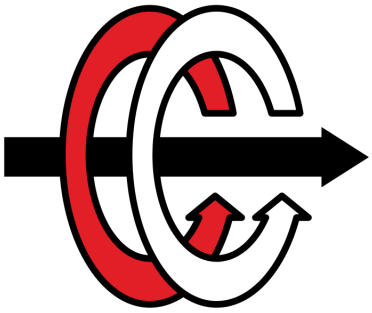
MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING



EE300

SUMMER PRACTICE REPORT

GURAY OZGUR
2167054

SP COMPANY: DARKBLUE TELECOMMUNICATION

COMPANY DIVISION: TELECOMMUNICATION

SP BEGINNING DATE: 25 JUNE 2018
SP ENDING DATE: 27 JULY 2018

SUPERVISOR: ALI MANCAR, alimancar@darkbluesystems.com

Contents

1 INTRODUCTION **2**

2 DESCRIPTION OF THE COMPANY **2**
- 2.1 Company Name . . . 2
- 2.2 Company Location . . . 2
- 2.3 Contact Information of the Company . . . 2
- 2.4 General Description of The Company . . . 3

3 WORK CONDUCTED AT THE SP LOCATION **3**
- 3.1 GETTING STARTED . . . 3
  - 3.1.1 What is USART? . . . 4
  - 3.1.2 What is SPI? . . . 4
  - 3.1.3 What is I^2C? . . . 5
  - 3.1.4 What is ICSP? . . . 6
  - 3.1.5 What is USB Bootloader? . . . 6
  - 3.1.6 What is ADC? . . . 7
- 3.2 FIRST CIRCUIT BOARD . . . 8
  - 3.2.1 Quartz Crystal Oscillators . . . 8
  - 3.2.2 Fixed-Voltage Regulator Application with MIC5209-5 . . . 8
  - 3.2.3 Analog Temperature Sensor, LMT87-Q1 . . . 9
  - 3.2.4 Analog to Digital Conversion with ADC081C021 . . . 9
  - 3.2.5 Driving RS232 with MAX232IDR . . . 10
  - 3.2.6 Constructing the Board . . . 10
  - 3.2.7 Programming the PIC18F26K22 . . . 13
- 3.3 SECOND CIRCUIT BOARD . . . 15
  - 3.3.1 Constructing the Board . . . 15
  - 3.3.2 Programming the PIC18F4550 . . . 18
  - 3.3.3 Making an Interface for USB HID . . . 20

4 CONCLUSION **23**

5 REFERENCES **24**

6 APPENDICES **25**
- A MPLAB Code, Project1 . . . 25
- B MPLAB Code, Project2 . . . 30
- C Microsoft Visual Studio Code and USB HID, Project2 . . . 32

1 INTRODUCTION

I have performed my first summer practice in Darkblue Telecommunication Systems, which is a company founded in 2001 by Alper Tosun. The reason why I chose Darkblue is that I was interested in embedded circuit design and Darkblue could give that opportunity to me. The company can be regarded as a small company, however having a small population should not imply that it does not have dedicated engineers and technicians working in collaboration and ambition and I had a chance to work with those great people for five weeks, from 25.06.2018 to 27.07.2018. During my internship period, my supervisor was Ali Mancar who has B.Sc. and M.Sc. degrees in our department.

What my supervisor wanted me to do was basically to design, produce, program and create an interface for a given purpose. While doing that, I needed to use Proteus 8.5, PICkit2, MPLAB X IDE, Ardunio IDE, and Visual Studio 2017 and for plotting the circuit board I used LPKF ProtoMat circuit plotter machine.

In this report, there are several sections about my internship period and my observations, which begins with some information about the company. Then, general concepts about embedded circuit design and programming with their comprehensive descriptions are discussed, work conducted in SP company is explained with detailed information. Various figures, schematics about the circuitry are included for clarifications.

2 DESCRIPTION OF THE COMPANY

2.1 Company Name

2.2 Company Location

Silicon Building, 10-A, METU Technopolis, 06800 Cankaya/Ankara

2.3 Contact Information of the Company

Telephone / Mail: (0312) 210 19 29 / a@darkbluesystems.com

2.4 General Description of The Company

Darkblue Telecommunication Systems has 34 employees including 22 engineers, 4 technicians and 1 industrial designer, which enables the company design and produce their own devices from scratch. 8 of them are electric and electronic engineers and they do R&D activities in telecommunication and RF areas. In this process, they produce printed circuit board (PCB) prototypes by using a circuit board plotter machine produced by LPKF, and a 3D printer is used by the industrial designer for the cover of the prototypes. After testing software on the prototype, other companies are hired for batch production, i.e. producing PCB and pick and place procedures. Moreover, the company has a Software Development Unit where 16 computer engineers work. Their job is mainly to create interfaces for these devices and develop mobile applications such as Secureforce, and Mobizoom. Since the company is located in Technopolis area of METU, naturally it is affected by the research environment of the university. That's why, R&D is the unit where most of the money is spent and earned. The company serves these end products mostly to companies located abroad. Just because of this, the company has two more branches in Dubai and Singapore. Furthermore, the company wants students to be knowledgeable in this area, hosts several university students who are eager to learn, and helps them to improve their knowledge about the field. As one of these students, I had a chance to see the environment in the Hardware Development Unit between R&D engineers, shown in Figure 1.

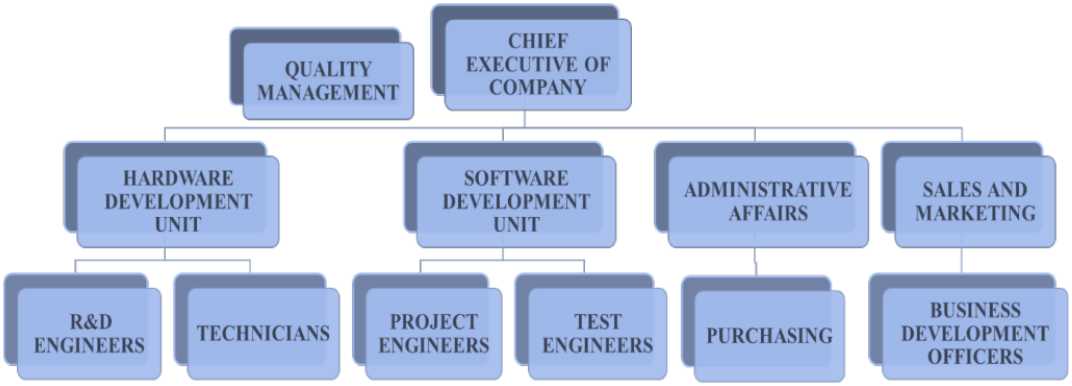


Figure 1: Administrative structure of Darkblue Telecommunication Systems

3 WORK CONDUCTED AT THE SP LOCATION

3.1 GETTING STARTED

During my internship period, I was given two main projects which aim to teach how to use microcontrollers and what can be done by using microcontrollers. That's why, I was advised

to make a research about PIC microcontrollers, USART, SPI, I^2C, ICSP, USB Bootloader, and ADC, which I was not aware of their existence before.

3.1.1 What is USART?

USART, Universal Synchronous-Asynchronous Receiver/Transmitter, is a circutry implementing serial communication either synchronously or asynchronously, that is, transmitting and receiving serial data with or without a clock signal. In more detail, USART takes parallel data, coming from a controlling device and converts it to a serial one. Then, transmitting it to another USART enables the data be converted back to the paralel form for the receiving device. Data flows from the TX (tranmitter) pin of the transmitting USART to RX (receiver) pin of the receiving USART. While transmission in UART communication, instead of clock signal a start and a stop bit are added to the data packet so that receiving UART understands the beginning of the data packet. Additionally, a parity bit is used for error checking. Another difference is in asynchronous mode transmission is full-duplex meaning that one can transmit and send data at the same time. Figure 2 shows an illustration in UART communication, which is the data transmission of 01001101 [1].

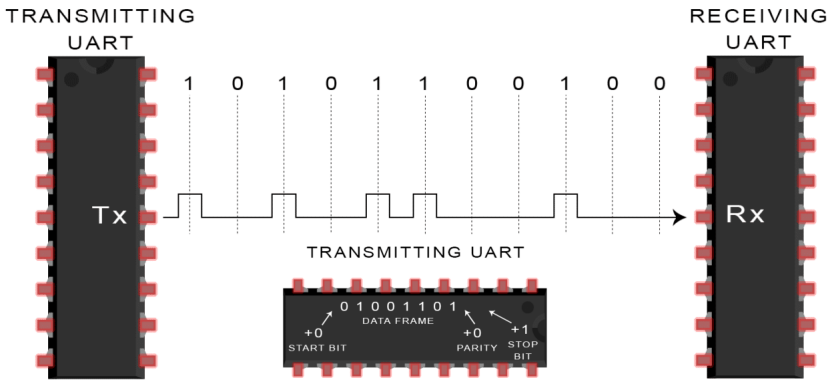


Figure 2: UART data transmission diagram

3.1.2 What is SPI?

SPI, Serial Peripheral Interface, is a synchronous communication protocol which transmits the data serially. In SPI protocol, there is a master-slave relationship between the devices trying to communicate, where master device is generally a controlling device. One master can exist and slaves can communication only with the master. In Figure 3, SPI is illustrated for a multiple slave system. Master transmits the data to the slave via MOSI (Master Output/Slave Input). Likewise, slave transmits the data to the master via MISO (Master Input/Slave Output). By having two seperate lines, transmission can be done at the same

time. SCLK (Clock) should send the clock signal since SPI is a synchronous protocol. SS/CS (Slave Select/Chip Select) enables the master choose which slave to communicate with for a multiple slave system. There is no start or stop bits, so data flow can keep going continuosly without any interruption, which is a clear advantage over UART. However, there is also no parity bits, which makes it error-prone [2].

