AN AES CORE IMPLEMENTATION FOR SECURED-DATA PORTABLE HARD DISK

KHANOB THONGKHOME

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN DATA STORAGE TECHNOLOGY

ENGLISH PROGRAM

INTERNATIONAL COLLEGE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2012

KMITL-2012-IC-M-005-007

หัวข้อวิทยานิพนธ์	การเข้าและถอดรหัสข้อมูลแบบเออีเอสสำหรับการรักษาความปลอดภัยของข้อมูลในฮาร์ดดิสก์แบบพกพา
นักศึกษา	นายขนบ ทองคำ
รหัสประจำตัว	52600601
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
ภาควิชา	เทคโนโลยีการบันทึกข้อมูล
พ.ศ.	2555
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร.สมศักดิ์ ชุมช่วย

บทคัดย่อ

งานวิจัยนี้ได้นำเสนอวิธีการออกแบบระบบสถาปัตยกรรมของการเข้าและถอดรหัสข้อมูลตามมาตรฐานเออีเอส เพื่อใช้ในการรักษาความปลอดภัยของข้อมูลบนฮาร์ดดิสก์แบบพกพา ด้วยการใช้ชิพวงจรรวมเอฟพีจีเอในการตรวจสอบความถูกต้องของการเข้าและถอดรหัส โดยการออกแบบจะพิจารณาถึงความสามารถของระบบทั้งด้านความเร็ว ขนาดของฮาร์ดแวร์ และอัตราการสิ้นเปลืองพลังงาน สำหรับรองรับความเร็วการทำงานในระดับ 5 กิกะบิตต่อวินาทีของระบบยูเอสบี 3.0. ซึ่งมาตรฐานเออีเอสที่ใช้ในงานวิจัยนี้ มีช่องทางการส่งผ่านกุญแจรหัสและข้อมูลที่ใช้ในการเข้าและถอดรหัส ขนาด 128 บิต ด้วยสถาปัตยกรรมสองรูปแบบ คือ ระบบเออีเอสมาตรฐานแบบวนซ้ำและระบบไปป์ไลน์ย่อยแบบวนซ้ำ เพื่อเลือกระบบสถาปัตยกรรมเออีเอสที่เหมาะสมและรองรับการทำงานของฮาร์ดดิสก์แบบพกพาได้ ผลจากการวิจัยพบว่าระบบเออีเอสมาตรฐานแบบวนซ้ำจะรองรับการส่งผ่านข้อมูลด้วยความเร็วสูงสุดที่ 5.2 กิกะบิตต่อวินาทีที่ความถี่ 340 เมกะเฮิร์ต โดยใช้ความจุของวงจรทั้งสิ้น 2969 ลอจิกบล็อค คิดเป็น 48 เปอร์เซ็นต์ของความจุวงจรทั้งหมดในเวอเท็กซ์โฟว์เอฟพีจีเอ ในขณะที่ระบบไปป์ไลน์ย่อยแบบวนซ้ำจะรองรับการส่งผ่านข้อมูลด้วยความเร็วสูงสุดที่ 8.3 กิกะบิตต่อวินาทีที่ความถี่ 545 เมกะเฮิร์ต โดยใช้ความจุของวงจรทั้งสิ้น 5641 ลอจิกบล็อค คิดเป็น 92 เปอร์เซ็นต์ของความจุวงจรทั้งหมดในเวอเท็กซ์โฟว์เอฟพีจีเอ โดยมีอัตราการสิ้นเปลืองพลังงานอยู่ที่ 10% - 20% ของพลังงานสูงสุดที่ใช้ในระบบยูเอสบี 3.0.

Thesis	An AES core implementation for secured-data portable hard disk
Student	Mr. Khanob Thongkhome
Student ID.	52600601
Degree	Master of Engineering
Program	Data Storage Technology
Year	2012
Thesis Advisor	Assoc.Prof.Dr.Somsak Choomchuay

ABSTRACT

This thesis presents an AES core architecture design for a portable hard disk drive data securing. The design has been verified for its encryption and decryption functionality with FPGA implementation. The design also has been optimized for speed, hardware size and power consumption to meet with minimum speed of 5 Gbps required by USB3.0 data communication port. The AES with 128 bits key size and data path width has been proposed with two possible alternatives; Basic iterative AES and One stage sub pipelined AES. The effort is to investigate the promised architecture that matches well the targeted and application. As a result, FPGA with the basic iterative AES encryption offers 5.2 Gbps throughput at 340 MHz clock speed. It occupies 2969 CLB slices (48% of overall CLB slices of Xilinx Virtex-4). The one stage sub pipelined AES offers 8.3 Gbps throughput at 545 MHz clock speed. It occupies 5641 CLB slices (92% of overall CLB slices of Xilinx Virtex-4). This architecture consumes 10% - 20% of the maximum power delivered by USB3.0 port.

Acknowledgements

First of all, I would like to thank you Associate Professor Dr. Somsak Choomchuay, my advisor for his helpful suggestions and constant supports during this research at KMITL. I am also thankful to all thesis committee members for their constructive comments and helpful discussions which gave me a better perspective on my own result. I should also mention that my graduate study in King Mongkut's Institute of Technology Ladkrabang was supported by NSTDA, KMITL and Seagate Technology (Thailand) Ltd.

And finally, I would like to acknowledge the support of all of my family and my friends for all of their help.

Khanob Thongkhome

Samutprakarn, Thailand

Table of Contents

Table of Contents (Cont.)

List of Tables

List of Figures

List of Figures (Cont.)

Acronyms, Definitions and Symbols

This thesis document uses the following set of Acronyms, Definitions and Symbols

AES	Advanced Encryption Standard
Affine Transformation	A mathematical operation of multiplication by a matrix followed by the addition of a vector
Array	A group of similar elements
Bit	A binary value consisting of 1 or 0
Block	Sequence of bits or array of bytes whose length is determined by the number of bits or bytes
Byte	An array of 8 bits
Cipher	Set of well defined steps to transform data from plain text to cipher text
Cipher Key	The secret key which is the password to encrypt the data. It also generates the set of keys for different iterations.
Ciphertext	The result of a cipher or input to inverse cipher
GF (p)	Finite field or Galois field over p which contains all the elements from 0 to p-1
Gbps	Gigabit per second

FPGA	Field programmable gate array
Plain text	Input to the cipher
Rijndael	Cryptographic algorithm specified in the AES
Round Key	The Key generated from Key Expansion
State	Intermediate results of a cipher which are stored as a block which is an array of bytes or bits
S-box	A non-linear substitution table used for byte substitution in the Cipher and Key Expansion
Nb	Number of columns consisting of 32 bits which comprises a State. Nb = 4 of this document
Nk	Number of columns consisting of 32 bits of Cipher Key. Nk = 4 bits of this document as this deals with only 128 bits key
Nr	Number of rounds which are 10 of this document
XOR	Exclusive or operations
$\oplus$	Exclusive or operations
$\otimes$	Multiplication of two polynomials (each with degree < 4) modulo $x^4 + 1$
$\bullet$	Finite field multiplication

CHAPTER 1

Introduction

1.1. Problem statement

The security of data in data storage devices for example hard disk drives are becoming one of the various main issues in computer security. The important main threat against is theft or loss due to their small size. An unauthorized access of confidential data evokes the organization to be risky. A recent computer security survey shows that two thirds of IT staffs who use a notebook or storage devices at work did not protect themselves with encryption [1]. What is an encryption?. The Encryption is the information converting process into an encrypted form. It is perceptible only to somebody who knows how to decrypt it to be the original message. It is generally used in the connection with electronic data, neither stored on a computer or transmitted over the Internet. The data encryption tools (usually in the form of computer software) are widely used to secure the data. Unfortunately, the software based encryption should be run under the operating system and on the CPU. So, it should be impacted with the overall performance of the PC and exposure to the security methods used to protect the information on the PC itself. The attacker can be stealth PC running processes which able to capture the encryption keys and even the non-encrypted data that is not a very good scenario to have [2]. The hardware based full disk encryption (FDE) is obtainable from many hard disk drive (HDD) manufacturing, including Seagate Technology, Western Digital and also solid-state drive manufacturing such as Samsung. The encryption key of FDE drive is stored independently from the CPU, to avoid removing computer memory that is a potential attack vector. The FDE drive encrypts the data in every write operation and decrypts it back to the original data in every read operation without user interference. The encryption and decryption process is done on the drive itself and near zero performance impact when the drive operates. The purpose of FDE

drive is to protect the data that stored in the device even if the host system is lost or stolen. This residue data from protection assures the system owner that their data will not be accessible without the correct authentication. All of them are used AES (Advanced Encryption Standard) encryption algorithm that was qualified by NIST (National Institute of Standard and Technology). AES is a symmetric block cipher that can process data block of 128 bits, using cipher keys with lengths of 128, 192 and 256 bits (depend on HDD manufacturing).

The disadvantage of Hardware-based full disk encryption system. The user will buy a new one for the FDE HDD unit to protect their critical data due to the current HDD of user was not supported encrypt/decrypt properties. And a security processor was identically fixed and didn't flexibly support another HDD unit. Thus, more questions around the market for the encryption properties of HDD. Can we secure them with hardware -based full disk encryption?. Are more energy should be focused on solution-level deployment issue when put encryption engine to the rest?. How is it operation speed?., etc. To answer that question, It's necessary to define "better" on given that we are deeply studying, focusing on technical details about FDE to establish data encryption/decryption core prototype for secured-data of current hard disk drive system or a portable hard disk system supporting and finding the way to improve some weakness of FDE hard disk drive to meet customer requirement.

1.2. Objective

To design AES core hardware based data encryption/decryption with FPGA chip for secured-data portable hard disk to support the new data communication technology or new hard disk interface technology.

1.3. Research Benefit

1.3.1 To obtain the hardware based data encryption/decryption prototype core for AES cryptography research development.

1.3.2 To obtain the AES core system that suitable for implementing with the secured data portable hard disk.

1.4. Research Methodology

1.4.1 To study the advance encryption standard algorithm.

1.4.2 To study and design advance encryption standard hardware based architecture that suitable for secured data on a portable hard disk.

1.4.3 To study VHDL language and its programming algorithm.

1.4.4 To validate designed AES architecture in each operation step on the FPGA chip with VHDL language.

1.4.5 To validate a whole system designed AES architecture of the FPGA chip with VHDL language.

1.4.6 Research conclusion.

1.5. Scope of Research

1.5.1. Apply the advanced encryption standard algorithm for hardware based system architecture design.

1.5.2. The system's core architecture could be operated and obtained high data rate with offered several techniques.

1.5.3. To Use VHDL language for circuit designing implementation.

1.6. Thesis Organization

This thesis is organized into six major parts in the following manner. Chapter 2 presents concise introduction of advanced encryption standard algorithm to each basic AES operation block with FPGA hardware implementation and background discusses related research and concepts that contribute to this thesis for example data storage disk encryption overview, block cipher operation mode of AES, portable or external hard disk interface, cryptanalysis attack, etc. As one of the major contribute, the proposing hardware architecture in each sub-operation of AES algorithm is presented in Chapter 3 to design suitable AES hardware architecture for supporting high data rate issue and interfacing unit to support portable hard disk properties with USB and SATA cores. Then, Chapter 4 represents the result of FPGA

implementation of proposing hardware architecture of each AES basic operation block and how to optimize the data path of the whole system to improve the overall throughput requirement. Finally, Chapter 5 concludes this research work by providing the conclusion and some important issue for future work.

CHAPTER 2

AES Implementation with FPGA for Data Storage Disk Encryption

2.1. Why do we determine AES?.

Advance Encryption Standard (AES) was announced in Federal of information processing standard publication (FIPS PUB 197) that are issued by the National Institute of Standards and Technology (NIST) on November 26, 2001. It was selected as the most suitable from fifteen competing designs were presented and evaluated. After that, it was approved by the Secretary of Commerce and became a proficient cipher as a Federal government standard on May 26, 2002. AES is the first public declaration and open block cipher that was approved by the National Security Agency (NSA) for confidentially the top secret information. The original name called Rijndael that was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and was submitted to the AES selection process. It's played on from the names of the two inventors. The primary criteria used by NIST to evaluate AES candidates included security, efficiency in software and hardware, and flexibility. Due to the relatively inconclusive the cryptanalysis results of the final five candidate software performance evaluations. So, the hardware efficiency evaluations among publication paper exhibited during the 3'rd AES conference provided a very abundant quantitative measure that clearly differentiated AES candidates among each other [3, 4, 5, 6]. Dandalis et al [3] verified Rijndael algorithm achieves the best time performance across the different platform (ASIC, FPGA and software) when compare with the other AES candidate finalist. Elbert et al [4] concluded the serpent algorithm performance is the best in both non feedback and feedback modes when considering the highest throughput with another AES candidate finalist. Gaj and Chodowiec [5] found that the Twofish seems to be the most suitable for applications where the primary requirement is the limited cost or an area of the cryptographic hardware. Serpent and Rijndael both offer superior performance for applications where the speed itself is a criterion of primary concern. Ichigawa et al [6] discovered about the Twofish algorithm will be most rewarding for the efforts of optimizing the

lookup tables than another AES candidate finalist. The importance of this evaluate was considered by a survey execute among the participants of the AES conference, in which the ranking of the candidate algorithms corresponds very well with their relative speed in hardware [7, 8].

In this chapter, we will review the AES algorithm fundamental from the viewpoint of knowledge required for efficient hardware implementations.

2.2. AES Algorithm

AES is an iterated data block cipher with a fixed data block size of 128 bits and a variable key length. Each transformation step operates on the mediator results, called state. The state is a rectangular array (dimensions 4x4, which is 16 bytes) of bytes from the total block size is 128 bits. The cipher key is a rectangular array with four rows as same as the state. The number of columns of the cipher key is equal to the key length divided by 32.

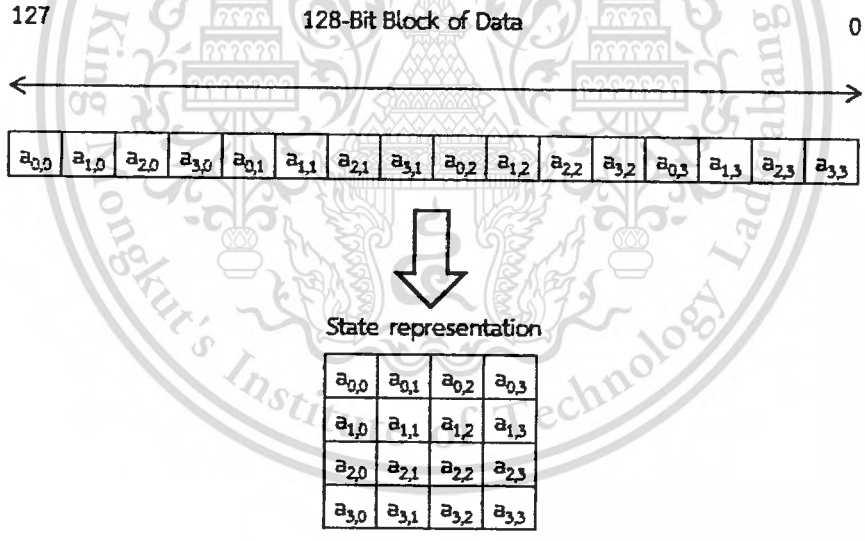


Figure 2.1 State representation of 128 bits data block

In Figure 2.1, it is significant to know that the input data bytes are converted to the state bytes array $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$ and the input cipher key are converted to the array $k_{0,0}$, $k_{1,0}$, $k_{2,0}$, $k_{3,0}$, $k_{0,1}$, $k_{1,1}$, $k_{2,1}$, $k_{3,1}$. At the last step of the

cipher operation, the cipher output is seceding from the state by ordering the state bytes. AES uses a variable number of rounds, which are fixed : key size 128, 192, 256 bits have 10, 12 and 14 rounds in respectively. During each round, the following operations apply to the state to follow the diagram that shows in Figure 2.2 and 2.3.

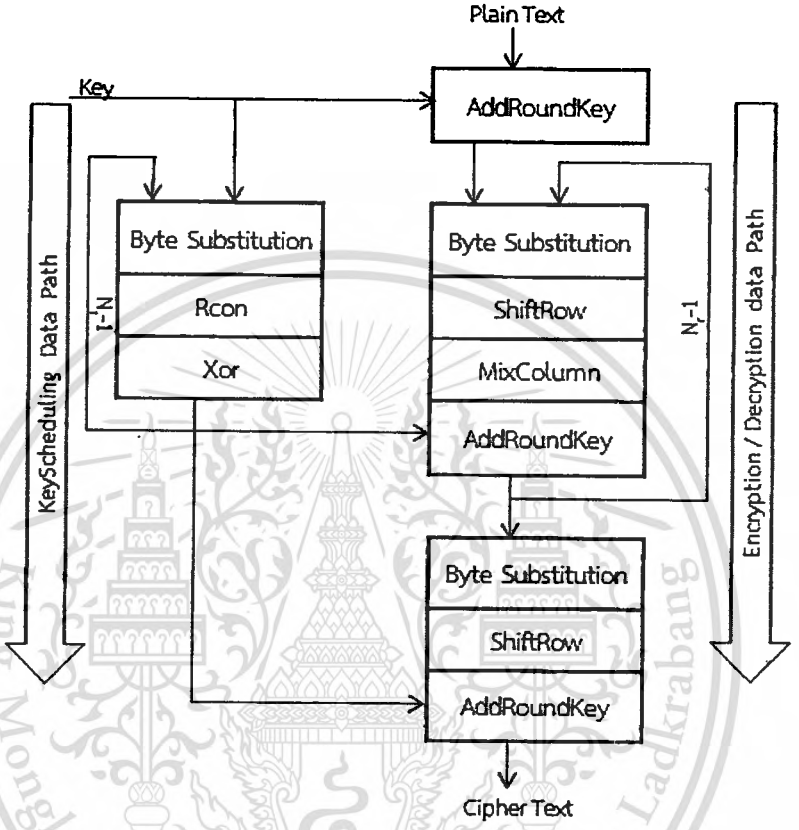


Figure 2.2 Block diagram of AES

- SubBytes: Each byte (16 bytes) in the state is replaced by another byte using the Rijndael S-Box substitution/LUT.
- ShiftRow: Each row (4 rows) in the state is shifted in a certain amount to the left direction.
- MixColumn: The Linear transformation operation on the columns (4 columns) of the state.
- AddRoundKey: Each byte of the state is performed XOR operation with a round key, which use different key of each round operation using the Rijndael key schedule.

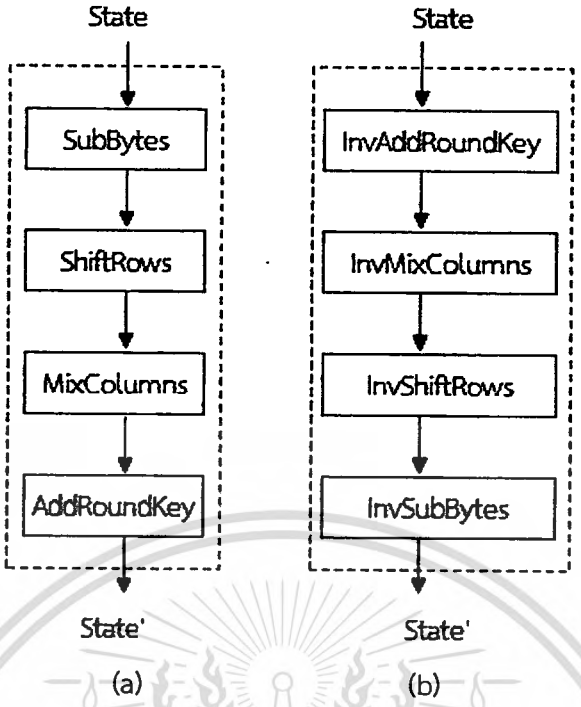


Figure 2.3 Structure of AES (a) encryption and (b) decryption round

2.2.1. The SubBytes and InvSubBytes Operation

The SubBytes operation is a byte substitution in non-linear basis and its operate independently on each byte of the state. The substitution table (S-Box) is an invertible and is constructed by the composition of two transformations process.

2.2.1.1 Take the multiplicative inverse in Galois field $GF(2^8)$ with the reduction polynomial m(x) specified by (1).

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (1)$$

2.2.1.2 Apply an affine transformation over GF(2).

Since the S-Box is independent of any input, pre-calculated forms are used, if enough memory (256 bytes for one S-Box) is available. Each byte in the array is updated using an 8 bits substitution box, the Rijndael S-box. This operation provides the nonlinearity in the cipher. The S-box used is derived from the multiplicative inverse over $GF(2^8)$, known to have a good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the

inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), and also any opposite fixed points.

$$b(_{i,j}) = S(a_{i,j}) \quad (2)$$

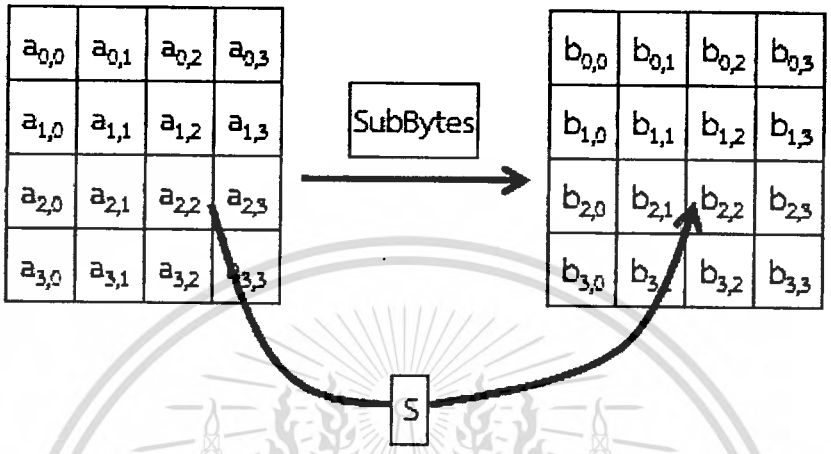


Figure 2.4 SubByte operation process

2.2.2. The ShiftRow Operation

In this operation, each row of the state is cyclically shifted to the left, depending on the row index.

- The 1st row is shifted 0 positions to the left.
- The 2nd row is shifted 1 positions to the left.
- The 3rd row is shifted 2 positions to the left.
- The 4th row is shifted 3 positions to the left.

