# Parallelised Decipher Engine

## Implementation of a cut-down AES encryption system on FPGA


**Prepared by:**

**Killian T. Mazibuko**

**MZBKIL001**

Department of Electrical Engineering
University of Cape Town

**Prepared for:**

**Dr. Simon Winberg**

Department of Electrical Engineering
University of Cape Town

**October 2012**

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Electrical and Computer Engineering.

**Key Words:** AES encryption, Cipher text, Plain text, Round transformation and Encryption key.

# Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this final year project reportfrom the work(s) of other people, has been attributed and has been cited and referenced.
3. This final year project report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof

**Name:** **Killian T. Mazibuko**

**Signature:** _____ **Date:** 20 October 2012

# Terms of Reference

Towards the end of July 2012, instructions were given to embark on a three-month project to develop a cut down parallelised encryption system which can be run on an FPGA platform.

I am expected to do the following during the course of the project:-

1. Do research on the mathematics involved in a typical encryption system and understand how the encryption system works. Select suitable hardware platform and encryption system.

2. Come up with a suitable design of an encryption system that is cut-down to fit a low area FPGA and still be able to produce encrypted data.

3. Implement a PC version of a working full version of the encryption system.

4. Implement a PC version of the cut-down design that is to be implemented on FPGA before being parallelised i.e. the gold version

5. Implement the FPGA version of the cut-down version of AES which has been named KES for the purposes of this project.

6. Compare the FPGA version with the cut-down version to determine the merits of parallelism on such a hardware platform.

7. Select suitable metrics for comparisons in terms of encryption strength and make suitable conclusions. The cut-down version is compared to the full version.

# Acknowledgements

God – for providing me with strength to do the work I was required to do.

Dr. S. Winberg, my supervisor - for the help and guidance that he has provided for this project.

My family - for providing me with support and funds to complete my studies.

My colleagues in the Digital Image Processing lab - for providing a conducive environment for me to work well and do what I was required to do.

# Abstract

This project is about the implementation of an encryption system which is parallelised on an FPGA platform. FPGA is a reconfigurable hardware platform that allows designers to build complex logic system and it allows parallelisation. This enables tasks to be done concurrently thus achieve a higher throughput of data. Encryption is the process of converting files to cryptic data known as cipher text. This secures the data. Decryption will require a key to decode the information. In this project, a suitable encryption process is investigated and implemented. This method is cut-down to fit a low area and performance comparisons are made between the one implemented on PC which executes serially and the one which is parallelised. Comparisons are also made between the full encryption version and the cut-down version in terms of speed and encryption power.

The system implemented in this project is thus cut-down to fit a low area. This project describes therefore the KES (Killian Encryption System) which is a cut-down version of the known selected encryption system. This KES encryption is implemented on a PC in a sequential manner. It is then implemented on an FPGA platform and comparisons are made to see the merits of parallelism. Correlation methods are used to compare cipher text and input text known as plain text. Comparisons are made to see how strong KES is compared to the selected encryption system.

Recommendations are made and conclusions drawn from the experiments done.

# Acronyms, Definitions and Symbols

| | |
|---|---|
| **ASIC** | **Application Specific Integrated Circuit** |
| **AES** | **Advanced Encryption Standard** |
| **KES** | **Killian Encryption Standard** |
| **XOR** | **Exclusive OR** |
| **FPGA** | **Field Programmable Gate Array** |
| **Cipher Key** | **Pass phrase or word used to encrypt data** |
| **Cipher Text** | **State output after encryption process** |
| **Plain Text** | **Sate input before encryption process with original data** |
| **State array** | **A 4 by 4 matrix used to store the 128 bits that AES encryption process operates on** |
| • | **Finite Field Multiplication** |
| **VHDL** | **Very High Speed Integrated Circuits Hardware Description Language** |
| **RFID** | **Radio Frequency Identification** |
| **RAM** | **Random Access Memory** |
| **DES** | **Data Encryption Standard** |

# Tables of Contents

# List of Figures

# List of Tables

# 1. Introduction

## 1.1 Background to the study

Encryption and cryptography plays a significant role in embedded systems these days. Data transfers are increasing with number of devices also increasing. These transfers require security of data being transferred. There are many encryption algorithms available which will be investigated in Chapter 2. These include the ones widely known i.e. the Data Encryption Standard (DES), Advanced Encryption Standard (AES) and Triple DES [16].

Towards the end of the 1990's, the US government called for a new encryption standard to be used to protect and secure information. This came after many loopholes had been discovered in the DES designed in 1977 which was widely used that time. So at the end of it all, Rijndael which was designed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted to the AES selection process, was selected to be their new standard. This was later known as the Advanced Encryption Standard and is now widely used to secure and protect information. Embedded systems also require protection of information and thus systems need to be designed that can implemented this encryption system on low cost and low area hardware platforms. This project investigates the encryption methods available including AES mentioned above. It then implements an encryption system on a personal computer first and then parallelizes it and implements it on hardware.

Encryption systems are classified into Symmetric and Asymmetric systems [35]. This depends on the key used. Asymmetric systems have a public key and a private key. The key is a sort of a password needed to secure the information. Symmetric systems only have one key. The Advanced Encryption Standard (AES) which is widely used today falls into this category.

This project implements a cut-down version of the chosen encryption standard which will be known as the Killian Encryption Standard (KES) for the purposes of the project. This system will be designed to fit a low area and low cost FPGA. FPGA is a reconfigurable hardware able to implement complex logic functions. It is also parallelised to capture the advantages of it over a similar system that is implemented on a PC. The encrypted text that is output on this KES will be compared to the input so that performance can be measured. This will also be done for the full version of the chosen encryption process. Suitable performance measurement parameters will be chosen and implemented to make suitable conclusions.

## 1.2 Objectives of this study

### *1.2.1 Problems to be investigated*

Embedded systems on low area and low cost hardware need to be fast to be able to meet real time needs. These systems also need to be designed to protect and secure information. These can be used in Wireless sensor networks, Wi-Fi systems or even card reader systems which handle sensitive information that needs to be secured. Data encryption is therefore necessary and needs to be done fast enough to meet real time demands of these systems. Data parallelism promises a lot of advantages over systems that runs sequentially on a PC. These issues are investigated in this project.

The questions to be answered are as follows:-

a) Is it advantageous to implement an encryption system on FPGA in terms of speed and performance?

b) Cutting down a full encryption version to fit a low area sacrifices encryption power? This shall be investigated

c) Does cutting down some parts of an encryption system bring improvement on speed and performance on a low area platform?

### *1.2.2 Purpose of the study*

It is relevant to do this study because data security is crucial in order to protect people. In the wake of the Wiki leaks and other reported cases of sensitive data and information leaking out to the general public; it is obvious that data security is crucial. The encryption must be done to ensure that data cannot leak to those who want to access it illegally. The encryption system must also execute within reasonable time so as to fit a system that may operate in real time. This project investigates the performance of implementing an encryption system on a hardware platform like an FPGA. The encryption system implemented in cut-down in order to fit a low area. Performance comparisons are crucial to determine the strength of the encryption system and also find out the merits of implementing the process on a hardware platform like an FPGA.

## 1.3 Scope and Limitations

The scope of the implementation on FPGA will be limited to the encryption process. Constructs are designed in the FPGA system to allow the quick implementation of the decryption process. However, due to time limitation, it was not possible to implements both the encryption process and the decryption as well on FPGA. However, implementations of these are available on a PC platform. The project will also be limited to implementation on a hardware platform like the FPGA.

## 1.4 Plan of development

The structure of the report is presented as follows:-

- Chapter One presents an Introduction to the study and background information. It also gives direction on what content is in the report

- Chapter Two presents the background study and Literature Review. It presents information searched to inquire on what hardware platform and encryption method to use before presenting the encryption process chosen.

- Chapter Three presents the Methodology. This shows the process done to conduct this research and some basic theory development of the procedure chosen.

- Chapter Four presents the detailed design of the system implemented on Personal Computer and on FPGA.

- Chapter Five presents the Results of the experiments done.

- Chapter Six discusses the experiments done so as to aid the process of making suitable conclusions.

- Chapter Seven presents the Conclusions made from the experiments.

- Chapter Eight presents Recommendations for future work.

# 2. Literature Review

## 2.1 Implementation of Encryption on FPGAs

### *2.1.1 Review of encryption methods*

A number of encryption methods were analysed to see which one could be implemented on FPGA. The following table shows how difficult it is to break the different types of encryption systems.

NUMBER OF SECONDS REQUIRED TO BREACH AES, DES AND TRIPLE DES ALGORITHMS.

| Key Length (bits) | DES (Seconds) | Triple DES (Seconds) | AES (Seconds) |
|---|---|---|---|
| **8** | 0.73 | 0.95 | 1.04 |
| **16** | 7.19 | 35.39 | 74.75 |
| **24** | 1330.84 | 2085.03 | 3081 |
| **32** | 2900.65 | 4252.33 | 8758.65 |
| **40** | 5768.67 | 9947.67 | 19141.12 |
| **48** | 9765.45 | 20086.58 | 41750.54 |
| **56** | 25789.12 | 42294.54 | 154277.87 |
| **64** | 60789.12 | 98294.54 | 554277.87 |

**Figure 2-1 Number of second to break encryption algorithm**

*a) Data Encryption Standard (DES)*

Before AES encryption was used, DES was another encryption method prevalent. It was designed in 1975 and was widely used. According to [16], it takes 64 bits at a time followed by an initial permutation. After 16 rounds are done on the data before final permutation is done. For a round, the 64 bits are split into; left and right. The right side goes through a function f mixed with key before being added to the left side. The sides are then swapped and it proceeds to the next stage. The right side is then expanded from 32 to 48 bits. 48 bits are added from the key. S-boxes reduce each state 6 bits to 4. After that, final permutation is done before the output comes out.

According to [17], it can be shown as also displayed on the figure above that it takes very few seconds to breach the algorithm. The algorithm is therefore not strong enough to withstand modern day attacks and will not be used in this project.

*b) DES Triple*

This algorithm is three encryption of DES to increase safety level. It is an enhancement of DES. It has been verified that this method is slower than other block ciphers [18]. As speed and throughput is such a crucial factor, it will not be used in this project considering that there are other faster and safer methods available.

*c) AES encryption*

This is the encryption method that will be used in this project. As shown in the figure, above it is safer than the above named ones which have been implemented on hardware before and it is also faster. This is very crucial in real time applications where throughput and latency are critical.

### *2.1.2 Review of Hardware Description Languages*

Two main Hardware description languages used in industry are Verilog and VHDL.

*a) Verilog*

Verilog is mainly based on C programming like constructs. It consists of about 50% C and 50% Ada in terms of its constructs and their similarity to other programming languages [19]. A C programmer would prefer it more than VHDL. Given the time available for this project, it would take time to get to the grips of this language before one can use it for a complex logic design like an Encryption system. The third year Computer Hardware course given in the department mainly focussed on VHDL. It would be therefore much appropriate to utilise VHDL for this project given the fact that there are a lot of designs that have been implemented. This provides a wealth of information to refer to.

*b) VHDL*

VHDL stands for Very High Speed Integrated Circuit Hardware Description Language. It specified the operation of logic systems. The students in the Electrical Engineering Department of UCT are more familiar with it and therefore it comes very easy and handy to use. However given more time, Verilog might be a worthwhile option considering its similarity to C programming style.

According to [19], you may place procedures and functions in a package in VHDL. This allows re-use in other modules. The idea of packages is not available in Verilog. For this project which is quite big, declaring a package at some point might be necessary.

The reasons given above mean that VHDL will be utilised to produce the encryption system for FPGA in this project.

### *2.1.3 Application of the encryption system*

For a small lightweight system like that which uses RFID (Radio Frequency Identification) tags, a low area 8 bit encryption system might be sufficient. This is the system that is going to be implemented in this project. A design like the one shown in [3] allows on the fly implementation of Round key calculation for AES encryption. This can save a lot of memory. Third year students in the Electrical Engineering department have recently been doing a project on the automation of loading and unloading of ships at Cape Town harbour. This can be done 24 hours a day allow an improvement in the throughput of ships at the harbour [20]. A lot of students identified the use of RFID tags to identify containers to be unloaded and unloaded as a possible addition to automate the system. Because this information needs to be secured, an encryption system like the one implemented in this project might be utilised. This will allow the use of locally produced systems which will support local engineers and minimise system costs.

Other application of encryption systems are documented in literature. This includes e-commerce servers which require high throughput and faster systems [7]. Obviously this will require implementation on more costly hardware platforms which are not available in the University of Cape Town Electrical Engineering Department. This might be considered in future. Other applications may be in cell-phones [6], point of sale hardware to encrypt personal data like credit card / debit card details.

In [20], applications in wireless sensor nodes and other computer networking hardware are given. This could be another area of application of this project.

### *2.1.4 Choice of FPGA platforms*

A number of FPGA platforms are available to implement encryption. The design shown in [11] shows that a small 32-bit encryption system can take about 148 slices on a Spartan 3. This is quite modest and fits the FPGA platforms available in the Department of Electrical Engineering at UCT. Most research projects in the department are using the Rhino board FPGA. This is based on the Nexys architecture which the Spartan 3 is based. It therefore is quite feasible to implement the encryption system on a Spartan 3. The boards are acquired at a reasonable price by the University which is affordable given the fact that the money allocated for this project is fixed at a level where buying extra FPGAs can be impossible.

The document in [23] shows that Xilinx offers better clock management that Altera based FPGAs. It also shows that a basic Spartan 3 has about 192 CLBs. One CLB makes four slices. Given this area size, this could accommodate a small encryption system like the one mentioned in [11].

The following figure table analyses the requirements of a typical encryption system on FPGA

| Design | Encryption only | Decryption only | Encryption and decryption |
|---|---|---|---|
| Slices | 626 | 645 | 830 |
| 4input LUTs | 858 | 872 | 1452 |
| BRAM | 14 | 14 | 14 |
| Latency (ns) | 264 | 281.6 | 321.8 |
| Max clock (MHz) | 166.7 | 156.2 | 136.7 |
| Throughput (Mbps) | 2133.76 | 2000 | 1749.76 |
| Efficiency (Mbps/slices) | 3.41 | 3.1 | 2.1 |

**Figure 2-2 Processing requirements of AES on FPGA [24]**

The above figure shows a typical encryption system that takes about 830 slices for encryption and decryption. The throughput is reasonably high and it also utilises LUT (Look up tables). Comparing with the slices given in [23] for a Spartan 3, it shows that a Spartan 3 would be fairly adequate for a cut-down version of an encryption system which is capable of accommodating Encryption and decryption. The system would be cut-down to fit a low cost FPGA.

## 2.2 Building blocks of encryption in FPGA

The AES encryption algorithm can be built in hardware and can operate in FPGAs or ASICs. This can be used for RFID cards, cell phones, ATM Machines or other embedded and special applications. A number of literatures were read to see how the logics of the system can be implemented in hardware.

In e-commerce servers, we need high throughput and the encryption can be implemented in a way that allows high throughput. Other applications like RFID cards can do with low area and low power at the cost of throughput. Medium throughput is available for cell phones [6]. A number of designs will be presented in this section which show how the encryption process is implemented.

### *2.2.1 High throughput applications*

Key sizes are used depending on the application and the strength required. For applications in secretive military operations, 256-bit key can be used [7]. For high throughput designs, we can double the throughput by using the loop unrolled architecture. The stages of AES encryption are replicated in hardware. The following diagram shows how the loop unrolled with pipelining is implemented in hardware. This is what allows the throughput to increase. This comes at a cost of area. We can also increase the pipelining stages. Instead of doing it after every stage, we can have divide each stage into a number of pipeline stages. This sub-pipelining can achieve high throughput in the range of tens of gigabyte which is particularly useful in e-commerce applications and their related servers [7]. Another implementation which includes a pipelined architecture is shown in [10]. It achieves a throughput of 28.4 GBits/s for the fully pipelined and 876 MBits/s for the sequential design.

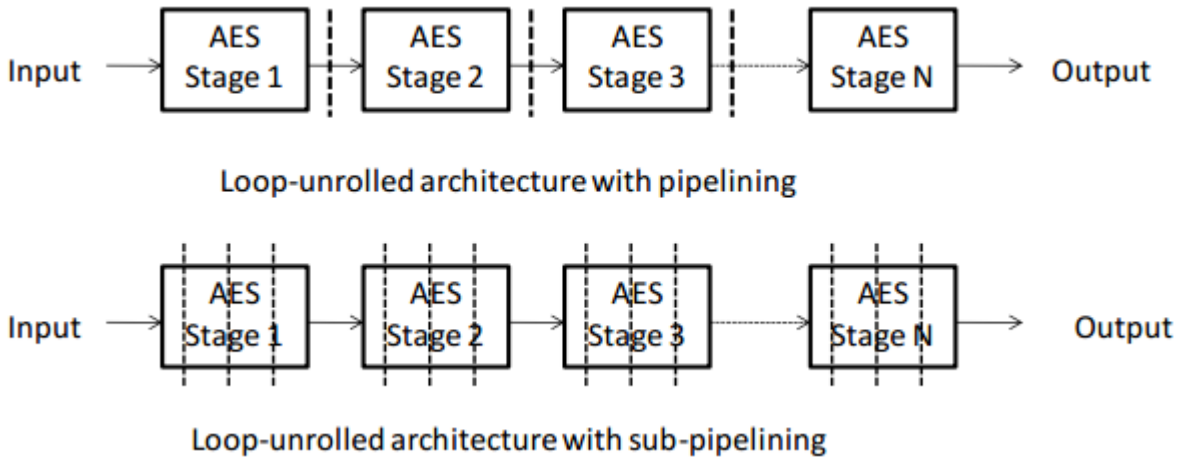


**Figure 2-3 AES encryption implementation of pipelining and sub-pipeline for high throughput**

### *2.2.2 Medium throughput applications*

Whilst high throughput applications may be desirable, they are quite costly as they require a large datapath of 128 bits. One might need medium throughput which can be implemented in low area and low power implementations. In this kind of application, the data-path is usually 32-bit but can also be 8-bit at the cost of more clock cycles and reduced throughput. The design in [5] shows a 32 bit system

which can achieve reasonable throughput of 876 Mbps. In this design the round keys from key expansion are stored in a BRAM and 2 shift registers are used to do shift rows.

This design uses 156 slices and the basic structure is shown below:-

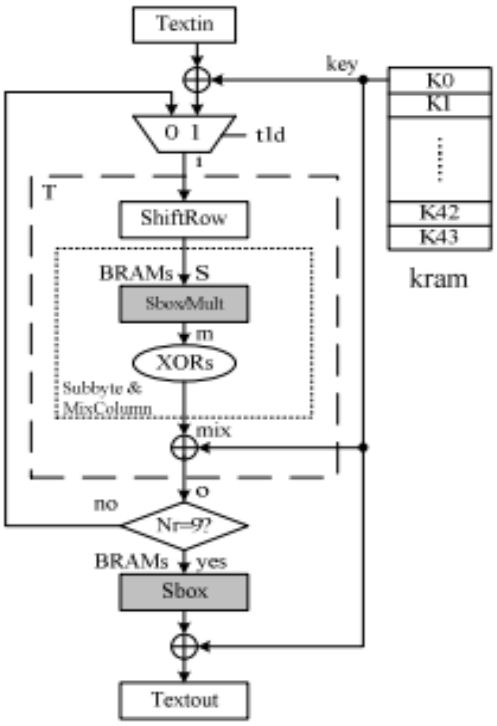


**Figure 2-4 32-bit implementation of AES encryption**

This shows that the keys are stored in a kram and retrieved when required. This obviously increases area. Look up tables are used to store the s-box required for sub bytes. A 44x32 dual-port BRAM is used for storing each key.

Another 32-bit implementation is also shown in [8]. In this design, tables are used to combine the function of mix-columns and sub-bytes. However, this depends on specific features of the Virtex-5 and the portability of the system may be difficult. Another one is shown in [11] which can be implemented on a Xilinx Spartan 3 platform. It uses block RAM to store key expansion keys. It achieves a steady throughput of 647 Mbps whilst using 148 slices. The one shown in [12] provides on the fly key expansion system which can be used for both encryption and decryption. A fairly high throughput of 768 Mbps can be achieved. However, this comes with some cost with regards to area.

The design shown in [13] achieves a steady throughput of 208 Mbps in a Spartan 3 FPGA. This design uses 3 RAM blocks and a 32-bit datapath. It also combines SubByte and MixColumn by making use of the dedicated 18-Kbit dual-port RAM blocks in Spartan-3 and Virtex-II. Even though this would be a suitable design for this project, its portability to other families of hardware could be compromised if it relies on the specific features mentioned.

The document shown in [14] contains helpful material that helps to build the state machine for the encryption core. A helpful documentation that helps in building an AES encryption system in FPGAs and ASIC was found in [15]. It gives guidelines how throughput and latency are calculated for the

iterative structures. It also contains various logic structures explaining how to build an AES encryption hardware core. The building up of pipelined cores is also adequately explained.

### *2.2.3 Low area and low power architecture*

For low area and low power application as in RFID card systems, we might sacrifice some clock cycles to save area. An eight-bit design is shown in [3]. This is particularly suited for low area and low power applications. One strength about this system is that it allows parallel operation of the key expansion unit along other AES operations. This allows the throughput to increase compared to other 8-bit systems.

This receives one byte at a time for data and key. A parallel-to-serial converter has to be used because mix columns operation produces a 32-bit word output which needs to be serially shifted into the unit for byte permutation. This system achieves a reasonable throughput of about 200 Mbps.

The eight-bit and 32-bit systems use an iterative design which goes in cycles till it finishes the encryption process compared with the loop unrolled and pipelined architectures described previously which are mainly used for very high throughput designs.

## 2.3 Essential Mathematics for AES

To implement the AES encryption algorithm, one has to understand the mathematics involved in translating the data from plain text to cipher text. A lot of literature was read to get this information so as to be in a position to implement the AES algorithm. This section explains this mathematics as it is crucial to understanding how the AES algorithm operates. The main document published shown in [1] represents the most basic literature available to understand how AES encryption operates.

