

Industrial Training Project Report

Firmware Design and Implementation of 8-Channel Digital Temperature Meter using LPC1768 ARM Cortex-M3



Research Project undertaken by:

Mukul C. Mahadik (17469) and Vivek Gusain (17445)
B. Tech 4th Year
Electronics and Communications Engineering,
National Institute of Technology, Hamirpur

Under the guidance of:

Dr. Joby Antony
B.Tech E.E MS (USA) PhD(IIT)
Former Visiting Scientist CERN
Engineer/Scientist F
Inter-University Accelerator Centre (IUAC), New Delhi

Internship Period:

1st May 2020 – 14th August 2020

**Submitted in fulfilment for
Industrial Training Viva**

ECD-419

B.Tech 4th Year

Semester 7 (2020)

Submitted To:

Dr. Rohit Dhiman
Assistant Professor
E&CED, NIT Hamirpur

Date of Submission: 05/11/2020

Table of Contents

Sr. No.	Title	Page No.
1.	Introduction	3
2.	Aim and Objectives	4
3.	Hardware Specifications	5
4.	Software Specifications	11
5.	Algorithms Developed	12
A.	C++ Libraries Used	12
B.	Primary Algorithm	13
i.	Data Display	14
ii.	Continuous Menu Display	14
iii.	Continuous Sensor Reading	16
iv.	Channel/Sensor Select	16
v.	Curve Data Entry	17
vi.	Data Interpolation	17
vii.	Data/Settings Save and Load	17
viii.	RPC Connectivity	18
6.	Results	19
7.	Conclusion	21
8.	Acknowledgment	22
9.	Bibliography	23

1. Introduction

This report summarizes the summer research internship project based on the **firmware design and implementation of an automated embedded device** as well as an insight into the practical and real-time applications of **digital temperature sensor devices** for **extremely low sub-zero temperatures**.

An **embedded system** is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger system. It is embedded as part of a complete device often including electrical or electronic hardware.

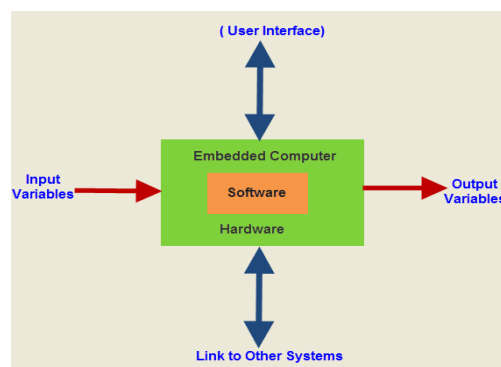


Fig. 1: Embedded System Components

The embedded device design we worked on receives or **fetches the voltage or resistance sensor readings** from up to **eight different channels**. Each of these eight channels may have different types of sensors depending on the requirements. These readings are then interpolated or converted to the appropriate temperature readings in the adjusted scale as per the pre-coded lookup table. This data is displayed on the screen as the default display that keeps **refreshing after an interval of 5.0 seconds** to show 4 channel sensor readings at a time alternatively.

As a part of the **User Interface (UI)** of the embedded system, the user can operate the device manually via the interfaced **4x3 keyboard** and view the corresponding menu options, data display by means of the **16x2 LCD display**. Additionally, the mbed LPC1768 microcontroller can be accessed via USB as it has a built-in micro USB interface. It can also be interfaced to provide **connectivity via remote procedural calls (RPC)** and Ethernet implemented using the standard RS232 port. The device can then be controlled or operated via the serial communication link via the PC terminal.

The data acquired by the device is continuously being logged and stored in a binary file with the **.bin** extension which is automatically converted to a human-readable text document file with the **.txt** extension. This log file can then be viewed on connecting the microcontroller to a client PC via the micro USB connection of the mbed board to the PC. Manual analysis can then be performed using the output data from the generated log file.

GitHub Link for the code: [https://github.com/MukuFlash03/MBED Temp Meter IUAC](https://github.com/MukuFlash03/MBED_Temp_Meter_IUAC)

2. Aim and Objectives

The **cryogenic temperature range** has been defined as from **-150 °C (-238 °F) to absolute zero (-273 °C or -460 °F)**, the temperature at which molecular motion comes as close as theoretically possible to ceasing completely.

Now, globally there exist many cryogenic labs and test facilities with extremely advanced technological devices and equipment that have been working with such low temperatures. These labs deal with low temperature substances and materials such as liquid helium and liquid nitrogen refrigerants, superconducting materials such as niobium which need to be handled with extreme precaution and under strict observations.

At **IUAC, India's premier research centre for nuclear accelerators, cryogenics is an important department** where the aforementioned facilities and research work is carried out. The **recent accelerator projects** include the heavy ion RF-superconducting LINAC booster for 15UD Pelletron Accelerator as well as Liquid Helium and Liquid Nitrogen Plant facilities.

The aim of the research project was working towards the development and implementation of an **automated Digital Temperature Meter to record as well as compute temperatures near absolute zero** (0 Kelvin or -273° C) which are difficult to measure with ordinary sensors, let alone be recorded in an organized manner, for future analysis.

Further, another reason was to emphasize on the **production of locally developed products** especially in the times of the global pandemic where the entire economy and imports have been affected. Hence, dependence on native products has been a major motivating factor behind this project.

The objectives of the research work are as follows:

- i. To interface various hardware components as per the low temperature conditions.
- ii. To design flow-charts and model the temperature meter device.
- iii. To develop and implement various software functionalities for the device.
- iv. To ensure client connectivity and data accessibility to the user.

3. Hardware Specifications



Fig 2: Temperature Meter Front Panel

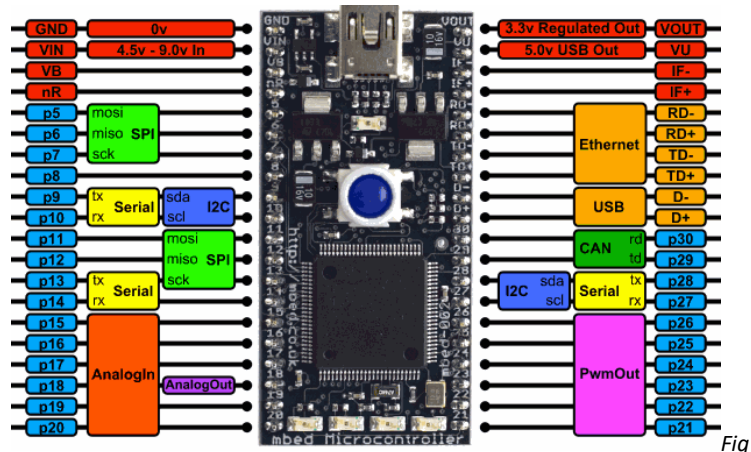


Fig 3: Temperature Meter Back Panel



Fig 4: Temperature Meter Wiring

A. Mbed LPC1768 Microcontroller



5: Mbed LPC1768 Pin Diagram

ARM Mbed OS is an open source embedded operating system designed specifically for the "things" in the Internet of Things. It includes all the features you need to develop a connected product based on an ARM Cortex-M3 microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.

