Project specifications

The purpose of this lab is to design a circuit and a suitable software for the AT89 MCU that will implement a temperature logger which samples every second and stored in a buffer. The sampled temperature will then be displayed onto a LCD display and exported via the RS232 serial port for data analysis on Matlab.

Introduction

The first part was to calibrate the thermistor to display a temperature from an ADC input. This will prepare the program for data sampling. The design choices I will be implementing will include: sending data before the buffer array fills up to minimise data loss and clearing the sent data to stop the AT89 from slowing down. A lookup table was considered for the temperature calibration. However, due to how memory intensive it can be, it was not considered as a viable trade off.

Methodology

Project outline and design

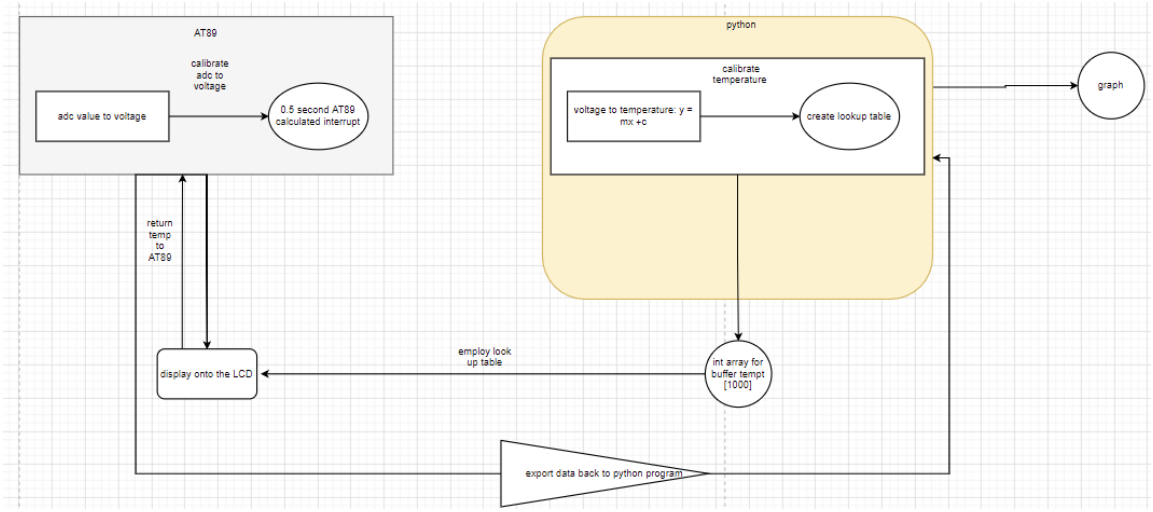

Figure 1: initial plan of the system

This plan (Figure 1) was first chosen to ensure that the AT89 does as little calculations as possible via lookup table. However, due to the ADC values of the thermistor a far more efficacious plan (Figure 2) was provided in order to compensate for potential packet loss. In addition, ADC values will be stored as integers and only be converted into strings before being sampled by the python program and turned into floats to be graphed. The plan only shows what will happen during the updating period.

