Appendix D

The PIC18LF45K20 Code

D.1 lcd.h

```
//LCD Functions Developed by electroSome

void Lcd_Port(char a)
{
if(a & 1)
D4 = 1;
else
D4 = 0;

if(a & 2)
D5 = 1;
else
D5 = 0;

if(a & 4)
D6 = 1;
else
D6 = 0;

if(a & 8)
D7 = 1;
else
D7 = 0;
}
void Lcd_Cmd(char a)
{
RS = 0;             // => RS = 0
Lcd_Port(a);
EN  = 1;             // => E = 1
        __delay_ms(4);
        EN  = 0;             // => E = 0
}

Lcd_Clear()
{
Lcd_Cmd(0);
Lcd_Cmd(1);
}

void Lcd_Set_Cursor(char a, char b)
{
char temp,z,y;
if(a == 1)
{
```

```
  temp = 0x80 + b - 1;
z = temp>>4;
y = temp & 0x0F;
Lcd_Cmd(z);
Lcd_Cmd(y);
}
else if(a == 2)
{
temp = 0xC0 + b - 1;
z = temp>>4;
y = temp & 0x0F;
Lcd_Cmd(z);
Lcd_Cmd(y);
}
}

void Lcd_Init()
{
  Lcd_Port(0x00);
   __delay_ms(20);
  Lcd_Cmd(0x03);
__delay_ms(5);
  Lcd_Cmd(0x03);
__delay_ms(11);
  Lcd_Cmd(0x03);
  ////////////////////////////////////////////////////
  Lcd_Cmd(0x02);
  Lcd_Cmd(0x02);
  Lcd_Cmd(0x08);
  Lcd_Cmd(0x00);
  Lcd_Cmd(0x0C);
  Lcd_Cmd(0x00);
  Lcd_Cmd(0x06);
}

void Lcd_Write_Char(char a)
{
   char temp,y;
   temp = a&0x0F;
   y = a&0xF0;
   RS = 1;             // => RS = 1
   Lcd_Port(y>>4);             //Data transfer
   EN = 1;
   __delay_us(40);
   EN = 0;
   Lcd_Port(temp);
```

```
   EN = 1;
   __delay_us(40);
   EN = 0;
}

void Lcd_Write_String(char *a)
{
int i;
for(i=0;a[i]!='\0';i++)
   Lcd_Write_Char(a[i]);
}

void Lcd_Shift_Right()
{
Lcd_Cmd(0x01);
Lcd_Cmd(0x0C);
}

void Lcd_Shift_Left()
{
Lcd_Cmd(0x01);
Lcd_Cmd(0x08);
}
```

D.2 UART.h

```
/*
    Purpose:  To provide UART functionality for PIC uC
*/
char UART_Init(const long int baudrate)
{
unsigned int x;
x = (_XTAL_FREQ - baudrate*64)/(baudrate*64);
if(x>255)
{
x = (_XTAL_FREQ - baudrate*16)/(baudrate*16);
BRGH = 1;
}
if(x<256)
{
  SPBRG = x;
  SYNC = 0;
  SPEN = 1;
```

```
          TRISC7 = 1;
          TRISC6 = 1;
          CREN = 1;
          TXEN = 1;
  return 1;
}
return 0;
}

char UART_TX_Empty()
{
  return TRMT;
}

char UART_Data_Ready()
{
   return RCIF;
}
char UART_Read()
{

  while(!RCIF);
  return RCREG;
}

void UART_Write(char data)
{
  while(!TRMT);
  TXREG = data;
}

void UART_Write_Text(char *text)
{
  int i;
  for(i=0;text[i]!='\0';i++)
  UART_Write(text[i]);
}
```

D.3 main.c

```
#define _XTAL_FREQ 20000000

#define RS LATCbits.LATC5
#define EN LATCbits.LATC4
#define D4 LATBbits.LATB7
#define D5 LATBbits.LATB6
#define D6 LATBbits.LATB5
#define D7 LATBbits.LATB4


#include <xc.h>
#include<string.h>
#include<p18f25k20.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "lcd.h"
#include "UART.h"


// BEGIN CONFIG
// CONFIG1H
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)

// CONFIG2L
#pragma config PWRT = OFF       // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON       // Brown-out Reset Enable bits (Brown-out Reset enabled and controlled by software (SBOREN is enabled))
#pragma config BORV = 18        // Brown Out Reset Voltage bits (VBOR set to 1.8 V nominal)

// CONFIG2H
#pragma config WDTEN = OFF      // Watchdog Timer Enable bit (WDT is controlled by SWDTEN bit of the WDTCON register)
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC   // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = OFF      // PORTB A/D Enable bit (PORTB<4:0> pins are configured as digital input channels on Reset)
#pragma config LPT1OSC = OFF    // Low-Power Timer1 Oscillator Enable bit (Timer1 configured for higher power operation)
#pragma config HFOFST = ON      // HFINTOSC Fast Start-up (HFINTOSC starts clocking the CPU without waiting for the oscillator to stablize.)