The **mbed Microcontroller is an ARM processor**, a comprehensive set of peripherals and a USB programming and communication interface provided in a small and practical DIP package.

The mbed **32-bit ARM Cortex-M3** NXP LPC1768 Microcontroller in particular is designed for prototyping all sorts of devices, especially those including Ethernet, USB, and the flexibility of lots of peripheral interfaces and FLASH memory.

It includes 512KB FLASH, 32KB RAM and lots of interfaces including built-in Ethernet, USB Host and Device, CAN, SPI, I2C, ADC, DAC, PWM and other I/O interfaces.

B. 16x2 LCD Display



Fig. 6: Displaying Temperature



Fig. 7: Displaying Channel Selection

An LCD (Liquid Crystal Display) screen is an electronic display module that uses liquid crystal to produce a visible image. LCD modules are very commonly used in most embedded projects, the reason being its cheap price, availability and programmer friendly.

A 16x2 LCD means it can display **16 characters per line** and there are **2 such lines**. In this LCD **each character** is displayed in **5x8 pixel matrix**.

This LCD has two registers:

- i. **Command register** stores various commands given to the display.
- ii. **Data register** stores data to be displayed.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The inbuilt library simplifies this for you so you don't need to know the low-level instructions. Contrast of the display can be adjusted by adjusting the potentiometer to be connected.

C. 4x3 Keypad



Fig. 8: Keypad Connections



Fig. 9: 4X3 Keypad

Keypad is most widely used as an input device generally used with microcontroller and microprocessor based devices. Metric keypads use in various types of embedded applications like door lock system, phones, as input devices etc.

The rows are the output lines while the columns are the input lines. Here R1, R2, R3 & R4 refers to row 1, row 2, row 3 & row 4 respectively and C1, C2 & C3 refers to column 1, column 2 & column 3 respectively.

Let us understand the logic to **detect the key and print numbers** of the first row having **three columns** and **one row**. So there are three possibilities:

Column 1 key pressed Column 2 key pressed Column 3 key pressed

If no key has been pressed then all columns will remain HIGH and if the key has been pressed then its corresponding column will give a LOW signal (because it shorted to ground via row).

If column 1 key has been pressed, as row R1 is output type and C1 is input, then microcontroller will read the logic state through C1 line. **Row 1** should be **LOW**, so that the microcontroller can detect logic. But if we have multiple rows then we should give logic **HIGH to them (R2, R3, R4)** to distinguish between R1 and other rows.

R1 → LOW R2 → HIGH R3 → HIGH R4 → HIGH

Similarly for all the other rows and columns combinations each of the keys has been outputted.

The following keypad functionalities have been implemented by us:

- i. **'2' - Curve Data Entry** : This key allows the user to proceed to defining their custom sensor specifications by facilitating them to enter the sensor data values for curve data interpolation.
- ii. **'4' - Unit Selection** : This key allows the user to select the output temperature unit from Kelvin (K), Celsius (°C).
- iii. **'6' - Sensor Selection** : This key allows the user to set the desired or interfaced sensor for the specific channel.
- iv. **'8' - Save Data** : This key allows the user to save the current data configurations and updates the log file.
- v. **'*' - Escape, Toggle, Minus** : This key serves multiple functions such as Escaping on double press, toggling through options in menus, entering the minus (-) sign or the decimal point (.) in curve data entry mode.
- vi. **'#' - Enter** : This key allows the user to confirm the current selection in a menu or confirm the entered values in curve data entry mode.

D. RS232 Ports

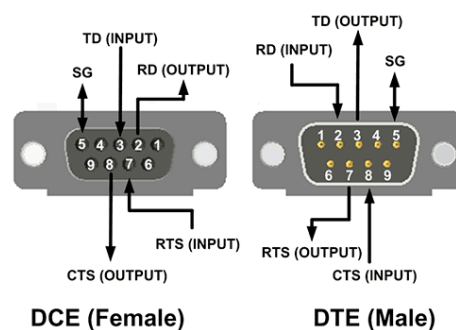


Fig. 10: RS232 Ports

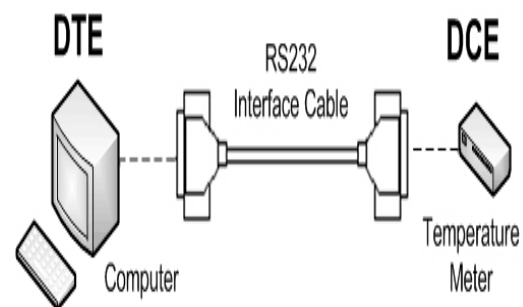


Fig. 11: RS232 Block Diagram

In telecommunication, the process of sending data sequentially over a computer bus is called as serial communication, which means the data will be transmitted bit by bit. While in parallel communication the data is transmitted in a byte (8 bit) or character on several data lines or buses at a time.

RS232 is a **“recommended standard” protocol used for serial communication**. The standard defines how computers (Data Terminal Equipment - DTEs) connect to modems/devices (Data Communication Equipment or DCEs) to allow serial data exchange between them.

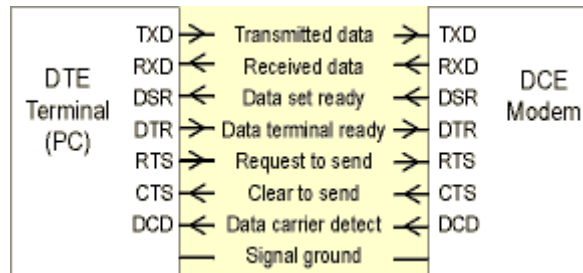


Fig. 12: RS232 Pin Functionalities

Both **hardware as well as software handshake arrangements** are available. These are used mainly so that the computer can prevent the device from sending data when it is not able to receive it or the device can prevent the computer from sending data when it is not ready for it.

