FPGA Implementation of RC6 Algorithm for IPSec protocol

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
VLSI DESIGN and EMBEDDED SYSTEM

By
SUDHEER REDDY ENUGU
Roll No : 20607006

Under the Guidance of
Prof.K.K.MAHAPATRA

Department of Electronics & Communication Engineering
National Institute of Technology
Rourkela
2006 - 2008

ACKNOWLEDGEMENT

This project is by far the most significant accomplishment in my life and it would be impossible without people who supported me and believed in me.

I would like to extend my gratitude and my sincere thanks to my honorable, esteemed supervisor **Prof. K.K.Mahapatra,** Department of Electronics and Communication Engineering. He is not only a great lecturer with deep vision but also and most importantly a kind person. I sincerely thank for his exemplary guidance and encouragement. His trust and support inspired me in the most important moments of making right decisions and I am glad to work with him.

I want to thank all my teachers **Prof. G.S. Rath**, **Prof. G.Panda, Prof. S.Mehar, Prof. S.K. Patra** and for providing a solid background for my studies and research thereafter. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would like to thank all my friends and especially my classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. I've enjoyed their companionship so much during my stay at NIT, Rourkela.

I would like to thank all those who made my stay in Rourkela an unforgettable and rewarding experience.

Last but not least I would like to thank my parents, who taught me the value of hard work by their own example. They rendered me enormous support during the whole tenure of my stay in NIT Rourkela.

SUDHEER REDDY. E

Abstract

With today's great demand for secure communications systems, there is a growing demand for real-time implementation of cryptographic algorithms. In this thesis we present a hardware implementation of the RC6 algorithm using VHDL Hardware Description Language. And the goal of the thesis was to implement a subset of the IPSec protocol using a Microcontroller and an FPGA. IPSEC is a framework for security that operates at the Network Layer by extending the IP packet header. IPSec protocol is to guarantee the security of data while traveling through the network. The motivation was to enable network application and cryptography to assembly and VHDL languages and to develop a prototype of their system. In this thesis many different sub-systems had to communicate with each other to achieve the final product: the PC and the Microcontroller through a serial connection, the Microcontroller and the FPGA through a bidirectional bus, and the Microcontroller and a terminal using a serial connection. Data was to be encrypted and decrypted using an RC6 algorithm including key scheduling application. The crypto-coprocessor (to implement RC6 algorithms) was implemented within an FPGA and connected to the Microcontroller bus.

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1. **INTRODUCTION**

RC6 is a symmetric key block cipher derived from RC5. It was designed by Ron Rivest, Matt Robshaw, Ray Sidney, and Yiqun Lisa Yin to meet the requirements of the Advanced Encryption Standard (AES) competition by the National Institute of Standards and Technology (NIST). The algorithm was one of the five finalists, and was also submitted to the NESSIE and CRYPTREC projects. Though the algorithm was not eventually selected, RC6 remains a good choice for security applications. It is proprietary of RSA Security.

The design of RC6 began with a consideration of RC5 as a potential candidate for an AES submission. Modifications were then made to meet the AES requirements, to increase security, and to improve performance. The inner loop, however, is based around the same “half-round" found in RC5. RC5 was intentionally designed to be extremely simple, to invite analysis shedding light on the security provided by extensive use of data-dependent rotations. Since RC5 was proposed in 1995, various studies provided a greater understanding of how RC5's structure and operations contribute to its security. While no practical attack on RC5 has been found, the studies provide some interesting theoretical attacks, generally based on the fact that the “rotation amounts" in RC5 do not depend on all of the bits in a register. RC6 was designed to thwart such attacks, and indeed to thwart all known attacks, providing a cipher that can offer the security required for the lifespan of the AES.

The philosophy of RC5 is to exploit operations (such as rotations) that are efficiently implemented on modern processors. RC6 continues this trend, and takes advantage of the fact that 32-bit integer multiplication is now efficiently implemented on most processors. Integer multiplication is a very effective “diffusion" primitive, and is used in RC6 to compute rotation amounts, so that the rotation amounts are dependent on all of the bits of another register, rather than just the low-order bits (as in RC5). As a result the new RC6 has much faster diffusion than RC5. This also allows RC6 to run with fewer rounds at increased security and with increased throughput.

RC6 is more exactly specified as RC6-w/r/b, where the parameters w, r, and b respectively express the word size (in bits), the number of rounds, and the size of the encryption key (in bytes). Since the AES submission is targeted at w = 32 and r = 20, we implemented this

version of RC6 algorithm, using a 32 bits word size, 20 rounds and 16 bytes (128 bits) encryption key lengths. The RC6 block cipher diagram as shown in the fig 1.1.

A key schedule generates 2r + 4 words (w bits each) from the b-bytes key provided by the user. These values (called round keys) are stored in an array S [0, 2r+3] and are used in both encryption and decryption. RC6 works on a block size of 128 bits and it is very similar to RC5 in structure, using data-dependent rotations, modular addition and XOR operations; in fact, RC6 could be viewed as interweaving two parallel RC5 encryption processes. However, RC6 does use an extra multiplication operation not present in RC5 in order to make the rotation dependent on every bit in a word, and not just the least significant few bits. The computation of $f(X) = (X \times (2X + 1)) \bmod 2^w$ is the most critical arithmetic operation of this block cipher. The goal of this thesis is to implement the RC6 Cipher with FPGA as the target technology.

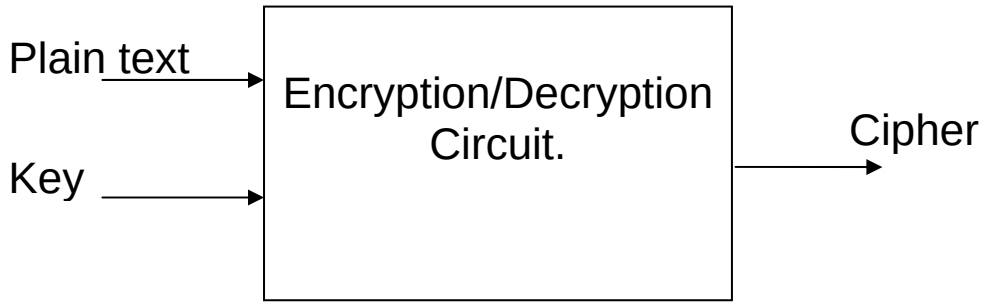

Fig 1.1: RC6 Cipher block diagram

The goal of the thesis was to implement on a microcontroller a subset of the IPSec protocol. IPSec is part of the IPv6 protocol to guarantee the security of data while traveling through the network (i.e. authentication, privacy and integrity). In this thesis two entities were communicating, a PC and a microcontroller. The PC was sending the data to the microcontroller using a point-to-point protocol over a serial link. Then the microcontroller processed the datagram, checking its validity and extracting the data. In dealing with IPSec, the data was encrypted so it was necessary to first decrypt the data to get the original plain text. Furthermore to speed up the decryption task, a crypto-coprocessor was considered. To manage the thesis several skills were necessary, from networking to micro-programming and hardware design. Generally IP is associated with TCP and well known as TCP/IP. In this

thesis in order to manage the complexity of the system the TCP layer was not considered and the data was provided directly after the IP layer as show in Figure 1.2. Thus from the PC side, the data was encrypted using the RC6 algorithm before being encapsulated into a datagram. To obtain the final datagram two layers were considered which are successively the IP and the PPP layers. The physical layer splits the datagram in order to meet serial link requirements. From the Microcontroller side the same steps were considered but in the reverse order. Once the data was extracted from the datagram it was sent to the crypto-coprocessor in order to retrieve the plain text. The crypto-coprocessor was implemented within an FPGA and connected to the Microcontroller bus.

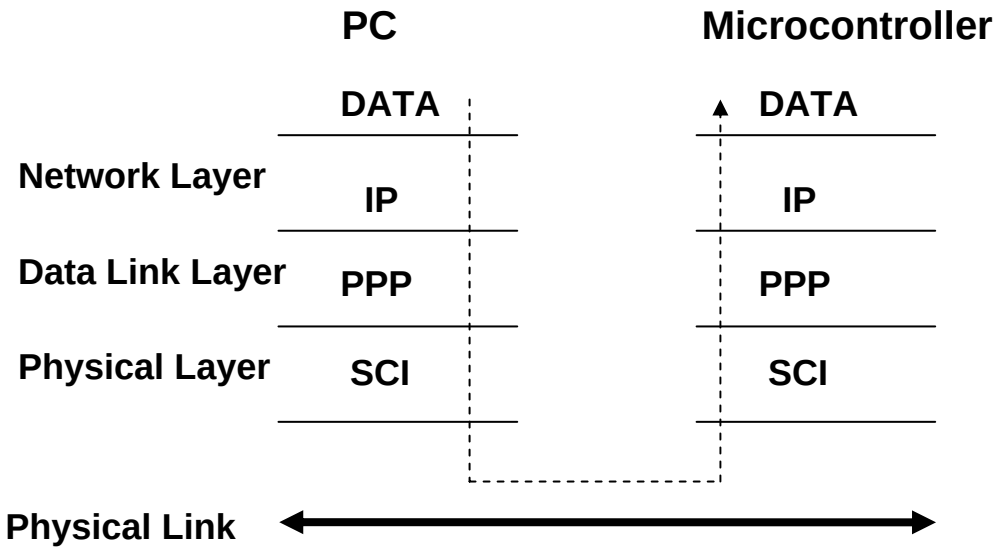

Fig 1.2: Layers involved during a communication with in the PC and Microcontroller

Chapter 2

MOTIVATION

2. MOTIVATION

To attack RC6 the best approach available to the cryptanalyst is that of exhaustive search for the b-byte encryption key. The more advanced attacks of differential and linear cryptanalysis, while being feasible on small-round versions of the cipher, do not extend well to attacking the full 20-round RC6 cipher. The RC6 key schedule is secure through mixing, one way function and no key separation. Therefore, RC6 provides a solid, well tuned margin for security.

RC6 facilitates and encourages analysis by allowing rapid understanding of security and making direct analysis straightforward. It also enables easy implementation by allowing compilers to produce high quality code for software implementations, and by preventing complicated optimizations and providing good performance with minimal effort for hardware implementations. RC6 is known to have good performance on 8, 16 and 32-bit platforms.

2.1 simplicity

The simplicity of RC5 has made it an attractive object for research. By being readily accessible to both crude and sophisticated analysis many people have been encouraged to look at the cipher and to assess the security it offers. RC6 was designed to build on the experience gained in using RC5 and to build on the security offered by a remarkably simple cipher. One can view the design of RC6 as progressing through the following steps:

1. Start with the basic half-round loop of RC5:

```
for i = 1 to r do
{
A = ((A  xor B)<<<B) + S[i]
(A, B) = (B, A)
}
```

2. Run two copies of RC5 in parallel: one on registers A, B and one on registers C,D.

```
for i = 1 to r do
{
A = ((A xor B)<<<B) + S[2i]
C = ((C xor D)<<<D) + S[2i+ 1]
(A,B) = (B,A)
(C,D) = (D,C)
}
```

3. At the swap stage, instead of swapping A with B and C with D, permute the registers by (A, B, C, D) = (B, C, D, A), so that the AB computation is mixed with the CD computation. At this stage the inner loop looks like:

```
for i = 1 to r do
{
A = ((A xor B)<<<B) + S[2i]
C = ((C xor D)<<<D) + S[2i+ 1]
(A, B, C, D) = (B, C, D, A)
}
```

4. Mix up the AB computation with the CD computation further, by switching where the rotation amounts come from between the two computations:

```
for i = 1 to r do
{
A = ((A xor B)<<<D) + S[2i]
C = ((C xor D)<<<B) + S[2i+ 1]
(A, B, C, D) = (B, C, D, A)
}
```

5. Instead of using B and D in a straightforward manner as above, we use transformed versions of these registers, for some suitable transformation. Our security goals are that the data-dependent rotation amount that will be derived from the output of this transformation should depend on all bits of the input word and that the transformation should provide good mixing within the word. The particular choice of this transformation for RC6 is the function $f(x) = x \times (2x + 1) \pmod{2^w}$ followed by a left rotation by five bit positions. This transformation appears to meet our security goals while taking advantage of simple primitives that are efficiently implemented on most modern processors. Note that f(x) is one-to-one modulo 2^w, and that the high-order bits of f(x), which determine the rotation amount used, depend heavily on all the bits of x. This gives us:

```
for i = 1 to r do
{
 t = (B × (2B + 1))<<<5
 u = (D × (2D + 1))<<<5
 A = ((A xor t)<<<u) + S[2i]
 C = ((C xor u)<<<t) + S[2i+ 1]
 (A, B, C, D) = (B, C, D, A)
}
```

6. At the beginning and end of the r rounds, add pre-whitening and post-whitening steps. Without these steps, the plaintext reveals part of the input to the first round of encryption and the cipher text reveals part of the input to the last round of encryption. The pre- and post-whitening steps help to disguise this and leaves us with RC6:

```
B = B + S[0]
D = D + S[1]
for i = 1 to r do
{
t = (B × (2B + 1))<<<5
u = (D × (2D + 1))<<<5
A = ((A xor t)<<<u) + S[2i]
C = ((C xor u)<<<t) + S[2i+ 1]
(A, B, C, D) = (B, C, D, A)
}
A = A + S[2r + 2]
C = C + S[2r + 3]
```

While it might appear that the evolution from RC5 to RC6 was straightforward, it in fact involved the design and analysis of literally dozens of alternatives. RC6 is the design that captures the spirit of our three goals of security, simplicity and performance the most effectively. Note that in the preceding development, the decision to expand to four 32-bit Registers was made first (for performance reasons), and then the decision to use the quadratic function $f(x) = x \times (2x + 1)(\text{mod } 2^w)$ was made later. If we had decided to stick with a two register version of RC6 then we might have had the following encryption scheme as an intermediate:

```
B = B + S[0]
for i = 1 to r do
{
t = B × (2B + 1)<<<5
A = ((A xor t)<<<t) + S[i]
(A, B) = (B, A)
}
A = A + S[r + 1]
```

This variant of RC6 may be of independent interest, particularly when support for 64-bit arithmetic in C improves. However we merely mention this as an aside here.

