[Group 1]

ECE-412-01

Final Project Report

Group 1
David Antosh, Calvin Lam, Andrew Branum, Ankit Kanotra

[Group 1]

ECE-412-01

Thermostat Project

[Group 1]

ECE-412-01

Abstract

The goal of the project is to gain a deeper understanding of the relationship between C and assembly code, command line software (Tera Term), circuitry, and to gain groupwork experience by creating something from scratch. Ideas are brainstormed and research is done to determine if said ideas are plausible. A is the final decision. The circuit is constructed using the given schematic and is connected to a Windows operating system using Atmel Studio 7. Research is done to figure out how to implement the various functions within C code. The code is tested within Tera Term using the LCD connected to the circuit. The creation of the project started with the hardware schematic and everything was tried and tested using a Mini Xplained 328pb microprocessor. The microprocessor's software is written via connection of a USB to a laptop with Atmel Studios. The results of the project were a success, as the returned temperatures matched the expected room temperature and extremities.

[Group 1]

ECE-412-01

Body

The hardware for this project was configured identically to that of lab 3. 11 pins (pin 4-14) of the LCD are connected to the Xplained Mini board. PBn and PDn pins are on the Xplained Mini Board. PB1 connects to RS, PB0 connects to RW, PB2 connects to E, PD0 connects to D0, PD1 connects to D1, PD2 connects to D2, PD3 connects to D3, PD4 connects to D4, PD5 connects to D5, PD6 connects to D6, and PD7 connects to D7. The backlight functions properly by connecting the cathode of the LCD to ground and the anode to a power source. The cathode and anode are the last two pins of the LCD. The brightness of the LCD screen is able to be changed by using the first three pins of the LCD and the use of a Single Turn 10K potentiometer.

The program gets the current signal from the analog to digital converter(ADC) and dynamically calculates the corresponding voltage and with the voltage, temperature can be calculated. Firstly, the calculation of resistance 2 (resist2), needs to be calculated in order to find the temperature with the formula : (B * 298.15) / (298.15 * log(resist2 / resist1) + B), B is derived from the rating of the 10k thermistor, B = 3950. But because this formula returns temperature in Kelvins, it is necessary to subtract 273.15 to get the value in celsius. From there the temperature needs to be dynamically built into a character array that is readable by the UART_PUT() and LCD_PUT() functions. This is done by taking the modulus 10 of the temperature and the remainder plus 48 would give us the corresponding ASCII value of the 1's place for temperature. Then by subtracting that value from the temperature and dividing by 10 and adding 48 will return what the 10's place will be in ASCII. Lastly, by feeding the dynamically created character array to the functions, the LCD will display temperature in celsius. The same steps could be replicated for Fahrenheit by multiplying temperature by 1.8 and adding 32 before all the ASCII conversions.

Once the program is initialized, a menu is displayed onto the terminal window. Two options can be selected using the keys 'F' and 'C'. Depending on which key is selected, a switch case will call the corresponding function to perform temperature calculation (via the thermistor). After the calculation is performed, the function will display the results on the terminal window and call another function to display those results on the LCD, *see figure 2 and 3*. After all calculations and visuals are completed, the program will return to the menu screen for another input to be received. This program flow can be visualized in *figure 1*.

[Group 1]

ECE-412-01

There were a few challenges when building this project. Firstly, when initially building the project, both Celsius and Farenheit were calculated using the same function. This was most likely due to shared variables between the two calculations, resulting in undesirable output in the LCD and terminal. To resolve the issue, separate functions were created and the menu was updated to ensure that only 1 temperature type was being calculated and displayed at one time. Another problem occurring was incorrect values being displayed on the LCD. As an example, if the celsius value on the terminal was 22, the LCD would display the value as 2<. This was being caused by parameter passing through functions inside the program. As a solution, values calculated from the Celsius and Fahrenheit were set to global variables and the LCD prep functions would pass in those globals to the final LCD print function.

