TEMPERATURE SENSOR AND LOGGER

Nnadika Chima Daniel
201077064 | ELEC5451M

CONTENTS

1. INTRODUCTION

Embedded systems in computer electronics are devices made to perform specific functions of the electrical and mechanical variety. They are found in or embedded onto larger electrical/electronic systems and mainly are capable of control and analysis of/for the larger system. [4]

This project entailed the use of such a system to perform the function of detecting, logging and displaying of current environment temperature. This embedded device is known as a Temperature Logger or Temperature Logging device. The embedded device makes use of a microcontroller device with CPU, memory and I/O peripheral characteristics. The microcontroller is embedded onto the device also containing a temperature sensor, LCD unit, buttons and switches, and a battery power unit.

The aim of the project was to configure the temperature sensor to be read by the micro-controller which then logs the data per minute into a CSV file in its memory. The Logged data is then to be displayed by the LCD unit on the device, with a plot of the real time data and logging state, with each state of the LCD to be navigable via button pushes. The switches on the device is also used to turn logging on/off.

2. HARDWARE DESCRIPTION

2.1 SYSTEM OVERVIEW

The temperature logging device designed and implemented in this project incorporated the use of a TMP102 temperature sensor to monitor the current temperature of the environment. This temperature reading was then sent to a micro-controller unit (NXP LPC1786) which logged the temperature data to an output file every one minute. For a simple user interface, an LCD display was also interfaced with the micro-controller on the device to show live logging and other features. A block diagram of the system is shown below (see appendix for schematic diagram).

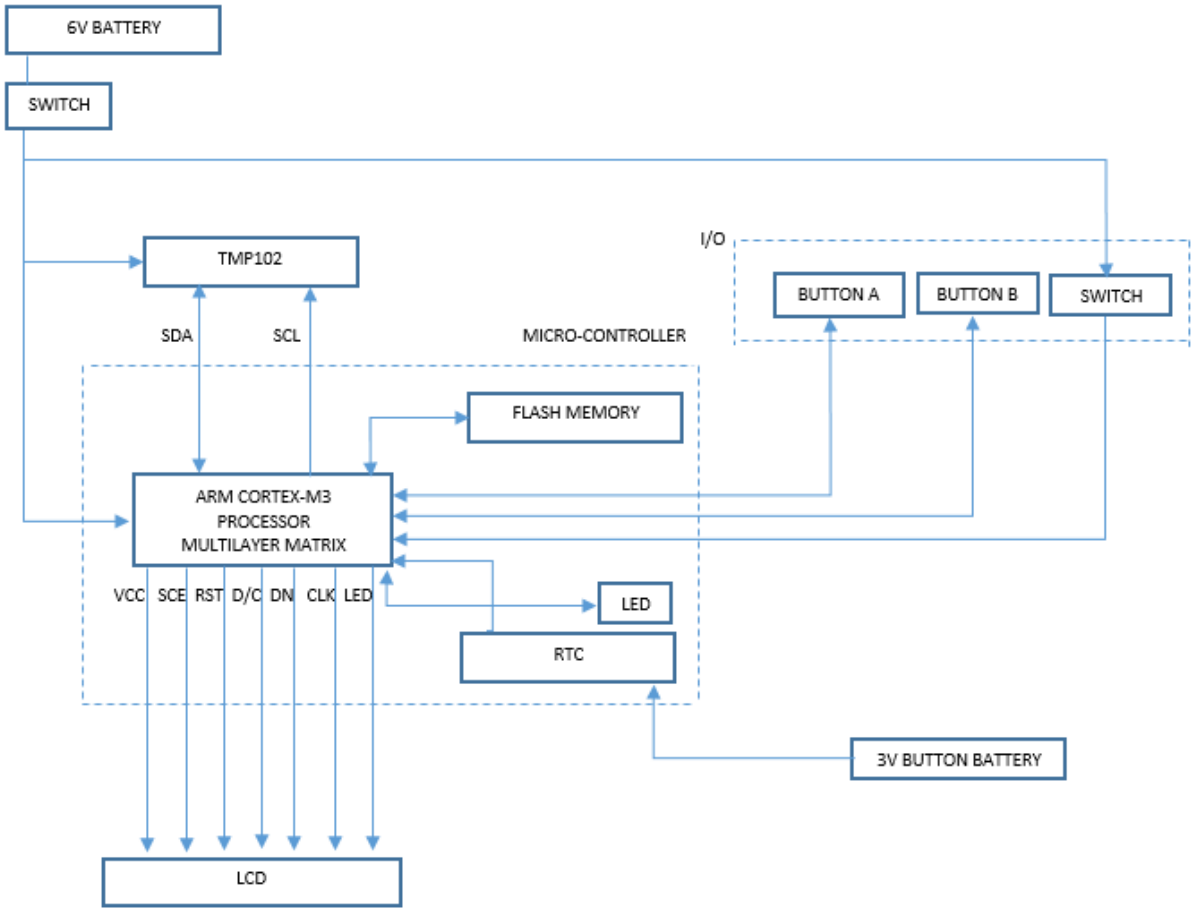


Figure 2.1: Block Diagram of Temperature Logger System

2.2 MICRO-CONTROLLER

The micro-controller unit used in this project was the LPC1786. The unit itself entails a vast array of components which cannot be extensively exhausted in this report. The components of the unit covered will be limited to those pertinent to the operation of the project.

The unit architecture boasts a 32-bit ARM Cortex-M3 processor (general Purpose, running on up to 120MHz frequency) with several interrupt control capabilities, low interrupt latency and consumes much lower power paired with more performance, and core buses that are able to be accesses at the same time. [1]

The unit also entails a flash memory of up to 512kB, an SRAM of 96kB, and an electrically erasable programmable read only memory (EEPROM) of 4032 bytes. The Unit also consists of an External memory controller (EMC) that helps interface with static memory devices, featuring read/write buffers to reduce data transfer latency. An LCD controller is also present in the unit, to support interfacing with LCD devices. [1]

The unit also boasts a USB interface, using a USB device/host/OTG controller, (device: enabling 12Mbits/data exchange rate. Host: enables full speed and low speed data exchange. OTG: integrates host, device and 1^2C interface, enabling dual-role functionality).

The micro-controller also entails a General Purpose parallel I/O which consists of programmable Pull up/down resistor with open-drain configurations. Also in the unit is an I^2C bus serial I/O controller with GPIO pins (400kB/sec rate), with a configurable master/slave system and programmable clock. Other elements of the micro-controller unit utilized in this project are its UARTs, SSP serial I/O controller, CAN controller, PWM, SD/MMC, Ticker Timer, and RTC backup register for clocking (32 KHz crystal oscillator providing 1Hz clock). [1] Below is a block diagram of the micro-controller system.

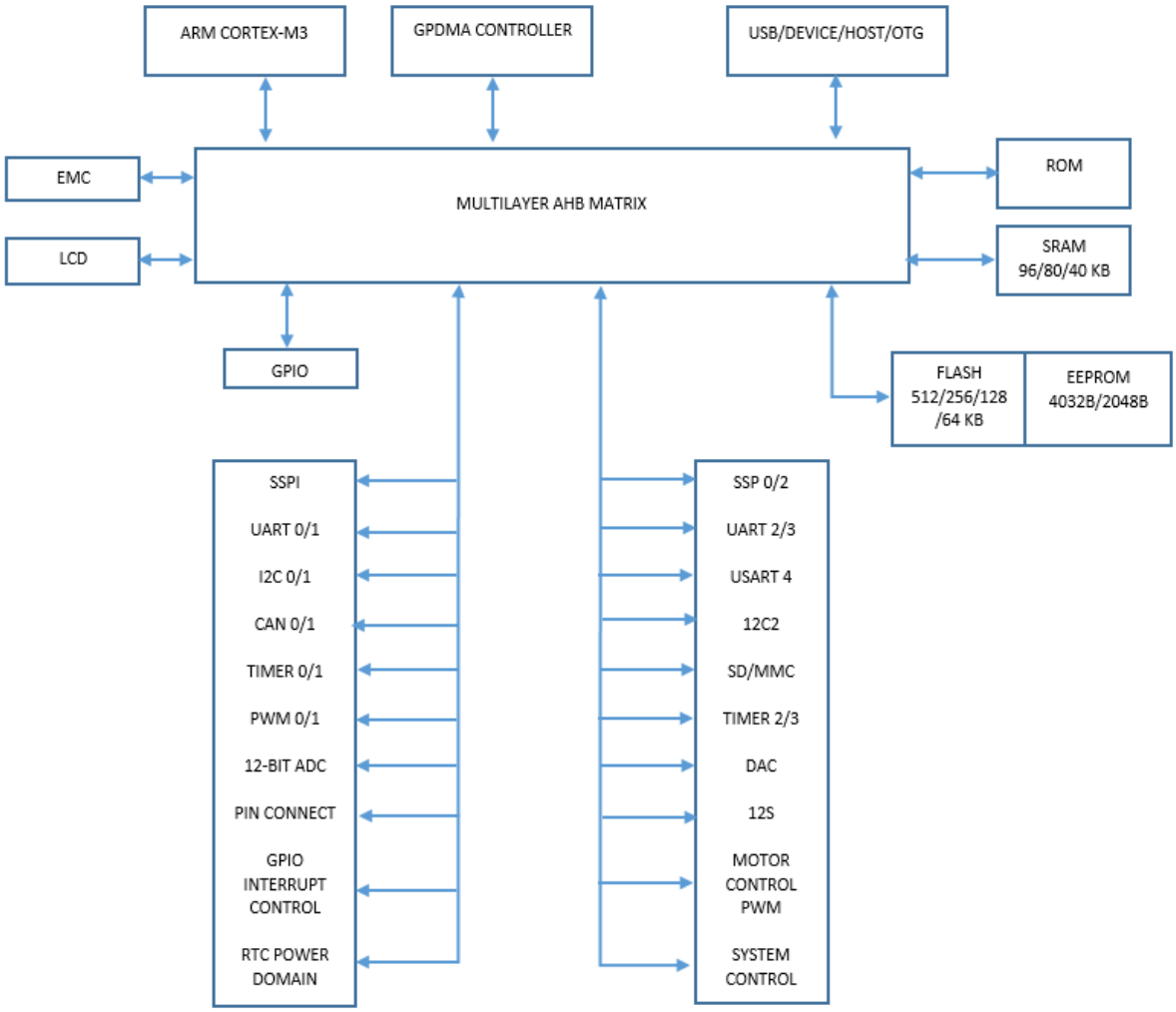


Figure 2.2: Block diagram of LPC1786 Micro-controller [1]