2.2 Good performance for a given level of security

While the latest techniques demonstrate that RC5-32/12/b, i.e. a 12-round version of RC5, might not be suitable for longer-term security needs, these attacks currently fall short of providing any real avenue for practical attack against a 16-round version. Most existing cryptanalytic results on RC5 depend on what might be viewed as a relatively slow avalanche of change between rounds. The integer addition helps to provide a reasonable amount of change due to the effect of the carry, but the most dramatic changes take place when two different rotation amounts are used at a similar point during the encryption of two related plaintexts. Typically an attacker would aim to control the evolution of the differences from round to round and, in versions of RC5 with fewer rounds, this can allow an attack to be mounted. The incremental changes in arriving at RC6 from RC5 have already been outlined. Two significant changes are the introduction of the quadratic function $B \times (2B + 1)$ (Similarly : $D \times (2D + 1)$) and the fixed rotation by five bits. The quadratic function is aimed at providing a faster rate of diffusion there by improving the chances that simple differentials will spoil rotation amounts much sooner than is accomplished with RC5. The quadratically transformed values of B and D are used in place of B and D to modify the registers A and C, increasing the nonlinearity of the scheme while not losing any entropy (since the transformation is a permutation). The fixed rotation by five bits plays a simple yet important role in complicating both linear and differential cryptanalysis.

2.3 Security

We conjecture that to attack RC6 the best approach available to the cryptanalyst is that of exhaustive search for the b-byte encryption key (or the expanded key array S[0; : : : ; 43] when the user-supplied encryption key is particularly long). The work effort required for this is $\min\{2^{8b}; 2^{1408}\}$ operations. Don Coppersmith observes, however, that at the expense of considerable memory and off-line pre-computation one can mount a meet-in-the-middle attack to recover the expanded key array S[0; : : : ; 43]. This would require 2^{704} on-line computations and so the work effort required to recover the expanded key array might best be estimated by $\min\{2^{8b}; 2^{704}\}$ operations.

The more advanced attacks of differential and linear cryptanalysis, while being feasible on small-round versions of the cipher, do not extend well to attacking the full 20-round RC6

cipher. The main difficulty is that it is hard to find good iterative characteristics or linear approximations with which an attack might be mounted.

It is an interesting challenge to establish the most appropriate goals for security against these more advanced attacks. To succeed, these attacks typically require large amounts of data, and obtaining 2^a blocks of known or chosen plaintext-cipher text pairs is a very different task from trying to recover one key from among 2^a possibilities (this latter task can be readily parallelized). It is worth observing that with a cipher running at the rate of one terabit per second (that is, encrypting data at the rate of 10^{12} bits/second), the time required for 50 computers working in parallel to encrypt 2^{64} blocks of data is more than a year; to encrypt 2^{80} blocks of data is more than 98, 000 years; and to encrypt 2^{128} blocks of data is more than 1019 years.

While having a data requirement of 2^{64} blocks of data for a successful attack might be viewed as sufficient in practical terms, we have aimed to provide a much greater level of security. The community as a whole will decide which level of security a cipher, in particular an AES candidate should satisfy. Should this be less than a data requirement of 2^{128} blocks of data then the number of rounds of RC6 could potentially be reduced from our initial suggestion of 20 rounds, thereby providing an improvement in performance.

For attacking an eight-round version of the cipher, RC6-32/8/b, one can construct six-round characteristics or linear approximations. Assuming that these could be used to attack the eight-round version of the cipher (an assumption that, while reasonable, overlooks a vast number of practical details) the estimated data required to mount a differential cryptanalytic attack on RC6-32/8/b would be around 256 chosen plaintext pairs, and to mount a linear cryptanalytic attack would be around 247 known plaintexts. This includes some consideration of more sophisticated phenomena such as differentials and linear hulls, but we might still expect more customized techniques to reduce these figures by a moderate amount. However they provide a reasonable illustration of the security that might be offered by a version of RC6 with a few rounds. Currently, it seems that a differential attack on the full 20-round RC6 cipher appears to be most easily accomplished by using a six-round iterative characteristic (although we have identified useful three- and four-round characteristics) together with some customized beginning and ending characteristics. Considering a variety of options, the probability of one of the best 18-round characteristics we are aware of in attacking RC6 is

around 2^{-238} and uses integer subtraction as the notion of difference. (For technical reasons, using exclusive-or as the notion of difference can be more problematical.) To use this characteristic in an attack would require more than the total number of available chosen plaintext/cipher text pairs. While we expect the amount of data required for an attack to drop as more detailed analysis takes place we do not believe that differential cryptanalysis can be successfully applied to RC6.

To mount a linear cryptanalytic attack, there appear to be two different options. The first might be to find a linear approximation over several rounds that uses a linear approximation across the quadratic function. Since there appear to be some very suitable linear approximations using the least significant bits of this function, this might be an appealing strategy. Indeed, one can establish useful six-round iterative linear approximations that can, at least in principle, be used to attack reduced-round versions of RC6. However, the bias of these approximations drops rapidly as more rounds are added, and soon the amount of data required for a successful attack exceeds the amount of data available. Instead, we note that an attacker might well pursue an alternative approach. It is possible to find a two-round iterative linear approximation that does not use an approximation across the combination of the quadratic function and fixed rotation by five bit positions. Using basic but established techniques to predict the bias of such an approximation, we observe that the data requirements to exploit this approximation over a version of RC6 with 16 rounds are about 2^{142} known plaintexts. Further analysis suggests that additional techniques might potentially be used to bring the data requirements down to a little under 2^{128} known plaintexts. This provided our rationale for choosing 20 rounds for RC6.

With our current knowledge, the most successful avenue for a linear cryptanalytic attack on RC6 would be to use the two-round iterative approximation we have just mentioned to build up an 18-round linear approximation with which to attack the cipher. Using the same techniques as before to predict the data requirements to use this approximation at first sight, we might need 2^{182} known plaintexts, an amount which exceeds the available data. Enhanced techniques might be useful in reducing this figure by a moderate amount (a pessimistic view suggests that such reductions would still leave an attack requiring 2^{155} known plaintexts) but in the final assessment we believe that the number of known plaintexts needed to exploit this approximation readily exceeds the maximum number of plaintexts available. We conclude that a linear cryptanalytic attack against RC6 is not possible using these techniques. Further,

we believe that the use of more sophisticated techniques are exceptionally unlikely to provide sufficient gains as to offer an attack requiring less than 2^{128} known plaintexts.

We are aware of several potential enhancements to the essential attacks we have described (in particular, the use of truncated and higher-order differentials), and we are also aware of some alternative approaches. However, all these techniques have so far failed to improve on the attacks outlined here, and we believe that all currently available sophisticated cryptanalytic attacks will require more data than there is available. A report on our work and findings is in reparation.

RC6 can easily be implemented in such a way as to be invulnerable to “timing attacks". Many modern processors have constant-time rotation and multiplication instructions. Other processors may have a rotation or shift time that depends linearly with the amount of rotation, but in this case it is usually easy to arrange the work so that the total compute time is data-independent (for example, by computing a rotate of t bits using a left-shift of t bits and a right-shift of w-t bits). In either case, the RC6 encrypt/decrypt time is data-independent, causing any potential timing attacks to fail.

Studies of RC5 have failed to reveal any weakness in the key setup. This provided one of the motivations for using the same key setup in RC6 as was used in RC5. The process of transforming the supplied key to the table of round keys appears to be well-modeled by a pseudo-random process. Thus, while there is no proof that no two keys yield the same table of round keys, it appears to be highly unlikely. It can be estimated that the chance that there exist two 256-bit keys yielding the same table of 44 32-bit round keys is approximately $2^{2\times256-44\times32} = 2^{-896} = 10^{-270}$ (approximately). We feel that there is value in the “one-way" structure of the key-setup routine that is more important than the (infinitesimal) chance that there might be two keys that yield the same table of round keys. One such value is the protection it provides against related-key attacks, for example.

We can summarize on the security of RC6 as follows:

1. The best attack on RC6 appears to be exhaustive search for the user-supplied Encryption key.
2. The data requirements to mount more sophisticated attacks on RC6 such as Differential and linear cryptanalysis exceed the available data.
3. There are no known examples of what might be termed “weak" keys.

Chapter 3

OUTLINE OF THE THESIS

3. OUTLINE OF THE THESIS

In this thesis as mentioned previously several aspects were considered, the first one was related to the datagram definition which requires a general understanding of the IP (and IPSec) and PPP layers. The original data was encrypted and gathered with the AH header and the IP header. Each of these headers contains specific information in order to provide a valid datagram. The resulting datagram was then encapsulated within the PPP layer to provide the final datagram. As for IP, PPP contains specific parameters that were defined. In order to reduce the complexity of the global system a simplified version of the IP and PPP layers was considered (the corresponding protocols can be very complex). For example the SA step was not considered and predefined key and algorithm for the cryptography solution were selected. Furthermore in a first step the authentication algorithm was not handled. Only the cryptography part was targeted. Obviously, the complexity of the system could have evolved depending on the results obtained during the thesis.

As an initial step, the plan was to manually write a text file corresponding to the data to be sent. Then it was necessary to transferred it through the serial interface to the microcontroller (P89C51RD2). The PC was sending the data to the microcontroller using a point-to-point protocol over a serial link. The microcontroller received the data, and stored the data in its memory. The original data was gathered with the AH header and the IP header. Once that step performed it was necessary to send the data to the crypto-coprocessor to encrypt the original data. All the tasks performed on the microcontroller required quite a large hand-written ASM program, so a rigorous test plan was required for debugging in order to manage the complexity of the code. Finally it was necessary to understand the RC6 cryptographic algorithm to be able to build the corresponding hardware design. For that purpose a VHDL code was defined. In order to implementation of the RC6 encryption we considering key scheduling also. Once the data was encrypted it was necessary to send it back to the microcontroller so that it was displayed on a terminal. Figure 3.1 illustrates the system that has been built.