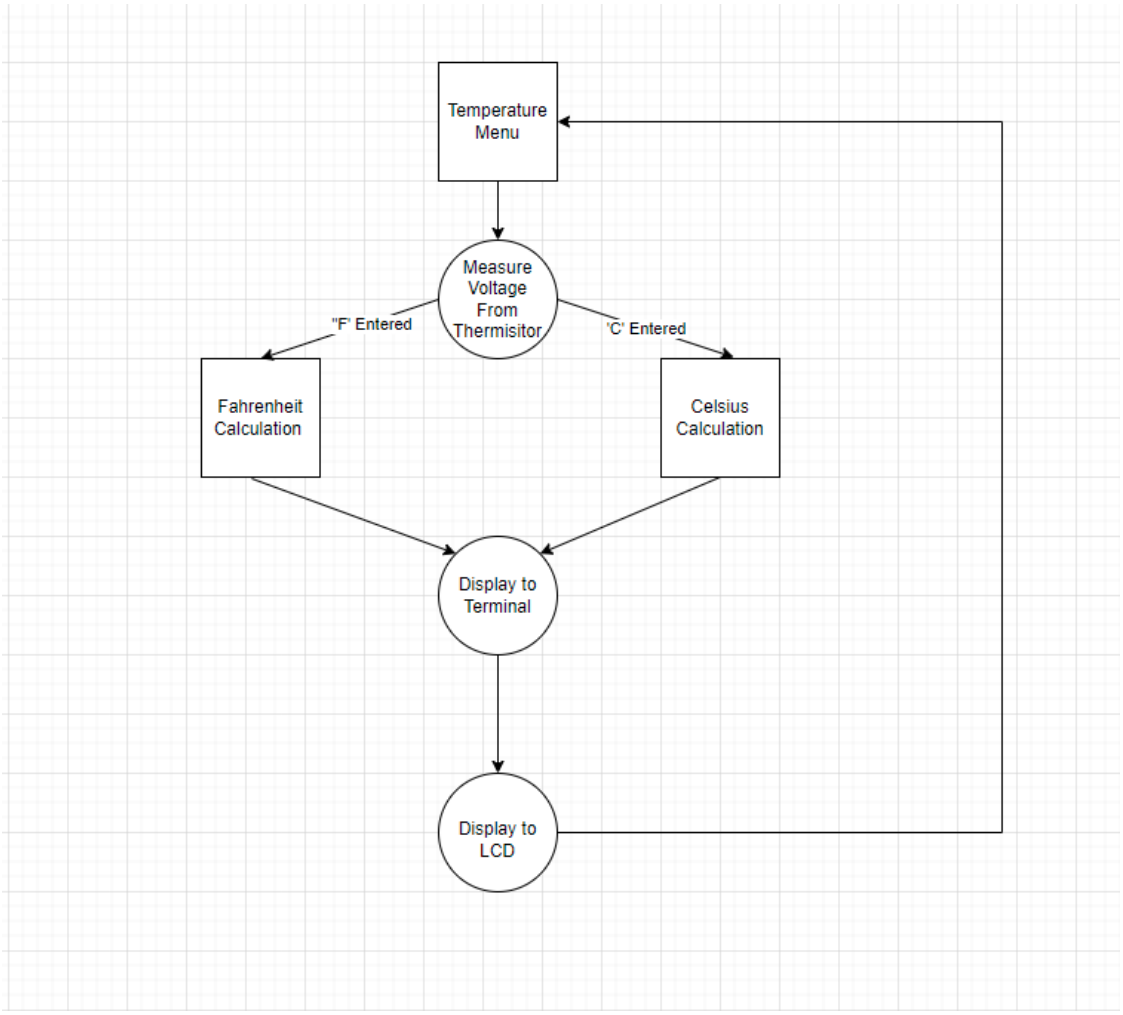


Figure 1: Flowchart of Program Logic

[Group 1]

