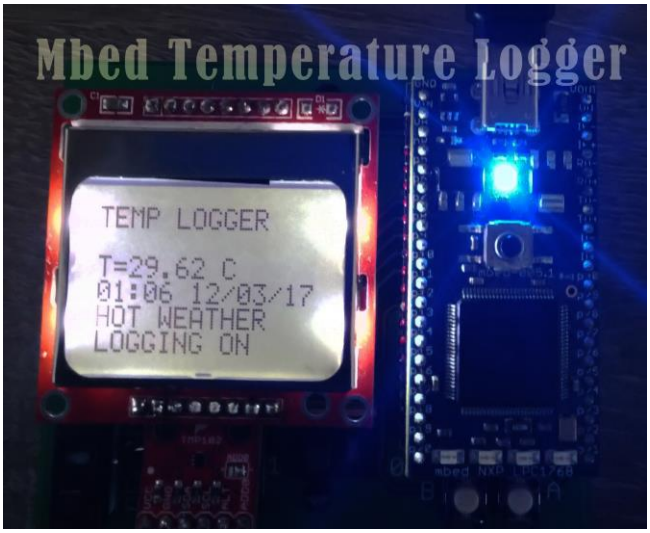


# Design and Construction of an Embed Temperature Logger

JANUARY 2018

**SCHOOL OF ELECTRONICS AND ELECTRICAL ENGINEERING**

DAGOGO GOWIN ORIFAMA- 201177661

**ELEC5451M Mini Projects and Laboratories**

# Table of Contents

## List of Figures



## Tables

# 1. Introduction

There is numerous definition when it comes to embedded systems. According to [1], it is a system which embeds a microprocessor into the device to be controlled to perform a specific task. An example is having a microprocessor placed inside products like a toaster, or camera. Continuing in the same vein, [2] defined embedded systems as a system which constitutes hardware and software, programmed for a specific task. It went on to elaborate on the part of an embedded system and are made up of two major elements namely

- Embedded system hardware: the embedded system requires a hardware part for it to run, this is usually based on a microprocessor or microcontroller. The hardware is made up of other elements to mention but a few are input, output interfaces and display [2].
- Embedded system software: This is written to perform a specific function. It can be written in either machine code or high-level languages (C and C++ are commonly used) [2].

This report is in the design, construction and programming of an embedded temperature logger. It focusses is on how a microcontroller will be programmed to seamlessly integrate the measurement of temperature from the surrounding using a TMP102 temperature sensor, display temperature status of the surrounding on a Nokia 5110 liquid crystal display (LCD), at the same time plots real-time values of the temperature on the LCD and updates data into an Excel spreadsheet every 60 seconds.

The project has applications in various sectors, one of which is in heating, ventilation and control (HVAC), which uses an automated temperature monitoring system, this project will play a vital role in such systems in tripping on cooling when the temperature of the surrounding falls below a threshold temperature or tripping it off when the ambient temperature is reached.

# 2. Hardware Description of the Embedded Temperature Logger

The block diagram of the temperature logger is shown in figure 1, it consists of a microcontroller, N5110 LCD, temperature sensor, serial interface and the excel spreadsheet. Also, table 1 shows the hardware requirement for the circuit construction and the number of each component required

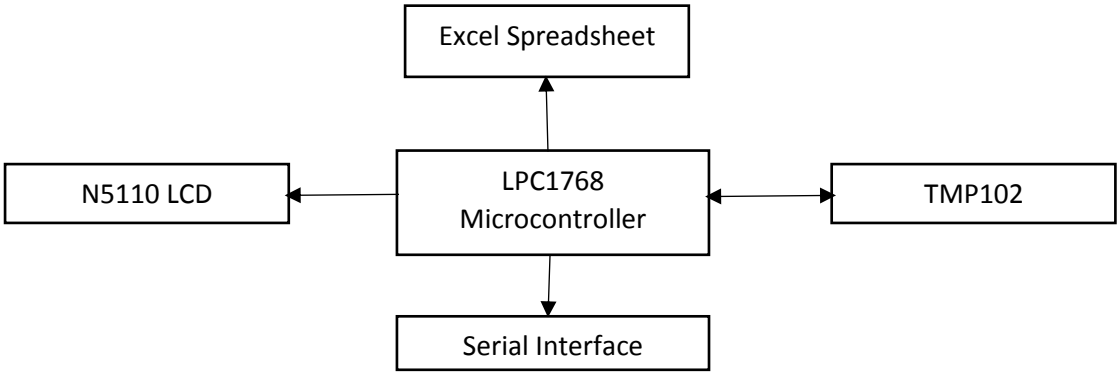


Figure 1: Block diagram of the Figure proposed temperature logger

| Name of hardware | Amount |
|---|---|
| LPC1768 | 1 |
| Temperature sensor (TMP 102) | 1 |
| N5110 LCD | 1 |
| CMOS battery | 1 |
| Printed circuit board (PCP) | 1 |
| Switch | 2 |
| Batter casing | 2 |
| Push button | 2 |
| IC sockets | 3 |
| Battery (AA) | 4 |

Table 1: hardware requirement of the temperature logger

A detailed block diagram showing the interconnection of the hardware is shown below. It is divided into 4 block sections which are power or battery, controller, sensor and the display block