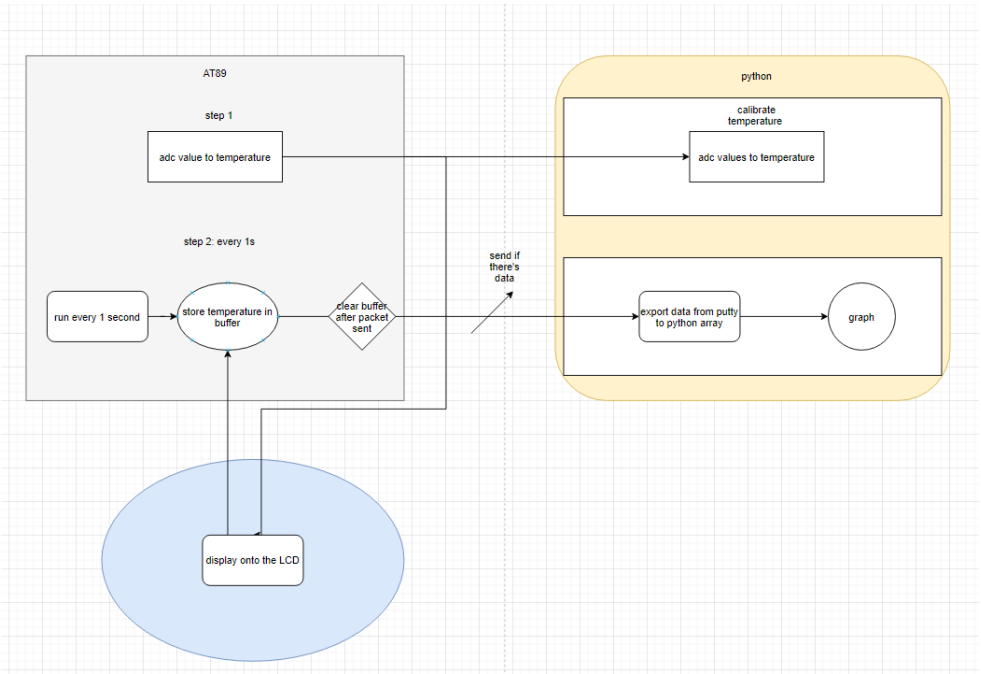

Figure 2: followed planned

Circuit set up

The first step of this project was to build a circuit. A voltage divider was used in order to measure a varying voltage (Figure 3). Which is needed from the thermoresistor (variable resistor that is temperature dependent). This circuit setup will allow the temperature dependent resistor to change the voltage which can provide the AT89 with an analogue input. This analogue input will be turned into digital input by the AT89 ADC convertor. Furthermore, this input will then be processed by the microcontroller to be displayed on to the LCD. Additionally, this input will then also be processed to be written onto the RS232 serial line to be later graphed by the python program.

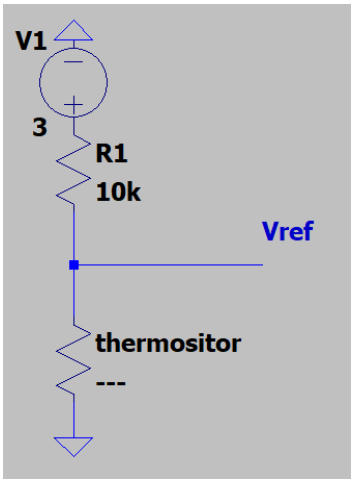

Figure 3: voltage divider

Temperature calibration

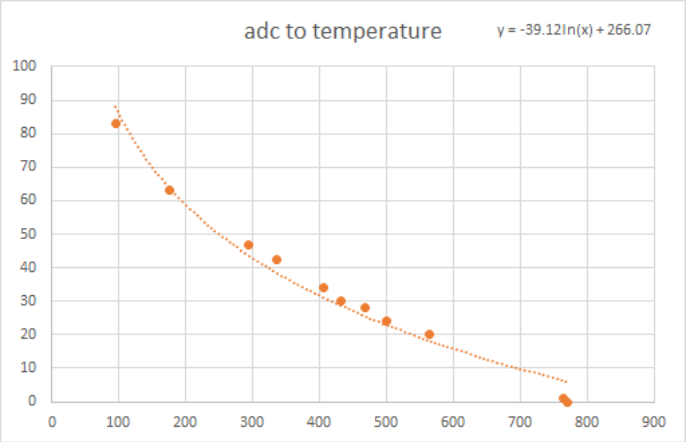

Figure 4: logothermic ADC values to temperature model

A logotherimic model was used to convert the readings from sampled adc to temperature as it looks like it fits that model. However, the logotherimic decay model has proved to be inefficacious in providing accurate readings. In addition to the less accurate readings, logothermic calculations will increase time complexity. As a result, a linear model was employed which yielded more accurate results. 15 values were taken between the temperatures of ~85°c and 0°c in order to improve the accuracy of this model. By using larger sample sizes, the model is less susceptible to outliers affecting the linear model.