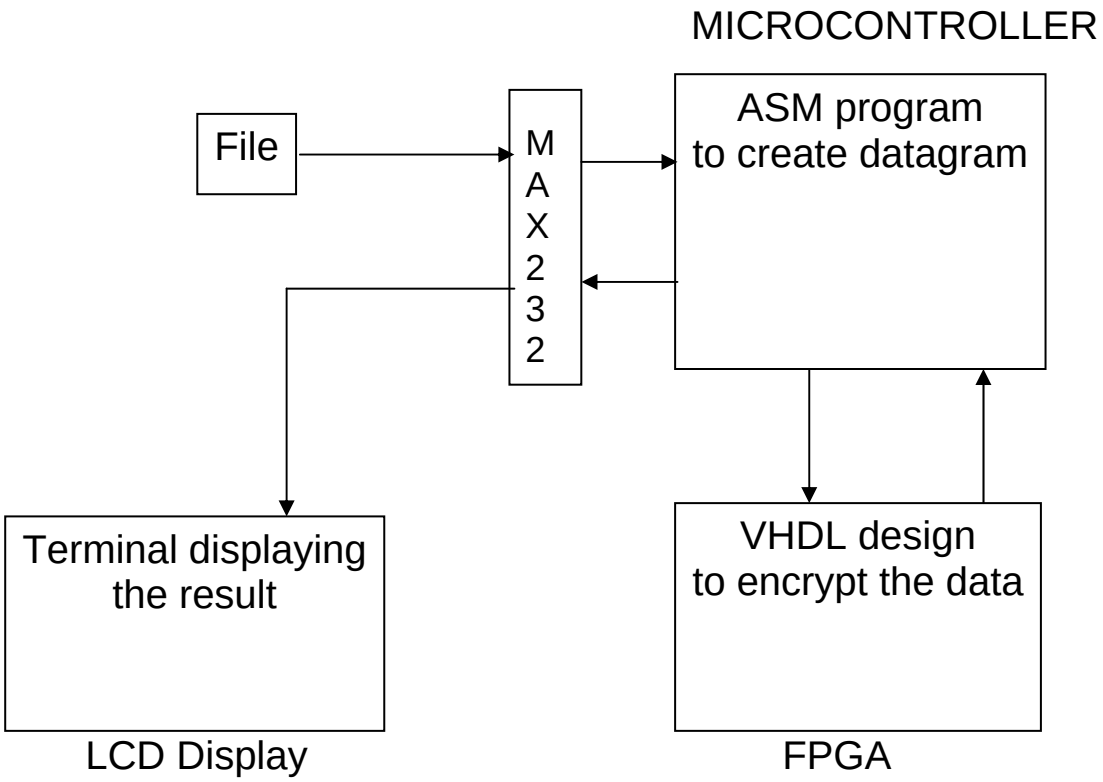

Fig 3.1: Design for creating the datagram and encryption process

After getting the datagram from the terminal it is necessary to decrypt to get original data, the plan was to manually write a text file corresponding to the data to be sent. Then it was necessary to transferred it through the serial interface to the microcontroller (P89C51RD2). The microcontroller received the datagram, checked its validity and stored the data in its memory. To provide this functionality it was necessary to configure the serial interface of the microcontroller in order to be able to receive the datagram. Then the various parameters from the headers were checked to verify the validity of the communication (for example, are the IP source and destination addresses correct). Once that step performed it was necessary to send the data to the crypto-coprocessor to determine the original data. Finally the RC6 cryptographic algorithm to be able to build the corresponding hardware design. For that purpose a VHDL code was defined. In order to help the implementation of the RC6 decryptor we considered the RC6 algorithm including key scheduling. Once the data was decrypted it was necessary to send it back to the microcontroller so that it was displayed on a terminal. Figure 3.2 illustrates the system that has been built.

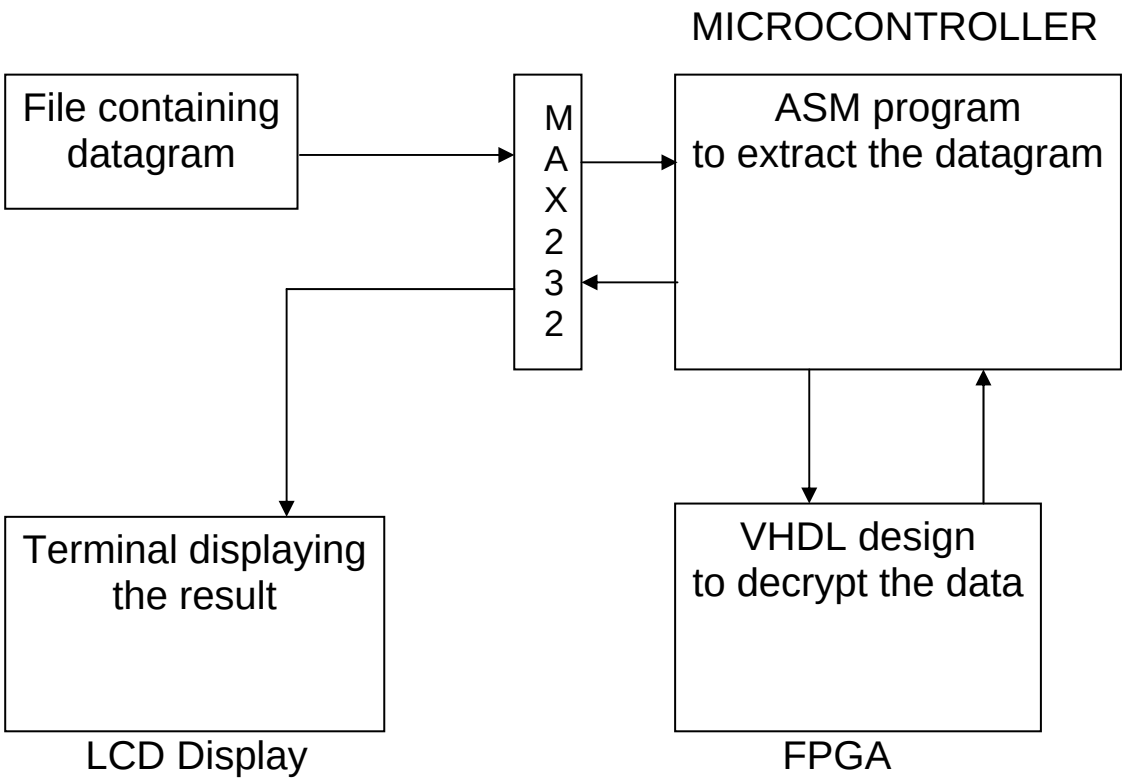

Fig 3.2: Design for extracting the data and decryption process

Chapter 4

STRUCTURE OF THE RC6 CIPHER ALGORITHM

4. STRUCTURE OF THE RC6 CIPHER ALGORITHM

4.1 BASIC OPERATIONS

RC6-*w/r/b* operates on units of four w-bit words using the following six basic operations. The base-two logarithm of w will be denoted by lg w.

.

- a + b integer addition modulo 2^w
- . a - b integer subtraction modulo 2^w
- . a xor b bitwise exclusive-or of w-bit words
- . a X b integer multiplication modulo 2^w
- a<<<b rotate the w-bit word a to the left by the amount given by the least significant lg w bits of b
- a>>>b rotate the w-bit word a to the right by the amount given by the least significant lg w bits of b

4.2 KEY SCHEDULE

The user supplies a key of b bytes. From this key, 2r + 4 words (w bits each) are derived and stored in the array S [0, 2r + 3]. This array is used in both encryption and decryption. Sufficient zero bytes are appended to give a key length equal to a non-zero integral number of words; these key bytes are then loaded in little-endian fashion into an array of c w-bit (w = 32 bits in our case) words L [0], … , L [c - 1]. Thus the first byte of key is stored as the low-order byte of L [0], etc., and L [c - 1] is padded with high-order zero bytes if necessary. The number of w bit (32 bit) words that will be generated for the additive round keys is 2r + 4 and these are stored in the array S [0; … ;2r + 3]. The constants P32 = B7E15163 and Q32 = 9E3779B9 (hexadecimal) are the same “magic constants" as used in the RC5 key schedule. Fig 4.2.1 shows how we are mixing the user supplied key with the stored array S [0, 2r+3] keys.

Procedure for Key Scheduling:

S [0] = P32
for i = 1 to 2r + 3 do
S [i] = S [i - 1] + Q32
A = B = i = j = 0
v = 3 X max{c, 2r + 4}
for s = 1 to v do
{
A = S [i] = (S [i] + A + B) <<< 3
B = L [j] = (L [j] + A + B) <<< (A + B)
i = (i + 1) mod (2r + 4)
j = (j + 1) mod c
}

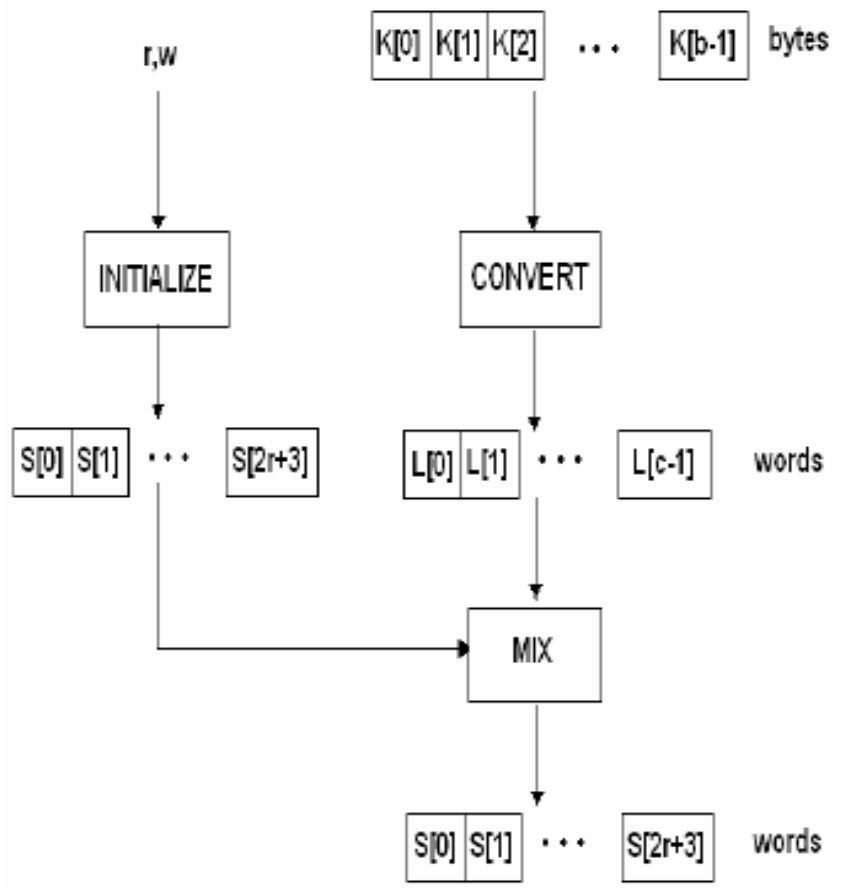

Fig 4.2.1 : RC6 Key Mix

4.3 ENCRYPTION

Encryption is the process of converting a plaintext message into cipher text which can be decoded back into the original message. An encryption algorithm along with a key is used in the encryption and decryption of data. There are several types of data encryptions which form the basis of network security. Encryption schemes are based on block or stream ciphers.

The type and length of the keys utilized depend upon the encryption algorithm and the amount of security needed. In conventional symmetric encryption a single key is used. With this key, the sender can encrypt a message and a recipient can decrypt the message but the security of the key becomes problematic. In asymmetric encryption, the encryption key and the decryption key are different. One is a public key by which the sender can encrypt the message and the other is a private key by which a recipient can decrypt the message.

RC6 works with four w-bit registers A, B, C, D which contain the initial input plaintext as well as the output cipher text at the end of encryption. The first byte of plaintext is placed in the least significant byte of A, the last byte of plaintext is placed into the most-significant byte of D. We use (A, B, C, D) = (B, C, D, A) to mean the parallel assignment of values on the right to registers on the left. Fig 4.3.1 show the RC6 algorithm.