ECE-412-01

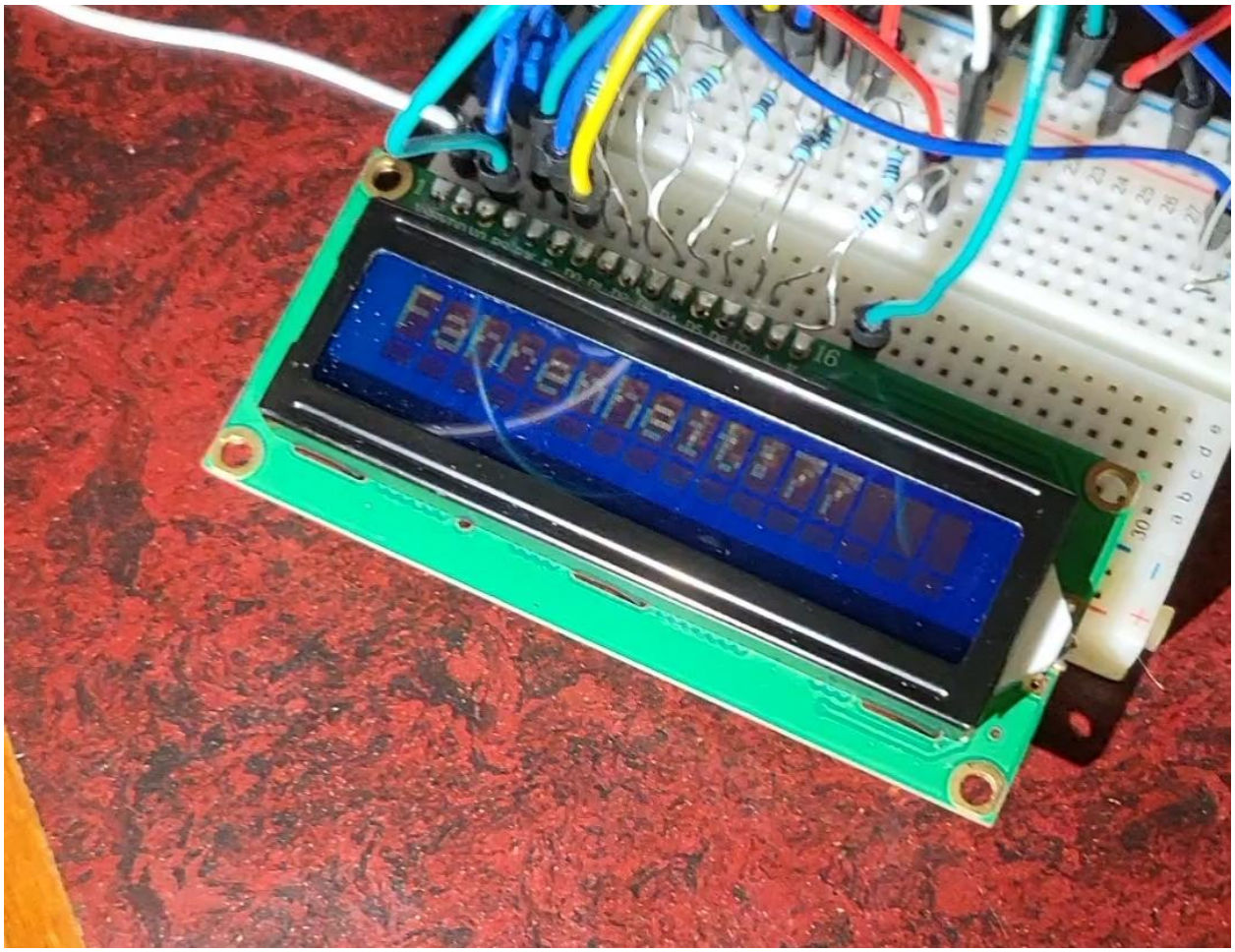


Figure 2: Fahrenheit Output Displayed on LCD Display

[Group 1]