For the block of size 128 bits and 192 bits the shifting pattern is the same. In this way, each column of the output state of the ShiftRow step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets). In the case of the 256 bits block, the first row is unchanged and the shifting for second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively - this change only applies for the Rijndael cipher when used with a 256 bits block, as AES does not use 256 bits block.

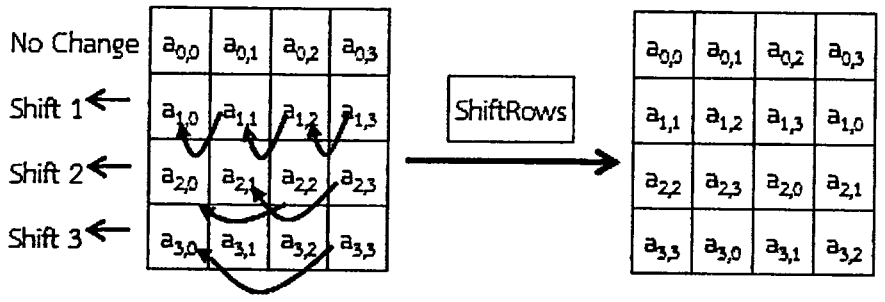


Figure 2.5 ShiftRow operation process

2.2.3 The AddRoundKey Operation

In this operation, a Round Key is applied to the state by a simple bitwise XOR. The Round Key is derived from the Cipher Key by the means of the key schedule. The Round Key length is equal to the block key length (=16 bytes).

$$b_{(i,j)} = a_{(i,j)} \text{ XOR } k_{(i,j)} \quad (3)$$

A graphical representation of this operation can be seen below.

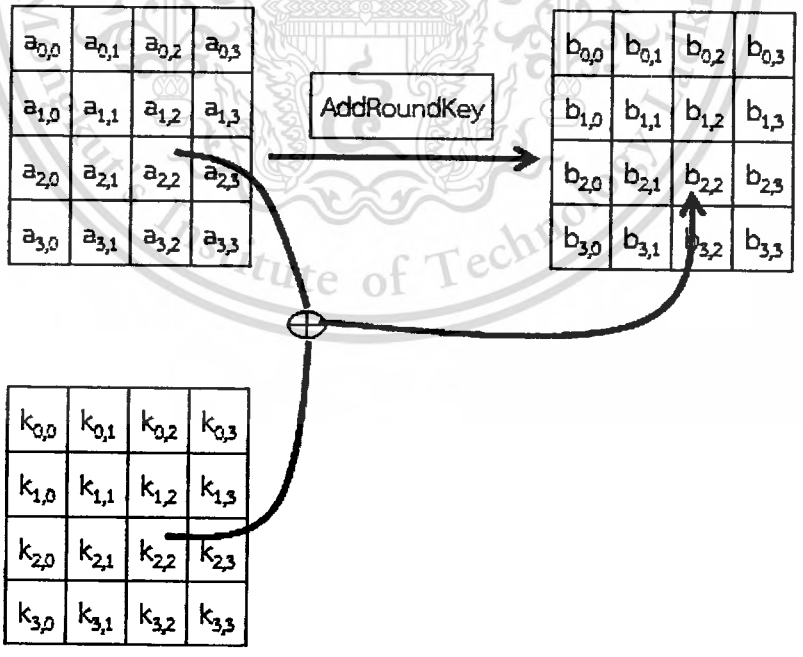


Figure 2.6 AddRoundKey operation process

2.2.4 The MixColumn Operation

In the MixColumns step, the four bytes of each column of the state are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with ShiftRows, MixColumns provides diffused in the cipher. Each column is treated as a polynomial over GF(2^8) and is then multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x) = 0x03x^3 + x^2 + x + 0x02$. (The coefficients are displayed in their hexadecimal equivalent of the binary representation of bit polynomials from GF(2)[x].) The MixColumns step can also be viewed as a multiplication by a particular MDS matrix in Finite Fields.

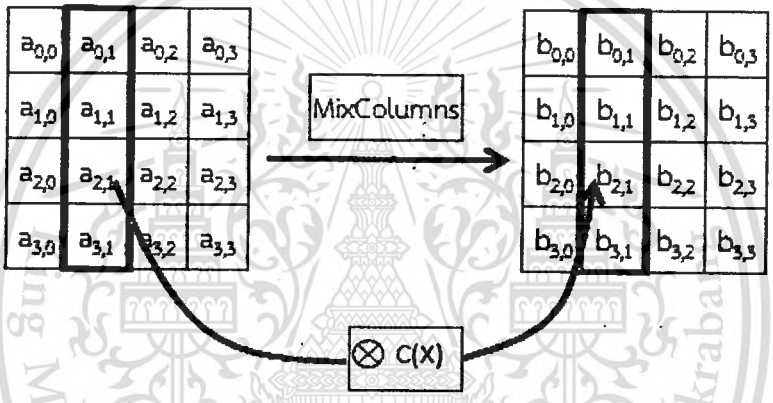


Figure 2.7 MixColumn operation process

2.3. FPGA and ASIC Technology

Cryptographic transformations can be implemented in both software and hardware. Software implementations are designed and coded in programming languages, such as C, C++, Java, and assembly language to be executed among others on general purpose microprocessors, digital signal processors, and smart cards. Hardware implementations are designed and coded in hardware description languages such as VHDL and Verilog HDL that are intended to be realized using two major implementation approaches: application-specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs).

Application-specific integrated circuits (ASICs) are designed all the way from the behavioral description to the physical layout and then sent for a fabrication in a

semiconductor foundry. Field programmable gate array (FPGA) can be bought off the shelf and reconfigured by designers themselves. With each reconfiguration, which takes only a fraction of a second, an integrated circuit can perform a completely different function. FPGA consists of thousands of universal reconfigurable logic blocks, connected using reconfigurable interconnects and switches, as shown in Figure 2.8. Additionally, modern FPGAs contain embedded higher-level components, such as memory blocks, multipliers, multipliers–accumulators, and even microprocessor cores. Reconfigurable input/output blocks provide a flexible interface with the outside world. Reconfiguration, which typically lasts only a fraction of a second, can change a function of each building block and interconnects between them, leading to a functionally new digital circuit.

In Table 2.1, we collect and contrast features of implementations of cryptographic transformations based on ASICs and FPGAs and microprocessors. The performance characteristics of ASICs and FPGAs are almost identical, as demonstrated by the first group of features, and substantially different from the performance characteristics of general purpose microprocessors. Both ASICs and FPGAs can make a full use of parallel processing and pipelining and operate on arbitrary size words. General purpose microprocessors, parallel processing and pipelining are limited by the number and internal structure of the processor functional units and by the instruction level parallelism. Additionally, all functional units operate on the fixed-size arguments only.

The primary difference between ASICs and FPGAs in terms of the performance characteristics is a smaller speed of FPGAs caused by the delays introduced by the circuitry required for reconfiguration. As a result of this speed penalty, any digital circuit implemented in an FPGA is typically slower than the same circuit implemented in an ASIC, assuming that both integrated circuits are fabricated using the same semiconductor technology (in particular, using the same transistor size).

The most modern FPGAs have high level embedded modules or example distributed RAM block, built-in multiplier ad so on is another important difference with ASICs. Moreover, in contrast to ASICs devices, it supports in-system reconfiguration and allows the design to be changed dynamically that useful fry stem updates.

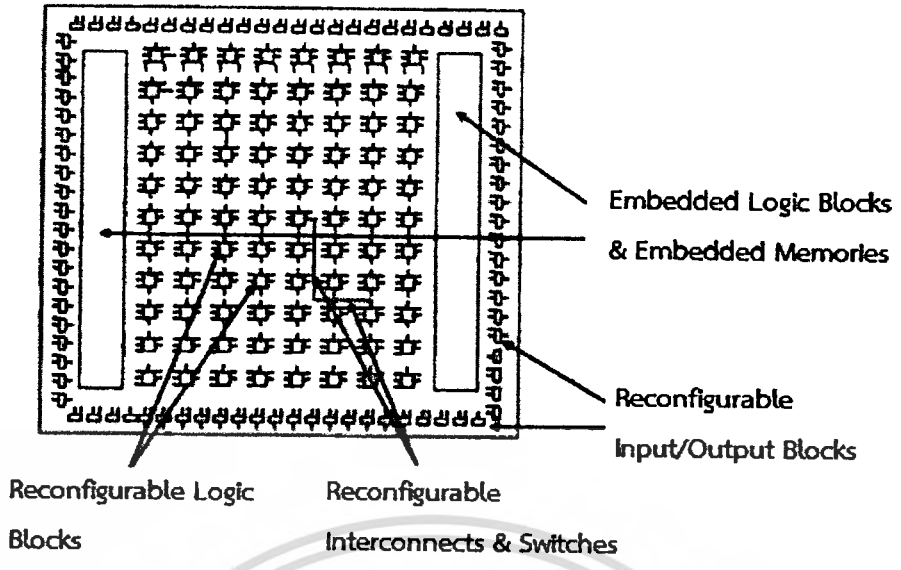


Figure 2.8 General structure and main building blocks of FPGA

Table 2.1 Characteristic features of implementations of cryptographic transformations in ASICs, FPGAs, and microprocessors.

	ASICs	FPGAs	Microprocessors
	Performance Characteristics		
Parallel Processing	Yes	Yes	Limnited
Pipelining	Yes	Yes	Limnited
Word Size	Variable	Variable	Fixed
Speed	Very Fast	Fast	Moderately Faste
	Development Process		
Description Language	VHDL, Verilog	VHDL, Verilog	C, C++, Java, Assembly
Design Cycle	Long	Moderately Long	Short
Design Tool	Very Expensive	Moderate Expensive	Inexpensive
Maintenance and Upgrade	Expensive	Inexpensive	Inexpensive
	Functionality		
Algorithm Agility	Yes	Yes	Yes
Tamper Resistance	Strong	Limited	Weak
Access Control to Key	Strong	Moderate	Weak

The recent study performed at the University of Toronto [9] quantified the performance differences between the current generation of ASICs and FPGAs. A set of 23 benchmarks covering applications in the area of cryptography, digital signal processing, and communications were included in the study. The study concluded that the circuits containing only combinational logic and flip-flops, the ratio of silicon area required to implement them in FPGAs and ASICs is on average 40. For circuits that could take advantage of dedicated blocks present in modern FPGAs, such as multiplier/accumulators and block memories, these blocks reduced the average area gap significantly in as little as 21. The ratio of critical path delay, from FPGA to ASIC, was found to be roughly 3–4, with less influence from embedded memories and embedded logic blocks. The dynamic power consumption ratio was approximately 12 times and, with hard blocks, this gap generally became smaller.

The common features of FPGAs and microprocessors concern most functionality and do not affect performance. Both general purpose microprocessors and FPGAs can be easily reconfigured in real time to perform a different algorithm. The disadvantage of this feature is a limited tamper resistance; the contents of an FPGA can be, at least in theory, modified by an unauthorized user. In practice, the contents of an FPGA are typically downloaded during the initialization from the read-only memory, such as EPROM, which cannot be easily tampered with, at least remotely. The access control of cryptographic keys in FPGAs is also stronger than in software, but weaker than in ASICs. The development process in both hardware implementation approaches is very similar. In both FPGAs and ASICs, the circuit is described using a hardware description language, verified using a digital circuit simulator, and then tested using a prototyping board. The primary difference between FPGAs and ASICs is that FPGA does not require the physical design (layout), fabrication, and testing for physical defects. As a result, the design cycle is significantly shorter and the design tools and testing much less expensive. The interesting similarity between FPGAs and software is a possibility of remote maintenance and upgrading, based on electronic patches.

2.4. Parameter of Hardware Implementation

Hardware implementations of AES data block ciphers can be characterized using several performance parameters. Below we provide our definitions of major

parameters and derive formulas that demonstrate mutual dependencies among these parameters.

2.4.1 Throughput and Latency

Encryption throughput is defined as the number of bits that were encrypted in a unit of time. Typically, the encryption and decryption throughputs are equal, and therefore only one parameter is reported. A typical unit of throughput is Mbps (megabit per second) or Gbps (gigabit per second). It is worth mentioning that 1 Mbps = 10^6 bit/s, and not 2^{20} bit/s, and 1 Gbps = 10^9 bit/s, and not 2^{30} bit/s.

Encryption (decryption) latency is defined as the time necessary to encrypt (decrypt) a single block of plaintext (ciphertext). The typical unit of latency in the current technology is NS (nanosecond). The encryption (or decryption) latency and throughput are related by the block size multiply by number of blocks processed simultaneously and then divide by latency.

In applications where large amounts of data are encrypted or decrypted, throughput determines the total encryption/decryption time and thus is the best measure of the cipher speed. In applications where a small number of plaintext (ciphertext) blocks are processed, the total encryption/decryption time depends on both throughput and latency.

2.4.2 Area

The area required for AES implementation is an important parameter for the following reasons:

- Cost

The area of an integrated circuit is a primary factor determining its cost. It is traditionally assumed that the cost of an integrated circuit is directly proportional to the circuit area. This dependence is not always accurate, especially taking into account the cost of a package, which is determined by the number of the circuit inputs and outputs.

- Limit on the maximum area

In certain hardware environments, there exists a limit on the maximum area of a cryptographic unit. This limit may be imposed by the cost, available fabrication technology, power consumption, or any combination of these factors. For example,

smart cards and microcontrollers. Both of cost and power consumption limit the area of the embedded encryption units; in FPGAs, the area is limited by the available fabrication technology and the cost of a programmable device.

In FPGA implementations, the only circuit size measures reported by the CAD tools are the number of basic configurable logic blocks and the number of equivalent logic gates. It is commonly believed that out of these two measures, the number of basic configurable logic blocks approximates the circuit area more accurately. Measurement and comparing circuit area in FPGAs are additionally complicated by the existence of embedded logic blocks and embedded memories. The specifications of FPGA devices typically do not provide any information about the relative ratio of the areas used by embedding blocks and basic reconfigurable logic blocks [10].

2.5. Data Storage Disk Encryption Overview

The data storage disk encryption can be categorized into two different methods. The first one is a software based encryption and the rest is hardware based encryption (can be sub-categorized into controller based and internal disk encryption also). The software based was add-on security product or software that modifies the hard drive drivers and encrypts all data as it is written to the drive. It requires the correct password before the data is decrypted and offers good protection but some system is expensive to purchase and deploy and impacts system performance which sometimes leads end users to turn it off. There is a potential for malware, trojans or rootkits to remotely turn off the software protection (the same as end users) without proper methods of protecting the software itself from attacks. Also worth noting, some software-based products require the encryption to be turned off whenever an operating system update must be installed—causing an administration burden and also risk of exposure. The hardware based encryption was built-in cryptographic hardware that encrypts all data as it is written to the drive. It requires the correct password to decrypt any data and does not impact performance. The system is extremely difficult to defeat when good passwords are used to offer excellent protection. The controller-based encryption has many advantages compared to software based encryption. It performs by an external hardware module or chip separately from the storage device as shown in

Figure 2.9. This property improves the overall performance and speed of the encryption process as a dedicated encryption controller or processor is used. The cryptographic operations are handled internally by the controller and the whole operation is not exposed. The cryptographic keys are stored inside the hardware and makes them bound to the underlying platform. Although controller based encryption is more secure than software based encryption, it still suffers from many limitations such as traffic-analysis attack since cryptographic controller and disk storage device are separated, the ciphertext is still accessible which may reveal important information to cryptanalysis. Moreover, it generally suffers from the additional cost and the inability to patch and upgrade products due to special additional hardware. The internal disk encryption (or Discryption) is the most reliable and efficient method. The cryptographic operations in a closed environment with restricted I/O and secured firmware. The encryption process was embedded inside the storage device itself as shown in Figure 2.10. The low power and high performance encryption at full storage device interface speed can be achieved using a dedicated encryption processor. As the encryption process is performed internally in the storage devices, security threats are virtually eliminated. In case of device theft, encrypted data and key are better secured deep inside the hardware. The sealed encrypted storage device would prevent bus or traffic-analysis attack which is a critical limitation of controller based encryption system [11]. This research plan to develop an AES core hardware architecture to support controller based hardware encryption to secure data of current hard disk that the users do not need to buy the new one of FDE hard disk.

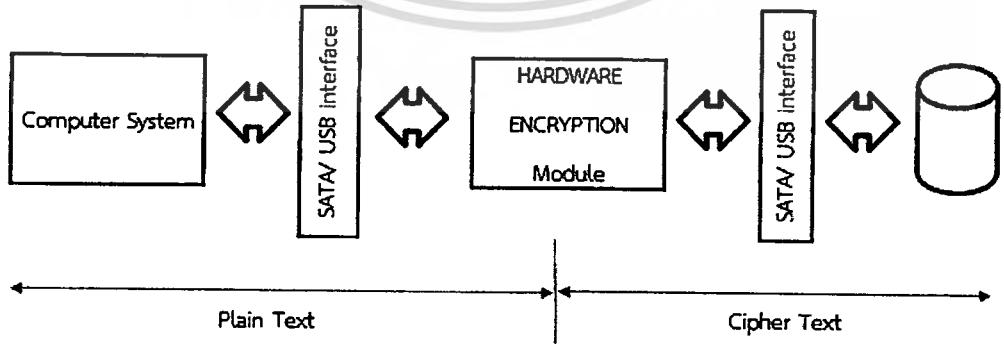


Figure 2.9 Controller based hardware encryption

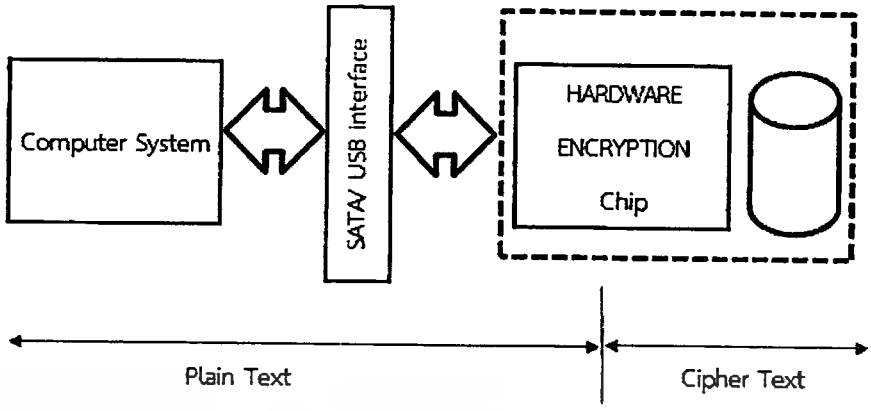


Figure 2.10 Internal hardware disk encryption

2.6. Block Cipher Operation Mode of AES

AES block cipher operation mode is an algorithm that characterizes the use of an AES algorithm to provide an information service, for example confidentiality or authentication. Currently, NIST is endorsing the nine modes of block cipher operation. The six confidentiality modes are ECB (Electronic Codebook), CBC (Cipher Block Chaining), OFB(Output Feedback), CFB(Cipher feedback), CTR(Counter mode), and XTS-AES(XEX-based Tweaked CodeBook (TCB) with CipherText Stealing (CTS), XEX is Xor encrypt Xor). The one authentication mode is CMAC(cipher-based MAC). And the two combined modes for confidentiality and authentication are CCM (Counter with Cipher Block Chaining-Message Authentication code) and GCM(Galois/Counter mode). From the viewpoint of AES hardware implementation, these confidentiality modes can be classified into two major groups. The first group is a Non-feedback mode(ECB and CTR).The next is Feedback mode(CBC, CFB and OFB).

In the non-feedback modes, all AES subsequent blocks can be encrypted the data in parallel and independently operate. But for the feedback modes, it should not to start encrypting the next block of data until the previous block encryption process is completed. So, all blocks must be respectively encrypted, with no have the capability for performing parallel processing. Using feedback modes did not serve to fully utilize the performance of secret key cipher hardware implementations, based on parallel processing of multiple blocks of data. The problem has been partially treated by using a CTR per the NIST recommendations.

ECB is the most uncomplicated block cipher mode due to it uses the same key to operate in each block of data and the input of this mode AES system is the current plaintext. CBC mode should XOR the first block of plain text and initialize a vector(IV) for the first time input of AES system and then XOR the current plaintext and preceding ciphertext to be the continuously next time input. The system should generate different ciphertext from the same plaintext. CFB mode uses AES as a stream cipher due to the input of this mode AES system is the preceding ciphertext and the ciphertext is the XOR between plaintext and the output of the AES system. OFB is similar CFB except the input of this mode AES system should be preceding output of the AES system. CTR uses counter to be input of this mode AES system. The counter should be initialized as the IV and increase for each block. The ciphertext is the XOR between plaintext and output AES system. The Figure 2.11 and Figure 2.12 show the block diagram of the ECB and CBC modes that the author select to be confidentiality mode implementation in this thesis.