2.3 TEMPERATURE SENSOR

The temperature sensor used in this project was a TMP102 Digital temperature sensor with thermal path around its chip body (for detection). It is useful in a variety of applications, from household to industry. The temperature range of the device is between -40°C to 125°C. It consist of 6 pins, SDA (Serial data), V+ (Supply Voltage), SLC (Serial Clock) ADD0 (Address Select), ALERT (over temperature Alert), and GND (Ground). [2]

Using a 12-bit temperature binary format, the temperature data detected is stored in device temperature register, with one bit of the data representing 0.0625°C. The device uses a slave configuration on the two bus system; Serial Data (SDA) and Serial Clock, (SCL), transmitting at a rate of 1KHZ to 400KHZ at fast mode, and 1KHZ to 2.85MHZ at high-speed mode. The Sensor data rate and temperature detection rate is set in its configuration register (up to 8HZ). The addressing system functions by connecting ADD0 pin to 4 other pin configurations on one bus (SDA, SLC, V+ and GND).The pin used in this project for read/write addressing is the GND pin (see table for pin addresses). [2]

ADDRESS PIN CONNECTION	DEVICE 2 WIRE ADDRESS
GND	1001000
V+	1001001
SDA	1001010
SLC	1001011

Table 2.1: Showing ADD0 address pins

An external device (master) starts the transfer process for read/write operations by sending the SCL signal in the bus with a START/STOP. This signal initiates an address protocol to configure the slave device. The address protocol entails a byte value, with each bit passed on a rising edge of the clock. At the ninth clock edge, an acknowledgment is sent via the slave with a low in the SDA signal. The data is then sent over 8 SCL cycles and an acknowledge bit. If a change in SDA occurs at a High in the SCL, it is interpreted as either a START or a STOP operation. After the transfer is finished, a STOP is send by pulling the SDA from low to high during a high SCL. [2]

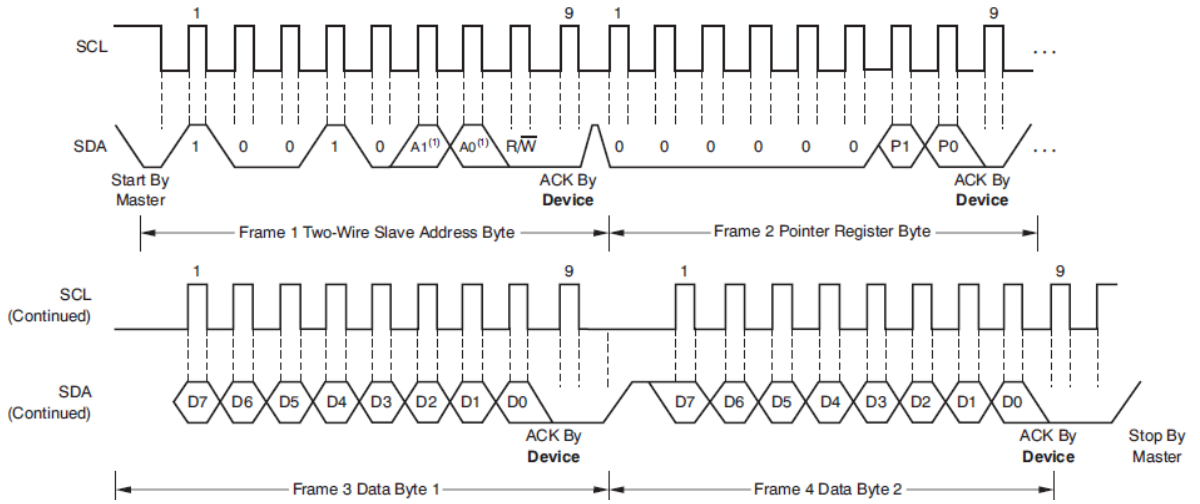


Figure 2.3: Showing typical 2-wire write operation timing diagram {Datasheet} [2]

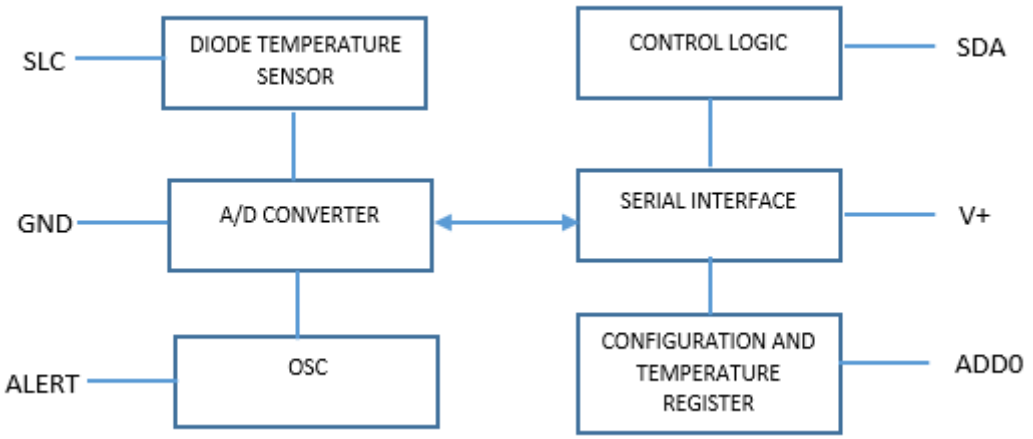


Figure 2.4: Block Diagram of Temperature Sensor System [2]

2.4 LCD DISPLAY

The LCD display used in this project is the Philips PCDB544 LCD commonly known as the Nokia N5110 LCD display module. The module consists of a CMOS LCD driver with a display of 48 rows and 84 columns with a Display Data RAM (DDRAM) of 48 by 84 bits. The system on-chip also contains the LCD supply voltage, intermediate LCD bias voltages and oscillators. The oscillators gives the display its clock signal. The bias voltages connect to the LCD driver with a sequence that corresponds to the data displayed on the LCD. [3]

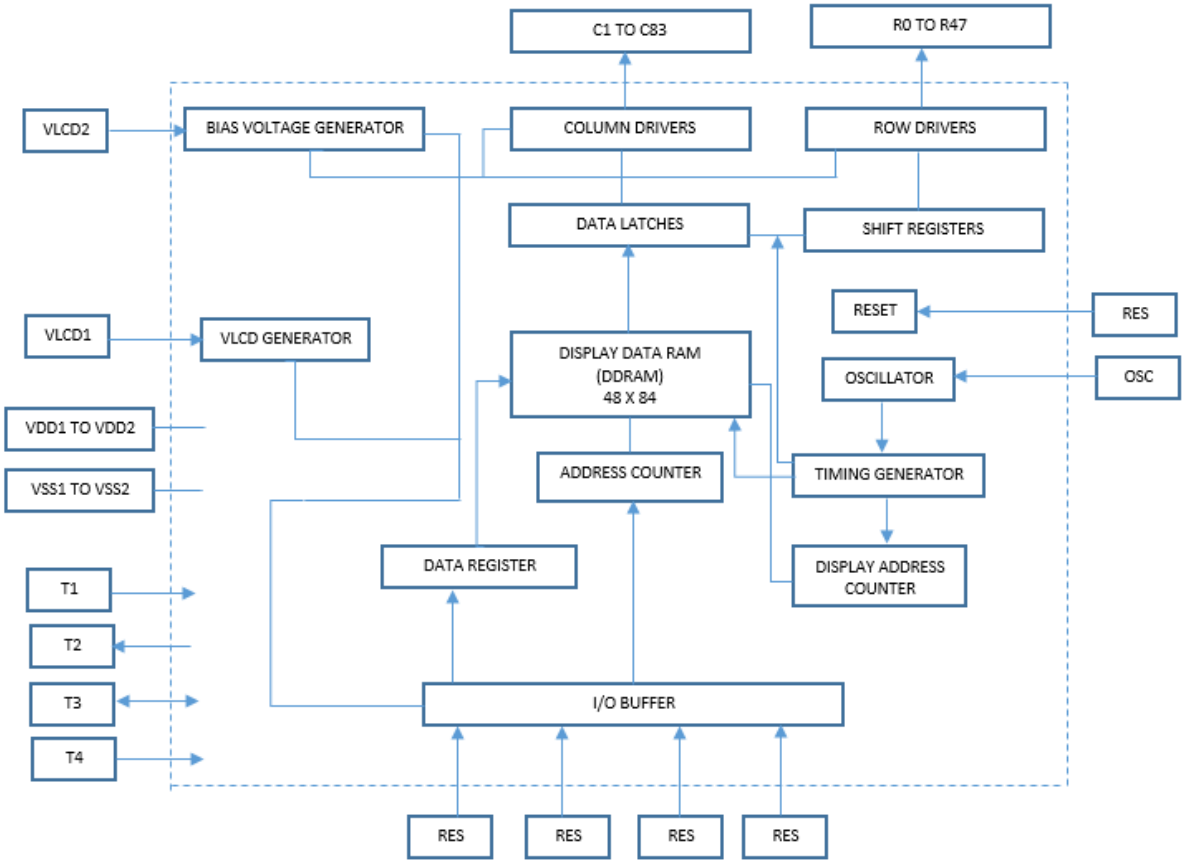


Figure 2.5: Showing LCD Block diagram [3]

2.5 POWER, LEDS AND SWITCHES

The micro-controller unit has embedded on its board 4 LEDs used in the project, mainly for error detection and logging state notification. Two switches are present in the Temperature Logging device, one for directing power to the LCD and the micro-controller board (Power on and Off), while the other is present as a spare for extra functionality. Two buttons are also present for extra functionality.

