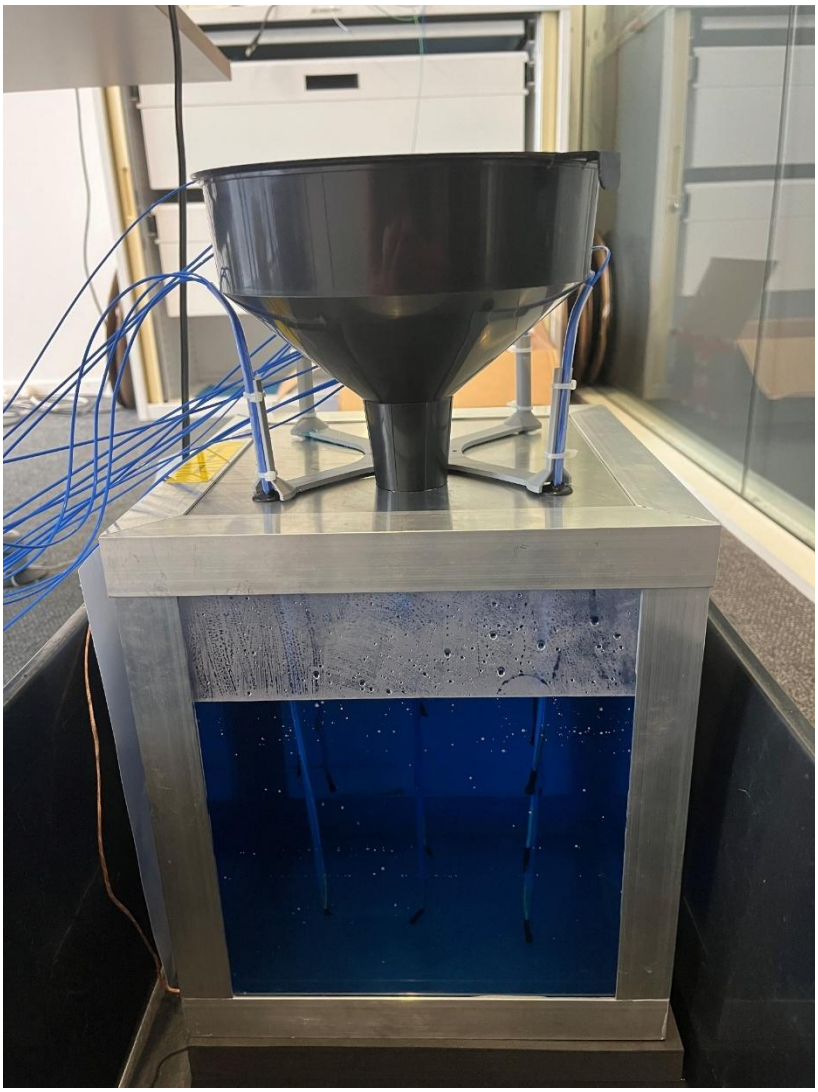
- The USB cable to allow the E: drive to be seen on the laptop was not plugged in fully which might have cause an issue with the data.
- On the JS server side not, enough delay was provided before after the RESET command was sent. This meant that the server was trying to read the E: drive during reconnected or at a bad time which cause data corruption.

```
10:10:36
Traceback (most recent call last):
  File "<stdin>", line 632, in <module>
  File "<stdin>", line 629, in main
  File "<stdin>", line 554, in run
  File "<stdin>", line 479, in handle
OSError: [Errno 5] EIO
>>>
```

The second test was conducted by pouring hot tea into the tank with cold water and the experimental results is what we had expected but the issue was that the sampling rate of the thermocouples was too low as it was once per seconds. This meant that the colour change on the screen was instant and you could not see a gradual change in the colours and therefore the quality of the results was low. The hot water was poured through via a funnel that was close to the centre of the tank but offset a bit. This allowed us to visually see that the bottom thermocouples getting hot first and then the other thermocouples would increase their colours gradually.