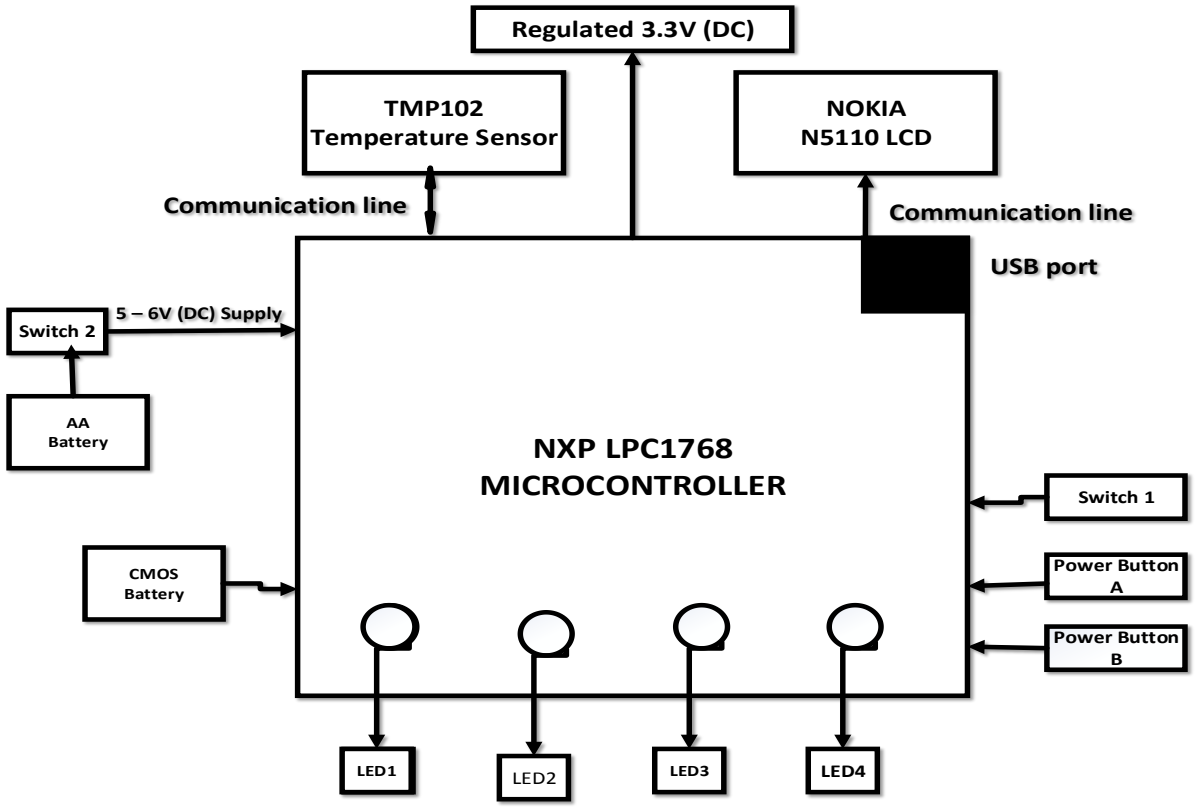


Figure 2: Detailed block diagram of hardware in temperature sensor

## 2.1. The TMP102 Sensor Block

A sensor is an electronic device which measures the changes of natural occurring parameters in the surrounding and converts it into an electrical signal. This block consists of a TMP102 temperature sensor which measures the change in temperature in the surrounding and converts it into bit. This bit is passed through an analogue to digital converter which generates the equivalent value of the temperature reading. The TMP102 temperature sensor uses 12bit in converting from analogue to digital reading, this yield a resolution of approximately

0.0625$^{o}$C for a range of temperature between -40$^{o}$C to 125$^{o}$C. For negatives values of temperature, 2's complement is applied in generating the bit. The microcontroller supplies a regulated voltage of 3.3V (DC) to the temperature sensor, TMP102 maximum allowable voltage is 3.6V (DC). The communication between the TMP102 sensor and the microcontroller is done using the inter-integrated circuit ($I^2C$) protocol.


Figure 3: TMP102 temperature sensor

## 2.2. The power or battery block

This block consists of the various hardware which supplies power to the temperature logger. They include a 4*1.5V battery bank, the CMOS battery and the USB 2.0 port. The 4_AA batteries serve as an alternative source of power to the temperature logger when it is disconnected from a computer. The 4_AA batteries rating 1.5V each has a series supply voltage of 6V, which is sufficient to power the device. The 4_AA batteries have a combined series supply current of 2.2AH at 0.5A discharge, this is sufficient for the microcontroller. The CR2032, CMOS battery is used for storing time using real-time clock (RTC). The CMOS ensures that the microcontroller's time is running correctly (even when the AA batteries are not attached) by generating the required pulse. When the temperature logger is connected to a computer through its USB 2.0 port, the USB supplies power to the device. The USB 2.0 which has 4 pins of which 2 are used for communication while the other 2 are for power supply, can supply a ground voltage of 0V and a +5V, with a maximum current capacity of 0.5A.


**(a)**


**(b)**

Figure 4:(a) 1.5V batteries (b) CMOS battery

## 2.3. The Controller Block

The controller block is made up of the microcontroller used or this project. The microcontroller is NXP LPC1768 belonging to the ARM Cortex-M3 family of microcontrollers.

It is a high performance, low cost, low power 32bit RISC processor. It has a 32KB static random-access memory (S-RAM) and a 512KB flash memory, also it operates at a CPU frequency of up to 100MHz. The LPC1768 microcontroller has an input supply voltage range of between 4.5V to 14V, it is powered by input pin or USB connections. The LPC1768 is made up 40pins on both side with 20pins on each side, it also has provision for an inter-integrated communication protocol, I2C (when communicating with the TMP102 sensor), a serial communication (when communicating with the Nokia N5110 LCD). The LPC1768 microcontroller has five light emitting diodes (LEDs), one is for the status of the microcontroller (ON when active) and the remaining 4 LEDs are for control. Furthermore, the LPC1768 microcontroller comes with a reset button and a USB port, the reset button is used to trigger a hardware restart of the microcontroller while the USB port is used when interfacing the microcontroller to a computer, it is basically used to load program scripts directly into the microcontroller. A screenshot of NXP LPC1768 is shown below

Figure 5: NXP LPC1768 Microcontroller

## 2.4. The display and communication block

This block is made up of Nokia N5110 LCD screen and Microsoft Excel spreadsheet for logging data.

### 2.4.1. The N5110 LCD Screen

The N5110 LCD is an 84 by a 48-pixel monochrome display that comes with backlight. It can display a total of 14 characters in a row for six rows, total 84 characters, this is because each displayed character uses 8 by 6 pixels for its vertical and horizontal dimension. The brightness of the backlight can be controlled by adjusting its setting programmatically. Special character can be printed by the LCD, this is done by a technique known as pixels manipulations, which involve the turning ON and OFF some of its pixels. The temperature limit of the embedded temperature logger is set by the N5110 which has an operational range of -25$^{o}$C to 70$^{o}$C. The N5110 LCD get its power supply of 3.3V from the microcontroller, but the backlight is from pin 21 (p21 pulse width modulated output, PWM) of the microcontroller. The N5110 LCD communicates with the microcontroller through a short distance serial interface protocol known as a serial peripheral interface (SPI).

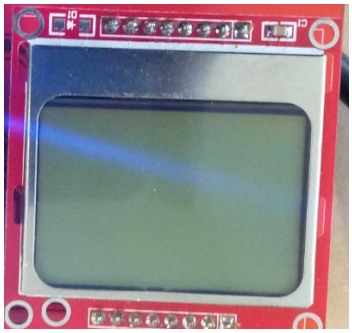

Figure 6: Nokia N5110 LCD screen

### 2.4.2. The Microsoft Excel Spreadsheet

This is a software developed by Microsoft it is basically used for storing, organising and manipulating data. The Excel spreadsheet is deemed to be one of the most complicated spreadsheets which can be used in graphing and charting data, formatting data, printing data and charts for use in reports and much more [3]. The embedded temperature logger has been implemented will log data into an excel (CSV file) which will be used to generate a temperature profile plot.

## 3. The Software Description

The software for the temperature logger was written in C++ programming language. The compilation of the code was done on the mbed online compiler; this was also used to generate the binary file which was loaded via the USB port into the microcontroller. The software ensures communication of data between the various connected hardware.

### 3.1. The design specification of the embedded temperature logger

The temperature logger can be powered using the 4AA batteries or through a USB cable to a computer. The main purpose of connecting the device to the computer is to be able to log the temperature and corresponding time every 60 seconds. There are other requirements in the design specification of the embedded temperature logger which include:

- Ability to retrieve current temperature reading from the TMP102 sensor and display on the N5110 LCD.
- Toggle between normal/default display view and plot view to show a real-time temperature plot
- Real-time weather status on the display to indicate when the temperature is below or above specified range of values.
- A switch to control data log into an Excel Spreadsheet
- A switch that controls the power supply from the battery
- Controlling the intensity of the N5110 LCD brightness using one of the buttons
- Display logging status (if data logging is ON or OFF) on the LCD
- LEDs displays as feedback during data logging

## 3.2. Constraint and Assumption of the proposed embedded temperature logger

During the development of the software for the temperature logger, it was assumed that the required hardware has been connected correctly and furthermore, it was established that the temperature range of embedded temperature logger is 0°C (set by N5110 LCD) [4] and 55°C (set by the AA batteries) [5].

## 3.3. The Software Structure of Embedded Temperature Logger

The general structure of the software that could achieve the stated specification is shown in Figure 7a, the structure is divided into functions in Figure 7b. Figure 8a represents the flowchart of the get temperature function that collects the current temperature reading. Figure 8b represents the flowchart for the interrupt that displays data on the LCD. Figure 8c represent chart for the interrupt that updates temperature plots every 0.5 seconds. Figure 9a represents an interrupt which controls data logging into a CSV file every 60 seconds, provided the logging switch is ON. Figure 9b displays the weather status on the LCD based on the value of temperature. Figure 9d controls the brightness of the LCD, this is dependent upon the number of times button B is pressed. Figure 10a is a function that checks if the data has been successfully retrieved from the TMP102 sensor, and flashes the LED if false. Figure 10b represents a function that generates the temperature values to be plotted on the LCD and Figure 10c is the flow chart for the function which writes data to a CSV file.