Power to the logging device stems either from 4 2XA batteries (1.5V each), or through a 5V USB power to the micro-controller board and the LCD unit. A small button battery cell is present in the temperature logging device is used to feed low power (3V) to the micro-controller board, to keep the RTC running even without external power.

3. SOFTWARE DESCRIPTION

3.1 SOFTWARE OVERVIEW

Programming the ARM micro-controller is made possible by the use of an online compiler toolchain for mbed that consists of all libraries and program classes needed to configure the specific micro-controller unit (using a C++ platform).

An overview of the program written to configure the temperature senor unit as a whole incorporated a multitude of functions all carrying out sets of duties. A list of all the functionalities implemented by the program are listed below

1. Error notification
2. Temperature sensor configuration and detection
3. Temperature Logging to file every minute.
4. Temperature Reading on LCD Display
5. Temperature Plotting on LCD Display

Each subroutine has within it details of configuring both the temperature sensor device, LCD display operation and use of GPIO pins for the buttons and switches. Details of all subroutines used was broken down into 4 sections, Temperature Sensor Configuration, Data Log Configuration, Display, and Main.

3.2 TEMPERATURE CONFIGURATION

Three subroutines are listed under this section: error(), initTMP102(), and temperature(). The error function is used by the other two as an acknowledgment check, it runs when either write or read actions to the temperature sensor don't return acknowledgment bits. The initTMP102 subroutine initializes and configures the sensor by setting conversion rates and bus speeds. The temperature subroutine gets the current temperature from the sensor.

Flowcharts of these three subroutines are draw below to show their operations.

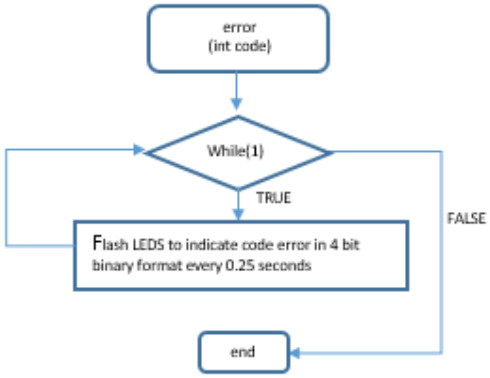


Figure 3.1: error function flowchart

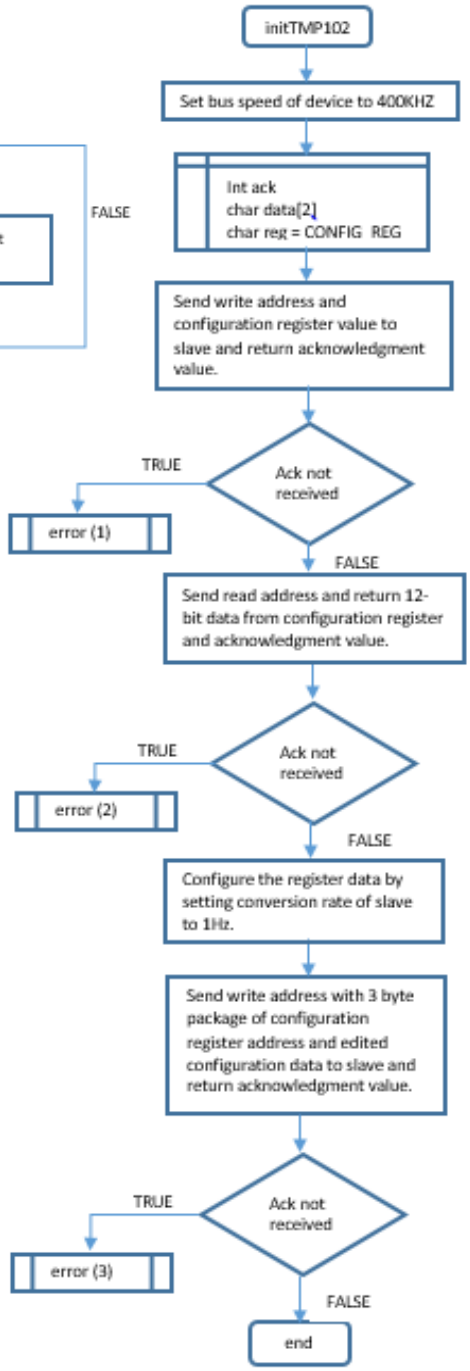


Figure 3.2: temperature sensor initialization function flowchart.

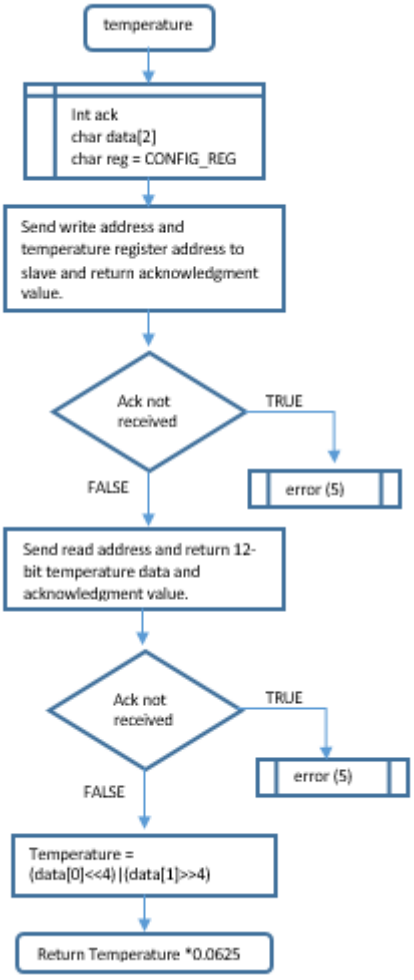


Figure 3.3: temperature read function flowchart

3.3 DATA LOGGING

The data logging of the device used 4 subroutines: logToFile(), timeTemp(), homeData() and plotData(). The logToFile subroutine takes the current temperature and time data and prints them into a CSV file with format (). The timeTemp subroutine gets the current temperature value from previously defined subroutine (temperature), and also gets the current time on the RTC of controller and sends them to the logToFile for logging. The homeData and plotData subroutines get and store current temperature and time values to be used for the LCD home and plot display operations. Flowcharts of each subroutine are shown below.

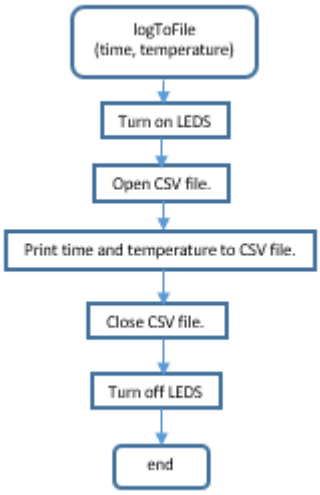


Figure 3.4: Log to File function flowchart

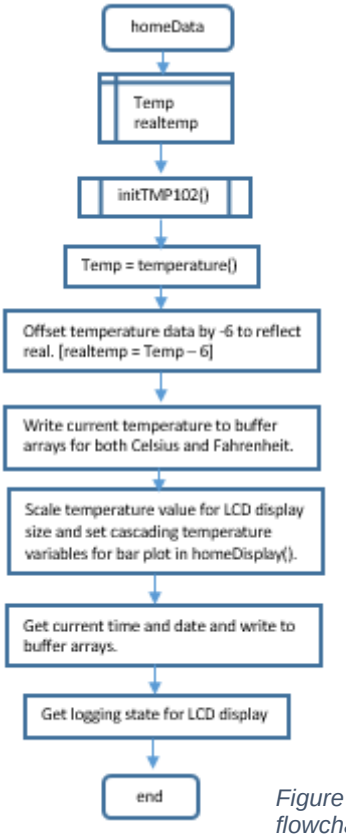


Figure 3.5: Home data function flowchart

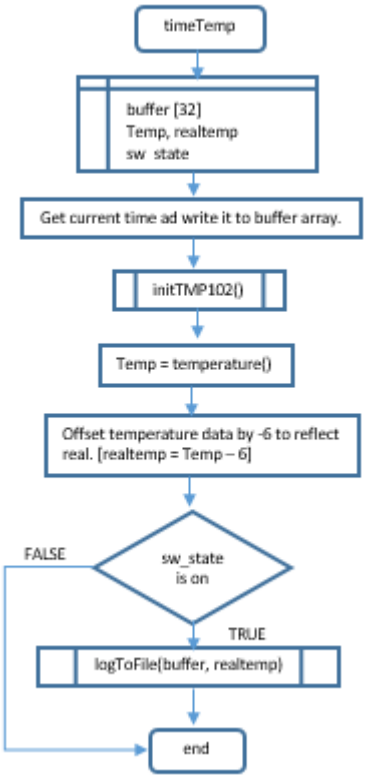


Figure 3.6: time and temperature logging function flowchart.

