People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering

Department of Electronics

Project Report Presented in Partial Fulfilment of

the Requirements of the Degree of

'LICENCE'

In Electrical and Electronics Engineering

Title:

PIC Microcontroller-based cryptosystem

Presented By:

- **NOUICER Walid**
- **BOUDJELTHIA Assam**
- **BENGANA Mohammed Nadir**

Supervisor:

Mr A. TERCHI.

Acknowledgment

To our beloved parents, brothers and sisters for their infinite support, consistent encouragement and unshakable trust in us throughout the hardest moments.

To our friends, all those who believed in us and made our years an enjoyable experience.

To all our teachers especially Mr Terchi, for the knowledge they provided us and answering our countless questions.

We dedicate this modest work.

Abstract

In today's world information is generated and transferred faster and greater than any other time in history, and the need for means to secure the enormous amount of information arises which makes encryption a vital process in almost all of data transfers happening in the modern world. In some applications data does not need lot of processing power for encryption, thus microcontrollers are quite enough to do the job.

In this project, a transmitter/receiver system of two PIC microcontrollers capable of encrypting and decrypting data using the eXtended Tiny Encryption Algorithm (XTEA) encryption algorithm. The encryption program is written in C language, and an LCD and a custom made keypad are used for I/O.

Contents

List of figures

List of tables

Introduction

As we move into the twenty first century, the information-processing and telecommunications revolutions set in motion will continue to gather more momentum. The flocks of digital packets containing work notes, statistic information and corporate plans are continually routed through the cyberspace. Most aspects of our lives depend on the successful transactions of these digital letters.

Such processes are based on rendering them into ones and zeroes before broadcasting to specific destinations. How to protect digital money when you can simply write the code of a dollar as easy as writing a word? How could we make a private conversation when our messages are sent to a satellite which broadcasts them so it can be picked from anywhere across the globe? How should a bank know that it is really Mark Zuckerberg requesting from his laptop in New York a transfer of $10,000,000 to another bank?

Fortunately, the mathematical power of cryptography can help. Cryptography provides techniques for keeping information secret, for determining that information has not been tampered with, and for determining who authored that information.

Cryptography with its combination of theory and practical application is a very interesting and attractive subject to study, especially since its practical use is essential in our information based society.

Information-protection protocols designed on theoretical foundations one year appear in products and standards documents the next.

Several encryption algorithms were developed in order to fulfil the demand for security in the domain of telecommunication. These algorithms are made such they can be efficiently programed into computers and microcontrollers.

In this project we developed a system that can encrypt inputted data, send it, decrypt it and display the original data. The system consists of two “stations”. Each station is basically a microcontroller (PIC18F4550) connected to a 20*4 LCD that functions as an output.

- Chapter I gives a general introduction about the history and basics of cryptography and PIC microcontroller .An overview of both with some details of each which we needed in our project.
- Chapter II concerns the PIC18F4550, a general background and the reasons behind choosing it.
- Chapter III covers the hardware part; in other words the components of the cryptosystem, how they are implemented and how they operate.
- In chapter IV we perform a simulation of the system, results are presented giving examples and showing limitations and weaknesses.
- The conclusion provides some remarks about the work carried out.

Chapter I: Cryptography

Encryption refers to the different methods used to make data that can be read and comprehended or "plain text" and render it unreadable in order to protect its contents, the encrypted data is called cipher-text. We use encryption to ensure that information is hidden from anyone for whom it is not intended, even those who can see the encrypted data. The process of reverting cipher-text to its original plaintext is called decryption. [1]

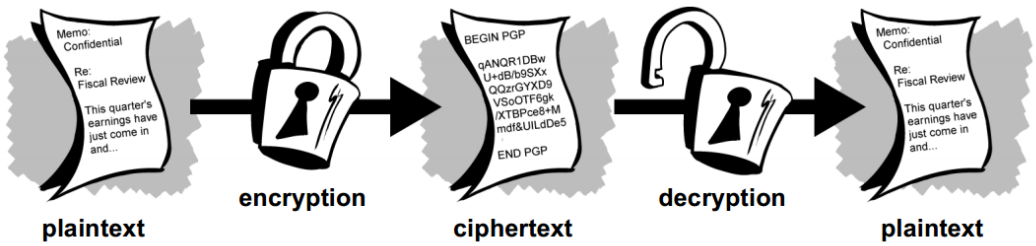

Figure I. 1: Encryption and decryption

The word cryptography comes from the Greek words κρυπτο (hidden or secret) and γραφη (writing). Cryptography is the art of secret writing. More generally, people think of cryptography as the art of mangling information into apparent unintelligibility in a manner allowing a secret method of unmangling. Basically, the main purpose of cryptography is to exchange information between certain individuals without it being acquired by irrelevant parties. [1]

I.1. Brief History of Cryptography

Although cryptography was used by Julius Caesar around 100 BC and by the Egyptians long before that, the first cipher which used an encryption key was supposedly made by Vigenere during the 16 century. In one of his ciphers, the encryption key was repeated multiple times spanning the entire message, and then the cipher text was produced by adding the message character with the key character modulo 26. Though poorly executed, Vigenere's cipher brought the idea of introducing encryption keys into the picture.

In the early 1970's, IBM designed a cipher called Lucifer. In 1973, the Nation Bureau of Standards now called The National Institute of Standards and Technology (NIST) put out a request for proposals for a block cipher which would become a national standard. Lucifer was eventually accepted and was called the Data Encryption Standard (DES).

In the years following 1997 DES was broken by an exhaustive search attack. The main problem with DES was the small size of the encryption key. As computing power increased it became easy to obtain a possible plain text message.

In 1997, NIST again put out a request for proposal for a new block cipher. It received 50 submissions. In 2000, it accepted Rijndael, and christened it as the Advanced Encryption Standard (AES). [2]

I.2. Cryptographic algorithms

A cryptographic algorithm, or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm uses a key—a word, number, or phrase—to encrypt the plaintext. The same plaintext encrypts to different cipher-text with different keys. The security of encrypted data depends on two things: the strength of the cryptographic algorithm and the secrecy of the key. [1]

There are two main types of encryption that are in use:

I.2.a. Conventional cryptography

Also called secret-key or symmetric-key encryption, one key is used both for encryption and decryption. The Data Encryption Standard (DES) is an example of a conventional cryptosystem. Figure I.2 illustrates conventional encryption. [1]

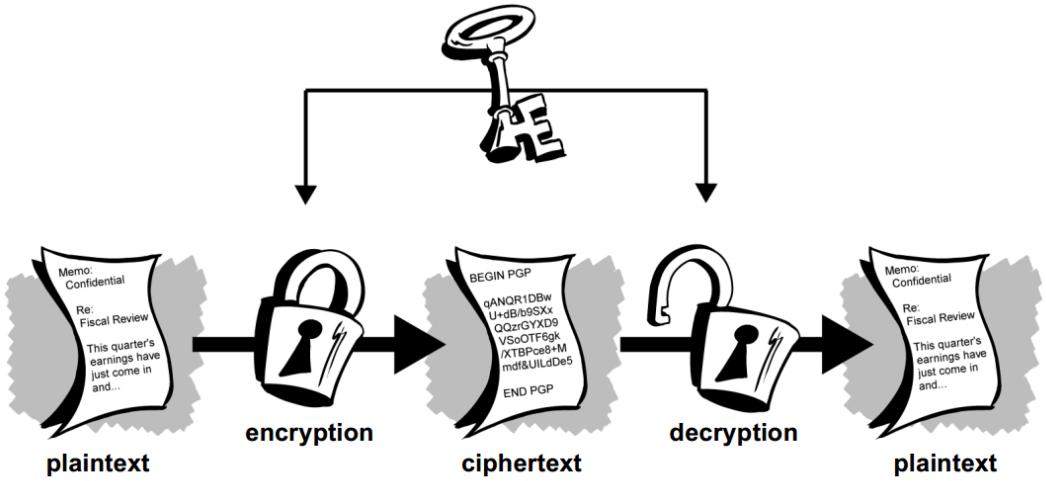

Figure I. 2: Conventional encryption

Advantages of symmetric-key encryption:

- It is faster.
- Encrypted data can be transferred on the link even if there is a possibility that the data will be intercepted. Since there is no key transmitted with the data, the chances of data being decrypted are null.

- Symmetric cryptosystem uses password authentication to prove the receiver's identity.
- A system only which possesses the secret key can decrypt a message.

Disadvantages:

- Symmetric cryptosystems have a problem of key transportation. The secret key is to be transmitted to the receiving system before the actual message is to be transmitted. Every means of electronic communication is insecure as it is impossible to guarantee that no one will be able to tap communication channels. So the only secure way of exchanging keys would be exchanging them personally.
- Cannot provide digital signatures that cannot be repudiated. [3][4]

An important distinction in symmetric cryptographic algorithms is between stream and block ciphers.

- Stream ciphers convert one symbol of plaintext directly into a symbol of cipher-text.
- Block ciphers encrypt a group of plaintext symbols as one block.

Simple substitution is an example of a stream cipher. Columnar transposition is a block cipher.

Most modern symmetric encryption algorithms are block ciphers. [5]

I.2.b. Public key cryptography

The problems of key distribution are solved by public key cryptography, the concept that was introduced by Whitfield Diffie and Martin Hellman in 1975. Public key cryptography is an asymmetric scheme that uses a pair of keys for encryption: a public key, for data encryption, and a corresponding private, or secret key for data decryption. A public key is published to the world while the private key is kept secret. Anyone with a copy of the public key can then encrypt information that only those of possession of the private key can read.