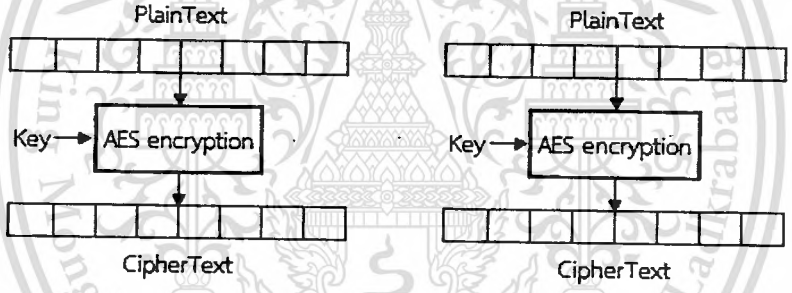


Figure 2.11 ECB mode block diagram

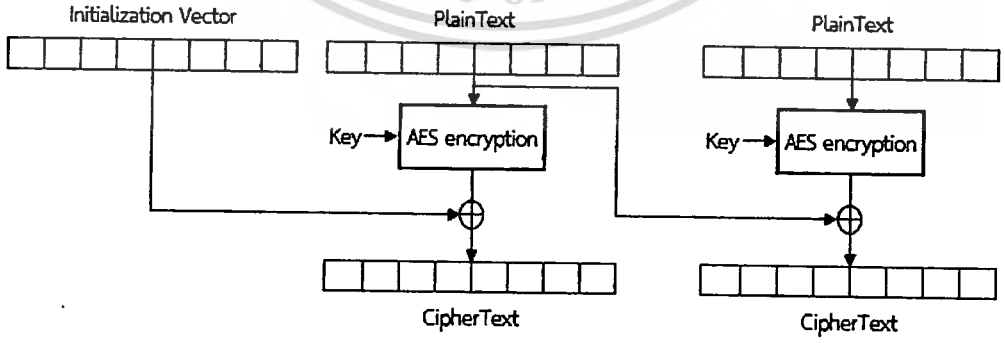


Figure 2.12 CBC mode block diagram

2.7. Portable/External Hard Disk Interface

The portable/external hard disk interfaces are mainly distinguished into 2 parts. The first one is device/hard disk to computer system interfacing for example USB, FireWire and eSATA, etc. These kinds of interface technique support the users to connect their portable/external hard disk with their PC. The USB, Universal Serial Bus, is the most popular industry standard that was designed to standardize the connection of all computer peripheral connectivity. The last revision of the USB is USB3.0 with maximum data transmission speed up to 5 Gbps and low power consumption due to the new "Superspeed" bus of USB3.0 works in a full duplex mode. FireWire is a serial bus interface for high speed data communication with 3.2 Gbps based on the last design revision IEEEP1394b. But it is not widely used because of it has expensive hardware than USB in the low end market computer peripheral. The eSATA, external Serial ATA, is designed to improve SATA (Serial ATA) that an internal PC interfacing technology by utilizing the shielded cable to connect internal SATA to outside the PC. The second parts of portable/external hard disk interfaces is USB to IDE/SATA hard disk interfacing. The IDE interface, Integrated Drive Electronics, is generally known as ATA or Parallel ATA that is a standard interface technique of IBM compatible hard disk but the use of this interface declined after the introduction of SATA since 2003. The SATA interfaces were designed to replace ISE/ATA and offering many advantages over than the old one. For example It has faster data transmission via over 2 pairs of conductor serial cable than ATA that used 16 bits wide data bus So It should be reduced cable size and cost, native hot swapping, etc. Currently, SATA has widely used in hard disk interface due to its maximum data transfer speed more over than other interfaces in the market. The last revision of SATA is SATA III with maximum data transmission speed up to 6 Gbps. For comparison among mainly used hard disk interface, we have concluded the in Table 2.2 as below.

There are 4 layers in SATA protocol architecture, i.e., Application layer, Transport layer, Link layer and Physical layer. The application layer is used for ATA command execution that include command block register access. The transport layer uses for laying control information and data to transfer between host and device in a package or frame that known as a Frame Information Structure (FIS). The link layer uses for taking data from FIS, encoding or decoding each byte of data using 8b/10b

and inserting control character such that the 10 bits stream of data should be correctly decoded. The physical layer uses for transmitting and receiving the encoded information as a serial data stream on the wire. The SATA communication layers block diagram was shown in Figure 2.13.

Table 2.2 Standard interfacing technique comparison of hard disk

Standard Interface	Raw Bandwidth (Mbps)	Transfer Speed (Mbps)	Max Cable length (m)	Power provided (W, V)
USB 2.0	480	60	5	2.5, 5
USB 3.0	5000	400	3	4.5, 5
SATA II	3000	300	1	- , -
SATA III	6000	600	1	- , -
eSATA, eSATAp	3000	300	2	- , 5
FireWire (IEEE 1394b)	3200	400	40	- , -

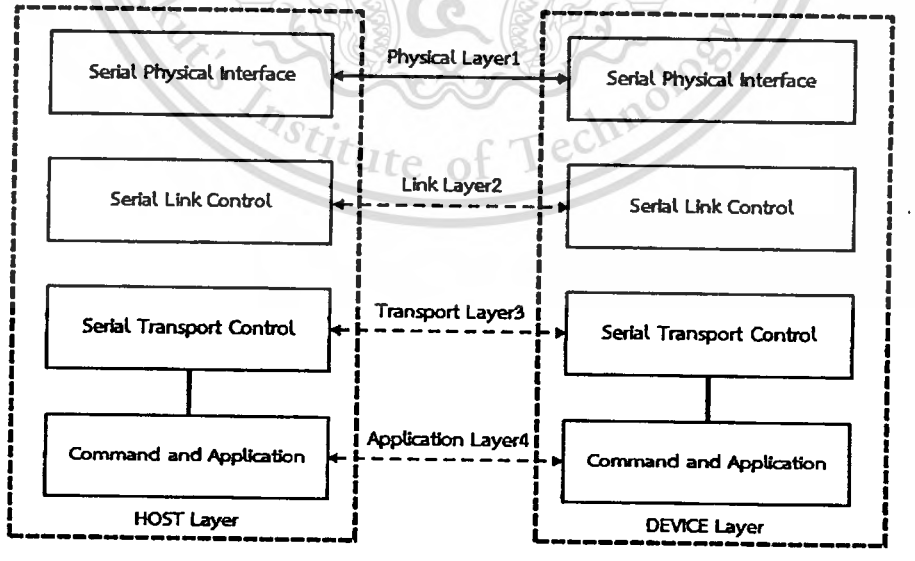


Figure 2.13 SATA communication layer

So, the designed AES system implementation for portable/external hard disk seem to be the bridge that capable of connecting the host computer over the connected SATA hard disk as shown in Figure 2.14. The AES module is placed between USB core and SATA link layer to transfer control information and data (FIS) and should be designed with the handshaking signal, is automated process of transaction on the communication channel to establish between two devices before normal data transfer step over the channel begins, to synchronize these communication devices.

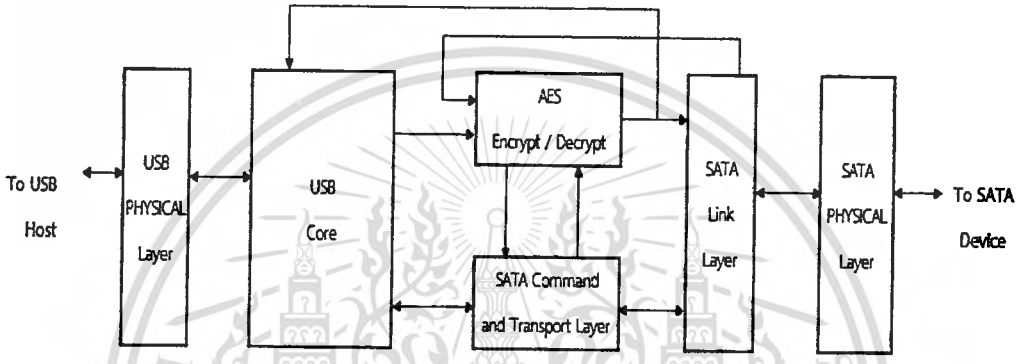


Figure 2.14 Secured portable hard disk with AES and SATA interfacing

2.8. Cryptanalysis Attack

Cryptanalysis mentions to the cultivation of ciphertexts to find the weaknesses in them that will allow retrieval of the plaintext from the ciphertext, without essentially knowing the cipher key or the encryption algorithm. It refers to finding a fault property in the design or implementation of the ciphertext that reduces the number of keys required in a foolish force attack that try to find every possible key until the correct one is found. For example, assume that a symmetric cipher implementation uses a key length of 128 bits. This means that a foolish force attack would need to try up to all 2^{128} possible combination rounds to be certain of finding the correct key to convert the ciphertext into plaintext, which is not possible given present and near future computing abilities. There are several techniques to perform cryptanalysis. It depends on what access the cryptanalyst has to the plaintext, ciphertext, or other aspects of the cryptosystem. The list below are some example of the most general types of the attacks.

2.8.1. Known-plaintext analysis : With this method, the attacker well known of a portion of the plaintext from the ciphertext. By using this information, the cryptanalyst intends to deduce the key used to produce the ciphertext.

2.8.2. Differential cryptanalysis (Chosen-plaintext analysis) : The attacker able to have any plaintext that was encrypted with a cipher key and obtain the resulting ciphertext. The cryptanalyst intends to deduce the key by comparing the entire ciphertext with the original plaintext. The Rivest-Shamir-Adleman encryption technique has been presented to be somewhat breakable to this type of cryptanalysis.

2.8.3. Ciphertext-only analysis : The attacker hasn't any knowledge of the plaintext and should be working only on the ciphertext. It requires an accurate guesswork to know how a message could be worded.

2.8.4. Man-in-the-middle attack : It differs from the above analysis in involves tricking individuals into surrendering their keys. The attacker places themselves in the communication channel between two parties who wish to exchange their keys for secure communication via public key infrastructure cryptography. Then the attacker performs a key exchange with each party, with the original parties believing they are exchanging keys with each other. The two parties then end up using keys that are known to the attacker. This type of attack can be defeated by the use of a hash function.

2.8.5. Timing/differential power analysis : This technique was public in 1998 and developed by Paul Kocher of Cryptography Research that measures differences in power consumption over a period of time when the system performs a function to secured-information. It can be used for conquest an information about the key computations that used in the encryption algorithm and other functions concerned with security. The technique can be rendered less effective by introducing random noise into the computations, or altering the sequence of the executable to make it harder to monitor the power fluctuations.

Beyond the above techniques, other techniques are available, for example convincing individuals to reveal passwords/keys, develop Trojan horse programs that steal a victim's secret key from their computer and send it back to the attacker. The AES IP core of this thesis should be helping the user to protect their secure data

from above cryptanalysis example to encrypt their data in the portable hard disk with the strong encryption key(128 bits), encryption algorithm as well.

2.9. Existing Approaches

2.8.1. A substantial progress in the development of the new architectures for AES has been made after the conclusion of the contest, as a result of focusing research efforts on a single secret-key encryption standard. This progress proceeded in several major directions. One direction was the development of high-speed, highly pipelined architectures for non-feedback cipher modes. This direction led to the development of AES implementations operating with the speeds of tens of Gigabit per second [12, 13]. For example, Jarvinen et al [12] designed a fully pipelined memoryless AES -128 encryption to meet 17.8 Gbps on Xilinx' Vertex-E device family. Morika and Satoh [13] designed the 10 Gbps full-AES crypto design with proposing the twisted-BDD S-Box architecture. The second direction was the development of compact architectures for AES, optimized for the minimum area. This effort led to the emergence of architectures with 64 bits, 32 bits, and even 8 bits data paths [14, 15]. Satoh et al [14] designed a compact Rijndael hardware architecture by sharing the data path between encryption and decryption, sharing S-Box with key expander and use composite field arithmetic to design S-box hardware architecture by implementing multiplicative inversion over a new composite field. The result observed the best performance design of 2.6 Gbps with 21.3 Kgates. Good and Benaissa [15] compared the two AES different architectures from the highest speed with fully parallel loop unrolled architecture with the result at 25 Gbps to the lowest area architecture with round based architecture using 32 bits data path with the result at 2.2 Mbps on the 264 total slices of Xilinx Spartan-II which is sufficient for wireless and home user application. The third direction was the optimization of basic operations of the AES, including logic-only implementation of *SubBytes* [16, 17] and optimizations and decompositions of the *MixColumns* and *InvMixColumns* transformations [18]. Canright et al [16] used logic gate optimization by computing of S-Box Galois inverse from $GF(2^8)$ into the ground based on GF(2) and merged S-box and inverse S-Box into one operation. The result observed 20% smaller area than previously most compact AES version. Mentens et al [17] proposed the compact S-box architecture using composite field arithmetic in $GF(((2^2)^2)^2)$ to describe systematic

irreducible polynomials that generate extension filed. Their result observed the most compact area of Satoh's S-box is at least 5% away from the optimal solution. Fischer et al [18] shown new relationship and enable efficient resource sharing between Mixcolumn and InvMixcolumn. Their proposed architecture resulted in reduction of the area up to 20%.

Many published research in AES core/architecture have been announced. All of them purpose the AES improvement way how to improve AES cores efficiency in the terms of speed, scale size, power consumption, low cost, etc. But a little bit publication shows the specific implementation of purposed AES core in hard disk or data storage device in actual.

2.8.2. FDE (Full disk encryption) hard disk drive made by HDD vendors using the OPAL which is a set of the storage workgroup specifications providing an extensive architecture for putting storage devices under policy control and Enterprise standards was developed by the Trusted Computing Group. The commercial FDE hard disk has introduced by Seagate on June 2005. It is hard disk drive with security processor on board, encrypting the data on the fly and FDE did not require any special support from the OS. The drive is fully autonomous. All it needs is a BIOS supporting ATA Security commands. The security ASIC chip on the drive generates a unique AES key to encrypt all data. When the drive is not locked, the key is always released by the security processor and data is transparently encrypted on writing and decrypted on reads. Hence it works just like normal drive, you can write / read anything to it. The disadvantage of the system, the user should buy a new one for the FDE HDD unit to protect their critical data, the current HDD of user was not supported encrypt/decrypt properties. And a security processor was identically fixed and didn't flexibly support another HDD unit.

2.8.3. The Commercial AES core software/hardware, It was developed and implemented in many applications for example, the cipher for wireless communications including IEEE 802.11i (Wi-Fi), electronic financial transactions, power line networks, security video surveillance systems, secured RFID/smart card , encrypted data storage, etc. [19] to support the various users with very expensive

cost but no more application clearly shown the actual implementation and efficiency of these commercial AES cores in a data storage device.

CHAPTER 3

AES Core Architecture Design Methodology

The author starts to design AES core hardware architecture by separately considering in the details of each AES basic operation to support the minimum speed of USB3.0 data communication port (> 5 Gbps) with compact architecture. We originally develop the VHDL code to describe the behavioral logic level and synthesis them to be gate level until completely works and comply with AES algorithm of NIST and then we look for an improvement techniques to improve the efficiency of each operation block, for example, resource sharing, parallel processing, pipelining, etc. After that, combine them together to be the AES data encryption/decryption completely whole system (bottom up design methodology). This chapter should be studied the AES algorithm hardware architecture in details as the following steps.

3.1. S-Box (Byte Substitution)Transformation Design

The S-Box operation is a nonlinear byte substitution. It composes of 2 sub-transformations; multiplicative inverse and affine transformation.

1) The Multiplicative inverse of each byte is taken – this stage is to compute $B(x) = A^{-1}(x)$ for an 8 bits input word (in $GF(2^8)$ where $m(x) = x^8 + x^4 + x^3 + x + 1$ is taken as a field polynomial; {00} is mapped to itself).

2) Affine Transformation: This sub-step is performed in GF(2) and defined by.

$$D(x) = \delta B(x)_{mod(x^8+1)} \oplus C(x) \quad (4)$$

Where $\delta = \{1F\} = x^4 + x^3 + x^2 + x + 1$ for the encryption process and $\delta = \{4A\} = x^6 + x^3 + x$ for the decryption process. The constant c(x) has been added in order

that the S-box has no fixed point (a map to a), and no opposite fixed point (a map to a-bar). $C(x) = \{63\} = x^6 + x^5 + x + 1$ for encryption, and $C(x) = \{05\} = x^2 + 1$ for S-box inversion. Let $B(x) = A^{-1}(x)$ then eqn. (4) can also be rewritten in the matrix form as given in eqn. (5) for encryption process and eqn. (6) for decryption process as follow.

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

In many AES implementations, these 2 steps are typically combined into a single 16 by 16 lookup table with the content 8 bits in length. The ROM size of 256x8 bits is not big for current technology and can be implemented in a fairly simple manner with modern design tools. However when the area is restricted or a ROM cannot be incorporated, the inversion hardware becomes necessary. Within this situation, the efficient S-box implementation is the major concern of our design. The affine transform, still requires a small number/delay of gates should be introduced.

3.1.1. LUT S-BOX

SubBytes is composed of 16 identical 8 X 8 S-boxes working in parallel. InvSub-Bytes is composed of the same number of 8 X 8 bits inverse S-boxes. Each of these S-boxes can be implemented independently using a 256 X 8 bits look-up table. A lookup table is implemented in digital systems using ROM (read-only memory). In this memory, input to an S-box is connected to the address lines, and the output is obtained at the data out bus. Each of the AES *SubBytes* look-up tables is about the size of 256 bytes = 2048 bits = 2 kilobits. If encryption and decryption are implemented together within the same circuit, both un-inverted and inverted 256 byte look-up tables can be placed within one 512 byte memory block. In this case, the most significant bit of an address is a control bit that distinguishes between encryption and decryption. The LUT example of both S-box and Inverse S-box are shown in Figure 3.1 (a) and 3.1 (b) in respectively.The VHDL source code of design S-Box and Inverse S-Box with LUT is shown as follow with its RTL schematic is shown in Figure 3.2.

		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
X	0	63	7c	77	7b	f2	6b	6f	c5	30	1	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	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
	4	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	2	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	6	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	8
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	3	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

Figure 3.1 (a) A LUT S-Box (SubByte Transform)

X \ Y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	9	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	8	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	0	8c	bc	d3	0a	f7	e4	58	5	b8	b3	45	6
7	d0	2c	1e	8f	ca	3f	0f	2	c1	af	bd	3	1	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	7	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	4	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 3.1 (b) A LUT Inverse S-Box (Inverse SubByte Transform)

S-box for Encryption Process :

```
FOR i in 0 to 15 LOOP
    Sbox_OUT(opr)((((i+1)*8)-1) downto(i*8)) <= SBOX(conv_integer
    (SubBytes_IN(opr)((((i+1)*8)-1) downto(i*8))));
END LOOP;
```

S-box for Decryption Process :

```
FOR i in 0 to 15 LOOP
    Inv_SubBytes_OUT(opr)((((i+1)*8)-1) downto(i*8)) <=INV_SBOX
    (conv_integer (Inv_SubBytes_IN(opr)((((i+1)*8)-1) downto(i*8))));
END LOOP;
```

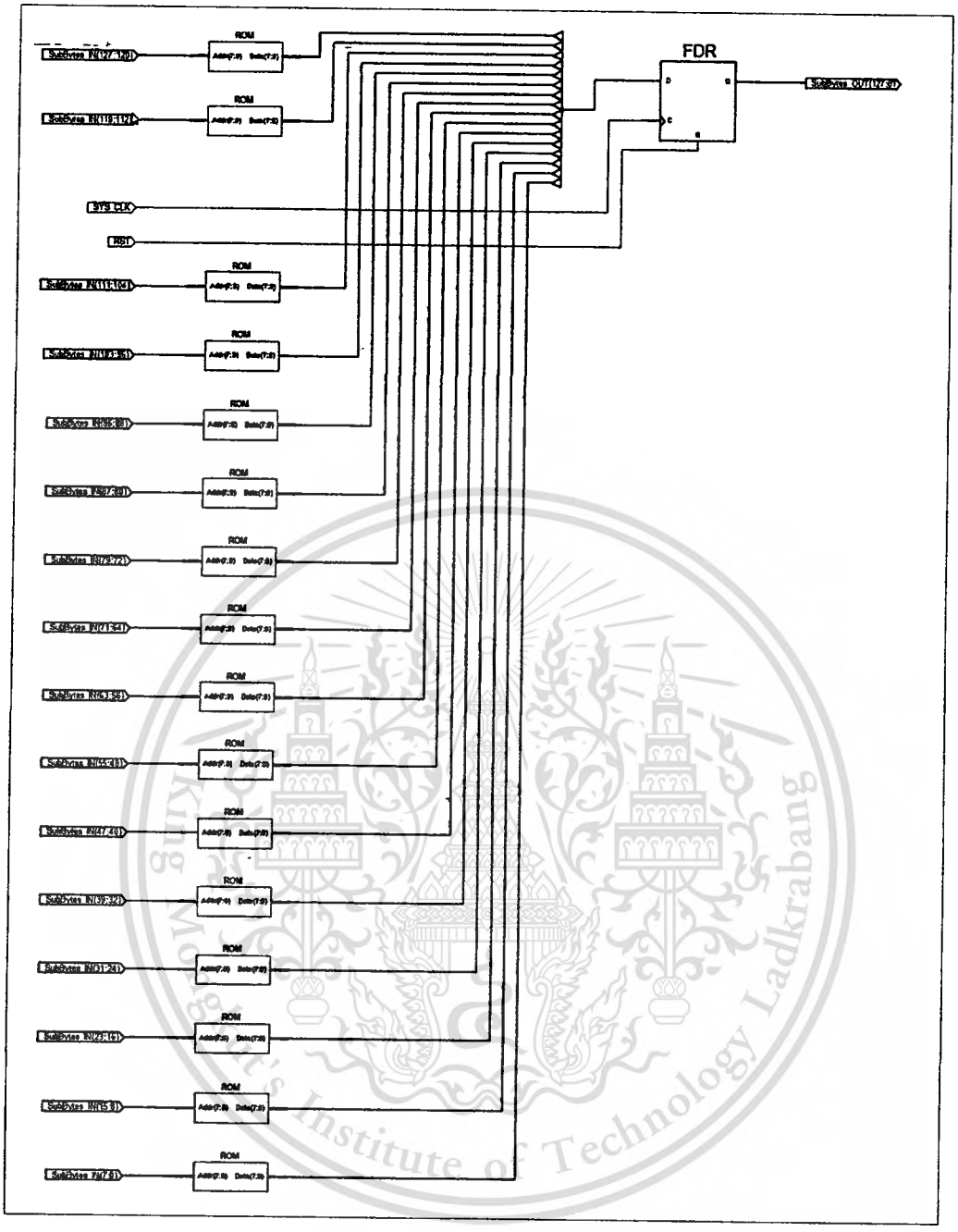


Figure 3.2 A RTL schematic of LUT S-Box (SubByte Transform)

3.1.2. Composite Field S-BOX

Defined by Rijndael, AES has adopted $m(x) = x^8 + x^4 + x^3 + x + 1$ as its field polynomial. Although such a polynomial is an irreducible one but it is not a primitive polynomial. The computation of $B(x) = A^{-1}(x)$ is considered to be intricate by most

authors and design engineers. Several techniques for S-box computation have been developed. These are, for instances; (1) To use table look up as mentioned above (2) To synthesize and optimized logic function by using CAD tools, and (3) To compute the inversion of elements in Galois field $GF(2^8)$ and optimize the logic functions. In the computation of element inversion in $GF(2^k)$ one can use either extended Euclid algorithm [20, 21] or composite field technique [22, 23, 24]. Rudra *et al.* [22], [23] has mapped all the operation except ShiftRow in the composite field of $GF(2^4)^2$. Morioka and Satoh [25] also have taken the advantage the used by composite field in the design of a low power S-box transformation. The Elements in $GF(2^8)$ are mapped to those defined in $GF(2^4)^2$. The Multiplication and inversion are optimized in the ground field. Our approach has drawn the techniques and many useful ideas reported in [25], [26], [27] and [28]. To reduce the unnecessary overhead, field transform is applied to the S-box computation only. It is not necessary to spoil it to the lowest ground field.