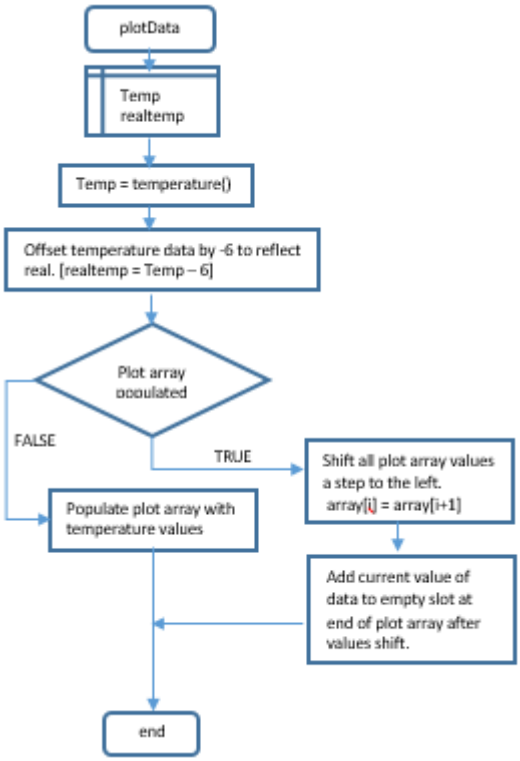


Figure 3.7: Plot Data function flowchart

3.4 DISPLAY

The Display operation to the LCD was programmed with 5 subroutines: homeDisplay(), plotDisplay(), aboutDisplay(), plotDetail() and homeDetail(). The home and plot Displays use set global variables configured periodically by prior data logging subroutines (homeData, plotData) to print and draw current time and temperature status on the home view, and real time plots on the plot view, on the LCD. The home and plot details subroutines print out descriptions of their corresponding prior views. The about Display subroutine simply prints out code authorship on the LCD. Flowcharts of each subroutine are shown below.

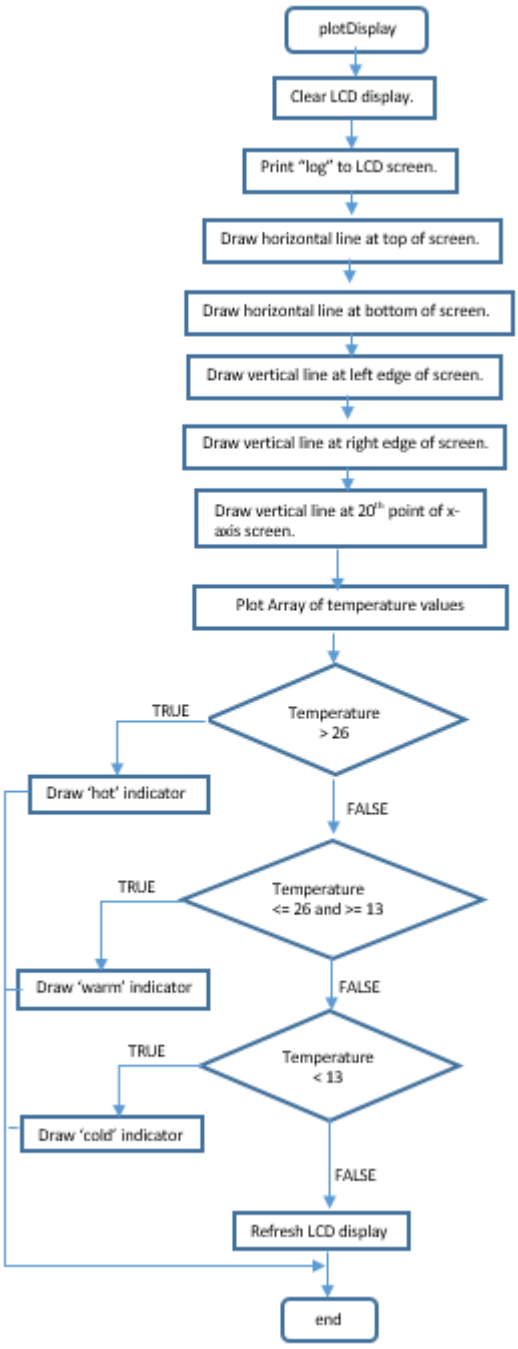


Figure 3.8: Plot Display function flowchart

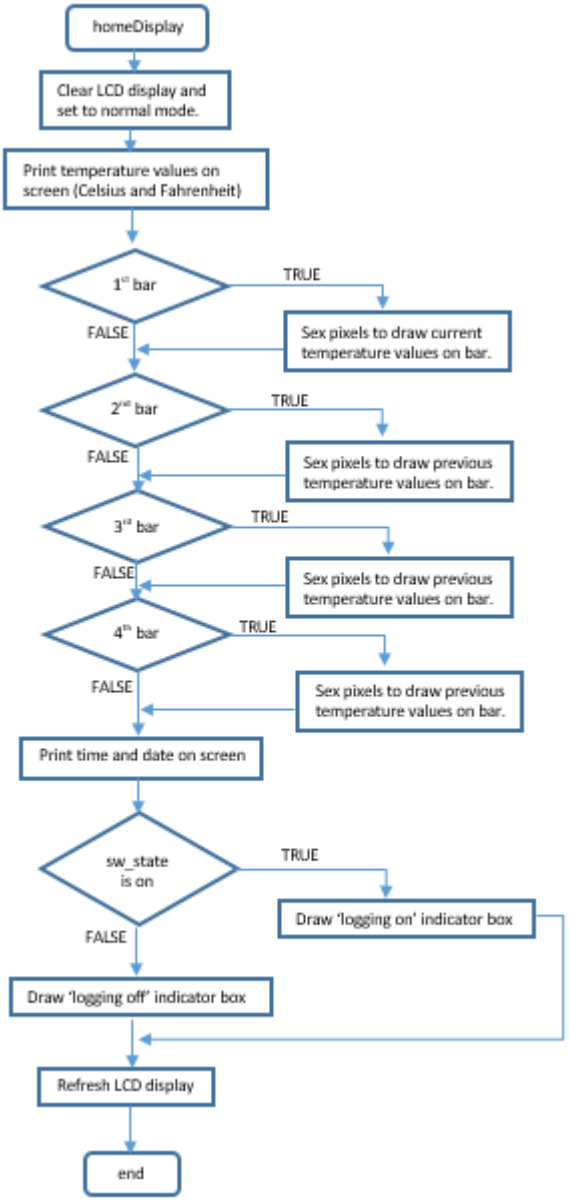


Figure 3.9: Home Display function flowchart

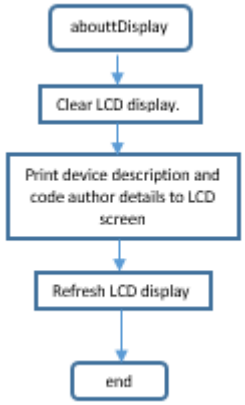


Figure 3.10: About Detail Function flowchart

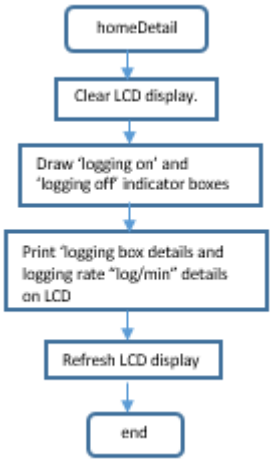


Figure 3.11: Home Detail function flowchart

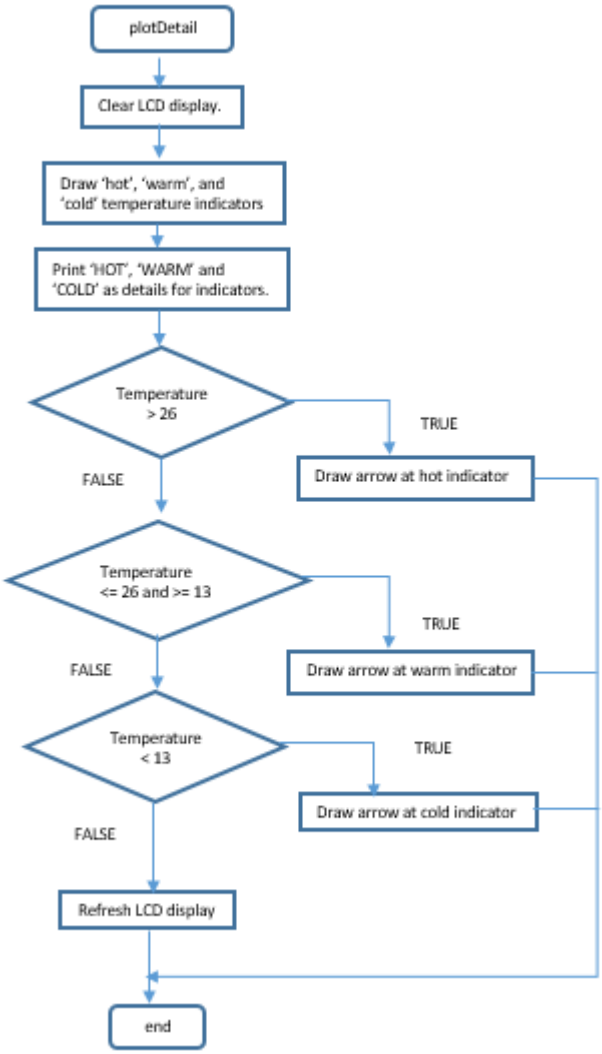


Figure 3.12: plot Detail Function flowchart

3.5 MAIN

The main function of the code takes all the prior defined subroutines and executes them in sequence. Global variables are used to store values used through the code by other subroutines (such as plot data array used by the plotData() and plotDisplay() subroutines, switch and button interrupts etc.). The data logging subroutines were attached to ticker interrupts to run alongside the code instead of within a while loop. The function initializes the LCD and prints out start messages before entering the while loop. In the while loop, each Display subroutine were placed in modes that were called once a button was pushed.