The third test was conducted like the second test but instead of using the brewed hot tea blue dye was poured into the hot water. This allowed for the visualisation of the flow of the hot water in the tank be of higher quality. The sampling rate for this test was increased to be 10 samples per seconds i.e. 10hz. This experiment improved upon the second test since there was a more gradual increase of the colours shown the browser page.

The setup of the third test is shown below.



24. Code Python Side

There have been over 20 iterations of the code for this project on the python side. Details regarding version are demonstrated in the past versions.

Shift register code

Below showcases the “bit bang” version of the shift register. This allows the shift register to shift a bit “0” or “1” to its outputs.

```
class SR74HC595_BITBANG:

    #Initiliases the shift register (bit bang version)
    def __init__(self, rclk_pin, ser_pin, srclk_pin, srclr_pin, oe_pin):

        self.rclk = machine.Pin(rclk_pin, machine.Pin.OUT)
        self.ser = machine.Pin(ser_pin, machine.Pin.OUT)
        self.srclk = machine.Pin(srclk_pin, machine.Pin.OUT)

        self.oe = machine.Pin(oe_pin, machine.Pin.OUT)
        self.srclr = machine.Pin(srclr_pin, machine.Pin.OUT)

        self.enable()   #Initiliases output enable

    #Triggers a clk pulse to write ser value to the register
    def _clock(self):
        self.srclk(1)
        self.srclk(0)

    #Writes a bit to the shift register depending on value
    def bit(self, value, latch = False):
        if value:
            self.ser.high()
        else:
            self.ser.low()
        #Triggers a clock pulse in order to write that bit the shift register
        self._clock()

        #Latches contents onto output registers
        if latch:
            self.latch()

    def bits(self, value, num_bits, latch=False):
        for i in range(num_bits):
            self.bit((value >> i) & 1)
        if latch:
            self.latch()


    #Pulse the RCLK (latch) pin to copy the current shift register contents to the output pins (Q0-Q7). This updates all outputs simultaneously
    def latch(self):
        self.rclk.high()
        self.rclk.low()

    #Clears the contents of the shift register
    def clear(self, latch = True):
        self.srclr.low()
        self.srclr.high()
        if latch:
            self.latch()

    #Enables the shift register
    def enable(self, enabled = True):
        self.oe.value(not enabled)

if __name__ == "__main__":
    print("Shift register File")
```

A simple example of how to use the shift register can be seen below:

1. Set the latch pin (RCLK) to LOW to prevent updating the output pins while shifting data.
2. Send data bit by bit to the serial data input pin (SER).
3. For each bit, pulse the clock pin (SRCLK) to shift data into the register.
4. Repeat steps 2 and 3 until all bits are shifted into the register.
5. When all eight pulses have been received, enable the 'Latch' pin to copy the values to the la tch register. Latch is the RCLK pin.

Below shows the code for the allowing data to be read and converted to degrees Celsius from the MAX31855 chips.