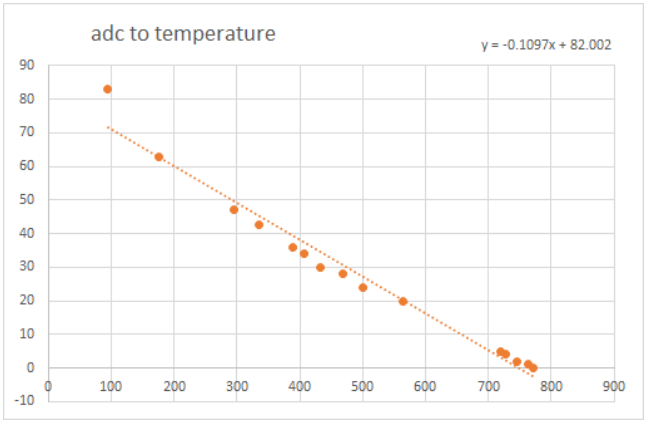

Figure 5: linear ADC to temperature model

$$equation\ 1:\ temperature\ =\ -\ 0.1097 sampleADC()\ +\ 82.002$$

As can be seen in Figure 5, there was an outlier of 83° C. Without the employment of a larger data set and linear model, it could really affect the accuracy of the digital thermometer like the model employed in Figure 4. A lookup table that follows this model was considered to speed up the calculations in exchange for memory space. However, this idea was scrap as it wouldn't provide accurate readings. In addition, the sampleADC() would still have to be modified to fit a

linear graph in order to work as a lookup table. This is because ADC inputs start at approximately 750. Temperature will then be rounded before turning it into an integer to minimise memory usage when stored in a buffer. However, ADC will be stored as strings before being written onto the serial I/O port.

Delay

I believe giving the AT89 1/10 of a second in order to execute all tasks at hand before allocating 9/10 of a second to be dead time. This can be achieved by adding 2 loops inside the preexisting main loop. First loop being the on part cycle while the second loop being the off part of the cycle.

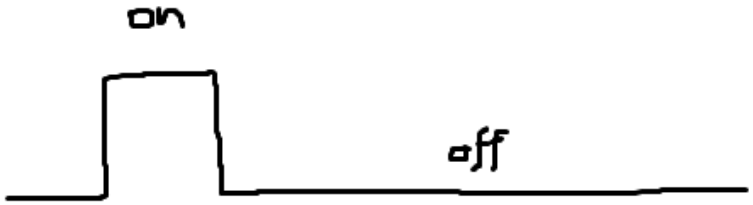

Figure 6: visual approximation of the on off period of the system

As displayed on Figure 6, the on part of the cycle will provide the system with sufficient amount of time to display and send a buffer through the serial IO port when a sufficient amount of data is collected. The off time will assist the system in saving power. Further measures could be taken to improve power efficiency. However, with the current configuration the system will save 90% of its power output while allocating sufficient resources to run the AT89. Timer 0 will be employed for this interrupt service routine (ISR) as an internal trigger is required.

$$equation\ 2\text{: } interrupt\ value = 2\ *\ Timer0\ frequency\ *\ duty\ off\ cycle\ *\ 1.024 = \sim 2\ *\ 2000\ *\ 0.9\ *\ 1.024 =$$

The timer0 frequency was observed through an oscilloscope and approximated to be 2000Hz (as shown in Figure 7); this frequency needs to be doubled as the system is a double edge trigger. An 8 bit timer would overflow every 256 counts. Since the timer is running at 12MHz/12 = 1MHz. We can confirm that the clock pulse is 1us and the timer would overflow every 256us. As a result, approximately every 1ms will pass 4 interrupts. To compensated for this 256 * 4 = 1024us = 1.024ms was multiplied into the equation to calculate the value that will allow the interrupt to trigger every second; thus updating the system every second.