Below is a diagram of the flowchart showing the operation of the main function; containing all subroutine calls used in the code.

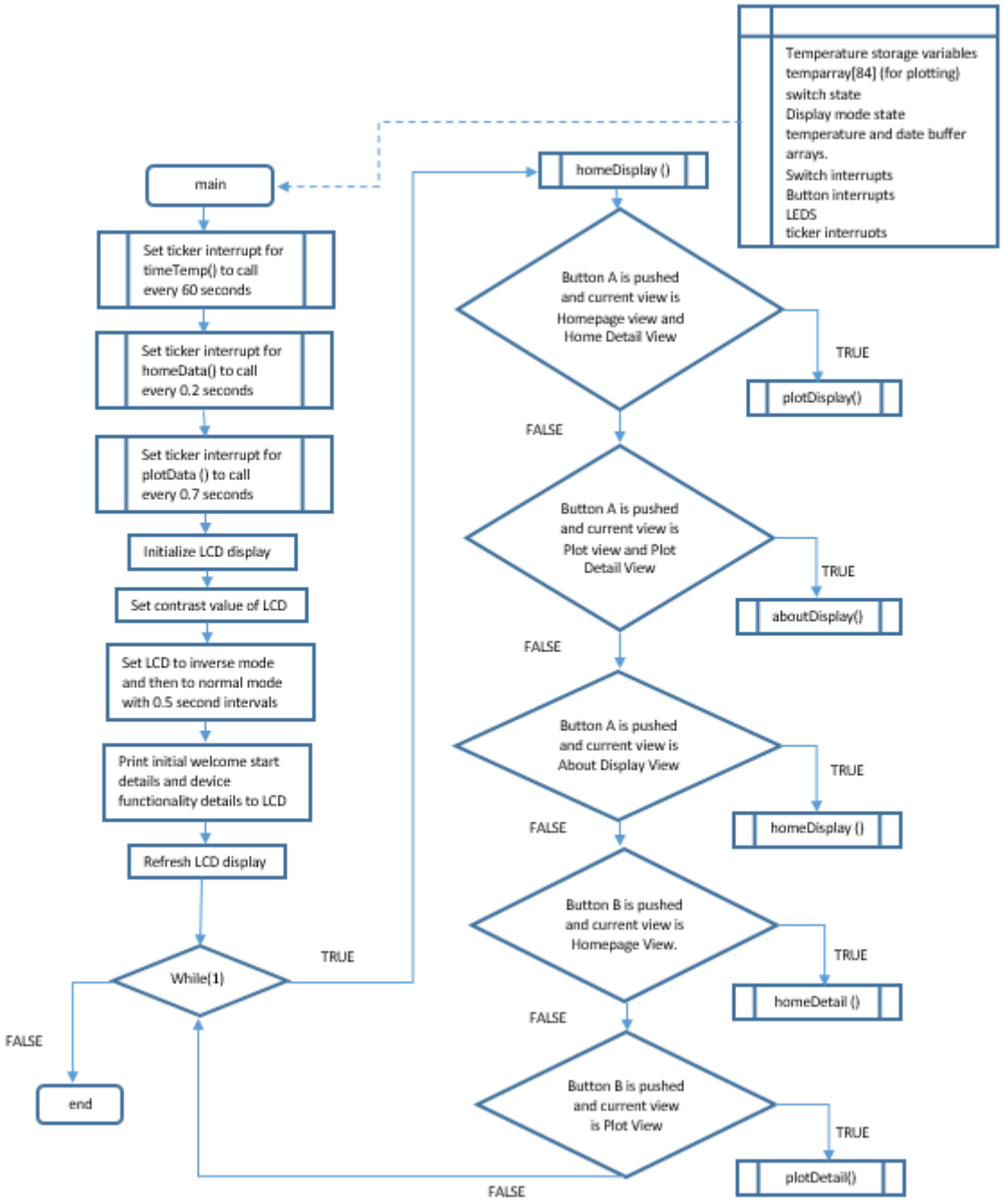


Figure 3.13: Main Function flowchart, showing global variables of code and all subroutines called for the program.

4. RESULTS AND TESTING

4.1 TEMPERATURE DATA LOG

The primary function of the temperature logging device in this project was the capacity to detect the current environmental temperature and log them into memory in a CSV file. The logging itself is to be done in one minute intervals and can be turned off and on with the use of a switch on the device. Meeting this objective for the device translates to the basis of the project.

The first aspect to the project was programming the micro-controller to read the temperature data from the TMP102 sensor and writing it to a file. The time interval with which the data is read was taken from the Real Time Clock of the controller. The Real time Clock Of the microcontroller was kept running at all time with the help of a 3V button battery power supply. The time on the clock was set to the current time programmatically (Using UNIX time format) and left to run indefinitely. To implement the switch control of the logging, the controller was programmed further to detect the state of the switch (Logic High or Low) and only log to the read temperature to file within the interval of the appropriate switch state. Using CoolTerm port terminal, the device capability was tested to see if it read both the temperature values, time and current switch state. The figure below shows a CoolTerm window showing these data.

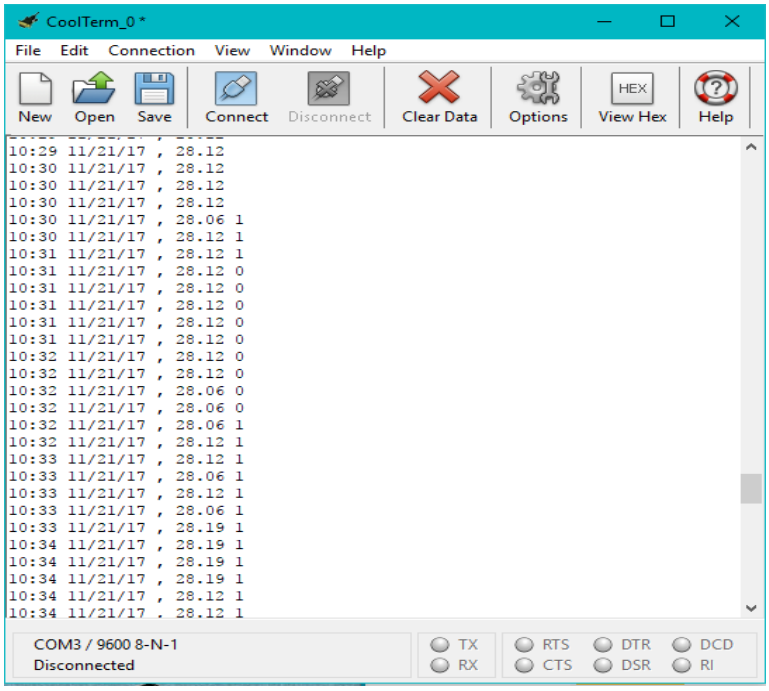
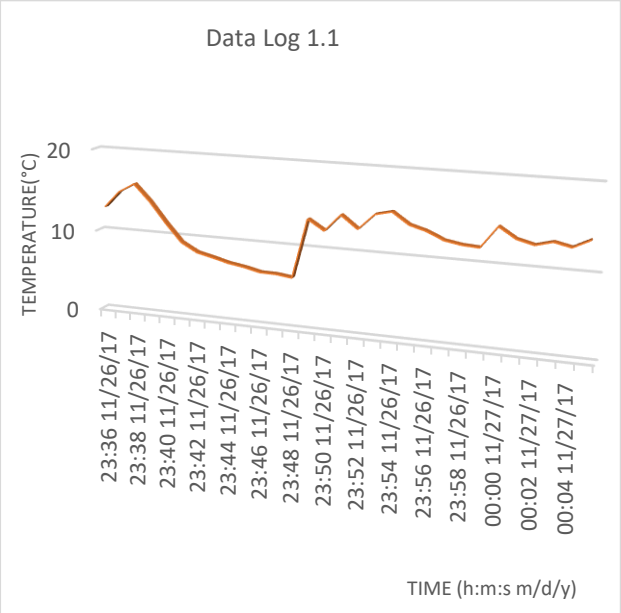


Figure 4.1: Showing CoolTerm Window for data log format test. The 1's and 0's show switch state testing.

The logging of the current temperature and time by the device was tested and implemented. The data was written to the CSV log file (with the format '10:20 20/11/2017, 20.60), only when the switch on the device was placed at a logic HIGH. The device was allowed to run for half an hour (30 data sets) to test the data logging to file capability. The CSV file created was extracted from the device via USB and opened in Excel. The values were plotted with current temperature against time. This

logging data is shown below with different plot formats (see appendix for another 30 minute data plot).