It is computationally infeasible to deduce the private key from the public key. Anyone who has a public key can encrypt in formation but cannot decrypt it. Only the person who has the corresponding private key can decrypt the information. [1]

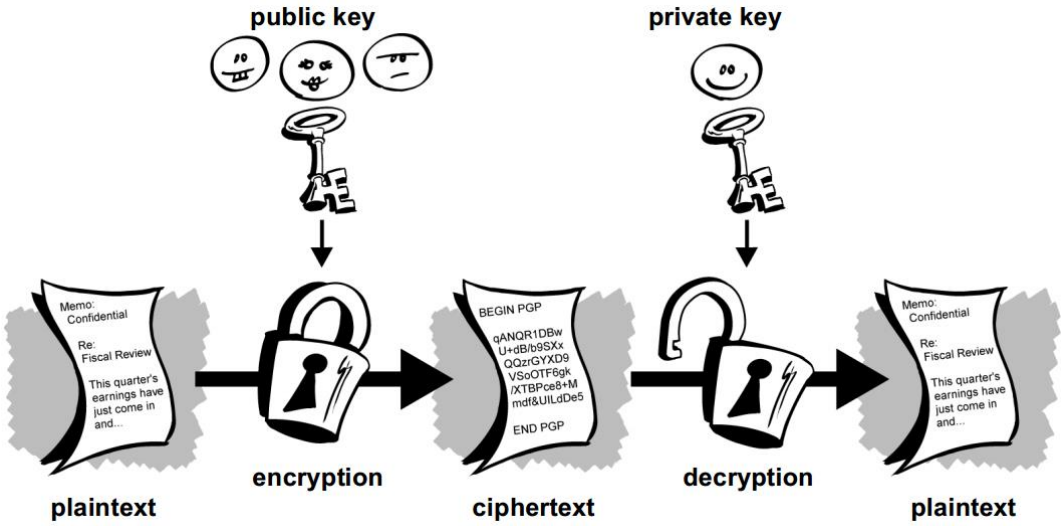

Figure I. 3: Public key encryption

Advantages of public key cryptography cryptosystem:

- In public key or asymmetric cryptography there is no need for exchanging keys, thus eliminating the key distribution problem.
- The primary advantage of public-key cryptography is increased security: the private keys do not ever need to be transmitted or revealed to anyone.
- Can provide digital signatures that can be repudiated

Disadvantages:

- A disadvantage of using public-key cryptography for encryption is speed: there are popular secret-key encryption methods which are significantly faster than any currently available public-key encryption method. [3][4]

I.3. The Tiny Encryption Algorithms

I.3.a. TEA

The Tiny Encryption Algorithm (TEA) is a block cipher that was designed by Wheeler and Needham in 1994 as a short C language program that would run safely on most machines. It has no preset tables or long set up times, and achieves a high performance by performing simple operations on 32-bit words. TEA has a simple Feistel structure, but it uses a large number (i.e. 64) rounds of iterations to make itself secure. Though written in C, TEA can readily be implemented in a range of languages, including assembler, we used in this project Micro C pro however. [6]

Feistel cipher is a symmetric structure used in the construction of block ciphers. This structure has the advantage that encryption and decryption operations are very

similar, even identical in some cases, requiring only a reversal of the key schedule. Therefore the size of the code or circuitry required to implement such a cipher is nearly halved.

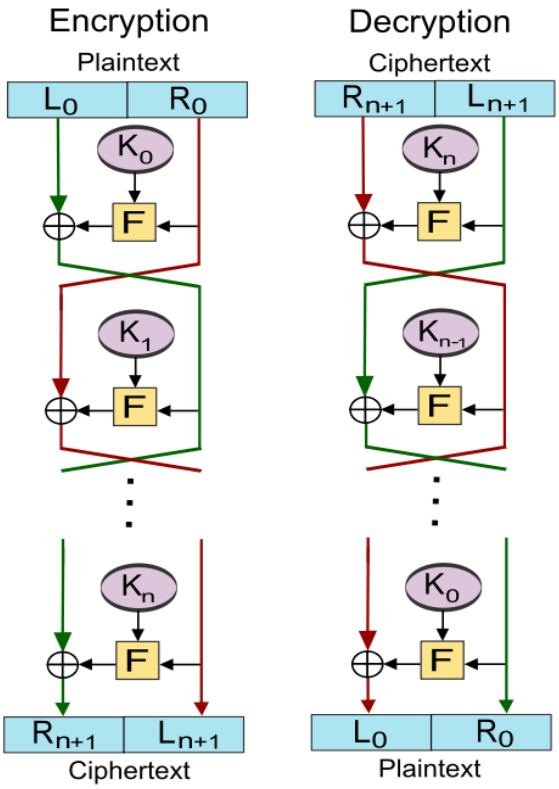

Figure I. 4: Feistel cipher diagram

In figure I.4, F is the round function and let $K_0, K_1, ..., K_n$ be the sub-keys for the rounds 0,1,…..,n respectively. Then the basic operation is as follows:

- Split the plaintext block into two equal pieces, (L_0, R_0)
- For each round i =0,1,….,n, compute:

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

Then the cipher-text is (R_{n+1}, L_{n+1}).

Decryption of a cipher-text (R_{n+1}, L_{n+1}) is accomplished by computing for i=n, n-1,……., 0

$$R_i = L_{i+1}$$

$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$$

Then (L_0, R_0) is the plaintext again. [8]

The Tiny Encryption Algorithm uses operations from mixed (orthogonal) algebraic groups. A dual shift causes all bits of the data and key to be mixed repeatedly. The key schedule algorithm is simple; the 128-bit key K is split into four 32-bit blocks K = (K[0], K[1], K[2], K[3]). TEA seems to be highly resistant to differential cryptanalysis(1) and achieves complete diffusion (where a one bit difference in the plaintext will cause approximately 32 bit differences in the cipher text). Time performance on a workstation is very impressive.

Wheeler et al. (1994) at the computer laboratory of Cambridge University developed the TEA encode routine. Figure I.5 presents the TEA encode routine in C language where the key value is stored in k[0] – k[2] and data are stored in v[0] – v[1]:

```
void code(long* v, long* k)  {
unsigned long y = v[0], z = v[1], sum = 0, /* set up */
delta = 0x9e3779b9, n = 32 ;  /* a key schedule constant */
while (n-->0) {         /* basic cycle start */
  sum += delta ;
  y +=  (z<<4)+k[0] ^ z+sum ^ (z>>5)+k[1] ;
  z +=  (y<<4)+k[2] ^ y+sum ^ (y>>5)+k[3] ;  /* end cycle */
     }
v[0] = y ; v[1] = z ; }
```

Figure I. 5: Encode routine

(1) Cryptanalysis is the science of analysing and breaking secure communication.

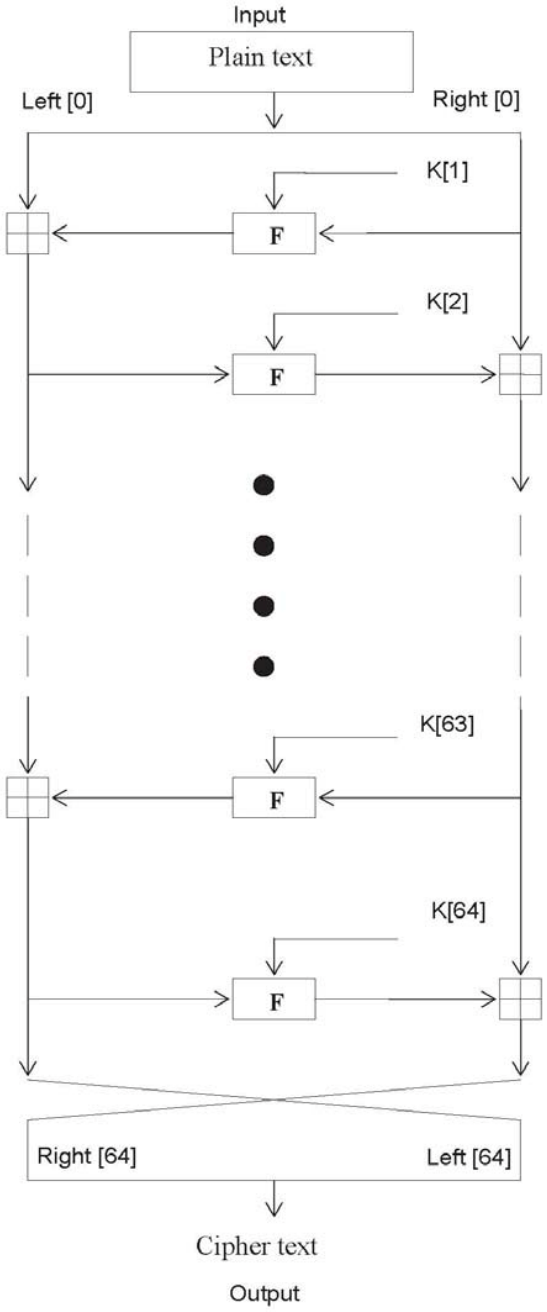

Figure I. 6: The abstract structure of TEA encryption routine

The notations used in Figure I.5 (and other figures) are summarized in Table I. 1.

Table I. 2: Notations

Symbol / Notation	Meaning
$\boxplus$	Addition modulo 2^{32}
$\oplus$	Exclusive-OR
$\ll$	Left shift
$\gg$	Right shift
$\|\|$	Concatenation
$\lfloor x \rfloor$	$\max_{y \in Z}(y \leq x)$, Z is the set of integers
LSB	Least significant bit
MSB	Most significant bit
[i]	Select bit i, i = 0 is the LSB
[j . . . i]	Select bits k where j ≥ k ≥ i, k = 0 is the LSB
0^k	Concatenation of k times the string '0'

Figure I.6 shows the structure of the TEA encryption routine. The inputs to the encryption algorithm consists of a plaintext block and a key K .The plaintext is P = (Left[0], Right[0]) and the cipher text is C = (Left[64], Right[64]). The plaintext block is split into two halves, Left[0] and Right[0]. Each half is used to encrypt the other half over 64 rounds of processing and then combine to produce the cipher text block.

- Each round *i* has inputs Left[*i*-1] and Right[*i*-1], derived from the previous round, as well as a sub key K[*i*] derived from the 128 bit overall K.
- The sub keys K[*i*] are different from K and from each other.
- The constant delta =$\left(\sqrt{5}-1\right) * 2^{31} = 9E3779B9_h$, is derived from the golden number ratio to ensure that the sub keys are distinct and its precise value has no cryptographic significance.
- The round function differs slightly from a classical Fiestel cipher structure in that integer addition modulo 2^{32} is used instead of exclusive-or as the combining operator.