### *2.3.1 Galois field*

The Galois field used in AES encryption is written as GF ($2^8$). This is a finite field that has a fixed of finite number of elements as defined in algebra. Finite field written as GF(a) denotes a field with integers that range from {0, 1, …, a-1}. The number of elements in total gives the order of the field. If the finite field has non-prime number integer orders, the finite field is denoted by prime number 'p' put to the power of n. The equation k = GF($p^n$) denotes a finite field with order k whilst using prime number p which is put to the power of n. In AES encryption, each data byte represents a range from $(00\text{-}FF)_H$. The representation of the byte as a polynomial is explained in the following section.

### *2.3.2 Bytes*

The basic element of computation in AES encryption is the byte. This is sequence of 8 bits grouped together. In finite field mathematics, this is represented by a group of polynomials like

$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1 x + b_0 ... 1$

For example {00011101} is represented as $x^4 + x^3 + x^2 + 1$

The equation shown above represents an 8 bit sequence with the coefficient of $x^7$ being the most significant. If the least significant bit is 0 then the last 1 is omitted. If any of the bits in between are zero then the power of x in that sequence is omitted. An eight bit sequence is represented in hexadecimal with braces like {02} for 2 in base 16 or {ae} for ae in base 16.

### *2.3.3 Addition*

Addition in finite field mathematics is performed by xoring individual bits on corresponding positions. XOR is a function that operates on two bits to return an answer. If the bits are opposite numbers i.e. the other one is 1 and the other 0, then it returns a one otherwise it returns a zero.

Adding finite field elements is characterised by modulo addition of bits that are corresponding [1].

An example is $(x^7 + x^5 + x^4 + x^3 + 1) + (x^6 + x^5 + x^3 + x) = x^7 + x^6 + x^4 + x + 1$ (in polynomial notation)
{10111001} xor {01101010} = {11010011} (in binary notation)
{b9} xor {6a} = {d3} (hexadecimal notation)

### *2.3.4 Multiplication*

Multiplication in GF ($2^8$) is done by multiplying polynomials modulo an irreducible polynomial of the eighth degree. It is said to be irreducible if its divisors are one and itself. In this AES encryption process, this irreducible polynomial is represented as:-

$m(x) = x^8 + x^4 + x^3 + x + 1$ which is {01} {0b} in hexadecimal notation. The {01} is added at the beginning to represent the first significant bit makes the whole sequence 9 bits.

For example {56} • {84} = {d0}

This is so because $(x^6 + x^5 + x^4 + x^2 + x)(x^7 + x^2)$ $= x^{13} + x^{12} + x^{11} + x^9 + x^8 + x^8 + x^7 + x^6 + x^4 + x^3$

$= x^{13} + x^{12} + x^{11} + x^9 + x^7 + x^6 + x^4 + x^3$

And $x^{13} + x^{12} + x^{11} + x^9 + x^7 + x^6 + x^4 + x^3$ modulo $(x^8 + x^4 + x^3 + x + 1)$

$= x^7 + x^6 + x^4$

We have to do the modulo operation to make sure that it is of degree less than 8 that a byte will represent.

The literature in [1] mentions the fact that if we have a polynomial a(x) of degree less than 8, there is a multiplicative inverse which is $a^{-1}(x)$. Extended Euclidean algorithm in [2] is used to find b(x) and c(x) so that:-

$A(x)b(x) + m(x)c(x) = 1$

From this $b(x) \bullet a(x) \bmod m(x) = 1$ therefore

$a^{-1}(x) = b(x) \bmod m(x)$. Equation 2

This expression is also true: $b(x) \bullet (a(x) + c(x)) = b(x) \bullet a(x) + b(x) \bullet c(x)$

### *2.3.5 Multiplication by x*

Multiplying by x will increase the exponent of each element in the polynomial by one. For example $a(x) = x^6 + x^5 + x^4$ becomes $x^7 + x^6 + x^5$.

Calculating x • a(x) given a polynomial will result in the result shown above modulo m(x). The polynomial is already in reduced form if bit with degree 8 is zero. If its 1 the result is found by xoring the polynomial m(x). The function xtime() represents this multiplication at byte level. This is done by a left shift followed by a conditional xor with {1b}. We can obtain multiplication of higher powers by repeated application of xtime().

In [1] the following example is provided where it shown how we obtain {57}•{13} = {fe}

This is so because {57}•{13} = {57}•({10} xor {02} xor {01})

We obtain {57}•{10} be deducing that:-

{57}•{02} = xtime({57}) = {ae}
{57}•{04} = xtime({ae}) = {47}

{57}•{08} = xtime({47}) = {8e}
{57}•{10} = xtime({8e}) = {07}

{57}•{10} = {57}•{10} xor {57}•{02} xor {57}•{01}
= {07} xor {ae} xor {57}
= {fe}

### *2.3.6 Four Term polynomials*

A 32-bit word can be represented by a four term polynomial. This is written as $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$. The coefficients are represented as [a3 a2 a1 a0]. Each coefficient will be representing a byte. Addition is done by xoring corresponding in a bitwise fashion. For example if we have another polynomial $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$. The result of addition is represented by $c(x) = (a_3 \text{ xor } b_3)\ x3 + (a_2 \text{ xor } b_2)\ x^2 + (a_1 \text{ xor } b_1)\ x + (a_0 \text{ xor } b_0)$.

For multiplication, this is done by first expanding and then collecting like terms. The result is an expression of degree six. This is represented according to [1] by :-

$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0.$

The coefficients are as follows:-

$c0 = a_0 \bullet b_0$ $\quad c_1 = a_1 \bullet b_0 \text{ xor } a_0 \bullet b_1$
$c2 = a_2 \bullet b_0 \text{ xor } a_1 \bullet b_1 \text{ xor } a_0 \bullet b_2$ $\quad c_3 = a_3 \bullet b_0 \text{ xor } a_2 \bullet b_1 \text{ xor } a_1 \bullet b_2 \text{ xor } a_0 \bullet b_3$
$c4 = a_3 \bullet b_1 \text{ xor } a_2 \bullet b_2 \text{ xor } a_1 \bullet b_3$ $\quad c_5 = a_3 \bullet b_2 \text{ xor } a_2 \bullet b_3$
$c6 = a_3 \bullet b_3$

However, this is of degree six therefore it has to be reduced to a polynomial of degree less than four by a modulo operation with a polynomial of degree 4. For AES algorithm, this polynomial is $x^4 + 1$. The final output of modular product is represented by $d(x) = d_3x^3 + d_2x^2 + d_1x + b_0$.

The output is represented by:-

$d0 = a_0 \bullet b_0 \text{ xor } a_3 \bullet b_1 \text{ xor } a_2 \bullet b_2 \text{ xor } a_1 \bullet b_3$
$d1 = a_1 \bullet b_0 \text{ xor } a_0 \bullet b_1 \text{ xor } a_3 \bullet b_2 \text{ xor } a_2 \bullet b_3$
$d2 = a_2 \bullet b_0 \text{ xor } a_1 \bullet b_1 \text{ xor } a_0 \bullet b_2 \text{ xor } a_3 \bullet b_3$
$d3 = a_3 \bullet b_0 \text{ xor } a_2 \bullet b_1 \text{ xor } a_1 \bullet b_2 \text{ xor } a_0 \bullet b_3$

In matrix form, given a fixed polynomial a(x), it is represented by:-

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

X4 + 1 is said to be not an irreducible polynomial over GF(28). This means if we multiply by a four byte polynomial, the process is not invertible. AES gave a fixed four term polynomial which is invertible. This is defined as $f(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.
The inverse is $f^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

In the key expansion algorithm to be explained later, there is a fixed term polynomial where $a_0$, $a_1$ and $a_2 = \{00\}$ and $a_3 = \{a3\}$. Multiplying by this effectively rotates the bytes by one place to the left. Given $[b_3\ b_2\ b_1\ b0]$, the output is $[b_2\ b_1\ b_0\ b_3]$. This function is called RotWord().

## 2.4 Encryption algorithm for AES

The following section will describe the AES algorithm in stages. The data that is to be encrypted is split into 128 bit sections. Each 128 bit section is encrypted using a key that is provided. The key can be 128 bits, 192 bits or 256 bits. The more the bits, the more the number of rounds that need to be performed on the data before cipher text comes out at the end. The first round involves xoring bitwise the 128 bits coming in with the initial key to form the state, which is intermediate 128 bits being processed before the cipher text comes out. This represents Round 0.

After that if the key is 128 bits, the state goes through nine rounds where the following steps are processed:- Shift rows, Sub bytes, Mix Columns and Add Round Key. Add Round Key adds a transformed key which is different for each round. The transformation of the key is done by the Key expansion routine which will be explained in the coming sections. After that the final round does shift rows, sub bytes and add the final round key whilst skipping the Mix columns process. Add round key involves xoring the state with the corresponding round key.

The 128 bit plain text input that comes in is transformed and represent by a 4 by 4 matrix shown by the following table.

| | | | |
|---|---|---|---|
| 0 | 4 | 8 | 12 |
| 1 | 5 | 9 | 13 |
| 2 | 6 | 10 | 14 |
| 3 | 7 | 11 | 15 |

**Table 2-1 Arrangement of bytes in a 4 by 4 state matrix.**

Each element in the table corresponds to a byte. The numbering shows how the bytes are arranged. Each column represents a 4 byte word which can be represented as a four element polynomial as discussed before.

The state data goes through a series of transformation before cipher text comes out. The process is reversed for decryption. The diagram below shows the encryption and decryption processes. What is written Byte Sub. is the byte substitution process or sub-bytes. The reverse of it is written as Inverse Byte Sub. in the decryption process. The number of iterations is dependent on the length of the key. If the key is 128 bits, 9 iterations are done with one last one without mixing columns. If the key is 192 bits, 12 iterations are done including the last one. If the key is 256 bits, 14 iterations are done.

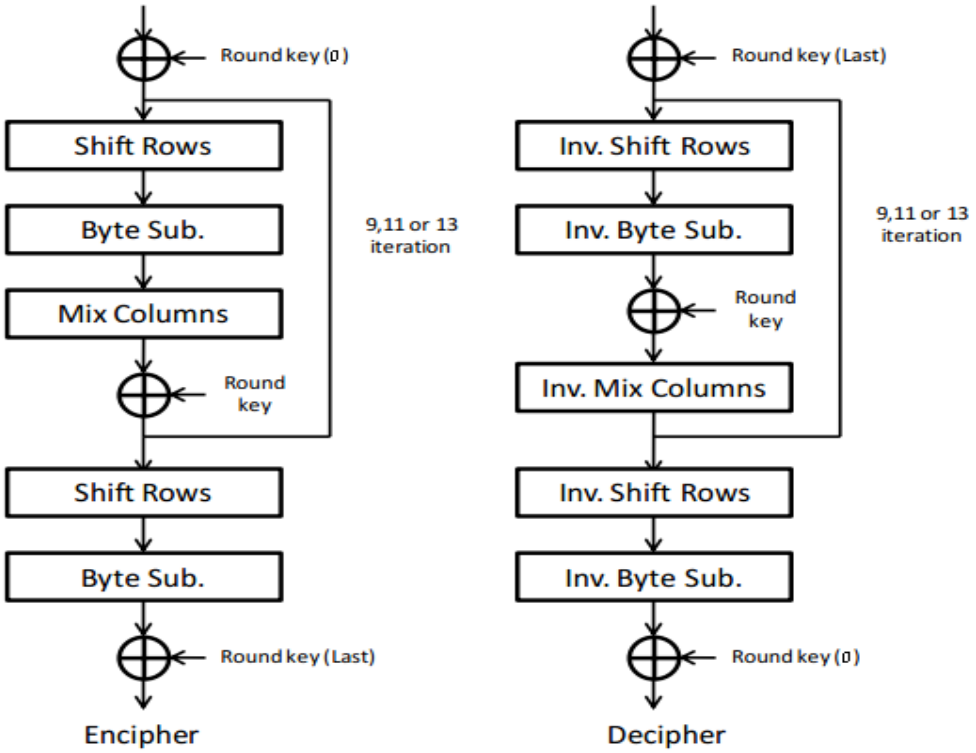


**Figure 2-5 AES encryption and decryption processes**

The following sub-sections will explain the shift rows, sub bytes, mix columns and the Key expansion routine.

### *2.4.1 Shift Rows / Inverse Shift Rows*

For shift rows, the first row is left as it is. The second row is shifted by one byte to the left. The third row is shifted by two bytes to the left. The fourth row is subsequently shifted by three bytes. After the shift rows operation, the state table changes as follows.

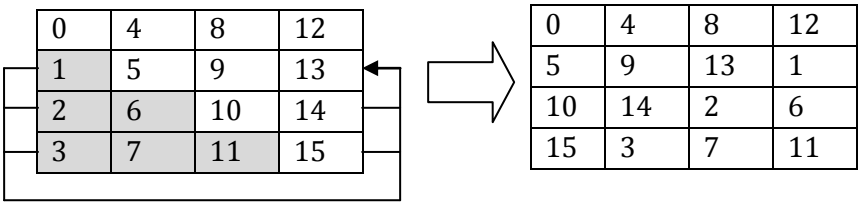


**Figure 2-6 Shift rows operation**

For inverse shift rows, the process is reversed as shown:-

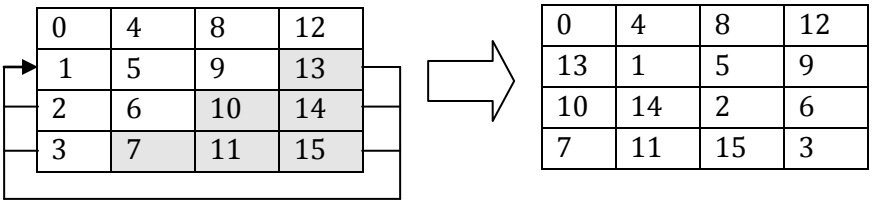


**Figure 2-7 Inverse rows operation**

We see that for inverse, the bytes on the extreme right come to the extreme left whilst its vice-versa for shift rows

### *2.4.2 Sub bytes / Inverse sub bytes*

For sub bytes, each byte in the state table is replaced by a corresponding byte from the S-box table. In programming languages, this S-box is held as an array. The two individual hexadecimal characters that make up a single byte in the state array are examined. The first one will direct to the required row in the S-box and the second one will direct to the column in the s-box. Where the row and column point to is the byte that will replace the previous one in the state array. This process is done for all the bytes. For inverse- sub bytes done for decryption, the process is reversed.

The S-box used is shown below:-

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 7C | 77 | 7B | F2 | 6B | 6F | C5 | 30 | 01 | 67 | 2B | FE | D7 | AB | 76 |
| **1** | CA | 82 | C9 | 7D | FA | 59 | 47 | F0 | AD | D4 | A2 | AF | 9C | A4 | 72 | C0 |
| **2** | B7 | FD | 93 | 26 | 36 | 3F | F7 | CC | 34 | A5 | E5 | F1 | 71 | D8 | 31 | 15 |
| **3** | 04 | C7 | 23 | C3 | 18 | 96 | 05 | 9A | 07 | 12 | 80 | E2 | EB | 27 | B2 | 75 |
| **4** | 09 | 83 | 2C | 1A | 1B | 6E | 5A | A0 | 52 | 3B | D6 | B3 | 29 | E3 | 2F | 84 |
| **5** | 53 | D1 | 00 | ED | 20 | FC | B1 | 5B | 6A | CB | BE | 39 | 4A | 4C | 58 | CF |
| **6** | D0 | EF | AA | FB | 43 | 4D | 33 | 85 | 45 | F9 | 02 | 7F | 50 | 3C | 9F | A8 |
| **7** | 51 | A3 | 40 | 8F | 92 | 9D | 38 | F5 | BC | B6 | DA | 21 | 10 | FF | F3 | D2 |
| **8** | CD | 0C | 13 | EC | 5F | 97 | 44 | 17 | C4 | A7 | 7E | 3D | 64 | 5D | 19 | 73 |
| **9** | 60 | 81 | 4F | DC | 22 | 2A | 90 | 88 | 46 | EE | B8 | 14 | DE | 5E | 0B | DB |
| **A** | E0 | 32 | 3A | 0A | 49 | 06 | 24 | 5C | C2 | D3 | AC | 62 | 91 | 95 | E4 | 79 |
| **B** | E7 | C8 | 37 | 6D | 8D | D5 | 4E | A9 | 6C | 56 | F4 | EA | 65 | 7A | AE | 08 |
| **C** | BA | 78 | 25 | 2E | 1C | A6 | B4 | C6 | E8 | DD | 74 | 1F | 4B | BD | 8B | 8A |
| **D** | 70 | 3E | B5 | 66 | 48 | 03 | F6 | 0E | 61 | 35 | 57 | B9 | 86 | C1 | 1D | 9E |
| **E** | E1 | F8 | 98 | 11 | 69 | D9 | 8E | 94 | 9B | 1E | 87 | E9 | CE | 55 | 28 | DF |
| **F** | 8C | A1 | 89 | 0D | BF | E6 | 42 | 68 | 41 | 99 | 2D | 0F | B0 | 54 | BB | 16 |

**Table 2-2 S-box used for encryption**

For example, {7A} in the state array will be replaced by {DA} after the sub-bytes function if doing the encryption process. This is so because the first 7 denotes the row and A will point to the column. Where they intersect is the byte that will replace the previous one.

For inverse sub-bytes, the process is reversed. The inverse sub bytes table is shown below. We can see that we will be able to get {7A} from {DA} as explained.

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 52 | 09 | 6A | D5 | 30 | 36 | A5 | 38 | BF | 40 | A3 | 9E | 81 | F3 | D7 | FB |
| **1** | 7C | E3 | 39 | 82 | 9B | 2F | FF | 87 | 34 | 8E | 43 | 44 | C4 | DE | E9 | CB |
| **2** | 54 | 7B | 94 | 32 | A6 | C2 | 23 | 3D | EE | 4C | 95 | 0B | 42 | FA | C3 | 4E |
| **3** | 08 | 2E | A1 | 66 | 28 | D9 | 24 | B2 | 76 | 5B | A2 | 49 | 6D | 8B | D1 | 25 |
| **4** | 72 | F8 | F6 | 64 | 86 | 68 | 98 | 16 | D4 | A4 | 5C | CC | 5D | 65 | B6 | 92 |
| **5** | 6C | 70 | 48 | 50 | FD | ED | B9 | DA | 5E | 15 | 46 | 57 | A7 | 8D | 9D | 84 |
| **6** | 90 | D8 | AB | 00 | 8C | BC | D3 | 0A | F7 | E4 | 58 | 05 | B8 | B3 | 45 | 06 |
| **7** | D0 | 2C | 1E | 8F | CA | 3F | 0F | 02 | C1 | AF | BD | 03 | 01 | 13 | 8A | 6B |
| **8** | 3A | 91 | 11 | 41 | 4F | 67 | DC | EA | 97 | F2 | CF | CE | F0 | B4 | E6 | 73 |
| **9** | 96 | AC | 74 | 22 | E7 | AD | 35 | 85 | E2 | F9 | 37 | E8 | 1C | 75 | DF | 6E |
| **A** | 47 | F1 | 1A | 71 | 1D | 29 | C5 | 89 | 6F | B7 | 62 | 0E | AA | 18 | BE | 1B |
| **B** | FC | 56 | 3E | 4B | C6 | D2 | 79 | 20 | 9A | DB | C0 | FE | 78 | CD | 5A | F4 |
| **C** | 1F | DD | A8 | 33 | 88 | 07 | C7 | 31 | B1 | 12 | 10 | 59 | 27 | 80 | EC | 5F |
| **D** | 60 | 51 | 7F | A9 | 19 | B5 | 4A | 0D | 2D | E5 | 7A | 9F | 93 | C9 | 9C | EF |
| **E** | A0 | E0 | 3B | 4D | AE | 2A | F5 | B0 | C8 | EB | BB | 3C | 83 | 53 | 99 | 61 |
| **F** | 17 | 2B | 04 | 7E | BA | 77 | D6 | 26 | E1 | 69 | 14 | 63 | 55 | 21 | 0C | 7D |

**Table 2-3 Inverse S-box**

Rather than storing the s-box as a table, the replacement value can also be calculated as shown in [3]. It is calculated by first finding the multiplicative inverse of the bytes and then applying an affine transformation on them. We can find the multiplicative inverse by using the Extended Euclidean algorithm [2]. We have shown in Equation 2 that the inverse can be found as:-

$a^{-1}(x)$ = b(x) mod m(x) where b(x) is found by the Extended Euclidean algorithm and $m(x) = x^8 + x^4 + x^3 + x + 1$.

The affine transformation is whereby a matrix which is constant is multiplied with the data followed by an addition with a vector which is also constant. The process is applied on a byte. The constants used for the affine transformation and its inverse are shown below:-

$$AF(a) = P1 * a + Q1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

The function AF represents the Affine transformation. P1 is the constant matrix and Q1 is the constant vector.