This is managed via the Clear-to-Send (CTS) and the Request-to-Send (RTS) signals in case of hardware handshakes whereas Xon/Xoff character-based handshaking is a software protocol that is often used to control data flow.

E. DT470, PT100 Temperature Sensors



Fig. 13: DT470 Sensor



Fig. 14: PT100 Sensor

i. DT470 Sensor

- The DT470 silicon diode cryogenic temperature sensors incorporate remarkably uniform sensing elements that exhibit precise, repeatable, monotonic temperature response over a wide range.
- The elements are mounted into rugged, hermetically sealed packages that have been specifically designed for proper thermal behavior in a cryogenic environment.
- The result is a family of cryogenic sensors with temperature characteristics so predictable, tightly grouped, and stable that the sensors in most applications are routinely interchangeable with one another.

ii. PT100 Sensor

- A Pt100 is a sensor used to measure temperature which falls into a group called Resistance Temperature Detectors (RTDs). RTDs require a small current to be passed through in order to determine the resistance.
- The first part, Pt, is the chemical symbol for Platinum and this shows that the sensor is Platinum-based. The second part, 100, relates to the resistance of the device at 0°C i.e. 100Ω.
- Although Platinum is a precious metal and therefore very expensive it gives the greatest linearity and stability of any other material.

F. MCP3208 8 Channel 12-bit ADC IC



Fig. 15: MCP3208 ADC IC

The MCP3208 12-bit (ADC) combines high performance and low power consumption in a small package, making it ideal for embedded control applications.

It can operate on both 3.3V and 5V and hence it can be used with 5V microcontroller as well as with 3.3V systems like the Raspberry Pi/Arduino.

The MCP3208 is an **8-Channel 12-bit Analog-to-Digital Converter (ADC)** IC, so it can measure 8 different analog voltage with a resolution of 12-bit. It measures the value of analog voltage which is then converted to a digital value from 0-2047. Then using SPI communication protocol we have to send the control bit data for selecting the channel number from which the ADC value has to be obtained and the IC will reply us back with the value.

This **ADC conversion formula** is given below:

$$\text{Analog Voltage Measured} = \text{ADC Reading} \times \frac{\text{System Voltage}}{\text{Resolution of ADC}}$$

Applications for the MCP3208 include data acquisition, instrumentation and measurement, multi-channel data loggers, industrial PCs, motor control, robotics, industrial automation, smart sensors, portable instrumentation and home medical appliances.

We have used the MCP3208 IC to read in the input sensor voltages from each of the eight different channels and pass on these analog values to the microcontroller in a digitised form as per the resolution of the IC, which is 12-bit = $2^{12} = 2048$. Thus the output value ranges from 0-2047 for each channel sensor voltage input.

4. Software Specifications

A. mbed Online Compiler

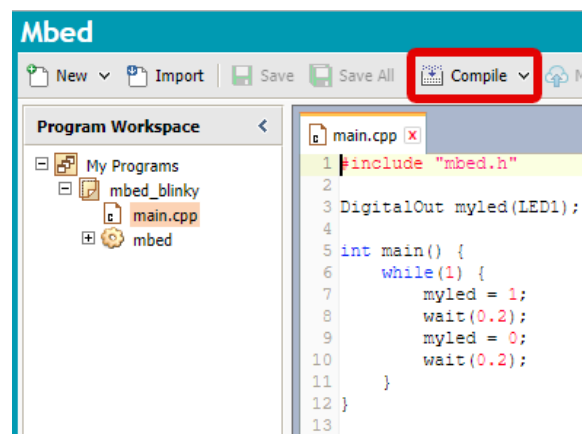


Fig. 16: MBED Online Compiler IDE

The **mbed online Compiler** lets you write programs in C++ and then compile and download them to run on the mbed NXP LPC1768 microcontroller. The online compiler platform has tons of libraries and example projects and codes for other developers to view and learn from thus facilitating a cooperative learning virtual environment.

You don't need to install extra setup on your computer because it is web IDE. This also **enables portability and ease of accessibility of the code anywhere in the world** along with synchronous updates to the code.

We ourselves have benefited from this feature as we were operating the embedded system **simultaneously from three locations – New Delhi, Dehradun, Pune.**

B. Tera Term Virtual Terminal (VT)

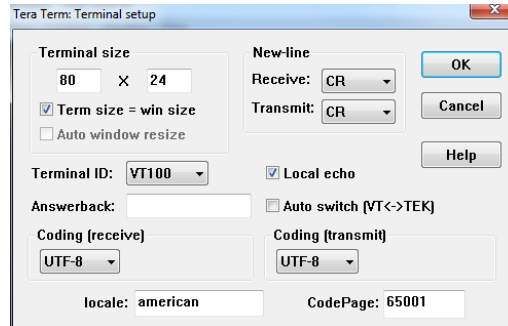


Fig. 17: Terminal Configuration



Fig. 18: RPC with Terminal

Tera Term is an open-source, free, software implemented, **terminal emulator** (communications) program. It emulates different types of computer terminals. It supports telnet, SSH 1 & 2 and serial port connections. It also has a built-in macro scripting language and a few other useful plugins.

This software was of great help to us in **debugging the firmware development of the temperature meter** as we had to check the various functionalities on a virtual terminal on the client PC before finalizing the features and wiring the hardware permanently on the actual device.

5. Algorithms Developed

A. C++ Libraries Used

Many libraries provided with the software and a large number of libraries developed by people from around the world are available for free. The functions implemented in these libraries can be used for quick development purposes.

This allows the user to use certain interfaces or modules without having in-depth knowledge about the internal structure or working. The point is that lack of technical or coding knowledge should not become an obstacle for development or creative thinking.

The various C++ libraries used in developing the firmware along with their purpose are mentioned below:

- i. **mbed.h:** The mbed Library provides an API-driven approach to coding that eliminates much of the low-level work normally associated with MCU code development. The **core mbed Library** supports the main LPC1768 peripherals, and the libraries already contributed by the mbed design community include USB, TCP/IP, and HTTP support.
- ii. **Keypad.h:** Keypad is a library **for using matrix style keypads** with microcontrollers and microprocessors and supports multiple keypresses. It was created to promote Hardware Abstraction. It improves readability of the code by hiding the pinMode and digitalWrite calls for the user.
- iii. **SerialRPCInterface.h:** The **RPC interface** can be really useful because it makes it possible to interface with mbed without having to write any of the **communication** code. When a serial communication is received it will pass the **commands** to the RPC and return the response.