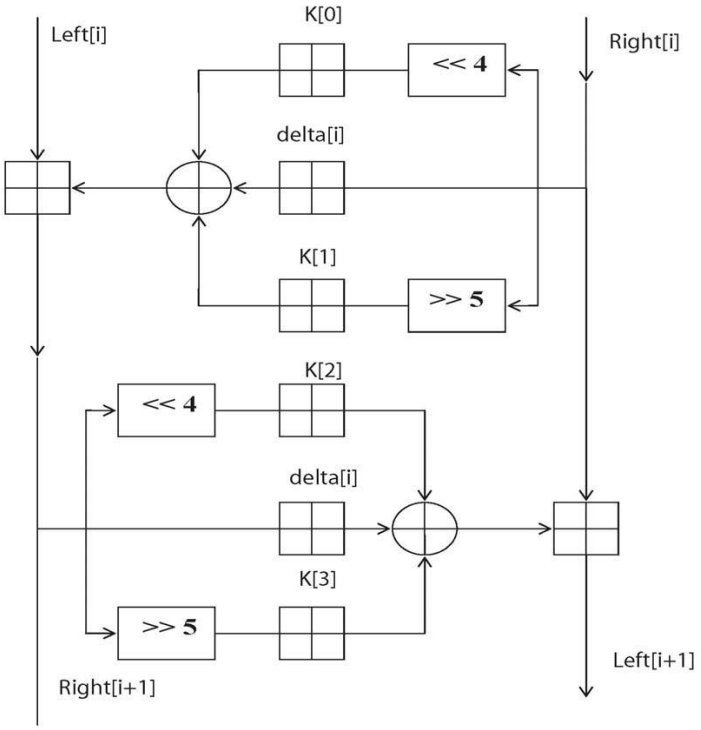

Figure I. 7: An abstraction of i-th cycle of TEA

Figure I.7 presents the internal details of the *i*-th cycle of TEA. The round function, F, consists of the key addition, bitwise XOR and left and right shift operation. We can describe the output (Left[i +1] , Right[i +1]) of the *i*th cycle of TEA with the input (Left[i] ,Right[i]) as follows:

$$\text{Left}\,[\text{i} + 1] = \text{Left[i]} \boxplus \text{F}\,(\,\text{Right[i]}, \text{K}\,[0, 1], \text{delta[i]}\,)$$

$$\text{Right}\,[\text{i} + 1] = \text{Right[i]} \boxplus \text{F}\,(\,\text{Right[i} + 1], \text{K}\,[2, 3], \text{delta[i]}\,)$$

$$\text{delta[i]} = (\text{i} + 1)/2 * \text{delta}$$

The round function, F, is defined by:

F(M, K[j,k], delta[i]) = ((M << 4) K[j]) $\oplus$ (M delta[i]) $\oplus$ ((M >> 5) K[k]).

The round function has the same general structure for each round but is parameterized by the round sub key K[i]. The key schedule algorithm is simple; the 128-bit key K is split into four 32-bit blocks K = (K[0], K[1], K[2], K[3]). The keys K[0] and K[1] are used in the odd rounds and the keys K[2] and K[3] are used in even rounds.

Decryption is essentially the same as the encryption process; in the decode routine the cipher text is used as input to the algorithm, but the sub keys K[i] are used in the reverse order, Figure I.8 shows the decryption routine code in C.

```
void decode(long* v, long* k) {
 unsigned long n = 32, sum, y = v[0], z = v[1], delta = 0x9e3779b9 ;
 sum = delta<<5 ;
 /* start cycle */
 while (n-->0) {
 z - = (y<<4)+k[2] ^ y+sum ^ (y>>5)+k[3] ;
 y -= (z<<4)+k[0] ^ z+sum ^ (z>>5)+k[1] ;
 sum -= delta ; }
 /* end cycle */
 v[0] = y ; v[1] = z ; }
```

Figure I. 8: Decryption routine code

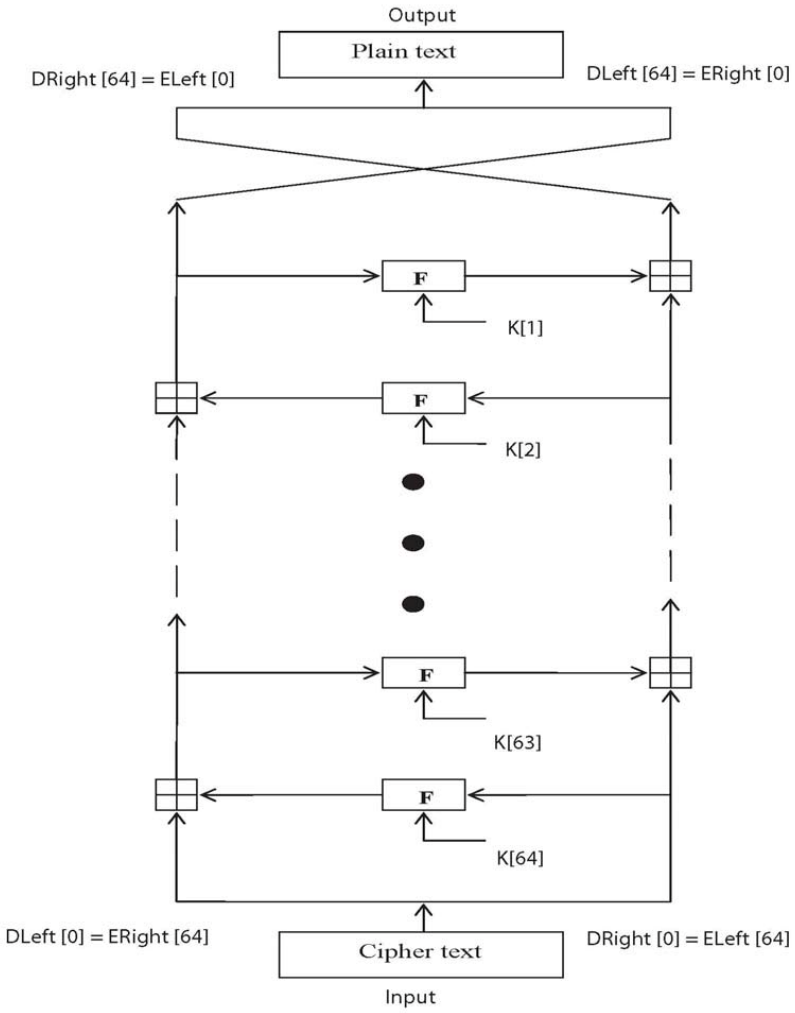

Figure I. 9: The abstract structure of TEA decryption routine

Figure I.9 presents the structure of the TEA decryption routine. The intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. For example, if the output of the nth encryption round is:

ELeft[i] || ERight[i] (ELeft[i] concatenated with ERight[i]).

Then the corresponding input to the (64-i)th decryption round is:

DRight[i] || DLeft[i] (DRight[i] concatenated with DLeft[i]).

After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is ERight[64] || ELeft[64], the output of that round is the final cipher text C. Now this cipher text is used as the input to the decryption algorithm. The input to the first round is ERight[64] || ELeft[64], which is equal to the 32-bit swap of the output of the 64th round of the encryption process.