For the inverse affine transformation, the constant matrix is P2 and the constant vector is Q2 as shown below:-

$$AF(a)^{-1} = P2 * a + Q2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

### *2.4.3 Mix Columns / Inverse Mix Columns*

For Mix columns, each column in the state matrix is treated as a four term polynomial using the mathematics described in the previous section. This is 32-bits and the operation is on each column. Each column is multiplied by the polynomial $c(x) = \{03\} x^3 + \{01\} x^2 + \{01\} x + \{02\}$. After this we get the resultant four term polynomial by applying modulo $k(x) = x^4 + 1$ as described before. The resultant 32-bit word is found by multiplying the four term word of the state matrix in each column as described by [4]. The process is shown below. S represents state array, the first number represents the row number and C represents the column number. S' represents the resultant vector after applying mix columns.

$$\begin{bmatrix} S_{0C}' \\ S_{1C}' \\ S_{2C}' \\ S_{3C}' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} * \begin{bmatrix} S_{0C} \\ S_{1C} \\ S_{2C} \\ S_{3C} \end{bmatrix}$$

These are expanded as shown below:-

$S_{0C}' = \{02\} \bullet S_{0C} \text{ xor } \{03\} \bullet S_{1C} \text{ xor } S_{2C} \text{ xor } S_{3C}$
$S_{1C}' = \{02\} \bullet S_{1C} \text{ xor } \{03\} \bullet S_{2C} \text{ xor } S_{0C} \text{ xor } S_{3C}$
$S_{2C}' = \{02\} \bullet S_{2C} \text{ xor } \{03\} \bullet S_{3C} \text{ xor } S_{0C} \text{ xor } S_{1C}$
$S_{3C}' = \{02\} \bullet S_{3C} \text{ xor } \{03\} \bullet S_{0C} \text{ xor } S_{1C} \text{ xor } S_{2C}$

The inverse mix columns is obtained by multiplying each state column with the polynomial $c^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

The matrix is represented as follows:-

$$\begin{bmatrix} S_{0C}' \\ S_{1C}' \\ S_{2C}' \\ S_{3C}' \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} * \begin{bmatrix} S_{0C} \\ S_{1C} \\ S_{2C} \\ S_{3C} \end{bmatrix}$$

The resultant calculations are as follows:-

$S_{0C}' = \{0e\} \bullet S_{0C} \text{ xor } \{0b\} \bullet S_{1C} \text{ xor } \{0d\} \bullet S_{2C} \text{ xor } \{09\} \bullet S_{3C}$
$S_{1C}' = \{0e\} \bullet S_{1C} \text{ xor } \{0b\} \bullet S_{2C} \text{ xor } \{0d\} \bullet S_{3C} \text{ xor } \{09\} \bullet S_{0C}$

$S_{2C}' = \{0e\} \bullet S_{2C} \text{ xor } \{0b\} \bullet S_{3C} \text{ xor } \{0d\} \bullet S_{0C} \text{ xor } \{09\} \bullet S_{1C}$
$S_{3C}' = \{0e\} \bullet S_{3C} \text{ xor } \{0b\} \bullet S_{0C} \text{ xor } \{0d\} \bullet S_{1C} \text{ xor } \{09\} \bullet S_{2C}$

### *2.4.4 Key Expansion Unit*

The Key expansion unit is used to generate round keys used in the encryption process. They key can be 128 bits, 192 bits or 256 bits. In this project, 128 bits will be used. The first 16 bits of round key are formed by the encryption key. This is Round Key 0 which is first xored with the incoming state bits to form the state matrix that enters the first iteration. A total of 176 bytes of round keys is needed for a 128 bit key. For 192 bit key, we would need 224 bytes and 240 bytes are needed for 256 bit key.

For a 128 bit key, 10 group of round keys are generated. Each has 16 bytes. The first column in each group is generated by the process shown below:-

- The column which is before the current one [p-1] if we are processing the pth column is put through the sub bytes calculation to get its first transformation.
- After the sub bytes operation, the RotWord() function which was discussed before is applied where the bytes are rotated by one byte to the left; $[a_0\ a_1\ a_2\ a_3]$ becomes $[a_1\ a_2\ a_3\ a_0]$.
- The round constant corresponding to the round is xored.
- The result is xored to the corresponding column in the previous group

According to [1], the key expansion generates a total of Nb(Nr + 1) words. This means a denoting the word number is defined as 0 <= p < Nb(Nr + 1). Nb is the number of words in the state matrix and Nr is the number of rounds. To find the word w[p], we XOR the word before it w[p-1] with the one Nk positions earlier w[p -Nk]. Nk is the number of words in the key.

It is also shown that the key expansion of a 256-bit key is a little different. We do sub bytes on each previous word w[p-1] i.e. SubWord()before doing XOR with the one Nk positions earlier. This is done if Nk is 8 and p-4 is a multiple of Nk [1].

These round keys in hardware can be generated on the fly in parallel with other AES operations as shown in [3]. This saves much area and memory if we intend to use low area as in an low area ASIC. However what is usually done is to store the round keys in BRAM blocks as described in [5]. The round constant array is called Rcon[p]. The values are given by $[ x^{p-1},\{ 00\},\{00\},\{00\}]$, with $x^{p-1}$ as powers of x ( x is given by{ 02}) in the $GF(2^8)$ field, as shown in the Mathematics section; p in this case starts from one.

# 3. Methodology

## 3.1 Basic Description and outline

The goal of the project is to implement a FPGA-based parallelised encryption and decryption system which can be tested against its serial version for performance characteristics. FPGAs have proved to be ideal hardware platforms for high performance and parallelism. It is an off-the-shelf programmable logic device [27] which the customer rather than the manufacturer can program. This gives it much greater flexibility [27]. It was thus chosen to implement the encryption system on an FPGA to take advantage of the ability to parallelise systems.

The plan is as follows:-

- Full version of AES running on a PC as the most complete and powerful based of comparison – AES was chosen as the method of encryption as described in the Literature Review.
- A cut-down version of AES that will run fast and take less space, but sacrificing encryption power. This tested on PC. The AES encryption will be examined to see what can be cut to make the process run faster.
- FPGA version of cut-down version. Then comparing this to the serial version, seeing if it is correct. Then speed comparison tests between the full version, serial (gold) version and FPGA version.

The serial version will be implemented in C++ and the parallelised version in VHDL on an FPGA. C++ was chosen as it is much easier to operate with bytes and binary files. It is also the language used for second and third year computer science at the University of Cape Town and the researcher is quite familiar with it. The implemented method on FPGA is a simplified version of the standard algorithm for encryption which actually uses more computing resources than those available. This simplified version will also be tested against the standard version to see how secure the data is when encrypted.

First, research will be done to discover the different number of encryption algorithms that are available and to find a suitable method. AES (Advanced encryption Standard) has been found to be the most stable and standardised method for encryption as explained in the Literature Review. After this, throughout the first two weeks of August, this method will be further examined. As explained in the Literature Review section, The Nexys 2 board was selected to be the FPGA hardware to use. The
After this, design for the experiments to be done will be completed throughout the rest of August. This will be followed by implementation of the experiment in September.

The cut-down version is referred to as the Killian Encryption Standard (KES) for the purposes of this project.

The following definitions are defined:-

Gold version: KES on a PC implemented as a C++ program. C++ was chosen as explained above.
FPGA version: KES in VHDL running on Nexys 2. These choices were selected as outlined in the Literature review.

Comparisons

Correlation methods will be used for comparing the input down and the encrypted data to see how strong the encryption is for complete version and the Gold version. The correlation method is explained together with reference literature given in the Experiments section. Comparisons will be done between the cipher text and the plain text. Entropy of the text will also be compared. This term is explained in the Experiments section. Comparisons are necessary to evaluate the performance of an encryption method compared to others as shown in [25] and [26]. The encoding and decoding code for Gold version and full version will be implemented for comparisons. For the FPGA version, due to time constraints only the encoding part will be implemented but constructs will be added in VHDL for the easy implementation of the decoding part.

## 3.2 Timetable

The chart below shows the project outline and timeline. In the next sections, each component will be explained to see what will be done.

| **Project Gaunt Chart** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Milestone** | **Start Date** | **End Date** | | | | | | | | | |
| **Complete Project** | 2012/08/01 | 2012/10/16 | | | | | | | | | |
| **Designing** | | | | | | | | | | | |
| Literature Review | 2012/08/01 | 2012/08/14 | | | | | | | | | |
| Design phase | 2012/08/15 | 2012/08/31 | | | | | | | | | |
| **Implementation** | | | | | | | | | | | |
| Implementation of design | 2012/09/01 | 2012/09/25 | | | | | | | | | |
| **Testing and Reporting** | | | | | | | | | | | |
| Testing | 2012/09/26 | 2012/10/02 | | | | | | | | | |
| Write up | 2012/10/03 | 2012/10/16 | | | | | | | | | |

**Figure 3-1Gaunt Chart for Project**

Since the Implementation part involves a lot of debugging and code, it will take the longest as shown above. A log book will be used to record notes and references will be recorded and stored on file so that it can be easier when writing up.

## 3.3 Work to be done on Literature Review

First of all some literatures will be read to determine which encryption method to implement. After that literature will be read to find out which hardware platform to implement the encryption system. After that, there will also be considerations on which hardware description language to use. Extensive research will be done to see how the encryption method functions. Extensive research will be done on encryption systems implemented before to see what sort of design can be implemented. They will also be research on the type of applications that the encryption system is going to be used for.

## 3.4 Work to be done on Design

From the literature review, suitable design option will be selected considering the hardware platform available and the type of application that the FPGA system is going to be used for. The book named in [28] will be used to help design in VHDL. Other reference information includes programming notes in VHDL and C++ from third year Electrical and Computer Engineering. The other book is [29] which shows how the cores are built and has a model AES encryption system implemented in Verilog.

The first design process involves designing the flow of the gold version of the encryption process to make sure it models the one to be implemented on the FPGA even though it's not parallelised. The full version's flow is changed and some rounds cut to ensure it runs faster and takes less space.

The design process will then go to the FPGA system. This is whereby various logic elements are selected that can do the encryption process. Various blocks are designed from the top block which is the encryption core to the parts that make it. As the encryption system will be mainly modelled for use in low area applications like RFID tags as mentioned in Chapter 2, the datapath will be limited to 8 bits for now. Various designs of 32-bit and 8 bit implementations were examined as shown in the Literature review to see how the encryption system is setup. Compared to the 32-bit designs explained in Chapter 2 [3] shows an 8 bit designs which suits the low area application which this project aims. Most components for the design to be implemented will thus refer to how this structure in [3] is built. This will then be followed to implement the design done on PC and parallelise it so that some tasks can execute concurrently.

## 3.5 Setup of Working Environment

The working environment needs to be correctly setup so as to accomplish the building up of the encryption system. The following items are to be setup:-

### *3.5.1 Personal Computer*

A personal computer running Windows XP or higher needs to be available that will be able to run the software that in needed. It needs to have at least 10 GB of space free on hard drive. The Xilinx ISE 12.1 consumes about 8 GB of hard drive. The other software Diligent does not consume much memory. It is also advisable to leave at least 2GB free to allow other simulators like ModelSim to be installed.

### *3.5.2 Xilinx ISE 12.1 or higher*

This is the main simulation environment used to write the VHDL code and modules. Xilinx ISim is used to simulate the test benches which are written to test the encryption system. Xilinx ISE was chosen because the Spartan 3 which is available for this project is a Xilinx product therefore is modelled in this edition of software. One can also install ModelSim which can be incorporated into Xilinx. However, this is advised when a professional version of ModelSim is available which can be linked to Xilinx. The freely available Student version of ModelSim was tried and did not integrate well into Xilinx but managed to operate very well separately. One would need a fully functional license to run Xilinx ISE which is linked to the MAC address of the host personal computer.

### *3.5.3 Diligent Adept 2.1.1 or higher*

This is the software required to connect to the FPGA. This is explained in [30] which shows a user's manual. After compiling the design in Xilinx, one is able to program and produce a bit file which contains instructions how the FPGA will be programmed. When the Nexys 2 board is connected to the host PC, the program is able to read it and one can specify the bit file produced by the Xilinx software to be able to implement the encryption system on FPGA.

### *3.5.4 Spartan 3 FPGA with USB*

This is the FPGA hardware that will be used to connect to the host PC. This was chosen as explained in Chapter 2. The datasheet is explained in [31] which show how its ports can be connected. One can connect a keyboard to the board to simulate the inputting of data onto the FPGA. Alternatively, one can enter the data via the USB port. The UCF file which links the ports of the encryption system defined in the Xilinx ISE to the ports on the FPGA will be examined and correctly programmed so that the FPGA can function correctly in a manner that is user friendly.

As the document in [31] shows, the Spartan 3 consists of five fundamental blocks which are listed below:-

- Configurable Logic blocks (CLBs) – These contain Look up tables. Logic functions are found here and also elements to store data like flip flops. Each CLB has four slices which will be presented in this project to show the area used by the design compared with others listed in Chapter 2.

- Input or Output Blocks (IOB) – flow of data from IO pins to device's logic is controlled here

- Block RAM – Data storage is provided using 18-Kbit dual-port blocks.

- Multiplier blocks – To perform multiplication of two 18-bit numbers of binary.

- Digital Clock Manager blocks (DCM) - solutions for manipulating clocks signals are provided.

The structure is shown by the following diagram:-

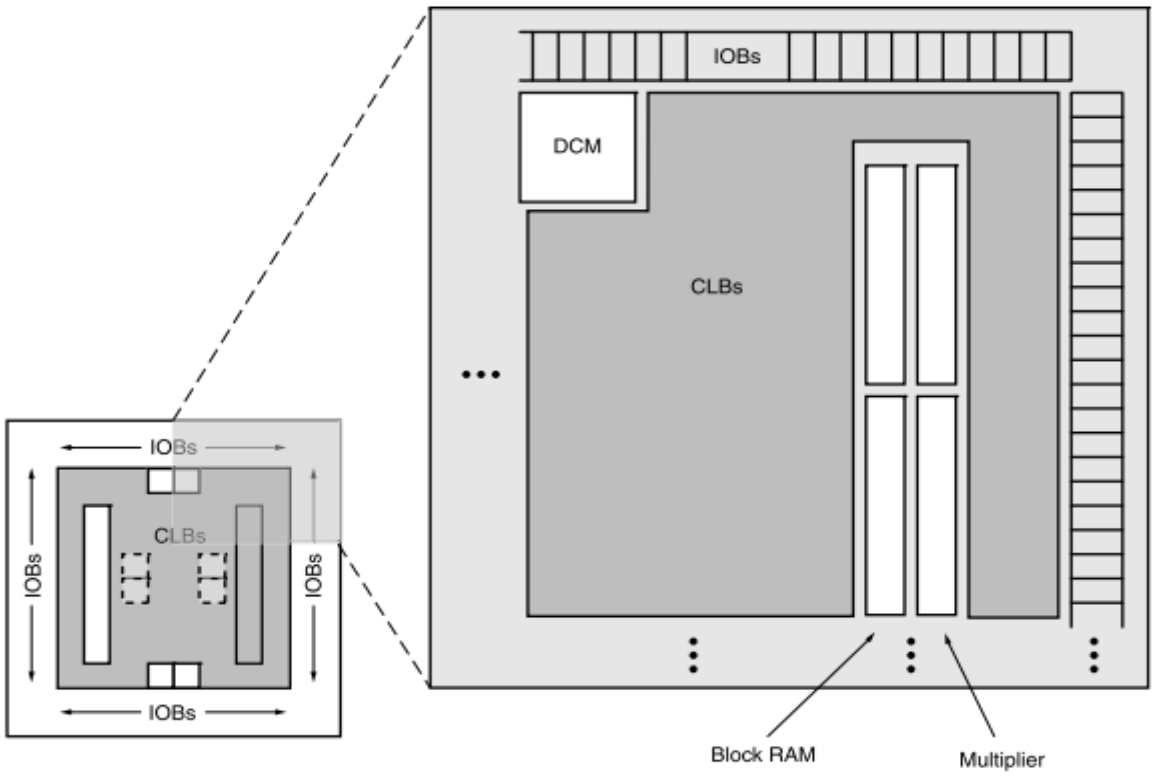


**Figure 3-2 Structure of a Spartan 3E FPGA**

The design will be tested on a Spartan-3E 500. However it is designed to fit the smallest of the family which has 960 slices as shown below. This is to increase its portability.

| Device | System Gates | Equivalent Logic Cells | CLB Array (One CLB = Four Slices) | | | | Distributed RAM bits(1) | Block RAM bits(1) | Dedicated Multipliers | DCMs | Maximum User I/O | Maximum Differential I/O Pairs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Rows | Columns | Total CLBs | Total Slices | | | | | | |
| XC3S100E | 100K | 2,160 | 22 | 16 | 240 | 960 | 15K | 72K | 4 | 2 | 108 | 40 |
| XC3S250E | 250K | 5,508 | 34 | 26 | 612 | 2,448 | 38K | 216K | 12 | 4 | 172 | 68 |
| XC3S500E | 500K | 10,476 | 46 | 34 | 1,164 | 4,656 | 73K | 360K | 20 | 4 | 232 | 92 |
| XC3S1200E | 1200K | 19,512 | 60 | 46 | 2,168 | 8,672 | 136K | 504K | 28 | 8 | 304 | 124 |
| XC3S1600E | 1600K | 33,192 | 76 | 58 | 3,688 | 14,752 | 231K | 648K | 36 | 8 | 376 | 156 |

**Figure 3-3 Spartan 3E FPGAs structure**

### *3.5.5 Matlab 2011 or higher*

This will be for examining the files produced from the encryption output. Scripts will be written that can process files to find correlation values and entropy which are explained in the coming Experiment setup section. This will also be used to produce histograms of how the values of bytes in the files vary. One can write correlation algorithms in C++. It was discovered that Matlab offer ready to use functions of this nature and would thus be ideal to use fitting the timeline of this project.

## 3.6 Building up of Project

The following steps outline what will be done to build the system.

1. Open source programs will be researched and used to implement the full version of the Advanced Encryption System. Since these have already been tested, it will save much time required to program and test. The program will be run to make sure that the functions of Encryption and Decryption are working.

2. The researcher will then follow the design of the gold solution provided to write code for it in C++. This will be run on PC and tests done to make sure that it produces expected results as verified independently. The decryption system for the gold solution will also be built and ensure that it is able to decrypt and recover the encrypted text using the same key. Timing variables will be added to measure how long the program takes to run.

3. Following the design provided in 2, the FPGA solution will be built. References will be made to similar projects done to see how the blocks can be implemented in VHDL and FPGA. Extensive reference will be made to the Open Core site to see if there are useful modules that can be used to build the design of the FPGA solution.

4. Software to measure the correlation of input versus Output will be installed and used to compare input versus cipher text for gold solution and also for full version. This is for comparisons to be made.

## 3.7 Tests to be done on the developed system

After the software is built, tests will be done on the system following the basic outline specified before. The tests to be done are listed before.

### *3.7.1 Experiment 1*

The first experiment involves testing that the AES encryption full version works. A number of plain and cipher text combinations will be tested and compared with the examples provided in [1]. If they all match it means the algorithm is implemented well. It will then be modified to take in plain text from files.

### *3.7.2 Experiment 2*

The second experiment will involve doing similar work done in 3.7.1 above on the Gold version. The document in [1] provides examples of what the output should be after every round. Since the encryption method is cut down to 7 rounds, The output after seven rounds shown in [1] will be checked and verified. The same will be done on the FPGA version. Simulation will be done in Xilinx ISE and output in hexadecimal examined to ensure it is the same with the one on the Gold version.

### *3.7.3 Experiment 3*

After modifying the full version and gold version to take in files, sample files are taken for testing the encryption. These are:-

- Document1.pdf (433 KB) – A pdf file which will be encrypted. The full version encryption will produce Document1.enc as output. The decryption part will take in Document1.enc and try to recover the Document as Doc1.pdf. Finally Doc1.pdf will be opened to see if it can reveal original contents. The same will be done for the Gold version. It will produce Document11.enc as encryption output and decryption will produce Doc11.pdf. The execution times for the full version and gold version will be recorded.

- Music1.mp3 (12.8 MB) – This is an MP3 files that contains voices. The full version encryption will produce Music1.enc as output. The decryption part will take in Music1.enc and try to recover the file as Mus1.pdf. Finally Mus1.pdf will be opened to see if it can play the original file. The same will be done for the Gold version. It will produce Music11.enc as encryption output and decryption will produce Mus11.pdf. The execution times for the full version and gold version will be recorded.

- Video3.mp4 (20 MB) – This is an MP4 music video. The full version encryption will produce Video1.enc as output. The decryption part will take in Video1.enc and try to recover the file as Vid1.mp4. Finally Vid1.mp4 will be opened to see if it can play the original file. The same will be done for the Gold version. It will produce Video11.enc as encryption output and decryption will produce Vid11.pdf. The execution times for the full version and gold version will be recorded.

- Picture1.jpg (130 KB) – This is a JPG image. The full version encryption will produce Picture1a.jpg as output. The decryption part will take in Picture1a.jpg and try to recover the file as Pic1.jpg. Finally Pic1.jpg will be opened to see if it can show the original picture. The same will be done for the Gold version. It will produce Picture1a.jpg as encryption output and decryption will produce Pic11.jpg. The execution times for the full version and gold version will be recorded.

- Sample1.txt (2 KB) – This is a text file. The full version will produce sample1.enc. The decryption part will take this and produce samp1.txt. Finally samp1.txt will be opened to see if can recover original contents. The same will be done for the gold version. It will produce sample11.enc as encryption output and decryption will produce samp11.txt, The execution times for the full version and gold version will be recorded.

- Sample2.txt (32 bytes) – This is a text file. The full version will produce sample2.enc. The decryption part will take this and produce samp2.txt. Finally samp2.txt will be opened to see if can recover original contents. The same will be done for the gold version. It will produce sample22.enc as encryption output and decryption will produce samp22.txt. The execution times for the full version and gold version will be recorded.

### *3.7.4 Experiment 4*

Correlation coefficient is a factor used to compare encryption methods [25]. It measures how two vectors are related to one another. A correlation value of 1 shows that they are very similar to one another. A correlation value of 0 shows they are very dissimilar. It is favourable to have cipher text that has a low correlation to the plain text and this will strengthen its security aspects [25].

The cross correlation coefficient is defined by [26] as:-

$$r = \frac{n\sum(xy) - \sum x \sum y}{\sqrt{\left[n\sum(x^2) - (\sum x)^2\right]\left[n\sum(y^2) - (\sum y)^2\right]}}$$

r is the correlation value. The vectors to be compared are x and y. n is the number of data pairs. The sum at the top represents sum of data whilst the one at the bottom represents sum squared.

A matlab script will be written to compare files produced in 3.7.3 above. The Matlab script will open them as binary files and put the contents in a vector. The vectors will be sized up to make sure they are of the same length for comparisons to take place. There is a Matlab function called corrcoef(x,y) which finds the correlation coefficient between vectors x and y. The plain and cipher text for all the files recorded in 3.7.3 above will be compared both for the full version and the gold version. Correlation will be recorded so that the performance of the KES encryption method can be compared to the full version encryption method.