ECE-412-01

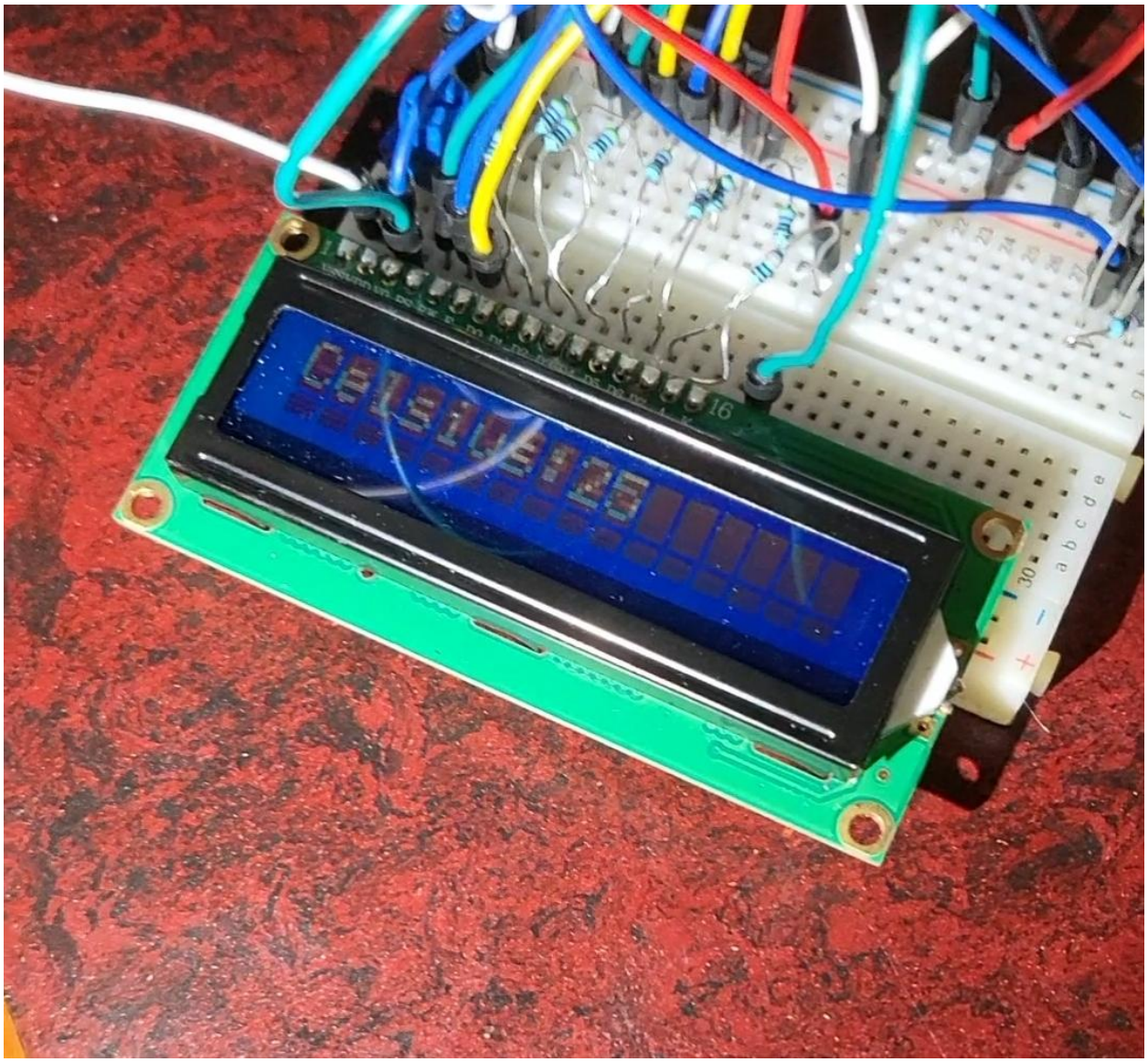


Figure 3: Celsius Output Displayed on LCD Display

[Group 1]

ECE-412-01

Source Code (Software)

```
#include <math.h>                    //Math Library

const char MS3[] = "\r\nPress (C)elsius (F)ahrenheit\r\n";          //Menu Screen
const char MS4[] = "\r\nThermostat: "; //Launch Message
const char MS5[] = "\r\nInvalid Command Try Again...";          //Message for incorrect input

char volts[5];                       //string buffer for ADC output
int Acc;                             //Accumulator for ADC use
double temp1;                        //Temperature conversion variables
double temp2;
double temp3;
double resist1 = 10000.0;            //The resistance of the resistor
double resist2;                      //Resistance of the thermistor
double B = 3950.0;                   //B value of the thermistor
char temp[3];
double v0 = 0.0;                     //Volt double placeholder
int a;
int b;
int c;                               //end of temperature variables

void LCD_Init(void);          //external Assembly functions
void UART_Init(void);
void UART_Clear(void);
void UART_Get(void);
void UART_Put(void);
void LCD_Write_Data(void);
void LCD_Write_Command(void);
void LCD_Read_Data(void);
void Mega328P_Init(void);
void ADC_Get(void);
void EEPROM_Read(void);
void EEPROM_Write(void);

unsigned char ASCII;          //shared I/O variable with Assembly
unsigned char DATA;           //shared internal variable with Assembly
char HADC;                    //shared ADC variable with Assembly
char LADC;                    //shared ADC variable with Assembly

char volts[5];                //string buffer for ADC output
```

[Group 1]