I.3.b. XTEA

To secure TEA against related-key attacks, Needham and Wheeler presented an extended TEA, known as XTEA, which retains the original objectives of simplicity and efficiency. XTEA accepts a 64-bit block size and a 128-bit user key, and has a total of 64 rounds as well. As one of the fastest and most efficient block ciphers in existence, XTEA is used for some real-life cryptographic applications.

```
XTEA( long * v, long * k, long N) {
unsigned long y = v[0];
unsigned z =v [1];
unsigned DELTA = 0x9e3779b9;
if (N>0) {
 /* coding */
 unsigned long limit = DELTA*N;
 unsigned long 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{
 /* decoding */
 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 I. 10: XTEA routine

Figure I.10 presents the XTEA routine in C language, where the array v represents the plain text of 2 words, the array k represents the key of 4 words, and N contains the value of number of cycles.

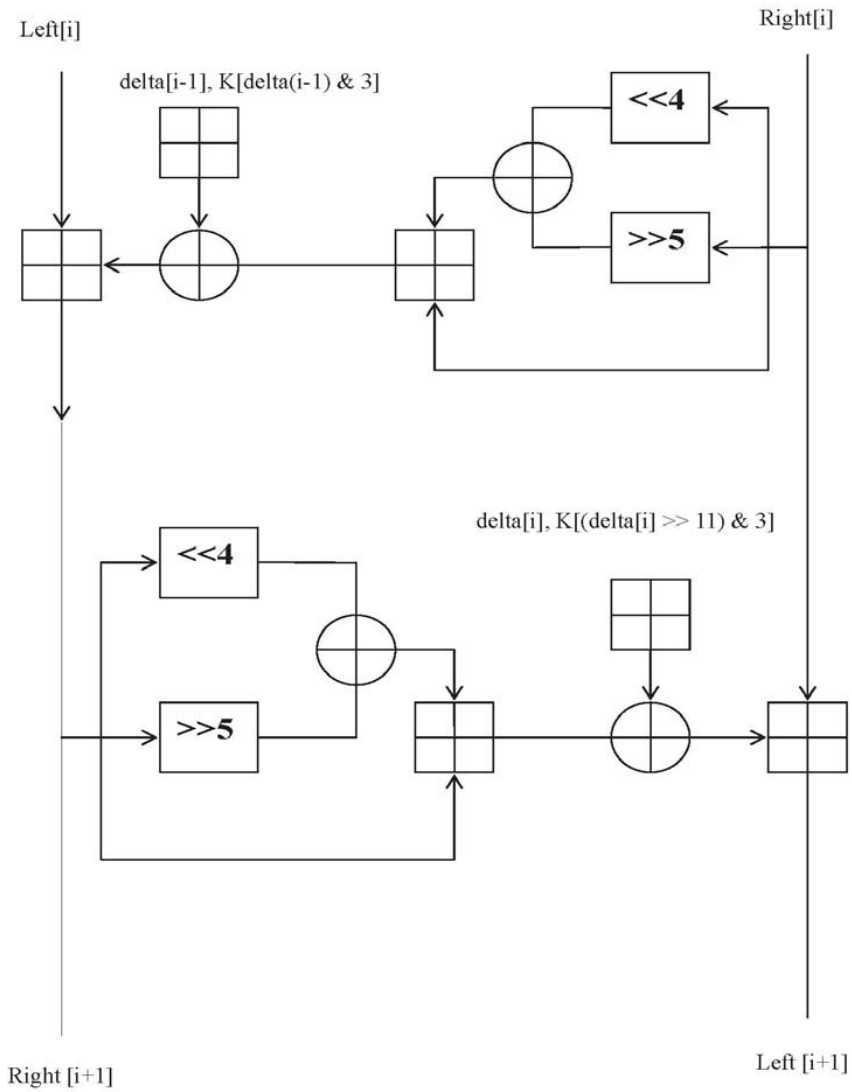

Figure I. 11: An abstraction of i-th cycle of XTEA

Figure I.11 shows the internal details of the i-th cycle of XTEA routine. The relation between the output (Left[i+1], Right[i+1]) and the input (Left[i], Right[i]) for the i-th cycle of XTEA is defined as follows:

$$\mathbf{Left[i+1]} = \mathbf{Left[i]} \boxplus \mathbf{F(\ Right[i], K[2i-1], delta[i-1])}$$

$$\mathrm{Right[i+1]} = \mathrm{Right[i]} \boxplus \mathrm{F(Left[i+1], K[2i], delta[i])}$$

$$\mathbf{delta[i]} = \mathbf{(i+1)/2} * \mathbf{delta}$$

The round function, F, is defined by:

$$F(M, K[*], delta[**]) = ((M<<4) \oplus (M>>5)) M \oplus delta[**] K[*]$$

The round keys are generated according to the algorithm shown in Figure I.12. [7]

- Split the 128-bit key K into four 32-bit blocks

K = (K[0], K[1], K[2], K[3]).

- for rounds r = 1,….,64

if (r is odd) then

key = K[delta((r-1)/2) & 3];

else

if (r is even) then

key = K[(delta((r/2)>>11) & 3]

Figure I. 12: Round key generation algorithm

Chapter II: PIC microcontroller

II.1. PIC characteristics (PIC18F4550):

The PIC18F4550 is a 32KB microcontroller with a 256B of PROM and a frequency up to 48 MHz with 40pin, it's a modern microcontroller that envelop multiple new features such as:

- Alternate Run Modes: By clocking the controller from the Timer1 source or the internal oscillator block, power consumption during code execution can be reduced by as much as 90%.
- Multiple Idle Modes: The controller can also run with its CPU core disabled but the peripherals still active. In these states, power consumption can be reduced even further, to as little as 4%, of normal operation requirements.
- Fail-Safe Clock Monitor: This option constantly monitors the main clock source against a reference signal provided by the internal oscillator. If a clock failure occurs, the controller is switched to the internal oscillator block, allowing for continued low-speed operation or a safe application shutdown.
- Two-Speed Start-up: This option allows the internal oscillator to serve as the clock source from Power-on Reset, or wake-up from Sleep mode, until the primary clock source is available
- Memory Endurance: The Enhanced Flash cells for both program memory and data EEPROM are rated to last for many thousands of erase/write cycles – up to 100,000 for program memory and 1,000,000 for EEPROM. Data retention without refresh is conservatively estimated to be greater than 40 years.
- Self-Programmability: This device can write to its own program memory spaces under internal software control. By using a boot-loader routine, located in the protected Boot Block at the top of program memory, it becomes possible to create an application that can update itself in the field.
- Serial communication: It supports the UART (being used in this project), SPI and I²C protocols. The SPI protocol include 3-wire support with all 4 modes. [9]

Table II. 1: PIC18F4550 characteristics

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP/ ECCP (PWM)	SPP	MSSP		EAUSART	Comparators	Timers 8/16-bit
	FLASH (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I^2C			
PIC18F4550	32K	16384	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3

Table II. 2: PIC18F4550 Features

Features	PIC18F4550
Operating Frequency	DC – 48 MHz
Program Memory (Bytes)	32768
Program Memory (Instructions)	16384
Data Memory (Bytes)	2048
Data EEPROM Memory (Bytes)	256