It is made up of 3 parts:

RPCFunction - A class which can be used to call user defined functions over RPC.

RPCVariable - A class which allows you to read and write the value of a variable on mbed using RPC.

SerialRPCInterface – This class sets up RPC over Serial Import library.

B. Primary Algorithm

Listed below are the major functionalities implemented in the design of the device. The algorithms for each of the following tasks are described in detail below:

- i. Data Display
- ii. Continuous Menu Display
- iii. Continuous Sensor Reading
- iv. Channel/Sensor Select
- v. Curve Data Entry
- vi. Data Interpolation
- vii. Data/Settings Save and Load
- viii. RPC Connectivity

i. Data Display:

This deals with the logic used to display all the data to be presented to the user via the user interface display device, which in this case is the 16x2 LCD Display.

The common step for each display sub-functionality includes **setting the chip select (cs)** to 0 to enable the LCD device which is **active LOW** and then disable it by setting **cs = 1** once the data has been displayed.

```
182 void displayinitialize()
183 {
184     int counter;
185     spi.format(16,3);
186     spi.frequency(1000000); // spi freq 1 MHz
187     cs = 0;
188     spi.write(0xF830);
189     cs=1;
```

The first step is initializing the LCD device by **setting the bits per data value to 16** and the **transfer data mode to 3**, during LOW SPI Clock (SCK) with a frequency of 1 MHz.

```
214 void displaygotofirstchar( int lineno )
215 {
216     int counter1,counter2;
217     if (lineno == 1)
218     {
219         cs = 0;
220         spi.write(0xF880);
221         cs=1;
```

The next operation involves **resetting the cursor** to the start of each line. This is done by sending a special preset hexadecimal instruction - "**0xF880**" for line 1 and "**0xF8C0**" for line 2, which instructs the device to

perform this task.

```
201 void displaychar(unsigned int mychar)
202 {
203     unsigned int myint= 0x0;
204     cs = 0;
205
206     myint= (0xFA00 | (mychar));
207     spi.write(myint);
208     cs=1;
```

This function is then used to **display a character** onto the screen by performing bit manipulation using a preset hexadecimal instruction code - "**0xFA00**" before writing using the SPI protocol.

```
237 void writestringspi_line1( char *s )
238 {
239     char *mystr;
240     mystr = s;
241
242     while (*mystr)
243     {
244         displaychar(*mystr);
245         mystr = mystr+1;
246     }
```

This function is used to take the string to be displayed on the screen and parses it character by character by sending it to the displaychar().

```
321 void blankline2()
322 { displaygotofirstchar(2) ;
323   sprintf(display," ");
324   writestringspi_line2(display);
```

This function is used to blank the LCD display whenever the data on the screen is to be cleared by outputting an empty string of 16 spaces for desired line.

ii. Continuous Menu Display:

This specifies the algorithm used to keep the device running continuously without human intervention and provide the normal operational functionality on user interaction.

```

1143 spi2.format(8, 0);
1144 spi2.frequency(100000); //0 // spi2 speed reduced by 10
1145
1146 load_saved_configurations();
1147
1148 //      r0  r1  r2  r3  c0  c1  c2  c3
1149 Keypad keypad(p21, p22, p23, p24, p25, p26, p27, NC);
1150 keypad.attach(&cbAfterInput);
1151 keypad.start(); // energize the columns c0-c3 of the keypad
1152
1153 displayinitialize();

```

First, the 8-channel ADC MCP3208 is initialized on its specified SPI connection. Next, the last used configurations and data are loaded back from the flash memory of the microcontroller.

Then, the keypad and the display are initialized and its connections to the MCU are specified.

Next, comes the main menu loop which continuously keeps running and **refreshes** the screen temperature sensor **values every 5.0 seconds** and keeps the device available for user interaction.

```

1171 while (1)
1172 {
1173     wait(5);
1174     continuousreadsensor();

```

This is followed by the input entered by the user via the interfaced keypad, which is done by a **keypad interrupt routine**. This input value is then passed to a **switch()** block as per the menu options for the primary functionalities of the device.

```

1176 if (Index > -1)
1177 {
1178     input = Index + 1;
1179     switch (input)

```

```

1181 case 2:
1182     //PC.printf("Calling sensor3curveDataEntry(
1183     id = '2';
1184
1185     do
1186     {
1187         //PC.printf("Enter no. of values (must
1188         writestringspi_line1(display);
1189         curveVal = (int)curveDataentry("n",0);
1190         if (!flag) break;
1191     }
1192     while (curveVal <= 1 || curveVal > 100);
1193
1194     for (i=0;i<curveVal;i++)
1195     {
1196         if (flag)
1197         {
1198             dx[i]=curveDataentry("x",i);
1199             if(!flag) break ;
1200             // PC.printf("x returned is = %f\n",
1201             dy[i]=curveDataentry("y",i);
1202             if(!flag) break ;
1203             //PC.printf("y returned is = %f\n",
1204         }
1205     }

```

Next, the first menu option - **“Curve Data Entry”** is designed such that the user is prompted to enter the number of points, say ‘n’, followed by ‘n’ pairs of (x,y) values for the custom sensor.

```

case 8:
    save_current_configurations();
    break;
case 6:
    // PC.printf("Pressed 6 to enter
    id = '6';
    sensortypeselect();
    break;

```

Lastly, the two menu options - “Setting the sensor for each channel” and “Saving the current configurations” are designed by means of their corresponding functions.

iii. Continuous Sensor Reading:

This specifies the logic designed to continuously keep updating the display with the temperatures recorded from each of the channels. The current device configurations are for two sensors namely, **DT470 and PT100, as well as a custom sensor – CCS3 sensor.**

Hence, the first function call is to the interpolation function which serves to calibrate the values for the custom sensor.

```
1009 void continuousreadsensor()
1010 {
1011     interpolateCCS3(r,T);
1012     for (int j = 1; j<9; j++)
1013         sensorVoltage[j] = readchannel(j-1);
1014     for (int j=1; j<9; j++)
1015     {
1016         if (channelsensortable[j] == 1)
1017             sensorTemp[j] = convert_D_to_T(sensorVoltage[j]);
1018         else if (channelsensortable[j] == 2)
1019             sensorTemp[j] = convert_P_to_T(sensorVoltage[j]);
1020         else if (channelsensortable[j] == 3)
1021             sensorTemp[j] = ccs_V2T(sensorVoltage[j])
```