ECE-412-01

```
int Acc;                        //Accumulator for ADC use


void UART_Puts(const char *str)    //Display a string in the PC Terminal Program
{
   while (*str)
   {
      ASCII = *str++;
      UART_Put();
   }
}

void LCD_Puts(const char *str)    //Display a string on the LCD Module
{
   while (*str)
   {
      DATA = *str++;
      LCD_Write_Data();
   }
}

void Banner(void)                  //Display Tiny OS Banner on Terminal
{
   UART_Puts(MS4);
}

void HELP(void)                   //Display available Tiny OS Commands on Terminal
{
   UART_Puts(MS3);
}

//Celsius Output Function
void LCD(void)
{
   DATA = 0x34;
   LCD_Write_Command();
   DATA = 0x08;
   LCD_Write_Command();
   DATA = 0x02;
   LCD_Write_Command();
   DATA = 0x06;
```

```
   LCD_Write_Command();
   DATA = 0x0f;
   LCD_Write_Command();
        //Writes Celsius and temp variable onto screen
        LCD_Puts("Celsius:");
        LCD_Puts(temp);
        //Writes just enough spaces to reach end of LCD screen so continual LCD writes will
clear
        LCD_Puts("      ");
}

//Fahrenheit Output Function
void LCD_f(void)
{
   DATA = 0x34;
   LCD_Write_Command();
   DATA = 0x08;
   LCD_Write_Command();
   DATA = 0x02;
   LCD_Write_Command();
   DATA = 0x06;
   LCD_Write_Command();
   DATA = 0x0f;
   LCD_Write_Command();
        //Writes Fahrenheit and temp variable onto screen
        LCD_Puts("Fahrenheit:");
        LCD_Puts(temp);
        //Writes just enough spaces to reach end of LCD screen so continual LCD writes will
clear
        LCD_Puts("      ");
}

//Celsius Recorder/Calculator Function
void ADC(void)   //Lite Demo of the Analog to Digital Converter
{
        volts[0x1]='.';
        volts[0x3]=' ';
        volts[0x4]= 0;
        ADC_Get();
        Acc = (((int)HADC) * 0x100 + (int)(LADC))*0xA;
        volts[0x0] = 48 + (Acc / 0x7FE);
        Acc = Acc % 0x7FE;
```

```
        volts[0x2] = ((Acc *0xA) / 0x7FE) + 48;
        Acc = (Acc * 0xA) % 0x7FE;

        if (Acc >= 0x3FF) volts[0x2]++;

        if (volts[0x2] == 58)
        {
                volts[0x2] = 48;
                volts[0x0]++;

        }

        v0 = volts[0x0] - 48 + (volts[0x2] - 48) / 10.0;

        //into a double

        resist2 = (v0 * resist1) / (5.0 - v0);   //calculate the resistance of

        //the thermistor
        //temperature calculation and conversion into Celsius

        temp1 = (B * 298.15) / (298.15 * log(resist2 / resist1) + B);
        temp2 = temp1 - 273.15;

        //ASCII conversion of temperature(Celsius) double
        b = floor(fmod(temp2, 10.0));
        c = b + 48;
        a = floor(((temp2 - b) / 10.0)) + 48;

        temp[0x0] = a;
        temp[0x1] = c;
        temp[0x2] = 0;

        //Outputs to Terminal and Calls LCD Function
        UART_Puts(" Celsius: ");
        UART_Puts(temp);
        LCD();
}

void ADC_f(void)   //Lite Demo of the Analog to Digital Converter
{
        volts[0x1]='.';
```

```
volts[0x3]=' ';
volts[0x4]= 0;
ADC_Get();
Acc = (((int)HADC) * 0x100 + (int)(LADC))*0xA;
volts[0x0] = 48 + (Acc / 0x7FE);
Acc = Acc % 0x7FE;
volts[0x2] = ((Acc *0xA) / 0x7FE) + 48;
Acc = (Acc * 0xA) % 0x7FE;

if (Acc >= 0x3FF) volts[0x2]++;

if (volts[0x2] == 58)
{
        volts[0x2] = 48;
        volts[0x0]++;
}

v0 = volts[0x0] - 48 + (volts[0x2] - 48) / 10.0;

//into a double

resist2 = (v0 * resist1) / (5.0 - v0);   //calculate the resistance of

//the thermistor
//temperature calculation and conversion into Celsius

temp1 = (B * 298.15) / (298.15 * log(resist2 / resist1) + B);
temp2 = temp1 - 273.15;

temp3 = ((temp2 * 9.0) / 5.0) + 32.0; //Celsius to Fahrenheit conversion

//ASCII conversion of temperature(Celsius) double
b = floor(fmod(temp3, 10.0));
c = b + 48;
a = floor(((temp3 - b) / 10.0)) + 48;

temp[0x0] = a;
temp[0x1] = c;
temp[0x2] = 0;

//Outputs to Terminal and Calls LCD Function
UART_Puts(" Fahrenheit: ");
```

```
        UART_Puts(temp);
        LCD_f();
}

void Command(void)              //command interpreter
{
   UART_Puts(MS3);
   ASCII = '\0';
   while (ASCII == '\0')
   {
      UART_Get();
   }
   switch (ASCII)
   {
      case 'C' | 'c': ADC();
      break;
                case 'F' | 'f': ADC_f();
      break;
      default:
      UART_Puts(MS5);
      HELP();
      break;
   }
}

int main(void)
{
   Mega328P_Init();
   Banner();
   while (1)
   {
      Command();            //infinite command loop
   }
}
```

```
.section ".data" //sets up constants in the memory
.equ DDRB,0x04 //Port B Data Direction Register as I/O, sets port to
//0x04 location
.equ            DDRD,0x0A               //DDRD = 10
.equ            PORTB,0x05              //PORTB = 5
.equ            PORTD,0x0B              //PORTD = 11
```

[Group 1]