The matlab script will also output the entropy. A function will be written that takes in a vector and calculates the entropy. The values will be tabulated. This is a measure of the amount of information which a file compression system must code [26]. Files with low entropy have similar byte intensity levels throughout. A low entropy will compress to a small size. The function is given by [26] as :-

$$H_e = -\sum_{k=0}^{G-1} P(k)\log_2\big(P(k)\big)$$

G is the intensity level of a byte which can go from 0to 255. P(k) is the probability that symbol k will occur. Entropy measures the randomness in a file. The higher it is the greater the randomness. Higher entropy represents good variability and we seek cipher text to have higher entropy than plain text.

For this experiment, histograms will be produced that shows the number of bytes and a certain byte intensity level measured from 0 to 255.

### *3.7.5 Experiment 5*

32 bytes of input will be put to the FPGA simulation. The amount of time it takes to get all results will be recorded. We also record the time it takes for 16 bytes to come out since it processes a block of 16 bytes at a time. The result will be compared to the one we got it Experiment 3 to calculate speed up. More bytes will be loaded to the system to see how speed up increases as more bytes are loaded in. This will reveal the performance of the parallelised version operating on the FPGA hardware.

## 3.8 Methods of analysing data

The following methods will be used to analyse the data that is produced.

### *3.8.1 Experiment 1*

- Output compared with the one shown in [1] given same input and key.
- Various combinations will be examined.

### *3.8.2 Experiment 2*

- Output compared with the one shown in [1] given same input and key.
- Various combinations will be examined.

### *3.8.3 Experiment 3*

- Running times of the code is noted for all the input files

### *3.8.4 Experiment 4*

- Correlation values tabulated and graphs plotted in MS Excel
- Entropy values tabulated and graphs plotted in MS Excel.

### *3.8.5 Experiment 5*

- Performance time for FPGA is noted
- Speedup for parallelism is calculated

## 3.9 Documenting the work output

At each stage of the Literature review, document names and where the document is fetched will be noted so as to correctly reference them when compiling the final report. Diagrams will be done to show the design in full and implemented blocks will be summarised so that anyone who can operate Digital systems will be able to replicate the system and understand how it functions.

# 4. Design

This chapter documents the structure of the algorithms coded in C++ on PC and the VHDL modules implemented written in Xilinx ISE and simulated there.

## 4.1 Full Version Encryption algorithm

The encryption process is documented as researched in Chapter 2. This sequence of steps is implemented in C++ so as to model the full version algorithm. Most of the code was adapted from open sources and modified to add functionality to read files and write output. The structure of the code is shown in the flow chart on the following page:-

Full version Encryption Flowchart

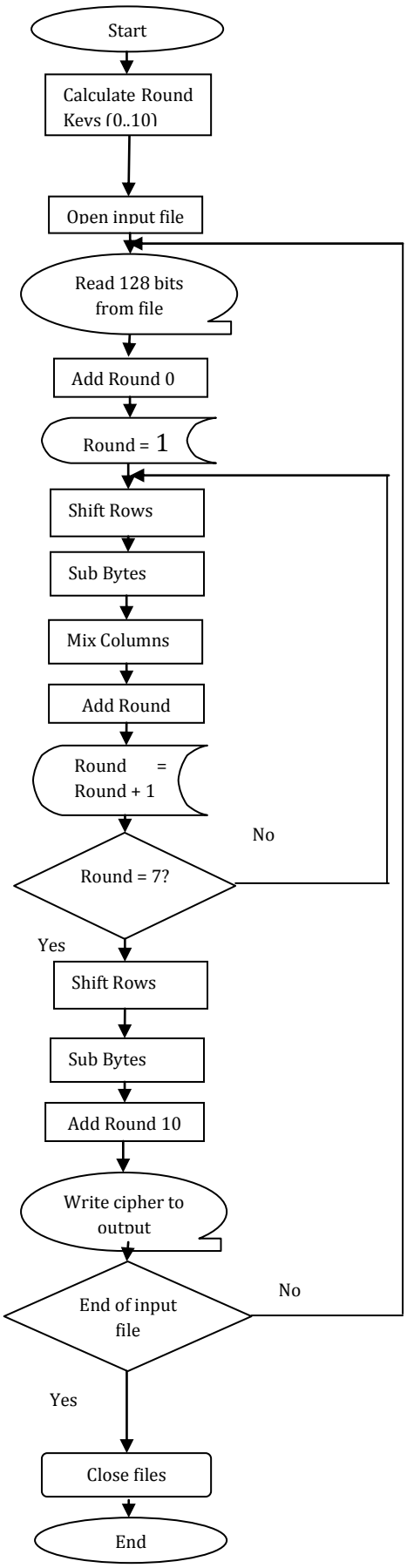


**Figure 4-1 Full Version Flowchart**

The program starts by first calculating the round keys for all the rounds. This is done by calling the Key expansion algorithm. The Cipher algorithm is then called which loops around as shown above. There are subroutines for Mixing columns, Sub bytes, Add Round, Shift rows and Add round. They follow the sequence of steps described in Chapter 2.

## 4.2 Cut Down Version encryption algorithm

The cut down version of the encryption algorithm is supposed to be the one that takes shorter time to run. It also cuts down some rounds from the one shown above and does seven rounds instead of ten. It is also takes one byte at a time instead of 16 bytes as the 128-bit data path. The modified flowchart for the cut-down version is shown in the next page:-

Cut Down Version Flowchart

Sequence buffer = [0;5;10;15;4;9;14;3;8;13;2;7;12;1;6;11]

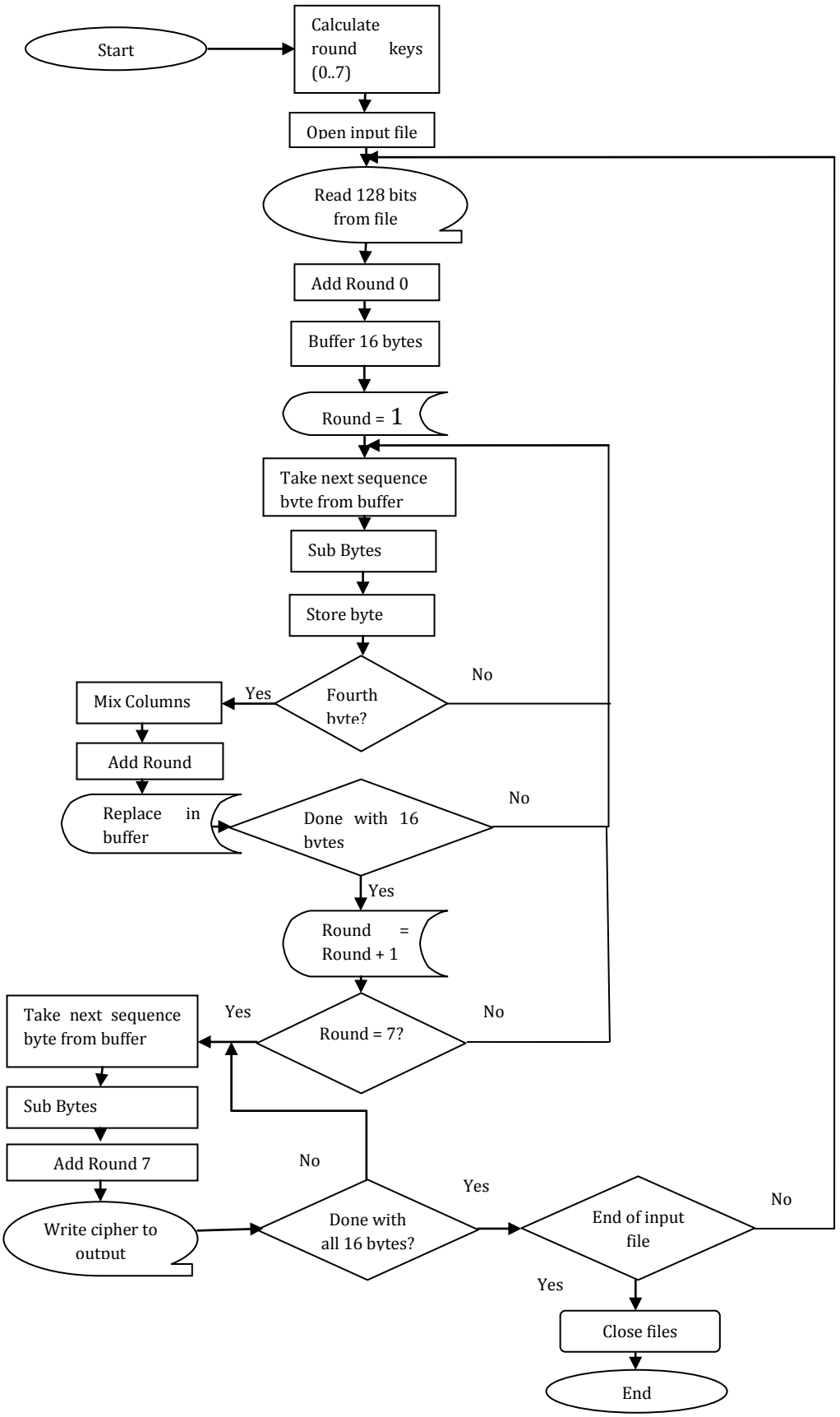


**Figure 4-2 Cut-down version Flowchart**

The sequence buffer stores the order in which bytes are processed from the state buffer. This is in the order in which bytes are taken after doing Shift rows. There is no need for a separate Shift rows function because the bytes are taken in the order specified in Chapter 2 in Figure 2.7

## 4.3 Inverse Operations

The inverse for the Full version is implemented almost as shown in the first flow chart. The mix columns is done after the Add round and the Shift rows is replaced by Inverse Shift rows. Sub bytes is replaced by inverse sub bytes. The Add round runs from 10 down to 0 instead of the other way for Forward operation. The matrix used for Mix columns also changes to the one specified in Chapter 2 under the Inverse operations. Round is first initialised to 10, then instead of saying Round = Round + 1 we say Round = Round -1.

For the gold version, the same changes are made. Round is first initialised to 7 and runs down to the initial round. The datapath is also change to take one byte at a time. This is to model the FPGA version which processes one byte at a time.

## 4.4 Cut Down Version FPGA Implementation

### *4.4.1 Choice of Design*

A number of designs were examined to see how the cores for encryption are built. Open cores available on the web provide a number of designs available which can be downloaded and built on. The choice of the design to use was based on the data path to use, portability of the system, the ability to implement parallelised architecture, the limit of the chosen FPGA (Spartan 3E) and the application for the system which is low area and low power. Most of the designs available on open-cores implement 128 bit data path designs which are not possible to implement on the Spartan 3. 32 bit architectures were also greatly considered. The design shown in [5] implements a high throughput 32-bit design. It was however decided to change all the datapath to 8 bit so that the system can be as portable as possible even though it takes a lot of clock cycles. The design in [3] proved to be a suitable reference for building an eight bit design. This is the reference which was largely as explained in the coming section.

A design similar to the one shown in [32] was also considered. The designer used an open core designed by Jerzy Gbur available on the website to create a multi-core system. This is quite ideal since it allows extensive parallelisation by implementing some cores for encryption and some for encryption. It was however found to be quite complex and possible for a project longer than three months. This was considered after having email discussions with the designer who struggled implementing it on the Xilinx board for his project but gladly made his cores available for download on Google. It was then decided that the 8 bit design provided in [3] could be initially implemented and then multi-core in future like what is done in [3] to increase the parallelisation. This could be of much help when working with bigger files or large data with other cores working on different sets of data at the same time.

The designer also had email discussions with Thomas Ruschival who implemented a 32 architecture. The portability of such a system was found to be less than the design provided in [3] as it utilises the Avalon slave interface which is suited for Altera based platforms as explained in [33].

From the facts shown above, the 8 bit architecture which references to the chosen architecture like that in [3] was implemented in VHDL. Other components were redesigned to fit the needs of the KES architecture as explained in the following section.

### *4.4.2 8 Bit KES Architecture*

The FPGA version implements the cut down version system shown in the previous section. Instead of first doing the Calculate Round Keys function, it is done in parallel with other encryption processes. This is expected to take advantage of the parallelising ability of FPGAs and possibly obtain results faster than the sequential version. According to the design, the round key calculation is ahead of the KES encryption process by 4 clock cycles, so by the time the Mix columns process finishes, the system fetches the round key calculation delayed by 4 cycles to add to it and it continues with KES encryption process. A design which is quite similar was used in [3].

Dedicated hardware like FPGAs will provide performance which is higher and consume less power. In Chapter 2, various FPGA designs were discussed which include loop unrolling and pipelined designs. These consume higher power and area and the design presented here is for low power and area suitable for the typical RFID (Radio Frequency Identification) application discussed earlier in Chapter 2. This design presents an iterative design which consumes less resources than other designs. The datapath used is 8 bits which is used for low area application even though it takes more clock cycles to finish than the 32 or 128 bit data-paths. The choice was also limited by the fact that the chosen FPGA cannot take 128-bits at a time as shown in [31]. It is possible to implement a 32-bit design on this FPGA but the 8-bit system was chosen to increase portability.

The overall architecture of the KES encryption system is shown in the diagram below:-

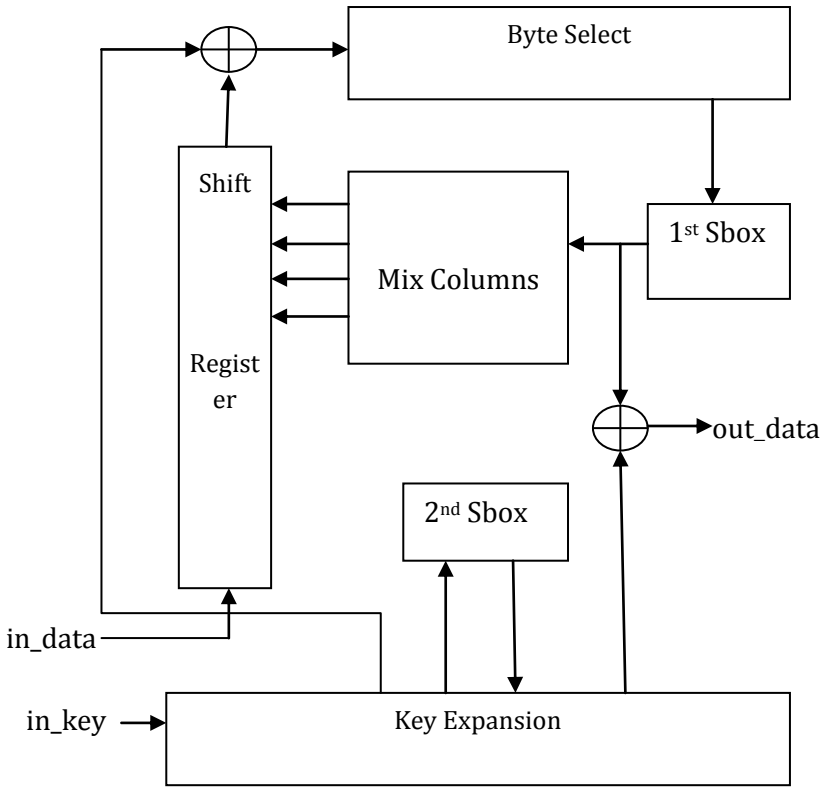


**Figure 4-3 Overall architecture of the KES encryption system**

The design was chosen following the design of the cut-down serial version. It operates on one byte at a time.

The lines represented above are 8-bit lines. The main components are Shift register, Byte select unit which buffers bytes and then selects them according to the Shift row sequence explained in the cut-down version flowchart. The bytes then go to the Sbox unit where the sub-bytes operation is done. The bytes then go to the Mix columns operations. After four bytes are received as shown in the Cut-down version flowchart, the output of Mix columns goes to the Shift register. After that, Add round is performed by xoring the Key from the Key expansion unit which is delayed by four cycles. The key expansion as explained also operates in parallel taking in keys and shifting them into the unit. On the last round, the bytes unwind following the Shift row sequence from the Shift register to the Byte select. Sub-bytes is performed and then the last key (round Key 7) is xored before data comes out as output.

The following components are explained in the following sections before the top block is presented. First is the Byte select unit followed by the Sbox, followed by the Mix Columns operation and then the Key Expansion. Understanding of how these components operate before putting it all together to present the top block.

### *4.4.3 Byte Select Unit*

This puts together the Shift rows and the storage of bytes. Shift rows is done by reordering the bytes as they are shifted in. It has space to store 12 bytes. Remaining four bytes are stored in the other registers of the encryption core. The byte select unit was adapted from the design presented in [9]. This represents register-based byte permutations. The one in [9] was chosen because it fits the 8-bit

architecture and allows shift rows function to be performed whilst data is being shifted in and avoid delays and data blocking. We reorder data by having the bytes circulate backwards from the last register using multiplexers. A multiplexor selects between input lines according to the m signal it gets as shown below. Certain bytes are read earlier on. No deadlocks are there. The structure from the design shown in [9] is shown in the diagram below:-

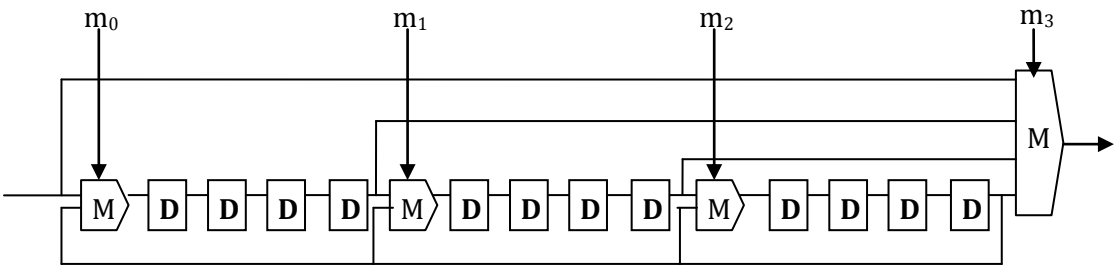


**Figure 4-4 Byte Select structure**

When the mux signal m is zero, the topmost signal is taken. When it increases, the signal are selected in that order going down. D represents a D flip flop and M represents a multiplexer. The D flip flop stores 8 bits of data as explained in [9]. The table below shows how the mux signals change as the clock cycles progress when doing encryption. This unit has a latency of 12 cycles. This means we need at least 12 registers.

| t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $m_0$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $m_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $m_2$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| $m_3$ | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 1 | 3 | 2 | 3 | 2 | 3 | 3 | 3 | 3 |

**Table 4-1 Multiplexer signals for byte select unit as time progresses**

The sequence of bytes taken is [0;5;10;15;4;9;14;3;8;13;2;7;12;1;6;11] following the sequence showing in the cut-down flowchart of the sequential version.

There is a constant array in VHDL to be put to set the sequence of operations to recover the bytes after Shift rows. The array is set to [3;2;1;0;3;2;1;1;3;2;3;2;3;3;3;3]. When it is 0, m0 to m3 are set to [1;0;0;0]. When it is 1, they are set to [0;1;0;1]. When it is 2, they are set to [0;0;1;2]. When it is 3, they are set to [0;0;0;3]. From table 4.1 above, we see that the first 11 cycles it is when the data is loaded into the unit. The byte select starts to get the data at 12 from above when it reflect to the option 3 which is the first option from the constant array shown above. The rest correspond to this pattern.

Below the component that will be implemented in VHDL is shown for this unit:-


**Figure 4-5 Byte Select unit component**

It shows that there is a signal for the loading of the data which happens when data is entering in. There is also a logic signal for shifting of data which also happens when data is being selected out. The sequence input is a 4 bit value which selects sequence. The decrypt_in value is set to one if the byte select is to select the sequence for the Inverse operation instead of the forward encryption process. This function was not included in this project due to time but can easily be added in, The port in_data takes in the incoming byte from the Shift register and the port out_data takes out the output data in order of the sequence shown in the cut-down version flowchart and sends it to the 1st Sbox for the sub-bytes operation. When need a clk (clock) input because the component operates one clock cycle after the other so these sequence steps have to occur in each clock cycle triggered by the rising edge of the clock as shown in the table above.

### *4.4.4 S-Box*

The 8-bit data path design shown in [3] from which this design derives its major components presents an Sbox implementation where the operation is broken into 2 computations as described in Chapter 2 under Sub-bytes operation. The calculation of the inverse is done in smaller subfields other than the GF(28). In this design, the Sbox is implemented as a look up table instead. The sort of implementation is presented in [11] which uses Block RAM. The in_data has the address component to fetch the required sub byte value. This can be easily implemented in FPGA. The structure of the S-box entity to be implemented in VHDL is shown in the diagram below:-

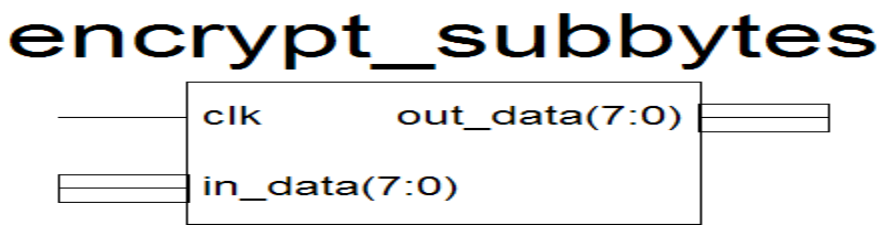


**Figure 4-6 Sub bytes entity**

It uses a clock input because it is activated on the clock edge. It takes an 8-bit input which represents the incoming byte from the byte select unit if it is the first Sbox as shown in Figure 4.3. If it is the second S-box it takes it from the Key expansion. It then output the byte to the port out_data which will

be connected to the Mix Columns box for the first S-box and connected back to the Key Expansion for the second S-box. In this design, we use to S-box because the encryption process and the key expansion process happen in parallel concurrently. It is therefore much more appropriate to separate the S-box implementations in the design.