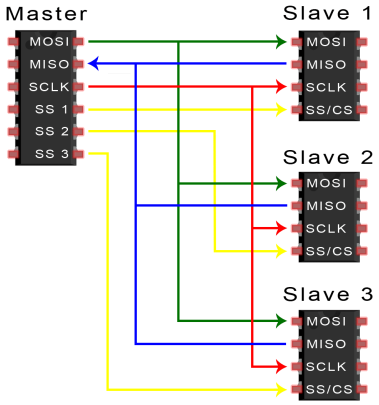


Figure 3: SPI master and slave configuration

3.1.3 What is I^2C?

I^2C Protocol, Inter-Integrated Circuit Protocol, is a synchronous, serial, multiple master, and multiple slave communication protocol. In Figure 4, I^2C for a multiple slave system is shown. SDA (Serial Data) is the line where the master and slaves exchange data alternatively and SCL (Serial Clock) is the line where the clock signal is transmitted. Since all the slaves are connected with a single line, I^2C needs a way to call its communication partner. After the start condition, the master sends the address of the slave it wants to communicate with. Then the slave answers its call with a ACK (acknowledge) bit, which follows with the master sending the data sequence. The slave informs the master by enabling ACK/NACK bit or not and the master stops the transmission with a stop bit. For a multiple master system, the master first checks the SDA line. In brief, I^2C uses only two lines to control the communication between multiple masters and multiple slaves, which indicates that I^2C is superior than UART. However, it is not as fast as SPI, since the data is interrupted constantly and unlike SPI, data sequence is limited to 8-bits [3].

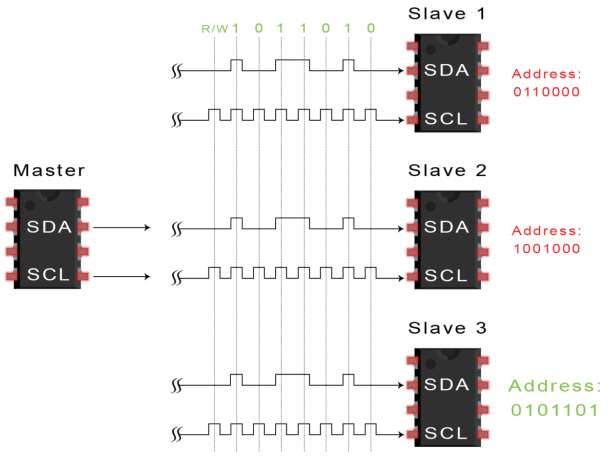


Figure 4: I^{2}C data transmission diagram

3.1.4 What is ICSP?

ICSP, In-Circuit Serial Programming, is a way to program the microcontroller without separating it from the circuit. A Surface-Mount Device (SMD) is very small and separation of a microcontroller of that size for programming is unrealizable. However, ICSP can done by using two lines for data and clock, and three lines for programming voltage, power, and ground.

3.1.5 What is USB Bootloader?

USB Bootloader provides a self-programming feature for microcontrollers by allocating some of the memory. Once microcontrollers' flash memory is allocated by the bootloader, it can reprogram the microcontroller without an external programmer or ICSP over and over again, which is shown in Figure 5. However, deficiency of using one is that memory for the main application reduces [4].

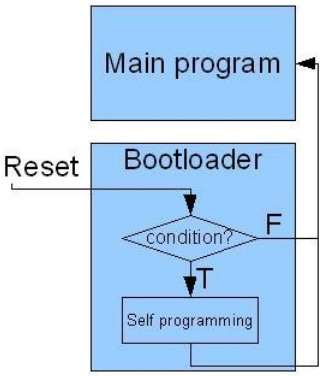


Figure 5: Working principle of a bootloader

3.1.6 What is ADC?

ADC, Analog to Digital Converter, transfers the analog signal to a digital signal by applying some processes, given as sampling, quantization and coding, as seen in Figure 6. Sampling is a procedure which takes the amplitude of the analog signal at discrete time intervals, and these amplitude values are fixed to certain levels by quantization. Assinging binary values to those levels is called coding. As a result, an analog signal is converted to a digital one [5].

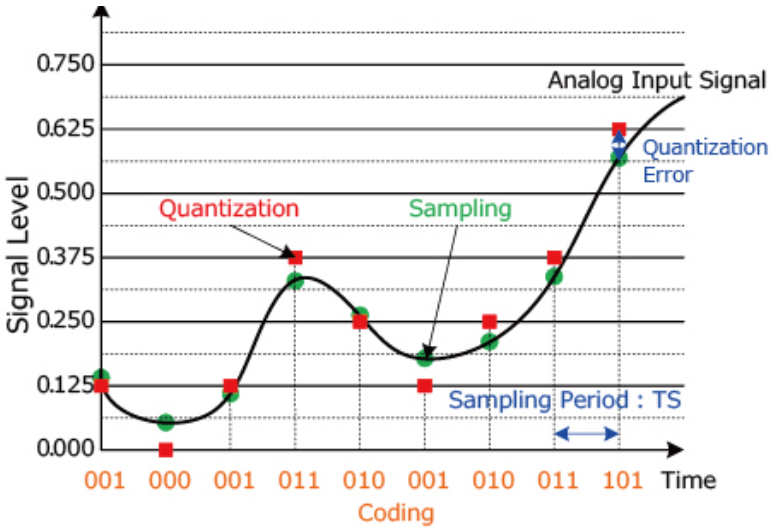


Figure 6: An illustration of ADC procedure

3.2 FIRST CIRCUIT BOARD

My first design project aimed to get the insight of the concepts mentioned earlier and learn about the essentials such as making schematics of the circuit and preparing the PCB layout by using Proteus. Gerber file obtained from the Proteus enabled me to print out the circuit, and I had an experience of soldering such small circuit elements. In the project, I prepared my code in MPLAB, which is provided by Microchip Inc. for their microcontrollers. For this case, it was PIC18F26K22. Also, there were several circuit elements, like MIC5209-5 for adopting the supply voltage, ADC081C021 for analog to digital conversion, LMT87-Q1 as an external temperature sensor, and MAX232IDR for driving RS232 connector.

3.2.1 Quartz Crystal Oscillators

Quartz Crystal Oscillators create a pulse train and crystal adjusts the frequency of the wave by its fundamental frequency. This wave is used by the microcontroller to do the system timing and clock signals. In our case, PIC18F26K22 has an internal oscillator having selectable frequencies from 31 kHz to 16 MHz and also by using PLL, Phase Locked Loop, its performance can be four times faster. Moreover, it supports two external oscillators up to 64 MHz [6]. For our application, one crystal oscillator whose fundamental frequency is 64 MHz is adopted. OSC1 and OSC2 pins are used with the circuit shown in Figure 7.

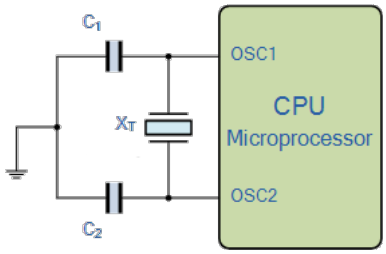


Figure 7: Crystal oscillator circuit diagram

3.2.2 Fixed-Voltage Regulator Application with MIC5209-5

In our circuit board, PIC18F26K22 should function at 5 Volts, that's why we have to search for a solution to convert the output voltage of the adapter, that is 12 Volts, to 5 Volts. For this purpose, MIC5209-5 is a suitable candidate. Checking the datasheet of MIC5209-5, I chose to use the circuit in Figure 8 with the addition of an indicator LED [7].

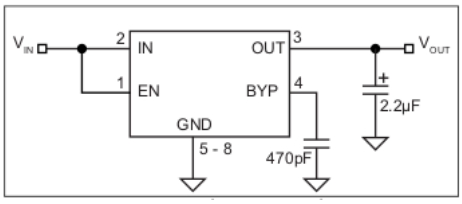


Figure 8: Ultra-low-noise fixed-voltage application of MIC5209-5

3.2.3 Analog Temperature Sensor, LMT87-Q1

Analog Temperature Sensors are required to sense the temperature of the surroundings to decide whether cooling fan should work or not. I chose to use LMT87-Q1 for this design. To understand the temperature value, we should look at the Equation 1, which is given in the datasheet of LMT87-Q1 [8].

$$T = \frac{13.582 - \sqrt{(-13.582)^2 + 4 * 0.00433 * (2230.8 - V_{TEMP}(mV))}}{2 * (-0.00433)} + 30 \qquad (1)$$

3.2.4 Analog to Digital Conversion with ADC081C021

ADC081C021 is an I^{2}C-Compatible, 8-bit Analog to Digital Converter, whose aim is simply to obtain a digital signal corresponding to an analog signal. To see how it works, it is asked to connect a potentiometer to the ADC module as an input. Moreover, LMT87-Q1 is an analog temperature sensor, so we have to make use of an ADC. Minimal required connections are shown in Figure 9 with pull-up resistors for I^{2}C [9]. One should also note that since ADC081C021 is a 8-bit converter, supplied 5 Volts corresponds to $2^8 - 1$, which is shown in Equation 2.

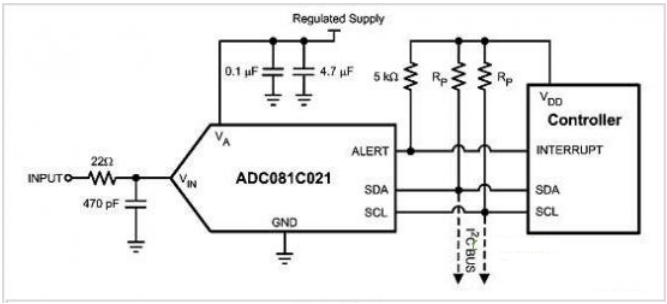


Figure 9: Typical application circuit of ADC081C021

$$\frac{255}{5} = \frac{ADCReading}{AnalogVoltageMeasured} \tag{2}$$

3.2.5 Driving RS232 with MAX232IDR

In telecommunications, RS232 whose pinout is given in Figure 11 is a recommended standard for connecting computer and the device to allow the flow of serial data between them. Choosing MAX232IDR for its driver was a sound decision. The minimal required connections of MAX232x is in Figure 10 [10].