#pragma config MCLRE = ON       // MCLR Pin Enable bit (MCLR pin enabled; RE3 input pin disabled)

// CONFIG4L
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = OFF        // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L
#pragma config CP0 = OFF        // Code Protection Block 0 (Block 0 (000800-001FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection Block 1 (Block 1 (002000-003FFFh) not code-protected)
#pragma config CP2 = OFF        // Code Protection Block 2 (Block 2 (004000-005FFFh) not code-protected)
#pragma config CP3 = OFF        // Code Protection Block 3 (Block 3 (006000-007FFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF       // Write Protection Block 0 (Block 0 (000800-001FFFh) not write-protected)
#pragma config WRT1 = OFF       // Write Protection Block 1 (Block 1 (002000-003FFFh) not write-protected)
```

```
#pragma config WRT2 = OFF       // Write Protection Block 2 (Block 2 (004000-005FFFh) not write-protected)
#pragma config WRT3 = OFF       // Write Protection Block 3 (Block 3 (006000-007FFFh) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot Block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not write-protected)

// CONFIG7L
 // Table Read Protection(Block 0, 1, 2 and 3 not protected from table reads executed in other blocks)
#pragma config EBTR0 = OFF
#pragma config EBTR1 = OFF
#pragma config EBTR2 = OFF
#pragma config EBTR3 = OFF

// CONFIG7H
#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot Block (000000-0007FFh) not protected from table reads executed in other blocks)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

//END CONFIG
```

```
#define forward_dir 0
#define reverse_dir 1

bit ecoMode = 0; // To enable ECO MODE

unsigned int tempToSet = 20; //initial value of the temperature to set

int currentTemp; //the temperature measured

char futureTemp[4], lastTemp[4];


char addressToConnect[20];

//int shaft_steps = 5;
int shaft_steps = 32;
unsigned char TempData;

char string[20];

char buffer[20];
int i = 0;



void myDelay_n10ms(int n){

    while(n > 0){
        __delay_ms(10);
        --n;
    }
}
```

```
void ADCinit(){
 ADCON1 =0b00000000;
 ADCON2 = 0b10001010;//ADCON2 setup: rigth justified, Tacq =2Tad, Tad=32*Tosc (or Fosc/32)
}

// reverses a string 'str' of length 'len'
void reverse(char *str, int len){
    int i=0, j=len-1, temp;
    while (i<j)
    {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++; j--;
    }
}

 // Converts a given integer x to string str[].  d is the number
 // of digits required in output. If d is more than the number
 // of digits in x, then 0s are added at the beginning.
int intToStr(int x, char str[], int d){
    int i = 0;
    while (x){
        str[i++] = (x%10) + '0';
        x = x/10;
    }
    // If number of digits required is more, then
    // add 0s at the beginning
    while (i < d)
        str[i++] = '0';

    reverse(str, i);
    str[i] = '\0';
    return i;
}

// Converts a floating point number to string.
void myftoa(float n, char *res, int afterpoint){
int temp_int[2] = {0};
    // Extract integer part
    int ipart = (int)n;
    // Extract floating part
    float fpart = n - (float)ipart;
    // convert integer part to string
    int i = intToStr(ipart, res, 0);
```

```
    // check for display option after point
    if (afterpoint != 0)    {
        res[i] = '.';  // add dot
        // Get the value of fraction part upto given no.