Interrupt Sources	20
I/O Ports	Ports A, B, C, D, E
Timers	4
Capture/Compare/PWM Modules	1
Enhanced Capture/ Compare/PWM Modules	1
Serial Communications	MSSP, Enhanced USART
Universal Serial Bus (USB) Module	1
Streaming Parallel Port (SPP)	Yes
10-Bit Analog-to-Digital Module	13 Input Channels
Comparators	2
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT
Programmable Low-Voltage Detect	Yes
Programmable Brown-out Reset	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set enabled
Packages	40-pin PDIP 44-pin QFN 44-pin TQFP

II.2. The choice of PIC18F4550:

The reason this chip was chosen is because it can hold more instructions and execute them fast thanks to the 48Mhz clock also the self-programmability gives us the ability to change the encryption algorithm anytime we want and without using more resources and for that matter if we are going to change the encryption algorithm often the memory endurance is very useful. And since our project is based on bidirectional communication between PICs, we need the UART protocol.

Chapter III: Hardware development

III.1. Parts selection:

For this project, we have the following list of components:

- Two PIC18F4550 microcontrollers (one as a MASTER and the second as a SLAVE) which are going to Send/Receive data from/to each other.

  (See figures III.1, III.2).

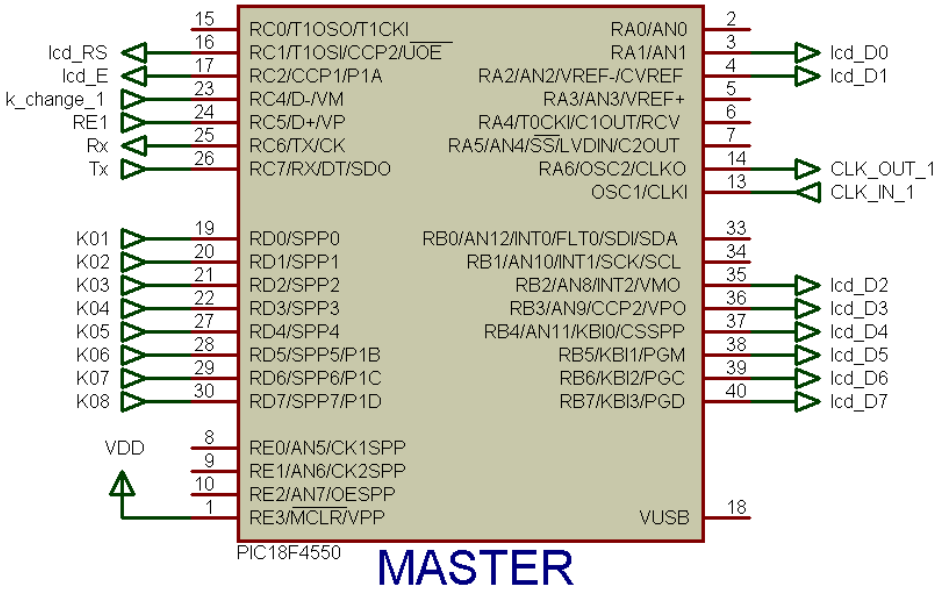

Figure III. 1: Master PIC pins connections

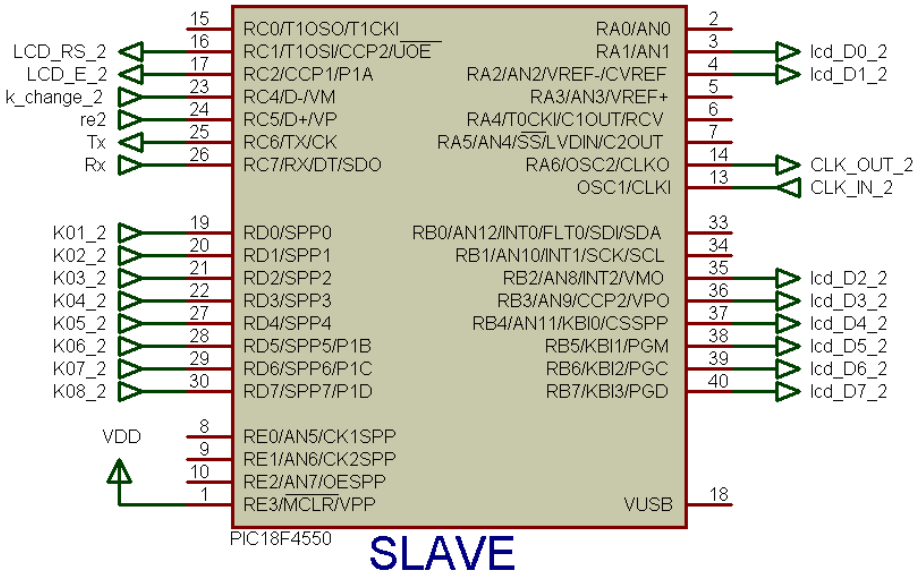

Figure III. 2: Slave PIC pins connections

- Two LCDs (20 x 4 characters in size) that serves as the output of the system.
- Two Keypads for data input. We have managed to add a button for extra characters since the type of keypad that is generally supported by the PIC18 and "MikroC KEYPAD library" is only for 4x4 keypads. By adding this switch we now are able to interface "16 more characters". (See figure III.3). [14][16]

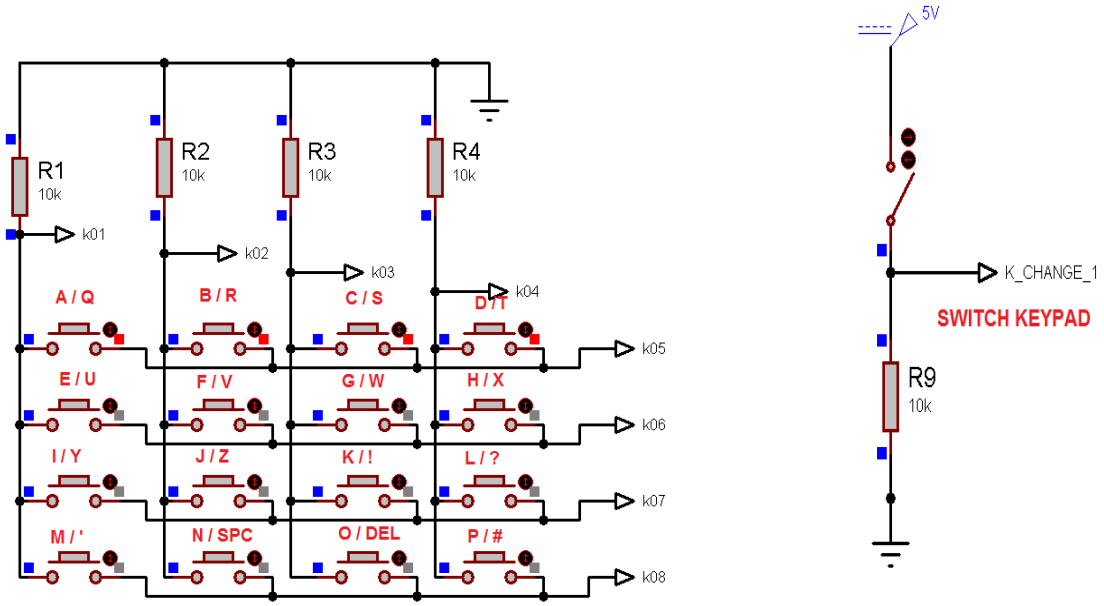

Figure III. 3: 26 characters keypad implementation

III.1.a. PIC Clock selection

For a proper work of PIC18F4550 we use an 20 MHz input clock with a Crystal oscillator with "type HS", Crystal 20MHz and two 15 pF capacitors [13] (see figure III.4)

The clock is connected to pins 13 and 14 as CLK_IN and CLK_OUT respectively.

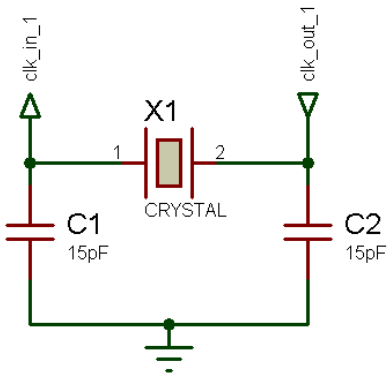

Figure III. 4: Crystal oscillator clock (20 MHz)

III.2. Interfacing components with PIC:

Keypad: Our keypad has 8 pins which are connected to PORTD of the microcontroller [RD0 to RD7], key switcher is assigned to pin 23 [RC4]. It's designed using buttons only that the PIC will keep scanning and assigning an ASCII code (a character) for each case (see chapter IV) (see figure III.5).LCD: because there's no free PORT of 8 pins, we interfaced LCD to [RA1-RA2 and RB2-RB7 for LCD D0-D7 pins and LCD RS-E with RC1-RC2]. (See figure III.5)

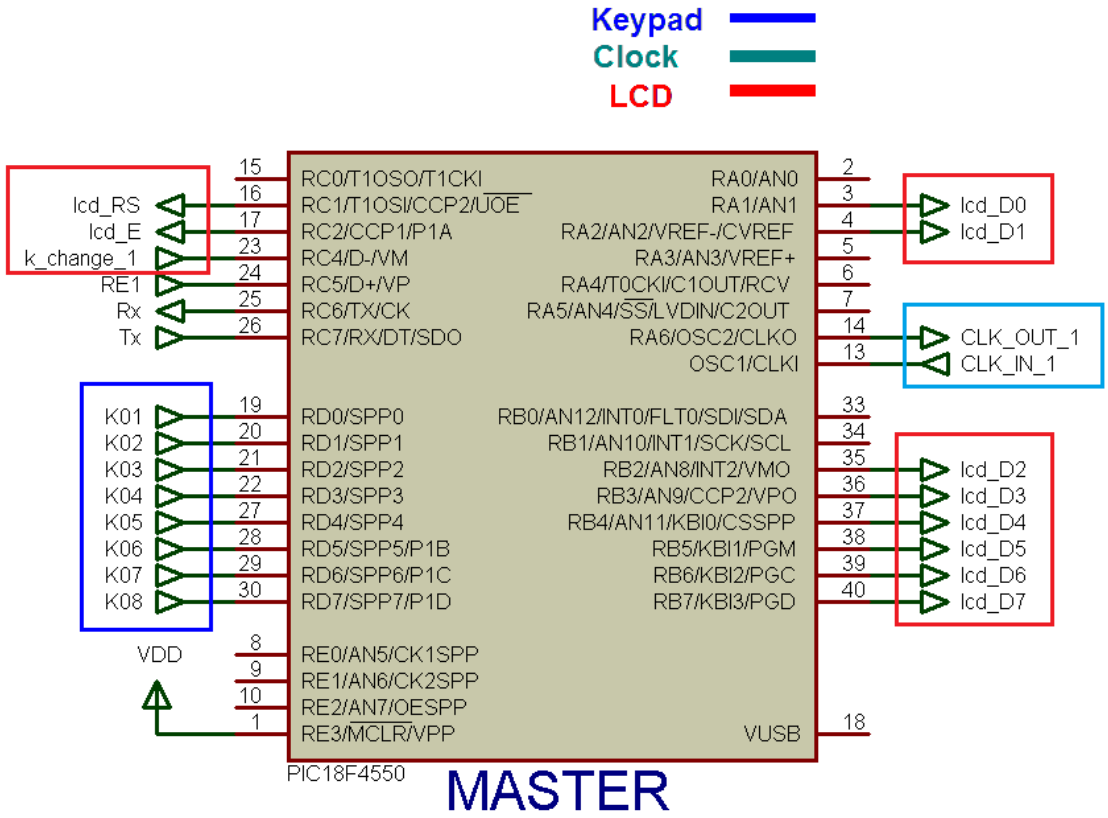

Figure III. 5: Microcontroller interfacing with different parts

III.3. Communication between PICs

There is many ways of communication available: Parallel and Serial communication. The parallel send whole block of data (8 bits).

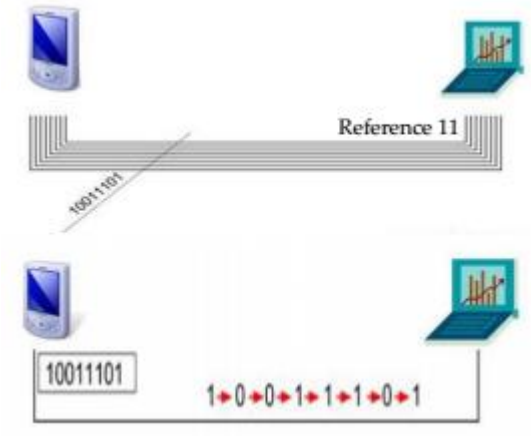

Figure III. 6: Parallel versus serial communication

Simultaneously, while serial sends bit by bit. (See figure III.6)

Since PIC18F supports different types of serial communication; UART, SPI, I2C and one wire protocols. (See figure III.7)

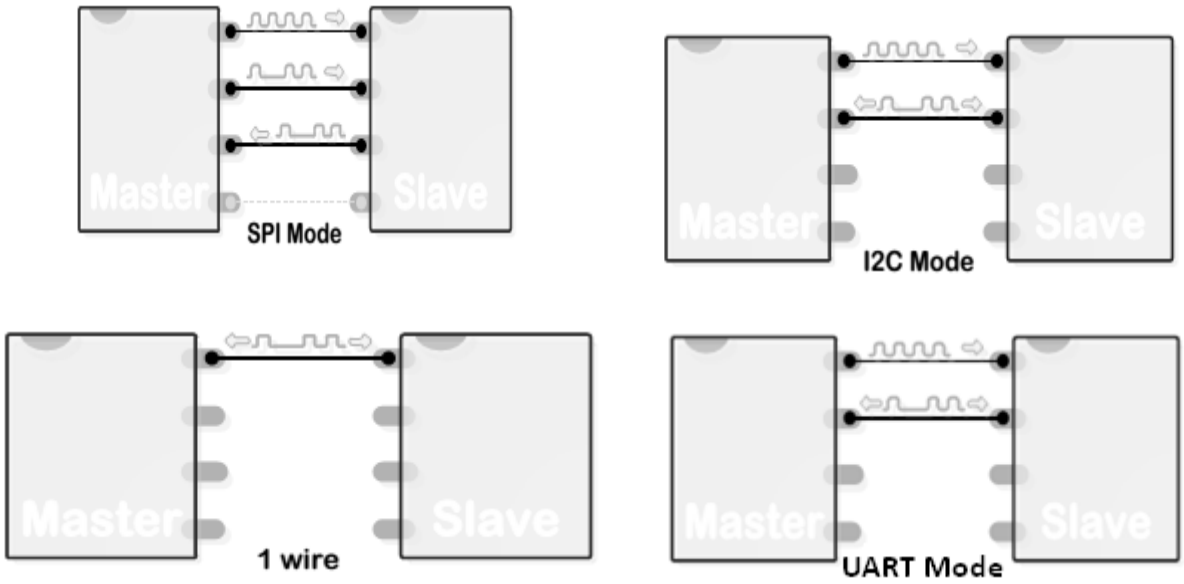

Figure III. 7: Serial communication types supported by PIC

Since UART protocol is the simplest one, and the most used and reliable we will use it on our project. [15]

III.4. Interfacing UART

The pins RX/TX are the UART pins that the PIC18F4550 use (see figure III.5) are used for UART protocol, (RX) on each side is used to receive data on pin 26 RC7, and TX on each side is used to send data to pin 25 RC6.

CHAPTER IV: Software development

IV.1. Integrate Development Environment (IDE)

The *mikroC PRO for PIC* is a development tool for PIC microcontrollers. It is designed to provide the programmer with solutions to developing applications for embedded systems, without compromising performance or control.

PIC and C fit together well: PIC is the most popular 8-bit chip in the world, used in a wide variety of applications, and C, prized for its efficiency, is the natural choice for developing embedded systems. *MikroC PRO for PIC* provides a successful match featuring highly advanced IDE, ANSI compliant compiler, broad set of hardware libraries, comprehensive documentation.