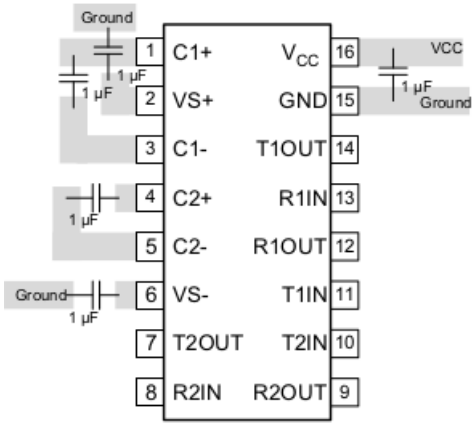


Figure 10: Layout schematic for MAX232x

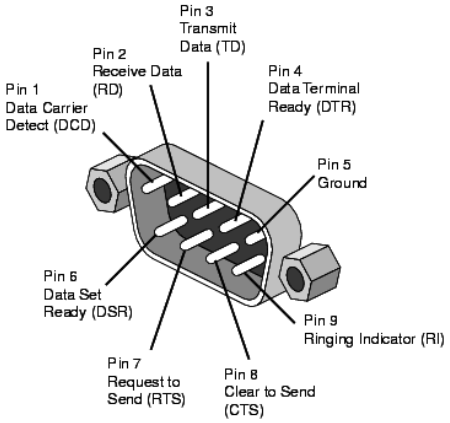


Figure 11: RS232 connector pinout

3.2.6 Constructing the Board

Now that we have all the information about individual parts, we can start to unite parts of the circuit as a whole. First and foremost, TBLOCK-2 is used for voltages coming from adapter. For ICSP, i.e. for connection with PICkit, a SIL5 connector is adopted. Two push buttons and two LEDs are included. To stabilize the data that is sent from LMT87-Q1, a resistor and a capacitor is placed to the input pin and two TUNIK-3 connectors are also placed since the temperature sensor is external. Two TUNIK-4 connectors are placed for GPIO, General Purpose Input/Output pins. Oscillator, MIC5209-5, ADC081C021, and MAX232IDR are formed as described earlier. Putting them all together, we obtained our schematics for the circuit as seen in Figure 12.

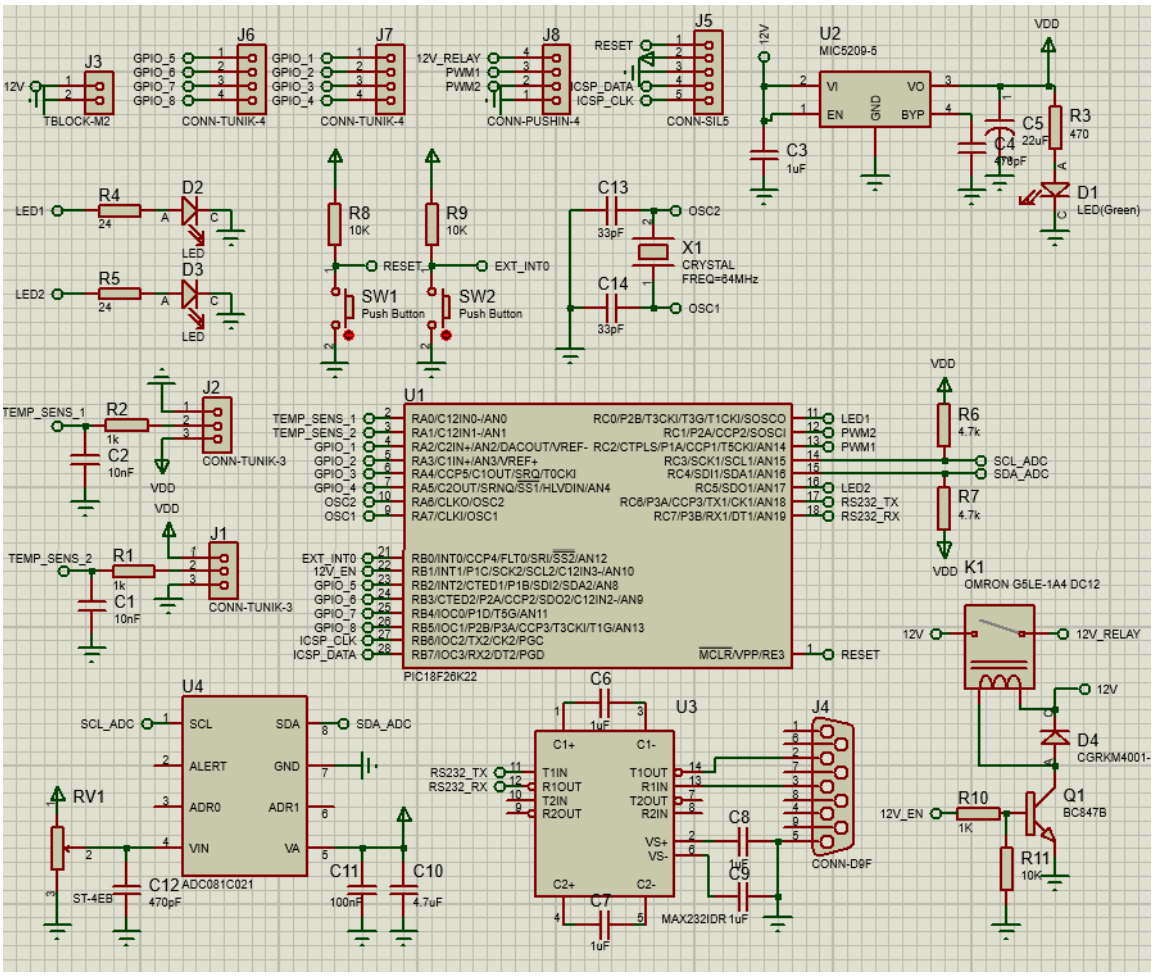


Figure 12: Overall schematics of the circuit with PIC18F26K22

Making schematics of the circuit is not enough to realize the circuit because this circuit can be made plenty of different ways thanks to their designers. Design of the PCB layout is crucial for that matter. One thing to remember is packagings of the elements, it could be PTH, Plated-Through Hole, or SMD, Surface-Mount Device. I wanted to use SMD packages for their small sizes and the company had a broad range of packages for the elements, which I chose from a list being handed out to me. Another thing is number of layers, PCBs can have various number of layers, such as 1-Layer, 2-Layer, or 4-Layer. I did not need a 2-Layer PCB, as my circuit board was a kind of test board and I managed to place them in a compact way by just drilling two vias for the crossing traces. PCB layout of the circuit is given in Figure 13 and 3D modeling can be seen in Figure 14.