Input:

- . Plain text stored in four w-bit input registers A, B, C, D
- . Number r of rounds
- . w-bit round keys S[0, … ,2r + 3]

Output:

- . Cipher text stored in A, B, C, D

Procedure for Encryption:

B = B + S [0]
D = D + S [1]
for i = 1 to r do
{
t = (B X (2B + 1)) <<< lg w
u = (D X (2D + 1)) <<< lg w
A = ((A ⊕ t) <<< u) + S [2i]
C = ((C ⊕ u) <<< t) + S [2i+ 1]
(A, B, C, D) = (B, C, D, A)
}
A = A + S [2r + 2]
C = C + S [2r + 3]

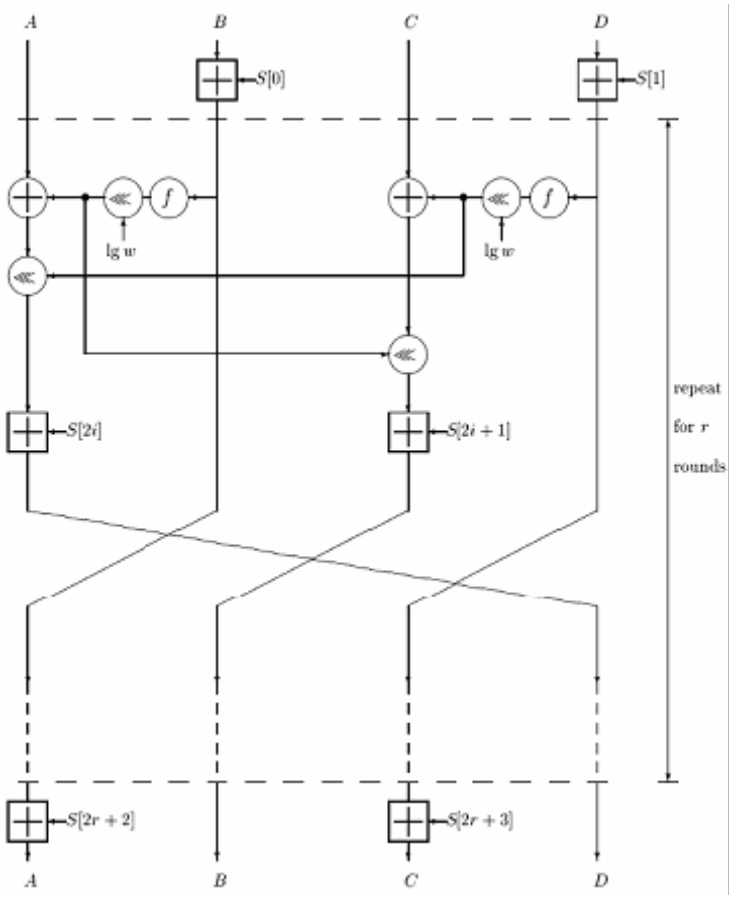

Fig. 4.3.1 : Encryption with RC6-w/r/b Here $f(X) = (X\ (2X + 1)) \bmod 2^w$

4.4 DECRYPTION

RC6 decryption works with four w-bit registers A, B, C, D which contain the initial input cipher text as well as the output plain text at the end of decryption. The first byte of cipher text is placed in the least significant byte of A, the last byte of cipher text is placed into the most-significant byte of D. We use (A, B, C, D) = (B, C, D, A) to mean the parallel assignment of values on the right to registers on the left.

Input:

- . Cipher text stored in four w-bit input registers A, B, C, D
- . Number r of rounds
- . w-bit round keys S[0; … ; 2r + 3]

Output:

- . Plaintext stored in A, B, C, D

Procedure for Decryption:

C = C – S [2r + 3]
A = A – S [2r + 2]
for i = r downto 1 do
{
(A, B, C, D) = (D, A, B, C)
u = (D X (2D + 1)) <<< lg w
t = (B X (2B + 1)) <<< lg w
C = ((C – S [2i + 1]) >>> t) □u
A = ((A – S [2i]) >>> u) □t
}
D = D – S [1]
B = B – S [0]

4.5 DESIGN ANALYSIS

4.5.1MULTIPLICATION

When implementing the RC6 algorithm, it was first determined that the RC6 modulo 2^{32} multiplication was the dominant element of the round function in terms of required logic resources. Each RC6 round requires two copies of modulo 2^{32} multiplier. However, it was found that the RC6 round function does not require a general modulo 2^{32} multiplier. The RC6 multipliers implement the function A (2A + 1) which may be implemented as $2A^2 + A$. Therefore, the multiplication operation was replaced with an array squarer with summed partial products, requiring fewer hardware resources and resulting in a faster implementation.

4.5.2 VARIABLE SHIFTING

Variable shifting operations have the potential to require considerable hardware resources, the 5-bit variable shifting required by the RC6 round function required few hardware resources. Instead of implementing a 32-to-1 multiplexer for each of the thirty-two rotation output bits (controlled by the five shifting bits), a multi-level multiplexing approach was used. The variable rotation is broken into multiple stages, each of which is controlled by one of the five shifting bits. For each rotation output bit of a given stage, a 2-to-1 multiplexer controlled by the stage's shifting bit is used. This implementation requires a total of 160 2-to-1 multiplexers as opposed to the thirty-two 32-to-1 multiplexers required for a one-stage implementation. However, using 2-to-1 multiplexers to form the five-stage barrel-shifter results in an overall implementation that is smaller and faster when compared to the one-stage barrel- shifter implementation.

4.5.3 OTHER OPERATIONS

The remaining components of the RC6 round functions, consisting of fixed shifting, bit-wise XOR, and modulo 2^{32} addition, were found to be simple in structure, and requiring few hardware resources.

4.6 DESIGN ARCHITECTURE

4.6.1 RC6 Key Schedule Module

The majority of the research papers done so far about the RC6 algorithm and its implementation in hardware, and more specifically in FPGAs, assume that key scheduling is done outside of the FPGA. All of the sub keys are downloaded to the key storage unit of the FPGA and are then used in both encryption and decryption. Our project is different in the sense that we are performing key scheduling and generating all of the sub keys inside the FPGA. Once the key schedule algorithm has executed and all of the sub keys have been generated, encryption and decryption will be started. If the user wishes to input a new key, the key schedule algorithm will run again and a new set of sub keys will be generated to be later used in en encryption and decryption. Fig 4.6.1 shows the diagram for RC6 Key Schedule Module

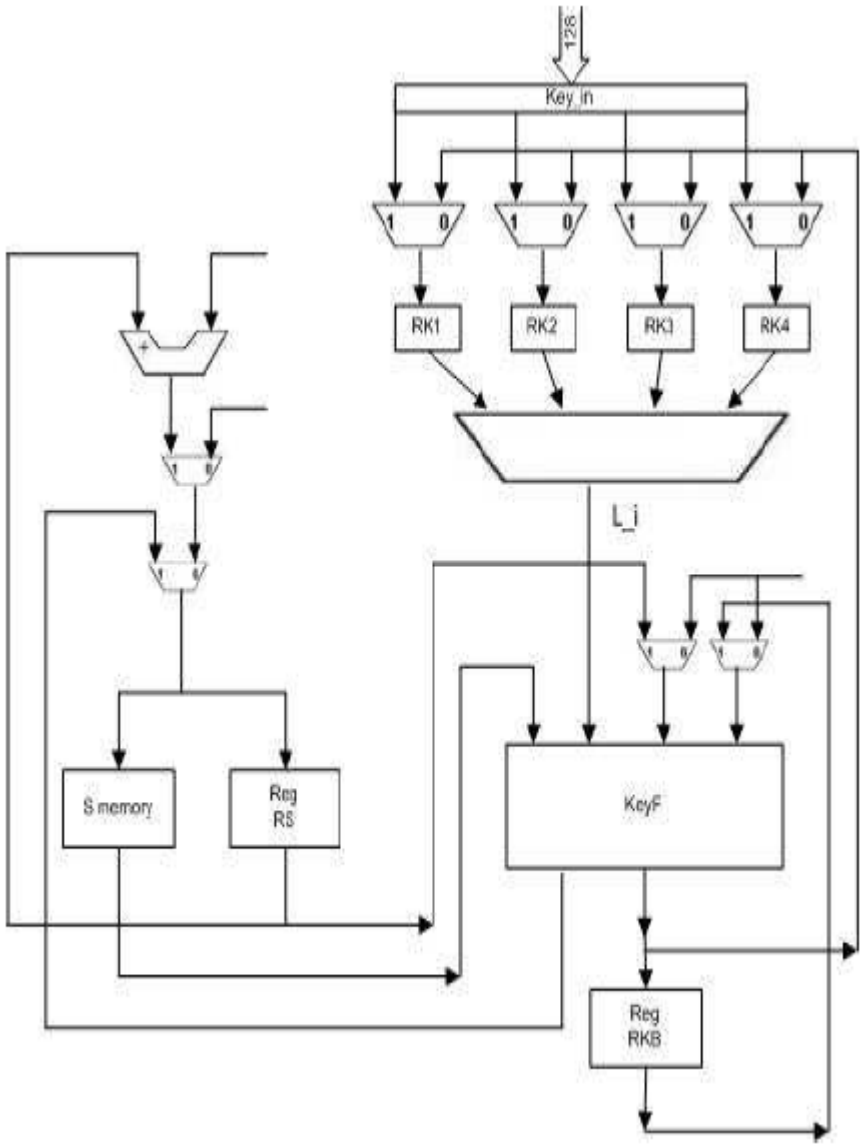

Fig.4.6.1 - RC6 Key Schedule Module

4.6.2 RC6 Main Module

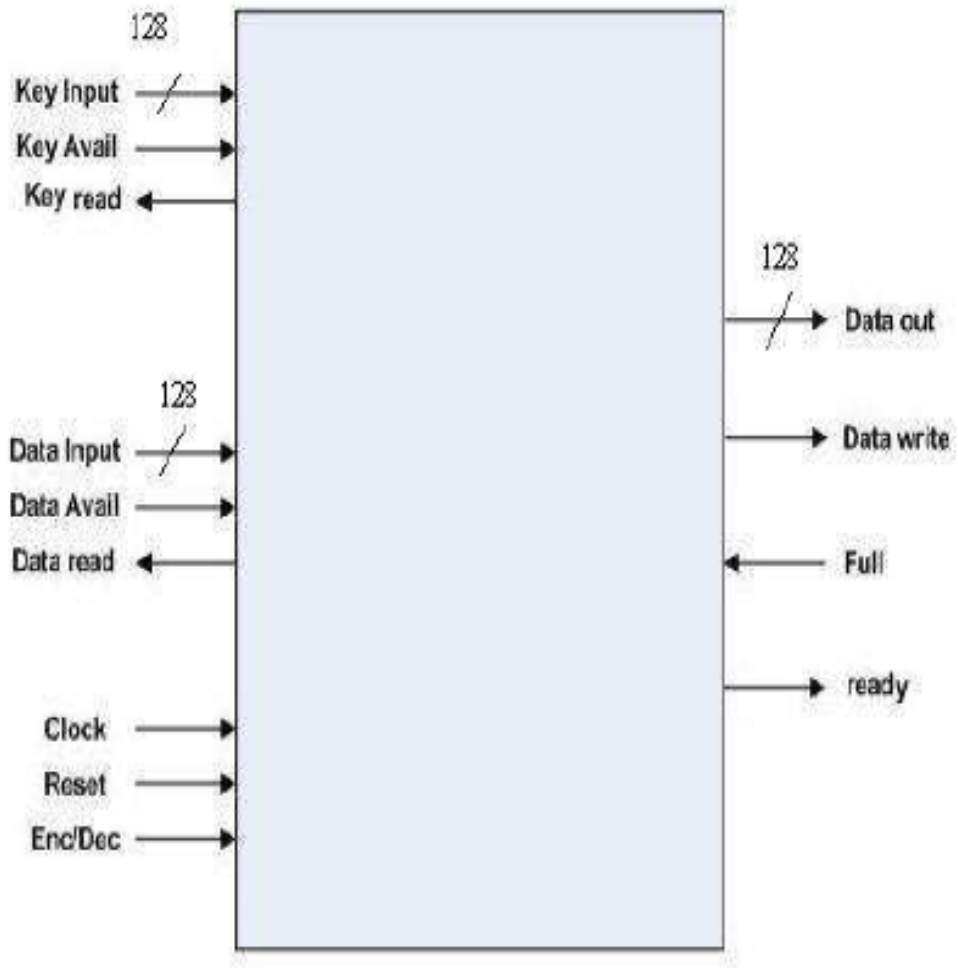

Fig. 4.6.2 - RC6 Main Module

Input:

- **Key Input:** Key to be used by ecnr/decr
- **Key Avail:** Indicates that the key is available to be read
- **Data Input:** Message/Cipher text is entered into the cipher
- **Data Avail:** Indicates data is available to be read for enc/dec
- **Clock:** Master Clock
- **Reset:** Master Reset

- **Enc/Dec:** Enc/Dec 0/1 encryption/ decryption selection
- **Full:** Indicates output full and cannot output data

Output:

- **Key Read:** Indicates the key has been read
- **Data Read:** Entered into the cipher
- **Data Out:** Cipher text/ Plaintext is output through this port
- **Data Write:** Data becomes available on output bus
- **Ready:** Indicates that the key has been generated and the unit is ready for enc/dec.

4.6.3 RC6 Core Module

The RC6 core module is where the function f(X) = (X × (2X + 1)) mod 2w is implemented. As we can see the data is first broken down to four words, each 32 bits wide represented by A, B, C and D. Key scheduler prepares two 32 bit words from the S array, one value from the even addresses and one from the odd addresses. In the case of encryption A and C are added with these two values from S. Also u and t are calculated using the function f. u and t are shifted by 5 before they are Xored with output from the barrel shifter. Fig 4.6.3 shows the diagram for core module for RC6 algorithm.