The files contained in this program include main.cpp, getData.cpp, getData.h, display.cpp, display.h, rtc.cpp and rtc.h.

GetData.h and GetData.cpp: getData.h is a header file which prototypes functions used in getData.cpp, getData.cpp contains four functions; temperror which flashes the LED3 and LED4 if data is not received from the temperature sensor it is represented in figure 10a, initTMP102 initialises the temperature reading, getTemperature is a function that leads to the acquisition of the reading from the temperature sensor it is represented in figure 8a, and getTime is a function that gets and formats current time and date.

Display.h and Display.cpp: display.h is a header file which prototypes all functions used in getData.cpp, disaplay.cpp contains four functions; togglePlot which checks if button A or B has been pressed, and sets the value of a variable plot to either 1 or 0. PlotArray is a function which generates the values of temperature to be plotted it is represented in figure 10b. Display1 is a function which handles everything that has to do with displaying data on the LCD, firstly it checks if the temperature logger is being on for the first time, if true, displays a welcome message and user guide for 5 seconds and then displays the normal/default view, which is the view displaying temperature reading, next the function checks if the button A or B has been pressed by considering the value of the plot variable from the togglePlot function, and then switch between the plot and the normal view. Next, is the brightness function which controls the brightness of the LCD when called upon by an interrupt (brightSet) which is triggered when the button B is pressed (LCD must be in default view).

RTC.h and RTC.cpp: rtc.h h is a header file which prototypes all functions used in rtc.cpp, these functions are for setting the Unix time.

Main.cpp: this is a very important file as all mbed program must have a main.cpp file. This file contains the main function which is the most important function in the program, this is because all other functions created can only be accessed from the main function. The main function of the temperature logger program has three ticker interrupts and the togglePlot function. Display ticker calls the display1 function to display and update data on the LCD every 0.5 seconds it is represented in figure 8b. Next, is the plotdata ticker which calls on the plotArray function to send updated temperature reading to the LCD graph plot every 1 second it is represented in figure 8c. Also, we have the file ticker interrupt (shown in figure 9a) which calls on the writeDataToFile function (present within main.cpp) every 60 seconds to log data into a CSV file.

The Nokia N5110 and the mbed libraries were included into the temperature logger program, this enabled the use of inbuilt functions like lcd.init, lcd.refresh, lcd.SetPixel, DigitalOut, DigitalIn, BusOut, InterruptIn, Ticker etc. Furthermore, the addresses of the temperature and configuration registers were configured.

Next, is the program compilation, after which the .bin file generated was uploaded into the LPC1768 microcontroller via USB port. To instruct the microcontroller to run the latest copied file the reset button is pressed.

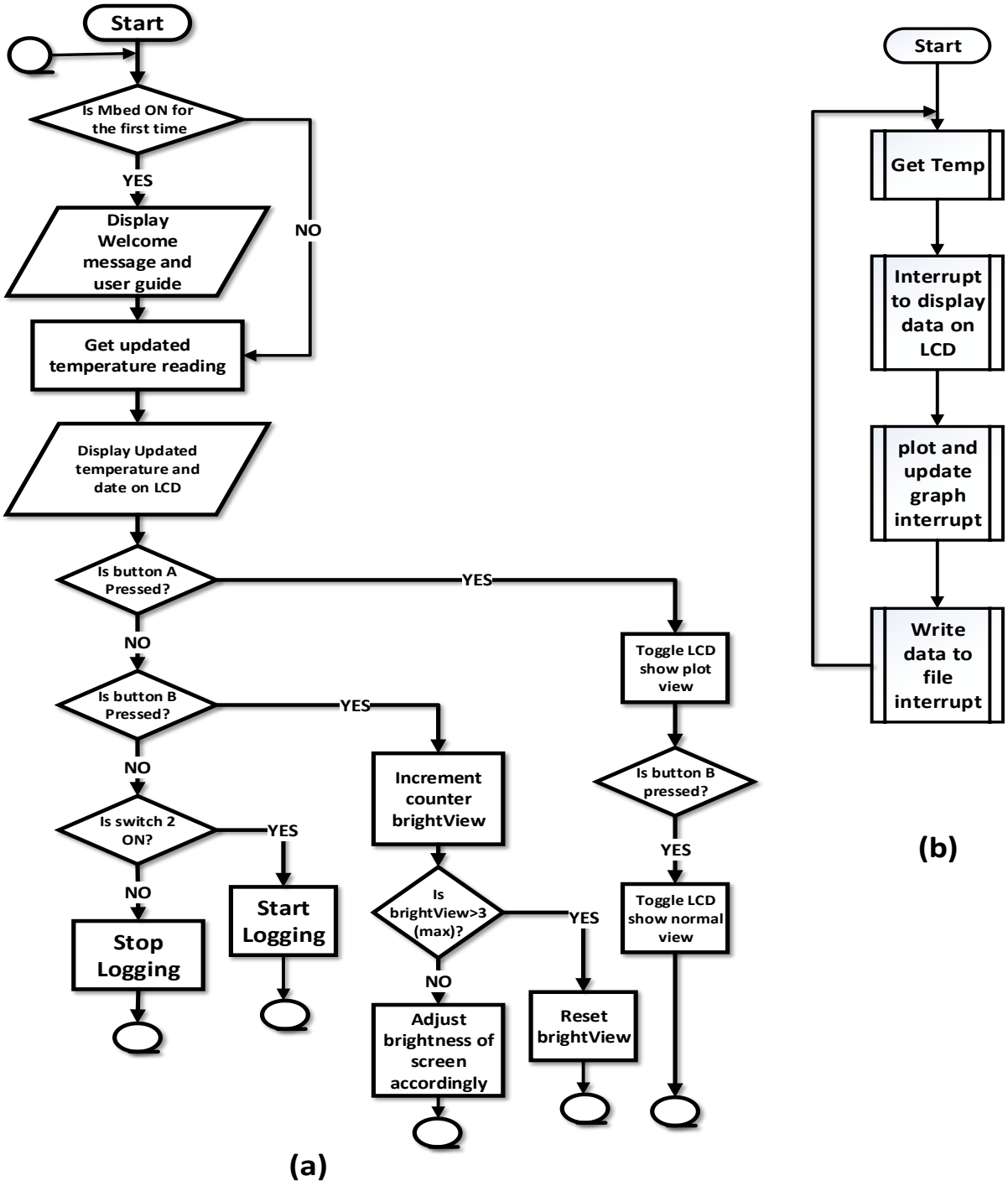


Figure 7: (a)Detailed software structure for temperature logger (b) Detailed software structure using functions