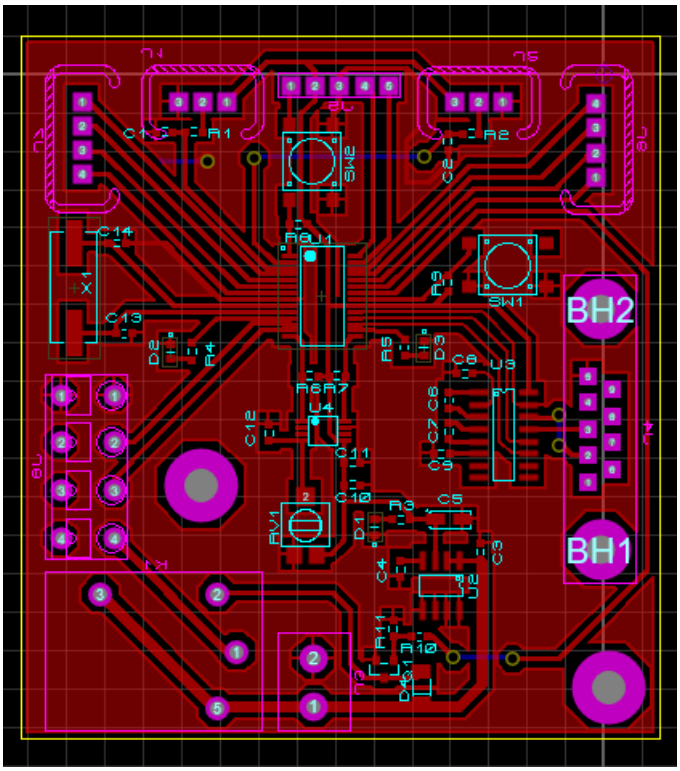


Figure 13: PCB layout of the circuit given in Figure 12

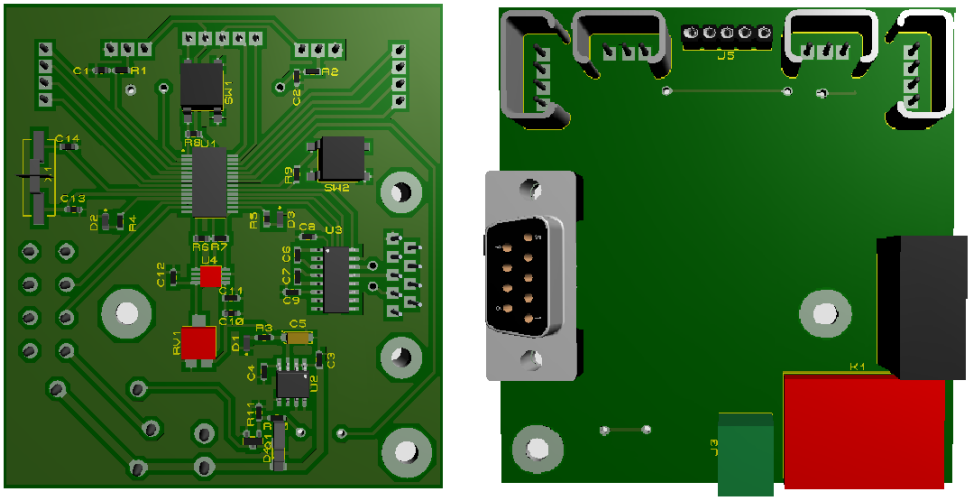


Figure 14: 3D modeling of the circuit board

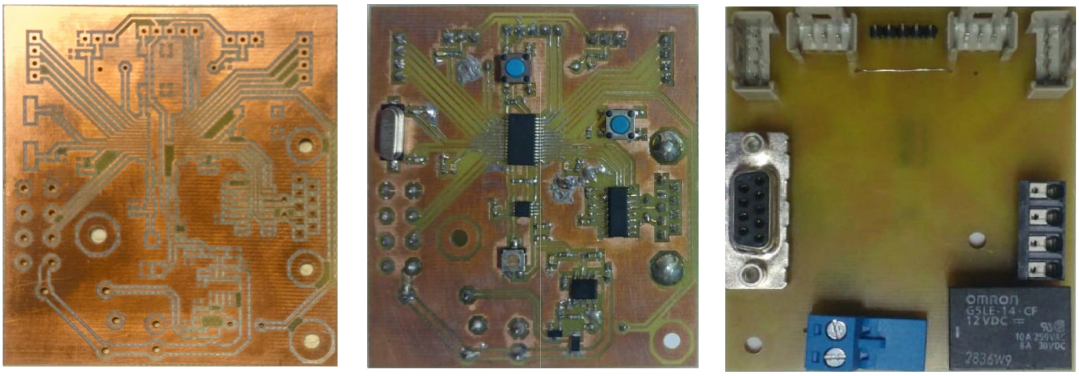


Figure 15: Prototype of the circuit board plotted with LPKF-S63

Challenging task was soldering, which I think that I was elated the most. After some guidance from the technicians of the company, I was able to make all soldering done, which was very satisfying when I compared the circuit board with 3D modeling. The circuit before the soldering and after the soldering can be seen in Figure 15.

3.2.7 Programming the PIC18F26K22

REGISTER 16-1: TxSTAx: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

bit 6 **TX9:** 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN:** Transmit Enable bit(1)
1 = Transmit enabled
0 = Transmit disabled

bit 4 **SYNC:** EUSART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 2 **BRGH:** High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode

Figure 16: Register Definitions: EUSART control

It is time to test the circuit board. I want to illustrate an example showing how to code the PIC18F26K22 with MPLAB X IDE. Every module or protocol is controlled by some registers defined in the datasheet of the microcontroller. As seen in Figure 16, TXSTAx is one of those registers containing 8 bits and I chose to show some of bits of the register TXSTAx, such as TX9, TXEN, SYNC, and BRGH. This register is one of the registers to adjust if one wants to configure the USART protocol [6]. Bits can be adjusted one by one or all register can be adjusted by a corresponding decimal number.

```
TXSTA1bits.SYNC = 0;    //Asynchronous mode
TXSTA1bits.TX9 = 0;     //Selects 8-bit transmission
TXSTA1bits.TXEN = 1;    //Transmit enabled
TXSTA1bits.BRGH = 1;    //High speed
```

is a sample for adjusting the bits one by one from the Appendix A, which is the code that I write for first circuit board. Datasheet also includes the protocols and what to do to enable those protocols step by step. ADC module is adjusted by three registers, i.e. ADCONx registers. CCP, Capture/Compare/PWM, module is adjusted by configuring CCPxCON, CCPRxL and T2CON registers. For the digital inputs or outputs, TRIS bit of that pin should be arranged so that the pin is assigned to an input or output. PORT and LAT bits are used for reading the input pin and writing the output pin respectively. USART protocol is adjusted from TXSTAx, RCSTAx, and BAUDCONx registers, also when TRMT is empty, data is sent through TX pin by writing it TXREG register byte by byte [6]. In the same manner, SPI and I^2C protocols are configured. I have tested CPP, ADC, UART, and I^2C, by sensing the temperature from external temperature sensors and controlling the cooling fan with PWM according to them and sending that information to the computer screen by using either UART or I^2C as seen in Figure 17. Besides, all code can be found in Appendix A for more detail.

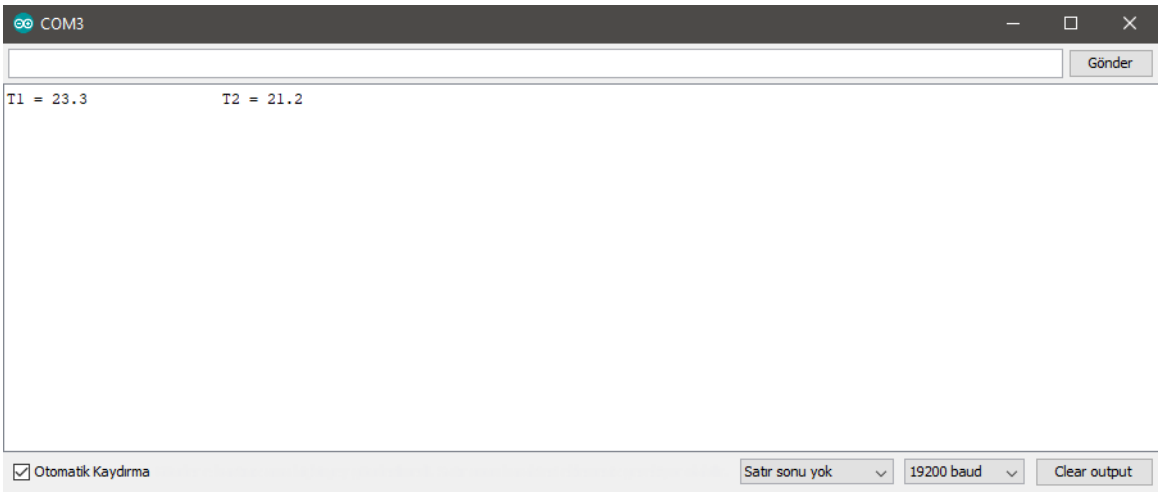


Figure 17: Arduino IDE screen showing the temperatures of the sensors, from RS232 port

3.3 SECOND CIRCUIT BOARD

For the design of the second circuit board, I had an actual purpose, making the board PICkit free. The reason why the company is asking such a request is that sometimes the products need to be field upgradeable several times or same circuit board can be used for different purposes in different environments and programming it with an external kit like PICkit 2 is not easy when it comes to numbers. Besides, the company may want the customer to update its software when it comes to distances and can not expect the customer to have a programmer device. The company designs a great number of boards as their products and those boards usually have USB sockets. Having an USB socket has a huge advantage if PIC has a special feature which is self-programmability and designer knows all about it. With this feature, the device can write their own memory by the help of an internal software control called bootloader routine. I was assigned to find a way how it can be done without using excessive components. If possible, I had to remove the additional elements and make it software-controlled through an interface.

3.3.1 Constructing the Board

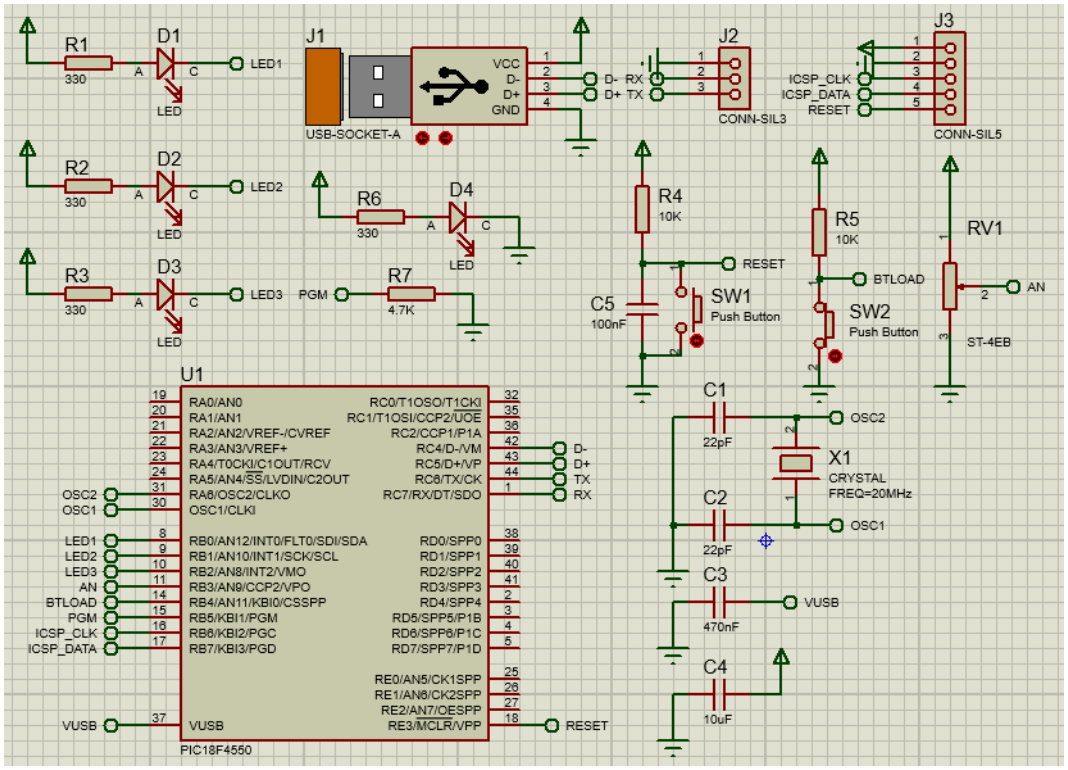


Figure 18: Overall schematics of the circuit with PIC18F4550

As you can see from Figure 18, since I was only trying to make the circuit be programmable from the USB, there are not many components. That's why, most of the pins of PIC18F4550 is left unconnected, which makes the circuit very small.