### *4.4.5 Key Expansion Section*

The key expansion process can be done by calculating all round keys and storing them. This demands a lot of space. This is presented in [5]. Since this project seeks to implement a low area implementation, another option was sought. The design for Key expansion unit selected follows closely the one specified in [3]. This implements a key expansion component which calculates the round keys on the fly in parallel with the other encryption operation. The key expansion component implemented here has a dedicated S-box which saves the number of cycles in total compared to the one when an S-box is shared by these operations which operate in parallel. It also reduces the number of control logic operations and makes the design easier to implement yet effective.

The operation of the Key expansion is explained in Chapter 2. The first 4 byte word of each round is formed by first doing a cyclic shift on the previous word(Rot word), doing sub bytes, and xoring with Round constant calculation before finally xoring with the corresponding word of the previous round. The other words in the round are calculated by xoring the previous word with the corresponding word of the previous round.

The component to be implemented in VHDL is shown in the figure below.

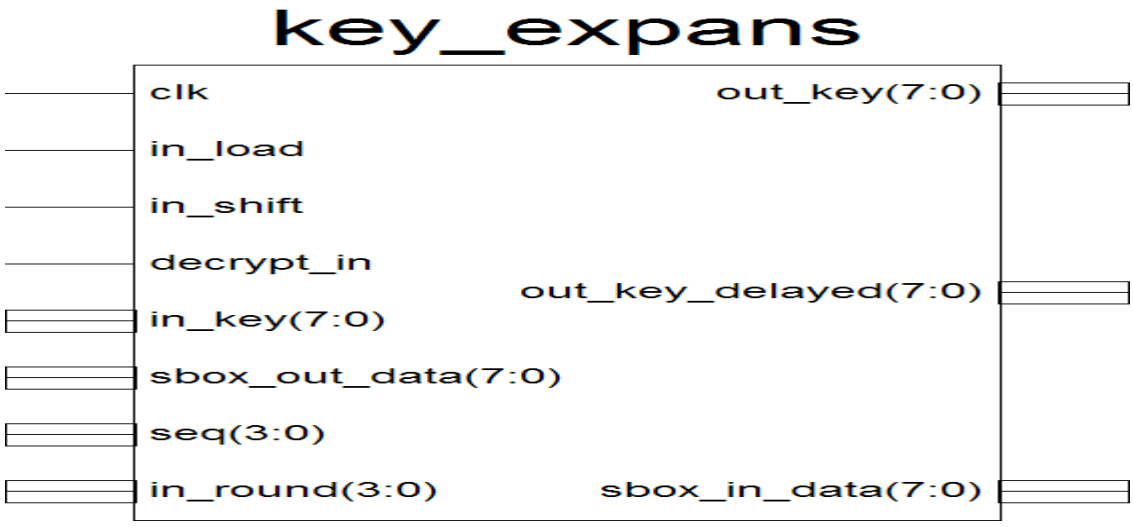


**Figure 4-7 Key Expansion component entity block**

The design shown in [3] was again referred to see how the Key expansion routine could be implemented. The port out_key has the key for the last round xored from output of sub-bytes before output data comes out as shown in Figure 4.3. The out_key_delayed port has the key to be xored with the output from the Mix columns which is put to the Shift register before going back to the Byte select unit for the other round. The Mix columns operation takes 4 clock cycles so the key has to be delayed by 4 clock cycles. The diagram for the Key expansion unit is shown in the figure below:-

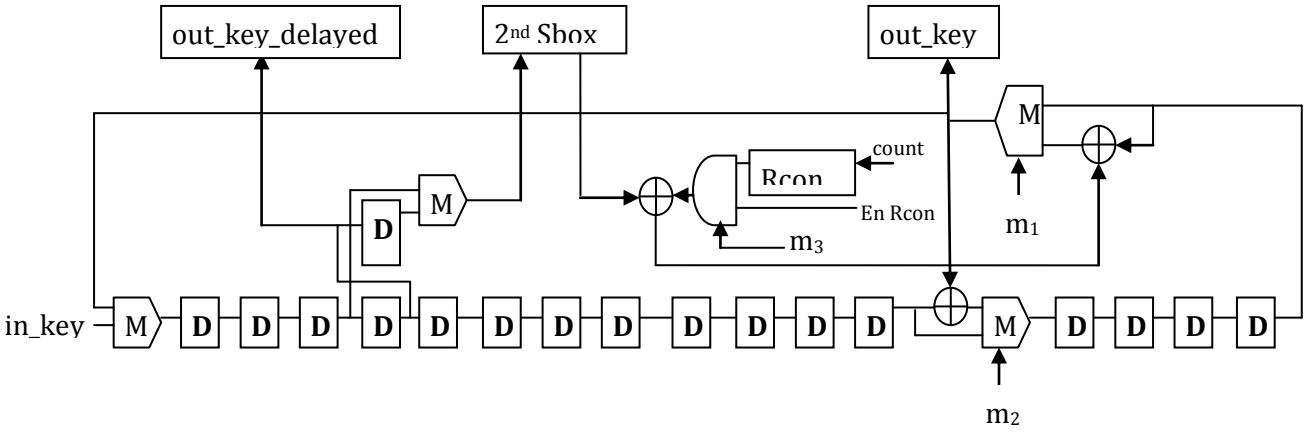


**Figure 4-8 Key expansion component**

Its operation is explained in the table below. Its registers represented by D-flip flops are numbered $K_0$ to $K_{15}$. $K_0$ is the one at the extreme right and $K_{15}$ is the one at the extreme left. When the key comes in, it is loaded in and the in_load is set to one to accept the input. The bytes get shifted in. The in_round port has the round number and the in_shift is set when bytes are being shifted as the round calculation proceeds. The En Rcon is set if the RCon calculator is to be operated for the first word in every round. Otherwise it is set to zero and the value xored to the Sbox output is zero for the other bytes of the first word as explained above.

| Time | $K_{15}$ | $K_{14}$ | $K_{13}$ | $K_{12}$ | $K_{11}$ ... $K_4$ | $K_3$ | $K_2$ | $K_1$ | $K_0$ | Work to be done |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $x_{15}$ | $x_{14}$ | $x_{13}$ | $x_{12}$ | $x_{11}$...$x_4$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ | $y_0 = x_0$ xor Sbox($x_{13}$) xor Rcon, $y_4 = x_4$ xor $y_0$ |
| 1 | $y_0$ | $x_{15}$ | $x_{14}$ | $x_{13}$ | $x_{12}$...$x_5$ | $y_4$ | $x_3$ | $x_2$ | $x_1$ | $y_1 = x_1$ xor Sbox($x_{14}$), $y_5 = x_5$ xor $y_1$ |
| 2 | $y_1$ | $y_0$ | $x_{15}$ | $x_{14}$ | $x_{13}$...$x_6$ | $y_5$ | $y_4$ | $x_3$ | $x_2$ | $y_2 = x_2$ xor Sbox($x_{15}$), $y_6 = x_6$ xor $y_2$ |
| 3 | $y_2$ | $y_1$ | $y_0$ | $x_{15}$ | $x_{14}$...$x_7$ | $y_6$ | $y_5$ | $y_4$ | $x_3$ | $y_3 = x_3$ xor Sbox($x_{12}$), $y_7 = x_7$ xor $y_3$ |
| 4 | $y_3$ | $y_2$ | $y_1$ | $y_0$ | $x_{15}$...$x_8$ | $y_7$ | $y_6$ | $y_5$ | $y_4$ | $y_8 = x_8$ xor $y_4$ |
| 5 | $y_4$ | $y_3$ | $y_2$ | $y_1$ | $y_0$...$x_9$ | $y_8$ | $y_7$ | $y_6$ | $y_5$ | $y_9 = x_9$ xor $y_5$ |
| . . . | . . . | . . . | . . . | . . . | . . . | . . . | . . . | . . . | . . . | . . . |
| 14 | $y_{13}$ | $y_{12}$ | $y_{11}$ | $y_{10}$ | $y_9$...$y_2$ | $y_1$ | $y_0$ | $y_{15}$ | $y_{14}$ | Nothing to be done |
| 15 | $y_{14}$ | $y_{13}$ | $y_{12}$ | $y_{11}$ | $y_{10}$...$y_3$ | $y_2$ | $y_1$ | $y_0$ | $y_{15}$ | Nothing to be done |

**Table 4-2 How the Key Expansion functions and how the register changes**

From the table above, we can see that one round takes 16 cycles to calculate. $x_{15}..x_0$ are the previous round keys whilst $y_{15}$ to $y_0$ are the new round keys. These represent their corresponding bytes.

There is also control sequence to control the multiplexer operations just like the byte select unit. The multiplexers are numbered as m0..m2. They are shown on the figure above. M3 corresponds to the rcon calculator. The in_round port has the value of the current round. This is needed by the process to be implemented to calculate the Rcon. As soon as the round changes, the process is sensitive to it and it will calculate the appropriate Rcon value. The values are stored in a look up table.

### *4.4.6 Mix Columns*

The Mix columns unit entity is shown in the diagram below:-

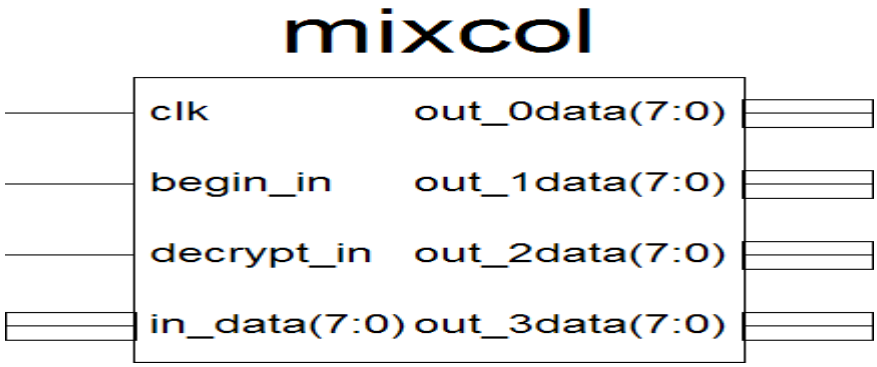


**Figure 4-9 Mix Columns entity**

The structure is shown in the logic diagram below. What is required is to process byte by byte perform mix columns when four bytes are received. The routine presented in the gold version will take a lot of area when implemented in FPGA. This is because it uses four bytes to store the bytes as they come in. It then performs Mix columns after all bytes are received. This will complicate the hardware and the design shown in [3] was found to be better.

The begin_in port is set to one when the first byte of every four byte word is received. This is where the registers of the mix columns operation are initialised. The design of the Mix columns shown in the cut-down version flowchart was adapted as represented in [3] so that it could operate on byte after byte. The structure of the Mix Columns is shown below:-

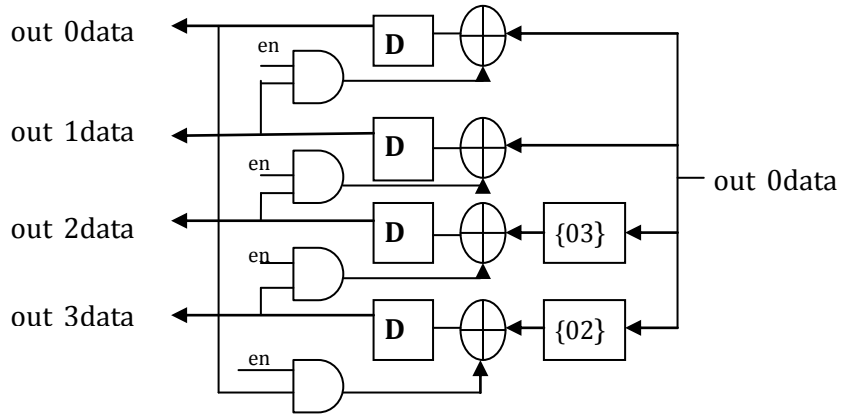


**Figure 4-10 Mix columns structure**

As explained before, it takes four clock cycles to do the mix columns process which is why the key has to be delayed by four cycles before Add round is performed. For the first cycle, the first by for the 4 byte word of the state comes in. The registers are numbered accumulator 0 to 3. Accumulator 0 is the top D flip flop. At the beginning, the begin_in port is set to one to signal the coming of the first byte of the four byte word which is processed. En is set to zero so that only the input byte is processed to update the register. Mix columns is done for every four byte word of the state array as explained in

Chapter 2. For the other 3 bytes, en is set to zero and the output from the other accumulator can be used for the processing of the incoming byte to obtain the Mix Columns output explained in Chapter 2. After 4 clock cycles, the out [0-3] data lines have the Mix columns result and the output is put to the Shift register so that it can be shifted in byte by byte to the Byte Select unit for the following round. {02} represents the xtime function or multiplication by two in the finite field scenario as explained in Chapter 2. {03} can be implemented by xtime followed by xoring with the incoming byte.

The decrypt_in port will be set to one when the decrypt function is added in. It needs a clock (clk) input because the processes are done at the rising edge of the clock cycle.

### *4.4.7 Shift Register*

The notes shown by [34] provide a simple structure of a shift register than can be implemented. This will be used in this project to take in 4 parallel lines from the Mix Columns sections and shift in the bits serially one by one into the Byte Select unit for the next round. This was selected to convert the 32 bits received from the Mix Columns sections to serial 8 bits output since we are using an 8 bit datapath. The shift register unit is used to convert the parallel lines which come out of the Mix columns unit to serial so that they can be shifted in into the Byte select unit. The structure of the entity is shown in the diagram below:-

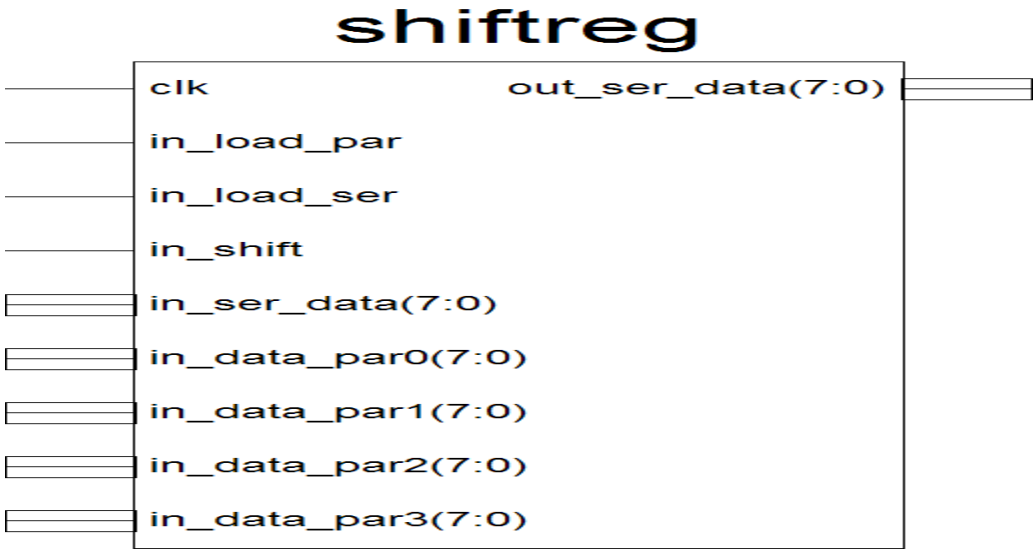


**Figure 4-11 Shift Register entity of the KES encryption system**

The ports labelled in_data_par[0..3] are connected to the Mix Columns output. The output port out_ser_data is the serial data output per clock cycle which is connected to the Byte select unit. When data first come in, it goes through the input port in_ser_data and it is shifted into the unit. When in_load_ser is set to one, data is shifted in per clock cycle into the Shift register. When in_load_par is set to one, the four registers in the Shift register are set by the input ports labelled in_data_par[0...3] from the Mix columns unit. Data is shifted in and out of the unit per clock cycle so we need a clock (clk) input to process this as per rising edge of the clock.

### *4.4.8 Top Block and Overall Structure*

The components mentioned in the previous sections are all part of the KES encryption entity which is the top block with signals (wires) that connect these entities. It also coordinates the starting of these entities using processes that are sensitive to the required watch signals. The structure of the top block to be implemented in VHDL is shown below:-

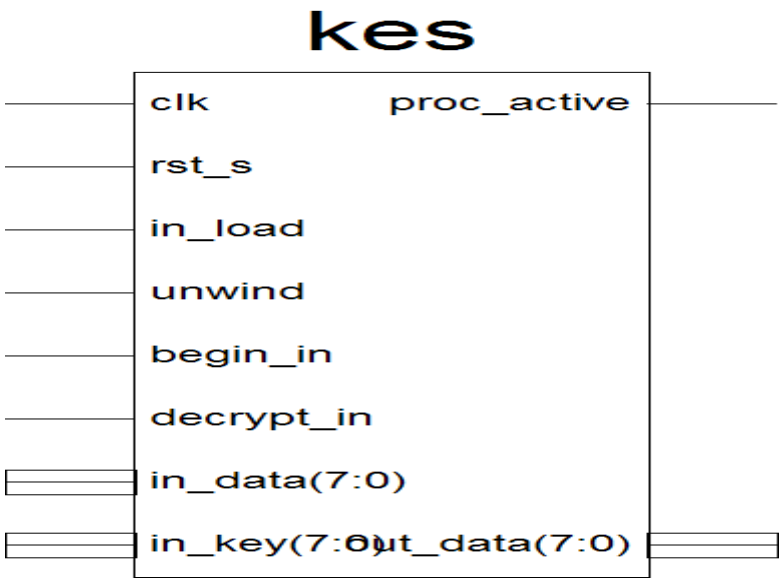


**Figure 4-12 Top block for KES encryption core**

The overall structure is shown below:-

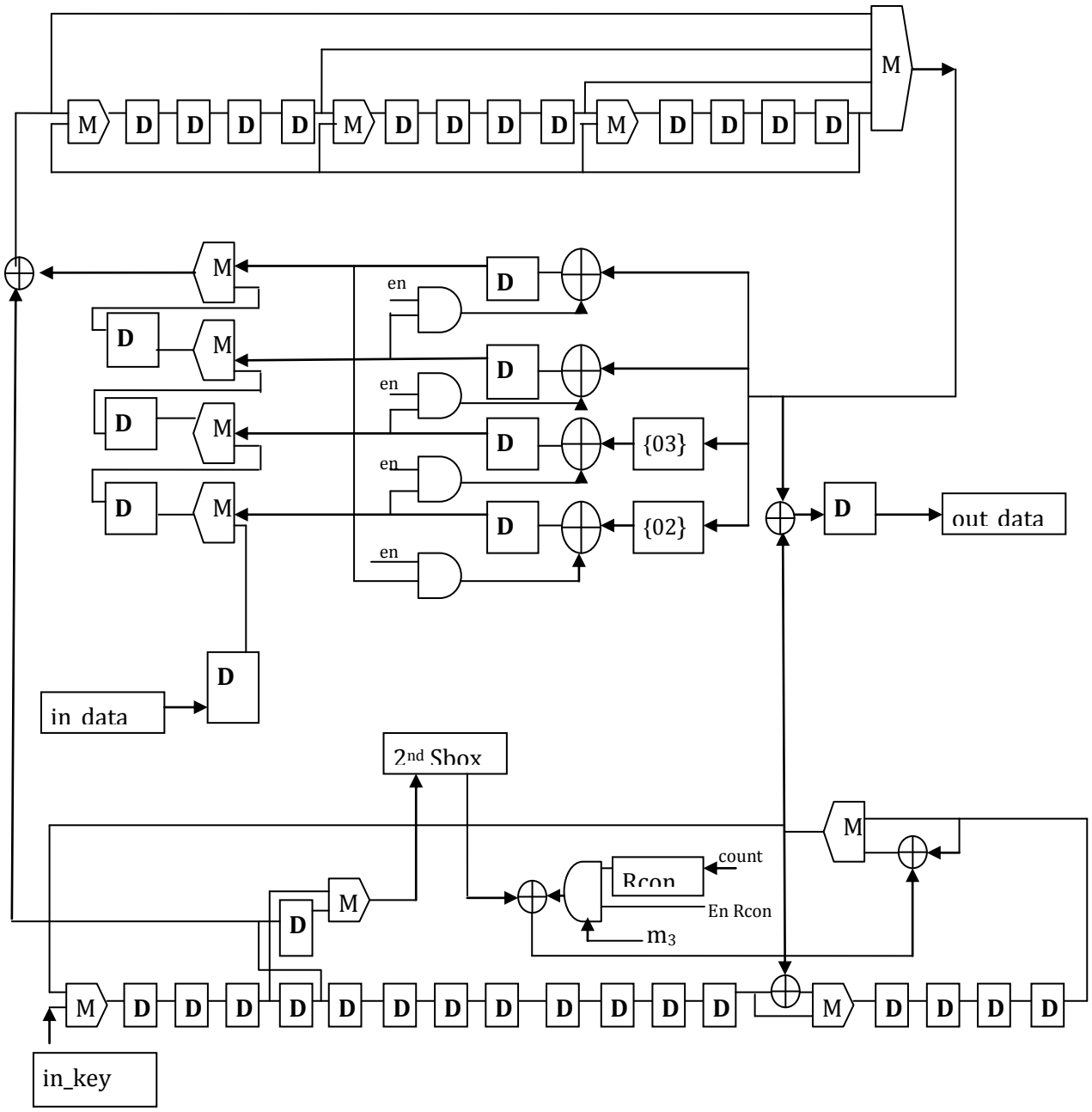


**Figure 4-13 Overall structure**

The clock input is needed to coordinate the components so that there operate in sequence. The in_key port has the incoming key which is connected to the Key expansion unit. The port in_data has the incoming data which is connected to the Shift register. The out_data port has the output data with the cipher text output byte by byte until all 16 bytes of the cipher output are out. The decrypt_in port is set to one if the Decrypt function is to be operated. In this project, the decrypt function is not implemented due to time constraints. This can be easily implemented in future by making use of this port and adding some components to implement decryption.