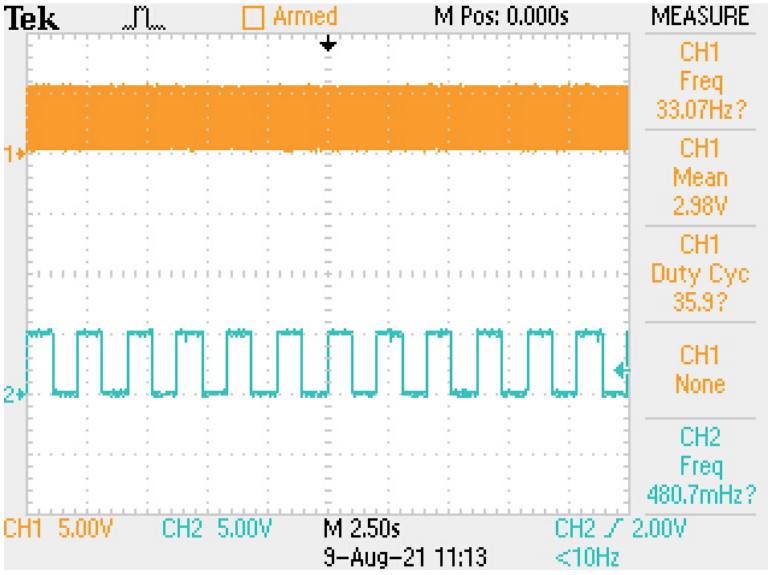

Figure 7: Orange: timer 0 frequency. Blue: ISR.

As observed in Figure 7, the blue duty cycle shows the frequency of ISR. Keep in mind that the ISR is both positive and negative edge triggered which means that it will trigger twice every duty cycle. Additionally, the blue duty cycle was approximated to have a frequency of 480mHz which is equivalent to 2.0833s meaning that the thermometer will update every 1.04s.

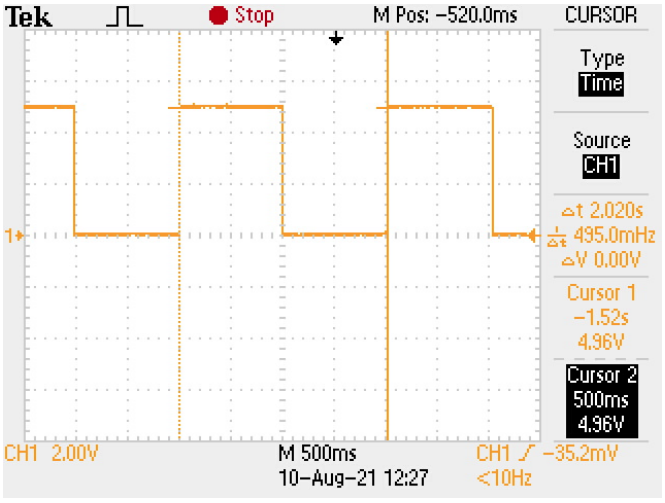

Figure 8: ISR duty cycle using interrupt value of 3600

Through further analysis of the timer 0 using an oscilloscope (Figure 8) the actual frequency timer 0 runs on was 1954 Hz. Substituting this into the equation provided us with an offTime value of 3600. This proved to be more accurate by updating the system every 1.01s instead of the previous 1.04s when the offTime had a value of 3686. As a result, improving the system by 30ms

$$equation\ 3:\ interrupt\ value\ =\ 2\ *\ Timer0\ frequency\ *\ duty\ off\ cycle\ *\ 1.024\ =\ \sim 2\ *\ 1954\ *\ 0.9\ *\ 1.024\ =$$

Sending buffer

It is ideal that a temperature display should also be sent as soon as it's display to free up as much buffer space as possible. Sending data as integers will also be faster compared to sending data as floats requires more processing power. To achieve this data received from python will be computed using the same equation used in the 8051 code to mimic the same temperature reading.

It has been discovered later that the AT89 only allows string to be written into the serial port via the serialWriteLine() function. As a result, the string version of the ADC value will be sent instead of the integer. E.g. 750 is sent as "750". This means that each element of the buffer will be turned into a string before being sent to python before being removed from the buffer to allow new elements to replace the previous one.