I did not want to use an external power supply while I can use USB port with this design. In Bus Power Only mode, all power for the application is drawn from the USB. This is effectively the simplest power method for the device as seen in Figure 19 [11].

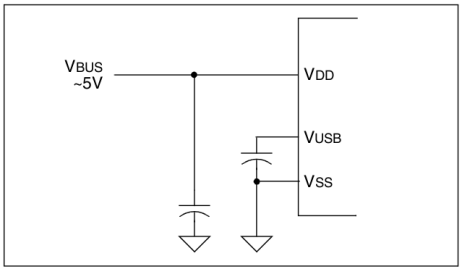


Figure 19: Bus Power Only mode of PIC18F4550

To see that the circuit is powered, power indicator LED is included. Three pins are reserved for three LEDs to check during programming for various purposes. For an analog signal, potentiometer is adopted. An external crystal oscillator having 20 MHz of fundamental frequency is used. Making communication easy with the computer, I chose to use an USB-TTL shown in Figure 20.

Figure 20: USB to TTL serial PCB assembly

Microchip Inc. distributes an interface to program the device for these type of PICs for bootloader routine. To attach the device to this interface, the device should be reset while RB4 is low. In other words, RB4 pin should be connected to ground to activate the bootloader. For that matter, a switch is adopted for that pin and another switch for reset. PGM pin might activate ICSP mode when LVP, Low Voltage Programming, bit is enabled [11]. That's why, it is better to connect a resistor from PGM to ground. Otherwise, PIC does not run the bootloader, which looks like it is disconnected. Lastly and unfortunately, SIL5 connector is adopted for ICSP since PIC does not include the bootloader at the beginning. For

once, there is no way but to use ICSP to program the bootloader to the device. However, it can be removed after installation of the bootloader.

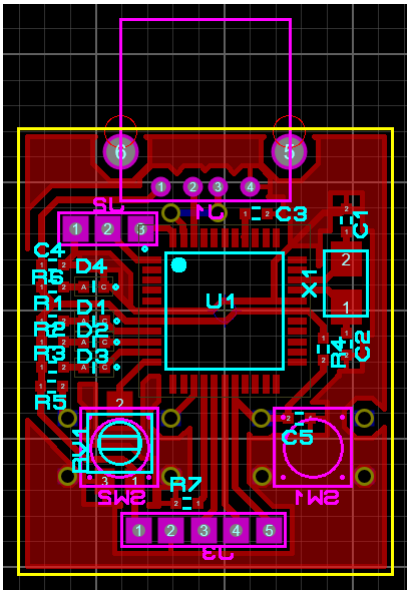


Figure 21: PCB layout of the circuit given in Figure 18

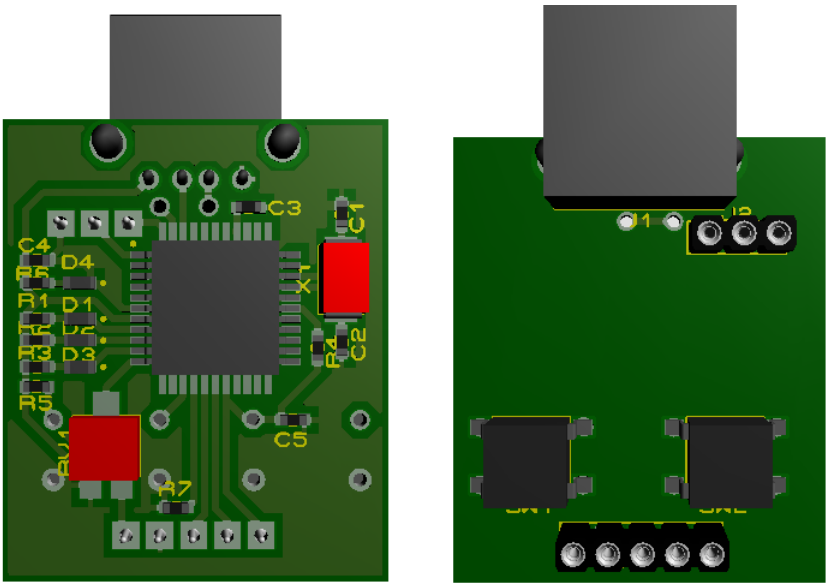


Figure 22: 3D modeling of the circuit board

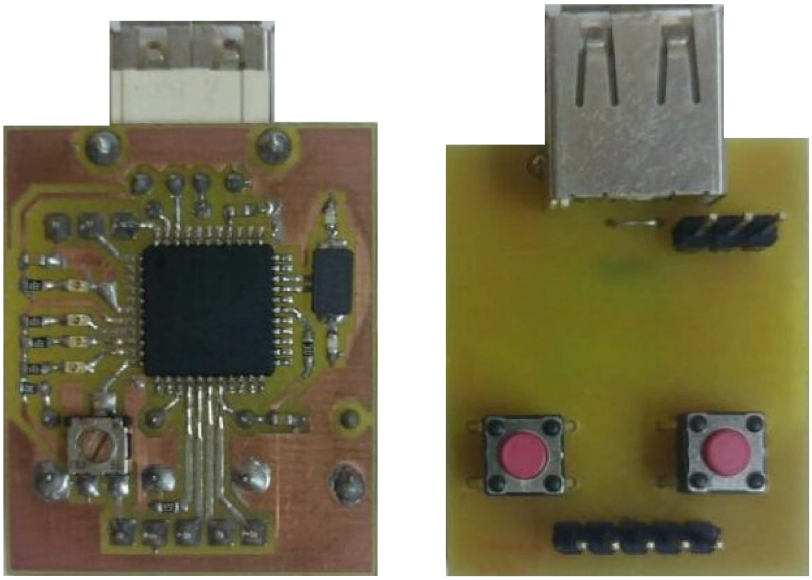


Figure 23: Prototype of the circuit board plotted with LPKF-S63

After making the PCB layout of the circuit board including PIC18F4550, in Figure 21, I plotted the circuit board in LPKF-S63 with its Gerber file and soldered it neatly. 3D modeling of the circuit is shown in Figure 22 and soldered version of the circuit is in Figure 23.

3.3.2 Programming the PIC18F4550

I have started with the old-fashioned way, i.e. programming the device by the help of a PICkit. I have tested my code written in MPLAP and shown in Appendix B to my device, which makes the circuit read the analog value from potentiometer, then convert it to its digital form and finally transfer the data via USB-TTL to the computer screen as shown in Figure 24. Once I checked whether it is working as it should be, I proceeded the main subject of the project.

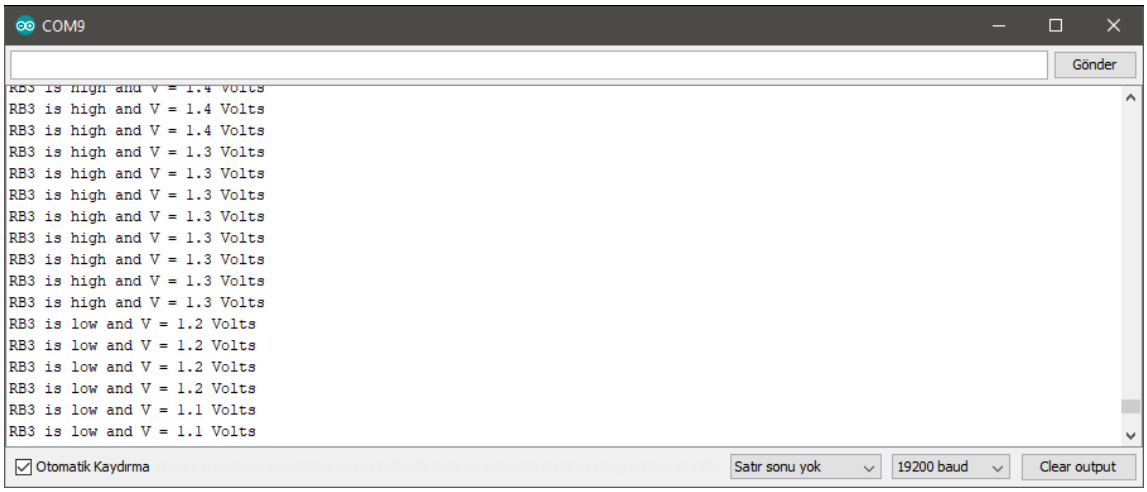


Figure 24: Arduino IDE screen showing analog voltage of the potentiometer, from USB-TTL

Microchip Libraries for Applications, MLA, is a library including various projects for different applications and it provides software packages consisting of drivers, demos, documentations, and utilities as well as source codes for projects [12]. Source code for the bootloader was too complicated, and MLA was uniquely suited to this application. However, it would not be possible if the company did not purchase the MPLAB XC8 Compiler PRO Dongle License as Microchip Inc. is only providing the source code and compiling the code without PRO mode does not leave enough space for user code, i.e. main application.