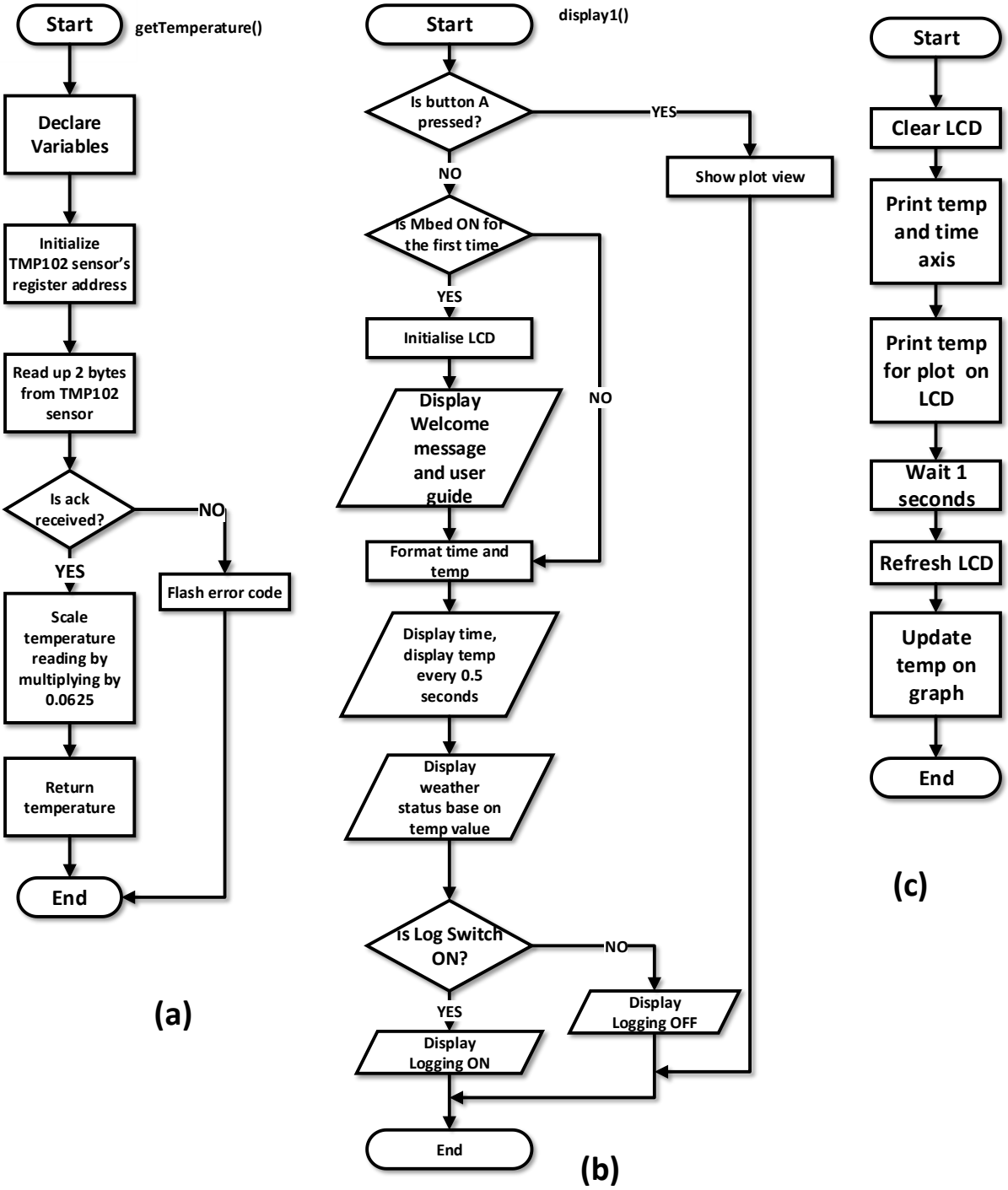


Figure 8: (a) Get temperature functions (b) Interrupt that displays data on LCD(c) Interrupt that updates temperature values on plot

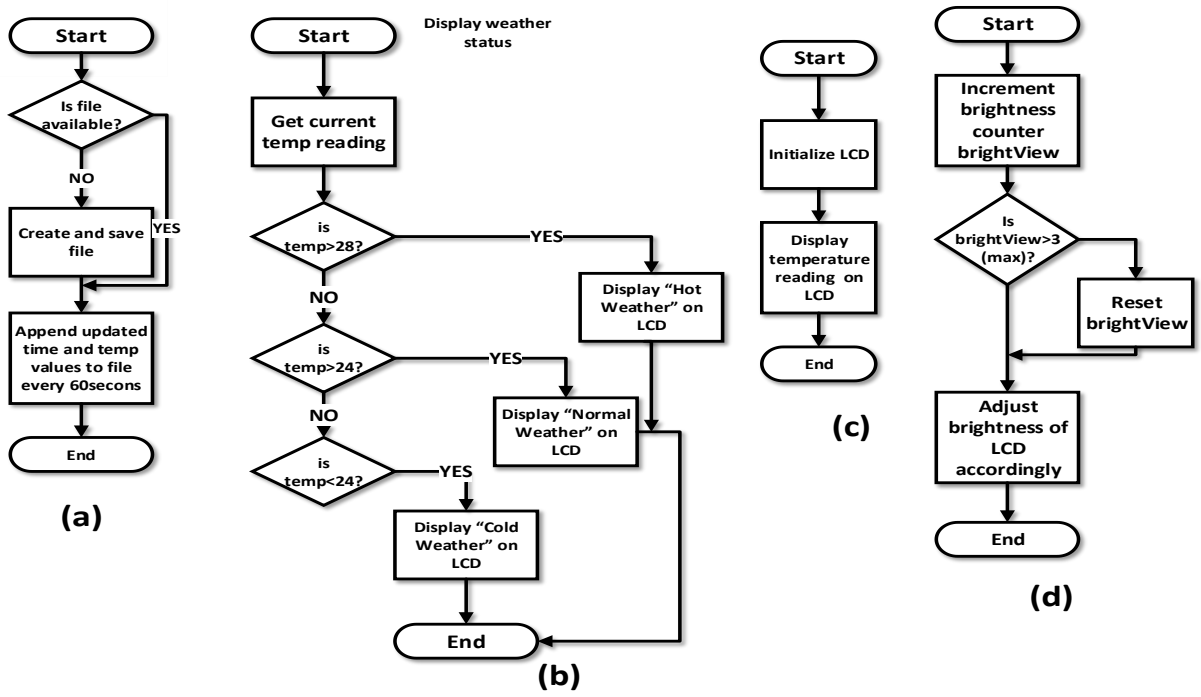


Figure 9: (a)Interrupt that writes data to file (b) Display weather status (c) Display temperature on LCD(d) Brightness control interrupt function

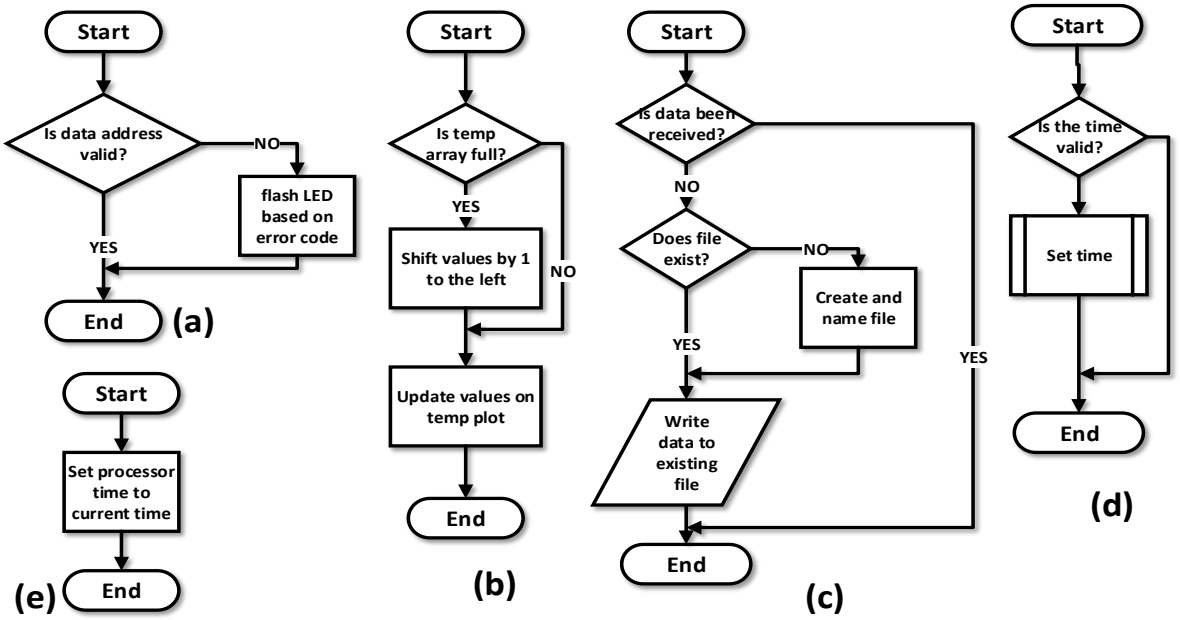


Figure 10: (a) temperror function flow chart (b)Plot Array flow chart (c) write data to file function flow chart (d) Serial ISR flow chart (e)Set time flow chart