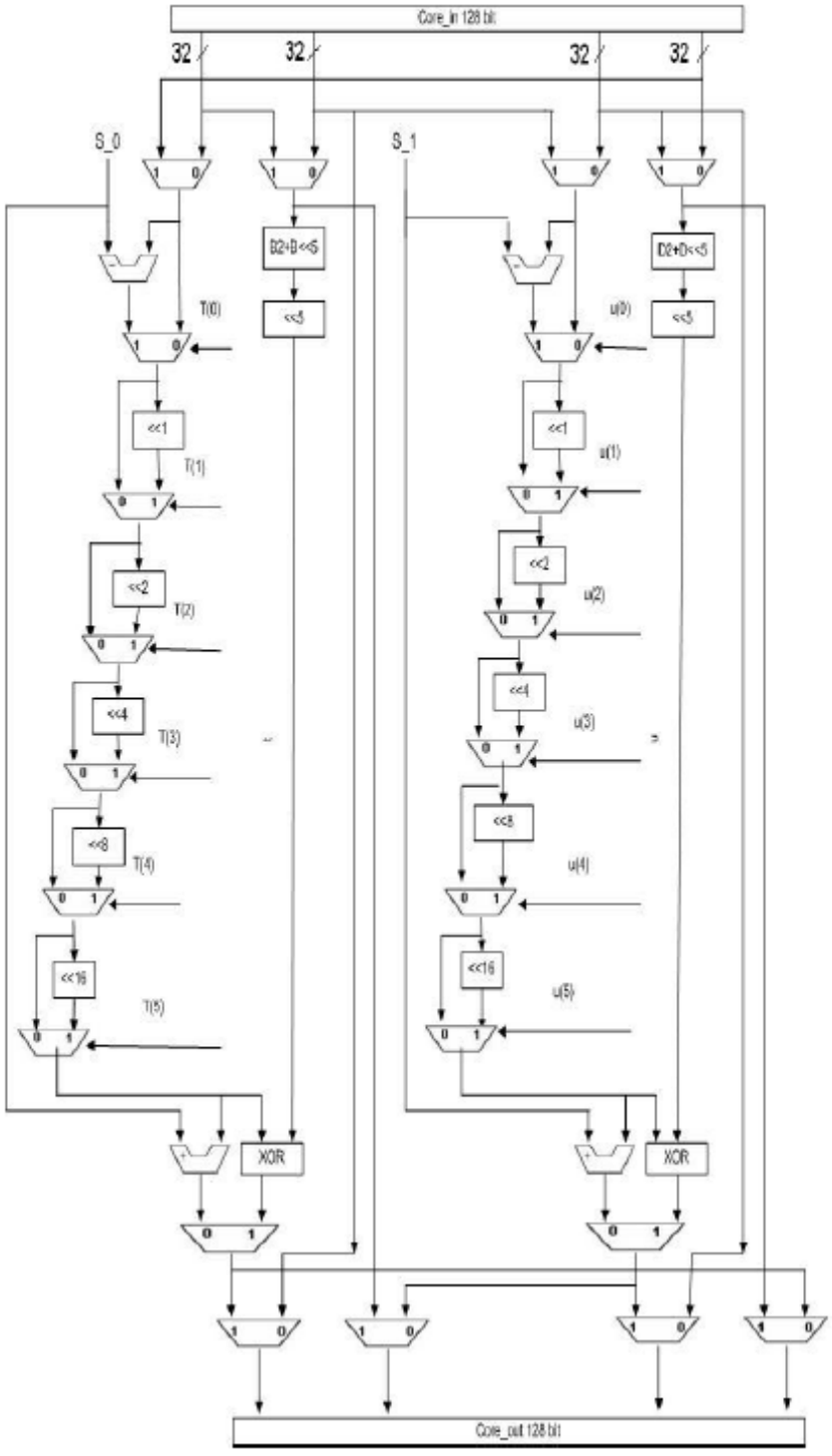

Fig. 4.6.3 - RC6 Core Module

4.6.4 RC6 Block diagram

To begin with, the data is first read in 128 bits and broken down to 4 x 32 bits words (A, B, C and D). Initially, and in case of encryption, the first two words in the S array are added to B and D. For Decryption, the two words are subtracted from C and A. These four blocks make the initial 128 bits that will be fed to a register before going into the core module through a multiplexer that controls the input for the core for every round. After completing all the rounds the output is sent to a register where it will be saved. Finally, this 128 bit is broken down to four blocks again, so the final addition and subtraction will be done before sending it as the cipher data. RC6 block diagram is shown in below Fig 4.6.4

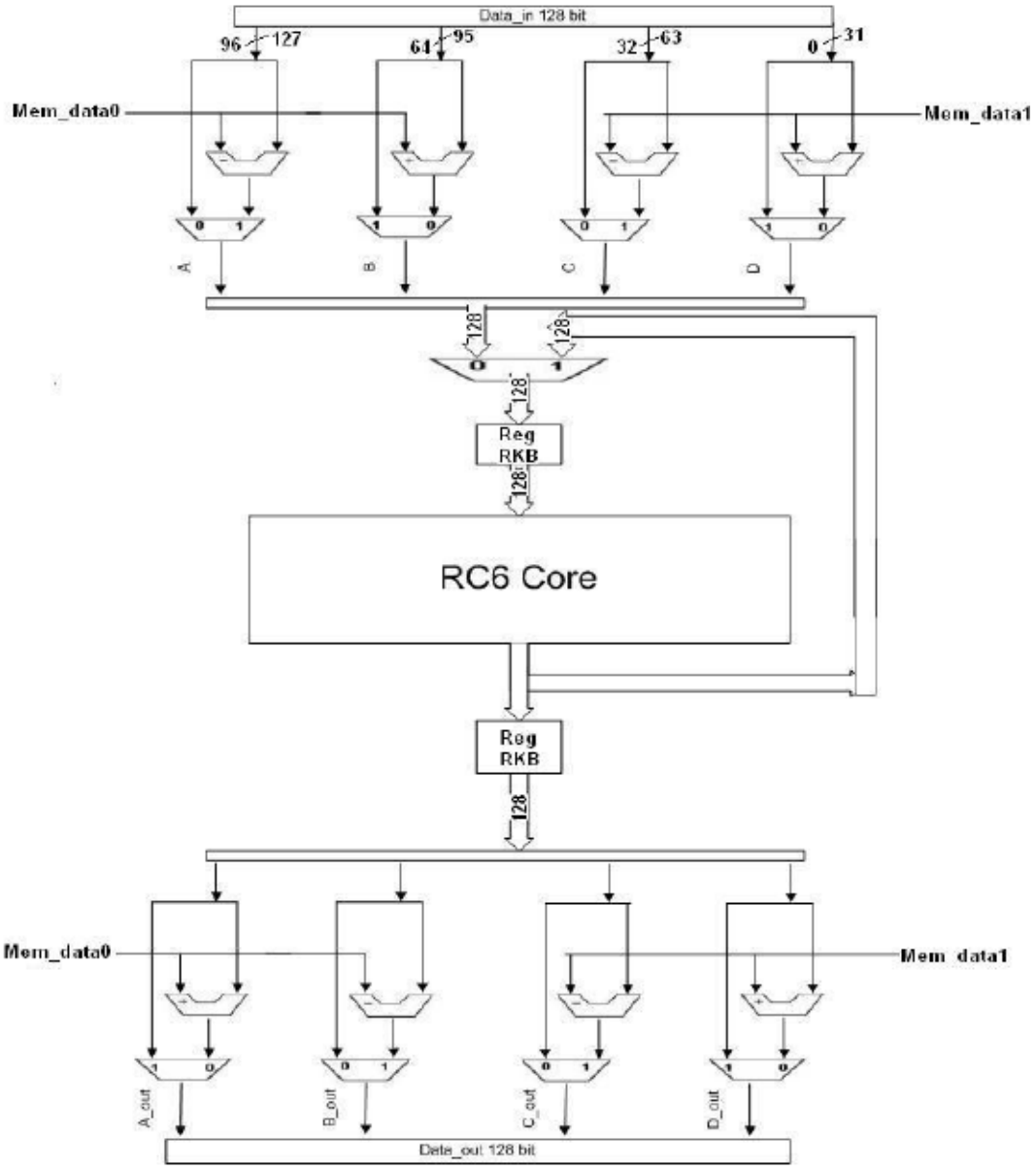

Fig 4.6.4 : RC6 Block diagram

4.6.5 Control Unit

The control unit for RC6 is a very complete one due to the fact that it also generates different signals for generating the array of S keys. Two counters are controlled using these signals. A 5 bit counter is used in key generating and preparing the array S of keys in the rounds. Each control signal is controlled by a state and in some cases by other values as well. This unit also generates output signals for feeding the data in and sending the data out. The ASM chart shows when the signals are set and reset. The diagram is shown in Figure 4.6.5.1.

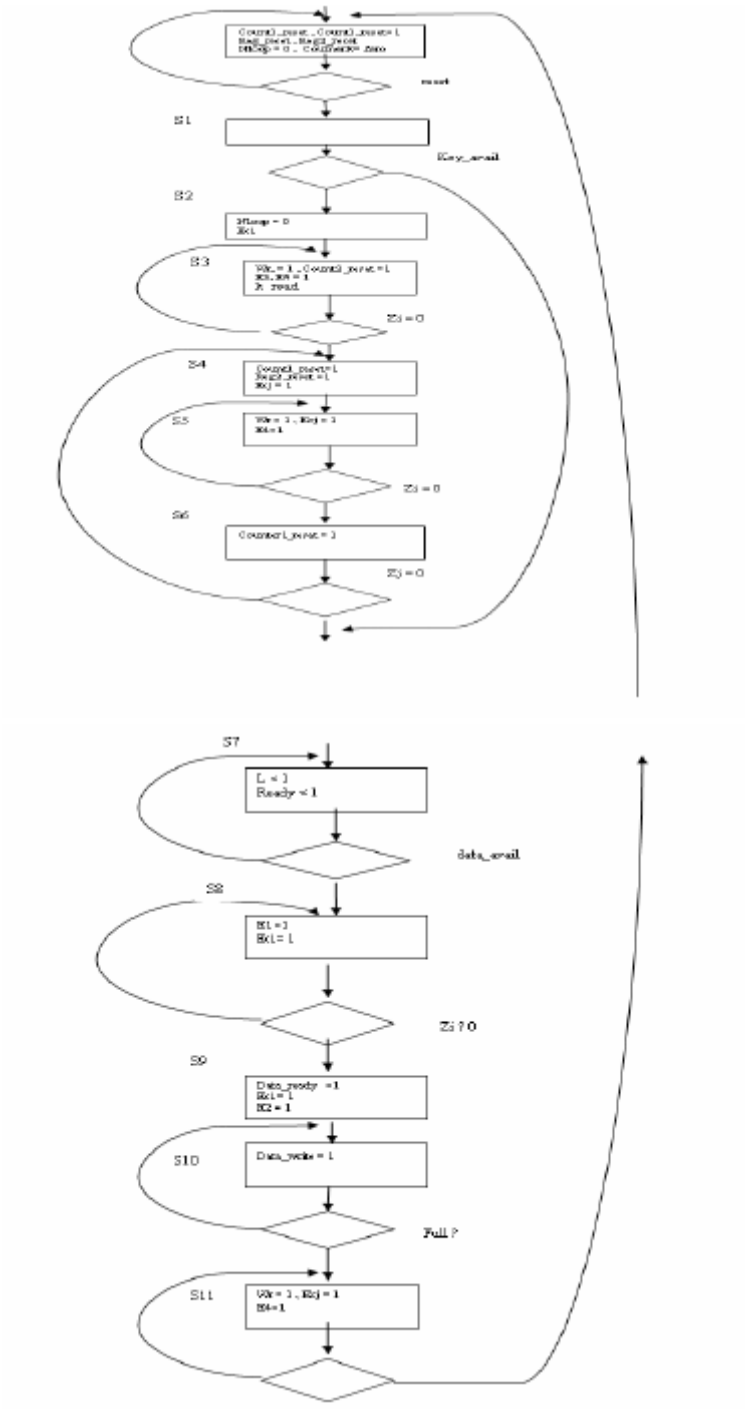

Fig. 4.6.5.1 – ASM chart of the Control Unit

The next block diagram in Fig. 4.6.5.2 shows the signals needed to control key generation for encryption/decryption units.