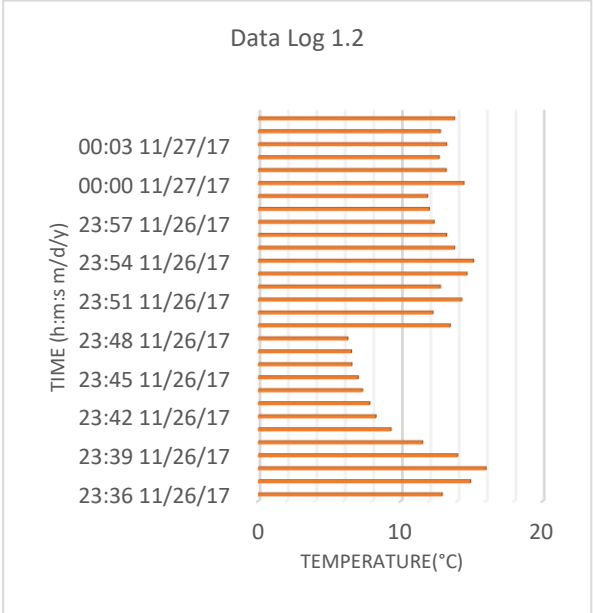


Figure 4.2a: Temperature data values with time. Format 1. *Figure 4.2b: Temperature data values with time. Format 2.*

4.2 DEVICE VIEW DYNAMICS

After the temperature and time logging to file was completely implemented, the programming of the LCD interface design was carried out by creating separate screens for user interaction with the device. The LCD screens were split into two main sections, namely the **Home** and **Plot** Views.

The Home views consisted of two views that corresponded with each other. The first view showed the initial status of the device. This status entailed the current time, data logging state, current temperature values in Degrees (Celsius and Fahrenheit), and a real time bar that changes its height fluidly as the environmental temperature changes in real time.

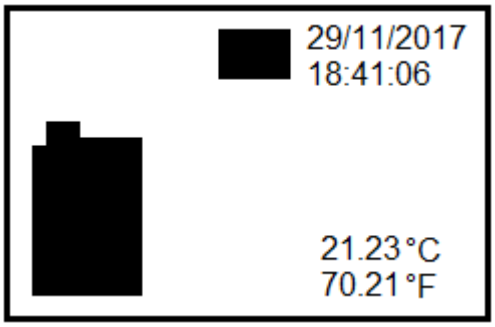


Figure 4.3: Showing Homepage View

The second view in this set showed the details of the previous view for user understanding. The logging state symbol was explained in this view and details of the logging time rate and use of switch to toggle logging (on/off).

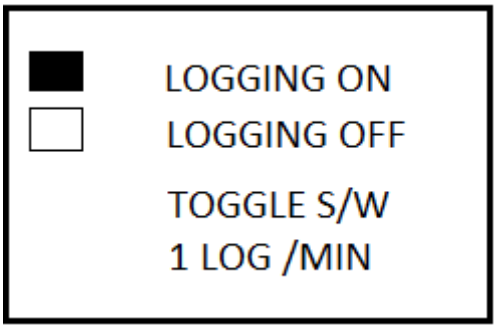


Figure 4.4: Showing Homepage details View.

The Plot Views were the second set of views programmatically implemented into the LCD. The first of the view set plotted a real time temperature log that shifted to the left every 0.7seconds. This gives the user a live look at the state of the temperature being read by the device. The view also has 3 indicators that come on to show how hot the environment is via an arbitrary threshold set. The values of temperature were adjusted in its ratio to fit the LDC screen Rows limit (48 X 64).

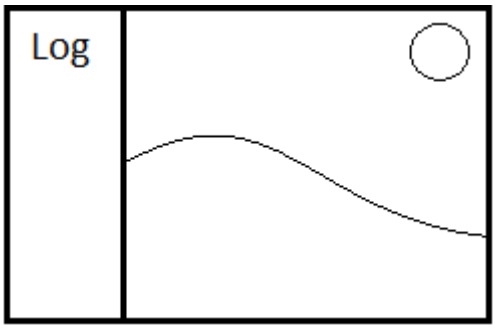


Figure 4.5: Showing Plot page View

The threshold sets shown in the first view is explained to the user of the device in the second view of the Plot View. The threshold set is to indicate HOT (via symbol) when the temperature is above 26°C, WARM when it is below 26°C and above 13°C, and COLD when it is below 13°C. This second view also prints the current temperature in degrees (Celsius).

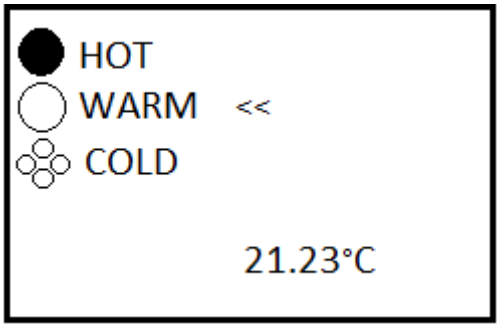


Figure 4.6: Showing Plot Page details view

Other view of the device are the **Welcome View** and the **About View** which are not dynamic views but shows simply the device name and authorship of the device.

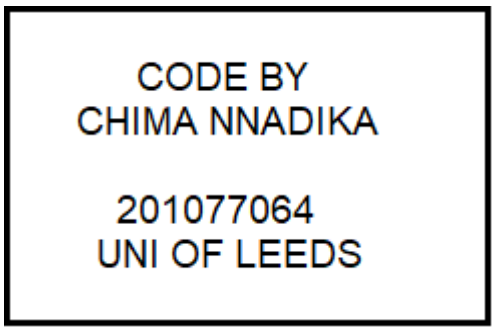


Figure 4.7a: Showing About View

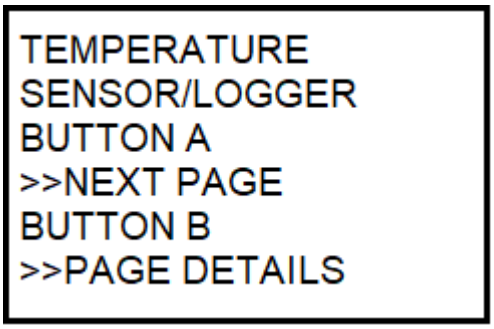


Figure 4.7b: Showing Welcome View

4.3 FINITE STATE MACHINE

The views of the device and logging of device are all capable of being navigated by the user via switches and buttons. The device has in its unit two buttons and one switch which control the state of the device. The finite state machine of the device is given below, showing how the buttons were integrated to control the LCD view states.

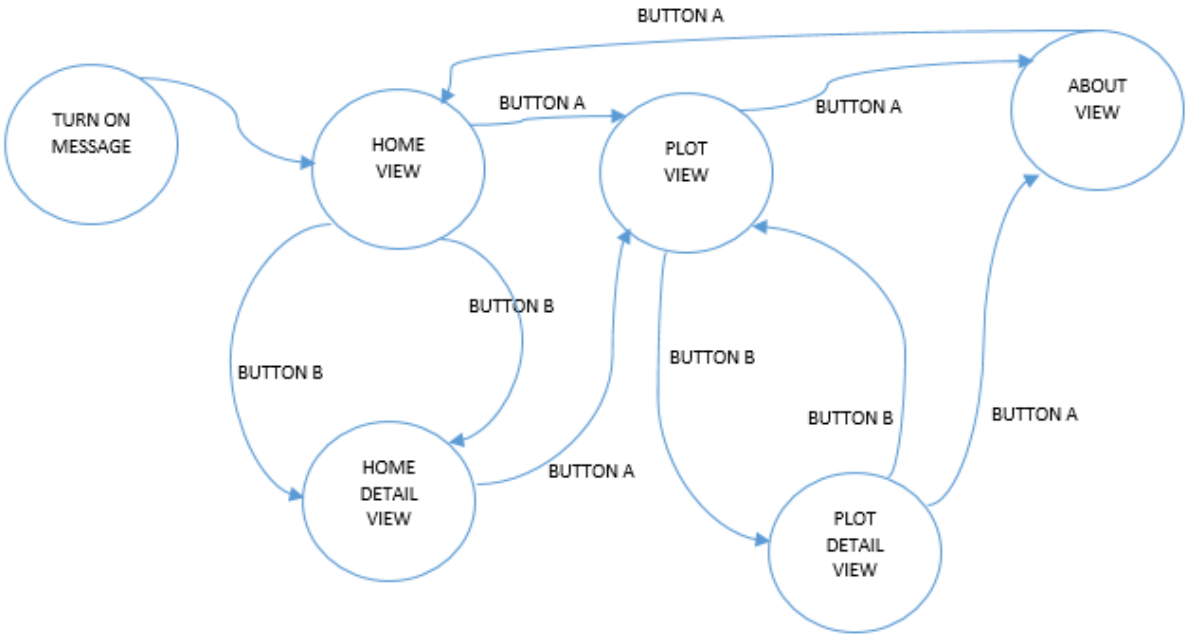


Figure 4.8: Showing LCD mode state-machine with Button functionality.

5. CONCLUSION

5.1 ACHIEVEMENTS

The goals set out for the project were all achieved, with the full use of all units in the temperature logging device and the printing to file of the current temperature and time. The buttons were fully implemented to navigate the LCD screen, acting as an interface for user input and interaction with the device. The use of switching to turn live logging on and off in the device was also implemented. The use of dynamic programming methods like ticker interrupts and function separation was also implemented. The device worked as intended, however, further work can be done on it.