ECE-412-01

```
.equ        U2X0,1              //U2X0 = 1
.equ        UBRR0L,0xC4         //UBRR0L = 196
.equ        UBRR0H,0xC5         //UBRR0H = 197
.equ        UCSR0A,0xC0         //UCSR0A = 192
.equ        UCSR0B,0xC1         //UCSR0B = 193
.equ        UCSR0C,0xC2         //UCSR0C = 194
.equ        UDR0,0xC6           //UDR0 = 198
.equ        RXC0,0x07           //RXC0 = 7
.equ        UDRE0,0x05          //UDRE0 = 5
.equ        ADCSRA,0x7A         //ADCSRA = 122
.equ        ADMUX,0x7C          //ADMUX = 124
.equ        ADCSRB,0x7B         //ADCSRB = 123
.equ        DIDR0,0x7E          //DIDR0 = 126
.equ        DIDR1,0x7F          //DIDR1 = 127
.equ        ADSC,6              //ADSC = 6
.equ        ADIF,4              //ADIF =4
.equ        ADCL,0x78           //ADCL = 120
.equ        ADCH,0x79           //ADCH = 121
.equ        EECR,0x1F           //EECR = 31
.equ        EEDR,0x20           //EEDR = 32
.equ        EEARL,0x21          //EEARL = 33
.equ        EEARH,0x22          //EEARH = 34
.equ        EERE,0              //EERE = 0
.equ        EEPE,1              //EEPE = 1
.equ        EEMPE,2             //EEMPE = 2
.equ        EERIE,3             //EERIE = 3
//variables accessible by the C code
.global     HADC                //high bit of ADC for C code
.global     LADC                //low bit of ADC for C code
.global     ASCII               //numbers will be stored to match the ASCII table
.global     DATA                //Used for communication between C and
Assembly instructions
.set        temp,0              //temp = 0
.section    ".text"             //Section used for program code and functions
.global Mega328P_Init
Mega328P_Init:
ldi         r16,0x07 ;PB0(R*W),PB1(RS),PB2(E) as fixed outputs
out         DDRB,r16            //writes value in r16 to DDRB port at 0x04
ldi         r16,0               //loads r16 with 0
out         PORTB,r16           //writes value (0) in r16 to PORTB
out         U2X0,r16            //initialize UART, 8bits, no parity, 1 stop, 9600
ldi         r17,0x0             //loads r17 with 0x0
```

[Group 1]

ECE-412-01

```
ldi             r16,0x67                //loads r16 with 0x67
sts             UBRR0H,r17              //stores value in r17 to data space at UBRR0H
                                         0xC5
sts             UBRR0L,r16               //stores value in r16 to data space at UBRR0L
                                         0xC4
ldi             r16,24                   //loads r16 with 24
sts             UCSR0B,r16              //stores value in r16 to data space at UCSR0B 0xC1
ldi             r16,6                    //loads r16 with 6
Sts             UCSR0C,r16              //stores value in r16 to data space at UCSR0C 0xC2
ldi             r16,0x87                 //initialize ADC, loads r16 with 0x87
sts             ADCSRA,r16               //stores value in r16 to data space ADCSRA 0x7A
ldi             r16,0x40                //loads r16 with 0x40
sts             ADMUX,r16               //stores value in r16 to data space ADMUX 0x7C
ldi             r16,0                    //loads r16 with 0
sts             ADCSRB,r16              //stores value in r16 to data space ADCSRB 0x7B
ldi             r16,0xFE                //loads r16 with 0xFE
sts             DIDR0,r16               //stores value in r16 to data space DIDR0 0x7E
ldi             r16,0xFF                //loads r16 with 0xFF
sts             DIDR1,r16                //stores value in r16 to data space DIDR1 0x7F
                ret                     //returns to line after init was called
.global LCD_Write_Command
LCD_Write_Command:
call            UART_Off                 //calls UART_Off to turn off the UART
ldi             r16,0xFF                ;PD0 - PD7 as outputs
out             DDRD,r16                //Stores value of r16 into DDRD address in SRAM
lds             r16,DATA                //loads value in data space DATA () into r16
out             PORTD,r16               //Stores value of r16 into PORTD address in SRAM
ldi             r16,4                   //load indirect 4 into r16
out             PORTB,r16                //Stores value of r16 into PORTB address in SRAM
call            LCD_Delay               //calls LCD_Delay function
ldi             r16,0                   //load indirect 0 into r16
out             PORTB,r16               //Stores value of r16 into PORTB address in SRAM
call            LCD_Delay               //calls LCD_Delay function
call            UART_On                 //calls UART_On to turn the UART on
ret                                     //returns to main function
LCD_Delay:
ldi             r16,0xFA                //loads r16 with 0xFA
D0:             ldi r17,0xFF            //start of D0 subroutine, loads r17 with 0xFF
D1:             dec r17                 //start of D1 subroutine, decreases r17 by 1
brne            D1                      //Repeat until r17 is 0
dec             r16                     //decreases r16 by 1
brne            D0                      //Repeat until r16 is 0
```

```
ret                             //returns to main function
.global LCD_Write_Data
LCD_Write_Data:
call        UART_Off            //calls UART_Off subroutine to turn off UART
ldi         r16,0xFF            //load indirect 0xFF into r16
out         DDRD,r16            //writes value in r16 to DDRD (0x0A)
lds         r16,DATA            //loads r16 with the value stored in DATA
out         PORTD,r16           //writes value in r16 to PORTD (0x0B)
ldi         r16,6               //load indirect 6 into r16
out         PORTB,r16           //writes value in r16 to PORTB (0x05)
call        LCD_Delay           //calls LCD_Delay subroutine
ldi         r16,0               //load indirect 0 into r16
out         PORTB,r16           //writes value in r16 to PORTB (0x05)
call        LCD_Delay           //calls LCD_Delay subroutine
call        UART_On             //calls UART_On subroutine
ret                             //returns to main function
.global LCD_Read_Data
LCD_Read_Data:
call        UART_Off            //calls UART_Off subroutine to turn off UART
ldi         r16,0x00            //loads r16 with 0x00
out         DDRD,r16            //writes value in r16 to DDRD port at 0x0A
out         PORTB,4             //writes value 4 to PORTB port at 0x05
in          r16,PORTD           //writes value at PORTD () to r16
sts         DATA,r16            //stores the value in r16 to data space DATA ()
out         PORTB,0             //writes value 0 to PORTB port at 0x05
call        UART_On             //calls UART_On subroutine to turn the UART on
ret                             //returns to main function
.global UART_On
UART_On:
ldi         r16,2               //Load immediate r16 with 2
out         DDRD,r16            //Copy bit 2 into port DDRD (port 10)
ldi         r16,24              //Load immediate r24 with 24
sts         UCSR0B,r16          //Copy r16(24) into UCSR0B SRAM Location(193)
ret                             //Return to Function this was called from
.global UART_Off
UART_Off:
ldi         r16,0               //Load immediate r16 with 0
sts         UCSR0B,r16          //Copy r16(0) into UCSR0B SRAM Location (193)
ret                             //Return to Function this was called from
.global UART_Clear
UART_Clear:
lds         r16,UCSR0A          //r16 is set with 192
```