In terms of memory storage, an integer would use 4 bytes of data while a string would use 1 byte per ASCII character. If we were to use the buffer as an array of strings it would be no different in terms of memory storage. E.g. 750 and '750,' would have the memory usage.

However, the advantage of keeping the buffer as an array of integers would mean that it can easily be cleared from an array instead of having to clear each char element in a string array. As a result, it would mean less time complexity. So it was decided that the ADC values will be stored as integers in the array buffer and only turning it into strings before being sent to python and removed from the buffer. A visual representation can be seen on Figure 9

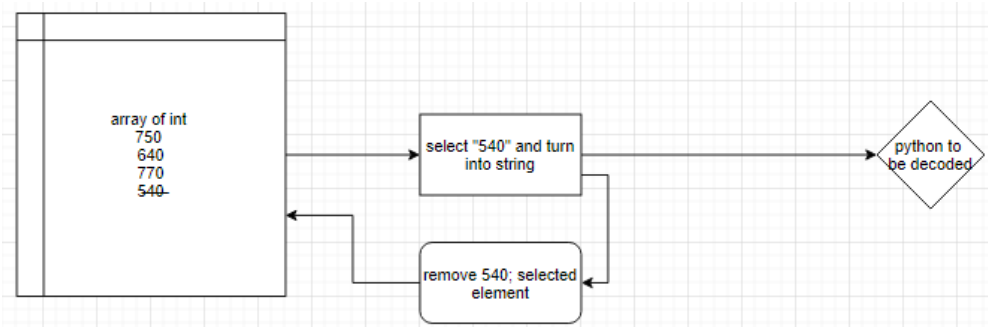

Figure 9: visual representation of the buffer element sending process

If there's more than 1 in the queue, it will also prioritise sending all elements of the buffer to python during the 1s interrupt service routine. This ensures data integrity and will prevent incorrect sampling. For example if the RS232 was disconnected from the AT89 this would mean that the MCU would be storing data in the buffer. When it connects to the serial port, the AT89 will start a catch up protocol to ensure the unsent buffer element is sent to reduce the chance of insufficient memory storage.

Data analysis and signal processing

When python receives the ADC value as a string buffer element it will turn the element into a float so that it can be processed through the ADC value to temperature model (equation 1). This ultimately removes the extra calculation needed to be processed by the AT89 “from float to string (5 char; e.g. 23.45 to ‘2’, ‘3’, ‘.’, ‘4’, ‘5’).

Allowing python to do the majority of the calculations will also be advantageous as the processing power of the computer is superior to the processing power of the 8051. The python program will do the realignment of data instead AT89 to reduce the burden. To achieve this the python will find the ADC value in the python array and make it a reference point for the rest of the AT89 buffer to rejoin (see Figure 10). This method will compensate for the packet loss in the event of a serial disconnection.

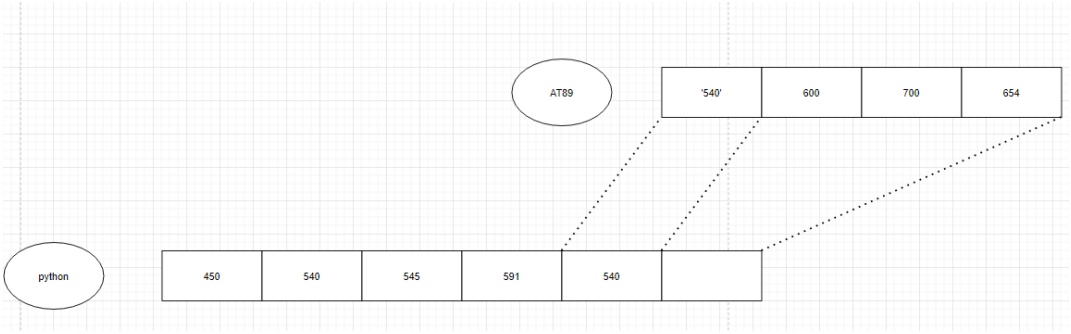

Figure 10: catch up protocol

Data is updated every second onto a live temperature vs time graph. 1Hz sampling frequency was chosen instead of Nyquist’s sampling frequency (2x of highest frequency; in this context 2x of interrupt service routine) as sampling twice as much as the update period would cause dips like the ones shown in Figure 11. As 0 ADC value is being sampled this gittering will form instead of the ideal smooth graph.