MAX31855 Code

```python
import pyb
import machine
import time

#Class for thermocouple chip (MAX31855)
class MAX31855:
    size = 8                        #Array size variable

    #Initiliase the thermocouple
    def __init__(self, cs_pin, spi_bus, raw_data):
        self.cs_pin = cs_pin        #Chip select pin for the IO expander
        self.spi_bus = spi_bus      #Spi bus lane
        self.raw_tc_data = raw_data #Raw data from the thermocouple
        self.raw_tc_integer = 0     #Raw data from the thermocouple as an integer
        self.tc_data = 0            #Thermocouple probe data in binary
        self.tc_ref_data = 0        #Thermocouple reference data in binary
        self.tc_c = 0               #Thermocouple probe temperature in celcius
        self.cj_c = 0               #Thermocouple reference temperature in celcius
        self.tc_buf = [0] * self.size   #Thermocouple probe temperature array
        self.error_flag = False #Thermocouple error flag
        self.error_data = 0 #Thermocouple error data
        self.tc_total = 0 #Thermocouple probe total temperature value
        self.tc_avg = 0 #Thermocouple probe temperature average

        self.counter = 0            #Array counter used to fill temperature array

        # Position data of the thermocouple
        self.x = 0
        self.y = 0
        self.z = 0

        self.convert_temp()

    def read_thermocouple(self):
        self.raw_tc_data = self.spi_bus.read(4)    #Read the thermocouple value from spi bus

    #Converts temperature of data recived from thermocouple into degrees
    def convert_temp(self):
        self.raw_tc_integer = int.from_bytes(self.raw_tc_data, 'big') #Converts raw thermocouple data from hex using big indiannes to a binary integer
        self.error_data = self.raw_tc_integer & 0xF #Stores the error data

        #Checks if there is error in the thermocouple and turns a flag true or false
        if(self.error_data != 0):
            self.error_flag = True
        else:
            self.error_flag = False

        self.tc_data = self.raw_tc_integer >> 18        #Shifting value by 18 since thermocouple data is only bits [31:18]
        self.tc_ref_data = (self.raw_tc_integer >> 4) & 0xFFF   #Bit masking for the reference temperature data since [15:4] is the reference temp data

        self.tc_c = self.tc_data * 0.25     #Each bit represents 0.25 degrees celcius of probe temperature
        self.cj_c = self.tc_ref_data * 0.0625   #Each bit represents 0.0625 degrees celcius of reference temperature
        #self.fill_array_tc_probe()             #Calls function to fill up thermocouple probe array

    def fill_array_tc_probe(self):
        #When counter is 0 initally populates buffer with first temperature value from probe else just populate one index
        if(self.counter == 0):
            # Fills buffer up with first value obtained from temperature probe
            for i in range(self.size):
                self.tc_buf[i] = self.tc_c
            self.tc_total = self.tc_c * self.size   #Obtains total of tc_buf
            self.counter = self.counter + 1         #Exits initialisation
        else:
            old_value = self.tc_buf[self.counter - 1]               #Obtains "old" value of array
            self.tc_buf[self.counter - 1] = self.tc_c               #Populates index counter - 1 with temperature probe value
            self.tc_total = self.tc_total + (self.tc_c - old_value) #Adds difference between new reading and old reading
            self.tc_avg = self.tc_total >> 3                        #Bit shift by 3 to divide by 8
            self.counter = (self.counter % 8) + 1                   #Increments counter by 1 and resets at counter = 8

    def print_temp(self, index = None):
        label = f"Thermocouple {index}" if index is not None else "Thermocouple" #Gives the thermocouple an index label if one is assigned
        print(f'{label} temp: {self.tc_c} °C')  #Prints current probe temp
        print(f'{label} Ref temp: {self.cj_c} °C\n')    #Prints current ref temp

    def average_info(self, index = None):
        label = f"Thermocouple {index}" if index is not None else "Thermocouple"
        print(f'{label} average temp: {self.tc_avg} °C')



if __name__ == "__main__":
    print("Thermocouple Module File")
#     spi_bus =
```

Above shows the classes for the two IC's utilised to read the temperature readings from the MAX31855 chips

Below is a simple explanation of how the python code is integrated:

25. System Integration Overview – Main structure

1. System Purpose

This system is a microcontroller-based temperature measurement system. It is designed to read temperature data from many thermocouples, allow their positions to be calibrated, and then store the results for later analysis. It also is able to send commands over UART in order to be visualised later

The system can:

- Detect which thermocouples are connected (via JS code)
- Read temperatures from all active sensors (via Python code)
- Save calibration and measurement data to files (via Python code)
- Communicate with a host computer over UART (via Python and JS code)

2. Overall Software Structure

The software is organised around a state machine, which means the system operates in different modes depending on what it is doing. Only one state is active at a time, this keeps behaviour predictable and easier to manage.