[Group 1]

ECE-412-01

```
sbrs        r16,RXC0        //Skip next instruction if bit 7 of r16 is 1
ret                         //return to main function
lds         r16,UDR0        //loads r16 with the value stored in the data space at
                            //URDR0 ()

rjmp        UART_Clear      //relative jump to UART_Clear subroutine
.global UART_Get
UART_Get:
lds         r16,UCSR0A      //r16 is set to 192
sbrs        r16,RXC0        //Skip next instruction if bit 7 of r16 is 1
rjmp        UART_Get        //jumps to UART_Get (unconditional jump)
lds         r16,UDR0        //loads the value in data space at UDR0 to r16
sts         ASCII,r16       //Store ASCII global with value of r16
ret                         //return to main function
.global UART_Put
UART_Put:
lds         r17,UCSR0A      //r17 is set to 192
sbrs        r17,UDRE0       //checks if the bit register has been set
rjmp        UART_Put        //restarts function
lds         r16,ASCII       //grabs the stored value in ASCII and loads it into
//r16
sts         UDR0,r16        //stores r16 into stack
ret                         //return to main function
.global ADC_Get
ADC_Get:
ldi         r16,0xC7        //Loads 199 to r16
sts         ADCSRA,r16      //Stores r16 in stack
A2V1:
lds         r16,ADCSRA      //Loads store value from stack into r16
sbrc        r16,ADSC        //Skips next line if the Bit register was cleared
rjmp        A2V1            //Loops back up to A2V1 it Bit register wasn't
//cleared
lds         r16,ADCL        //loads 120 from .data section
sts         LADC,r16        //loads 120 into LADC so C code can access
lds         r16,ADCH        //loads 121 from .data section
sts         HADC,r16        //loads 121 into HADC for C code
ret                         //return to main function
.global EEPROM_Write
EEPROM_Write:
sbic        EECR,EEPE
rjmp        EEPROM_Write ;  Wait for completion of previous write
ldi         r18,0x00        ; Set up address (r18:r17) in address register
```

[Group 1]