# 4. Testing and Results

The Figure 13a shows the welcome screen of the temperature logger when powered ON, the welcome screen only displays when the temperature logger is powered for the first time, the welcome screen show name and student identification number, a user guide for navigating/operating the temperature logger. The welcome screen was set to display for 5 seconds, after which the normal/default view of the temperature logger is displayed. This view contains the temperature reading and corresponding time, weather status and logging status.

The normal view of the temperature logger is shown in figure 13a to 13e.The real-time temperature plot is shown in figure 13f. The vertical axis represents temperature axis while the time axis is on the horizontal axis. Figure 13b, 13c and 13d show the status of the weather on the LCD; when the temperature is below 24$^{o}$C the status indicates a cold weather and temperature reading between 24 -28$^{o}$C and above 28$^{o}$C show a status of normal and hot weather respectively. Similarly, logging status on the LCD when the data is being logged into the Excel Spreadsheet is shown in figure 13e. Also, a screenshot of the Excel Spreadsheet showing the temperature and the corresponding time is in figure 11.

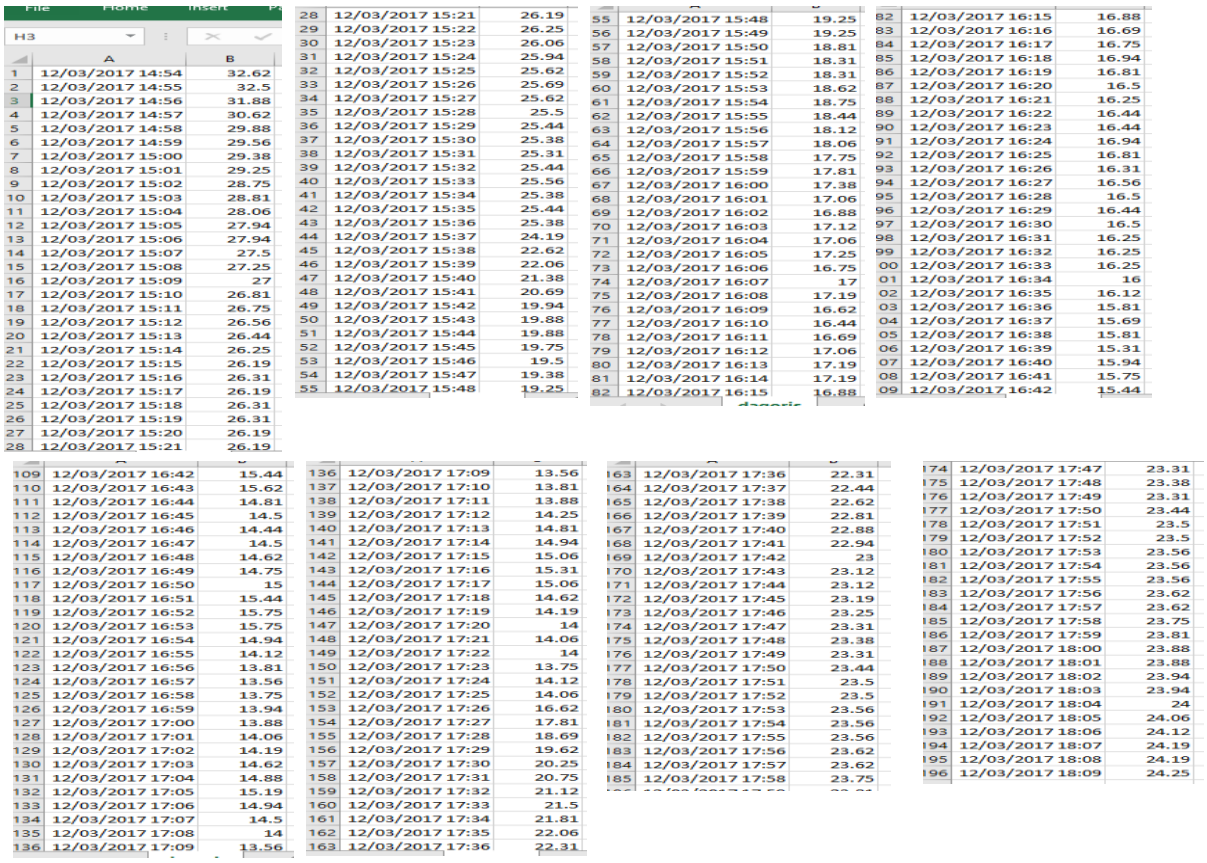

Figure 11: Logged in values using the Excel CSV file

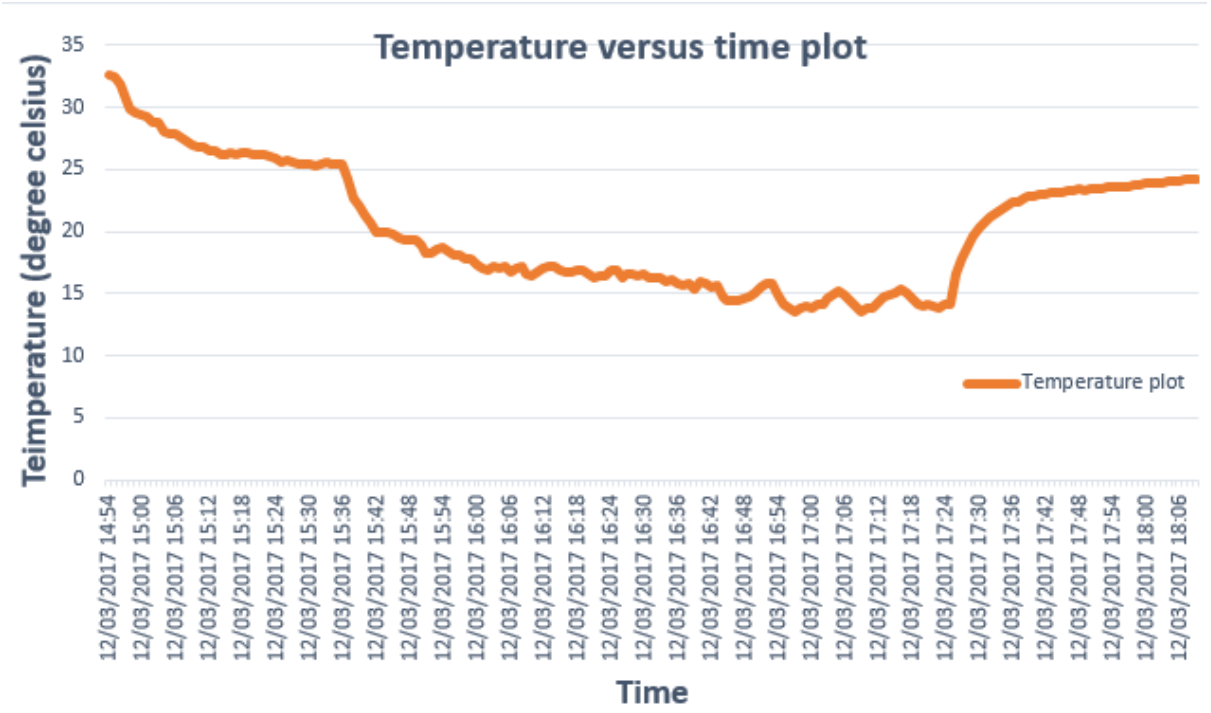


Figure 12: Temperature versus time plot

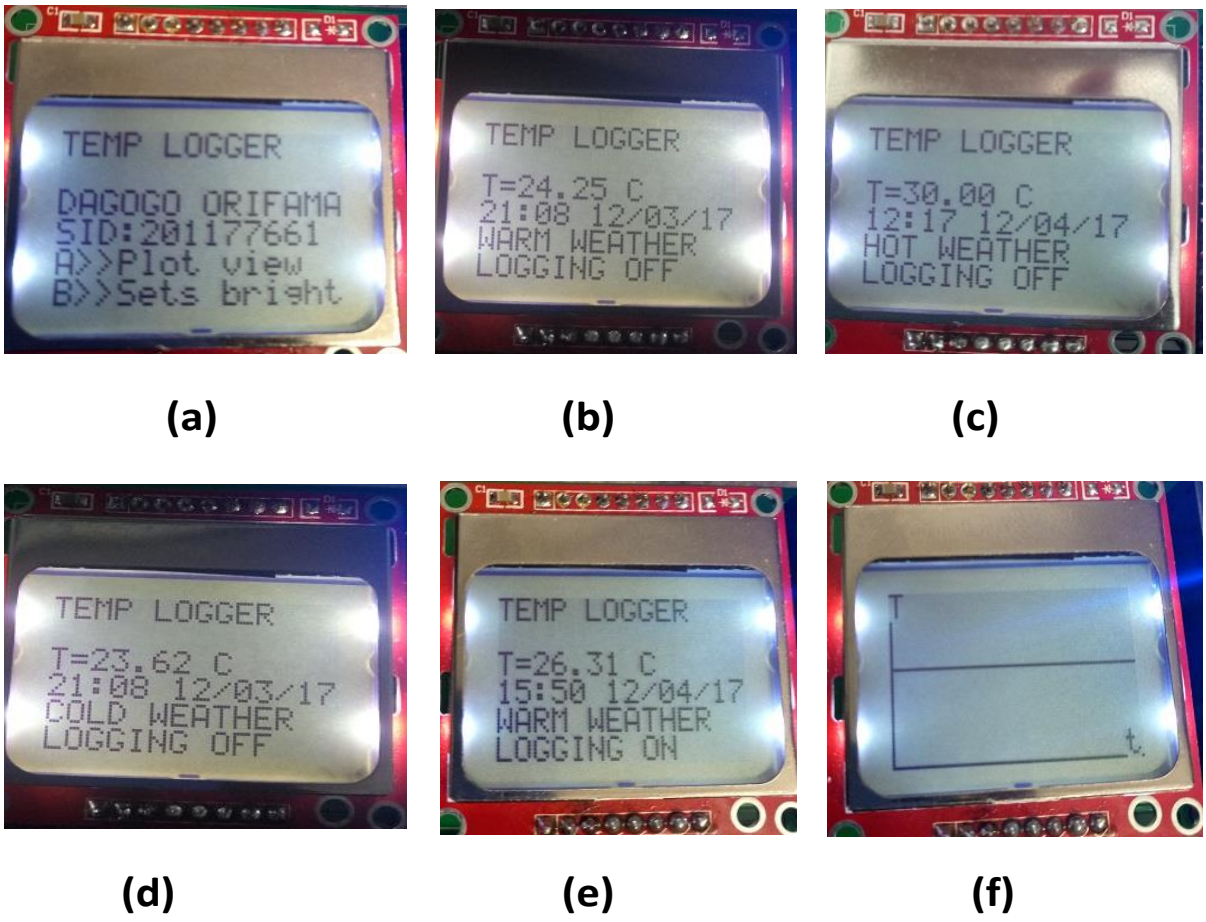