Working in the composite field, multiplicative inverse is leisured. However, forth and back, we have to map elements in $GF(2^k)$ into $GF(2^n)^m$ where k =mn. Therefore both transform and inverse transform matrices are needed. Elements in $GF(2^8)$ can be mapped to an element in $GF(2^4)^2$ by using the polynomial $r(x) = x^2 + x + \beta^{14}$ where β^{14} denotes the element in $GF(2^4)$ of which $I(x) = x^4 + x + 1$ is the primitive irreducible polynomial. The resulted mapping and inverse mapping matrices are given in eqn. (6) and (7) respectively.

$$T = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (6)$$

And

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (7)$$

An advantage of mapping elements from $GF(2^8)$ to be $GF((2^4)^2)$ is the simpler multiplicative inverse computation since inversion is performed in $GF(2^4)$. For such a small field size, inversion using either the direct truth-table mapping or table look up consumes small area. Moreover, in the Rijndael system, the data are treated naturally in byte format. Let data (byte) be expressed as $A = \{bc\} = bx + c$, the inversion of A, say $B = A^{-1} = \{pq\} = px + q$. For the field polynomial $r(x) = x^2 + Cx + D$, one can have.

$$p = b\Delta^{-1} \quad (8)$$

$$q = (Cb \oplus c)\,\Delta^{-1} \quad (9)$$

Where

$$\Delta = c\,(Cb \oplus c) \oplus b^2 D \quad (10)$$

Or

$$\Delta = bcC \oplus c^2 \oplus b^2 D \quad (11)$$

For $GF((2^n)^2)$, the polynomial in the form of $r(x) = x^2 + x + \lambda$ always exists [24]. As such, C and D can be set to {1} and {9} (in $GF(2^4)$) respectively. Fixed-coefficient multiplication (i.e., b^2D) as well as squaring units are relatively simple according to their small field size. The multiplications required in computing (5), (6) and (7) can be done straight away in $GF(2^4)$ or can be further simplified by making use of composite field $GF((2^2)^2)$ [25].

3.2. ShiftRow Transformation Design

The ShiftRow process operates on an individual row with an individual offset byte of state. In the state arrangement, the data are fed into a square matrix in row order. To operate the ShiftRow transformation, we need register#1 to store the whole data before byte swapping. This can result in the unsmooth data flow. However, the implementation is not very difficult. Due to we are designing the ShiftRow transform throughput in 128 bits per clock cycle to support the high throughput of hard disk. We used register#2 to be a pipeline arrangement (see Figure 3.3 below) such that the data are arranged in order and ready for the following operation, MixColumn transformation. The VHDL source code of design ShiftRow is shown as follow with its RTL schematic is shown in Figure 3.4.

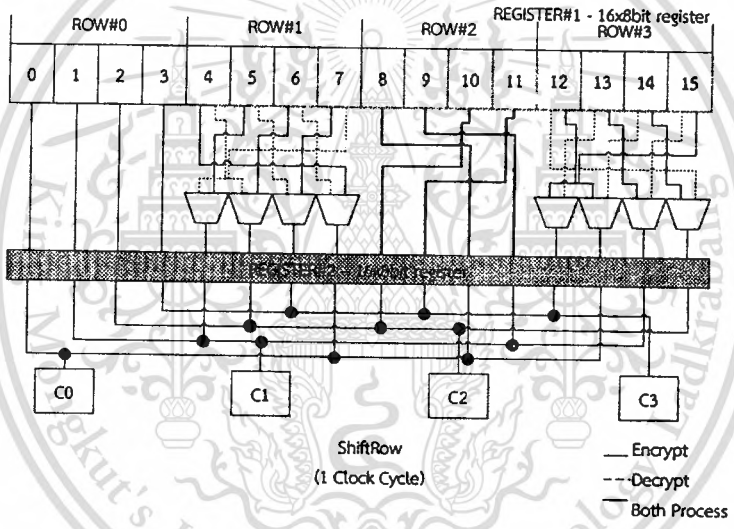


Figure 3.3 An architecture of ShiftRow and Inverse ShiftRow (dash line) switches with MIxColumn-Transform ready.

Example of ShiftRows for Encryption Process :

```
ShiftRows_In(opr) <= Sbox_OUT(opr);

ShiftRows_OUT(opr) <= ShiftRows_In(opr)(127 downto 120) &

ShiftRows_In(opr)(87 downto 80)              &
```

```
ShiftRows_In(opr)(47 downto 40)          &
ShiftRows_In(opr)(7 downto 0)            &
ShiftRows_In(opr)(95 downto 88)          &
ShiftRows_In(opr)(55 downto 48)          &
ShiftRows_In(opr)(15 downto 8)           &
ShiftRows_In(opr)(103 downto 96)         &
ShiftRows_In(opr)(63 downto 56)          &
ShiftRows_In(opr)(23 downto 16)          &
ShiftRows_In(opr)(111 downto 104)        &
ShiftRows_In(opr)(71 downto 64)          &
ShiftRows_In(opr)(31 downto 24)          &
ShiftRows_In(opr)(119 downto 112)        &
ShiftRows_In(opr)(79 downto 72)          &
ShiftRows_In(opr)(39 downto 32);
```

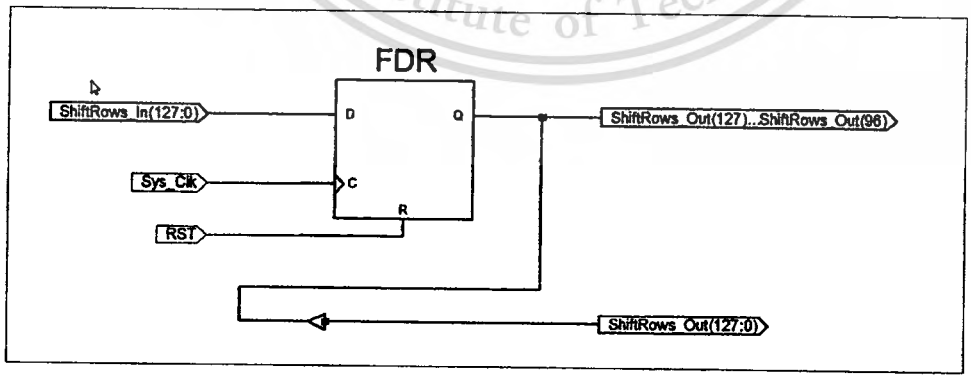


Figure 3.4 A RTL schematic of ShiftRow Process.

3.3. MixColumn Transformation Design

The mix column transformation operates on each column individually. Each byte in the column is mapped into a new that can be expressed by.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} c'_{0,0} & c'_{0,1} & c'_{0,2} & c'_{0,3} \\ c'_{1,0} & c'_{1,1} & c'_{1,2} & c'_{1,3} \\ c'_{2,0} & c'_{2,1} & c'_{2,2} & c'_{2,3} \\ c'_{3,0} & c'_{3,1} & c'_{3,2} & c'_{3,3} \end{bmatrix} \quad (12)$$

The forward MixColumn transform of an AES can be expressed as $C(x)a(x)_{mod(x4+1)} = C'(x)$ and each column is considered as a polynomial with coefficients Ci,c defines in $GF(2^8)$. The multiplication is modulo $x^4 + 1$ and a(x) is given by $a(x)= a_0 + a_1x + a_2x^2 + a_3x^3$ where a0 = {02}, a1 = a2 = {01} and a3 = {03} respectively. The computation given by (12) can be implemented with a circuit shown in Figure 3.8.Let $C_{i,c} = B(x)$ be an element to be multiplied. The B(x) can also be written in the term of polynomial form as.

$$B(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + b_4x^4 + b_5x^5 + b_6x^6 + b_7x^7 \quad (13)$$

Where $b_k \in (0,1)$. The Multiplication transformation used in the forward MixColumn is {03}B(x) = (x+1)B(x) and {02}B(x) = xB(x). The result of multiplication is.

$$\{03\}B(x) = (b_0 \oplus b_7) + (b_0 \oplus b_1 \oplus b_7)x + (b_1 \oplus b_2)x^2 + (b_2 \oplus b_3 \oplus b_7)x^3 + (b_3 \oplus b_4 \oplus b_7)x^4 + (b_4 \oplus b_5)x^5 + (b_5 \oplus b_6)x^6 + (b_6 \oplus b_7)x^7 \quad (14)$$

$$\{02\}B(x) = b_7 + (b_0 \oplus b_7)\,x + b_1x^2 + (b_0 \oplus b_7)\,x^3 + (b_3 \oplus b_7)x^4 + b_4x^5 + b_5x^6 + b_6x^7 \quad (15)$$

The implementation of the above equation is simple since additions are simply XORs. As an example the circuit to compute xB_i is shown in Figure 3.5 below. The implementation of $(x+1)B_i$ shown in Figure 3.6 can be done similarly.[29]

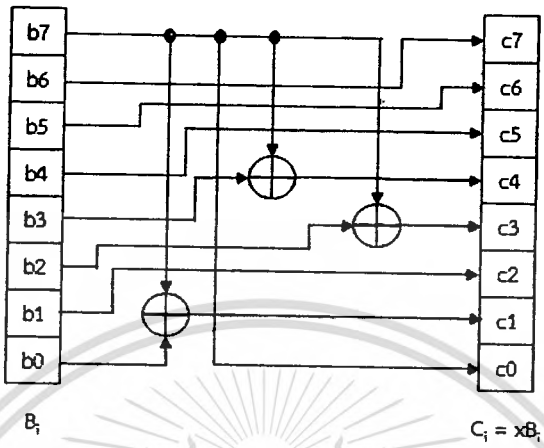


Figure 3.5 A x2 Fixed coefficient multiplier

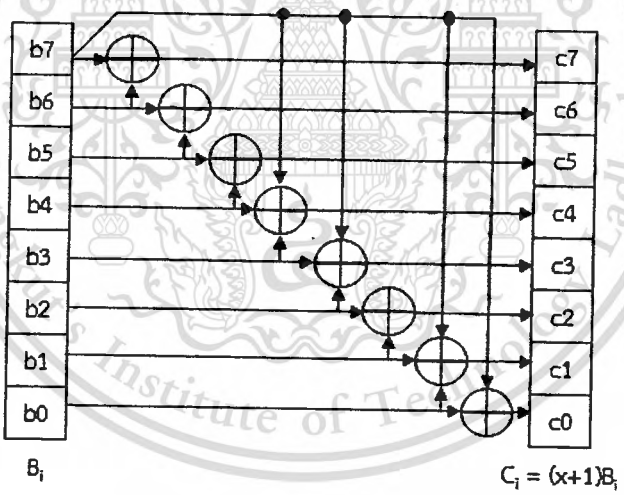


Figure 3.6 A x3 Fixed coefficient multiplier

The inverse MixColumn matrix can be written. In similar way $b(x) = b_0 + b_1x + b_2x^2 + b_3x^3$ is defined as the inverse transform polynomial where b0 = {0E} ,b1 = {09}, b2 = {0D} and b3 = {0B} respectively.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} \dot{c}_{0,0} & \dot{c}_{0,1} & \dot{c}_{0,2} & \dot{c}_{0,3} \\ \dot{c}_{1,0} & \dot{c}_{1,1} & \dot{c}_{1,2} & \dot{c}_{1,3} \\ \dot{c}_{2,0} & \dot{c}_{2,1} & \dot{c}_{2,2} & \dot{c}_{2,3} \\ \dot{c}_{3,0} & \dot{c}_{3,1} & \dot{c}_{3,2} & \dot{c}_{3,3} \end{bmatrix} \quad (16)$$

The computation given by (12) can be implemented with a circuit shown in Figure 3.8. Unfortunately, element C_i of the inverse MixColumn transform are not as simple as forward MixColumn transform. This can cause a restriction of hardware sharing between these two transforms. The required multiplication is treated as before, i.e. multiplication in $GF(2^8)$ modulo $m(x) = x^8 + x^4 + x^3 + x + 1$. Similarly let Let $C_{i,c} = D(x)$ be an element to be multiplied. The result of multiplications is.

$$\{0D\}D(x) = (d_0 \oplus d_5 \oplus d_6) + (d_1 \oplus d_5 \oplus d_7)x + (d_0 \oplus d_2 \oplus d_6)x^2 + (d_0 \oplus d_1 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_7)x^3 + (d_1 \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_7)x^4 + (d_2 \oplus d_3 \oplus d_5 \oplus d_6)x^5 + (d_3 \oplus d_4 \oplus d_6 \oplus d_7)x^6 + (d_4 \oplus d_5 \oplus d_7)x^7 \quad (17)$$

$$\{0E\}D(x) = (d_5 \oplus d_6 \oplus d_7) + (d_0 \oplus d_5)x + (d_0 \oplus d_1 \oplus d_6)x^2 + (d_0 \oplus d_1 \oplus d_2 \oplus d_5 \oplus d_6)x^3 + (d_1 \oplus d_2 \oplus d_3 \oplus d_5)x^4 + (d_2 \oplus d_3 \oplus d_4 \oplus d_6)x^5 + (d_3 \oplus d_4 \oplus d_5 \oplus d_7)x^6 + (d_4 \oplus d_5 \oplus d_6)x^7 \quad (18)$$

$$\{0B\}D(x) = (d_0 \oplus d_5 \oplus d_7) + (d_0 \oplus d_1 \oplus d_5 \oplus d_6 \oplus d_7)x + (d_1 \oplus d_2 \oplus d_6 \oplus d_7)x^2 + (d_0 \oplus d_2 \oplus d_3 \oplus d_5)x^3 + (d_1 \oplus d_3 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_7)x^4 + (d_2 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_7)x^5 + (d_3 \oplus d_5 \oplus d_6 \oplus d_7)x^6 + (d_4 \oplus d_6 \oplus d_7)x^7 \quad (19)$$

$$\{09\}D(x) = (d_0 \oplus d_5) + (d_1 \oplus d_5 \oplus d_6)x + (d_2 \oplus d_6 \oplus d_7)x^2 + (d_0 \oplus d_3 \oplus d_5 \oplus d_7)x^3 + (d_1 \oplus d_4 \oplus d_5 \oplus d_6)x^4 + (d_2 \oplus d_5 \oplus d_6 \oplus d_7)x^5 + (d_3 \oplus d_6 \oplus d_7)x^6 + (d_4 \oplus d_7)x^7 \quad (20)$$

The maximum addition terms in the fixed coefficient multiplication are six terms. This can cause the delay of 2τ unit, where τ is the unit delay time of a single

XOR gate (2 or 3 inputs). Above fixed coefficient multiplications can be realized as shown in Figure 3.7.

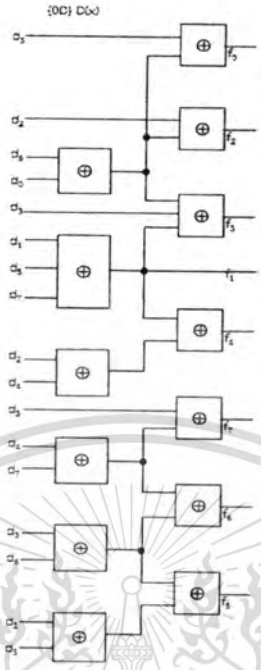


Figure 3.7. Fixed coefficient implementation example of F(x) = {0D}D(x)

C_i,C is computed in one clock cycle (or one column per one clock cycle) with four parallel fixed-coefficient multipliers, followed by a summing operation as shown in Figure 3.8. A compact size hardware with one column transform per 4 clock cycles can be arranged similarly.

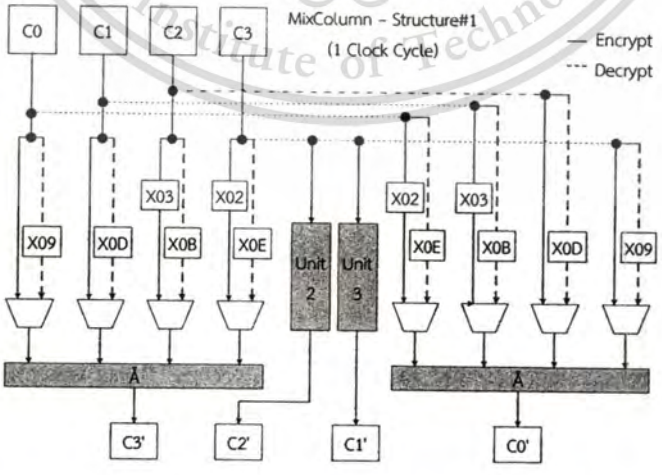


Figure. 3.8. MixColumn and Inverse MixColumn transform

The VHDL source code of design MixColumn and Inverse MixColumn Transform is shown as follows with its RTL schematics is shown in Figure 3.9.

Example of MixColumn for Encryption Process :

```
COLUMN_IN(opr) <= ShiftRows_OUT(opr)(127 downto 0);
a0(opr) <= COLUMN_IN(opr)(127 downto 120);
a1(opr) <= COLUMN_IN(opr)(119 downto 112);
a2(opr) <= COLUMN_IN(opr)(111 downto 104);
a3(opr) <= COLUMN_IN(opr)(103 downto 96);
----Fixed Coefficient Multiplication (a0x2, a1x2, a2x2, a3x2)
IF (a0(opr)(7) = '1') then    ---Each byte  is it's own conditin
        a0x2(opr) <=(a0(opr)(6 downto 0) & '0') XOR X"1B";
ELSE
        a0x2(opr) <= a0(opr)(6 downto 0) & '0';
END IF;
IF (a1(opr)(7) = '1') then
        a1x2(opr) <=(a1(opr)(6 downto 0) & '0') XOR X"1B";
ELSE
        a1x2(opr) <= a1(opr)(6 downto 0) & '0';
END IF;
IF (a2(opr)(7) = '1') then
        a2x2(opr) <=(a2(opr)(6 downto 0) & '0') XOR X"1B";
ELSE
        a2x2(opr) <= a2(opr)(6 downto 0) & '0';
END IF;
IF (a3(opr)(7) = '1') then
        a3x2(opr) <=(a3(opr)(6 downto 0) & '0') XOR X"1B";
ELSE
        a3x2(opr) <= a3(opr)(6 downto 0) & '0';
END IF;
----Fixed Coefficient Multiplication (a0x3, a1x3, a2x3, a3x3)
        a0x3(opr) <= a0x2(opr) XOR a0(opr);
```

```
a1x3(opr) <= a1x2(opr) XOR a1(opr);
a2x3(opr) <= a2x2(opr) XOR a2(opr);
a3x3(opr) <= a3x2(opr) XOR a3(opr);
r0(opr) <= a0x2(opr) XOR a1x3(opr) XOR a2(opr) XOR a3(opr);
r1(opr) <= a1x2(opr) XOR a2x3(opr) XOR a3(opr) XOR a0(opr);
r2(opr) <= a2x2(opr) XOR a3x3(opr) XOR a0(opr) XOR a1(opr);
r3(opr) <= a3x2(opr) XOR a0x3(opr) XOR a1(opr) XOR a2(opr);
```

Example of InvMixColumn for Decryption Process :

```
Inv_COLUMN_IN(opr) <= Inv_ShiftRows_OUT(opr)(127 downto 0);
a0(opr) <= Inv_COLUMN_IN(opr)(127 downto 120);
----Fixed Coefficient Multiplication (a0xD)
a0xD(opr)(0) <= a0(opr)(0) XOR a0(opr)(5) XOR a0(opr)(6);
a0xD(opr)(1) <= a0(opr)(1) XOR a0(opr)(5) XOR a0(opr)(7);
a0xD(opr)(2) <= a0(opr)(0) XOR a0(opr)(2) XOR a0(opr)(6);
a0xD(opr)(3) <= a0(opr)(0) XOR a0(opr)(1) XOR a0(opr)(3) XOR
a0(opr)(5) XOR a0(opr)(6) XOR a0(opr)(7);
a0xD(opr)(4) <= a0(opr)(1) XOR a0(opr)(2) XOR a0(opr)(4) XOR a0(opr)(5)
XOR a0(opr)(7);
a0xD(opr)(5) <= a0(opr)(2) XOR a0(opr)(3) XOR a0(opr)(5) XOR
a0(opr)(6);
a0xD(opr)(6) <= a0(opr)(3) XOR a0(opr)(4) XOR a0(opr)(6) XOR
a0(opr)(7);
a0xD(opr)(7) <= a0(opr)(4) XOR a0(opr)(5) XOR a0(opr)(7);
----Fixed Coefficient Multiplication (a0xB)
a0xB(opr)(0) <= a0(opr)(0) XOR a0(opr)(5) XOR a0(opr)(7);
a0xB(opr)(1) <= a0(opr)(0) XOR a0(opr)(1) XOR a0(opr)(5) XOR
a0(opr)(6) XOR a0(opr)(7);
a0xB(opr)(2) <= a0(opr)(1) XOR a0(opr)(2) XOR a0(opr)(6) XOR
a0(opr)(7);
a0xB(opr)(3) <= a0(opr)(0) XOR a0(opr)(2) XOR a0(opr)(3) XOR
a0(opr)(5);
```

```
                a0xB(opr)(4) <= a0(opr)(1) XOR a0(opr)(3) XOR a0(opr)(4) XOR a0(opr)(5)
XOR a0(opr)(6) XOR a0(opr)(7);
                a0xB(opr)(5) <= a0(opr)(2) XOR a0(opr)(4) XOR a0(opr)(5) XOR a0(opr)(6)
XOR a0(opr)(7);
                a0xB(opr)(6) <= a0(opr)(3) XOR a0(opr)(5) XOR a0(opr)(6) XOR
a0(opr)(7);
                a0xB(opr)(7) <= a0(opr)(4) XOR a0(opr)(6) XOR a0(opr)(7);
----Fixed Coefficient Multiplication (a0x9)
                a0x9(opr)(0) <= a0(opr)(0) XOR a0(opr)(5);
                a0x9(opr)(1) <= a0(opr)(1) XOR a0(opr)(5) XOR a0(opr)(6);
                a0x9(opr)(2) <= a0(opr)(2) XOR a0(opr)(6) XOR a0(opr)(7);
                a0x9(opr)(3) <= a0(opr)(0) XOR a0(opr)(3) XOR a0(opr)(5) XOR
a0(opr)(7);
                a0x9(opr)(4) <= a0(opr)(1) XOR a0(opr)(4) XOR a0(opr)(5) XOR
a0(opr)(6);
                a0x9(opr)(5) <= a0(opr)(2) XOR a0(opr)(5) XOR a0(opr)(6) XOR
a0(opr)(7);
                a0x9(opr)(6) <= a0(opr)(3) XOR a0(opr)(6) XOR a0(opr)(7);
                a0x9(opr)(7) <= a0(opr)(4) XOR a0(opr)(7);
----Fixed Coefficient Multiplication (a0xE)
                a0xE(opr)(0) <= a0(opr)(5) XOR a0(opr)(6) XOR a0(opr)(7);
                a0xE(opr)(1) <= a0(opr)(0) XOR a0(opr)(5);
                a0xE(opr)(2) <= a0(opr)(0) XOR a0(opr)(1) XOR a0(opr)(6);
                a0xE(opr)(3) <= a0(opr)(0) XOR a0(opr)(1) XOR a0(opr)(2) XOR a0(opr)(5)
XOR a0(opr)(6);
                a0xE(opr)(4) <= a0(opr)(1) XOR a0(opr)(2) XOR a0(opr)(3) XOR a0(opr)(5);
                a0xE(opr)(5) <= a0(opr)(2) XOR a0(opr)(3) XOR a0(opr)(4) XOR a0(opr)(6);
                a0xE(opr)(6) <= a0(opr)(3) XOR a0(opr)(4) XOR a0(opr)(5) XOR a0(opr)(7);
                a0xE(opr)(7) <= a0(opr)(4) XOR a0(opr)(5) XOR a0(opr)(6);
```