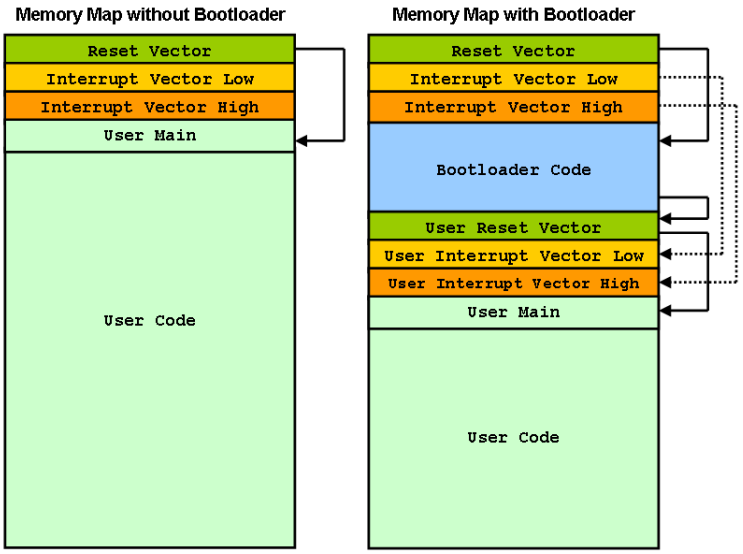
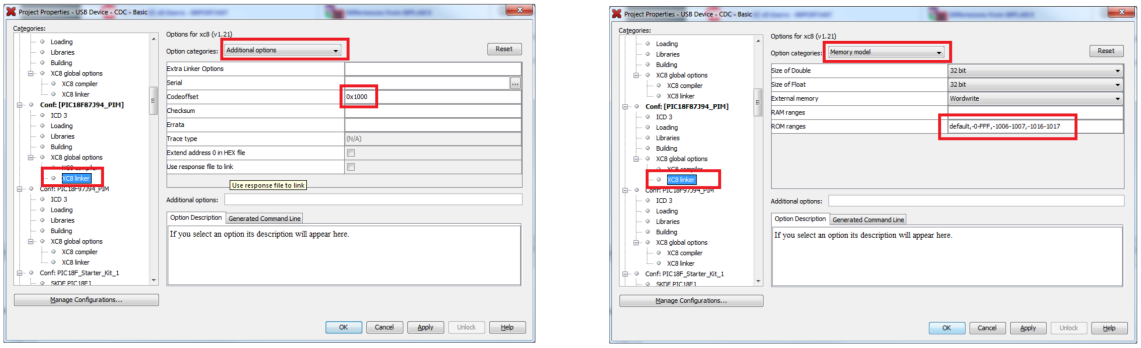


Figure 25: Memory map of PIC18F4550 with and without bootloader

As we can see from Figure 25, an illustration of memory map is shown with the effect of bootloader, which also indicates that user code should be offsetted.



(a) Codeoffset option

(b) ROM ranges option

Figure 26: Options that should be made before compiling

As shown in Figure 26, Codeoffset must be set to "0x1000" and ROM ranges must be set to "default,-0-FFF,-1006-1007,-1006-1007" [12].

After compiling the source code by adding the .hex file from the source code of bootloader to the "Loadables", main.hex file is obtained. PICkit 3 is used for the last time to program. Now, programming can be done by USB Bootloader v2.15, which is the interface provided by Microchip Inc. mentioned earlier. The device is attached by pressing reset button while keeping pressed the bootloader button in USB Bootloader v2.15. Then, the source code of the main application is selected, programming will be done with the Erase/Program/VerifyDevice sequence, which is illustrated in Figure 28.

3.3.3 Making an Interface for USB HID

Now, the second part of this project is optimization. We still need a way to get rid of two big buttons that are necessary to program the device, which will be inside of a box containing several circuit boards. Reset button is connected to RE3 pin and bootloader button is connected to RB4 pin. If we can open the interface, USB Bootloader v2.15, which is in a fixed directory, and then change the RE3 pin from high to low by keeping RB4 low, the device will be attached to the interface, and will be ready. We will be also needing a reset button again after the code is installed, since we do not want to unplug it every time to reset the device.

I wanted to make a software for this device so that we do not need those buttons. To do that, the device and the computer should communicate with each other. Since USB can also be used for data transmission, it is well suited for this task and we do not need anything else. By using Microsoft Visual Stduio and C++, I have prepared USB HID Interactive Software, which is shown in Figure 27. I have also included the Microsoft Visual Stduio code and software itself to the Appendix C for more information. I tried to show everything in the software, however this circuit is only made for understanding of bootloader routine, so the software only controls 3 LEDs, reset button and bootloader button. ENTER BOOTLOADER MODE button makes things easier since it includes all the combination to enter the mode, and opens the interface when pressed like in Figure 28. Voltage indicator shows the voltage over potentiometer, which was shown with Arduino IDE port screen earlier in Figure 24, and Pushbutton State shows whether bootloader button is pressed or not.

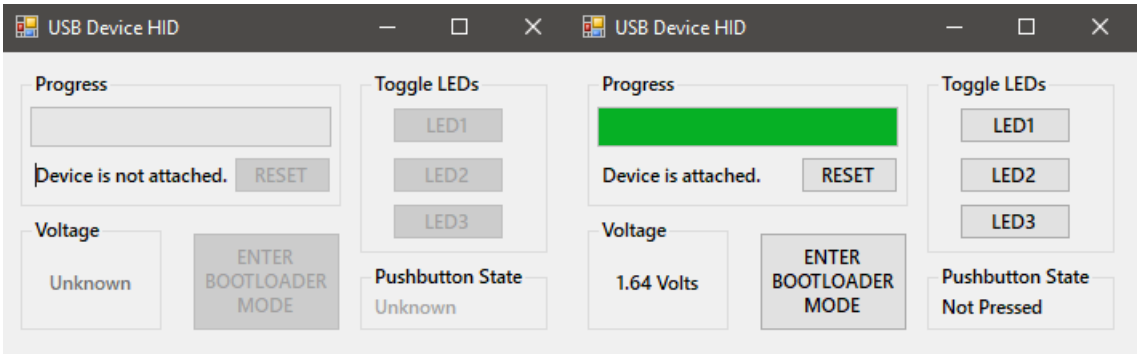


Figure 27: USB Human Interface Device Interactive Software

In this way, we could omit the buttons and make the circuit even smaller and easier to use every time when it is connected. I have prepared a report for this work on request and submit it to the person who can make use of this information.

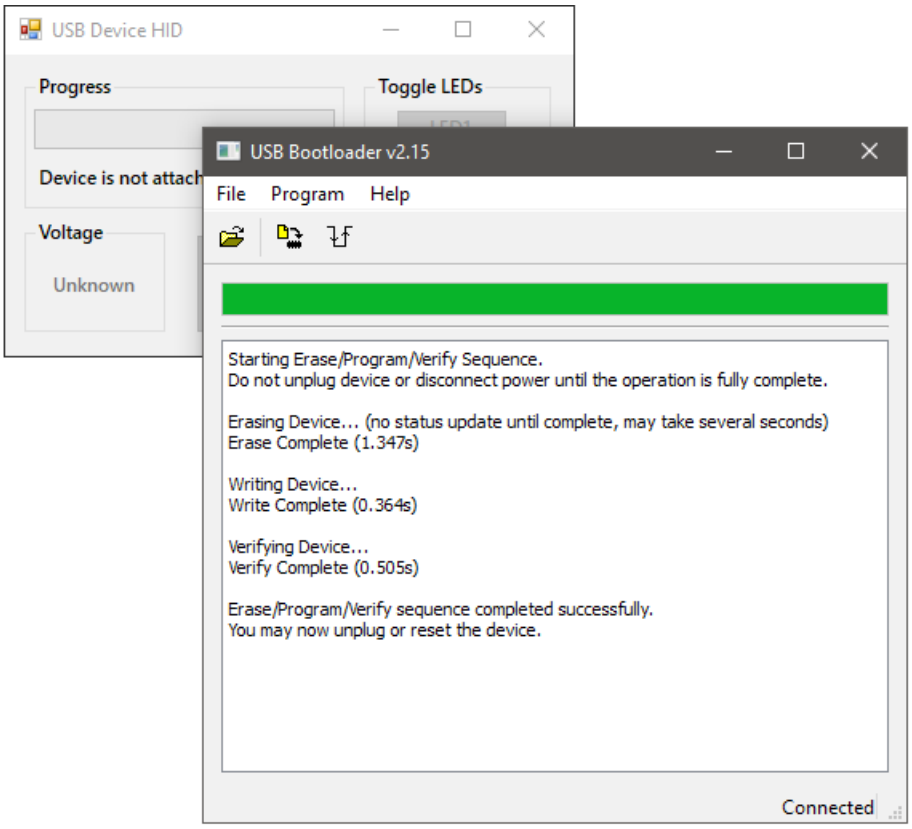


Figure 28: USB Human Interface Device Interactive Software, Bootloader Mode Activated

4 CONCLUSION

I spent my five weeks interning with DarkBlue Telecommunication Systems, which was my first experience at the field. Although I did not have a comprehensive engineering background, since I was in the second year of my degree, I was eager to learn and employees helped me to improve my knowledge about the field during the internship period and gave me that further exposure I need to advance in my career.

Additionally, I felt like I was able to contribute to the company by assisting on projects throughout my summer practice. I observed all the production process and tried to duplicate that procedure from the beginning. In this journey, I learned how to use PIC microcontrollers and had a knowledge about USART, SPI, I^2C protocols, etc. I designed two circuit boards, prepared their PCB layouts, printed out the circuit boards and soldered the circuit elements to the boards. Then I wrote codes in MPLAB for the microcontrollers. To be able to do these, I benefited from datasheet of each circuit element and vast knowledge of my supervisor, which contributed to me a lot.

Not only did I gain practical skills but I felt the great atmosphere at the office, which was always welcoming. Everyone was allowed to go to the kitchen at any time and have their favourite snacks, the hazelnuts especially. Generously, the boss organized a lunch for the employees and all interns were also invited, which made us feel like a part of the company.