This is followed by reading in the values from each of the sensors via **readchannel()**. Next, the analog voltage values read in from each of the sensors are passed to their respective sensor functions that return a digitized temperature value as the output.

```
1033     if (channelswitchflag == 0)
1034     {
1035         displaygotofirstchar(1);
1036         sprintf(display,"1 %5.1fK %5.1fK", sensorTemp[1],sensorTemp[2]);
1037         writestringspi_line1(display);
1038         sprintf(display,"3 %5.1fK %5.1fK", sensorTemp[3],sensorTemp[4]);
1039         writestringspi_line2(display);
1040     }
```

These digitized temperature values are then displayed on the LCD display alternating between the first four channels and the last four channels with a time interval of 5.0 seconds defined in main().

iv. Channel/Sensor Select:

This specifies the algorithm designed for setting the sensor for each of the 8 channels. It comprises two parts – **choosing the channel and then the sensor for that channel.**

The first part involves choosing the channel which is again done by a switch menu configuration, providing the options for each of the 8 channels along with a **toggle key ‘*’ [Case 10] to cycle through the options.** Finally, the **‘#’ key [Case 12] is used to confirm channel selection and go to sensor selection menu.**

```
switch (input)
{
case 1:
    displaygotofirstchar(2);
    sprintf(display,"CH1<*><ENTER>");
    writestringspi_line2(display);

    channelselected = input;
    togglecounter = input;
    break;
```

```
case 10:
    togglecounter++;
    displaygotofirstchar(2);
    sprintf(display,"CH%d<*><ENTER>",togglecounter);
    writestringspi_line2(display);

    channelselected = togglecounter;

    if (togglecounter>=8)
        togglecounter=0;
```

```
case 12:
    selectSensor();
    break1 = 1;
    break;
```

Similarly, the switch case is used for the **sensor selection menu** following which the corresponding sensor for each selected channel is updated.

```
displaygotofirstchar(2);
sprintf(display,"DT470<*><ENTER>");
writestringspi_line2(display);
```

```
channelsensortable[channelselected] = sensorselected;
```

v. Curve Data Entry:

This specifies the algorithm designed for a key functionality of the device – **setting up a custom sensor by taking in prerecorded data values for the sensor.**

First, the number of points - 'n' is taken as input following which the points themselves are inputted; negative, positive as well as floating point values are allowed using the '*' to add a minus sign or a decimal point.

Finally on pressing '#' - ENTER key, the 'n' values are stored in the 'x' and 'y' variables.

```
case 1:
    a1 = 0;
    //PC.printf("1");
    onechar='1';

    displaychar_line2(onechar);
    strcat(packet_data,"1");
    break;
```

```
case 10:
    if(ctr == 1)
    {
        onechar='-';
        displaychar_line2(onechar);
        strcat(packet_data, "-" );
    }
    else
    {
        onechar='.';
        displaychar_line2(onechar);
        strcat(packet_data, ".");
    }
}
```

```
case 12:
    if (a1 == 1)
    {
        switch ( *s )
        {
            case 'x':
                kybstrnum = x[ind];
                break ;
            case 'y':
                kybstrnum=y[ind];
                break ;
        }
    }
}
```

vi. Data Interpolation:

This serves to convert the analog voltage values read in from the sensors into digitized temperature values for each of the corresponding sensors. Here, three sensors have been installed and hence three separate interpolation functions have been defined. Each of these functions has the **interpolation formulation on the basis of prerecorded temperature vs voltage values for each sensor as per research data.**

```
for ( i=0; i < curveCCS3-2; i++)
{
    if (volt <= r[i] && volt > r[i+1])
    {
        temp = m_cs[i]*volt + c_cs[i];
        return temp;
    }
}
```

```
float convert_P_to_T( float voltinD ) // pt 100 curve
{
    voltinD = voltinD * 409.5; // converting the volt into 0 to 4095
    float f;
    f = 30.05575 + 0.08533 * voltinD + 0.00000180114 * voltinD * voltinD ;
    return f;
}
```

```
float convert_D_to_T(float D)
{
    float T;
    D *= 409.5;

    if (D > 3522)
        T = 481 - (454 * D) / 4096;
    else if (D > 3145)
        T = 754 - (779 * D) / 4096;
    else if (D > 2762)
        T = 810 - (844 * D) / 4096;
    else if (D > 2721)
        T = 2207 - (2930 * D) / 4096;
}
```

vii. Data/Settings Save and Load:

This offers one of the key functionalities provided in the device i.e. to **save the existing data** values as well as **load the last updated** sensor selection **configuration settings**. This is triggered by pressing the '8' key in the keypad. This is done via the **"File" class provided in C++** to create the log file while saving data and access an already present log file while loading data.

The values stored in the log file include the sensors for each of the 8 channels, the voltage (x) and temperature (y) values, and the slope (m) and intercept (c) values for the custom sensor obtained during interpolation.

```
void save_current_configurations()
{
    FILE *fp;
    fp = fopen("/local/datafile.txt", "w");
    int i;

    for (i=0;i<9;i++)
        fprintf( fp, "%d\t",channelsensortable[i] );
    fprintf(fp, "\n");

    for (i=0;i<100;i++)
    {
        fprintf(fp, "%lf\t",x[i]);
        fprintf(fp, "%lf\n",y[i]);
    }
    fprintf(fp, "\n");

    for (i=0;i<100;i++)
    {
        fprintf(fp, "%lf\t",m[i]);
        fprintf(fp, "%lf\n",c[i]);
    }
    fprintf(fp, "\n");

    fclose(fp);
}
```

```
void load_saved_configurations()
{
    FILE *fp;
    fp = fopen("/local/datafile.txt", "r");
    if (fp == NULL) return;
    int i;

    for (i=0;i<9;i++)
        fscanf(fp, "%d\t",&channelsensortable[i]);

    chs1 = channelsensortable[1] ;

    for (i=0;i<100;i++)
        fscanf(fp, "%lf\t%lf\n",&x[i],&y[i]);

    for (i=0;i<100;i++)
        fscanf(fp, "%lf\t%lf\n",&m[i], &c[i]);

    fclose(fp);
}
```

viii. RPC Connectivity:

There are many occasions when its useful to be able to **talk to your mbed from a computer** to gather data using the sensors or create remote applications over a network. Creating an interface between mbed and a computer can be difficult because it requires you to specify a communication format and then to write code on both the mbed and the computer.

Remote Procedure Call (RPC) is an **inter-process communication technique** used for client-server apps without the programmer coding the details for the remote interaction.