Figure 13: (a) Welcome screen of temperature logger (b) Normal view of temperature logger showing warm weather status (c) Normal view of

temperature logger showing hot weather status (d) Normal view of temperature logger showing cold weather status (e) Normal view of temperature logger showing logging on(f) Temperature plot view

## 4.2. Discussion on the result

The following were achieved after a careful exploration of the features of the NXP LPC1768 microcontroller:

- The ability to display the temperature and the corresponding time on the N5110 LCD, and show a temperature plot when the button A is pressed. The setPixel function was used for plotting the graph on the LCD and a Ticker function updates the temperature on the LCD every 0.5 seconds.
- The ability to control the logging of temperature for every minute into an excel CSV file with a switch and used of LED1, LED2, LED3 and LED4 as an indication of data being logged.
- Ability to determine the weather status and displaying on the LCD
- Controlling the brightness of the LCD when button B is pressed

A screenshot of the embedded temperature logger completely designed and fully functional is shown in figure 14, with a detailed procedure on its operation explained subsequently.

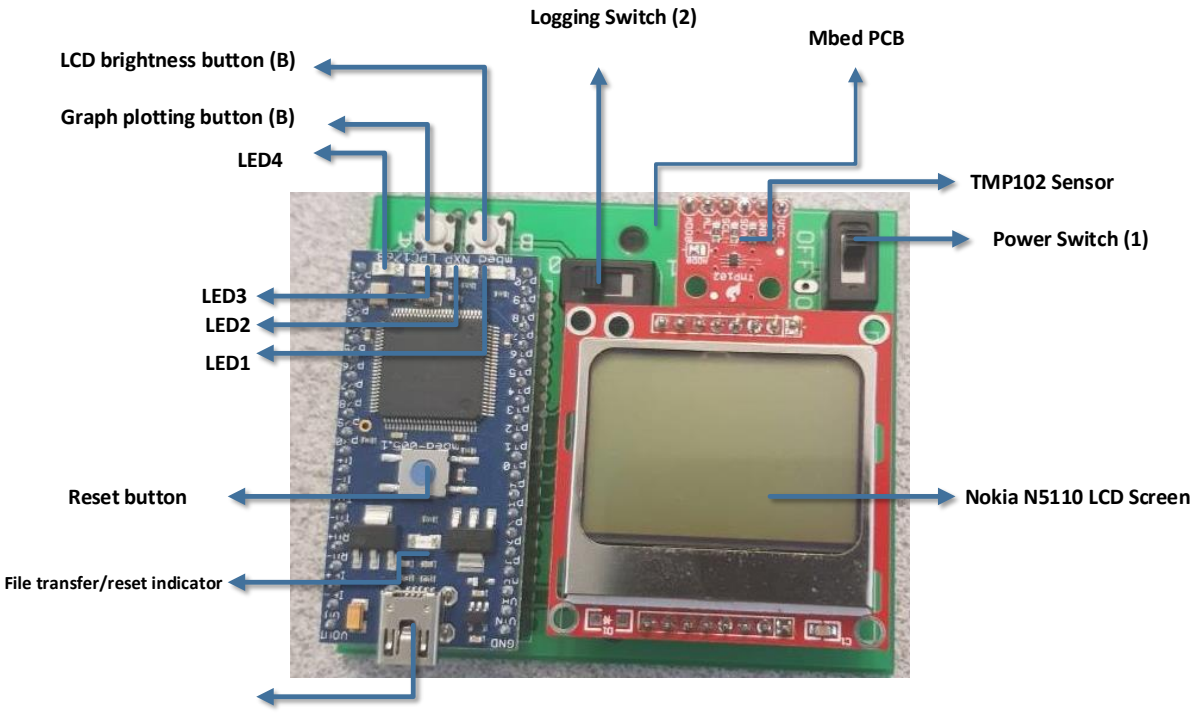


Figure 14: Embedded temperature logger showing features listing of its front view

## 4.3. Features and Usage

**Power switch (1):** The switch is shifted to the right to turn it ON. When ON, power is supplied from the 4AA batteries.