All in all, I can say that my summer practice has been a success. I had an great opportunity to learn new things, feel the work environment and meet with incredible people. I could not be more thankful, and definitely I recommend DarkBlue Telecommunication Systems for a summer practice location.

5 REFERENCES

[1] "Basics of UART Communication," Apr 2017. [Online]. Available: http://www.circuitbasics.com/basics-uart-communication/

[2] "Basics of the SPI Communication Protocol," May 2018. [Online]. Available: http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/

[3] "Basics of the I^2C Communication Protocol," Apr 2017. [Online]. Available: http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

[4] R. M. Fosler and R. Richey, *A FLASH Bootloader for PIC16 and PIC18 Devices*, Microchip Technology Inc., Aug 2004.

[5] "A/D Converter and D/A Converter." [Online]. Available: https://www.rohm.com/electronics-basics/ad-da-converters/what-are-ad-da-converters/

[6] 28/40/44-Pin, Low-Power, High-Performance Microcontrollers with XLP Technology, PIC18(L)F2X/4XK22 datasheet, Microchip Technology Inc., 2016.

[7] 500 mA Low-Noise LDO Regulator, MIC5209 datasheet, Microchip Technology Inc., 2017.

[8] LMT87-Q1 2.7-V, SC70, Analog Temperature Sensors With Class-AB Output, LMT87-Q1 datasheet, Texas Instruments, Oct 2017.

[9] ADC081C021/ADC081C027 I 2 C-Compatible, 8-Bit Analog-to-Digital Converter with Alert Function, ADC081C021 datasheet, Texas Instruments, Mar 2013.

[10] MAX232x Dual EIA-232 Drivers/Receivers, MAX232 datasheet, Texas Instruments, Nov 2014.

[11] 28/40/44-Pin, High-Performance, Enhanced Flash, USB Microcontrollers with nanoWatt Technology, PIC18F2455/2550/4455/4550 datasheet, Microchip Technology Inc., 2006.

[12] Microchip Libraries for Applications, Microchip Technology Inc., 2018. [Online]. Available: https://www.microchip.com/mplab/microchip-libraries-for-applications

6 APPENDICES

A MPLAB Code, Project1

```
// PIC18F26K22 Configuration Bit Settings
// CONFIG1H
#pragma config FOSC = HSHP      // Oscillator Selection bits (HS oscillator (high
power > 16 MHz))
#pragma config PLLCFG = OFF     // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON    // Primary clock enable bit (Primary clock enabled)
#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 PWRTEN = OFF     // Power-up Timer Enable bit (Power up timer
disabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset
enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 190       // Brown Out Reset Voltage bits (VBOR set to 1.90 V
nominal)

// CONFIG2H
#pragma config WDTEN = OFF      // Watchdog Timer Enable bits (Watch dog timer is
always disabled. SWDTEN has no effect.)
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC1  // CCP2 MUX bit (CCP2 input/output is multiplexed
with RC1)
#pragma config PBADEN = ON      // PORTB A/D Enable bit (PORTB<5:0> pins are
configured as analog input channels on Reset)
#pragma config CCP3MX = PORTB5  // P3A/CCP3 Mux bit (P3A/CCP3 input/output is
multiplexed with RB5)
#pragma config HFOFST = ON      // HFINTOSC Fast Start-up (HFINTOSC output and
ready status are not delayed by the oscillator stable status)
#pragma config T3CMX = PORTC0   // Timer3 Clock input mux bit (T3CKI is on RC0)
#pragma config P2BMX = PORTB5   // ECCP2 B output mux bit (P2B is on RB5)
#pragma config MCLRE = EXTMCLR  // 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-
003FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection Block 1 (Block 1 (004000-
007FFFh) not code-protected)
#pragma config CP2 = OFF        // Code Protection Block 2 (Block 2 (008000-
00BFFFh) not code-protected)
#pragma config CP3 = OFF        // Code Protection Block 3 (Block 3 (00C000-
00FFFFh) 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-
003FFFh) not write-protected)
```

```
#pragma config WRT1 = OFF       // Write Protection Block 1 (Block 1 (004000-
007FFFh) not write-protected)
#pragma config WRT2 = OFF       // Write Protection Block 2 (Block 2 (008000-
00BFFFh) not write-protected)
#pragma config WRT3 = OFF       // Write Protection Block 3 (Block 3 (00C000-
00FFFFh) 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
#pragma config EBTR0 = OFF      // Table Read Protection Block 0 (Block 0 (000800-
003FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Table Read Protection Block 1 (Block 1 (004000-
007FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR2 = OFF      // Table Read Protection Block 2 (Block 2 (008000-
00BFFFh) not protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF      // Table Read Protection Block 3 (Block 3 (00C000-
00FFFFh) not protected from table reads executed in other blocks)

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

#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <math.h>
#include <string.h>

#define _XTAL_FREQ 64000000L
#define LED1 LATCbits.LATC0
#define LED2 LATCbits.LATC5
#define ADDRESS 0x50

//uart_puts
//Takes a string and sends each character to be sent to USART0
void uart_puts(char *str) {
    int i = 0;
    // Loop through string, sending each character
    while(str[i] != '\0')
    {
        while (TXSTA1bits.TRMT == 0);    // wait the end of transmission
        TXREG1 = str[i];                 // start sending the new byte
        i++;
    }
}

//convert_temp_LMT87Q1
float convert_temp_LMT87Q1(int v){
#define RATIO 4.8828125
#define X 13.582
#define Y 0.00433
#define Z 2230.8
    float t1 = (float)sqrt((float)pow(X, 2)+(4*Y*(Z-(float)(RATIO*v))));
    return 30-(X-t1)/(2*Y);
}

//pwm_setup
void pwm_setup(int duty_cyc, int pwm_ch){
    //Setup for PWM Operation
```

```
    switch(pwm_ch){
        case 1:{                        //CCP1 active
            TRISCbits.RC2 = 0;          //RC2 is output
            PR2 = 99;                   //Period of PWM Signal
            CCP1CONbits.CCP1M = 0b1100; //PWM mode
            CCPTMRS0bits.C1TSEL = 0b00; //PWM modes use Timer2
            CCP1CONbits.DC1B = 0b00;    //the two LSbs of the PWM duty cycle
            if(duty_cyc >= 0 && duty_cyc <= 100)
                CCPR1L = (unsigned short)duty_cyc;
            else if(duty_cyc > 100 )CCPR1L = 100;
            else CCPR1L = 0;

            break;
        }
        case 2:{                        //CCP2 active
            TRISCbits.RC1 = 0;          //RC1 is output
            PR2 = 99;                   //Period of PWM Signal
            CCP2CONbits.CCP2M = 0b1100; //PWM mode
            CCPTMRS0bits.C1TSEL = 0b00; //PWM modes use Timer2
            CCP2CONbits.DC2B = 0b00;    //the two LSbs of the PWM duty cycle
            if(duty_cyc >= 0 && duty_cyc <= 100)
                CCPR2L = (unsigned short)duty_cyc;
            else if(duty_cyc > 100 )CCPR2L = 100;
            else CCPR2L = 0;

            break;
        }
        default:;
    }
}

//i2c_master_init
void i2c_master_init(const unsigned long c){
    SSP1CON1 = 0b00101000;          //SSP enable, CLK=FOSC/(4*(SSPADD+1))
    SSP1CON2 = 0;                   //0 initially
    SSP1ADD = (_XTAL_FREQ/(4*c))-1; //Setting Clock Speed
    SSP1STAT = 0;                   //0 initially
    TRISCbits.RC3 = 1;              //Setting as input as given in datasheet
    TRISCbits.RC4 = 1;              //Setting as input as given in datasheet
}

//i2c_master_wait
void i2c_master_wait(){
    while ((SSP1STAT & 0b00000100) || (SSP1CON2 & 0x1F)); //Transmit is in progress
}

//i2c_master_start
void i2c_master_start(){
    i2c_master_wait();
    SSP1CON2bits.SEN = 1;       //Initiate start condition
}

//i2c_master_repeatedstart
void i2c_master_repeatedstart(){
    i2c_master_wait();
    SSP1CON2bits.RSEN = 1;      //Initiate repeated start condition
}

//i2c_master_stop
void i2c_master_stop(){
    i2c_master_wait();
    SSP1CON2bits.PEN = 1;       //Initiate stop condition
}

//i2c_master_write
void i2c_master_write(unsigned d){
```

```
    i2c_master_wait();
    SSP1BUF = d;                //Write data to SSPBUF
}

//i2c_master_read
unsigned short i2c_master_read(unsigned short a){
    unsigned short temp;
    i2c_master_wait();
    SSP1CON2bits.RCEN = 1;
    i2c_master_wait();
    temp = SSP1BUF;             //Read data from SSPBUF
    i2c_master_wait();
    SSP1CON2bits.ACKDT =(a)?0:1;//Acknowledge bit
    SSP1CON2bits.ACKEN = 1;     //Acknowledge sequence
    return temp;
}

int main(){

    //Oscillator Control
    OSCCONbits.SCS = 0b00;      //Primary clock

    //Timer Control
    T2CONbits.T2CKPS = 0b01;    //Prescaler is 4
    T2CONbits.TMR2ON = 1;       //Timer2 is on

    //LED Control
    TRISC0 = 0;                 //RC0 is output
    TRISC5 = 0;                 //RC5 is output
    LED1 = 1;                   //LED1 on
    for(int i=0;i<10;i++){      //LED1 Blinking
        LATC0 = !LATC0;
        __delay_ms(20);
    }
    LED1 = 0;                   //LED1 off
    LED2 = 0;                   //LED2 off


    //Setup UART
    TRISCbits.TRISC6 = 0;       //TX is output
    TRISCbits.TRISC7 = 1;       //RX is input

    TXSTA1bits.SYNC = 0;        //Asynchronous mode
    TXSTA1bits.TX9 = 0;         //Selects 8-bit transmission
    TXSTA1bits.TXEN = 1;        //Transmit enabled

    RCSTA1bits.RX9 = 0;         //Selects 8-bit reception
    RCSTA1bits.CREN = 1;        //Enables receiver
    RCSTA1bits.SPEN = 1;        //Serial port enabled

    PIE1bits.RC1IE = 1;         //Enables the EUSART1 receive interrupt

    //setting for 19200 BPS
    SPBRG1 = 325;               //SPBRGx value for SYNC = 0, BRGH = 1, BRG16 = 0
    BAUDCON1bits.BRG16 = 0;     //8-bit Baud Rate Generator is used, SPBRG1
    TXSTA1bits.BRGH = 1;        //High speed
    BAUDCON1bits.CKTXP = 0;     //Idle state for transmit is low

    //Relay
    TRISBbits.TRISB1 = 0;       //RB1 is output
    PORTBbits.RB1 = 0;          //Turns off Relay

    //A/D Conversion Procedure
    TRISAbits.RA0 = 1;          //RA0 is input
    TRISAbits.RA1 = 1;          //RA1 is input
```

```
    ADCON2bits.ADCS = 0b000;    //Clock Select Bits, FOSC/2
    ADCON1bits.PVCFG = 0b00;    //A/D VREF+ connected to internal signal, AVDD
    ADCON2bits.ACQT = 0b010;    //Acquisition time select bits, 010
    ADCON2bits.ADFM = 1;        //Right justified

    //Temperature Detection
    ADCON0bits.ADON = 1;        //ADC is enabled
    ADCON0bits.CHS = 0b00000;   //Analog Channel Select bits, AN0
    ADCON0bits.GO_NOT_DONE = 1; //A/D conversion cycle in progress. Setting this
bit starts an A/D conversion cycle

    while (ADCON0bits.GO_NOT_DONE);

    float v1 = ADRESH * 256 + ADRESL;
    char str1[50];
    sprintf(str1, "T1 = %.1f \t\t", convert_temp_LMT87Q1(v1));
    uart_puts(str1);

    ADCON0bits.ADON = 1;        //ADC is enabled
    ADCON0bits.CHS = 0b00001;   //Analog Channel Select bits, AN1
    ADCON0bits.GO_NOT_DONE = 1; //A/D conversion cycle in progress. Setting this
bit starts an A/D conversion cycle

    while (ADCON0bits.GO_NOT_DONE);

    float v2 = ADRESH * 256 + ADRESL;
    char str2[50];
    sprintf(str2, "T2 = %.1f\n", convert_temp_LMT87Q1(v2));
    uart_puts(str2);

    //I2C
    while(1){

    INTCONbits.PEIE = 1;
    INTCONbits.GIE  = 1;
    i2c_master_init(100000);
    LED2 = 1;                   //LED2 on
    __delay_ms(300);
    i2c_master_start();
    SSP1IF = 0;
    i2c_master_write(ADDRESS << 1);
    i2c_master_write(0x00);

    i2c_master_repeatedstart();
    i2c_master_write((ADDRESS << 1) + 1);
    __delay_us(2);
    pwm_setup(i2c_master_read(1),1);
    SSP1IF = 0;
    i2c_master_read(0);
    i2c_master_stop();

    for(int i=0;i<5;i++){       //LED2 Blinking
        LATC5 = !LATC5;
        __delay_ms(20);}
    LED2 = 0;                   //LED2 off
    }
    return 0;
}
```