        // of points after dot. The third parameter is needed
        // to handle cases like 233.007
        fpart = fpart * pow(10, afterpoint);
        intToStr((int)fpart, res + i + 1, afterpoint);
    }
}

unsigned int ADCRead(unsigned char ch){
   if (ch>13) return 0;  //Invalid Channel
   ADCON0=0x00;
   ADCON0=(ch<<2);   //Select ADC Channel
   ADCON0bits.ADON=1;  //switch on the adc module
   ADCON0bits.GO_DONE=1;//Start conversion
   while(ADCON0bits.GO_DONE); //wait for the conversion to finish
   ADCON0bits.ADON=0;  //switch off adc
   return ADRES;
}

float temp(){

float T;
int result;
float conv;
char res[4];
char temp[3];
   ADCinit();
    result = ADCRead(0);//Channel 0 for the temperature
    conv = (result*3.3)/1024.0;
    T = 25.0 +(conv -2.98)*100; //float
    currentTemp = T;
    tempToSet = T; //to be abble to increase (decrease) the temperature from currentTemp;
if (T < 0.0) {
myftoa(-T, temp, 0);
Lcd_Set_Cursor(2, 6);
     Lcd_Write_Char('-');
Lcd_Set_Cursor(2, 1);
    Lcd_Write_String("Temp:");
        Lcd_Set_Cursor(2, 7);
    Lcd_Write_String(temp);
        UART_Write_Text(temp); // Send string on UART
        UART_Write_Text("\r\n");
}
```

```
else if(T < 100.0 && T >= 0.0){
     myftoa(T, res, 0);
Lcd_Set_Cursor(2, 1);
    Lcd_Write_String("Temp:");
        Lcd_Set_Cursor(2, 6);
   Lcd_Write_String(res);
        UART_Write_Text(res); // Send string on UART
        UART_Write_Text("\r\n");
  }
else if(T >= 100.0){
    myftoa(T, res, 0);
Lcd_Set_Cursor(2, 1);
    Lcd_Write_String("Temp:");
        Lcd_Set_Cursor(2, 6);
    Lcd_Write_String(res);
        UART_Write_Text(res); // Send string on UART
        UART_Write_Text("\r\n");
}
    return T;
}


void battery(){

int result;
float conv, limit;
     ADCinit();
    result = ADCRead(1);// channel 1 for the battery measurements
    conv = (result*3.3)/1024.0;
    limit = 1.88; // (3.3+045)/2 where 0.75 is the droptout of the regulator

if (conv > limit){
Lcd_Set_Cursor(2, 10);
    Lcd_Write_String("BAT:");
        Lcd_Set_Cursor(2, 14);
    Lcd_Write_String("OK");
}
else if(conv < limit){
Lcd_Set_Cursor(2, 10);
    Lcd_Write_String("BAT:");
        Lcd_Set_Cursor(2, 14);
    Lcd_Write_String("!!");
  }
}
```

```
void motor(int direction, int steps){
    if (direction==0){
        LATCbits.LATC1 = 0;
        LATCbits.LATC2 = 0;
        myDelay_n10ms(200);
        while(steps > 0){
            LATCbits.LATC1 = 1;
            myDelay_n10ms(100);
            --steps;
        }
        LATCbits.LATC1 = 0;
    }
    else if (direction==1){
        LATCbits.LATC1 = 0;
        LATCbits.LATC2 = 0;
        myDelay_n10ms(200);
        while(steps > 0){
            LATCbits.LATC2 = 1;
            myDelay_n10ms(100);
            --steps;
        }
        LATCbits.LATC2 = 0;
    }
}


void temp_learning(){

    int n = shaft_steps, address = 0x01;
    int tempValue;

    motor(reverse_dir, shaft_steps); //opens the valve completely
    motor(forward_dir, shaft_steps); //closes the valve completely

    while(n > 0){
        motor(reverse_dir, 1); //opens (1/shft_steps) of the total length of the shaft
        //myDelay_n10ms(90000); //Delay of 15minutes for the equilibration of the temperature
        myDelay_n10ms(200); //Delay of 15minutes for the equilibration of the temperature