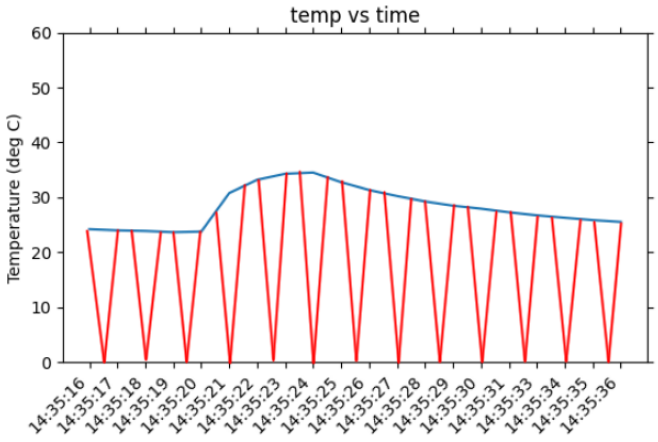

Figure 11: replica of 2Hz sampling frequency (2x ISR frequency) blue being ideal and red being 2Hz sampling

A major issue that was discovered was forgetting to convert the temperature from a string to a float which causes a graph that doesn't follow the order of values as shown in Figure 12. However, this was fixed by changing the ADC values to floats before storing it as an element of an array.

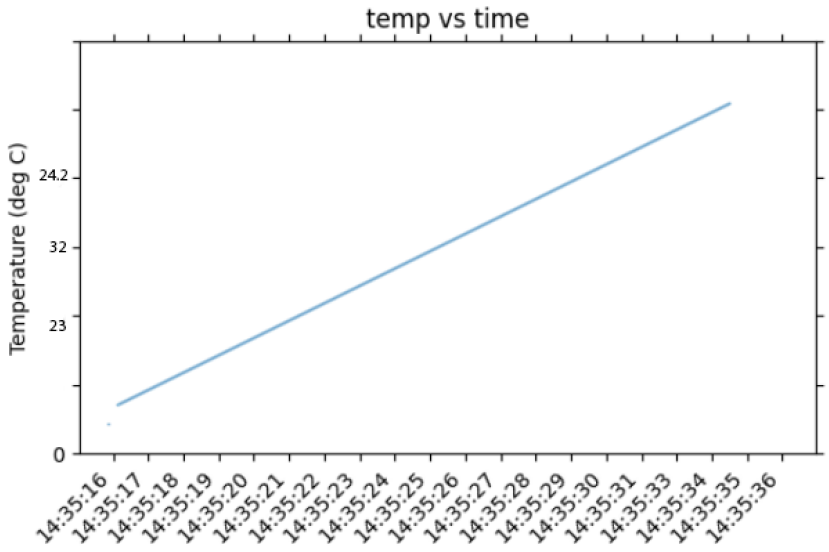

Figure 12: string array error

The final product resulted in a live graph that scrolls when new data is produced at the same time the thermometer is being updated (Figure 13). Although the final product does meet the specification deployed by this lab, After updating the graph for a while the graph can slow down due to the print overflow which was fixed by removing all print statements from the program