The main loop runs continuously and:

1. Executes the logic for the current state
2. Checks for and processes incoming UART commands

Time-based actions, such as temperature scanning, are triggered using a hardware timer.

3. Main System Controller

The System class controls the whole application. It does the following:

- Setting up all hardware (SPI, UART, GPIO, RTC, timers)
- Managing the current system state
- Sending UART commands over to the laptop

This keeps all high-level control in one place and avoids mixing hardware setup with application logic.

4. State Machine Operation

4.1 Initialisation State

The initialisation state runs when the system starts. It:

- Sets up hardware and software
- Creates the thermocouple manager
- Detects which thermocouples are connected

The system waits in this state until a USB connection is detected before moving on.

4.2 Calibration State

The calibration state is used to work with individual thermocouples. In this state, the system:

- Selects and reads one thermocouple at a time
- Receives position data (x, y, z) from the host computer
- Stores thermocouple positions in memory
- Saves position data to a CSV file once all data is received
- Can reload saved position data when requested

During calibration, temperature scanning is paused to avoid interference.

4.3 Measurement State

The measurement state is responsible for collecting temperature data. It:

- Uses a timer to trigger regular measurements
- Reads all active thermocouples one by one
- Converts raw sensor data into degrees Celsius
- Sends live temperature data over UART
- Saves timestamped temperature data to CSV files

5. Thermocouple Manager

The TC_MANAGER class handles all thermocouple-related hardware control. It:

- Finds which thermocouples are connected
- Controls which thermocouple is active using shift registers

- Selects the correct PCB for each thermocouple
- Reads temperature data using SPI
- Provides formatted temperature data to outer classes

6. Shift Register Control

- Shift registers are used to select which thermocouple is active
- This allows many thermocouples to be controlled using only a few microcontroller pins
- Only one thermocouple is enabled at a time to avoid signal conflicts
- Sensor selection is reliable and repeatable
- The shift registers are controlled using a bit-banged software approach
- The microcontroller manually drives the data and clock lines
- Hardware SPI is not used for shift register control

7. Thermocouple Interface

Each thermocouple uses a MAX31855 interface chip. The class for this chip allows:

- Reads raw temperature data over SPI
- Checks for sensor errors
- Converts the data into temperature values
- Stores probe and reference temperatures
- Stores position data for calibration

8. Timing and Measurements

A timer triggers temperature measurements by setting a flag. The main loop checks this flag and performs the measurement when needed.

The system uses a software timer (Timer (-1)) to periodically trigger temperature scans, with all time-critical and blocking operations handled in the main loop.

9. Communication and Data Storage

UART Communication

UART is used to:

- Send commands to the system
- Receive live temperature data
- Transfer a selected thermocouple's data and the measurement data
- Request system status

A helper class manages buffering and message handling.

File Storage

The system stores:

- Thermocouple position data in position.csv
- Temperature measurements in time-stamped CSV files

Files are flushed after writing to reduce the risk of data loss. This is what might have caused problems in the past where data was corrupted whilst writing to the SD card