B MPLAB Code, Project2

```
// PIC18F4550 Configuration Bit Settings
// CONFIG1L
#pragma config PLLDIV = 5       // PLL Prescaler Selection bits (Divide by 5 (20
MHz oscillator input))
#pragma config CPUDIV = OSC1_PLL2// System Clock Postscaler Selection bits
([Primary Oscillator Src: /1][96 MHz PLL Src: /2])
#pragma config USBDIV = 2       // USB Clock Selection bit (used in Full-Speed USB
mode only; UCFG:FSEN = 1) (USB clock source comes from the 96 MHz PLL divided by 2)

// CONFIG1H
#pragma config FOSC = HSPLL_HS  // Oscillator Selection bits (HS oscillator, PLL
enabled (HSPLL))
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe
Clock Monitor enabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit
(Oscillator Switchover mode enabled)

// CONFIG2L
#pragma config PWRT = OFF      // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOR = ON        // Brown-out Reset Enable bits (Brown-out Reset
enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 3        // Brown-out Reset Voltage bits (Minimum setting
2.05V)
#pragma config VREGEN = ON     // USB Voltage Regulator Enable bit (USB voltage
regulator disabled)

// CONFIG2H
#pragma config WDT = OFF        // Watchdog Timer Enable bit (WDT disabled (control
is placed on the SWDTEN bit))
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:128)

// CONFIG3H
#pragma config PBADEN = OFF     // PORTB A/D Enable bit (PORTB<4:0> pins are
configured as analog input channels on Reset)
#pragma config LPT1OSC = OFF    // Low-Power Timer 1 Oscillator Enable bit (Timer1
configured for low-power operation)
#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 bit (Block 0 (000800-001FFFh) is
not code-protected)
#pragma config CP1 = OFF        // Code Protection bit (Block 1 (002000-003FFFh) is
not code-protected)

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

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

// CONFIG6H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit
(Configuration registers (300000-3000FFh) are not write-protected)
```

```
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot block
(000000-0007FFh) is not write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF      // Table Read Protection bit (Block 0 (000800-
001FFFh) is not protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Table Read Protection bit (Block 1 (002000-
003FFFh) is not protected from table reads executed in other blocks)

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


#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <math.h>
#include <string.h>

#define _XTAL_FREQ 48000000L

#define LED1 LATBbits.LATB0
#define LED2 LATBbits.LATB1
#define LED3 LATBbits.LATB2

//uart_puts
//Takes a string and sends each character to be sent to USART0
void uart_puts(char *str) {
    int i = 0;
    // Loop through string, sending each character
    while(str[i] != '\0')
    {
        while (TXSTAbits.TRMT == 0);    // wait the end of transmission
        TXREG = str[i];                 // start sending the new byte
        i++;
    }
}

int main(){

    //Oscillator Control
    OSCCONbits.SCS = 0b00;      //Primary clock

    //Timer Control
    T2CONbits.T2CKPS = 0b01;    //Prescaler is 4
    T2CONbits.TMR2ON = 1;       //Timer2 is on

    //LED Control
    TRISBbits.RB0 = 0;          //RB0 is output
    TRISBbits.RB1 = 0;          //RB1 is output
    TRISBbits.RB2 = 0;          //RB2 is output
    LED1 = 0;                   //LED1 on
    LED2 = 0;                   //LED2 on
    LED3 = 1;                   //LED3 off
    for(int i=0;i<10;i++){      //LED1 Blinking
        LATB0 = !LATB0;
        __delay_ms(20);
    }
    LED1 = 1;                   //LED1 off
    LED2 = 1;                   //LED2 off


    //Setup UART
    TRISCbits.TRISC6 = 0;       //TX is output
    TRISCbits.TRISC7 = 1;       //RX is input
```

```
    TXSTAbits.SYNC = 0;         //Asynchronous mode
    TXSTAbits.TX9 = 0;          //Selects 8-bit transmission
    TXSTAbits.TXEN = 1;         //Transmit enabled

    RCSTAbits.RX9 = 0;          //Selects 8-bit reception
    RCSTAbits.CREN = 1;         //Enables receiver
    RCSTAbits.SPEN = 1;         //Serial port enabled

    //setting for 19200 BPS
    SPBRGH = 2;                 //SPBRGH value for SYNC = 0, BRGH = 1, BRG16 = 1
    SPBRG = 103;                //SPBRG value for SYNC = 0, BRGH = 1, BRG16 = 1
    BAUDCONbits.BRG16 = 1;      //8-bit Baud Rate Generator is used, SPBRG
    TXSTAbits.BRGH = 1;         //High speed
    BAUDCONbits.RXDTP = 1;      //RX data is inverted

    TRISBbits.RB3 = 1;          //RB3 is input
    LED3 = 0;                   //LED3 on

    //A/D Conversion Procedure
    TRISBbits.RB3 = 1;          //RB3 is input

    ADCON2bits.ADCS = 0b000;    //Clock Select Bits, FOSC/2
    ADCON1bits.VCFG = 0b00;     //A/D VREF+ connected to internal signal, AVDD
    ADCON2bits.ACQT = 0b010;    //Acquisition time select bits, 010
    ADCON2bits.ADFM = 1;        //Right justified

    while(1){
    ADCON0bits.ADON = 1;        //ADC is enabled
    ADCON0bits.CHS = 0b1001;    //Analog Channel Select bits, AN9
    ADCON0bits.GO_NOT_DONE = 1; //A/D conversion cycle in progress. Setting this
bit starts an A/D conversion cycle

    float v = ADRESH * 256 + ADRESL;
    v = 0.0048828125 * v;
    while (ADCON0bits.GO_NOT_DONE);
    char str[50];

    if(PORTBbits.RB3 == 1){
        sprintf(str, "RB3 is high and V = %.1f Volts\n", v);
        uart_puts(str);
    }
    else{
        sprintf(str, "RB3 is low and V = %.1f Volts\n", v);
        uart_puts(str);
    }
    }


    return 0;

}
```

C Microsoft Visual Studio Code and USB HID, Project2

The document can be found in the following drive link.

url: https://drive.google.com/open?id=1323saIjLyOm9PTCCGmQXm9lOfHIOCOAA