The unwind port is set to one when bytes are being output for the last round. This is where the whole system unwinds to output data as cipher text. This represents the last round in the AES encryption routine. Mix Columns is skipped and the last key is xored to the output of the first S-box. The port in_load is set to one when loading in data and the key. The port rst_s is set to low to reset the system otherwise throughout the operation of the encryption process, it is high. Initially, it is temporarily set to zero on the first test bench so that the signals can be initialised. The proc_active port is set to one when the encryption system is busy meaning process active. This comes after loading all data and calculations begin going through all rounds until the calculations are complete before unwinding.

The encryption system uses states which are feed, start and busy. Feed is when the bytes are being fed into the system and being loaded. Start is when processing starts and busy is when it is busy processing before outputting the results.

The encryption system contains three processes running in parallel. In VHDL, a process is a code section that may execute sequentially like in conventional programming languages. The processes are shown below

Process: check_c
Sensitivity list: clock (clk), Reset (rst_s)

Sequence of steps:-

- Get the next state and update it
- If feed state, set sequence to zero. If unwinding, sequence should increase by one per clock cycle so that the Byte select unit can perform Shift rows for the last round before doing sub-bytes and xoring the last key.
- If start state, set round to zero to show it is starting first round.
- If busy state, sequence should increase by one as the Byte select unit is in operation. If all sequence of steps are done then round should increase by one

Process: finalout
Sensitivity list: clock

Sequence of steps:-

- As per rising edge, if the byte select is active update output. This is to output something when the system is active.

Process: comb_cont (Combined Control unit)
Sensitivity List: state_cur (current state), ext_comb (combination of load and unwind bits. Unwind is least significant), rnd_ (round), seq_r (sequence), begin_in.

Sequence of Steps

- Initialise signals to zero i.e proc_active, and the components load and shift ports
- Select case of Current state
- If Feed, Set shift ports for Shift register, key expansion and byte select to shift the units as data unwinds. If load bit set to one, set load ports for byte select, shift register and key expansion. If load and unwind are set, Set load and shift for byte select and key expansion. Set load for

Shift register. If begin_in set one, the state should change to start otherwise the next state remains feed to continue loading data in.

- If current state is start, the next state should be busy as it is busy processing. If it is the beginning of every four byte word in sequence, the mix columns unit should be activated to initialise its accumulator and begin calculation.
- If busy is the current state, the sequence of steps are similar to the ones mentioned on the point above. Shift register parallel lines are loaded with data from mix columns when at the beginning of the four byte word. Shift bits are set for byte select unit, key expansion and shift register. Change state to feed when at the last round otherwise continue processing.

# 5. Results

This chapter presents the results of the experiments outlined in the Methodology chapter. They are presented in the order in which they have been stated in that chapter.

## 5.1 Experiment 1

The first experiment was about testing that the full version encryption algorithm is working correctly. In [1] an example is presented where the following input is supplied in hexadecimal:-

Key:- [2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c]
This is the key which was used for encryption for the other experiments that followed.

Plain Text:- [32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34]

The cipher text (output) was [39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32]

This is the correct output as shown in [1].

The following was also put in :-

Key:-[ 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f]

Plain Text:- [00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff]

The output was [69 c4 e0 d8 6a 7b 04 30 d8 cd b7 80 70 b4 c5 5a]

This is the correct output as shown in [1]

This shown the full version is working as expected.

## 5.2 Experiment 2

The experiment shown above was repeated for the Gold version. Since the full version is cut-down to seven rounds for the gold version, the output expected is different from the one shown above. It is the outputs of seven rounds with the last add round adding the round key for the seventh part.

The output was [b9 d7 b7 31 78 1c f4 03 1f 13 7e 4d 1a 0d 75 9c]

The other output was [a0 a1 62 56 8b e9 68 8d 0f 93 27 63 11 bc 95 6a ]

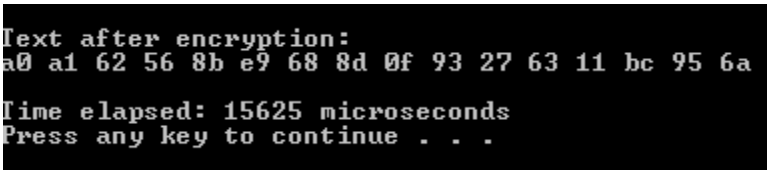


**Figure 5-1 Screen shot for gold version serial execution**

This is the correct output as shown in [1] if the output of round seven before mix columns operation is xored to the seventh round key.

## 5.3 Experiment 3

The following table shows the results of the execution times for the encryption algorithms for the gold and the full version.

| File | Full version (Milliseconds) | Gold version (Milliseconds) |
|---|---|---|
| Picture1.jpg (130 KB) | 268.63 | 243.621 |
| Document1.pdf (433 KB) | 1107.501 | 815.215 |
| Music1.mp3 (12.8 MB) | 23 991.049 | 23 354.087 |
| Video1.mp4 (20 MB) | 37 216.152 | 36474.769 |
| Sample1.txt (2 KB) | 12.428 | 7.751 |
| Sample2.txt (32 bytes) | 0.545 | 0.406 |

**Table 5-1 Encryption Times**

The following table shows the decryption times

| File | Full version (Milliseconds) | Gold version (Milliseconds) |
|---|---|---|
| Picture1.jpg (130 KB) | 464.201 | 309.216 |
| Document1.pdf (433 KB) | 1539.481 | 991.249 |
| Music1.mp3 (12.8 MB) | 43120.51 | 30863.914 |
| Video1.mp4 (20 MB) | 67577.14 | 48288.194 |
| Sample1.txt (2 KB) | 14.418 | 9.721 |
| Sample2.txt (32 bytes) | 0.439 | 0.375 |

**Table 5-2 Decryption Times**

The following graph compares the execution times for encryption and decryption for the full version.

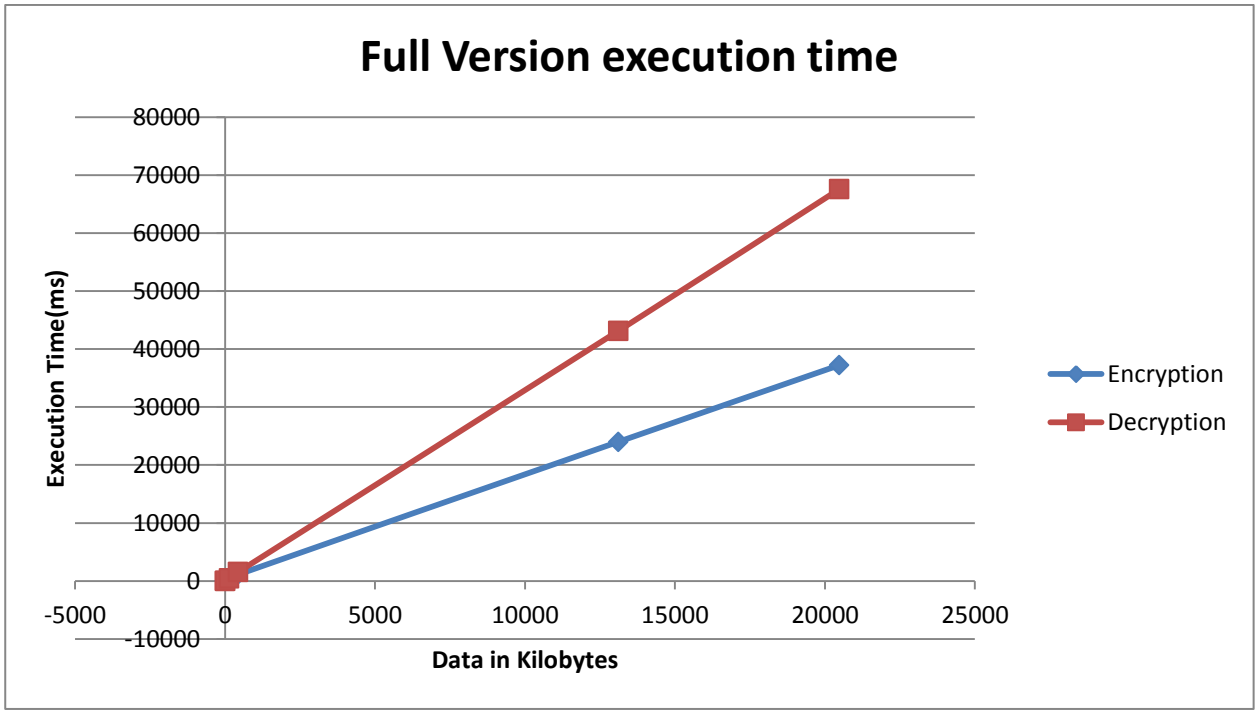


**Figure 5-2 Full Version Execution time for encryption and decryption**

The following graph compares the execution times for encryption and decryption for the cut-down (gold) version.

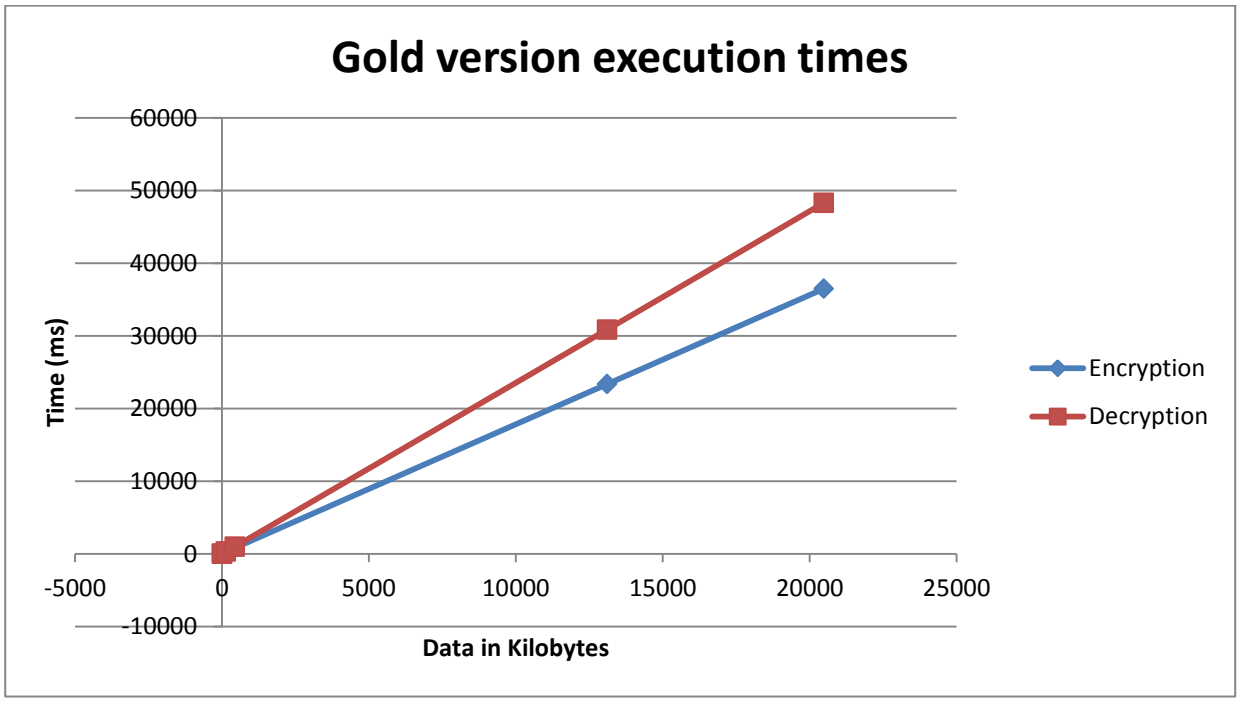


**Figure 5-3 Gold version Execution time for encryption and decryption Experiment 4**

### *5.3.1 Comparisons*

The results from the Matlab correlation tests were tabulated as shown below:-

| File | Full Version | Gold Version |
|---|---|---|
| Picture1.jpg (130 KB) | 0.001 | 0.0017 |
| Document1.pdf (433 KB) | 0.0007 | 0.0003 |
| Music1.mp3 (12.8 MB) | 0.0001 | 0.0002 |
| Video1.mp4 (20 MB) | 0.0005 | 0.0012 |
| Sample1.txt (2 KB) | 0.0155 | 0.0050 |
| Sample2.txt (32 bytes) | 0.2358 | 0.1751 |

**Table 5-3 Tabulation of correlation values**

This is displayed in the graph below:-

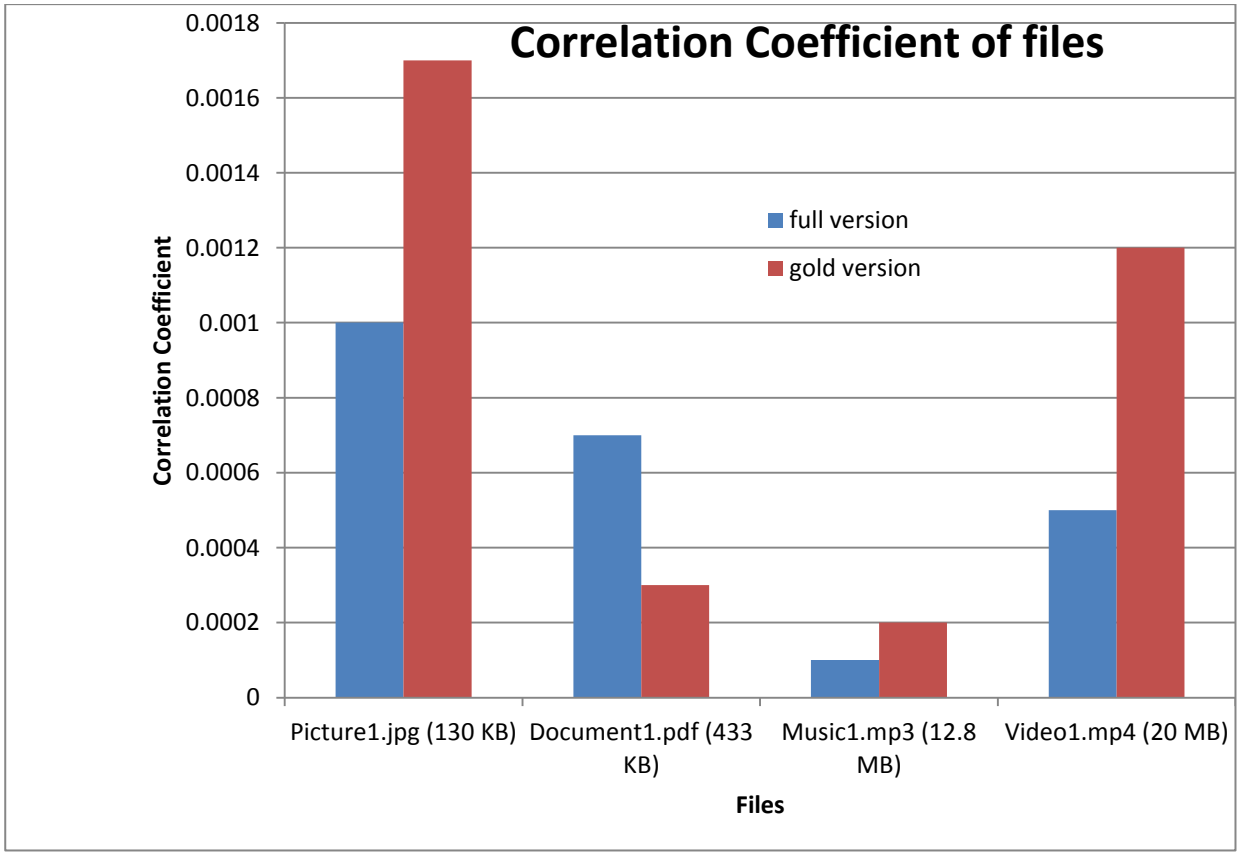


**Figure 5-4 Correlation of encrypted text and plain text**

The following table shows a tabulation of the entropy measured values

| File | Plain Text | Full Version encrypted | Gold Version encrypted |
|---|---|---|---|
| Picture1.jpg (130 KB) | 7.9882 | 7.9988 | 7.9987 |
| Document1.pdf (433 KB) | 7.9824 | 7.9995 | 7.9995 |
| Music1.mp3 (12.8 MB) | 7.9675 | 7.9999 | 7.9999 |
| Video1.mp4 (20 MB) | 7.9531 | 8 | 8 |
| Sample1.txt (2 KB) | 4.4852 | 7.9086 | 7.8919 |
| Sample2.txt (32 bytes) | 3.1014 | 4.875 | 4.9375 |

**Table 5-4 Entropy values**

### *5.3.2 Histograms*

The following histograms show the hexadecimal values representing data in files before encryption and after encryption. The values representing the bytes range from 0 to 255 as shown. What is written as Intensity level usually applies to images but to general files reflect the value at that byte which represents the information in the file. These histograms were generated in Matlab. They compare the original file to the encrypted file. Selected ones are presented here. The rest of them are found in Appendix A in Chapter 10.

