Palestine Polytechnic University

College of Engineering & Technology
Electrical & Computer Engineering Department

Graduation Project

FPGA Based Compression /Security Card

Project Team

Hiyam Badr

Bayan Al-Qadi

Bayan Al-Fatafta

Project Supervisor
Eng. Elayan Abu Gharbeyeh

Co-supervisor
Eng. Radwan Tahboob

Hebron-Palestine

June, 2005

Palestine Polytechnic University

College of Engineering & Technology
Electrical & Computer Engineering Department

Graduation Project

FPGA Based Compression /Security Card

Project Team

Hiyam Badr

Bayan Al-Qadi

Bayan Al-Fatafta

Project Supervisor
Eng. Elayan Abu Gharbeyeh

Co-supervisor
Eng. Radwan Tahboob

Hebron-Palestine

June, 2005

جامعة بوليتكنيك فلسطين

الخليل-فلسطين

كلية الهندية والتكنولوجيا

دائرة الكهرباء والحاسوب

اسم المشروع:

FBGA Based Compression/Security Card

أسماء الطلبة:

هيام بدر بيان القاضي بيان فطافطة

بناء على نظام كلية الهندسة والتكنولوجيا وإشراف ومتابعة المشرف المباشر على المشروع وموافقة أعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الكهرباء والحاسوب وذلك للوفاء بمتطلبات درجة البكالوريوس في الهندسة تخصص هندسة حاسوب.

توقيع المشرف

.....

توقيع اللجنة الممتحنة

....

توقيع رئيس الدائرة

.....

Abstract

Security and speed are the main concerns for data transmitted over the internet.

This system aims to provide these two characteristics, by performing means of encryption and compression to the data comes from the micro web server to be transmitted over the internet.

This system also aims to provide an interface with microweb server's serial port.

These goals are achieved by implementing LZW compression algorithm, and hybrid encryption algorithm which includes Diffie-Hellman public key exchange and AES private encryption algorithm on FPGA by using Verilog hardware description language.

يهدف هذا النظام الى تحقيق السرعة و الامن للبيانات التي تنقل عبر الانترنت و القادمة من الميكروويب سيرفر ، من خلال ضغط و تشفيرهذه البيانات باستخدام خوازرمية الضغط (LZW) و خوارزميات التشفير (Diffie-Hellman public key exchange) و (AES private encryption) لبرمجة FPGA بواسطة لغة Verilog.

Dedication

To our parents, brothers and sisters.

To our teachers and colleagues.

To every one give us the encouragement and support to reach our goals.

Project team

Hiyam Badr

Bayan Qadi

Bayan Fatafta

Acknowledgement

There were a lot of anonymous soldiers behind our project who we want to thank. Starting with our supervisor **Eng.Elayan AbuGharbeyeh** and ending with each and every single person who attributed in letting this project to express what we have gained during our college educational journey.

We would like also to thank our university "Palestine Polytechnic University" which has provided us with needed equipment and knowledge.

Project team

Hiyam Badr

Bayan Qadi

Bayan Fatafta

Table of contents

List of Tables

List of Figures

Abbreviations

FPGA: Field Programmable Gate Array

AES: Advanced Encryption Standard

LZW: Lempel Ziv Welch Lossless Algorithm

CPLD: Complex Programmable Logic Device

ASIC: Application Specific Integration Circuit

HDL: Hardware Description Language

RX: Receiver

TX: Transmitter

ECU: Encryption Compression Unit

Chapter One

Introduction

1.1 Overview

1.2 Literature Review

1.3 Estimated Cost

1.4 Time Plan

1.5 Report Contents

1.1 Overview

Data transmission over the internet became a demand of many companies and individuals to facilitate communications and improve their work, but the transmission must have two characteristics: security and speed.

The first goal which is security can be achieved by encrypting data using one of encryption algorithm, so the encrypted data can't be read without decrypting it using the encryption key which is only known by the sender and the receiver.

Speed can be achieved be compressing data and reducing its size so transmitting more data in less time; the receiver must decompress data before reading it.

The main objective of this project is to secure and accelerate data transmission between micro web server and the internet by performing data compression/ decompression, encryption/ decryption, and serial port interfacing, which will be implemented by field programmable gate array (FPGA).

1.2 Literature review

There are many projects concerned with encryption and data compression using FPGA , in our university there are many projects about encryption in both hardware and software domains.

Projects in our university:

1. Hardware Encrypto Unit

This project involves implementing an encryption algorithm called RSA on a programmable logic device (FPGA/CPLD), writing software to interface it with pc serial port and developing a simple application which will utilize the chip for sending encrypted data over the internet.[1]

2. PCI Encryption Card

This project involves encrypting data using RSA algorithm on FPGA and interface the FPGA with PCI card in the personal computer and utilize sending encrypted data between computers. [2]

External projects:

There are few projects concerning implementation of both encryption and compression using FPGA the following project is the most similar to our project.

SCAN-Based Compression-Encryption-Hiding for Video on demand

They present a scan-based method for image and video compression-encryption hiding with application to digital video-on-demand. The software SCAN implementation running on a Pentium IV takes about one second for 25 video frames. As an alternative solution, however, they have developed a FPGA-based architecture, which operates in real-time. [3]

1.3 Estimated cost

The following are the costs of the software components which we used for the project.

Component	Cost
Xilinx web PACK 6.0a ModelSimIII XE Starter	$300
ISE 7.1 software package	$500

Table 1.1 Estimated cost

1.4 Time plan

The time scheduling for the project is described as following:

Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Hardware design																
Software design																
implementation																
testing																
Documentation																

Table 1.2 Time plan

1.5 Report contents

Here is a brief description of the contents of the remaining chapters:

Chapter two is about the theoretical background of the project; containing theoretical subjects related to the main ideas of the project and information about components that we used in our project.

Chapter three is about the project's design concepts including project objectives, general block diagram, and how this system works.

Chapter four is about hardware which includes discussing design options and justifying those chosen for the project and detailed description of project parts, also it describes the function of system components.

Chapter five is the software system design; it contains the general algorithms and flowcharts.

Chapter six is implementation and testing, it includes the hardware and software implementation of the system including simulation.

Chapter seven is conclusion and future work.

Chapter Two

Theoretical Background

2.1 Background

2.2 Data Encryption

2.3 Data compression

2.4 Field Programmable Gate Array (FPFA)

2.5 Verilog Language

2.6 Microweb Server

2.1 Background

During the last century the industry of protecting information and increasing the speed of transmitting information has grown incredibly with the need to protect and increasing the speed of transmitting data

Several algorithms are used for data encryption and compression, so in our project we chose AES algorithm and Deffie-Hellman for encryption and LZW algorithm for compression.

2.2 Data Encryption

2.2.1 What is encryption?

Encryption is the process of obscuring information to make it unreadable without special knowledge. This is usually done for secrecy, and typically for confidential communications. Encryption can also be used for authentication. Even when encrypted, messages can still be subject to traffic analysis although this cannot typically be used to reveal the actual contents of the message.

2.2.2 How Encryption Works

Encryption is done using cryptography - the study of sending 'messages' in a secret form so that only those authorized to receive the 'message' are able to read it. The easy part of encryption is applying a mathematical function to the plaintext and converting it to an encrypted cipher. The harder part is to ensure that the people who are supposed to decipher this message can do so with ease, yet only those authorized are able to decipher it. The mathematical function should be sufficiently complex and mathematically sound to give us a high degree of safety. [4]

2.2.3 Cipher

A cipher (sometimes spelt cypher) is an algorithm for performing encryption (and the reverse, decryption) — a series of well-defined steps that can be followed as a procedure. An alternative term is encipherment. The original information is known as plaintext, and the encrypted form as cipher text. The cipher text message contains all the information of the plaintext message, but is not in a format readable by a human or computer without the proper mechanism to decrypt it; it should resemble random gibberish to those not intended to read it.

Ciphers are usually parameterized by a piece of auxiliary information, called a key. The encrypting procedure is varied depending on the key which changes the detailed operation of the algorithm. Without the key, the cipher cannot be used to encrypt, or more importantly, to decrypt.

2.2.4 Types of cipher

There are a variety of different types of encryption. Algorithms used earlier in the history of cryptography are different to modern methods, and modern ciphers can be classified according to how they operate and whether they use one or two keys.

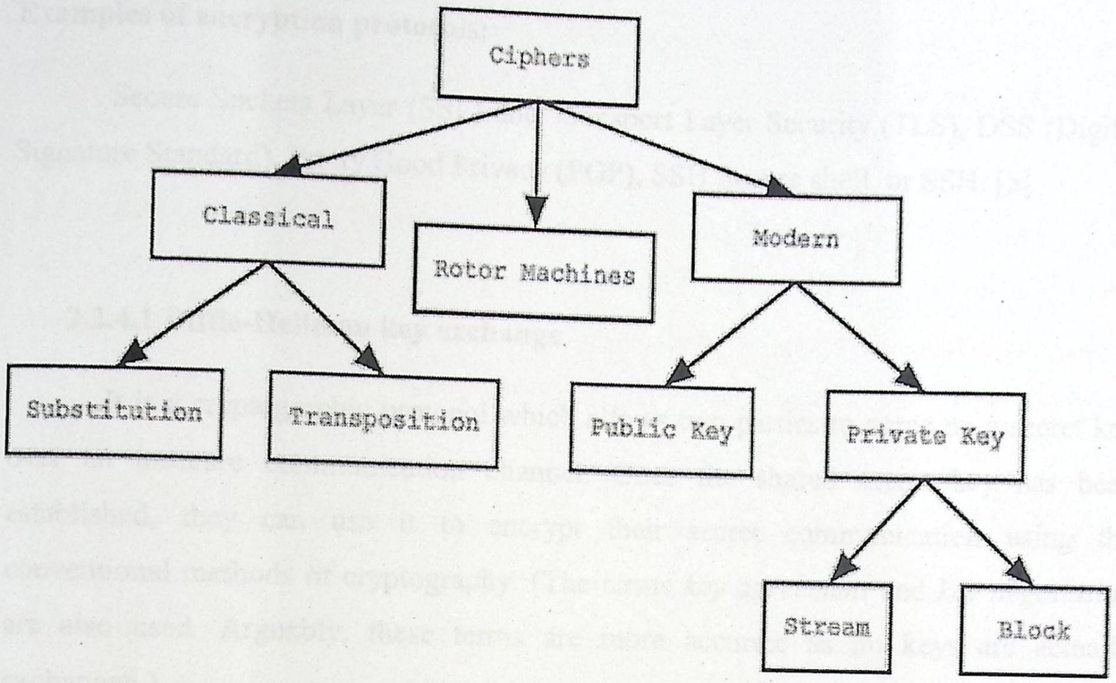
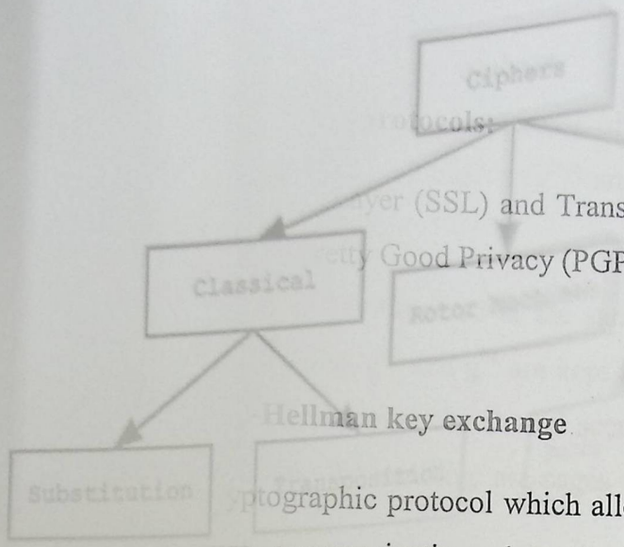


Figure 2.1 Types of ciphers

Historical pen and paper ciphers used in the past are sometimes known as classical ciphers. They include substitution ciphers and transposition ciphers. During the early 1900s, more sophisticated machines for encryption were used, rotor machines, which were more complex than previous schemes.

Encryption methods can be divided into symmetric key algorithms and asymmetric key algorithms. In a symmetric key algorithm (e.g., DES and AES), the sender and receiver must have a shared key set up in advance and kept secret from all other parties; the sender uses this key for encryption, and the receiver uses the same key for decryption. In an asymmetric key algorithm (e.g., RS, AElGamal, Elliptic curve cryptography, Paillier cryptosystem), there are two separate keys: a public key is published and enables any sender to perform encryption, while a private key is kept secret by the receiver and enables him to perform decryption. Common asymmetric encryption algorithms available today are all based on the Diffie-Hellman public key agreement algorithm.

Symmetric key ciphers can be distinguished into two types, depending on whether they work on blocks of symbols usually of a fixed size (block ciphers), or on a continuous stream of symbols (stream ciphers).



yer (SSL) and Transport Layer Security (TLS), DSS (Digital
ty Good Privacy (PGP), SSH Secure shell, or SSH. [5]

Hellman key exchange

ptographic protocol which allows two parties to agree on a secret key
re communication channel. Once the shared secret key has been
hey can use it to encrypt their secret communication using the
methods of cryptography. (The terms *key agreement* and *key negotiation*
ed. Arguably, these terms are more accurate as no keys are actually

iffie-Hellman key agreement is used, in conjunction with several alternative
ication methods, in the IKE component of the IPsec protocol suite, for securing
t Protocol communications.

cription:

The simplest, and original, implementation of the protocol uses the ultiplicative group of integers modulo p, where p is a prime. That simply means that he integers between 1 and p − 1 are used with normal multiplication, exponentiation and division, except that after each operation the result keeps only the remainder after dividing by p. Here is an example of the protocol:

1. Both sides agree on a finite cyclic group G and a generating element g in G. (This is usually done long before the rest of the protocol; g is assumed to be known by all attackers.

2. Side A picks a random natural number a and sends g^a to side A.

3. Side B side picks a random natural number b and sends g^b to side B.

4. A computes $(g^b)^a \bmod p$.

5. B computes $(g^a)^b \bmod p$.

Examples of encryption protocols:

Secure Sockets Layer (SSL) and Transport Layer Security (TLS), DSS (Digital Signature Standard), Pretty Good Privacy (PGP), SSH Secure shell, or SSH. [5]

2.2.4.1 Diffie-Hellman key exchange

It is a cryptographic protocol which allows two parties to agree on a secret key over an insecure communication channel. Once the shared secret key has been established, they can use it to encrypt their secret communication using the conventional methods of cryptography. (The terms *key agreement* and *key negotiation* are also used. Arguably, these terms are more accurate as no keys are actually exchanged.)

Diffie-Hellman key agreement is used, in conjunction with several alternative authentication methods, in the IKE component of the IPsec protocol suite, for securing Internet Protocol communications.

Description:

The simplest, and original, implementation of the protocol uses the multiplicative group of integers modulo p, where p is a prime. That simply means that the integers between 1 and p − 1 are used with normal multiplication, exponentiation and division, except that after each operation the result keeps only the remainder after dividing by p. Here is an example of the protocol:

1. Both sides agree on a finite cyclic group G and a generating element g in G. (This is usually done long before the rest of the protocol; g is assumed to be known by all attackers.

2. Side A picks a random natural number a and sends g^a to side A.

3. Side B side picks a random natural number b and sends g^b to side B.

4. A computes $(g^b)^a \bmod p$.

5. B computes $(g^a)^b \bmod p$.

Both A and B are now in possession of the group element g^{ab} which can serve as the shared secret key. The values of $(g^b)^a$ and $(g^a)^b$ are the same because groups are power associative. [6][7]

Both A and B have arrived at the same value, this is because g^{ab} and g^{ba} are equal. Note that only a, b, g^{ab} and g^{ba} are kept secret. All the other values are sent in the clear. Once A and B compute the shared secret they can use it as an encryption key, known only to them, for sending messages across the same open communications channel.

Of course, much larger values of a,b, and p would be needed to make this example secure, since it's easy to try all the possible values of g^{ab} mod 23. If p was a prime of more than 300 digits, and a and b were at least 100 digits long, then even the best known algorithms for finding a given only g, p, and g^a mod p (known as the discrete logarithm problem) would take longer than the lifetime of the universe to run.

2.2.4.2 Advanced Encryption Standard (AES)

It is a block cipher adopted as an encryption standard by the US government, and is expected to be used worldwide and analyzed extensively, as was the case with its predecessor, the Data Encryption Standard (DES). It was adopted by National Institute of Standards and Technology (NIST) as US FIPS PUB 197 in November 2001 after a 5-year standardization process.

The cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted to the AES selection process under the name "Rijndael", a portmanteau comprised of the names of the inventors.

Description of the cipher

Strictly speaking, AES is not precisely Rijndael (although in practice they are used interchangeably) as Rijndael supports a larger range of block and key sizes; AES has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits, whereas Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits.

AES operates on a 4×4 array of bytes, termed the *state* (versions of Rijndael with a larger block size have additional columns in the state). For encryption, each round of AES (except the last round) consists of four stages:

1. SubBytes — a non-linear substitution step where each byte is replaced with another according to a lookup table.

2. ShiftRows — a transposition step where each row of the state is shifted cyclically a certain number of steps.

3. MixColumns — a mixing operation which operates on the columns of the state, combining the four bytes in each column using a linear transformation.

4. AddRoundKey — each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule.

The final round omits the MixColumns stage.

For more details on how AES works see appendix D

Security

As of 2004, no successful attacks against AES have been recognised. The National Security Agency (NSA) reviewed all the AES finalists, including Rijndael, and stated that all of them were secure enough for US Government non-classified data. In June 2003, the US Government announced that AES may be used for classified information.

This marks the first time that the public has had access to a cipher approved by NSA for TOP SECRET information.

The most common way to attack block ciphers is to try various attacks on versions of the cipher with a reduced number of rounds. AES has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. As of 2004, the best known attacks are on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys. [8][9]

2.3 Data compression

2.3.1 What is data compression?

In computer science, data compression is the process of encoding information using fewer bits, or information units, thanks to specific encoding schemes.

Compression only works when both the sender and receiver of the information message have agreed on the encoding scheme.

A popular encoding scheme used on almost all modern operating systems is the ZIP. It can be used to reduce the size of an attachment to an e-mail, facilitating its transmission. However, both the sender and receiver must be aware of this format, and must use the appropriate encoding / decoding program.

Compression is possible because most real-world data is very redundant, or represented in its human-interpretable form in a not concise way. Compression is important because it helps reduce the consumption of expensive resources, such as disk space or connection bandwidth. However, compression requires information processing power, which can also be expensive. Therefore, many data compression schemes have been designed for various purposes. Some schemes are reversible so that the original data can be reconstructed (lossless compression), while others accept some loss of data in order to achieve higher compression (lossy compression). [10][11]

2.3.2 Compression algorithms

2.3.2.1 Lossless data compression

Lossless data compression is a class of data compression algorithms that allow the original data to be reconstructed *exactly* compressed data. Contrast with lossy data compression.

Lossless data compression is used in software compression tools such as the highly popular Zip format, used by PKZIP, and Mac OS 10.3, and the Unix programs bzip2, gzip, compress. Other popular formats include Stuffet and RAR.

Lossless compression is used when it is important that the original and the decompressed data are exactly identical, or when no assumption can be made on whether certain deviation is uncritical. Typical examples are executable programs and source code. Some image file formats, notably PNG, use only lossles compression, while others like TIFF and MNG may use either lossless or lossy methods.[12]

2.3.2.1.1 Lossless compression Techniques:

Lossless compression methods may be categorized according to the type of data they are designed to compress. The three main types of targets for compression algorithms are text, images, and sound.

Most lossless compression programs use two different kinds of algorithm: One which generates a *statistical model* for the input data, and another which maps the input data to bit strings using this model in such a way that "probable" (e.g. frequently encountered) data will produce shorter output than "improbable" data. Often, only the former algorithm is named, while the second is implied (through common use, standardization etc.) or unspecified.

Statistical modeling algorithms for text (or text-like binary data such as executables) include:

1-Brrows-Wheeler Transform:

(BWT, also called block-sorting compression), is an algorithm used in data compression techniques such as bzip2. It was invented by Michael Burrows and David Wheeler.[13]

2-DEFLATE:

It is a lossless data compression algorithm that uses a combination of the LZ77 algorithm and Huffman coding to take advantage of the different frequencies of occurrence of byte sequences in the file. [14]

3-LZW:

The original Lempel Ziv approach to data compression was first published in in1977. TerryWelch's refinements to the algorithm were published in 1984. The algorithm is surprisingly simple. In a nutshell, LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.

The code that the LZW algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds. [15][16]

LZW compression can be used in a variety formats:

- TIFF files
- GIF files

To Compress

1. Initialize the table to contain all single-byte strings
2. Read the first byte, c, from the input file. Set the prefix, w, to that byte.
3. Read the next input byte into c.
4. If at end of file then go to step 7.
5. If «w»c is in the string table then set «w» to «w»c and go to 3.
6. Output Code («w»); Put «w»c in the string table; Set «w» to c; go to 3
7. Output Code («w») and signify completion

To Decompress

The de-compressor incrementally reconstructs the same string table that the compressor used. Unfortunately, there is an abnormal case for which this algorithm does not work. The abnormal case occurs whenever an input character string containing a cyclic sequence of the form byte-code-byte-code-byte already appears in the compressor string table. The decompression algorithm is modified (see step 3 of the decompression description, below) to handle the special case.

1. Read the first input code and store in both code and oldcode. Since the first code is known to be atomic (cf. 2, above), code is the encoding of c, output c, and set the extension to c.

2. Read the next code and save the value in incode. If at end of file then go to 8.

3. If the code is not in the string table, we have the special case so, output extension, set code to oldcode, and set incode to the coding of «w»c

4. If the retrieved code is the code for «w»c then push c onto the stack, set code to code of «w» and repeat this step.

5. If the code is atomic, output the byte it represents and set the extension to that byte.

6. While the stack is non-empty, output the byte on top and pop the stack.

7. Put «oldcode»c into the string table, set oldcode to incode and go to 2.

8. Signify completion.

Encoding algorithms:

These algorithms are used optionally to produce bit sequences to be compressed

1- Huffman coding: in computer science this code is an entropy encoding algorithm used for data compression that finds the optimal system of encoding strings based on the relative frequency of each character. [17]

2- Arithmetic coding: is a method for lossless data compression. It is a form of entropy encoding , but where other entropy encoding techniques separate the input message into its component symbols and replace each symbol with a code word, arithmetic coding encodes the entire message into a single number, a fraction n where ($0.0 \leq n < 1.0$). [18]

2.3.2.2 Lossy data compression

A *lossy* data compression method is one where compressing a file and decompressing it retrieves a file that may well be different to the original, but is "close enough" to be useful in some way. This type of compression is used a lot on the Internet and especially in streaming media and telephony applications.

The advantage of lossy methods over lossless methods is that in some cases a lossy method can produce a much smaller compressed file than any known lossless method, while still meeting the requirements of the application.

Lossy methods are most often used for compressing sound or images. In these cases, the retrieved file can be quite different to the original at the bit level while being indistinguishable to the human ear or eye for most practical purposes. Many methods focus on the idiosyncrasies of the human anatomy, taking intoaccount, for example,

Lossy compression methods:

A- Direct cosine transform.

B- Fractal compression.

C- Wavelet compression. [19]

2.4 Field Programmable Gate Array (FPFA)

2.4.1 What are FPGA and CPLD?

In 60's systems were built from lots of individual chips with a spaghetti-like maze of wiring between them. It was difficult to modify such a system after you built it. After a week or two it was difficult to remember what each of the chips was for.

Manufacturing such a system took a lot of time because each design change required that the wiring be redone which usually meant building a new printed circuit board. The chip makers solved this problem by placing an unconnected array of AND-OR gates in a single chip called a programmable logic device (PLD).

The PLD contained an array of fuses that could be blown open or left closed to connect various inputs to each AND gate. You could program a PLD with a set of Boolean sum-of-product equations so it would perform the logic functions you needed in your system. Since the PLDs could be rewired internally, there was less of a need to change the printed circuit boards which held them.

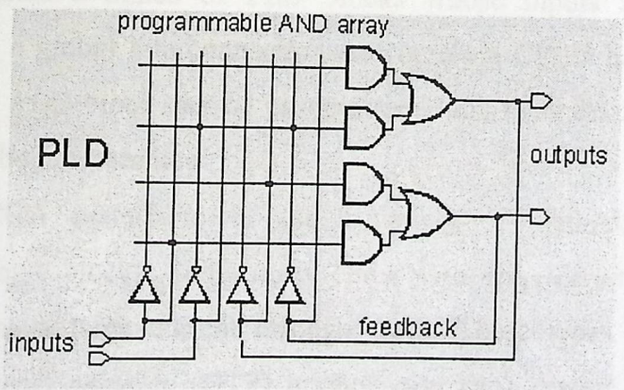


Figure2.2 1 PLD Architecture

Simple PLDs could only handle up to 10–20 logic equations, so you couldn't fit a very large logic design into just one of them. You had to figure out how to break your larger designs apart and fit them into a set of PLDs. This was time-consuming and meant you had to interconnect the PLDs with wires. Wiring was a big problem because

eventually you would make some design change that couldn't be handled just by reprogramming the PLDs and then you would have to build a new circuit board.

The chip makers came to the rescue again by building much larger programmable chips called *complex programmable logic devices* (CPLDs) and *field-programmable gate arrays* (FPGAs). With these, you could essentially get a complete system onto a single chip.

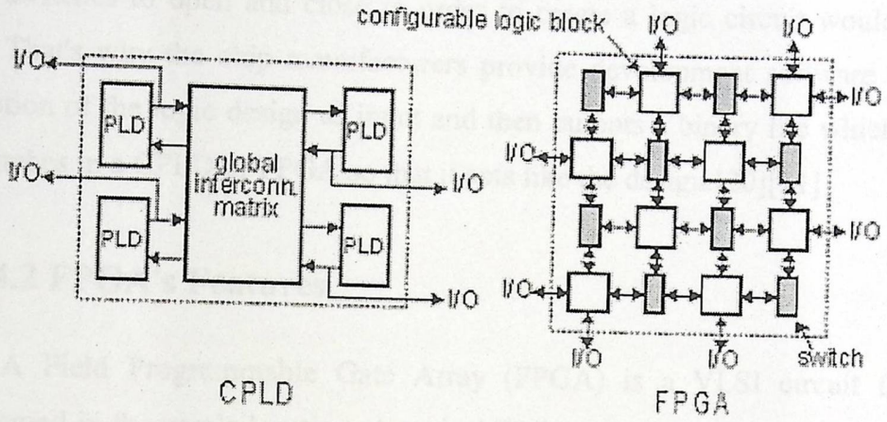


Figure 2.3 CPLD and FPGA Architecture

A CPLD contains a bunch of PLD blocks whose inputs and outputs are connected together by a global interconnection matrix. So a CPLD has two levels of programmability: each PLD block can be programmed, and then the interconnections between the PLDs can be programmed

CPLD and FPGA manufacturers use a variety of methods to make the connections between logic blocks. Some make chips with *fuses* or *anti-fuses* that are programmed by passing a large current through them. These types of CPLDs and FPGAs are *one-time programmable* (OTP) because you can't rewire them internally once the fuses are blown.

Other manufacturers make the connections using pass transistors that are opened or closed by storing a charge on their gate electrodes using a high-voltage pulse. This type of programmable device resembles an EPROM or EEPROM: you can erase it and then place it in a special programmer socket and reprogram it. That's fine unless you have the CPLD or FPGA soldered into a circuit board.

Finally, some manufacturers use static RAM or Flash bits to control the pass transistors for each interconnection. By loading each bit with a 1 or a 0, you can control whether the switch is closed or opened and, therefore, whether two logic elements are connected or not. CPLDs and FPGAs built using RAM/Flash switches can be reprogrammed without removing them from the circuit board. They are often said to be *in-circuit reconfigurable* or *in-circuit programmable.*

Regardless of the interconnection method used, you can see that figuring out which switches to open and close in order to create a logic circuit would be quite a chore. That's why the chip manufacturers provide development software that takes a description of the logic design as input and then outputs a binary file which configures the switches in a CPLD or FPGA so that it acts like the design.[20][21]

2.4.2 FPGA's Features

A Field Programmable Gate Array (FPGA) is a VLSI circuit that can be programmed in the user's location. A typical FPGA consist of any array of hundreds or thousands of logic blocks, surrounded by programmable input and output blocks and connected together via programmable interconnections. There is a wide variety of internal configurations within this group of devices. The performance of each device type depends on the circuit contained in their logic blocks and the efficiency of their programmed interconnection.

An FPGA consists of an array of configurable logic blocks that implement the logical functions of gates. Logic gates are like switches with multiple inputs and 1 output that perform the basic logical operations (AND, OR, XOR, NAND, NOR). In FPGA's, the logic functions performed within the logic blocks, and sending signals to the chip can alter the connections between the blocks.

FPGA designs are not all the same. The size and capacitate performance of the logic blocks that make up the FPGA vary between manufacture. For an FPGA that consists of blocks that are large and powerful, it is said to have a "coarse grained" design. Conversely, a chip consisting of small but simple blocks is said to be "fine grained".

In other word a field-programmable gate array or FPGA is a gate array that can reprogrammed after it is manufactured, rather than during the manufacturing — a programmable logic device. Device manufacturers include Xilinx, Altera, Lattice Semiconductor, Actel, Cypress, Atmel and QuickLogic.

Many modern FPGAs have the ability to be reprogrammed at 'run time', and this is leading to the idea of reconfigurable computing or reconfigurable systems - CPUs that reconfigure themselves to suit the task at hand.

Applications of FPGAs include DSP, Software-defined radio, Aerospace and defense systems, ASIC Prototyping, Medical imaging and a growing range of other areas.

To define the behavior of the FPGA it is required to use a Hardware Description Language (HDL) or a schematic designed using an Electronic design automation tool. Either of these, when compiled, will generate a net list, that can be mapped to the actual FPGA architecture. When done the binary file generated is used to (re)configure the FPGA device. Common HDL's are VHDL and Verilog.

A typical FPGA logic block consists of look up tables, multiplexers, gates, and flip-flops. The look up table is a truth table stored in SRAM and provides the combinational circuit functions for the logic block. These functions are realized from the truth table stored in the SRAM, similar to the manner that combinational circuit functions are implemented with ROM. For example a 16x 2SRAM can store the truth table of combinational circuit that has four inputs and two outputs. The combinational logic section along with number of programmable multiplexers is used to configure the input equation for the flip flop and the out of logic block. [22]

2.4.3 What are the differences between FPGA and CPLD?

1- FPGAs are "fine-grain" devices. That means that they contain a lot (up to 100000) of tiny blocks of logic with flip-flops. CPLDs are "coarse-grain" devices. They contain relatively few (a few 100's max) large blocks of logic with flip-flops.

2- FPGAs are RAM based. They need to be "downloaded" (configured) at each power-up. CPLDs are EEPROM based. They are active at power-up (i.e. as long as they've been programmed at least once...).

3- CPLDs have a faster input-to-output timings than FPGAs (because of their coarse-grain architecture, one block of logic can hold a big equation), so are better suited for microprocessor decoding logic for example than FPGAs.

4- FPGAs have special routing resources to implement efficiently binary counters and arithmetic functions (adders, comparators...). CPLDs do not.

5- FPGAs can contain very large digital designs, while CPLDs can contain small designs only.[23]

2.4.4 What are the differences between FPGA and ASIC?

1- The blocks that are used in FPGA are similar in structure to the gate arrays used in some ASIC's, but whereas standard gate arrays are configured and fixed during manufacture, the configurable logic blocks in new FPGA's can be rewired and reprogrammed repeatedly in around a microsecond.

2- FPGA is generally slower than their ASIC counterparts, and draws more power. However, they have several advantages such as a shorter time-to-market, and lower development costs (for quantities < 10k).

3- An ASIC can be made that is a so-called hard copy of an FPGA - that is, an integrated circuit with the same functionality as the FPGA, but faster and consuming less power. [24]

2.4.5 How to implement a logic design with FPGA

Implementing a logic design with the FPGA or CPLD development software usually consists of the following steps (depicted in the figure below):

1- Enter a description of your logic circuit using a hardware description language (HDL) such as verilog. You can also draw your design using a schematic editor.

2- Use a logic synthesizer program to transform the HDL or schematic into a netlist. The netlist is just a description of the various logic gates in your design and how they are interconnected.

3- Use the implementation tools to map the logic gates and interconnections into the FPGA.the configurable logic blocks (CLBs) in the FPGA can be further decomposed into look-up tables (LUTs) that perform logic operations. The CLBs and LUTs are interwoven with various routing resources. The mapping tool collects your netlist gates into groups that fit into the LUTs and then the place & route tool assign the gate collections to specific CLBs while opening or closing the switches in the routing matrices to connect the gates together.

4- Once the implementation phase is complete, a program extracts the state of the switches in the routing matrices and generates a bit stream where the ones and zeroes correspond to open or closed switches.

5- The bitstream is downloaded into a physical FPGA chip. The electronic switches in the FPGA are open or close in response to the binary bits in the bitstream. Upon completion of the downloading, the FPGA will perform the operations specified by your Hardware Design Language (HDL) code or schematic. You can apply input signals to the I/O pins of the FPGA to check the operation of your design.[25]

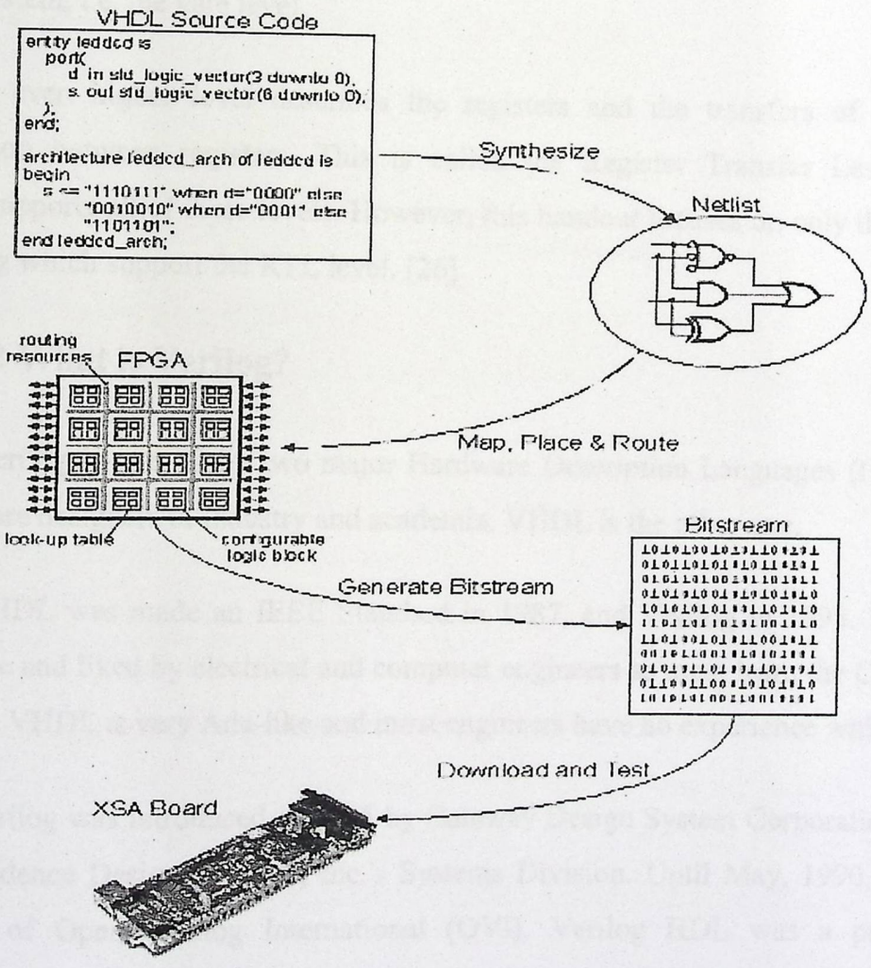


Figure 2.4 FPGA Programming

2.5 Verilog Language

2.5.1. Introduction

Verilog HDL is a Hardware Description Language (HDL). A Hardware Description Language is a language used to describe a digital system, for example, a computer or a component of a computer.

One may describe a digital system at several levels. For example, an HDL might describe the layout of the wires, resistors and transistors on an Integrated Circuit (IC) chip, i.e. the switch level. Or, it might describe the logical gates and flip flops in a digital system, i.e. the gate level.

An even higher level describes the registers and the transfers of vectors of information between registers. This is called the Register Transfer Level (RTL). Verilog supports all of these levels. However, this handout focuses on only the portions of Verilog which support the RTL level. [26]

2.5.2 What is Verilog?

Verilog is one of the two major Hardware Description Languages (HDL) used by hardware designers in industry and academia. VHDL is the other one.

VHDL was made an IEEE Standard in 1987, and Verilog in 1995. Verilog is very C-like and liked by electrical and computer engineers as most learn the C language in college. VHDL is very Ada-like and most engineers have no experience with Ada.

Verilog was introduced in 1985 by Gateway Design System Corporation, now a part of Cadence Design Systems, Inc.'s Systems Division. Until May, 1990, with the formation of Open Verilog International (OVI), Verilog HDL was a proprietary language of Cadence.

Cadence was motivated to open the language to the Public Domain with the expectation that the market for Verilog HDL-related software products would grow

2.5 Verilog Language

2.5.1. Introduction

Verilog HDL is a Hardware Description Language (HDL). A Hardware Description Language is a language used to describe a digital system, for example, a computer or a component of a computer.

One may describe a digital system at several levels. For example, an HDL might describe the layout of the wires, resistors and transistors on an Integrated Circuit (IC) chip, i.e. the switch level. Or, it might describe the logical gates and flip flops in a digital system, i.e. the gate level.

An even higher level describes the registers and the transfers of vectors of information between registers. This is called the Register Transfer Level (RTL). Verilog supports all of these levels. However, this handout focuses on only the portions of Verilog which support the RTL level. [26]

2.5.2 What is Verilog?

Verilog is one of the two major Hardware Description Languages (HDL) used by hardware designers in industry and academia. VHDL is the other one.

VHDL was made an IEEE Standard in 1987, and Verilog in 1995. Verilog is very C-like and liked by electrical and computer engineers as most learn the C language in college. VHDL is very Ada-like and most engineers have no experience with Ada.

Verilog was introduced in 1985 by Gateway Design System Corporation, now a part of Cadence Design Systems, Inc.'s Systems Division. Until May, 1990, with the formation of Open Verilog International (OVI), Verilog HDL was a proprietary language of Cadence.

Cadence was motivated to open the language to the Public Domain with the expectation that the market for Verilog HDL-related software products would grow

more rapidly with broader acceptance of the language. Cadence realized that Verilog HDL users wanted other software and service companies to embrace the language and develop Verilog-supported design tools.

Verilog HDL allows a hardware designer to describe designs at a high level of abstraction such as at the architectural or behavioral level as well as the lower implementation levels (i. e. , gate and switch levels) leading to Very Large Scale Integration (VLSI) Integrated Circuits (IC) layouts and chip fabrication.

A primary use of HDLs is the simulation of designs before the designer must commit to fabrication. This handout does not cover all of Verilog HDL but focuses on the use of Verilog HDL at the architectural or behavioral levels. The handout emphasizes design at the Register Transfer Level (RTL).[27]

2.6 Microweb Server

The microweb server (IPμ8930) is a TCP controller module that contains all you need to add intranet/Internet-based monitoring and control of data and applications; it can also be used as a serial-to-Ethernet bridge.

The system is to be connected with the microweb server serially so we will talk about the serial port of microweb server.

2.6.1 Micro web server Serial Port

The IPμ8930 has a standard UART serial port, which used to communicate with or control a variety of external serial devices. The signal levels on the RX, TX, RTS, and CTS pins of the 8930 module are 5V TTL levels, not true RS-232 levels. To connect the 8930 to a PC or other standard RS-232 device, a level-shifting device must be used.

The IPμ8930 Developer Board has this chip (e.g., MAX232A) built in, and routed to the RJ-45 serial connector.

more rapidly with broader acceptance of the language. Cadence realized that Verilog HDL users wanted other software and service companies to embrace the language and develop Verilog-supported design tools.

Verilog HDL allows a hardware designer to describe designs at a high level of abstraction such as at the architectural or behavioral level as well as the lower implementation levels (i. e. , gate and switch levels) leading to Very Large Scale Integration (VLSI) Integrated Circuits (IC) layouts and chip fabrication.

A primary use of HDLs is the simulation of designs before the designer must commit to fabrication. This handout does not cover all of Verilog HDL but focuses on the use of Verilog HDL at the architectural or behavioral levels. The handout emphasizes design at the Register Transfer Level (RTL).[27]

2.6 Microweb Server

The microweb server (IPμ8930) is a TCP controller module that contains all you need to add intranet/Internet-based monitoring and control of data and applications; it can also be used as a serial-to-Ethernet bridge.

The system is to be connected with the microweb server serially so we will talk about the serial port of microweb server.

2.6.1 Micro web server Serial Port

The IPμ8930 has a standard UART serial port, which used to communicate with or control a variety of external serial devices. The signal levels on the RX, TX, RTS, and CTS pins of the 8930 module are 5V TTL levels, not true RS-232 levels. To connect the 8930 to a PC or other standard RS-232 device, a level-shifting device must be used.

The IPμ8930 Developer Board has this chip (e.g., MAX232A) built in, and routed to the RJ-45 serial connector.

more rapidly with broader acceptance of the language. Cadence realized that Verilog HDL users wanted other software and service companies to embrace the language and develop Verilog-supported design tools.

Verilog HDL allows a hardware designer to describe designs at a high level of abstraction such as at the architectural or behavioral level as well as the lower implementation levels (i. e. , gate and switch levels) leading to Very Large Scale Integration (VLSI) Integrated Circuits (IC) layouts and chip fabrication.

A primary use of HDLs is the simulation of designs before the designer must commit to fabrication. This handout does not cover all of Verilog HDL but focuses on the use of Verilog HDL at the architectural or behavioral levels. The handout emphasizes design at the Register Transfer Level (RTL).[27]

2.6 Microweb Server

The microweb server (IPμ8930) is a TCP controller module that contains all you need to add intranet/Internet-based monitoring and control of data and applications; it can also be used as a serial-to-Ethernet bridge.

The system is to be connected with the microweb server serially so we will talk about the serial port of microweb server.

2.6.1 Micro web server Serial Port

The IPμ8930 has a standard UART serial port, which used to communicate with or control a variety of external serial devices. The signal levels on the RX, TX, RTS, and CTS pins of the 8930 module are 5V TTL levels, not true RS-232 levels. To connect the 8930 to a PC or other standard RS-232 device, a level-shifting device must be used.

The IPμ8930 Developer Board has this chip (e.g., MAX232A) built in, and routed to the RJ-45 serial connector.

The baud rate (SERBAUD) and flow control type (SERCTRL) are user-settable and located in the internal EEPROM .The data bits, parity, and stop bits are fixed at 8, N, and 1.

The IPμ8930 serial port can operate in one of two modes: Serial Tunneling mode and Peripheral mode..[28]

Chapter Three

Design Concepts

3.1 Project Objectives

3.2 General Block Diagram

3.3 How System Works

3.1 Project Objectives

Our project has the following purposes:

1. Studying encryption and compression algorithms.
2. Coding encryption and compression algorithms.
3. Designing a serial interface with microweb server.
4. Simulating the design using FPGA software package.

3.2 General Block Diagram

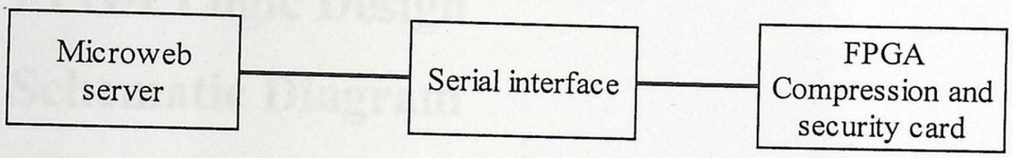


Figure 3.1 System General Block diagram

3.3 How System Works

The FPGA compression and security card receives original data from the microweb server through the serial interface, and then performs data encryption and compression to have secure data which will be transmitted to the microweb server.

On the other hand the card receives secured data from the microweb server through the serial interface, and then performs data decompression and decryption to get the original data which will be transmitted to the microweb server

Chapter Four

Hardware System Design

4.1 Design Options

4.2 FPGA Logic Design

4.3 Schematic Diagram

4.1 Design Options

Because the system is very huge and needs a large number of logic cells (nearly over 40000 logic cells) we decided to use FPGA chip (FG680–XCV2000E) rather than PLDs which have a limitation in terms of logic cells that will not be able to fit the algorithm.

Also we will not use ASIC as it not reconfigurable; it is programmed only once and can't be programmed again.

The Lempel Ziv Welch (lzw) algorithm is used for compression/decompression core and advanced encryption standard (AES) is used for encryption/decryption algorithm.

4.1.1 Why we choose FPGA?

The FPGA chip has the following advantages:

- FPGA device technology setting is maintained across the design flow.

- Provides the availability of building the whole digital system on single device.

- Accelerates design creation and implementation.

- Easy to learn and use.

- Low cost of ownership. [23]

4.2 FPGA Logic Design

After choosing the suitable encryption and compression algorithms, the design was targeted to the hardware.

This section shows system core and the functionality of each core.

4.2.1 System Block diagram

This system consists of three main units:

- Encryption/compression unit
- Serialization/desrialization
- Serial port interface

Figure 4.1 shows these cores which will be discussed in details in the following sections.

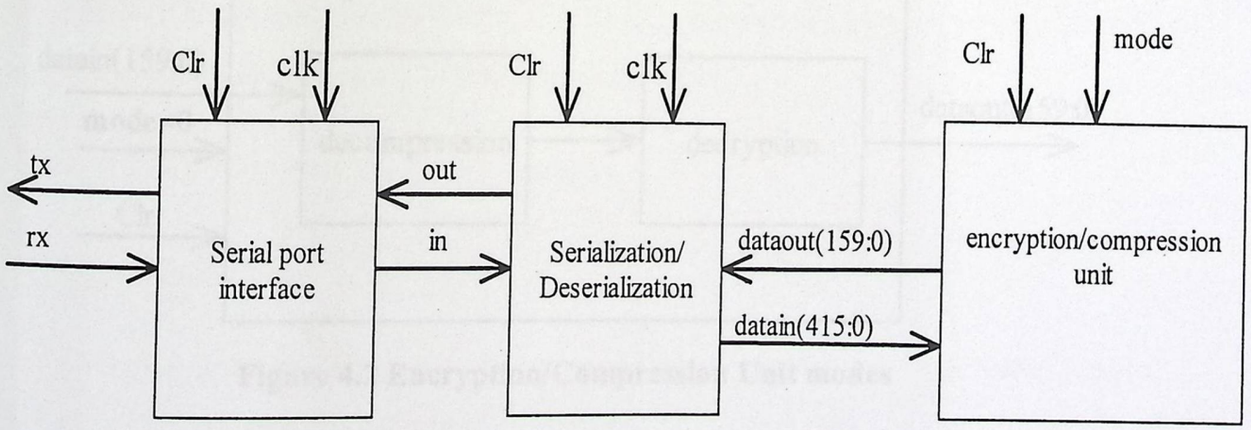


Figure 4.1 System Block Diagram

4.2.2 Encryption/Compression Unit (ECU)

The ECU is the main unit in the system, it works on two modes; the first mode is encryption/compression and the second mode is decompression/decryption.

The two modes are illustrated in figure 4.2.

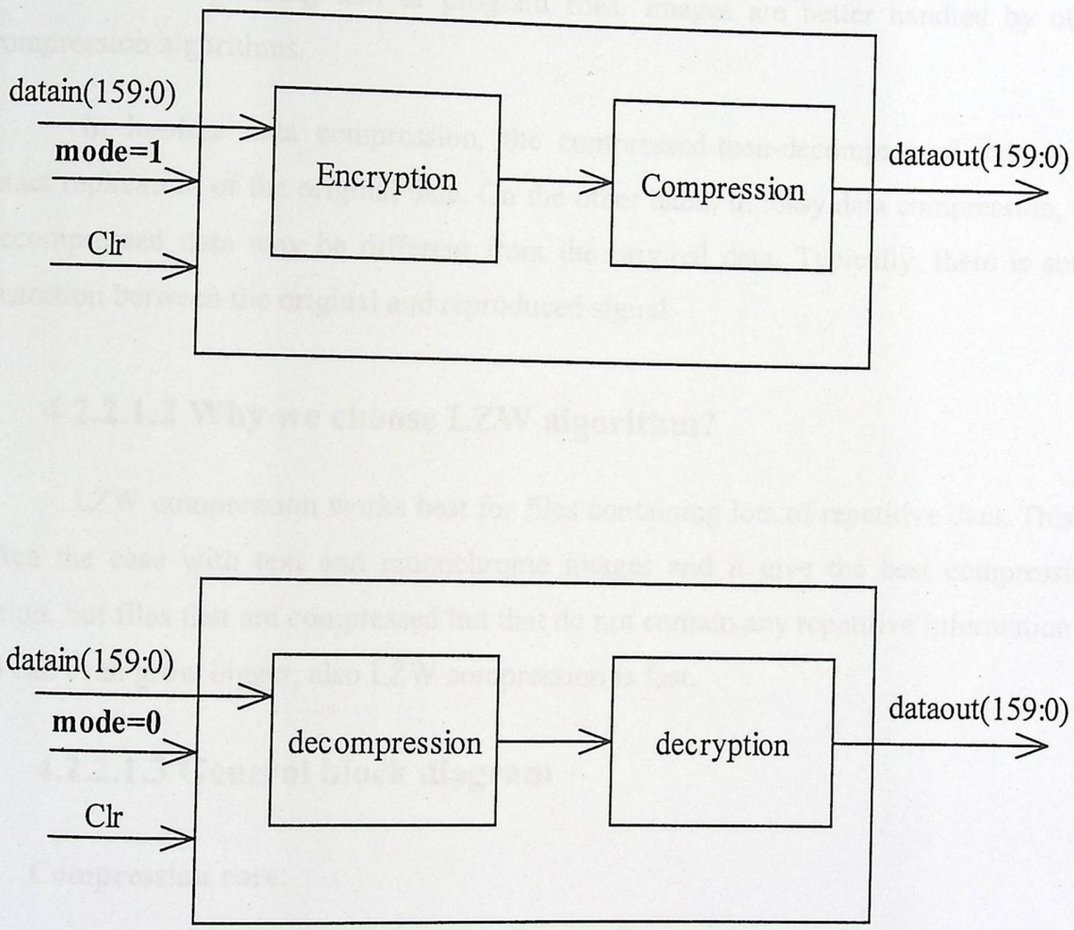


Figure 4.2 Encryption/Compression Unit modes

4.2.2 Encryption/Compression Unit (ECU)

The ECU is the main unit in the system, it works on two modes; the first mode is encryption/compression and the second mode is decompression/decryption.

The two modes are illustrated in figure 4.2.

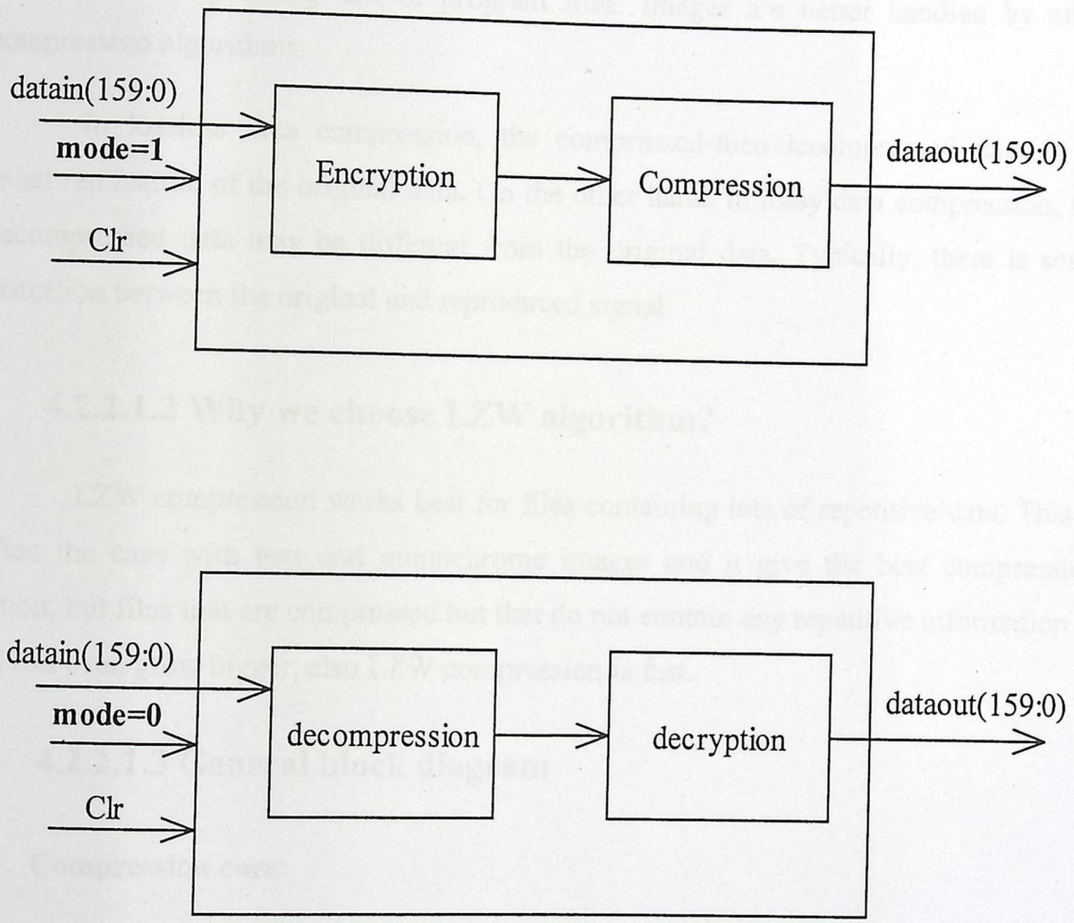


Figure 4.2 Encryption/Compression Unit modes

4.2.2 Encryption/Compression Unit (ECU)

The ECU is the main unit in the system, it works on two modes; the first mode is encryption/compression and the second mode is decompression/decryption.

The two modes are illustrated in figure 4.2.

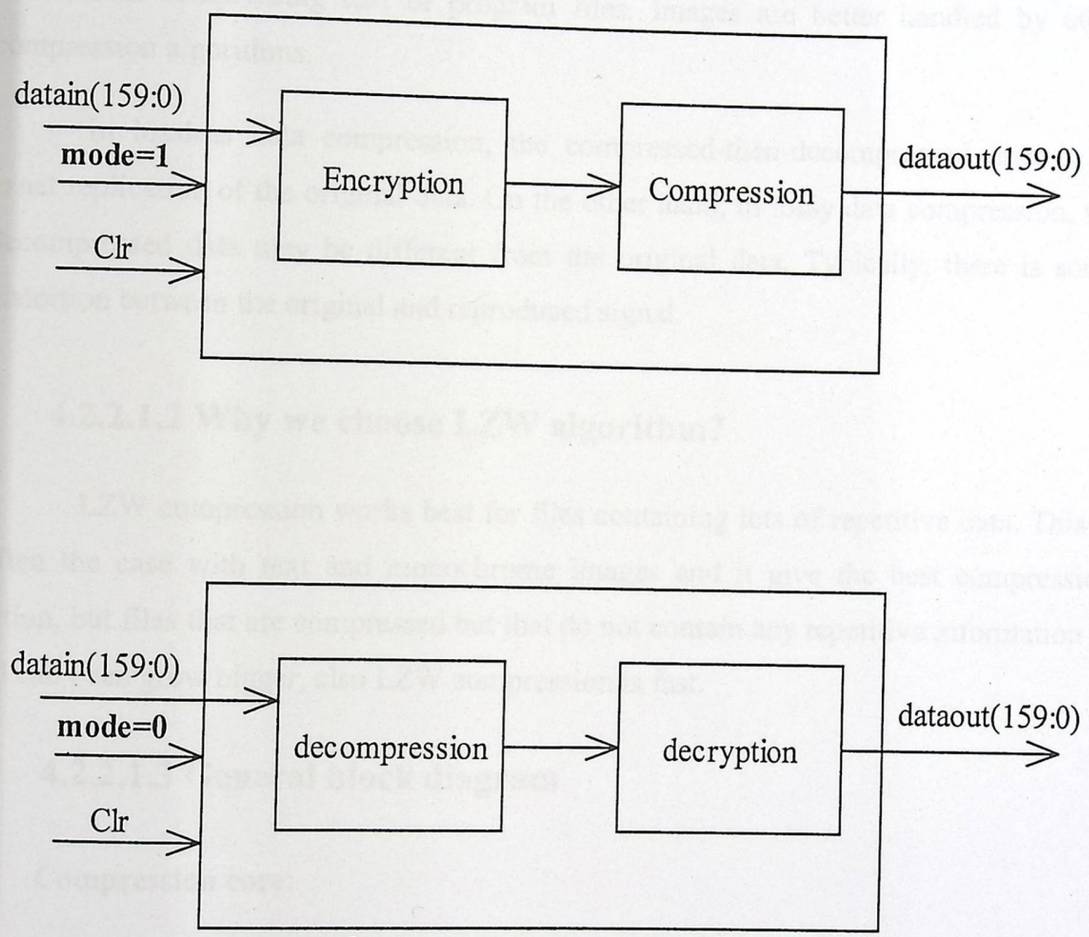


Figure 4.2 Encryption/Compression Unit modes

4.2.2.1 Compression/Decompression cores

In this section we will describe the compression/decompression cores, we chose to implement these cores using LZW lossless compression algorithm.

4.2.2.1.1 Why we choose lossless method?

We chose lossless compression algorithm in our project because it is mainly efficient in compressing text or program files. Images are better handled by other compression algorithms.

In lossless data compression, the compressed-then-decompressed data is an exact replication of the original data. On the other hand, in lossy data compression, the decompressed data may be different from the original data. Typically, there is some distortion between the original and reproduced signal.

4.2.2.1.2 Why we choose LZW algorithm?

LZW compression works best for files containing lots of repetitive data. This is often the case with text and monochrome images and it give the best compression ration, but files that are compressed but that do not contain any repetitive information at all can even grow bigger, also LZW compression is fast.

4.2.2.1.3 General block diagram

- **Compression core:**

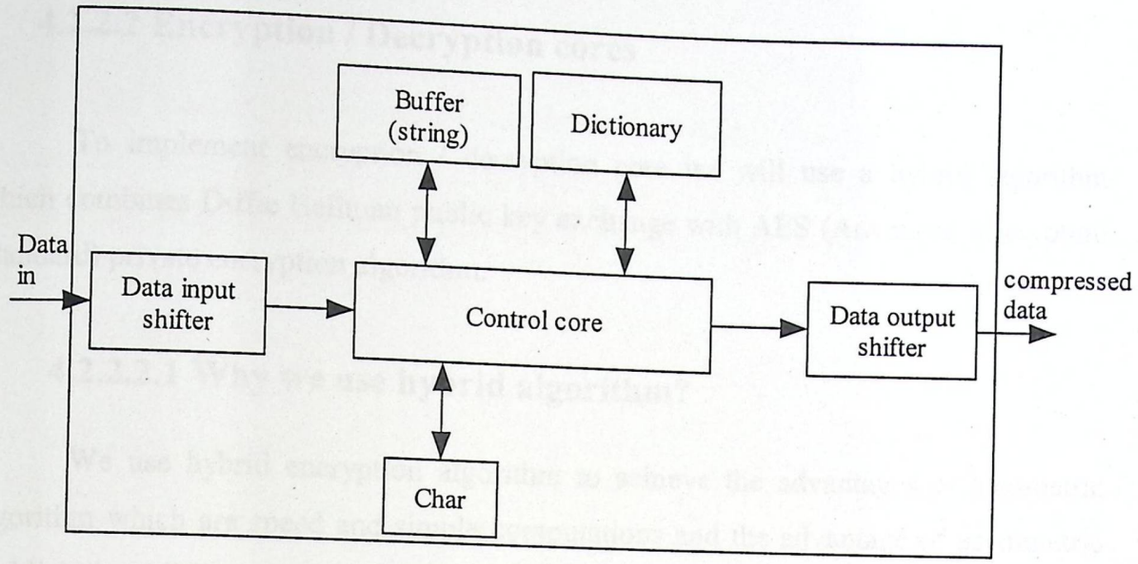
Compression core includes input data shifter, compressed data shifter, two buffers, dictionary which is a memory component, and control core.

The data shifter is used to divide the incoming data into 4 bits each time, the 4 bits is sent the buffer (char) to be compressed , the buffer (string) is used during the compression process, then the data sent to data output shifter, the dictionary is used to store the compression codes.

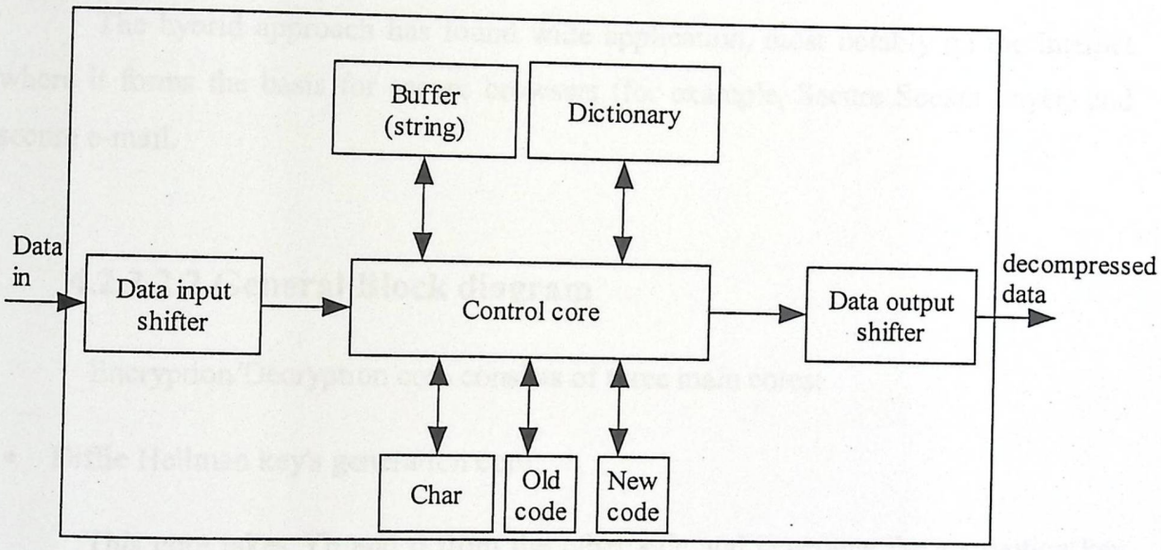
- **Decompression core:**

Decompression core includes input data shifter, decompressed data shifter, 4 buffers, dictionary which is a memory component, and control core.

The data shifter is used to divide the incoming data into 5 bits each time, the 5 bits is sent to newcode to be decompressed the buffers (oldcode, char , string) are used in the decompression process, then the decompressed data is sent to data output shifter, the dictionary is used to store the compression codes.



Compression Core



Decompression Core

Figure 4.3 Compression/Decompression Cores

4.2.2.2 Encryption / Decryption cores

To implement encryption / decryption core we will use a hybrid algorithm which combines Diffie Hellman public key exchange with AES (Advanced Encryption Standard) private encryption algorithm.

4.2.2.2.1 Why we use hybrid algorithm?

We use hybrid encryption algorithm to achieve the advantages of symmetric algorithm which are speed and simple computations and the advantage of asymmetric (public) algorithms which provides the best security.

The hybrid approach has found wide application, most notably on the Internet where it forms the basis for secure browsers (for example, Secure Socket Layer) and secure e-mail.

4.2.2.2.2 General Block diagram

Encryption/Decryption core consists of three main cores:

- Diffie Hellman key's generation core:

This core takes Yb and p from the other side and generates the encryption key which is sent to the encryption or decryption core (depending on ECU mode).

In general a practical Diffie Hellman solution is composed of at least three basic components:

1. Modulus core.

2. Exponentiation core.

3. Data management and control.

The function of these components will be explained by the algorithm flowchart, in the following chapter.

- AES Encryption core:

AES encryption core includes the following components: Byte substituter, row shifter, column mixer, key expander, and encryption controller which control the other components, each component has a basic function in the encryption process , this is to be explained in the following chapter.

- AES Decryption core:

AES encryption core includes the following components: Inverse byte substituter, inverse row shifter, inverse column mixer, key expander, and decryption controller which control the other components, the functions of these components are the inverse of the functions of the encryption components.

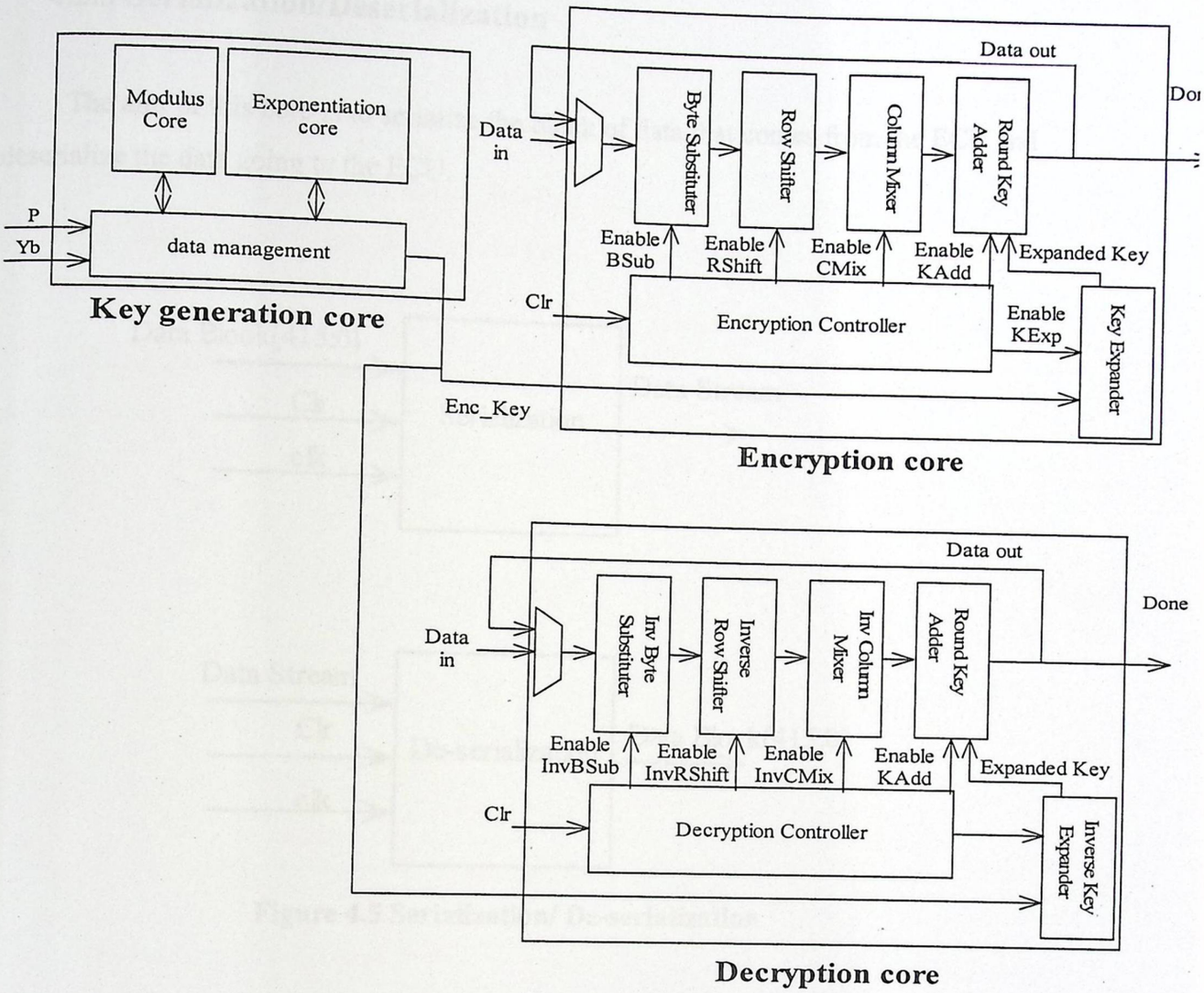


Figure 4.4 Encryption/Decryption Core

4.2.3 Serialization/Deserialization

The aim of this core is to serialize the block of data that comes from the ECU and deserialize the data going to the ECU.

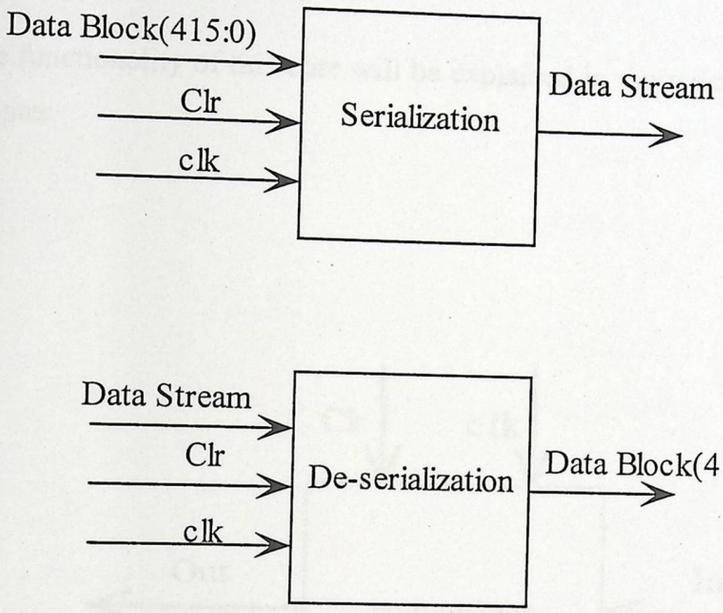


Figure 4.5 Serialization/ De-serialization

4.2.4 Serial port interface

This core interfaces the system with the serial port of the microweb server.

The serial port configuration of data bits, parity, and stop bits are fixed at 8, N, and 1, this means that the size of transmitted data unit equals 8 bits and the start bit is at logic zero and the stop bit is at logic one .

The functionality of this core will be explained in more details by the flowchart in next chapter.

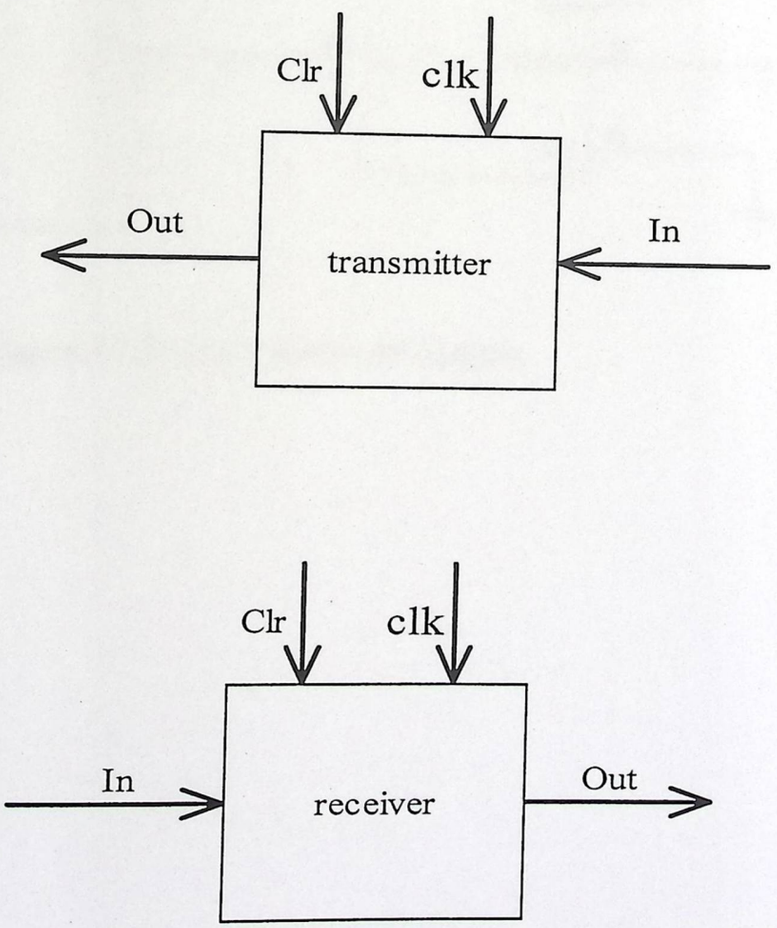


Figure 4.6 Serial port interface

4.3 Schematic Diagram

Figure 4.6 shows the schematic diagram of the system, the FPGA chip (xcv2000E-fg680) is interfaced with the serial port of microweb server.

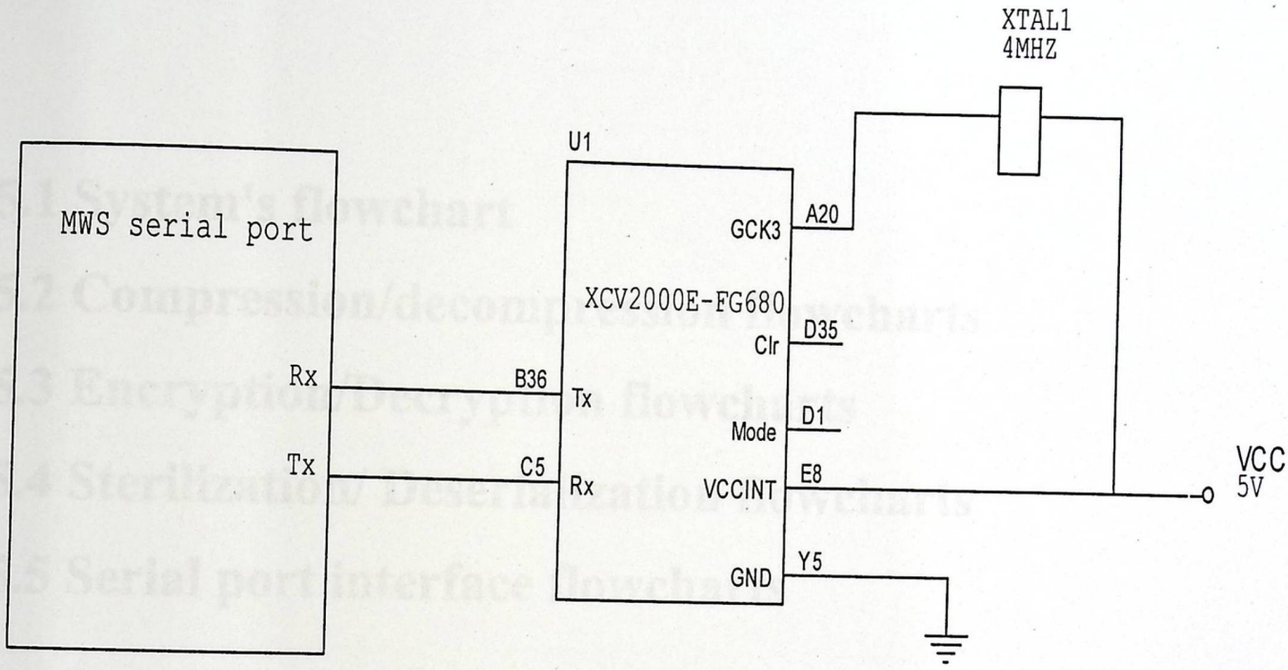


Figure 4.7 System's schematic diagram

Chapter Five

Software system design

5.1 System's flowchart

5.2 Compression/decompression flowcharts

5.3 Encryption/Decryption flowcharts

5.4 Sterilization/ Deserialization flowcharts

5.5 Serial port interface flowcharts

After designing the system in hardware this chapter describes the functionality of the system's algorithms, and it explains in details how each core of the system works.

5.1 System's flowchart

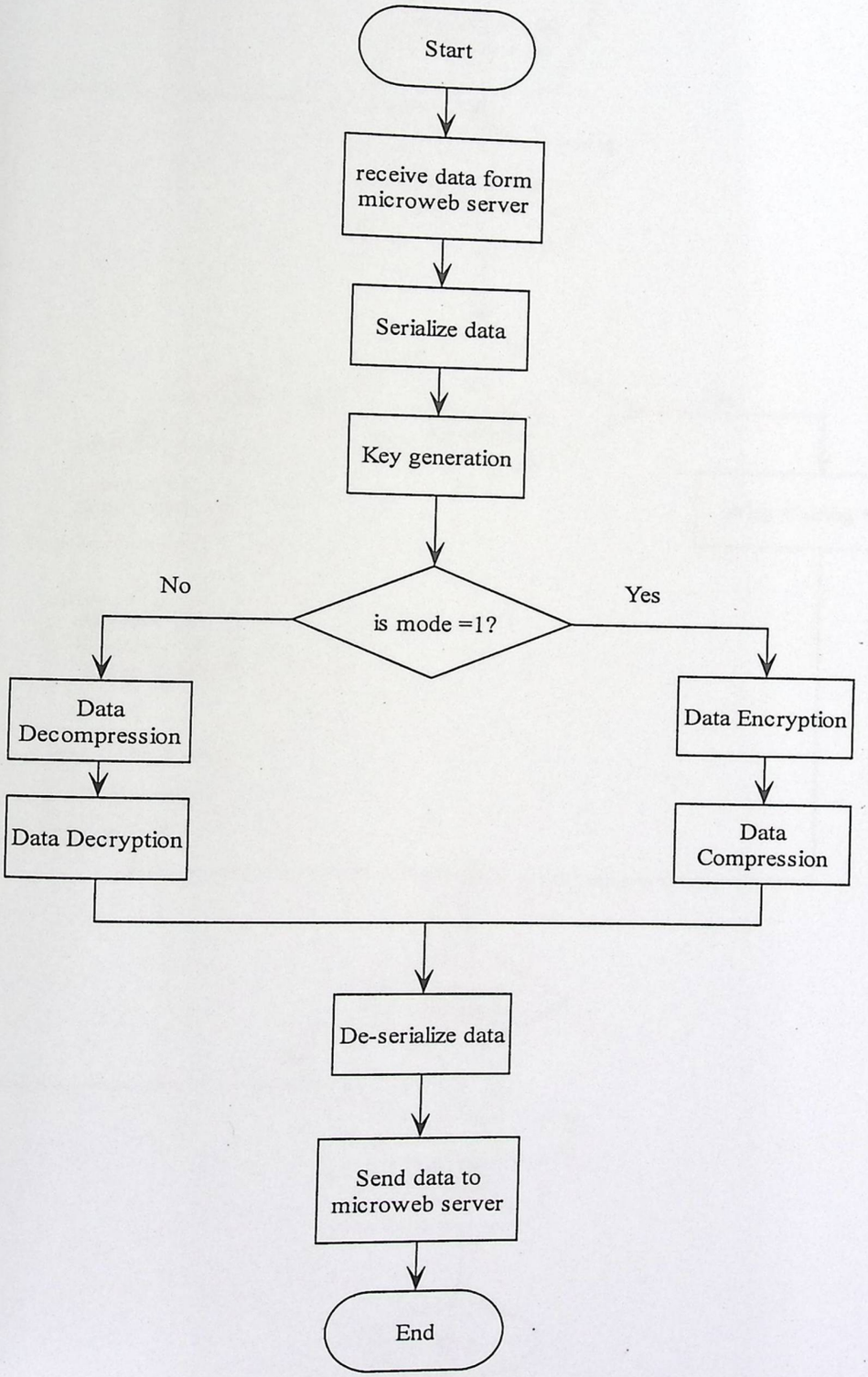


Figure 5.1 System's flow chart

5.2 Compression/decompression flowcharts

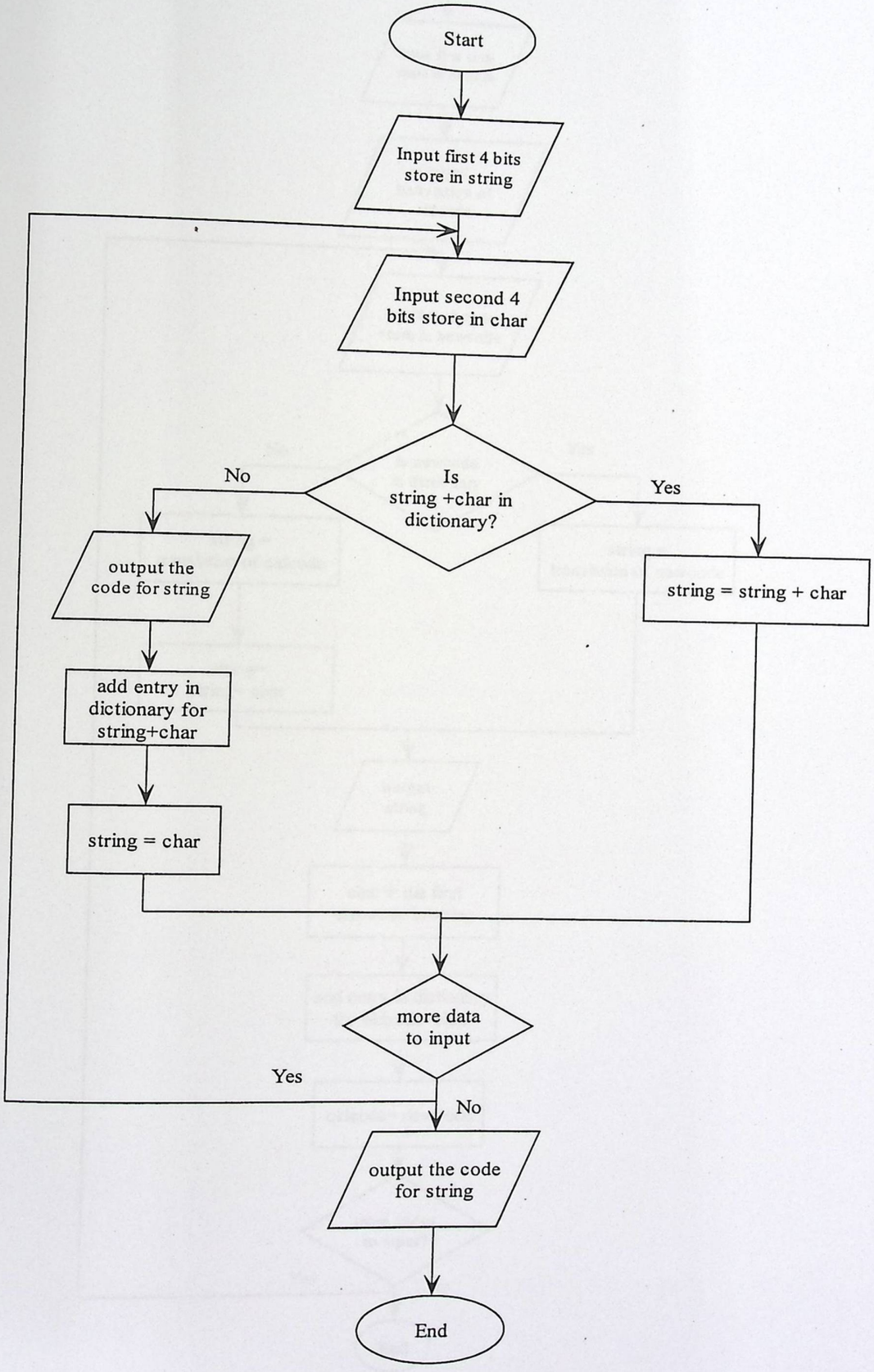


Figure 5.2 LZW compression flow chart

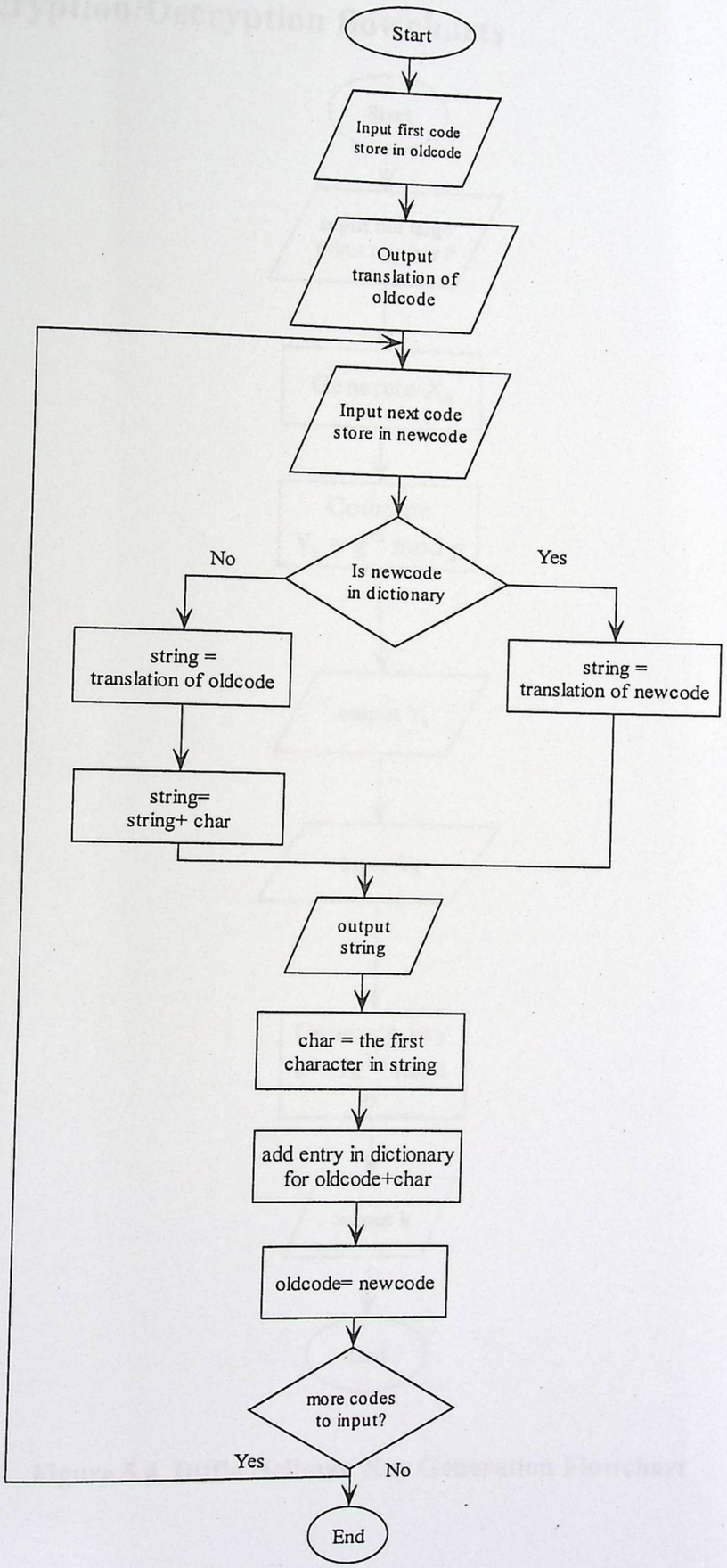


Figure 5.3 LZW decompression flow chart

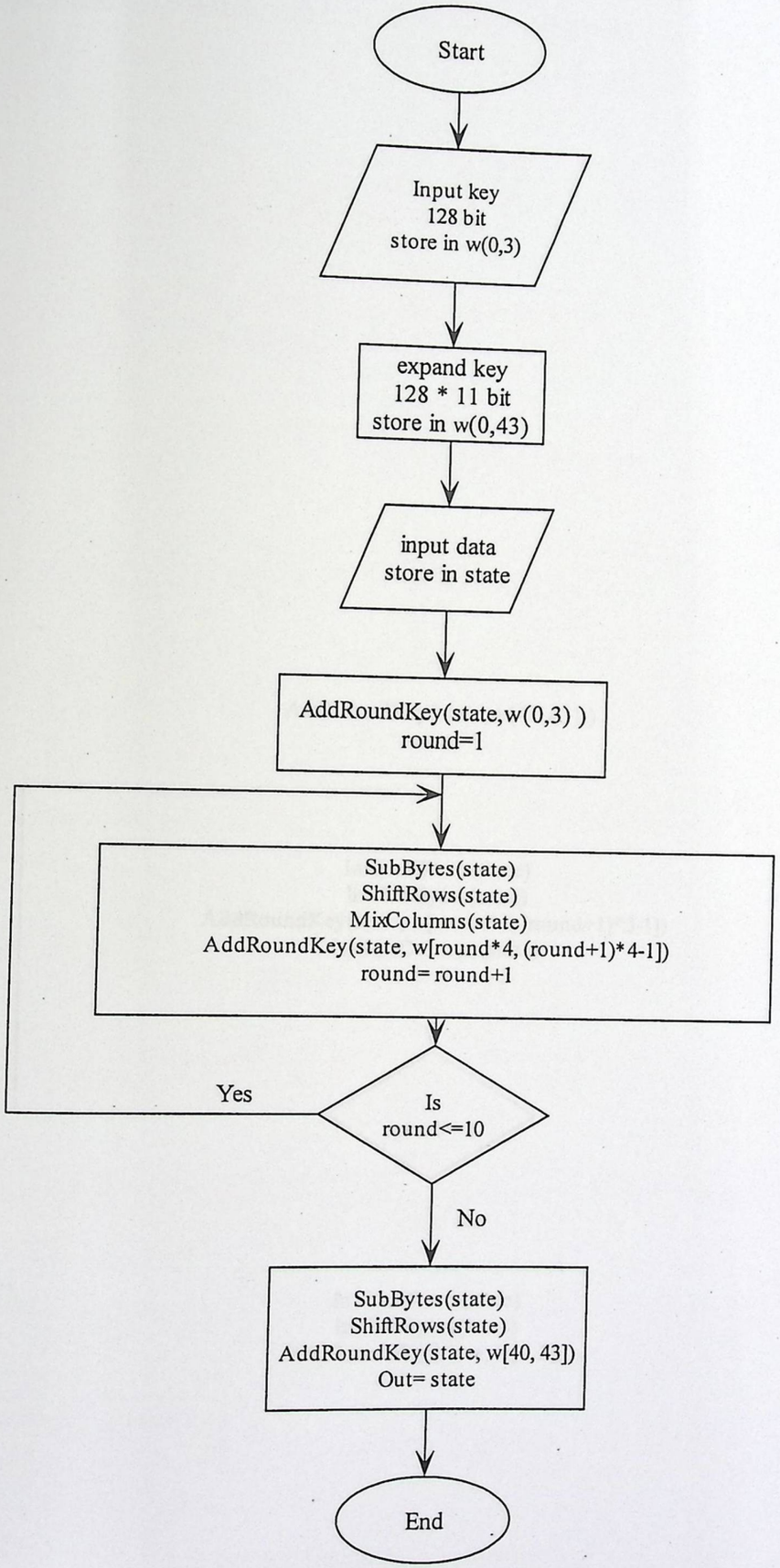


Figure 5.5 AES encryption flowchart

5.3 Encryption/Decryption flowcharts

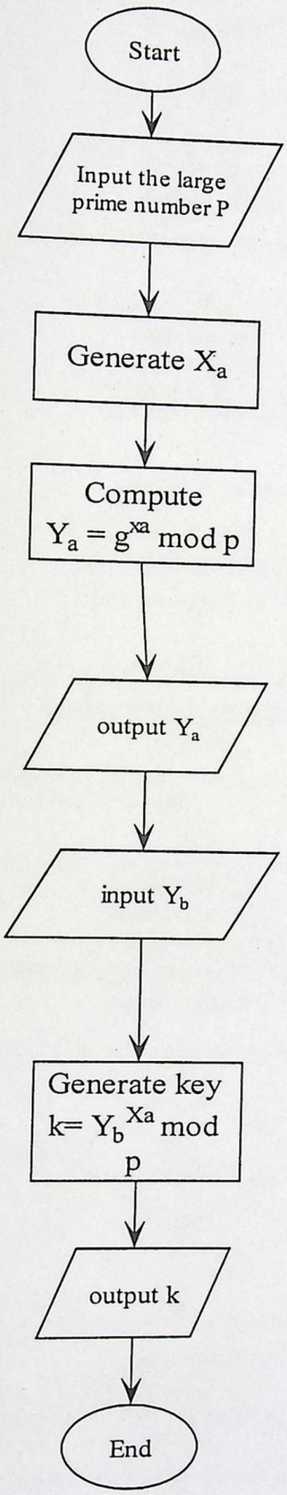


Figure 5.4 Diffie Hellman Key Generation Flowchart

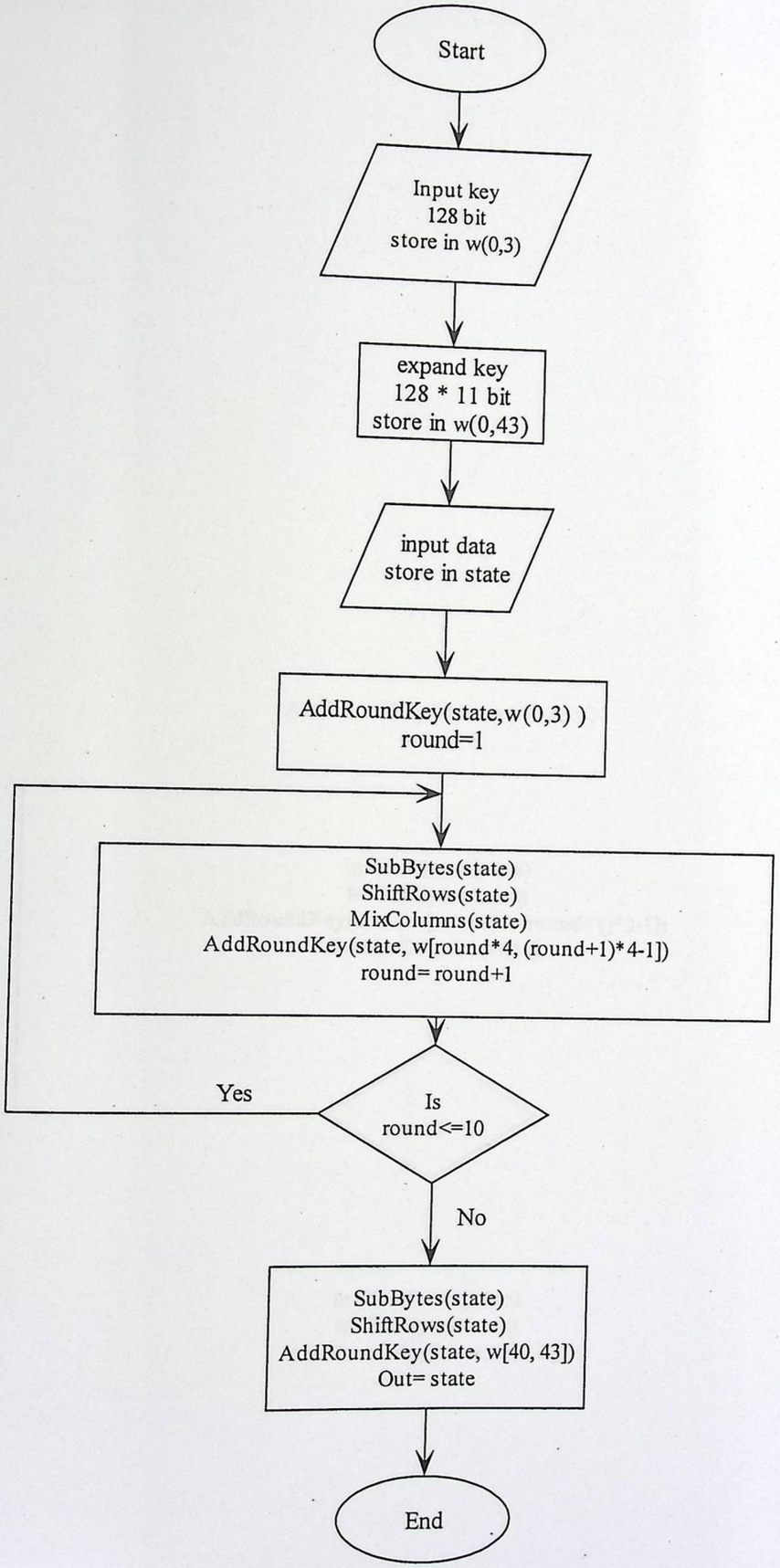


Figure 5.5 AES encryption flowchart

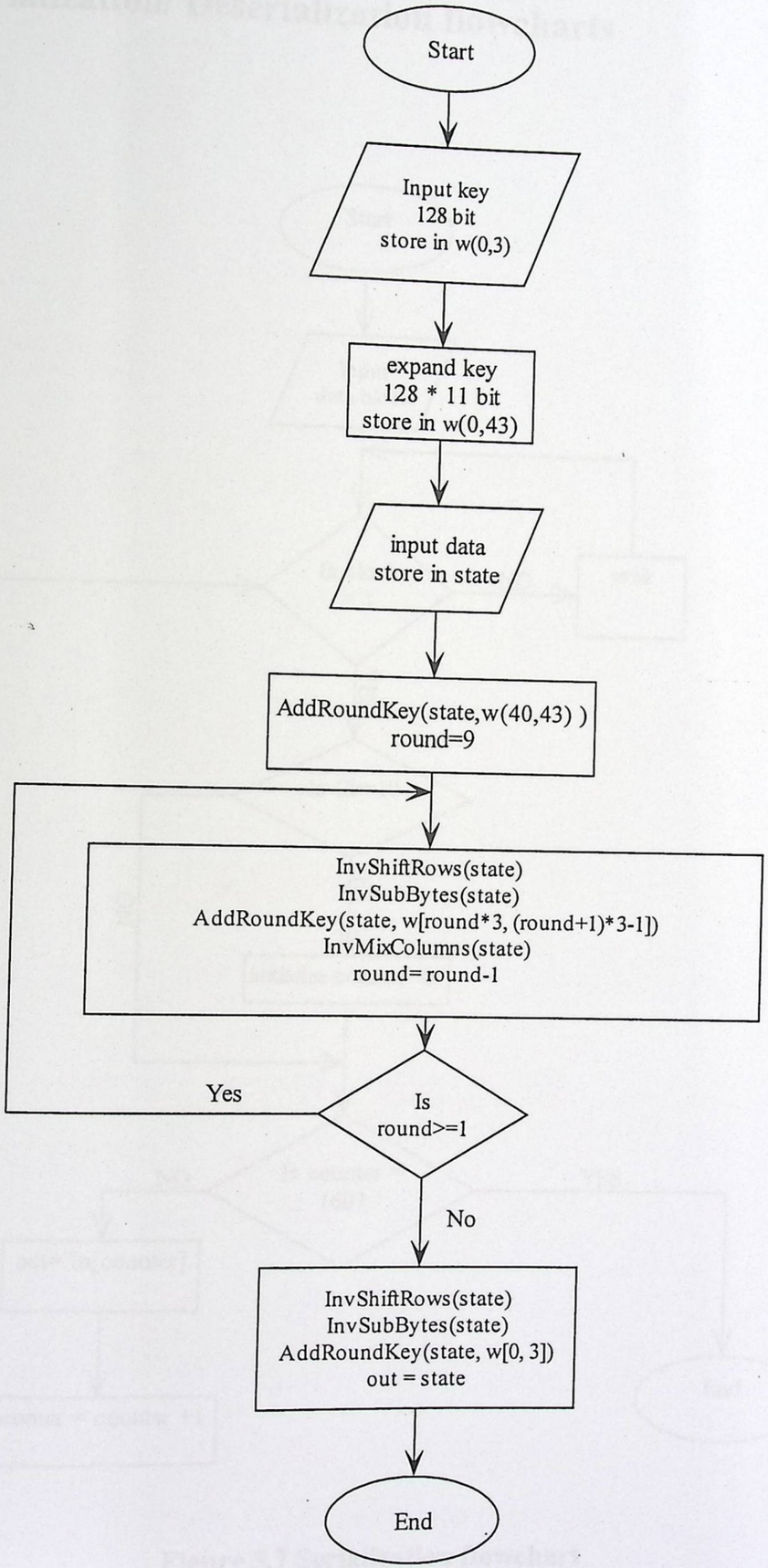


Figure 5.6 AES decryption flowchart

5.4 Serialization/ Deserialization flowcharts

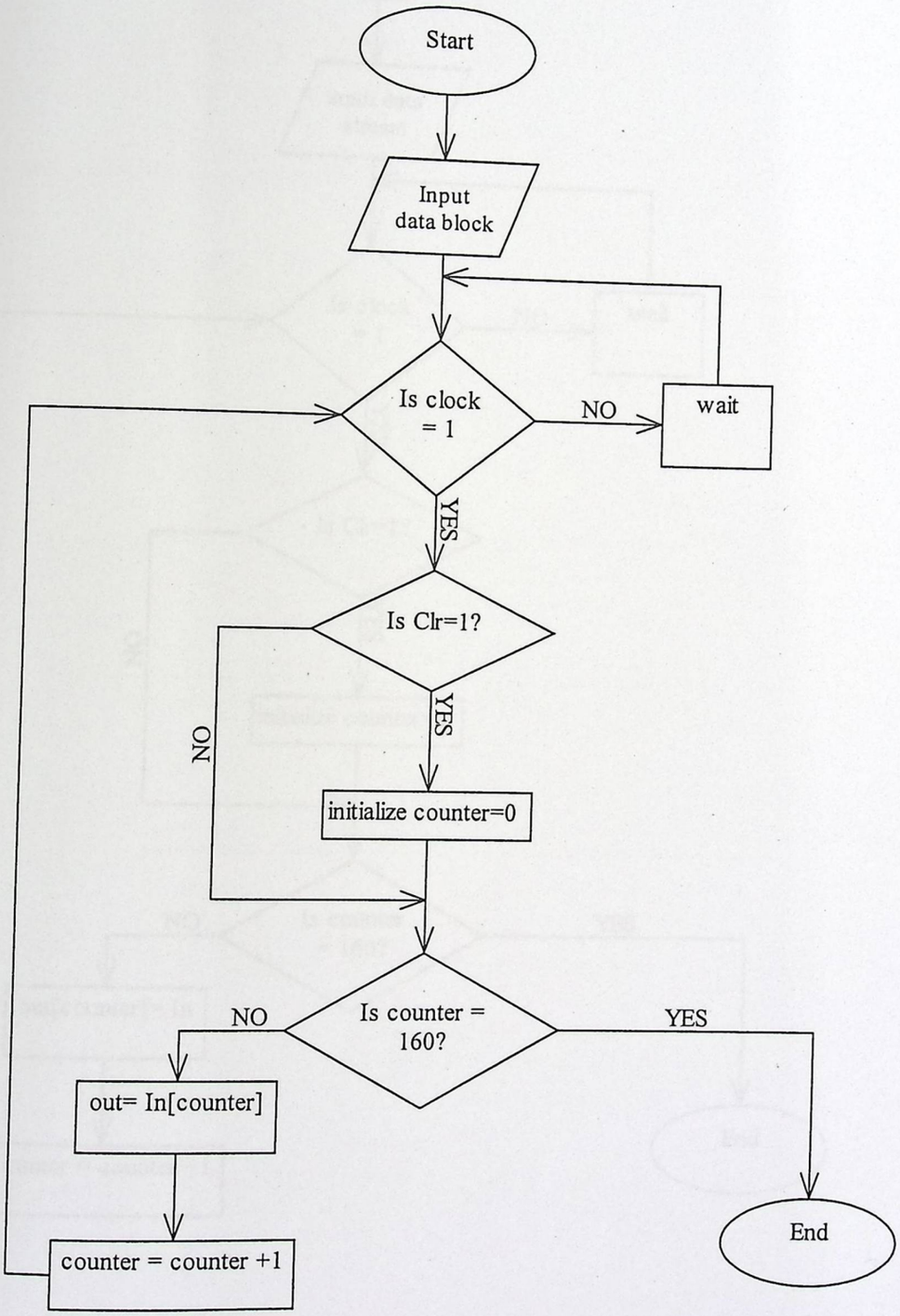


Figure 5.7 Serialization flowchart

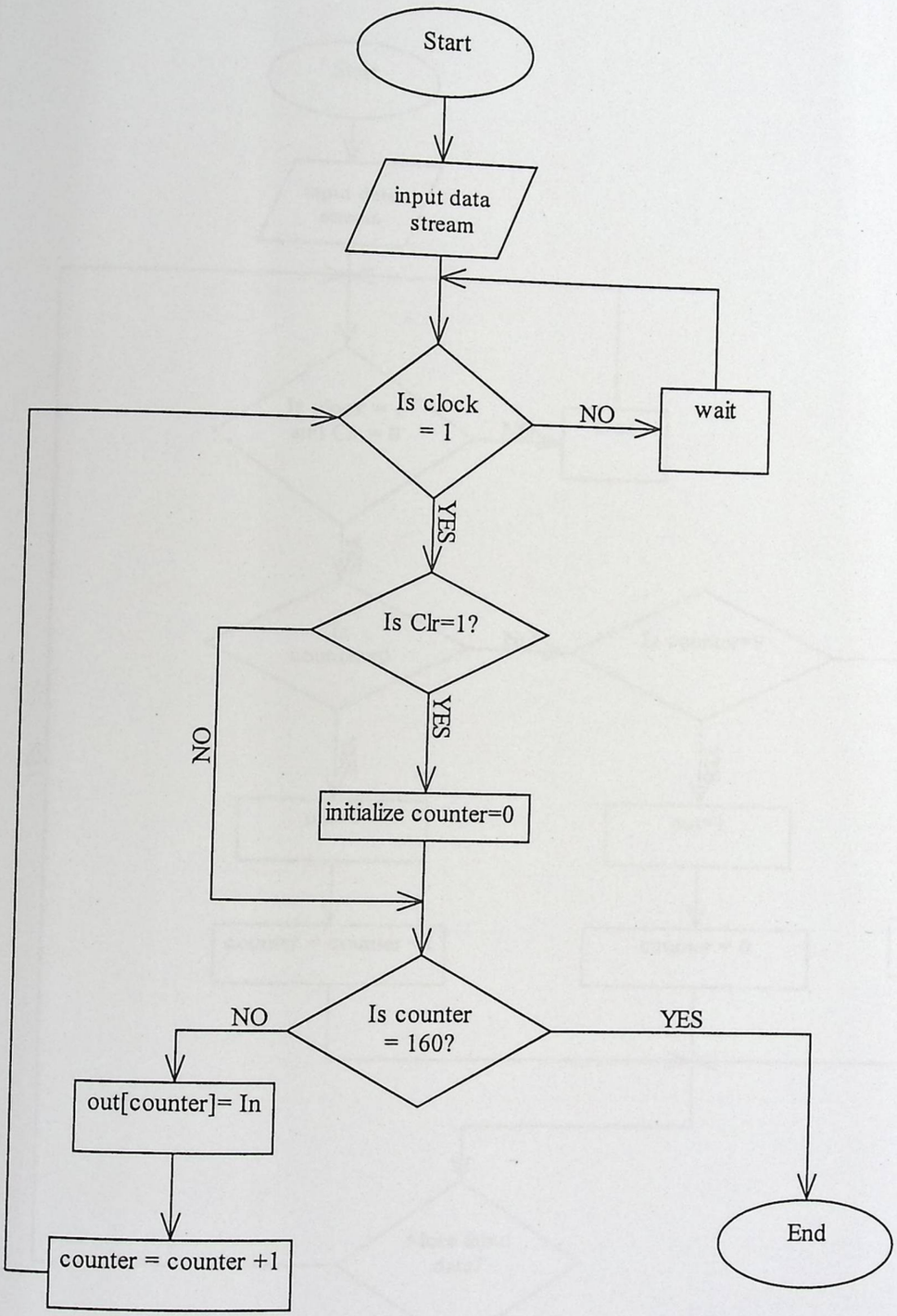


Figure 5.8 Deserialization flowchart

5.5 Serial port interface flowcharts

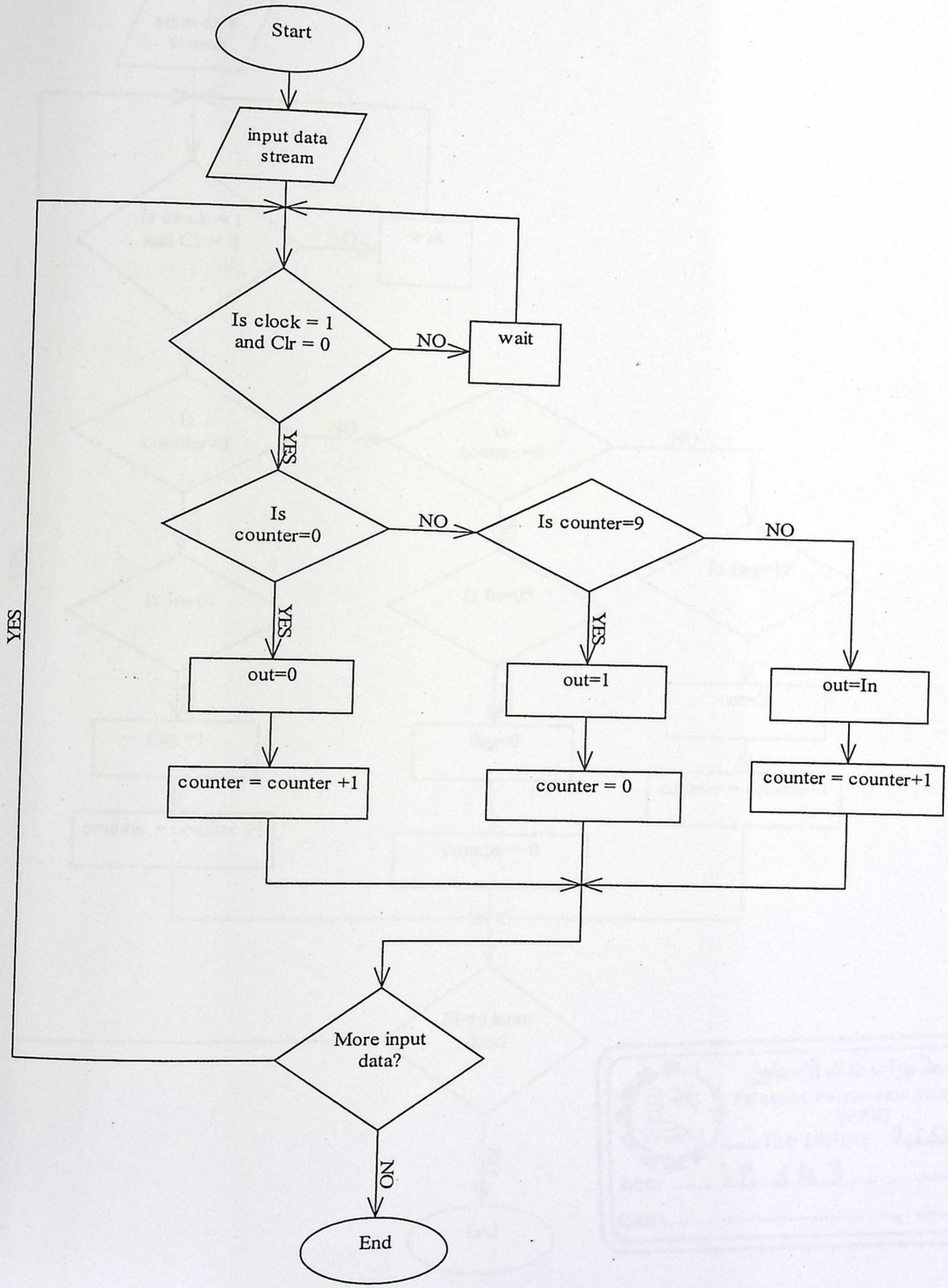


Figure 5.9 Transmitter flowchart

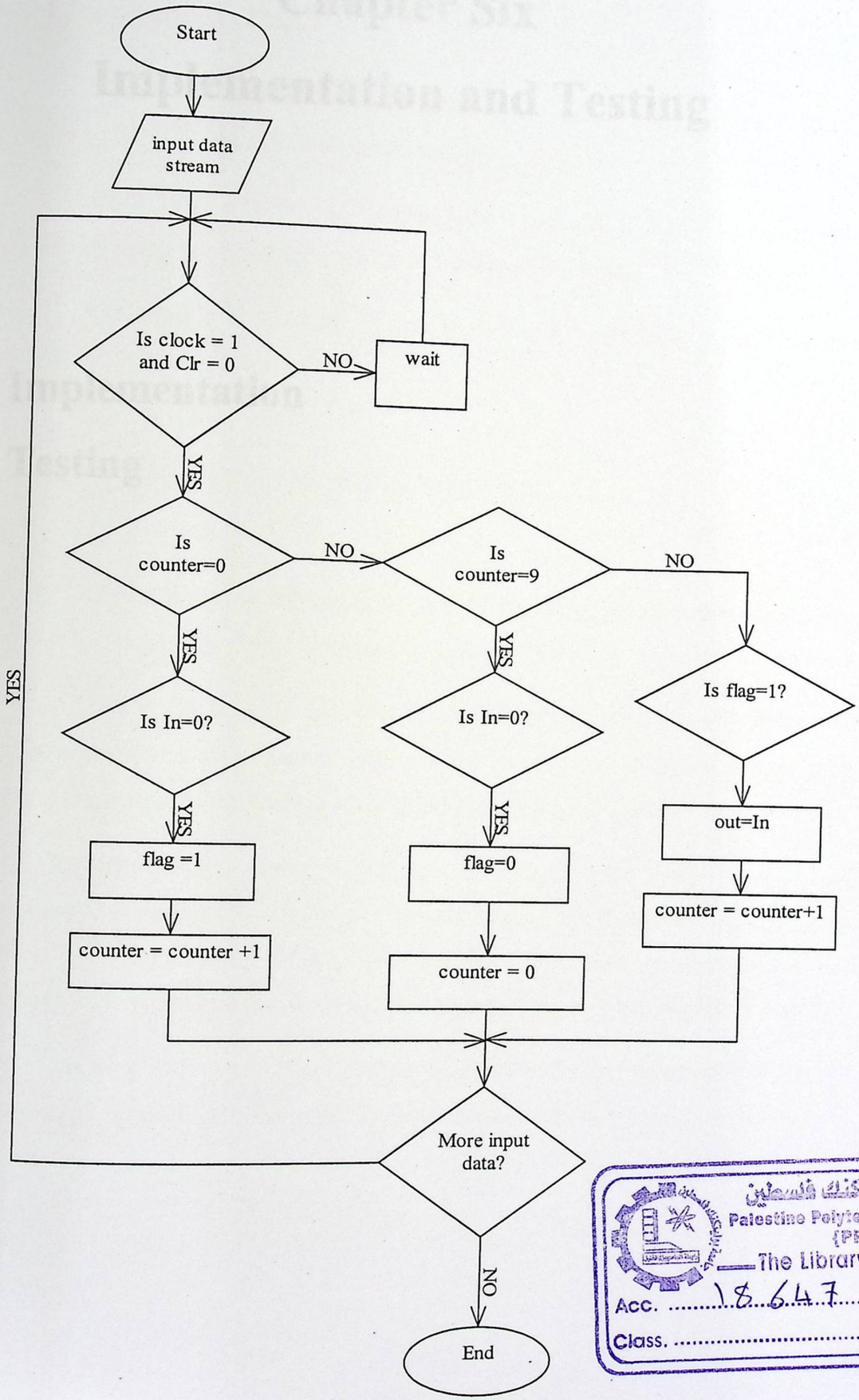


Figure 5.10 Receiver flowchart

Chapter Six

Implementation and Testing

6.1 Implementation

6.2 Testing

6.1 Implementation

The system modules were implemented by verilog hardware description language using Xilinx ISE7 software package.

Since FPGA (FG680–XCV2000E) is not available in PPU; the system was just simulated using Xilinx WebPack 6.0a ModelSim XE Starter.

The Verilog code of the system is included in appendix B, and the system's schematic design is in appendix C.

6.1.1 Why we choose verilog language?

Verilog is the top HDL used by over 10,000 designers at such hardware vendors as Sun Microsystems, Apple Computer and Motorola. Industrial designers like Verilog.

The Verilog language provides the digital designer with a means of describing a digital system at a wide range of levels of abstraction, and, at the same time, provides access to computer-aided design tools to aid in the design process at these levels.

Verilog allows hardware designers to express their design with behavioral constructs, deterring the details of implementation to a later stage of design in the design. An abstract representation helps the designer explore architectural alternatives through simulations and to detect design bottlenecks before detailed design begins.

Verilog also allows the designer to specific designs at the logical gate level using gate constructs and the transistor level using switch constructs.

6.2 Testing

After coding and implementation stage, the system is tested to validate the functionality of the system's modules separately and for the system as one unit.

6.2.1 Unit Testing

Each module in the system was tested individually and all modules worked as expected.

6.2.1.1 Encryption/decryption module

The encryption module is simulated for certain data input, then the decryption is simulated to decrypt the result of encryption module, the output of decryption module was the original data (the data inputted to the encryption module).

The results are shown in the figures 6.1 and 6.2.

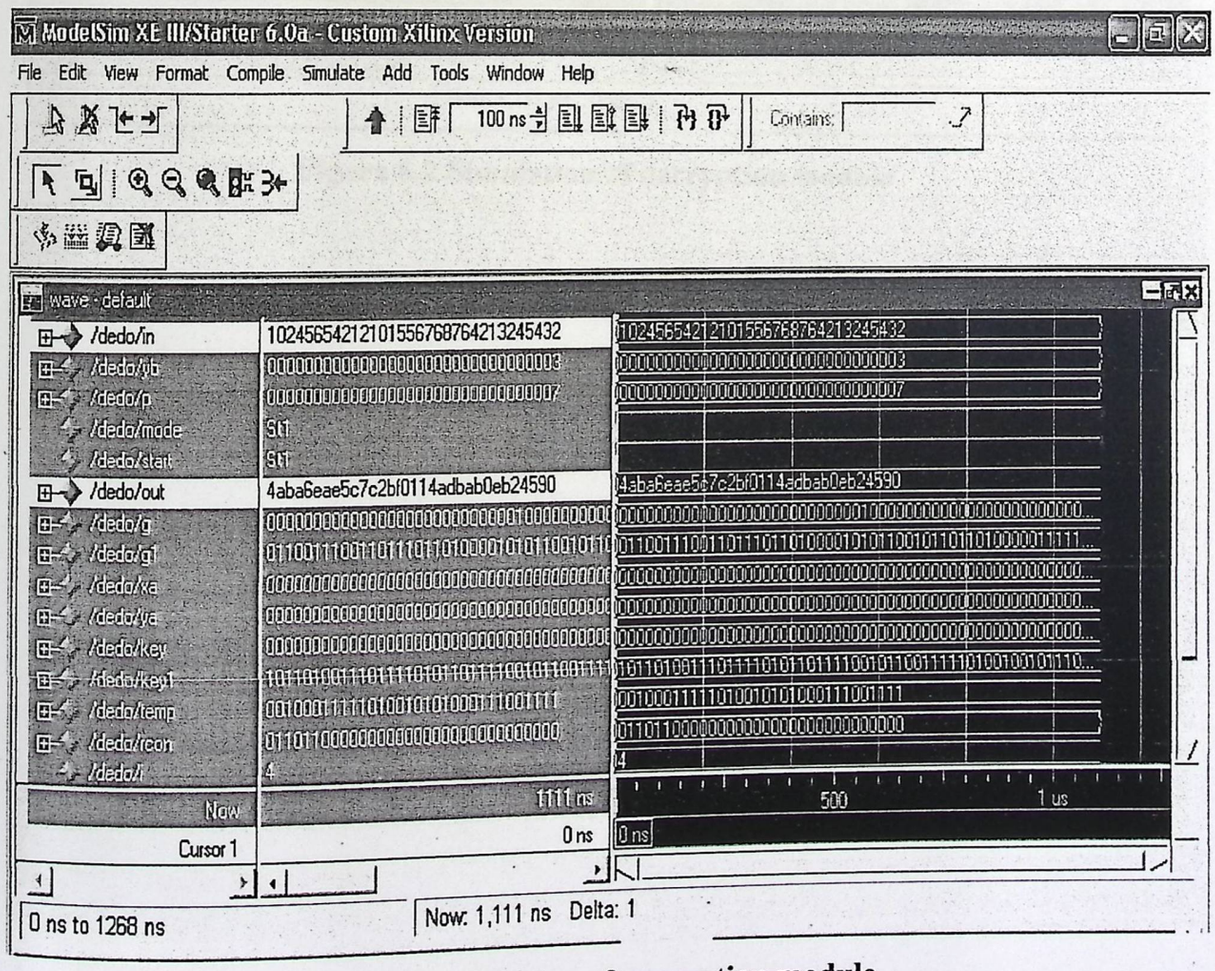


Figure 6.1 Simulation of encryption module

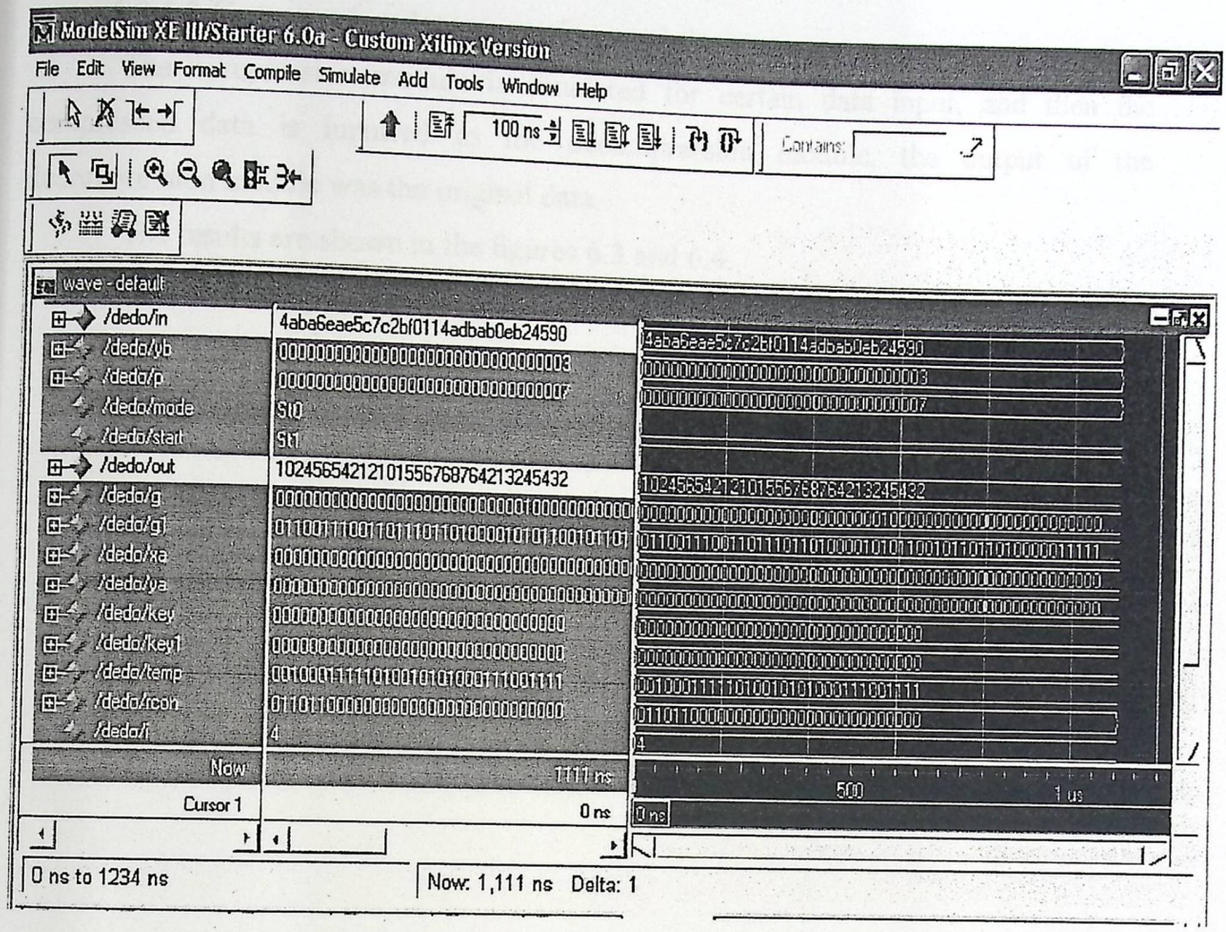


Figure 6.2 Simulation of decryption module

6.2.1.2 Compression/decompression module

The compression module is simulated for certain data input, and then the compressed data is inputted to the decompression module, the output of the decompression module was the original data.

The results are shown in the figures 6.3 and 6.4.

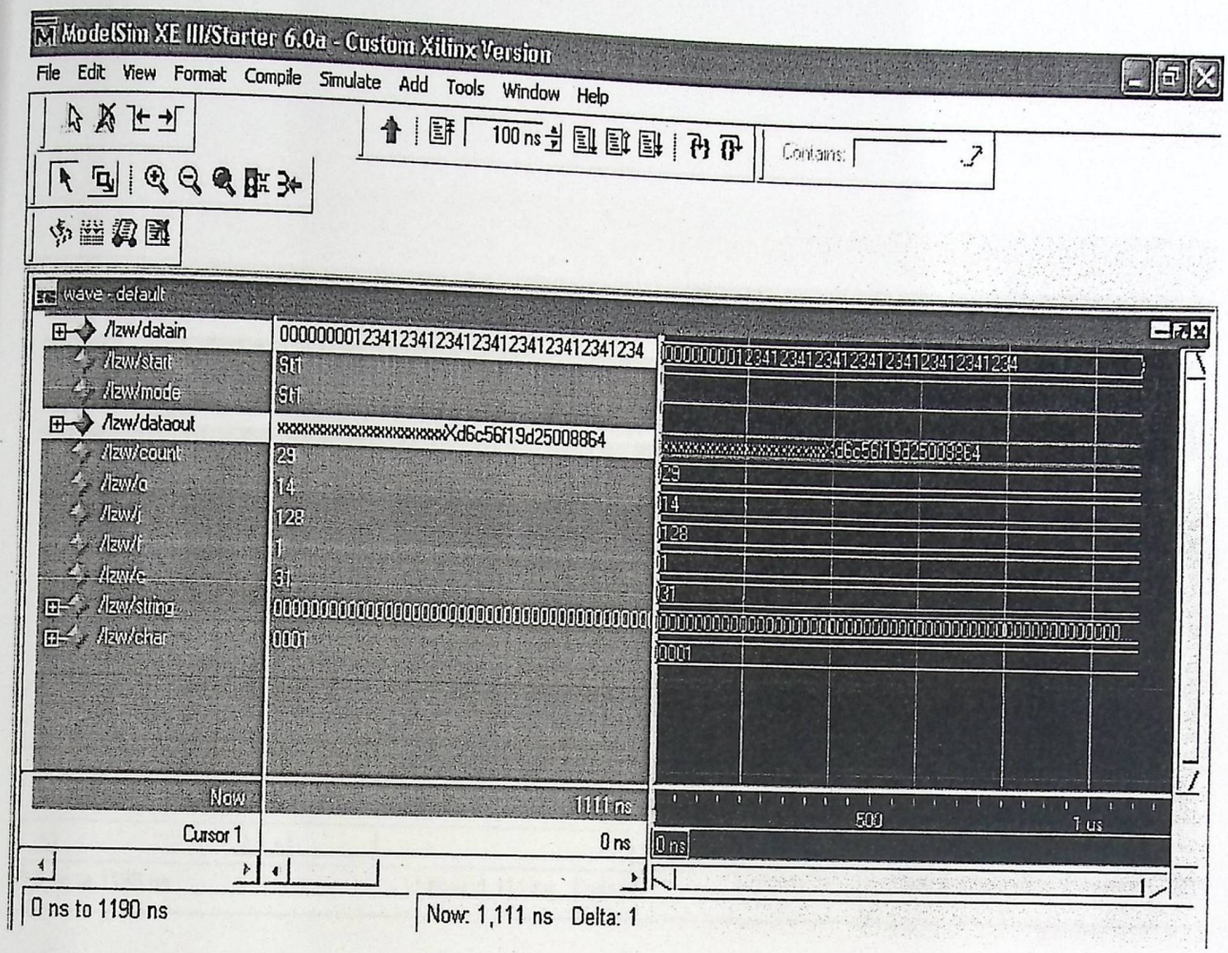


Figure 6.3 Simulation of compression module

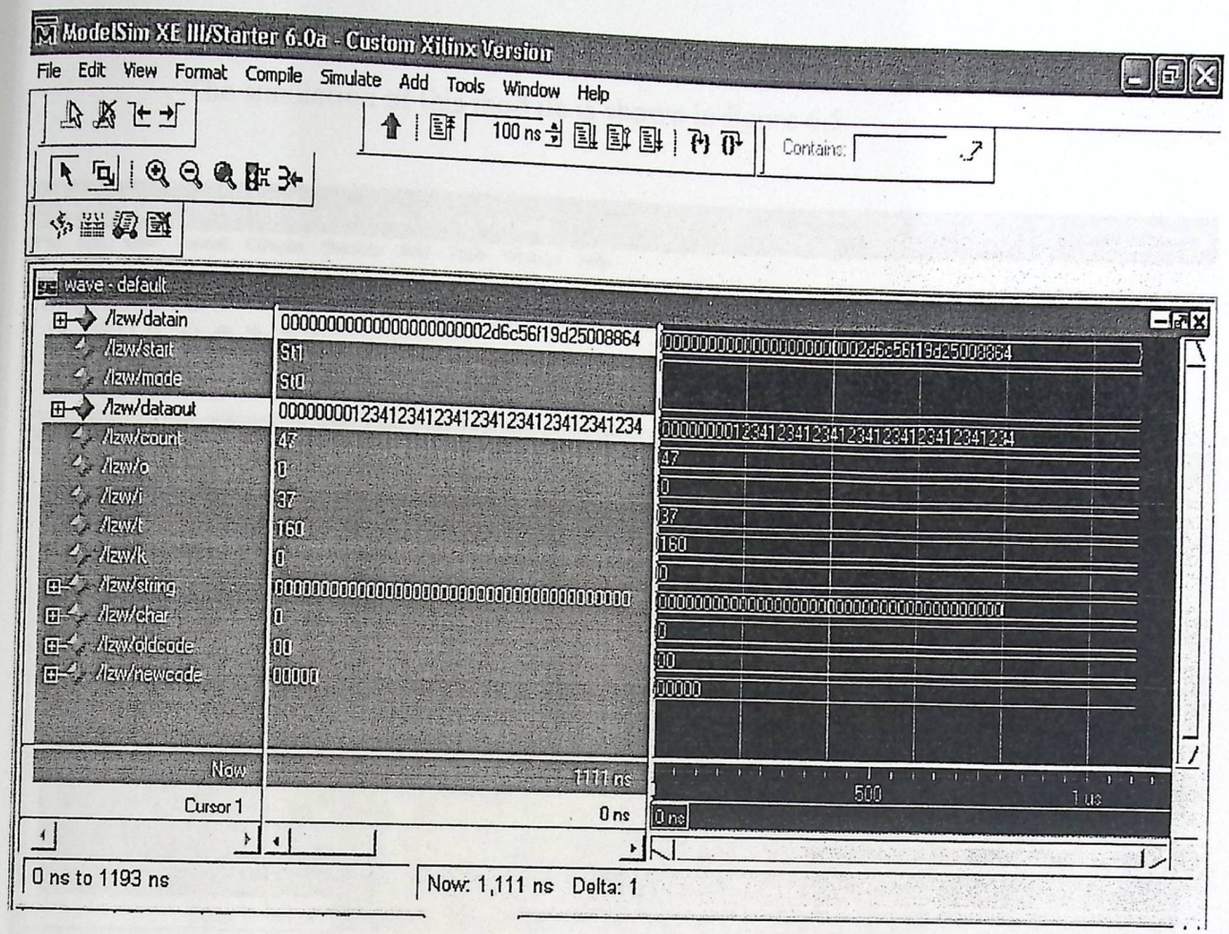


Figure 6.4 Simulation of decompression module

6.2.1.3 Parallel to serial module

The simulation of this module is shown in figure 6.5

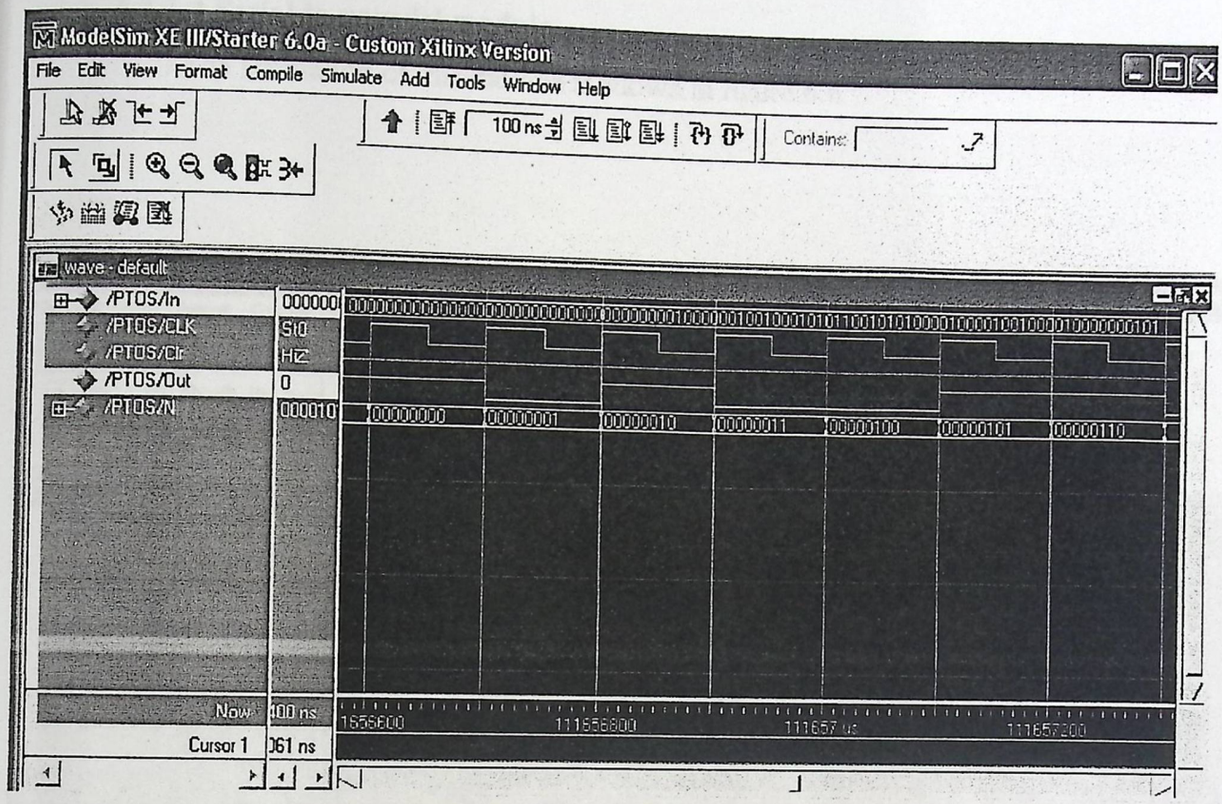


Figure 6.5 Simulation of ptos module

6.2.1.4 Serial to parallel module

The simulation of this module is shown in figure 6.6

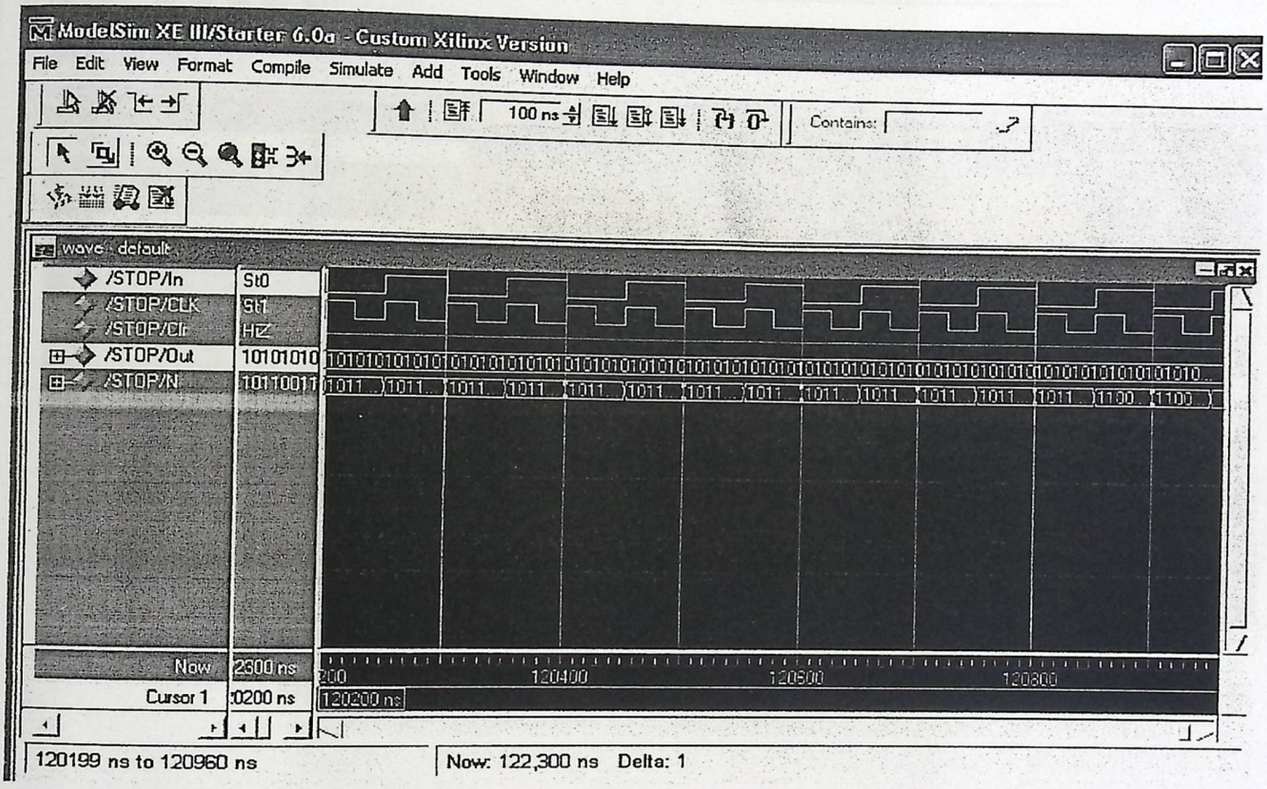


Figure 6.6 Simulation of stop module

6.2.1.4 Serial to parallel module

The simulation of this module is shown in figure 6.6

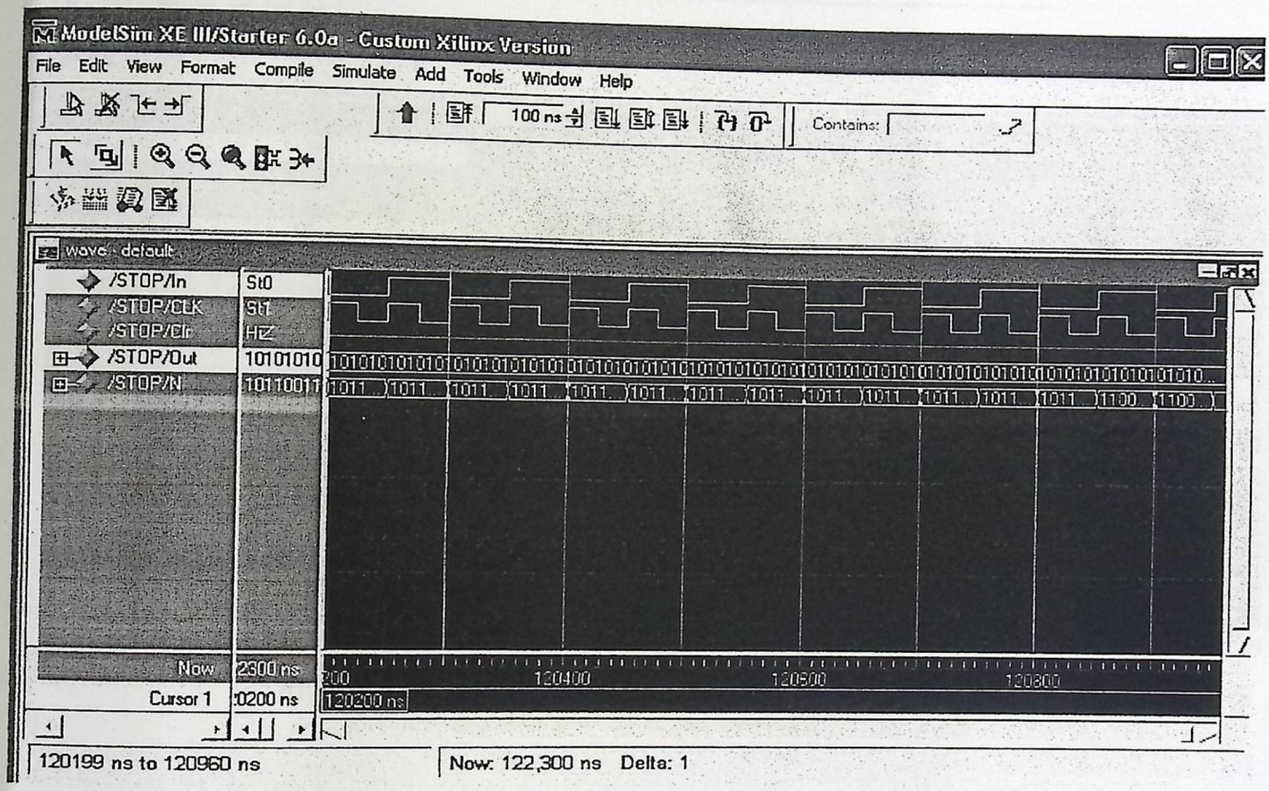


Figure 6.6 Simulation of stop module

6.2.1.5 Transmitter module

This module transmits data to the serial port of micro web server; the simulation of this module is shown in figure 6.7

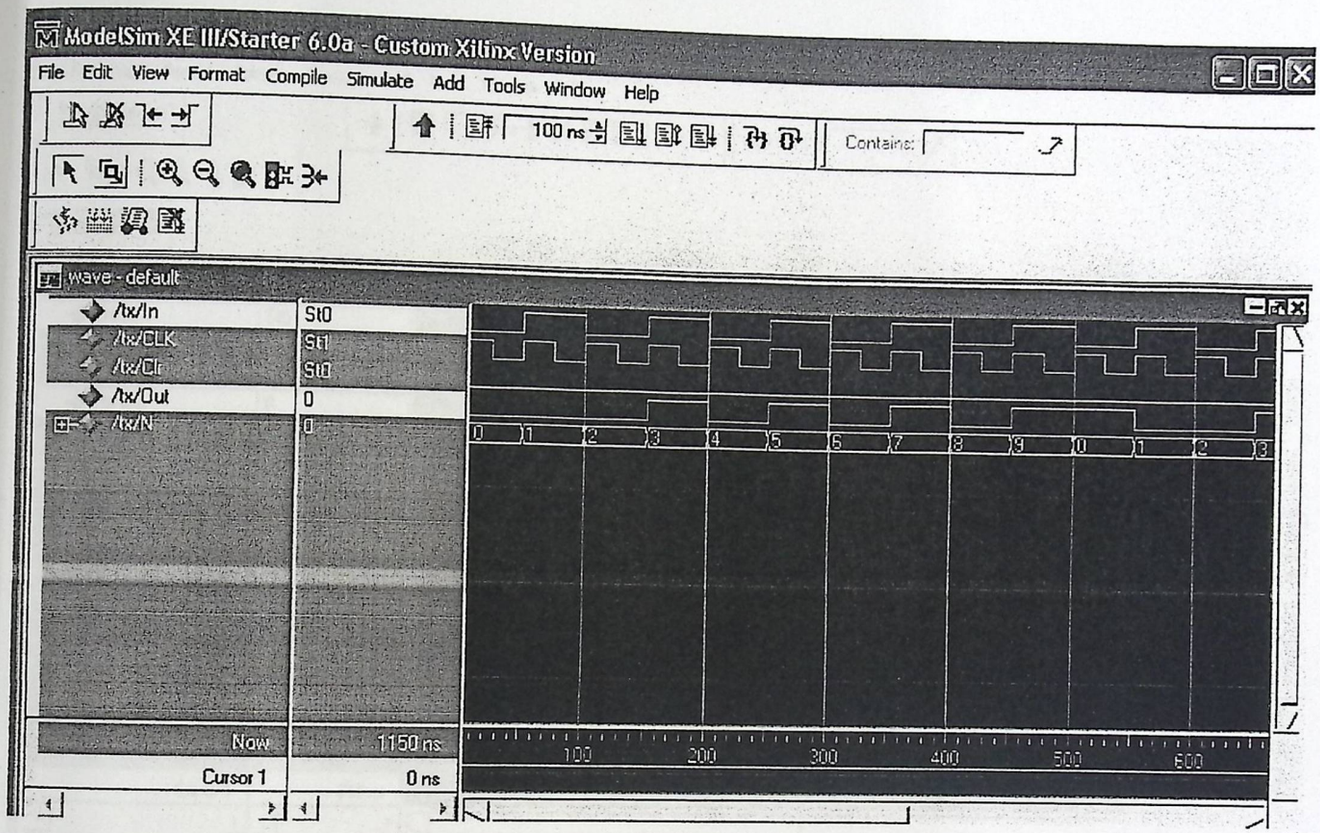


Figure 6.7 Simulation of transmitter module

6.2.1.6 Receiver module

This module receives data to the serial port of micro web server; the simulation of this module is shown in figure 6.8

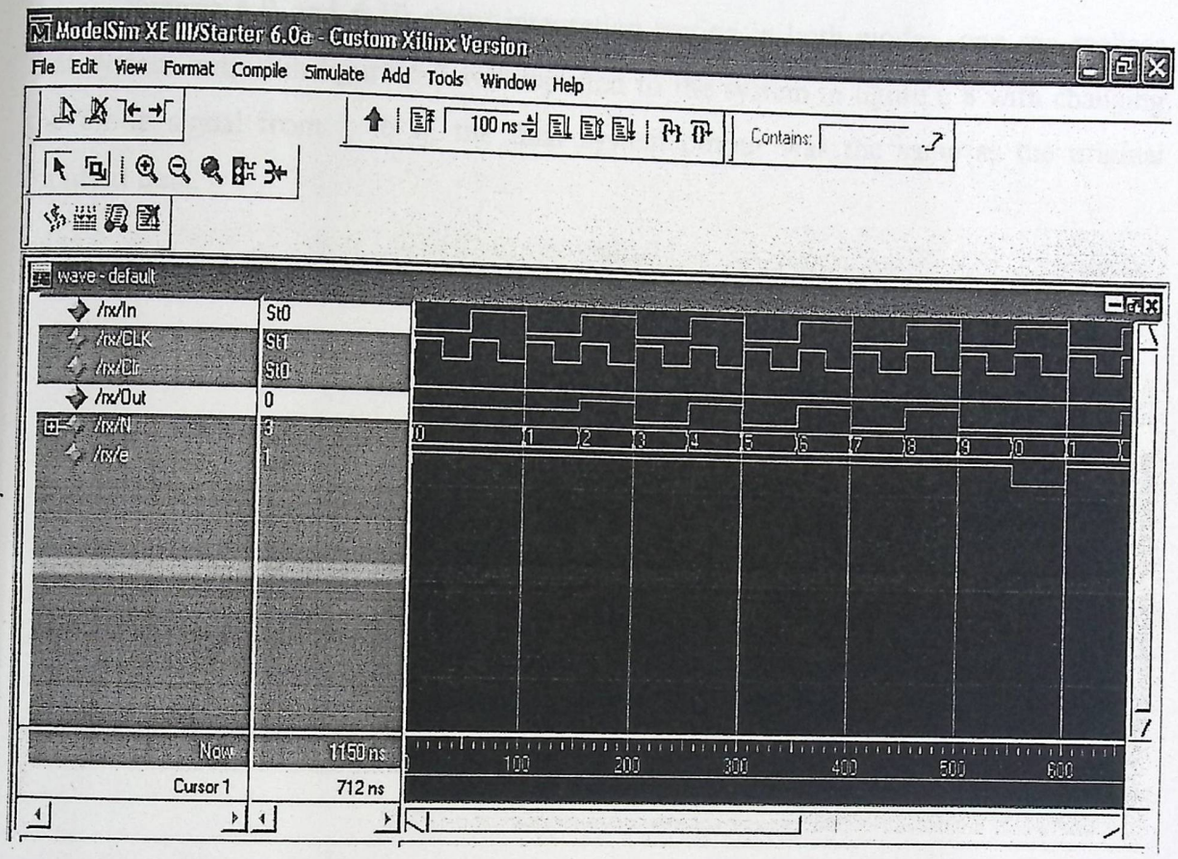


Figure 6.8 Simulation of receiver module

6.2.2 Integration Testing

After testing each module and insuring that they function as expected, these modules were integrated together and tested as one unit.

Figures 6.9 and 6.10 show integration testing in both modes, one can realizes that system output in figure 6.7 was inputted to the system in figure 6.8 with changing the mode signal from 1 to 0, the final system output was the same as the original inputted data.

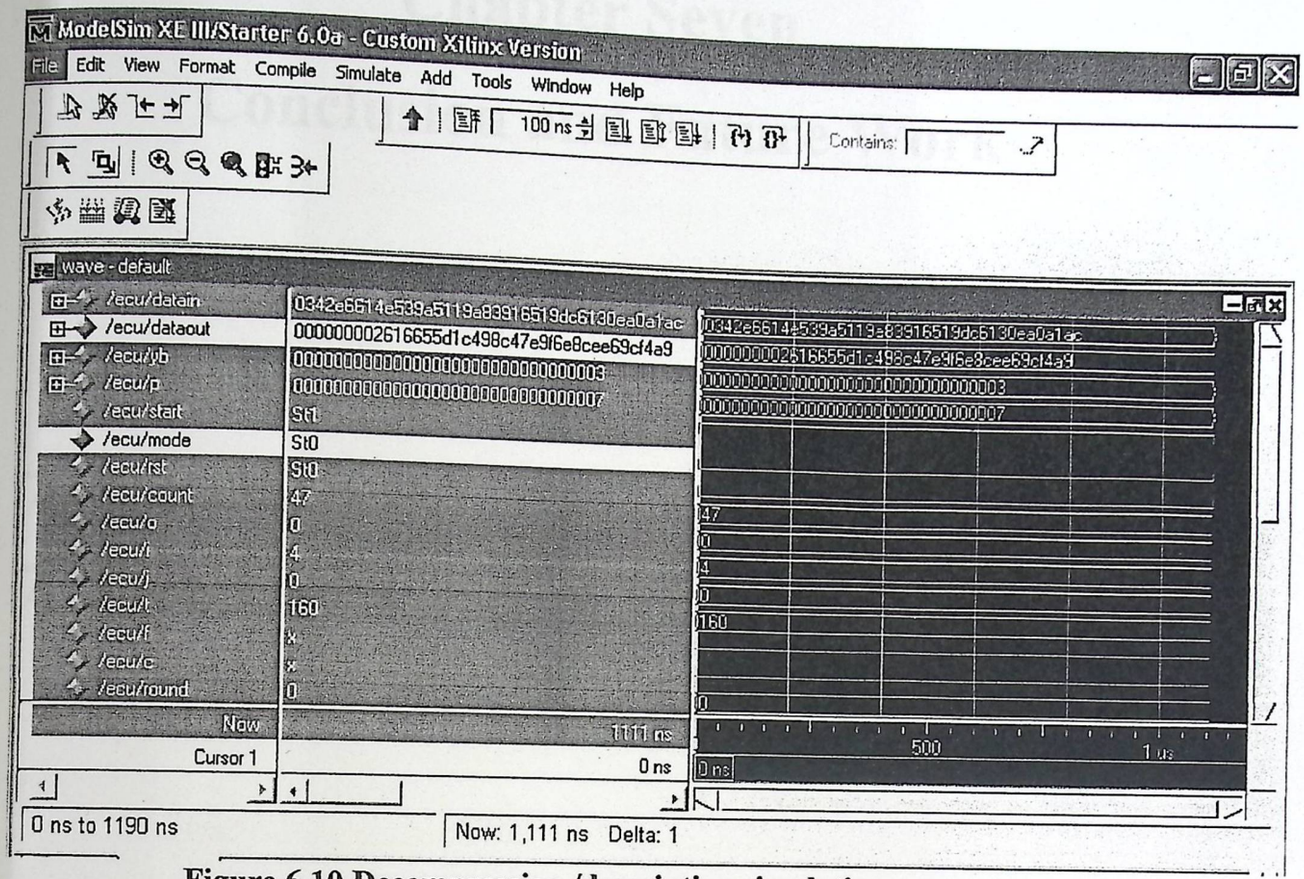


Figure 6.10 Decompression /description simulation

Chapter Seven
Conclusion and Future Work

7.1 Conclusion

7.2 Future work

7.1 Conclusion

After studying encryption and compression algorithms, and choosing AES and diffie-hellman for encryption and lzw for compression, verilog HDL was written for the system.

The implementation of the design couldn't be implemented on hardware because the required FPGA (xcv2000E) and its board it's not found in the university.

We can conclude the following:

- According to our search for literature studies we didn't found any system that implements encryption/decryption, compression/decompression, and microweb server serial port interface together using FPGA.
- Although this system is relatively large, it can be implemented in only one chip.
- According to the testing process the system functions correctly, the results was as expected.
- The system functionality can be modified easily by modifying the code and reprogramming the same FPGA chip.

7.2 Future work

We suggest the following future works for this system are some suggested future works:

- Other new encryption and compression algorithms can be used to give more speed and security
- The design can be interfaced with micro web server parallel port rather than serial port.
- The compression/encryption unit can be interfaced with other devices for different kinds of data.

References

References

1. Demah Badawee,Shorooq Abud dayya and Rami atawneh - Hardware Encrypto Unit - Palestine Polytechnic University – Hebron- 2003
2. Neveen Aqeelan, Rania Halayqa and Aseel Noman- PCI Encryption Card - Palestine Polytechnic University – Hebron- 2004
3. http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/mags/mu/&toc=comp/mags/mu/2003/03/u3toc.xml&DOI=10.1109/MMUL.2003.1218259
4. http://www.mycrypto.net
5. http://en.wikipedia.org/wiki/encryption
6. http://en.wikipedia.org/wiki/Diffie-Hellman
7. http://www.scramdisk.clara.net/pgpfaq.htm#SubDH
8. http://en.wikipedia.org/wiki/Aes
9. http://www.cs.eku.edu/faculty/styer/460/Encrypt/JS-AES.html
10. http://en.wikipedia.org/wiki/data_compression
11. http://www.data-compression.com
12. http://en.wikipedia.org/wiki/data_lossless
13. http://en.wikipedia.org/wiki/burrows_wheeler_transform
14. http://en.wikipedia.org/wiki/DEFLATE
15. http://en.wikipedia.org/wiki/LZW
16. http://en.wikipedia.org/wiki/dspguide.com/datacomp.htm
17. http://cs.sfu.ca/CC/365/li/squeeze/Huffman.html
18. http://en.wikipedia.org/wiki/arithmetixc_coding
19. http://en.wikipedia.org/wiki/lossy_data
20. Morris Mano – Digital Design – 3rd edition – Prentice hall- New Jersey- 2002
21. http://andraka.com/whatisan.htm
22. http://edn.com/archives/1996/101096/21df_07.htm
23. http://www.fpga4fun.com/WhatAreFPGAs.html
24. http://infoeng.ee.ic.ac.uk/~malikz/surprise2001/as399e/questions
25. http://www.xess.com/fpgatut.htm
26. http://www.eg.bucknell.edu/~cs320/1995-fall/verilog-manual.html
27. http://www.verilog.com/v-faq.html
28. http://www.ipsil.com

Appendices

Appendix A

XCV2000E -FG680

Virtex™-E 1.8 V Field Programmable Gate Arrays

DS022-1 (v2.3) July 17, 2002

Production Product Specification

Features

- Fast, High-Density 1.8 V FPGA Family
  - Densities from 58 k to 4 M system gates
  - 130 MHz internal performance (four LUT levels)
  - Designed for low-power operation
  - PCI compliant 3.3 V, 32/64-bit, 33/66-MHz
- Highly Flexible SelectI/O+™ Technology
  - Supports 20 high-performance interface standards
  - Up to 804 singled-ended I/Os or 344 differential I/O pairs for an aggregate bandwidth of > 100 Gb/s
- Differential Signalling Support
  - LVDS (622 Mb/s), BLVDS (Bus LVDS), LVPECL
  - Differential I/O signals can be input, output, or I/O
  - Compatible with standard differential devices
  - LVPECL and LVDS clock inputs for 300+ MHz clocks
- Proprietary High-Performance SelectLink™ Technology
  - Double Data Rate (DDR) to Virtex-E link
  - Web-based HDL generation methodology
- Sophisticated SelectRAM+™ Memory Hierarchy
  - 1 Mb of internal configurable distributed RAM
  - Up to 832 Kb of synchronous internal block RAM
  - True Dual-Port BlockRAM capability
  - Memory bandwidth up to 1.66 Tb/s (equivalent bandwidth of over 100 RAMBUS channels)
  - Designed for high-performance Interfaces to External Memories
  - 200 MHz ZBT* SRAMs
  - 200 Mb/s DDR SDRAMs
  - Supported by free Synthesizable reference design
- High-Performance Built-In Clock Management Circuitry
  - Eight fully digital Delay-Locked Loops (DLLs)
  - Digitally-Synthesized 50% duty cycle for Double Data Rate (DDR) Applications
  - Clock Multiply and Divide
  - Zero-delay conversion of high-speed LVPECL/LVDS clocks to any I/O standard
- Flexible Architecture Balances Speed and Density
  - Dedicated carry logic for high-speed arithmetic
  - Dedicated multiplier support
  - Cascade chain for wide-input function
  - Abundant registers/latches with clock enable, and dual synchronous/asynchronous set and reset
  - Internal 3-state bussing
  - IEEE 1149.1 boundary-scan logic
  - Die-temperature sensor diode
- Supported by Xilinx Foundation™ and Alliance Series™ Development Systems
  - Further compile time reduction of 50%
  - Internet Team Design (ITD) tool ideal for million-plus gate density designs
  - Wide selection of PC and workstation platforms
- SRAM-Based In-System Configuration
  - Unlimited re-programmability
- Advanced Packaging Options
  - 0.8 mm Chip-scale
  - 1.0 mm BGA
  - 1.27 mm BGA
  - HQ/PQ
- 0.18 µm 6-Layer Metal Process
- 100% Factory Tested

* ZBT is a trademark of Integrated Device Technology, Inc.

resources. The abundance of routing resources permits the Virtex-E family to accommodate even the largest and most complex designs.

Virtex-E FPGAs are SRAM-based, and are customized by loading configuration data into internal memory cells. Configuration data can be read from an external SPROM (master serial mode), or can be written into the FPGA (SelectMAP™, slave serial, and JTAG modes).

The standard Xilinx Foundation Series™ and Alliance Series™ Development systems deliver complete design support for Virtex-E, covering every aspect from behavioral and schematic entry, through simulation, automatic design translation and implementation, to the creation and downloading of a configuration bit stream.

Higher Performance

Virtex-E devices provide better performance than previous generations of FPGAs. Designs can achieve synchronous system clock rates up to 240 MHz including I/O or 622 Mb/s using Source Synchronous data transmission architechtures. Virtex-E I/Os comply fully with 3.3 V PCI specifications, and interfaces can be implemented that operate at 33 MHz or 66 MHz.

While performance is design-dependent, many designs operate internally at speeds in excess of 133 MHz and can achieve over 311 MHz. Table 2 shows performance data for representative circuits, using worst-case timing parameters.

Table 2: Performance for Common Circuit Functions

Function	Bits	Virtex-E (-7)
Register-to-Register		
Adder	16	4.3 ns
	64	6.3 ns
Pipelined Multiplier	8 x 8	4.4 ns
	16 x 16	5.1 ns
Address Decoder	16	3.8 ns
	64	5.5 ns
16:1 Multiplexer		4.6 ns
Parity Tree	9	3.5 ns
	18	4.3 ns
	36	5.9 ns
Chip-to-Chip		
HSTL Class IV		
LVTTL,16mA, fast slew		
LVDS		
LVPECL		

Virtex-E Device/Package Combinations and Maximum I/O

Table 3: Virtex-E Family Maximum User I/O by Device/Package (Excluding Dedicated Clock Pins)

	XCV 50E	XCV 100E	XCV 200E	XCV 300E	XCV 400E	XCV 600E	XCV 1000E	XCV 1600E	XCV 2000E	XCV 2600E	XCV 3200E
CS144	94	94	94								
PQ240	158	158	158	158	158						
HQ240						158	158				
BG352		196	260	260							
BG432				316	316	316					
BG560					404	404	404	404	404		
FG256	176	176	176	176							
FG456			284	312							
FG676					404	444					
FG680						512	512	512	512		
FG860							660	660	660		
FG900						512	660	700			
FG1156							660	724	804	804	804

Virtex-E Ordering Information

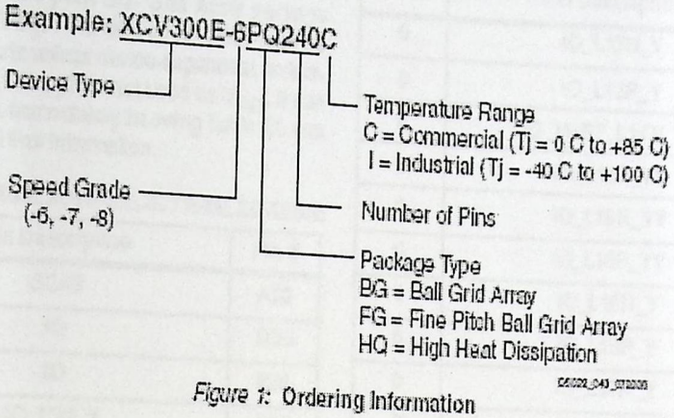


Figure 1: Ordering Information

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/7/99	1.0	Initial Xilinx release.
1/10/00	1.1	Re-released with spd.txtv. 1.18, FG860/900/1156 package information, and additional DLL, Select RAM and SelectI/O information.
1/28/00	1.2	Added Delay Measurement Methodology table, updated SelectI/O section, Figures 30, 54, & 55, text explaining Table 5, T_{BYP} values, buffered Hex Line info, p. 8, I/O Timing Measurement notes, notes for Tables 15, 16, and corrected F1156 pinout table footnote references.
2/29/00	1.3	Updated pinout tables, V_{CC} page 20, and corrected Figure 20.
5/23/00	1.4	Correction to table on p. 22.
7/10/00	1.5	• Numerous minor edits. • Data sheet upgraded to Preliminary. • Preview -8 numbers added to Virtex-E Electrical Characteristics tables.
8/1/00	1.6	• Reformatted entire document to follow new style guidelines. • Changed speed grade values in tables on pages 35-37.
9/20/00	1.7	• Min values added to Virtex-E Electrical Characteristics tables. • XCV2600E and XCV3200E numbers added to Virtex-E Electrical Characteristics tables (Module 3). • Corrected user I/O count for XCV100E device in Table 1 (Module 1). • Changed several pins to "No Connect in the XCV100E" and removed duplicate V_{CCINT} pins in Table ~ (Module 4). • Changed pin J10 to "No connect in XCV600E" in Table 74 (Module 4). • Changed pin J30 to "VREF option only in the XCV600E" in Table 74 (Module 4). • Corrected pair 18 in Table 75 (Module 4) to be "AO in the XCV1000E, XCV1600E".

FG680 Fine-Pitch Ball Grid Array Package

XCV600E, XCV1000E, XCV1600E, and XCV2000E devices in the FG680 fine-pitch Ball Grid Array package have footprint compatibility. Pins labeled IO_VREF can be used as either in all parts unless device-dependent as indicated in the footnotes. If the pin is not used as V_{REF}, it can be used as general I/O. Immediately following Table 22, see Table 23 for Differential Pair Information.

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
0	GCK3	A20
0	IO	D35
0	IO	B36
0	IO_L0N_Y	C35
0	IO_L0P_Y	A36
0	IO_VREF_L1N_Y	C34[1]
0	IO_L1P_Y	B35
0	IO_L2N_YY	C34
0	IO_L2P_YY	A35
0	IO_VREF_L3N_YY	D33
0	IO_L3P_YY	B34
0	IO_L4N	C33
0	IO_L4P	A34
0	IO_L5N_Y	D32
0	IO_L5P_Y	B33
0	IO_L6N_YY	C32
0	IO_L6P_YY	D31
0	IO_VREF_L7N_YY	A33
0	IO_L7P_YY	C31
0	IO_L8N_Y	B32
0	IO_L8P_Y	B31
0	IO_VREF_L9N_Y	A32[2]
0	IO_L9P_Y	D30
0	IO_L10N_YY	A31
0	IO_L10P_YY	C30
0	IO_VREF_L11N_YY	B30
0	IO_L11P_YY	D29
0	IO_L12N_Y	A30
0	IO_L12P_Y	C29

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
0	IO_L13N_Y	A29
0	IO_L13P_Y	B29
0	IO_VREF_L14N_YY	E28
0	IO_L14P_YY	A28
0	IO_L15N_YY	C28
0	IO_L15P_YY	B27
0	IO_L16N_Y	D27
0	IO_L16P_Y	A27
0	IO_L17N_Y	C27
0	IO_L17P_Y	B26
0	IO_L18N_YY	D26
0	IO_L18P_YY	C26
0	IO_VREF_L19N_YY	A25[1]
0	IO_L19P_YY	D25
0	IO_L20N_Y	E25
0	IO_L20P_Y	C25
0	IO_L21N_Y	A25
0	IO_L21P_Y	D24
0	IO_L22N_YY	A24
0	IO_L22P_YY	B23
0	IO_VREF_L23N_YY	C24
0	IO_L23P_YY	A23
0	IO_L24N_Y	E24
0	IO_L24P_Y	B22
0	IO_L25N_Y	E23
0	IO_L25P_Y	A22
0	IO_L26N_YY	D23
0	IO_L26P_YY	B21
0	IO_VREF_L27N_YY	C23
0	IO_L27P_YY	A21
0	IO_L28N_Y	E22
0	IO_L28P_Y	B20
0	IO_LVDS_DLL_L29N	C22
0	IO_VREF	C22[3]
1	GCK2	D21

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
1	IO	C5
1	IO_LVDS_DLL_L02P	A19
1	IO_L03N_Y	C21
1	IO_VREF_L03P_Y	B12[2]
1	IO_L31N_Y	C19
1	IO_L31P_Y	A18
1	IO_L32N_YY	D19
1	IO_VREF_L32P_YY	B19
1	IO_L33N_YY	C18
1	IO_L33P_YY	A17
1	IO_L34N_Y	D18
1	IO_L34P_Y	B17
1	IO_L35N_Y	E19
1	IO_L35P_Y	A16
1	IO_L36N_YY	C17
1	IO_VREF_L36P_YY	D17
1	IO_L37N_YY	B16
1	IO_L37P_YY	E17
1	IO_L38N_Y	A15
1	IO_L38P_Y	C16
1	IO_L39N_Y	B15
1	IO_L39P_Y	D16
1	IO_L40N_YY	A14
1	IO_VREF_L40P_YY	B14[1]
1	IO_L41N_YY	C15
1	IO_L41P_YY	A13
1	IO_L42N_Y	D15
1	IO_L42P_Y	B13
1	IO_L43N_Y	C14
1	IO_L43P_Y	A12
1	IO_L44N_YY	D14
1	IO_L44P_YY	C13
1	IO_L45N_YY	B12
1	IO_VREF_L45P_YY	D13
1	IO_L46N_Y	A11
1	IO_L46P_Y	C12

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
1	IO_L47N_Y	B11
1	IO_L47P_Y	C11
1	IO_L48N_YY	A10
1	IO_VREF_L48P_YY	D11
1	IO_L49N_YY	B10
1	IO_L49P_YY	C10
1	IO_L50N_Y	A9
1	IO_VREF_L50P_Y	C10[2]
1	IO_L51N_Y	E9
1	IO_L51P_Y	C9
1	IO_L52N_YY	A8
1	IO_VREF_L52P_YY	B8
1	IO_L53N_YY	D9
1	IO_L53P_YY	A7
1	IO_L54N_Y	C8
1	IO_L54P_Y	B7
1	IO_L55N_Y	D8
1	IO_L55P_Y	A6
1	IO_L56N_YY	C7
1	IO_VREF_L56P_YY	B6
1	IO_L57N_YY	D7
1	IO_L57P_YY	A5
1	IO_L58N_Y	C6
1	IO_VREF_L58P_Y	B5[1]
1	IO_L59N_Y	D6
1	IO_L59P_Y	A4
1	IO_WRITE_L60N_YY	B4
1	IO_CS_L60P_YY	D5
2	IO	D1
2	IO	F4
2	IO_DOUT_BUSY_L61P_YY	E3
2	IO_DIN_D0_L61N_YY	C2
2	IO_L62P_Y	D3
2	IO_L62N_Y	F3
2	IO_VREF_L63P	D2[1]

Table 22: FG680 - XCV600E, XCV1000E, XCV1500E, XCV2000E

Bank	Pin Description	Pin #
2	IO_L83N	G4
2	IO_L84P	G3
2	IO_L84N	E2
2	IO_VREF_L85P_Y	H4
2	IO_L85N_Y	E1
2	IO_L86P_YY	H3
2	IO_L86N_YY	F2
2	IO_L87P	J4
2	IO_L87N	F1
2	IO_L88P_Y	J3
2	IO_L88N_Y	G2
2	IO_VREF_L89P_YY	G1
2	IO_L89N_YY	K4
2	IO_L70P_YY	H2
2	IO_L70N_YY	K3
2	IO_VREF_L71P	H1[2]
2	IO_L71N	L4
2	IO_L72P	J2
2	IO_L72N	L3
2	IO_VREF_L73P_YY	J1
2	IO_L73N_YY	M3
2	IO_L74P_YY	K2
2	IO_L74N_YY	N4
2	IO_L75P	K1
2	IO_L75N	N3
2	IO_VREF_L76P_YY	L2
2	IO_D1_L76N_YY	P4
2	IO_D2_L77P_YY	P3
2	IO_L77N_YY	L1
2	IO_L78P_Y	R4
2	IO_L78N_Y	M2
2	IO_L79P	R3
2	IO_L79N	M1
2	IO_L80P	T4
2	IO_L80N	N2
2	IO_VREF_L81P_Y	N1[1]

Table 22: FG680 - XCV600E, XCV1000E, XCV1500E, XCV2000E

Bank	Pin Description	Pin #
2	IO_L81N_Y	T3
2	IO_L82P_YY	P2
2	IO_L82N_YY	U5
2	IO_L83P	P1
2	IO_L83N	U4
2	IO_L84P_Y	R2
2	IO_L84N_Y	U3
2	IO_VREF_L85P_YY	V5
2	IO_D3_L85N_YY	R1
2	IO_L86P_YY	W4
2	IO_L86N_YY	T2
2	IO_L87P	V3
2	IO_L87N	T1
2	IO_L88P	W4
2	IO_L88N	U2
2	IO_VREF_L89P_YY	W3
2	IO_L89N_YY	U1
2	IO_L90P_YY	AA3
2	IO_L90N_YY	V2
2	IO_VREF_L91P	AA4[2]
2	IO_L91N	V1
2	IO_L92P_YY	AB2
2	IO_L92N_YY	W2
3	IO	AF3
3	IO	AT3
3	IO	AE3
3	IO_L93P	AE4
3	IO_VREF_L93N	W1[2]
3	IO_L94P_YY	AB5
3	IO_L94N_YY	Y2
3	IO_L95P_YY	AC2
3	IO_VREF_L95N_YY	Y1
3	IO_L96P	AC3
3	IO_L96N	AA1
3	IO_L97P	AC4

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
3	IO_L97N	AA2
3	IO_L98P_YY	AC5
3	IO_L98N_YY	AB1
3	IO_D4_L99P_YY	AD3
3	IO_VREF_L99N_YY	AC1
3	IO_L100P_Y	AD1
3	IO_L100N_Y	AD4
3	IO_L101P	AD2
3	IO_L101N	AE3
3	IO_L102P_YY	AE1
3	IO_L102N_YY	AE4
3	IO_L103P_Y	AE2
3	IO_VREF_L103N_Y	AF3[1]
3	IO_L104P	AF4
3	IO_L104N	AF1
3	IO_L105P	AG3
3	IO_L105N	AF2
3	IO_L106P_Y	AG4
3	IO_L106N_Y	AG1
3	IO_L107P_YY	AH3
3	IO_D5_L107N_YY	AG2
3	IO_D6_L108P_YY	AH1
3	IO_VREF_L108N_YY	AJ2
3	IO_L109P	AH2
3	IO_L109N	AJ3
3	IO_L110P_YY	AJ1
3	IO_L110N_YY	AJ4
3	IO_L111P_YY	AK1
3	IO_VREF_L111N_YY	AK3
3	IO_L112P	AK2
3	IO_L112N	AK4
3	IO_L113P	AL1
3	IO_VREF_L113N	AL2[2]
3	IO_L114P_YY	AM1
3	IO_L114N_YY	AL3
3	IO_L115P_YY	AM2

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
3	IO_VREF_L115N_YY	AL4
3	IO_L116P_Y	AH3
3	IO_L116N_Y	AH1
3	IO_L117P	AH4
3	IO_L117N	AP1
3	IO_L118P_YY	AM2
3	IO_L118N_YY	AP2
3	IO_L119P_Y	AN3
3	IO_VREF_L119N_Y	AR1
3	IO_L120P	AN4
3	IO_L120N	AT1
3	IO_L121P	AR2
3	IO_VREF_L121N	AP4[1]
3	IO_L122P_Y	AT2
3	IO_L122N_Y	AR3
3	IO_D7_L123P_YY	AR4
3	IO_INIT_L123N_YY	AU2
4	GCK0	AW10
4	IO	AW3
4	IO_L124P_YY	AU4
4	IO_L124N_YY	AW5
4	IO_L125P_Y	AT6
4	IO_L125N_Y	AV4
4	IO_VREF_L126P_Y	AU6[1]
4	IO_L126N_Y	AW4
4	IO_L127P_YY	AT7
4	IO_L127N_YY	AW5
4	IO_VREF_L128P_YY	AU7
4	IO_L128N_YY	AW6
4	IO_L129P_Y	AT8
4	IO_L129N_Y	AW8
4	IO_L130P_Y	AU8
4	IO_L130N_Y	AV7
4	IO_L131P_YY	AT9
4	IO_L131N_YY	AW7

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
4	IO_VREF_L132P_YY	AW8
4	IO_L132N_YY	AU9
4	IO_L133P_Y	AW9
4	IO_L133N_Y	AT10
4	IO_VREF_L134P_Y	AU8[3]
4	IO_L134N_Y	AU10
4	IO_L135P_YY	AW7
4	IO_L135N_YY	AT11
4	IO_VREF_L136P_YY	AW10
4	IO_L136N_YY	AU11
4	IO_L137P_Y	AW10
4	IO_L137N_Y	AU12
4	IO_L138P_Y	AW11
4	IO_L138N_Y	AT13
4	IO_VREF_L139P_YY	AW11
4	IO_L139N_YY	AU13
4	IO_L140P_YY	AT14
4	IO_L140N_YY	AW12
4	IO_L141P_Y	AU14
4	IO_L141N_Y	AW12
4	IO_L142P_Y	AT15
4	IO_L142N_Y	AW13
4	IO_L143P_YY	AU15
4	IO_L143N_YY	AW13
4	IO_VREF_L144P_YY	AW14[1]
4	IO_L144N_YY	AT16
4	IO_L145P_Y	AW14
4	IO_L145N_Y	AU16
4	IO_L146P_Y	AW15
4	IO_L146N_Y	AR17
4	IO_L147P_YY	AW15
4	IO_L147N_YY	AT17
4	IO_VREF_L148P_YY	AU17
4	IO_L148N_YY	AW16
4	IO_L149P_Y	AR18
4	IO_L149N_Y	AW16

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
4	IO_L150P_Y	AT16
4	IO_L150N_Y	AW17
4	IO_L151P_YY	AU18
4	IO_L151N_YY	AW17
4	IO_VREF_L152P_YY	AT19
4	IO_L152N_YY	AW18
4	IO_L153P_Y	AU19
4	IO_L153N_Y	AW18
4	IO_VREF_L154P	AU21[2]
4	IO_L154N	AW19
4	IO_LVDS_DLL_L155P	AT21
5	GCK1	AU22
5	IO	AT34
5	IO	AW20
5	IO_LVDS_DLL_L156N	AT22
5	IO_VREF_L156P_Y	AW22[2]
5	IO_L156N_Y	AR22
5	IO_L157P_YY	AW23
5	IO_VREF_L157N_YY	AW21
5	IO_L158P_YY	AU23
5	IO_L158N_YY	AW21
5	IO_L159P_Y	AT23
5	IO_L159N_Y	AW22
5	IO_L160P_Y	AR23
5	IO_L160N_Y	AW22
5	IO_L161P_YY	AW24
5	IO_VREF_L161N_YY	AW23
5	IO_L162P_YY	AW24
5	IO_L162N_YY	AU24
5	IO_L163P_Y	AW25
5	IO_L163N_Y	AT24
5	IO_L164P_Y	AW25
5	IO_L164N_Y	AU25
5	IO_L165P_YY	AW26
5	IO_VREF_L165N_YY	AT25[1]

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
5	IO_L166P_YY	AH26
5	IO_L166N_YY	AH27
5	IO_L167P_Y	AU28
5	IO_L167N_Y	AH27
5	IO_L168P_Y	AT26
5	IO_L168N_Y	AH28
5	IO_L169P_YY	AU27
5	IO_L169N_YY	AH26
5	IO_L170P_YY	AH29
5	IO_VREF_L170N_YY	AT27
5	IO_L171P_Y	AH30
5	IO_L171N_Y	AU29
5	IO_L172P_Y	AH30
5	IO_L172N_Y	AH29
5	IO_L173P_YY	AH31
5	IO_VREF_L173N_YY	AU29
5	IO_L174P_YY	AH31
5	IO_L174N_YY	AT29
5	IO_L175P_Y	AH32
5	IO_VREF_L175N_Y	AU30[2]
5	IO_L176P_Y	AH33
5	IO_L176N_Y	AT30
5	IO_L177P_YY	AH33
5	IO_VREF_L177N_YY	AU31
5	IO_L178P_YY	AT31
5	IO_L178N_YY	AH34
5	IO_L179P_Y	AH32
5	IO_L179N_Y	AH34
5	IO_L180P_Y	AU32
5	IO_L180N_Y	AH35
5	IO_L181P_YY	AT32
5	IO_VREF_L181N_YY	AH35
5	IO_L182P_YY	AU33
5	IO_L182N_YY	AH36
5	IO_L183P_Y	AT33
5	IO_VREF_L183N_Y	AH37[1]

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
5	IO_L184P_Y	AU34
5	IO_L184N_Y	AU35
6	IO	W20
6	IO	AR37
6	IO	AR32
6	IO_L185N_YY	AR33
6	IO_L185P_YY	AT36
6	IO_L186N_Y	AR35
6	IO_L186P_Y	AP35
6	IO_VREF_L187N	AT32[1]
6	IO_L187P	AP37
6	IO_L188N	AP33
6	IO_L188P	AP32
6	IO_VREF_L189N_Y	AN35
6	IO_L189P_Y	AN33
6	IO_L190N_YY	AN37
6	IO_L190P_YY	AN32
6	IO_L191N	AM36
6	IO_L191P	AM35
6	IO_L192N_Y	AM37
6	IO_L192P_Y	AL26
6	IO_VREF_L193N_YY	AM30
6	IO_L193P_YY	AL27
6	IO_L194N_YY	AL28
6	IO_L194P_YY	AK35
6	IO_VREF_L195N	AL32[2]
6	IO_L195P	AK37
6	IO_L196N	AK33
6	IO_L196P	AJ36
6	IO_VREF_L197N_YY	AK32
6	IO_L197P_YY	AJ37
6	IO_L198N_YY	AJ33
6	IO_L198P_YY	AH37
6	IO_L199N	AJ30
6	IO_L199P	AH33

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
6	IO_VREF_L202N_YY	AH32
6	IO_L202P_YY	AG33
6	IO_L201N_YY	AG35
6	IO_L201P_YY	AG32
6	IO_L200N_Y	AG37
6	IO_L200P_Y	AF30
6	IO_L203N	AF36
6	IO_L203P	AE33
6	IO_L204N	AF27
6	IO_L204P	AF28
6	IO_VREF_L205N_Y	AE30[1]
6	IO_L205P_Y	AE35
6	IO_L206N_YY	AD33
6	IO_L206P_YY	AE37
6	IO_L207N	AD32
6	IO_L207P	AD35
6	IO_L208N_Y	AC33
6	IO_L208P_Y	AC32
6	IO_VREF_L209N_YY	AD37
6	IO_L209P_YY	AB33
6	IO_L210N_YY	AC35
6	IO_L210P_YY	AB32
6	IO_L211N	AC36
6	IO_L211P	AA33
6	IO_L212N	AC37
6	IO_L212P	AA32
6	IO_VREF_L213N_YY	AB35
6	IO_L213P_YY	Y30
6	IO_L214N_YY	AB36
6	IO_L214P_YY	Y29
6	IO_VREF_L215N	AB37[2]
6	IO_L215P	AA36
7	IO	C33
7	IO	B37
7	IO	F37

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
7	IO_L216N_YY	AA37
7	IO_L216P_YY	W29
7	IO_L217N	W27
7	IO_VREF_L217P	W32[2]
7	IO_L218N_YY	W26
7	IO_L218P_YY	U30
7	IO_L219N_YY	V28
7	IO_VREF_L219P_YY	U33
7	IO_L220N	V27
7	IO_L220P	T29
7	IO_L221N	V26
7	IO_L221P	T28
7	IO_L222N_YY	V25
7	IO_L222P_YY	R30
7	IO_L223N_YY	U27
7	IO_VREF_L223P_YY	U28
7	IO_L224N_Y	R33
7	IO_L224P_Y	U25
7	IO_L225N	P30
7	IO_L225P	T27
7	IO_L226N_YY	P28
7	IO_L226P_YY	T26
7	IO_L227N_Y	N30
7	IO_VREF_L227P_Y	N36[1]
7	IO_L228N	R27
7	IO_L228P	N32
7	IO_L229N	R26
7	IO_L229P	N33
7	IO_L230N_Y	P27
7	IO_L230P_Y	L30
7	IO_L231N_YY	P26
7	IO_L231P_YY	N27
7	IO_L232N_YY	L36
7	IO_VREF_L232P_YY	N28
7	IO_L233N	K30
7	IO_L233P	N37

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
7	IO_L234N_YY	K38
7	IO_L234P_YY	L37
7	IO_L235N_YY	J39
7	IO_VREF_L235P_YY	L36
7	IO_L236N	J38
7	IO_L236P	K37
7	IO_L237N	H39
7	IO_VREF_L237P	K36[2]
7	IO_L238N_YY	H38
7	IO_L238P_YY	J37
7	IO_L239N_YY	G39
7	IO_VREF_L239P_YY	G38
7	IO_L240N_Y	J36
7	IO_L240P_Y	F39
7	IO_L241N	H37
7	IO_L241P	F38
7	IO_L242N_YY	H36
7	IO_L242P_YY	E39
7	IO_L243N_Y	G37
7	IO_VREF_L243P_Y	E38
7	IO_L244N	G36
7	IO_L244P	D39
7	IO_L245N	D38
7	IO_VREF_L245P	F36[1]
7	IO_L246N_Y	D37
7	IO_L246P_Y	E37
2	CCLK	E4
3	DONE	AU3
NA	DXN	AM37
NA	DXP	AU36
NA	M0	AT37
NA	M1	AU38
NA	M2	AT35
NA	PROGRAM	AT5
NA	TCK	C36

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
NA	TDI	B3
2	TDO	C4
NA	TMS	E36
NA	VCCINT	E8
NA	VCCINT	E9
NA	VCCINT	E15
NA	VCCINT	E16
NA	VCCINT	E24
NA	VCCINT	E25
NA	VCCINT	E31
NA	VCCINT	E32
NA	VCCINT	H5
NA	VCCINT	H35
NA	VCCINT	J5
NA	VCCINT	J35
NA	VCCINT	R5
NA	VCCINT	R35
NA	VCCINT	T5
NA	VCCINT	T35
NA	VCCINT	AD5
NA	VCCINT	AD35
NA	VCCINT	AE5
NA	VCCINT	AE35
NA	VCCINT	AL5
NA	VCCINT	AL35
NA	VCCINT	AM5
NA	VCCINT	AM35
NA	VCCINT	AR9
NA	VCCINT	AR2
NA	VCCINT	AR15
NA	VCCINT	AR16
NA	VCCINT	AR24
NA	VCCINT	AR25
NA	VCCINT	AR31
NA	VCCINT	AR32

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
0	VCCO	E34
0	VCCO	E33
0	VCCO	E30
0	VCCO	E29
0	VCCO	E27
0	VCCO	E26
1	VCCO	E10
1	VCCO	E11
1	VCCO	E13
1	VCCO	E14
1	VCCO	E6
1	VCCO	E7
2	VCCO	P5
2	VCCO	N5
2	VCCO	L5
2	VCCO	K5
2	VCCO	G5
2	VCCO	F5
3	VCCO	AP5
3	VCCO	AN5
3	VCCO	AK5
3	VCCO	AJ5
3	VCCO	AG5
3	VCCO	AF5
4	VCCO	AR10
4	VCCO	AR11
4	VCCO	AR13
4	VCCO	AR14
4	VCCO	AR6
4	VCCO	AR7
5	VCCO	AR34
5	VCCO	AR33
5	VCCO	AR30
5	VCCO	AR29
5	VCCO	AR27

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
5	VCCO	AR26
6	VCCO	AP35
6	VCCO	AN35
6	VCCO	AK35
6	VCCO	AJ35
6	VCCO	AG35
6	VCCO	AF35
7	VCCO	P35
7	VCCO	N35
7	VCCO	L35
7	VCCO	K35
7	VCCO	G35
7	VCCO	F35
NA	GND	Y5
NA	GND	Y4
NA	GND	Y27
NA	GND	Y26
NA	GND	Y25
NA	GND	Y3
NA	GND	W5
NA	GND	W35
NA	GND	M5
NA	GND	M4
NA	GND	M33
NA	GND	M32
NA	GND	E5
NA	GND	E35
NA	GND	E28
NA	GND	E21
NA	GND	E20
NA	GND	E19
NA	GND	E12
NA	GND	D4
NA	GND	D33
NA	GND	D28

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
NA	GND	D29
NA	GND	D12
NA	GND	C32
NA	GND	C27
NA	GND	C3
NA	GND	C29
NA	GND	C1
NA	GND	B30
NA	GND	B28
NA	GND	E2
NA	GND	B1
NA	GND	AW32
NA	GND	AW28
NA	GND	AW27
NA	GND	AW2
NA	GND	AW32
NA	GND	AW1
NA	GND	AV30
NA	GND	AV28
NA	GND	AV2
NA	GND	AV1
NA	GND	AU32
NA	GND	AU27
NA	GND	AU3
NA	GND	AU29
NA	GND	AU1
NA	GND	AT4
NA	GND	AT30
NA	GND	AT26
NA	GND	AT20
NA	GND	AT12
NA	GND	AR5
NA	GND	AR30
NA	GND	AR28
NA	GND	AR21
NA	GND	AR20

Table 22: FG680 - XCV600E, XCV1000E, XCV1600E, XCV2000E

Bank	Pin Description	Pin #
NA	GND	AR19
NA	GND	AR12
NA	GND	AH5
NA	GND	AH4
NA	GND	AH30
NA	GND	AH26
NA	GND	AA5
NA	GND	AA30
NA	GND	A29
NA	GND	A28
NA	GND	A27
NA	GND	A3
NA	GND	A2
NA	GND	A1

Notes:

1. V_{REF} or I/O option only in the XCV1000E, 1600E, 2000E; otherwise, I/O option only.
2. V_{REF} or I/O option only in the XCV1600E, 2000E; otherwise, I/O option only.
3. V_{REF} or I/O option only in the XCV2000E; otherwise, I/O option only.

Appendix B

Verilog source code for the system

Appendix B

Verilog source code for the system

```
/******************ENCRYPTION- COMPRESSION UNIT**********************/

module ecu(data ,mode ,Clr,dataout);
  input [415:0] data;          // input data
  output [159:0] dataout;      //output data
  input [159:0] datain;
  input Clr,mode;              //input controls
  wire [159:0] dataout,datain;
    wire [415:0] data;
    wire [127:0] yb,p;         // Used to generate encryption key
  integer count,o,i,j,t,f,c,round;


/***Variables needed for compression***/
 reg [155:0] string;
reg[4:0]out [0:31];
reg [3:0] char;
reg [4:0] oldcode,newcode;
reg[127:0]dictionary [0:31];
reg [3:0] outs [0:31];
reg [159:0] reg1,data;


/***Variables needed for encryption***/
reg [127:0] xa,ya,g,g1,key;
reg [127:0] key1;
reg [31:0] k[0:3],s[3:0],s1[3:0];
reg [31:0] w [0:43];
reg[31:0] temp,rcon;
reg [7:0] mul2[0:255],mul3[0:255],sbox [0:255],invsbox [0:255],mule[0:255],mul9[0:255],
mulb[0:255],muld[0:255];
```

```
/********************** The beginning of the code*****************************/
always @(datain)
    if (!Clr) begin
/***************** Initialization of SBox************* *********/
sbox[0]='h63;sbox[1]='h7c;sbox[2]='h77;sbox[3]='h7b;sbox[4]='hf2;sbox[5]='h6b;
sbox[6]='h6f;sbox[7]='hc5;sbox[8]='h30;sbox[9]='h01;sbox[10]='h67;sbox[11]='h2b;
sbox[12]='hfe;sbox[13]='hd7;sbox[14]='hab;sbox[15]='h76;sbox[16]='hca;sbox[17]='h82;
sbox[18]='hc9;sbox[19]='h7d;sbox[20]='hfa;sbox[21]='h59;sbox[22]='h47;sbox[23]='hf0;
sbox[24]='had;sbox[25]='hd4;sbox[26]='ha2;sbox[27]='haf;sbox[28]='h9c;sbox[29]='ha4;
sbox[30]='h72;sbox[31]='hc0;sbox[32]='hb7;sbox[33]='hfd;sbox[34]='h93;sbox[35]='h26;
sbox[36]='h36;sbox[37]='h3f;sbox[38]='hf7;sbox[39]='hcc;sbox[40]='h34;sbox[41]='ha5;
sbox[42]='he5;sbox[43]='hf1;sbox[44]='h71;sbox[45]='hd8;sbox[46]='h31;sbox[47]='h15;
sbox[48]='h04;sbox[49]='hc7;sbox[50]='h23;sbox[51]='hc3;sbox[52]='h18;sbox[53]='h96;
sbox[54]='h05;sbox[55]='h9a;sbox[56]='h07;sbox[57]='h12;sbox[58]='h80;sbox[59]='he2;
sbox[60]='heb;sbox[61]='h27;sbox[62]='hb2;sbox[63]='h75;sbox[64]='h09;sbox[65]='h83;
sbox[66]='h2c;sbox[67]='h1a;sbox[68]='h1b;sbox[69]='h6e;sbox[70]='h5a;sbox[71]='ha0;
sbox[72]='h52;sbox[73]='h3b;sbox[74]='hd6;sbox[75]='hb3;sbox[76]='h29;sbox[77]='he3;
sbox[78]='h2f;sbox[79]='h84;sbox[80]='h53;sbox[81]='hd1;sbox[82]='h00;sbox[83]='hed;
sbox[84]='h20;sbox[85]='hfc;sbox[86]='hb1;sbox[87]='h5b;sbox[88]='h6a;sbox[89]='hcb;
sbox[90]='hbe;sbox[91]='h39;sbox[92]='h4a;sbox[93]='h4c;sbox[94]='h58;sbox[95]='hcf;
sbox[96]='hd0;sbox[97]='hef;sbox[98]='haa;sbox[99]='hfb;sbox[100]='h43;sbox[101]='h4d;
sbox[102]='h33;sbox[103]='h85;sbox[104]='h45;sbox[105]='hf9;sbox[106]='h02;
sbox[107]='h7f;sbox[108]='h50;sbox[109]='h3c;sbox[110]='h9f;sbox[111]='ha8;
sbox[112]='h51;sbox[113]='ha3;sbox[114]='h40;sbox[115]='h8f;sbox[116]='h92;
sbox[117]='h9d;sbox[118]='h38;sbox[119]='hf5;sbox[120]='hbc;sbox[121]='hb6;
sbox[122]='hda;sbox[123]='h21;sbox[124]='h10;sbox[125]='hff;sbox[126]='hf3;
sbox[127]='hd2;sbox[128]='hcd;sbox[129]='h0c;sbox[130]='h13;sbox[131]='hec;
sbox[132]='h5f;sbox[133]='h97;sbox[134]='h44;sbox[135]='h17;sbox[136]='hc4;
sbox[137]='ha7;sbox[138]='h7e;sbox[139]='h3d;sbox[140]='h64;sbox[141]='h5d;
sbox[142]='h19;sbox[143]='h73;sbox[144]='h60;sbox[145]='h81;sbox[146]='h4f;
sbox[147]='hdc;sbox[148]='h22;sbox[149]='h2a;sbox[150]='h90;sbox[151]='h88;
sbox[152]='h46;sbox[153]='hee;sbox[154]='hb8;sbox[155]='h14;sbox[156]='hde;
sbox[157]='h5e;sbox[158]='h0b;sbox[159]='hdb;sbox[160]='he0;sbox[161]='h32;
sbox[162]='h3a;sbox[163]='h0a;sbox[164]='h49;sbox[165]='h06;sbox[166]='h24;
sbox[167]='h5c;sbox[168]='hc2;sbox[169]='hd3;sbox[170]='hac;sbox[171]='h62;
sbox[172]='h91;sbox[173]='h95;sbox[174]='he4;sbox[175]='h79;sbox[176]='he7;
sbox[177]='hc8;sbox[178]='h37;sbox[179]='h6d;sbox[180]='h8d;sbox[181]='hd5;
```

```
sbox[182]='h4e;sbox[183]='ha9;sbox[184]='h6c;sbox[185]='h56;sbox[186]='hf4;
sbox[187]='hea;sbox[188]='h65;sbox[189]='h7a;sbox[190]='hae;sbox[191]='h08;
sbox[192]='hba;sbox[193]='h78;sbox[194]='h25;sbox[195]='h2e;sbox[196]='h1c;
sbox[197]='ha6;sbox[198]='hb4;sbox[199]='hc6;sbox[200]='he8;sbox[201]='hdd;
sbox[202]='h74;sbox[203]='h1f;sbox[204]='h4b;sbox[205]='hbd;sbox[206]='h8b;
sbox[207]='h8a;sbox[208]='h70;sbox[209]='h3e;sbox[210]='hb5;sbox[211]='h66;
sbox[212]='h48;sbox[213]='h03;sbox[214]='hf6;sbox[215]='h0e;sbox[216]='h61;
sbox[217]='h35;sbox[218]='h57;sbox[219]='hb9;sbox[220]='h86;sbox[221]='hc1;
sbox[222]='h1d;sbox[223]='h9e;sbox[224]='he1;sbox[225]='hf8;sbox[226]='h98;
sbox[227]='h11;sbox[228]='h69;sbox[229]='hd9;sbox[230]='h8e;sbox[231]='h94;
sbox[232]='h9b;sbox[233]='h1e;sbox[234]='h87;sbox[235]='he9;sbox[236]='hce;
sbox[237]='h55;sbox[238]='h28;sbox[239]='hdf;sbox[240]='h8c;sbox[241]='ha1;
sbox[242]='h89;sbox[243]='h0d;sbox[244]='hbf;sbox[245]='he6;sbox[246]='h42;
sbox[247]='h68;sbox[248]='h41;sbox[249]='h99;sbox[250]='h2d;sbox[251]='h0f;
sbox[252]='hb0;sbox[253]='h54;sbox[254]='hbb;sbox[255]='h16;

/***************** Initialization of mul2********** *********/
mul2[0]='h00;mul2[1]='h02;mul2[2]='h04;mul2[3]='h06;mul2[4]='h08;mul2[5]='h0A;
mul2[6]='h0C;mul2[7]='h0E;mul2[8]='h10;mul2[9]='h12;mul2[10]='h14;mul2[11]='h16;
mul2[12]='h18;mul2[13]='h1A;mul2[14]='h1C;mul2[15]='h1E;mul2[16]='h20;
mul2[17]='h22;mul2[18]='h24;mul2[19]='h26;mul2[20]='h28;mul2[21]='h2A;
mul2[22]='h2C;mul2[23]='h2E;mul2[24]='h30;mul2[25]='h32;mul2[26]='h34;
mul2[27]='h36;mul2[28]='h38;mul2[29]='h3A;mul2[30]='h3C;mul2[31]='h3E;
mul2[32]='h40;mul2[33]='h42;mul2[34]='h44;mul2[35]='h46;mul2[36]='h48;
mul2[37]='h4A;mul2[38]='h4C;mul2[39]='h4E;mul2[40]='h50;mul2[41]='h52;
mul2[42]='h54;mul2[43]='h56;mul2[44]='h58;mul2[45]='h5A;mul2[46]='h5C;
mul2[47]='h5E;mul2[48]='h60;mul2[49]='h62;mul2[50]='h64;mul2[51]='h66;
mul2[52]='h68;mul2[53]='h6A;mul2[54]='h6C;mul2[55]='h6E;mul2[56]='h70;
mul2[57]='h72;mul2[58]='h74;mul2[59]='h76;mul2[60]='h78;mul2[61]='h7A;
mul2[62]='h7C;mul2[63]='h7E;mul2[64]='h80;mul2[65]='h82;mul2[66]='h84;
mul2[67]='h86;mul2[68]='h88;mul2[69]='h8A;mul2[70]='h8C;mul2[71] = 'h8E;
mul2[72]='h90;mul2[73]='h92;mul2[74]='h94;mul2[75]='h96;mul2[76]='h98;
mul2[77]='h9A;mul2[78]='h9C;mul2[79]='h9E;mul2[80]='hA0;mul2[81]='hA2;
mul2[82]='hA4;mul2[83]='hA6;mul2[84]='hA8;mul2[85]='hAA;mul2[86]='hAC;
mul2[87]='hAE;mul2[88]='hB0;mul2[89]='hB2;mul2[90]='hB4;mul2[91]='hB6;
mul2[92]='hB8;mul2[93]='hBA;mul2[94]='hBC;mul2[95]='hBE;mul2[96] = 'hC0;
mul2[97]='hC2;mul2[98]='hC4;mul2[99]='hC6;mul2[100]='hC8;mul2[101]='hCA;
```

```
mul2[102]='hCC;mul2[103]='hCE;mul2[104]='hD0;mul2[105]='hD2;mul2[106]='hD4;
mul2[107]='hD6;mul2[108]='hD8;mul2[109]='hDA;mul2[110]='hDC;mul2[111]='hDE;
mul2[112]='hE0;mul2[113]='hE2;mul2[114]='hE4;mul2[115]='hE6;mul2[116]='hE8;
mul2[117]='hEA;mul2[118]='hEC;mul2[119]='hEE;mul2[120]='hF0;mul2[121]='hF2;
mul2[122]='hF4;mul2[123]='hF6;mul2[124]='hF8;mul2[125]='hFA;mul2[126]='hFC;
mul2[127]='hFE;mul2[128]='h1B;mul2[129]='h19;mul2[130]='h1F;mul2[131]='h1D;
mul2[132]='h13;mul2[133]='h11;mul2[134]='h17;mul2[135]='h15;mul2[136]='h0B;
mul2[137]='h09;mul2[138]='h0F;mul2[139]='h0D;mul2[140]='h03;mul2[141]='h01;
mul2[142]='h07;mul2[143]='h05;mul2[144]='h3B;mul2[145]='h39;mul2[146]='h3F;
mul2[147]='h3D;mul2[148]='h33;mul2[149]='h31;mul2[150]='h37;mul2[151]='h35;
mul2[152]='h2B;mul2[153]='h29;mul2[154]='h2F;mul2[155]='h2D;mul2[156]='h23;
mul2[157]='h21;mul2[158]='h27;mul2[159]='h25;mul2[160]='h5B;mul2[161]='h59;
mul2[162]='h5F;mul2[163]='h5D;mul2[164]='h53;mul2[165]='h51;mul2[166]='h57;
mul2[167]='h55;mul2[168]='h4B;mul2[169]='h49;mul2[170]='h4F;mul2[171]='h4D;
mul2[172]='h43;mul2[173]='h41;mul2[174]='h47;mul2[175]='h45;mul2[176]='h7B;
mul2[177]='h79;mul2[178]='h7F;mul2[179]='h7D;mul2[180]='h73;mul2[181]='h71;
mul2[182]='h77;mul2[183]='h75;mul2[184]='h6B;mul2[185]='h69;mul2[186]='h6F;
mul2[187]='h6D;mul2[188]='h63;mul2[189]='h61;mul2[190]='h67;mul2[191]='h65;
mul2[192]='h9B;mul2[193]='h99;mul2[194]='h9F;mul2[195]='h9D;mul2[196]='h93;
mul2[197]='h91;mul2[198]='h97;mul2[199]='h95;mul2[200]='h8B;mul2[201]='h89;
mul2[202]='h8F;mul2[203]='h8D;mul2[204]='h83;mul2[205]='h81;mul2[206]='h87;
mul2[207]='h85;mul2[208]='hBB;mul2[209]='hB9;mul2[210]='hBF;mul2[211]='hBD;
mul2[212]='hB3;mul2[213]='hB1;mul2[214]='hB7;mul2[215]='hB5;mul2[216]='hAB;
mul2[217]='hA9;mul2[218]='hAF;mul2[219]='hAD;mul2[220]='hA3;mul2[221]='hA1;
mul2[222]='hA7;mul2[223]='hA5;mul2[224]='hDB;mul2[225]='hD9;mul2[226]='hDF;
mul2[227]='hDD;mul2[228]='hD3;mul2[229]='hD1;mul2[230]='hD7;mul2[231]='hD5;
mul2[232]='hCB;mul2[233]='hC9;mul2[234]='hCF;mul2[235]='hCD;mul2[236]='hC3;
mul2[237]='hC1;mul2[238]='hC7;mul2[239]='hC5;mul2[240]='hFB;mul2[241]='hF9;
mul2[242]='hFF;mul2[243]='hFD;mul2[244]='hF3;mul2[245]='hF1;mul2[246]='hF7;
mul2[247]='hF5;mul2[248]='hEB;mul2[249]='hE9;mul2[250]='hEF;mul2[251]='hED;
mul2[252]='hE3;mul2[253]='hE1;mul2[254]='hE7;mul2[255]='hE5;
```

```
/***************** Initialization of mul3************ *********/
mul3[0]= 'h00;mul3[1]='h03;mul3[2]='h06;mul3[3]='h05;mul3[4]='h0C;mul3[5]='h0F;
mul3[6]='h0A;mul3[7]='h09;mul3[8]='h18;mul3[9]='h1B;mul3[10]='h1E;mul3[11]='h1D;
mul3[12]='h14;mul3[13]='h17;mul3[14]='h12;mul3[15]='h11;mul3[16]='h30;mul3[17]='h33;m
ul3[18]='h36;mul3[19]='h35;mul3[20]='h3C;mul3[21]='h3F;mul3[22]='h3A;
mul3[23]='h39;mul3[24]='h28;mul3[25]='h2B;mul3[26]='h2E;mul3[27]='h2D;
mul3[28]='h24;mul3[29]='h27;mul3[30]='h22;mul3[31]='h21;mul3[32]='h60;
mul3[33]='h63;mul3[34]='h66;mul3[35]='h65;mul3[36]='h6C;mul3[37]='h6F;
mul3[38]='h6A;mul3[39]='h69;mul3[40]='h78;mul3[41]='h7B;mul3[42]='h7E;
mul3[43]='h7D;mul3[44]='h74;mul3[45]='h77;mul3[46]='h72;mul3[47]='h71;
mul3[48]='h50;mul3[49]='h53;mul3[50]='h56;mul3[51]='h55;mul3[52]='h5C;
mul3[53]='h5F;mul3[54]='h5A;mul3[55]='h59;mul3[56]='h48;mul3[57]='h4B;
mul3[58]='h4E;mul3[59]='h4D;mul3[60]='h44;mul3[61]='h47;mul3[62]='h42;
mul3[63]='h41;mul3[64]='hC0;mul3[65]='hC3;mul3[66]='hC6;mul3[67]='hC5;
mul3[68]='hCC;mul3[69]='hCF;mul3[70]='hCA;mul3[71]='hC9;mul3[72]='hD8;
mul3[73]='hDB;mul3[74]='hDE;mul3[75]='hDD;mul3[76]='hD4;mul3[77]='hD7;
mul3[78]='hD2;mul3[079]='hD1;mul3[80]='hF0;mul3[81]='hF3;mul3[82]='hF6;
mul3[83]='hF5;mul3[84]='hFC;mul3[85]='hFF;mul3[86]='hFA;mul3[87]='hF9;
mul3[88]='hE8;mul3[89]='hEB;mul3[90]='hEE;mul3[91]='hED;mul3[92]='hE4;
mul3[93]='hE7;mul3[94]='hE2;mul3[95]='hE1;mul3[96]='hA0;mul3[97]='hA3;
mul3[98]='hA6;mul3[99]='hA5;mul3[100]='hAC;mul3[101]='hAF;mul3[102]='hAA;
mul3[103]='hA9;mul3[104]='hB8;mul3[105]='hBB;mul3[106]='hBE;mul3[107]='hBD;
mul3[108]='hB4;mul3[109]='hB7;mul3[110]='hB2;mul3[111]='hB1;mul3[112]='h90;
mul3[113]='h93;mul3[114]='h96;mul3[115]='h95;mul3[116]='h9C;mul3[117]='h9F;
mul3[118]='h9A;mul3[119]='h99;mul3[120]='h88;mul3[121]='h8B;mul3[122]='h8E;
mul3[123]='h8D;mul3[124]='h84;mul3[125]='h87;mul3[126]='h82;mul3[127]='h81;
mul3[128]='h9B;mul3[129]='h98;mul3[130]='h9D;mul3[131]='h9E;mul3[132]='h97;
mul3[133]='h94;mul3[134]='h91;mul3[135]='h92;mul3[136]='h83;mul3[137]='h80;
mul3[138]='h85;mul3[139]='h86;mul3[140]='h8F;mul3[141]='h8C;mul3[142]='h89;
mul3[143]='h8A;mul3[144]='hAB;mul3[145]='hA8;mul3[146]='hAD;mul3[147]='hAE;
mul3[148]='hA7;mul3[149]='hA4;mul3[150]='hA1;mul3[151]='hA2;mul3[152]='hB3;
mul3[153]='hB0;mul3[154]='hB5;mul3[155]='hB6;mul3[156]='hBF;mul3[157]='hBC;
mul3[158]='hB9;mul3[159]='hBA;mul3[160]='hFB;mul3[161]='hF8;mul3[162]='hFD;
mul3[163]='hFE;mul3[164]='hF7;mul3[165]='hF4;mul3[166]='hF1;mul3[167]='hF2;
mul3[168]='hE3;mul3[169]='hE0;mul3[170]='hE5;mul3[171]='hE6;mul3[172]='hEF;
mul3[173]='hEC;mul3[174]='hE9;mul3[175]='hEA;mul3[176]='hCB;mul3[177]='hC8;
mul3[178]='hCD;mul3[179]='hCE;mul3[180]='hC7;mul3[181]='hC4;mul3[182]='hC1;
```

```
/***************** Initialization of mul3************* *********/
mul3[0]= 'h00;mul3[1]='h03;mul3[2]='h06;mul3[3]='h05;mul3[4]='h0C;mul3[5]='h0F;
mul3[6]='h0A;mul3[7]='h09;mul3[8]='h18;mul3[9]='h1B;mul3[10]='h1E;mul3[11]='h1D;
mul3[12]='h14;mul3[13]='h17;mul3[14]='h12;mul3[15]='h11;mul3[16]='h30;mul3[17]='h33;m
ul3[18]='h36;mul3[19]='h35;mul3[20]='h3C;mul3[21]='h3F;mul3[22]='h3A;
mul3[23]='h39;mul3[24]='h28;mul3[25]='h2B;mul3[26]='h2E;mul3[27]='h2D;
mul3[28]='h24;mul3[29]='h27;mul3[30]='h22;mul3[31]='h21;mul3[32]='h60;
mul3[33]='h63;mul3[34]='h66;mul3[35]='h65;mul3[36]='h6C;mul3[37]='h6F;
mul3[38]='h6A;mul3[39]='h69;mul3[40]='h78;mul3[41]='h7B;mul3[42]='h7E;
mul3[43]='h7D;mul3[44]='h74;mul3[45]='h77;mul3[46]='h72;mul3[47]='h71;
mul3[48]='h50;mul3[49]='h53;mul3[50]='h56;mul3[51]='h55;mul3[52]='h5C;
mul3[53]='h5F;mul3[54]='h5A;mul3[55]='h59;mul3[56]='h48;mul3[57]='h4B;
mul3[58]='h4E;mul3[59]='h4D;mul3[60]='h44;mul3[61]='h47;mul3[62]='h42;
mul3[63]='h41;mul3[64]='hC0;mul3[65]='hC3;mul3[66]='hC6;mul3[67]='hC5;
mul3[68]='hCC;mul3[69]='hCF;mul3[70]='hCA;mul3[71]='hC9;mul3[72]='hD8;
mul3[73]='hDB;mul3[74]='hDE;mul3[75]='hDD;mul3[76]='hD4;mul3[77]='hD7;
mul3[78]='hD2;mul3[079]='hD1;mul3[80]='hF0;mul3[81]='hF3;mul3[82]='hF6;
mul3[83]='hF5;mul3[84]='hFC;mul3[85]='hFF;mul3[86]='hFA;mul3[87]='hF9;
mul3[88]='hE8;mul3[89]='hEB;mul3[90]='hEE;mul3[91]='hED;mul3[92]='hE4;
mul3[93]='hE7;mul3[94]='hE2;mul3[95]='hE1;mul3[96]='hA0;mul3[97]='hA3;
mul3[98]='hA6;mul3[99]='hA5;mul3[100]='hAC;mul3[101]='hAF;mul3[102]='hAA;
mul3[103]='hA9;mul3[104]='hB8;mul3[105]='hBB;mul3[106]='hBE;mul3[107]='hBD;
mul3[108]='hB4;mul3[109]='hB7;mul3[110]='hB2;mul3[111]='hB1;mul3[112]='h90;
mul3[113]='h93;mul3[114]='h96;mul3[115]='h95;mul3[116]='h9C;mul3[117]='h9F;
mul3[118]='h9A;mul3[119]='h99;mul3[120]='h88;mul3[121]='h8B;mul3[122]='h8E;
mul3[123]='h8D;mul3[124]='h84;mul3[125]='h87;mul3[126]='h82;mul3[127]='h81;
mul3[128]='h9B;mul3[129]='h98;mul3[130]='h9D;mul3[131]='h9E;mul3[132]='h97;
mul3[133]='h94;mul3[134]='h91;mul3[135]='h92;mul3[136]='h83;mul3[137]='h80;
mul3[138]='h85;mul3[139]='h86;mul3[140]='h8F;mul3[141]='h8C;mul3[142]='h89;
mul3[143]='h8A;mul3[144]='hAB;mul3[145]='hA8;mul3[146]='hAD;mul3[147]='hAE;
mul3[148]='hA7;mul3[149]='hA4;mul3[150]='hA1;mul3[151]='hA2;mul3[152]='hB3;
mul3[153]='hB0;mul3[154]='hB5;mul3[155]='hB6;mul3[156]='hBF;mul3[157]='hBC;
mul3[158]='hB9;mul3[159]='hBA;mul3[160]='hFB;mul3[161]='hF8;mul3[162]='hFD;
mul3[163]='hFE;mul3[164]='hF7;mul3[165]='hF4;mul3[166]='hF1;mul3[167]='hF2;
mul3[168]='hE3;mul3[169]='hE0;mul3[170]='hE5;mul3[171]='hE6;mul3[172]='hEF;
mul3[173]='hEC;mul3[174]='hE9;mul3[175]='hEA;mul3[176]='hCB;mul3[177]='hC8;
mul3[178]='hCD;mul3[179]='hCE;mul3[180]='hC7;mul3[181]='hC4;mul3[182]='hC1;
```

```
mul3[183]='hC2;mul3[184]='hD3;mul3[185]='hD0;mul3[186]='hD5;mul3[187]='hD6;
mul3[188]='hDF;mul3[189]='hDC;mul3[190]='hD9;mul3[191]='hDA;mul3[192]='h5B;
mul3[193]='h58;mul3[194]='h5D;mul3[195]='h5E;mul3[196]='h57;mul3[197]='h54;
mul3[198]='h51;mul3[199]='h52;mul3[200]='h43;mul3[201]='h40;mul3[202]='h45;
mul3[203]='h46;mul3[204]='h4F;mul3[205]='h4C;mul3[206]='h49;mul3[207]='h4A;
mul3[208]='h6B;mul3[209]='h68;mul3[210]='h6D;mul3[211]='h6E;mul3[212]='h67;
mul3[213]='h64;mul3[214]='h61;mul3[215]='h62;mul3[216]='h73;mul3[217]='h70;
mul3[218]='h75;mul3[219]='h76;mul3[220]='h7F;mul3[221]='h7C;mul3[222]='h79;
mul3[223]='h7A;mul3[224]='h3B;mul3[225]='h38;mul3[226]='h3D;mul3[227]='h3E;
mul3[228]='h37;mul3[229]='h34;mul3[230]='h31;mul3[231]='h32;mul3[232]='h23;
mul3[233]='h20;mul3[234]='h25;mul3[235]='h26;mul3[236]='h2F;mul3[237]='h2C;
mul3[238]='h29;mul3[239]='h2A;mul3[240]='h0B;mul3[241]='h08;mul3[242]='h0D;
mul3[243]='h0E;mul3[244]='h07;mul3[245]='h04;mul3[246]='h01;mul3[247]='h02;
mul3[248]='h13;mul3[249]='h10;mul3[250]='h15;mul3[251]='h16;mul3[252]='h1F;
mul3[253]='h1C;mul3[254]='h19;mul3[255]='h1A;


/****************** Initialization of inverse SBox************ *********/
invsbox[0]='h52;invsbox[1]='h09;invsbox[2]='h6a;invsbox[3]='hd5;invsbox[4]='h30;
invsbox[5]='h36;invsbox[6]='ha5;invsbox[7]='h38;invsbox[8]='hbf;invsbox[9]='h40;
invsbox[10]='ha3;invsbox[11]='h9e;invsbox[12]='h81;invsbox[13]='hf3;invsbox[14]='hd7;
invsbox[15]='hfb;invsbox[16]='h7c;invsbox[17]='he3;invsbox[18]='h39;invsbox[19]='h82;
invsbox[20]='h9b;invsbox[21]='h2f;invsbox[22]='hff;invsbox[23]='h87;invsbox[24]='h34;
invsbox[25]='h8e;invsbox[26]='h43;invsbox[27]='h44;invsbox[28]='hc4;invsbox[29]='hde;
invsbox[30]='he9;invsbox[31]='hcb;invsbox[32]='h54;invsbox[33]='h7b;invsbox[34]='h94;
invsbox[35]='h32;invsbox[36]='ha6;invsbox[37]='hc2;invsbox[38]='h23;invsbox[39]='h3d;
invsbox[40]='hee;invsbox[41]='h4c;invsbox[42]='h95;invsbox[43]='h0b;invsbox[44]='h42;
invsbox[45]='hfa;invsbox[46]='hc3;invsbox[47]='h4e;invsbox[48]='h08;invsbox[49]='h2e;
invsbox[50]='ha1;invsbox[51]='h66;invsbox[52]='h28;invsbox[53]='hd9;invsbox[54]='h24;
invsbox[55]='hb2;invsbox[56]='h76;invsbox[57]='h5b;invsbox[58]='ha2;invsbox[59]='h49;
invsbox[60]='h6d;invsbox[61]='h8b;invsbox[62]='hd1;invsbox[63]='h25;invsbox[64]='h72;
invsbox[65]='hf8;invsbox[66]='hf6;invsbox[67]='h64;invsbox[68]='h86;invsbox[69]='h68;
invsbox[70]='h98;invsbox[71]='h16;invsbox[72]='hd4;invsbox[73]='ha4;invsbox[74]='h5c;
invsbox[75]='hcc;invsbox[76]='h5d;invsbox[77]='h65;invsbox[78]='hb6;invsbox[79]='h92;
invsbox[80]='h6c;invsbox[81]='h70;invsbox[82]='h48;invsbox[83]='h50;invsbox[84]='hfd;
invsbox[85]='hed;invsbox[86]='hb9;invsbox[87]='hda;invsbox[88]='h5e;invsbox[89]='h15;
invsbox[90]='h46;invsbox[91]='h57;invsbox[92]='ha7;invsbox[93]='h8d;invsbox[94]='h9d;
```

invsbox[95]='h84;invsbox[96]='h90;invsbox[97]='hd8;invsbox[98]='hab;invsbox[99]='h00;
invsbox[100]='h8c;invsbox[101]='hbc;invsbox[102]='hd3;invsbox[103]='h0a;
invsbox[104]='hf7;invsbox[105]='he4;invsbox[106]='h58;invsbox[107]='h05;
invsbox[108]='hb8;invsbox[109]='hb3;invsbox[110]='h45;invsbox[111]='h06;
invsbox[112]='hd0;invsbox[113]='h2c;invsbox[114]='h1e;invsbox[115]='h8f;
invsbox[116]='hca;invsbox[117]='h3f;invsbox[118]='h0f;invsbox[119]='h02;
invsbox[120]='hc1;invsbox[121]='haf;invsbox[122]='hbd;invsbox[123]='h03;
invsbox[124]='h01;invsbox[125]='h13;invsbox[126]='h8a;invsbox[127]='h6b;
invsbox[128]='h3a;invsbox[129]='h91;invsbox[130]='h11;invsbox[131]='h41;
invsbox[132]='h4f;invsbox[133]='h67;invsbox[134]='hdc;invsbox[135]='hea;
invsbox[136]='h97;invsbox[137]='hf2;invsbox[138]='hcf;invsbox[139]='hce;
invsbox[140]='hf0;invsbox[141]='hb4;invsbox[142]='he6;invsbox[143]='h73;
invsbox[144]='h96;invsbox[145]='hac;invsbox[146]='h74;invsbox[147]='h22;
invsbox[148]='he7;invsbox[149]='had;invsbox[150]='h35;invsbox[151]='h85;
invsbox[152]='he2;invsbox[153]='hf9;invsbox[154]='h37;invsbox[155]='he8;
invsbox[156]='h1c;invsbox[157]='h75;invsbox[158]='hdf;invsbox[159]='h6e;
invsbox[160]='h47;invsbox[161]='hf1;invsbox[162]='h1a;invsbox[163]='h71;
invsbox[164]='h1d;invsbox[165]='h29;invsbox[166]='hc5;invsbox[167]='h89;
invsbox[168]='h6f;invsbox[169]='hb7;invsbox[170]='h62;invsbox[171]='h0e;
invsbox[172]='haa;invsbox[173]='h18;invsbox[174]='hbe;invsbox[175]='h1b;
invsbox[176]='hfc;invsbox[177]='h56;invsbox[178]='h3e;invsbox[179]='h4b;
invsbox[180]='hc6;invsbox[181]='hd2;invsbox[182]='h79;invsbox[183]='h20;
invsbox[184]='h9a;invsbox[185]='hdb;invsbox[186]='hc0;invsbox[187]='hfe;
invsbox[188]='h78;invsbox[189]='hcd;invsbox[190]='h5a;invsbox[191]='hf4;
invsbox[192]='h1f;invsbox[193]='hdd;invsbox[194]='ha8;invsbox[195]='h33;
invsbox[196]='h88;invsbox[197]='h07;invsbox[198]='hc7;invsbox[199]='h31;
invsbox[200]='hb1;invsbox[201]='h12;invsbox[202]='h10;invsbox[203]='h59;
invsbox[204]='h27;invsbox[205]='h80;invsbox[206]='hec;invsbox[207]='h5f;
invsbox[208]='h60;invsbox[209]='h51;invsbox[210]='h7f;invsbox[211]='ha9;
invsbox[212]='h19;invsbox[213]='hb5;invsbox[214]='h4a;invsbox[215]='h0d;
invsbox[216]='h2d;invsbox[217]='he5;invsbox[218]='h7a;invsbox[219]='h9f;
invsbox[220]='h93;invsbox[221]='hc9;invsbox[222]='h9c;invsbox[223]='hef;
invsbox[224]='ha0;invsbox[225]='he0;invsbox[226]='h3b;invsbox[227]='h4d;
invsbox[228]='hae;invsbox[229]='h2a;invsbox[230]='hf5;invsbox[231]='hb0;
invsbox[232]='hc8;invsbox[233]='heb;invsbox[234]='hbb;invsbox[235]='h3c;
invsbox[236]='h83;invsbox[237]='h53;invsbox[238]='h99;invsbox[239]='h61;
invsbox[240]='h17;invsbox[241]='h2b;invsbox[242]='h04;invsbox[243]='h7e;

```
invsbox[244]='hba;invsbox[245]='h77;invsbox[246]='hd6;invsbox[247]='h26;
invsbox[248]='he1;invsbox[249]='h69;invsbox[250]='h14;invsbox[251]='h63;
invsbox[252]='h55;invsbox[253]='h21;invsbox[254]='h0c;invsbox[255]='h7d;


/***************** Initialization of mul9********** *********/
mul9[0]='h00;mul9[1]='h09;mul9[2]='h12;mul9[3]='h1B;mul9[4]='h24;mul9[5]='h2D;
mul9[6]='h36;mul9[7]='h3F;mul9[8]='h48;mul9[9]='h41;mul9[10]='h5A;mul9[11]='h53;
mul9[12]='h6C;mul9[13]='h65;mul9[14]='h7E;mul9[15]='h77;mul9[016]='h90;
mul9[17]='h99;mul9[18]='h82;mul9[19]='h8B;mul9[20]='hB4;mul9[21]='hBD;
mul9[22]='hA6;mul9[23]='hAF;mul9[24]='hD8;mul9[25]='hD1;mul9[26]='hCA;
mul9[27]='hC3;mul9[28]='hFC;mul9[29]='hF5;mul9[30]='hEE;mul9[31]='hE7;
mul9[32]='h3B;mul9[33]='h32;mul9[34]='h29;mul9[35]='h20;mul9[36]='h1F;
mul9[37]='h16;mul9[38]='h0D;mul9[39]='h04;mul9[40]='h73;mul9[41]='h7A;
mul9[42]='h61;mul9[43]='h68;mul9[44]='h57;mul9[45]='h5E;mul9[46]='h45;
mul9[47]='h4C;mul9[48]='hAB;mul9[49]='hA2;mul9[50]='hB9;mul9[51]='hB0;
mul9[52]='h8F;mul9[53]='h86;mul9[54]='h9D;mul9[55]='h94;mul9[56]='hE3;
mul9[57]='hEA;mul9[58]='hF1;mul9[59]='hF8;mul9[60]='hC7;mul9[61]='hCE;
mul9[62]='hD5;mul9[63]='hDC;mul9[64]='h76;mul9[65]='h7F;mul9[66]='h64;
mul9[67]='h6D;mul9[68]='h52;mul9[69]='h5B;mul9[70]='h40;mul9[71]='h49;
mul9[72]='h3E;mul9[73]='h37;mul9[74]='h2C;mul9[75]='h25;mul9[76]='h1A;
mul9[77]='h13;mul9[78]='h08;mul9[79]='h01;mul9[80]='hE6;mul9[81]='hEF;
mul9[82]='hF4;mul9[83]='hFD;mul9[84]='hC2;mul9[85]='hCB;mul9[86]='hD0;
mul9[87]='hD9;mul9[88]='hAE;mul9[89]='hA7;mul9[90]='hBC;mul9[91]='hB5;
mul9[92]='h8A;mul9[93]='h83;mul9[94]='h98;mul9[95]='h91;mul9[96]='h4D;
mul9[97]='h44;mul9[98]='h5F;mul9[99]='h56;mul9[100]='h69;mul9[101]='h60;
mul9[102]='h7B;mul9[103]='h72;mul9[104]='h05;mul9[105]='h0C;mul9[106]='h17;
mul9[107]='h1E;mul9[108]='h21;mul9[109]='h28;mul9[110]='h33;mul9[111]='h3A;
mul9[112]='hDD;mul9[113]='hD4;mul9[114]='hCF;mul9[115]='hC6;mul9[116]='hF9;
mul9[117]='hF0;mul9[118]='hEB;mul9[119]='hE2;mul9[120]='h95;mul9[121]='h9C;
mul9[122]='h87;mul9[123]='h8E;mul9[124]='hB1;mul9[125]='hB8;mul9[126]='hA3;
mul9[127]='hAA;mul9[128]='hEC;mul9[129]='hE5;mul9[130]='hFE;mul9[131]='hF7;
mul9[132]='hC8;mul9[133]='hC1;mul9[134]='hDA;mul9[135]='hD3;mul9[136]='hA4;
mul9[137]='hAD;mul9[138]='hB6;mul9[139]='hBF;mul9[140]='h80;mul9[141]='h89;
mul9[142]='h92;mul9[143]='h9B;mul9[144]='h7C;mul9[145]='h75;mul9[146]='h6E;
mul9[147]='h67;mul9[148]='h58;mul9[149]='h51;mul9[150]='h4A;mul9[151]='h43;
mul9[152]='h34;mul9[153]='h3D;mul9[154]='h26;mul9[155]='h2F;mul9[156]='h10;
```

```
mul9[157]='h19;mul9[158]='h02;mul9[159]='h0B;mul9[160]='hD7;mul9[161]='hDE;
mul9[162]='hC5;mul9[163]='hCC;mul9[164]='hF3;mul9[165]='hFA;mul9[166]='hE1;
mul9[167]='hE8;mul9[168]='h9F;mul9[169]='h96;mul9[170]='h8D;mul9[171]='h84;
mul9[172]='hBB;mul9[173]='hB2;mul9[174]='hA9;mul9[175]='hA0;mul9[176]='h47;
mul9[177]='h4E;mul9[178]='h55;mul9[179]='h5C;mul9[180]='h63;mul9[181]='h6A;
mul9[182]='h71;mul9[183]='h78;mul9[184]='h0F;mul9[185]='h06;mul9[186]='h1D;
mul9[187]='h14;mul9[188]='h2B;mul9[189]='h22;mul9[190]='h39;mul9[191]='h30;
mul9[192]='h9A;mul9[193]='h93;mul9[194]='h88;mul9[195]='h81;mul9[196]='hBE;
mul9[197]='hB7;mul9[198]='hAC;mul9[199]='hA5;mul9[200]='hD2;mul9[201]='hDB;
mul9[202]='hC0;mul9[203]='hC9;mul9[204]='hF6;mul9[205]='hFF;mul9[206]='hE4;
mul9[207]='hED;mul9[208]='h0A;mul9[209]='h03;mul9[210]='h18;mul9[211]='h11;
mul9[212]='h2E;mul9[213]='h27;mul9[214]='h3C;mul9[215]='h35;mul9[216]='h42;
mul9[217]='h4B;mul9[218]='h50;mul9[219]='h59;mul9[220]='h66;mul9[221]='h6F;
mul9[222]='h74;mul9[223]='h7D;mul9[224]='hA1;mul9[225]='hA8;mul9[226]='hB3;
mul9[227]='hBA;mul9[228]='h85;mul9[229]='h8C;mul9[230]='h97;mul9[231]='h9E;
mul9[232]='hE9;mul9[233]='hE0;mul9[234]='hFB;mul9[235]='hF2;mul9[236]='hCD;
mul9[237]='hC4;mul9[238]='hDF;mul9[239]='hD6;mul9[240]='h31;mul9[241]='h38;
mul9[242]='h23;mul9[243]='h2A;mul9[244]='h15;mul9[245]='h1C;mul9[246]='h07;
mul9[247]='h0E;mul9[248]='h79;mul9[249]='h70;mul9[250]='h6B;mul9[251]='h62;
mul9[252]='h5D;mul9[253]='h54;mul9[254]='h4F;mul9[255]='h46;


/***************** Initialization of mulb************* *********/
mulb[0]='h00;mulb[1]='h0B;mulb[2]='h16;mulb[3]='h1D;mulb[4]='h2C;mulb[5]='h27;
mulb[6]='h3A;mulb[7]='h31;mulb[8]='h58;mulb[9]='h53;mulb[10]='h4E;mulb[11]='h45;
mulb[12]='h74;mulb[13]='h7F;mulb[14]='h62;mulb[15]='h69;mulb[16]='hB0;
mulb[17]='hBB;mulb[18]='hA6;mulb[19]='hAD;mulb[20]='h9C;mulb[21]='h97;
mulb[22]='h8A;mulb[23]='h81;mulb[24]='hE8;mulb[25]='hE3;mulb[26]='hFE;
mulb[27]='hF5;mulb[28]='hC4;mulb[29]='hCF;mulb[30]='hD2;mulb[31]='hD9;
mulb[32]='h7B;mulb[33]='h70;mulb[34]='h6D;mulb[35]='h66;mulb[036]= 'h57;
mulb[37]='h5C;mulb[38]='h41;mulb[39]='h4A;mulb[40]='h23;mulb[41]='h28;
mulb[42]='h35;mulb[43]='h3E;mulb[44]='h0F;mulb[45]='h04;mulb[46]='h19;
mulb[47]='h12;mulb[48]='hCB;mulb[49]='hC0;mulb[50]='hDD;mulb[51]='hD6;
mulb[52]='hE7;mulb[53]='hEC;mulb[54]='hF1;mulb[55]='hFA;mulb[56]='h93;
mulb[57]='h98;mulb[58]='h85;mulb[59]='h8E;mulb[60]='hBF;mulb[61]='hB4;
mulb[62]='hA9;mulb[63]='hA2;mulb[64]='hF6;mulb[65]='hFD;mulb[66]='hE0;
mulb[67]='hEB;mulb[68]='hDA;mulb[69]='hD1;mulb[70]='hCC;mulb[71]='hC7;
```

mulb[72]='hAE;mulb[73]='hA5;mulb[74]='hB8;mulb[75]='hB3;mulb[76]='h82;
mulb[77]='h89;mulb[78]='h94;mulb[79]='h9F;mulb[80]='h46;mulb[81]='h4D;
mulb[82]='h50;mulb[83]='h5B;mulb[84]='h6A;mulb[85]='h61;mulb[86]='h7C;
mulb[87]='h77;mulb[88]='h1E;mulb[89]='h15;mulb[90]='h08;mulb[91]='h03;
mulb[92]='h32;mulb[93]='h39;mulb[94]='h24;mulb[95]='h2F;mulb[96]='h8D;
mulb[97]='h86;mulb[98]='h9B;mulb[99]='h90;mulb[100]='hA1;mulb[101]='hAA;
mulb[102]='hB7;mulb[103]='hBC;mulb[104]='hD5;mulb[105]='hDE;mulb[106]='hC3;
mulb[107]='hC8,mulb[108]='hF9;mulb[109]='hF2;mulb[110]='hEF;mulb[111]='hE4;
mulb[112]='h3D;mulb[113]='h36;mulb[114]='h2B;mulb[115]='h20;mulb[116]='h11;
mulb[117]='h1A;mulb[118]='h07;mulb[119]='h0C;mulb[120]='h65;mulb[121]='h6E;
mulb[122]='h73;mulb[123]='h78;mulb[124]='h49;mulb[125]='h42;mulb[126]='h5F;
mulb[127]='h54;mulb[128]='hF7;mulb[129]='hFC;mulb[130]='hE1;mulb[131]='hEA;
mulb[132]='hDB;mulb[133]='hD0;mulb[134]='hCD;mulb[135]='hC6;mulb[136]='hAF;
mulb[137]='hA4;mulb[138]='hB9;mulb[139]='hB2;mulb[140]='h83;mulb[141]='h88;
mulb[142]='h95;mulb[143]='h9E;mulb[144]='h47;mulb[145]='h4C;mulb[146]='h51;
mulb[147]='h5A;mulb[148]='h6B;mulb[149]='h60;mulb[150]='h7D;mulb[151]='h76;
mulb[152]='h1F;mulb[153]='h14;mulb[154]='h09;mulb[155]='h02;mulb[156]='h33;
mulb[157]='h38;mulb[158]='h25;mulb[159]='h2E;mulb[160]='h8C;mulb[161]='h87;
mulb[162]='h9A;mulb[163]='h91;mulb[164]='hA0;mulb[165]='hAB;mulb[166]='hB6;
mulb[167]='hBD;mulb[168]='hD4;mulb[169]='hDF;mulb[170]='hC2;mulb[171]='hC9;
mulb[172]='hF8;mulb[173]='hF3;mulb[174]='hEE;mulb[175]='hE5;mulb[176]='h3C;
mulb[177]='h37;mulb[178]='h2A;mulb[179]='h21;mulb[180]='h10;mulb[181]='h1B;
mulb[182]='h06;mulb[183]='h0D;mulb[184]='h64;mulb[185]='h6F;mulb[186]='h72;
mulb[187]='h79;mulb[188]='h48;mulb[189]='h43;mulb[190]='h5E;mulb[191]='h55;
mulb[192]='h01;mulb[193]='h0A;mulb[194]='h17;mulb[195]='h1C;mulb[196]='h2D;
mulb[197]='h26;mulb[198]='h3B;mulb[199]='h30;mulb[200]='h59;mulb[201]='h52;
mulb[202]='h4F;mulb[203]='h44;mulb[204]='h75;mulb[205]='h7E;mulb[206]='h63;
mulb[207]='h68;mulb[208]='hB1;mulb[209]='hBA;mulb[210]='hA7;mulb[211]='hAC;
mulb[212]='h9D;mulb[213]='h96;mulb[214]='h8B;mulb[215]='h80;mulb[216]='hE9;
mulb[217]='hE2;mulb[218]='hFF;mulb[219]='hF4;mulb[220]='hC5;mulb[221]='hCE;
mulb[222]='hD3;mulb[223]='hD8;mulb[224]='h7A;mulb[225]='h71;mulb[226]='h6C;
mulb[227]='h67;mulb[228]='h56;mulb[229]='h5D;mulb[230]='h40;mulb[231]='h4B;
mulb[232]='h22;mulb[233]='h29;mulb[234]='h34;mulb[235]='h3F;mulb[236]='h0E;
mulb[237]='h05;mulb[238]='h18;mulb[239]='h13;mulb[240]='hCA;mulb[241]='hC1;
mulb[242]='hDC;mulb[243]='hD7;mulb[244]='hE6;mulb[245]='hED;mulb[246]='hF0;
mulb[247]='hFB;mulb[248]='h92;mulb[249]='h99;mulb[250]='h84;mulb[251]='h8F;
mulb[252]='hBE;mulb[253]='hB5;mulb[254]='hA8;mulb[255]= 'hA3;

```
/****************** Initialization of muld**********************/
muld[0]='h00;muld[1]='h0D;muld[2]='h1A;muld[3]='h17;muld[4]='h34;muld[5]='h39;
muld[6]='h2E;muld[7]='h23;muld[8]='h68;muld[9]='h65;muld[10]='h72;muld[11]='h7F;
muld[12]='h5C;muld[13]='h51;muld[14]='h46;muld[15]='h4B;muld[16]='hD0;
muld[17]='hDD;muld[18]='hCA;muld[19]='hC7;muld[20]='hE4;muld[21]='hE9;
muld[22]='hFE;muld[23]='hF3;muld[24]='hB8;muld[25]='hB5;muld[26]='hA2;
muld[27]='hAF;muld[28]='h8C;muld[29]='h81;muld[30]='h96;muld[31]='h9B;
muld[32]='hBB;muld[33]='hB6;muld[34]='hA1;muld[35]='hAC;muld[36]='h8F;
muld[37]='h82;muld[38]='h95;muld[39]='h98;muld[40]='hD3;muld[41]='hDE;
muld[42]='hC9;muld[43]='hC4;muld[44]='hE7;muld[45]= 'hEA;muld[46]='hFD;
muld[47]='hF0;muld[48]='h6B;muld[49]='h66;muld[50]='h71;muld[51]='h7C;
muld[52]='h5F;muld[53]='h52;muld[54]='h45;muld[55]='h48;muld[56]='h03;
muld[57]='h0E;muld[58]='h19;muld[59]='h14;muld[60]='h37;muld[61]='h3A;
muld[62]='h2D;muld[63]='h20;muld[64]='h6D;muld[65]='h60;muld[66]='h77;
muld[67]='h7A;muld[68]='h59;muld[69]='h54;muld[70]='h43;muld[71]= 'h4E;
muld[72]='h05;muld[73]='h08;muld[74]='h1F;muld[75]='h12;muld[76]='h31;
muld[77]='h3C;muld[78]='h2B;muld[79]='h26;muld[80]='hBD;muld[81]='hB0;
muld[82]='hA7;muld[83]='hAA;muld[84]='h89;muld[85]='h84;muld[86]='h93;
muld[87]='h9E;muld[88]='hD5;muld[89]='hD8;muld[90]='hCF;muld[91]='hC2;
muld[92]='hE1;muld[93]='hEC;muld[94]='hFB;muld[95]='hF6;muld[96]='hD6;
muld[97]='hDB;muld[98]='hCC;muld[99]='hC1;muld[100]='hE2;muld[101]='hEF;
muld[102]='hF8;muld[103]='hF5;muld[104]='hBE;muld[105]='hB3;muld[106]='hA4;
muld[107]='hA9;muld[108]='h8A;muld[109]='h87;muld[110]='h90;muld[111]='h9D;
muld[112]='h06;muld[113]='h0B;muld[114]='h1C;muld[115]='h11;muld[116]='h32;
muld[117]='h3F;muld[118]='h28;muld[119]='h25;muld[120]='h6E;muld[121]='h63;
muld[122]='h74;muld[123]='h79;muld[124]='h5A;muld[125]='h57;muld[126]='h40;
muld[127]='h4D;muld[128]='hDA;muld[129]='hD7;muld[130]='hC0;muld[131]='hCD;
muld[132]='hEE;muld[133]='hE3;muld[134]='hF4;muld[135]='hF9;muld[136]='hB2;
muld[137]='hBF;muld[138]='hA8;muld[139]='hA5;muld[140]='h86;muld[141]='h8B;
muld[142]='h9C;muld[143]='h91;muld[144]='h0A;muld[145]='h07;muld[146]='h10;
muld[147]='h1D;muld[148]='h3E;muld[149]='h33;muld[150]='h24;muld[151]='h29;
muld[152]='h62;muld[153]='h6F;muld[154]='h78;muld[155]='h75;muld[156]='h56;
muld[157]='h5B;muld[158]='h4C;muld[159]='h41;muld[160]='h61;muld[161]='h6C;
muld[162]='h7B;muld[163]='h76;muld[164]='h55;muld[165]='h58;muld[166]='h4F;
muld[167]='h42;muld[168]='h09;muld[169]='h04;muld[170]='h13;muld[171]='h1E;
muld[172]='h3D;muld[173]='h30;muld[174]='h27;muld[175]='h2A;muld[176]='hB1;
muld[177]='hBC;muld[178]='hAB;muld[179]='hA6;muld[180]='h85;muld[181]='h88;
```

```
muld[182]='h9F;muld[183]='h92;muld[184]='hD9;muld[185]='hD4;muld[186]='hC3;
muld[187]='hCE;muld[188]='hED;muld[189]='hE0;muld[190]='hF7;muld[191]='hFA;
muld[192]='hB7;muld[193]='hBA;muld[194]='hAD;muld[195]='hA0;muld[196]='h83;
muld[197]='h8E;muld[198]='h99;muld[199]='h94;muld[200]='hDF;muld[201]='hD2;
muld[202]='hC5;muld[203]='hC8;muld[204]='hEB;muld[205]='hE6;muld[206]='hF1;
muld[207]='hFC;muld[208]='h67;muld[209]='h6A;muld[210]='h7D;muld[211]='h70;
muld[212]='h53;muld[213]='h5E;muld[214]='h49;muld[215]='h44;muld[216]='h0F;
muld[217]='h02;muld[218]='h15;muld[219]='h18;muld[220]='h3B;muld[221]='h36;
muld[222]='h21;muld[223]='h2C;muld[224]='h0C;muld[225]='h01;muld[226]='h16;
muld[227]='h1B;muld[228]='h38;muld[229]='h35;muld[230]='h22;muld[231]='h2F;
muld[232]='h64;muld[233]='h69;muld[234]='h7E;muld[235]='h73;muld[236]='h50;
muld[237]='h5D;muld[238]='h4A;muld[239]='h47;muld[240]='hDC;muld[241]='hD1;
muld[242]='hC6;muld[243]='hCB;muld[244]='hE8;muld[245]='hE5;muld[246]='hF2;
muld[247]='hFF;muld[248]='hB4;muld[249]='hB9;muld[250]='hAE;muld[251]='hA3;
muld[252]='h80;muld[253]='h8D;muld[254]='h9A;muld[255]='h97;

/***************** Initialization of mule**********************/
mule[0]='h00;mule[1]='h0E;mule[2]='h1C;mule[3]='h12;mule[4]='h38;mule[5]='h36;
mule[6]='h24;mule[7]='h2A;mule[8]='h70;mule[9]='h7E;mule[10]='h6C;mule[11]='h62;
mule[12]='h48;mule[13]='h46;mule[14]='h54;mule[15]='h5A;mule[16]='hE0;
mule[17]='hEE;mule[18]='hFC;mule[19]='hF2;mule[20]='hD8;mule[21]='hD6;
mule[22]='hC4;mule[23]='hCA;mule[24]='h90;mule[25]='h9E;mule[26]='h8C;
mule[27]='h82;mule[28]='hA8;mule[29]='hA6;mule[30]='hB4;mule[31]='hBA;
mule[32]='hDB;mule[33]='hD5;mule[34]='hC7;mule[35]='hC9;mule[36]='hE3;
mule[37]='hED;mule[38]='hFF;mule[39]='hF1;mule[40]='hAB;mule[41]='hA5;
mule[42]='hB7;mule[43]='hB9;mule[44]='h93;mule[45]='h9D;mule[46]= 'h8F;
mule[47]='h81;mule[48]='h3B;mule[49]='h35;mule[50]='h27;mule[51]='h29;
mule[52]='h03;mule[53]='h0D;mule[54]='h1F;mule[55]='h11;mule[56]='h4B;
mule[57]='h45;mule[58]='h57;mule[59]='h59;mule[60]='h73;mule[61]='h7D;
mule[62]='h6F;mule[63]='h61;mule[64]='hAD;mule[65]='hA3;mule[66]='hB1;
mule[67]='hBF;mule[68]='h95;mule[69]='h9B;mule[70]='h89;mule[71]='h87;
mule[72]='hDD;mule[73]='hD3;mule[74]='hC1;mule[75]='hCF;mule[76]='hE5;
mule[77]='hEB;mule[78]='hF9;mule[79]='hF7;mule[80]='h4D;mule[81]='h43;
mule[82]='h51;mule[83]='h5F;mule[84]='h75;mule[85]='h7B;mule[86]='h69;
mule[87]='h67;mule[88]='h3D;mule[89]='h33;mule[90]='h21;mule[91]='h2F;
mule[92]='h05;mule[93]='h0B;mule[94]='h19;mule[95]='h17;mule[96]='h76;
mule[97]='h78;mule[98]='h6A;mule[99]='h64;mule[100]='h4E;mule[101]='h40;
```

mule[102]='h52;mule[103]='h5C;mule[104]='h06;mule[105]='h08;mule[106]='h1A;
mule[107]='h14;mule[108]='h3E;mule[109]='h30;mule[110]='h22;mule[111]='h2C;
mule[112]='h96;mule[113]='h98;mule[114]='h8A;mule[115]='h84;mule[116]='hAE;
mule[117]='hA0;mule[118]='hB2;mule[119]='hBC;mule[120]='hE6;mule[121]='hE8;
mule[122]='hFA;mule[123]='hF4;mule[124]='hDE;mule[125]='hD0;mule[126]='hC2;
mule[127]='hCC;mule[128]='h41;mule[129]='h4F;mule[130]='h5D;mule[131]='h53;
mule[132]='h79;mule[133]='h77;mule[134]='h65;mule[135]='h6B;mule[136]='h31;
mule[137]='h3F;mule[138]='h2D;mule[139]='h23;mule[140]='h09;mule[141]='h07;
mule[142]='h15;mule[143]='h1B;mule[144]='hA1;mule[145]='hAF;mule[146]='hBD;
mule[147]='hB3;mule[148]='h99;mule[149]='h97;mule[150]='h85;mule[151]='h8B;
mule[152]='hD1;mule[153]='hDF;mule[154]='hCD;mule[155]='hC3;mule[156]='hE9;
mule[157]= 'hE7;mule[158]='hF5;mule[159]='hFB;mule[160]='h9A;mule[161]='h94;
mule[162]='h86;mule[163]='h88;mule[164]='hA2;mule[165]='hAC;mule[166]='hBE;
mule[167]='hB0;mule[168]='hEA;mule[169]='hE4;mule[170]='hF6;mule[171]='hF8;
mule[172]='hD2;mule[173]='hDC;mule[174]='hCE;mule[175]='hC0;mule[176]='h7A;
mule[177]='h74;mule[178]='h66;mule[179]='h68;mule[180]='h42;mule[181]='h4C;
mule[182]='h5E;mule[183]='h50;mule[184]='h0A;mule[185]='h04;mule[186]='h16;
mule[187]='h18;mule[188]='h32;mule[189]='h3C;mule[190]='h2E;mule[191]='h20;
mule[192]='hEC;mule[193]='hE2;mule[194]='hF0;mule[195]='hFE;mule[196]='hD4;
mule[197]='hDA;mule[198]='hC8;mule[199]='hC6;mule[200]='h9C;mule[201]='h92;
mule[202]='h80;mule[203]='h8E;mule[204]='hA4;mule[205]='hAA;mule[206]='hB8;
mule[207]='hB6;mule[208]='h0C;mule[209]='h02;mule[210]='h10;mule[211]='h1E;
mule[212]='h34;mule[213]='h3A;mule[214]='h28;mule[215]='h26;mule[216]='h7C;
mule[217]='h72;mule[218]='h60;mule[219]='h6E;mule[220]='h44;mule[221]='h4A;
mule[222]='h58;mule[223]='h56;mule[224]='h37;mule[225]='h39;mule[226]='h2B;
mule[227]='h25;mule[228]='h0F;mule[229]='h01;mule[230]='h13;mule[231]='h1D;
mule[232]='h47;mule[233]='h49;mule[234]='h5B;mule[235]='h55;mule[236]='h7F;
mule[237]='h71;mule[238]='h63;mule[239]='h6D;mule[240]='hD7;mule[241]='hD9;
mule[242]='hCB;mule[243]='hC5;mule[244]='hEF;mule[245]='hE1;mule[246]='hF3;
mule[247]='hFD;mule[248]='hA7;mule[249]='hA9;mule[250]='hBB;mule[251]='hB5;
mule[252]='h9F;mule[253]='h91;mule[254]='h83;mule[255]='h8D;

```
rcon=32'h01000000;     // Constant used for encryption
    t=0;
datain={data[159],data[158],data[157],data[156],data[155],data[154],data[153],data[152],
data[151],data[150],data[149],data[148],data[147],data[146],data[145],data[144],data[143],
data[142],data[141],data[140],data[139],data[138],data[137],data[136],data[135],data[134],
data[133],data[132],data[131],data[130],data[129],data[128],data[127],data[126],data[125],
data[124],data[123],data[122],data[121],data[120],data[119],data[118],data[117],data[116],
data[115],data[114],data[113],data[112],data[111],data[110],data[109],data[108],data[107],
data[106],data[105],data[104],data[103],data[102],data[101],data[100],data[99],data[98],
data[97],data[96],data[95],data[94],data[93],data[92],data[91],data[90],data[89],data[88],
data[87],data[86],data[85],data[84],data[83],data[82],data[81],data[80],data[79],data[78],
data[77],data[76],data[75],data[74],data[73],data[72],data[71],data[70],data[69],data[68],
data[67],data[66],data[65],data[64],data[63],data[62],data[61],data[60],data[59],data[58],
data[57],data[56],data[55],data[54],data[53],data[52],data[51],data[50],data[49],data[48],
data[47],data[46],data[45],data[44],data[43],data[42],data[41],data[40],data[39],data[38],
data[37],data[36],data[35],data[34],data[33],data[32],data[31],data[30],data[29],data[28],
data[27],data[26],data[25],data[24],data[23],data[22],data[21],data[20],data[19],data[18],
data[17],data[16],data[15],data[14],data[13],data[12],data[11],data[10],data[9],data[8],data[7],
data[6],data[5],data[4],data[3],data[2],data[1],data[0]};

yb={data[287],data[286],data[285],data[284],data[283],data[282],data[281],data[280],
data[279],data[278],data[277],data[276],data[275],data[274],data[273],data[272],data[271],
data[270],data[269],data[268],data[267],data[266],data[265],data[264],data[263],data[262],
data[261],data[260],data[259],data[258],data[257],data[256],data[255],data[254],data[253],
data[252],data[251],data[250],data[249],data[248],data[247],data[246],data[245],data[244],
data[243],data[242],data[241],data[240],data[239],data[238],data[237],data[236],data[235],
data[234],data[233],data[232],data[231],data[230],data[229],data[228],data[227],data[226],
data[225],data[224],data[223],data[222],data[221],data[220],data[219],data[218],data[217],
data[216],data[215],data[214],data[213],data[212],data[211],data[210],data[209],data[208],
data[207],data[206],data[205],data[204],data[203],data[202],data[201],data[200],data[199],
data[198],data[197],data[196],data[195],data[194],data[193],data[192],data[191],data[190],
data[189],data[188],data[187],data[186],data[185],data[184],data[183],data[182],data[181],
data[180],data[179],data[178],data[177],data[176],data[175],data[174],data[173],data[172],
data[171],data[170],data[169],data[168],data[167],data[166],data[165],data[164],data[163],
data[162],data[161],data[160]};
```

```
p={data[415],data[414],data[413],data[412],data[411],data[410],data[409],data[408],data[407],
data[406],data[405],data[404],data[403],data[402],data[401],data[400],data[399],data[398],
data[397],data[396],data[395],data[394],data[393],data[392],data[391],data[390],data[389],
data[388],data[387],data[386],data[385],data[384],data[383],data[382],data[381],data[380],
data[379],data[378],data[377],data[376],data[375],data[374],data[373],data[372],data[371],
data[370],data[369],data[368],data[367],data[366],data[365],data[364],data[363],data[362],
data[361],data[360],data[359],data[358],data[357],data[356],data[355],data[354],data[353],
data[352],data[351],data[350],data[349],data[348],data[347],data[346],data[345],data[344],
data[343],data[342],data[341],data[340],data[339],data[338],data[337],data[336],data[335],
data[334],data[333],data[332],data[331],data[330],data[329],data[328],data[327],data[326],
data[325],data[324],data[323],data[322],data[321],data[320],data[319],data[318],data[317],
data[316],data[315],data[314],data[313],data[312],data[311],data[310],data[309],data[308],
data[307],data[306],data[305],data[304],data[303],data[302],data[301],data[300],data[299],
data[298],data[297],data[296],data[295],data[294],data[293],data[292],data[291],data[290],
data[289], data[288]};
```

```
/***************** Generate encryption key**********************/
    xa= 100;
    g=1;
    g1=1;
    for (i=0;i<xa;i=i+1)begin
    g=g*2;
    end
    ya= g%p;
    for (i=0;i<xa;i=i+1)begin
    g1=g1*yb;
    end
   key= g1%p;


/***************** Expand encryption key**********************/
while (t<4) begin
w[t]={key[(3-t)*32+31],key[(3-t)*32+30],key[(3-t)*32+29],key[(3-t)*32+28],
key[(3-t)*32+27],key[(3-t)*32+26],key[(3-t)*32+25],key[(3-t)*32+24],key[t*32+23],
key[(3-t)*32+22],key[(3-t)*32+21],key[(3-t)*32+20],key[(3-t)*32+19],
key[(3-t)*32+18],key[(3-t)*32+17],key[(3-t)*32+16],key[(3-t)*32+15],
key[(3-t)*32+14],key[(3-t)*32+13],key[(3-t)*32+12],key[(3-t)*32+11],
key[(3-t)*32+10],key[(3-t)*32+9],key[(3-t)*32+8],key[(3-t)*32+7],
key[(3-t)*32+6],key[(3-t)*32+5],key[(3-t)*32+4],key[(3-t)*32+3],
key[(3-t)*32+2],key[(3-t)*32+1],key[(3-t)*32]};

t=t+1;
end

for(i=4;i<44;i=i+1)
begin
    temp=w[i-1];
    if(i%4 ==0)
    begin
temp={temp[23],temp[22],temp[21], temp[20], temp[19], temp[18], temp[17], temp[16],
temp[15], temp[14], temp[13], temp[12], temp[11], temp[10], temp[9], temp[8], temp[7],
temp[6], temp[5], temp[4], temp[3], temp[2], temp[1], temp[0], temp[31], temp[30], temp[29],
temp[28], temp[27], temp[26], temp[25], temp[24]};
```

```
temp={sbox[{temp[31], temp[30], temp[29], temp[28], temp[27], temp[26], temp[25],
temp[24]}],sbox[{temp[23],temp[22],temp[21], temp[20], temp[19], temp[18], temp[17],
temp[16]}],sbox[{temp[15], temp[14], temp[13], temp[12], temp[11], temp[10], temp[9],
temp[8]}],sbox[{temp[7], temp[6], temp[5], temp[4], temp[3], temp[2], temp[1], temp[0]}]};

temp=temp^rcon;
if(rcon==32'h80000000)
    rcon=32'h1b000000;
else
    rcon=rcon*2;
end
w[i]=w[i-4]^temp;
end


if (mode) begin

/***If mode = 1: first encrypt then compress; if mode = 0: first decompress then decrypt***/

/*****************************ENCRYPTION*****************************/

s[0]={datain[127],datain[126],datain[125],datain[124],datain[123],datain[122],datain[121],
datain[120],datain[95],datain[94],datain[93],datain[92],datain[91],datain[90],datain[89],
datain[88],datain[63],datain[62],datain[61],datain[60],datain[59],datain[58],datain[57],
datain[56],datain[31],datain[30],datain[29],datain[28],datain[27],datain[26],datain[25],
datain[24]};

s[1]={datain[119],datain[118],datain[117],datain[116],datain[115],datain[114],datain[113],
datain[112],datain[87],datain[86],datain[85],datain[84],datain[83],datain[82],datain[81],
datain[80],datain[55],datain[54],datain[53],datain[52],datain[51],datain[50],datain[49],
datain[48],datain[23],datain[22],datain[21],datain[20],datain[19],datain[18],datain[17],
datain[16]};
```

```
s[2]={datain[111],datain[110],datain[109],datain[108],datain[107],datain[106],datain[105],
datain[104],datain[79],datain[78],datain[77],datain[76],datain[75],datain[74],datain[73],
datain[72],datain[47],datain[46],datain[45],datain[44],datain[43],datain[42],datain[41],
datain[40],datain[15],datain[14],datain[13],datain[12],datain[11],datain[10],datain[9],
datain[8]};

s[3]={datain[103],datain[102],datain[101],datain[100],datain[99],datain[98],datain[97],
datain[96],datain[71],datain[70],datain[69],datain[68],datain[67],datain[66],datain[65],
datain[64],datain[39],datain[38],datain[37],datain[36],datain[35],datain[34],datain[33],
datain[32],datain[7],datain[6],datain[5],datain[4],datain[3],datain[2],datain[1],datain[0]};


k[0]={key[127],key[126],key[125],key[124],key[123],key[122],key[121],key[120],key[95],
key[94],key[93],key[92],key[91],key[90],key[89],key[88],key[63],key[62],key[61],key[60],
key[59],key[58],key[57],key[56],key[31],key[30],key[29],key[28],key[27],key[26],key[25],
key[24]};

k[1]={key[119],key[118],key[117],key[116],key[115],key[114],key[113],key[112],key[87],
key[86],key[85],key[84],key[83],key[82],key[81],key[80],key[55],key[54],key[53],key[52],
key[51],key[50],key[49],key[48],key[23],key[22],key[21],key[20],key[19],key[18],key[17],
key[16]};

k[2]={key[111],key[110],key[109],key[108],key[107],key[106],key[105],key[104],key[79],
key[78],key[77],key[76],key[75],key[74],key[73],key[72],key[47],key[46],key[45],key[44],
key[43],key[42],key[41],key[40],key[15],key[14],key[13],key[12],key[11],key[10],key[9],
key[8]};
    k[3]={key[103],key[102],key[101],key[100],key[99],key[98],key[97],key[96],key[71],
key[70],key[69],key[68],key[67],key[66],key[65],key[64],key[39],key[38],key[37],key[36],
key[35],key[34],key[33],key[32],key[7],key[6],key[5],key[4],key[3],key[2],key[1],key[0]};
```

```
/**********************Add round key**********************/

for (i=0;i<4;i=i+1)
begin
s[i]=s[i]^k[i];
end
/*********************************************************/
for(round=1;round<10;round=round+1)begin                    //10 ROUNDS

/***************Sub bytes**************/
s[0]={sbox[{s[0][31],s[0][30],s[0][29],s[0][28],s[0][27],s[0][26],s[0][25],s[0][24]}],
sbox[{s[0][23],s[0][22],s[0][21],s[0][20],s[0][19],s[0][18],s[0][17],s[0][16]}],
sbox[{s[0][15],s[0][14],s[0][13],s[0][12],s[0][11],s[0][10],s[0][9],s[0][8]}],
sbox[{s[0][7],s[0][6],s[0][5],s[0][4],s[0][3],s[0][2],s[0][1],s[0][0]}]};

s[1]={sbox[{s[1][31],s[1][30],s[1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24]}],
sbox[{s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16]}],
sbox[{s[1][15],s[1][14],s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8]}],
sbox[{s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][2],s[1][1],s[1][0]}]};

s[2]={sbox[{s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],s[2][26],s[2][25],s[2][24]}],
sbox[{s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],s[2][16]}],
sbox[{s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8]}],
sbox[{s[2][7],s[2][6],s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],s[2][0]}]};

s[3]={sbox[{s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],s[3][25],s[3][24]}],
sbox[{s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][17],s[3][16]}],
sbox[{s[3][15],s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8]}],
sbox[{s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][1],s[3][0]}]};


/***************Shift rows**************/

s[1]={s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16],s[1][15],s[1][14],s[1
][13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8],s[1][7],s[1][6],s[1][5],s[1][4],s[1][3]
s[1][2],s[1][1],s[1][0],s[1][31],s[1][30],s[1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24]};
```

```
s[2]={s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8],s[2][7],s[2][6],
s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],s[2][0],s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],
s[2][26],s[2][25],s[2][24],s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],
s[2][16]};
    s[3]={s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][1],s[3][0],s[3][31],s[3][30],s[3][29],
s[3][28],s[3][27],s[3][26],s[3][25],s[3][24],s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],
s[3][18],s[3][17],s[3][16],s[3][15],s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],
s[3][8]};

/***************Mix Column**************/

for(i=0;i<4;i=i+1) begin

{s1[0][i*8+7],s1[0][i*8+6],s1[0][i*8+5],s1[0][i*8+4],s1[0][i*8+3],s1[0][i*8+2],s1[0][i*8+1],s
1[0][i*8]}=mul2[{s[0][i*8+7],s[0][i*8+6],s[0][i*8+5],s[0][i*8+4],s[0][i*8+3],s[0][i*8+2],
s[0][i*8+1],s[0][i*8]}]^mul3[{s[1][i*8+7],s[1][i*8+6],s[1][i*8+5],s[1][i*8+4],s[1][i*8+3],
s[1][i*8+2],s[1][i*8+1],s[1][i*8]}]^{s[2][i*8+7],s[2][i*8+6],s[2][i*8+5],s[2][i*8+4],
s[2][i*8+3],s[2][i*8+2],s[2][i*8+1],s[2][i*8]}^{s[3][i*8+7],s[3][i*8+6],s[3][i*8+5],
s[3][i*8+4],s[3][i*8+3],s[3][i*8+2],s[3][i*8+1],s[3][i*8]};

{s1[1][i*8+7],s1[1][i*8+6],s1[1][i*8+5],s1[1][i*8+4],s1[1][i*8+3],s1[1][i*8+2],s1[1][i*8+1],s
1[1][i*8]}={s[0][i*8+7],s[0][i*8+6],s[0][i*8+5],s[0][i*8+4],s[0][i*8+3],s[0][i*8+2],
s[0][i*8+1],s[0][i*8]}^mul2[{s[1][i*8+7],s[1][i*8+6],s[1][i*8+5],s[1][i*8+4],s[1][i*8+3],
s[1][i*8+2],s[1][i*8+1],s[1][i*8]}]^mul3[{s[2][i*8+7],s[2][i*8+6],s[2][i*8+5],s[2][i*8+4],
s[2][i*8+3],s[2][i*8+2],s[2][i*8+1],s[2][i*8]}]^{s[3][i*8+7],s[3][i*8+6],s[3][i*8+5],
s[3][i*8+4],s[3][i*8+3],s[3][i*8+2],s[3][i*8+1],s[3][i*8]};

{s1[2][i*8+7],s1[2][i*8+6],s1[2][i*8+5],s1[2][i*8+4],s1[2][i*8+3],s1[2][i*8+2],s1[2][i*8+1],s
1[2][i*8]}={s[0][i*8+7],s[0][i*8+6],s[0][i*8+5],s[0][i*8+4],s[0][i*8+3],s[0][i*8+2],
s[0][i*8+1],s[0][i*8]}^{s[1][i*8+7],s[1][i*8+6],s[1][i*8+5],s[1][i*8+4],s[1][i*8+3],
s[1][i*8+2],s[1][i*8+1],s[1][i*8]}^mul2[{s[2][i*8+7],s[2][i*8+6],s[2][i*8+5],s[2][i*8+4],
s[2][i*8+3],s[2][i*8+2],s[2][i*8+1],s[2][i*8]}]^mul3[{s[3][i*8+7],s[3][i*8+6],s[3][i*8+5],
s[3][i*8+4],s[3][i*8+3],s[3][i*8+2],s[3][i*8+1],s[3][i*8]}];
```

```
{s1[3][i*8+7],s1[3][i*8+6],s1[3][i*8+5],s1[3][i*8+4],s1[3][i*8+3],s1[3][i*8+2],s1[3][i*8+1],s
1[3][i*8]}=mul3[{s[0][i*8+7],s[0][i*8+6],s[0][i*8+5],s[0][i*8+4],s[0][i*8+3],s[0][i*8+2],
s[0][i*8+1],s[0][i*8]}]^{s[1][i*8+7],s[1][i*8+6],s[1][i*8+5],s[1][i*8+4],s[1][i*8+3],
s[1][i*8+2],s[1][i*8+1],s[1][i*8]}^{s[2][i*8+7],s[2][i*8+6],s[2][i*8+5],s[2][i*8+4],
s[2][i*8+3],s[2][i*8+2],s[2][i*8+1],s[2][i*8]}^mul2[{s[3][i*8+7],s[3][i*8+6],s[3][i*8+5],
s[3][i*8+4],s[3][i*8+3],s[3][i*8+2],s[3][i*8+1],s[3][i*8]}];
end
{s[0],s[1],s[2],s[3]}={s1[0],s1[1],s1[2],s1[3]};

/*************** Add round key**************/
key1={w[round*4],w[round*4+1],w[round*4+2],w[round*4+3]};

k[0]={key1[127],key1[126],key1[125],key1[124],key1[123],key1[122],key1[121],key1[120],ke
y1[95],key1[94],key1[93],key1[92],key1[91],key1[90],key1[89],key1[88],key1[63],
key1[62],key1[61],key1[60],key1[59],key1[58],key1[57],key1[56],key1[31],key1[30],
key1[29],key1[28],key1[27],key1[26],key1[25],key1[24]};

k[1]={key1[119],key1[118],key1[117],key1[116],key1[115],key1[114],key1[113],key1[112],ke
y1[87],key1[86],key1[85],key1[84],key1[83],key1[82],key1[81],key1[80],key1[55],
key1[54],key1[53],key1[52],key1[51],key1[50],key1[49],key1[48],key1[23],key1[22],
key1[21],key1[20],key1[19],key1[18],key1[17],key1[16]};

k[2]={key1[111],key1[110],key1[109],key1[108],key1[107],key1[106],key1[105],key1[104],ke
y1[79],key1[78],key1[77],key1[76],key1[75],key1[74],key1[73],key1[72],key1[47],
key1[46],key1[45],key1[44],key1[43],key1[42],key1[41],key1[40],key1[15],key1[14],
key1[13],key1[12],key1[11],key1[10],key1[9],key1[8]};

k[3]={key1[103],key1[102],key1[101],key1[100],key1[99],key1[98],key1[97],key1[96],key1[7
1],key1[70],key1[69],key1[68],key1[67],key1[66],key1[65],key1[64],key1[39],key1[38],
key1[37],key1[36],key1[35],key1[34],key1[33],key1[32],key1[7],key1[6],key1[5],key1[4],
key1[3],key1[2],key1[1],key1[0]};

    for (i=0;i<4;i=i+1)
    begin
      s[i]=s[i]^k[i];
    end
end  // End ROUND
```

```
/***************Sub bytes**************/
s[0]={sbox[{s[0][31],s[0][30],s[0][29],s[0][28],s[0][27],s[0][26],s[0][25],s[0][24]}],sbox[{s[0][
23],s[0][22],s[0][21],s[0][20],s[0][19],s[0][18],s[0][17],s[0][16]}],sbox[{s[0][15],s[0][14],s[0][
13],s[0][12],s[0][11],s[0][10],s[0][9],s[0][8]}],sbox[{s[0][7],s[0][6],s[0][5],s[0][4],s[0][3],s[0][
2],s[0][1],s[0][0]}]};

s[1]={sbox[{s[1][31],s[1][30],s[1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24]}],sbox[{s[1][
23],s[1][22],s[1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16]}],sbox[{s[1][15],s[1][14],s[1][
13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8]}],sbox[{s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][
2],s[1][1],s[1][0]}]};

s[2]={sbox[{s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],s[2][26],s[2][25],s[2][24]}],sbox[{s[2][
23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],s[2][16]}],sbox[{s[2][15],s[2][14],s[2][
13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8]}],sbox[{s[2][7],s[2][6],s[2][5],s[2][4],s[2][3],s[2][
2],s[2][1],s[2][0]}]};

s[3]={sbox[{s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],s[3][25],s[3][24]}],sbox[{s[3][
23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][17],s[3][16]}],sbox[{s[3][15],s[3][14],s[3][
13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8]}],sbox[{s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][
2],s[3][1],s[3][0]}]};


/***************Shift rows**************/
s[1]={s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16],s[1][15],s[1][14],s[1
][13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8],s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][2],s[1][
1],s[1][0],s[1][31],s[1][30],s[1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24]};

s[2]={s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8],s[2][7],s[2][6],s[2][5],
s[2][4],s[2][3],s[2][2],s[2][1],s[2][0],s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],s[2][26],s[2][2
5],s[2][24],s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],s[2][16]};

s[3]={s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][1],s[3][0],s[3][31],s[3][30],s[3][29],s[3]
[28],s[3][27],s[3][26],s[3][25],s[3][24],s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][
17],s[3][16],s[3][15],s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8]};
```

```
/**************** Add round key***************/
    key1={w[40],w[41],w[42],w[43]};

k[0]={key1[127],key1[126],key1[125],key1[124],key1[123],key1[122],key1[121],key1[120],ke
y1[95],key1[94],key1[93],key1[92],key1[91],key1[90],key1[89],key1[88],key1[63],
key1[62],key1[61],key1[60],key1[59],key1[58],key1[57],key1[56],key1[31],key1[30],
key1[29],key1[28],key1[27],key1[26],key1[25],key1[24]};

k[1]={key1[119],key1[118],key1[117],key1[116],key1[115],key1[114],key1[113],key1[112],ke
y1[87],key1[86],key1[85],key1[84],key1[83],key1[82],key1[81],key1[80],key1[55],
key1[54],key1[53],key1[52],key1[51],key1[50],key1[49],key1[48],key1[23],key1[22],
key1[21],key1[20],key1[19],key1[18],key1[17],key1[16]};

k[2]={key1[111],key1[110],key1[109],key1[108],key1[107],key1[106],key1[105],key1[104],ke
y1[79],key1[78],key1[77],key1[76],key1[75],key1[74],key1[73],key1[72],key1[47],
key1[46],key1[45],key1[44],key1[43],key1[42],key1[41],key1[40],key1[15],key1[14],
key1[13],key1[12],key1[11],key1[10],key1[9],key1[8]};

k[3]={key1[103],key1[102],key1[101],key1[100],key1[99],key1[98],key1[97],key1[96],
key1[71],key1[70],key1[69],key1[68],key1[67],key1[66],key1[65],key1[64],key1[39],
key1[38],key1[37],key1[36],key1[35],key1[34],key1[33],key1[32],key1[7],key1[6],key1[5],
key1[4],key1[3],key1[2],key1[1],key1[0]};

for (i=0;i<4;i=i+1)
    begin
      s[i]=s[i]^k[i];
    end
s1[0]={s[0][31],s[0][30],s[0][29],s[0][28],s[0][27],s[0][26],s[0][25],s[0][24],s[1][31],s[1][30],s[
1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24],s[2][31],s[2][30],s[2][29],s[2][28],
s[2][27],s[2][26],s[2][25],s[2][24],s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],
s[3][25],s[3][24]};

s1[1]={s[0][23],s[0][22],s[0][21],s[0][20],s[0][19],s[0][18],s[0][17],s[0][16],s[1][23],s[1][22],s[
1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16],s[2][23],s[2][22],s[2][21],s[2][20],
s[2][19],s[2][18],s[2][17],s[2][16],s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],
s[3][17],s[3][16]};
```

```
s1[2]={s[0][15],s[0][14],s[0][13],s[0][12],s[0][11],s[0][10],s[0][9],s[0][8],s[1][15],s[1][14],
s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8],s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],
s[2][10],s[2][9],s[2][8],s[3][15],s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8]};

s1[3]={s[0][7],s[0][6],s[0][5],s[0][4],s[0][3],s[0][2],s[0][1],s[0][0],s[1][7],s[1][6],s[1][5],
s[1][4],s[1][3],s[1][2],s[1][1],s[1][0],s[2][7],s[2][6],s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],
s[2][0],s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][1],s[3][0]};
data={s1[0],s1[1],s1[2],s1[3]};

/**************************COMPRESSION*****************************/
count=0;
o=0;

/***Initialize dictionary***/
while(count<16) begin
dictionary[count]=count;
count=count+1;
end
/*********************/

string ={data[3],data[2],data[1],data[0]};

for(j=4;j<128;j=j+4)
begin
char={data[j+3],data[j+2],data[j+1],data[j]};

f=0;
for (c=0;c<=30;c=c+1)
begin
if({string,char}==dictionary[c])
f=1;
end
if (f) begin
string= {string,char};
end
```

```
else
begin
for (c=0;c<=30;c=c+1) begin
if (string==dictionary[c]) begin
out[o]=c;
o=o+1;
end
end

dictionary[count]={string,char};
if(count<30)
count=count+1;
string= char;
char=0;
end
end
if (string)
for (c=0;c<=30;c=c+1) begin
if (string==dictionary[c]) begin
out[o]=c;
o=o+1;
end
end

reg1={out[31],out[30],out[29],out[28],out[27],out[26],out[25],out[24],out[23],out[22],
out[21],out[20],out[19],out[18],out[17],out[16],out[15],out[14],out[13],out[12],out[11],
out[10],out[9],out[8],out[7],out[6],out[5],out[4],out[3],out[2],out[1],out[0]};
end
```

```
else begin              // mode = 0, (decompress, decrypt)
/************************DECOMPRESSION*****************************/
 count=0;
 i=1;
 /***Initialize dictionary***/

while(count<16) begin
 dictionary[count]=count;
 count=count+1;
 end
/**********************/
 oldcode= {datain[4],datain[3],datain[2],datain[1],datain[0]};
 outs[0]=dictionary[oldcode];

for(t=5;t<160;t=t+5) begin
    newcode={datain[t+4],datain[t+3],datain[t+2],datain[t+1],datain[t]};
    if (dictionary[newcode]!=0)
    begin
            string=dictionary[newcode];
            char={string[3],string[2],string[1],string[0]};
    end
    else
    begin
            string= dictionary[oldcode];
            char={string[3],string[2],string[1],string[0]};
            string={string,char};
    end
    j=0;
    while({string[j+3],string[j+2],string[j+1],string[j]}!=0)
    begin
            outs[i]={string[j+3],string[j+2],string[j+1],string[j]};
            j=j+4;
            i=i+1;
    end
    dictionary[count]=dictionary[oldcode];
    j=0;
    o=0;
```

```
while({dictionary[count][j+3],dictionary[count][j+2],dictionary[count][j+1],dictionay[count][j]
}!=0)
    begin
        j=j+4;
    end

{dictionary[count][j+3],dictionary[count][j+2],dictionary[count][j+1],dictionary[count][j]}=cha
r;
    count=count+1;
    oldcode=newcode;

 end
 data={outs[31],outs[30],outs[29],outs[28],outs[27],outs[26],outs[25],outs[24],outs[23],
outs[22],outs[21],outs[20],outs[19],outs[18],outs[17],outs[16],outs[15],outs[14],outs[13],
outs[12],outs[11],outs[10],outs[9],outs[8],outs[7],outs[6],outs[5],outs[4],outs[3],outs[2],
outs[1],outs[0]};


/**************************DECRYPTION*****************************/

s[0]={data[127],data[126],data[125],data[124],data[123],data[122],data[121],data[120],
data[95],data[94],data[93],data[92],data[91],data[90],data[89],data[88],data[63],data[62],
data[61],data[60],data[59],data[58],data[57],data[56],data[31],data[30],data[29],data[28],
data[27],data[26],data[25],data[24]};

s[1]={data[119],data[118],data[117],data[116],data[115],data[114],data[113],data[112],
data[87],data[86],data[85],data[84],data[83],data[82],data[81],data[80],data[55],data[54],
data[53],data[52],data[51],data[50],data[49],data[48],data[23],data[22],data[21],data[20],
data[19],data[18],data[17],data[16]};

s[2]={data[111],data[110],data[109],data[108],data[107],data[106],data[105],data[104],
data[79],data[78],data[77],data[76],data[75],data[74],data[73],data[72],data[47],data[46],
data[45],data[44],data[43],data[42],data[41],data[40],data[15],data[14],data[13],data[12],
data[11],data[10],data[9],data[8]};
```

```
s[3]={data[103],data[102],data[101],data[100],data[99],data[98],data[97],data[96],data[71],
data[70],data[69],data[68],data[67],data[66],data[65],data[64],data[39],data[38],data[37],
data[36],data[35],data[34],data[33],data[32],data[7],data[6],data[5],data[4],data[3],data[2],
data[1],data[0]};


  key1={w[40],w[41],w[42],w[43]};

k[0]={key1[127],key1[126],key1[125],key1[124],key1[123],key1[122],key1[121],key1[120],ke
y1[95],key1[94],key1[93],key1[92],key1[91],key1[90],key1[89],key1[88],key1[63],
key1[62],key1[61],key1[60],key1[59],key1[58],key1[57],key1[56],key1[31],key1[30],
key1[29],key1[28],key1[27],key1[26],key1[25],key1[24]};

k[1]={key1[119],key1[118],key1[117],key1[116],key1[115],key1[114],key1[113],key1[112],ke
y1[87],key1[86],key1[85],key1[84],key1[83],key1[82],key1[81],key1[80],key1[55],
key1[54],key1[53],key1[52],key1[51],key1[50],key1[49],key1[48],key1[23],key1[22],
key1[21],key1[20],key1[19],key1[18],key1[17],key1[16]};

k[2]={key1[111],key1[110],key1[109],key1[108],key1[107],key1[106],key1[105],key1[104],ke
y1[79],key1[78],key1[77],key1[76],key1[75],key1[74],key1[73],key1[72],key1[47],
key1[46],key1[45],key1[44],key1[43],key1[42],key1[41],key1[40],key1[15],key1[14],
key1[13],key1[12],key1[11],key1[10],key1[9],key1[8]};

k[3]={key1[103],key1[102],key1[101],key1[100],key1[99],key1[98],key1[97],key1[96],
key1[71],key1[70],key1[69],key1[68],key1[67],key1[66],key1[65],key1[64],key1[39],
key1[38],key1[37],key1[36],key1[35],key1[34],key1[33],key1[32],key1[7],key1[6],key1[5],
key1[4],key1[3],key1[2],key1[1],key1[0]};

/********************** Add round key**************/
  for (i=0;i<4;i=i+1)
  begin
    s[i]=s[i]^k[i];
  end
```

```
for(round=9;round>0;round=round-1) begin        //10 ROUNDS

/**********************        Inverse        shift        rows**************/
s[1]={s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][2],s[1][1],s[1][0],s[1][31],s[1][30],s[1][29],
s[1][28],s[1][27],s[1][26],s[1][25],s[1][24],s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],
s[1][18],s[1][17],s[1][16],s[1][15],s[1][14],s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],
s[1][8]};

s[2]={s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8],s[2][7],s[2][6],
s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],s[2][0],s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],
s[2][26],s[2][25],s[2][24],s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],
s[2][16]};

s[3]={s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][17],s[3][16],s[3][15],
s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8],s[3][7],s[3][6],s[3][5],s[3][4],
s[3][3],s[3][2],s[3][1],s[3][0],s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],s[3][25],
s[3][24]};

/**********************        Inverse        sub        bytes**************/
s[0]={invsbox[{s[0][31],s[0][30],s[0][29],s[0][28],s[0][27],s[0][26],s[0][25],s[0][24]}],
invsbox[{s[0][23],s[0][22],s[0][21],s[0][20],s[0][19],s[0][18],s[0][17],s[0][16]}],
invsbox[{s[0][15],s[0][14],s[0][13],s[0][12],s[0][11],s[0][10],s[0][9],s[0][8]}],
invsbox[{s[0][7],s[0][6],s[0][5],s[0][4],s[0][3],s[0][2],s[0][1],s[0][0]}]};

s[1]={invsbox[{s[1][31],s[1][30],s[1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24]}],
invsbox[{s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16]}],
invsbox[{s[1][15],s[1][14],s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8]}],
invsbox[{s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][2],s[1][1],s[1][0]}]};

s[2]={invsbox[{s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],s[2][26],s[2][25],s[2][24]}],
invsbox[{s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],s[2][16]}],
invsbox[{s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8]}],
invsbox[{s[2][7],s[2][6],s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],s[2][0]}]};
```

```
s[3]={invsbox[{s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],s[3][25],s[3][24]}],
invsbox[{s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][17],s[3][16]}],
invsbox[{s[3][15],s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8]}],
invsbox[{s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][1],s[3][0]}]};


/********************** Add round key**************/
  key1={w[round*4],w[round*4+1],w[round*4+2],w[round*4+3]};

k[0]={key1[127],key1[126],key1[125],key1[124],key1[123],key1[122],key1[121],key1[120],ke
y1[95],key1[94],key1[93],key1[92],key1[91],key1[90],key1[89],key1[88],key1[63],
key1[62],key1[61],key1[60],key1[59],key1[58],key1[57],key1[56],key1[31],key1[30],
key1[29],key1[28],key1[27],key1[26],key1[25],key1[24]};

k[1]={key1[119],key1[118],key1[117],key1[116],key1[115],key1[114],key1[113],key1[112],ke
y1[87],key1[86],key1[85],key1[84],key1[83],key1[82],key1[81],key1[80],key1[55],
key1[54],key1[53],key1[52],key1[51],key1[50],key1[49],key1[48],key1[23],key1[22],
key1[21],key1[20],key1[19],key1[18],key1[17],key1[16]};

k[2]={key1[111],key1[110],key1[109],key1[108],key1[107],key1[106],key1[105],key1[104],ke
y1[79],key1[78],key1[77],key1[76],key1[75],key1[74],key1[73],key1[72],key1[47],
key1[46],key1[45],key1[44],key1[43],key1[42],key1[41],key1[40],key1[15],key1[14],
key1[13],key1[12],key1[11],key1[10],key1[9],key1[8]};

k[3]={key1[103],key1[102],key1[101],key1[100],key1[99],key1[98],key1[97],key1[96],
key1[71],key1[70],key1[69],key1[68],key1[67],key1[66],key1[65],key1[64],key1[39],
key1[38],key1[37],key1[36],key1[35],key1[34],key1[33],key1[32],key1[7],key1[6],key1[5],
key1[4],key1[3],key1[2],key1[1],key1[0]};


  for (i=0;i<4;i=i+1)
  begin
    s[i]=s[i]^k[i];
  end
```

```
/********************* Inverse mix column**************/
  for(i=0;i<4;i=i+1) begin

{s1[0][i*8+7],s1[0][i*8+6],s1[0][i*8+5],s1[0][i*8+4],s1[0][i*8+3],s1[0][i*8+2],s1[0][i*8+1],s
1[0][i*8]}=mule[{s[0][i*8+7],s[0][i*8+6],s[0][i*8+5],s[0][i*8+4],s[0][i*8+3],s[0][i*8+2],
s[0][i*8+1],s[0][i*8]}]^mulb[{s[1][i*8+7],s[1][i*8+6],s[1][i*8+5],s[1][i*8+4],s[1][i*8+3],
s[1][i*8+2],s[1][i*8+1],s[1][i*8]}]^muld[{s[2][i*8+7],s[2][i*8+6],s[2][i*8+5],s[2][i*8+4],
s[2][i*8+3],s[2][i*8+2],s[2][i*8+1],s[2][i*8]}]^mul9[{s[3][i*8+7],s[3][i*8+6],s[3][i*8+5],
s[3][i*8+4],s[3][i*8+3],s[3][i*8+2],s[3][i*8+1],s[3][i*8]}];

{s1[1][i*8+7],s1[1][i*8+6],s1[1][i*8+5],s1[1][i*8+4],s1[1][i*8+3],s1[1][i*8+2],s1[1][i*8+1],s
1[1][i*8]}=mul9[{s[0][i*8+7],s[0][i*8+6],s[0][i*8+5],s[0][i*8+4],s[0][i*8+3],s[0][i*8+2],
s[0][i*8+1],s[0][i*8]}]^mule[{s[1][i*8+7],s[1][i*8+6],s[1][i*8+5],s[1][i*8+4],s[1][i*8+3],
s[1][i*8+2],s[1][i*8+1],s[1][i*8]}]^mulb[{s[2][i*8+7],s[2][i*8+6],s[2][i*8+5],s[2][i*8+4],
s[2][i*8+3],s[2][i*8+2],s[2][i*8+1],s[2][i*8]}]^muld[{s[3][i*8+7],s[3][i*8+6],s[3][i*8+5],
s[3][i*8+4],s[3][i*8+3],s[3][i*8+2],s[3][i*8+1],s[3][i*8]}];

{s1[2][i*8+7],s1[2][i*8+6],s1[2][i*8+5],s1[2][i*8+4],s1[2][i*8+3],s1[2][i*8+2],s1[2][i*8+1],s
1[2][i*8]}=muld[{s[0][i*8+7],s[0][i*8+6],s[0][i*8+5],s[0][i*8+4],s[0][i*8+3],s[0][i*8+2],
s[0][i*8+1],s[0][i*8]}]^mul9[{s[1][i*8+7],s[1][i*8+6],s[1][i*8+5],s[1][i*8+4],s[1][i*8+3],
s[1][i*8+2],s[1][i*8+1],s[1][i*8]}]^mule[{s[2][i*8+7],s[2][i*8+6],s[2][i*8+5],s[2][i*8+4],
s[2][i*8+3],s[2][i*8+2],s[2][i*8+1],s[2][i*8]}]^mulb[{s[3][i*8+7],s[3][i*8+6],s[3][i*8+5],
s[3][i*8+4],s[3][i*8+3],s[3][i*8+2],s[3][i*8+1],s[3][i*8]}];

{s1[3][i*8+7],s1[3][i*8+6],s1[3][i*8+5],s1[3][i*8+4],s1[3][i*8+3],s1[3][i*8+2],s1[3][i*8+1],s
1[3][i*8]}=mulb[{s[0][i*8+7],s[0][i*8+6],s[0][i*8+5],s[0][i*8+4],s[0][i*8+3],s[0][i*8+2],
s[0][i*8+1],s[0][i*8]}]^muld[{s[1][i*8+7],s[1][i*8+6],s[1][i*8+5],s[1][i*8+4],s[1][i*8+3],
s[1][i*8+2],s[1][i*8+1],s[1][i*8]}]^mul9[{s[2][i*8+7],s[2][i*8+6],s[2][i*8+5],s[2][i*8+4],
s[2][i*8+3],s[2][i*8+2],s[2][i*8+1],s[2][i*8]}]^mule[{s[3][i*8+7],s[3][i*8+6],s[3][i*8+5],
s[3][i*8+4],s[3][i*8+3],s[3][i*8+2],s[3][i*8+1],s[3][i*8]}];

end
  {s[0],s[1],s[2],s[3]}={s1[0],s1[1],s1[2],s1[3]};

end
```

```
/********************* Inverse shift rows**************/
   s[1]={s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][2],s[1][1],s[1][0],s[1][31],s[1][30],s[1][29],
s[1][28],s[1][27],s[1][26],s[1][25],s[1][24],s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],
s[1][18],s[1][17],s[1][16],s[1][15],s[1][14],s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],
s[1][8]};

s[2]={s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8],s[2][7],s[2][6],
s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],s[2][0],s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],
s[2][26],s[2][25],s[2][24],s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],
s[2][16]};

s[3]={s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][17],s[3][16],s[3][15],s[3][14],s[3
][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8],s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][
1],s[3][0],s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],s[3][25],s[3][24]};

/********************* Inverse sub bytes**************/
s[0]={invsbox[{s[0][31],s[0][30],s[0][29],s[0][28],s[0][27],s[0][26],s[0][25],s[0][24]}],
invsbox[{s[0][23],s[0][22],s[0][21],s[0][20],s[0][19],s[0][18],s[0][17],s[0][16]}],
invsbox[{s[0][15],s[0][14],s[0][13],s[0][12],s[0][11],s[0][10],s[0][9],s[0][8]}],
invsbox[{s[0][7],s[0][6],s[0][5],s[0][4],s[0][3],s[0][2],s[0][1],s[0][0]}]};

s[1]={invsbox[{s[1][31],s[1][30],s[1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24]}],
invsbox[{s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16]}],
invsbox[{s[1][15],s[1][14],s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8]}],
invsbox[{s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][2],s[1][1],s[1][0]}]};

s[2]={invsbox[{s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],s[2][26],s[2][25],s[2][24]}],
invsbox[{s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],s[2][16]}],
invsbox[{s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8]}],
invsbox[{s[2][7],s[2][6],s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],s[2][0]}]};

s[3]={invsbox[{s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],s[3][25],s[3][24]}],
invsbox[{s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][17],s[3][16]}],
invsbox[{s[3][15],s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8]}],
invsbox[{s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][1],s[3][0]}]};
```

```
/********************** Inverse shift rows**************/
   s[1]={s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][2],s[1][1],s[1][0],s[1][31],s[1][30],s[1][29],
s[1][28],s[1][27],s[1][26],s[1][25],s[1][24],s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],
s[1][18],s[1][17],s[1][16],s[1][15],s[1][14],s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],
s[1][8]};

s[2]={s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8],s[2][7],s[2][6],
s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],s[2][0],s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],
s[2][26],s[2][25],s[2][24],s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],
s[2][16]};

s[3]={s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][17],s[3][16],s[3][15],s[3][14],s[3
][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8],s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][
1],s[3][0],s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],s[3][25],s[3][24]};

/**********************  Inverse  sub  bytes**************/
s[0]={invsbox[{s[0][31],s[0][30],s[0][29],s[0][28],s[0][27],s[0][26],s[0][25],s[0][24]}],
invsbox[{s[0][23],s[0][22],s[0][21],s[0][20],s[0][19],s[0][18],s[0][17],s[0][16]}],
invsbox[{s[0][15],s[0][14],s[0][13],s[0][12],s[0][11],s[0][10],s[0][9],s[0][8]}],
invsbox[{s[0][7],s[0][6],s[0][5],s[0][4],s[0][3],s[0][2],s[0][1],s[0][0]}]};

s[1]={invsbox[{s[1][31],s[1][30],s[1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24]}],
invsbox[{s[1][23],s[1][22],s[1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16]}],
invsbox[{s[1][15],s[1][14],s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8]}],
invsbox[{s[1][7],s[1][6],s[1][5],s[1][4],s[1][3],s[1][2],s[1][1],s[1][0]}]};

s[2]={invsbox[{s[2][31],s[2][30],s[2][29],s[2][28],s[2][27],s[2][26],s[2][25],s[2][24]}],
invsbox[{s[2][23],s[2][22],s[2][21],s[2][20],s[2][19],s[2][18],s[2][17],s[2][16]}],
invsbox[{s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],s[2][10],s[2][9],s[2][8]}],
invsbox[{s[2][7],s[2][6],s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],s[2][0]}]};

s[3]={invsbox[{s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],s[3][25],s[3][24]}],
invsbox[{s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],s[3][17],s[3][16]}],
invsbox[{s[3][15],s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8]}],
invsbox[{s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][1],s[3][0]}]};
```

```
/********************* Add round key**************/
 key1={w[0],w[1],w[2],w[3]};
k[0]={key1[127],key1[126],key1[125],key1[124],key1[123],key1[122],key1[121],key1[120],ke
y1[95],key1[94],key1[93],key1[92],key1[91],key1[90],key1[89],key1[88],key1[63],
key1[62],key1[61],key1[60],key1[59],key1[58],key1[57],key1[56],key1[31],key1[30],
key1[29],key1[28],key1[27],key1[26],key1[25],key1[24]};

k[1]={key1[119],key1[118],key1[117],key1[116],key1[115],key1[114],key1[113],key1[112],ke
y1[87],key1[86],key1[85],key1[84],key1[83],key1[82],key1[81],key1[80],key1[55],
key1[54],key1[53],key1[52],key1[51],key1[50],key1[49],key1[48],key1[23],key1[22],
key1[21],key1[20],key1[19],key1[18],key1[17],key1[16]};

k[2]={key1[111],key1[110],key1[109],key1[108],key1[107],key1[106],key1[105],key1[104],ke
y1[79],key1[78],key1[77],key1[76],key1[75],key1[74],key1[73],key1[72],key1[47],
key1[46],key1[45],key1[44],key1[43],key1[42],key1[41],key1[40],key1[15],key1[14],
key1[13],key1[12],key1[11],key1[10],key1[9],key1[8]};

k[3]={key1[103],key1[102],key1[101],key1[100],key1[99],key1[98],key1[97],key1[96],
key1[71],key1[70],key1[69],key1[68],key1[67],key1[66],key1[65],key1[64],key1[39],
key1[38],key1[37],key1[36],key1[35],key1[34],key1[33],key1[32],key1[7],key1[6],key1[5],
key1[4],key1[3],key1[2],key1[1],key1[0]};

  for (i=0;i<4;i=i+1)
  begin
    s[i]=s[i]^k[i];
  end

s1[0]={s[0][31],s[0][30],s[0][29],s[0][28],s[0][27],s[0][26],s[0][25],s[0][24],s[1][31],s[1][30],s[
1][29],s[1][28],s[1][27],s[1][26],s[1][25],s[1][24],s[2][31],s[2][30],s[2][29],s[2][28],
s[2][27],s[2][26],s[2][25],s[2][24],s[3][31],s[3][30],s[3][29],s[3][28],s[3][27],s[3][26],
s[3][25],s[3][24]};

s1[1]={s[0][23],s[0][22],s[0][21],s[0][20],s[0][19],s[0][18],s[0][17],s[0][16],s[1][23],s[1][22],s[
1][21],s[1][20],s[1][19],s[1][18],s[1][17],s[1][16],s[2][23],s[2][22],s[2][21],s[2][20],
s[2][19],s[2][18],s[2][17],s[2][16],s[3][23],s[3][22],s[3][21],s[3][20],s[3][19],s[3][18],
s[3][17],s[3][16]};
  s1[2]={s[0][15],s[0][14],s[0][13],s[0][12],s[0][11],s[0][10],s[0][9],s[0][8],s[1][15],s[1][14],
```

```
s[1][13],s[1][12],s[1][11],s[1][10],s[1][9],s[1][8],s[2][15],s[2][14],s[2][13],s[2][12],s[2][11],
s[2][10],s[2][9],s[2][8],s[3][15],s[3][14],s[3][13],s[3][12],s[3][11],s[3][10],s[3][9],s[3][8]};

s1[3]={s[0][7],s[0][6],s[0][5],s[0][4],s[0][3],s[0][2],s[0][1],s[0][0],s[1][7],s[1][6],s[1][5],
s[1][4],s[1][3],s[1][2],s[1][1],s[1][0],s[2][7],s[2][6],s[2][5],s[2][4],s[2][3],s[2][2],s[2][1],
s[2][0],s[3][7],s[3][6],s[3][5],s[3][4],s[3][3],s[3][2],s[3][1],s[3][0]};

reg1={s1[0],s1[1],s1[2],s1[3]};
end
  end

assign dataout=reg1;
endmodule

/**********************************SERIALIZATION**********************/
module PTOS(In,CLK,Out,N,Clr);
input [159:0]In;
input CLK,Clr;
output Out;
output [7:0]N;
reg Out;
reg [7:0]N;
always @ (posedge CLK)
 if (~Clr) N= 8'b00000000;
 else if (N == 8'b11111111) N= 8'b00000000;
 else
  begin
    Out=In[N];
    N=N+1'b1;
  end
Endmodule
```

```
/*****************************DE-SERIALIZATION**********************/
module STOP(In,CLK,Out,N,Clr);
input In;
input CLK,Clr;
output [159:0] Out;
output [7:0]N;
reg [159:0]Out;
reg [7:0]N;
always @ (posedge CLK)
 if (~Clr) N= 4'b00000000;

 else
  begin
    Out[N]=In;
    N=N+1'b1;
  end
endmodule
```

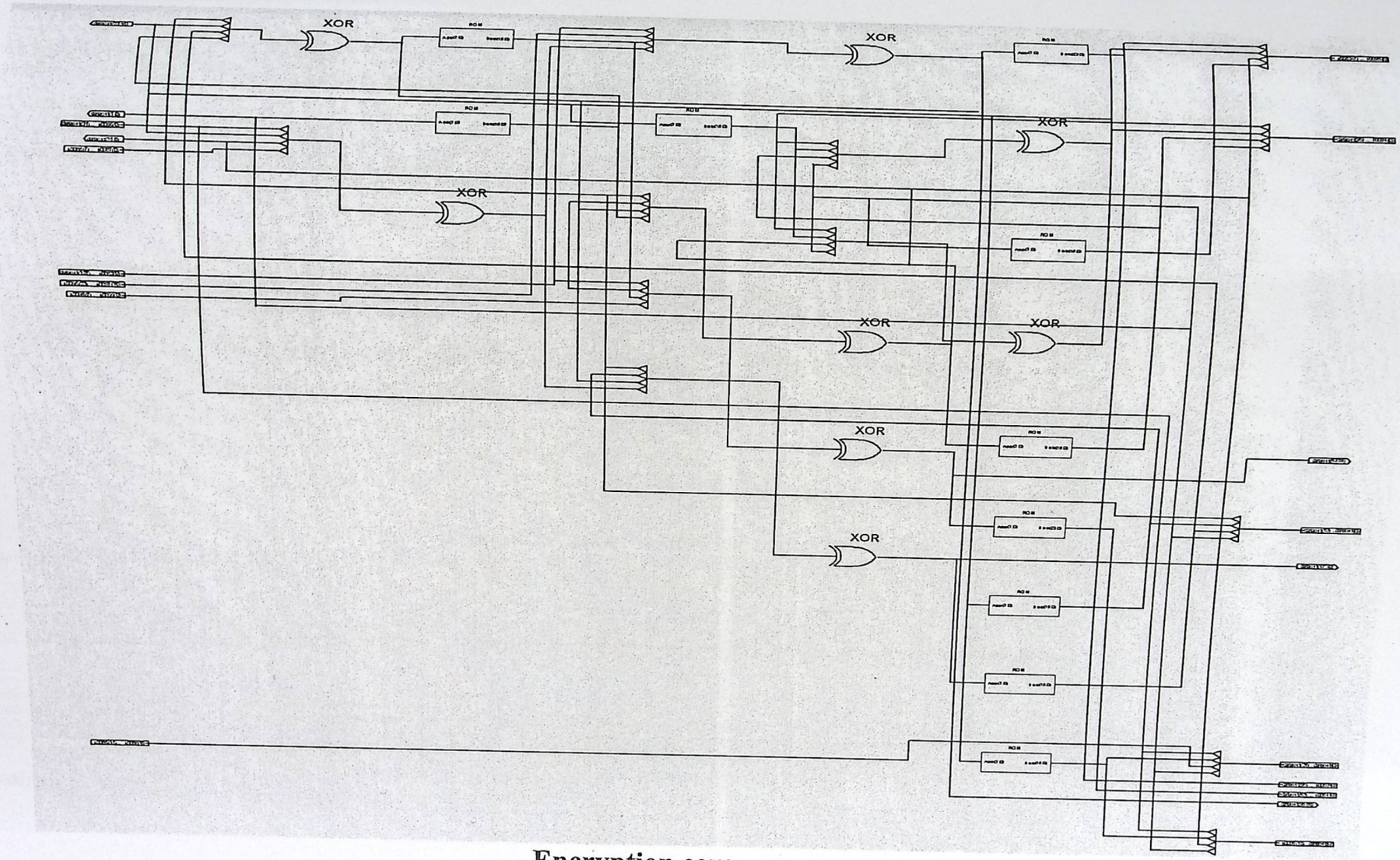
```
/********************* transmitter*********************8/
module tx(In,Out,CLK,Clr);
input In,CLK,Clr;
output Out;
reg[3:0] N;
reg  Out;
always @ (posedge CLK)begin
 if(N==0) begin Out=0;  N=N+1; end
 else if (N==9)begin Out=1;N=0;end
 else begin
 Out=In;
 N=N+1;
 end

  end
endmodule
```

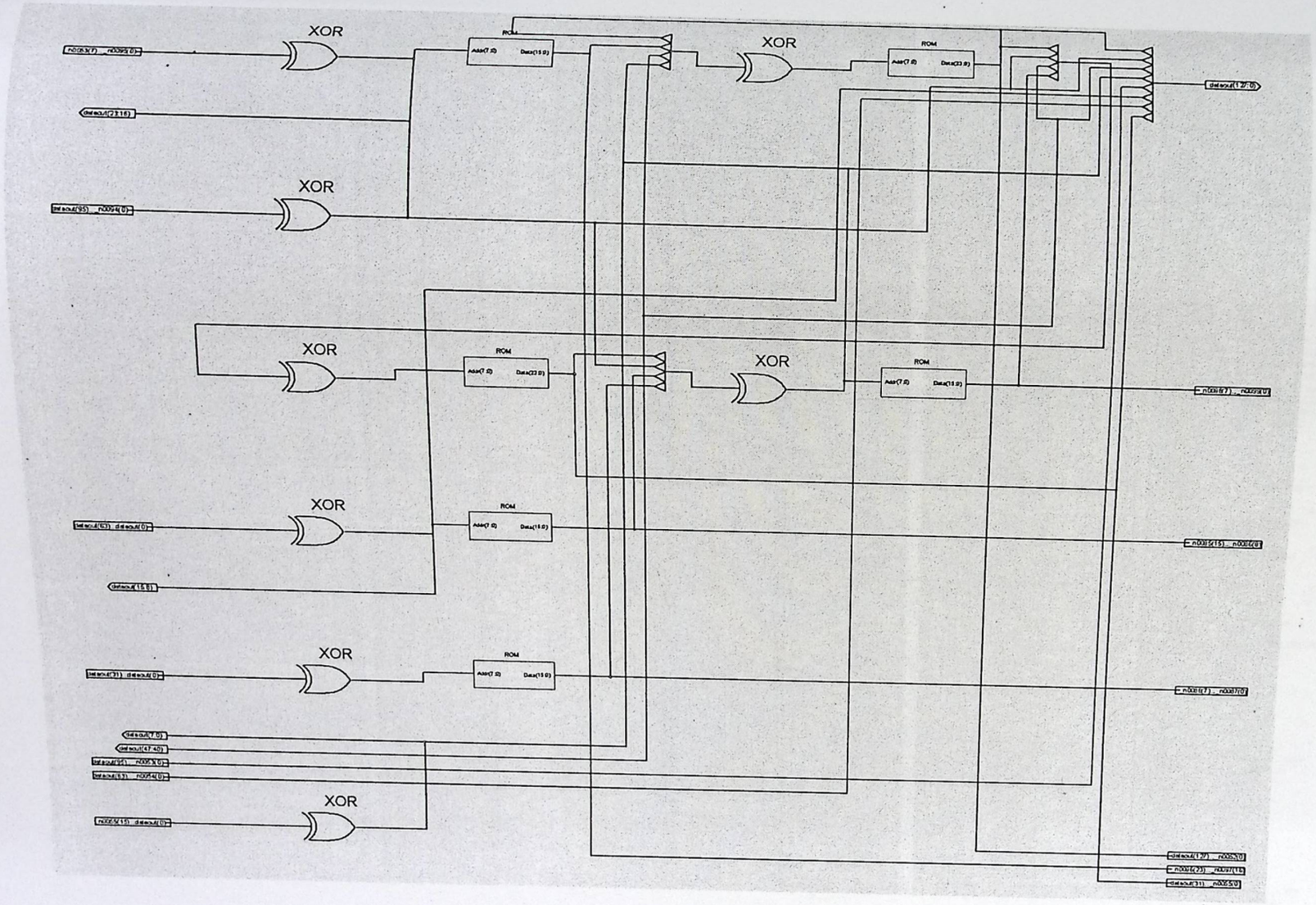
```
/********************** receiver*****************************/
module rx(In,Out,CLK,Clr);
input In,CLK,Clr;
output Out;
reg[3:0] N;
reg  Out,e;
always @ (posedge CLK,!Clr)begin
 if(N==0) begin if (In == 0) begin e=1; N=N+1;end  end
 else if (N==9)begin if (In==1) begin e=0;N=0;end end
 else  if(e) begin
 Out=In;
 N=N+1;
 end
  end
Endmodule
```

Appendix C

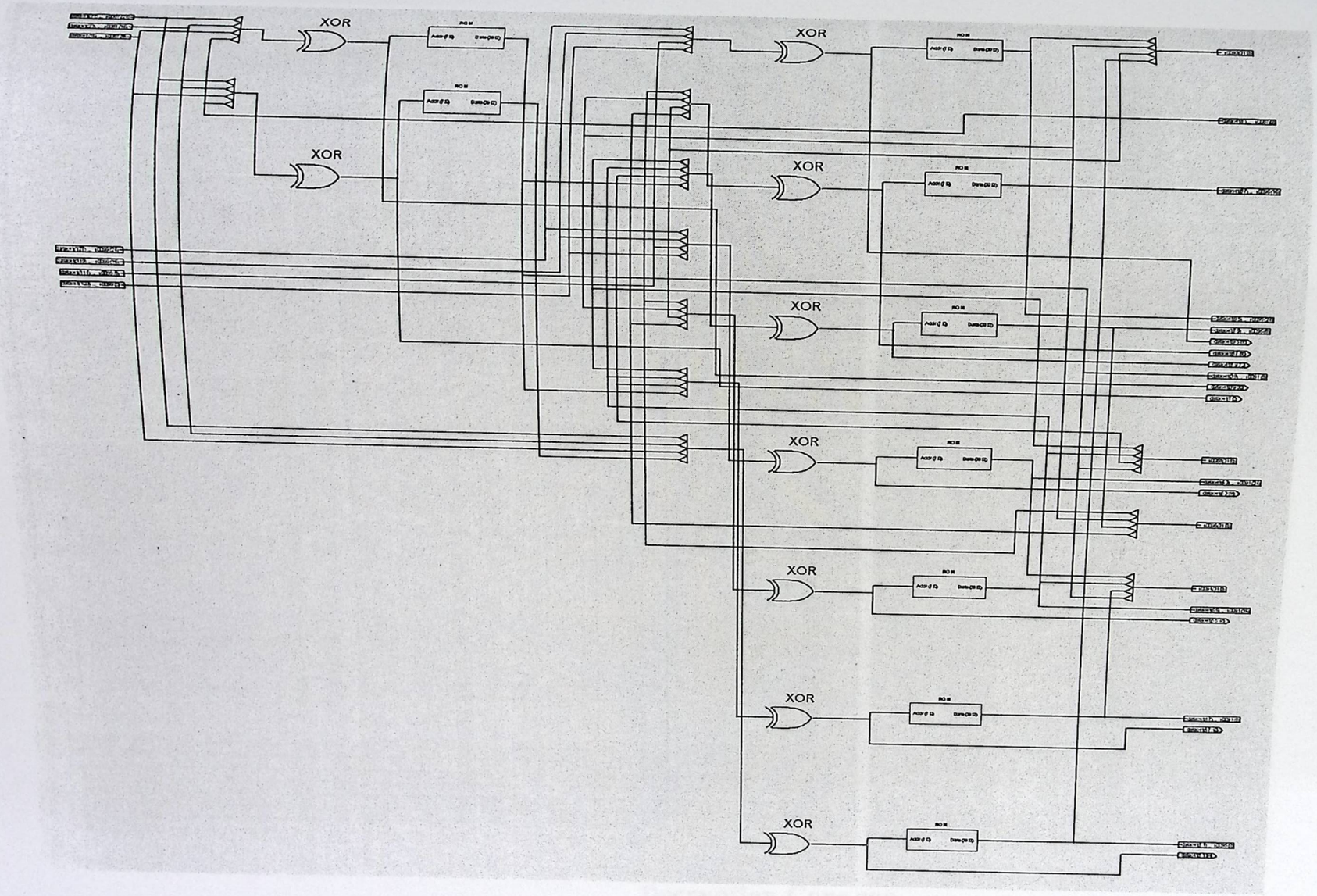
The schematic design of the system



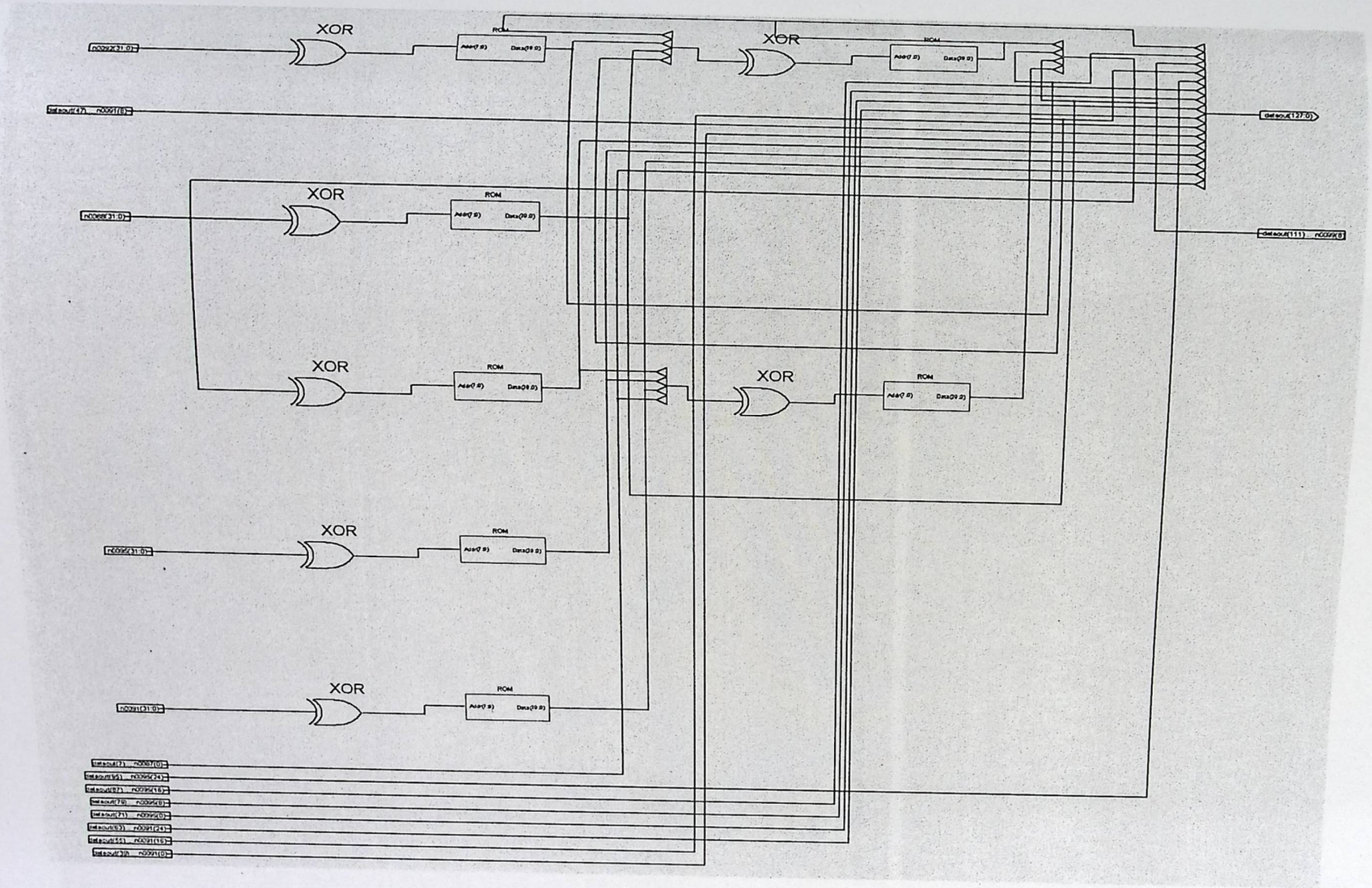
Encryption core



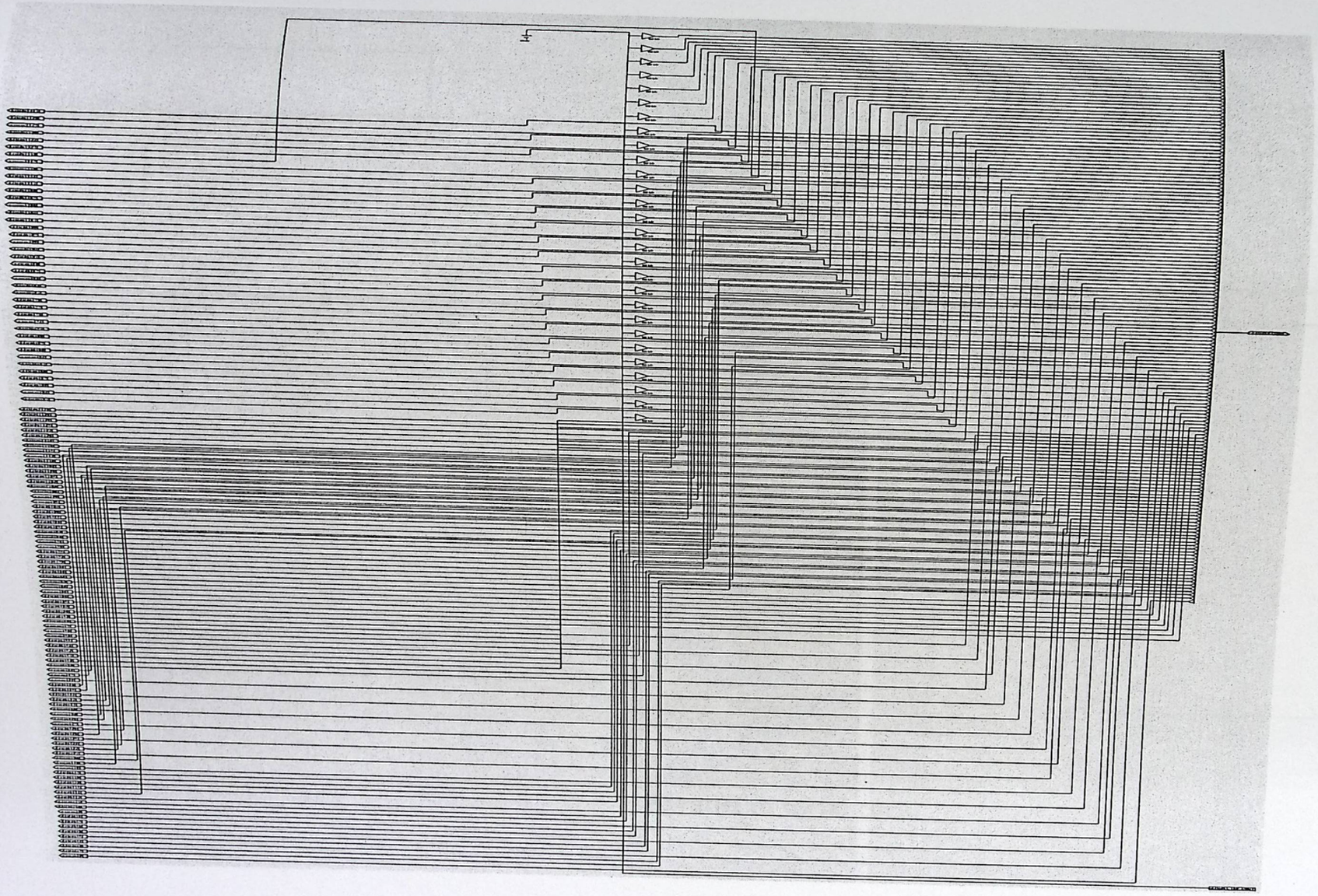
Encryption core con.



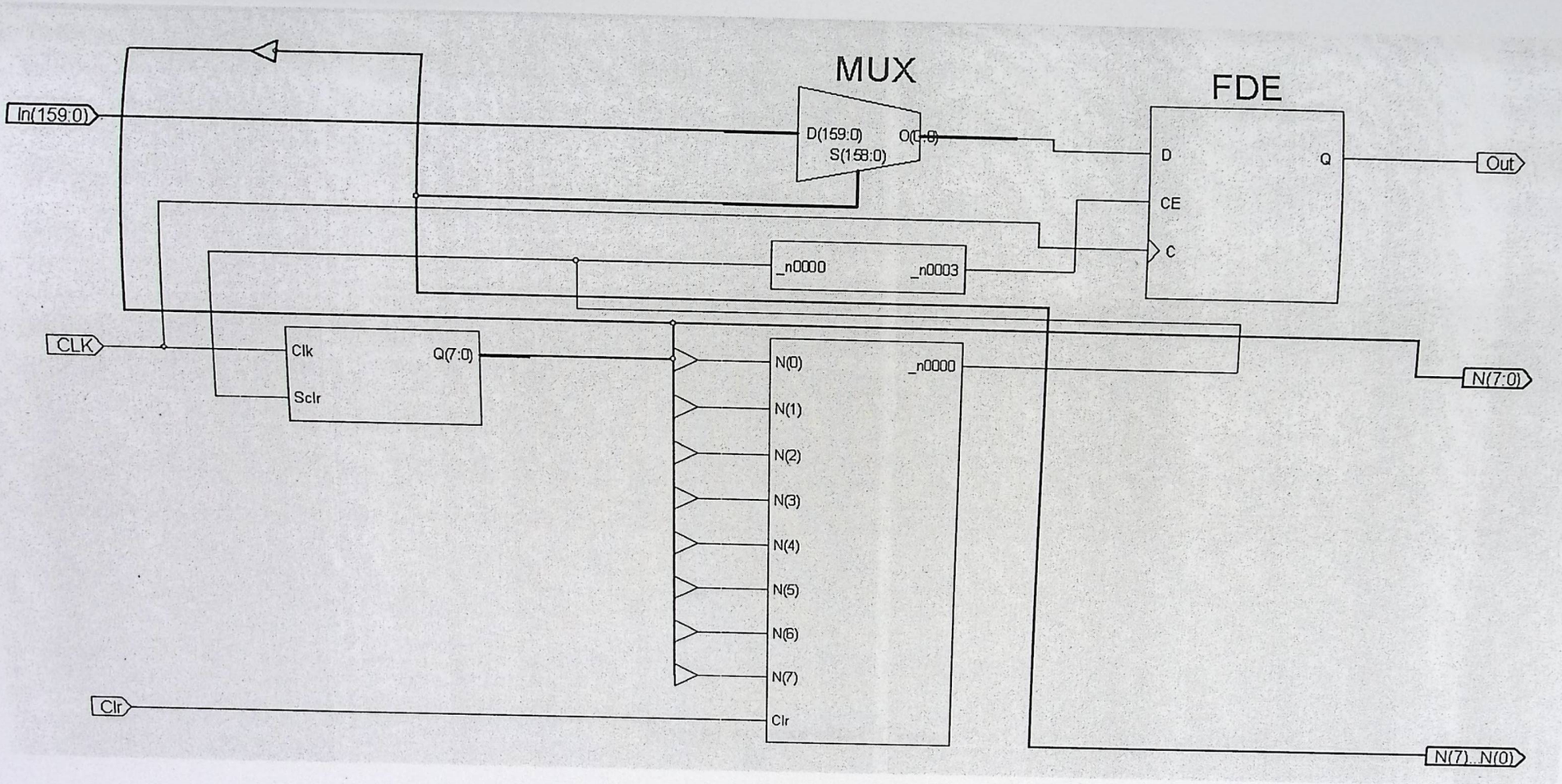
Decryption Core



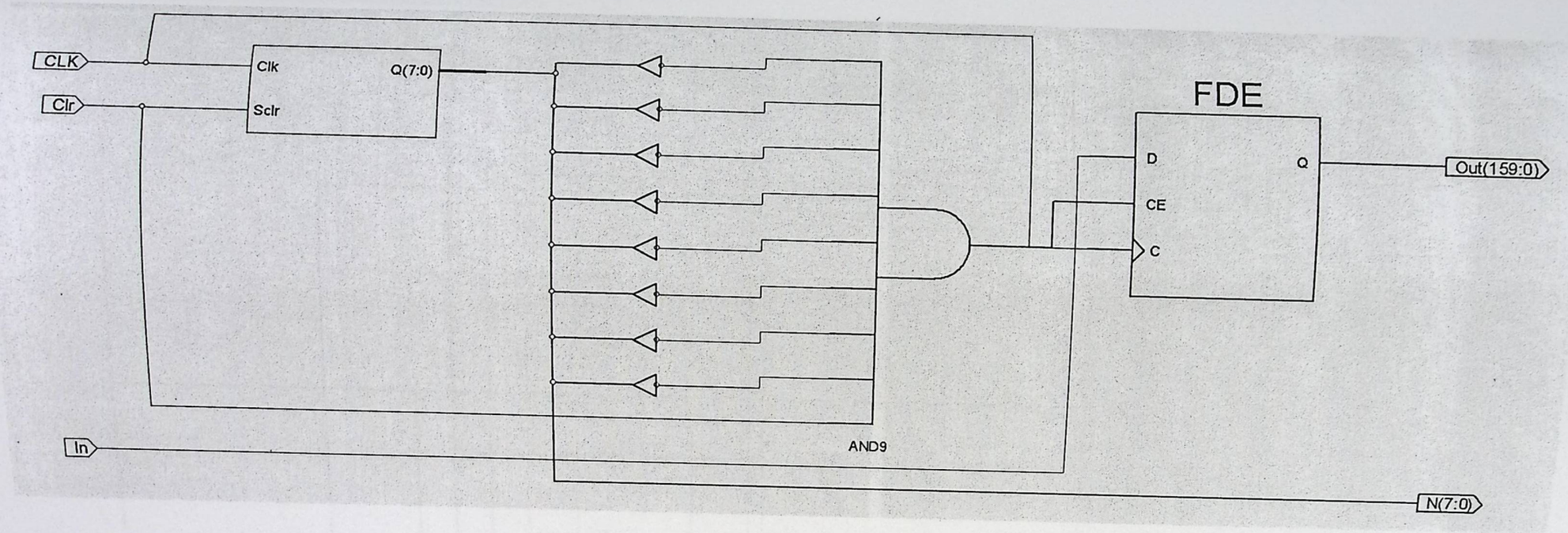
Decryption Core con.



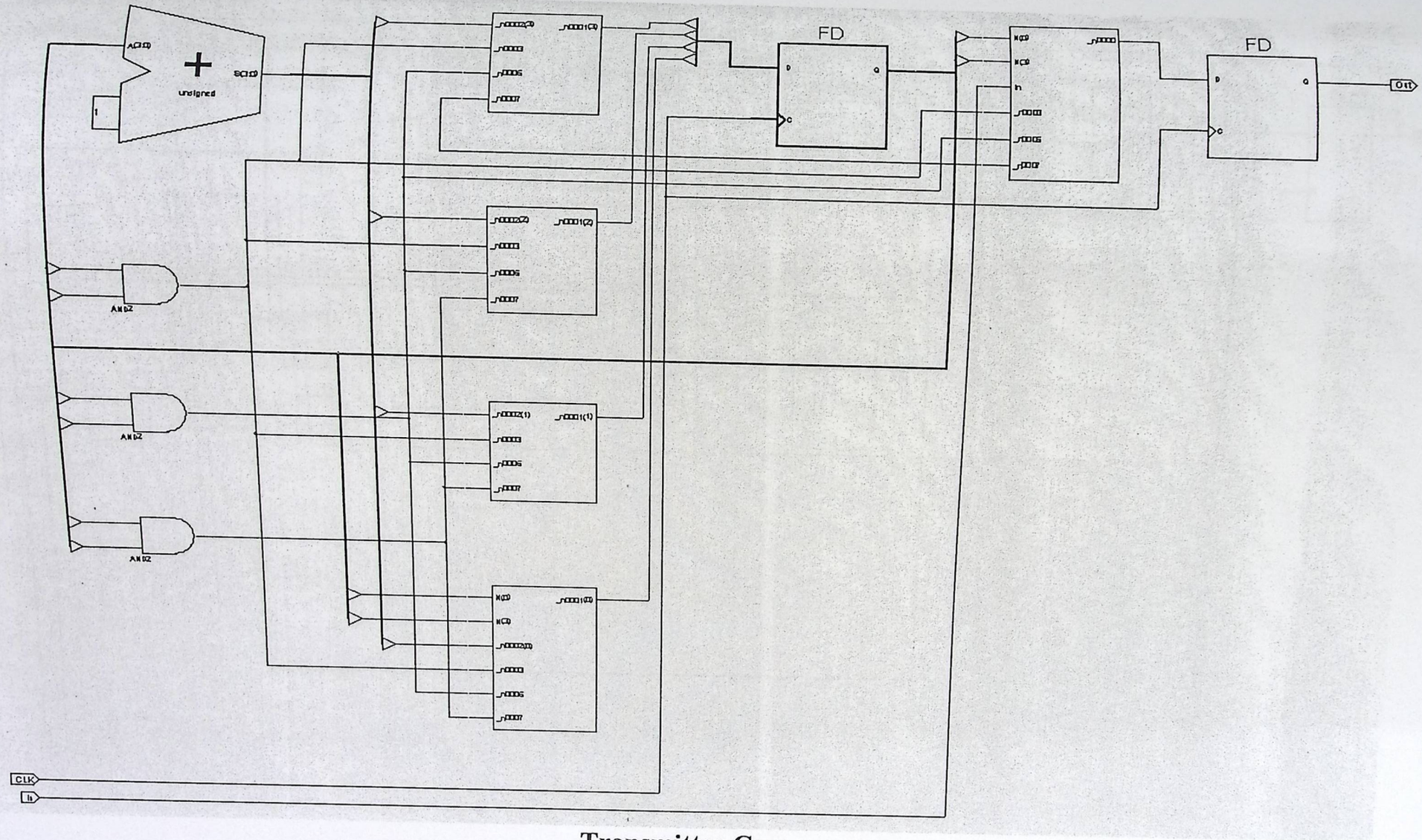
Compression/decompression core



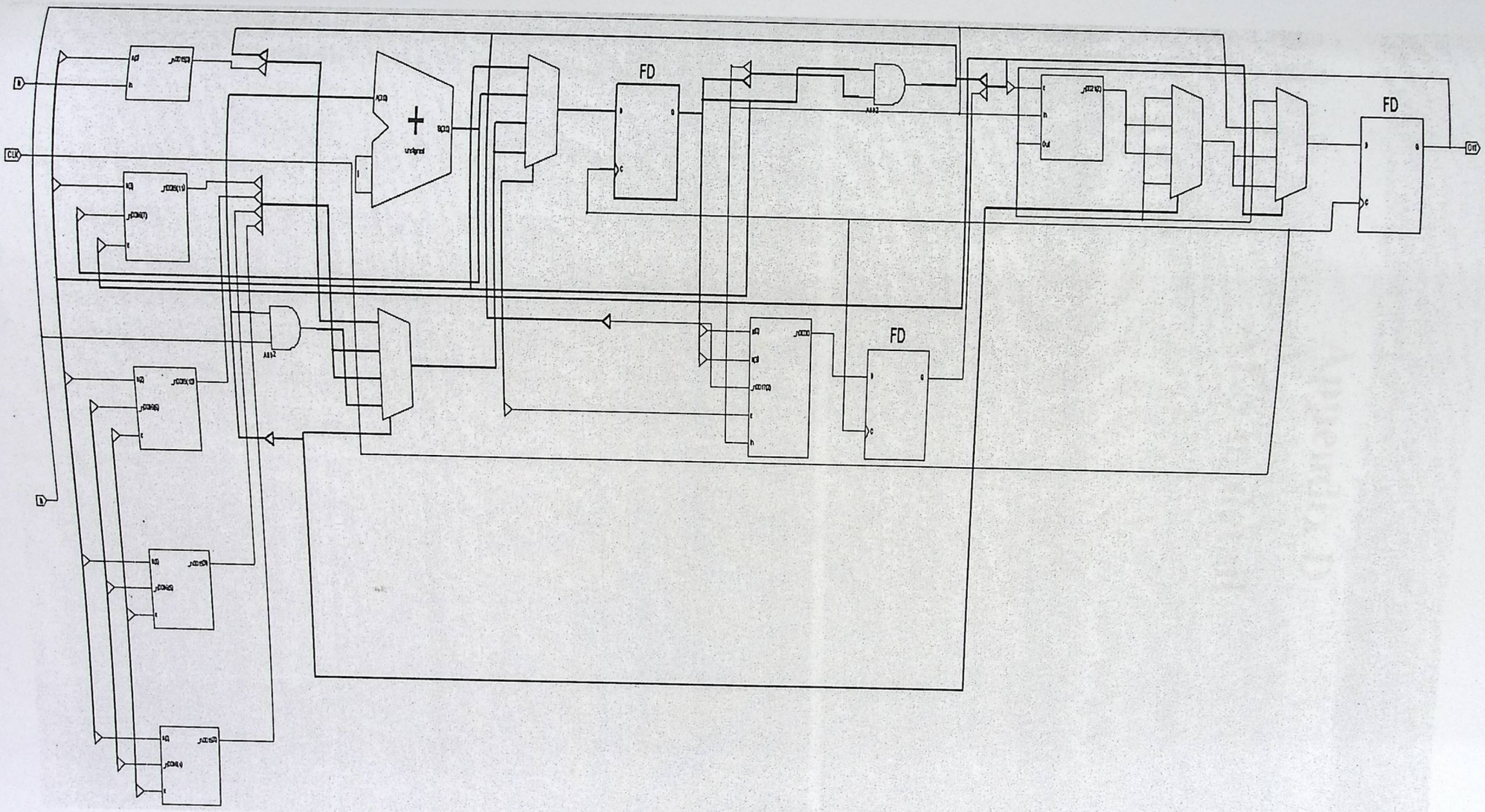
Parallel to serial core



Serial to parallel Core



Transmitter Core



Receiver Core

Appendix D
AES in detail

How AES Works

AES is designed to work on bytes. However, each byte is interpreted as a representation of the polynomial:

$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$

Where each b_i is either 0 or 1.

Addition and Multiplication

Addition then becomes exclusive-or, but multiplication is defined as polynomial multiplication modulo $x^8 + x^4 + x^3 + x + 1$. For example **2d * a3** would be calculated as follows (remembering $x^y + x^y = 0$):

2d = 00101101 =x5 + x3 + x2 + 1

a3 = 10100011 = $x^7 + x^5 + x + 1$

2d * a3 = $(x^{12} + x^{10} + x^9 + x^7) + (x^{10} + x^8 + x^7 + x^5) + (x^6 + x^4 + x^3 + x) + (x^5 + x^3 + x^2 + 1)$

= $x^{12} + x^9 + x^8 + x^6 + x^4 + x^2 + x + 1$

- modulo * $x^4 = x^9 + x^7 + x^6 + x^5 + x^2 + x + 1$
- modulo * x = $x^7 + x^6 + x^4 + 1$

2d * a3 = 11010001 = d1

Although this seems not efficient, all multiplications are by a constant, so they can be calculated in advance and turned into a simple table lookup.

Algorithm State

The 128-bit state can be represented as a 4 by 4 table of bytes. The cipher will perform various operations on this array.

Encryption Algorithm (128-bit version)

```
Cipher(byte in[16], byte out[16], word w[44])
begin
byte state[4,4]
state = in
AddRoundKey(state, w[0, 3])
for round = 1 step 1 to 10
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*4, (round+1)*4-1])
end for
SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[40, 43])
out = state
end
```

SubBytes Routine

In this routine, each byte of the state is replaced according to the following formula:

For each bit i, set b_i to:

$b_i \; xor \; b_{(i+4) \bmod 8} \; xor \; b_{(i+5) \bmod 8} \; xor \; b_{(i+6) \bmod 8} \; xor \; b_{(i+7) \bmod 8} + c_i$ where c = 63 hex.

As with multiplication, this is usually implemented as a table lookup.

ShiftRows Routine

This routine modifies each row of the state matrix. The top row is not changed, the next row is rotated left one position, the following row two positions, and the bottom row three positions.

MixColumns Routine

This function mixes up the data in each column according to the following formulas:

- Set $s_{0,c}$ to $2*s_{0,c}$ *xor* $3*s_{1,c}$ *xor* $s_{2,c}$ *xor* $s_{3,c}$
- Set $s_{1,c}$ to $_{0,c}$ *xor* $2*s_{1,c}$ *xor* $3*s_{2,c}$ *xor* $s_{3,c}$
- Set $s_{2,c}$ to $s_{0,c}$ *xor* $s_{1,c}$ *xor* $2*s_{2,c}$ *xor* $3*s_{3,c}$
- Set $s_{3,c}$ to $3*s_{0,c}$ *xor* $s_{1,c}$ *xor* $s_{2,c}$ *xor* $2*s_{3,c}$

AddRoundKey Routine

This function does an XOR between each column of the state and a 32-bit word from the key schedule.

Key Expansion

The key schedule *w* is generated in the following form:

- The first four words (*w[0]* through *w[3]*) of the key are the incoming cipher key.
- To calculate *w[i]* for *i* from 4 to 43:
  - Set *temp* = *w[i-1]*
  - If *i* = 4, 8, 12, 16, ..., 40 (multiples of 4)
    - Rotate this word left one byte
    - Replace each byte (using the same substitution function as SubBytes)
    - Do an exclusive-or with the round constant *Rcon[i]*
  - Set *w[i]* = *w[i-4]* xor *temp*

AES Decryption

Decryption basically consists of performing each of the encryption steps in reverse, using the following algorithm:

```
InvCipher(byte in[16], byte out[16], word w[44])])
begin
byte state[4,4]
state = in
AddRoundKey(state, w[40, 43])
for round = 9 step -1 downto 1
InvShiftRows(state)
InvSubBytes(state)
AddRoundKey(state, w[round*3, (round+1)*3-1]) InvMixColumns(state)
end for
InvShiftRows(state)
InvSubBytes(state)
AddRoundKey(state, w[0, 3])
out = state
end
```

Each of the `inverse` functions is the inverse of the corresponding encryption functions. `InvSubBytes` becomes another table lookup, and the equations for `InvMixColumns` are:

- Set $s_{0,c}$ to $0x0e*s_{0,c}$ xor $0x0b*s_{1,c}$ xor $0x0d*s_{2,c}$ xor $0x09*s_{3,c}$
- Set $s_{1,c}$ to $0x09*s_{0,c}$ xor $0x0e*s_{1,c}$ xor $0x0b*s_{2,c}$ xor $0x0d*s_{3,c}$
- Set $s_{2,c}$ to $0x0d*s_{0,c}$ xor $0x09*s_{1,c}$ xor $0x0e*s_{2,c}$ xor $0x0b*s_{3,c}$
- Set $s_{3,c}$ to $0x0b*s_{0,c}$ xor $0x0d*s_{1,c}$ xor $0x09*s_{2,c}$ xor $0x0e*s_{3,c}$

The algorithm can be rewritten so it looks similar to the encryption algorithm, with a few simple modifications.