**Logging switch (2):** The switch is shifted to the right to turn it ON. It is referred to as the log switch for the temperature logger, this is because when the switch is ON, data (temperature and the corresponding time) is stored into the excel CSV file every 1 minutes. The LED1, LED2, LED3 and LED4 blinks as an indication that data logging is activated.

**Graph plotting button (A):** it is activated when pressed when this happens the temperature plot is displayed on the screen.

**Brightness control button (B):** it is activated when pressed, depending on the number of times it is pressed, the brightness is reflective accordingly. It was programmed to increase the brightness in step of 10%, 50% and 100% for 1,2 and 3 presses respectively

**Reset Button:** when pressed the microcontroller loads the recent .bin file copied into it.

**Serial port/USB:** The USB 2.0 cable when connected to the temperature logger from a computer, it serves as a source of power supply and for data transfer.

## 5. Conclusion

The embedded temperature logger has been implemented, this was done using the TMP102 temperature sensor. The TMP102 sensor was integrated with the NXP LPC1768 microcontroller using the inter-integrated circuit ($I^2C$) communication protocol. Values gotten from the TMP102 configuration register was scaled by a conversion factor of 0.0625. A Ticker function was used to plot data from the TMP102 sensor on the N5110 LCD, display data on the LCD and log data to file. The Nokia N5110 LCD was configured to communicate with the microcontroller through the SPI bus. The plotting of data on the LCD was achieved using an inbuilt function on the N5110 library known as setPixel. Similarly, LCD brightness control was implemented in this project.

The temperature range of operation for the embedded temperature logger was determined as 0 – 55$^{o}$C. The project can, therefore, be said conclusively to have met the set aim and objectives in terms of specifications.

## 6. Recommendations

Although the project has been implemented to meet specifications, improvements can be made to increase the project flexibility and user experience. The design could include a menu setting option that allows for setting the upper and lower temperature limit, Adjust the plotting scale of the graph on the LCD, adjust LCD brightness and the logging time instead of hand coding.

# References

[1] T. W. Rob Toulson, Fast and Effective Embedded System Design - Applying the ARM Mbed, Oxford: Elsevier, 2017.

[2] “Embedded System Basic Tutorial,” www.radio-electronics.com, [Online]. Available: http://www.radio-electronics.com/info/processing-embedded/embedded-systems/basics-tutorial.php. [Accessed 11 12 2017].

[3] “What is Microsoft Excel and What would I use it for,” www.thoughtco.com, [Online]. Available: https://www.thoughtco.com/what-is-microsoft-excel-3573533. [Accessed 12 12 2017].

[4] “Temperature Limits of LCD Displays,” www.lcd-enclosure.com, [Online]. Available: http://www.lcd-enclosure.com/temperature-limits-of-lcd-displays/. [Accessed 12 12 2017].

[5] “Energizer AA Alkaline Battery,” www.medicbatteries.com, [Online]. Available: https://www.medicbatteries.com/aa-energizer-battery-energize-aa. [Accessed 12 12 2017].

# Appendix

Display.cpp

```
#include "mbed.h"
#include "N5110.h"
#include "getData.h"
#include "display.h"

//    VCC,SCE,RST,D/C,MOSI,SCLK,LED
N5110 lcd(p7,p8,p9,p10,p11,p13,p21);  // LPC1768 - pwr from GPIO
InterruptIn brightSet(p17); // Brightness Interrupt subroutine
DigitalIn plotview(p16, PullUp); // Initialising button A as input to display plot view when button is pressed
DigitalIn defaultview(p17, PullUp); // Initialising button B as input to display default view when button is pressed
DigitalIn mylogstatus(p18);   // Initialising switch on Pin 18 as input, this is used to log data to file when ON

//Global variable
int brightVal=0;
int firstTime = 1;
int plot = 0;
int plot_array[84];


void togglePlot() {
      while(1) {
         if(!plotview) // checks if botton A is pressed
```

```
        {
          plot = 1;
        }
        if(!defaultview)  // checks if botton B is pressed
        {
          plot = 0;
        }
      }
}
void plotArray(){
   for (int i=0; i<83; i++)
   {
     plot_array[i] = plot_array[i+1];
   }
   plot_array[83] = (int)getTemperature();
}
void display1() {
   if (plot == 0) // show normal view
   {
   if (firstTime == 1)  // if the LCD is displaying for the first time
   {
   lcd.init();   //  initialise display
   lcd.clear();  // clear LCD
   lcd.setContrast(0.5);  // Sets LCD contrast
   brightSet.rise(&brightness);  // calls on the brightness function to set the brightness of LCD
   //if LCD is ON for the first time display a welcome message
   lcd.printString("TEMP LOGGER", 0, 0);
   lcd.printString("DAGOGO ORIFAMA", 0, 2);
   lcd.printString("SID:201177661", 0, 3);
   lcd.printString("A>>Plot view", 0, 4);
   lcd.printString("B>>Sets bright", 0, 5);
   lcd.refresh();
   wait(5.0);
   firstTime = 0;
   }
   lcd.clear();
   char tempbuffer [14];
   char timebuffer [15];
   int length1 = sprintf(tempbuffer, "T=%2.2f C", getTemperature()); // Print formatted
temperature to tempbuffer
   int length2 = getTime(timebuffer);      //get the corresponding time
//  lcd.printString("SID:201177661", 0, 0);
   lcd.printString("TEMP LOGGER", 0, 0);    //Display "TEMP LOGGER" on LCD
   if(length1<=14){
       lcd.printString(tempbuffer,0,2);     //Display formatted temperature in tempbuffer on LCD
   }

   if(length2<=14){
       lcd.printString(timebuffer,0,3);     //Display the corresponding time on LCD
   }
```

```
   float newTemp= getTemperature();     //get temperature in real time to determine weather
status
    if (newTemp>28.00){              //checks if newTemp is greater than maximum temperature
setting of 28 degree celsius
      lcd.printString("HOT WEATHER",0,4);  // if newTemp is greater than 28 display "HOT
WEATHER" on LCD
      }
   if (newTemp>24.00 & newTemp<28.00){  //checks if newTemp is within the range of 24 to 28
degree celsius
       lcd.printString("WARM WEATHER",0,4); // if newTemp is within the range of 24 to 28 degree
celsius, display "WARM WEATHER" on LCD
      }
   if (newTemp<24.00){              //checks if temperature is lower than normal temperature
setting of 24 degree celsius
       lcd.printString("COLD WEATHER",0,4);  // if newTemp is lower than 24 degree celsius, display
"COLD WEATHER" on LCD
      }

   if(mylogstatus)  // Checks if the switch on P18 is ON, if true
   { lcd.printString("LOGGING ON",0,5); // display "LOGGING ON" on LCD
   }
   else //if false
   { lcd.printString("LOGGING OFF",0,5); // display "LOGGING OFF" on LCD
   }

   lcd.refresh();
   }
   else  // plot graph
   {
      int y;
      lcd.clear();
      for (int x=0; x<=83; x++) {  //Plot temperature
         if (plot_array[x]>=47) {
            y = 0;
         } else {
            y = 47 - plot_array [x];
         }
         lcd.setPixel(x, y);
      }
      for (int x = 0; x <=83; x ++)
      {
         lcd.setPixel(x, 47); // set time axis
      }
      for (int y = 0; y <=47; y ++)
      {
         lcd.setPixel(0, y);    // plots the Temperature axis
      }
      lcd.printString("T",0,0);  //print Temperature axis label
      lcd.refresh();           //show changes
      lcd.printString("t",78,5); //print time axis label
```

```
        lcd.refresh();
    }
}
void brightness(){
    brightVal=brightVal+1;          //Sets brightness Variable
    if(brightVal==1){               //brightness level 1
    lcd.setBrightness(0.2);         //sets lcd brightness level to 20%
    wait(1.0);                      //delay for 1 second
    }
    if(brightVal==2){               //brightness level 2
    lcd.setBrightness(0.5);         //sets lcd brightness level to 50%
    wait(1.0);                      //delay for 1 second
    }
    if(brightVal==3){               //brightness level 3
    lcd.setBrightness(1.0);         //sets lcd brightness level to 100%
    brightVal=0;                    // reset brightness counter
    wait(1.0);                      //delay for 1 second
    }
}
```