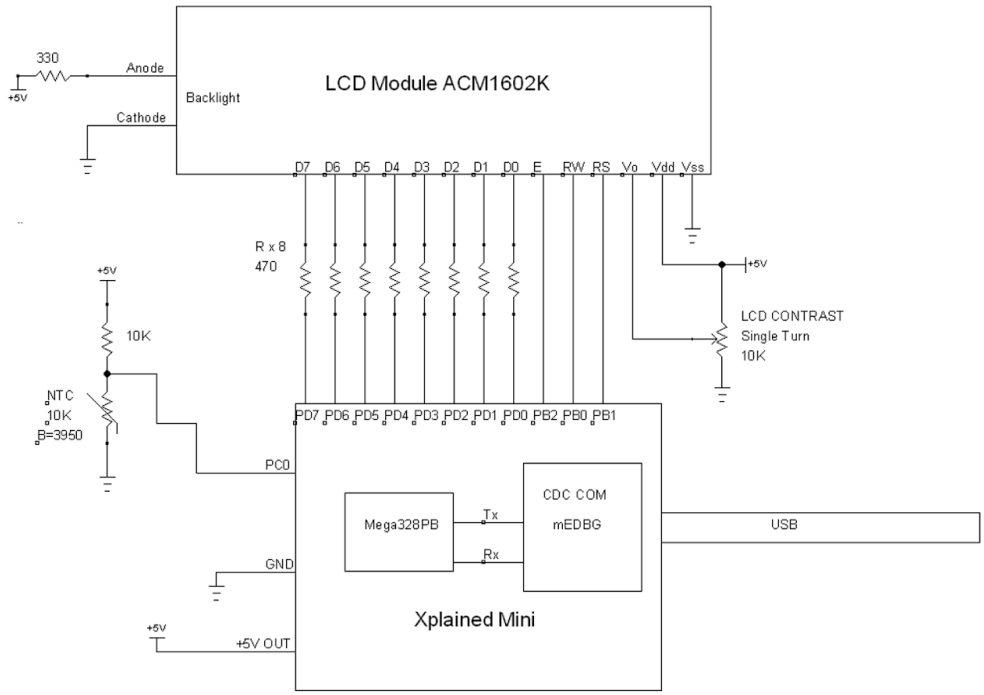
ECE-412-01

```
ldi             r17,0x05
ldi             r16,DATA                ; Set up data in r16
out             EEARH, r18
out             EEARL, r17
out             EEDR,r16                ; Write data (r16) to Data Register
sbi             EECR,EEMPE              ; Write logical one to EEMPE
sbi             EECR,EEPE               ; Start eeprom write by setting EEPE
ret
.global EEPROM_Read
EEPROM_Read:
sbic            EECR,EEPE
rjmp            EEPROM_Read             ; Wait for completion of previous write
ldi             r18,0x00                ; Set up address (r18:r17) in EEPROM address
;register
ldi             r17,0x05
ldi             r16,0x00
lds             r16,DATA
out             EEARH, r18
out             EEARL, r17
sbi             EECR,EERE               ; Start eeprom read by writing EERE
in              r16,EEDR                ; Read data from Data Register
sts             ASCII,r16
ret
.end
```

[Group 1]

ECE-412-01

Schematics (Hardware)



Title: ECE412 Lab 3 Circuit		
Author: Eugene Rockey		
File: C:\Users\Donkey\Desktop\Lab3demo.dsn		Document
Revision: 1.0	Date:	Sheets: 1 of 1

[Group 1]

ECE-412-01

Application-Based Analysis

Two important aspects of the project were to ensure correct and accurate measurement and the ability to display the temperature quickly/swap between the two types. Firstly, to ensure the accuracy of the measurement, calibration of the temperature was done. By viewing the ambient temperature reading in the library from the thermostat, values recorded from the 328PB could be compared and later altered. Secondly, to ensure the ability to swap measurements quickly, as soon as the display of the LCD was completed, the menu was instantly brought back up to start another measurement again. Additionally, every consecutive measurement would clear the screen LCD to only display the current temperature.

[Group 1]

ECE-412-01

Conclusion

The purpose of this project was to use the accumulation of all knowledge acquired in previous labs and apply it to create a final project that consists of input(s) entered by the user and the resulting output, which is relayed through the LCD display. The resulting project is able to convert the voltage acquired by the ADC to a temperature in both Celsius and Fahrenheit. Based on repeated experiments, the room temperature tended to be around 77 degrees Fahrenheit while the temperature of measured extremities tended to be around 93 degrees Fahrenheit.

[Group 1]

ECE-412-01

References

Atmel, AVR Assembler: User Guide. AVR instruction set manual. (n.d.). Retrieved April 25, 2022, from http://atmel-studiodoc.s3-website-us-east-1.amazonaws.com/webhelp/GUID-0B644D8F-67E7-49E6-82C9- 1B2B9ABE6A0D-en-US-1/index.html

Specifications of LCD Module. Retrieved April 25, 2022 from https://components101.com/sites/default/files/component_datasheet/16x2%20LCD%20Data sheet.pdf