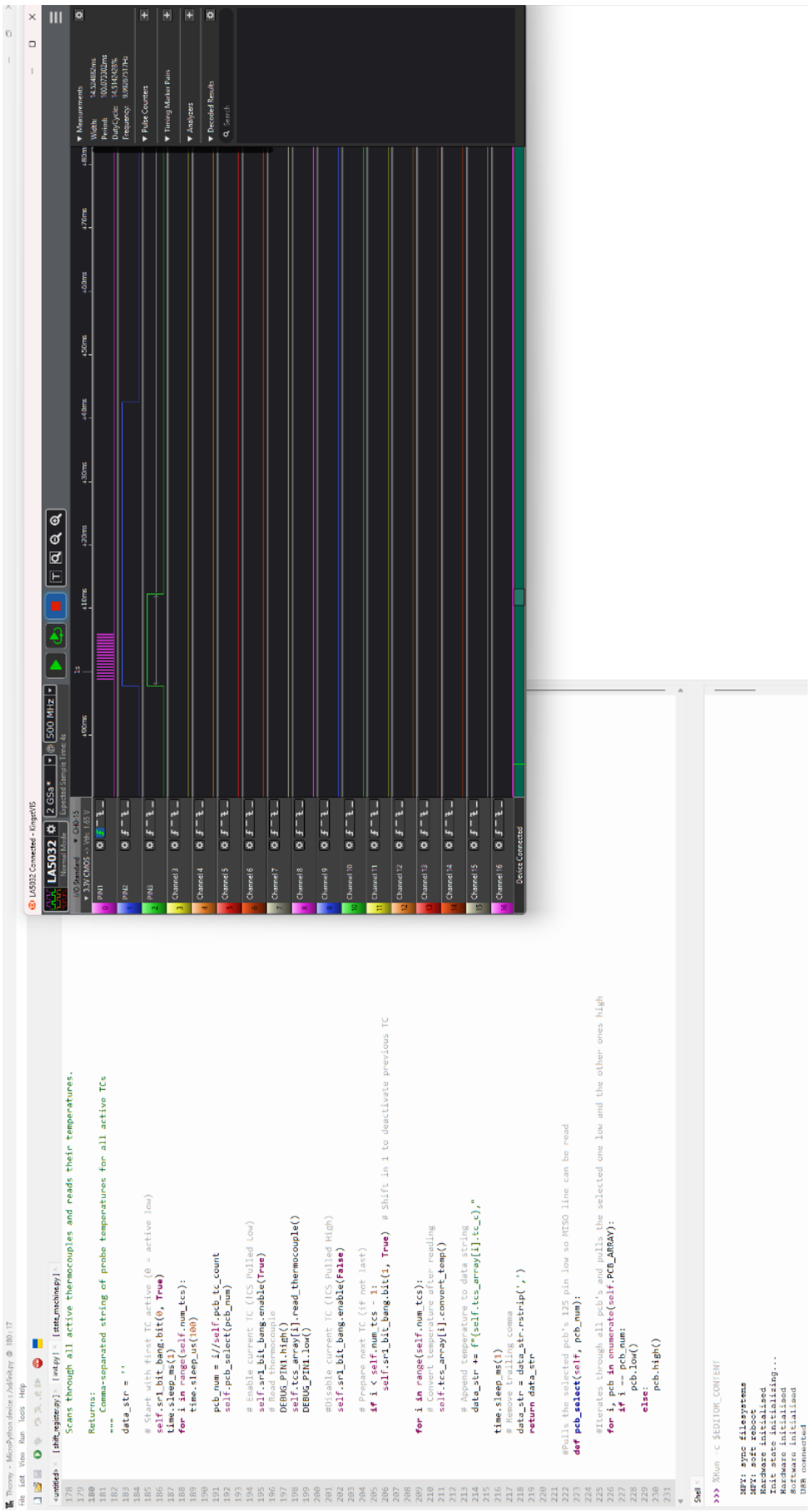
10. Summary

The Heat Cube system combines:

- A state machine for clear control flow
- A thermocouple manager to handle hardware
- Shift registers for scalability
- Timers for regular measurements
- UART and file storage for data output

This design keeps the system modular, scalable, and suitable for measuring temperature across many thermocouples.

Timing of reading each thermocouple



With this setup we can see that the time it takes to read 16 thermocouples is ~15ms, and total

operation of reading thermocouples, updating and generating strings for the thermocouples, writing to a file of the data and writing over UART takes ~50ms. But each read of a singular thermocouple is ~120us. This means that if 256 thermocouples data was desired to be read it would take ~30ms. But the highest resolution desired is 8 readings per second of 256 thermocouple which is a reading every 125ms which we are well below,

Errors still to solve:

Sometimes the opening and closing of the port does not sync that well on the JS side therefore to hard enforce it to close and open the read and write ports change a line of either js, html or css code and it will automatically update and correct itself.