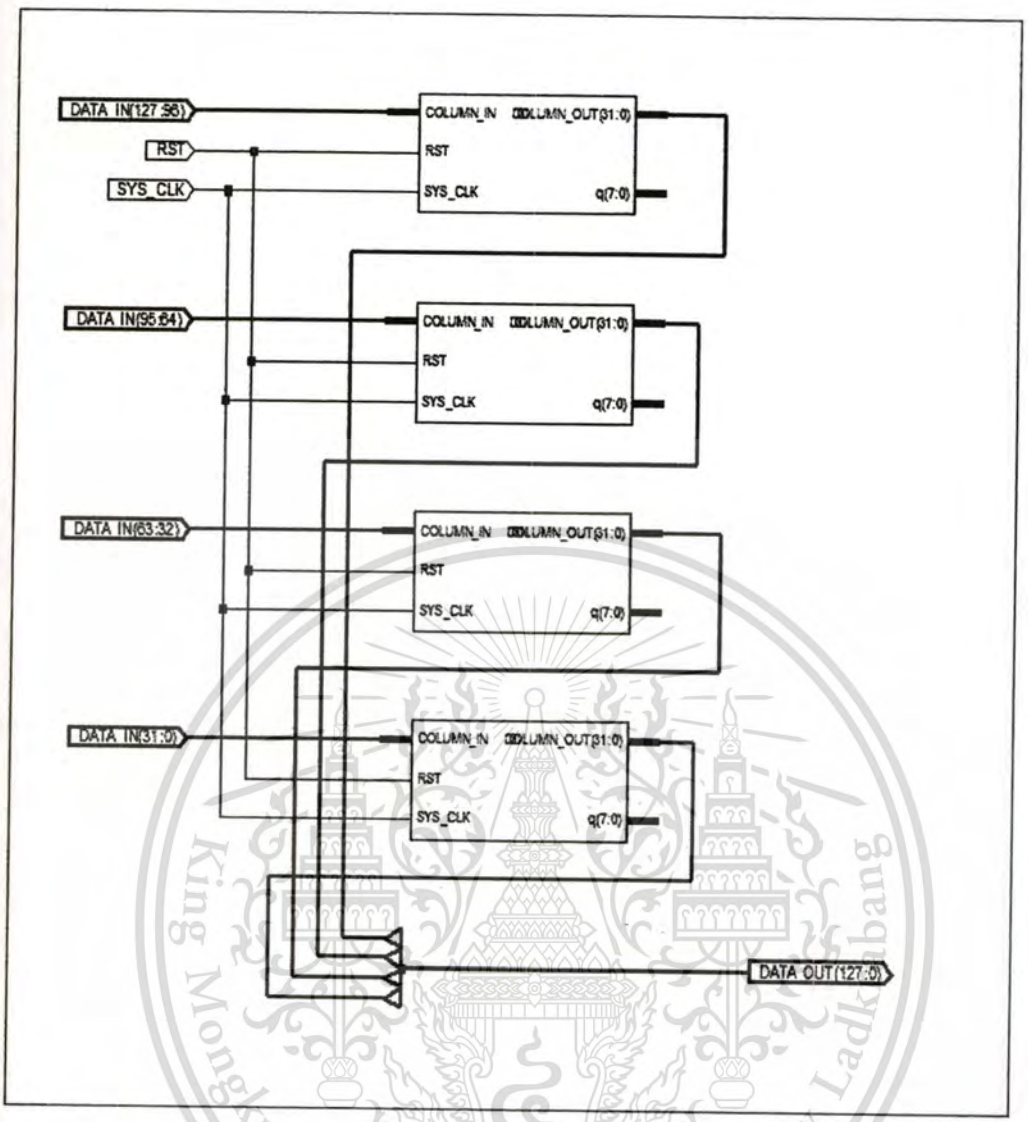


Figure. 3.9. A RTL Schematic of MixColumn Transform

3.4. KeyScheduling and AddRoundKey Transformation Design

The KeyScheduling expands the initial 128 bits cipher keys to generate the round keys. The two methods for the key expansion are commonly used, i.e., the round keys can be generated on-the-fly with the data transformation, or they are pre-calculated and stored for later use. In this paper, the round keys applied to the data transformation for encryption or decryption are calculated on-the-fly. The agility

of the key expansion deal with the situation that the cipher keys are changed frequently.

The implementation of the key generation for encryption is illustrated in Figure 3.10. Every word (32 bits) of the next state is the XOR of the current word in the same position with its left neighboring word. For example, the word in C_1 is calculated as $w'[C_1] = w[C_1] \oplus w[C_0]$. For the words in the position C_1, its neighboring word is in position C_3. A transformation RotWord is applied to the word in position C_3 prior the XOR, followed by an XOR with the round constant Rcon. So, we need two numbers of 128 bits register to store round keys. The S-box can be shared with the S-box as details in section 2. The RotWord register is a circular word shift register. And The RCon is a feedback word shift register. For the one stage sub pipelined AES structure, we have two different KeyScheduling modules which share the load for ten iterations. The KeyScheduling#1 generates the first five keys and the KeyScheduling#2 generates the last five keys. The entire KeyScheduling module generates three sets of keys for iteration. Two sets of keys from the KeyScheduling#1, #2 and the third, by accumulating the four 32 bits of the outputs of the input block.

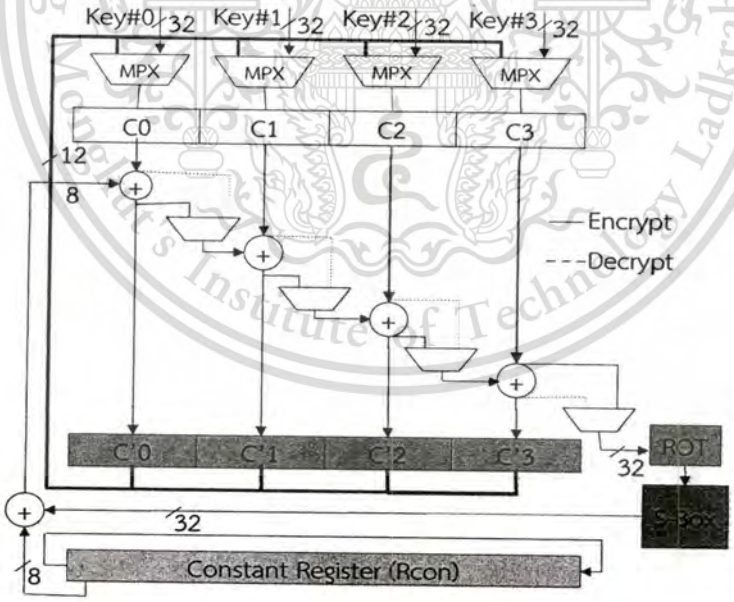


Figure 3.10 KeyScheduling Structure

The KeyScheduling VHDL source code is shown as below and RTL schematic is shown in Figure 3.11.

```
FOR opr IN 0 TO 9 LOOP
        IF opr = 0 THEN
                ACTIVE_KEY(opr) <= KEY_128;
        ELSE
                ACTIVE_KEY(opr) <= KEY_BUF(opr-1);
        END IF;
    ROTWRDSUB(opr)  <= SBOX(conv_integer(ACTIVE_KEY(opr)(23 downto 16))) &
    SBOX(conv_integer(ACTIVE_KEY(opr)(15 downto 8))) &
    SBOX(conv_integer(ACTIVE_KEY(opr)(7 downto 0))) &
    SBOX(conv_integer(ACTIVE_KEY(opr)(31 downto 24)));
    ACT_RCON(opr) <= RCON(KEY_INTRVL);
    KEY_COL_0(opr) <= ACTIVE_KEY(opr)(127 downto 96) XOR ROTWRDSUB(opr)
    XOR (ACT_RCON(opr) & X"000000");
    KEY_COL_1(opr) <= ACTIVE_KEY(opr)(95 downto 64) XOR KEY_COL_0(opr);
    KEY_COL_2(opr) <= ACTIVE_KEY(opr)(63 downto 32) XOR KEY_COL_1(opr);
    KEY_COL_3(opr) <= ACTIVE_KEY(opr)(31 downto 0) XOR KEY_COL_2(opr);
    KEY_BUF(opr) <= KEY_COL_0(opr) & KEY_COL_1(opr) & KEY_COL_2(opr) &
    KEY_COL_3(opr);
    KEY_INTRVL <= KEY_INTRVL + 1;
END LOOP;
```

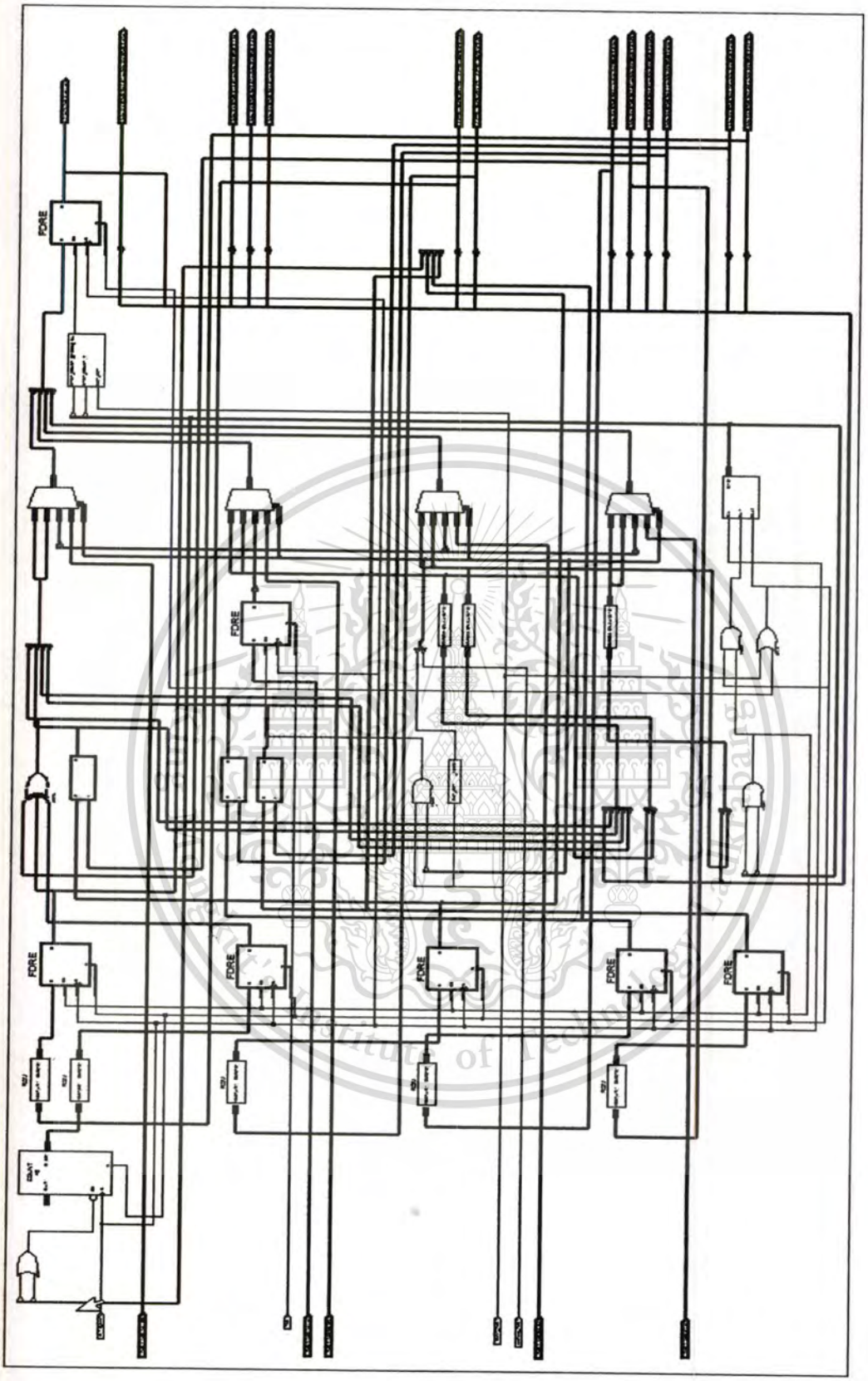


Figure. 3.11. A RTL Schematic of KeyScheduling

Too simple for AddRoundKey operation. Its VHDL source code is shown as below and RTL schematic is shown in Figure 3.12.

```
IF opr < 9 THEN
        output_each_round(opr) <= COLUMN_OUT(opr) XOR KEY_IN(opr);
ELSE
        output_each_round(opr) <= ShiftRows_OUT(opr) XOR KEY_IN(opr);
END IF;
```

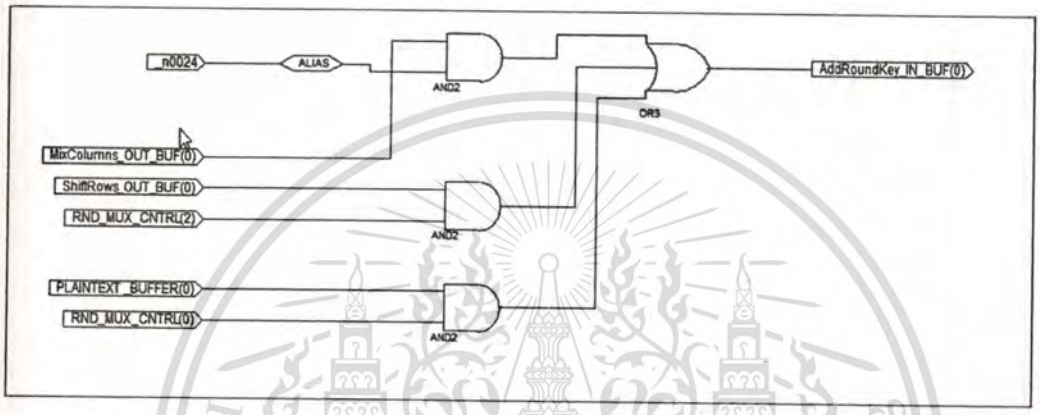


Figure. 3.12. A RTL Schematic of AddRoundKey

3.5. Design of Whole System AES Architecture

To obtain the lowest power and higher data rate AES (Gbit/second), the researcher has drawn many useful ideas and modified some technique reported in [30] to optimize the area size and design decision to obtain the high throughput of the system. The iterative structure design is chosen because of the small area of circuit sequentially running 10 rounds of AES operations in one data path. The encryption/decryption key length is limited to 128 bits to reduce the critical data path of the KeyScheduling data path. Since AES is efficiently used in non-feedback block cipher mode of operation, our architectural design can be pipelined also.

3.5.1. Compact Architecture

Due to AES is an 128 bits iterated data block cipher with a fixed data block size of 128 bits. So, the fully parallelism of 128 bits AES round operation should be

compounded 16 of 8 bits S-box and 4 of 32 bits MixColumn operation, working independently (except the ShiftRow operation that span throughout on 128 bits block). In order to compact the AES hardware architecture, the resources should be ideally cut by four but it should be used the four clock cycles to operate in one round. This approach will result in lower performance system than fully parallelism AES architecture. After that the author designs the whole hardware architecture by using the basic iterative AES with full 128 bits data path is shown in Figure 3.14 (Data Block#1). The same set of hardware is reused for all the ten iterations. This architecture is entirely based on the iterative approach of design for encryption algorithms. The key expansion block generates the key required for the corresponding iteration on the fly. This design harnesses the parallelism in the AES algorithm. The Multi Stage Sub-pipeline structure is shown in Figure 3.14 (Data Block#1, #2, ... , #n) is an improvement of the basic iterative architecture with respect to speed. It has just n stage of the pipeline within the data block. The data block is replicated once. In each stage the same workload is shared by two data blocks. The hardware used in the data block is just more than n times the hardware used in Basic iterative design. Next, the designing AES can be operated in both non-feedback block cipher mode of operation (ECB and CBC) to provide confidential information service that shown the block diagram as Figure 3.13.

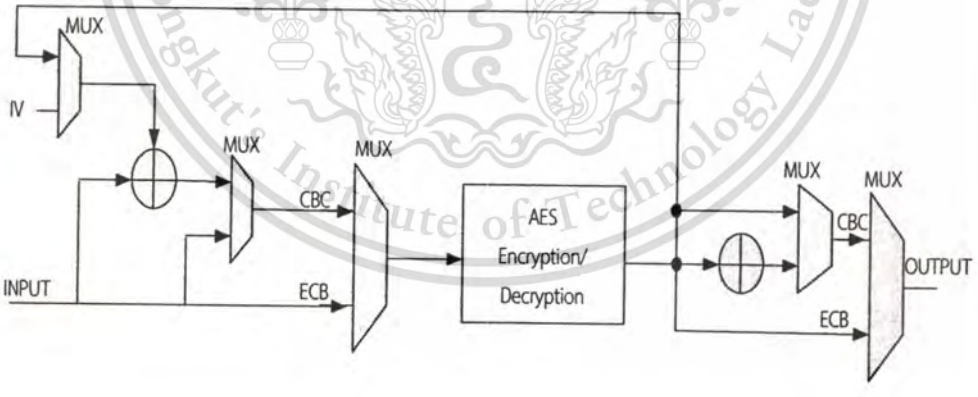


Figure 3.13. AES in non-feedback block cipher modes of operation

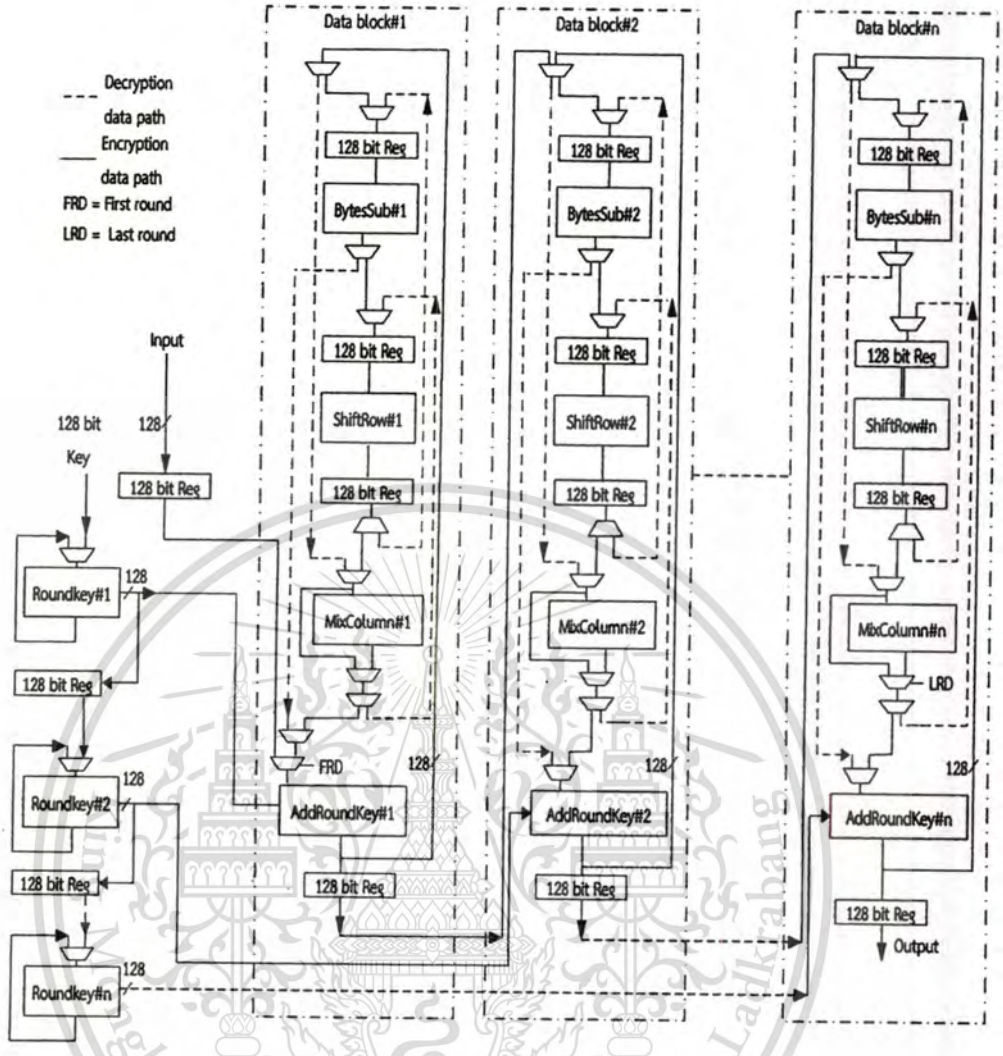


Figure 3.14. A 128 bits data path of multi stage sub pipelined AES

Furthermore, the system was designed to prepare the interfacing input, output and another control signal to support portable hard disk properties with USB and SATA cores that shown in Figure 3.15.

