Wireless Digital Voltmeter

A Major Qualifying Project Report

Submitted to the faculty

Of the

Worcester Polytechnic Institute

Worcester, MA

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

On this day of

April, 2010

By

Mairaj Aftab Malik

Electrical and Computer Engineering '11

Omar Kiyani

Electrical and Computer Engineering '11

Project Advisor: Dr. John Mcneill

Electrical and Computer Engineering

TABLE OF CONTENTS

Index of Figures

Index of Tables

Abstract

The goal of this MQP was to design and implement a hand-held DC Digital Voltmeter (DVM) with wireless capabilities. Some of the specifications we sought to achieve in our final device were 5-digit voltage readings, 0.1% accuracy and approximately 100m wireless range. By implementing this DVM, we are showcasing the capabilities of some of the Analog Devices Inc integrated circuits namely the AD7799 and ADUC845. The AD7799, consisting of a 24 bit ∑-Δ ADC with an on chip IN-AMP and 3- differential inputs, serves as a perfect voltmeter front-end in our design, the ADUC 845, a Micro-Controller, serves as the brain for the system. Whereas, a windows application designed in C# serves as the visual user interface for the system.

Executive Summary

Recording data from Digital Voltmeters available in market today can be exasperating at times. Voltmeters that are not accompanied with GPIB or USB ports cannot be connected to external devices to log data. Hence, the data has to be recorded manually. Even the expensive ones with USB or GPIB connections have limitations because of the length of cables, their use of wall power and driver issues.

Moreover, some applications involve the use of digital voltmeters (DVM's) to measure voltage in hazardous conditions where it is not practical or possible to take the voltage readings in person.

Keeping this problem statement in mind, we designed and implemented a DC Wireless *Digital Voltmeter* that would solve the aforementioned problems by making use of Bluetooth Technology to transmit data wirelessly to an external Device where it can be logged.

In doing so, we also highlighted the features of some of the Integrated Circuits (IC's) made by Analog Devices Inc (ADI), who serve as the sponsors for this project.

In the initial stages of the design, it was decided to build a palm sized DC Voltmeter with 5-digit voltage readings and 0.1% accuracy, that would be able to interact wirelessly with external devices including Laptops, PDA's at a distance of around 100m. It was also decided that these external devices would provide the visual user interface with our DVM.

To achieve this functionality, the design of the wireless voltmeter was broken into four major blocks namely;

- *Data Acquisition (Voltmeter Front-End).*
- *Wireless communication block.*
- *System Processing (Micro-controller)*
- *System Power Block.*

Given the sheer range of options available for every block, thorough background research was conducted to determine the most cost-effective yet efficient solution in terms of power consumption, board space, data accuracy and ease of use.

The research revealed an Integrated circuit AD7799 produced by Analog Devices, as the most suitable contender for the Data Acquisition block based on the fact that it brought together several different components including a 24 bit $\sum$-Δ ADC, an Instrumentation Amplifier (IN-AMP), a 6-2 multiplexer and SPI interface in one tiny TSSOP-16 package. Therefore this chip alone could act as the voltmeter front-end for our DVM.

For the communication block, it was determined that Bluetooth communication would suffice for our requirements and Wi-Fi would be an overkill which would increase our power consumption by a factor of 10.

Knowing from experience we knew that a micro-controller would be necessary to act as the system brain to control other IC's including the Bluetooth transceiver and AD7799.

Lastly, the system power block was figured out based on the power requirements of the other two blocks. It was decided that our DVM would have a rechargeable battery with about 10 - 12 hour battery life. The battery would be charged using USB to make our DVM more versatile and easy to use.

At this point we were in a position to make design choices more specifically. A micro-Controller and a Bluetooth transceiver was decided upon and development boards for these components and AD7799 were ordered. These development boards were then all hooked up together to test the functionality of the entire system.

For the user interface for our DVM a windows based application was made in Visual Studio 2008 using the object oriented programming language C#. The Application displayed the voltage being measured by our DVM and it also enabled the user to control the DVM features including changing ranges, initiating calibration procedure and switching it ON/OFF.

Building upon our success with the development boards a Printed circuit board (PCB) was designed. The PCB included all the subsystems including battery monitoring, USB battery charging, external multiplexing for calibration etc. In spite of containing all the subsystems the PCB was well within our initial design requirements and measured only 3.5* 1.8inches (8.9*4.6cm).

Intensive testing was now performed on the board to verify its functionality. The testing showed that the system was functional and worked seamlessly.

Acknowledgements

This Project would not have been possible without the support and guidance of many.

Firstly, we would like to thank the New England Center for Analog and Mixed Signal Integrated Circuit Design (NECAMSID) for sponsoring this project, and specifically Analog Devices in Wilmington -MA for providing the necessary financial support to carry out this project.

Secondly, we would like to thank Mike Coln and Paul Blanchard from Analog Devices for their constant support throughout the Project.

Thirdly, we would like to thank Professor Gene Bogdanov for his guidance at times when it was most needed.

Finally, we would like to extend our deepest gratitude to our project advisor, Professor John McNeill for his constant support and guidance. He set a very high standard of excellence for us right from the start, and having him as our mentor motivated us to work even harder and try to achieve the highest quality of work possible.

Introduction

This project explores the design and development of a Wireless Digital Voltmeter (DVM). It encompasses a wide variety of technology areas ranging from the IC board design to wireless communications. The three major technologies that were employed in the development of the DVM are

(i) Integrated Circuits (IC's) for the Voltmeter, Data Type Conversions, Battery Management etc.

(ii) Wireless Communication using Bluetooth Technology.

(iii) C# object-oriented programming language to design real time windows based application to serve as the user interface to the DVM.

In the initial stages of the project, it was decided to build a palm sized Voltmeter with 5-digit voltage readings and 0.1% accuracy, that would be able to interact wirelessly with external devices including Laptops, PDA's at a distance of around 100m. In an attempt to save board space, it was also decided that the DVM itself won't have a display; instead external devices would provide the visual user interface with our DVM.

After thorough research an Integrated circuit AD7799 produced by Analog Devices, was chosen to be provide the Voltmeter Front End, based on the fact that it brought together several different components including a 24 bit ∑-Δ ADC, an Instrumentation Amplifier (IN-AMP), a 6-2 multiplexer and SPI interface in one tiny TSSOP-16 package.

A Bluetooth transceiver was found to be the most suitable communication choice for our system as most modern devices incorporate Bluetooth communication protocol therefore it enables our DVM to be compatible with greater number of external products.

A microcontroller was found to be a necessary part of the system to configure the AD7799 and also to act as a data converter between the SPI interface of the AD7799 and UART interface of the Bluetooth transceiver.

After the data had been transmitted wirelessly by the Bluetooth chip, it could be acquired by any device that incorporated a Bluetooth transceiver. In our case it was a PC. A Windows based application that served as the user interface was made in Visual Studio 2008 using C# to demonstrate the two-way link between the DVM and the PC.

This report discusses the details of this project through all the major steps: Conception, Project definition, design, implementation and evaluation.

1. Background

Our project began with the aim of developing an Integrated Circuit solution to solve the problem of getting voltage readings from a voltmeter into an external device where they can be continuously monitored and logged. After the initial couple of meetings we had decided that instead of modifying an existing Voltmeter; we will design and implement our own device which will have the ability to wirelessly would transmit data to an external device where it can be logged.

This chapter provides a basic overview of how we set certain project goals and what background research we had to perform in order to finalize our product design.

1.1. Defining Project Goals

The goals given below were set at the start of the project and later were quintessential in molding the design of our product.

1. To develop a fully functional Digital Voltmeter (DVM).
2. To have two way wireless communication capability in the DVM.
3. To build a software application for an external device (such as a PC), that would serve as the visual user interface.
4. Having data logging and calibration capability in the software application that was built.
5. To keep the size of this voltmeter as small as possible.

Following the goals and initial requirements, we started doing our background research on all the different parts of the project.

1.2. Background Research

Based on our project goals, we divided our product into different modules:

1.) Data Acquisition (Voltmeter Front-End)

2.) Wireless Communication

3.) Microcontroller

4.) System power (battery)

Therefore, a thorough background research was conducted in all the above areas, which is presented in the following pages of this chapter.

1.2.1. Digital Voltmeters (DVM’s)

Digital voltmeters are instruments that measure voltage or voltage drop in a circuit. They use solid-state components and display values digitally. Typically, digital voltmeters are used to locate excessive resistance that may indicate an open circuit or ground. They are also used to identify low voltage or voltage drops that may indicate a poor connection.

Digital voltmeters are connected in parallel with the circuit being tested so that the meter can tap a small amount of current. The positive lead is connected to the circuits positive side and the negative lead is connected to the circuits ground. The digital voltmeters internal resistance is the input impedance, which is usually expressed in ohms per volt. This amount is relatively high in order to prevent the device from drawing significant current and disturbing the operation of the circuit being tested. The sensitivity of the current meter and the value of the series resistance determine the range of voltages that digital voltmeters can measure.

Digital voltmeters are usually designed around a special type of analog-to-digital converter called an integrating converter. Voltmeter accuracy is affected by many factors, including temperature and supply voltage variations. They can measure a range of alternating current (AC) voltages, direct current (DC) voltages, or both AC and DC voltages. Devices typically display between three and seven digits. Some digital voltmeters can also capture minimum and maximum voltages called spike readings.

DVM’s in Market

There are quite a few types of digital voltmeters present in the market today. We took a look at several examples, compared their specifications and based on that figured out the requirements for our project. Voltmeters present in market can be divided into two broad categories:

(i) Ready to use DVM’s

(ii) Assembled DVM’s

These can be further sub-categorized according to their other properties. We started by looking at the ready to use DVM’s:

(i) Ready to use

These DVM’s do not require any circuit building from the user’s part and they come inside their own package. These can be further divided into two categories:

- Handheld
- Bench

Handheld

These are the voltmeters that the user can hold in his/her hand while taking measurements. Their advantages include small size and portability. We will take a look at one example for such kind of a voltmeter:

- **DATEL - DMS-40PC Series:**

  DMS-40PC Series, 4½ Digit, LED Display, Miniature DPM's are fully self-contained, component-like, plug-in meters that provide scientific-grade accuracy (typically ±2 counts or

±0.005% of full scale) and outstanding reliability at a very affordable price. Within its miniature (2.17" x 0.92" x 0.56"), epoxy-encapsulated package, each meter contains a precision reference circuit; a high-resolution, auto-zeroing, factory calibrated A/D converter; and a large (0.52"/13.2mm), easy-to-read, LED display. The basic specifications for this voltmeter can be found in Table1-1. Figure 1-1 shows a DATEL- DMS-40PC.

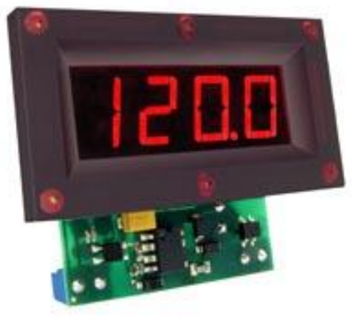


Fig.1-1: (DATEL - DMS-40PC DVM)[1]

Bench

These are the voltmeters mostly used in labs. They operate on the voltages available in regular power sockets (100 – 120V). Following are examples of some of the bench DVM’s that are available in market today:

- **Tektronix CDM250 Digital Multimeter:**

The CDM250 Digital Multimeter measures and displays voltage, current and resistance using a 3 1/2 digit LED display. Sine wave alternating voltages and currents are displayed in RMS values. The CDM250 is safe and easy to use. This meter has overload protection and is UL listed and CSA certified. Its specifications can be found in Table1-1. Figure 1-2 shows a

[1] www.datelmeters.com

Tektronix CDM250 Digital Multimeter.

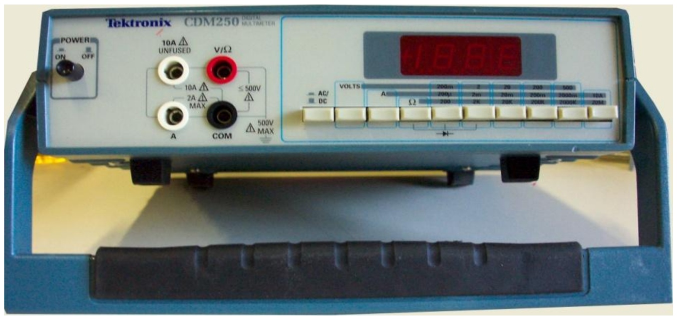


Fig.1-2: (Tektronix CDM250 DVM)[2]

- **Fluke 8845A/8846A Digital Multimeters:**

The Fluke 8845A, 6.5 digit precision multimeters, have the precision and versatility to handle the most demanding measurements, on the bench or in a system. These meters are both high performance and yet also remarkably easy to use. These digital multi-meters perform the functions you would expect to see in a multifunction DMM, including measuring volts, ohms, and amps. Some basic specifications can be found in Table1-1. Figure 1-3 shows a Fluke 8845A/8846A Digital Multimeter.

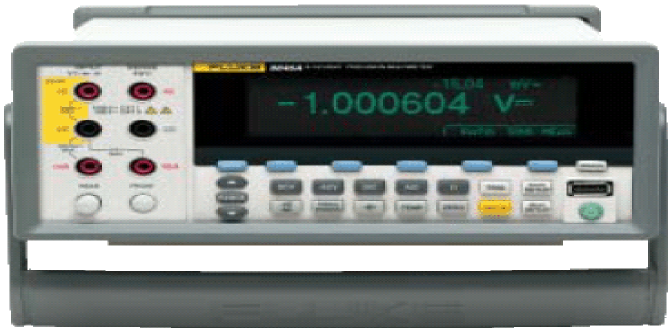


Fig.1-3: (Fluke 8845A/8846A DVM)[3]

[2] www.tek.com

[3] Ca.fluke.com

- **Agilent 34970A Data Acquisition/Switch Unit:**

The Agilent 34970A consists of a three-slot mainframe with a built-in 6 1/2 digit multimeter. Each channel can be configured independently to measure one of 11 different functions without the added cost or hassles of signal-conditioning accessories. We can choose from eight optional plug-in modules to create a compact data logger, full-featured data acquisition system or low-cost switching unit. On-module screw-terminal connections eliminate the need for terminal blocks and a unique relay maintenance feature counts every closure on every switch for easy, predictable relay maintenance. For specifications, refer to Table1-1. Figure 1-4 shows an Agilent 34970A Data Acquisition/Switch Unit.

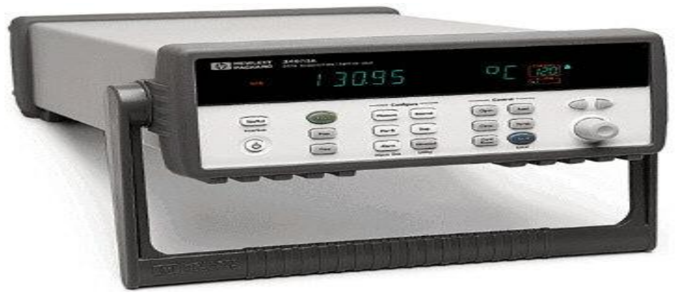


Fig.1-4: (Agilent 34970A DVM)[4]

(ii) Assembled

These kinds of DVM's use discrete components and require some sort of parts assembling by the user in order for them to able to use them. These can be further divided into two categories:

- Ones that do not use a DVM IC
- Ones that use the DVM IC

[4] Inotek.com

Non-DVM IC

These voltmeters usually use the analog-to-digital converter IC's made by companies like MAXIM, Intersil etc. Following are a couple of examples of such sort of DVM's:

- **ICL7107 - DIGITAL VOLTMETER (Intersil IC):**

  This is an easy to build, but very accurate and useful digital voltmeter. It has been designed as a panel meter and can be used in DC power supplies or anywhere else it is necessary to have an accurate indication of the voltage present. The circuit employs the ADC (Analogue to Digital Converter) I.C. CL7107 made by INTERSIL. This IC incorporates in a 40 pin case all the circuitry necessary to convert an analogue signal to digital and can drive a series of four seven segment LED displays directly. The components built into the IC are an analog to digital converter, a comparator, a clock, a decoder and a seven segment LED display driver. This DVM cab measure and display a DC voltage in the range of 0-1999 Volts. Other specifications for this voltmeter can be found in Table 1-1. Figure 1-5 shows an ICL7107 – Digital Voltmeter.

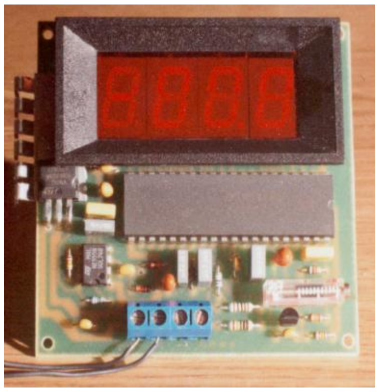


Fig.1-5: (ICL7107 DVM)[5]

[5] www.Qsl.net

DVM IC

Some IC chips are designed specifically so that may be used as digital voltmeters, after they are connected with certain external components. An example of such a chip is:

- **AD7799 – Digital Voltmeter Front-End Chip (by Analog Devices, Inc.)**

  The AD7799 can help us make a DVM of our own**.** It is a low power, low noise, complete analog front end for high precision measurement applications. The AD7799 contains a low noise, 24-bit Σ-Δ ADC with three differential analog inputs. The on-chip, low noise instrumentation amplifier means that signals of small amplitude can be interfaced directly to the ADC.

  On-chip features include a low-side power switch, reference detect, programmable digital output pins, burnout currents, and an internal clock oscillator. The output data rate from the part is software-programmable and can be varied from 4.17 Hz to 470 Hz.

  Technical Specifications:

Supply Voltage: .... 2.7V – 5.25V	Input Current: .... 380 µA
Temperature Range: ..... -40 °C to +105 °C	Update rate: .... 4.17 Hz to 470 Hz

  Applications:

-Weigh scales	-Portable instrumentation
-Pressure measurement	**-6-digit DVM**
-Strain gauge transducers	

  As can be seen above, the AD7799 has numerous applications. But the one in which we are interested in is the 6-digit DVM application (which is highlighted in the list above).

DVM Summary:

All the DVM categories that we discussed so far can be summed up in the following flow-chart:

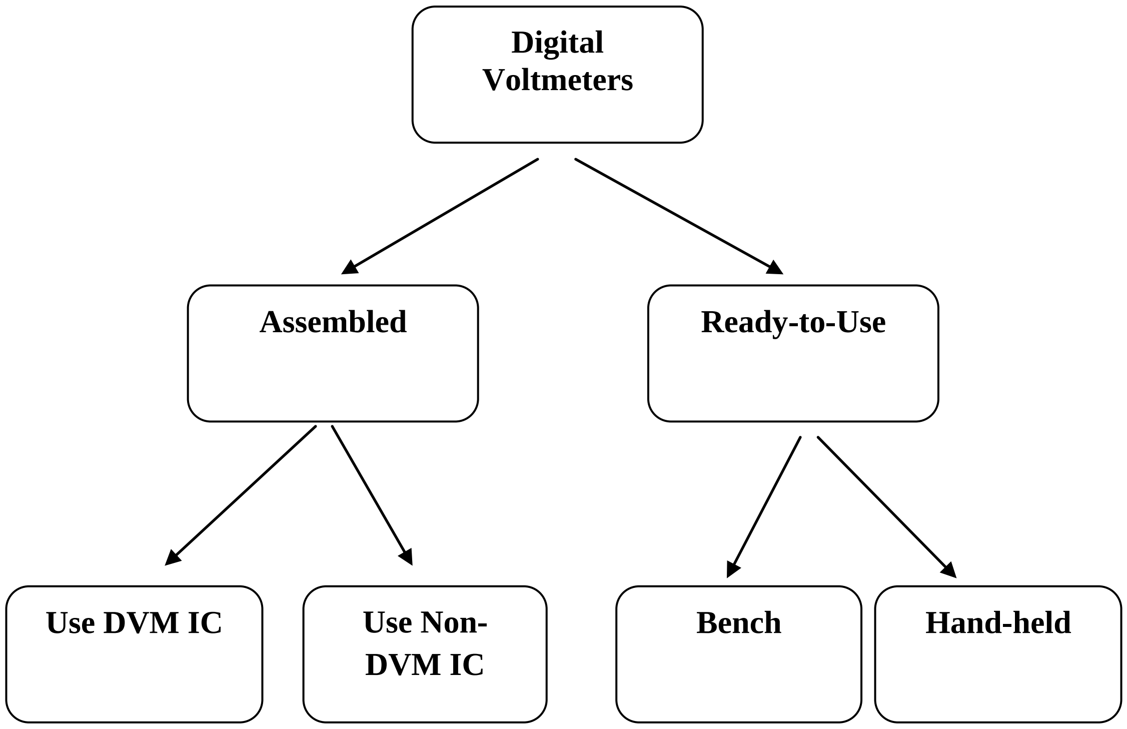


Fig.1-6: (DVM categories)

The specifications for all the DVM's that we looked at while conducting our market research can be found in the following table:

Property	Display digits (digits)	Supply Voltage (V)	Power Input (Watts)	Measuring Range (V)	Accuracy
ICL 7107 DVM	3.5	5	1	0 – 1,999	0.1%
Tektronix CDM250	3.5	110	220	0.2 – 500	0.5%
Datel DMS-40PC	4.5	5	2.5	0 – 200	0.1%
Fluke 8845A	6.5	120	12	0.1 – 1000	0.002%
Agilent 34970A	6.5	100/120/220	12	0.1– 300	0.004%

Table 1-1: (Comparison of DVM specifications)

Typical Circuitry of the existing DVM's

While carrying out the market research for DVM's, we also studied their functionalities and figured out how they operated. We shall take a look at the internal functional blocks of some of the voltmeters mentioned above, and see how they actually operate. It was very helpful to look at the schematics of the existing DVM's, as it helped us with our own DVM design.

- **ICL7107 - DIGITAL VOLTMETER (Intersil IC):**

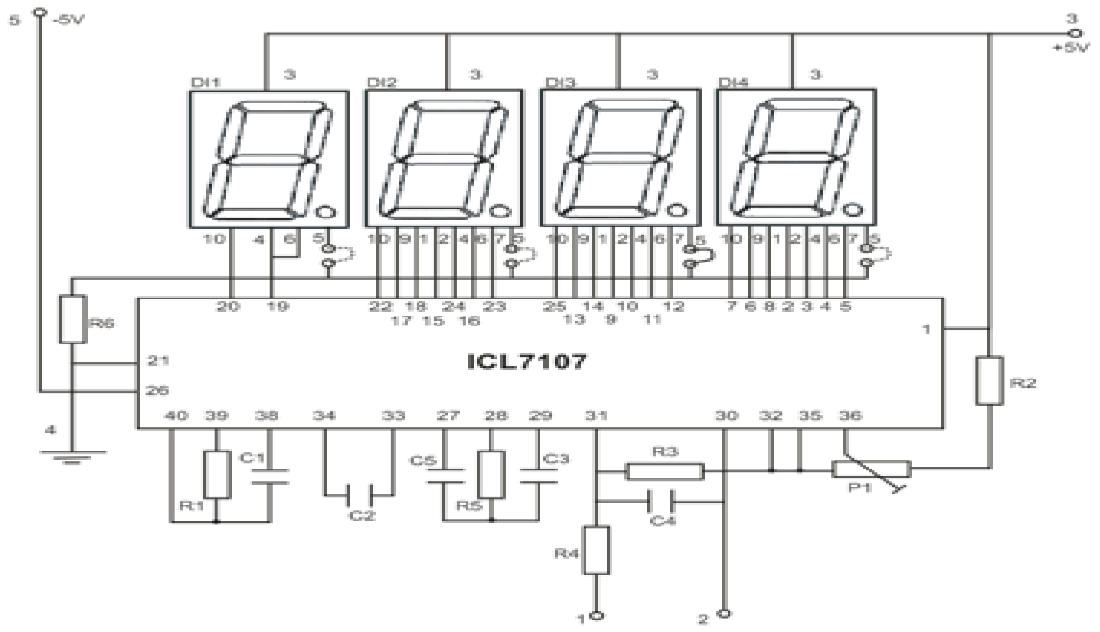


Fig.1-7: (ICL7107 DVM Schematic)[6]

An Analog to Digital Converter, (ADC) is better known as a dual slope converter or integrating converter. This type of converter is generally preferred over other types as it offers accuracy and simplicity in design and a relative indifference to noise which makes it very reliable. The operation of the circuit is better understood if it is described in two stages. During the first stage and for a given period the input voltage is integrated, and in the output of the integrator at the end of this period, there is a voltage which is directly proportional to the input voltage. At the

[6] www.qsl.net

end of the preset period the integrator is fed with an internal reference voltage and the output of the circuit is gradually reduced until it reaches the level of the zero reference voltage. This second phase is known as the negative slope period and its duration depends on the output of the integrator in the first period. As the duration of the first operation is fixed and the length of the second is variable it is possible to compare the two and this way the input voltage is in fact compared to the internal reference voltage and the result is coded and is sent to the display.

- **DATEL - DMS-40PC Series:**

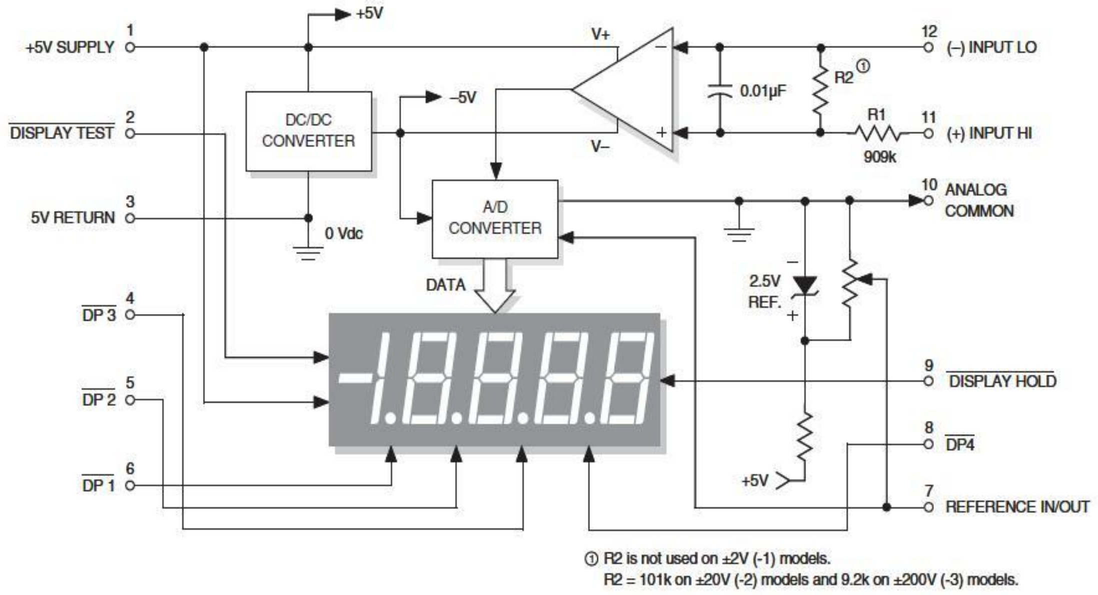


Fig.1-8: (DMS-40PC DVM Schematic)[7]

The DMS-40PC can measure single-ended signals with either positive or negative polarities. True single-ended inputs always have one of their two terminals at the same potential as the DMS-40PC's 5V RETURN (pin 3). Single-ended inputs are usually derived from power supplies that have a common ground with the meter's +5V supply. In that case, Pin 12, (–) INPUT LO, is

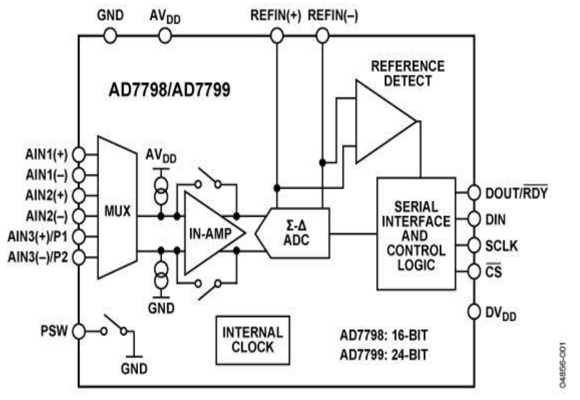
[7] www.datelmeters.com

shown directly tied to ground. This connection to ground must be a separate wire or pc-board trace originating at VIN's negative terminal. This hook-up will normally eliminate display instabilities and errors caused by ground-loop currents which can occur if (–) INPUT LO is tied to ground at, or near, pin 3.

Differential inputs can also be measured with DMS-40PC meters. Differential inputs must also originate from power supplies that have a common ground with the meter's 5V RETURN (pin 3). However, differential inputs usually have both terminals above and/or below 5V RETURN.

The AD7799 – A Complete Voltmeter Front-End

The AD7799 with an on chip instrumentation amplifier, a low noise 24-bit Σ-Δ ADC and three differential analog inputs has been designed as a complete data acquisition analog front end for high precision measurement applications. Figure 1.9, shows the functional block diagram of the AD7799 and also the dimensions of the TSSOP package it is contained in.



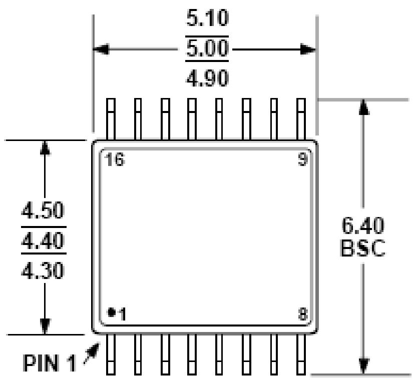
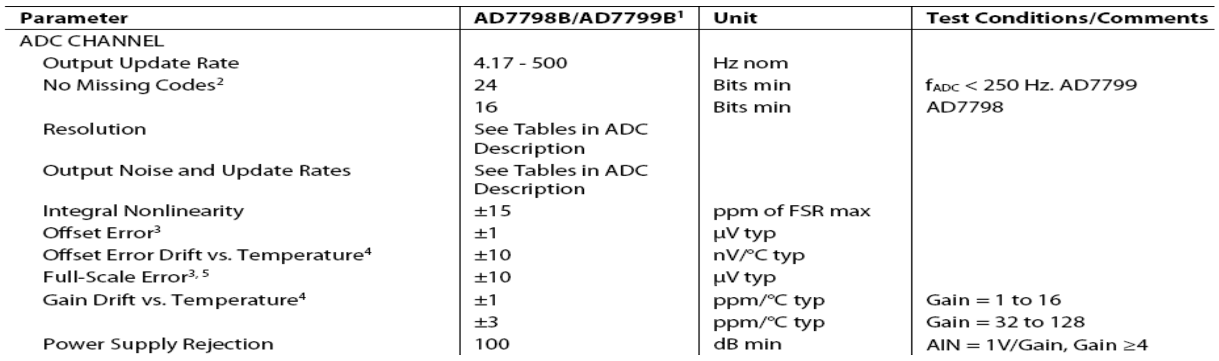


Fig.1-9: (Block Diagram for AD7799 (left) Tssop-16 package dimension in mm (right))[8]

Every function of the chip ranging from gain selection to refresh rate and channel selection is software programmable. The chip uses a SPI interface for this purpose which is also built into it. The SPI interface is used to configure the on chip registers which control the behavior of the chip and is also used to read the output from the ADC conversions.

[8] www.Analog.com

The ADC present on AD7799 is a highly accurate 24 bit Σ-Δ ADC.



Parameter	AD7798B/AD7799B[1]	Unit	Test Conditions/Comments
ADC CHANNEL			
Output Update Rate	4.17 - 500	Hz nom	
No Missing Codes[2]	24	Bits min	f_{ADC} < 250 Hz. AD7799
	16	Bits min	AD7798
Resolution	See Tables in ADC Description		
Output Noise and Update Rates	See Tables in ADC Description		
Integral Nonlinearity	±15	ppm of FSR max	
Offset Error[3]	±1	µV typ	
Offset Error Drift vs. Temperature[4]	±10	nV/°C typ	
Full-Scale Error[3, 5]	±10	µV typ	
Gain Drift vs. Temperature[4]	±1	ppm/°C typ	Gain = 1 to 16
	±3	ppm/°C typ	Gain = 32 to 128
Power Supply Rejection	100	dB min	AIN = 1V/Gain, Gain ≥4

Table 1-2: (ADC characteristics)[9]

The part operates with a power supply from 2.7V to 5.25V and consumes a current of 400uA typically.

All these features make the AD7799 an ideal system on-chip solution for our design.

[9] AD7799 datasheet

1.2.2. Wireless Communication

According to our initial design specifications, we were required to have a two way communication link between our DVM and the external device. A lot of the potential external devices were looked at Including Laptops, PDA's, PC's, iPhone/iTouch, etc. All of them had either a built a built in Wi-Fi module or a Bluetooth module or both. As was the case with IPODs which had a Broadcom BCM4325 chip incorporated in them. The Broadcom BCM4325 family of single-chip devices integrates IEEE 802.11a/b/g with a Bluetooth 2.1 + enhanced data rate (EDR) on a single chip. Figure 1-10 shows the internal circuitry of an iTouch containing the BCM 4325 chip.

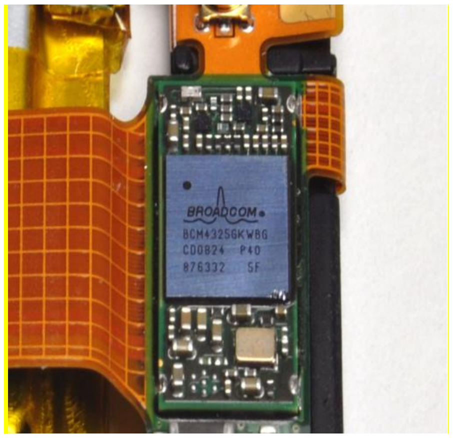


Fig.1-10: (BCM4325)[10]

Thus in order to make our product more adaptable with these external devices it was decided to limit our communication options to Bluetooth and IEEE 802.11 (Wi-Fi). Both of these communication methods were analyzed separately and keeping our system requirements in mind, we tried to determine which method would serve our needs better.

[10] www.ifixit.com

Bluetooth:

Bluetooth Wireless technology is a short range communications technology intended to replace the cables connecting portable or fixed devices. It transmits data via low-power radio waves. Bluetooth technology has the ability to handle both data and voice transmissions simultaneously. The key features of the *Bluetooth* technology are robustness, low power, and low cost.

Frequency Spectrum:

Bluetooth transmits data via low-power radio waves using a frequency of **2.45 GHz** (between 2.402 GHz and 2.480 GHz).This frequency band has been set aside by international agreement for the use of industrial, scientific and medical devices (ISM).

Interference:

A number of household devices, including baby monitors and cordless phones also take advantage of the same radio-frequency band as the Bluetooth. However the Bluetooth makes use of a technique called *Advanced Frequency Hopping* (AHF) that makes it highly unlikely for more than one device to be transmitting on the same frequency at the same time. In this technique, a Bluetooth device will use 79 individual randomly chosen frequencies within the band, ranging from 2.402 GHz to 2.480 GHz. In the case of Bluetooth transceiver, it changes frequencies 1,600 times every second. Since every Bluetooth transmitter is continuously hopping frequencies, it’s unlikely that two transceivers will be on the same frequency at the same time.

Range:

The operating range in the case of Bluetooth depends upon the Device class. The mostly commonly used class 2 transmitter has a range of 10m (32 Feet). The higher end class 1 transmitters can transmit up to 100m (300 Feet). The fundamental strength of Bluetooth technology is that unlike infrared it doesn't require line of sight between communicating devices.

Data Rate:

The version 2.1 + EDR which is present in the IPod touch can support a data transmission rate of up to 3 Mbps.

Power:

Bluetooth technology is designed to have very low power consumption. The most commonly used Class 2 Bluetooth transmitter uses around 35mW of power.

Cost:

A USB Bluetooth 2.1 EDR adapter is available in the market for $15.

Bluetooth options:

We had several Bluetooth options available to us if we decided to use Bluetooth transmission in our device. We could either use an off the shelf Bluetooth USB adapter as the one shown in Figure 1-11 or we could use a Bluetooth chip such as BCM2048 with an interface board as shown in Figure 1-12.

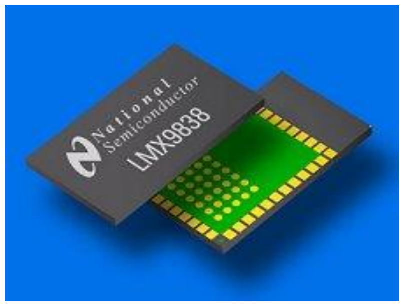
Fig.1-11: (Belkin Bluetooth Transmitter)[11]

Fig.1-12: (BCM2048)[12]

[11] Belkin.com
[12] Broadcom.com

Some other examples of Bluetooth chips that could be used in our design are LMX9838 (manufactured by National Semiconductor) and RN-41 (made by Roving Networks). These can be seen respectively in Figure1-13and Figure1-14:

<u>Fig.1-13:</u> RN-41 (Roving Networks)[13]



<u>Fig.1-14:</u> LMX9838 (National Semiconductor)[14]

<u>802.11 Network (Wi-Fi):</u>

A Wi-Fi is a wireless network based on the 802.11 networking protocol in which radio waves of high frequencies are used to send and receive data. The 802.11 networking protocol has 3 different standards namely 802.11a, 802.11b and 802.11g. These standards vary in terms of their transmitting frequencies and data transmission rates.

Frequency Spectrum:

Wi-Fi Radio transmitters use low power radio waves of frequency 2.4GHz or 5GHz to transmit data. The 2.4GHz frequency is used by the 802.11b/g network, whereas the 802.11a transmits data at 5 GHz.

[13] Rovingnetworks.com
[14] National.com

Interference:

Wi-Fi Transmissions can face interference from many household devices such as microwaves, cordless telephones and Bluetooth devices. Most of this interference is in the 2.4GHz band. Therefore the 802.11/a network which uses 5GHz frequency is immune to such interference.

Range:

A typical Wi-Fi transmitter using 802.11b or 802.11g with a stock antenna would have a range of 32m indoors and 90m outdoors. Whereas a Wi-Fi Transmitter (using 802.11a), which transmits at 5 GHz frequency band would have slightly lesser range. As with Bluetooth transmission, Wi-Fi does not require line of sight between the communicating devices.

Data Rate:

The 802.11a and 802.11g can handle data speeds of up to 54Mbps, whereas the 802.11b network which is the slowest and least expensive can move up to 11 Mbps.

Power:

A low power Wi-Fi transceiver uses approximately 290mW power while receiving and 425mW power while transmitting.

Cost:

An 802.11b/g Wi-Fi Transceiver having data transmission rate of 54 Mbps costs about 20$.

Options:

A few different options are available to us when it comes to using Wi-Fi Transceiver. We can use a Wi-Fi USB adapter, which is easily available in the market for around $20. Figure 1-15 is a picture of a Wi-Fi USB adapter and figure 1-16 shows its internal circuitry.

Fig.1-15: (Wi-Fi USB Adapter)[15]

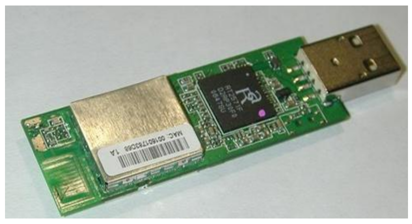


Fig.1-16: (Wi-Fi Adapter Circuitry)[16]

The Second option is to use a Wi-Fi chip such as the BCM 2049. In this case we would have to design and build the interface board for this chip. However, it would greatly reduce the power consumption and the overall size of the product. The Broadcom chip BCM 2049 can be seen in figure 1-17:

Fig.1-17: (BCM2049)[17]

[15] Belkin.com
[16] Broadcom.com
[17] www.broadcom.com

The comparison between Bluetooth and Wi-Fi is summarized in the following table:

Property	**Bluetooth**	**802.11 Wi-Fi**
Frequency Spectrum	2.45GHz(between 2.402GHz and 2.480GHz)	2.4 GHz or 5GHz
Interference	Household devices (baby monitors and cordless phones)	Household devices such as microwaves, cordless telephones and Bluetooth devices.
Range	Class 2 up to 10m (32 Feet). Class 1 up to 100m	32m indoors and 90m outdoors
Data Rate	3 Mbps	802.11a/g = 54 Mbps 802.11b = 11 Mbps
Power	35mW for receiving and transmitting	290mW while receiving. 425mW while transmitting
Cost	Appx. $12 for a USB adapter. Quantity does not affect price	Appx. $20 for a USB adapter. Quantity does not affect price.

Table 1-3: (Bluetooth Vs Wi-Fi)

1.2.3. Microcontroller

From previous experience we knew that a microcontroller would be essential to serve as the brain of our system by controlling all other IC’s. In our case, if we were to use the AD7799 chip a Microcontroller would be necessary to configure the on chip registers on the AD7799. Moreover it would also be required to convert the data coming out of the AD7799 which is following the SPI protocol to data which follows UART protocol as majority of the Bluetooth or Wi-Fi transceivers have only UART ports.

Therefore we decided that any microcontroller we considered must have the following features

1. A SPI port (to communicate with AD7799).
2. A UART port (to communicate with Bluetooth/Wi-Fi Transceiver).
3. A good number of general purpose Input output pins (to control any other IC’s).
4. Low power consumption (Portable device so limited battery capacity)
5. Small physical package (take up less board space).
6. Should be easily programmable.
7. Low cost

As Analog Devices Inc, serve as the sponsors of our project we decided to use a microcontroller made by them as by doing so we would be able to highlight the features of their microcontrollers.

A variety of microcontrollers produced by ADI were considered. However most of them did not have both SPI and UART interface thus were not suitable for our design.

One microcontroller which did have all the required features is discussed on the next page.

ADUC845: Precision Analog Microcontroller

The ADUC 845 had the following feature set

- Dual 24-Bit, Precision Sigma-Delta ADCs
- 10-Channel Single-Ended Input or 5-Channel Differential Input Options
- 8051 based core.
- Single 12-Bit Rail-to-Rail Voltage-Output DAC
- **34 GPIO Pins**
- **UART, I2C & SPI ports**
- **Ultra low power consumption.**
- **14*10mm MQFP-52 Package**
- **Easy Programming through Emulation Access pin.**

The features required for our design are shown in bold.

The Functional block diagram of the ADUC 845 is shown in the figure below.

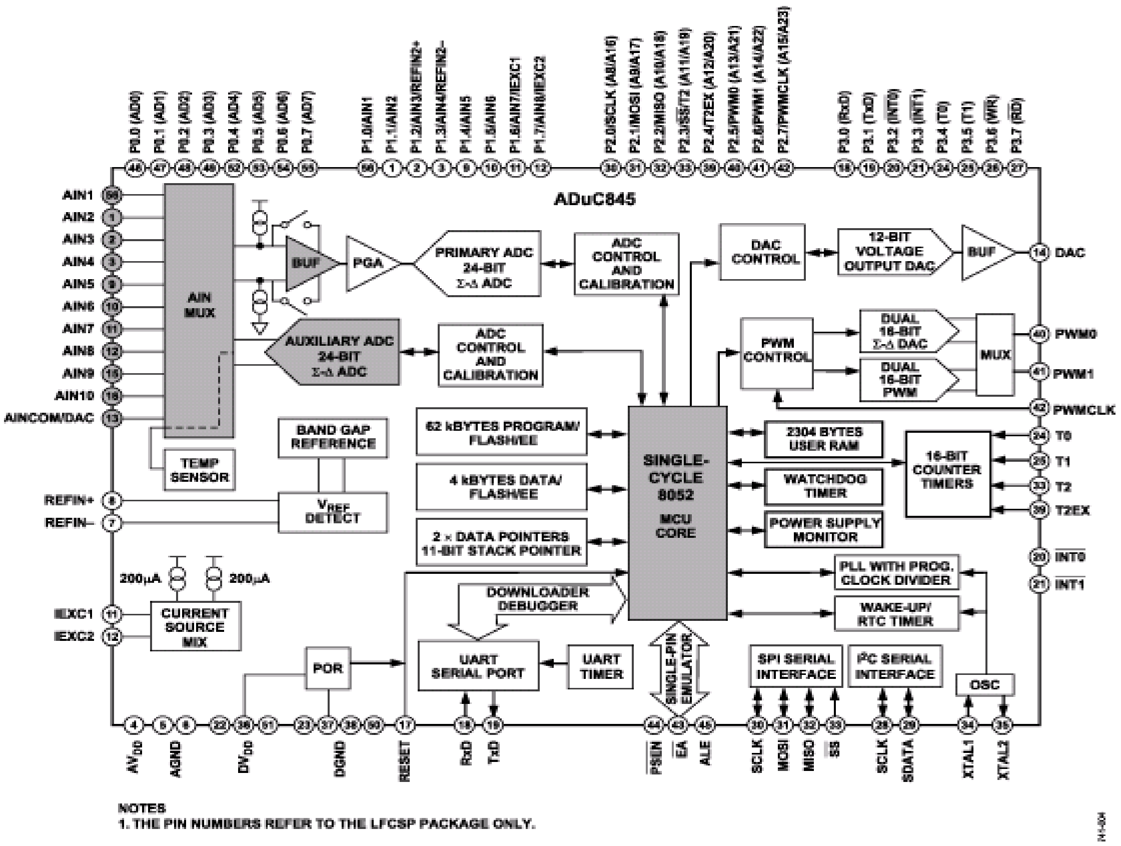

ADuC845
AIN MUX
BUF
PGA
PRIMARY ADC 24-BIT Σ-Δ ADC
ADC CONTROL AND CALIBRATION
AUXILIARY ADC 24-BIT Σ-Δ ADC
DAC CONTROL
12-BIT VOLTAGE OUTPUT DAC
PWM CONTROL
DUAL 16-BIT Σ-Δ DAC
DUAL 16-BIT PWM
MUX
TEMP SENSOR
BAND GAP REFERENCE
VREF DETECT
62 kBYTES PROGRAM/FLASH/EE
4 kBYTES DATA/FLASH/EE
2 × DATA POINTERS 11-BIT STACK POINTER
SINGLE-CYCLE 8052 MCU CORE
2304 BYTES USER RAM
WATCHDOG TIMER
POWER SUPPLY MONITOR
16-BIT COUNTER TIMERS
PLL WITH PROG. CLOCK DIVIDER
WAKE-UP/RTC TIMER
CURRENT SOURCE MIX
DOWNLOADER DEBUGGER
POR
UART SERIAL PORT
UART TIMER
SINGLE-PIN EMULATOR
SPI SERIAL INTERFACE
I²C SERIAL INTERFACE
OSC
AIN1 AIN2 AIN3 AIN4 AIN5 AIN6 AIN7 AIN8 AIN9 AIN10 AINCOM/DAC
REFIN+ REFIN−
IEXC1 IEXC2
200μA 200μA
DAC PWM0 PWM1 PWMCLK T0 T1 T2 T2EX INT0 INT1
AVDD AGND DVDD DGND RESET RxD TxD PSEN EA ALE SCLK MOSI MISO SS SCLK SDATA XTAL1 XTAL2
NOTES
1. THE PIN NUMBERS REFER TO THE LFCSP PACKAGE ONLY.

Fig.1-18: (Functional Block Diagram ADUC845)[18]

[18] ADUC845 datasheet

1.2.4. System Power (Battery)

A thorough market research for disposable and rechargeable battery types was performed to get a general idea of the trends existing in batteries and also to determine which would be the best one for our system. The research results are included in appendix A. These results were used to perform battery analysis.

Battery Analysis

Disposable Batteries

A Matlab plot for Capacity (mAh) against Volume (cm^3) for the disposable batteries that were surveyed can be seen in Figure 1-19. A bigger battery is likely to have a larger capacity, and hence a close to linear trend was observed between the two variables.

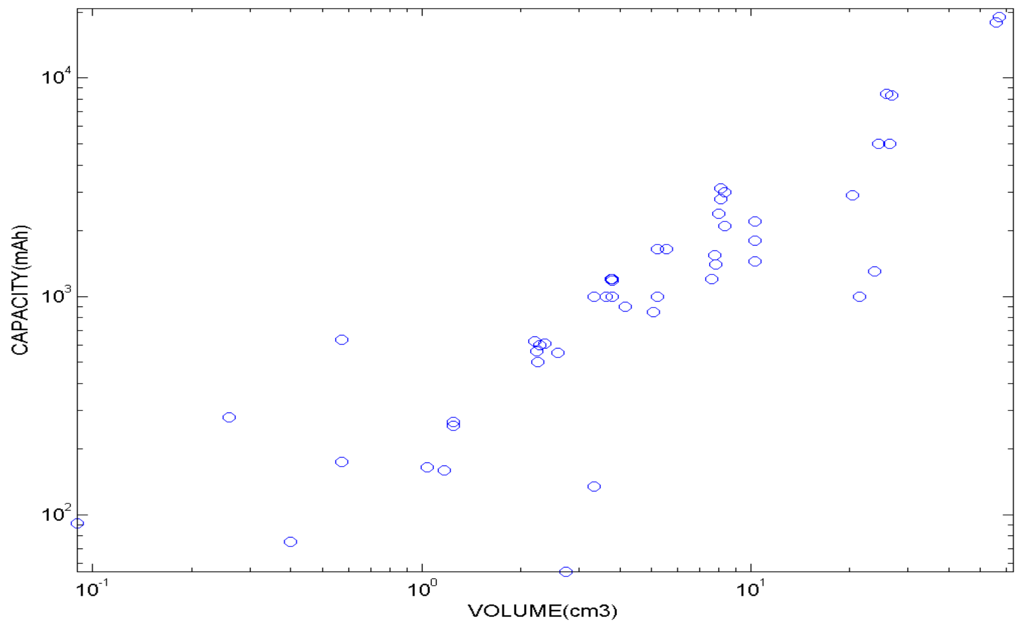
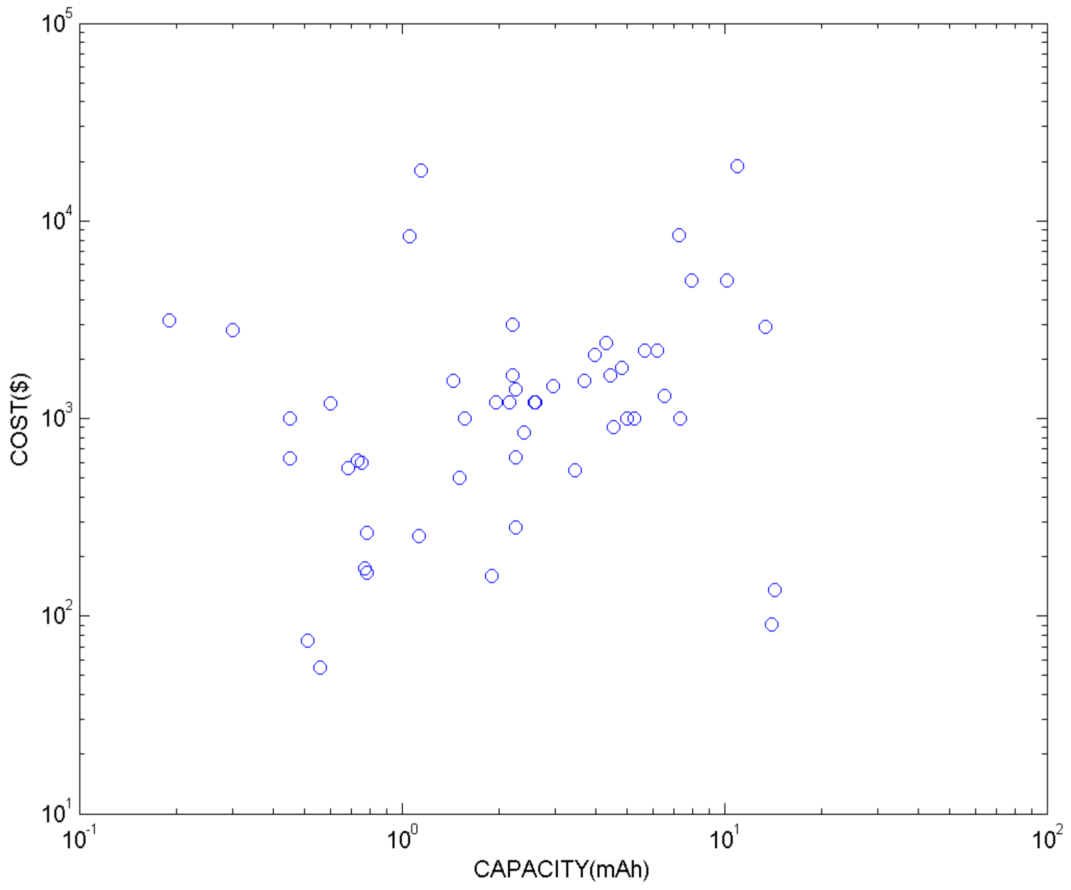


Fig.1-19: Capacity (mAh) Vs Vol. (cm^3)

The graph for the cost of the disposable batteries against the capacity can be seen in Figure 1-20. It doesn't follow any particular trend, but it can be concluded that generally a higher capacity battery would cost more (as the scatter of the points suggests).



<u>Fig.1-20:</u> Cost ($) Vs Capacity (mAh)

Rechargeable Batteries

The graph for the capacity against the volume for the rechargeable batteries that were surveyed is shown in Figure1-21. A close to linear trend can be seen, which means that the larger sized rechargeable batteries are likely to have a greater capacity.

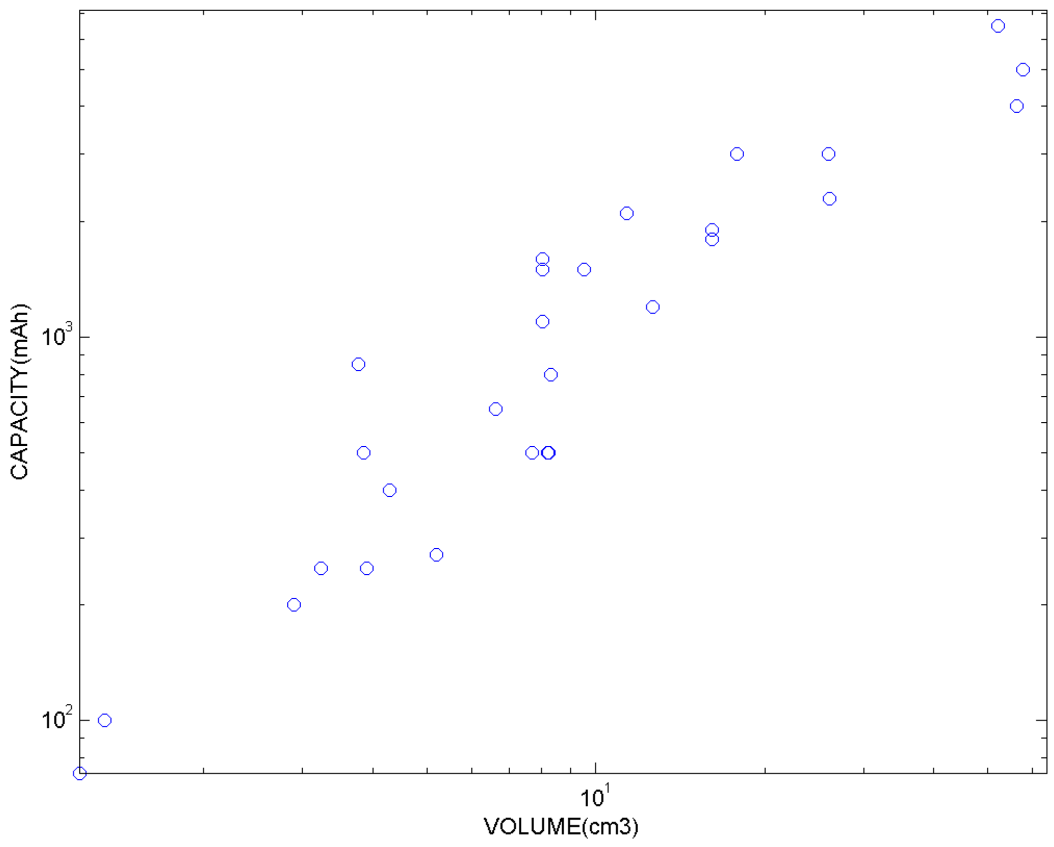


Fig.1-21: Capacity (mAh) Vs Vol. (cm^3)

The graph for cost against capacity of the rechargeable batteries can be seen in Figure 1-22. The scattered points do not seem to follow any specific trend, but a general rise in the cost of the cell can be seen as the capacity increases.

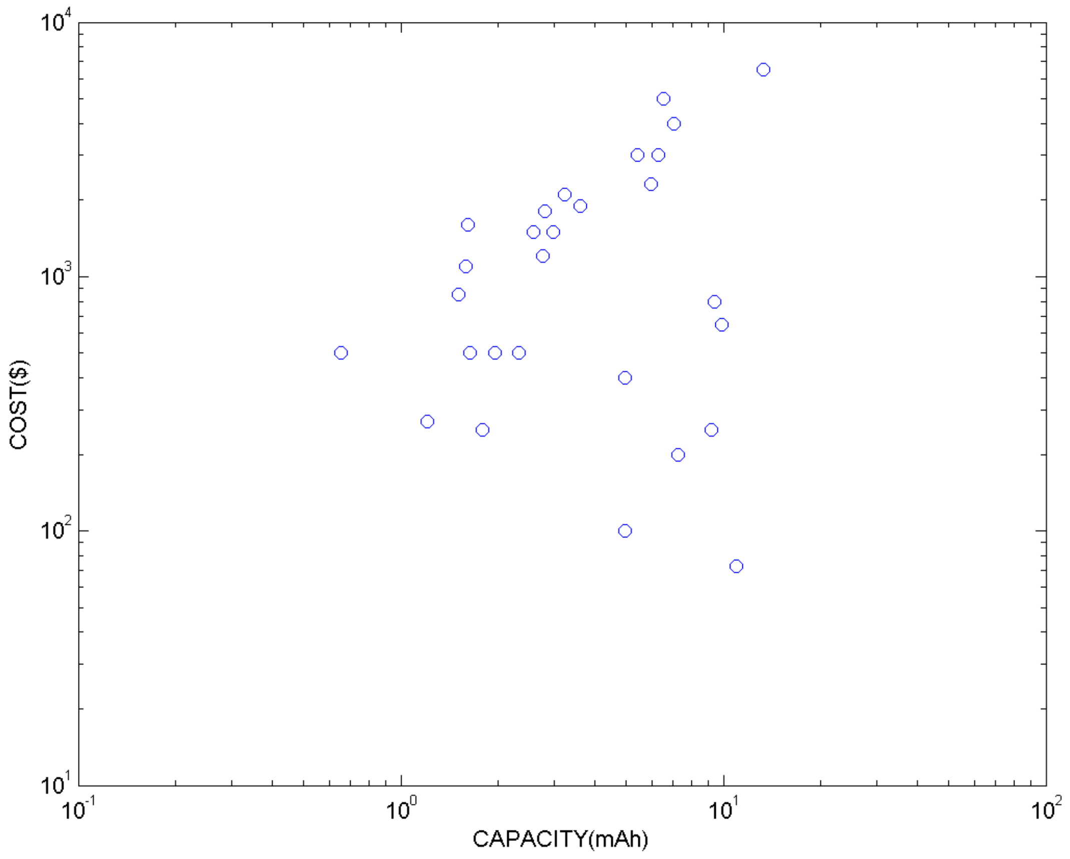


Fig.1-22: Cost ($) Vs Capacity (mAh)

This concludes the market research that we carried out for the different modules of our device: namely the DVM chip, the Microcontroller, the Wireless transceiver and the battery. In the next chapter, we would discuss which one of these options we decided to incorporate in our final product and what the reasons for their selection were.

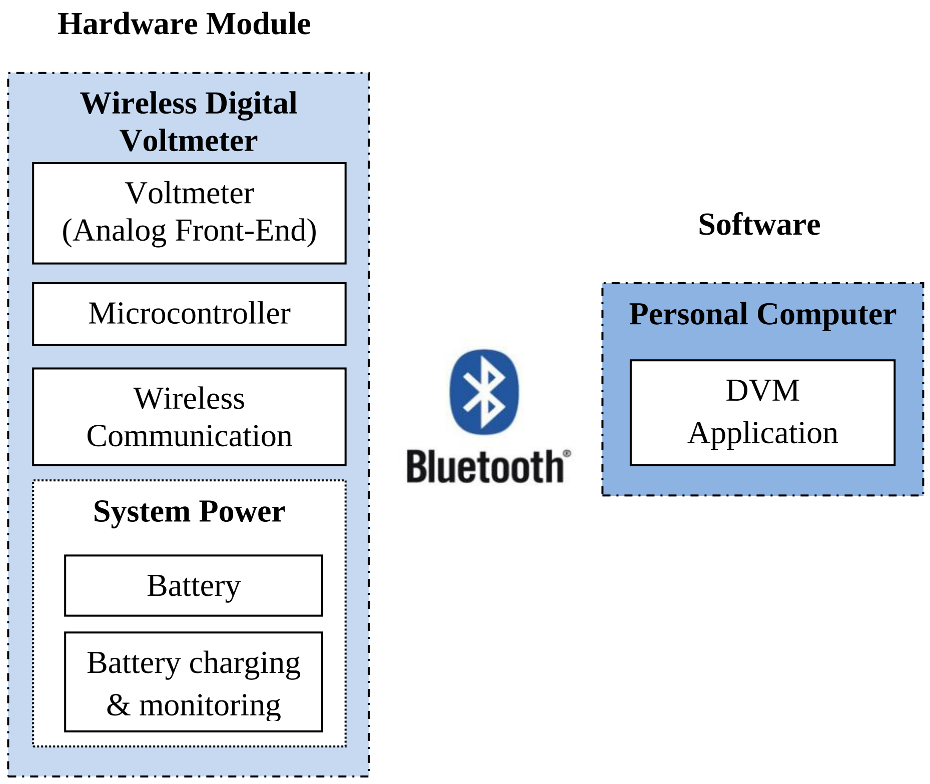
2. Design Details

In this chapter, we will discuss the overall design process of our system. After conducting the background research, we were in a position to come up with the design of the essential systems and the background research had also given us an idea of the critical sub systems that would be needed to make our entire device function.

In the following sections of this chapter we will take a look at the overall design of our project to provide an understanding of how the individual components come together. We shall first discuss why each component was selected and how it performed its respective function in the overall project design.

2.1. Overall Design

To better understand the whole system we decided to break our design into blocks which are shown in the block diagram (figure 2-1)



<u>Fig.2-1:</u> (Initial System Block Diagram)

As can be seen in the diagram our system has now been divided into two major blocks. One block is the Wireless Digital Voltmeter Hardware block which contains all the physical circuitry for our system. The other block is the external device which is going to have the software application which is going to act as our user interface with the Digital Voltmeter.

2.1.1. DVM Block (Analog Front-End)

Our first task was to determine the specifications for our Voltmeter as the rest of our design was going to be dependent on these specifications. Hence, the specifications for our Voltmeter Front-End were chosen to be as following:

- Supply Voltage = 3 – 5V (As it a portable DVM)
- Input power < 10mW (least power should be spent on the DVM, as the wireless transmission device would require a lot of power from the battery as well)
- Accuracy = 0.1% (0.01V resolution)
- Display Digits = 5 (this would give us a reasonable level of accuracy and precision)
- Range = 0 – 200.00V
- 3 voltage ranges (0-2V, 0-20V & 0-200V)
- Small size of the DVM front End so it would take less board space.

Based on these requirements the AD7799 chip seemed to be the best option to implement the analog Voltmeter Front-End of our DVM. That is because it required a small input voltage (2.7 – 5.25V) and current (380uA) and hence a small input power (which satisfied our <10mW condition). Its 24-bit analog to digital converter was capable of providing us with a 6 digit display, although we required only 5 digits in our DVM. Moreover, it had a very small size (about 5x5mm) which made it one of the smallest chips we could possibly use to build a DVM.

2.1.2. Wireless Communication

There were three basic communication requirements of our system, which were the data transmission rate, the range and the power consumption of the transceiver.

Data Rate:

The data rate was based on our DVM design specifications. According to our design, the DVM was going to have 5-digit display. So these 5 digits have to be transmitted from the DVM module. After looking at the AD7799 datasheet, we were aware that each voltage reading consisted of 24bits of data. We wanted our device to refresh about 40 times per second to make it as real time as possible. So the required data transmission rate was:

1 voltage reading = 24 bits

Update rate = 20 readings/sec

Bits transmitted/second = 40*20 = **800Bps.**

So our required data rate was **800Bps.**

Range:

Our DVM should at least be able to communicate with the external device within a large sized room. So, approximately a range of 50-60 m would suffice.

Power:

According to our initial plan our product should have a battery life of at least 8-10 hours and hence our wireless transceiver needed to be as power efficient as possible.

<u>Choice of the Communication method.</u>

By looking at our requirements we could see that both Bluetooth and Wi-Fi had enough data speed and range to support our application. Moreover both communication options were cheap, robust and reliable. However, the only differentiating factor between the two is the power consumption. Wi-Fi consumes almost 10 times more power than a Bluetooth transceiver and hence that makes it a very expensive option for us to have in our design. Power consumption was a very important consideration for us as our product was portable and needed to have a good battery life. Therefore we decided to use **Bluetooth** as our method of communication.

<u>Choice of transceiver for Bluetooth technology.</u>

Within the Bluetooth transceivers category, we had some options to choose from: BCM2048, LMX9838 and RN-41 to name a few. We rejected the Broadcom chip BCM2048 mainly because it was too expensive for our product and it really exceeded our needs, as it integrated Bluetooth, and FM capabilities. LMX9838 (manufactured by National Semiconductors) was rejected because of its high power consumption as it targeted video and audio transmission applications. Therefore, we decided to use RN-41 (manufactured by Roving Networks) in our design as it satisfied the wireless requirements of system at very low cost and low power consumption. Some of its important features are

- Fully qualified Bluetooth 2.1/2.0/1.2/1.1 module
- Bluetooth v2.0+EDR support
- Postage stamp sized form factor, 13.4mm x25.8 mm x2mm
- Low power *(30mA connected,, <10mA sniff mode)*
- UART and USB data connection interfaces.
- Sustained SPP data rates - 240Kbps (slave),300Kbps (master)
- 8MB on board flash, HCI mode, or SPP/DUN software stacks available.

The Bluetooth IC can be seen in figure 2-2:

<u>Fig.2-2:</u> (Roving Networks RN-41 Bluetooth Module)[19]

[19] RN41 Datasheet

2.1.3. Microcontroller (Analog Devices ADuC845 Micro Converter)

Our choice of microcontroller was determined by the following factors:

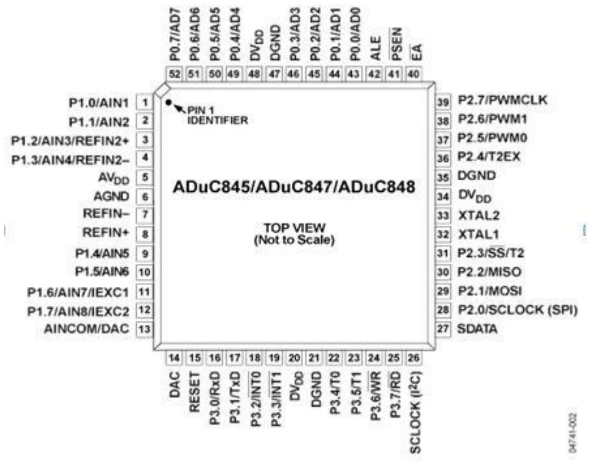
1. Availability of SPI and UART ports
2. Good number of GPIO pins.
3. Low power
4. Small size
5. Low cost
6. Easy to program.

The microprocessor manufactured by our sponsors Analog Devices met all of our requirements as can be seen from its specifications given below.

- Dual 24-Bit, Precision Sigma-Delta ADCs
- 10-Channel Single-Ended Input or 5-Channel Differential Input Options
- 8051 based core.
- Single 12-Bit Rail-to-Rail Voltage-Output DAC
- **Easy Programming through Emulation Access pin.**
- **34 GPIO Pins**
- **UART, I2C & SPI ports**
- **Ultra low power consumption.**
- **14*10mm MQFP-52 Package**

Moreover ADUC845 included dual 24 bit ADCs (analog to digital converter) which could be utilized to monitor the battery life of our battery. Hence it was decided to proceed forward with this product.

The pin configuration of ADuC845 and its physical package can be seen in figure 2-3:



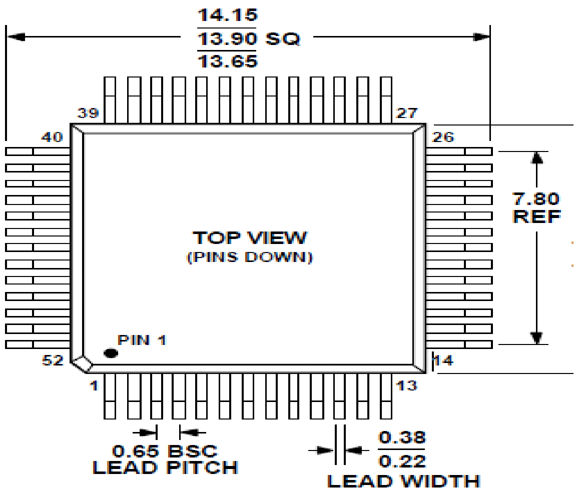


Fig.2-3: (Pin Configuration for ADUC845 (left) MQFP-52 package dimensions in mm (right))

2.1.4. System Power

The system power block consisted

I. Battery.

II. A charging source

III. A charge management and monitoring chip.

IV. A voltage Regulator chip.

i) Battery

To make the task simpler of choosing a battery simpler, we first decided to calculate the power consumption of the major systems in our device.

These include the three main components namely;

- AD7799 (Voltmeter Front-End)
- ADUC845 (Microcontroller)
- RN-41 (Bluetooth Transceiver)

Calculating Power Consumption:

To calculate the power consumption we decided to run our 3 main components at a voltage of 3.3V which lied comfortable within the input voltage range of all three chips.

AD7799 (Voltmeter Front-End):
Input voltage = 3.3V
Input Current = 380uA
Power = V.I = (3.3V) (380uA) = **1.254mW**

ADUC845 (Microcontroller):
Input voltage = 3.3V
Input Current = 4.6mA
Power = V.I = (3.3V) (4.6mA) = **15.18mW**

<u>RN41 (Bluetooth Transceiver):</u>
Input voltage = <u>3.3V</u>
Input Current = <u>30mA</u>
Power = V.I = (3.3V) (30mA) = **<u>99mW</u>**

Combined Power Consumption =1.254mW + 15.18mW+99mW =**115.43mW**

Total Power Consumption including other components = 115.432 + 15 %(115.43) = **132.74mW**

Total Current = 132.74/3.3 = **40.2mA.**

Minimum battery capacity required to have 12 hour battery life = 40.2mA*12 = **482.4mAh.**

Based on the calculations we divided our battery requirements into two parts:

1.) Fixed Requirements:

- 3 – 4V Voltage Range
- Battery capacity > 483mAH.
- Instantaneous current drawn > 45mA
- Should be rechargeable

2.) Flexible Requirements:

- Size should be small enough to hide under a 3.5*1.8inch PCB.
- Cost of the battery should be preferably less than $10

We then compared our requirements to the rechargeable battery research shown in table 2-1

Battery P/N	**Family**	**Type**	**Rechg Y/N**	**Volume (cm^3)**	**Weight (grams)**	**Voltage (V)**	**Capacity (mAh)**	**Series R (mohms)**	**Cost ($)**
IPod battery	Lithium	IPod shuffle Gen 2	Y	2.90	4.3	3.7	200	180	7.25
IPod battery	Lithium	IPod shuffle Gen 1	Y	3.24	6.0	3.7	250	200	9.20
Ultralife UBP001	**Lithium**	**Cell Phone/IPod**	**Y**	**20.0**	**31.0**	**3.7**	**1700**	**40**	**8.00**
IPod Battery	Lithium	IPod Nano Gen 2	Y	4.30	8.0	3.7	400	300	4.95
PDA-67LI	Lithium	IPod Mini	Y	6.63	28.0	3.7	550	200	9.90
BLI-919-.8	Lithium	Cell phone battery	Y	8.33	50.0	3.7	800	250	9.40

Table 2-1: (Final Battery Options)

Amongst these, the battery which caught our attention was the Ultralife UBP001 battery as it provided the greatest capacity at a price less the $10, at the same time satisfying all the other requirements such as small size and also providing the lowest series resistance. With a capacity of 1700mAh it could provide a battery life of about 40 hours as according to calculations our final circuit withdrew a total current of 40mA.

Fig.2-4: (Ultralife UBP001 Lithium-Ion Battery)[20]

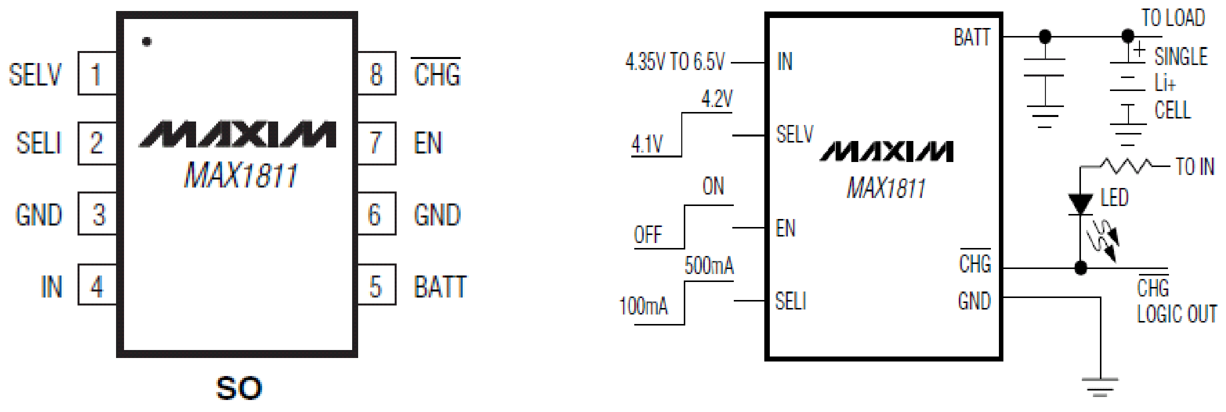
[20] www.mouser.com

ii) *Charging Source:*

To make our product more versatile we decided to implement a USB charging system in our device. Our Device would be designed with a USB connector which could be hooked up with a USB port on a Laptop or a PC to charge our battery.

iii) *Charge Management and Monitoring*

To protect our battery from over charging we had to use a charge management chip in our design. We had a number of chips (LTC4095, ADP3820 and MAX1811) available for this purpose which would was specifically designed to charge a 3-4V battery with a 5V USB input. However we decided to proceed forward with the Max1811 chip produced by Maxim because of its low cost, ease of availability and simple external circuit design.



<u>Fig.2-5:</u> (Pin Configuration for MAX1811 (left) typical operating circuit (right))[21]

The simple external circuit was the main attraction of this chip and is shown in figure 2-5(right).

Another important aspect of our product would be its ability to indicate to the user the battery life remaining. This simply requires an Analog to Digital converter which samples the battery

[21] MAX1811 Datasheet

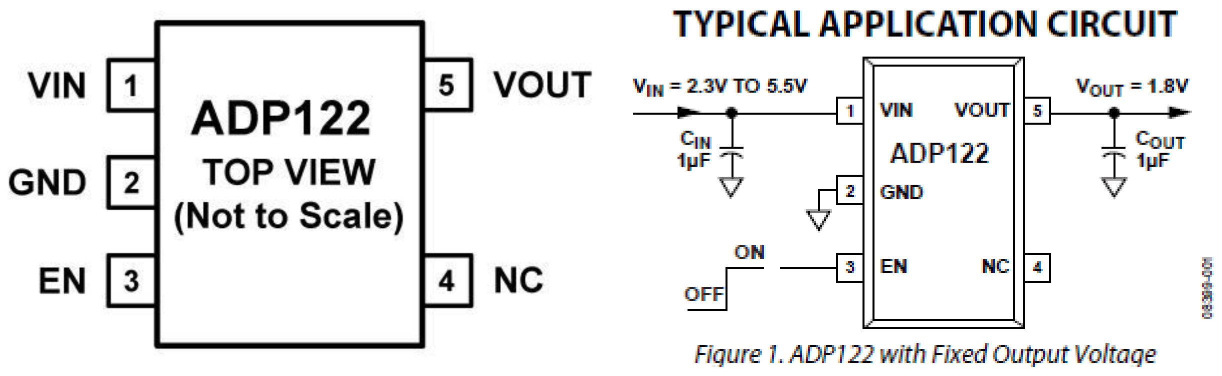
voltage and compares it to a known voltage or the full charged value of the battery. Based on that if it’s a linear battery the battery life can be estimated.

To incorporate battery monitoring in our system we had two options, either we could use and external chip to continuously sample the battery or we can use the 24 bit ADC present on the ADUC845 microcontroller which would sample the battery when the user would send a battery measure command through the software.

To avoid getting an extra chip that would increase the costs and take up more board space we decided to go ahead with using the ADC present on our microcontroller chip.

iv) Voltage Regulator

After finalizing our battery, we had to choose a linear regulator chip for our device that would regulate the voltage from the battery and make sure that it supplied all the parts of the circuit with a fixed DC voltage, as to avoid any surges and peaks. We used another Analog Devices chip for this purpose, called ADP122 which is a CMOS Linear Regulator with a fixed output voltage. We chose that ADP122 part which had a voltage output of 3.3V, as this value is smaller than our nominal battery voltage (3.7V) and is enough to drive all the components in our circuit. Figure 2-6 shows the pin configuration of ADP122:



<u>Fig.2-6:</u> (Pin Configuration for ADP122 (left) typical operating circuit (right))[22]

[22] Analog.com

2.2. System Block Diagram

Now that we knew which components we were using for the various blocks of the system, we were in a position to update the block diagram and make it more detailed.

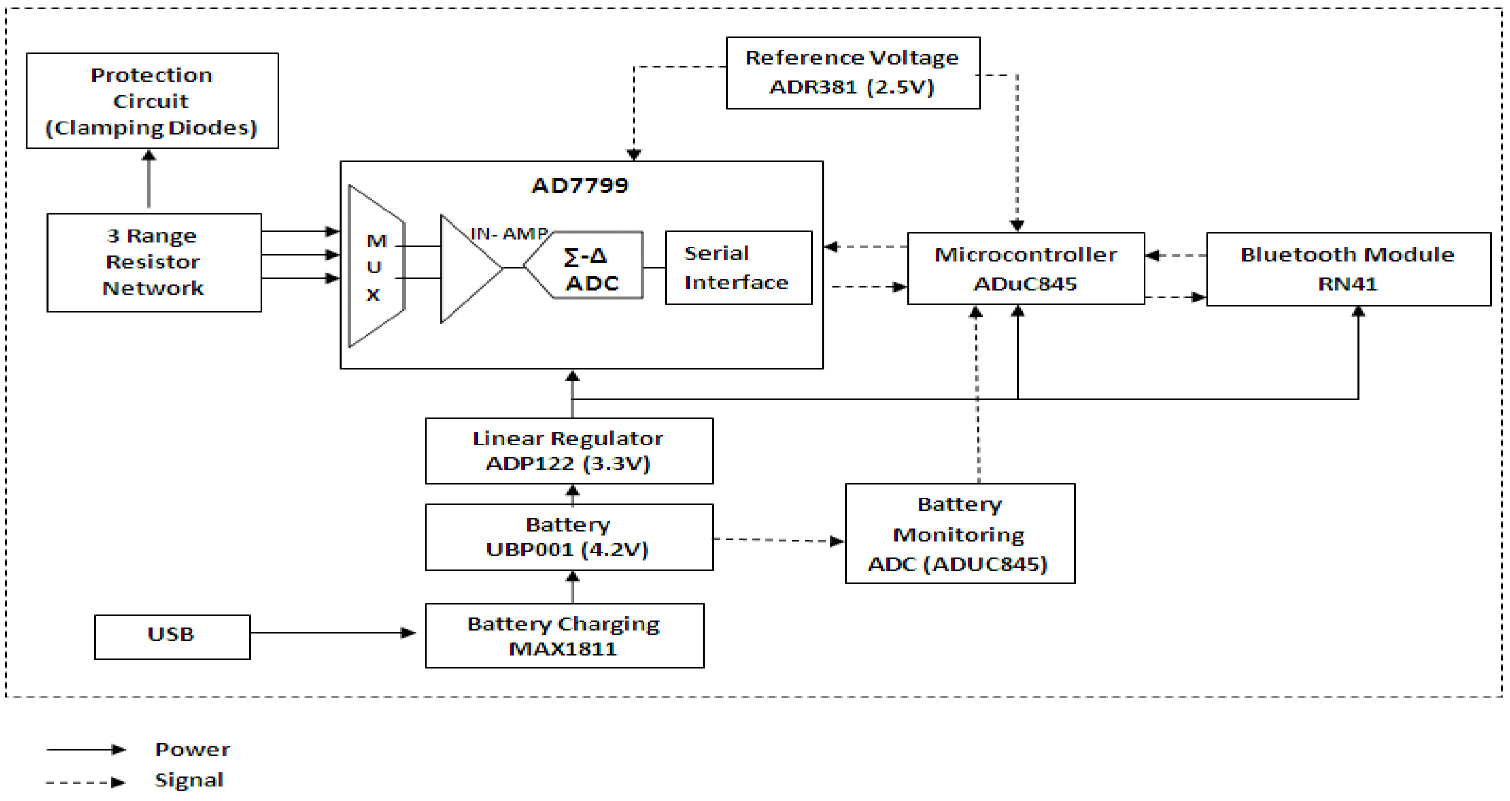


Fig.2-7: (Detailed Block Diagram)

The solid lines show the flow of power through our circuit whereas the dotted lines indicate the flow of signals and data.

3. Functionality Testing and Results

In this chapter we will discuss the nature of the functionality tests that we performed on our different subsystems, the procedures in which these tests were performed and the respective results that we obtained for each one. Our design consisted of three main subsystems: the voltmeter front end (AD7799), the Bluetooth Module (RN-41) and the Microprocessor (ADuC845). For these specific parts, we obtained their evaluation boards from their vendors and using them performed individual testing on each one of them.

3.1. The AD7799 (Voltmeter Front-End)

As mentioned above, all of the functionality testing for AD7799 was performed using its evaluation board that was obtained from Analog Devices. A picture of the board can be seen in figure 3-1. The software to configure and test the IC came with the evaluation board.

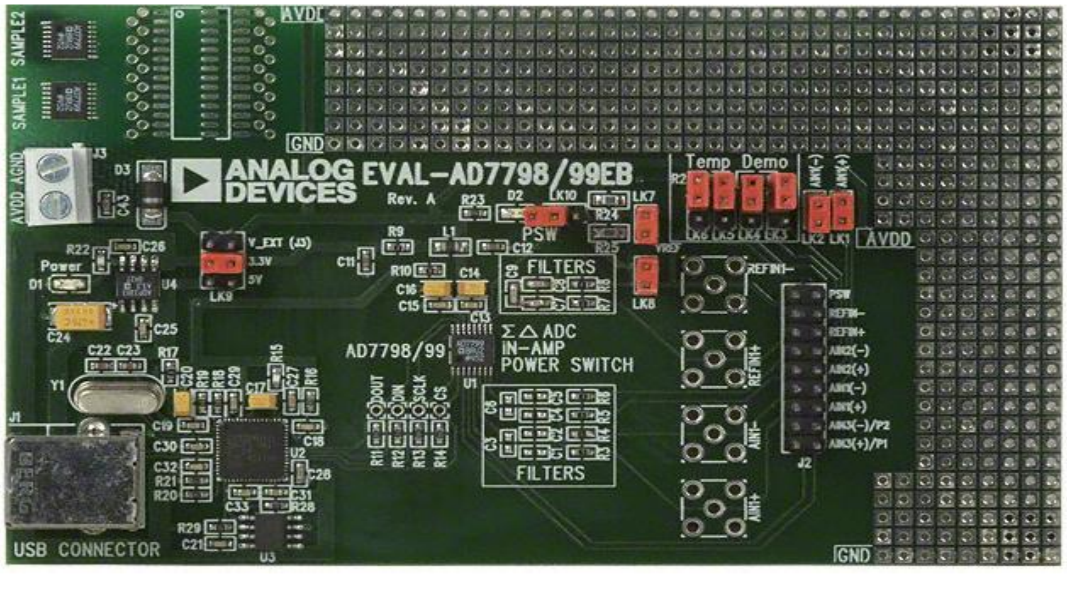
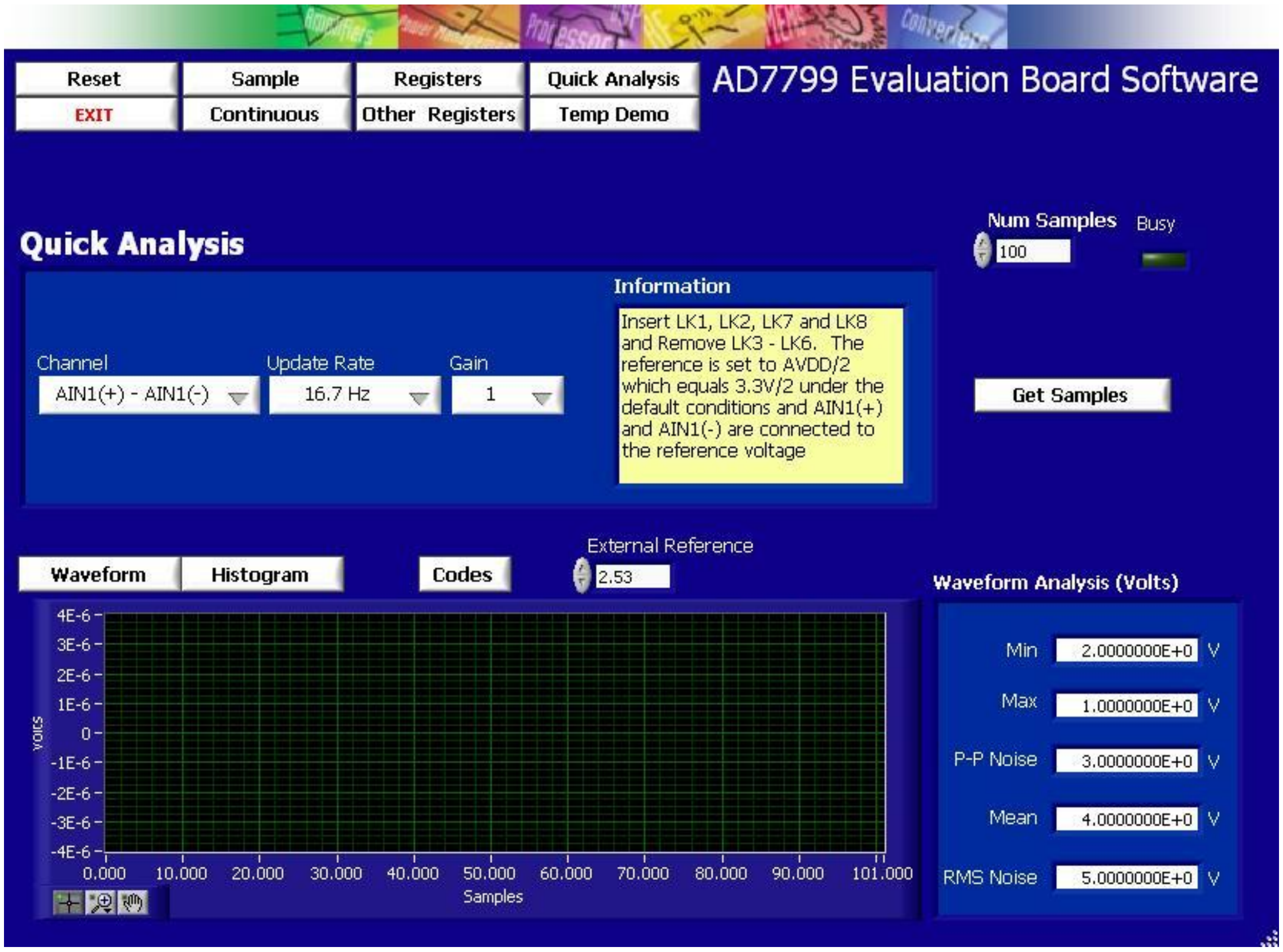


Fig.3-1: (Analog Devices AD7799 Evaluation Board)[23]

[23] Digikey.com

All the different specifications of AD7799 were tested using this board. We obtained a set of voltage readings for each different value of gain that was available to us: 1, 2, 4, 8, 16, 32, 64 and 128 to see which setting would work the best in our design. We observed the different results and figured out what factors influenced the voltage readings in what ways. A screenshot of the AD7799 evaluation board software can be seen in figure 3-2. It shows that the voltmeter has been configured for a gain of 1, an update rate of 16.7Hz and a voltage reference of 2.53V.



AD7799 Evaluation Board Software

Quick Analysis

Num Samples: 100

Channel: AIN1(+) - AIN1(-)
Update Rate: 16.7 Hz
Gain: 1

Information
Insert LK1, LK2, LK7 and LK8 and Remove LK3 - LK6. The reference is set to AVDD/2 which equals 3.3V/2 under the default conditions and AIN1(+) and AIN1(-) are connected to the reference voltage

External Reference: 2.53

Waveform Analysis (Volts)

Min	2.0000000E+0	V
Max	1.0000000E+0	V
P-P Noise	3.0000000E+0	V
Mean	4.0000000E+0	V
RMS Noise	5.0000000E+0	V

Fig.3-2: (AD7799 Evaluation Software)

After our initial testing of the board was complete, we knew that we needed to supply a Common-Mode voltage (V_{CM}) to our AD7799 inputs in order to get more accurate results, which in our case was: $V_{CM} = V_{CC}/2 = 5V/2 = 2.50V$. We also found out that we needed to perform Zero and Full-Scale calibration on AD7799 to make sure that our results were within the 0.1% accuracy requirement of our device. Voltage reference was kept at 2.53V throughout the testing, as was recommended. With all these requirements met, we were able to extract the desired results from the AD7799 and our 0.1% accuracy condition was being met. The voltage readings that were obtained after configuring AD7799 according to our needs can be seen in table 3-1.

It can be seen that all the voltage readings obtained in the -2.5V to +2.5V have a percentage error of less than 0.1%. This was sufficient for our needs, as our DVM was supposed to be 0.1% accurate.

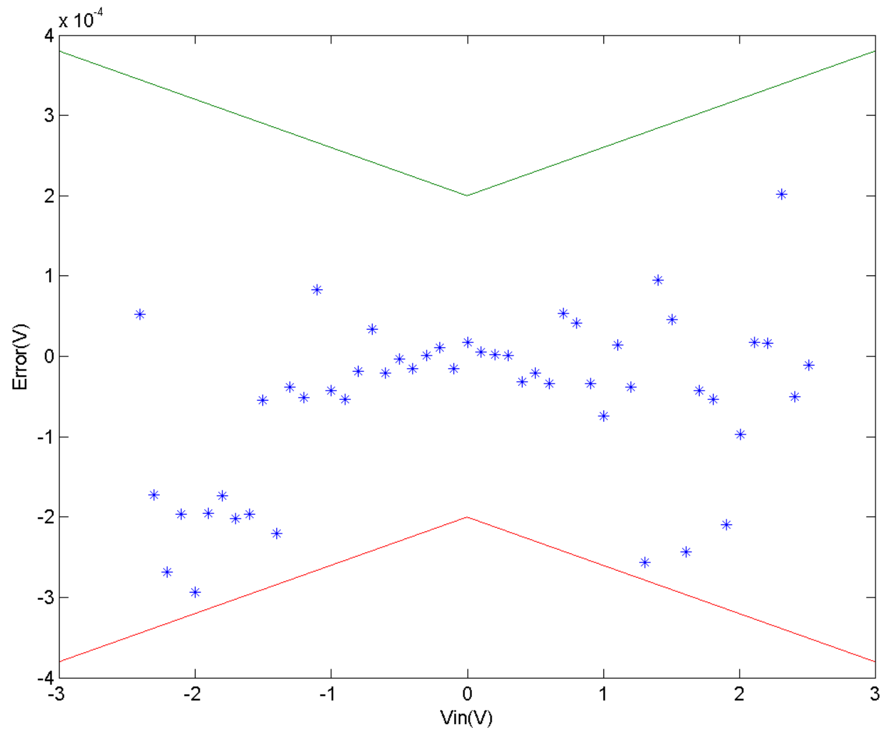
Gain = 1	Gain correction = 5508FF= 1.004664V	Offset = 7FFFC0=0.000025V

Vin(V)	Vout(V)	Error (V)	%age error
-2.50977	-2.50960	-0.000174	0.007
-2.40890	-2.40895	0.000052	0.002
-2.30860	-2.30843	-0.000172	0.007
-2.20850	-2.20823	-0.000268	0.012
-2.10813	-2.10793	-0.000196	0.009
-2.00604	-2.00575	-0.000294	0.015
-1.90577	-1.90557	-0.000195	0.010
-1.80549	-1.80532	-0.000173	0.010
-1.70520	-1.70500	-0.000202	0.012
-1.60493	-1.60473	-0.000196	0.012
-1.50463	-1.50457	-0.000055	0.004
-1.40441	-1.40419	-0.000221	0.016
-1.30415	-1.30411	-0.000038	0.003
-1.20388	-1.20383	-0.000051	0.004
-1.10360	-1.10368	0.000083	0.007
-1.00334	-1.00330	-0.000043	0.004
-0.90307	-0.90302	-0.000053	0.006

Vin(V)	Vout(V)	Error (V)	%age error
-0.80280	-0.80278	-0.000019	0.002
-0.70201	-0.70205	0.000034	0.005
-0.60170	-0.60168	-0.000021	0.004
-0.50138	-0.50138	-0.000003	0.001
-0.40111	-0.40109	-0.000015	0.004
-0.30086	-0.30086	0.000001	0.000
-0.20058	-0.20059	0.000011	0.006
-0.09990	-0.09989	-0.000015	0.015
0.00040	0.00042	0.000017	4.240
0.10073	0.10074	0.000006	0.006
0.20151	0.20151	0.000002	0.001
0.30184	0.30184	0.000001	0.000
0.40222	0.40219	-0.000032	0.008
0.50257	0.50255	-0.000021	0.004
0.60290	0.60287	-0.000034	0.006
0.70330	0.70335	0.000054	0.008
0.80422	0.80426	0.000041	0.005
0.90460	0.90457	-0.000034	0.004
1.00488	1.00481	-0.000074	0.007
1.10515	1.10516	0.000014	0.001
1.20547	1.20543	-0.000038	0.003
1.30585	1.30559	-0.000257	0.020
1.40611	1.40621	0.000095	0.007
1.50644	1.50649	0.000046	0.003
1.60676	1.60652	-0.000243	0.015
1.70715	1.70711	-0.000043	0.003
1.80748	1.80743	-0.000054	0.003
1.90780	1.90759	-0.000210	0.011
2.00811	2.00801	-0.000097	0.005
2.11038	2.11040	0.000017	0.001
2.21066	2.21068	0.000016	0.001
2.31105	2.31125	0.000202	0.009
2.41135	2.41130	-0.000050	0.002
2.51160	2.51159	-0.000011	0.000

Table 3-1: (Voltage Readings obtained from the AD7799 Eval.Board)

Figure 3-3 shows how our voltage readings were within the threshold limits that were set in order to get 0.1% accuracy from our DVM. For our application, we only needed to make sure that our voltmeter gave us the accurate results for the range: -2.5V to +2.5V, as our external resistor network would always downscale the input voltage to within this range.

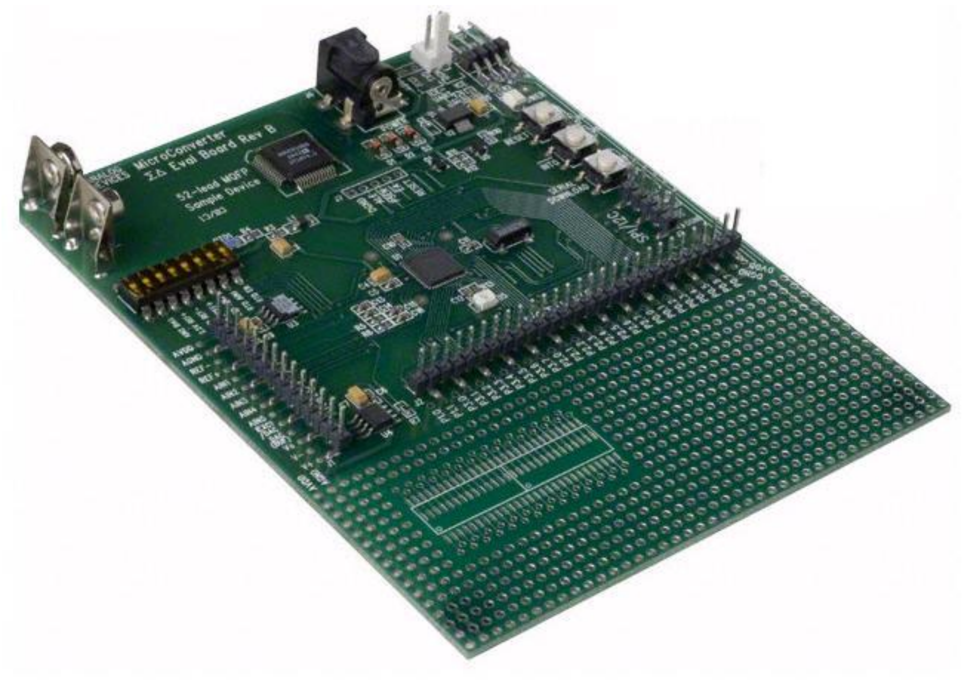


<u>Fig.3-3:</u> (Error values in the -2.5V to 2.5V range)

Therefore, we performed the complete functionality testing for AD7799 using its evaluation board, and kept on testing and editing the voltmeter settings until we achieved the desired results. After the AD7799 configuration was finalized, we were in a position to move on to the other subsystems in our design and configure them.

3.2. The ADuC845 (Microprocessor)

The microprocessor that we used in our design was the Analog Devices ADuC845. Just like AD7799, we performed all of our functionality testing for ADuC845 using its evaluation board obtained from Analog Devices. This evaluation board can be seen in figure 3-4.



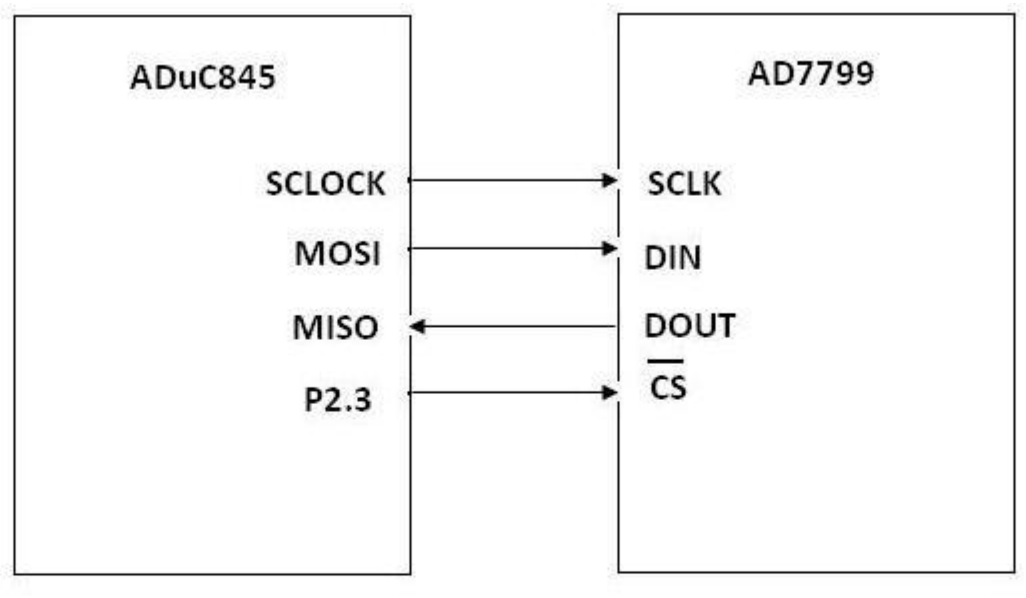
<u>Fig.3-4:</u> (Analog Devices ADuC845 Evaluation Board)[24]

The first step was to install the required software that would download programs on to the ADuC845 evaluation board. We chose to use IAR KickStart, as it was compatible with our kit and we had the option of using C as the development language, a software language with which we had considerable experience.

After obtaining the hardware and installing the required software, one of the first tasks was to make sure that our ADuc845 had the capability to program our AD7799 IC using the SPI (Serial

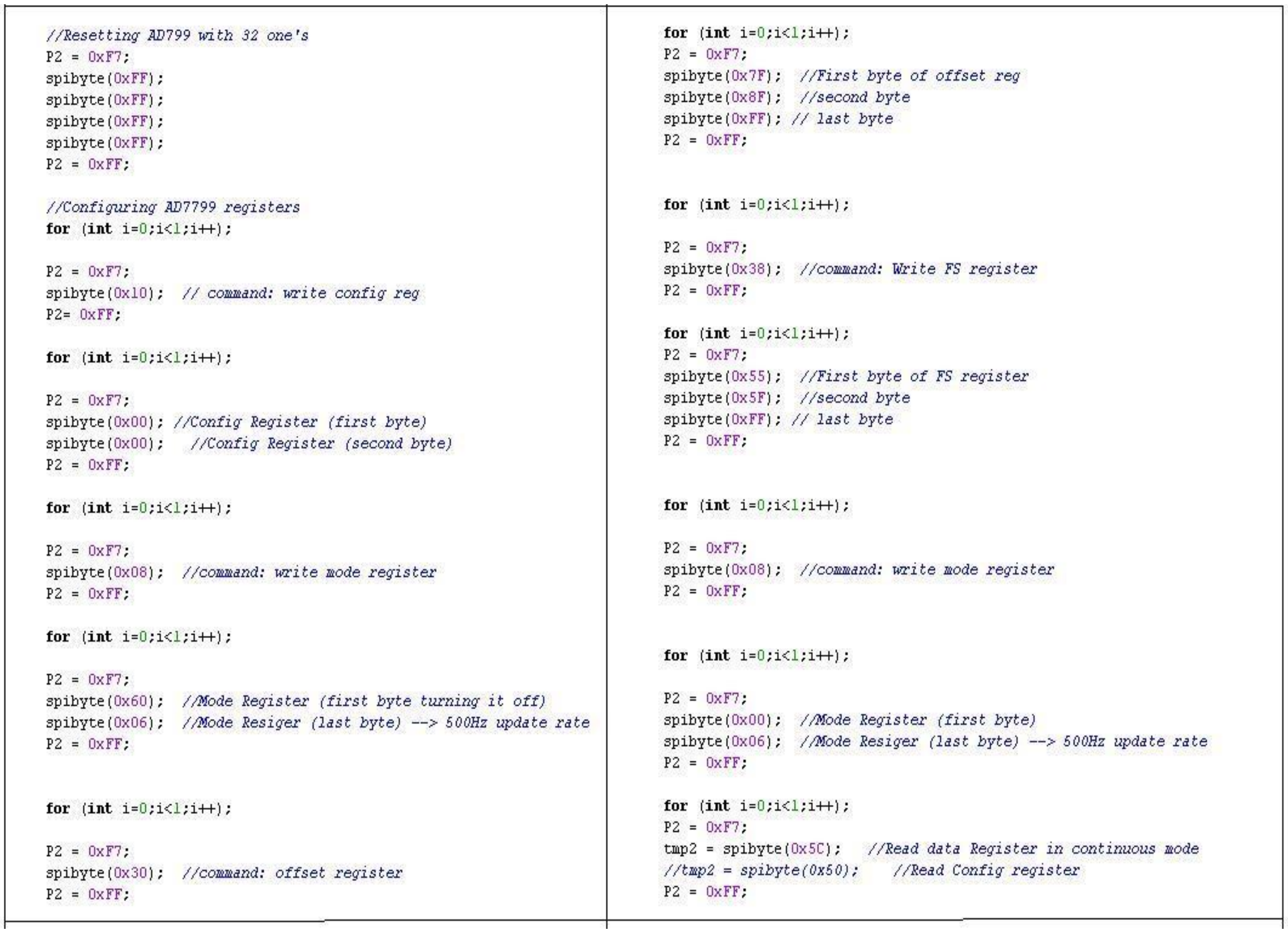
[24] Digikey.com

Peripheral Interface). In this SPI connection, the ADuC845 acted as the master block and AD7799 was the slave peripheral. Therefore, the MOSI (Master Out Slave In) pin of the ADuC845 board was connected to the D_{IN} (Data in) pin of the AD7799 board, and the D_{OUT} (Data out) pin of the AD7799 was connected to MISO (Master In Slave Out) pin of the ADuC845. The SCLOCK (Serial Interface Clock) pin of the ADuC845 was connected to the SCLK (Serial Clock Input) pin of the AD7799 board to synchronize both the systems, so that both of them followed the same clock. The SPI connections between the two boards can be seen in figure 3-5. An I/O pin (P2.3) from ADuC845 was connected to the Chip Select (CS) pin of AD7799 so that the microprocessor had the power to select the AD7799 whenever it needed to.



<u>Fig.3-5:</u> (SPI Connection between ADuC845 and AD7799)

The complete C code that was written for ADuC845 can be found in the Appendix E at the end of the report. The snippet of code which configured the AD7799 for bipolar operation to meet our voltmeter settings of: Gain = 1, Update Rate = 39.2Hz, and Continuous Conversion Mode can be seen in figure 3-6.



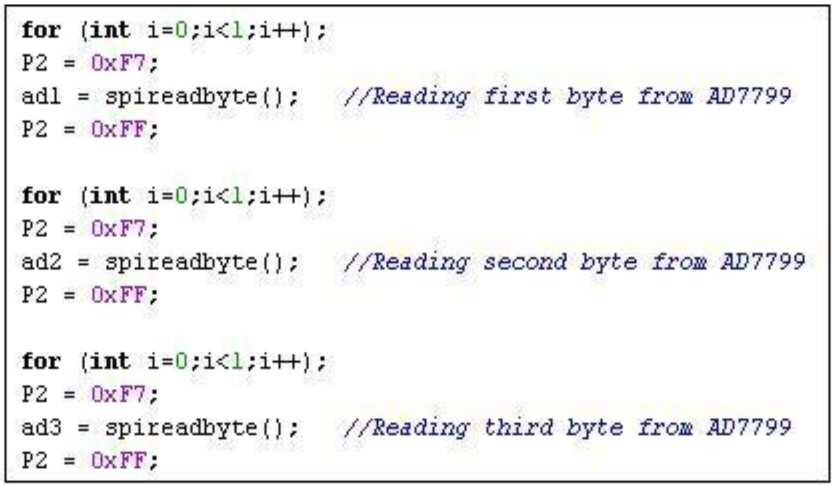
```
//Resetting AD799 with 32 one's
P2 = 0xF7;
spibyte(0xFF);
spibyte(0xFF);
spibyte(0xFF);
spibyte(0xFF);
P2 = 0xFF;

//Configuring AD7799 registers
for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x10);  // command: write config reg
P2= 0xFF;

for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x00); //Config Register (first byte)
spibyte(0x00);   //Config Register (second byte)
P2 = 0xFF;

for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x08);  //command: write mode register
P2 = 0xFF;

for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x60);  //Mode Register (first byte turning it off)
spibyte(0x06);  //Mode Resiger (last byte) --> 500Hz update rate
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x30);  //command: offset register
P2 = 0xFF;
```

```
for (int i=0;i<1;i++);
P2 = 0xF7;
spibyte(0x7F);  //First byte of offset reg
spibyte(0x8F);  //second byte
spibyte(0xFF); // last byte
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x38);  //command: Write FS register
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
spibyte(0x55);  //First byte of FS register
spibyte(0x5F);  //second byte
spibyte(0xFF); // last byte
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x08);  //command: write mode register
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x00);  //Mode Register (first byte)
spibyte(0x06);  //Mode Resiger (last byte) --> 500Hz update rate
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
tmp2 = spibyte(0x5C);   //Read data Register in continuous mode
//tmp2 = spibyte(0x50);    //Read Config register
P2 = 0xFF;
```

Fig.3-6: (C code for ADuC845 to configure the AD7799 registers)

Once the AD7799 had been placed in continuous conversion mode the next task was to verify the accuracy of the serial data in HEX format that was being placed at the DOUT pin of AD7799 and thus coming into the microcontroller through the MISO pin. Initially, a Logic Analyzer placed at the SPI interface was used to verify the input, the outputs and serial clock. However, to decipher these hex values into voltage readings a *SPIreadbyte()* method was incorporated in our software. This part of the code can be seen in figure 3-7. As each AD7799 reading consisted of

24 bits of data, it required three variables (8 bits each) in the IAR KickStart to hold the value of one voltage measurement. This can be seen in fig.3-7.



```
for (int i=0;i<1;i++);
P2 = 0xF7;
ad1 = spireadbyte();   //Reading first byte from AD7799
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
ad2 = spireadbyte();   //Reading second byte from AD7799
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
ad3 = spireadbyte();   //Reading third byte from AD7799
P2 = 0xFF;
```

<u>Fig.3-7:</u> (C code to read SPI data and store it in different variables)

Once these Hex values had been read into the microcontroller, they were converted into voltage values by using the algorithm

$$Code = 2^{N-1} \times [(AIN \times GAIN/V_{REF}) + 1]$$

where:
AIN is the analog input voltage.
$N = 24$ for the AD7799.

These voltage values were found to be exactly what they were expected to be.

After having established a two-way SPI connection between the most important modules of our system (AD7799 and ADuC845), the next step was to transmit this data from the ADuC845 to the Bluetooth module (RN-41) using the UART (Universal Asynchronous Receiver/Transmitter) interface. The connection between the ADuC845 and the Bluetooth Module (Rn-41) is discussed in the next section.

3.3. RN-41 (Bluetooth Module)

To make it easier to hook up the Bluetooth transceiver to the external components without having a PCB it was decided to use the development board for RN-41.The board is shown in figure 3-8:

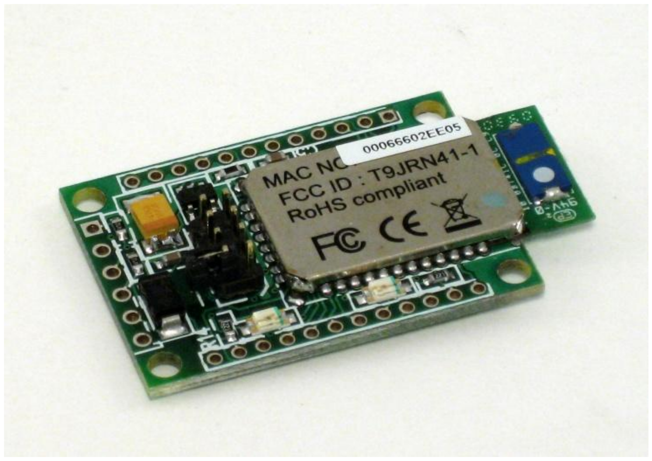
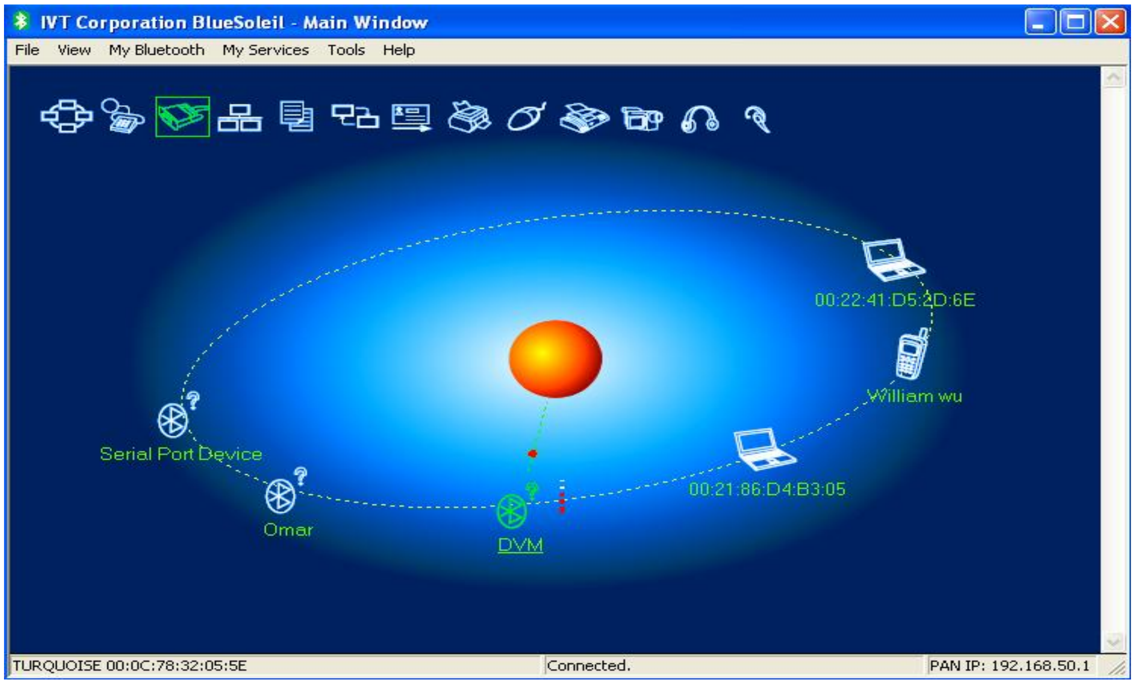


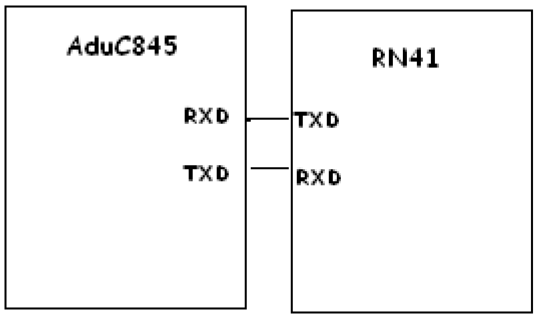
Fig.3-8: (Roving Networks RN-41 Bluetooth Board)

RN-41 Bluetooth transceiver comes with the complete Bluetooth protocol stack already programmed on chip. According to factory setting the device was in slave mode with auto detection feature enabled. Additionally the device came with external USB Bluetooth transceiver which in our case was connected to a PC to allow data transmission between the PC and the RN-41.The external USB Transceiver had the ability to map itself as a serial port in the PC and allow serial transmission of data. A generic off the shelf Bluetooth software made by IVT Corporation called Blue-Soleil was used to identify the RN-41 Bluetooth device and establish a connection with it. A screenshot of the Blue-Soleil main page, showing the established connection between the PC and RN-41(DVM) can be seen in figure 3-9.


IVT Corporation BlueSoleil - Main Window
File View My Bluetooth My Services Tools Help
00:22:41:D5:2D:6E
William wu
Serial Port Device
Omar
DVM
00:21:86:D4:B3:05
TURQUOISE 00:0C:78:32:05:5E
Connected.
PAN IP: 192.168.50.1

<u>Fig.3-9</u> (BlueSoleil Software for RN-41)

For our project, we wanted the RN-41 module to transmit the voltage readings read by the micro-controller from the AD7799 to the PC. Therefore, it was essential to establish a UART connection between ADuC845 and RN-41. The layout of the physical connection is shown below



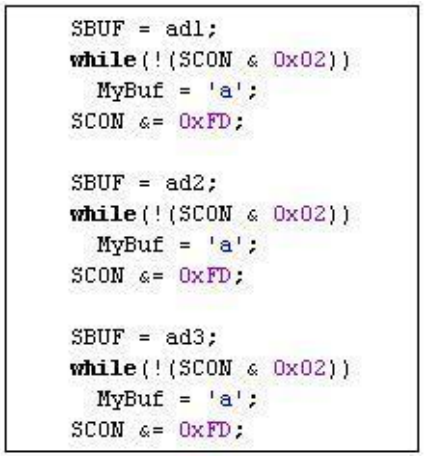
<u>Fig.3-10</u> (UART Connection)

Once the hardware connection had been established, modifications were made to the C program made earlier for the SPI interface.

The UART Port of the ADUC845 was enabled and configured for the following settings

Data = 8 Bit.
1 start and 1 stop bit.
Parity =None.
Baud Rate =9600bps.

Once configured, the microprocessor placed the data read from the SPI interface on to the Data register (SBUF) of the UART port and was sent to the RN41. As SBUF register could transmit 8 bits at a time, it took three UART transmissions to send one complete voltage reading to the Bluetooth module. This part of the code can be seen in figure 3-10. The C code that was written in IAR KickStart to configure this UART functionality can be found in the Appendix E.



```
SBUF = ad1;
while(!(SCON & 0x02))
  MyBuf = 'a';
SCON &= 0xFD;

SBUF = ad2;
while(!(SCON & 0x02))
  MyBuf = 'a';
SCON &= 0xFD;

SBUF = ad3;
while(!(SCON & 0x02))
  MyBuf = 'a';
SCON &= 0xFD;
```

Fig.3-11: (C code to send data from ADuC845 to RN-41 using UART)

At this stage, all of our evaluation boards (AD7799, ADuC845 and RN-41) were working together as one system and were performing the voltmeter functionality that we expected from our final product. But at this point, we needed a software application on our PC to display the Voltmeter reading being sent over by the Bluetooth. Moreover to enable us to control the whole system from the interface.

3.4. C# Application (Windows Application on a PC)

Since we got our Bluetooth module to send the voltage readings from AD7799 to the PC, we needed a software application on the PC that would display these results clearly, to ensure their accuracy and precision. As mentioned above the data was being received by the PC from the USB Bluetooth transceiver which was mapped as a serial port.

It was essential to provide a visual user interface therefore which would be updated real time. To achieve this functionality, it was decided to make the Windows application in Microsoft Visual Studio 2008 using the development language C-Sharp (C#).

This Microsoft Visual Studio project that we built had a very simple GUI (Graphical User Interface) so that anyone could use it without much difficulty. It had one main “Voltage” textbox that displayed the real-time voltage readings, as they were received from the Bluetooth module. It refreshed and updated itself ten times a second. The textbox can be seen in figure 3-11.

There were three radio buttons available on the form for the user to choose which voltage range he/she wanted to operate the DVM in. The three available ranges were: 0 – 2V, 0 – 20V and 0 - 200V. After one of these radio buttons was selected, the software knew what range the DVM was going to operate in and hence it sent the specific command to the DVM block so that the AD7799 was configured into the correct mode and measuring range. These buttons can also be seen in figure 3-12.

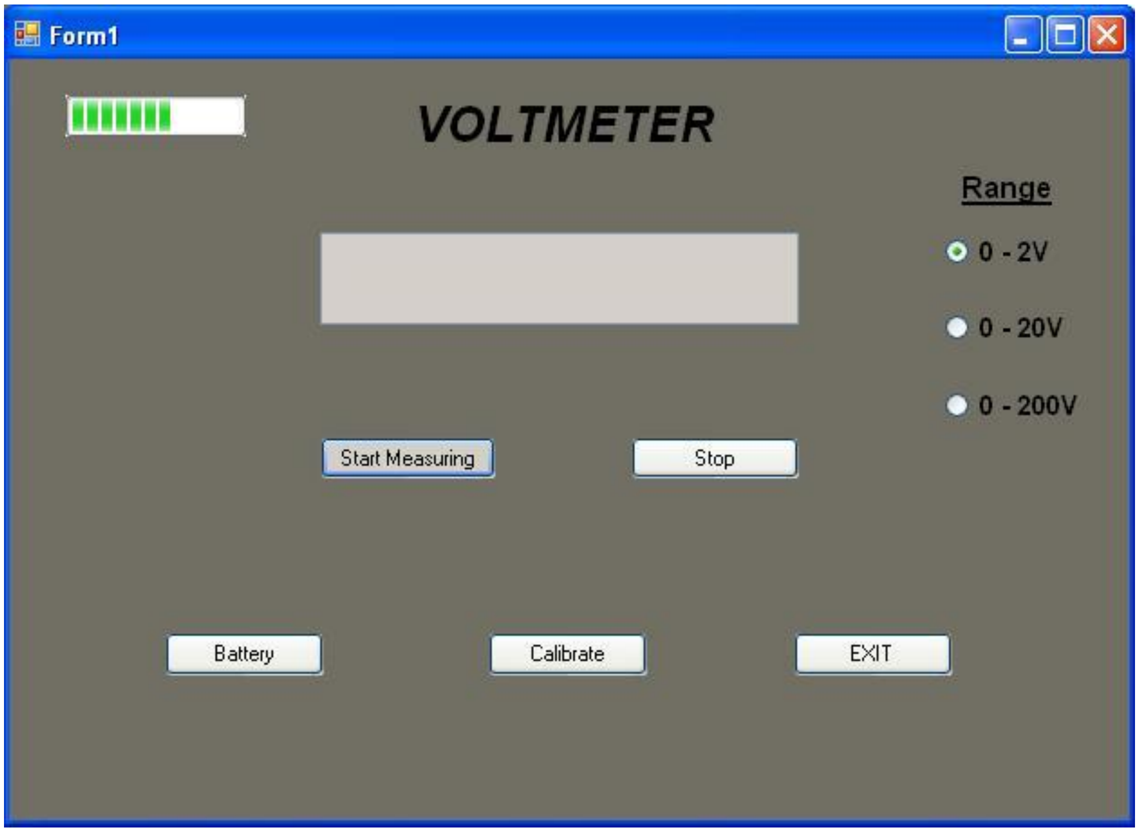


Fig.3-12: (C# Windows application form for our DVM)

Other than that, there were five buttons on the form called:

1.) **Start:** To start getting the voltage readings from the DVM block in our textbox.

2.) **Stop:** To stop the textbox from displaying the voltage readings.

3.) **Calibrate:** To send a message to the DVM block to perform complete system calibration.

4.) **Battery:** To display the remaining battery life on the form, in the form of bars.

5.) **Exit:** To exit the application.

This application basically used a serial port connection from the port that had the Bluetooth dongle connected to it, and used that to import the DVM data into Microsoft Visual Studio. The complete C# code that was written to achieve the full functionality from this application can be found in Appendix F.

4. Final Product

So far we have been taking a look at how the different blocks of our design functioned individually, and also how the complete system functionality was tested using the evaluation boards for AD7799, ADuC845 and RN-41. Having established that proof of concept, it was now time to combine all the evaluation board circuits together and make our own PCB (Printed Circuit Board) that would perform in the same manner. In this chapter we shall take a look at how we combined all the individual component circuitry to build our final DVM block, and discuss some of the new features and subsystems that we decided to incorporate in our device.

4.1 Complete Schematic

The complete schematic for our DVM block can be seen in figure 4-1. It can also be found at the end of the report in the Appendix B. All the connections between the different parts of our design can be seen clearly. The SPI connection between AD7799 and ADuC845, the UART connection between the ADuC845 and RN-41, the battery charging circuit and the voltage regulator circuit can be seen in figure 4-1.

There are three different LED’s in our design: one to show that the power is on (LED1), the second to show that the battery is being charged (LED2) and the third one to show that the data is being transmitted from the RN-41 Bluetooth module (LED3). All of these can be seen on the schematic.

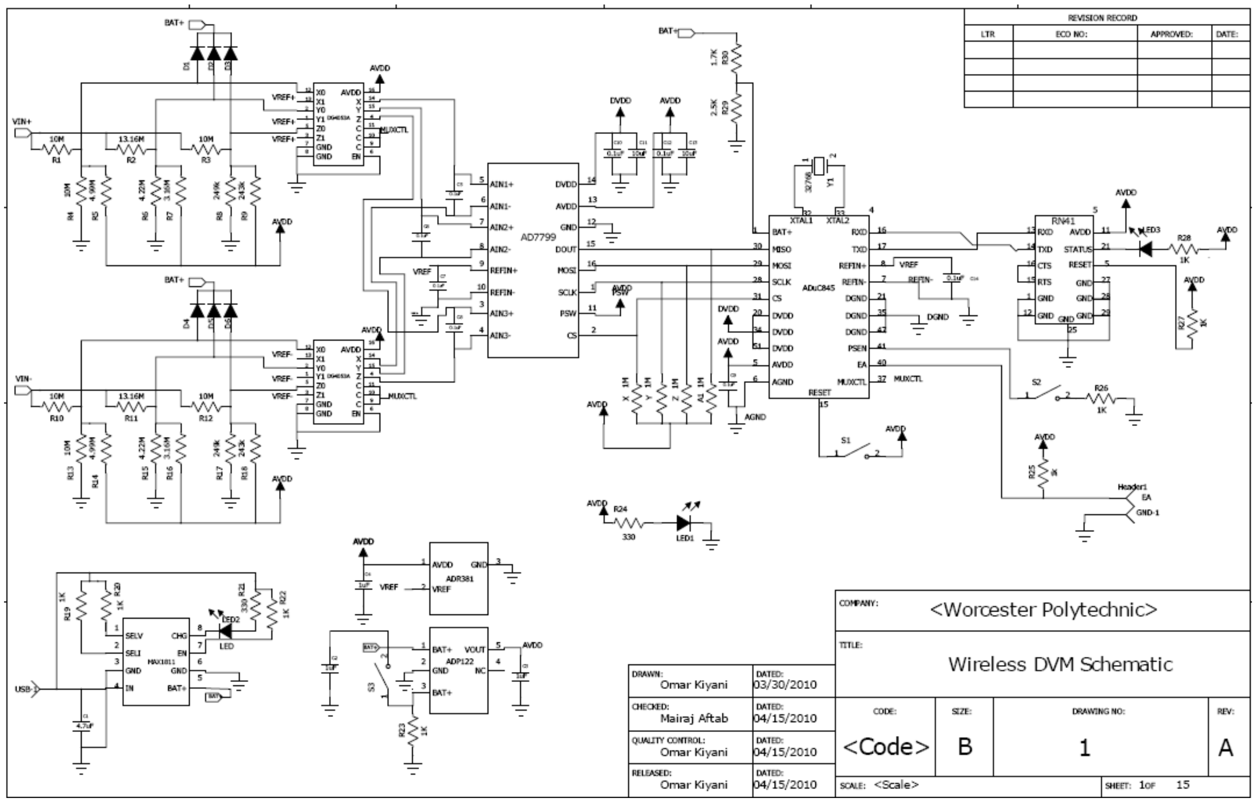


Fig.4-1: (Final Schematic)

4.1.1. Analog Front-End (AD7799) and Microprocessor Circuit

As can be seen in figure 4-1, all of the six inputs of the AD7799 were used in our design. That was because we needed to satisfy the three input voltage ranges requirement of our DVM. The analog inputs AIN1+ and AIN1- were used for range 1 (0 – 2V), AIN2+ and AIN2- were used for range 2 (0 – 20V) and AIN3+ and AIN3- were used for range 3 (0 – 200V). All of these inputs had a 0.1uF bypass capacitor connected across them in order to reduce the effect of noise. The analog inputs also went through an external resistor network, designed separately for each range, so that the input voltage for any of these analog inputs never exceeded the IC specification of AV_{DD} + 1.0V. In our case, AV_{DD} was equal to 3.3V. This was the output of the ADP122 voltage regulator that supplied the whole circuit with power. Therefore, our resistor network was designed specifically for each particular range to make sure that the input voltage for each pin would always be less than the absolute maximum rating for that pin.

We were supplying a Common-mode voltage equal to $AV_{DD}/2$ (3.3V/2 = 1.65V) to all of our analog inputs to make sure that our AD7799 performed optimally. The external voltage that was to be measured rode on top of this value and their sum went into the AD7799 analog inputs. The resistor network was chosen so as to keep the analog input smaller than 4.15V for each analog front-end of AD7799. The attenuated voltage values for each voltage reading and what range they actually represented can be seen in the following table 4-1:

Range	**Voltage Range**	**Attenuated Voltage** (Analog input to AD7799)
1	0 - 2V	1.65V – 2.05V
2	0 – 20V	1.65V – 4.15V
3	0 – 200V	1.65V – 4.15V

Table 4-1: DVM Voltage Ranges and the corresponding actual input voltages

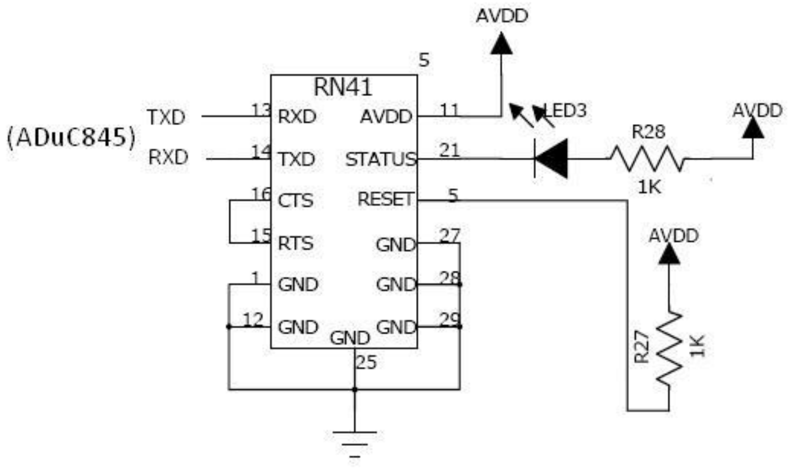
We also had certain “clamping” diodes connected to the external resistor network to protect our system from large input voltages. These diodes made sure that the voltages in the analog inputs of the AD7799 never exceeded the chip specified absolute maximum rating of 4.3V. These diodes can also be seen in the schematic in figure 4-1.

The two multiplexer IC’s (DG4035A) that can be seen in the schematic right before the AD7799 inputs were there for the calibration purpose. Each internal MUX inside the IC’s outputted the external voltage that it was getting from the resistor network when it was in the regular measuring mode. The multiplexers outputted the reference voltage (V_{REF}) when the system calibration was needed to be done. When this was done, our microprocessor knew what voltage value to expect (V_{REF}) and it then recorded the error or difference between the expected and actual voltage value. All the future voltage readings were then multiplied by this gain-error value to obtain the correct and accurate voltage reading. For this reason, these multiplexer IC's were a significant part of our product design.

All the SPI connection pins for AD7799 (DOUT, MOSI, SCLK and CS) were pulled up high using 1MΩ pull-up resistors, as can be seen in figure 4-1. The microprocessor had a switch connected to the RESET pin (no.15) to give the user an option to reset the ADuC845 settings to default. There was another switch connected at the PSEN pin (no.41) of ADuC845 which enabled the serial download of the program to the microprocessor. Push-buttons were used in both of these cases. The 32.768kHz clock crystal was connected across the XTAL1 and XTAL2 pins (no. 32 and 33) of the ADuC845 to provide the processor with a clock signal. All these connections can be seen in figure 4-1.

4.1.2. Bluetooth RN-41 Circuit

The isolated RN-41 circuit can be seen in figure 4-2. The RXD and TXD pins (no.13 and 14) were connected to the TXD and RXD pins of the ADuC845 to establish the UART link between the two modules. The CTS and RTS pins (no.16 and 15) were connected to each other, and the RESET pin (no.5) was permanently pulled up high using a 1kΩ pull up resistor. A blue LED was connected to the STATUS pin (no.21) of RN-41 so that it could display the mode of operation of the Bluetooth module. It stopped blinking when a connection with an external device had been established. All the pin connections can be seen in figure 4-2:



<u>Fig.4-2:</u> (Bluetooth RN-41 Circuit)

4.1.3. <u>Battery Charging Circuit (using MAX1811)</u>

The MAX1811 was the main IC for our battery charging circuit. The MAX 1811 had 4 input pins consisting of

- SELV (Used for selecting the battery regulation Voltage)
- SELI (Used for selecting battery regulation current)
- IN (Input supply voltage from the USB)
- EN (Used for enabling the device)

To meet our charging requirements the pins SELV and SELI had to be pulled high to set the regulation voltage at 4.2V and the regulation current at 500mA. The enable pin has to be pulled high as well. All three of these pins were pulled up high by connecting them to the USB 5V supply through a 1K resistor. The IC also had a charge indicator pin on which a LED was connected to indicate the stage of charge.

The battery charging circuit consisting of MAX1811 is shown in figure 4-3

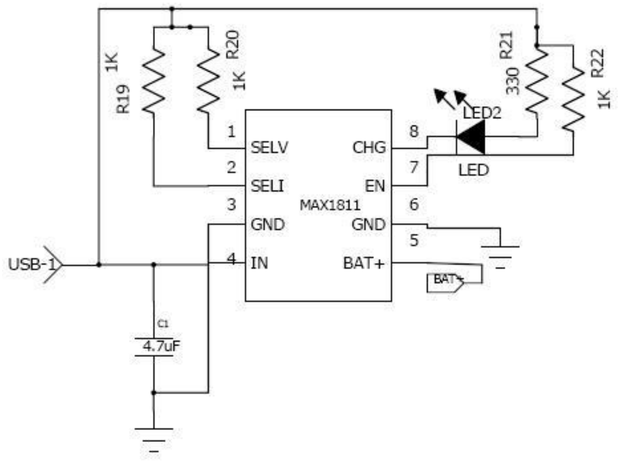


Fig.4-3: (Battery Charging Circuit (MAX1811)

4.1.4. Voltage Regulation (for AV_{DD} and V_{REF})

We used two Analog Devices. (ADI) IC's to provide our system with AV_{DD} and V_{REF} voltages. ADP122 was used to provide the system with the AV_{DD}, and ADR381 was used for the V_{REF} voltage. The positive end of the battery (BAT+) was connected to the input pin (no.1) of ADP122. The V_{OUT} pin (no.5) of ADP122 supplied the whole circuit with AV_{DD} as it was connected to the power plane of the PCB. The power switch for our whole voltmeter device was connected to the enable pin (no.3) of ADP122, which meant that if this IC was disabled (Enable pin = low), then there would be no AV_{DD} in the circuit and hence no power.

ADR381 used to provide the system with the V_{REF} voltage, which was used by both AD7799 and ADuC845. It simply took the AV_{DD} as the voltage input and outputted the V_{REF} , which in our case was 2.50V. Both of these circuits can be seen in figure 4-4:

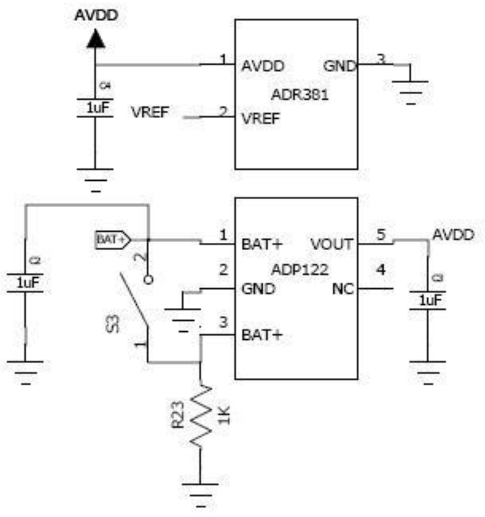


Fig.4-4: (Voltage Regulation (ADP122 and ADR381)

4.2. Final PCB Layout

After finishing the schematic for our product, we started to design the PCB (Printed Circuit Board) layout for our device. We used the ExpressPCB software to make our complete PCB layout. We decided to make a four layered board which would have a top surface, a ground plane, a power plane and a bottom surface.

A 4 layer board option was chosen to provide better noise immunity. Moreover by making a 4 layered board it was easier for us to separate the analog and digital circuitry. The ground plane also helped in protecting the Bluetooth module from any external interference.

The top layer contained the footprints for AD7799, the ADuC845, the multiplexer IC's, the clamping diodes, the LED's, the power switch, the USB charging port and the headphone jack. Its layout can be seen in figure 4-5:

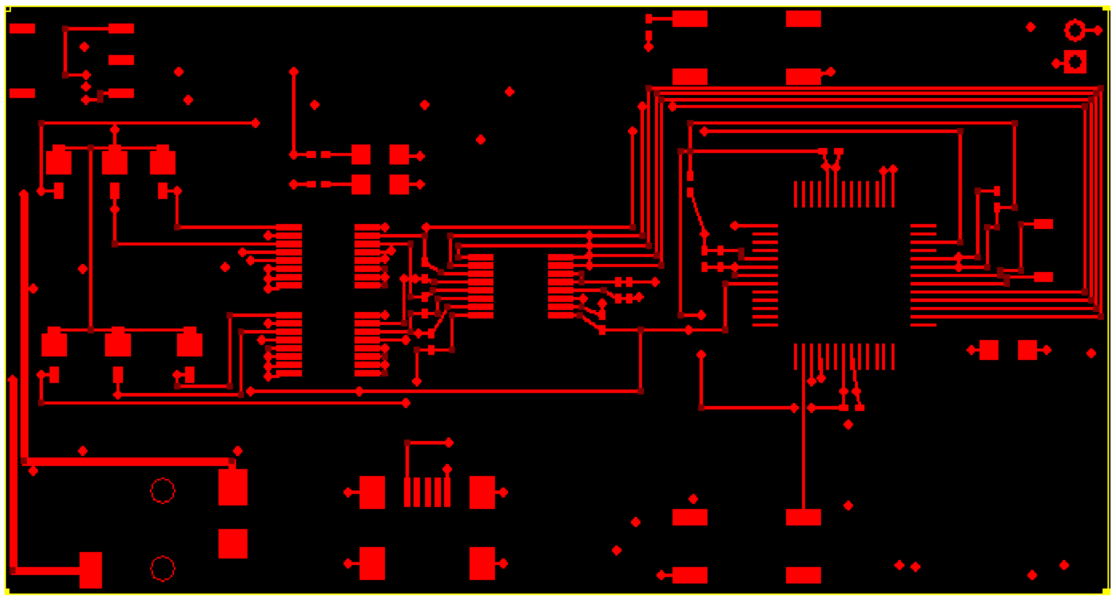


Fig.4-5: (Top layer of the PCB)

The bottom layer contained the footprints for RN-41, the MAX1811, the ADP122 and the external resistor network. It can be seen in figure 4-6:

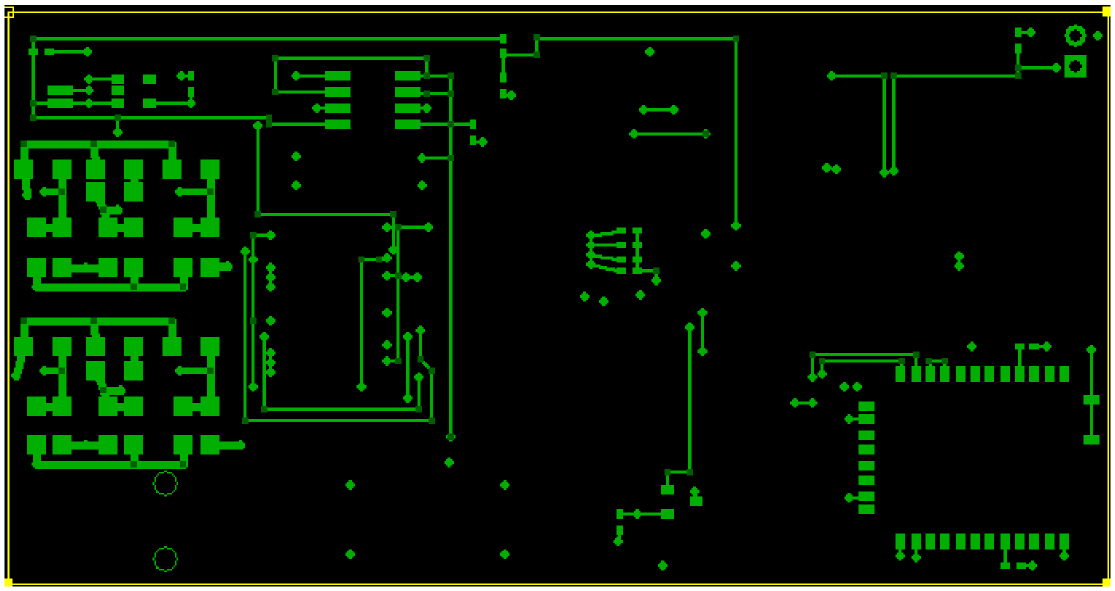


Fig.4-6: (Bottom Layer of the PCB)

The combined PCB layout of the board including both the top and bottom layers can be seen in figure 4-7. All of these PCB layouts can also be found in Appendix C.

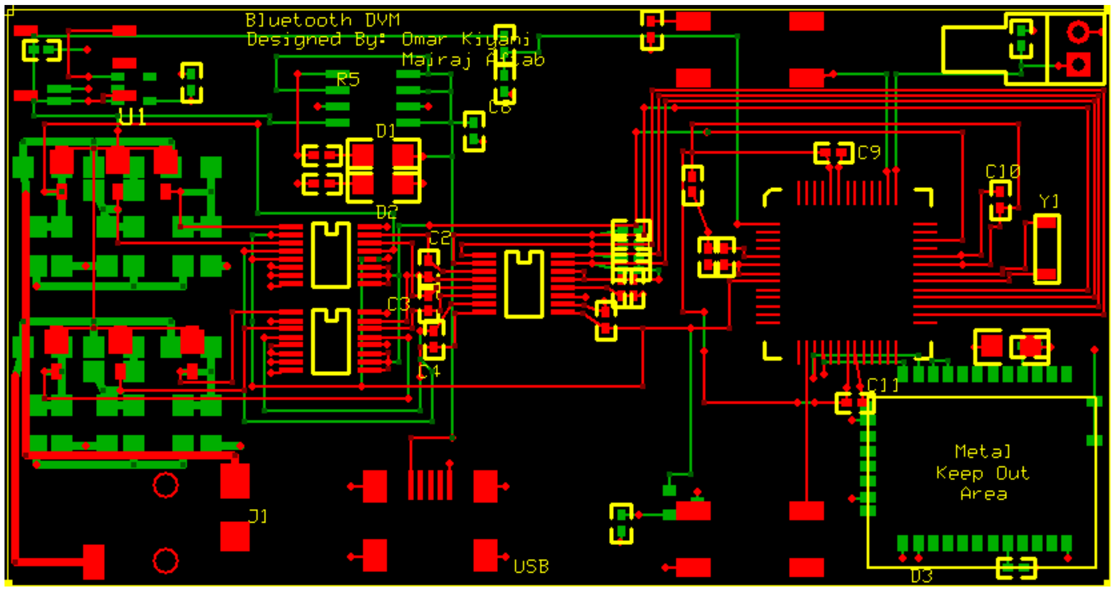
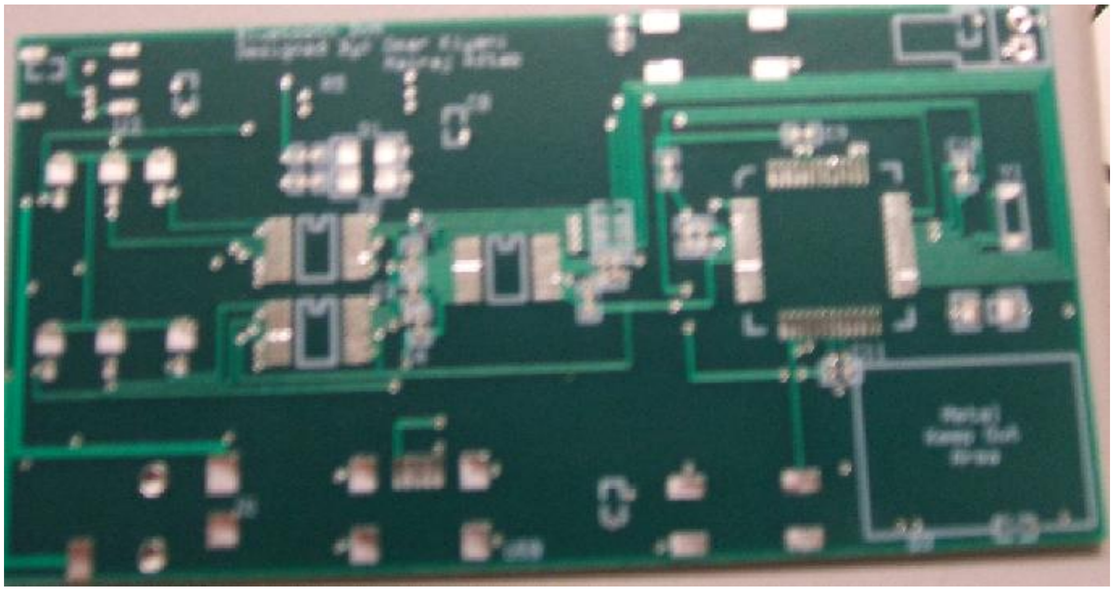


Fig.4-7: (Complete PCB Layout (top and bottom layers)

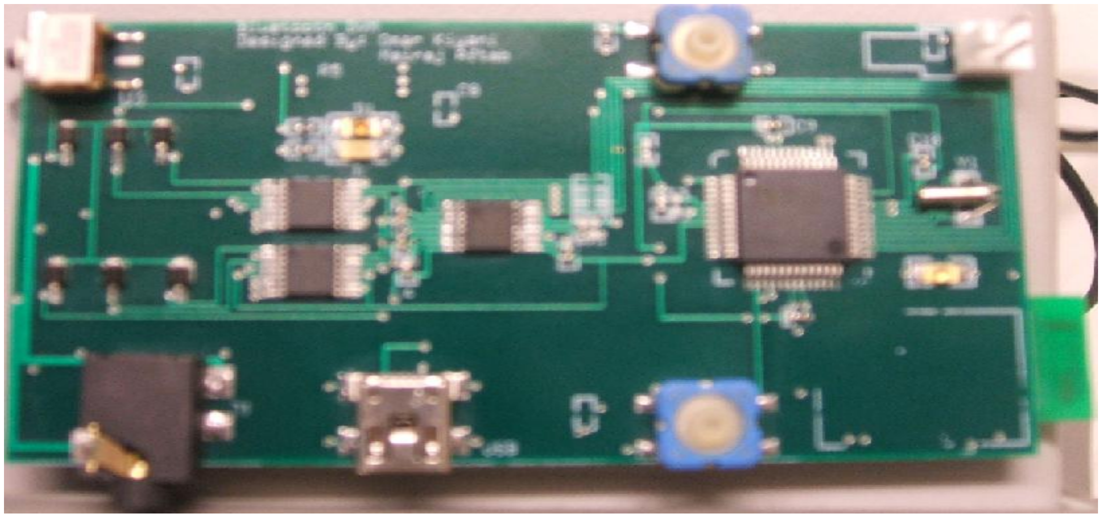


<u>Fig.4-8:</u> (Final PCB - top layer)

4.3. Product Testing

Once the Printed circuit board was received our next task was to solder all the components on to the board. This was a slightly hard task as some of the components we were using were had a 0406 footprint and they can be pretty hard to hand solder.

However working carefully all the parts were soldered on to the board. Figure 4-9 shows the actual photograph of the top view of our PCB after the parts had been soldered on it.



<u>Fig.4-9:</u> (Photograph of the PCB with parts (top view))

Once everything was connected we programmed our on board ADUC845 microprocessor using the EA (Emulation Access) pin present on the programming header on our PCB.

The next part was to test the functionality of our PCB. For this purpose a set of probes were made which had a 1.5mm headphone jack on one end a pair of alligator clips on the other end. The custom probes are shown in figure 4-10

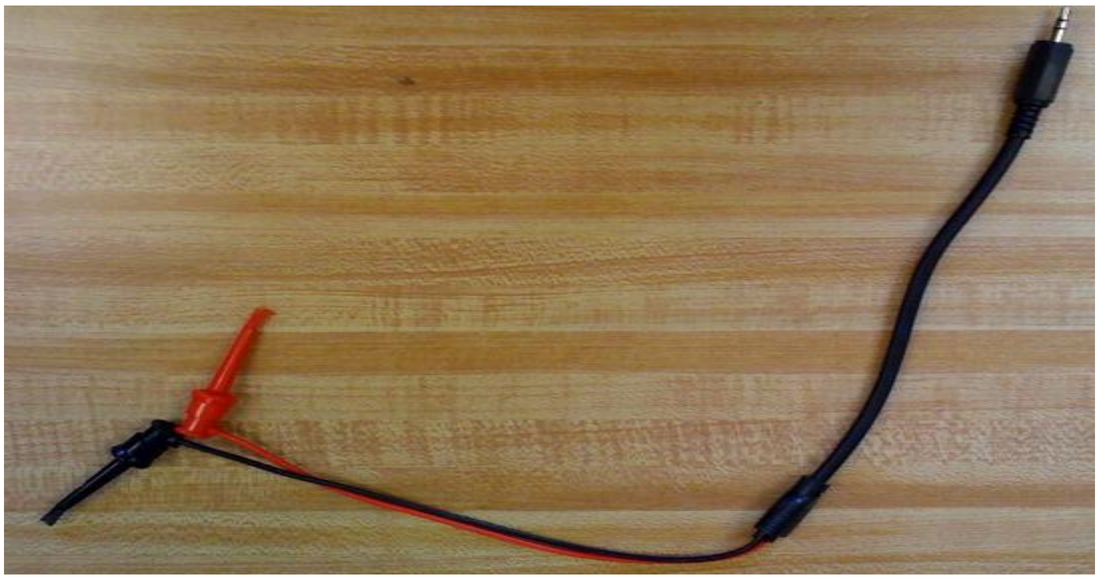
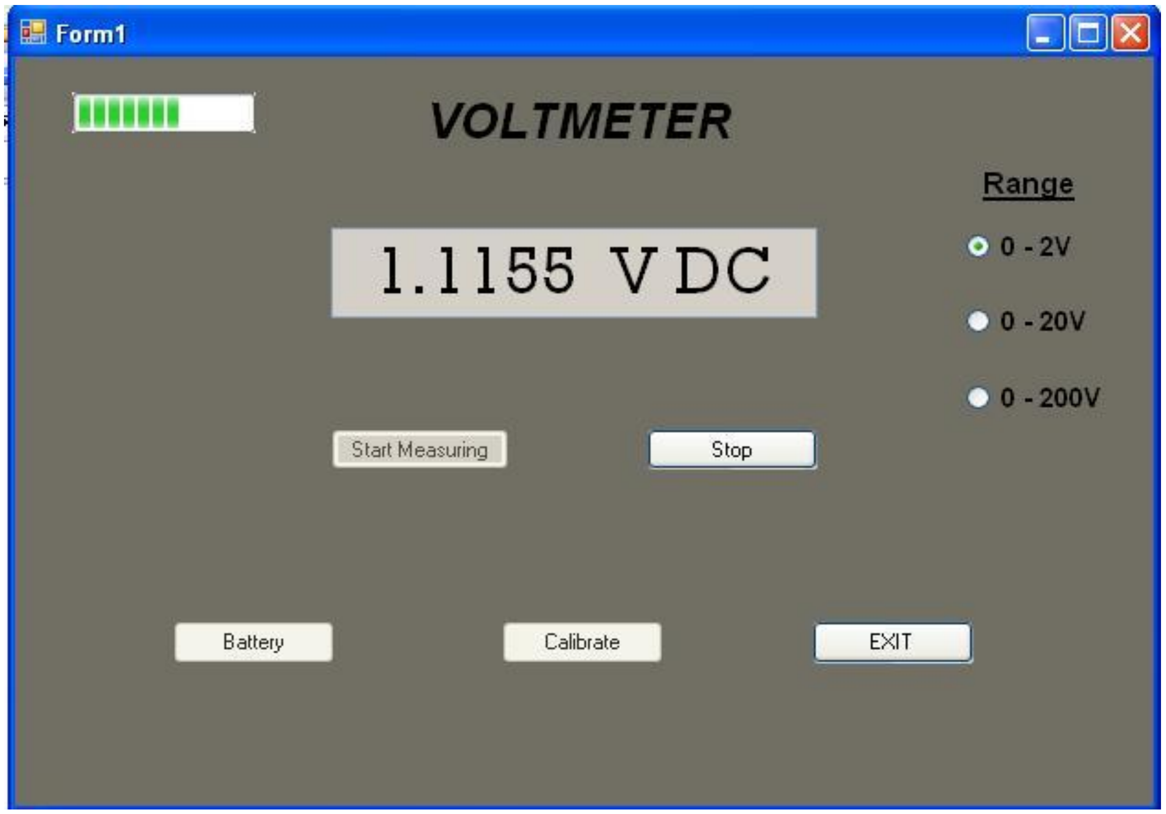


Fig.4-10: (Custom probes)

The male headphone jack was inserted in the female jack present of the PCB. The alligator clips were connected to an external power supply and a varying voltage ranging from 0 – 60 V was applied. The C# application on the PC was run and as soon as the ‘Start’ button was clicked, we started getting 5-digit voltage readings. Figure 4-11 shows a screenshot of the C# application while it was displaying real-time voltage values:



1.1155 VDC

<u>Fig.4-11:</u> (C# Application showing voltage readings)

5. Future Improvements & Recommendations

We at this time have a fully functional 0.1% accurate digital Voltmeter which is capable of transmitting data wirelessly to any external device with Bluetooth transceiver. Moreover we have a functional real-time windows based application that provides the graphical user interface (GUI). By doing this we have Identified promising opportunities for future enhancement and development as a comprehensive product.

There is still room for improvement in our device. Moreover certain features can be added to make it more useful and adaptable for various applications.

We can divide the future work on this product into two categories.

Short run

In the immediate future we are planning to develop an itouch/iphone based DVM application similar to our windows based application. This would then enable users of our device to use our DVM with IPod related products. This would add to the general appeal of the device and also make it more versatile.

Long Run

In the long run similar technology can be used to implement other kind of measurement devices including a Multimeter that would enable users to measure resistance, capacitance, current etc wirelessly. It would also be worth to design an oscilloscope using this technology.

In the long run we can even try to improve the accuracy of our product by using a higher resolution ADC.

6. Conclusion:

The project involved the development of a complete hardware and software solution to a given problem. We had the task of designing a complete product consisting of all the major systems and also the sub systems which were critical for the functionality of the main blocks.

By going through this process we got exposure to product development we learnt how each and every aspect of the product was ultimately linked together and therefore how to go about the process of designing a product from the start.

The major engineering fields we encompassed in designing our product included

(i) Integrated Circuits (IC's) for the Voltmeter, Data Type Conversions, Battery Management etc.

(ii) Wireless Communication using Wi-Fi/ Bluetooth technology.

(iii) C# object-oriented programming language to design real time windows based application to serve as the user interface to the DVM.

Initially after the first couple of meeting we weren't really sure of how to proceed forward with the process of designing such an elaborate system, however through background research we were able to figure out the functionality of DVM's. This helped us to come up with specifications for own DVM and also in making the right choice in choosing AD7799 as the Analog Front – End for data acquisition.

The choice of ADUC845 might have looked like an over kill in the beginning as it included a lot of extra features, however in our application we were able to make use of these features by using the on chip 24 bit ADC for measuring the battery voltage and thus determining the battery life remaining this helped us to avoid using an extra IC for battery life monitoring.

The choice of C# and Microsoft visual studio proved to be feasible as well as its visual interface made it easier to develop the application.

At this time we have a fully functional 0.1% accurate digital Voltmeter on our own customized PCB which is capable of transmitting data wirelessly to any external device with Bluetooth transceiver. Moreover we have a functional real-time windows based application that provides the graphical user interface.

If we were to look at the goals of our project and compare them to the outcomes we can consider it to be highly successful. All of our goals were met if not exceeded. However, there is always room for future improvement and development and we will continue to figure out ways to make our product even more efficient, reliable and more marketable.

Appendix A – Battery Research

Table A-1: Disposable Batteries

Battery P/N	Family	Type	Rechg Y/N	Volume (cm^3)	Weight (grams)	Voltage (V)	Capacity (mAh)	Series R (mohms)	Cost ($)
A23	Alkaline	A	N	2.75	8.0	12	55	200	0.56
CR1620	Lithium	Coin	N	0.40	1.3	3	75	110	0.51
AC10EZ-8	Zinc	Coin	N	0.09	0.3	1.4	91	112	14.05
TLM-1520HP	Lithium	1/2 AA	N	3.35	9.0	3.6	135	250	14.30
CR-1-3N	Lithium	Camera	N	1.17	3.3	3	160	200	1.90
BR-2325	Lithium	Coin	N	1.04	3.2	3	165	100	0.78
357-303HVP	Silver-Oxide	Coin	N	0.57	2.3	1.5	175	103	0.77
BR-2330 /VCN	Lithium	Coin	N	1.25	3.2	3	255	100	1.13
CR2330	Lithium	Coin	N	1.25	3.8	3	265	100	0.78
AC13-4AP	Zinc	Coin	N	0.26	0.8	1.4	280	100	2.25
BR-3032	Lithium	Coin	N	2.26	5.5	3	500	102	1.51
TLH-2450	Lithium	Wafer 24.5mm	N	2.60	8.8	3.6	550	290	3.45
CR2354	Lithium	Coin	N	2.24	2.0	3	560	100	0.68
DURACELL 15035925	Alkaline	AAAA	N	2.29	7.0	1.5	600	325	0.75
CR2450	Lithium	Coin	N	2.36	6.9	3	610	102	0.73
E96BP	Alkaline	AAAA	N	2.20	6.5	1.5	625	300	0.45
AC675-4AP	Zinc	Coin	N	0.57	1.9	1.4	635	102	2.25

CR2	Lithium	Camera	N	5.07	11.0	3	850	140	2.40
TLH-5902/S	Lithium	AA	N	4.16	9.6	3.6	900	150	4.52
LR03PA	Alkaline	AAA	N	3.80	11.0	1.5	1000	140	0.45
BR-1/2AAE5P	Lithium	1/2 AA	N	3..35	8.0	3	1000	700	5.00
TL-5934	Lithium	1/10D	N	5.20	16.3	3.6	1000	300	5.27
CR2477	Lithium	Coin	N	3.63	10.5	3	1000	100	1.57
CR-V3A /1BOEM	Lithium	Camera	N	21.40	39.0	3	1000	160	7.31
EN92	Alkaline	AAA	N	3.80	11.8	1.5	1185	200	0.60
L92	Lithium	AAA	N	3.80	7.6	1.5	1200	120	1.96
BR-2/3A	Lithium	2/3A	N	7.60	13.5	3	1200	200	2.58
BR-2/3A	Lithium	A	N	7.60	13.5	3	1200	80	2.59
L92	Lithium	AAA	N	3.78	7.6	1.5	1200	140	2.16
CR-P2PA /1BOEM	Lithium	Camera	N	23.77	37.0	6	1300	160	6.54
CR123A	Lithium	Camera	N	7.83	17.0	3	1400	160	2.26
BR-2 /3AG	Lithium	A-Type	N	10.33	18.0	3	1450	170	2.94
CR-123APA /1B	Lithium	Camera	N	7.76	17.0	3	1550	160	3.67
CR-123A	Lithium	Camera	N	7.76	17.0	3	1550	160	1.44
TL-5955	Lithium	2/3AA	N	5.20	12.5	3.6	1650	300	4.43
TL-5955/S	Lithium	AA	N	5.53	12.5	3.6	1650	150	2.20
BR-A	Lithium	A	N	10.33	18.0	3	1800	170	4.81
TL-2100/S	Lithium	AA	N	8.34	17.6	3.6	2100	150	3.96

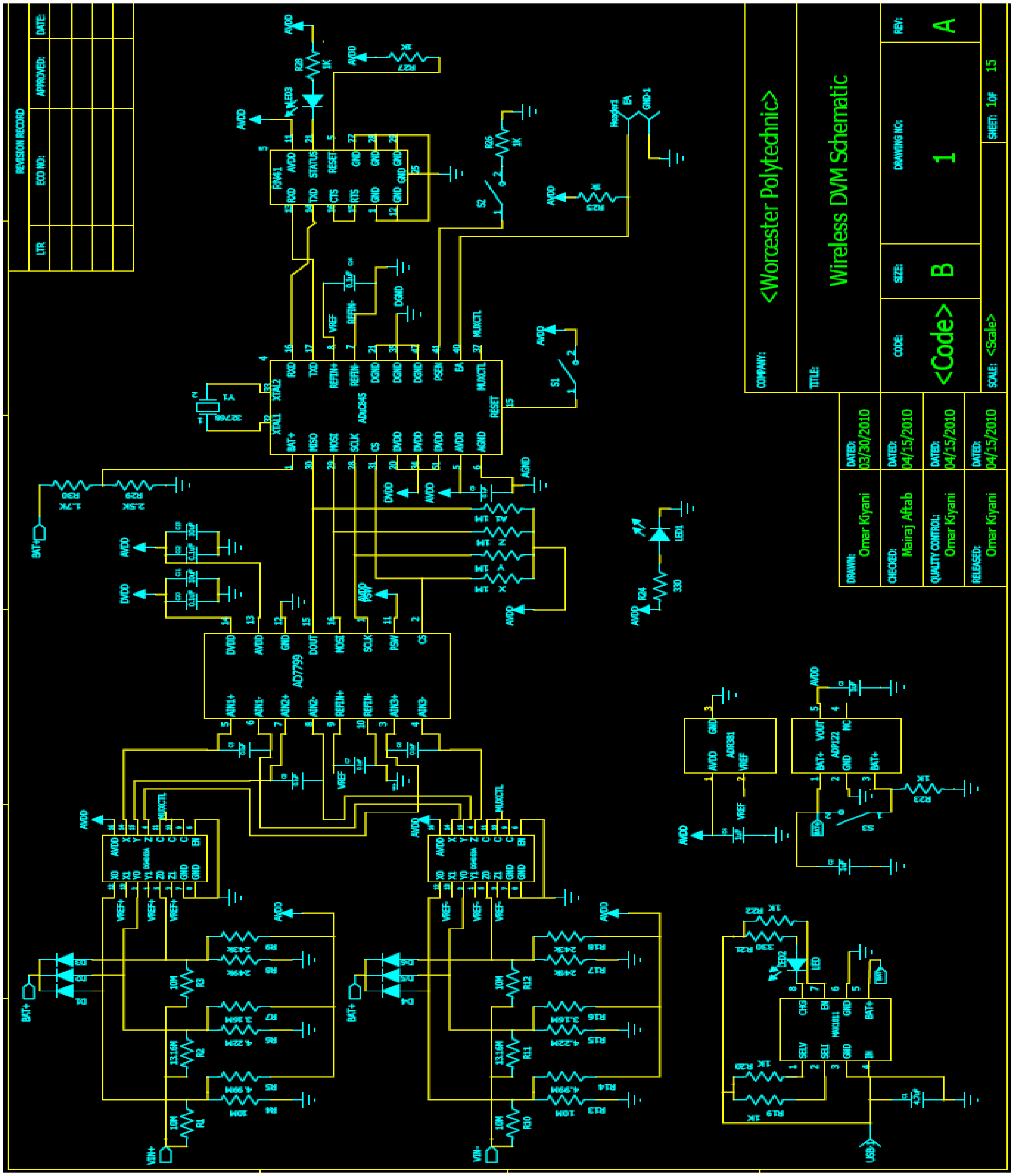
BR-A	Lithium	A	N	10.32	18.0	3	2200	300	5.65
BR-AG	Lithium	A	N	10.33	42.0	3	2200	170	6.21
TL-5903	Lithium	AA	N	8.00	18.0	3.6	2400	400	4.29
EN91	Alkaline	AA	N	8.10	23.0	1.5	2799	250	0.30
ELCRV3BP2	Lithium	Camera	N	20.37	32.0	3	2900	160	13.44
L91	Lithium	AA	N	8.34	14.5	1.5	3000	180	2.20
LR6XWA/B	Alkaline	AA	N	8.10	23.0	1.5	3125	300	0.19
BR-C	Lithium	C	N	26.54	42.0	3	5000	300	7.90
BR-CSSP	Lithium	C	N	24.50	42.0	3	5000	300	10.20
EN93	Alkaline	C	N	26.90	66.0	1.5	8350	200	1.06
TL-5920	Lithium	C	N	26.00	49.5	3.6	8500	250	7.23
LR20XWA	Alkaline	D	N	55.90	141.0	1.5	18000	1100	1.15
TL-5930	Lithium	D	N	57.00	105.0	3.6	19000	300	11.00

Table A-2: Rechargeable Batteries

Battery P/N	**Family**	**Type**	**Rechg Y/N**	**Volume (cm^3)**	**Weight (grams)**	**Voltage (V)**	**Capacity (mAh)**	**Series R (mohms)**	**Cost ($)**
IPod Battery	Lithium	IPod shuffle Gen 3	Y	1.20	2.0	3.7	73	120	11.00
UBP001	Lithium	Cell Phone/IPod	Y	1.40	31.0	3.7	1700	40	8.00
VL-3032 /F2N	Lithium	Coin	Y	1.33	6.2	3.0	100	50	4.95
IPod battery	Lithium	IPod shuffle Gen 2	Y	2.90	4.3	3.7	200	180	7.25
P254	NiCd	AAA	Y	3.90	9.8	1.2	250	13	1.79
IPod battery	Lithium	IPod shuffle Gen 1	Y	3.24	6.0	3.7	250	200	9.20
N-270AA	NiCd	2/3AA	Y	5.20	14.0	1.2	270	15	1.21
IPod Battery	Lithium	IPod Nano Gen 2	Y	4.30	8.0	3.7	400	300	4.95
P-50AAH	NiCd	AA	Y	8.20	21.0	1.2	500	20	0.65
N-500A	NiCd	2/3A	Y	7.70	19.0	1.2	500	9	1.96
P-50AAH/A7B	Ni-Cd	AA	Y	8.26	21.0	1.2	500	19	2.33
HHR-60AAAH	NiMH	AAA	Y	3.85	13.0	1.2	500	14	1.64
PDA-67LI	Lithium	IPod Mini	Y	6.63	13.0	3.7	550	200	9.90
BLI-919-.8	Lithium	Cell phone	Y	8.33	50.0	3.7	800	250	9.40

		battery							
NH12	NiMH	AAA	Y	3.78	13.0	1.2	850	13	1.51
KR-1100AAU	NiCd	AA	Y	8.03	24.0	1.2	1100	19	1.59
N-1250SCRL	Ni-Cd	SC	Y	12.66	43.0	1.2	1200	14	2.76
KR-1500AUL	NiCd	A	Y	9.53	45.0	1.2	1500	16	2.57
HR-AAU	NiMH	AA	Y	8.03	28.0	1.2	1500	20	2.97
HR-3U-1700	NiMH	AA	Y	8.03	27.0	1.2	1600	20	1.62
KR1800SCET	Ni-Cd	SC	Y	16.11	49.0	1.2	1800	15	2.79
N-1900SCRT	Ni-Cd	SC	Y	16.11	54.0	1.2	1900	15	3.60
HHR-210A	NiMH	A	Y	11.35	38.0	1.2	2100	8	3.22
P-230CH	NiCd	C	Y	26.10	78.0	1.2	2300	6	5.96
KR-CH(3.0)	NiCd	C	Y	26.00	78.0	1.2	3000	5.9	6.30
HHR-30SCPY20T	NiMH	SC	Y	17.86	57.0	1.2	3000	15	5.43
P-400DH	NiCd	D	Y	56.20	139.0	1.2	4000	7	7.03
P-500DR	NiCd	D	Y	57.80	145.0	1.2	5000	5	6.53
HHR-650DA08T	NiMH	D	Y	52.17	170.0	1.2	6500	7	13.30

Appendix B – Final Schematic



Appendix C – PCB Layout

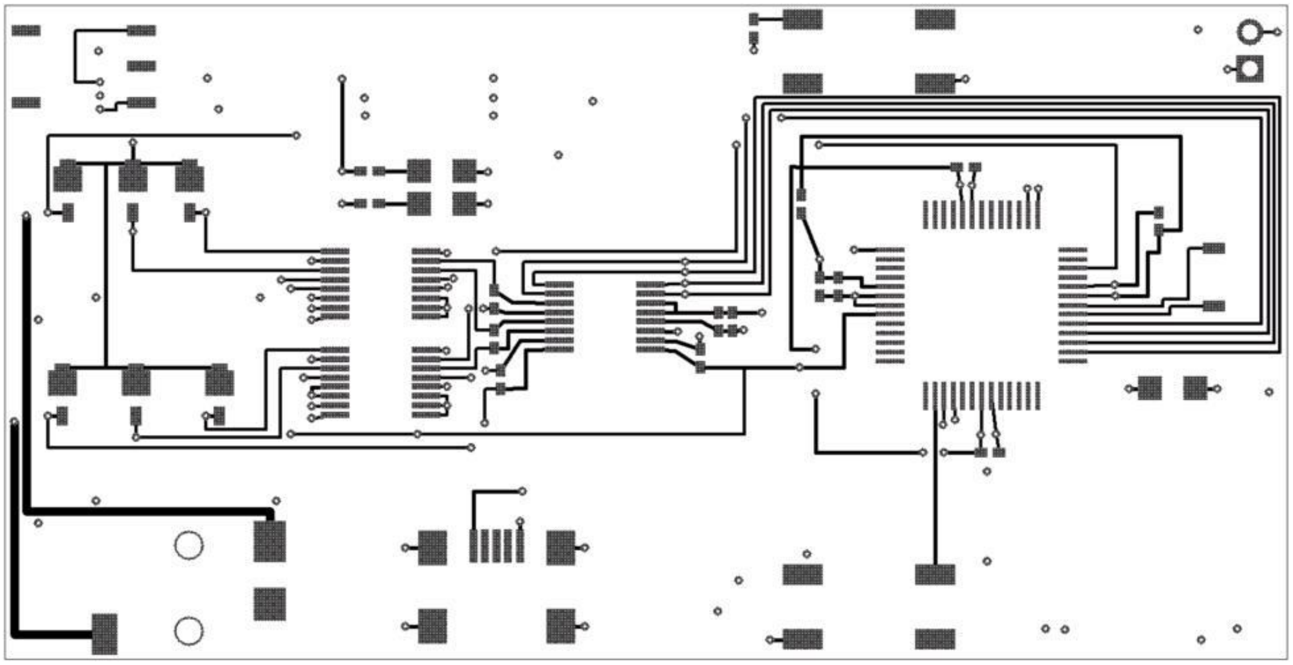


Fig.A-1: (Top Layer of PCB)

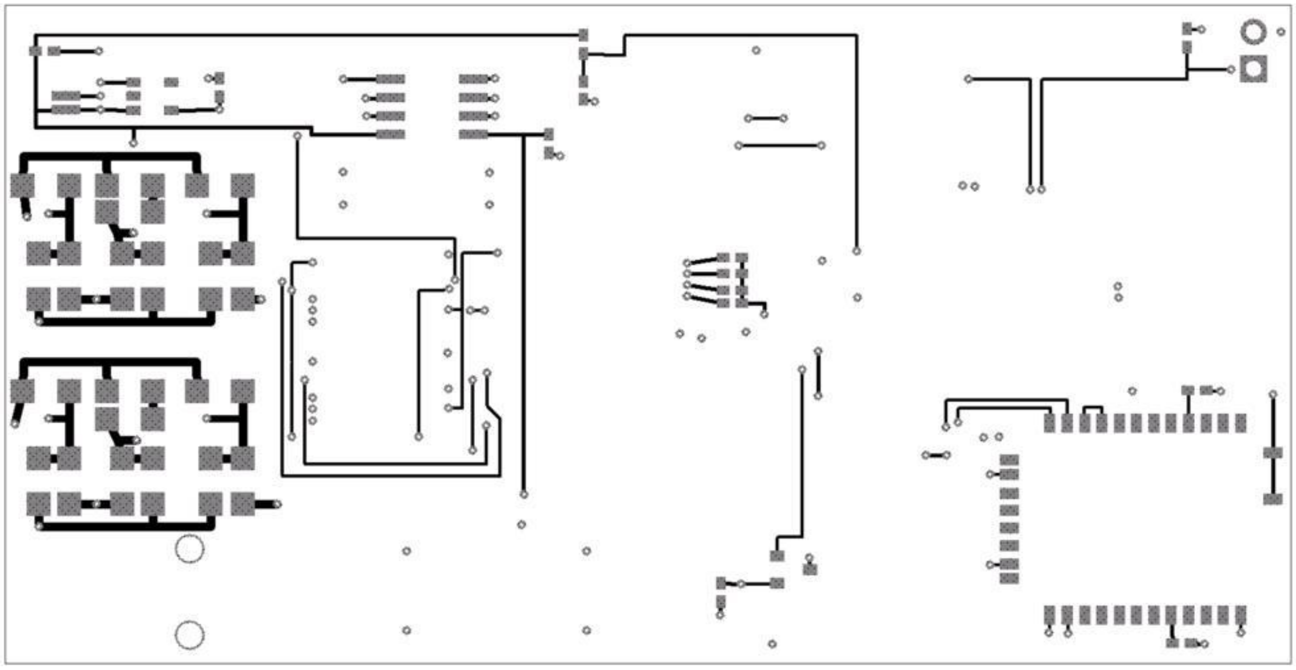


Fig.A-2: (Bottom Layer of PCB)

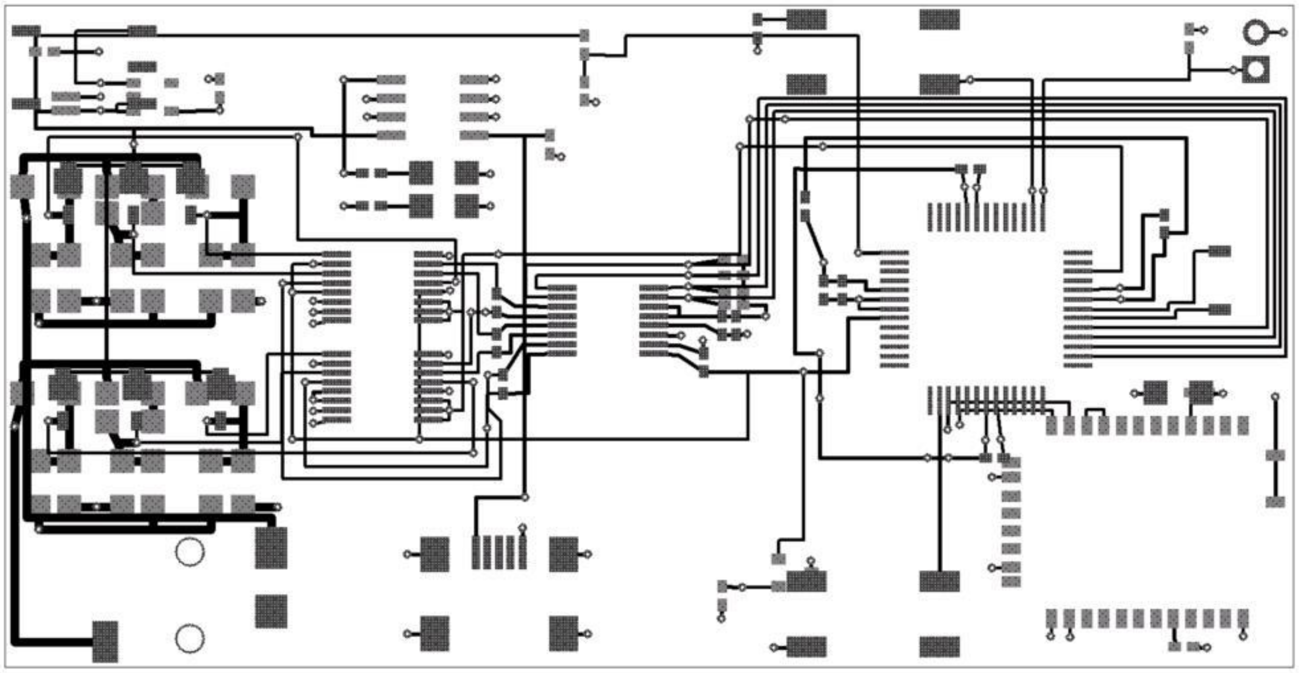
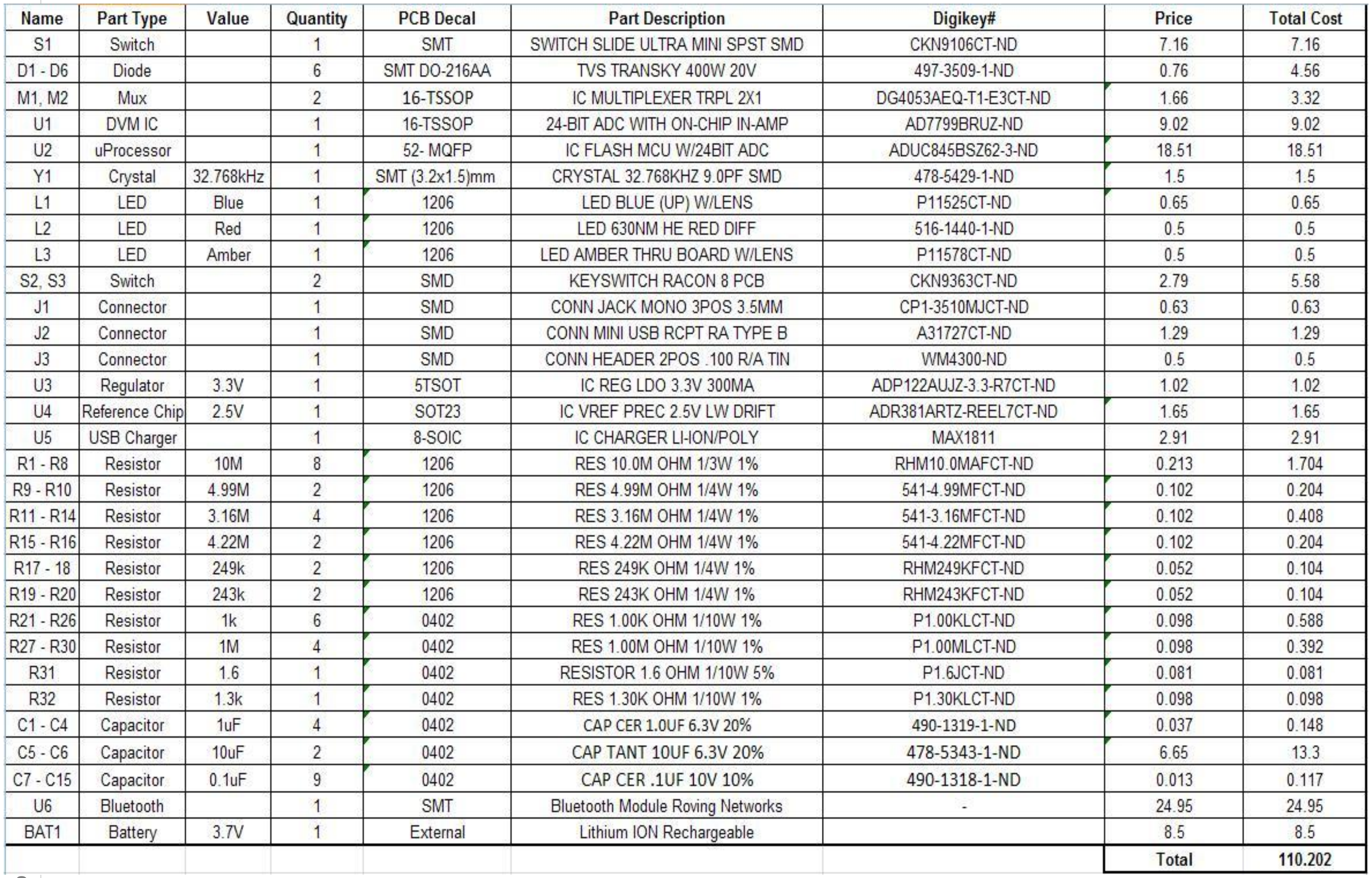


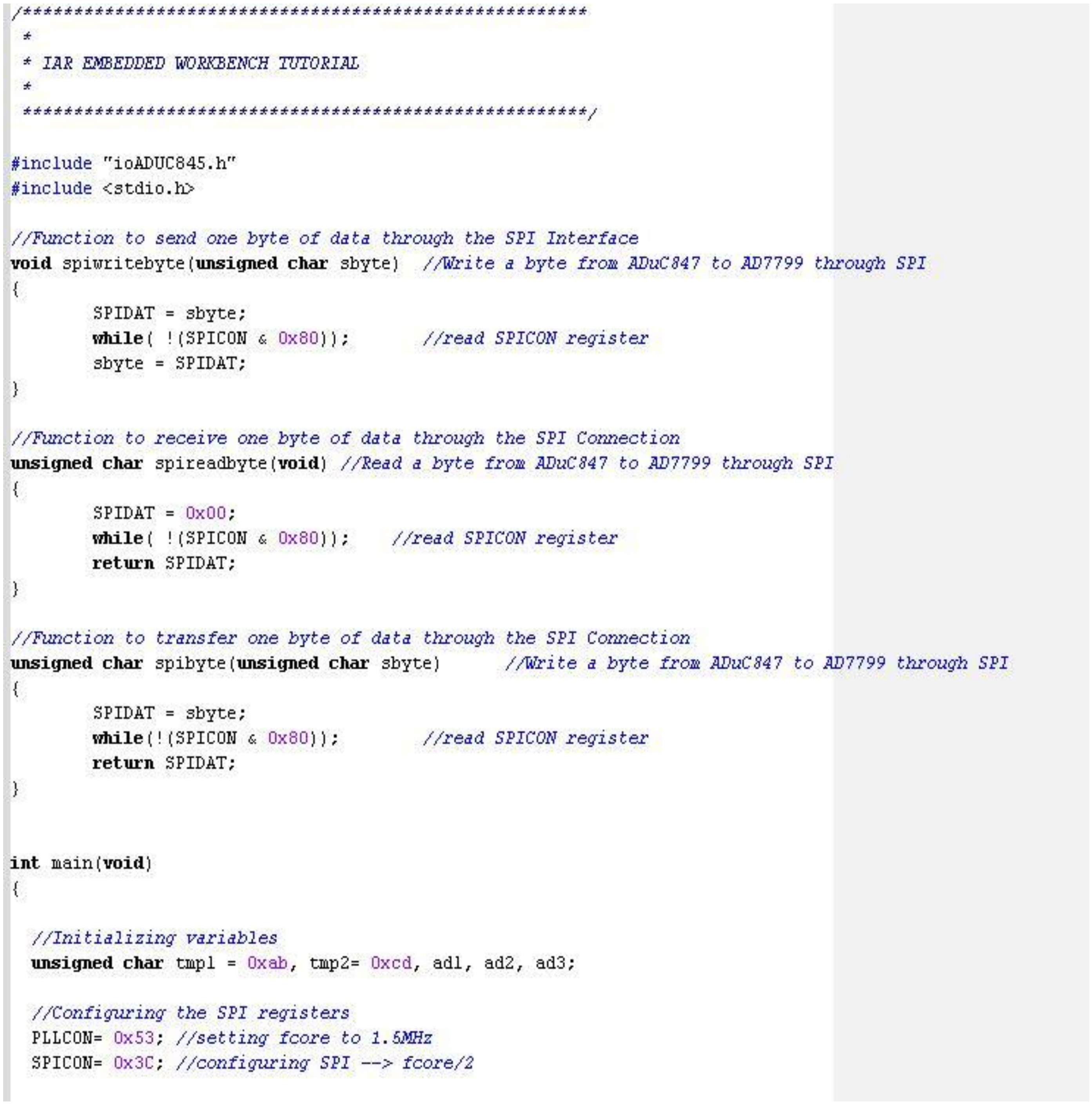
Fig.A-3: (Complete Layout (Top and Bottom Layer) of PCB)

Appendix D – Part List

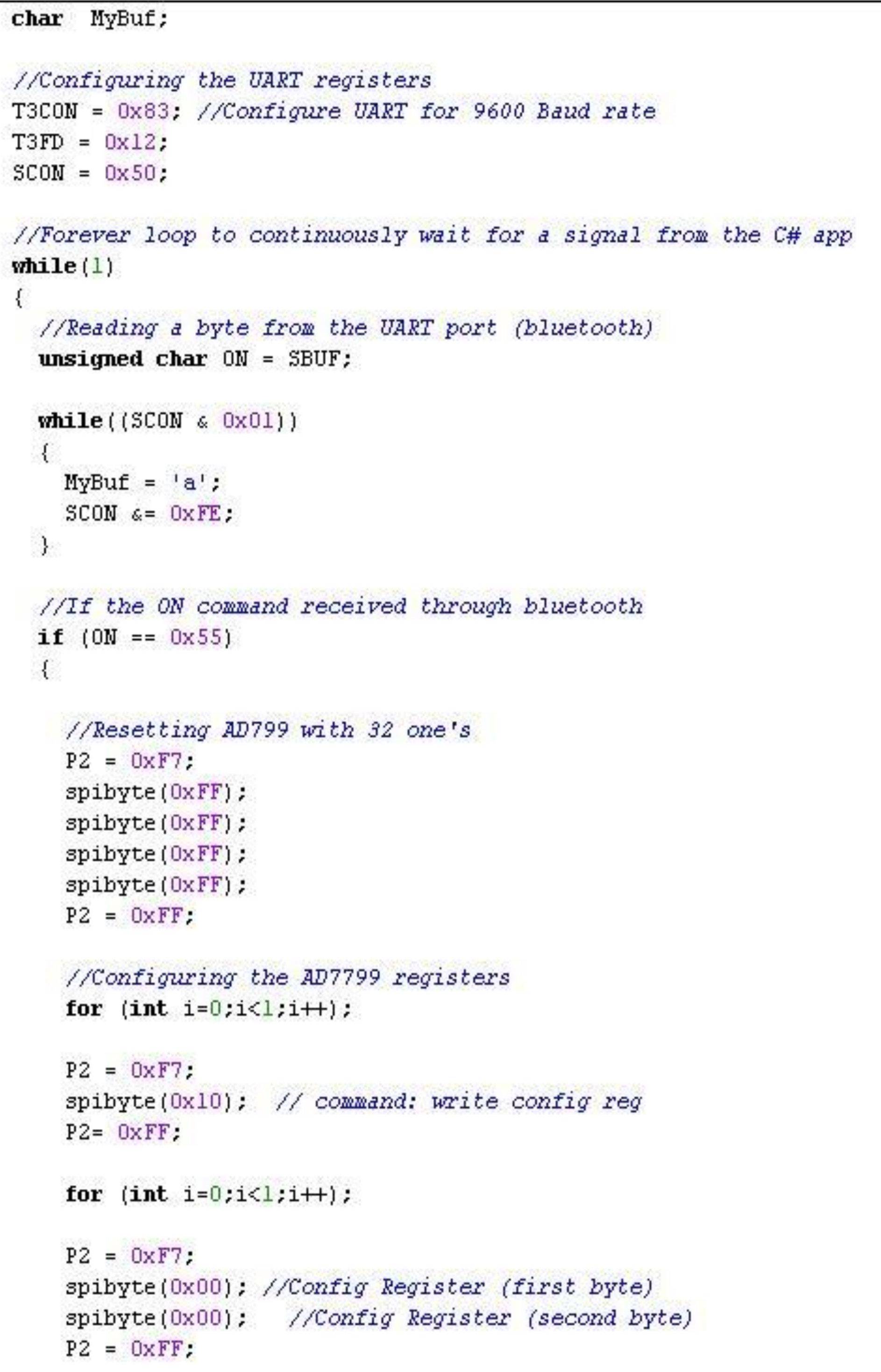


Name	Part Type	Value	Quantity	PCB Decal	Part Description	Digikey#	Price	Total Cost
S1	Switch		1	SMT	SWITCH SLIDE ULTRA MINI SPST SMD	CKN9106CT-ND	7.16	7.16
D1 - D6	Diode		6	SMT DO-216AA	TVS TRANSKY 400W 20V	497-3509-1-ND	0.76	4.56
M1, M2	Mux		2	16-TSSOP	IC MULTIPLEXER TRPL 2X1	DG4053AEQ-T1-E3CT-ND	1.66	3.32
U1	DVM IC		1	16-TSSOP	24-BIT ADC WITH ON-CHIP IN-AMP	AD7799BRUZ-ND	9.02	9.02
U2	uProcessor		1	52- MQFP	IC FLASH MCU W/24BIT ADC	ADUC845BSZ62-3-ND	18.51	18.51
Y1	Crystal	32.768kHz	1	SMT (3.2x1.5)mm	CRYSTAL 32.768KHZ 9.0PF SMD	478-5429-1-ND	1.5	1.5
L1	LED	Blue	1	1206	LED BLUE (UP) W/LENS	P11525CT-ND	0.65	0.65
L2	LED	Red	1	1206	LED 630NM HE RED DIFF	516-1440-1-ND	0.5	0.5
L3	LED	Amber	1	1206	LED AMBER THRU BOARD W/LENS	P11578CT-ND	0.5	0.5
S2, S3	Switch		2	SMD	KEYSWITCH RACON 8 PCB	CKN9363CT-ND	2.79	5.58
J1	Connector		1	SMD	CONN JACK MONO 3POS 3.5MM	CP1-3510MJCT-ND	0.63	0.63
J2	Connector		1	SMD	CONN MINI USB RCPT RA TYPE B	A31727CT-ND	1.29	1.29
J3	Connector		1	SMD	CONN HEADER 2POS .100 R/A TIN	WM4300-ND	0.5	0.5
U3	Regulator	3.3V	1	5TSOT	IC REG LDO 3.3V 300MA	ADP122AUJZ-3.3-R7CT-ND	1.02	1.02
U4	Reference Chip	2.5V	1	SOT23	IC VREF PREC 2.5V LW DRIFT	ADR381ARTZ-REEL7CT-ND	1.65	1.65
U5	USB Charger		1	8-SOIC	IC CHARGER LI-ION/POLY	MAX1811	2.91	2.91
R1 - R8	Resistor	10M	8	1206	RES 10.0M OHM 1/3W 1%	RHM10.0MAFCT-ND	0.213	1.704
R9 - R10	Resistor	4.99M	2	1206	RES 4.99M OHM 1/4W 1%	541-4.99MFCT-ND	0.102	0.204
R11 - R14	Resistor	3.16M	4	1206	RES 3.16M OHM 1/4W 1%	541-3.16MFCT-ND	0.102	0.408
R15 - R16	Resistor	4.22M	2	1206	RES 4.22M OHM 1/4W 1%	541-4.22MFCT-ND	0.102	0.204
R17 - 18	Resistor	249k	2	1206	RES 249K OHM 1/4W 1%	RHM249KFCT-ND	0.052	0.104
R19 - R20	Resistor	243k	2	1206	RES 243K OHM 1/4W 1%	RHM243KFCT-ND	0.052	0.104
R21 - R26	Resistor	1k	6	0402	RES 1.00K OHM 1/10W 1%	P1.00KLCT-ND	0.098	0.588
R27 - R30	Resistor	1M	4	0402	RES 1.00M OHM 1/10W 1%	P1.00MLCT-ND	0.098	0.392
R31	Resistor	1.6	1	0402	RESISTOR 1.6 OHM 1/10W 5%	P1.6JCT-ND	0.081	0.081
R32	Resistor	1.3k	1	0402	RES 1.30K OHM 1/10W 1%	P1.30KLCT-ND	0.098	0.098
C1 - C4	Capacitor	1uF	4	0402	CAP CER 1.0UF 6.3V 20%	490-1319-1-ND	0.037	0.148
C5 - C6	Capacitor	10uF	2	0402	CAP TANT 10UF 6.3V 20%	478-5343-1-ND	6.65	13.3
C7 - C15	Capacitor	0.1uF	9	0402	CAP CER .1UF 10V 10%	490-1318-1-ND	0.013	0.117
U6	Bluetooth		1	SMT	Bluetooth Module Roving Networks	-	24.95	24.95
BAT1	Battery	3.7V	1	External	Lithium ION Rechargeable		8.5	8.5
							Total	110.202

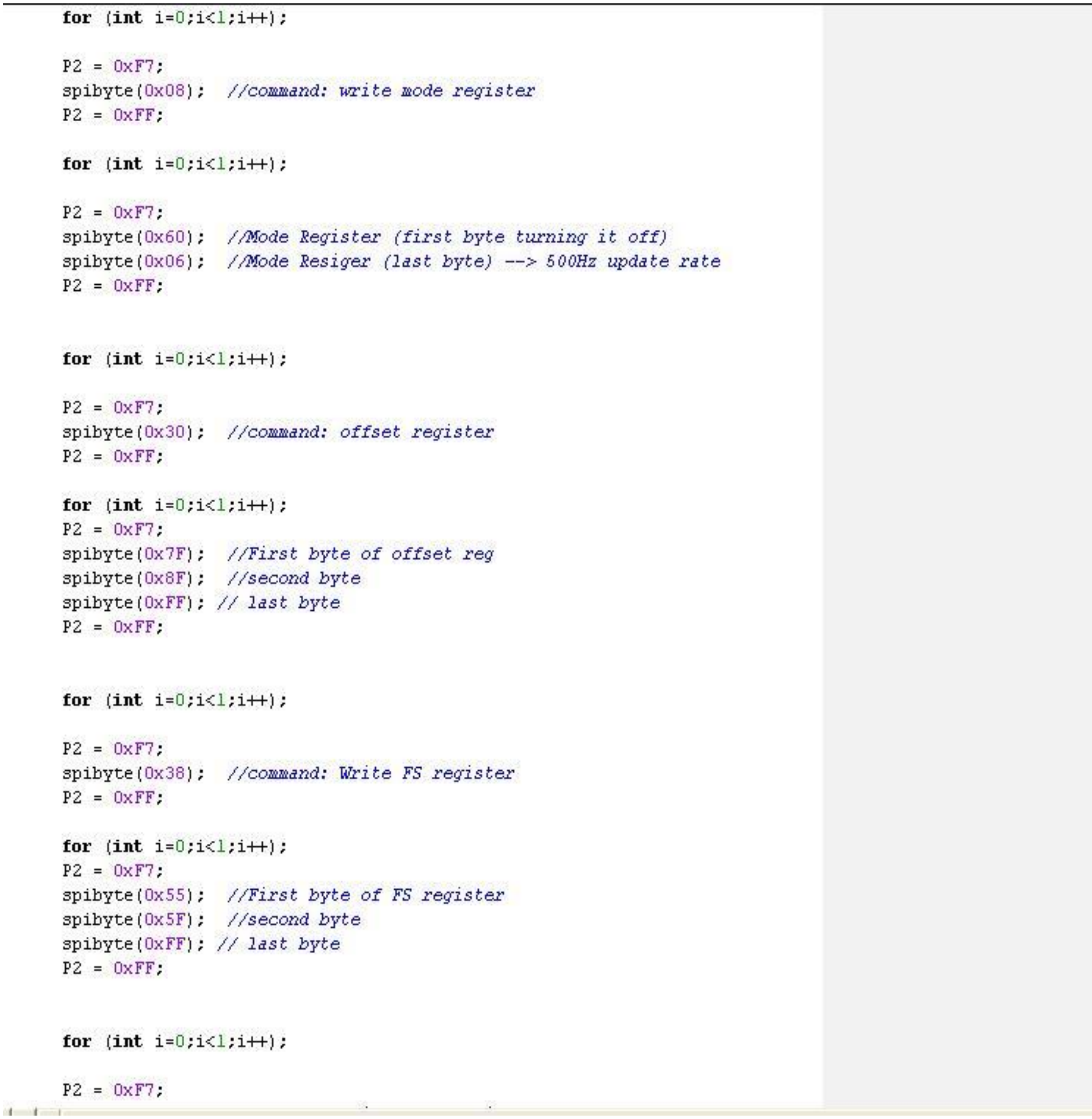
Appendix E – ADuC845 (C code in IAR KickStart)



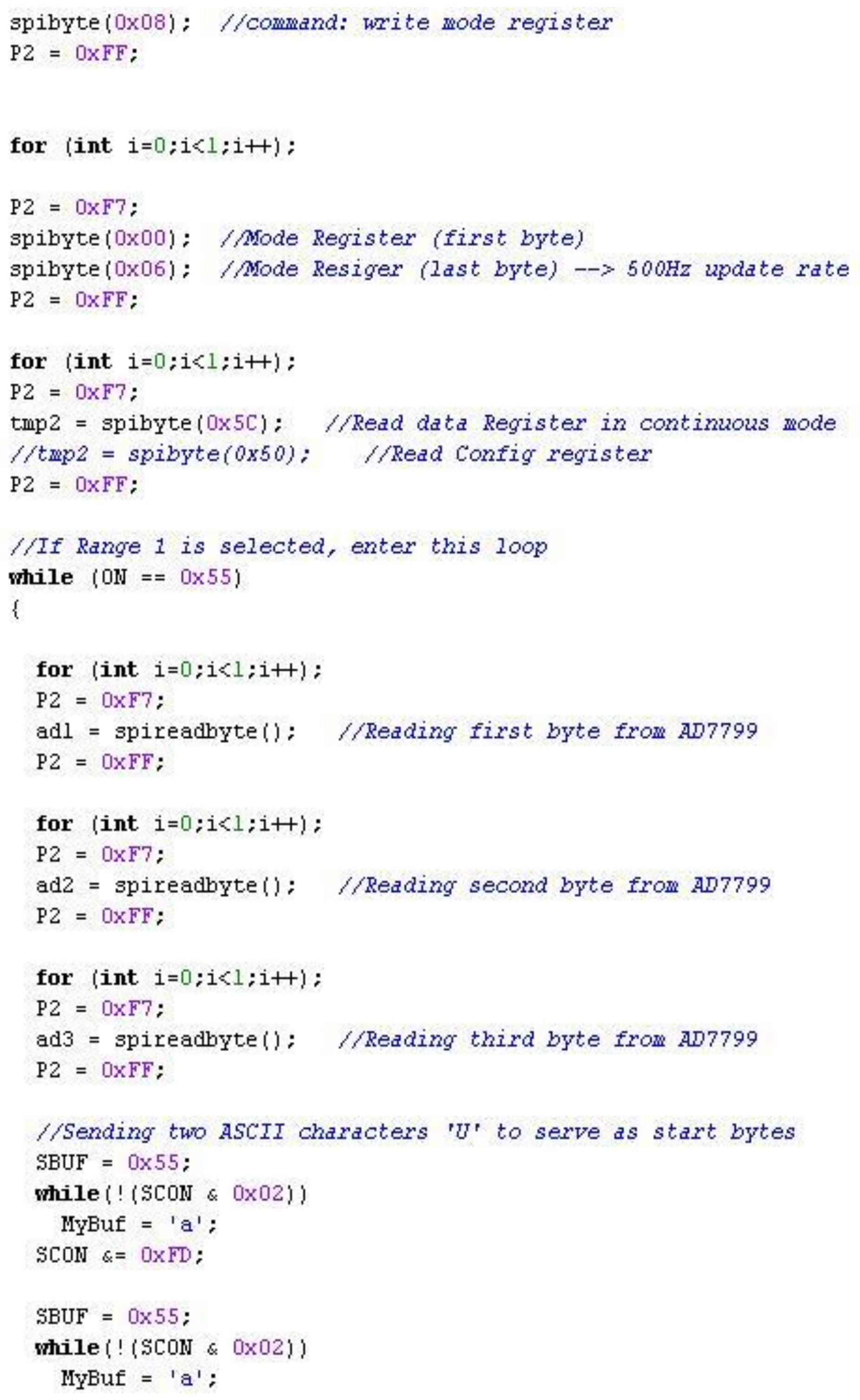
```c
/*******************************************************
 *
 * IAR EMBEDDED WORKBENCH TUTORIAL
 *
 *******************************************************/

#include "ioADUC845.h"
#include <stdio.h>

//Function to send one byte of data through the SPI Interface
void spiwritebyte(unsigned char sbyte)  //Write a byte from ADuC847 to AD7799 through SPI
{
        SPIDAT = sbyte;
        while( !(SPICON & 0x80));       //read SPICON register
        sbyte = SPIDAT;
}

//Function to receive one byte of data through the SPI Connection
unsigned char spireadbyte(void) //Read a byte from ADuC847 to AD7799 through SPI
{
        SPIDAT = 0x00;
        while( !(SPICON & 0x80));    //read SPICON register
        return SPIDAT;
}

//Function to transfer one byte of data through the SPI Connection
unsigned char spibyte(unsigned char sbyte)      //Write a byte from ADuC847 to AD7799 through SPI
{
        SPIDAT = sbyte;
        while(!(SPICON & 0x80));        //read SPICON register
        return SPIDAT;
}


int main(void)
{

  //Initializing variables
  unsigned char tmp1 = 0xab, tmp2= 0xcd, ad1, ad2, ad3;

  //Configuring the SPI registers
  PLLCON= 0x53; //setting fcore to 1.5MHz
  SPICON= 0x3C; //configuring SPI --> fcore/2
```



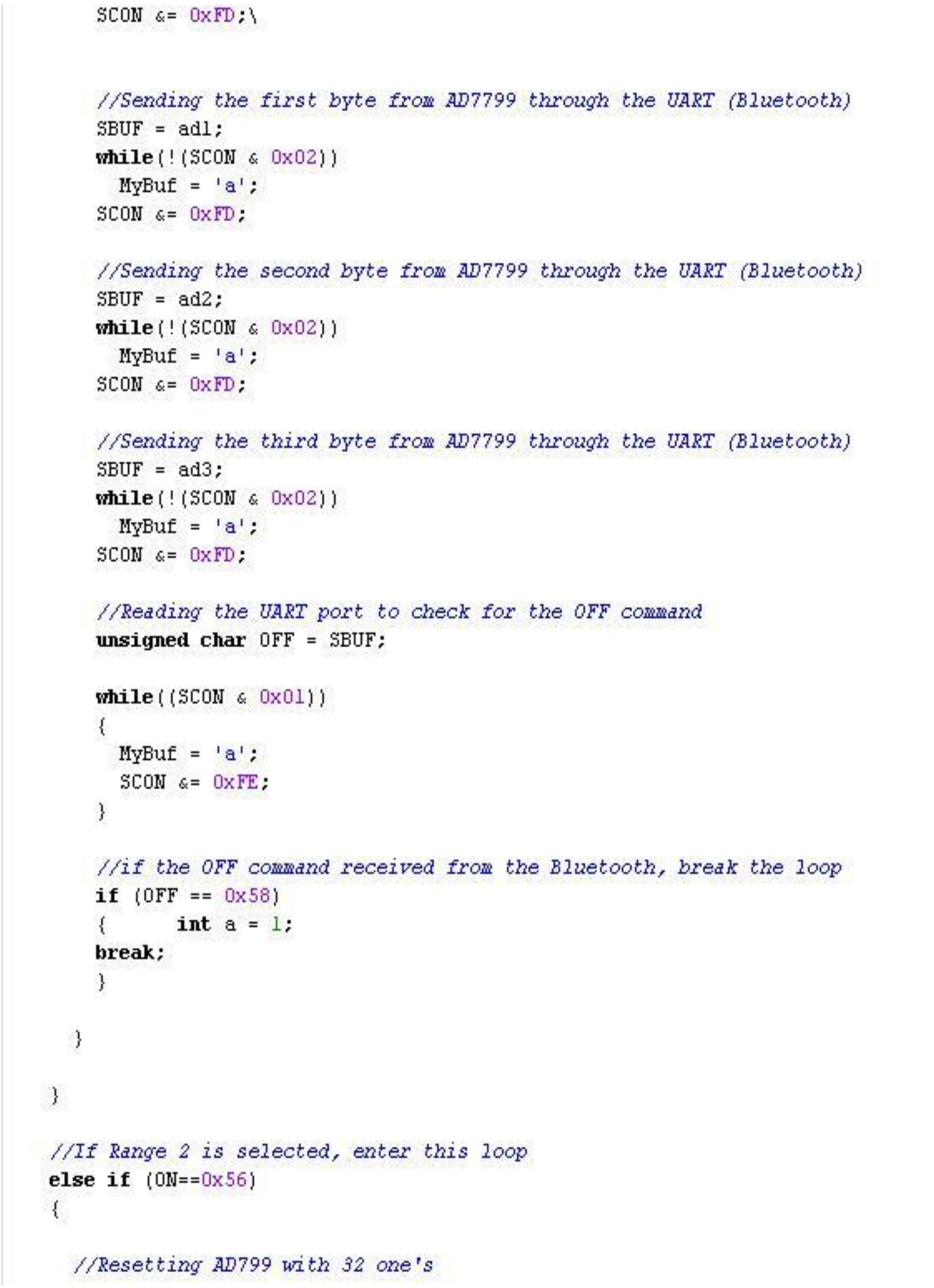
```
char  MyBuf;

//Configuring the UART registers
T3CON = 0x83; //Configure UART for 9600 Baud rate
T3FD = 0x12;
SCON = 0x50;

//Forever loop to continuously wait for a signal from the C# app
while(1)
{
  //Reading a byte from the UART port (bluetooth)
  unsigned char ON = SBUF;

  while((SCON & 0x01))
  {
    MyBuf = 'a';
    SCON &= 0xFE;
  }

  //If the ON command received through bluetooth
  if (ON == 0x55)
  {

    //Resetting AD799 with 32 one's
    P2 = 0xF7;
    spibyte(0xFF);
    spibyte(0xFF);
    spibyte(0xFF);
    spibyte(0xFF);
    P2 = 0xFF;

    //Configuring the AD7799 registers
    for (int i=0;i<1;i++);

    P2 = 0xF7;
    spibyte(0x10);  // command: write config reg
    P2= 0xFF;

    for (int i=0;i<1;i++);

    P2 = 0xF7;
    spibyte(0x00); //Config Register (first byte)
    spibyte(0x00);   //Config Register (second byte)
    P2 = 0xFF;
```



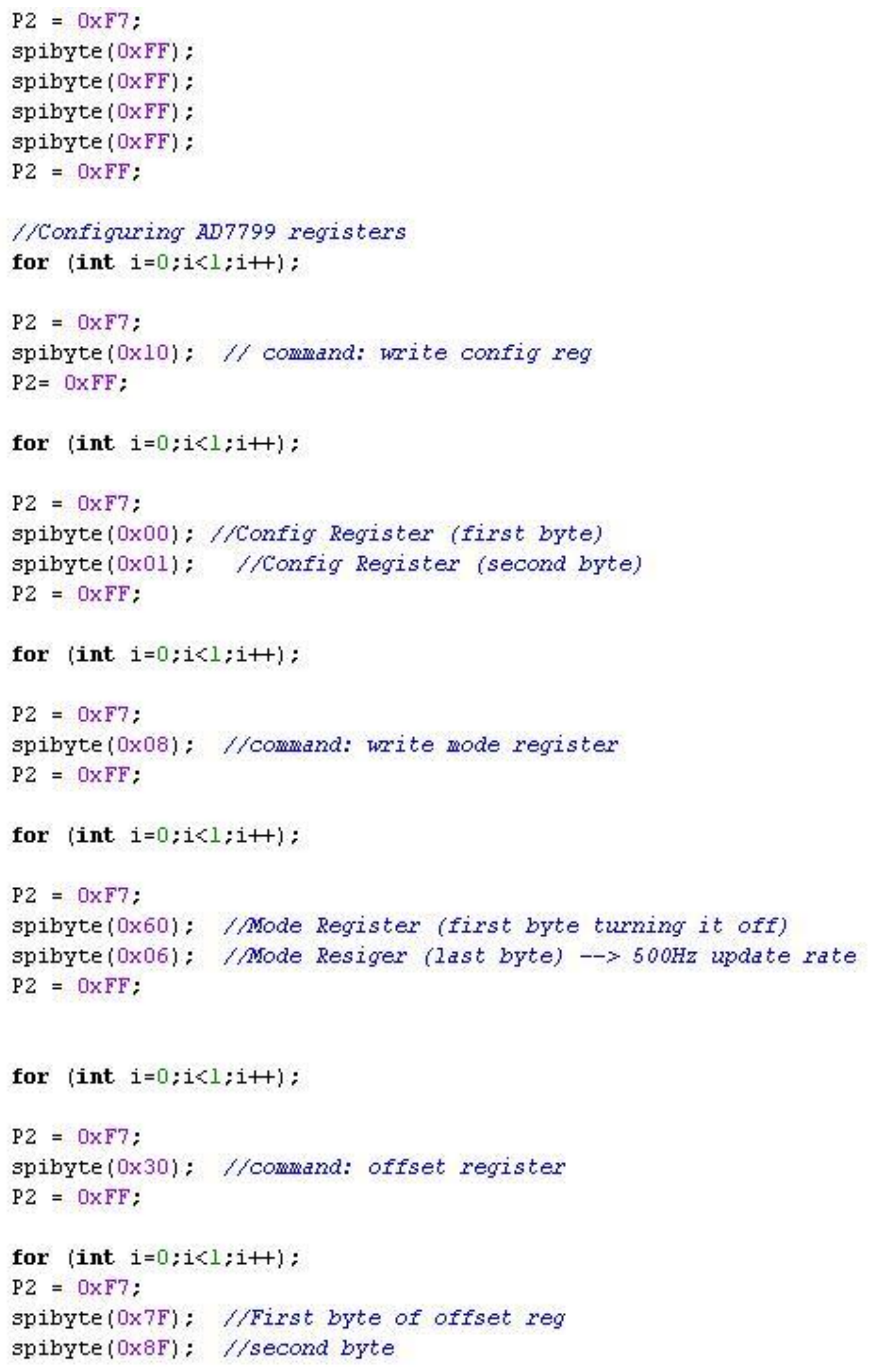
```
for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x08);  //command: write mode register
P2 = 0xFF;

for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x60);  //Mode Register (first byte turning it off)
spibyte(0x06);  //Mode Resiger (last byte) --> 500Hz update rate
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x30);  //command: offset register
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
spibyte(0x7F);  //First byte of offset reg
spibyte(0x8F);  //second byte
spibyte(0xFF); // last byte
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x38);  //command: Write FS register
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
spibyte(0x55);  //First byte of FS register
spibyte(0x5F);  //second byte
spibyte(0xFF); // last byte
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
```



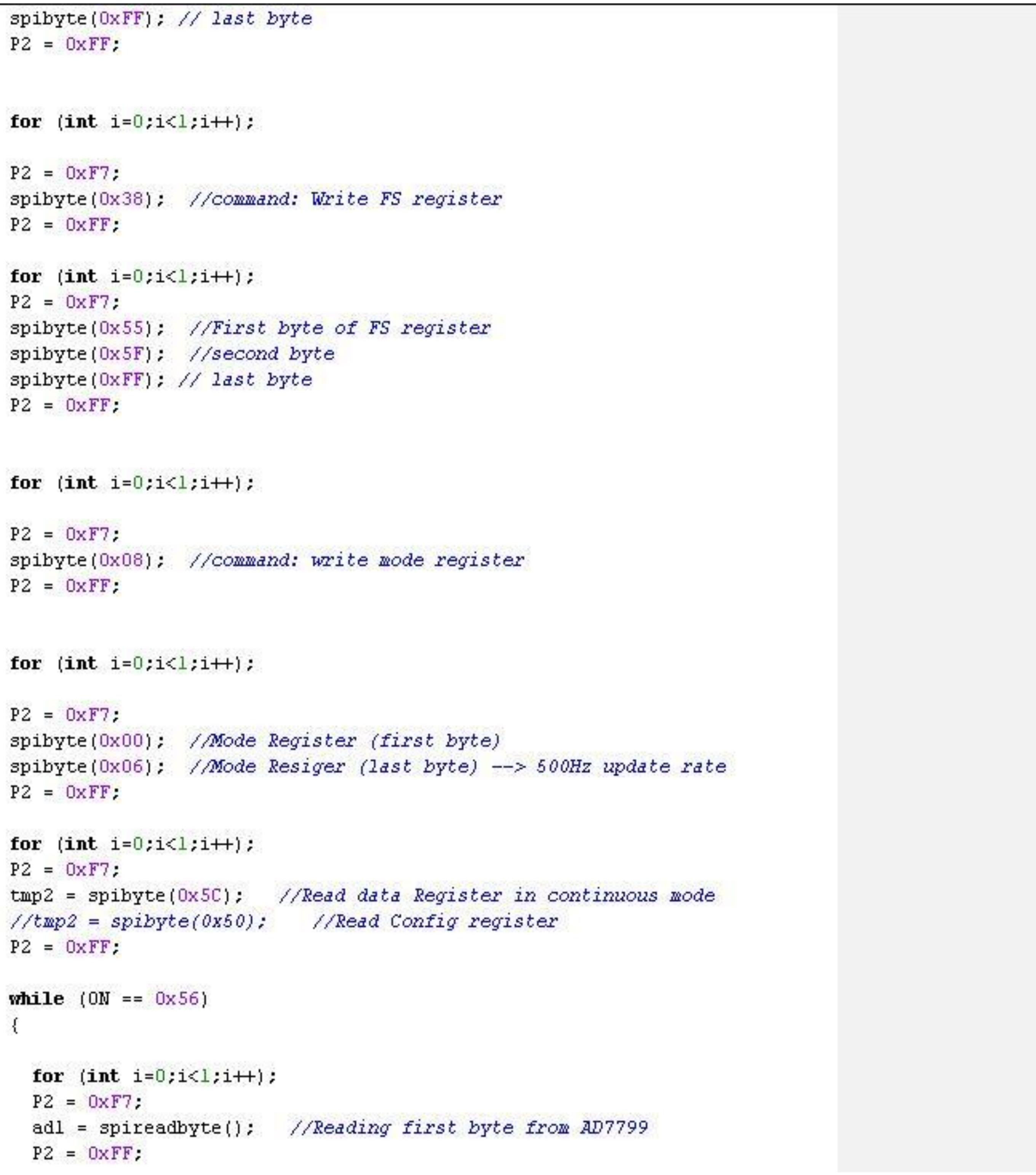
```
spibyte(0x08);  //command: write mode register
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x00);  //Mode Register (first byte)
spibyte(0x06);  //Mode Resiger (last byte) --> 500Hz update rate
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
tmp2 = spibyte(0x5C);   //Read data Register in continuous mode
//tmp2 = spibyte(0x50);    //Read Config register
P2 = 0xFF;

//If Range 1 is selected, enter this loop
while (ON == 0x55)
{

  for (int i=0;i<1;i++);
  P2 = 0xF7;
  ad1 = spireadbyte();   //Reading first byte from AD7799
  P2 = 0xFF;

  for (int i=0;i<1;i++);
  P2 = 0xF7;
  ad2 = spireadbyte();   //Reading second byte from AD7799
  P2 = 0xFF;

  for (int i=0;i<1;i++);
  P2 = 0xF7;
  ad3 = spireadbyte();   //Reading third byte from AD7799
  P2 = 0xFF;

  //Sending two ASCII characters 'U' to serve as start bytes
  SBUF = 0x55;
  while(!(SCON & 0x02))
    MyBuf = 'a';
  SCON &= 0xFD;

  SBUF = 0x55;
  while(!(SCON & 0x02))
    MyBuf = 'a';
```



```
      SCON &= 0xFD;\


      //Sending the first byte from AD7799 through the UART (Bluetooth)
      SBUF = ad1;
      while(!(SCON & 0x02))
        MyBuf = 'a';
      SCON &= 0xFD;

      //Sending the second byte from AD7799 through the UART (Bluetooth)
      SBUF = ad2;
      while(!(SCON & 0x02))
        MyBuf = 'a';
      SCON &= 0xFD;

      //Sending the third byte from AD7799 through the UART (Bluetooth)
      SBUF = ad3;
      while(!(SCON & 0x02))
        MyBuf = 'a';
      SCON &= 0xFD;

      //Reading the UART port to check for the OFF command
      unsigned char OFF = SBUF;

      while((SCON & 0x01))
      {
        MyBuf = 'a';
        SCON &= 0xFE;
      }

      //if the OFF command received from the Bluetooth, break the loop
      if (OFF == 0x58)
      {       int a = 1;
      break;
      }

    }

  }

  //If Range 2 is selected, enter this loop
  else if (ON==0x56)
  {

    //Resetting AD799 with 32 one's
```

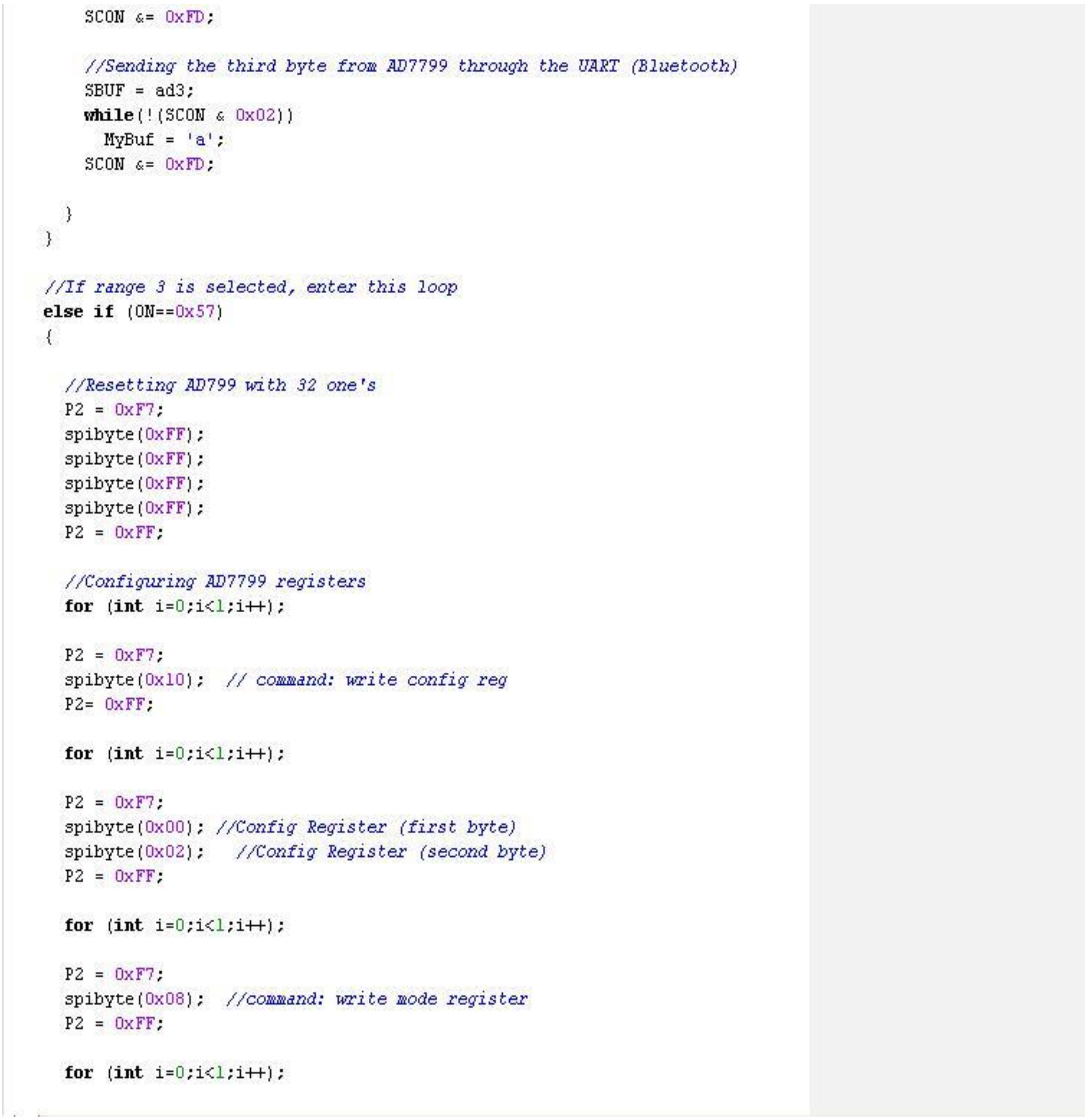


```
P2 = 0xF7;
spibyte(0xFF);
spibyte(0xFF);
spibyte(0xFF);
spibyte(0xFF);
P2 = 0xFF;

//Configuring AD7799 registers
for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x10);  // command: write config reg
P2= 0xFF;

for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x00); //Config Register (first byte)
spibyte(0x01);   //Config Register (second byte)
P2 = 0xFF;

for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x08);  //command: write mode register
P2 = 0xFF;

for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x60);  //Mode Register (first byte turning it off)
spibyte(0x06);  //Mode Resiger (last byte) --> 500Hz update rate
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x30);  //command: offset register
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
spibyte(0x7F);  //First byte of offset reg
spibyte(0x8F);  //second byte
```



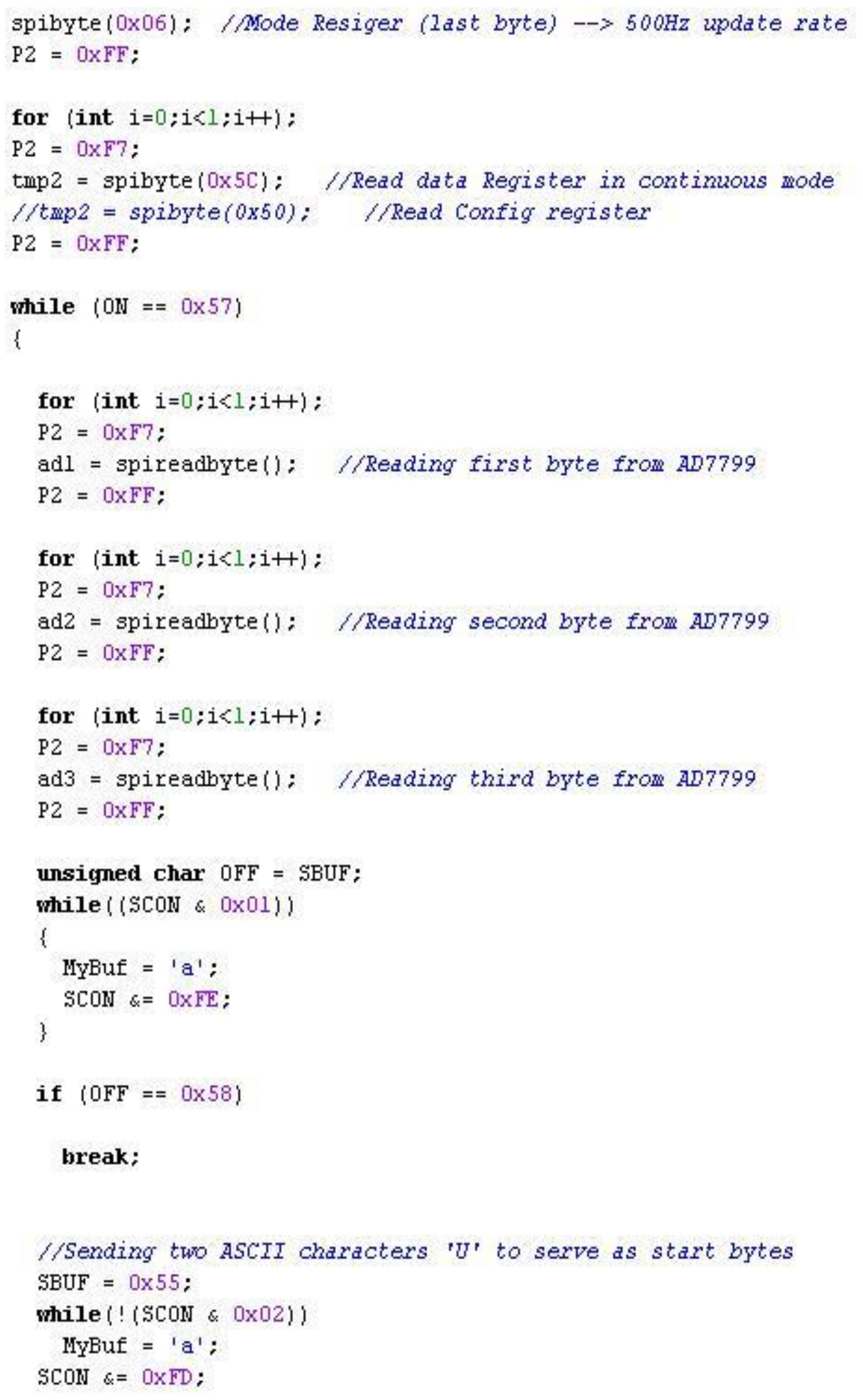
```
spibyte(0xFF); // last byte
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x38);  //command: Write FS register
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
spibyte(0x55);  //First byte of FS register
spibyte(0x5F);  //second byte
spibyte(0xFF); // last byte
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x08);  //command: write mode register
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x00);  //Mode Register (first byte)
spibyte(0x06);  //Mode Resiger (last byte) --> 500Hz update rate
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
tmp2 = spibyte(0x5C);   //Read data Register in continuous mode
//tmp2 = spibyte(0x50);    //Read Config register
P2 = 0xFF;

while (ON == 0x56)
{

  for (int i=0;i<1;i++);
  P2 = 0xF7;
  ad1 = spireadbyte();   //Reading first byte from AD7799
  P2 = 0xFF;
```

```
for (int i=0;i<1;i++);
P2 = 0xF7;
ad2 = spireadbyte();    //Reading second byte from AD7799
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
ad3 = spireadbyte();    //Reading third byte from AD7799
P2 = 0xFF;

unsigned char OFF = SBUF;
while((SCON & 0x01))
{
  MyBuf = 'a';
  SCON &= 0xFE;
}

if (OFF == 0x58)
{
  break;
}

//Sending two ASCII characters 'U' to serve as start bytes
SBUF = 0x55;
while(!(SCON & 0x02))
  MyBuf = 'a';
SCON &= 0xFD;

SBUF = 0x55;
while(!(SCON & 0x02))
  MyBuf = 'a';
SCON &= 0xFD;\


//Sending the first byte from AD7799 through the UART (Bluetooth)
SBUF = ad1;
while(!(SCON & 0x02))
  MyBuf = 'a';
SCON &= 0xFD;

//Sending the second byte from AD7799 through the UART (Bluetooth)
SBUF = ad2;
while(!(SCON & 0x02))
  MyBuf = 'a';
```



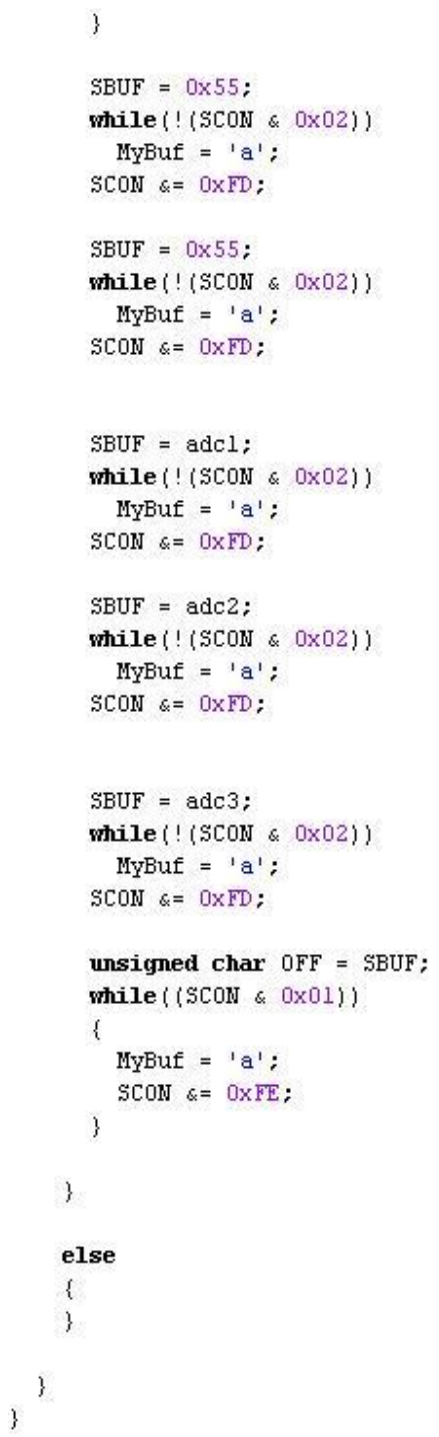
```
    SCON &= 0xFD;

    //Sending the third byte from AD7799 through the UART (Bluetooth)
    SBUF = ad3;
    while(!(SCON & 0x02))
      MyBuf = 'a';
    SCON &= 0xFD;

  }
}

//If range 3 is selected, enter this loop
else if (ON==0x57)
{

  //Resetting AD799 with 32 one's
  P2 = 0xF7;
  spibyte(0xFF);
  spibyte(0xFF);
  spibyte(0xFF);
  spibyte(0xFF);
  P2 = 0xFF;

  //Configuring AD7799 registers
  for (int i=0;i<1;i++);

  P2 = 0xF7;
  spibyte(0x10);  // command: write config reg
  P2= 0xFF;

  for (int i=0;i<1;i++);

  P2 = 0xF7;
  spibyte(0x00); //Config Register (first byte)
  spibyte(0x02);   //Config Register (second byte)
  P2 = 0xFF;

  for (int i=0;i<1;i++);

  P2 = 0xF7;
  spibyte(0x08);  //command: write mode register
  P2 = 0xFF;

  for (int i=0;i<1;i++);
```

```
P2 = 0xF7;
spibyte(0x60);  //Mode Register (first byte turning it off)
spibyte(0x06);  //Mode Resiger (last byte) --> 500Hz update rate
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x30);  //command: offset register
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
spibyte(0x7F);  //First byte of offset reg
spibyte(0x8F);  //second byte
spibyte(0xFF); // last byte
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x38);  //command: Write FS register
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
spibyte(0x55);  //First byte of FS register
spibyte(0x5F);  //second byte
spibyte(0xFF); // last byte
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x08);  //command: write mode register
P2 = 0xFF;


for (int i=0;i<1;i++);

P2 = 0xF7;
spibyte(0x00);  //Mode Register (first byte)
```



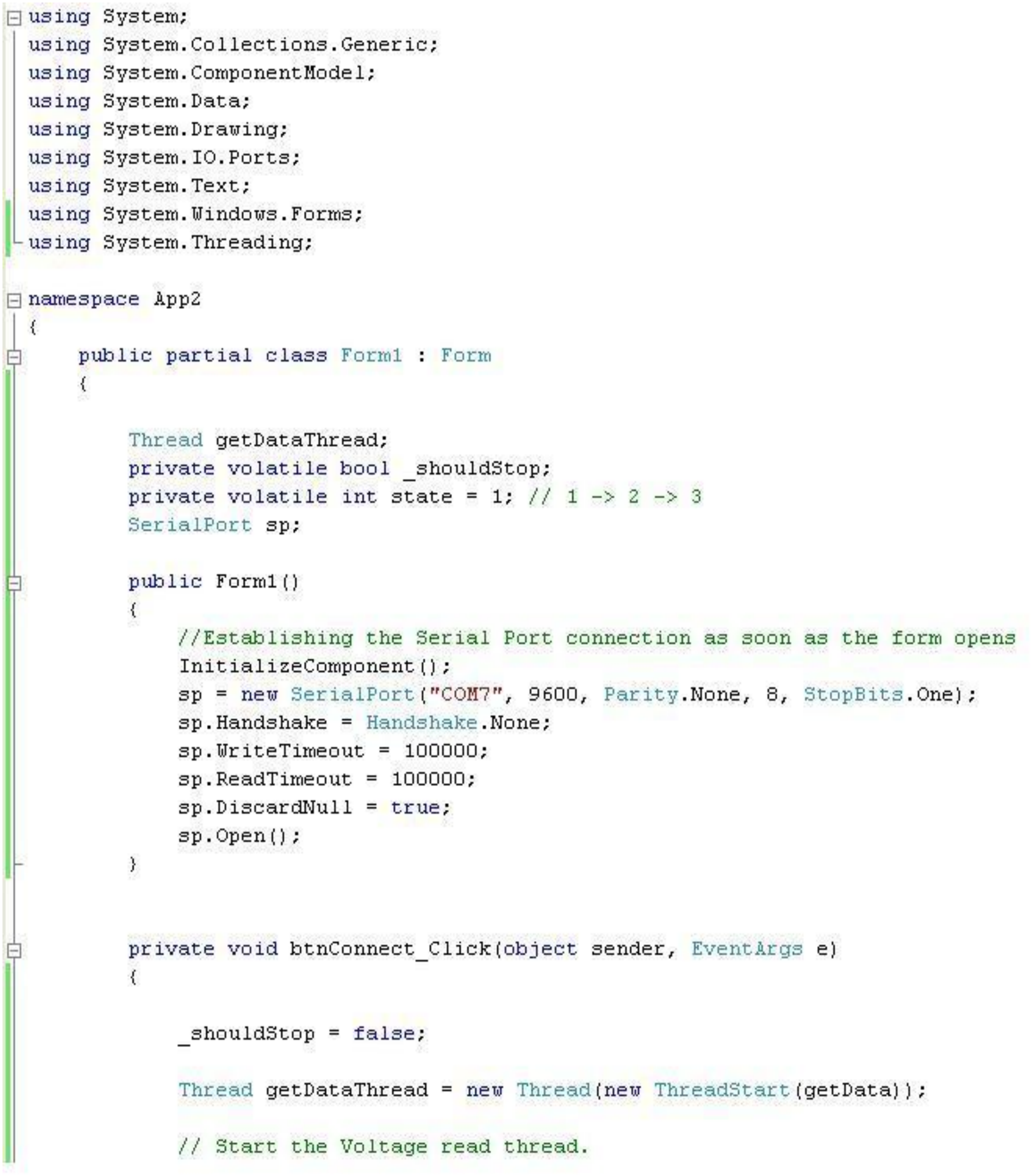
```
spibyte(0x06);  //Mode Resiger (last byte) --> 500Hz update rate
P2 = 0xFF;

for (int i=0;i<1;i++);
P2 = 0xF7;
tmp2 = spibyte(0x5C);   //Read data Register in continuous mode
//tmp2 = spibyte(0x50);    //Read Config register
P2 = 0xFF;

while (ON == 0x57)
{

  for (int i=0;i<1;i++);
  P2 = 0xF7;
  ad1 = spireadbyte();   //Reading first byte from AD7799
  P2 = 0xFF;

  for (int i=0;i<1;i++);
  P2 = 0xF7;
  ad2 = spireadbyte();   //Reading second byte from AD7799
  P2 = 0xFF;

  for (int i=0;i<1;i++);
  P2 = 0xF7;
  ad3 = spireadbyte();   //Reading third byte from AD7799
  P2 = 0xFF;

  unsigned char OFF = SBUF;
  while((SCON & 0x01))
  {
    MyBuf = 'a';
    SCON &= 0xFE;
  }

  if (OFF == 0x58)

    break;


  //Sending two ASCII characters 'U' to serve as start bytes
  SBUF = 0x55;
  while(!(SCON & 0x02))
    MyBuf = 'a';
  SCON &= 0xFD;
```

```
    SBUF = 0x55;
    while(!(SCON & 0x02))
      MyBuf = 'a';
    SCON &= 0xFD;\


    //Sending the first byte from AD7799 through the UART (Bluetooth)
    SBUF = ad1;
    while(!(SCON & 0x02))
      MyBuf = 'a';
    SCON &= 0xFD;

    //Sending the second byte from AD7799 through the UART (Bluetooth)
    SBUF = ad2;
    while(!(SCON & 0x02))
      MyBuf = 'a';
    SCON &= 0xFD;

    //Sending the third byte from AD7799 through the UART (Bluetooth)
    SBUF = ad3;
    while(!(SCON & 0x02))
      MyBuf = 'a';
    SCON &= 0xFD;


  }
}

//If the calibration button is pressed, perform this
else if (ON==0x59)
{

  //Configuring ADC0
  ADC0CON1 = 0x22;
  ADC0CON2 = 0x4A;
  ADCMODE = 0x22;

  unsigned char adc1, adc2, adc3;


  while(!(ADCSTAT & 0x80))
  { adc1 = ADC0H;
  adc2 = ADC0M;
  adc3 = ADC0L;
  ADCSTAT &= 0x7F;
```

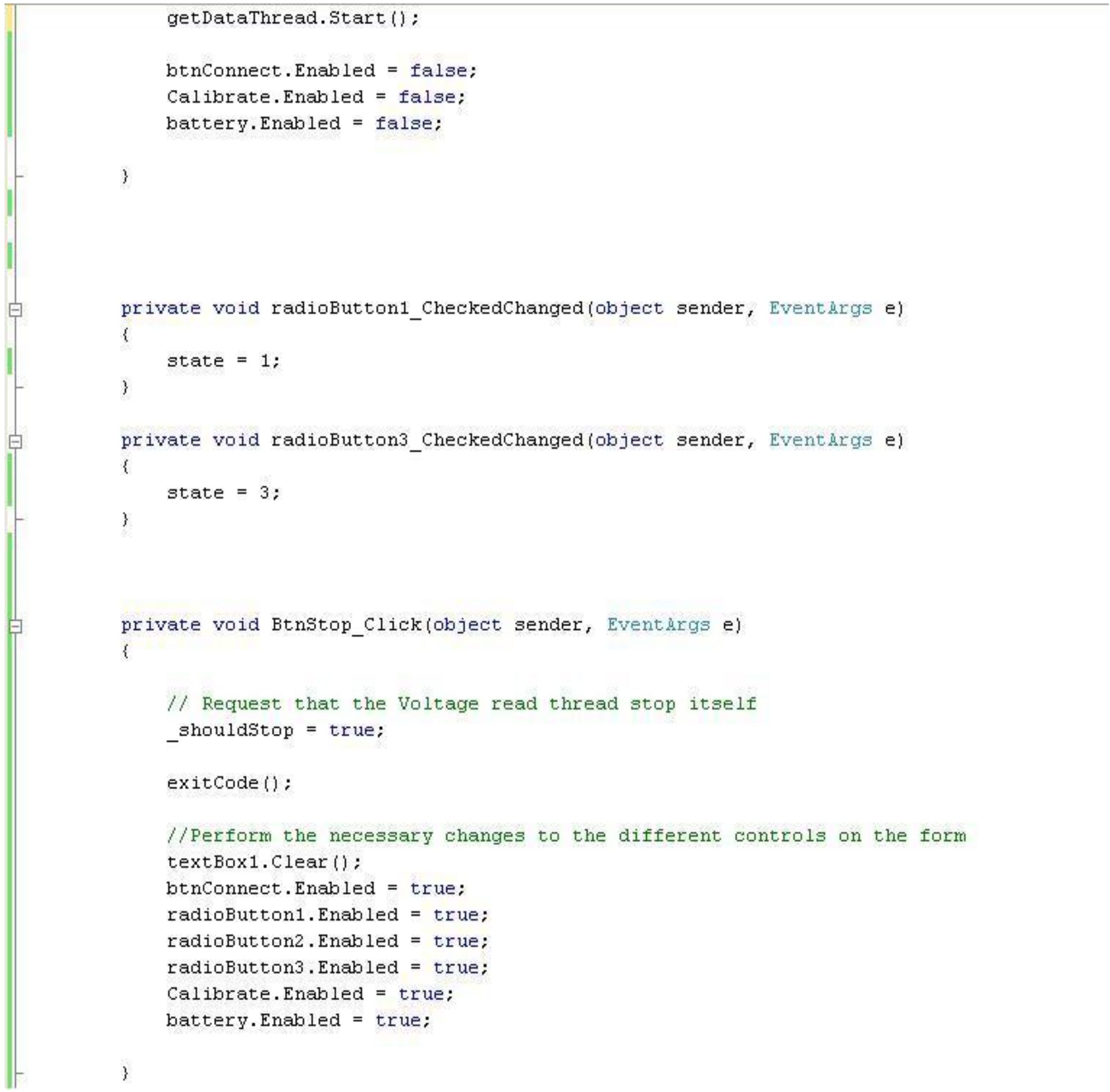


```
      }

      SBUF = 0x55;
      while(!(SCON & 0x02))
        MyBuf = 'a';
      SCON &= 0xFD;

      SBUF = 0x55;
      while(!(SCON & 0x02))
        MyBuf = 'a';
      SCON &= 0xFD;


      SBUF = adc1;
      while(!(SCON & 0x02))
        MyBuf = 'a';
      SCON &= 0xFD;

      SBUF = adc2;
      while(!(SCON & 0x02))
        MyBuf = 'a';
      SCON &= 0xFD;


      SBUF = adc3;
      while(!(SCON & 0x02))
        MyBuf = 'a';
      SCON &= 0xFD;

      unsigned char OFF = SBUF;
      while((SCON & 0x01))
      {
        MyBuf = 'a';
        SCON &= 0xFE;
      }

    }

    else
    {
    }

  }
}
```

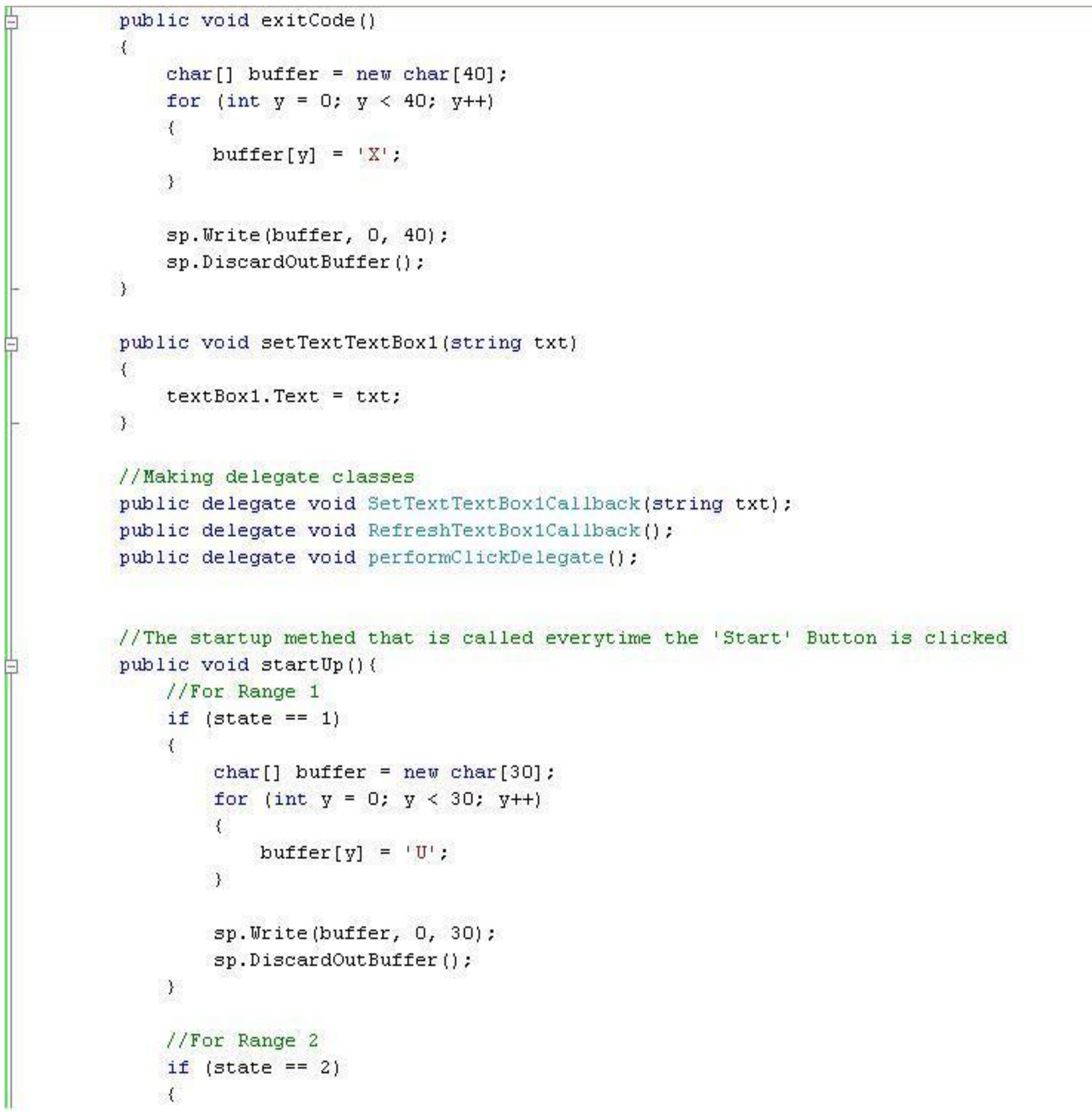
Appendix F – C# Code (Microsoft Visual Studio)



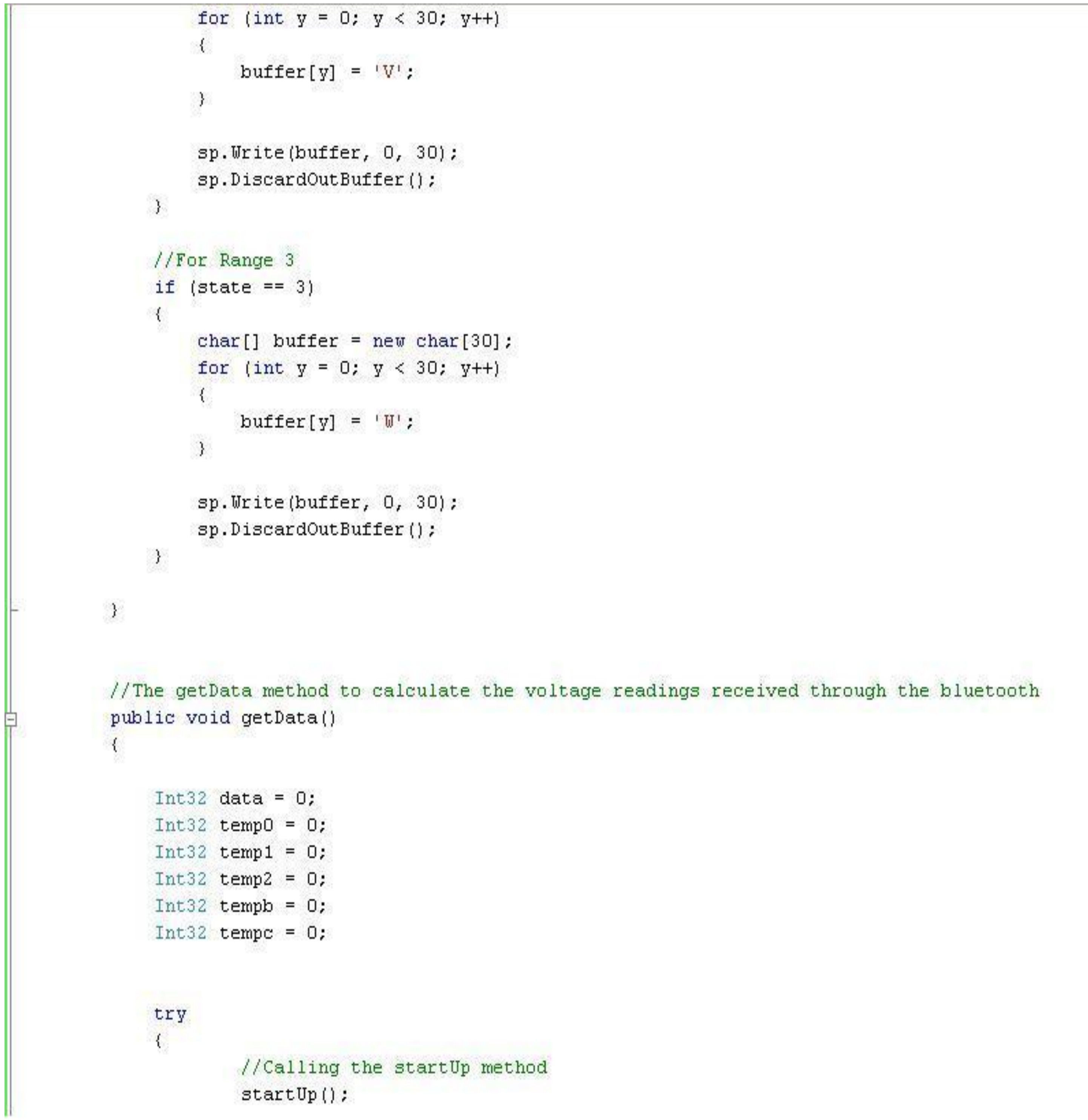
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO.Ports;
using System.Text;
using System.Windows.Forms;
using System.Threading;

namespace App2
{
    public partial class Form1 : Form
    {

        Thread getDataThread;
        private volatile bool _shouldStop;
        private volatile int state = 1; // 1 -> 2 -> 3
        SerialPort sp;

        public Form1()
        {
            //Establishing the Serial Port connection as soon as the form opens
            InitializeComponent();
            sp = new SerialPort("COM7", 9600, Parity.None, 8, StopBits.One);
            sp.Handshake = Handshake.None;
            sp.WriteTimeout = 100000;
            sp.ReadTimeout = 100000;
            sp.DiscardNull = true;
            sp.Open();
        }


        private void btnConnect_Click(object sender, EventArgs e)
        {

            _shouldStop = false;

            Thread getDataThread = new Thread(new ThreadStart(getData));

            // Start the Voltage read thread.
```

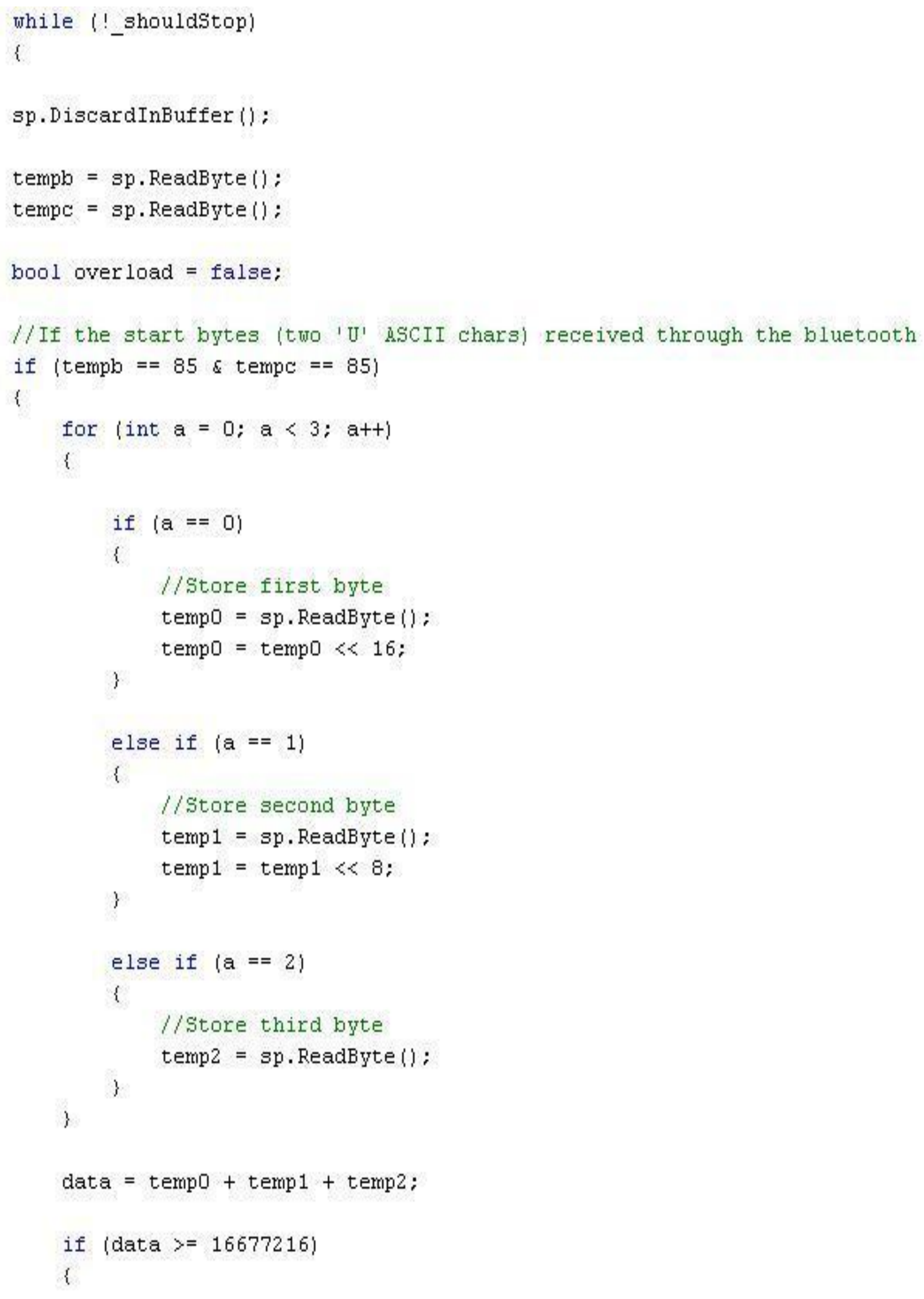


```
            getDataThread.Start();

            btnConnect.Enabled = false;
            Calibrate.Enabled = false;
            battery.Enabled = false;

        }




        private void radioButton1_CheckedChanged(object sender, EventArgs e)
        {
            state = 1;
        }

        private void radioButton3_CheckedChanged(object sender, EventArgs e)
        {
            state = 3;
        }



        private void BtnStop_Click(object sender, EventArgs e)
        {

            // Request that the Voltage read thread stop itself
            _shouldStop = true;

            exitCode();

            //Perform the necessary changes to the different controls on the form
            textBox1.Clear();
            btnConnect.Enabled = true;
            radioButton1.Enabled = true;
            radioButton2.Enabled = true;
            radioButton3.Enabled = true;
            Calibrate.Enabled = true;
            battery.Enabled = true;

        }
```

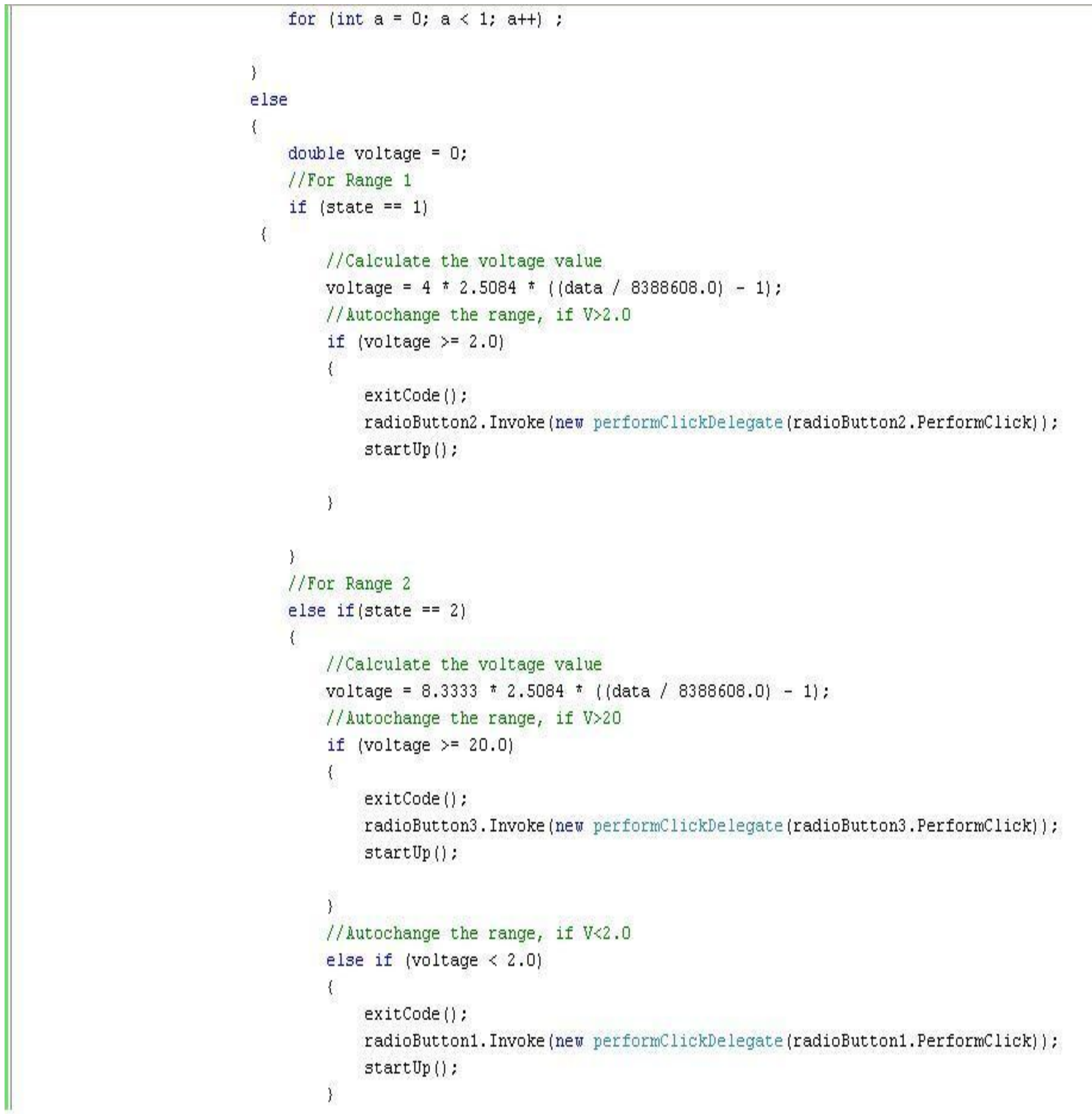


```
        public void exitCode()
        {
            char[] buffer = new char[40];
            for (int y = 0; y < 40; y++)
            {
                buffer[y] = 'X';
            }

            sp.Write(buffer, 0, 40);
            sp.DiscardOutBuffer();
        }

        public void setTextTextBox1(string txt)
        {
            textBox1.Text = txt;
        }

        //Making delegate classes
        public delegate void SetTextTextBox1Callback(string txt);
        public delegate void RefreshTextBox1Callback();
        public delegate void performClickDelegate();


        //The startup methed that is called everytime the 'Start' Button is clicked
        public void startUp(){
            //For Range 1
            if (state == 1)
            {
                char[] buffer = new char[30];
                for (int y = 0; y < 30; y++)
                {
                    buffer[y] = 'U';
                }

                sp.Write(buffer, 0, 30);
                sp.DiscardOutBuffer();
            }

            //For Range 2
            if (state == 2)
            {
```

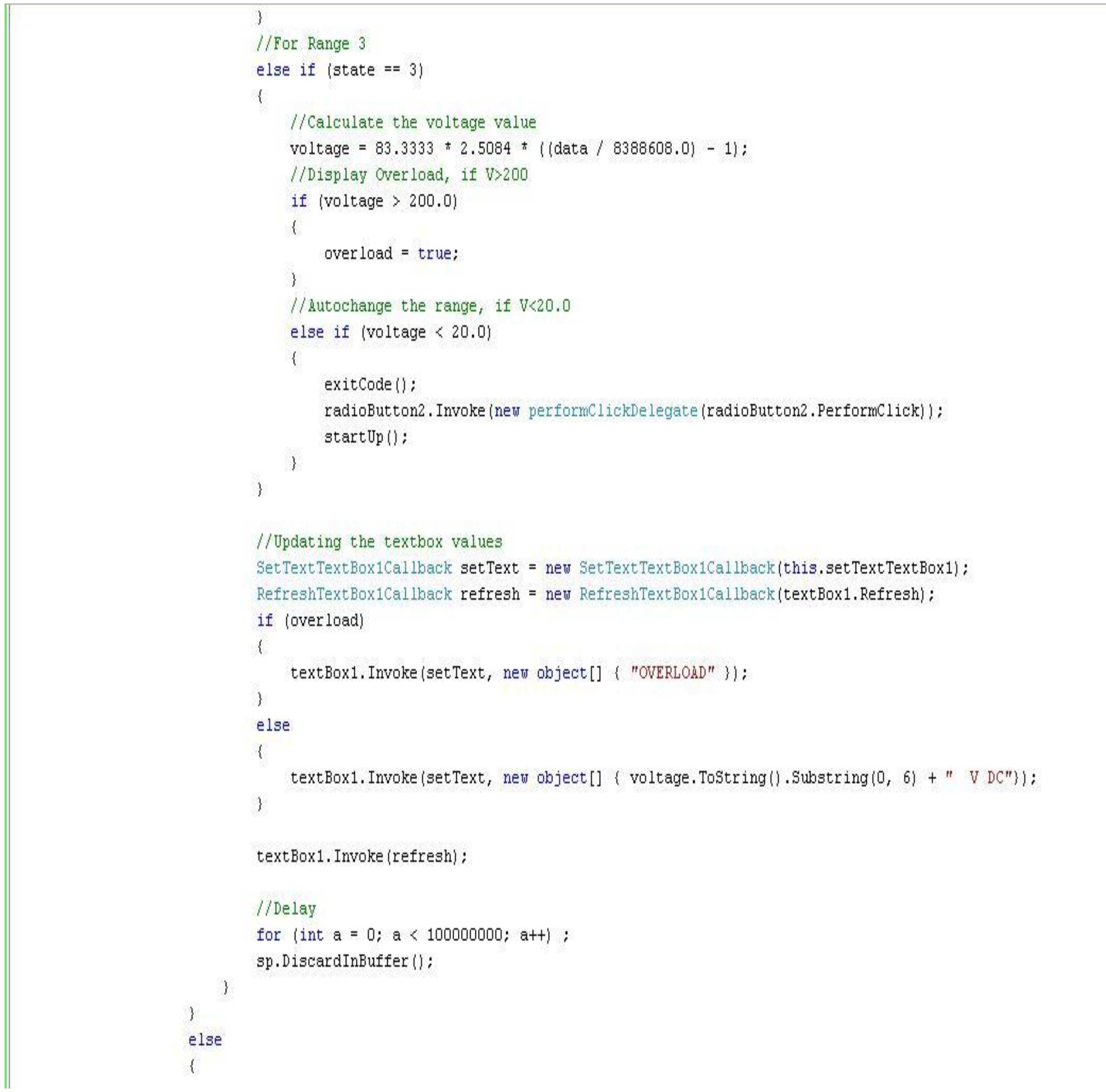

```
            for (int y = 0; y < 30; y++)
            {
                buffer[y] = 'V';
            }

            sp.Write(buffer, 0, 30);
            sp.DiscardOutBuffer();
        }

        //For Range 3
        if (state == 3)
        {
            char[] buffer = new char[30];
            for (int y = 0; y < 30; y++)
            {
                buffer[y] = 'W';
            }

            sp.Write(buffer, 0, 30);
            sp.DiscardOutBuffer();
        }

    }


    //The getData method to calculate the voltage readings received through the bluetooth
    public void getData()
    {

        Int32 data = 0;
        Int32 temp0 = 0;
        Int32 temp1 = 0;
        Int32 temp2 = 0;
        Int32 tempb = 0;
        Int32 tempc = 0;


        try
        {
                //Calling the startUp method
                startUp();
```



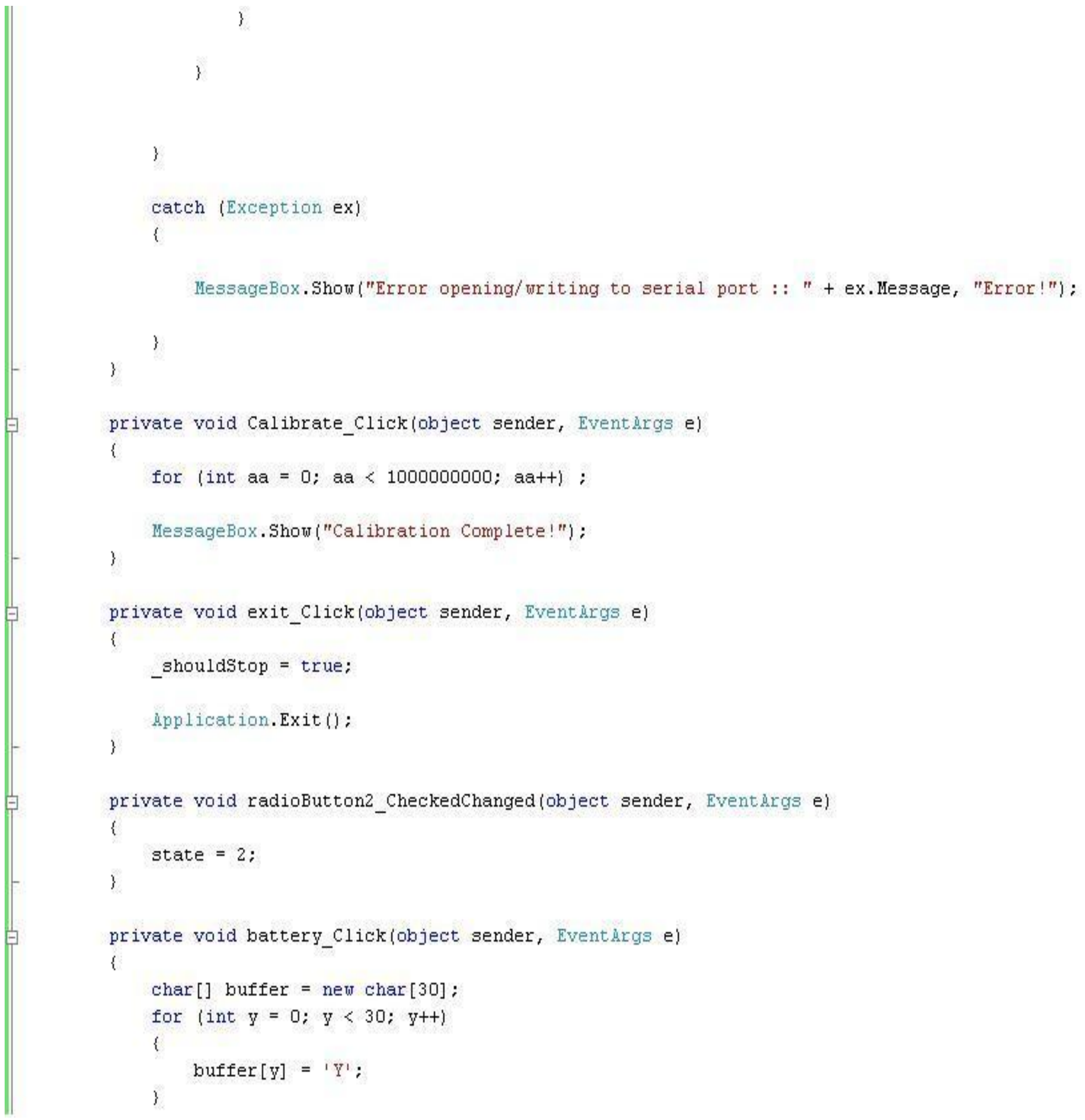
```
while (!_shouldStop)
{

sp.DiscardInBuffer();

tempb = sp.ReadByte();
tempc = sp.ReadByte();

bool overload = false;

//If the start bytes (two 'U' ASCII chars) received through the bluetooth
if (tempb == 85 & tempc == 85)
{
    for (int a = 0; a < 3; a++)
    {

        if (a == 0)
        {
            //Store first byte
            temp0 = sp.ReadByte();
            temp0 = temp0 << 16;
        }

        else if (a == 1)
        {
            //Store second byte
            temp1 = sp.ReadByte();
            temp1 = temp1 << 8;
        }

        else if (a == 2)
        {
            //Store third byte
            temp2 = sp.ReadByte();
        }
    }

    data = temp0 + temp1 + temp2;

    if (data >= 16677216)
    {
```



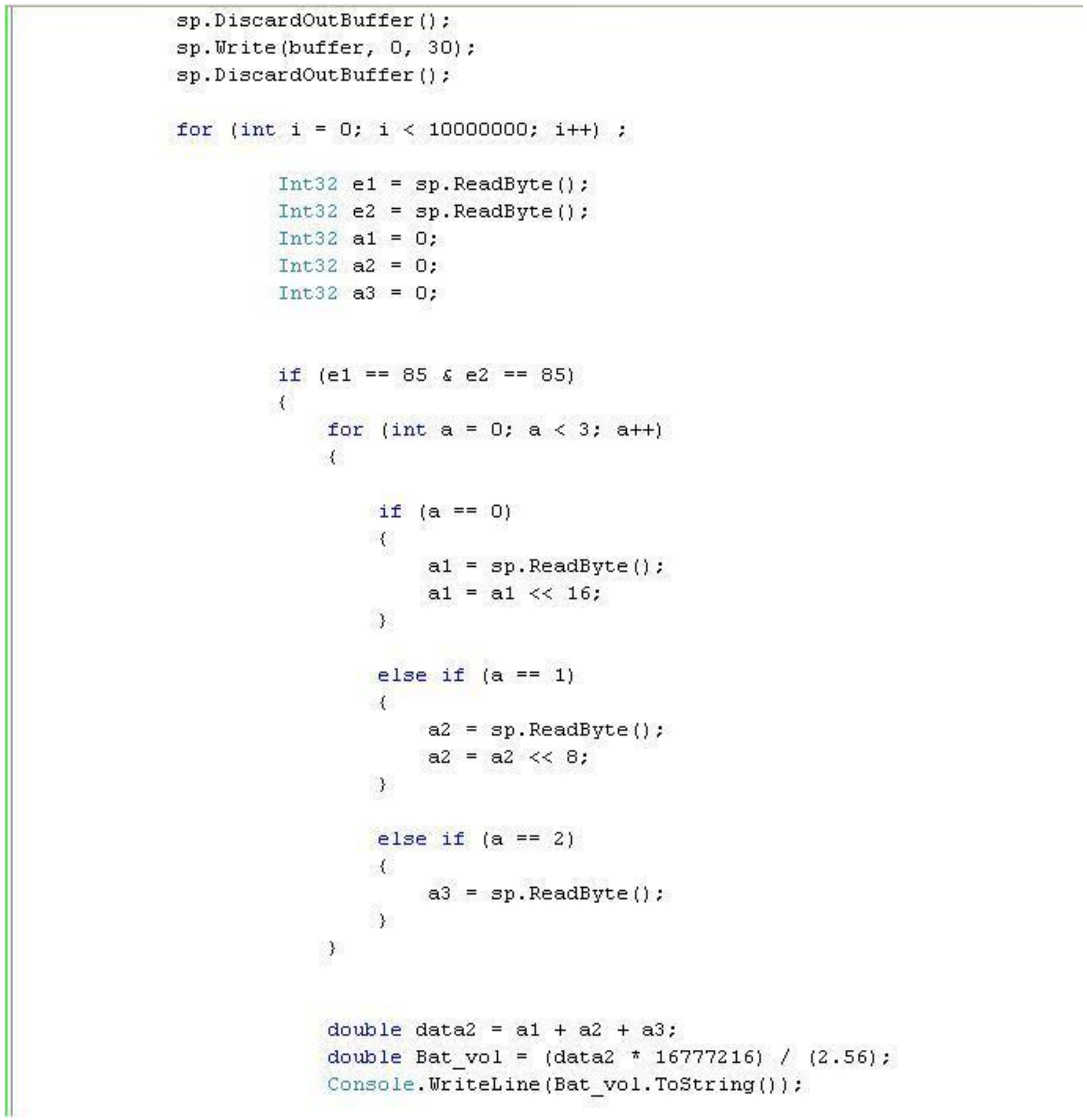
```
    for (int a = 0; a < 1; a++) ;

}
else
{
    double voltage = 0;
    //For Range 1
    if (state == 1)
 {
        //Calculate the voltage value
        voltage = 4 * 2.5084 * ((data / 8388608.0) - 1);
        //Autochange the range, if V>2.0
        if (voltage >= 2.0)
        {
            exitCode();
            radioButton2.Invoke(new performClickDelegate(radioButton2.PerformClick));
            startUp();

        }

    }
    //For Range 2
    else if(state == 2)
    {
        //Calculate the voltage value
        voltage = 8.3333 * 2.5084 * ((data / 8388608.0) - 1);
        //Autochange the range, if V>20
        if (voltage >= 20.0)
        {
            exitCode();
            radioButton3.Invoke(new performClickDelegate(radioButton3.PerformClick));
            startUp();

        }
        //Autochange the range, if V<2.0
        else if (voltage < 2.0)
        {
            exitCode();
            radioButton1.Invoke(new performClickDelegate(radioButton1.PerformClick));
            startUp();
        }
```

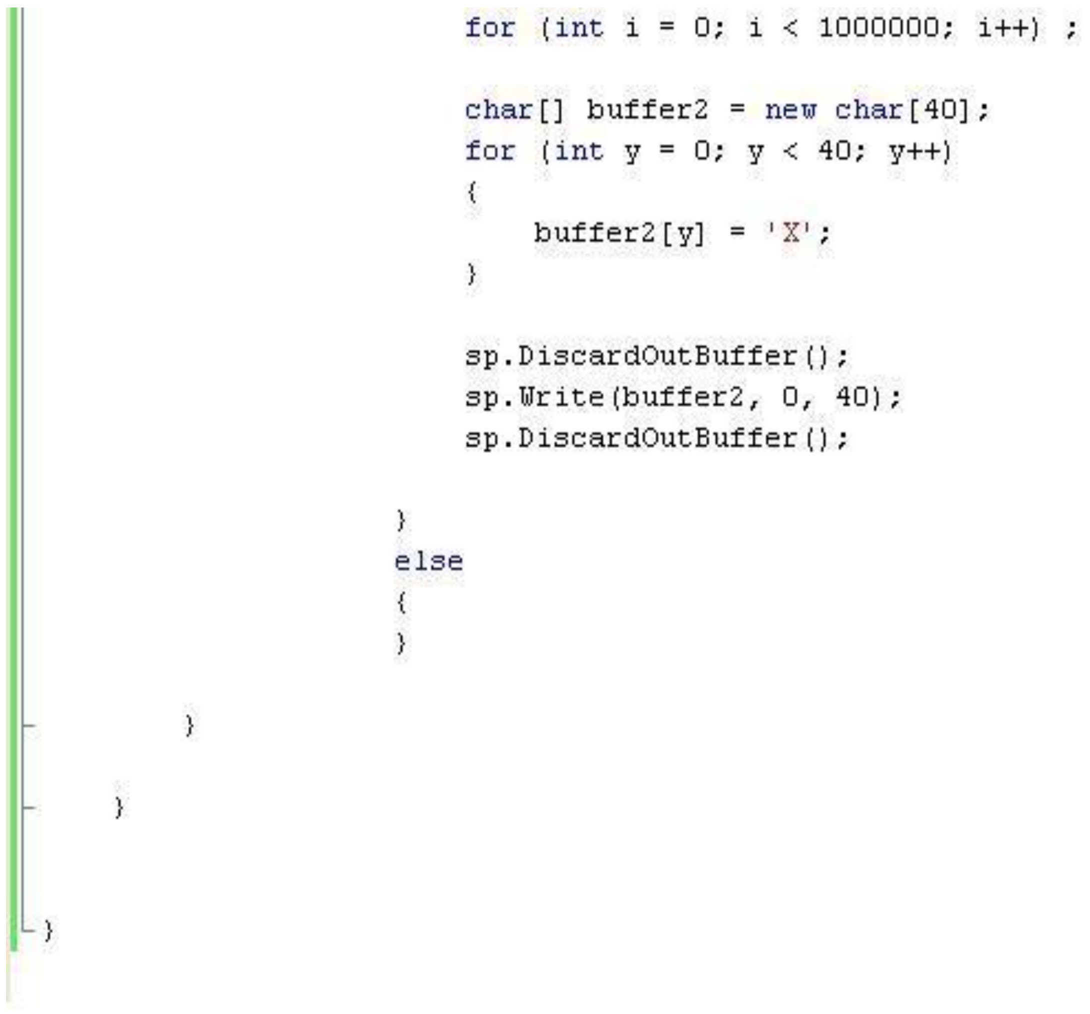


```
        }
        //For Range 3
        else if (state == 3)
        {
            //Calculate the voltage value
            voltage = 83.3333 * 2.5084 * ((data / 8388608.0) - 1);
            //Display Overload, if V>200
            if (voltage > 200.0)
            {
                overload = true;
            }
            //Autochange the range, if V<20.0
            else if (voltage < 20.0)
            {
                exitCode();
                radioButton2.Invoke(new performClickDelegate(radioButton2.PerformClick));
                startUp();
            }
        }

        //Updating the textbox values
        SetTextTextBox1Callback setText = new SetTextTextBox1Callback(this.setTextTextBox1);
        RefreshTextBox1Callback refresh = new RefreshTextBox1Callback(textBox1.Refresh);
        if (overload)
        {
            textBox1.Invoke(setText, new object[] { "OVERLOAD" });
        }
        else
        {
            textBox1.Invoke(setText, new object[] { voltage.ToString().Substring(0, 6) + "  V DC"});
        }

        textBox1.Invoke(refresh);

        //Delay
        for (int a = 0; a < 100000000; a++) ;
        sp.DiscardInBuffer();
    }
}
else
{
```



```
            }

        }


    }

    catch (Exception ex)
    {

        MessageBox.Show("Error opening/writing to serial port :: " + ex.Message, "Error!");

    }
}

private void Calibrate_Click(object sender, EventArgs e)
{
    for (int aa = 0; aa < 1000000000; aa++) ;

    MessageBox.Show("Calibration Complete!");
}

private void exit_Click(object sender, EventArgs e)
{
    _shouldStop = true;

    Application.Exit();
}

private void radioButton2_CheckedChanged(object sender, EventArgs e)
{
    state = 2;
}

private void battery_Click(object sender, EventArgs e)
{
    char[] buffer = new char[30];
    for (int y = 0; y < 30; y++)
    {
        buffer[y] = 'Y';
    }
```


```
            sp.DiscardOutBuffer();
            sp.Write(buffer, 0, 30);
            sp.DiscardOutBuffer();

            for (int i = 0; i < 10000000; i++) ;

                Int32 e1 = sp.ReadByte();
                Int32 e2 = sp.ReadByte();
                Int32 a1 = 0;
                Int32 a2 = 0;
                Int32 a3 = 0;


                if (e1 == 85 & e2 == 85)
                {
                    for (int a = 0; a < 3; a++)
                    {

                        if (a == 0)
                        {
                            a1 = sp.ReadByte();
                            a1 = a1 << 16;
                        }

                        else if (a == 1)
                        {
                            a2 = sp.ReadByte();
                            a2 = a2 << 8;
                        }

                        else if (a == 2)
                        {
                            a3 = sp.ReadByte();
                        }
                    }


                    double data2 = a1 + a2 + a3;
                    double Bat_vol = (data2 * 16777216) / (2.56);
                    Console.WriteLine(Bat_vol.ToString());
```



```
                        for (int i = 0; i < 1000000; i++) ;

                        char[] buffer2 = new char[40];
                        for (int y = 0; y < 40; y++)
                        {
                            buffer2[y] = 'X';
                        }

                        sp.DiscardOutBuffer();
                        sp.Write(buffer2, 0, 40);
                        sp.DiscardOutBuffer();

                    }
                    else
                    {
                    }

        }

    }


}
```