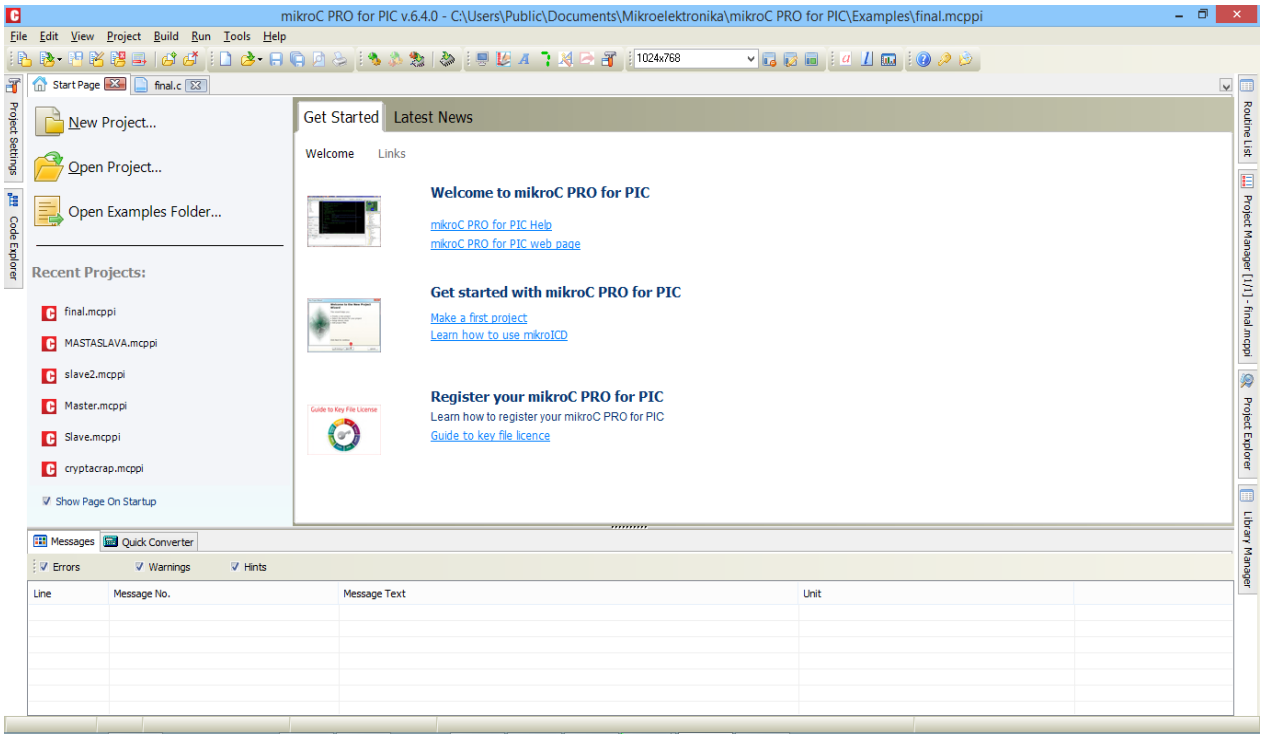

Figure IV. 1: MikroC pro IDE interface

IV.2. Simulator

Proteus 8 is a single application with many service modules offering different functionality (schematic capture, PCB layout, etc.).

In Proteus 8, the relationship between Schematic Design and PCB Layout involves a shared database and is far more integrated. It therefore have a single project file rather than separate design and layout files.

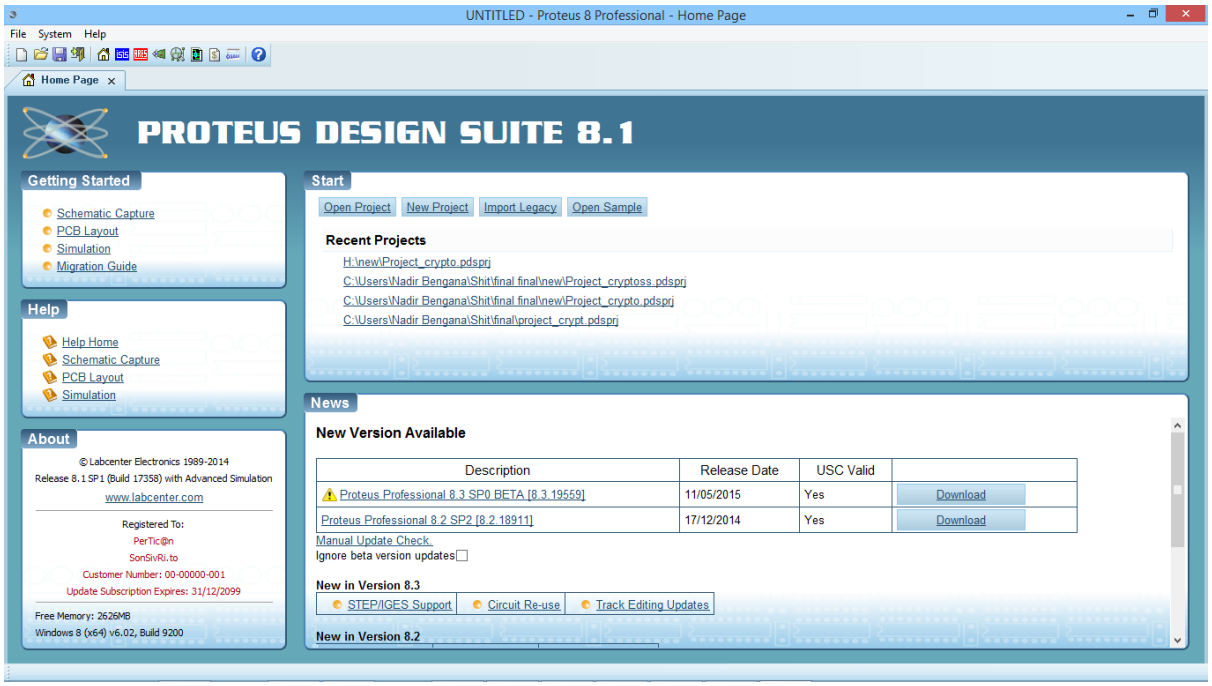

Figure IV. 2: Proteus interface

IV.3. Project's Flow chart

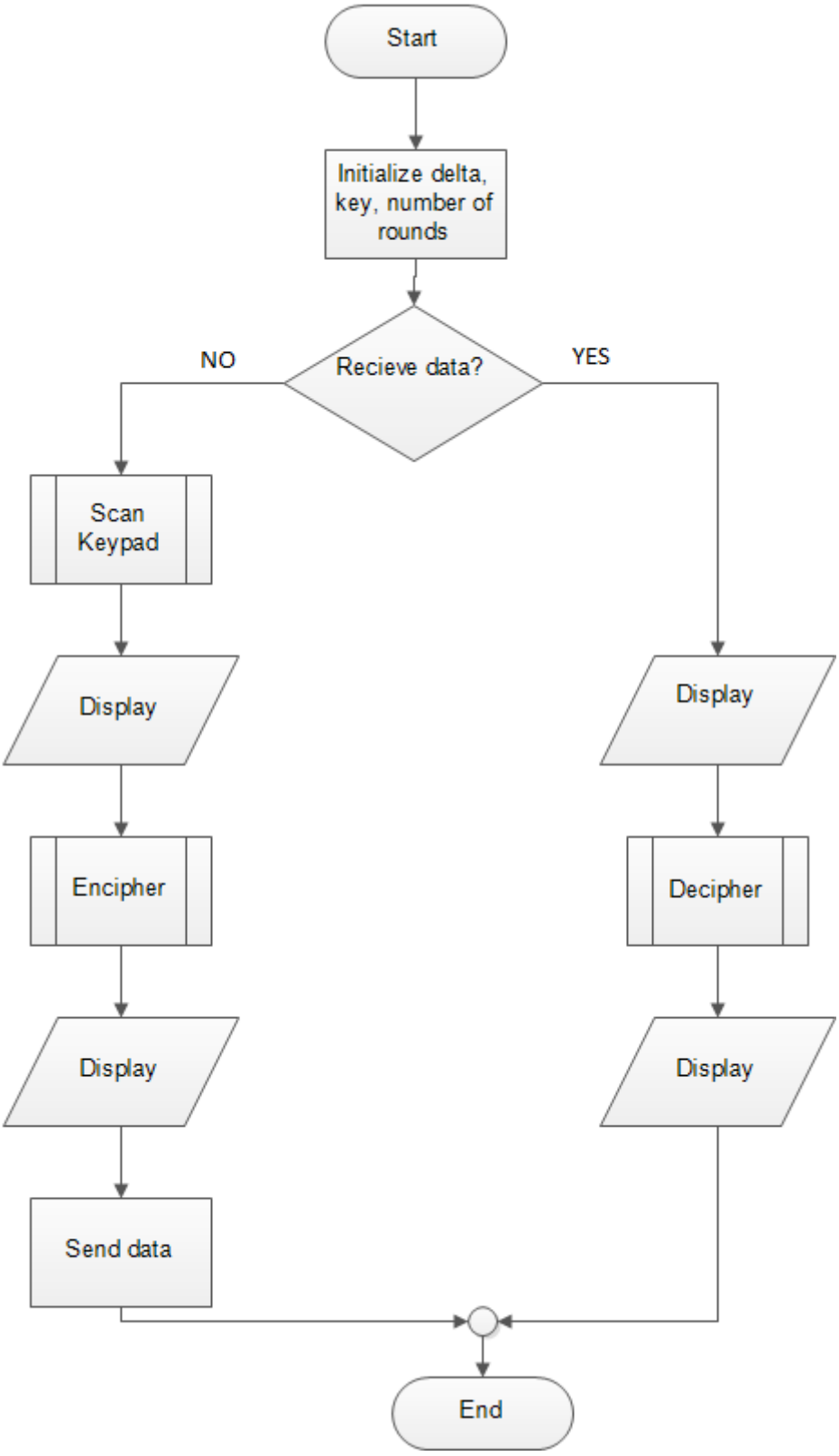

Figure IV. 3: Project's flowchart

IV.4. Procedure

Both microcontrollers will use the same code. To be able to drive them by the same code a condition is set by a hardware switch, the algorithm used is the one shown in figure IV.3.

In this version editing the key, delta, number of rounds is performed using the code. In future update those parameters will be accessible to the user!

The Pin RC5 will decide whether the PIC is Master or Slave. The Master is the one sending data while the slave is the one receiving it. The protocol used for communicating is the UART, which uses RC6 and RC7 pins of each PIC connected reversely.

The microcontrollers will proceed through the following steps:

- Initialize the parameters such as the key and number of rounds
- Check if there's data coming from the other microcontroller
- In case data is coming:
    - Display the received (ciphered) data
    - Decipher the received data
    - Display the deciphered data
- In case data isn't coming:
    - Scan the keypad
    - Display the data written by the keypad
    - Encipher the data written by the keypad
    - Display the ciphered data
    - Send the data via UART

IV.5. XTEA in PIC18F4550:

To make the PIC18F4550 compatible with the XTEA algorithm we followed the following procedure

The data being sent is 8 bits (unsigned short), and since the XTEA algorithm needs to divide the input data into two parts (left and right) we need to do that manually for our data before encrypting it. The sample 8 bits is of the form D_2D_1 so we take D_1 and put it as the right input and D_2 as the left input. This can be done by the following (see figure also):

$$D_2 = (D_2D_1)/10$$

$$D_1 = (D_2D_1)\%10$$

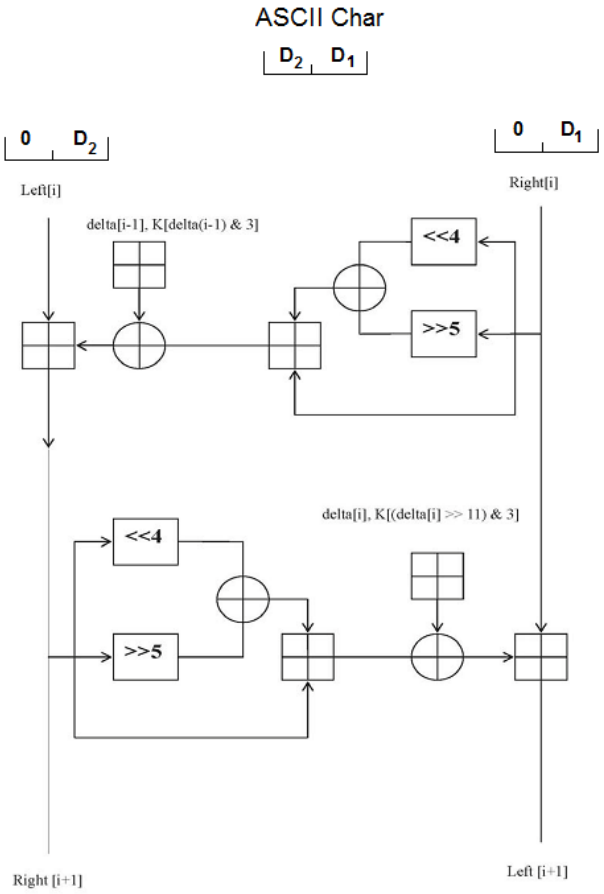

Figure IV. 4: Abstraction of 8 bits cryptography with XTEA

All codes including keypad, LCD, cypher, decipher are in a single code executed in both PIC18F4550 microcontrollers (see Appendix A).

IV.6. Execution and simulation

This is a screenshot when the whole system is executed. (See figure III.8).

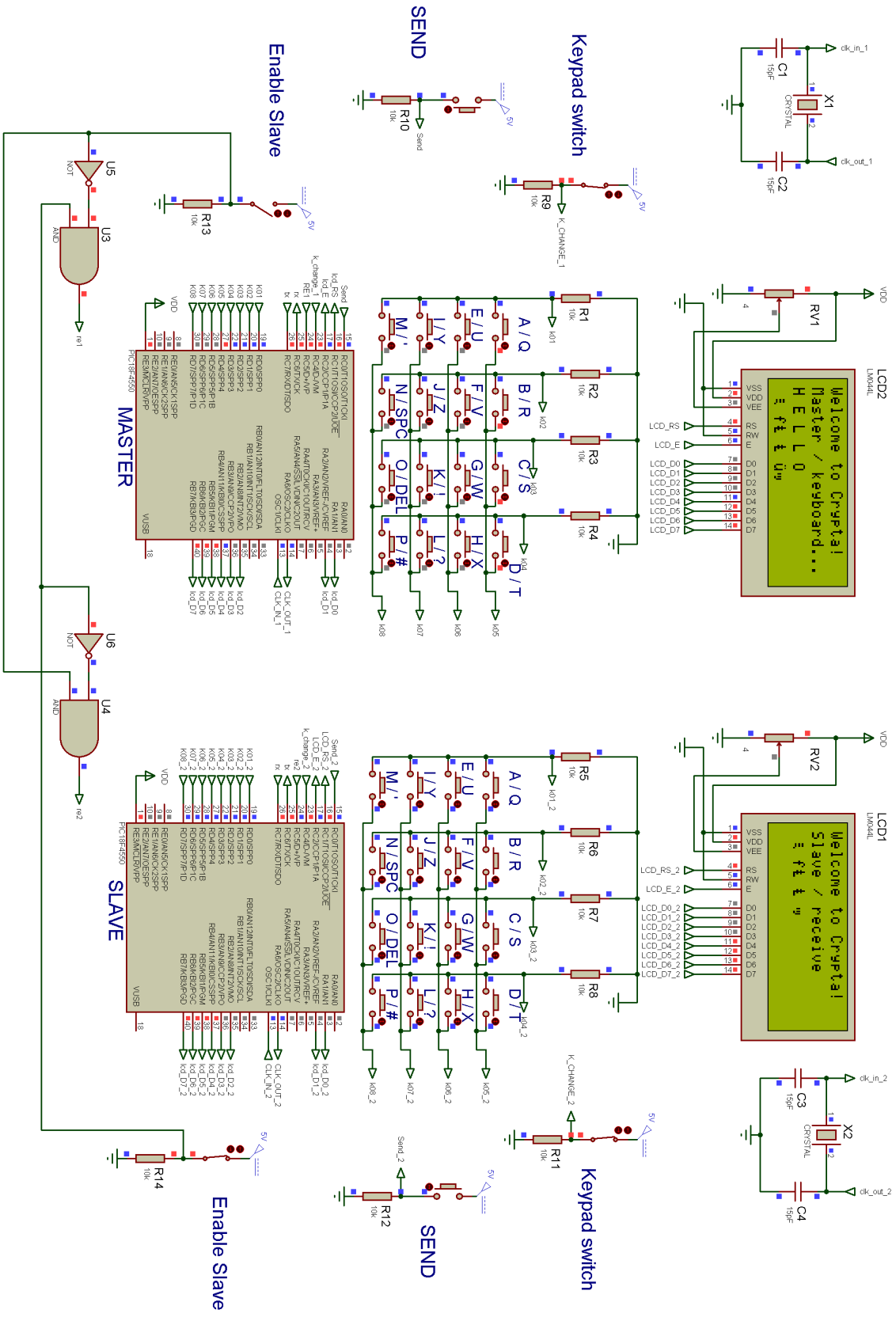

Figure IV. 5: Master, Slave initial state

When the Master sends data (8 bits) by keyboard, it will be encrypted using the XTEA algorithm and it will be sent using UART module. On the other side the Slave receives the data whenever it become ready (it uses a function that check data availability), it decodes the received data and display it afterward.

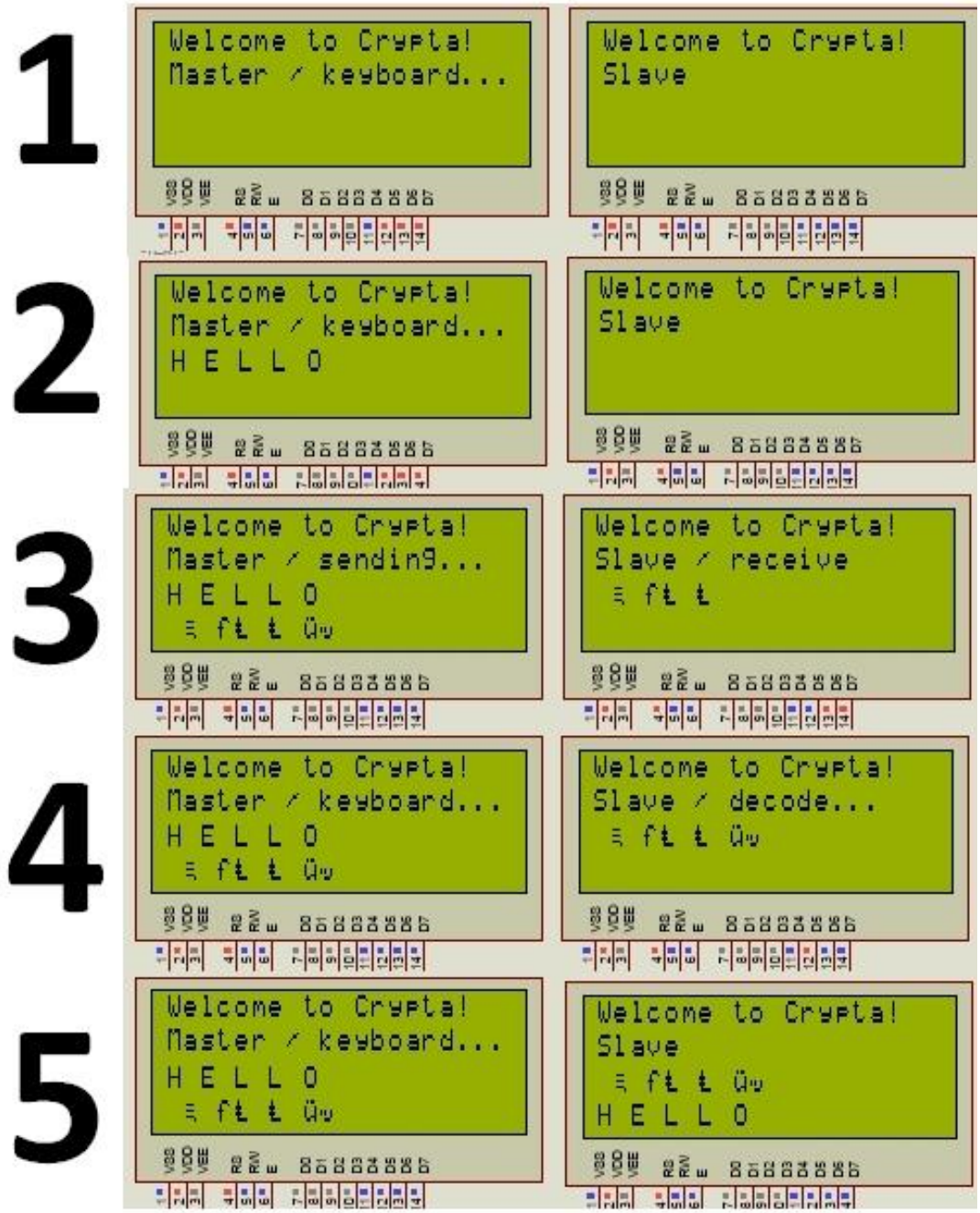

Figure IV. 6: The different steps of the "Crypta" system

In figure IV.5, the steps the system follows are:

1. Initial state
2. Inputing data (typing text)

3. Sending encrypted data to Slave
4. Decrypting received data
5. Displaying the plain-text

The results obtained by the PIC and the encryption algorithm are very good, and the process speed is also good (since 20 MHz is enough for the PIC to do the needed calculations in a small amount of time).

IV.7. Analysis and discussion

We send a sample of data, in the simulation, and the result was the following:

- Cipher key: "0x00112233445566778899AABBCCDDEEFF"
- Number of rounds: "32 rounds"
- Plaintext: "MR TERCHI"
- Plaintext Hex: "0x4D5220544552434849"
- Cipher 1: "0xC523D3244C234D6110"
- Cipher 2: "0x8BD55B2687D5C721AF"

Table IV. 1: Simulation results in ASCII and HEX

	Char 1	Char 2	Char 3	Char 4	Char 5	Char 6	Char 7	Char 8	Char 9
Sample data	M	R		T	E	R	C	H	I
ASCII code	77	82	32	84	69	82	67	72	73
HEX	0x4D	0x52	0x20	0x54	0x45	0x52	0x43	0x48	0x49
D_1 ASCII code	197	35	211	36	76	35	77	97	16
D_1 HEX	0xC5	0x23	0xD3	0x24	0x4C	0x23	0x4D	0x61	0x10
D_2 ASCII code	139	213	91	38	135	213	199	33	175
D_2 HEX	0x8B	0xD5	0x5B	0x26	0x87	0xD5	0xC7	0x21	0xAF

Since the LCD uses only 8 bit ASCII codes some characters appear as different characters, but that is not a problem since the written message can be displayed correctly.

The process of encoding and decoding is relatively fast. It takes few milliseconds for encoding decoding but since the simulation was not running in real time it is hard to measure the overall time for encoding and decoding.

Conclusion

The purpose of this project was to implement a cryptosystem using PIC microcontrollers. The chosen algorithm was the extended tiny algorithm (XTEA) and a suitable code was written to encrypt the data as well as manage and interface the different components of the system (keyboards as input, LCDs as outputs).

As expected the PIC18F4550 was able to handle the execution of the XTEA algorithm very well but with some modifications to the algorithm. The data (words) were ciphered, sent, and deciphered in very acceptable time periods.

The UART protocol seemed perfect for communicating easily between the 2 microcontrollers. One code was enough to drive both the Master (transmitter) and Slave (Receiver) and switching between them can be done with a simple switch.

The project encountered some issues during development. One problem was the synchronization of the two PICs when sending and receiving, since there was some delays or "data loss" sometimes but that was mainly because the simulation was not running in real-time but these losses happened rarely. The data loss problem can be fixed by sending a parity check bit with the data to ensure that the received data is correct. Such problems occurred in the simulation but wouldn't happen in real life because most of them was the result of Proteus lacking resources to run properly.

Another problem was implementing a keyboard suitable for inputting data. And because Proteus did not provide a ready letter-keyboard (only number-keypads) we went ahead and made one from scratch, it consists of 16 buttons each with double functionality that can be toggled between using a switch and thus we were able to send type words.

The main limitation that was encountered was that the XTEA could not be executed with its original version because of the limited data length in the PIC18F4550. This lead to editing the algorithm to make it more compatible with the requirements.

Future work:

This project can be further expanded and given more scope for practical applications. Communication between the microcontrollers could be done using a radio frequency module (RF) and to more than one microcontroller, and for more security the key and number of rounds would be accessible to the user or maybe updating the whole algorithm to a more secure stable one. Encrypting types of data other than written could be more challenging using only PIC microcontrollers but it is not completely impossible providing access to better components.

References

[1] Philip R, Zimmermann (October 1998). Cryptography for the Internet.

[2] Huzaifa Sidhpurwala (2013/08/14). A Brief History of Cryptography. From https://securityblog.redhat.com/2013/08/14/a-brief-history-of-cryptography/

[3] Cryptosystems. From http://en.wikipedia.org/wiki/Cryptosystem

[4] Miguel Hidalgo. Compare and Contrast of Symmetric Cryptosystems and Public-Key Cryptosystems. From http://www.informweb.com/webportal/articles/pkicc.htm

[5] Dr. Bill Young. Foundations of Computer Security, Lecture 45: Stream and Block Encryption. Department of Computer Sciences University of Texas at Austin.

[6] Jiqiang Lu. Related-key rectangle attack on 36 rounds of the XTEA block cipher. International Journal of Information Security.

[7] Vikram Reddy Andem (2003). A cryptanalysis of the tiny encryption algorithm a thesis. Department of Computer Science, the University of Alabama.

[8] Feistel cipher. From http://en.wikipedia.org/wiki/Feistel_cipher

[9] PIC18F4550 data-sheet

[10] Dogan Ibrahim. Advanced PIC microcontroller

[11] MikroElectronica support forums

[12] MikroElecronica Help

[13] PIC18F4550 datasheet

[14] Sultan AbderlKhaleq. Microcontroller encyclopedia "ميكروبيديا" .

[15] Mostafa Hadja AbdeRahman. Serial communication with PIC18.

[16] Ahmed Smir Faid. "المفتاح البسيط للتعامل مع أنواع مختلفة من الميكروكنترولر".

Appendices

A. The source code