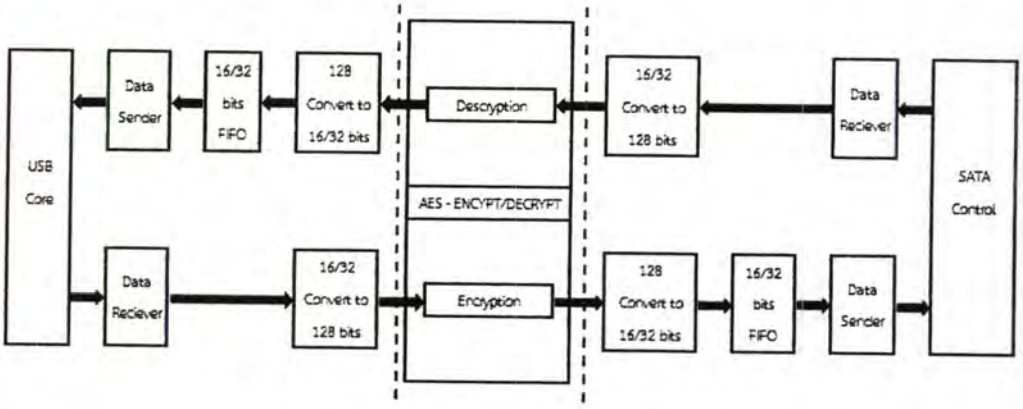


Figure 3.15. A framework of data encryption and decryption module interfacing

The FIFO (First in, First out) in Figure 3.16 uses for data bridging to transfer the data among designing AES, USB core and SATA core control. The author selects the FIFOs with static memory architecture design of Texas Instrument as shown a block diagram in Figure 3.16 to be a control signal interface design of designing an AES system to solve the long fall-through time in long FIFOs. Due to It uses a circular FIFO concept in Figure 3.17. The memory addressing of the incoming data that should be written/store is in the write pointer. The addressing of the 1'st data word in the FIFO that should be read out is in the read pointer. After reset, both pointers (write and read) indicate the same memory address location. After each write operation, the write pointer should be set to the next memory location and the reading of a data word should be set the read pointer to the next data word that is to be read out. The read pointer constantly follows the write pointer. When the read pointer reaches the write pointer, the FIFO is empty. If the write pointer catches up with the read pointer, the FIFO is full. All input signal listing are 16/32 bits INPUT DATA, WRITE CLOCK, CLEAR and READ CLOCK.And all output signal listing are 16/32 bits OUTPUT DATA, FULL, HALF FULL and EMPTY. The Read addresses are generated by the READ POINTER and the write addresses are generated by the WRITE POINTER. The write-control and read-control blocks control operation during write and read access. The FULL and EMPTY status signals are generated by separate flag logic. This FIFOs design also offers the status signals HALF FULL, ALMOST FULL, and ALMOST EMPTY. A FIFO with static memory also has to be initialized before it is used to set the two pointers and the status logic to define initial states.

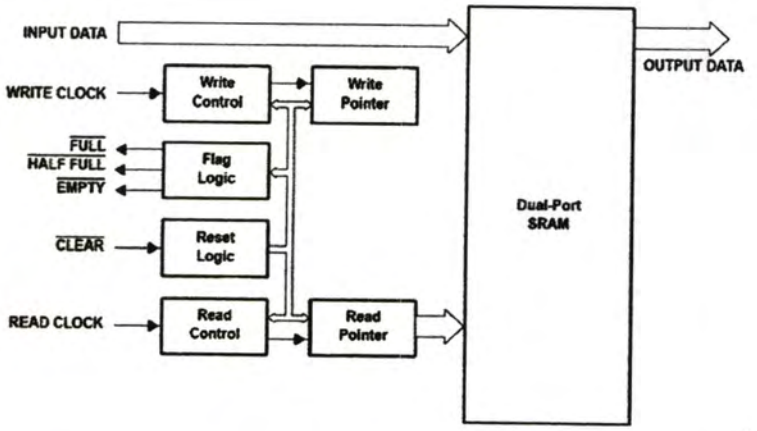


Figure 3.16. A FIFOs with static memory architecture of Texas Instruments

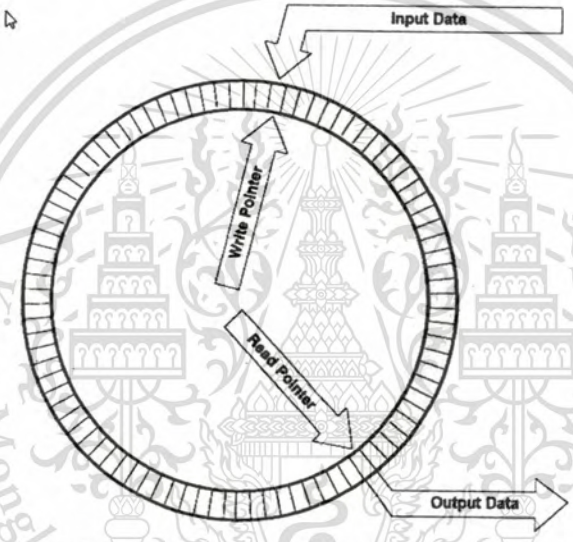


Figure 3.17. A Circular FIFOs with 2 pointers of Texas Instrument

CHAPTER 4

Architecture Design for AES

The strategy for AES core implementation on the FPGA device in this thesis is only focused on architectural optimization. The author exploits a design technique for example iterative, pipelining, sub-pipelining to optimize each AES operation block performance. Due to each AES block operation include fixed-rotation operation for ShiftRow/InvShiftRow that is hardwired and doesn't consume FPGA resource, the simple XOR operation for AddRoundKey/ InvAddroundKey, polynomial multiplication in GF operation for S-Box/Inv S-Box and constant polynomial multiplication in GF operation for MixColumn/InvMixColumn. Thus S-Box/Inv S-Box and MixColumn/ InvMixColumn operations are two key main units of AES implementation. It has been estimated that both AES operations block take more than 65% of the total area in the AES encryption/decryption system [31]. Therefore, both of S-Box/Inv S-Box and MixColumn/InvMixColumn operations are good candidates for improving overall performance of the system.

4.1. S-Box Implementation

The power related limitations of USB-powered portable hard disk is the main issue for solving it. Another system attached for example the designing AES data encryption/decryption with an FPGA device should be a big deal issue. In this thesis, we start to design a single module S-box with a composite field technique that implemented directly by referring to eqn.(6) through eqn.(11). This might be seen as a cheaper solution to reduce the FPGA resource than LUT approach. Most sub-circuits were taken from [28]. As the S-Box will be massively paralleled to process 128 byte data in few clock cycles, this part of the system must be carefully designed. The composite filed inversion used in this paper composes of few $GF(2^4)$ adders, multipliers, and a field inversion circuit as shown in Figure 4.1. The complexity detail is given in Table 4.1 below. The design was implemented in the low end Xilinx

Spartan2E FPGA devices(XC2S600E). Its resource utilizes as well as the obtained performances are given in Table 4.2.

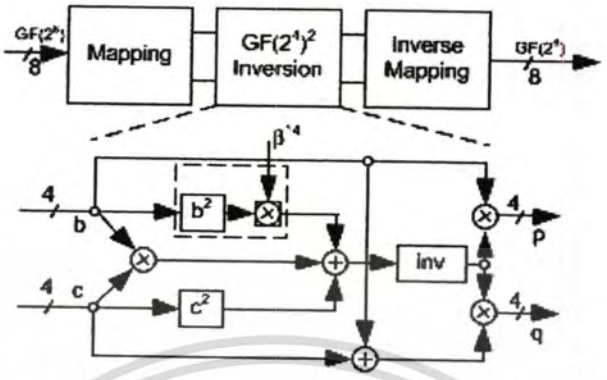


Figure 4.1 S-Box and S-Box^{-1} computation in GF($(2^4)^2$)

Table 4.1. Hardware complexity of a $GF(2^4)^2$ inversion

Component List	Complexity*			Unit
	AND	XOR	τ	Required
Squaring	-	2	1	2
Fixed-Coeff. Mul.	-	1	1	1
Inversion	11	11	3	1
Multiplication	16	10	3	3

* 1) 2 input gate and 3 input gate are equivalent

2) Gate delay of 1, 2 and 3 input is assumed to be equal

Table 4.2. Utilization Summary of Designing Single S-Box (no Sub-pipeline).

Utilization Summary	Designing - No Pipeline S-Box		
	Used	Available	Utilization
Number of Slices	814	6912	11%

Number of Slice Flip Flops	392	13824	2%
Number of 4 input LUTs	1536	13824	11%
Number of bonded IOBs	258	514	50%
Number of GCLKs	1	4	25%
Minimum period	13.637ns (Maximum Freq 73.330 MHz)		
Minimum input arrival time before clock	4.454ns		
Maximum output required time after clock	7.165ns		
Maximum combinational path delay	No path found		
Throughput	1.12 Gbps		

Affine and inverse affine transform circuits are fairly simple since they contain only a small number of XOR gates. The forward affine transform can be implanted directly using eqn.(2). Similarly the inverse affine transform can be done. The delay time of those circuits is 3τ and 2τ respectively. Here τ is a unit delay of a single gate (less than 3 inputs). For a 32 bits data path, 4 parallel S-Box is required. In total, 16 S-Box circuits (including affine transforms) are needed.

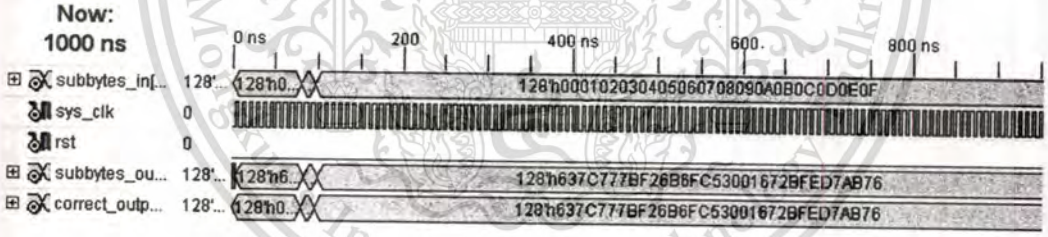


Figure 4.2 Single S-Box (no Sub-pipeline) function verification

For higher throughput, the S-box can be sub-pipelined to accommodate the throughput requirement of the system as illustrate in Figure 4.2. However the maximum delay between stages can be made down to as the lowest. This is limited by the delay of the $GF(2^4)$ inversion circuit as well as a multiplier. Pipeline option is given in Table 4.3. Upon the preliminary investigation, the obtained throughput is given in the last column. Obviously increasing the number of pipeline state boosts

the throughput; however at the penalty of increasing the circuit size as well as power consumption.

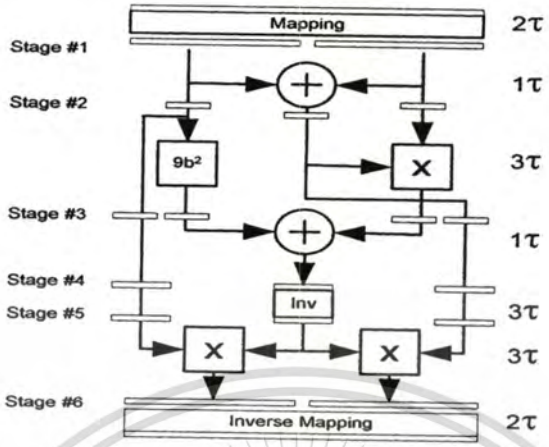


Figure 4.3 Sub-pipelining of the inversion circuit

Table 4.3. Pipelining Scheme of the Inversion Circuit

No. of Pipeline stage	Stages used	Max. delay between stages	Throughput (Gbps)
-	-	15τ	1.12
2	#4	8τ	1.87
3	#2, #4, #5	5τ	2.31
4	#2, #4, #5, #6	4τ	2.76
5	#2 - #6	3τ	2.93

The proposed sub-pipeline options can be chosen upon speed or throughput requirement. A non-pipelined S-Box can give the throughput of more than 1 Gbps as high. The designing S-Box is also shared by a KeyScheduling process. Due to the internal data path is 128 bits wide. So, 16 S-Box are required.

From Table 4.3., the system throughput still is not meet and far away from USB3.0 (5 Gbps) and SATA III (6 Gbps) bandwidth based on Table 2.2. The FPGA

device provides the huge number of flip-flops that should be used to put the register inside each step of a single round for a pipelining design strategy. The increase in performance is folded as much as the number of stages. The problem is that all stages must have similar delays which is not true for AES. So, The author tries to improve system throughput of the designing AES with LUT S-box. These LUT of both encryption and decryption was demonstrated in Chapter3. The 256x8 ROM is configured by using CLB to operate in memory mode for performing S-box , KeyScheduling. The LUT S-box utilization summary and verification result are shown in Table 4.4 and Figure 4.4 in respectively. The design doesn't use the FPGA dedicated resources (BRAM, etc.), so it has a high flexibility and can be implemented virtually in every commercial FPGA device. The maximum throughput is about 2.76 Gbps when compare with composite field S-box with non-pipelined on the same FPFA family at 1.12 Gbps but the CLB slice is greater than composite field S-box about 6%.

Table 4.4. Utilization Summary of Designing Single S-Box (LUT).

Utilization Summary	LUT S-Box		
	Used	Available	Utilization
Number of Slices	1158	6912	16%
Number of Slice Flip Flops	8	13824	0%
Number of 4 input LUTs	2316	13824	16%
Number of bonded IOBs	266	514	51%
Number of GCLKs	1	4	25%
Minimum period	5.522ns (Maximum Freq 181.09 MHz)		
Minimum input arrival time before clock	13.134ns		
Maximum output required time after clock	7.744ns		
Maximum combinational path delay	No path found		
Throughput	2.76 Gbps		

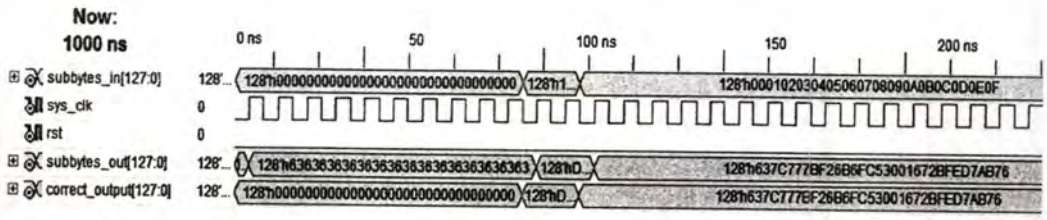


Figure 4.4 Single LUT S-Box function verification

Although the LUT S-box give us a higher system throughput greater than composite field S-box but it should be consumed more area. So, the designer should be a tradeoff between them again. It's depend on applications that could be applied.

4.2. ShiftRow Implementation

For the validation of previous described ShiftRow, we design it by using VHDL code as same as S-Box module that implemented directly by referring to Figure 3.9 Most sub-circuits were taken from [26, 27]. Resource utilization as well as the obtained performances is given in Table 4.5.

Table 4.5. ShiftRow FPGA Utilization

Utilization Summary	Designing - No Pipeline S-Box		
	Used	Available	Utilization
Number of Slices	74	6912	1%
Number of bonded IOBs	258	514	50%
Number of GCLKs	1	24	25%
Minimum input arrival time before clock	10.067ns		
Maximum output required time after clock	6.514ns		
Maximum combinational path delay	No path found		

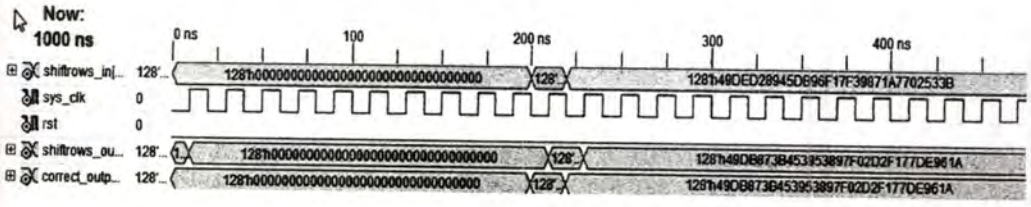


Figure 4.5 ShiftRow function verification

4.3. MixColumn Implementation

For the validation of previous described MixColumn, we design it by using VHDL code as same as other modules that implemented directly by referring to eqn.(14) through eqn.(20). Most sub-circuits were taken from [26, 27]. Resource utilization as well as the obtained performances is given in Table 4.6.

Table 4.6. MixColumn FPGA Utilization

Utilization Summary	Designing - No Pipeline S-Box		
	Used	Available	Utilization
Number of Slices	147	6912	2%
Number of 4 input LUTs	256	13824	1%
Number of bonded IOBs	258	514	50%
Number of GCLKs	1	4	25%
Minimum input arrival time (Bef CLK)	10.067ns		
Maximum output required time (Aft CLK)	6.514ns		
Maximum combinational path delay	No path found		

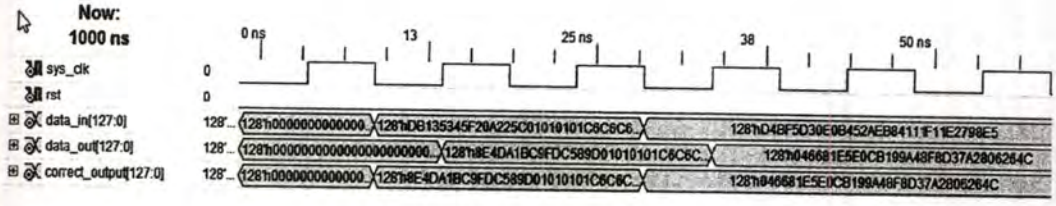


Figure 4.6 MixColumn function verification

4.4. AddRoundKey Implementation

For the validation of previous described AddRoundKey, we design it by using VHDL code as same as other modules that implemented directly by referring to Figure 3.8. Most sub-circuits were taken from [26, 27]. Resource utilization as well as the obtained performances is given in Table 4.7.

Table 4.7. AddRoundKey FPGA Utilization

Utilization Summary	Designing - No Pipeline S-Box		
	Used	Available	Utilization
Number of Slices	74	6912	1%
Number of 4 input LUTs	128	13824	0%
Number of bonded IOBs	386	514	75%
Number of GCLKs	1	4	25%
Minimum input arrival time (Bef CLK)	10.067ns		
Maximum output required time (Aft CLK)	6.514ns		
Maximum combinational path delay	No path found		

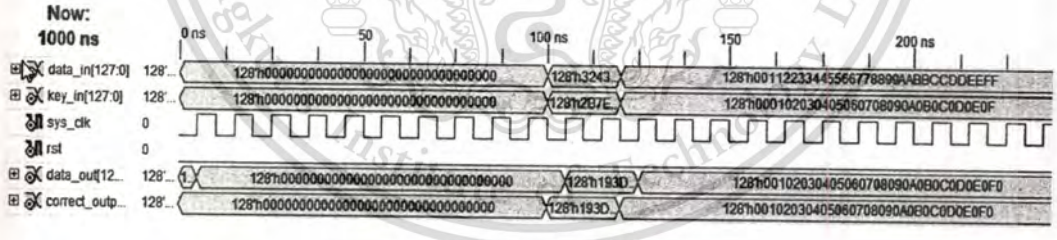


Figure 4.7 AddRoundKey function verification

4.5. KeyScheduling Implementation

For the validation of previous described KeyScheduling, we design it by using VHDL code as same as S-Box module that implemented directly by referring to

Figure 3.8. Most sub-circuits were taken from [26, 27]. Resource utilization as well as the obtained performances is given in Table 4.8.

Table 4.8. KeyScheduling FPGA Utilization

Utilization Summary	Designing - No Pipeline S-Box		
	Used	Available	Utilization
Number of Slices	609	6912	8%
Number of Slice Flip Flops	515	13824	3%
Number of 4 input LUTs	891	13824	6%
Number of bonded IOBs	260	514	50%
Number of GCLKs	1	4	25%
Minimum period	8.571ns (Maximum Frequency116.673 MHz)		
Minimum input arrival time (Bef Clk)	14.335ns		
Maximum output required time (Aft Clk)	8.944ns		
Maximum combinational path delay	No path found		
Throughput	1.79 Gbps		

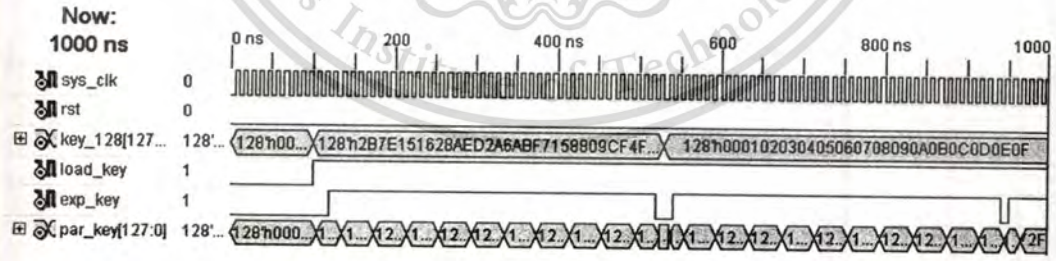


Figure 4.8 KeyScheduling function verification

The FPGA resource of each AES operation block was summarized in Table 4.9. The S-Box is the highest resource consumable than another block due to more complex

algorithm calculation (11% - 16% of overall resource). The second is KeySchduling. This operation block is still required S-Box operation for cipher key generating (8% of overall resource).

Table 4.9. Each designing AES block FPGA resource utilization implements on Xilinx Spartan2E FPGA devices (XC2S600E).

Operation	Number of Slice	% Resource
S-Box (LUT)	1158	16%
S-Box (Composite Field)	814	11%
ShiftRow	74	1%
MixColumn	147	2%
AddRoundKey	74	1%
KeyScheduling	609	8%

4.6. AES Data Path Optimization

One factor concerns about designing AES system throughput is the data path that use for transferring the data from one operation block to another operation block within the system. Although the system was designed with the maximum 128 bits data path architecture. Because of the different algorithm complexity in each AES operation. So, the smoothly data flow design within the designing AES system is needed. The Author measures the delay times of each designing AES operation to observe where is the bottleneck operation of the system and try to optimize delay times of each AES block operation as equal as we can. The Table 4.10 shows the delay times of each AES operation block.