5.2 SETBACKS AND FUTURE IMPLEMENTATIONS

The device is incapable of changing the rate of logging time intervals. This means that no matter what, it logs the current temperature at a constant time of 1 minute per second. The way the program is setup currently, ticker interrupts are set to run the specific logging function every 60 seconds, and that value cannot be changed within the code (since interrupts set themselves before the code enters an infinite loop. The logging function would have to be detached from the interrupt object and placed in a simple while loop with the interval variable changeable via button set by using a wait function. This method has its drawback due to wait functions not being reliable methods of constant time operation (program might take longer than intended to execute specific wait function). Certain variables and objects used to test the code are commented into the final code version (like serial bus for CoolTerm), and s few unused variables remain in the final code. Further code tampering to eliminate unnecessary memory allocation by the controller could have been done.

The device itself is a portable and useful unit that can be operated in isolated locations, logging temperature data for as long as the memory allocated allows. This means it has uses in industry, such as testing the temperature status of a sub-system, such as being used for analysis solar panels for power converter configurations. It can also be used for household applications like alarm systems, or as part of a temperature regulation system.

6. REFERENCES

[1] N. X. P. Semiconductors, “LPC178X/7X Data Sheet,” *Semicond. N X P*, vol. Rev. 5.5, 2016.

[2] T. Instrument and D. Sheet, “TMP102 Low-Power Digital Temperature Sensor With SMBus ™ and Two-Wire Serial Interface in SOT-563,” *Texas Instrum.*, 2015.

[3] SparkFun Electronics Data Sheet, “Graphic LCD 84x48 - Nokia 5110 - LCD-10168,” *SparkFun Electron.*, 1999.

[4] Heath, Steve, "Embedded systems design". EDN series for design engineers (2 ed.). Newnes. p. 2. ISBN 978-0-7506-5546-0. (2003).

7. APPENDIX

7.1 HARDWARE SCHEMATIC

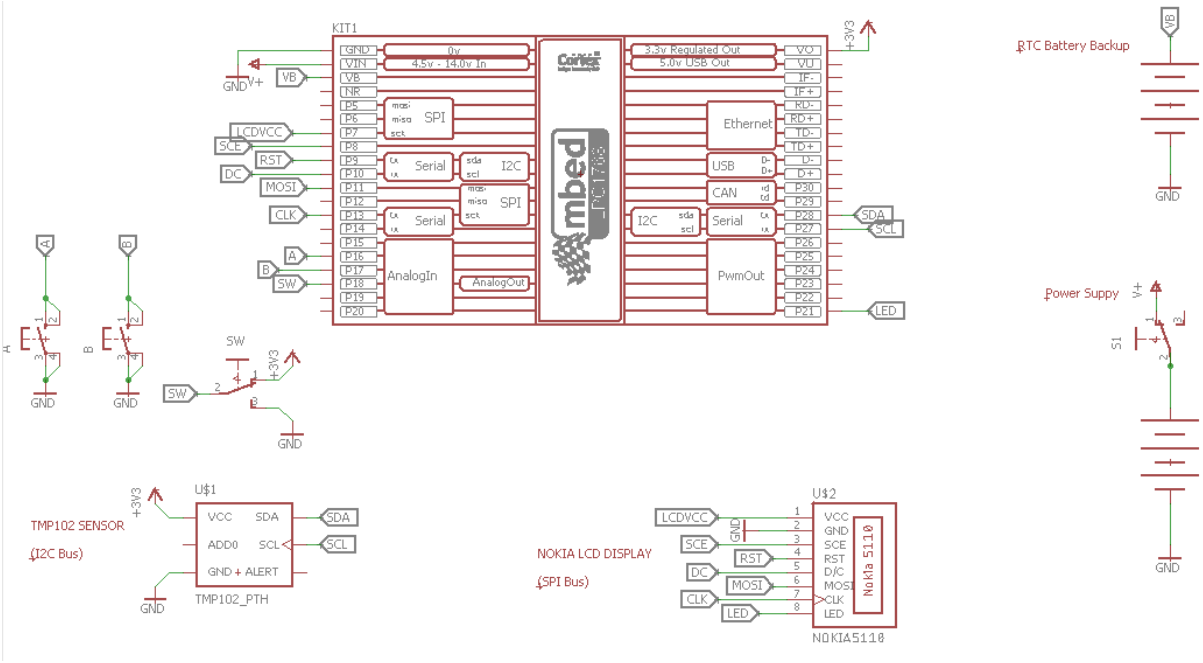


Figure 7.1: Showing full Temperature Logger System Schematic using Eagle Software.

7.2 DATA LOG

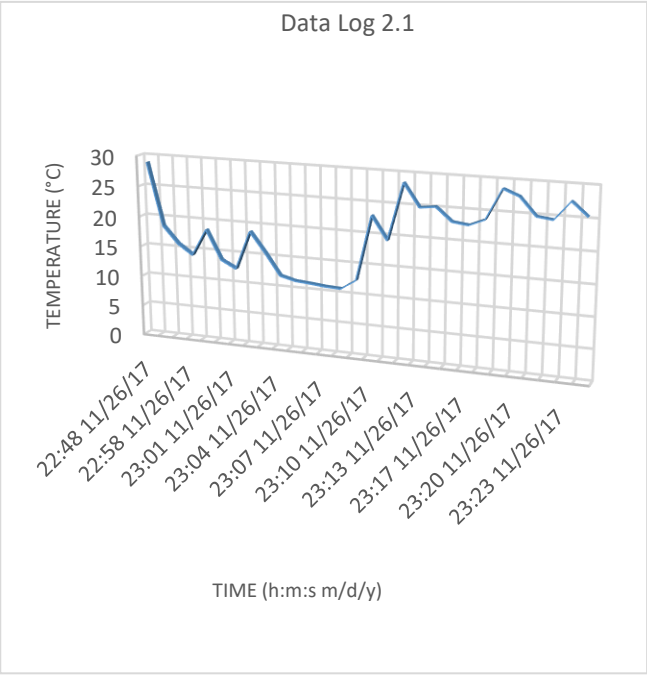


Figure 7.2a: Temperature data values with time. Format 1.

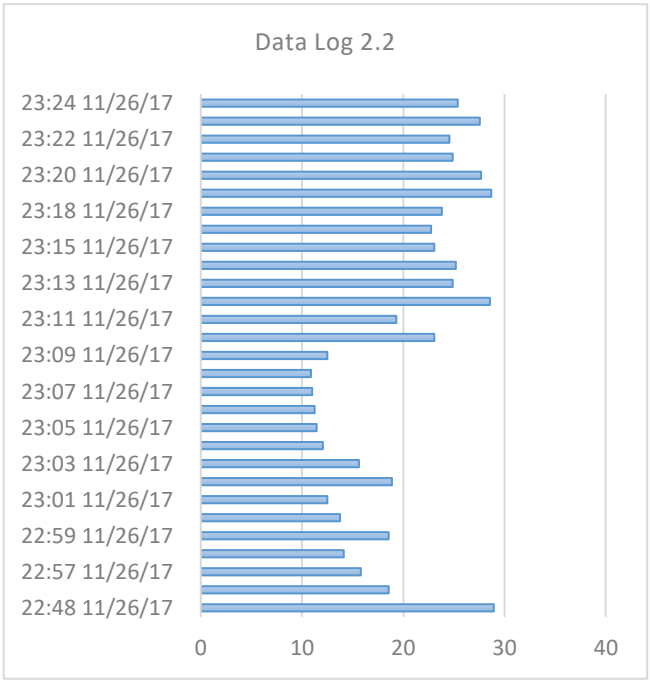


Figure 7.2b: Temperature data values with time. Format 1.