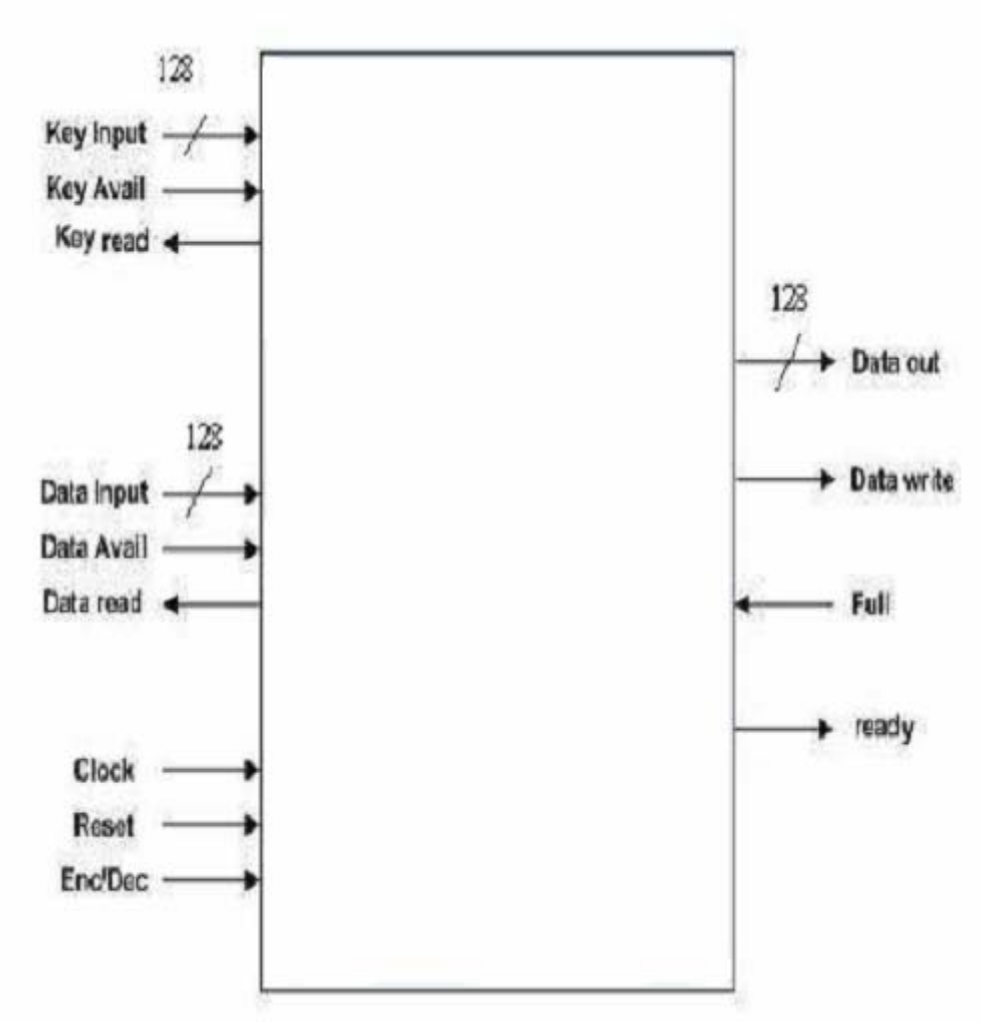

Fig. 4.6.5.2 – Control Unit

Chapter 5

STRUCTURE OF THE IPSec PROTOCOL

5. STRUCTURE OF THE IPSec PROTOCOL

IPSEC is a framework for security that operates at the Network Layer by extending the IP packet header (using additional protocol numbers, not options). This gives it the ability to encrypt any higher layer protocol, including arbitrary TCP and UDP sessions, so it offers the greatest flexibility of all the existing TCP/IP cryptosystems. Flexibility, however, often comes at the price of complexity, and IPSEC is not an exception. Configuring which addresses and ports to encrypt using which IPSEC options often begins to look like configuring packet filtering, then add in the additional complexities of key management. While conceptually simple, setting up IPSEC is much more complex that installing SSH, for example. The IP security architecture uses the concept of a security association as the basis for building security functions into IP. A security association is simply the bundle of algorithms and parameters (such as keys) that is being used to encrypt and authenticate a particular flow in one direction. Therefore, in normal bi-directional traffic, the flows are secured by a pair of security associations. The actual choice of encryption and authentication algorithms (from a defined list) is left to the IPSec administrator.

There are two modes of **IPSec** operation: **transport mode** and **tunnel mode**.

5.1 Transport mode

In transport mode, only the payload (the data you transfer) of the IP packet is encrypted and/or authenticated. The routing is intact, since the IP header is neither modified nor encrypted; however, when the authentication header is used, the IP addresses cannot be translated, as this will invalidate the hash value. The transport and application layers are always secured by hash, so they cannot be modified in any way. Transport mode is used for host-to-host communications.

5.2 Tunnel mode

In tunnel mode, the entire IP packet (data plus the message headers) is encrypted and/or authenticated. It must then be encapsulated into a new IP packet for routing to work. Tunnel

mode is used for network-to-network communications or host-to-network and host-to-host communications over the internet.

Two protocols have been developed to provide packet-level security for IPv6.

- The IP Authentication Header provides integrity and authentication and non-repudiation, if the appropriate choice of cryptographic algorithms is made.
- The IP Encapsulating Security Payload provides confidentiality, along with optional (but strongly recommended) authentication and integrity protection.

5.3 Authentication header (AH)

The AH is intended to guarantee connection less integrity and data origin authentication of IP data grams. Further, it can optionally protect against replay attacks by using the sliding window technique and discarding old packets. AH protects the IP payload and all header fields of an IP datagram except for mutable fields.

0 - 7 bit	8 - 15 bit	16 - 23 bit	24 - 31 bit
Next header	Payload length	RESERVED	
Security parameters index (SPI)			
Sequence number			
Authentication data (variable)			

Fig 5.3 : AH packet diagram

Field meanings:

Next header
Identifies the protocol of the transferred data.
Payload length
Size of AH packet.
RESERVED
Reserved for future use (all zero until then).

Security parameters index (SPI)

Identifies the security parameters, which, in combination with the IP address, then identify the security association implemented with this packet.

Sequence number

A monotonically increasing number, used to prevent replay attacks.

Authentication data

Contains the integrity check value (ICV) necessary to authenticate the packet; it may contain padding.

5.4 Encapsulating Security Payload (ESP)

The ESP protocol provides origin authenticity, integrity, and confidentiality protection of a packet. ESP also supports encryption-only and authentication-only configurations, but using encryption without authentication is strongly discouraged because it is insecure. Unlike AH, the IP packet header is not protected by ESP.

<table>
<tr><th>0 – 7 bit</th><th>8 – 15 bit</th><th>16 – 23 bit</th><th>24 – 31 bit</th></tr>
<tr><td colspan="4">Security Parameter Index (SPI)</td></tr>
<tr><td colspan="4">Sequence number</td></tr>
<tr><td colspan="4">Payload data (variable)</td></tr>
<tr><td></td><td colspan="2">Padding (0 to 255 bytes)</td><td></td></tr>
<tr><td></td><td></td><td>Pad length</td><td>Next Header</td></tr>
<tr><td colspan="4">Authentication data (variable)</td></tr>
</table>

Fig 5.4 : An ESP packet diagram

Field meanings:

Security parameters index (SPI)
Identifies the security parameters in combination with IP address.
Sequence number
A monotonically increasing number, used to prevent replay attacks.
Payload data
The data to be transferred.
Padding
Used with some block ciphers to pad the data to the full length of a block.
Pad length
Size of padding in bytes.
Next header
Identifies the protocol of the transferred data.
Authentication data
Contains the data used to authenticate the packet.

5.5 Point-to-Point Protocol

The Point-to-Point Protocol (PPP) originally emerged as an encapsulation protocol for transporting IP traffic over point-to-point links. PPP also established a standard for the assignment and management of IP addresses, asynchronous (start/stop) and bit-oriented synchronous encapsulation, network protocol multiplexing, link configuration, link quality testing, error detection, and option negotiation for such capabilities as network layer address negotiation and data-compression negotiation. PPP supports these functions by providing an extensible Link Control Protocol (LCP) and a family of Network Control Protocols (NCPs) to negotiate optional configuration parameters and facilities. In addition to IP, PPP supports other protocols, including Novell's Inter network Packet Exchange (IPX) and DECnet.

PPP Components

PPP provides a method for transmitting data grams over serial point-to-point links. PPP contains three main components:

- A method for encapsulating data grams over serial links. PPP uses the High-Level Data Link Control (HDLC) protocol as a basis for encapsulating data grams over point-to-point links.
- An extensible LCP to establish, configure, and test the data link connection.
- A family of NCPs for establishing and configuring different network layer protocols. PPP is designed to allow the simultaneous use of multiple network layer protocols.

General Operation

To establish communications over a point-to-point link, the originating PPP first sends LCP frames to configure and (optionally) test the data link. After the link has been established and optional facilities have been negotiated as needed by the LCP, the originating PPP sends NCP frames to choose and configure one or more network layer protocols. When each of the chosen network layer protocols has been configured, packets from each network layer protocol can be sent over the link. The link will remain configured for communications until explicit LCP or NCP frames close the link, or until some external event occurs (for example, an inactivity timer expires or a user intervenes).

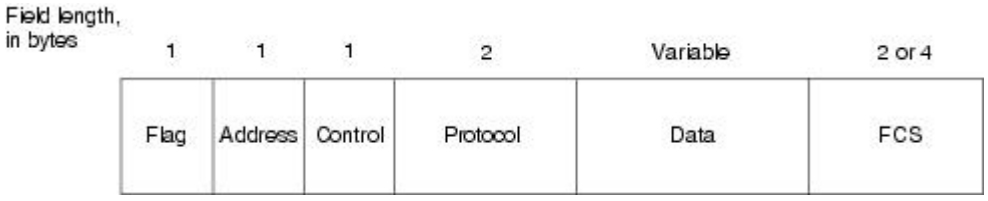

Fig 5.5 : Six Fields Make Up the PPP Frame

The following descriptions summarize the PPP frame fields illustrated in Figure 5.5:

- **Flag**— A single byte that indicates the beginning or end of a frame. The flag field consists of the binary sequence 01111110.

- **Address**— A single byte that contains the binary sequence 11111111, the standard broadcast address. PPP does not assign individual station addresses.

- **Control**— A single byte that contains the binary sequence 00000011, which calls for transmission of user data in an unsequenced frame. A connectionless link service similar to that of Logical Link Control (LLC) Type 1 is provided.

- **Protocol**— Two bytes that identify the protocol encapsulated in the information field of the frame. The most up-to-date values of the protocol field are specified in the most recent Assigned Numbers Request For Comments (RFC).

- **Data**— Zero or more bytes that contain the datagram for the protocol specified in the protocol field. The end of the information field is found by locating the closing flag sequence and allowing 2 bytes for the FCS field. The default maximum length of the information field is 1,500 bytes. By prior agreement, consenting PPP implementations can use other values for the maximum information field length.

- **Frame check sequence (FCS)**—Normally 16 bits (2 bytes). By prior agreement, consenting PPP implementations can use a 32-bit (4-byte) FCS for improved error detection.

The LCP can negotiate modifications to the standard PPP frame structure. Modified frames, however, always will be clearly distinguishable from standard frames.

PPP Link-Control Protocol

The PPP LCP provides a method of establishing, configuring, maintaining, and terminating the point-to-point connection. LCP goes through four distinct phases.

First, link establishment and configuration negotiation occur. Before any network layer datagrams (for example, IP) can be exchanged, LCP first must open the connection and negotiate configuration parameters. This phase is complete when a configuration-acknowledgment frame has been both sent and received.

This is followed by link quality determination. LCP allows an optional link quality determination phase following the link-establishment and configuration-negotiation phase. In this phase, the link is tested to determine whether the link quality is sufficient to bring up

network layer protocols. This phase is optional. LCP can delay transmission of network layer protocol information until this phase is complete.

At this point, network layer protocol configuration negotiation occurs. After LCP has finished the link quality determination phase, network layer protocols can be configured separately by the appropriate NCP and can be brought up and taken down at any time. If LCP closes the link, it informs the network layer protocols so that they can take appropriate action.

Finally, link termination occurs. LCP can terminate the link at any time. This usually is done at the request of a user but can happen because of a physical event, such as the loss of carrier or the expiration of an idle-period timer.

Three classes of LCP frames exist. Link-establishment frames are used to establish and configure a link. Link-termination frames are used to terminate a link, and link-maintenance frames are used to manage and debug a link. These frames are used to accomplish the work of each of the LCP phases.

Chapter 6

STEPS OF THE THESIS

6. STEPS OF THE THESIS

There are five several steps in this thesis as fallows

A. PC-Microcontroller communication
B. Datagram definition
C. Crypto-coprocessor to encrypt the data
D. Datagram validation and extraction
E. Crypto-coprocessor to decrypt the data
F. Complete system

From one PC side the data is encrypted by using crypto-coprocessor, then defining the necessary information to the data, i.e. adding the headers to the data. Then sending this encrypted data serially to the Microcontroller. The Microcontroller will receive the data and extract the data. After extracting the data it sends to the crypto-coprocessor to decrypt the data. After getting the original data it will displayed on the terminal.

6.1 PC – Microcontroller communication

The PC was sending the data to the microcontroller using a point-to-point protocol over a serial link. In this part of the thesis, the task was to write C code which would enable the serial transmission of a text file (the data), to the Microcontroller and then to write assembly code which would take in the serial transmission and store it into Microcontroller memory. Without any data being sent, there would be no way of testing the Microcontroller code for storing the data into memory. Other reasons for C code being done first was that it was simpler, not requiring any hardware other than a PC. Serial transmission of the data was accomplished by using the windows.h library, which allowed for com ports to be selected and used to send or receive data at specified baud rates. Once the com port was selected and setup, the rest of the code was simply a matter of opening and preparing the datagram file to be sent through it serially to the Microcontroller. The assembly was then written for storage of the

data sent by the C code. Once the first byte was detected, it would be stored into Microcontroller memory.

6.2 Datagram definition

In a general network stack, data is encapsulated inside of a frame by appending fields to the beginning and end of the data. The datagram represents the final result of data that has been encapsulated by the following process:

1. The original data is wrapped in an IPv6 packet
2. This IPv6 packet is encrypted, which encrypts only the data portion of the packet, and the corresponding Authentication headers are inserted in the packet.
3. This encrypted IPv6 packet is then encapsulated in a PPP frame, which includes data appended to both the beginning and end of the frame.

The IP security architecture uses the concept of a security association as the basis for building security functions into IP. A security association is simply the bundle of algorithms and parameters (such as keys) that is being used to encrypt and authenticate a particular flow in one direction. Therefore, in normal bi-directional traffic, the flows are secured by a pair of security associations. The actual choice of encryption and authentication algorithms (from a defined list) is left to the IPsec administrator.

This complete structure is the final datagram. This structure requires several fields for each step from the above list. The first step in creating this datagram was to define each of the values for the protocol headers. After determining each constant value and computing the dynamic values, a ASM program was defined to construct the datagram. The initial ASM program was intended to construct a single static datagram in memory, then save that datagram as a file. Starting in this manner provided an easy upgrade path for the planned modification to the code to produce a dynamic datagram based on user input or an input file. After writing the initial ASM code to hold the required data, the required functions for the

dynamic fields were researched. Specifically, an RC6 encryption algorithm inserted into the code.

After completing the encryption task the data was sent into the datagram file and store into the Microcontroller memory. Once the data is received from the crypto-processor it is displayed on a terminal as shown in Figure 2.

6.3 Crypto-coprocessor to encrypt the data

The first step in the design of the FPGA coprocessor was to define a bus protocol between the Microcontroller and the FPGA. The protocol was to take into account the asynchronous properties of the two devices. In order to implement such a protocol, a hand shake method was used. With the bus protocol decision finalized a high level FSM (Finite State Machine) was designed. The FSM was then split into separate modules: the data input; the encryptor, and the data output. The block diagram of crypto-coprocessor was shown in figure 6.3. The beginning of the FPGA consisted of the serial to parallel converter. This module was designed to take one bit at a time and to output 4 x 32 bits to the encryptor. This is also named as serial to parallel converter. Each set of 32 bits output to the decryptor consisted of 32 originally input, so the input sequence required four buffers, each capable of holding 32 bits. The middle of the FPGA code was a encryptor which encrypted 128-bit plain text blocks into 128-bit cipher text blocks using the RC6 algorithm. Here we are using RC6 algorithm including key scheduling. The protocol between the input/output modules and the encryptor had to be established. The end of the FPGA consisted of the output. The output took in and stored 4 x 32 bits at once. Each 32-bit word was stored into a 32-bitwide register. Again here one parallel serial converter is need to send the data. Each bit then sent to the Microcontroller each time the Microcontroller requested data, serially. The encryption process is as fallows.

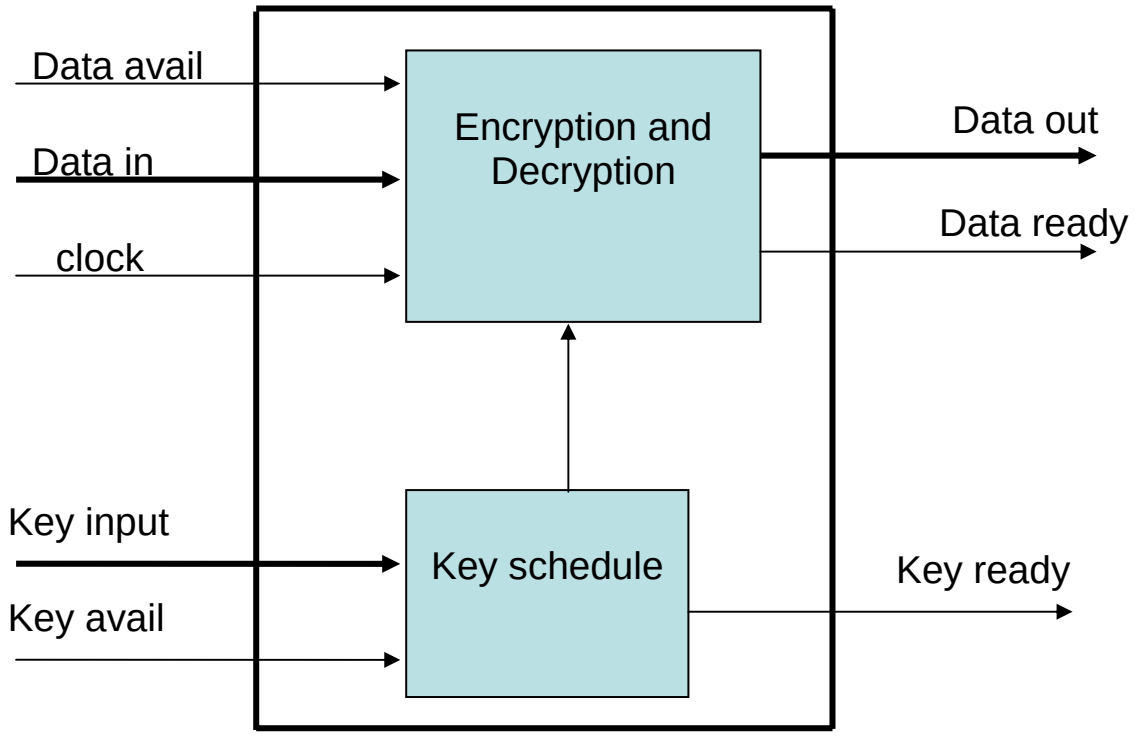

Fig 6.3 : Crypto-processor block diagram

6.4 Datagram validation and data extraction

Following the process of storing the datagram in Micocontroller memory, it is required that the Microcontroller checks the validity of certain parameters within the datagram, most importantly the Payload, Destination Address, and Data portions. These parameters are checked to verify the validity of the communication. The first step in extracting this datagram was to define each of the values for the protocol headers. After determining each constant value and computing the dynamic values, a ASM program was defined to extract the datagram. Following the extraction and checking of each parameter, the next step is to send

the encrypted data portion of the datagram to the crypto-processor. The crypto-processor performs decryption and sends the original data (the data before encryption) back to the microcontroller to be stored back in Microcontroller memory. The encrypted data will be written over the decrypted data in the same memory range. Once the data is received from the crypto-processor it is displayed on a terminal as shown in Figure 3.

6.5 Crypto-coprocessor to decrypt the data

In this task also we will use same procedure as in encrypting the data. The first step was to define a bus protocol between the Microcontroller and the FPGA. The protocol was to take into account the asynchronous properties of the two devices. In order to implement such a protocol, a hand shake method was used. With the bus protocol decision finalized a high level FSM (Finite State Machine) was designed. The FSM was then split into separate modules: the data input; the decryptor; and the data output. The beginning of the FPGA consisted of the input. This module was designed to take one bit at a time and to output 4 x 32 bits to the decryptor. Each set of 32 bits output to the decryptor consisted of 32 originally input, so the input sequence required four buffers, each capable of holding 32 bits. The middle of the FPGA code was a decryptor which decrypted 128-bit cipher text blocks into 128-bit plaintext blocks using the RC6 algorithm. The protocol between the input/output modules and the decryptor had to be established. The end of the FPGA consisted of the output. The goal of the output was to perform the reverse of the input module. The output took in and stored 4 x 32 bits at once. Each 32-bit word was stored into a 32-bitwide register. Each bit sent to the Microcontroller each time the Microcontroller requested data, serially. The decryption process is as fallows.

6.6 Complete system

The task is to integrating the parts of the system into a whole. This involved a debugging stage. Many bugs were encountered and dealt with. It was necessary that all of the portions fit and work together as though they were one homogonous piece of code. This involved aligning the variable names, adding code to join two parts, removing parts what were unnecessary or duplicated, and making sure the bus communications matched on both sides. Another design choice used when combining the different parts of the project was being consistent for what banks of memory were used for what purposes.

Chapter 7

RESULTS

7. RESULTS

Use VHDL to simulate hardware implementation. Field Programmable Gate Arrays (FPGAs) consist of arrays of configurable logic blocks that implement logical functions of gates that are easily reconfigurable. In contrast, Application Specific Integrated Circuits (ASICs) provide only the functionality needed for a specific task. An ASIC chip will support a particular application for which it is designed, but not a modified version of the same application introduced after the ASIC design is completed. On the other hand, the configuration of an FPGA can be easily reprogrammed to accommodate a design modification. Other key factors that favor the use of FPGAs for hardware implementation of ciphers include faster turnaround design time, scalable security, and variable architecture parameters. For those reasons we have chosen FPGAs as the target technology.

Selection of a target FPGA

- Xilinx Spartan3E XC3S500E

Selection of a target Microcontroller

- Philips P89C51RD2

7.1 Testing

The first step in the design process is to check for the functional correctness of the design using simulations. Then, the FPGA synthesis tool is used to interpret logic components from the VHDL code. The synthesis tool produces a net list which does not have accurate timing information since placement and routing of the logic components on the FPGA is not yet determined. The post-synthesis net list can however be used to check the correct inference of logic components from the VHDL code. The final step in the process is to map the design to the target FPGA. The FPGA implementation tool, which is vendor-specific, generates a net list which has accurate timing as well as logic information. The final net list is used for simulation to check if the design will actually work when configured physically on the FPGA. This type of simulation is known as timing simulation. Our design did pass timing simulation under the control of test bench. Test benches were written in VHDL in order to verify the functionality of the design. The test benches were designed to read test vectors from a file and

compare the produced output with expected outputs stored in another file. Extensive testing was done for checking both the encryption as well as decryption functionality.

7.2 Waveforms

Key setup: The following waveform shows when signal ready turns to 1 meaning that the key schedule is done and the data is being read. Here we read the input
"075978ABDEA7863946BCFA273D763DEC"
And the key input is "9876543210abcdef9876543210abcdef"

Encryption: The following waveform shows when the input "075978ABDEA7863946BCFA273D763DEC" is encrypted and the cipher text "2DE1684C2658B2E7892D7633C4E6A5A6" is outputted.

Decryption: The following waveform shows when the cipher "2DE1684C2658B2E7892D7633C4E6A5A6" is decrypted and the original text "075978ABDEA7863946BCFA273D763DEC" is outputted.

7.3 Obtained Results

After running the VHDL code, checking for functionality, synthesizing and then implementing the code, we got the following results which are summarized in the following.