Table 4.10. Delay times of each designing AES operation block

AES Operation	Delay Time (ns)
- LUT S-Box	1.935
- GF (No PipeLine) S-Box	5.091
- GF (3 PipeLine) S-Box	3.53
- GF (4 PipeLine) S-Box	2.32
- GF (5 PipeLine) S-Box	2.04
- GF (7 PipeLine) S-Box	1.849
ShiftRow	1.872
MixColumn (Old Design)	2.273
KeyScheduling (Old Design)	2.784
AddRoundKey	1.872

From Table 4.10 the Mixcolumn and KeyScheduling block are critical data path due to both of them have higher delay time than another block while LUT S-Box, ShiftRow and AddRoundKey have comparable delay time at about 1.9 ns. The delay time optimization process of these operation blocks should be designed as close as with S-Box delay time due to its algorithmic complexity.

4.6.1 MixColumn and Inverse MixColumn data path optimization

For MixColumn operation, The fixed coefficient multiplier in Galois field is a main operation that should be concerned with the delay times. So, the multiplication by 2 in Galois field is done with a left shift and a conditional XOR with X"1B" if the MSB is equal 1. And the multiplication by 3 in Galois field is done by first doing the multiplication by two and then XOR with (equivalent by adding) the original value are the best way that the author uses for improving delay times. For Inverse MixColumn operation, many addition terms in the fixed coefficient multiplication can cause the delay times. The author uses the method in [29] to realize the fixed coefficient multiplications complexity to compute them in one clock

cycle with four parallel fixed-coefficient multipliers, followed by a summing operation. The result after implementing this methodology shown in Figure 4.9 observe the delay time of MixColumn operation is running around 1.871 ns to close with another AES operation block.

VHDL source code modification :

```
---Multiply by 2 in the Galois Field is done with a left shift and a conditional XOR
with X"1B" if the MSB is 1
begin
    IF (a0(7) = '1') then    ---Each byte is its own condition
        a0x2 <=(a0(6 downto 0) & '0') XOR X"1B";
    ELSE
        a0x2 <= a0(6 downto 0) & '0';
    END IF;

    IF (a1(7) = '1') then
        a1x2 <=(a1(6 downto 0) & '0') XOR X"1B";
    ELSE
        a1x2 <= a1(6 downto 0) & '0';
    END IF;

    IF (a2(7) = '1') then
        a2x2 <=(a2(6 downto 0) & '0') XOR X"1B";
    ELSE
        a2x2 <= a2(6 downto 0) & '0';
    END IF;

    IF (a3(7) = '1') then
        a3x2 <=(a3(6 downto 0) & '0') XOR X"1B";
    ELSE
        a3x2 <= a3(6 downto 0) & '0';
    END IF;
```

--Multiply by 3: Multiply by 3 is done by first doing the Multiply by 2 and then XOR with (equivalent to adding) the original value

```
a0x3 <= a0x2 XOR a0;
a1x3 <= a1x2 XOR a1;
a2x3 <= a2x2 XOR a2;
a3x3 <= a3x2 XOR a3;
```

4.6.2 KeyScheduling data path optimization

The author observes some bug in VHDL source code that didn't calculate the key in one clock cycle. After modifying the source code again, the new delay time measure is about 1.856 ns to close with another AES operation block as the result shown in Figure 4.9.

VHDL source code modification :

```
KEY_COL_0(opr) <= ACTIVE_KEY(opr)(0) XOR ROTWRDSUB(opr) XOR
(ACT_RCON(opr) & X"000000");
KEY_COL_1(opr) <= ACTIVE_KEY(opr)(1) XOR KEY_COL_0(opr);
KEY_COL_2(opr) <= ACTIVE_KEY(opr)(2) XOR KEY_COL_1(opr);
KEY_COL_3(opr) <= ACTIVE_KEY(opr)(3) XOR KEY_COL_2(opr);
KEY_BUF(opr) <=  KEY_COL_0(opr) & KEY_COL_1(opr) & KEY_COL_2(opr) &
KEY_COL_3(opr);
```

Modifying Source Code :

```
KEY_COL_0(opr) <= ACTIVE_KEY(opr)(127 downto 96) XOR ROTWRDSUB(opr)
XOR (ACT_RCON(opr) & X"000000");
KEY_COL_1(opr) <= ACTIVE_KEY(opr)(95 downto 64) XOR KEY_COL_0(opr);
KEY_COL_2(opr) <= ACTIVE_KEY(opr)(63 downto 32) XOR KEY_COL_1(opr);
KEY_COL_3(opr) <= ACTIVE_KEY(opr)(31 downto 0) XOR KEY_COL_2(opr);
KEY_BUF(opr) <=  KEY_COL_0(opr) & KEY_COL_1(opr) & KEY_COL_2(opr) &
KEY_COL_3(opr);
```

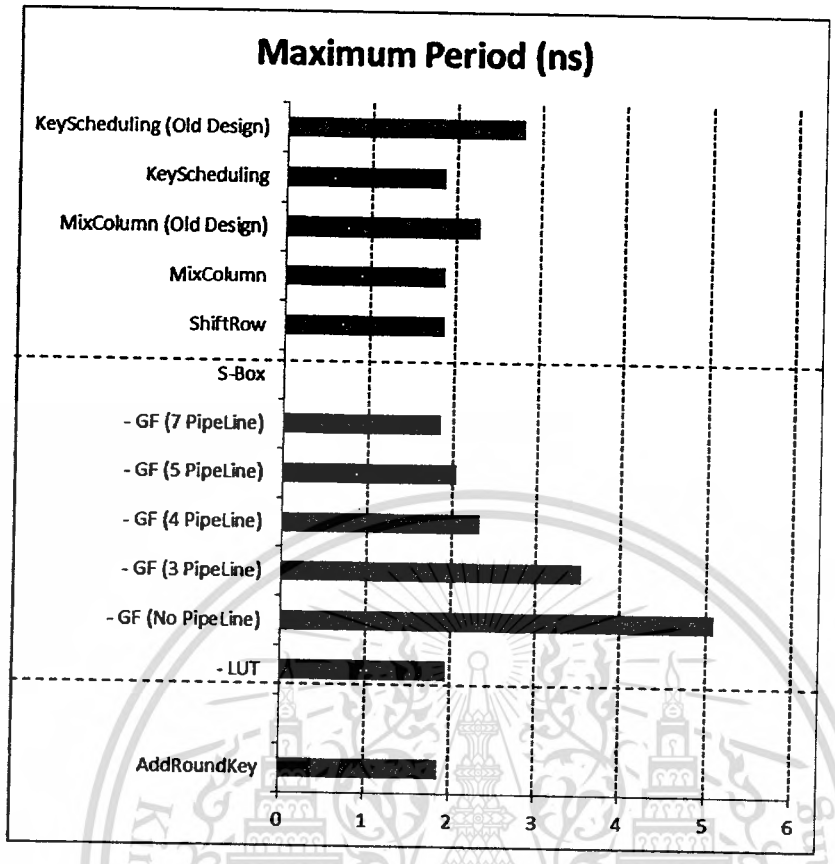


Figure 4.9 AES's data path optimization result

4.7. AES Core Entity

The entity of the AES core FPGA implementation is shown in Figure 4.10. Almost internal data path within the AES core entity are 128 bits. The top module provides three separate status signals for interfacing. There are DONE, NEAR_DONE and BUSY that to be the system status signaling to show encryption or decryption progress. When the encryption/decryption process is complete and valid ciphertext/plaintext are present on the LCD display. At this point, both a new key and new plaintext/ciphertext can be loaded, starting on the next rising clock edge. If a key was initially loaded, the core will continuously process the input plaintext/ciphertext with only a single cycle of delay between processing. This

The whole system AES resource utilization and function verification are given in Table 4.11 and Figure 4.11 in respectively.

Signal Pin :

Clock, Reset ; Input : AES system clock and Initial all internal registers signaling

Plaintext_In; Input : Input data path for encryption/decryption process

Enc/Dec; Input : Operation selective between Encryption/Decryption

Key_In, Key_Load ; Input : Input path for encryption/decryption key

Busy, Done, Near_Done; Output : System status flag/signaling

Full, Half_Full, Empty; Input : Handshakes signaling

Ciphertext_Out; Output : Output data path for encryption/decryption process

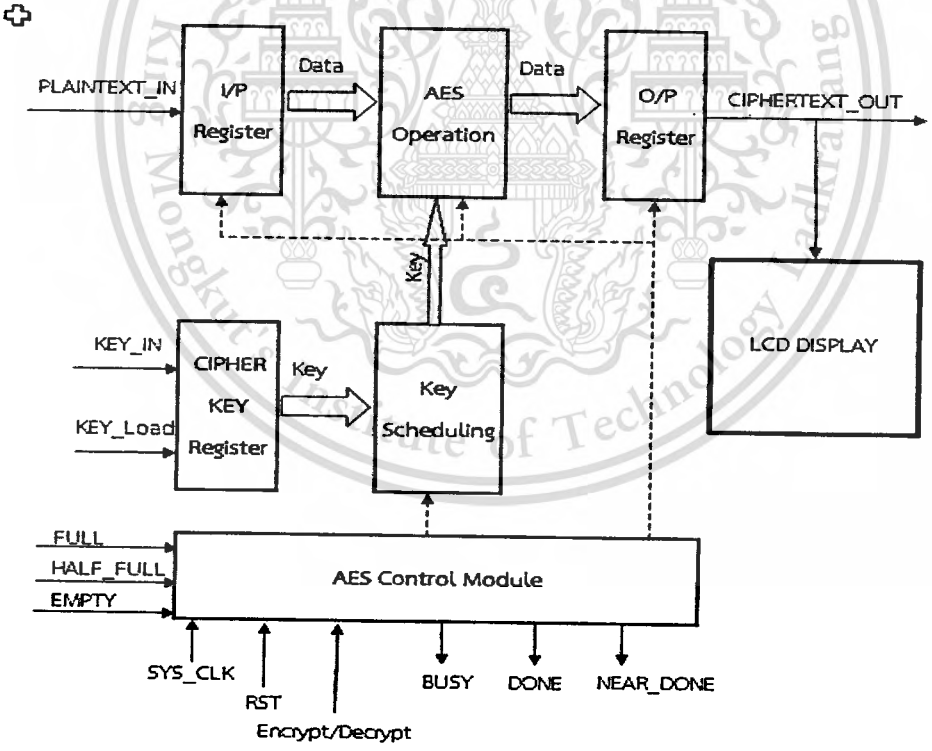


Figure 4.10. AES's core entity

Table 4.11. AES system FPGA Utilization (SPARTAN 2E, XC2S600E-6FG676)

Utilization Summary	AES Encryption System		
	Used	Available	Utilization
Number of Slices	2274	6912	32%
Number of Slice Flip Flops	1449	13824	10%
Number of 4 input LUTs	3731	13824	26%
Number of bonded IOBs	391	514	76%
Number of GCLKs	1	4	25%
Minimum period	10.09 ns (Maximum Freq 99.157 MHz)		
Minimum input arrival time before clock	12.120ns		
Maximum output required time after clock	9.632ns		
Maximum combinational path delay	No path found		
Throughput	1.51 Gbps		

Utilization Summary	AES Decryption System		
	Used	Available	Utilization
Number of Slices	2525	6912	36%
Number of Slice Flip Flops	1905	13824	13%
Number of 4 input LUTs	4091	13824	29%
Number of bonded IOBs	391	514	76%
Number of GCLKs	1	4	25%
Minimum period	10.187 ns (Maximum Freq 98.164MHz)		
Minimum input arrival time before clock	12.12ns		
Maximum output required time after clock	9.632ns		
Maximum combinational path delay	No path found		
Throughput	1.497 Gbps		

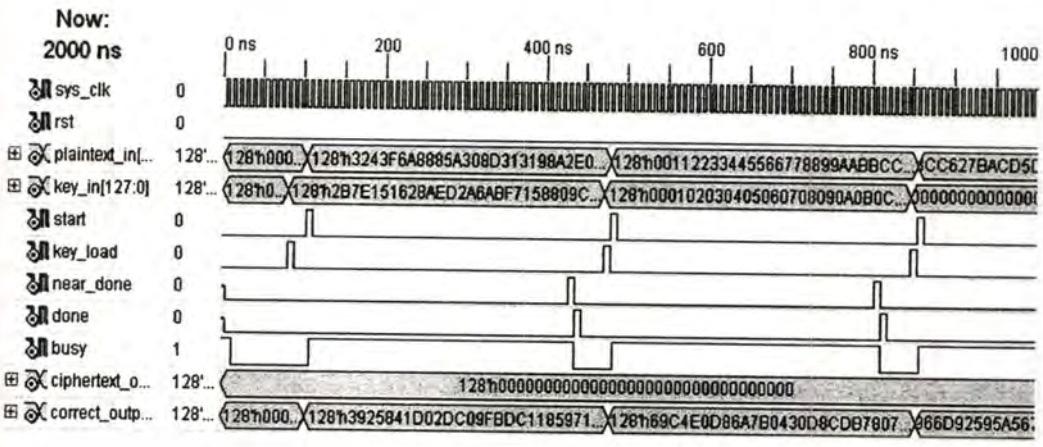


Figure 4.11 Whole basic AES iteration system function verification

Due to the designing system throughput is still far away from USB3.0 and SATA III operation speed. The Multi Stage Sub-pipeline structure is shown in Figure 3.14 (Data Block#1, #2, ... , #n) is a candidate to improve system throughput. It has just n stage of the pipeline within the data block. But the data block is replicated n times. The author uses one stage sub-pipelining to improve designing AES system throughput to tradeoff between speed and area to meet USB3.0 (5 Gbps) and SATA III (6 Gbps) speed also. The result observes the throughput of one stage sub-pipeline structure is 2.573 Gbps on SPARTAN-2E but the same system implementation with VERTEX 4 FPGA , the throughput is up to 8 Gbps as shown in Table 4.12 as below.

Table 4.12. Whole AES system comparison

AES Structure	Xilinx Device	CLB Slice	Delay (ns)	Fmax (MHz)	TP (Gbps)
Basic AES Iterations	SPARTAN2E with LUT S-Box	2274	2.32	99.16	1.51
	VERTEX4 with LUT S-Box	2969	0.864	340.599	5.197
	SPARTAN2E with GF S-Box	1689	2.23	55.402	0.845
	VERTEX4 with GF S-Box	1569	0.947	148.85	2.27
Basic AES Iterations	SPARTAN2E with LUT S-Box	4320	1.21	168.67	2.573
	VERTEX4 with LUT S-Box	5641	0.465	544.958	8.309

with One Stage sub Pipelined	SPARTAN2E with GF S-Box	3209	1.18	94.29	1.438
	VERTEX4 with GF S-Box	2981	0.477	238.75	3.64

4.8. Implementation Result

For the validation of all AES system module, we implement them on Warrior III (ALTERA – EP3C10E144C8) FPGA board of ASTRONLOGIC Co, Ltd. and display encryption/decryption result in the LCD display. The result observed the plaintext and ciphertext were complied with the cipher example in FIPS -197 and shown in Figure 4.12 to 4.15 in respectively.

Encryption Result :

PlainText := 3243f6a8885a308d313198a2e0370734

Cipher Key := 2b7e151628aed2a6abf7158809cf4f3c

Output :=

Figure 4.12 Encryption result

Decryption Result :

Cipher Text := 69c4e0d86a7b0430d8cdb78070b4c55a

Cipher Key := 13111d7fe3944a17f307a78b4d2b30c5

Output :

Figure 4.13 Decryption result

Encryption Result with ECB mode :

PlainText := 6bc1bee22e409f96e93d7e117393172a

Cipher Key := 2b7e151628aed2a6abf7158809cf4f3c

Output :=

Figure 4.14 Encryption with ECB result

Encryption Result with CBC mode :

PlainText := 6bc1bee22e409f96e93d7e117393172a

Cipher Key := 2b7e151628aed2a6abf7158809cf4f3c

Initial Vector := 000102030405060708090a0b0c0d0e0f

Output :=

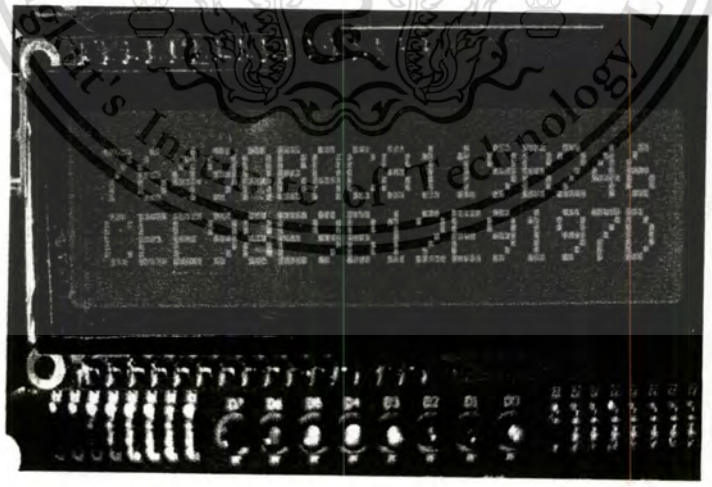


Figure 4.15 Encryption with CBC result

CHAPTER 5

Result and Conclusion

5.1. Performance Review

Based on an architecture optimization strategy to implement the iterative loop AES hardware architecture of FPGA chip to perform data encryption/decryption process in portable hard disks and support their power consumption issue [38] due to this architecture provides the small area design rather than the speed [31]. The initial Iterative Loop AES is synthesized on SPARTAN 2E (XC2S600E-6FG676) and VERTEX4 (XCVVLX15-12FF676) with composite field versus LUT byte substitution design occupies the CLB slices, I/O blocks, system clock in data encryption/ decryption modes as shown in Table 5.1. Each AES operation block is designed and considered delay time optimization especial MixColumn and KeyScheduling to compare with S-Box that is the most algorithm complexity in AES operations of smooth data flow. But It is still not meet the data transfer rate target. Then the one stage sub-pipeline iterative loop architecture that processed multiple AES block at the same time and at different stages to improve the data transfer rate of the initial IL AES system is implemented. The one stage sub-pipeline Iterative Loop AES is synthesized on SPARTAN 2E (XC2S600E-6FG676) and VERTEX4 (XC4VLX15-12FF676) with composite field versus LUT byte substitution design occupies the CLB slices, I/O blocks, System clock in data encryption/decryption modes as shown in Table 5.2. Its result met the data transfer rate target as well. Finally, the AES whole system is implemented on the evaluation FPGA board (Warrior Cyclone III, ALTERA - EP3C10E144C8 with 10,320 LEs) including ECB and CBC security mode. The result is complied with FIP197 example vector with the maximum speed at 8.31/8.22 Gbits/sec with Vertex4 occupies 5641/5688 CLB slices in the encryption / decryption mode in respectively. This design does not use of FPGA dedicated resources (BRAMs, etc.) So it has a high portability/efficiency and can be implemented in almost every commercial FPGA device and its performance is supported USB3.0 and SATA III operation speed to suitable for implementing in portable hard disk.

Table 5.1 Iterative Loop AES implementation result

Iterative Loop AES		Encryption				Decryption			
		CLB Slices	Delay (ns)	fmax (GHz)	TP Gbit/Sec	CLB Slices	Delay (ns)	fmax (GHz)	TP Gbit/Sec
SPARTAN 2E	Composite Field S-Box without Sub-Pipelining	1689	2.23	55.4	0.85	1796	2.2	55.4	0.845
	Composite Field S-Box with 7 stages Sub-Pipelining	1791	1.9	99.2	1.51	1782	1.8	96	1.46
	LUT S-Box	2274	2.32	99.2	1.51	2399	2.2	87.6	1.34
VERTEX4	Composite Field S-Box without Sub-Pipelining	1569	0.95	198.85	3.05	1584	0.94	198.46	3.03
	Composite Field S-Box with 7 stages Sub-Pipelining	2362	0.95	338.13	5.14	2545	0.95	315.09	4.81
	LUT S-Box	2969	0.87	340.59	5.197	2914	0.86	326.16	4.97

Table 5.2 Iterative Loop one stage sub-pipeline AES implementation

Iterative Loop AES with one stage sub-pipeline		Encryption				Decryption			
		CLB Slices	Delay (ns)	fmax (GHz)	TP Gbit/Sec	CLB Slices	Delay (ns)	fmax (GHz)	TP Gbit/Sec
SPARTAN 2E	Composite Field S-Box without Sub-Pipelining	3209	1.18	94.3	1.44	3412	1.02	92.4	1.41
	Composite Field S-Box with 7 stages Sub-Pipelining	3403	1.15	170	2.59	3386	1.12	169.4	2.5
	LUT S-Box	4320	1.21	168.7	2.57	4558	1.09	165.7	2.53
VERTEX4	Composite Field S-Box without Sub-Pipelining	2981	0.48	238.75	3.64	3009	0.46	232.15	3.54
	Composite Field S-Box with 7 stages Sub-Pipelining	4487	0.46	536.4	8.18	4835	0.46	520.52	7.95
	LUT S-Box	5641	0.47	544.96	8.31	5688	0.46	538.9	8.22

5.2. Other Design

To compare the designing AES system performance with another AES design implementation on FPGA chip would be a fair comparison if all designed were tested under the same environment. Ideally, the performance of AES core should be the same FPGA, same design strategies and same specification. The author classifies them in term of efficient designs according to the throughput over area ratio as shown in Table 5.3. So, we stress that to ignore the usage of BRAMs in our comparison. Our design is clearly more efficient than the other one AES system design.

5.3. Power Analysis

As designing an AES system for portable hard disk implemented on FPGAs, the power consumption is increasingly a concern for the system that will be connected to the USB port on the host computer. Normally, this communication port consumes the maximum power around 2.5 - 4.5 watt. Thus, obtaining an accurate estimate of power consumption is essential to design a system's power supply, and is crucial in developing an appropriate power budgeting for the whole system. The author uses XPE (Xilinx Power Estimator) tool to estimate an accurate power model of VERTEX4 (XC4VLX15-12FF676) on both composite field S-box with 7 stage sub-pipeline and LUT S-box designs (iterative loop AES with one stage pipeline). The power analysis results shown in Figure 5.1 and Figure 5.2 in respectively.

The designing AES power consumption based on VERTEX4 FPGA consumes 0.466 watts on composite S-Box 7 stage sub-pipeline and 0.447 watts on the LUT S-box to support 50°C ambient temperature that is the maximum possible temperature expected inside the hard disk enclosure box for housing the FPGA of designing AES. The designing AES is designed without active air flow across the chip that is measured in Linear Feet per Minutes(LFM). So, our design is more consuming about 10% - 20% of overall system power consumption.