Display.h

```
#include "mbed.h"

void display1(); // Function that displays data on the LCd
void plotArray(); // Function responsible for data on the LCD plot
void togglePlot(); //Function which determines the view to display based on wheather button A is
pressed or not
void brightness(); // Fucntion that controls the LCD backlight
```

getData.cpp

```
#include "mbed.h"
#include "getData.h"

BusOut mytempleds(LED3, LED4); // Function that declares LED 3 and LED4 as output
I2C tmp102(p28,p27);          // SDA, SCL

///hang in infinite loop flashing error code
void temperror(int code){
    while(1){
        mytempleds=0;  // OFF LED3 and LED4
        wait(0.25);    // wait
        mytempleds=code;   // ON LED3 and LED4 base on the value 'code'
        wait(0.25);    // wait
}
}
void initTMP102()
{
 tmp102.frequency(400000); // set bus speed to 400 kHz
 int ack; // used to store acknowledgement bit
```

```
 char data[2]; // array for data
 char reg = CONFIG_REG; // register address
 //////// Read current status of configuration register ///////

 ack = tmp102.write(TMP102_W_ADD,&reg,1); // send the slave write address and the
configuration register address
 if (ack)
 temperror(1); // if we don't receive acknowledgement, flash error message
 ack = tmp102.read(TMP102_R_ADD,data,2); // read default 2 bytes from configuration register
and store in buffer
 if (ack)
 temperror(2); // if we don't receive acknowledgement, flash error message
 ///////// Configure the register //////////
 // set conversion rate to 1 Hz
 data[1] |= (1 << 6); // set bit 6
 data[1] &= ~(1 << 7); // clear bit 7

 //////// Send the configured register to the slave ////////////

 char tx_data[3] = {reg,data[0],data[1]}; // create 3-byte packet for writing (p12 datasheet)
 ack = tmp102.write(TMP102_W_ADD,tx_data,3); // send the slave write address, config reg
address and contents
 if (ack)
 temperror(3); // if we don't receive acknowledgement, flash error message

}

float getTemperature()
{
 int ack; // used to store acknowledgement bit
 char data[2]; // array for data
 char reg = TEMP_REG; // temperature register address
 ack = tmp102.write(TMP102_W_ADD,&reg,1); // send temperature register address
 if (ack)
 temperror(5); // if we don't receive acknowledgement, flash error message
 ack = tmp102.read(TMP102_R_ADD,data,2); // read 2 bytes from temperature register and store
in array
 if (ack)
 temperror(6); // if we don't receive acknowledgement, flash error message
 int temperature = (data[0] << 4) | (data[1] >> 4);
 return temperature*0.0625;
}

int getTime(char *buffer) {
   time_t seconds = time(NULL); // get current time
 // format time into a string (time and date)
   int length = strftime(buffer, 15 , "%R %D", localtime(&seconds));
   return length;
}
```

getData.h

```
#include "mbed.h"
//Temperature addresses for ADD0 connected to ground
#define TMP102_ADD 0x48
#define TMP102_R_ADD 0x90
#define TMP102_W_ADD 0x91
//Register addresses
#define TEMP_REG 0X00
#define CONFIG_REG 0X01
#define THIGH_REG 0X03
#define TLOW_REG 0X02


void temperror(int code); // flashes the LED3 and LED4 if data is not read from the temperature sensor
void initTMP102();      // function that initialises the temperature
float getTemperature();  //subroutine that leads to the acquisition of the temperature from sensor
int getTime(char *buffer); // function that gets and format current time and date
```

main.cpp

```
#include "mbed.h"
#include "getData.h"
#include "display.h"
#include "rtc.h"

Ticker file, display, plotdata; // timer that control temperature acquisition, display of data to screen and graph plotting
BusOut mywriteleds(LED1, LED2, LED3, LED4); // Initialising LED1, LED2, LED3, LED4 as output
LocalFileSystem local("local"); // create local filesystem
DigitalIn mylogswitch(p18);     // sets Switch connected to P18 as input
void writeDataToFile();         // Function to write data to file

int main() {
   // void rtcTime(1511954097); //current unix time stamp
   file.attach(&writeDataToFile, 60.0); // Log data (time, data and temperature) to file every 60 seconds
   display.attach(&display1, 0.5);     // Display new temperature reading from sensor every 0.5seconds
   plotdata.attach(&plotArray,1.0);    // send new data every 1seconds to be plotted on the graph
   togglePlot();                    // toggle between default view and plot view
}
void writeDataToFile(){
   if(mylogswitch) { // if switch is ON, log data to file

float temperature = getTemperature();
 char timeBuffer [15];
 getTime(timeBuffer);
 mywriteleds = 15; // turn on LEDs for feedback
```

```
 FILE *fp = fopen("/local/Log.csv", "a"); // open 'log.csv' for appending
 // if the file doesn't exist it is created, if it exists, data is appended to the end
 fprintf(fp,"=\"%s\", %.2f\r\n",timeBuffer,temperature); // print string to file
 fclose(fp); // close file
 mywriteleds = 0; // turn off LEDs to signify file access has finished
}
}
```

rtc.cpp

```
#include "mbed.h"
#include "rtc.h"

Serial serial(USBTX,USBRX);
int setTimeFlag = 0; // flag for ISR
char rxString[16]; // buffer to store received string
void rtcTime()
{
 serial.attach(&serialISR); // attach serial ISR
 char buffer[30]; // buffer used to store time string

 set_time(1512223589); // initialise time to 2nd December, 2017
 while(1) {
 time_t seconds = time(NULL); // get current time
 // format time into a string (time and date)
 strftime(buffer, 30 , "%X %D", localtime(&seconds));
 // print over serial
 serial.printf("Time = %s\n",buffer);
 wait(1.0); // delay for a second

 if (setTimeFlag) { // if updated time has been sent
 setTimeFlag = 0; // clear flag
 setTime(); // update time
 }
 }
}
void setTime() {
 // print time for debugging
 serial.printf("set_time - %s",rxString);
 // atoi() converts a string to an integer
 int time = atoi(rxString);
 // update the time
 set_time(time);
}
void serialISR() {
 // when a serial interrupt occurs, read rx string into buffer
 serial.gets(rxString,16);
 // set flag
 setTimeFlag = 1;
}
```

rtc.h

```
#include "mbed.h"

void serialISR(); // ISR that is called when serial data is received
void setTime(); // function to set the UNIX time
void rtcTime(); // RTC time function
```