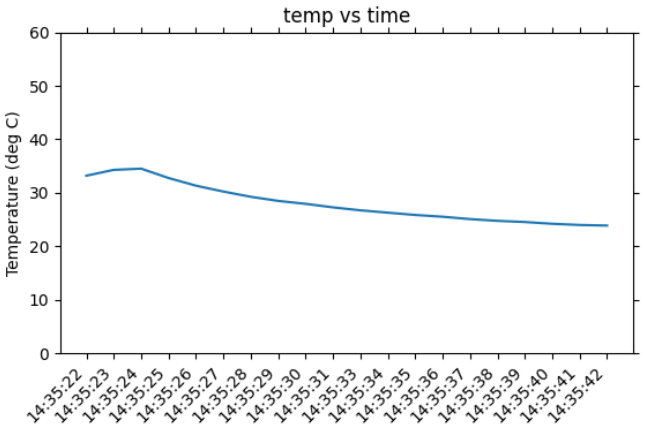

Figure 13: final graph and result

Discussion and conclusion

Although Figure 2 plan was followed through, implementing the catch up protocol was very tricky and puts a lot of processing burden onto the AT89 which is unideal, (suddenly prioritising a huge amount of packets being sent). If this project was to be remade, I would probably use a better microcontroller with a higher clock speed. In addition, a better clock speed would also mean a more accurate thermometer updating time. E.g. 1.000001s instead of 1.01s. A better Microcontroller would also mean that data integrity could be done on the MCU instead of relying on python to fix. Another thing that could be improved is being able to write integers onto the serial lines instead of strings. This will also improve the time complexity of the program.

Something that went well was employing a linear model for the ADC to temperature values instead of the logothermic model. This improved the accuracy of the digital thermometer by treating the high value as an outlier which could've been caused by the material in the thermistor. Additionally, the ISR frequency was well done as it didn't use as much power as it could have. If we were to leave the clock ticking while doing nothing there would be more power drawage. As a result, increasing energy efficiency.

In conclusion, the digital thermometer project completed with python and C program has proven to be quite challenging. Although it does perform well given the processing power it was provided; accurate timer update, minimal packet loss and accurate readings. There's still a few more steps to be made to reach an acceptable product on the market.

Appendix

Due to level 4 lockdown I am unable to retrieve my code for both the python program and the 8051 program . However, this one code below shows one of the progress I've made on the 8051 program (not final).

```
/***  Include Files  ******************************************************/

#include <t89c51ac3.h>
#include "phys340libkeil.h"
#include <string.h>
#include <stdio.h>
#include <math.h>

const char BlankString[] = "                                ";
char outputText [33];
char * txt = "C";
int timecollab = 0;

float adcValue = 0;
unsigned int ticks = 0;
//everything goes to 1954
//unsigned int onTime = 400; //10ms 1/10 of a second on time negative and positive edges 1000ms * 0.75 / 2 positive
unsigned int offTime = 3686; //Timer0 off time 4000 ticks; * 0.9 *1.024
unsigned int bufferSize = 10; /**if bufferSize = buffer.size() then update it as well*/

/***  Function to sample an analog voltage  ********************************/

void myIntHandler(void) interrupt 1
{
  ticks++;
  P1_3 = ~P1_3;
}

void initTimer(){
  IEN0 = 0x82; /**enable all interrupt bit timer 0 overflow interrupt enable bit 1000 0010*/
  TMOD |= 0x02; /**mode 2 8 bit auto reload 0000 0010* don't override tmod*/
  TH0 = 0x00;

  TCON |= 0x10 ; //start timer 0 doesn't override tcon register level trigger
}
```

```
float recalibration(){
  float y;
  y = sampleADC() * -0.1097 + 82.002;
  adcValue = y;
}

/**display Temperature */
void displayTemp(){
  clearLCD();
  recalibration(); //calibrates the system to show adc to temp value
  sprintf(outputText,"%d %d",   sampleADC(), timecollab);
  setLCDPos(0);
  writeLineLCD(outputText);

}

/**  Main Function  ******************************************************/


void main()
{
  initLCD();
  initTimer();
  initSerial(1200); //1.2kHz baud rate

  while(1){
    if(ticks >= offTime){ //this is new delay maybe make this on time?
      //P1 = 0x01; /** interrupt on*/
      displayTemp();//displays temperature reading
      writeLineSerial(outputText);
      timecollab++;
      ticks = 0;
      P1_4 = ~P1_4;
    }


  }

}
```