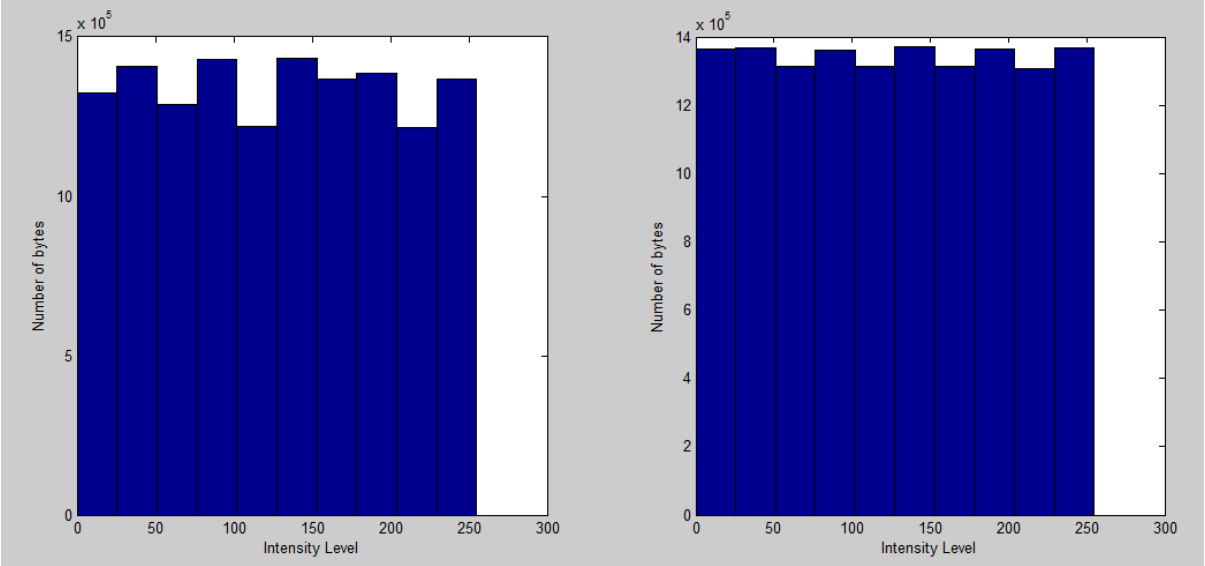


**Figure 5-5 Music.mp3 (12.8 MB) Full Version Histogram**

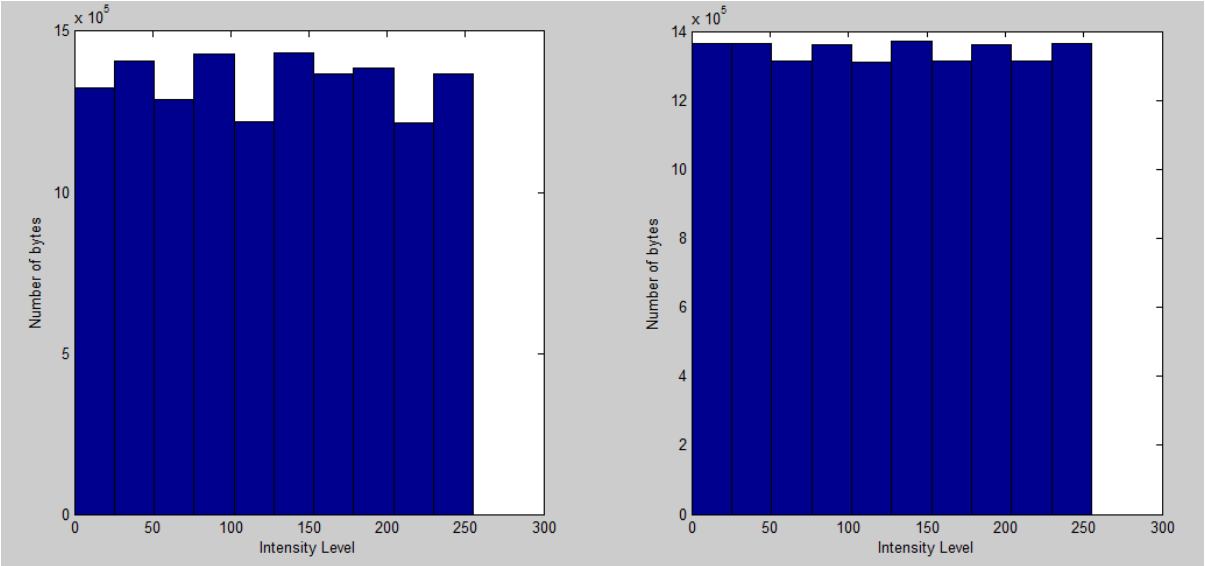


**Figure 5-6 Music.mp3 (12.8 MB) Gold Version Histogram**

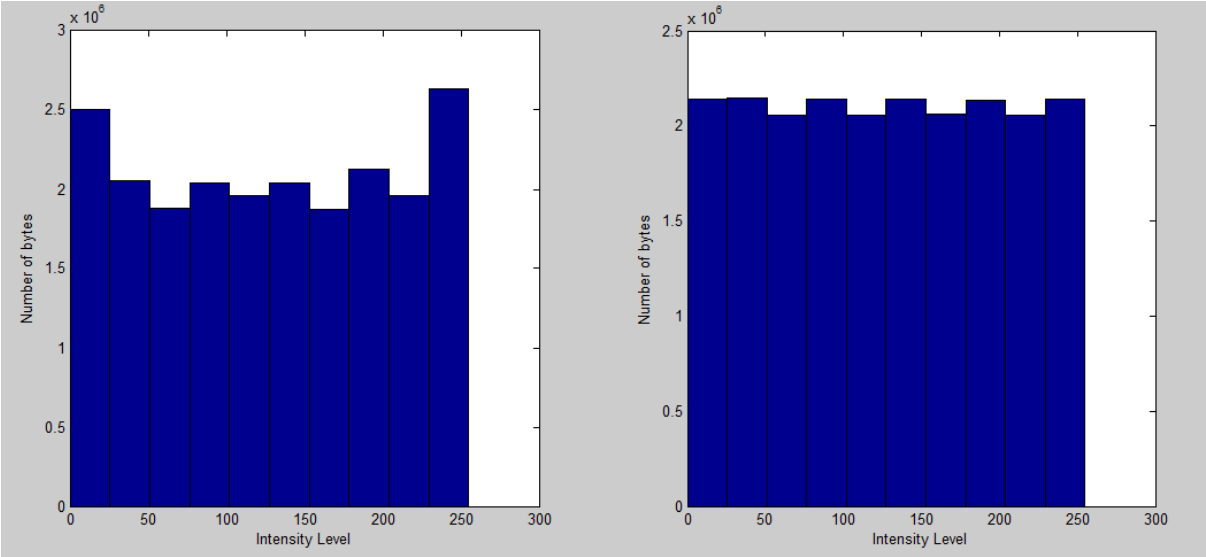


**Figure 5-7 Video1.mp4 (20 MB) Full Version Histogram**

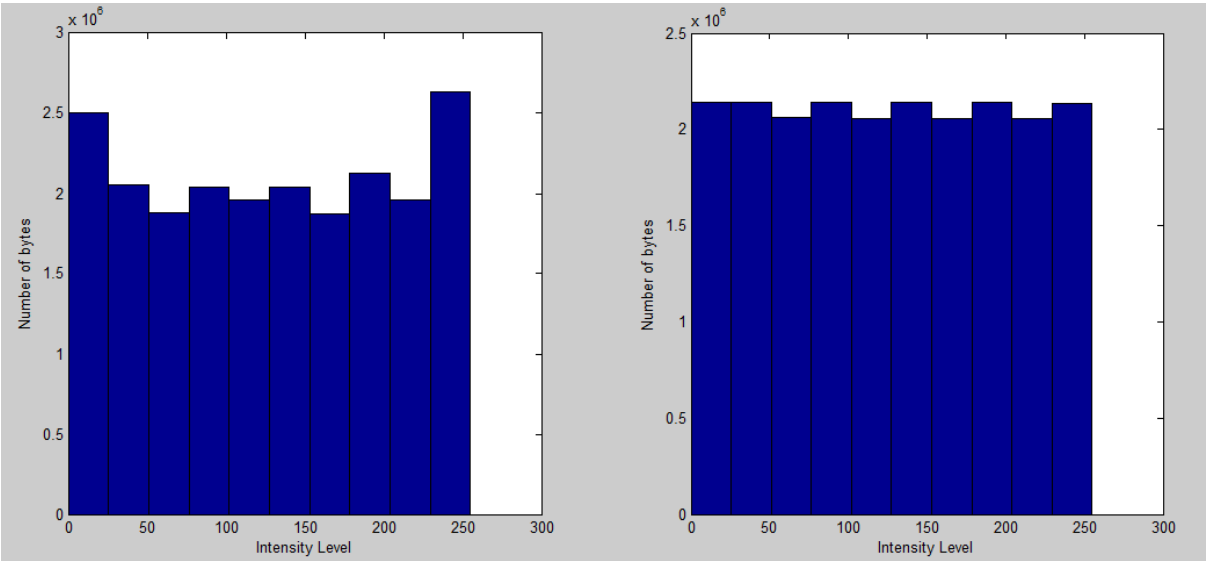


**Figure 5-8 Video1.mp4 (20 MB) Gold Version Histogram**

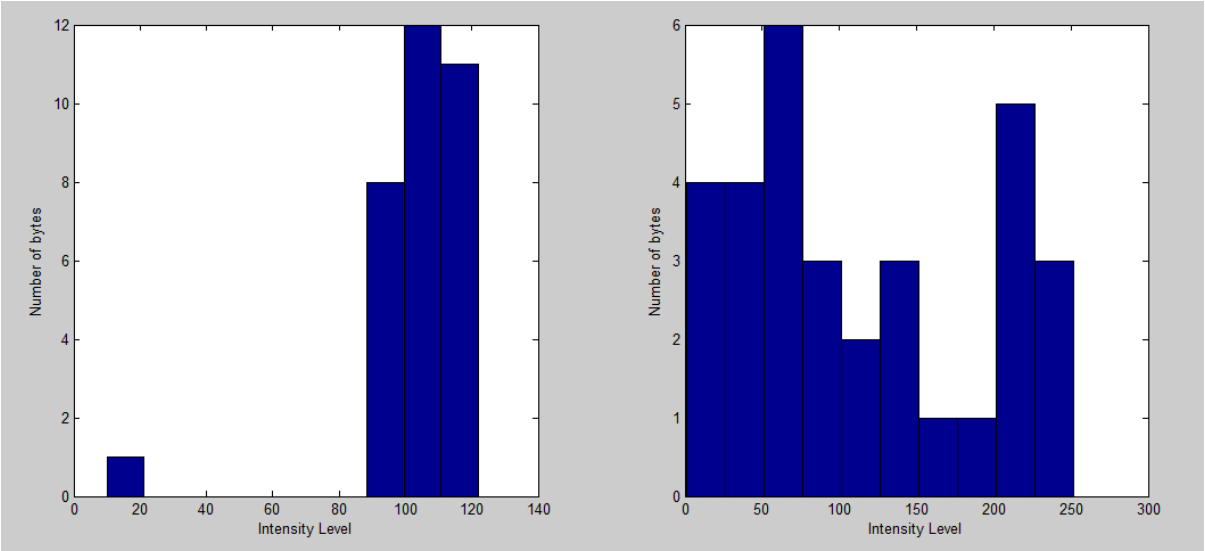


**Figure 5-9 Sample2.txt (32 bytes) Full Version Histogram**

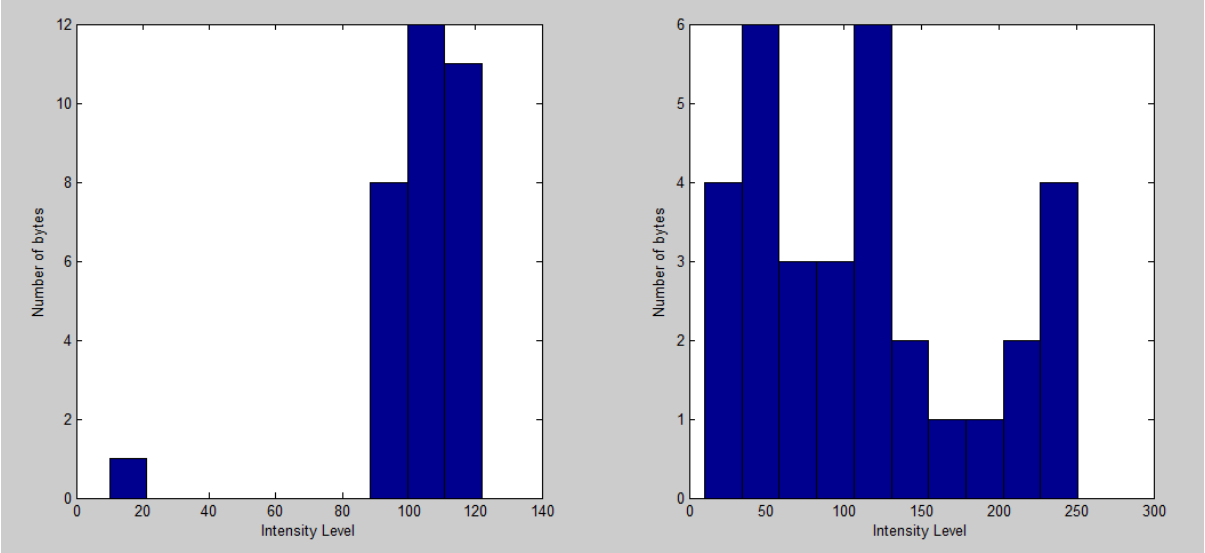


**Figure 5-10 Sample2.txt (32 bytes) Gold Version Histogram**

## 5.4 Experiment 5

The data recorded in Experiment 2 was fed to the FPGA system to see that it was getting the correct output. The following shows a screen shot of the simulation that shows it was getting the correct results displayed in the screen shot in 5.2.

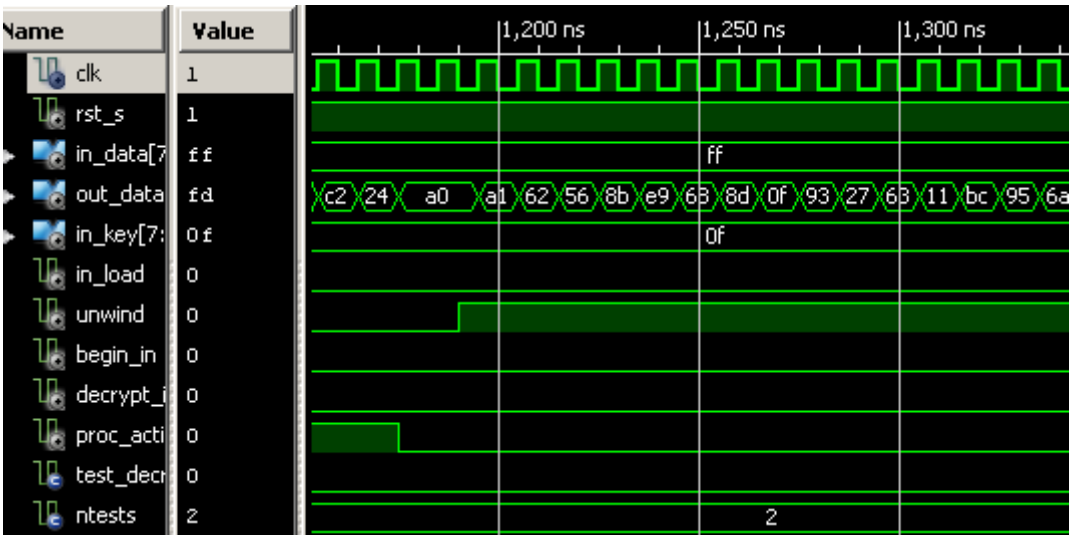


**Figure 5-11 Simulation output for FPGA version**

This shows the system unwinding and producing output. The port written out_data is for the KES top block which shows the output when unwind becomes 1. This shows that the output is [a0 a1 62 56 8b e9 68 8d 0f 93 27 63 11 bc 95 6a] as indicated before which shows the FPGA version is accurate. This was also verified for the other result.

The FPGA version was fed with 32 bytes. It took 2665 ns according to the simulation. This is compared to the gold version's value of 406 µs. This shows a speed up of (Gold Version / FPGA version) = 406 µs / 2665 ns = 152.35.

One cycle to output results takes about 1345 ns. It was found that 128 bytes take about 8525 ns. This represents a throughput of 14.319 Mbytes/s. This shows that over a number of clock cycles, speed up will increase because of the parallelising ability.

Compilation output:-

```
Device utilization summary:
---------------------------

Selected Device : 3s500efg320-4

 Number of Slices:                      470
 Number of Slice Flip Flops:            191
 Number of 4 input LUTs:                873
    Number used as logic:               825
    Number used as Shift registers:      48
 Number of IOs:                          31
 Number of bonded IOBs:                  30
 Number of GCLKs:                         1
```

**Figure 5-12 Device utilization and Compilation output from simulation**

# 6. Discussion

This chapter discusses the results obtained in Chapter 5 so as to aid to make conclusions and answer the questions presented in 1.2.1.

## 6.1 Experiment 1 and 2

The results of the experiments show that the implementation of the algorithms was done correctly. They also verify the correctness of the routines mentioned in [1] which are used to do the encryption process. After this was verified, the algorithms were modified to add features to read from files and encrypt all the data.

## 6.2 Experiment 3

The execution times for the encryption process for the full version and the gold version show a linear relationship between the size of data being encrypted and the time it takes to encrypt. This is was disadvantage of the serial encryption process without the implementation of parallelism. The gold version takes less time to compress because some rounds have been cut. However, it also goes through some loops as it handles one byte at a time to model the FPGA version which will have an 8 bit data path.

The execution times for the decryption part show that the decryption part takes longer to execute than the encryption part. This is partly due to the Mix columns process which takes longer on the decrypt part than the encrypt part. This is so because there are a lot of different values to multiply and calculate. This is also expected for the FPGA version that when the decrypt part is added, it will use more resources than the encrypt part.

The graphs represented in Figures 5.2 and 5.3 show that the gold version executes faster than the full version. This is due to the fact that some rounds have been cut. It also shows the linear relationship mentioned between the size of data and the time it takes to encrypt it.

## 6.3 Experiment 4

Experiment 4 was performed in Matlab to measure the correlation coefficient between encrypted files and the plain text or original files. As explained in the Methodology chapter, this measures the performance of an encryption system. It measures the security feature of the encryption as it is difficult to break into encrypted data and recover the original information without having the cipher key and not knowing the encryption process used.

From the results, it can be seen that there is a very low correlation between encrypted data and original data which means the AES encryption process is safe as discussed in the Literature review compared with other encryption systems.

It can be shown that correlation measurement is not reliable for small sizes of data. This is shown by the measurements taken on the 32 bytes file and the 2 KB file. The small sizes show the correlation coefficient of the cut-down version being lower than that of the Full version. This cannot be expected since some rounds are cut-off meaning the encryption power is weaker.

The results are more reliable on the large sizes of data. The correlation coefficient measured on the video and music files show that the value is lower for the Full version than on the Gold version. This is due to the fact that encryption power is sacrificed on the cut-down version to allow it to complete the process faster. The exception could be the result of the Document1.pdf which showed a correlation value of the gold version as lower than the Full version. It would also be better to take random samples of the bytes and obtain better values as explained in [25-27]. Widening the number of samples will also reveal the fact that the cut-down version is weaker in terms of encryption power.

The entropy values do not reveal much about the comparisons between the Full and the gold version. This is because they are almost equal but in most instances reveal that the full version has a higher entropy because there is more data variability. This is as a result of the processing of more rounds which tends to strengthen the encryption. The results show that the encrypted files have a higher entropy than original files. This means there is more data variability which means it is much safer. It also means when compressing it would take larger space than the original files.

Analysis of Histograms

The histograms reveal that for encrypted files, the frequency of bytes with the values ranging from 0 to 255 is almost level and uniform. They compare the original file and the encrypted file. This reveals more variability as seen from the entropy results showing that the data is much more difficult to predict.

The small sizes of data have text that has hex values that are bound in certain ranges as shown in the graphs. The encryption process tries to spread through and increase the data variability so as to strengthen it. It can be seen from these histograms that there is greater variability in the full version encrypted files than in the gold version showing stronger encryption.

## 6.4 Experiment 5

The simulations done in the Xilinx ISE show that the FPGA version was able to encrypt the data as expected because the output revealed the correct results.

It took 2665 ns to encrypt 32 bytes whereas it took 406 µs for the gold version. This shows an incredible speed up of 152.35. The speed up increases when more bytes are loaded. It took 8525ns to encrypt 128 bytes on FPGA. This would take roughly 1624 µs on PC considering the linear relationship. This would represent a speed up of 190.5. This is due to the fact that the operation is parallelised. It allows simultaneous loading and unloading of data. So the execution time does not follow a linear relationship.

The compilation report shows that it takes up 470 slices on FPGA. This represents about 10% utilization on a Spartan 3 of 4656 slices. Compared to other encryption systems referred to in the Literature Review in 2.1.4, it takes much less area than the one shown in [24]. The utilization will

increase when the decryption part is added. After that, a multi-core system can be designed to increase the parallelisation and utilization of an FPGA system. Otherwise if implemented on an ASIC of low area for the application stated of RFID tags, it could be quite ideal. The multi-core system can also be added to make it applicable to a number of applications. William Whitehouse's project shown in [32] is an example of how a single core system like the one implemented in this project can be made to be multi-core. This would greatly increase the throughput.

# 7. Conclusions

From the results gathered, we got enough information to answer the questions initially posed in 1.2.1. The questions are answered in the sections below.

## 7.1 Advantages of Parallelisation

The first question posed was to compare the performance of the parallelised system compared to the gold version running on PC which does the work albeit slower. It can be seen from the results and the speed up that was calculated that parallelisation has the advantage of speeding up processing. This shows that FPGAs are an ideal hardware platform to implement high performance systems like the Encryption system investigated in this project.

## 7.2 Encryption Strength of cutting down the encryption process

It can be seen from the graphs obtained and the correlation values that the cut-down version has slightly weaker encryption strength compared to the full version. However this is quite adequate to encrypt ordinary data. It might not be sufficient to apply on sensitive information which needs to be highly secured. However, it can be sufficient for a low area platform when we are limited by area. In this project, we were not much limited by area as shown on the compilation result of the FPGA used. It would therefore be recommended to use the full version on the supplied FPGA even though it can take quite longer if we are to encrypt sensitive information which demands higher security.

## 7.3 Improvement of Speed of the Cut-down version

It can be seen from the initial tabulated results that the cut-down version executes faster than the full version. It thus shows that when we have an application that operates in real time on ordinary data and when latency is important, the cut-down version is quite ideal. However for an application where we would mind doing it for a longer time to put more encryption power, the full version is recommend.

## 7.4 Final Conclusion

From the results obtained, it can be seen that the developed system satisfies the project requirements meaning the project was a success. All the terms of reference listed at the beginning of this report were satisfied as shown by the results.

# 8. Recommendations

The following recommendations are made for future work:-

- It would be appropriate to change the design from an 8 bit system to maybe 32 bit of higher. This will allow the encryption process to complete quicker. The processing of Mix columns takes 32 bits at a time. For the 8 bit system it operates in four clock cycles. However, if the 32 bit system is implemented, it will take a single cycle. The system will also be able to utilize more area on the provided Spartan 3 since it is not utilised much in this project.

- It would also be appropriate to increase utilization by changing the design of the Key expansion routine so that round keys can be stored and retrieved as it operates. The fact that it has to calculate on the fly will repeat some work already done and therefore waste some clock cycles. For higher performance and to utilise the FPGA platform much more, the pre-storing of round keys can be better.

- It would also be appropriate to change the design to a multi-core system. This was demonstrated by the project recorded in [32]. In this project, a 11 core system is produced out of Jerzy Gbur's project shown on the Open Cores site. This allowed a 7x improvement on throughput. It is thus quite advantageous to implement this multi-core design and increase the parallelisation for higher performance.

# 9. List of References


[1] FIPS Publication 197, “Announcing the Advanced Encryption Standard”, November 26 2001.

[2] A. Menezes, P. van Oorschot and S. Vanstone, “Handbook of Applied Cryptography”, CRC Press, New York, 1997, p. 81-83.

[3] Panu Hämäläinen, Timo Alho, Marko Hännikäinen, Timo D. Hämäläinen,"Design and Implementation of Low-area and Low-power AES Encryption Hardware Core", 9th Euromicro Conference on Digital System Design - Architectures, Methods and Tools (DSD 2006), Cavtat, Croatia, August 30, 2006 - September 1, 2006.

[4] Daemen, Joan and Rijmen Vincent, “The Design of Rijndael – The Advanced Encryption Standard”, 2002, Springer.

[5] Chi-Jeng Chang , Chi-Wu Huang , Kuo-Huang Chang , Yi-Cheng Chen and Chung-Cheng Hsieh, “High Throughput 32-bit AES Implemenation in FPGA” , Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference.

[6] Tim Good and Mohammed Benaissa, “Very Small FPGA Application-Specific Instruction Processor for AES”, IEEE Transactions on Circuit and Systems-I, Vol. 53, No. 7, July 2006.

[7] Seagate – Technology Paper, “128 Bit Versus 256 Bit AES Encryption”, Practical business reasons why 128 bit solution provide comprehensive security for every need. www.seagate.com/staticfiles/docs/pdf/whitepaper/tp596_128_bit_versus_256_bit.pdf.

[8] Philippe Bulens, Fran¸cois-Xavier Standaert, Jean-Jacques Quisquater, Pascal Pellegrin, Ga¨el Rouvroy, “Implementation of the AES-128 on Virtex-5 FPGAs”, UCL Crypto Group, Belgium.

[9] K. K. Parhi, “Systematic synthesis of DSP data format converters using life-time analysis and forwardbackward register allocation,” IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing, vol. 39, no. 7, pp. 423–440, Jul. 1992.

[10] Chih-Peng Fan and Jun-Kui Hwang, “FPGA implementations of high throughput sequential and fully pipelined AES algorithm”, Department of Electrical Engineering, National Chung Hsing University, 2008.

[11] Chi-Wu Huang, Chi-Jeng Chang, Mao-Yuan Lin, Hung-Yun Tai, “Compact FPGA Implementation of 32-bits AES Algorithm Using Block RAM”, Institute of Applied Electronics Technology, Department of Industrial Education: National Taiwan Normal University.

[12] Somsak Choomchuay, Surapong Pongyupinpanich and Somsanouk Pathumvanh, “A Compact 32-bit Architecture for an AES System”, ECTI transactions on computer and information theory vol.1, no.1 May 2005.

[13] Gael Rouvroy, Francois-Xavier Standaert, Jean-Jacques Quisquater and Jean-Didier Legat, “Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very well suited for small embedded applications”, UCL Crypto group.

[14]Ashwini M. Deshpande, Mangesh S. Deshpande and Devendra N. Kayatanavar, “FPGA Implementation of AES Encryption and Decryption”, International conference on Control, Automation, Communication and Energy conservation, 2009.

[15] Kris Gaj and Pawel Chodowiec, “FPGA and ASIC Implementations of AES”, ECE Department, George Mason University.

[16] Kutin Sandy, “DES, Triple-DES, and AES”, University of Chicago, 2001.