        tempValue = temp();
        eeprom_write(address, tempValue);
        ++address;
        --n;
    }
    //value = eeprom_read (address);
}
```

```
/*
 *This function reads the current shaft position and the
 *position to get the temperature set by the user. It computes
 * the number of steps needed and move the shaft to the rigth position
 */
void setTemp(){

    int currentStep, toSetStep, num_steps;
    currentStep = eeprom_read (currentTemp); //reads the position of the shaft
    toSetStep = eeprom_read (tempToSet); //reads the next position of the shaft
    num_steps = currentStep - toSetStep;

    if(num_steps > 0){
        motor(forward_dir, num_steps);
    }
    else if(num_steps < 0){
        num_steps = 0 - num_steps;
        motor(reverse_dir, num_steps);
    }
    if(num_steps == 0){
        ;
    }
}

void display(){
    Lcd_Set_Cursor(1,1);
    Lcd_Write_String("set To: ");
    Lcd_Write_String(futureTemp);
    myDelay_n10ms(1000);
}

void ecoMod(){

    do{
        // disable all the interrupts except INT0 which manage EcoMode
        RCIE = 0;
        INT1IE = 0;
        INT2IE = 0;
        motor(forward_dir, shaft_steps); //closes the valve completely
    }while(ecoMode == 1);
    if(ecoMode == 0){
        RCIE = 0;
        INT1IE = 0;
        INT2IE = 0;
```

```
        setTemp();
    }
}

int main()
{
    TRISA = 0xFF; //sets all the portA as inputs
    TRISB = 0x0F; //RB0-3 are inputs and the others are outputs


    TRISCbits.TRISC1 = 0; //
    TRISCbits.TRISC2 = 0;
    TRISCbits.TRISC4 = 0;
    TRISCbits.TRISC5 = 0;


    LATCbits.LATC1 = 0; //to be sure that the motor is disabled
    LATCbits.LATC2 = 0; //to be sure that the motor is disabled

    RCONbits.IPEN = 1; //enable interruption priority

    INTCONbits.INT0IE = 1; //enable Interrupt 0 (RB0 as interrupt)
    INTCON2bits.INTEDG0 = 0; //cause interrupt at falling edge
    INTCONbits.INT0IF = 0; //reset interrupt flag

    INTCON3bits.INT1IE = 1; //enable Interrupt 1 (RB1 as interrupt)
    INTCON2bits.INTEDG1 = 0; //cause interrupt at falling edge
    INTCON3bits.INT1IF = 0; //reset interrupt flag
    INTCON3bits.INT1IP = 0; //low priority

    INTCON3bits.INT2IE = 1; //enable Interrupt 2 (RB2 as interrupt)
    INTCON2bits.INTEDG2 = 0; //cause interrupt at falling edge
    INTCON3bits.INT2IF = 0; //reset interrupt flag
    INTCON3bits.INT2IP = 0; //low priority

    RCIE = 1; //Enable USART Receiver Interrupt
    RCIF = 0; // Receiver Flag

    RCIP = 1; // High priority for Receiver Interrupt

    Lcd_Init();
    ADCinit();
    UART_Init(9600);

    PEIE = 1;   // Enable Peripheral Interrupts
    GIE  = 1;   // Enable global interrupts
```

```
 int i = 0;


 UART_Write_Text("abort = true"); // Reset the ESP to prevent troubleshooting
 UART_Write_Text("\r\n");
 myDelay_n10ms(1000);

UART_Write_Text("abort = true"); // Reset the ESP to prevent troubleshooting
UART_Write_Text("\r\n");
myDelay_n10ms(1000);

UART_Write_Text("abort = true"); // Reset the ESP to prevent troubleshooting
UART_Write_Text("\r\n");
myDelay_n10ms(1000);

 UART_Write_Text("dofile(\"init.lua\")\r\n");// Asks ESP8266 to start running its code
 Lcd_Set_Cursor(2,1);
 Lcd_Write_String("dofile(\"init.lua\")");
 myDelay_n10ms(2000);