```
// Keypad module connections
char  keypadPort at PORTD;
// End Keypad module connections

// LCD module connections
sbit LCD_RS at RC1_bit;
sbit LCD_EN at RC2_bit;
sbit LCD_D0 at RA0_bit;
sbit LCD_D1 at RA2_bit;
sbit LCD_D2 at RB2_bit;
sbit LCD_D3 at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_RS_Direction at TRISC1_bit;
sbit LCD_EN_Direction at TRISC2_bit;
sbit LCD_D0_Direction at TRISA0_bit;
sbit LCD_D1_Direction at TRISA2_bit;
sbit LCD_D2_Direction at TRISB2_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
// End LCD module connections

#include <stdint.h>
unsigned short v0, v1, dat[2];
//int delta = 0x9E3779B9;
unsigned short delta = 0x9E3779B9;
unsigned short key[4] = {0x00112233,0x44556677,0x8899aabb,0xccddeeff};
unsigned short send[9], rec[9], word_e[9], word_s[9], kp;
short rows = 1,rowm = 1, col = 3;
int i, j, length, sum, num_rounds =32;

void main() {
  TRISC.B4 = 1;
  TRISC.B5 = 1;
  TRISC.B0 = 1;
  Keypad_Init();                          // Initialize Keypad
  ADRESL  = 0;                             // Configure AN pins as digital I/O
  ADRESH = 0;
  ADCON1=0;
  ADCON2=0;
  Lcd_Init(); // Initialize LCD
  Lcd_Cmd(_LCD_CLEAR); // Clear display
  Lcd_Cmd(_LCD_CURSOR_OFF); // Cursor off
  Lcd_Out(1, 1, "Welcome to Crypta!   ");

  do {
   while(PORTC.B5 == 0){
     Lcd_Out(2, 1, "Slave               ");
      UART1_Init(9600);
      delay_ms(100);
      length = UART1_Read();
      if(length < 9){
           Lcd_Out(2, 1, "Slave / receive     ");
```

```
        for(i=0;i<(2*length);i++){
             delay_ms(100);
             rec[i] = UART1_Read();
           }

    // Descryption Algorithm
    for(j=0;j<length;j++){
         Lcd_Out(2, 1, "Slave / decode...   ");
         dat[0] = rec[2*j];
         dat[1] = rec[2*j+1];
         v0=dat[0], v1=dat[1], sum=delta*num_rounds;
         for (i=0; i < num_rounds; i++) {
             v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
             sum -= delta;
             v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
         }
         dat[0]=v0; dat[1]=v1;
         word_s[j] = dat[0]*10 + dat[1];
     }
     // End of Decryption Algorithm

     // Display decrypted text
     for(i=0;i<length;i++){
          Lcd_Chr(col, rows, word_s[i]);

          // Organize Display
          rows += 1;
          if (rows > 20) {
              col++;
            }
          if(rows > 20 & col == 4){
              Lcd_Out(3, 1, "                    ");
              Lcd_Out(4, 1, "                    ");
            }
        }
     }
   }

while(PORTC.B5 == 1){
     Lcd_Out(2, 1, "Master    ");
     for(i=0;i<20;i++){
          kp = 0;

          Scan:
          Lcd_Out(2, 1, "Master / keyboard...");

          do {
               kp = Keypad_Key_Click();
                    if(PORTC.B0 == 1){
                          goto ENC;
                      }
            } while (!kp);

            if (PORTC.B4 == 1){
                 switch (kp) {
                        case  1: kp = 65; break; // A
                        case  2: kp = 66; break; // B
```

```
                case  3: kp = 67; break; // C
                case  4: kp = 68; break; // D
                case  5: kp = 69; break; // E
                case  6: kp = 70; break; // F
                case  7: kp = 71; break; // G
                case  8: kp = 72; break; // H
                case  9: kp = 73; break; // E
                case 10: kp = 74; break; // J
                case 11: kp = 75; break; // K
                case 12: kp = 76; break; // L
                case 13: kp = 77; break; // M
                case 14: kp = 78; break; // N
                case 15: kp = 79; break; // O
                case 16: kp = 80; break; // P
              }
        }

        else if (PORTC.B4 == 0) {
             switch (kp) {
                case  1: kp = 81; break; // Q
                case  2: kp = 82; break; // R
                case  3: kp = 83; break; // S
                case  4: kp = 84; break; // T
                case  5: kp = 85; break; // U
                case  6: kp = 86; break; // V
                case  7: kp = 87; break; // W
                case  8: kp = 88; break; // X
                case  9: kp = 89; break; // Y
                case 10: kp = 90; break; // Z
                case 11: kp = 33; break; // !
                case 12: kp = 63; break; // ?
                case 13: kp = 39; break; // '
                case 14: kp = 32; break; // SPC
                case 15: {rowm-=1;i--; kp = 0; break;} // DEL
                case 16: kp = 35; break; // #
                }
             if (kp == 0) {
                    Lcd_Chr(col, rowm, kp);
                     goto Scan;
               }
             }

         Lcd_Out(2, 1, "Master / displaying.");

         // Print key ASCII value on LCD
         word_e[i] = kp;
         Lcd_Chr(col, rowm, word_e[i]);
         length++;
         rowm += 1;
         if (rowm > 20) {
            col++;
           }
         if(rowm > 20 & col == 4){
            Lcd_Out(3, 1, "                    ");
            Lcd_Out(4, 1, "                    ");
           }
}
```

```
        ENC:
        Lcd_Out(2, 1, "Master / encoding...");
        for(j=0;j<length;j++){
             // Encryption Algorithm
             dat[0]= word_e[j]/10 ;
             dat[1]= word_e[j]%10;
             v0=dat[0], v1=dat[1], sum=0;
             for (i=0; i < num_rounds; i++) {
                  v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
                  sum += delta;
                   v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum >> 11) & 3]);
                }
             dat[0]=v0; dat[1]=v1;
             send[2*j] = dat[0];
             send[2*j+1] = dat[1];
             // End of encryption algorithm
          }

            // Sending data using UART
             Lcd_Out(2, 1, "Master / sending... ");
             UART1_Init(9600);
             Delay_ms(100);
             UART1_Write(length);
             for(i=0;i<length;i++){
                  UART1_Write(send[2*i]);
                  Delay_ms(100);
                  UART1_Write(send[2*i+1]);
                  Delay_ms(100);
               }
             for(i=0;i<length;i++){
                  send[i] = ' ';
               }
             length = 0;
      }
    } while (1);
}
```

B. PIC18F4550 pin assignment