7.3 TEMPERATURE DEVICE CODE

```
/*Temperature sensor and Data Logging
    By Chima Daniel Nnadika
    201077064
*/

#include "mbed.h"
#include "N5110.h"


//Address for ADD0 connected to GND
#define TMP102_ADD 0x48
#define TMP102_R_ADD 0x91
#define TMP102_W_ADD 0x090

//Register Addresses
#define TEMP_REG 0x00
#define CONFIG_REG 0x01
#define THIGH_REG 0x02
#define TLOW_REG 0x03

//initialize timer interrupts to call analysis functions
Ticker getdat;
Ticker plotdat;
Ticker homedat;

//set serial I/O for debbugging
Serial serial(USBTX,USBRX);
//set I2C object for temperature data and clock (SDA, SLC)
I2C tmp102(p28, p27);

//set I/O buttons, Switches and LEDS
InterruptIn sw(p18);
DigitalIn buttonA(p16, PullUp);
DigitalIn buttonB(p17, PullUp);
BusOut leds(LED1, LED2, LED3, LED4);

//Power up LCD display and declare method
N5110 lcd(p7,p8,p9,p10,p11,p13,p21);
LocalFileSystem local("local"); // create local filesystem

//global variables used between functions to store time and temperature
//data for plotting and printing to LCD display
float tempo0, tempo1, tempo2, tempo3,tempo4, tempo5, tempo6; //temperature storage
variables for plot
float temparray[84]; //temperature plot array
int j = 0, sw_state; // plot counter and switch variabele
char tempBuffer[14],  tempBuffer2[14], timeBuffer[14], dateBuffer[14]; //data buffers to
store temperature and time
volatile bool isPushed = false; //boolean for storing button push instance
int mode = 0; //mode counter

//file logging function prototype
void logToFile(char t1[], float t2);


// hang in infinite loop flashing error code
void error(int code){
    while(1) {
        leds = 0;
        wait(0.25);
        leds = code;
        wait(0.25);
    }
```

```
}

//Initialize TMP102
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)
    error(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)
    error(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)
    error(3); // if we don't receive acknowledgement, flash error message
}

float temperature(){
    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)
        error(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)
        error(6); // if we don't receive acknowledgement, flash error message
    int temperature = (data[0] << 4) | (data[1] >> 4);
    return temperature*0.0625;
}


void logToFile(char *t1, float t2){
    leds = 15; // turn on LEDs for feedback
    FILE *fp = fopen("/local/log.csv", "a"); // open 'log.txt' for appending
    // if the file doesn't exist it is created, if it exists, data is appended to the end
    fprintf(fp,"=\"%s\" , %.2f\n",t1,t2); // print string to file
    fclose(fp); // close file
    leds = 0; // turn off LEDs to signify file access has finished
}

//This function gets the current time and temperature
//and send it t the logtofile function for logging
//
void timeTemp (){
    char buffer[32]; // buffer used to store time string
    float Temp, realtemp;
    int sw_state = sw.read();
    time_t seconds = time(NULL); // get current time
    strftime(buffer, 32 ,"%R %d/%m/%y", localtime(&seconds));
```

```
    initTMP102();
    Temp = temperature(); //ges temperature value from sensor
    realtemp = Temp-6; //adjusts read temperature to measured
    // print over serial
    //serial.printf("%s , %.2f %d\n",buffer,Temp,sw_state);
    while(sw_state){ //start log only when switch is on a high (1)
        logToFile(buffer,realtemp); //send data to logtofile function
        break;
    }
}

//this function gets current data for the home display
void homeData(){
    //data for temp on home screen
    float Temp, realtemp;
    initTMP102();
    Temp = temperature();
    realtemp = Temp-6;
    sprintf(tempBuffer,"%.2f%cC",realtemp,(char)167); //gets data instring format for lcd
    sprintf(tempBuffer2,"%.2f%cF",(realtemp*1.8)+32,(char)167); //gets farenheit value for
lcd

    //the tempo variables are used to draw bar on lcd
    //showing instantaneous temperature changes
    tempo0 = (47*realtemp)/60; //ratio of data to screensize for temperature bar
    tempo5 = tempo4;
    tempo4 = tempo3;
    tempo3 = tempo2;
    tempo2 = tempo1;
    tempo1 = tempo0;
    //get time data for home screen
    time_t seconds = time(NULL); // get current time
    strftime(timeBuffer, 14 , "%X", localtime(&seconds));
    strftime(dateBuffer, 14 , "%D", localtime(&seconds));
    //get logging state for gui
    sw_state = sw.read();
}

//This function displays the data on the home screen
void homeDisplay(){
    //display temp on home screen
    lcd.clear();
    lcd.normalMode();

    //print temperature (celcius and farenheit)
    lcd.printString(tempBuffer,40,4);
    lcd.printString(tempBuffer2,37,5);

    //display temperature bar
    int i;
    for (i = 47; i > (47-tempo0); i--) {
        lcd.setPixel(12, i); lcd.setPixel(11, i);
    }

    for (i = 47; i > (47-tempo1); i--){
         lcd.setPixel(10, i); lcd.setPixel(9, i);
        }

    for(i = 47; i > (47-tempo2); i--){
        lcd.setPixel(8, i); lcd.setPixel(7, i);
    }

    for(i = 47; i > (47-tempo3); i--){
            lcd.setPixel(6, i); lcd.setPixel(5, i);
    }
```

```
    for(i = 47; i > (47-tempo4); i--){
        lcd.setPixel(4, i); lcd.setPixel(3, i);
    }

    for (i = 47; i > (47-tempo5); i--){
        lcd.setPixel(2, i); lcd.setPixel(1, i);
    }

    //display time
    lcd.printString(dateBuffer,36,0);
    lcd.printString(timeBuffer,36,1);

    //display logging
    if(sw_state){
        lcd.drawRect(20,5,10,10,FILL_BLACK);
    }else{lcd.drawRect(20,5,10,10,FILL_TRANSPARENT);}
    lcd.refresh();
}

//this function gets the plot data for the display
void plotData(){
    float Temp;
    float realtemp;
    Temp = temperature();
    realtemp = Temp-6;
    tempo6 = realtemp;

    //we offset the data to start plotting  from the
    //20th pixel in the display.
    if(j > 63){
        for(int i = 20; i <= 82; i++){
            temparray[i] = temparray[i+1]; //shift array values
        }
        //append new data value to array
        temparray[83] = realtemp/60;
        j = 64;
    }
    else{
        //initialize and append
        temparray[j+20] =  realtemp/60;
        j++;
    }
}

//this function displays the plot on the plot screen
void plotDisplay(){
       lcd.clear();

       //set environment
        lcd.printString("log",2,0);
        int i;
        for(i = 0; i <= 83; i++){
            lcd.setPixel(i,0);
            lcd.setPixel(i,47);
        }
        for(i = 0; i <= 47; i++){
            lcd.setPixel(20,i);
            lcd.setPixel(0,i);
            lcd.setPixel(83,i);
        }

        //plot data (84 elements)
        lcd.plotArray(temparray);

        //draw temperature level identifier
        if(tempo6>26){
```

```
        lcd.drawCircle(74,7,5,FILL_BLACK);
        }
        else if (tempo6<=26 && tempo6>=13){
            lcd.drawCircle(74,7,5,FILL_TRANSPARENT);
        }
        else if(tempo6<13){
            lcd.drawCircle(77,7,1,FILL_TRANSPARENT);
            lcd.drawCircle(71,7,1,FILL_TRANSPARENT);
            lcd.drawCircle(74,5,1,FILL_TRANSPARENT);
            lcd.drawCircle(74,9,1,FILL_TRANSPARENT);
        }

        lcd.refresh();
}

//this function displays the "About" screen
void aboutDisplay(){
    lcd.clear();
    lcd.printString("CODE BY",20,0);
    lcd.printString("CHIMA NNADIKA",3,1);
    lcd.printString("201077064",15,3);
    lcd.printString("UNI OF LEEDS",8,5);
    lcd.refresh();
}

//this function displays the home detail screen
void homeDetail(){
    lcd.clear();

    //draws and explains the logging identifier
    lcd.drawRect(2,1,10,10,FILL_BLACK);
    lcd.drawRect(2,13,10,10,FILL_TRANSPARENT);
    lcd.printString("LOGGING ON",16,0);
    lcd.printString("LOGGING OFF",16,2);
    lcd.printString("TOGGLE S/W",16,4);
    lcd.printString("1 LOG /MIN",16,5);
    lcd.refresh();
}

//this function displays the plot details page
void plotDetail(){
    lcd.clear();

    //draws the temperature level identifiers
    lcd.drawCircle(5,4,4,FILL_BLACK);
    lcd.drawCircle(5,12,4,FILL_TRANSPARENT);
    lcd.drawCircle(8,19,1,FILL_TRANSPARENT);
    lcd.drawCircle(2,19,1,FILL_TRANSPARENT);
    lcd.drawCircle(5,21,1,FILL_TRANSPARENT);
    lcd.drawCircle(5,17,1,FILL_TRANSPARENT);
    // explains the level identfier meanings
    lcd.printString("HOT",16,0);
    lcd.printString("WARM",16,1);
    lcd.printString("COLD",16,2);
    lcd.printString("MAXPLOT:60 DEG",0,5);

    //points to current temperature level
    if(tempo6>26){lcd.printString("<<",44,0);}
    else if(tempo6<=26 && tempo6>=13){lcd.printString("<<",44,1);}
    else if(tempo6<13){lcd.printString("<<",44,2);}
    //prints temperature in current screen
    lcd.printString(tempBuffer,40,4);
    lcd.refresh();
}
int main(){
    //Aquire data via ticker interrupts
```

```
getdat.attach(&timeTemp, 60.0);
homedat.attach(&homeData, 0.2);
plotdat.attach(&plotData, 0.71);

//initialise screen and set home
lcd.init();
lcd.setContrast(0.5);
lcd.inverseMode();
wait(0.5);
lcd.normalMode();
wait(0.5);
lcd.inverseMode();
wait(0.5);
lcd.normalMode();
lcd.printString("TEMPERATURE",0,0);
lcd.printString("SENSOR/LOGGER",0,1);
lcd.printString("BUTTON A",0,2);
lcd.printString(">>NEXT PAGE",0,3);
lcd.printString("BUTTON B",0,4);
lcd.printString(">>PAGE DETAILS",0,5);
lcd.refresh();
wait(5);


while(1) {
    lcd.setBrightness(0.5);
    lcd.clear();

    //detect button state
    //and switch between modes depending on button push
    if(buttonA == 0 && (mode == 0 | mode == 4)){
        while(!buttonA){} //wait for button release
        mode = 1;
    }
    else if(buttonA == 0 && (mode == 1 | mode == 3)){
        while(!buttonA){}
        mode = 2;
    }
    else if(buttonA == 0 && mode == 2){
        while(!buttonA){}
        mode = 0;
    }else if(buttonB == 0 && mode == 1){
        while(!buttonB){}
        mode = 3;
    }else if(buttonB == 0 && mode == 3){
        while(!buttonB){}
        mode = 1;
    }else if(buttonB == 0 && mode == 0){
        while(!buttonB){}
        mode = 4;
    }else if(buttonB == 0 && mode == 4){
        while(!buttonB){}
        mode = 0;
    }


    //display states
    if(mode == 1){
        plotDisplay();
    }
    else if(mode == 2){
        aboutDisplay();
    }
    else if(mode == 3){ //activated on Button B push while on plot display
        plotDetail();   //plot details page expands on plot display
    }else if(mode == 4){ //activated on Button B push while on home display
```

```
            homeDetail();   //home detail page expands on home display
        }else{
            homeDisplay();
        }
    }
}
```