RPC commands are in a predefined format and can be sent over any transport mechanism that can send a stream of text.

RPCVariable allows you to read and write the value of any variable on mbed. You do this by creating an RPCVariable object and then attaching the variable you want to access over RPC.

RPCFunction object makes it possible to call a custom function over RPC. To use you first create the function, you then create the RPCFunction object when you attach

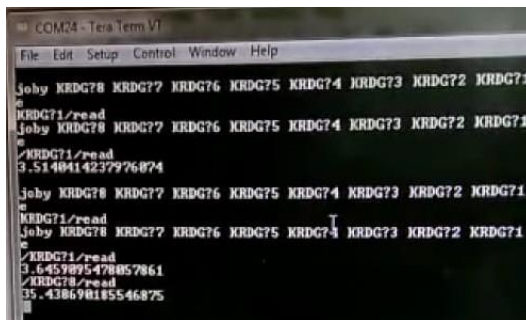
the function and give it a name by which it will be called over RPC.

RPC Commands are in the format: **"/<Object name>/<Method name> <Arguments separated by spaces>**

If you send just "/" mbed will return a list of objects that can be used.

If you send **"/<object name>/"** then mbed will return the methods which can be used on this object.

As seen below, the sensor temperature variables are defined as **RPCVariables** accessible via the Tera Term Virtual Terminal using the variables **"KRDG?1"** and so on. These are then accessed by using the command – **KRDG?1/read** in the terminal.



```
COM24 - Tera Term VT
File Edit Setup Control Window Help

Joby KRDG78 KRDG77 KRDG76 KRDG75 KRDG74 KRDG73 KRDG72 KRDG71
KRDG71/read
3.5140414237976874
Joby KRDG78 KRDG77 KRDG76 KRDG75 KRDG74 KRDG73 KRDG72 KRDG71
KRDG71/read
3.6459895478857861
Joby KRDG78 KRDG77 KRDG76 KRDG75 KRDG74 KRDG73 KRDG72 KRDG71
KRDG78/read
35.438698185546875
```

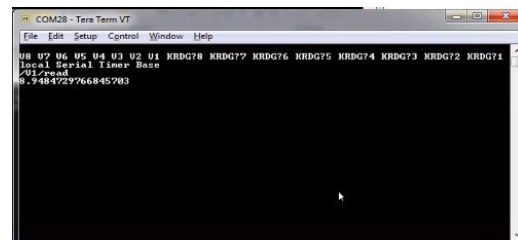
```
RPCVariable<float> rpcTdata1(&sensorTemp[1], "KRDG?1");
RPCVariable<float> rpcTdata2(&sensorTemp[2], "KRDG?2");
RPCVariable<float> rpcTdata3(&sensorTemp[3], "KRDG?3");
RPCVariable<float> rpcTdata4(&sensorTemp[4], "KRDG?4");
RPCVariable<float> rpcTdata5(&sensorTemp[5], "KRDG?5");
RPCVariable<float> rpcTdata6(&sensorTemp[6], "KRDG?6");
RPCVariable<float> rpcTdata7(&sensorTemp[7], "KRDG?7");
RPCVariable<float> rpcTdata8(&sensorTemp[8], "KRDG?8");
```

6. Results

Each of the algorithms was found to be working with full efficiency with some minor bugs such as improper escape key operation and data loading which have been rectified completely. The functionalities listed earlier have all been observed to be working with great accuracy and without any major glitches, almost negligible glitches in fact. The temperatures and voltages have been read and displayed with a **precision of up to two decimal places**.

Also, for each operation or anytime a user operates the device, adequate instructions have been provided on the 16x2 LCD display despite the limit on the number of characters such that the user can have a smooth experience while operating the device.

Some of the images of the working device have been provided below:



Further, the **calibration values have been logged which are stored during the interpolation process** of any of the sensors. These values are compared to the untouched non-interpolated values in order **to quantify the error** in the interpolated readings as compared to the actual readings. These error quantities have **helped us in formulating the most appropriate interpolation techniques with a minimized error** in the calibration values.

An excerpt from the logged values before and after interpolation along with the error has been attached below:

A	B	C	D	E	F	G
Temp in Kelvin	Resistance according Internet eqn	voltage for 657uA	gain 100	ADC decimal value	Interpolated T	ERROR f2-a2
31	0.050	3.27203953444799E-05	0.0032720395	1	30.2	-0.8
32	0.499	0.000327710837216	0.0327710837	13	31.2	-0.8
33	0.947	0.0006223987008	0.0622398701	25	32.2	-0.8
34	1.395	0.000916785702521	0.0916785703	38	33.3	-0.7
35	1.843	0.00121087355234	0.1210873552	50	34.3	-0.7
36	2.290	0.001504663953752	0.1504663954	62	35.3	-0.7
37	2.737	0.00179815860379	0.1798158604	74	36.4	-0.6
38	3.185	0.00209112310047	0.2091123100	86	37.5	-0.5
39	3.632	0.00238359400792	0.2383594008	98	38.6	-0.4
40	4.080	0.00267606491519	0.2676064915	110	39.7	-0.3
41	4.527	0.00296853582264	0.2968535823	122	40.8	-0.2
42	4.975	0.00326100673011	0.3261006730	134	41.9	-0.1
43	5.422	0.00355347763766	0.3553477638	146	43.0	0.0
44	5.870	0.00384594854513	0.3845948545	158	44.1	0.1
45	6.317	0.00413841945260	0.4138419453	170	45.2	0.2
46	6.765	0.00443089036007	0.4430890360	182	46.3	0.3
47	7.212	0.00472336126754	0.4723361268	194	47.4	0.4
48	7.660	0.00501583217501	0.5015832175	206	48.5	0.5
49	8.107	0.00530830308248	0.5308303082	218	49.6	0.6
50	8.555	0.00560077398995	0.5600773990	230	50.7	0.7
51	9.002	0.00589324489742	0.5893244897	242	51.8	0.8
52	9.450	0.00618571580489	0.6185715805	254	52.9	0.9
53	9.897	0.00647818671236	0.6478186712	266	54.0	1.0
54	10.345	0.00677065761983	0.6770657620	278	55.1	1.1
55	10.792	0.00706312852730	0.7063128527	290	56.2	1.2
56	11.240	0.00735560343477	0.7355603435	302	57.3	1.3
57	11.687	0.00764807434224	0.7648074342	314	58.4	1.4
58	12.135	0.00794054524971	0.7940545250	326	59.5	1.5
59	12.582	0.00823301615718	0.8233016157	338	60.6	1.6
60	13.030	0.00852548706465	0.8525487065	350	61.7	1.7
61	13.477	0.00881795797212	0.8817957972	362	62.8	1.8
62	13.925	0.00911042887959	0.9110428880	374	63.9	1.9
63	14.372	0.00940290378706	0.9402903787	386	65.0	2.0
64	14.820	0.00969537469453	0.9695374695	398	66.1	2.1
65	15.267	0.00998784560200	0.9987845602	410	67.2	2.2
66	15.715	0.01028031650947	0.0102803165	422	68.3	2.3
67	16.162	0.01057278741694	0.0105727874	434	69.4	2.4
68	16.610	0.01086525832441	0.0108652583	446	70.5	2.5
69	17.057	0.01115772923188	0.0111577292	458	71.6	2.6
70	17.505	0.01145020013935	0.0114502001	470	72.7	2.7
71	17.952	0.01174267104682	0.0117426710	482	73.8	2.8
72	18.399	0.01203514195429	0.0120351419	494	74.9	2.9
73	18.847	0.01232761286176	0.0123276128	506	76.0	3.0
74	19.294	0.01262008376923	0.0126200837	518	77.1	3.1
75	19.742	0.01291255467670	0.0129125546	530	78.2	3.2
76	20.189	0.01320502558417	0.0132050255	542	79.3	3.3
77	20.637	0.01349749649164	0.0134974964	554	80.4	3.4
78	21.084	0.01379031139911	0.0137903113	566	81.5	3.5
79	21.532	0.01408312630658	0.0140831263	578	82.6	3.6
80	21.979	0.01437559721405	0.0143755972	590	83.7	3.7
81	22.427	0.01466806812152	0.0146680681	602	84.8	3.8
82	22.874	0.01496053902899	0.0149605390	614	85.9	3.9
83	23.322	0.01525300993646	0.0152530099	626	87.0	4.0
84	23.769	0.01554548084393	0.0155454808	638	88.1	4.1
85	24.217	0.01583795175140	0.0158379517	650	89.2	4.2
86	24.664	0.01613042265887	0.0161304226	662	90.3	4.3
87	25.112	0.01642289356634	0.0164228935	674	91.4	4.4
88	25.559	0.01671536447381	0.0167153644	686	92.5	4.5
89	26.007	0.01700783538128	0.0170078353	698	93.6	4.6
90	26.454	0.01730030628875	0.0173003062	710	94.7	4.7
91	26.902	0.01759277719622	0.0175927771	722	95.8	4.8
92	27.349	0.01788524810369	0.0178852481	734	96.9	4.9
93	27.797	0.01817771901116	0.0181777190	746	98.0	5.0
94	28.244	0.01847018991863	0.0184701899	758	99.1	5.1
95	28.692	0.01876266082610	0.0187626608	770	100.2	5.2
96	29.139	0.01905513173357	0.0190551317	782	101.3	5.3
97	29.587	0.01934760264104	0.0193476026	794	102.4	5.4
98	30.034	0.01964007354851	0.0196400735	806	103.5	5.5
99	30.482	0.01993254445598	0.0199325444	818	104.6	5.6
100	30.929	0.02022501536345	0.0202250153	830	105.7	5.7
101	31.377	0.02051748627092	0.0205174862	842	106.8	5.8
102	31.824	0.02081031117839	0.0208103111	854	107.9	5.9
103	32.272	0.02110313608586	0.0211031360	866	109.0	6.0
104	32.719	0.02139560699333	0.0213956069	878	110.1	6.1
105	33.167	0.02168807790080	0.0216880779	890	111.2	6.2
106	33.614	0.02198054880827	0.0219805488	902	112.3	6.3
107	34.062	0.02227301971574	0.0222730197	914	113.4	6.4
108	34.509	0.02256549062321	0.0225654906	926	114.5	6.5
109	34.957	0.02285796153068	0.0228579615	938	115.6	6.6
110	35.404	0.02315043243815	0.0231504324	950	116.7	6.7
111	35.852	0.02344290334562	0.0234429033	962	117.8	6.8
112	36.299	0.02373537425309	0.0237353742	974	118.9	6.9
113	36.747	0.02402784516056	0.0240278451	986	120.0	7.0
114	37.194	0.02432031606803	0.0243203160	998	121.1	7.1
115	37.642	0.02461278697550	0.0246127869	1010	122.2	7.2
116	38.089	0.02490525788297	0.0249052578	1022	123.3	7.3
117	38.537	0.02519772879044	0.0251977287	1034	124.4	7.4
118	38.984	0.02549020369791	0.0254902036	1046	125.5	7.5
119	39.432	0.02578267460538	0.0257826746	1058	126.6	7.6
120	39.879	0.02607514551285	0.0260751455	1070	127.7	7.7
121	40.327	0.02636761642032	0.0263676164	1082	128.8	7.8
122	40.774	0.02666008732779	0.0266600873	1094	129.9	7.9
123	41.222	0.02695255823526	0.0269525582	1106	131.0	8.0
124	41.669	0.02724502914273	0.0272450291	1118	132.1	8.1
125	42.117	0.02753750005020	0.0275375000	1130	133.2	8.2
126	42.564	0.02783031139911	0.0278303113	1142	134.3	8.3
127	43.012	0.02812312274812	0.0281231227	1154	135.4	8.4
128	43.459	0.02841593409713	0.0284159340	1166	136.5	8.5
129	43.907	0.02870874544614	0.0287087454	1178	137.6	8.6
130	44.354	0.02900155679515	0.0290015567	1190	138.7	8.7
131	44.802	0.02929436814416	0.0292943681	1202	139.8	8.8
132	45.249	0.02958717949317	0.0295871794	1214	140.9	8.9
133	45.697	0.02988031139911	0.0298803113	1226	142.0	9.0
134	46.144	0.03017312274812	0.0301731227	1238	143.1	9.1
135	46.592	0.03046593409713	0.0304659340	1250	144.2	9.2
136	47.039	0.03075874544614	0.0307587454	1262	145.3	9.3
137	47.487	0.03105155679515	0.0310515567	1274	146.4	9.4
138	47.934	0.03134436814416	0.0313443681	1286	147.5	9.5
139	48.382	0.03163717949317	0.0316371794	1298	148.6	9.6
140	48.829	0.03193031139911	0.0319303113	1310	149.7	9.7
141	49.277	0.03222312274812	0.0322231227	1322	150.8	9.8
142	49.724	0.03251593409713	0.0325159340	1334	151.9	9.9
143	50.172	0.03280874544614	0.0328087454	1346	153.0	10.0
144	50.619	0.03310155679515	0.0331015567	1358	154.1	10.1
145	51.067	0.03339436814416	0.0333943681	1370	155.2	10.2
146	51.514	0.03368717949317	0.0336871794	1382	156.3	10.3
147	51.962	0.03398031139911	0.0339803113	1394	157.4	10.4
148	52.409	0.03427312274812	0.0342731227	1406	158.5	10.5
149	52.857	0.03456593409713	0.0345659340	1418	159.6	10.6
150	53.304	0.03485874544614	0.0348587454	1430	160.7	10.7
151	53.752	0.03515155679515	0.0351515567	1442	161.8	10.8
152	54.199	0.03544436814416	0.0354443681	1454	162.9	10.9
153	54.647	0.03573717949317	0.0357371794	1466	164.0	11.0
154	55.094	0.03603031139911	0.0360303113	1478	165.1	11.1
155	55.542	0.03632312274812	0.0363231227	1490	166.2	11.2
156	55.989	0.03661593409713	0.0366159340	1502	167.3	11.3
157	56.437	0.03690874544614	0.0369087454	1514	168.4	11.4
158	56.884	0.03720155679515	0.0372015567	1526	169.5	11.5
159	57.332	0.03749436814416	0.0374943681	1538	170.6	11.6
160	57.779	0.03778717949317	0.0377871794	1550	171.7	11.7
161	58.227	0.03808031139911	0.0380803113	1562	172.8	11.8
162	58.674	0.03837312274812	0.0383731227	1574	173.9	11.9
163	59.122	0.03866593409713	0.0386659340	1586	175.0	12.0
164	59.569	0.03895874544614	0.0389587454	1598	176.1	12.1
165	60.017	0.03925155679515	0.0392515567	1610	177.2	12.2
166	60.464	0.03954436814416	0.0395443681	1622	178.3	12.3
167	60.912	0.03983717949317	0.0398371794	1634	179.4	12.4
168	61.359	0.04013031139911	0.0401303113	1646	180.5	12.5
169	61.807	0.04042312274812	0.0404231227	1658	181.6	12.6
170	62.254	0.04071593409713	0.0407159340	1670	182.7	12.7

7. Conclusion

The algorithms highlighted and described in section 5 have been successfully implemented as a part of the firmware design of the 8-channel digital temperature meter embedded device. The device has been tested with real-time sensor data readings under varying conditions and after detailed analysis, the appropriate firmware algorithms have been designed to make the device capable of handling real-time operations. Despite the limiting factor of the results having the surety of varying from system to system as well as with varying operating conditions, the **device has been built** to suitably handle such fluctuations and **to provide relatively precise results** with respect to the expected theoretical results.

Thus, this makes the research work a successful endeavor. Nevertheless, it is always open to suggestions, modifications so as to facilitate **future applications as well as updates** when it comes to real-life scenarios which are constantly being updated.

For instance, the **RPC connectivity feature can be further extended** such that the device can be controlled or **communicated with over the Internet** by means of the Ethernet and secure HTTPS protocols. These can be developed either from scratch or built on top of the existing RPC architecture as the RPC Ethernet communication protocol.

With such connectivity features as well as accurate sensor readings with fast refresh rates, the device turned out to be way more efficient than we imagined it would during the initial design phases. This in itself is a huge achievement as the **device is self-sufficient and ready to be manufactured for supply to relevant organizations**. Regardless, the device is efficient but can surely be improved upon so as to be the best version of itself to be utilized with perfection in real-time real-world scenarios to offer the best features of its class.

8. Acknowledgments

The time we spent working as remote summer interns at IUAC New Delhi from May 2020 to August 2020 was truly a memorable one as it was rich in building experience over time and helped us discover our potential. We believe that the experience that we've gained as an intern by working under the guidance of our esteemed mentor will forever shape and influence our professional life while fostering personal growth and development.

First of all, as always, we will always be grateful to our parents who have always been there by our side, to encourage us, to support us; being the constant source of motivation and guidance throughout our lives. Thank you for being there.

Finally, we would like to express our sincere thanks and gratitude for our mentor, Dr. Joby Antony, for giving us this essential opportunity to work as an intern under him. Dr. Joby has spent his invaluable time and knowledge with us throughout the period of the internship and his wonderful experiences have greatly motivated us to keep working towards the completion of this project. Thank you, sir for being so patient throughout the period of our internship while also being available anytime for your constant guidance and support. We believe we have greatly benefited from the great rapport that we shared and will definitely implement all that you have taught us. We will always be grateful to you for this opportunity.

To all those who have provided us with any sort of help that made this internship and the completion of the research work assigned to us possible, we extend a sincere thanks and gratitude for your contribution and support.

9. Bibliography

1. mbed LPC1768 - <https://os.mbed.com/platforms/mbed-LPC1768/>
2. Understanding RS232 Serial Port Communication - <https://www.windmill.co.uk/rs232-communication.html>
3. What is RS232 - <https://www.bb-elec.com/Learning-Center/All-White-Papers/Serial/What-Is-RS-232.aspx>
4. 4X3 Keypad Operating Mechanism - <https://pijaeducation.com/arduino/keypad/4x3-keypad-operating-mechanism/>
5. Working of 16X2 LCD - <https://www.electronicsforu.com/resources/learn-electronics/16x2-lcd-pinout-diagram>
6. Introduction to ARM MBED LPC1768 - <https://www.electronicwings.com/mbed/introduction-to-arm-mbed-lpc1768>
7. Platinum Resistance Thermometers - <https://www.peaksensors.co.uk/resources/resistance-thermometer-information/>
8. Silicon Diodes DT470 - http://www.ic72.com/pdf_file/d/134778.pdf
9. Cryogenics - <https://www.britannica.com/science/cryogenics>
10. IUAC Cryogenics - <https://www.iuac.res.in/Cryogenics>
11. MCP3208 IC - <https://components101.com/ics/mcp3008-adc-pinout-equivalent-datasheet>
12. SerialRPCInterface Library - <https://os.mbed.com/cookbook/RPC-Interface-Library>
13. RPC in OS - <https://www.guru99.com/remote-procedure-call-rpc.html>
14. Cryogenic temperature measurement with microcontrollers - https://www.researchgate.net/publication/336715237_A_low-cost_cryogenic_temperature_measurement_system_using_Arduino_microcontroller