 Lcd_Clear();
 Lcd_Set_Cursor(1,1);
 Lcd_Write_String("WLAN Thermostat");
 myDelay_n10ms(100);
 Lcd_Set_Cursor(2,1);
 Lcd_Write_String("ULg 2016");
 myDelay_n10ms(500);

 motor(reverse_dir, shaft_steps); //opens the valve completely
 motor(forward_dir, shaft_steps); //closes the valve completely

 Lcd_Clear();
 Lcd_Set_Cursor(1,1);
 Lcd_Write_String("Learn Temp?");
 while(i < 20){      // gives 5seconds to switch to the learning algorithm
     if(PORTBbits.RB3 == 0){
         Lcd_Clear();
         Lcd_Set_Cursor(1,1);
         Lcd_Write_String("Learning Temp");
         temp_learning();
         i = 20;
     }
     myDelay_n10ms(25);
     ++i;
 }
```

```
    Lcd_Clear();

  while(1){
    temp();
    battery();
    display();
    myDelay_n10ms(90000); //15 minutes

  }
  return 0;
}



void interrupt high_priority HighISR(){

    if(RCIF){  // If UART Rx Interrupt
if(OERR){ // If over run error, then reset the receiver
CREN = 0;
CREN = 1;
}
        int count;
        //Read USART data
        //PIR1bits.RCIF;//Data has been passed to RCREG
        buffer[count] = RCREG; //Read RX register
        count++;
         if(RCREG=='\n'){
            if(buffer[0]=='N'){
                futureTemp[0] = buffer[2];
                futureTemp[1] = buffer[3];
                count = 0;
            }
            if((buffer[0]=='1')&&(buffer[1]=='9')&&(buffer[2]=='2')){
                int j = 0;
                while(buffer[j]=='\r'){
                    addressToConnect[j]= buffer[j];
                    ++j;
                }
                Lcd_Clear();
                Lcd_Set_Cursor(1,1);
                Lcd_Write_String(addressToConnect); // Displays the IP address to connect the ESP
            }
            count = 0;
        }
        RCSTAbits.CREN = 0; //clear error (if any)
        RCSTAbits.CREN = 1; //Enables Receiver
```

```
        RCIE = 0;
        RCIE = 1;
}

    if(INT0IF)  // If INT0 Interrupt
{
        INTCONbits.INT0IF = 0;
        myDelay_n10ms(50); //delay of 500ms to debounce
        if(ecoMode == 0){
            Lcd_Clear();
            Lcd_Set_Cursor(1,1);
            Lcd_Write_String("ENTER ECOMODE");
            myDelay_n10ms(100);
            ecoMode = 1;
        }
        else if(ecoMode == 1){
            Lcd_Clear();
            Lcd_Set_Cursor(1,1);
            Lcd_Write_String("END ECOMODE");
            myDelay_n10ms(100);
            ecoMode = 0;
        }
        ecoMod();
}
}

void interrupt low_priority LowISR(void){
    float a = tempToSet;
    char tmp[3];

    if(INT1IF){ // If INT1 Interrupt
        INTCON3bits.INT1IF = 0;
        --tempToSet;
        myftoa(a, tmp, 0); //converts a float to an array of char
        myDelay_n10ms(50); //delay of 500ms to debounce
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("set to:");
        Lcd_Set_Cursor(1,9);
        Lcd_Write_String(tmp);
        myDelay_n10ms(100);
}
    if(INT2IF){  // If INT2 Interrupt
        INTCON3bits.INT2IF = 0;
        ++tempToSet;
        myftoa(a, tmp, 0); //converts a float to an array of char
```

```
        myDelay_n10ms(50); //delay of 500ms to debounce
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("set to:");
        Lcd_Set_Cursor(1,9);
        Lcd_Write_String(tmp);
        myDelay_n10ms(100);
}
}
```