[17] Majithia Sachin, Dinesh Kumar, “Implementation and Analysis of AES, DES and Triple DES on GSM Network”, IJCSNS International Journal of Computer Science and Network Security, January 2010.

[18] W.Stallings, ''Cryptography and Network Security 4th Ed,'' Prentice Hall , 2005,PP. 58-309.

[19] Douglas J. Smith, “VHDL and Verilog Compared and Contrasted – Plus modeled example written in VHDL, Verilog and C.”, VeriBest Incorporated.

[20] Kahlen F J Prof., “Project Management: MEC2026S Assignment and Supplementary Information”, University of Cape Town, 2012.

[21] Suhas J Manangi, Parul Chaurasia, Mahendra Pratap Singh, “Simplified AE S f or Low Memory Embedded Processors”, Global Journal of Computer Science and Technology, 2010.

[22] Joseph Zambreno, David Nguyen, and Alok Choudhary, “Exploring Area/Delay Tradeoffs in an AES FPGA Implementation”, Northwestern University.

[23] Ognjen Šćekić, “FPGA Comparative Analysis”, Belgrade 2005

[24] Li, C., Q. Zhou, Y. Liu and Q. Yao, “Cost-efficient data cryptographic engine based on FPGA”, 4th International Conference on Ubi-Media Computing, July 3-4, 2011.
[25] K.Sumathy, R.Tamilselvi, “Comparison of Encryption Levels for Image Security Using Various Transforms”, RMK Engineering College, 2011.
[26] Varsha Bhatt, Gajendra Singh Chandel, “Implementation of new advance image Cryption algorithm to enhance security of Multimedia component”, Sai Institute Of Science & Engineering.
[27] Karen Parnell, Roger Bryner, “Comparing and Contrasting FPGA and Microprocessor System Design and Development”, Xilinx white paper, July 2004.
[28] Ashenden, Peter J., “The designer's guide to VHDL”, Morgan, 2008
[29] Kilts, Steve, “Advanced FPGA design: architecture, implementation, and optimization”, Wiley, 2007.
[30] Digilent Adept, “Digilent Adept Suite User’s Manual”, 2006.
[31] Xilinx, “Spartan-3E FPGA Family: Data Sheet”, August 2009.
[32] Whitehouse, William, “FPGA Implementation of Multicore AES 128/192/256”, 2010.
[33] Thomas Ruschival, “AES 128/192/256 (ECB) Avalon® -MM Slave”, Open cores, 2011.
[34] CSIT, “Reconfigurable Digital Systems Lectures”, Politechnica University of Bucharest, 2012.
[35] Journal of Systems Architecture, “Embedded Hardware for Cryptosystems “, Volume 53, Issues2-3, February-March 2007, Pages 69-71.
.

# 10. Appendices

## 10.1 Appendix A

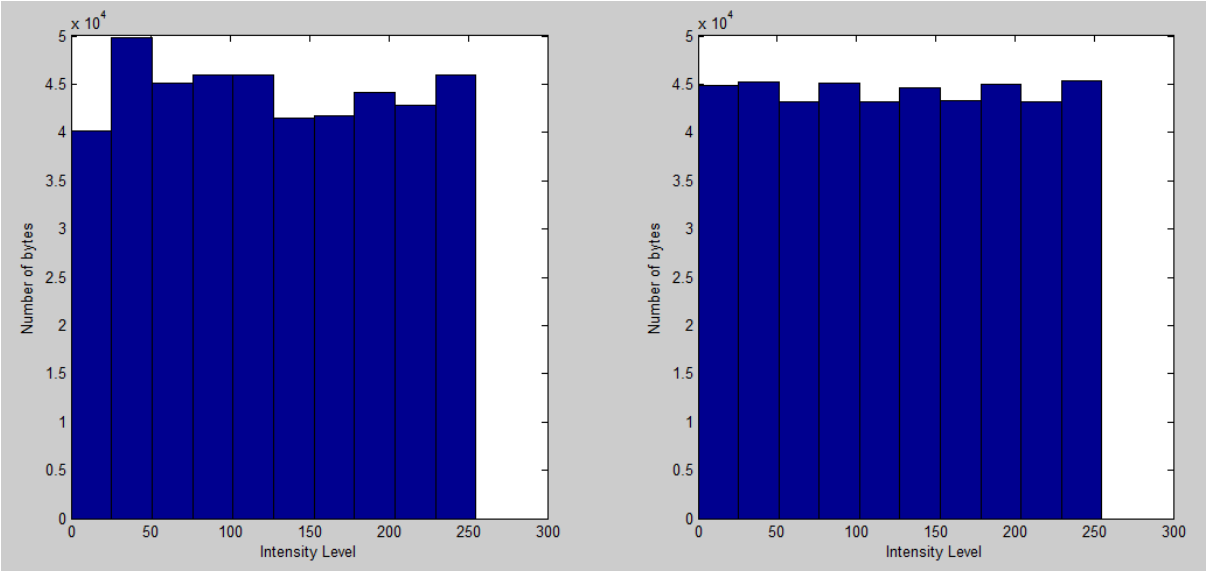


**Figure 10-1 Document1.pdf (433 KB) Full Version Histogram**

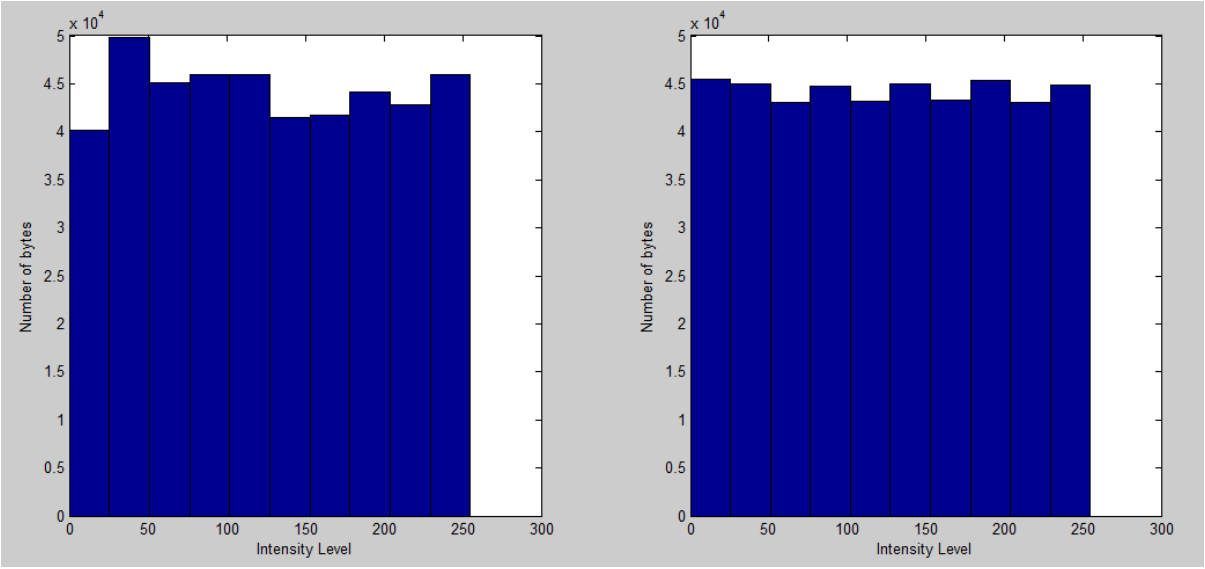


**Figure 10-2 Document1.pdf (433 KB) Gold Version Histogram**

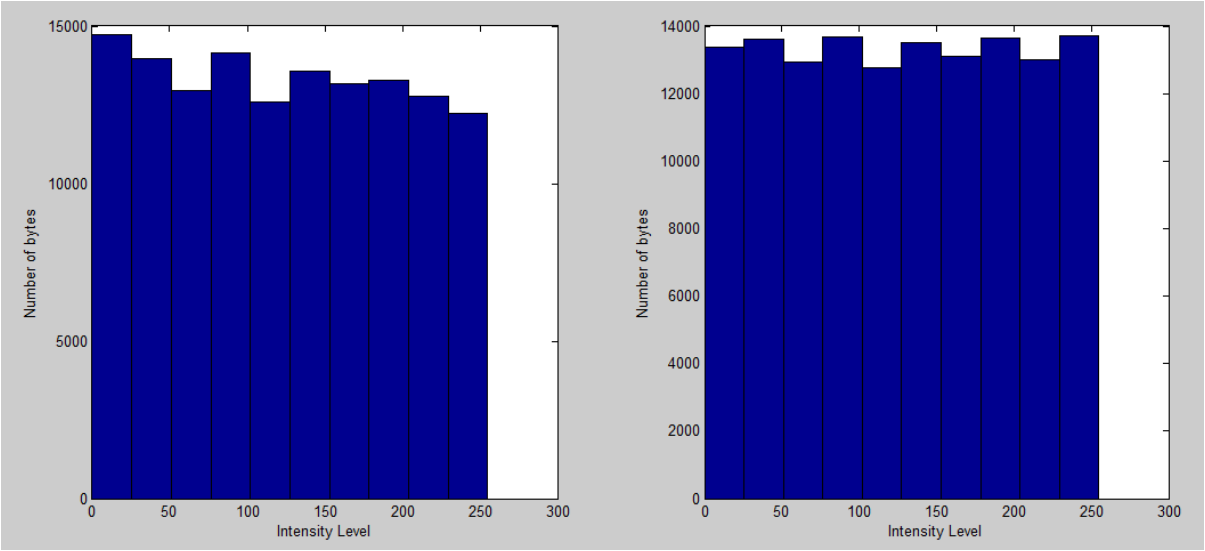


**Figure 10-3 Picture1.jpg (130 KB) Full Version Histogram**

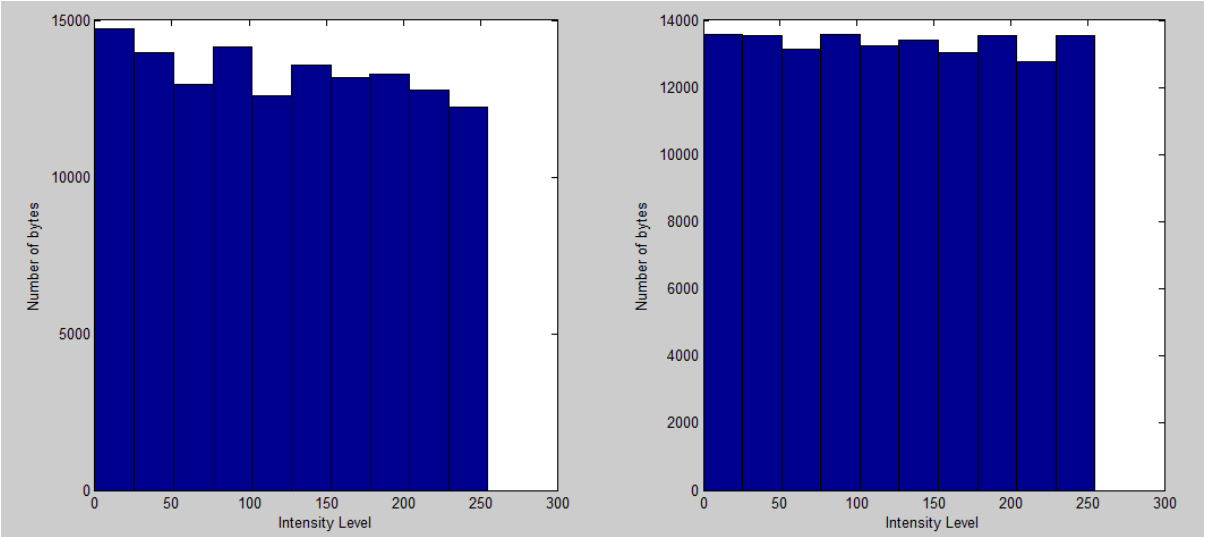


**Figure 10-4 Picture1.jpg (130 KB) Gold Version Histogram**

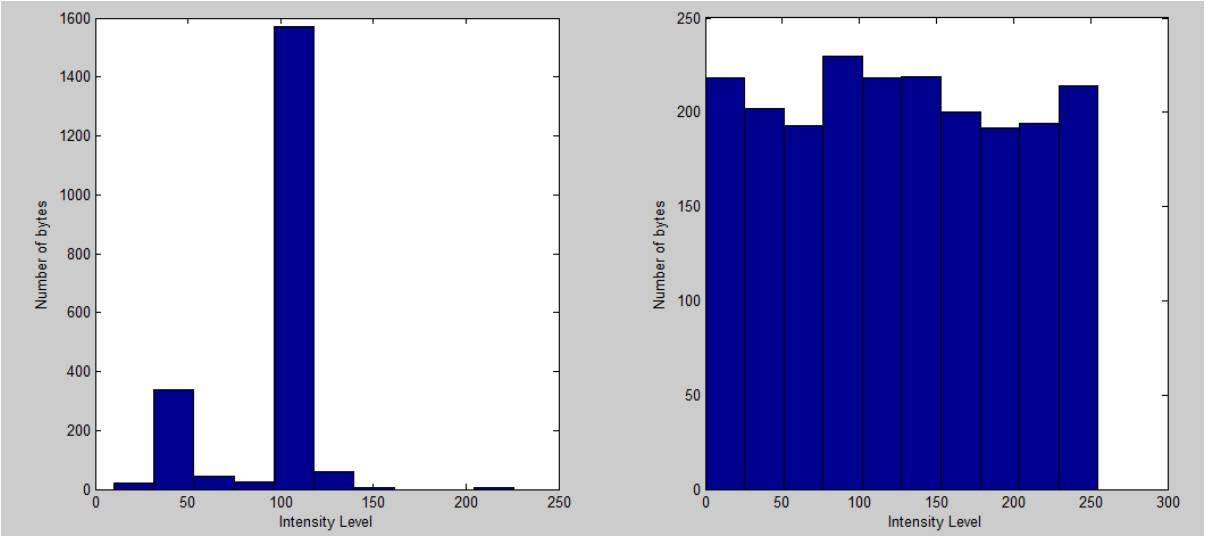


**Figure 10-5 Sample1.txt (2 KB) Full Version Histogram**

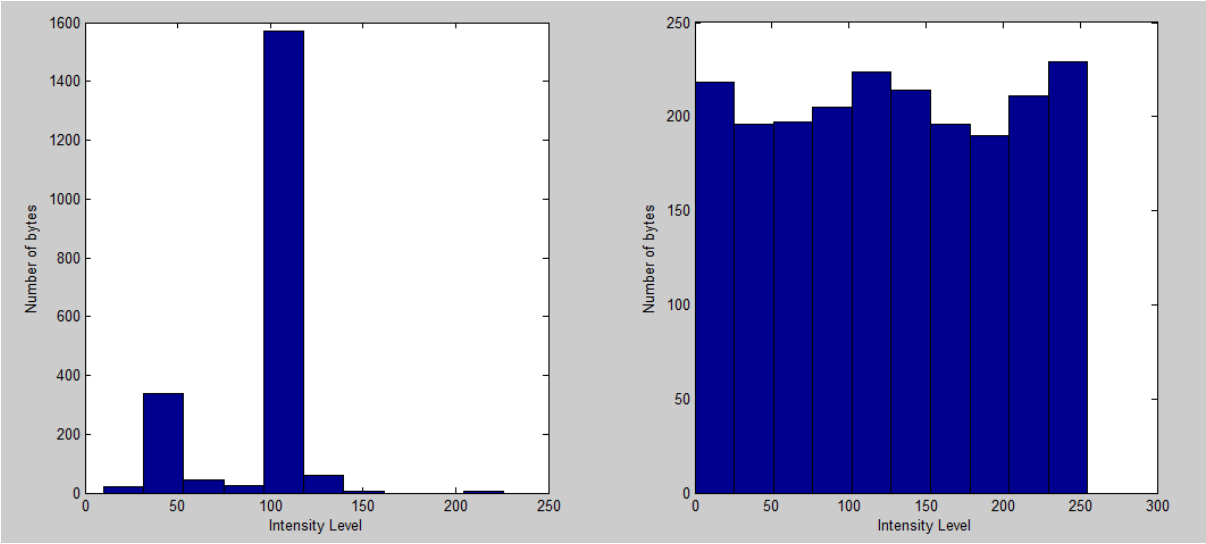


**Figure 10-6 Sample1.txt (2 KB) Gold Version Histogram**

## 10.2 Appendix B

The code used in this project is found in the attached CD.

# 11. EBE Faculty: Assessment of Ethics in Research Projects

Any person planning to undertake research in the Faculty of Engineering and the Built Environment at the University of Cape Town is required to complete this form before collecting or analysing data. When completed it should be submitted to the supervisor (where applicable) and from there to the Head of Department. If any of the questions below have been answered YES, and the applicant is NOT a fourth year student, the Head should forward this form for approval by the Faculty EIR committee: submit to Ms Zulpha Geyer (Zulpha.Geyer@uct.ac.za; ChemEng Building, Ph 021 650 4791).Students must include a copy of the completed form with the final year project when it is submitted for examination.

**Name of Principal Researcher/Student:** Killian T. Mazibuko **Department:** ELECTRICAL ENGINEERING

**If a Student:** YES **Degree:** BscEng Electrical and Computer **Supervisor:** Dr Simon Winberg

**If a Research Contract indicate source of funding/sponsorship:** N/A

**Research Project Title:** SW-13 PARALLELISED DECIPHER ENGINE

Overview of ethics issues in your research project:

| | | |
|---|---|---|
| **Question 1: Is there a possibility that your research could cause harm to a third party (i.e. a person not involved in your project)?** | YES | NO |
| **Question 2: Is your research making use of human subjects as sources of data?** If your answer is YES, please complete Addendum 2. | YES | NO |
| **Question 3: Does your research involve the participation of or provision of services to communities?** If your answer is YES, please complete Addendum 3. | YES | NO |
| **Question 4: If your research is sponsored, is there any potential for conflicts of interest?** If your answer is YES, please complete Addendum 4. | YES | NO |

If you have answered YES to any of the above questions, please append a copy of your research proposal, as well as any interview schedules or questionnaires (Addendum 1) and please complete further addenda as appropriate.

I hereby undertake to carry out my research in such a way that

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

Signed by:

| | **Full name and signature** | **Date** |
|---|---|---|
| **Principal Researcher/Student:** | **Killian T. Mazibuko** | 20 October 2012 |

This application is approved by:

| | | |
|---|---|---|
| **Supervisor (if applicable):** | **Dr. Simon Winberg** | 20 October 2012 |
| **HOD (or delegated nominee):** Final authority for all assessments with NO to all questions and for all undergraduate research. | **Janine Buxey** | 20 October 2012 |
| **Chair : Faculty EIR Committee** For applicants other than undergraduate students who have answered YES to any of the above | | |

**ADDENDUM 1:**

| | |
|---|---|
| ID: | **SW-13** |
| TITLE: | **Parallelized Decipher Engine** |
| DESCRIPTION: | There are two main objectives to this project: developing an understanding of some of the math in data encryption, and implementing this on a parallel processing device. The plan is that multiple implementations will be devised using the same general decryption algorithm, but using different parallelization techniques. Suitable performance metrics need to be established, these include the obvious one of speed, but also aspects such as resource and power use. The top-level design, and using a good modular approach for portability, is essential for this project in order to facilitate porting the algorithm to different platforms and getting performance testing done quickly. The proposed platforms are: single-processor PC (i.e. for a baseline measure); multi-processor PC (threaded version); with a CUDA or FPGA highly parallelized version. |
| DELIVERABLES: | A variety of prototyped systems to decrypt a block of encrypted data, performance testing and comparisons. |
| SKILLS/REQUIREMENTS: | Programming |
| EXTRA INFORMATION: | |
| AREA: | Computer Engineering |

**ADDENDUM 2:** To be completed if you answered YES to Question 2:

It is assumed that you have read the UCT Code for Research involving Human Subjects (available at http://web.uct.ac.za/depts/educate/download/uctcodeforresearchinvolvinghumansubjects.pdf) in order to be able to answer the questions in this addendum.

| | | |
|---|---|---|
| 2.1 Does the research discriminate against participation by individuals, or differentiate between participants, on the grounds of gender, race or ethnic group, age range, religion, income, handicap, illness or any similar classification? | YES | NO |
| 2.2 Does the research require the participation of socially or physically vulnerable people (children, aged, disabled, etc) or legally restricted groups? | YES | NO |
| 2.3 Will you not be able to secure the informed consent of all participants in the research?<br>(In the case of children, will you not be able to obtain the consent of their guardians or parents?) | YES | NO |
| 2.4 Will any confidential data be collected or will identifiable records of individuals be kept? | YES | NO |
| 2.5 In reporting on this research is there any possibility that you will not be able to keep the identities of the individuals involved anonymous? | YES | NO |
| 2.6 Are there any foreseeable risks of physical, psychological or social harm to participants that might occur in the course of the research? | YES | NO |
| 2.7 Does the research include making payments or giving gifts to any participants? | YES | NO |

If you have answered YES to any of these questions, please describe below how you plan to address these issues:

**ADDENDUM 3:** To be completed if you answered YES to Question 3:

| | | |
|---|---|---|
| 3.1 Is the community expected to make decisions for, during or based on the research? | YES | NO |
| 3.2 At the end of the research will any economic or social process be terminated or left unsupported, or equipment or facilities used in the research be recovered from the participants or community? | YES | NO |
| 3.3 Will any service be provided at a level below the generally accepted standards? | YES | NO |

If you have answered YES to any of these questions, please describe below how you plan to address these issues:

**ADDENDUM 4:** To be completed if you answered YES to Question 4

| | | |
|---|---|---|
| 4.1 Is there any existing or potential conflict of interest between a research sponsor, academic supervisor, other researchers or participants? | YES | NO |
| 4.2 Will information that reveals the identity of participants be supplied to a research sponsor, other than with the permission of the individuals? | YES | NO |
| 4.3 Does the proposed research potentially conflict with the research of any other individual or group within the University? | YES | NO |

If you have answered YES to any of these questions, please describe below how you plan to address these issues: