EMBEDDED PC BASED WIRELESS COMMUNICATION USING XTEA

KHOK JESS LYN

This thesis is submitted as partial fulfillment of the requirements for the award of Bachelor of Electrical Engineering (Electronics)

Faculty of Electrical & Electronics Engineering
Universiti Malaysia Pahang

NOVEMBER, 2010

CHAPTER 1

INTRODUCTION

1.1 Introduction

Men are social beings. Communication has and will always play an integral part in their lives. As men evolve and progress, so does its communication skills. These communication skills have developed to the extent that the information passed must be at times concealed and protected for reasons such as integrity, authenticity and confidentiality. The increasing need and interest in information protection have given rise to a new scientific field called cryptology. Cryptology is divided to two areas: cryptography and cryptanalysis.

Cryptography, in essence is the art of hiding information. It is a technique used to hide the meaning of a message and is derived from the union of the Greek words ‘kryptó’ which means hidden and ‘gráfo’ which means to write. This is different from steganography techniques in that one is not hiding the actual message, only the meaning of the message. If a message were to fall into the hands of the wrong person, cryptography should ensure that that message could not be read. Typically the sender and receiver agree upon a message scrambling protocol and methods for encrypting and decrypting the messages beforehand.

On the other hand, cryptanalysis deals with the breaking of the encrypted information. It is the study of methods for obtaining the meaning of encrypted information, without access to the secret information. Typically, this involves knowing how the system works and finding a secret key. In other words, this is the practice of code breaking or cracking the code. The process flow of a cryptosystem is simple in principle. However, the difficult and challenging part is to ensure that the cryptosystem is strong enough to withstand attacks.

Today, cryptanalysis is practiced by a broad range of organizations: governments try to break other governments' diplomatic and military transmissions; companies developing security products send them to cryptanalysts to test their security features and to a hacker or cracker to try to break the security of web sites by finding weaknesses in the securing protocols. It is this constant battle between cryptographers trying to secure information and cryptanalysts trying to break cryptosystems that moves the entire body of cryptology knowledge forward.

For this project, the Extended Tiny Encryption Algorithm (XTEA) has been chosen as the cryptography method. This is due to the reason of it being rather flexible and small in size which makes it run fast in software. XTEA is the precursor of Tiny Encryption Algorithm (TEA) designed by David Needham and Roger Wheeler to correct the weaknesses in TEA.

1.2 Problem Statement

In the last few decades, the information and communication technology industry has evolved such that it is increasingly permeating all aspects of our everyday life from emails, social networking to e-commerce. As we get progressively dependent on computers and communications networks, demand for ever more sophisticated mobile and wireless technologies in terms of security expands. However, whether or not the communication happens with the internet does not matter because as long as there is interaction between A and B, online or offline, wireless communication puts users at risk of data interception. Data, regardless of its level of confidentiality causes one vulnerable to attacks once it is passed to the wrong hand.

1.3 Objective

This project aims to produce a safe measure for wireless data communication which is fast and easy to use. This is achieved by implementing XTEA in an embedded system for PC based wireless communication using the XTend OEM RF module. When data is input from one end user, the microcontroller should be able to immediately encrypt the input and transmit it over to the other end user. The microcontroller then decrypts the encrypted data and the original message will be displayed.

1.4 Scope of Project

The project encompasses interfacing XTEA in an embedded system using the PIC18F14K50 for PC based wireless communication. The wireless module used is the XTend OEM RF module by MaxStream.

XTEA is an encryption/decryption algorithm which is has little memory footprint, fast and is simple to use. Data is sent from one PC to another wirelessly after it is encrypted by the PIC. The encrypted data prevents leakage of secret or vital information should there be an intercept because it is only decrypted when it has reached the other end user.

1.5 Organization of Thesis

This thesis is a combination of five chapters which includes the introduction, literature review, methodology, result and conclusion.

Chapter 1 is the introduction to the project. In this chapter, the project is given an overview whereby it discusses the background leading to the project, problem statement, and the objectives to be achieved.

Chapter 2 is a compilation of literatures related to the project. Past developments, achievements and the milestone of cryptography, notably the XTEA, are discussed in this chapter.

Chapter 3 comprises of the project methodology. There are two main parts: hardware development and software development. Hardware development of the project includes the hardware components and how they are set up in stages. Software development relates the program development in C language.

Chapter 4 presents the result and discussion of this project. This chapter is mainly made up of simulations and their analysis to test the versatility of the project.

Chapter 5 is the final chapter in which this project is wrapped up. An executive summary is given and recommendations are made for future improvement of the project.

CHAPTER 2

LITERATURE REVIEW

2.1 Cryptography: A Brief History

Natural language ciphers have existed for over 4,000 years, with the Egyptians being the first that is known to engage in the practice. Not long after the first text was encrypted, the art of decrypting the cipher- cryptanalysis; emerged. The history of cryptography can be broadly divided into three phases:

1. From ancient civilizations to the nineteenth century and the first part of the twentieth century, with relatively simple algorithms that was designed and implemented by hand.
2. Extensive use of encrypting electro-mechanical machines, around the period of the Second World War.
3. Ever more pervasive use of computers, about in the last fifty years, supported by solid mathematical basis [3].

Early ciphers used substitution to replace letters and eventually evolved to also include transpositions of characters as well as simple substitution. Julius Caesar was the first recorded user of the substitution technique. Many experts in the field of cryptography have argued that all good cryptographers are also good at cryptanalysis. The idea is that if one has experience with methods of attack and cracking codes, he or she probably has better ideas on how to construct codes that cannot be cracked.

After World War I had ended, in 1926 the German Navy adopted the use of a family of electro-mechanical encryption devices that became collectively known as the Enigma machine. The device proved to be so effective that by World War II their use had spread to every branch of the German military. The enigma machine is said to have kick start the cryptography revolution.

The dawn of the computer age has brought about the invention of new mathematical techniques to accelerate the cryptography process to even better methods of encryption. In fact today, cryptography has become so advanced that many of the ciphers are considered to be unbreakable, in a relative term. Although it might take as long as several decades for one of today's computers to systematically decode a ciphertext produced by some of today's leading methods of encrypting, there is no guarantee that it would never be cracked.

In order for the ciphertext to be at all useful to the intended receiver, it must have some form of order to it, no matter how obscure this sense of order may be. Herein lies the problem. If there is an order to the cipher, regardless of how remote, there will always be the possibility that someone will find a way to exploit this order, and eventually crack the cipher [11]. Thus, the discovery of an absolutely secure means of communication continues to elude cryptologists today.

2.2 Standards

Over the course of the years, cryptography has gained worldwide interests so much so that a standard encryption algorithm was requested to standardize the presently available cryptography methods back in the early 70s. From then on, the Data Encryption Standard (DES) was proposed. However over time, the DES was found to be insecure in many applications. This is chiefly due to the 56-bit key size being too small and in 1999 where the collaboration between distributed.net and the Electronic Frontier Foundation managed to publicly break a DES key in 22 hours and 15 minutes. Therefore eventually in 2001, a new method called the Rijndael was announced as the new Advanced Encryption Standard (AES) to replace DES.

2.2.1 Data Encryption Standard (DES)

The Data Encryption Standard (DES) [8] is a cipher selected as an official Federal Information Processing Standard (FIPS) for the United States in 1976. As a block cipher DES operates on blocks with a size of 64 bits. The key also consists of 64 bits; only 56 of these are actually used by the algorithm, the other ones are parity check bits. The overall structure consists of a so-called Feistel network with 16 identical base rounds with 8 substitution boxes (S-Boxes), an initial permutation, a final permutation, and a separate key schedule.

The whole cipher consists only of bit operations, namely shifts, bit-permutations and exclusive-or operations. The introduction of DES is considered to have been a catalyst for the academic study of cryptography, particularly of methods to crack block ciphers. Figure 2.1 shows the feistel function of DES.

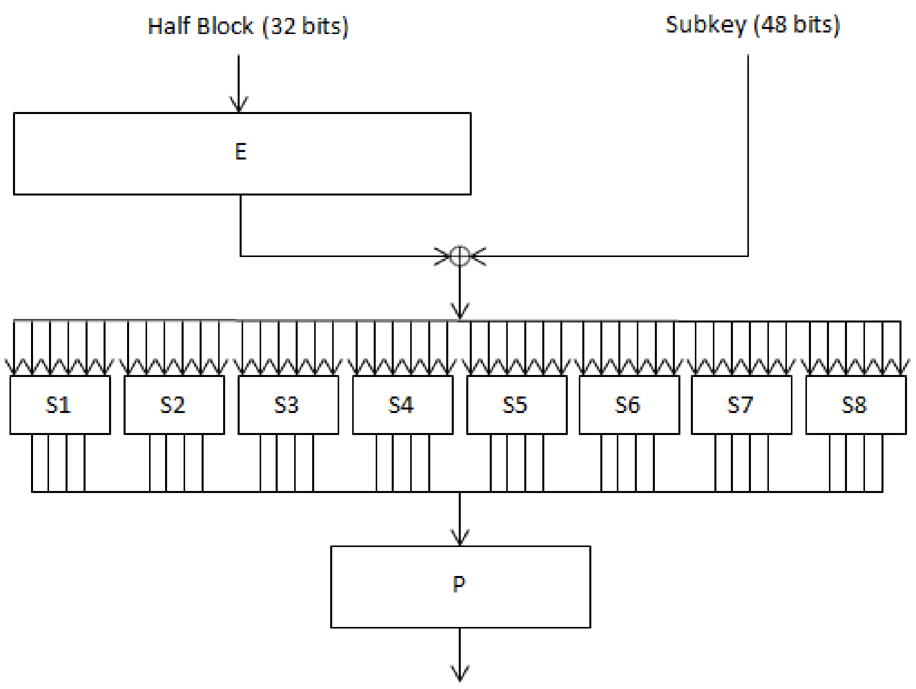


Figure 2.1: The feistel function of DES

2.2.2 Advanced Encryption Standard (AES)

AES was announced by National Institute of Standards and Technology (NIST) on 26 November 2001 after a 5 year standardization process before Rijndael was selected as the most suitable. It became effective as a Federal government standard after approval by the Secretary of Commerce. The Rijndael cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen.

AES is based on a design principle known as a substitution permutation network and is fast in both software and hardware. Unlike its predecessor, DES, AES does not use a Feistel network. AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, whereas Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits. The block size has a maximum of 256 bits, but the key size has no theoretical maximum.

The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of ciphertext. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

2.3 Tiny Encryption Algorithm (TEA)

In cryptography, TEA is a block cipher notable for its simplicity of description and implementation, typically a few lines of code. It was designed by David Wheeler and Roger Needham of the Cambridge Computer Laboratory and was first published in the proceedings of the Fast Software Encryption workshop in 1994. The main goal of their design at that time was to produce a cipher that is simple, short and does not rely on large tables or pre-computations [1].

TEA uses only simple addition, XOR and shifts functions, and has very small code size. This makes TEA an ideal candidate to provide data security services for wireless sensor network (WSN). Even though TEA was proposed mainly for software implementations, its simple design makes it also very suitable for hardware implementation.

TEA operates on 64-bit blocks and uses a 128-bit key. It has a Feistel structure with a suggested 64 rounds, typically implemented in pairs termed cycles. It has an extremely simple key schedule, mixing all of the key material in exactly the same way for each cycle. Different multiples of a magic constant are used to prevent simple attacks based on the symmetry of the rounds. Figure 2.2 shows the adaptation of the TEA in C code.

```
#include <stdint.h>

void encrypt (uint32_t*v, uint32_t*k)
{
   uint32_t v0=v[0], v1=v[1], sum=0, i;
   uint32_t delta=0x9e3779b9;
   uint32_t ko=[0], k1=k[1], k2=k[2], k3=k[3];
   for (i=0; i<32; i++)
   {
   sum += delta;
   v0 += ((v1<<4)+k0) ^ (v1+sum) ^ ((v1>>5)+k1);
   v1 += ((v0<<4)+k2) ^ (v0+sum) ^ ((v0>>5)+k3);
   }
   v[0]=v0; v[1]=v1;
}

#include <stdint.h>

void decrypt (uint32_t*v, uint32_t*k)
{
   uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i;
   uint32_t delta=0x9e3779b9;
   uint32_t ko=[0], k1=k[1], k2=k[2], k3=k[3];
   for (i=0; i<32; i++)
   {
   v1 -= ((v0<<4)+k2) ^ (v0+sum) ^ ((v0>>5)+k3);
   v0 -= ((v1<<4)+k0) ^ (v1+sum) ^ ((v1>>5)+k1);
   sum -= delta;
   }
   v[0]=v0; v[1]=v1;
}
```

Figure 2.2: Adaptation of the reference encryption and decryption routines in C

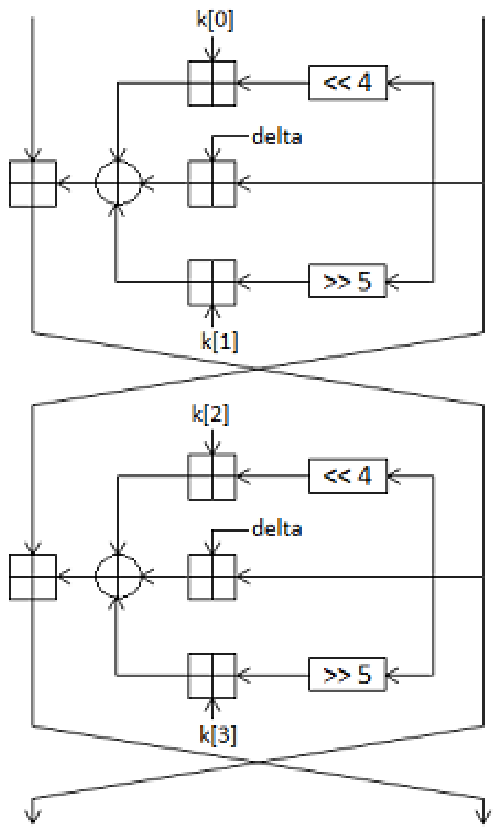


Figure 2.3: Two Feistel rounds of the TEA

Shortly after TEA was published, a few weaknesses were found. The mixing portion of TEA seems unbroken but related key attacks are possible even though the construction of 232 texts under two related keys seems impractical. The second weakness is that the effective length of the keys is 126 bits not 128 does affect certain potential applications but not the simple cypher decipher mode [2]. The authors rectified those weaknesses in a new version of TEA called XTEA.

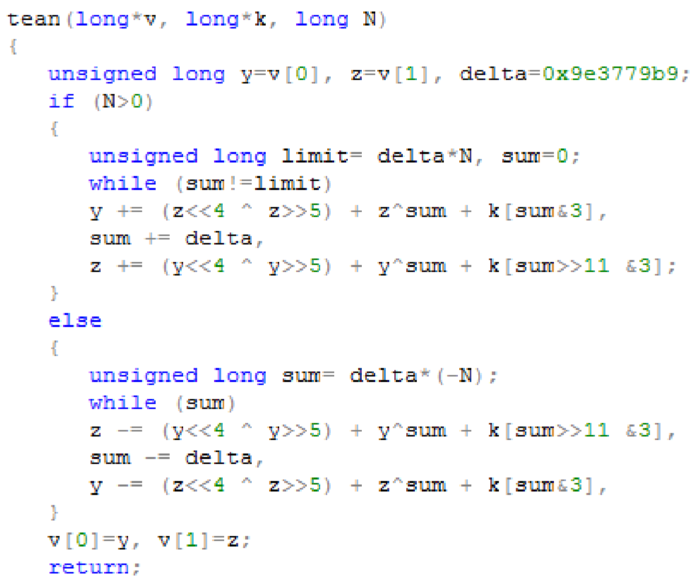
2.4 Extended Tiny Encryption Algorithm (XTEA)

XTEA is designed by Needham and Wheeler to correct the weaknesses of TEA and was presented in an unpublished technical report in 1997. The XTEA is a block cipher that uses a cryptographic key of 128 bits to encrypt or decrypt data in blocks of 64 bits. Each input block is split into two halves, y and z, which are then applied to a routine similar to a Feistel network for N rounds where N is typically 32. Most Feistel networks apply the result of a mixing function to one half of the data using XOR as a reversible function. XTEA uses for the same purpose integer addition during encryption and subtraction during decryption.

Several differences from TEA are apparent, including a somewhat more complex key-schedule and a rearrangement of the shifts, XORs, and additions. Presented along with XTEA was a variable-width block cipher termed Block TEA, which uses the XTEA round function but applies it cyclically across an entire message for several iterations. Because it operates on the entire message, Block TEA has the property that it does not need a mode of operation.

2.4.1 Mode of Operation

Figure 2.4 shows the C source code for XTEA as it was introduced in [4]. Additional parentheses were added to clarify the precedence of the operators. The main variables y, z, and sum, which assist with the subkey generation, have a length of 32 bits. All additions and subtractions within XTEA are modulo 2^{32}. Logical left shifts of z by 4 bits are denoted as $z \leq 4$ and logical right shift by 5 bits as $z \geq 5$. The bitwise XOR function is denoted as "^" in the source code and $\oplus$ in Figure 2.4 and Figure 2.5.



```
tean(long*v, long*k, long N)
{
   unsigned long y=v[0], z=v[1], delta=0x9e3779b9;
   if (N>0)
   {
      unsigned long limit= delta*N, sum=0;
      while (sum!=limit)
      y += (z<<4 ^ z>>5) + z^sum + k[sum&3],
      sum += delta,
      z += (y<<4 ^ y>>5) + y^sum + k[sum>>11 &3];
   }
   else
   {
      unsigned long sum= delta*(-N);
      while (sum)
      z -= (y<<4 ^ y>>5) + y^sum + k[sum>>11 &3],
      sum -= delta,
      y -= (z<<4 ^ z>>5) + z^sum + k[sum&3],
   }
   v[0]=y, v[1]=z;
   return;
```

Figure 2.4: Standard C source code for XTEA

The first part of the algorithm is the encryption routine and the second part is the decryption routine. The while-loop constitutes the round function. The formulae that compute the new values for y and z can be split into a permutation function f(z) = $(z \leq 4 \oplus z \geq 5) + z$ and a subkey generation function sum + k(sum).

The function k(sum) selects one block out of the four 32-bit blocks that comprise the key, depending on either bits 1 and 0 or bits 12 and 11 of sum. The results of the permutation function and the subkey generation function are XORed and then applied to y and z respectively, by addition in the case of encryption or subtraction in the case of decryption. This leads to the simplified block diagram shown in Figure 2.5.

For encryption, z is applied to the left side, y to the right side, and all adder/subtractors are in addition mode. For decryption, the opposite is applied. The permutation function is shown as ‘f’ and the subkey generation as ‘keygen’. One round of XTEA computes a new value for y and z. [1]

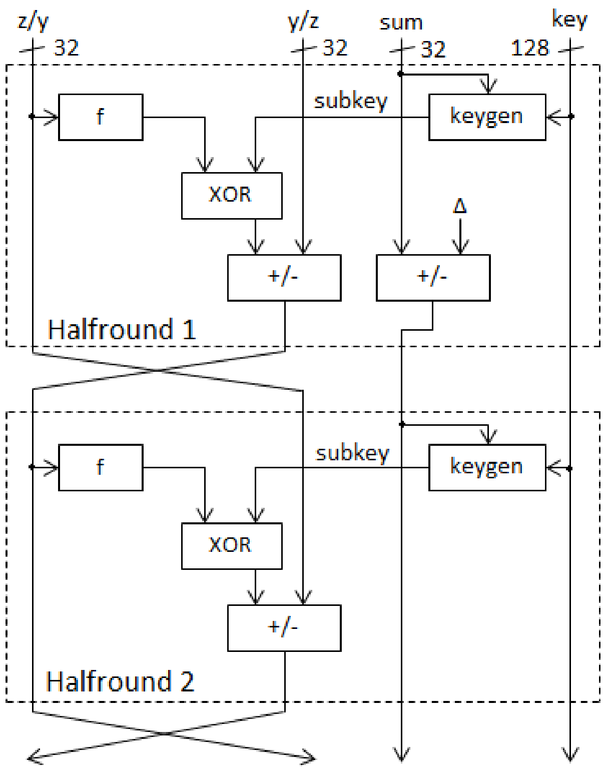


Figure 2.5: Two Feistel rounds (one cycle) of the XTEA

Therefore, the computation of one value can be viewed as a halfround. A new value for sum is computed between the first and the second half round. It is incremented by a constant Δ during encryption and decremented during decryption.

This computation is included into the first halfround as it can be performed concurrently with the final addition/subtraction of the data in this half round.

CHAPTER 3

METHODOLOGY

3.1 Introduction

Embedded PC based wireless communication using XTEA is an incorporation of GUI, cryptography algorithm and wireless modules. Figure 3.1 shows an overview of the project while Figure 3.2 is the project flow chart.

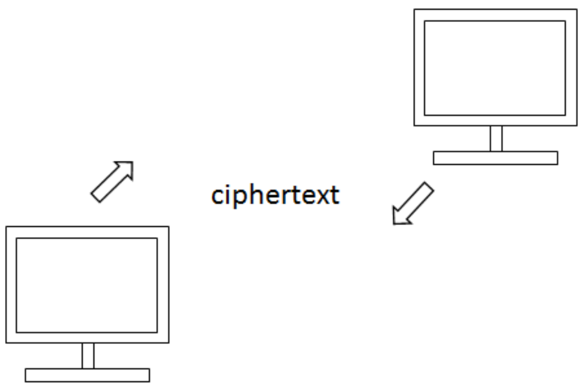


Figure 3.1: Project overview

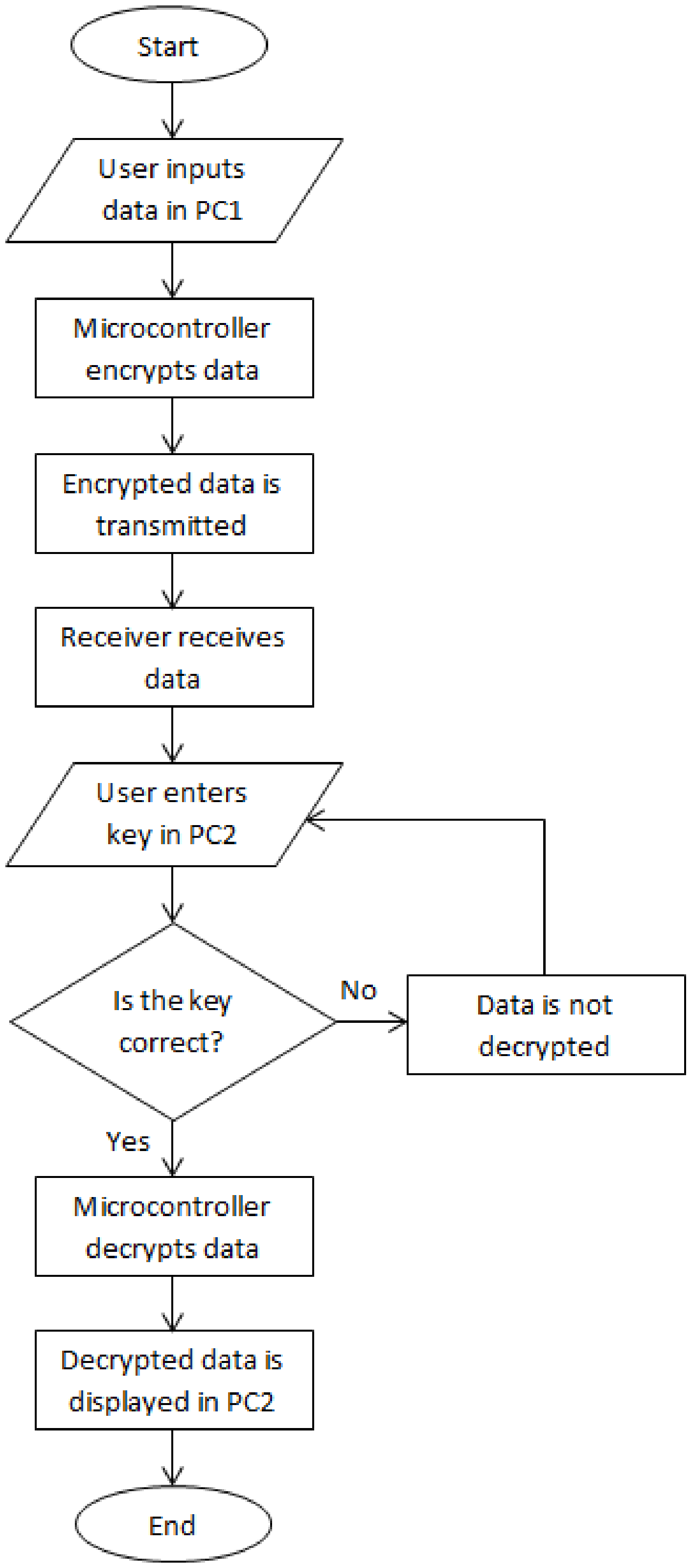

Start
User inputs data in PC1
Microcontroller encrypts data
Encrypted data is transmitted
Receiver receives data
User enters key in PC2
Is the key correct?
No
Data is not decrypted
Yes
Microcontroller decrypts data
Decrypted data is displayed in PC2
End

Figure 3.2: Flow Chart

The two PCs relay information in the form of ciphertext. GUI provides a platform for users to enter and display data. When data is entered in the PC, the input data is encrypted using XTEA and then transmitted by the XTend OEM RF module. The receiver on the other PC receives the ciphertext. If the correct key is entered, again using XTEA, the microcontroller decrypts the ciphertext. The original data is then displayed in the PC. The encryption/decryption process is bidirectional.

3.2 Hardware Development

The project utilizes three types of components:

- UC00A (USB to UART converter)
- PIC18F14K50 microcontroller
- XTend OEM RF module

Figure 3.3 shows how the three components are connected to the PC. The PC and microcontroller interface through serial communication. However, the PC's serial port uses the RS232 protocol but the microcontroller uses TTL UART. In that case, the UC00A bridges the PC and microcontroller by acting as a level shifter for the interface. The XTend OEM RF module is the transmitter/receiver for the wireless communication between the PCs.

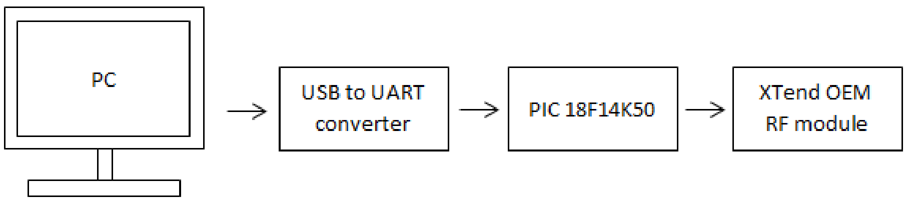


Figure 3.3: Hardware setup

3.2.1 PIC18F14K50

The PIC18F14K50 as shown in Figure 3.4 is an 8-bit microcontroller by Microchip Technology. This microcontroller is chosen because it is high performance while economically priced, has 16 Kbytes of flash memory up to 8192 single-word instructions, and is small in size. The microcontroller is programmed using the PICkit2 programmer. Figure 3.5 is the hex file while Figure 3.6 shows how the microcontroller is programmed.

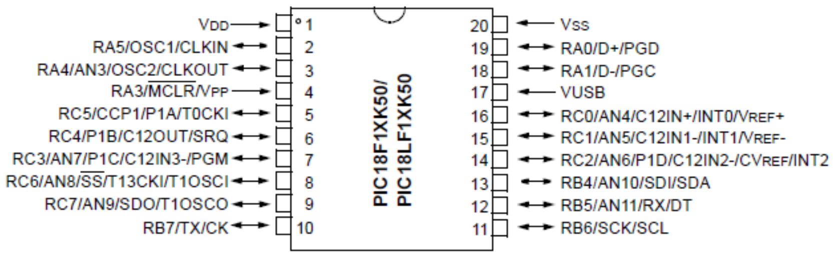
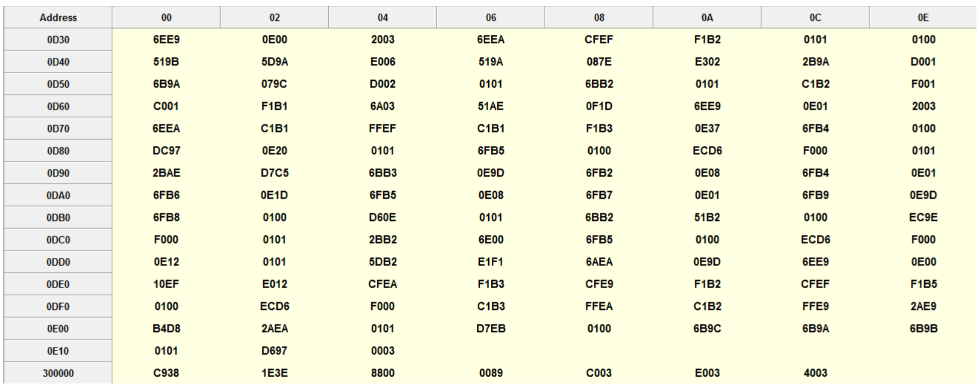
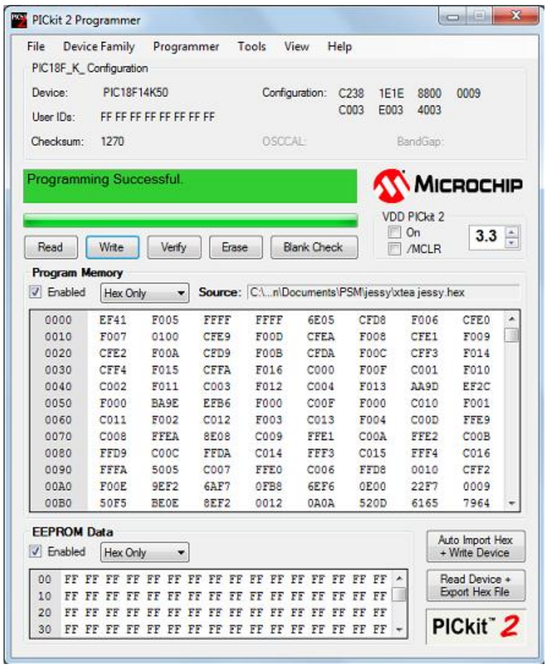


Figure 3.4: PIC18F14K50 pin diagram



Address	00	02	04	06	08	0A	0C	0E
0D30	6EE9	0E00	2003	6EEA	CFEF	F1B2	0101	0100
0D40	519B	5D9A	E006	519A	087E	E302	2B9A	D001
0D50	6B9A	079C	D002	0101	6BB2	0101	C1B2	F001
0D60	C001	F1B1	6A03	51AE	0F1D	6EE9	0E01	2003
0D70	6EEA	C1B1	FFEF	C1B1	F1B3	0E37	6FB4	0100
0D80	DC97	0E20	0101	6FB5	0100	ECD6	F000	0101
0D90	2BAE	D7C5	6BB3	0E9D	6FB2	0E08	6FB4	0E01
0DA0	6FB6	0E1D	6FB5	0E08	6FB7	0E01	6FB9	0E9D
0DB0	6FB8	0100	D60E	0101	6BB2	51B2	0100	EC9E
0DC0	F000	0101	2BB2	6E00	6FB5	0100	ECD6	F000
0DD0	0E12	0101	5DB2	E1F1	6AEA	0E9D	6EE9	0E00
0DE0	10EF	E012	CFEA	F1B3	CFE9	F1B2	CFEF	F1B5
0DF0	0100	ECD6	F000	C1B3	FFEA	C1B2	FFE9	2AE9
0E00	B4D8	2AEA	0101	D7EB	0100	6B9C	6B9A	6B9B
0E10	0101	D697	0003					
300000	C938	1E3E	8800	0089	C003	E003	4003	

Figure 3.5: Hex file



```
PIC18F_K_ Configuration
Device:      PIC18F14K50                Configuration:  C238  1E1E  8800  0009
                                                        C003  E003  4003
User IDs:    FF FF FF FF FF FF FF FF
Checksum:    1270                       OSCCAL:               BandGap:

Programming Successful.

VDD PICkit 2
[ ] On
[ ] /MCLR        3.3

Program Memory
[✓] Enabled   Hex Only   Source: C:\...n\Documents\PSM\jessy\xtea jessy.hex

0000   EF41   F005   FFFF   FFFF   6E05   CFD8   F006   CFE0
0010   F007   0100   CFE9   F00D   CFEA   F008   CFE1   F009
0020   CFE2   F00A   CFD9   F00B   CFDA   F00C   CFF3   F014
0030   CFF4   F015   CFFA   F016   C000   F00F   C001   F010
0040   C002   F011   C003   F012   C004   F013   AA9D   EF2C
0050   F000   BA9E   EFB6   F000   C00F   F000   C010   F001
0060   C011   F002   C012   F003   C013   F004   C00D   FFE9
0070   C008   FFEA   8E08   C009   FFE1   C00A   FFE2   C00B
0080   FFD9   C00C   FFDA   C014   FFF3   C015   FFF4   C016
0090   FFFA   5005   C007   FFE0   C006   FFD8   0010   CFF2
00A0   F00E   9EF2   6AF7   0FB8   6EF6   0E00   22F7   0009
00B0   50F5   BE0E   8EF2   0012   0A0A   520D   6165   7964

EEPROM Data
[✓] Enabled   Hex Only

00  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
10  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
30  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Figure 3.6: Programming the PIC

3.2.2 Serial Communication

Serial communication is the most popular interface between devices and this applies to microcontroller and computer. UART is one of those serial interfaces. Traditionally, most serial interface from microcontroller to computer is done through serial port (DB9). However, since computer serial ports use RS232 protocol and microcontrollers use TTL UART, a level shifter is needed between these interfaces.

Recently, computer serial port has been phased out. It has been replaced with USB. Of course most developer chooses USB to serial converter to obtain virtual serial port. The level shifter is still necessary for UART interface. Thus, the UC00Aproduced by Cytron is developed as a USB to UART converter which offers USB plug and play, direct interface with microcontroller and provides low current 5V supply from USB port.

Traditional method:

$$\boxed{\text{Computer} \rightarrow \text{Serial Port} \rightarrow \text{MAX232} \rightarrow \text{Microcontroller}}$$

Or

$$\boxed{\text{Computer} \rightarrow \text{USB} \rightarrow \text{Serial Port} \rightarrow \text{MAX232} \rightarrow \text{Microcontroller}}$$

Using UC00A:

$$\boxed{\text{Computer} \rightarrow \text{UC00A} \rightarrow \text{Microcontroller}}$$

The connection is made so much simpler with UC00A as there is no need for additional connections.

3.2.3 XTend OEM RF Module

The XTend OEM RF Module interface to a host device through a TTL-level asynchronous serial port. Through its serial port, the module can communicate with any UART voltage compatible device or through a level translator to any serial device (For example: RS-232/485/422 or USB interface board) [6]. Devices that have a UART interface can connect directly to the pins of the RF module as shown in the Figure 3.7. Some of the key features of the Xtend OEM RF module are as follow:

- Low power: 2.8 – 5.5 supply voltage
- Indoor/Urban range up to 900 m
- Its small form-factor enables it to be easily designed into a wide range of data systems

- There are no master/slave setup dependencies

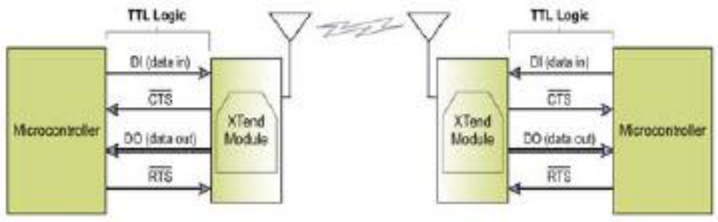


Figure 3.7: System Data Flow Diagram in a UART-interfaced environment

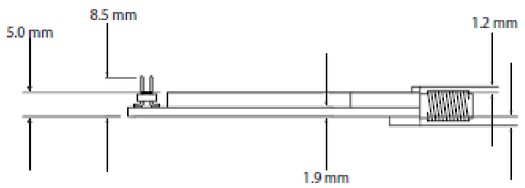


Figure 3.8: Side view of the XTend OEM RF module

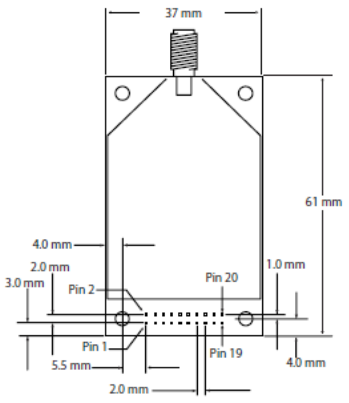
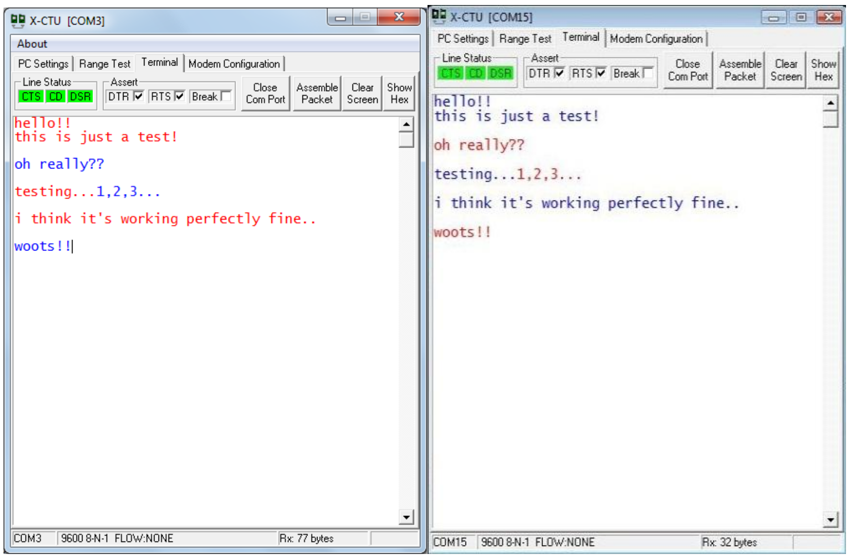


Figure 3.9: Top view of the XTend OEM RF module

To check for the functionality of the modules, X-CTU, a testing and configuration software is used for this purpose. Figure 3.10 shows the communication between two properly working modules in which they were embedded on a USB interface board and are connected to the PC through USB cables. Once connected to a PC, the module would automatically assume a COM port. The words in red denote input data while the words in blue denote data received. The exchange seems to be almost immediate and there was no lagging.



```
hello!!
this is just a test!

oh really??

testing...1,2,3...

i think it's working perfectly fine..

woots!!
```

```
COM3   9600 8-N-1  FLOW:NONE        Rx: 77 bytes
```

```
hello!!
this is just a test!

oh really??

testing...1,2,3...

i think it's working perfectly fine..

woots!!
```

```
COM15   9600 8-N-1  FLOW:NONE       Rx: 32 bytes
```

Figure 3.10: X-CTU

The hardware for the project is set up in stages until it eventually is as shown in Figure 3.3. The setup is done in stages to ensure that each and every component is working properly. Throughout the stages, X-CTU is used to test the wireless communication between the XTend OEM RF modules.

In the first stage, the XTend OEM RF modules are connected directly to the PCs through a USB interface board, as what has been described above. This is shown in Figure 3.11 below. The output is as shown in Figure 3.10 and the expected output for all stages goes the same.

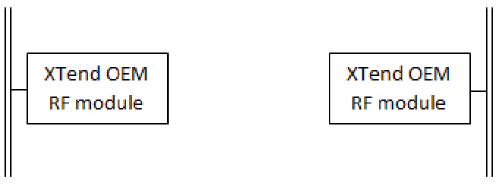


Figure 3.11: Stage 1

For the second stage, the UC00A USB to UART converter is placed between the XTend OEM RF module and PC without the USB interface board as is visualized in Figure 3.12.

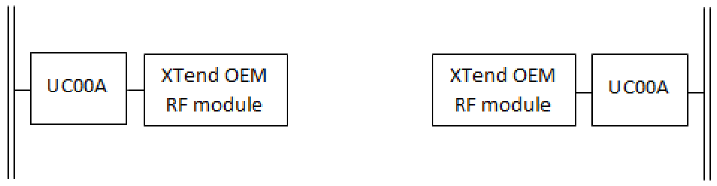


Figure 3.12: Stage 2

The final stage is shown in Figure 3.13. With the programmed microcontroller placed in between UC00A and the XTend OEM RF module, data encryption/decryption is incorporated in the wireless communication. Before the data is transmitted from the PC, the microcontroller encrypts the data based on the XTEA cryptography algorithm. The wireless module on the other PC receives the ciphertext but does not decrypt it until the correct key has been entered by the receiver (user). The original data is then displayed on the PC.

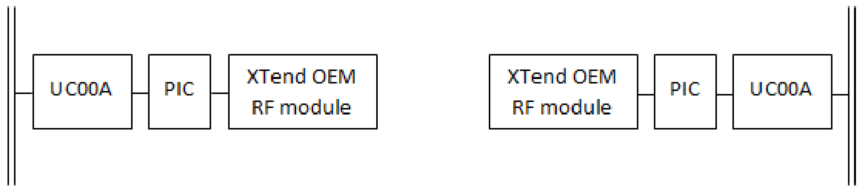


Figure 3.13: Stage 3

3.3 Software

This project involves PC to PC wireless communication using XTend OEM RF modules. Since this is a program centric project, the early part of this project consists of simulations using software such as:

- CCS C Compiler
- Proteus ISIS Professional
- Eltima Virtual Serial Port Driver
- Microsoft Visual Studio

These softwares work together running simulations in such a way that Proteus ISIS Professional and Microsoft Visual Studio are connected by Eltima Virtual Serial Port Driver to send and receive data serially between the two softwares.

3.3.1 CCS C Compiler

CCS provides a complete integrated tool suite for developing and debugging embedded applications running on Microchip PIC® MCUs and dsPIC® DSCs. The heart of this development tools suite is the CCS intelligent code optimizing C compiler which frees developers to concentrate on design functionality instead of having to become an MCU architecture expert. The C language program is written and compiled using this software to generate the hex file which would then be programmed into the PIC18F14K50.

3.3.2 Proteus ISIS Professional

Proteus ISIS Professional is a complete electronics design system which lets users simulate entire microprocessor/microcontroller designs running on actual processor machine code in real-time. It features a range of simulator models for popular microcontrollers, and a set of animated models for related peripheral devices such as LED and LCD displays, and keypads.

Two special features which are the highlights in the simulations are the COMPIM and virtual terminal functions. COMPIM is a COM port physical interface model which acts as a RS232 terminal. The virtual terminal acts like some sort of GUI whereby user can input data and it can also read data received. In other words, it forms the basis of the GUI later.

3.3.3 Eltima Virtual Serial Port Driver

Virtual Serial Port Driver (VSPD) emulates physical COM ports by creating virtual serial ports and connects them in pairs via virtual null-modem cable. Applications on both ends of the pair will be able to exchange data in such a way, that everything written to the first port will appear in the second one and backwards. All virtual ports work and behave exactly like real ones, emulating all their settings.

3.3.4 Microsoft Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

For this project, it is used to build the GUIs using the C# language in which it will input data and display the output. A unique function to note here is the serialPort tool. The serialPort tool enables the GUI to be connected to USB devices to transmit data over the COM port.

3.4 C Programming

The C language is a general purpose computer programming language mainly used to develop application software. In this project, the C language is chosen to develop the XTEA encryption/decryption routine for the microcontroller. Table 3.1 shows the comparison of the C language with other common programming language; assembly and basic, in terms of code complexity, software speed, program size, and development time for a basic application.

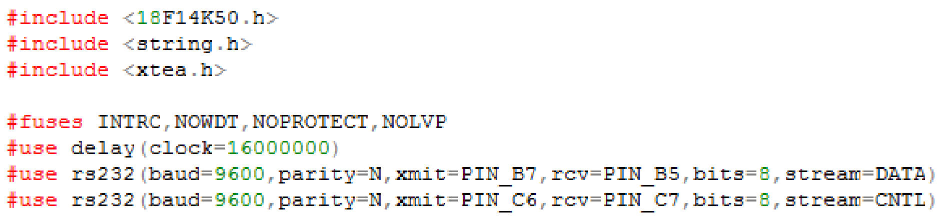
Language	Assembly	Basic	C
Code complexity	Hard	Easy	
Speed	Very fast	Slow	Fast
Program size	Very small	Big	Small
Development time	Long	Short	

Table 3.1: Programming language comparison

From the table above it is no surprise why the C language is the most widely used programming language. There are also other variations to the C language such as C# and C++.

3.4.1 Program Code

The principle of the XTEA cryptography algorithm has been explained in Chapter 2 (2.4.1 Mode of Operation). Figure 3.14 shows the header for the program.



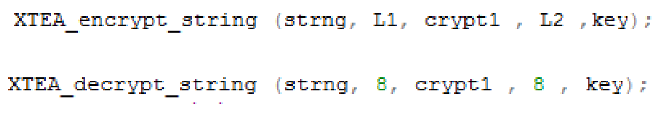
```
#include <18F14K50.h>
#include <string.h>
#include <xtea.h>

#fuses INTRC,NOWDT,NOPROTECT,NOLVP
#use delay(clock=16000000)
#use rs232(baud=9600,parity=N,xmit=PIN_B7,rcv=PIN_B5,bits=8,stream=DATA)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=CNTL)
```

Figure 3.14: Header

#use rs232 is a directive that tells the compiler the baud rate and other options like transmit, receive and enable pins. More than one rs232 statements can be used to specify different streams. For this project, it is used twice to differentiate the data from PC to microcontroller and vice versa, and to and fro from microcontroller to XTend OEM RF module. Pins RB7 and RC6 transmits data out from the microcontroller while pins RB5 and RC7 receives data into the microcontroller.

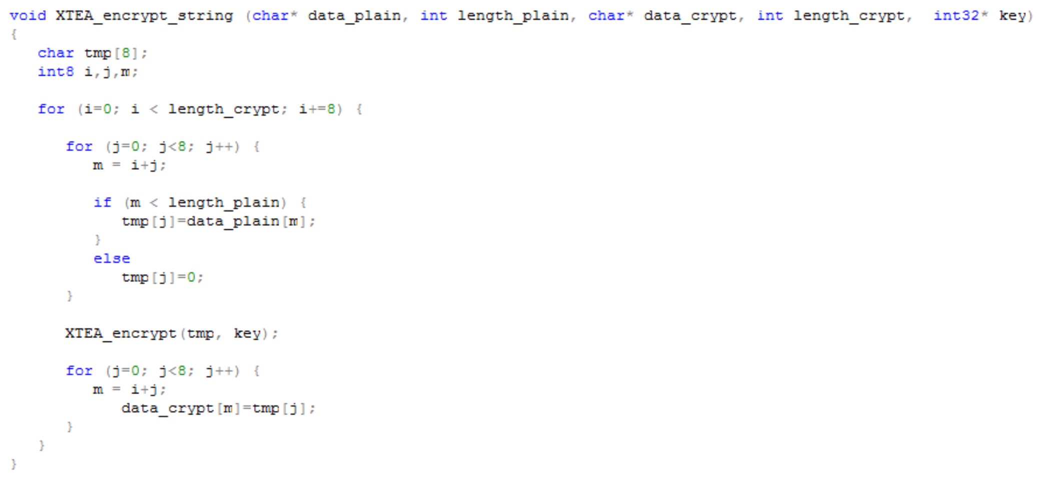
One point to note here however is the use of the xtea.h driver. This driver contains the XTEA algorithm which forms the back bone of the program. Whenever the following lines are called, these routines below in the xtea.h driver is executed.

XTEA_encrypt and XTEA_decrypt are the algorithms as have been stated in Chapter 2.



```
XTEA_encrypt_string (strng, L1, crypt1 , L2 ,key);

XTEA_decrypt_string (strng, 8, crypt1 , 8 , key);
```

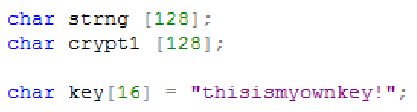
Figure 3.15: Function to call for XTEA



```
void XTEA_encrypt_string (char* data_plain, int length_plain, char* data_crypt, int length_crypt,  int32* key)
{
   char tmp[8];
   int8 i,j,m;

   for (i=0; i < length_crypt; i+=8) {

      for (j=0; j<8; j++) {
         m = i+j;

         if (m < length_plain) {
            tmp[j]=data_plain[m];
         }
         else
            tmp[j]=0;
      }

      XTEA_encrypt(tmp, key);

      for (j=0; j<8; j++) {
         m = i+j;
            data_crypt[m]=tmp[j];
      }
   }
}
```

Figure 3.16: Program code for string encryption

As can be seen in Figure 3.16, the plaintext is represented as string, and the length of both the plaintext and encrypted data are set to blocks of 8 bytes. This means that even though the input data is less than 8 bytes long (less than 8 characters) it will still be encrypted to a length of 8 bytes. If however the input data is more than 8 bytes long (more than 8 characters) the data will be encrypted to full two sets (meaning two blocks).



```
char strng [128];
char crypt1 [128];

char key[16] = "thisismyownkey!";
```

Figure 3.17: Key

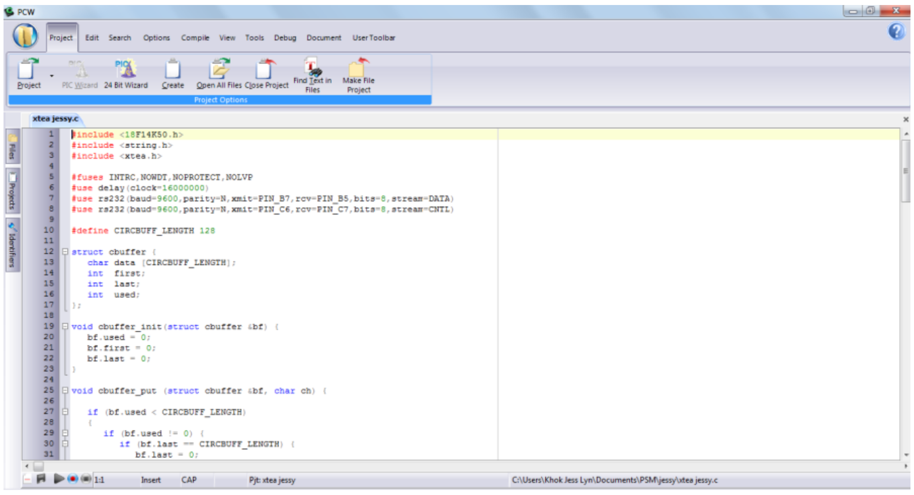
The key length is 128 bit. For this program, the key is set as ‘thisismyownkey!’ which has 16 characters. These 16 characters when entered are translated into a routine of decryption whereby the output is 32 hexadecimal characters. This key functions like a password in which case if the ciphertext is to be decrypted to its original data, the correct key must be entered.

3.5 Software Development

Simulations play a very huge part in completing this project. In fact, a big chunk of time is invested in simulating the project to especially test out the functionality of the program. By simulating first the project, time is saved in troubleshooting the software and virtual hardware before implementing it into the real hardware. The simulations are run in steps by slowly building it up to the end product.

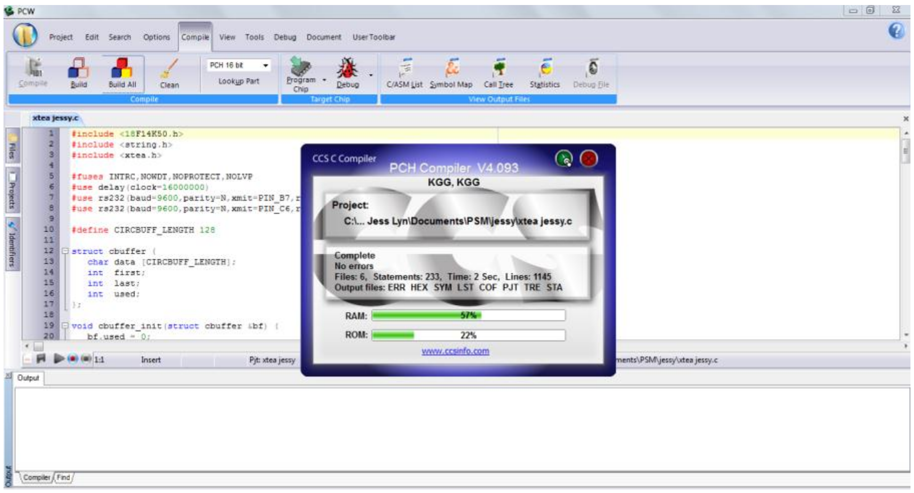
3.5.1 Step 1: Program

The program is written and compiled using CCS C Compiler to generate the hex file. Figure 3.18 shows the program written while Figure 3.19 shows the message box upon successful compilation. The hex file generated is saved by the name of ‘xtea jessy’ at the destination address.



```
#include <18F14K50.h>
#include <string.h>
#include <xtea.h>

#fuses INTRC,NOWDT,NOPROTECT,NOLVP
#use delay(clock=16000000)
#use rs232(baud=9600,parity=N,xmit=PIN_B7,rcv=PIN_B5,bits=8,stream=DATA)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=CNTL)

#define CIRCBUFF_LENGTH 128

struct cbuffer {
   char data [CIRCBUFF_LENGTH];
   int  first;
   int  last;
   int  used;
};

void cbuffer_init(struct cbuffer &bf) {
   bf.used = 0;
   bf.first = 0;
   bf.last = 0;
}

void cbuffer_put (struct cbuffer &bf, char ch) {

   if (bf.used < CIRCBUFF_LENGTH)
   {
      if (bf.used != 0) {
         if (bf.last == CIRCBUFF_LENGTH) {
            bf.last = 0;
```

Figure 3.18: Program in CCS C Compiler



```
CCS C Compiler
PCH Compiler  V4.093
KGG, KGG
Project:
   C:\... Jess Lyn\Documents\PSM\jessy\xtea jessy.c
Complete
No errors
Files: 6,  Statements: 233,  Time: 2 Sec,  Lines: 1145
Output files: ERR  HEX  SYM  LST  COF  PJT  TRE  STA
RAM: 57%
ROM: 22%
www.ccsinfo.com
```

Figure 3.19: Successful compilation

3.5.2 Step 2: Test Run on Virtual Circuit

A circuit which mirrors the actual circuit is constructed in Proteus ISIS Professional as shown in Figure 3.20. Even though there are no tools to represent the XTend OEM RF modules or any other wireless modules, the two PICs are connected by wire, assuming that they are communicating wirelessly. The hex file generated is 'programmed' into the PIC as shown in Figure 3.21. VT1 and VT2 in Figure 3.20 are the representation of GUI.

When the data received in VT2 is as expected from the input in VT1, the program is said to be successful. The PIC encrypts the input in VT1 before transmitting it to the second PIC which would then decrypt the data it receives. The decrypted data is displayed in VT2.

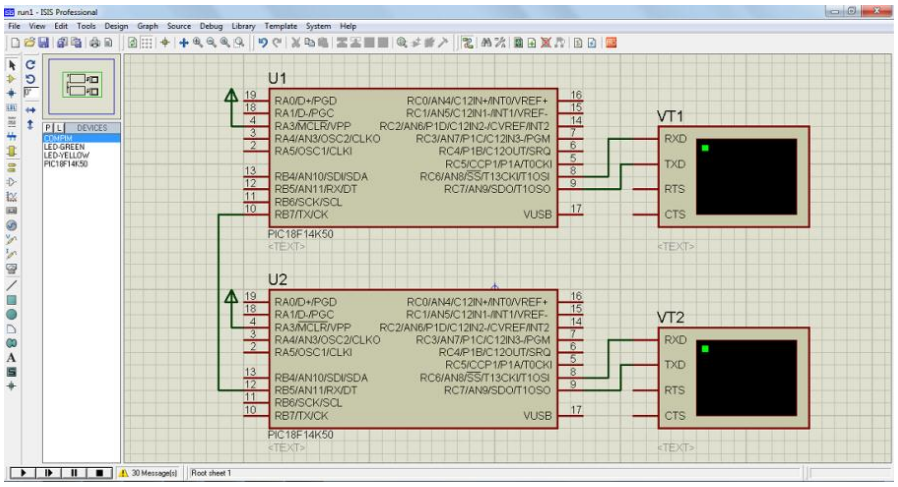
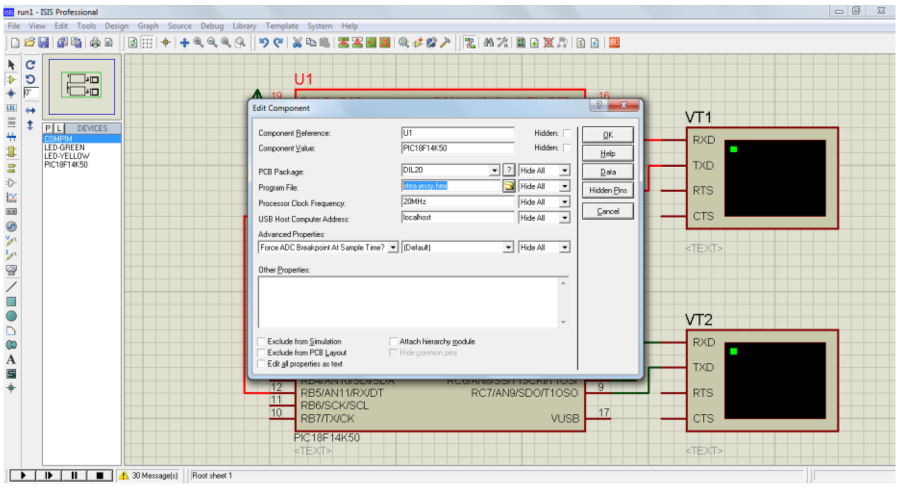


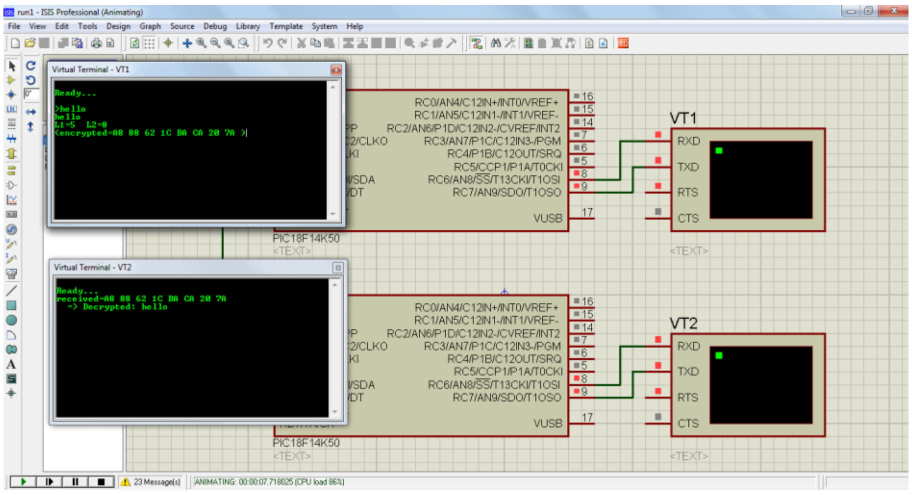
Figure 3.20: Circuit in Proteus ISIS Professional


Edit Component

Component Reference: U1 Hidden: ☐
Component Value: PIC18F14K50 Hidden: ☐
PCB Package: DIL20 Hide All
Program File: xtea jessy.hex Hide All
Processor Clock Frequency: 20MHz Hide All
USB Host Computer Address: localhost Hide All
Advanced Properties:
Force ADC Breakpoint At Sample Time? (Default) Hide All
Other Properties:

☐ Exclude from Simulation ☐ Attach hierarchy module
☐ Exclude from PCB Layout ☐ Hide common pins
☐ Edit all properties as text

Figure 3.21: 'Programming' the PIC



```
Ready...

>hello
hello
L1=5  L2=8
<encrypted=A8 88 62 1C BA CA 20 7A >
```

```
Ready...
received-A8 88 62 1C BA CA 20 7A
  => Decrypted: hello
```

Figure 3.22: Circuit running on xtea jessy.hex program

3.5.3 Step 3: GUI

Two simple GUIs, one to send data and another to receive data, are constructed in Microsoft Visual Studio using the C# language. The serialPort's COM port is set to the other half of the virtual COM port pair.

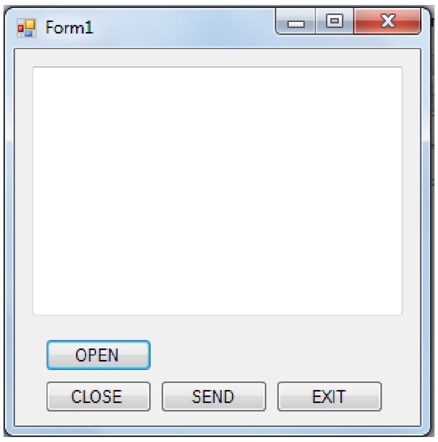


Figure 3.23: GUI to send data serially over COM port

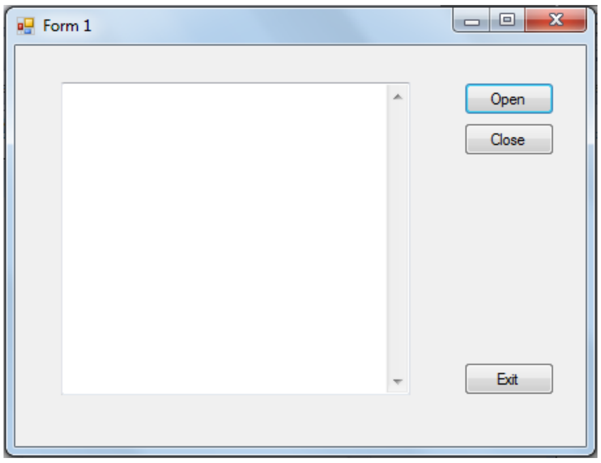


Figure 3.24: GUI to receive data serially over COM port

3.5.4 Step 4: Serial Port Communication

The virtual terminals in the Proteus ISIS Professional simulation are substituted with COMPIMs as shown in Figure 3.25 below. In this step, the VSPD acts as a bridge connecting the GUIs in Microsoft Visual Studio (serialPort) and the virtual circuit in Proteus ISIS Professional (COMPIM) by creating virtual COM port pairs.

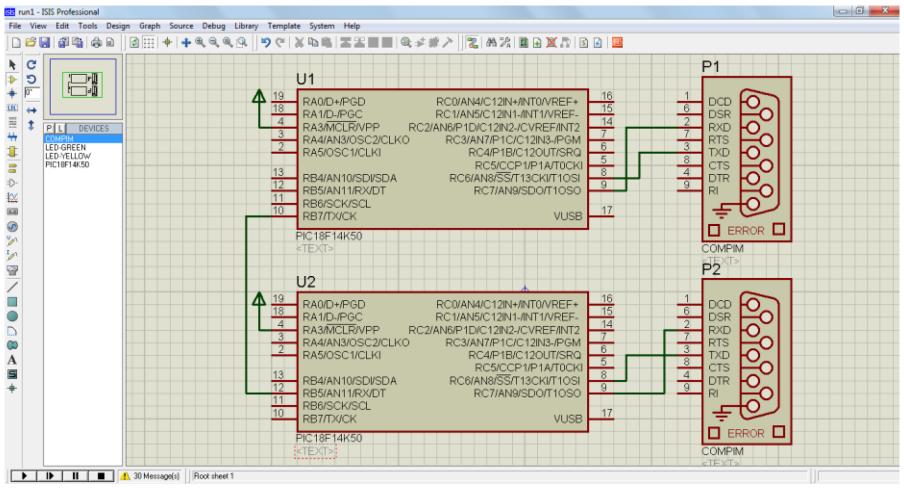
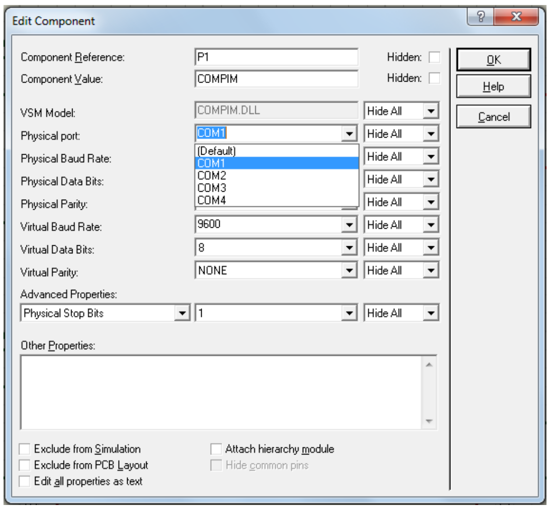


Figure 3.25: Circuit with COMPIM


Edit Component

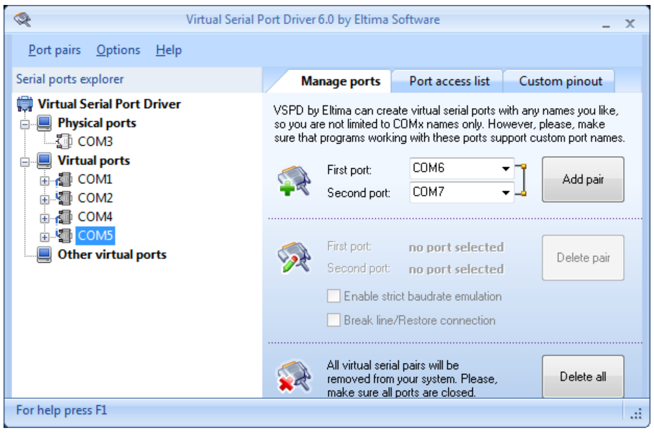
Component Reference:	P1	Hidden: ☐	
Component Value:	COMPIM	Hidden: ☐	
VSM Model:	COMPIM.DLL	Hide All	
Physical port:	COM1	Hide All	
Physical Baud Rate:	(Default) / COM1 / COM2 / COM3 / COM4	Hide All	
Physical Data Bits:		Hide All	
Physical Parity:		Hide All	
Virtual Baud Rate:	9600	Hide All	
Virtual Data Bits:	8	Hide All	
Virtual Parity:	NONE	Hide All	
Advanced Properties: Physical Stop Bits	1	Hide All	

Other Properties:

☐ Exclude from Simulation ☐ Attach hierarchy module
☐ Exclude from PCB Layout ☐ Hide common pins
☐ Edit all properties as text

Figure 3.26: Edit COMPIM properties

Figure 3.26 shows the COMPIM's properties which can be edited by user. For the simulation, the baud rate is set to 9600 and the COM port is set according to the virtual COM port created such as shown in Figure 3.27.


Virtual Serial Port Driver 6.0 by Eltima Software

VSPD by Eltima can create virtual serial ports with any names you like, so you are not limited to COMx names only. However, please, make sure that programs working with these ports support custom port names.

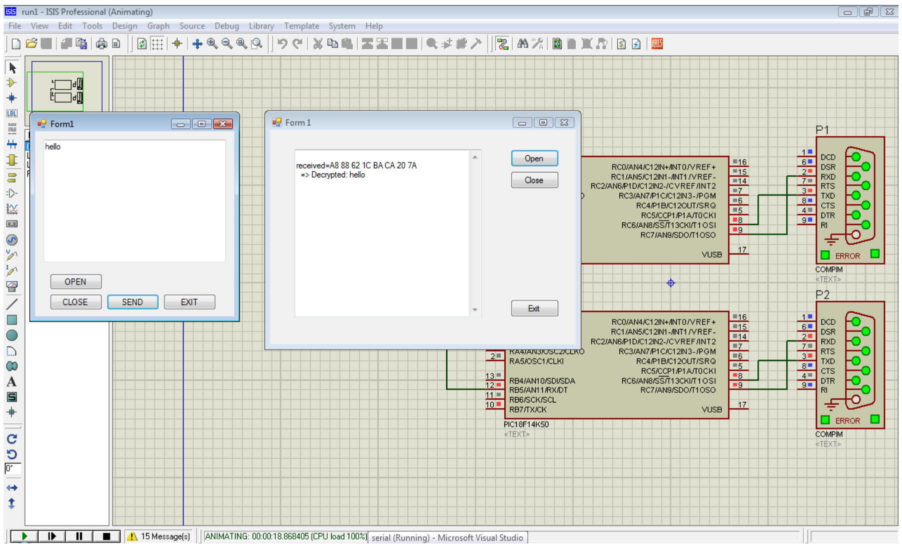
First port: COM6
Second port: COM7

First port: no port selected
Second port: no port selected

☐ Enable strict baudrate emulation
☐ Break line/Restore connection

All virtual serial pairs will be removed from your system. Please, make sure all ports are closed.

Figure 3.27: Eltima Virtual Serial Port Driver


run1 - ISIS Professional (Animating)
File View Edit Tools Design Graph Source Debug Library Template System Help
Form1
OPEN
CLOSE
SEND
EXIT
Form 1
Open
Close
Exit
P1
P2
DCD
DSR
RXD
RTS
TXD
CTS
DTR
RI
ERROR
COMPIM
PIC18F14K50
VUSB
15 Message(s)
ANIMATING: 00:00:18.868405 (CPU load 100%)
serial (Running) - Microsoft Visual Studio

```
hello
```

```
received=A8 88 62 1C BA CA 20 7A
  => Decrypted: hello
```

Figure 3.28: Simulation using all software

The simulation is then performed by running the GUIs in Microsoft Visual Studio and the virtual circuit in Proteus ISIS Professional as shown in Figure 3.28 above. Once the simulation is successful, producing the desired output, the simulation is translated into the hardware.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

The objective of the project is to produce a safe measure for wireless communication. The embedded PC based wireless communication using XTEA is a method which encompasses both software and hardware. PC based communication is achieved by the GUI where user input data and data is displayed.

When data is entered in the GUI, it is transmitted serially to the microcontroller through the UC00A USB to UART converter. The microcontroller then encrypts the data using the XTEA cryptography algorithm producing a ciphertext. The ciphertext is transmitted wirelessly from one PC to another with the XTend OEM RF modules acting as the transmitter and receiver. Unless the correct key is entered, the ciphertext remains encrypted. The original data after it has been decrypted is then displayed in the GUI in the PC. Figure 4.1 shows the hardware and figure 4.2 shows the GUI.

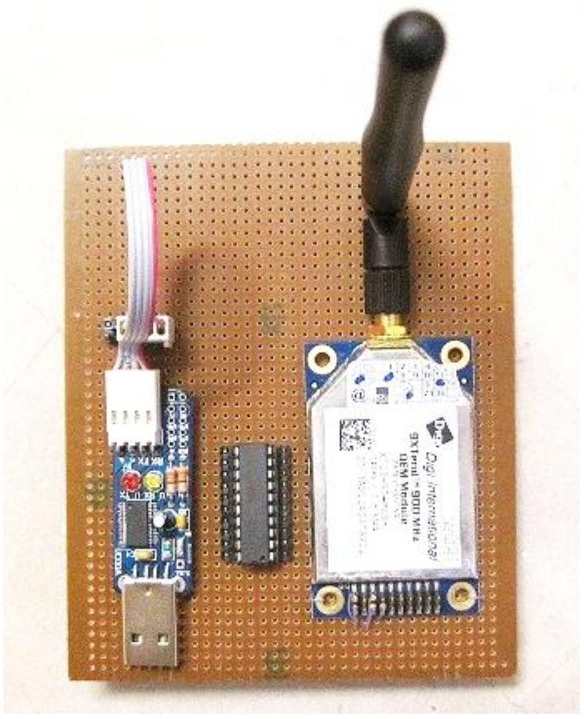


Figure 4.1: Hardware

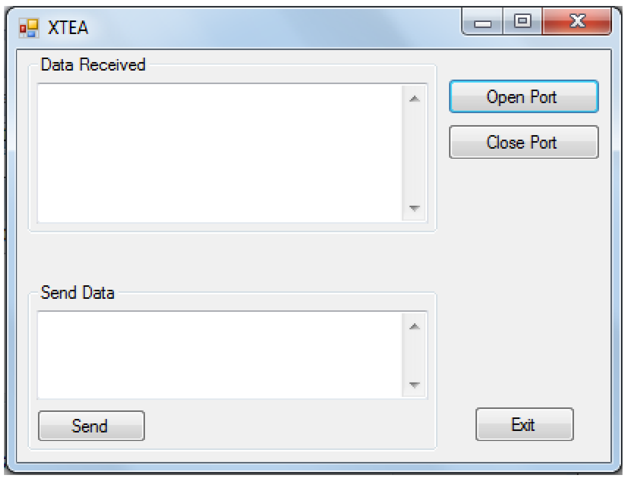


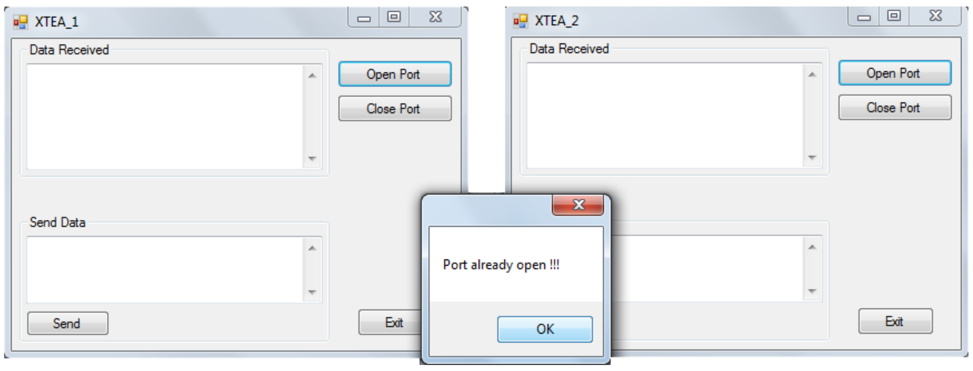
Figure 4.2: GUI

4.2 PC to PC Wireless Communication

Users are able to communicate in a distance wirelessly through the XTend OEM RF modules. However, the confidentiality of the data transmitted is protected by the XTEA. The analysis of the encryption/decryption process is discussed below.

4.2.1 XTEA Encryption/Decryption

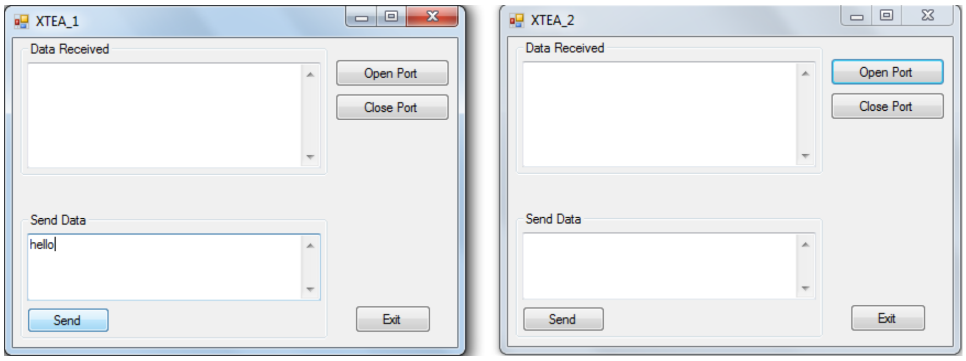
The figures below show the communication between the two PC. XTEA_1 is the GUI for PC1 while XTEA_2 is the GUI for PC2. First, just like during simulation, the COM port is opened. Should the 'Open Port' button clicked more than once, a message box notifying that the COM port has already been opened will appear as shown in Figure 4.3.



Port already open !!!

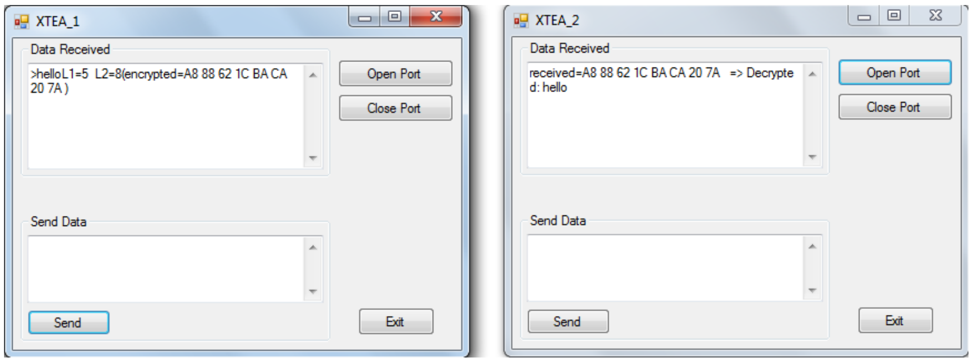
Figure 4.3: Message box

The figures below show how to send data and how data is received step by step. When the user input data in the 'Send Data' text box such as shown in Figure 4.4, upon clicking the 'Send' button the word 'hello' is transmitted from the PC to the microcontroller to be encrypted. The encrypted data is then transmitted wirelessly by the XTend OEM RF module to another. The latter would then receive the encrypted data and the microcontroller decrypts it before being transmitted back to the PC. The decrypted message is displayed in the ‘Data Received’ section as shown in Figure 4.5. Figure 4.6 and Figure 4.7 show the data being sent from the receiver PC instead.



```
hello
```

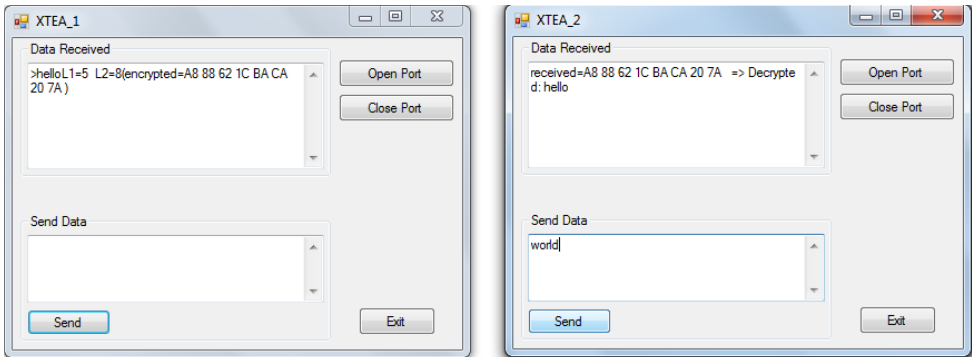
Figure 4.4: Input from PC1



```
>helloL1=5  L2=8(encrypted=A8 88 62 1C BA CA
20 7A )
```

```
received=A8 88 62 1C BA CA 20 7A   => Decrypte
d: hello
```

Figure 4.5: Data sent from PC1

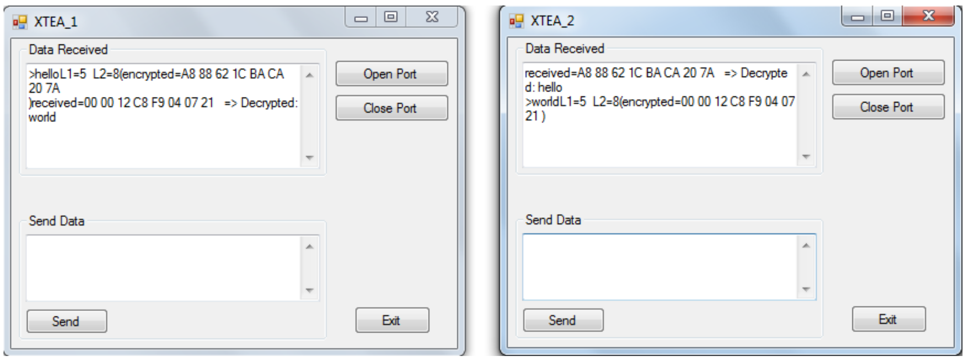


```
>helloL1=5  L2=8(encrypted=A8 88 62 1C BA CA
20 7A )
```

```
received=A8 88 62 1C BA CA 20 7A   => Decrypte
d: hello
```

```
world
```

Figure 4.6: Input from PC2



Data Received

```
>helloL1=5  L2=8(encrypted=A8 88 62 1C BA CA
20 7A
)received=00 00 12 C8 F9 04 07 21   => Decrypted:
world
```

Send Data

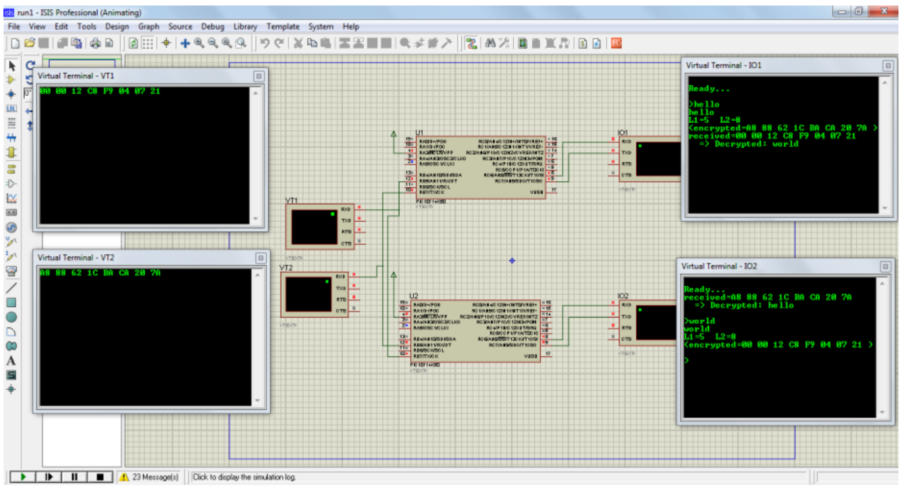
Data Received

```
received=A8 88 62 1C BA CA 20 7A   => Decrypte
d: hello
>worldL1=5  L2=8(encrypted=00 00 12 C8 F9 04 07
21 )
```

Send Data

Figure 4.7: Data sent from PC2

Since, there were no means to intercept the data transmitted wirelessly between the two modules, what could be done to prove that the data is indeed encrypted when being transmitted is to 'read' it from the simulation. As shown in Figure 4.8 below, virtual terminals are placed on the wires connecting the two microcontrollers. What transpired between the input and output of the microcontrollers are captured by the virtual terminals VT1 and VT2 as shown in Figure 4.9.


run1 - ISIS Professional (Animating)
File View Edit Tools Design Graph Source Debug Library Template System Help
VT1
VT2
U1
U2
IO1
IO2
23 Message(s)
Click to display the simulation log.

Virtual Terminal - VT1

```
00 00 12 C8 F9 04 07 21
```

Virtual Terminal - VT2

```
A8 88 62 1C BA CA 20 7A
```

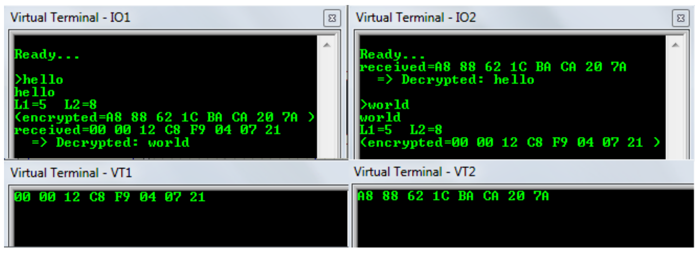
Virtual Terminal - IO1

```
Ready...

>hello
hello
L1=5  L2=8
(encrypted=A8 88 62 1C BA CA 20 7A )
received=00 00 12 C8 F9 04 07 21
  => Decrypted: world
```

Virtual Terminal - IO2

```
Ready...
received=A8 88 62 1C BA CA 20 7A
  => Decrypted: hello

>world
world
L1=5  L2=8
(encrypted=00 00 12 C8 F9 04 07 21 )

>
```

Figure 4.8: Bidirectional circuit



```
Ready...

>hello
hello
L1=5  L2=8
<encrypted=A8 88 62 1C BA CA 20 7A >
received=00 00 12 C8 F9 04 07 21
  => Decrypted: world
```

```
Ready...
received=A8 88 62 1C BA CA 20 7A
  => Decrypted: hello

>world
world
L1=5  L2=8
<encrypted=00 00 12 C8 F9 04 07 21 >
```

```
00 00 12 C8 F9 04 07 21
```

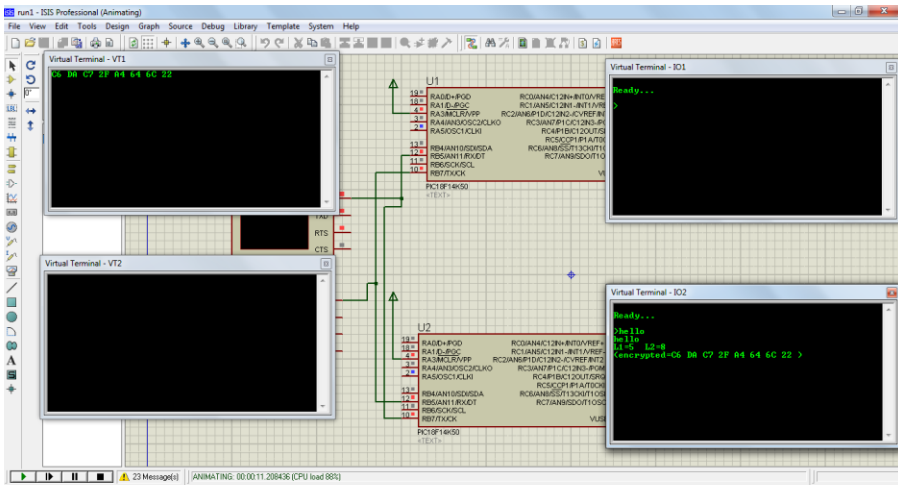
```
A8 88 62 1C BA CA 20 7A
```

Figure 4.9: Virtual terminals

The encrypted data input in IO1 captured by VT2 are the same. The same goes for the encrypted data input in IO2 captured by VT1 are the same. This shows that, the data when transmitted wirelessly would be in its encrypted form also.

4.2.2 Cryptography Key

The cryptography key of XTEA is a 128 bits long key used to ‘unlock’ the encryption. Unless a correct key is entered by the receiver user, the ciphertext remains encrypted. Figure 4.10 shows the simulation when data is entered in IO2. This depicts how it is like when data is sent from one PC but the receiver user has not entered the cryptography key. The data received by the wireless module is in its encrypted form.

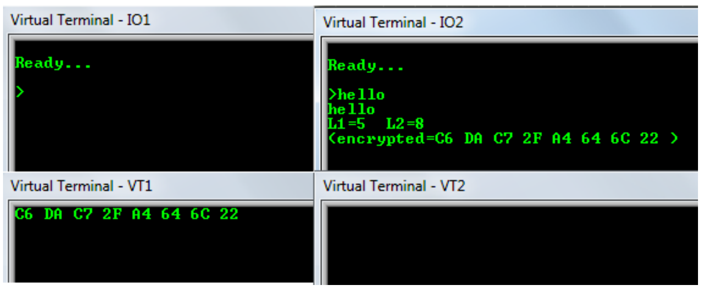

run1 - ISIS Professional (Animating)
File View Edit Tools Design Graph Source Debug Library Template System Help
Virtual Terminal - VT1
Virtual Terminal - VT2
Virtual Terminal - IO1
Virtual Terminal - IO2
U1
U2
PIC18F14K50
RTS
CTS
23 Message(s)
ANIMATING: 00:00:11.208436 (CPU load 88%)

Virtual Terminal - VT1
```
C6 DA C7 2F A4 64 6C 22
```

Virtual Terminal - IO1
```
Ready...
>
```

Virtual Terminal - IO2
```
Ready...
>hello
hello
L1=5  L2=8
<encrypted=C6 DA C7 2F A4 64 6C 22 >
```

Figure 4.10: Data entered


Virtual Terminal - IO1
```
Ready...
>
```

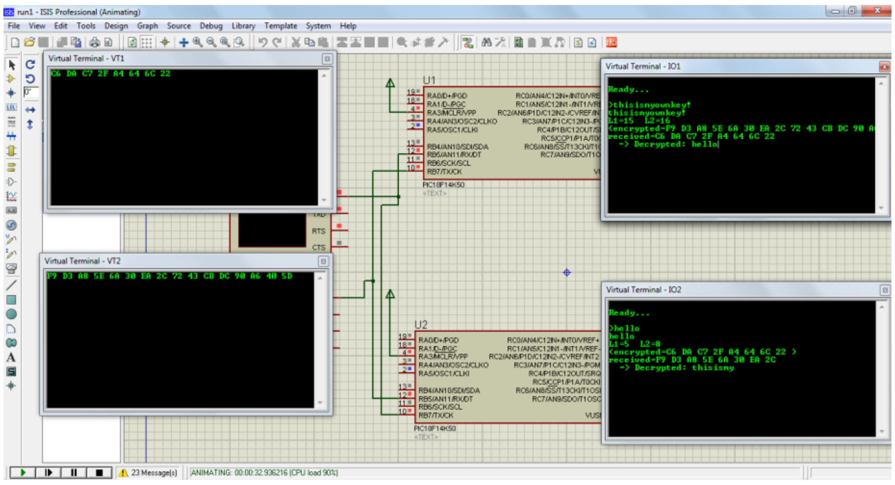
Virtual Terminal - IO2
```
Ready...
>hello
hello
L1=5  L2=8
<encrypted=C6 DA C7 2F A4 64 6C 22 >
```

Virtual Terminal - VT1
```
C6 DA C7 2F A4 64 6C 22
```

Virtual Terminal - VT2

Figure 4.11: Virtual terminals

When the correct key is entered, the ciphertext would then be decrypted to its original data as shown in Figure 4.12 and Figure 4.13.



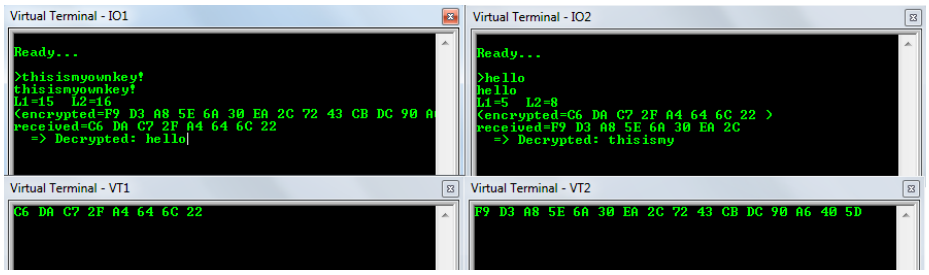
```
Virtual Terminal - VT1
C6 DA C7 2F A4 64 6C 22
```

```
Virtual Terminal - VT2
F9 D3 A8 5E 6A 30 EA 2C 72 43 CB DC 90 A6 40 5D
```

```
Virtual Terminal - IO1
Ready...
>thisismyownkey!
thisismyownkey!
L1=15  L2=16
<encrypted=F9 D3 A8 5E 6A 30 EA 2C 72 43 CB DC 90 A
received=C6 DA C7 2F A4 64 6C 22
  => Decrypted: hello
```

```
Virtual Terminal - IO2
Ready...
>hello
hello
L1=5  L2=8
<encrypted=C6 DA C7 2F A4 64 6C 22 >
received=F9 D3 A8 5E 6A 30 EA 2C
  => Decrypted: thisismy
```

Figure 4.12: Key entered



```
Virtual Terminal - IO1
Ready...
>thisismyownkey!
thisismyownkey!
L1=15  L2=16
<encrypted=F9 D3 A8 5E 6A 30 EA 2C 72 43 CB DC 90 A
received=C6 DA C7 2F A4 64 6C 22
  => Decrypted: hello
```

```
Virtual Terminal - IO2
Ready...
>hello
hello
L1=5  L2=8
<encrypted=C6 DA C7 2F A4 64 6C 22 >
received=F9 D3 A8 5E 6A 30 EA 2C
  => Decrypted: thisismy
```

```
Virtual Terminal - VT1
C6 DA C7 2F A4 64 6C 22
```

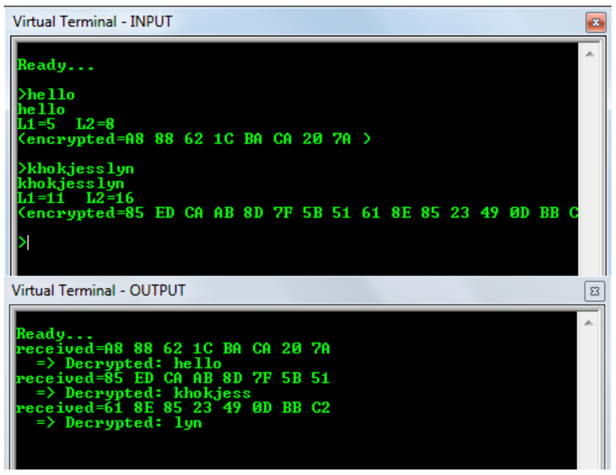
```
Virtual Terminal - VT2
F9 D3 A8 5E 6A 30 EA 2C 72 43 CB DC 90 A6 40 5D
```

Figure 4.13: Virtual terminals

4.2.3 Encryption Length

In XTEA, data is encrypted in blocks of 8 characters; or 64 bits long. Figure 4.14 shows how it is being done. When the data entered is less than 8 characters, as what that has been shown in all the simulations beforehand, the algorithm would still encrypt the data in blocks of 8 characters by considering the remaining spaces as characters. L1 is the length of the data entered by the user while L2 is the length of the ciphertext.

Should the data is longer than 8 characters, the data will be encrypted in the next multiplication of 8. For a data length of 11 characters as what is shown in Figure 4.14, the XTEA will run in two rounds. The data ‘khokjesslyn’ has 11 characters altogether. But since it is longer than the first multiplication of 8, it will be encrypted in the next multiplication of 8, which is in the length of 16. Thus, ‘khokjess’ is produced from the decryption of the first round of ciphertext and ‘lyn’ the remaining data is produced from the second round.



```
Ready...

>hello
hello
L1=5  L2=8
<encrypted=A8 88 62 1C BA CA 20 7A >

>khokjesslyn
khokjesslyn
L1=11  L2=16
<encrypted=85 ED CA AB 8D 7F 5B 51 61 8E 85 23 49 0D BB C

>
```

```
Ready...
received=A8 88 62 1C BA CA 20 7A
  => Decrypted: hello
received=85 ED CA AB 8D 7F 5B 51
  => Decrypted: khokjess
received=61 8E 85 23 49 0D BB C2
  => Decrypted: lyn
```

Figure 4.14: Virtual terminals

CHAPTER 5

CONCLUSION

5.1 Conclusion

In this modern age, cryptography is ever more advanced and pervasive as the information technology industry grows. It is now widely used for military, political and economic reasons. People are getting very obsessed with information security especially since now that the internet has become an almost integral part in many lives. There are many cryptography methods but the XTEA cryptographic algorithm used for this project outputs a satisfactorily secured wireless data transmission.

The embedded wireless system using XTEA cryptographic algorithm provides a secure measure for PC based wireless data transmission. When data is sent from one user to another the data is encrypted and then only transmitted. The receiver at the other end decrypts the data automatically upon receiving the encryption. The result is pretty impressive as the time taken for the whole process; data encryption, wireless transmission, and data decryption; happens altogether in real time immediately. This is proven true even when tested for the range of more than 300m. Suffice to say, XTEA is able to be implemented in almost any kind of system as its unusually small size and flexibility makes it versatile for many types of application.

This project produces two small and lightweight wireless modules which are easy to use and portable. These are the basics for a PC based wireless communication. The modules can communicate up to a range of 900m due to the restriction of the wireless module. However, for a longer communication distance this matter could be rectified easily by opting for a bigger scale wireless module.

Therefore, the project can be said to have achieved its objective of producing a PC based wireless communication which is fast and easy to use.

5.2 Recommendation

There are a few recommendations to further enhance this project for better application.

The implementation of multiple keys which enables keys unique to each user be used. This would provide a higher security and lesser chances of being hacked. This implementation however would require a database in which the key cannot be determined directly in the C programming.

The successor of XTEA, the Corrected Block Tiny Encryption Algorithm (XXTEA), can replace XTEA as it is said to be more likely to be more efficient for longer messages. The difference between XTEA and XXTEA is that the former’s round function to each word in the block and combines it additively with its leftmost neighbour. The slow diffusion rate of the decryption process was immediately exploited to break the cipher in [10]. XXTEA uses a more involved round function which makes use of both immediate neighbours in processing each word in the block.

REFERENCES

[1] Jens-Peter Kaps, *Chai-tea, Cryptographic Hardware Implementations of xTEA*, Volgenau School of IT&E, George Mason University, Fairfax, VA, USA

[2] David J. Wheeler and Roger M. Needham, (October 1997). *TEA Extensions*, Technical report, Computer Laboratory, University of Cambridge.

[3] http://www.logicalsecurity.com/resources/whitepapers/Cryptography.pdf
Accessed: 14 May 2010

[4] Sören Rinne, Thomas Eisenbarth, and Christof Paar *Performance Analysis of Contemporary Light-Weight Block Ciphers on 8-bit Microcontrollers*, Horst G¨ortz Institute for IT Security, Ruhr University Bochum

[5] http://www.ridex.co.uk/cryptology/#_Toc439908852
Accessed: 14 May 2010

[6] Vikram Reddy Andram, (2003), *A Cryptanalysis of the Tiny Encryption Algoritm* University of Alabama

[7] XTend OEM RF Module datasheet
Available at: http://www.digi.com/
Accessed: 26 March 2010

[8] PIC18F14K50 datasheet
Available at:
http://www.datasheetpro.com/1073385_view_PIC18F14K50_datasheet.html
Accessed: 24 March 2010

[9] Matthew D. Russell, (2004). *Tinyness: An Overview of TEA and Related Ciphers*

[10] David J. Wheeler and Roger M. Needham, (October 1998). *Correction to XTEA*, Technical report, Computer Laboratory, University of Cambridge.

[11] Chris Savarese, Brian Hart (1999) *Cryptography: Introduction* Trinity College of Hartford, CT

APPENDIX

Datasheet

1.3. Pin Signals

Figure 1-01. XTend OEM RF Module Pin Numbers

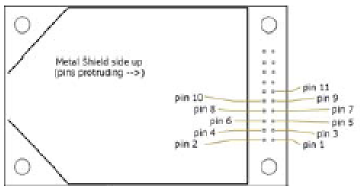


Table 1-03. Pin Signal Descriptions
(Low-asserted signals distinguished with a horizontal line over signal name.)

Pin Number	Mnemonic	I/O	High Impedance during Shutdown	Must Connect	Function
1	GND	-	-	yes	**Ground**
2	VCC	I	-	yes	**Power:** 2.8 - 5.5 VDC
3	GPO2 / RX LED	O	yes	-	**General Purpose Output 2:** <Default (CD=2)> Pin is driven low. Refer to the CD Command [p24] for other configuration options.
					RX LED: Pin is driven high during RF data reception; otherwise, the pin is driven low. Refer to the CD Command [p24] to enable.
4	$\overline{\text{TX}}$_PWR	O	yes	-	**Transmit_Power:** Pin pulses low during RF transmission; otherwise, the pin is driven high to indicate power is on and the module is not in Sleep or Shutdown Mode.
5	DI	I	yes	yes	**Data In:** Serial data entering the module (from the UART host). Refer to the Serial Communications [p9] section for more information.
6	DO	O	yes	-	**Data Out:** Serial Data exiting the module (to the UART host). Refer to the Serial Communications [p9] section for more information.
7	$\overline{\text{SHDN}}$	I	no	yes	**Shutdown:** Pin is driven high during operation and low during Shutdown. Shutdown enables the lowest power mode (~5 µA) available to the module. Refer to the Shutdown Mode [p14] section for more information.
8	GPI2 / SLEEP	I	yes	-	**General Purpose Input 2:** reserved for future use
					SLEEP: By default, SLEEP is not used. To configure this pin to enable Sleep Modes, refer to the Sleep Mode [p15], SM Command [p37] & PW Command [p32] sections.
9	GPO1 / $\overline{\text{CTS}}$ / RS-485 TX_EN	O	yes	-	**General Purpose Output 1:** reserved for future use
					$\overline{\text{CTS}}$ (Clear-to-Send): <Default (CS=0)> When pin is driven low, the UART host is permitted to send serial data to the module. Refer to the Serial Communications [p9] & CS Command [p25] sections for more information.
					RS-485 Transmit Enable: To configure this pin to enable RS-485 half and full-duplex communications. Refer to the Serial Communications [p9] & CS Command [p25] sections.
10	GPI1 / $\overline{\text{RTS}}$ / CMD	I	yes	-	**General Purpose Input 1:** reserved for future use
					$\overline{\text{RTS}}$ (Request-to-Send): By default, is not used. To configure this pin to regulate the flow of serial data exiting the module, refer to the Serial Communications [p9] & RT Command [p36] sections.
					CMD (Command): By default, CMD is not used. To configure this pin to enable binary command programming, refer to the Binary Commands [p18] & RT Command [p36] sections.
11	$\overline{\text{CONFIG}}$ / RSSI	I*	no	-	**Configuration:** Pin can be used as a backup method for entering Command Mode during power-up. Refer to the Command Mode [p17] section for more information.
		O*	no	-	**Receive Signal Strength Indicator:** By default, pin is used as an RSSI PWM output after at the conclusion of the power-up sequence. Refer to the RP Command [p35] for more information.
12-20	reserved / do not connect				

* RF module has 10K Ω internal pull-up resistor

Note: When integrating the module with a Host PC board, all lines not used should be left disconnected (floating).

PIC18F1XK50/PIC18LF1XK50

Device	Program Memory		Data Memory		I/O[1]	10-bit A/D (ch)[2]	ECCP (PWM)	MSSP		EUSART	Comp.	Timers 8/16-bit	USB
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI	Master I^2C^{TM}				
PIC18F13K50/ PIC18LF13K50	8K	4096	512[3]	256	15	11	1	Y	Y	1	2	1/3	Y
PIC18F14K50/ PIC18LF14K50	16K	8192	768[3]	256	15	11	1	Y	Y	1	2	1/3	Y

Note 1: One pin is input only.
2: Channel count includes internal Fixed Voltage Reference (FVR) and Programmable Voltage Reference (CVREF) channels.
3: Includes the dual port RAM used by the USB module which is shared with the data memory.

Pin Diagrams

20-pin PDIP, SSOP, SOIC (300 MIL)

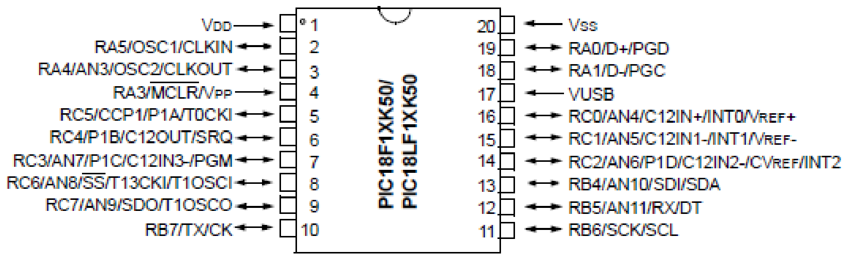


TABLE 1: PIC18F1XK50/PIC18LF1XK50 PIN SUMMARY

Pin	I/O	Analog	Comparator	Reference	ECCP	EUSART	MSSP	Timers	Interrupts	Pull-up	USB	Basic
19	RA0								IOC		D+	PGD
18	RA1								IOC		D-	PGC
4	RA3[1]								IOC	Y		MCLR/VPP
3	RA4	AN3							IOC	Y		OSC2/CLKOUT
2	RA5								IOC	Y		OSC1/CLKIN
13	RB4	AN10					SDI/SDA		IOC	Y		
12	RB5	AN11				RX/DT			IOC	Y		
11	RB6						SCL/SCK		IOC	Y		
10	RB7					TX/CK			IOC	Y		
16	RC0	AN4	C12IN+	VREF+					INT0			
15	RC1	AN5	C12IN1-	VREF-					INT1			
14	RC2	AN6	C12IN2-	CVREF	P1D				INT2			
7	RC3	AN7	C12IN3-		P1C							PGM
6	RC4		C12OUT		P1B							SRQ
5	RC5				CCP1/P1A			T0CKI				
8	RC6	AN8					SS	T13CKI/T1OSCI				
9	RC7	AN9					SDO	T1OSCO				
17											VUSB	
1												VDD
20												VSS

Note 1: Input only

5. PRODUCT SPECIFICATION

UC00A is designed to ease communication between microcontroller and PC. The specifications are as listed below:

5.1 4 ways 2510 header pin

Pin	Label	Definition	Function
1	+	5V Power output from UC00A	5V supply from USB, optional for user to power external device, maximum current 200mA.
2	–	Ground or negative	Ground of power and signal. This pin should be connected to device's GND pin.
3	TX	UC00A UART Transmit pin	This is UC00A's transmitter pin (5V TTL). It should be connected to device's receiver pin.
4	RX	UC00A UART Receive pin	This is UC00A's receiver pin (5V TTL). It should be connected to device's transmitter pin.

Absolute Maximum Rating

Symbol	Parameter	Min	Max	Unit
+	Power output pin	5.0	5.0	V
-	Operating voltage	0	0	V
TX	Transmitter pin of UC00A	0	5.5	V
RX	Receiver pin of UC00A	0	5.5	V

Cautions: "+" on UC00A is 5V supply directly from USB port of computer; it is advised not to use this power source to power application circuit or device. Wrong connection such as wrong polarity, wrong voltage, shorted might permanently damage computer.

Signature	: ______________________________
Author	: KHOK JESS LYN
Date	: 26 NOVEMBER 2010

To my family

ACKNOWLEDGEMENT

Words cannot describe how indebted I am to my supervisor, Encik Mohd Zamri Ibrahim, for all the help and guidance he has given me throughout this project. He is truly one talented man, and is perhaps one of the most gracious person I have ever come across to. For the inspiration, support and mentor he has been, this is my heartfelt gratitude.

To my family and best friends who has been my greatest strength and support throughout my journey here in Universiti Malaysia Pahang, this thesis is my way of saying thank you and appreciating all that they have done for me. They have helped nurture and shape me into who I am today.

ABSTRACT

This project presents an embedded cryptosystem for PC based wireless communication using the XTend OEM RF modules. It focuses mainly on cryptography and its implementation in an embedded system. The Extended Tiny Encryption Algorithm (XTEA) is the chosen encryption method. This is due to the fact that it is a cryptographic algorithm designed to minimize memory footprint and maximize speed. Being flexible and fast in software, XTEA is fast in hardware as well. A Graphical User Interface (GUI) is designed for the application whereby the user communicates with another user through it. It is able to send and also display the data exchange between two PC users. When data is entered in one of the PC, the PIC18F14K50 microcontroller encrypts the data before it is transmitted wirelessly from one PC to another through the XTend OEM RF modules. The ciphertext is then received by the XTend OEM RF module at the other PC but will remain encrypted unless the receiver inputs the correct key. This key is used to decrypt the ciphertext into its original data. When the receiver from the other PC enters the correct key, the ciphertext will be decrypted by the PIC18F15K50 microcontroller and the original data will be displayed in the GUI. The resultant hardware is two sets of identical cryptography wireless module which are able to perform both the encryption and decryption process, connected to the GUI in two different PC. This project produces a secure mean for safe wireless communication on PC level.

ABSTRAK

Projek ini menyampaikan kriptosistem untuk komunikasi tanpa wayar tahap PC dengan menggunakan modul XTend OEM RF. Ia berfokuskan kriptografi dan implikasinya dalamsistem terbenam. Extended Tiny Encryption Algorithm (XTEA) merupakan method enkripsi yang terpilih. Hal ini demikian kerana ia adalah suatu logaritma kriptografi direka untuk meminimakan saiz memori dan memaksimakan kelajuan. Oleh kerana ia fleksibel dan cepat dalam software, XTEA juga adalah cepat dalam hardware. Satu ‘Graphical User Interface’ (GUI) dibina untuk aplikasi di mana pengguna berkomunikasi antara satu sama lain melaluinya. Ia boleh menghantar dan memapar data perhubungan antara dua PC. Apabila data dihantar, mikrokontroller PIC18F14K50 mengenkrip data tersebut sebelum ditransmisi dari satu PC ke PC yang lain melalui modul XTend OEM RF. ‘Ciphertext’ akan diterima oleh modul XTend OEM RF pada PC yang lagi satu tetapi tidak akan didikrip melainkan kata kunci yang betul dimasuk oleh pengguna kedua. Data yang telah didikrip kemudiannya akan terpapar pada GUI. Hasilnya ialah dua set modul tanpa wayar yang boleh menjalani proses enkripsi dan dikripsi dengan menghubung dengan GUI pada PC. Projek ini menghasilkan suatu kaedah yang selamat untuk komunikasi tanpa wayar pada tahap PC.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLE

LIST OF ABBREVIATIONS

XTEA	Extended Tiny Encryption Algorithm
TEA	Tiny Encryption Algorithm
PC	Personal Computer
GUI	Graphical User Interface
PIC	Programmable Interface Controller
XXTEA	Corrected Block Tiny Encryption Algorithm

LIST OF APPENDICES