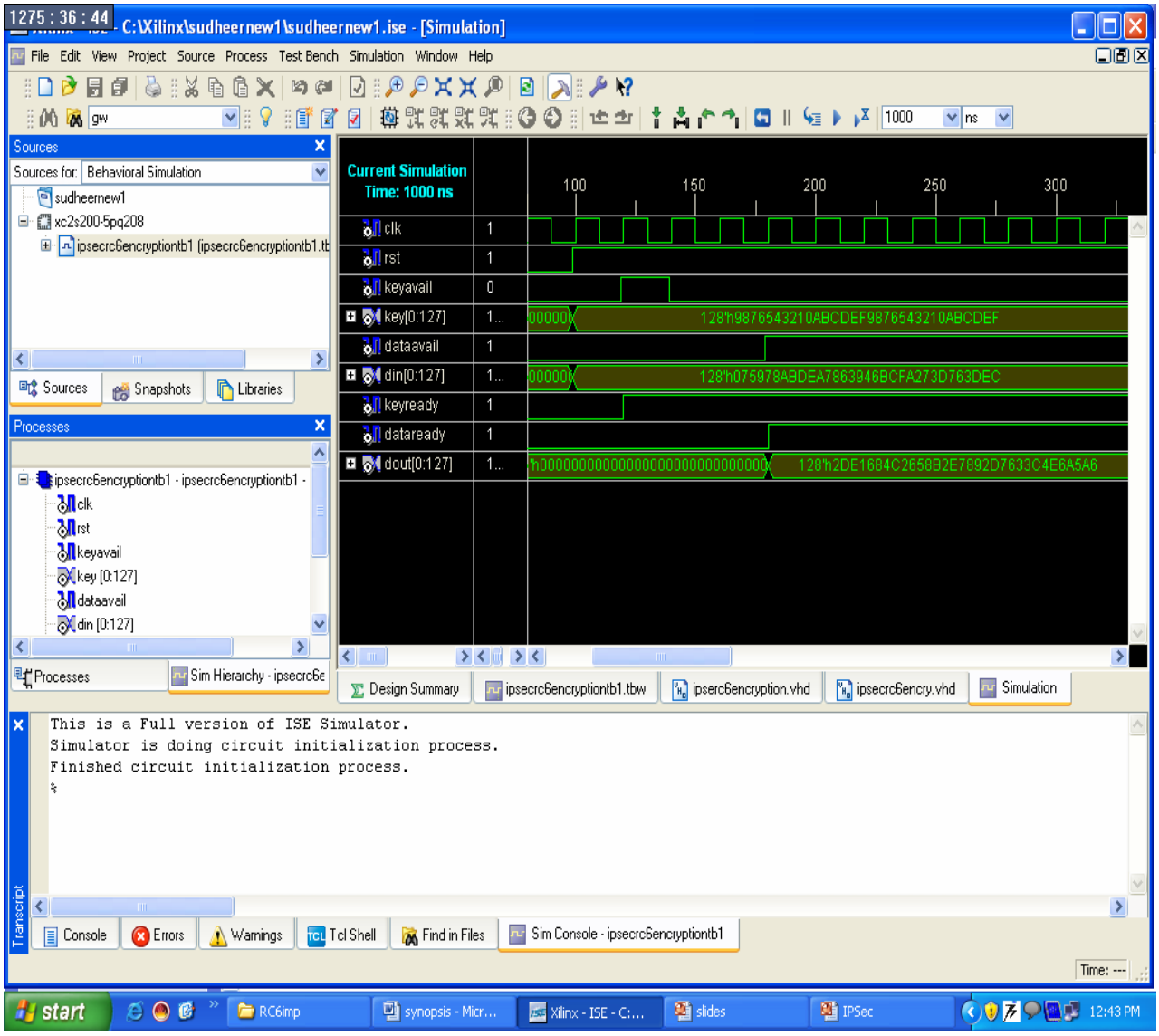

Fig 7.3.1: The result of the encryption process

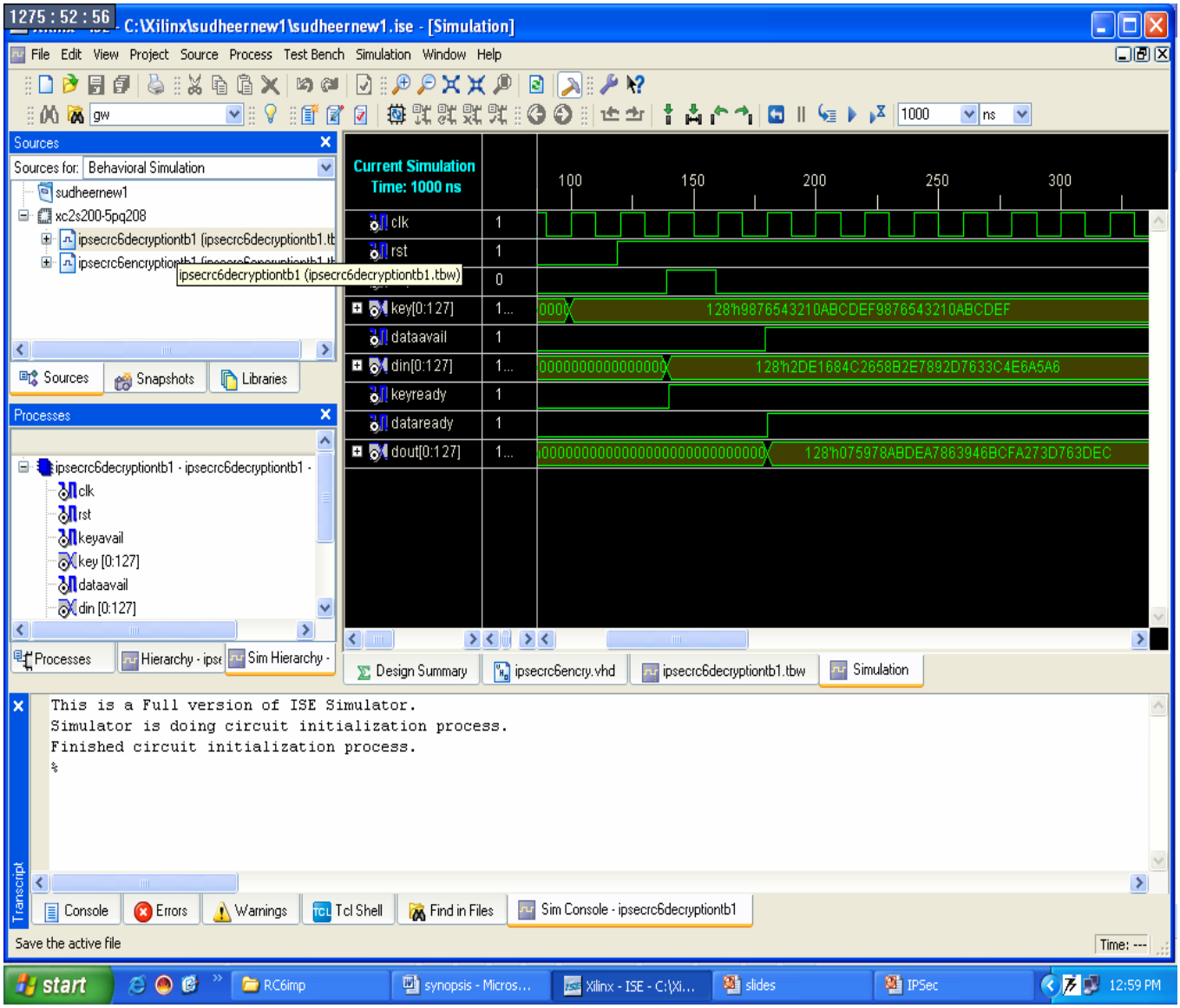

Fig 7.3.2: The result of the decryption process

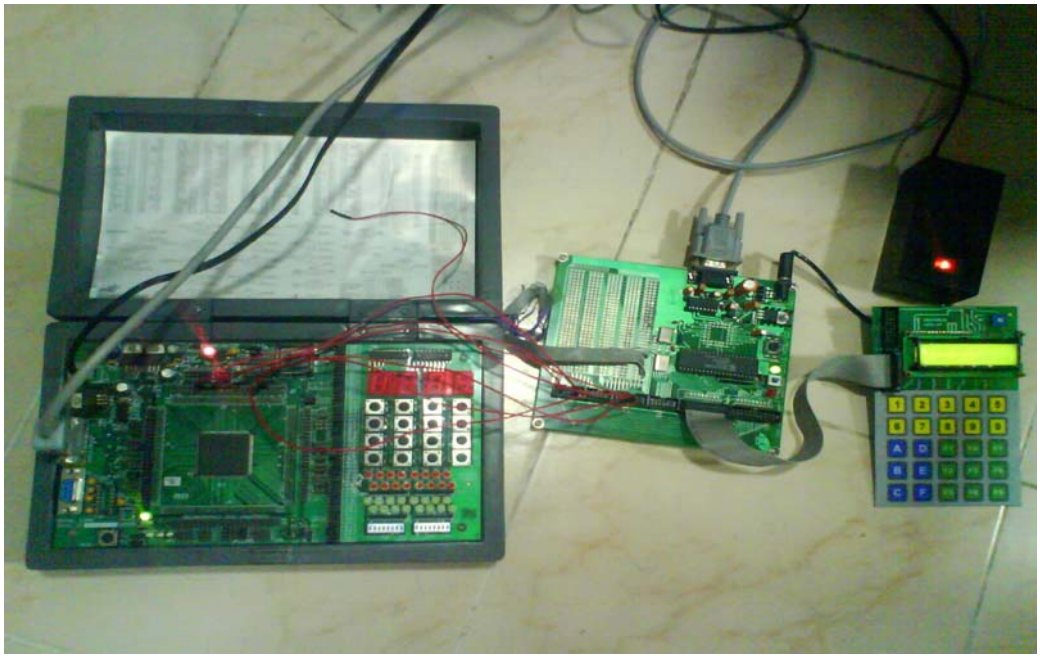

Fig 7.3.3 : Microcontroller and the links to the FPGA and the terminal

Chapter 7

CONCLUSION AND FUTURE WORK

8. CONCLUSION AND FUTURE WORK

8.1 CONCLUSIONS

RC6 is a secure, compact and simple block cipher. It offers good performance and considerable flexibility. To handle networking, processor based design and reconfigurable architecture which provide a good overview of a system level design. Another extension of the project could be to dynamically adapt the cryptography algorithms in order to take benefit of dynamic reconfiguration.

	Advantages	**Disadvantages**
With Key Schedule	No need to reload the Round keys every time the user changes the input Key	1- Lower throughput 2- More hardware resources
Without Key Schedule	1- Higher throughput 2- Less hardware resources	Every time the input key is changed, the round keys must be reloaded to the FPGA

8.2 FUTURE WORK

What we would recommend as future work/extension to this thesis is to implement the key schedule in a way that it will be able to generate the sub keys faster by pipelining the intitial S generate values in the S array. In our design, encryption/ decryption cannot be started until the key schedule algorithm has completed and all of the subkeys are generated. Key scheduling takes the most amount of time. So a possible extension to this thesis is to try

and generate the sub keys that is used for first few rounds and then generating the rest while encryption starting to use these. We see from the algorithm and in our implementation that the keys are needed faster than the time it takes to generate them. This will save a significant amount of time and improve the performance of the design.

REFERENCES

REFERENCES

[1] Mang, I. Mang, G.E.," Properties of the RC6 cipher for a BIST hardware implementation", IEEE Trans on computer, Volume 2, 7-10 Oct. 2002 Page(s):1356 - 1361 vol.2.

[2] Mang, I. Mang, G.E.," Hardware implementation with off-line test capabilities of the RC6 block cipher" , IEEE Trans on computer, Volume 2, 7-10 Oct. 2002 Page(s):1362 – 1367 vol.2

[3] J. Goodman, A. Chandrakasan, An Energy-Efficient Reconfigurable Public-Key Cryptography Processor, IEEE Journal of Solid-State Circuits, vol. 36, no. 11, November 2001, pp. 1808-1820.

[4] R.L. Rivest, "The RC5 Encryption Algorithm," available At website http://theory.lcs.mit.edu/~rivest/Rivestrc5rev.pdf

[5] Internetworking Technology Handbook. [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/

[6] IP Authentication Header. [Online]. Available: http://www.cse.ohio-state.edu/cgi-bin/rfc/rfc2402.html#sec-2

[7] NIST Advanced Encryption Standard (AES) Development Effort available at website http://www.nist.qov/aes

[8] R.L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin. The RC6 Block Cipher. In First Advanced Encryption Standard (AES) Conference, 1998.

[9] R.L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L.Yin, "The RC6 Block Cipher," available at website http://theory.csail.mit.edu/~rivest/rc6.pdf

[10] NIST Advanced Encryption Standard (AES) Development Effort available at website http://www.nist.gov/aes

[11] R.L. Rivest, "The RC5 Encryption Algorithm," Proceedings of Fast Software Encryption - 2nd International Workshop, Leuven, Belgium, Springer Verlag LNCS 1008, pp. 86-96, 1995.

[12] J.-P. Kaps and C. Paar, "Fast DES Implementation for FPGAs and its Application To a Universal Key-search Machine," presented at Workshop in Selected Areas of Cryptography (SAC'98), Kingston, Ont., Aug. 1998.

[13] K. Gaj, P. Chodowiec: Comparison of the Hardware Performance of the AES Candidates using Reconfigurable Hardware: The Third Advanced Encryption Standard Candidate Conference, April 13-14,2000, New York, USA.

[14] Wallace, C. S., “A Suggestion for a Fast Multiplier," IEEE Trans. on Computer, Vol. EC-13, pp.14-17, 1964.