Table 5.3. AES Comparison : Efficient Designs

Author	Core	Type	Device	Mode	Slices (BRAMs)	T(Mbps)	T/A
Saqib et al. [32]	E	P	XCV812E	ECB	2136(100)	5193	2.43
Pravin et al. [33]	E/D	IL	VERTEX - XCV600	ECB	1853	352	0.189
Lopez et al [34]	E	IL	SPARTAN 3 3s4000	ECB	633(53)	1067	1.68
Samir et al. [35]	E	IL	VERTEX5-XC5VLX50	ECB	587(2)	426	0.73
Alaoui-Ismaili et al. [36]	E	IL	VERTEX2 - XC2VP100	ECB	1250(40)	976	0.8
Calderon et al. [37]	E	IL	Altera EPFlOK	ECB	1584	637.24	0.4
This Thesis - 1	E	IL	**SPARTAN 2E -XC2S600E**	ECB	4320	2570	**0.6**
This Thesis - 2	E	IL	**VERTEX4X15**	ECB	5641	8310	**1.47**

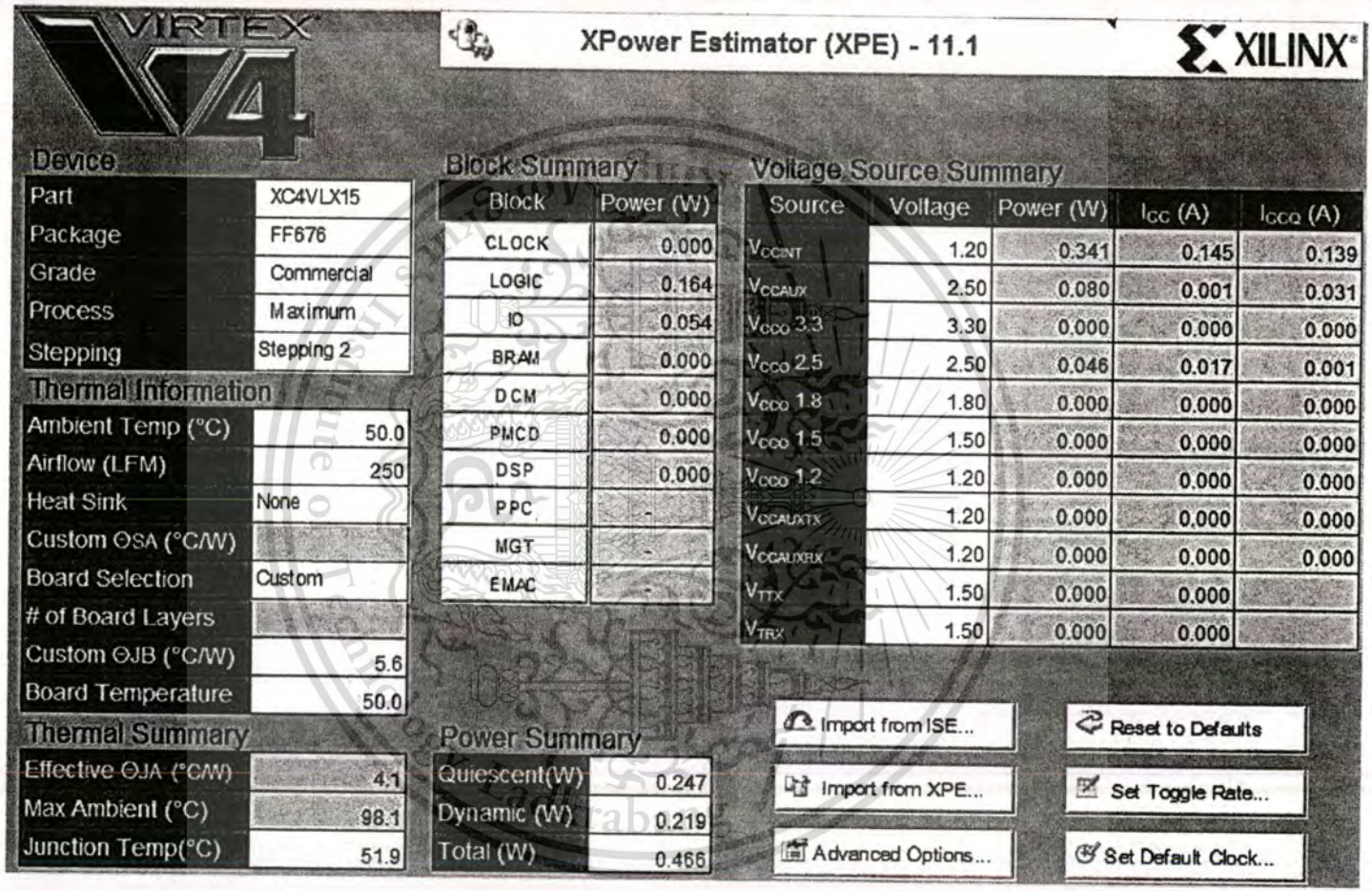


Figure 5.1. Composite S-Box with 7 Stages sub-pipeline power analysis result

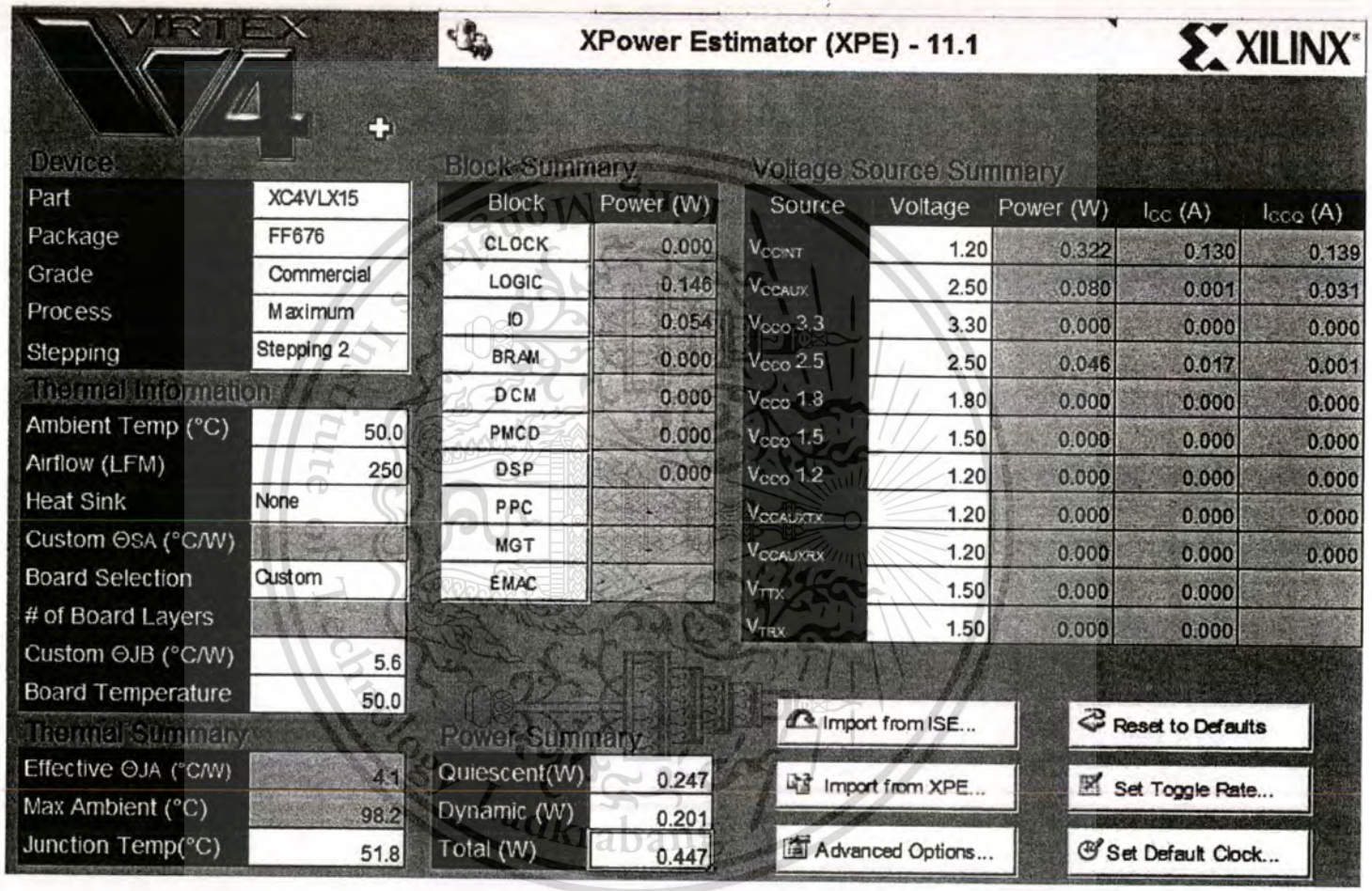


Figure 5.2. LUT S-Box power analysis result

5.4. Conclusion and Future work

The proposing AES core architecture, one stage sub-pipeline iteration with LUT S-Box architecture can give us the throughput more than 6 Gbps as high while the power consumption consumes 10% - 20% of the maximum power of USB communication port that operates under the ambient temperature as 50° C. Our system claims to be a high security with ECB or CBC cipher modes and high portability and can be implemented in every commercial FPGA device. It should be more efficient to implement for portable hard disk data encryption and decryption process supporting as well.

Many algorithmic strategies for efficient hardware implementation of AES based on FPGA devices have been proposed. Each strategy tries to obtain some mathematical expressions to take the advantage of FPGA structure. So, the future to broad work of this thesis could be applied/selected/optimized these proposing algorithmic strategies to improve AES system throughput with minimal area.

REFFERENCE

[1] I. Watson. "Security portable storage devices, Network Security." , IBM Security Intelligence., Vol. 7, 2006. Pp. 8–11.

[2] J. A. Halderman, S. D. Schoen, H. Nadia, C. William, P. William , J. A. Calandrino, A. J. Feldman, A. Jacob and E W. Felten. "Lest We Remember: Cold Boot Attacks on Encryption Keys.", 17th USENIX Security Symposium., August, 2008. Pp. 45 – 60.

[3] A. Dandalis, V. K. Prasanna and J. D. Rolim. "A comparative study of Performance of AES final candidates using FPGAs.",Cryptographic Hardware and Embedded Systems Workshop., vol. 1965, 2000. Pp. 125–140.

[4] A. J. Elbirt, W. Yip, B. Chetwynd and C. Paar. "An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists." , In Proc. The Third Advanced Encryption Standard Candidate Conference (AES3), New York, USA, April, 2000. Pp. 13–27.

[5] K. Gaj and P. Chodowiec. "Comparison of the hardware performance of the AES candidates using reconfigurable hardware.", In Proc. Third Advanced Encryption Standard Candidate Conference (AES3), New York, USA, April, 2000. Pp. 40–54.

[6] T. Ichikawa, T. Kasuya and M. Matsui. "Hardware evaluation of the AES finalists." , In Proc. Third Advanced Encryption Standard Candidate Conference (AES3), New York, USA, April, 2000. Pp. 279–285.

[7] B.Weeks, M. Bean, T. Rozylowicz and C. Ficke. "Hardware performance simulations of Round 2 Advanced Encryption Standard algorithms.", In Proc. Third Advanced Encryption Standard Candidate Conference (AES3), New York, USA, April, 2000. Pp. 286–304.

[8] K. Gaj and P. Chodowiec. "Hardware performance of the AES finalists-survey and analysis results." [Online]. Available : http://ece.gmu.edu/crypto/AES survey.pdf.

[9] I. Kuon and J.Rose. "Measuring the gap between FPGAs and ASICs.", IEEE Trans. Computer-Aided Design, February, 2007. Pp. 203–215.

[10] Cetin Kaya Koc, Editor. Cryptographic Engineering. New York: Springer Science + business media. 2009. Pp. 235–318.

[11] A. A. Mohammad. "A study on encryption algorithms and modes for disk encryption.", International conference on signal processing system, 2009. Pp. 793 – 797.

[12] K. U. Jarvinen, M. T. Tommiska and J. O. Skytta. "A fully pipelined memoryless 17.8 Gbps AES-128 encryptor.", In Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2003), Monterey, CA, February, 2003. Pp. 207–215.

[13] S. Morioka and A. Satoh. "A 10 Gbps full-AES crypto design with a twisted-BDD S-Box architecture.", In Proc. IEEE International Conference on Computer Design : VLSI in Computers and Processors, Freiburg, Germany, 2002. Pp. 98 –103.

[14] A. Satoh, S. Morioka, K. Takano and S. Munetoh. "A compact Rijndael hardware architecture with S-Box optimization.", In Proc. Theory and Application of Cryptology and Information Security (ASIACRYPT'01), vol. 2248, 2001. Pp. 239–254.

[15] T. Good and M. Benaissa. "AES FPGA from the fastest to the smallest.", Proc. International Workshop on Cryptographic Hardware and Embedded System (CHES'05), vol. 3659, 2005. Pp. 427–440.

[16] D. Canright. "A very compact S-box for AES." , Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES'05), vol. 3659, 2005. Pp. 441–455.

[17] N. Mentens, L. Batina, B. Preneel and I. Verbauwhede. "A systematic evaluation of compact hardware implementations for the Rijndael S-box.", Proc. (CT-RSA '05), vol. 3376, 2005. Pp. 323–333.

[18] V. Fischer, M. Drutarovsky, P. Chodowiec and F. Gramain. "InvMixColumn decomposition and multilevel resource sharing in Rijndael implementation.", IEEE Transactions on VLSI Systems, 2005. Pp.989–992.

[19] Helion. "Documentation of cryptographic cores." [Online]. Available : http://www.heliontech.com.

[20] K. Araki, I. Fujita and M. Morisue. "Fast Inverter Over Finite Fields Based on Euclid's Algorithm.", Trans. IEICE, November, vol. E-72, 2009. Pp. 1230–1234.

[21] H. Brunner, A. Curiger and M. Hofstetter. "On Computing Multiplicative Inverse in $GF(2^m)$.", IEEE Trans. on Computer, August, vol. 42., 1993. Pp. 1010–1015.

[22] C. Jutla, V. Kumar and A. Rudra. "On the Complexity of Isomorphic Galois Field Transforms.", IBM Research Report, Vol. RC22652 (W0211-243), November, 2002. Pp. 1-14.

[23] A. Rudra, P K. Dubey , C. Jutla , V. Kumar, J R. Rao and P. Rohatgi. "Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic.", Proc. CHES 2001, Vol. 2162, 2001. Pp. 175-188.

[24] C. Paar. "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields.", IEEE Trans. on Comp, vol. 45, 1996. Pp. 856–861.

[25] Sumio Morioka and Akashi Satoh. "An Optimized S-Box Circuit Architecture for Low Power AES Design.", Tokyo Research Laboratory, Japan, vol.2523, 2002. Pp. 172-186.

[26] Khanob Thongkhome, Chalermwat Thanavijitpun and Somsak Choomchuay. "An Implementation of S-Box for a Compact AES System.", Proc. of 25th Int. Conf. on Circuits/Systems, Computers, and Communications (ITC-CSCC2010), Pattaya, Thailand, July, 2010. Pp. 911-915.

[27] Chalermwat Thanavijitpun, Khanob Thongkhome and Somsak Choomchuay. "FPGA Implementation of FDE-Portable hard disk System.", The Int. Conf. on Information and Communication Technology for Embedded Systems, Pattaya, Thailand, January, 2011. Pp. 189-191.

[28] S. Chantarawong, P. Noo-intara and S. Choomchuay. "An Architecture for S-Box Computation in the AES.", Proc of Information and Computer Engineering Workshop 2004 (ICEP2004), January, 2004. Pp.157-162.

[29] P. Noo-intara, S. Chantarawong and S. Choomchuay. "Architecture for MixColumn Transform for the AES.", Proc of Information and Computer Engineering Workshop 2004 (ICEP2004), January, 2004. Pp.152-156.

[30] Alireza Hodjat and Ingrid Verbauwhede. "Minimum area cost for a 30 to 70 Gbits/s AES processor.", Proc. of the 2004 IEEE Computer society Annual Symposium on VLSI, February, 2004. Pp. 83-88.

[31] Francisco Rodriguez-Henriquez, N.A. Saqib and Díaz Pérez, Editors. Cryptographic Algorithms on Reconfigurable Hardware. New York: Springer business media, 2009. Pp. 259.

[32] N. A. Saquib, F. Rodriguez-Henriquez and A. Diaz-Perez. "AES Algorithm Implementation - An Efficient Approach for Sequential and Pipeline Architectures.", In Fourth Mexican International Conference on Computer

Science, Tlaxcala, Mexico, September, 2003. Pp.126-130.

[33] Pravin B. Ghewari, Jaymala K. Patil and Amit B. Chougule. "Efficient Hardware Design and Implementation of AES Cryptosystem.", International Journal of Engineering Science and Technology, Vol. 2(3), 2010. Pp. 213-219.

[34] E. Lopez-Trejo, F. Rodriguez Henriquez and A. Diaz-Perez. "An Efficient FPGA Implementation of CCM Mode Using AES.", In International Conference on Information Security and Cryptology, Seoul, Korea, vol. 3935, 2005. Pp. 208-215.

[35] Samir El Adib and *Naoufal Raissoun*. "AES Encryption Algorithm Hardware Implementation : Throughput and Area Comparison of 128, 192 and 256-bits Key.", International Journal of Reconfigurable and Embedded Systems (IJRES), Vol. 1, July, 2012. Pp. 67-74.

[36] Z. Alaoui-Ismaili. "Flexible Hardware Architecture for AES Cryptography Algorithm.", Multimedia Computing and Systems ICMCS'09, April, 2009. Pp. 438-442.

[37] G. J. Calderon, J. Velasco-Medina and J. Lopez-Hernandez. "Implementacion en Hardware del Algoritmo Rijndael.", In *X* Workshop IBERCHIP, 2004. Pp. 113.

[38] H.Anthony, S. Ripduman, R.Andrew and J.Brian. "An analysis of hard drive energy consumption .", IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. Pp. 1-10.

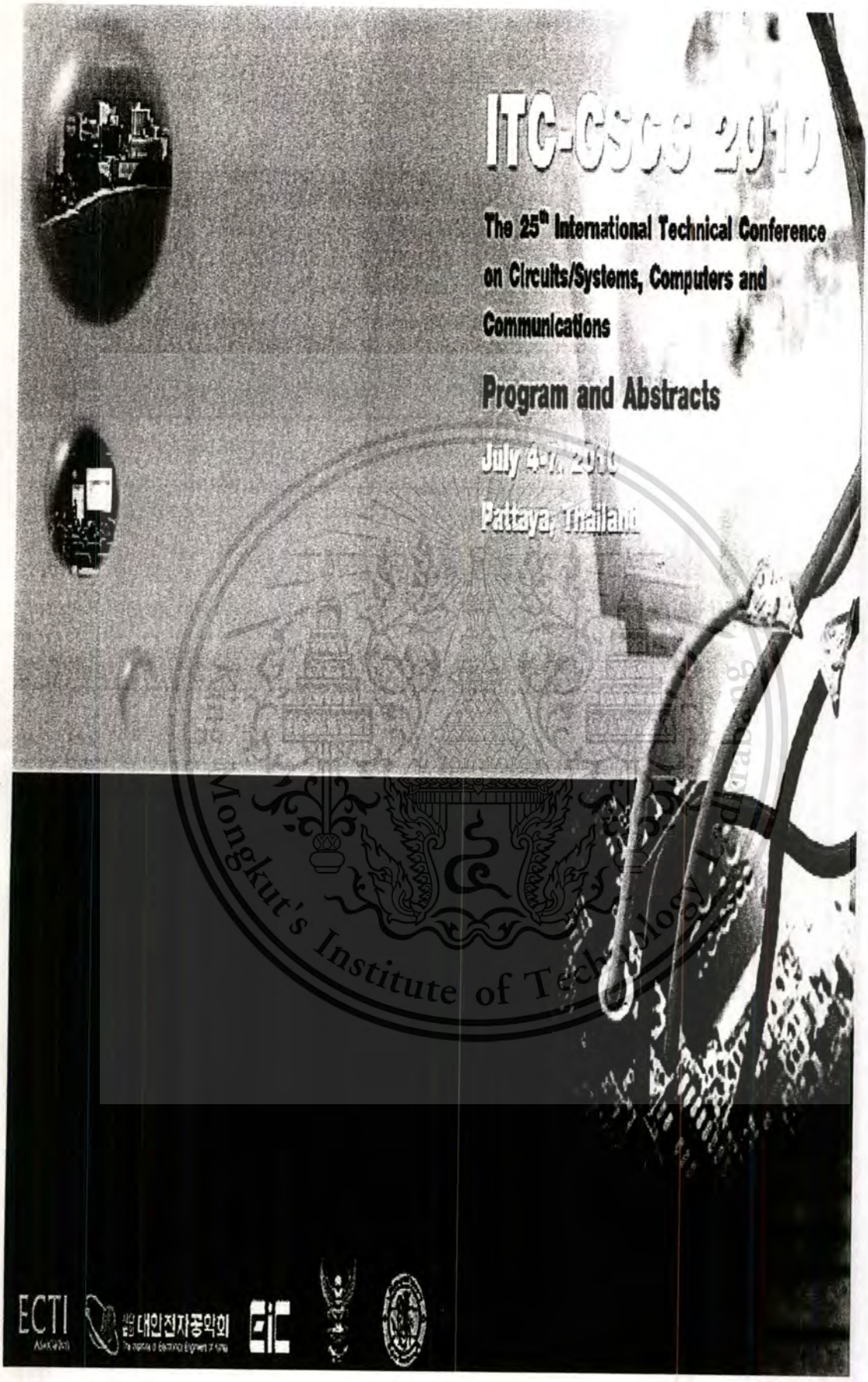
Appendix A

Publication List

[1] Khanob Thongkhome, Chalermwat Thanavijitpun, and Somsak Choomchuay, "An Implementation of S-box for a Compact AES System," Proc. of 25th Int. Conf. on Circuits/Systems, Computers, and Communications (ITC-CSCC2010), Pattaya, Thailand, July 2010.

[2] Chalermwat Thanavijitpun, Khanob Thongkhome, and Somsak Choomchuay, "FPGA Implementation of FDE-Portable hard Disk System," The Int. Conf. on Information and Communication Technology for Embedded Systems (IC-ICTES2011), Pattaya, Thailand, January, 2011

[3] Khanob Thongkhome, Chalermwat Thanavijitpun, and Somsak Choomchuay, "A FPGA Design of AES Core Architecture for Portable Hard Disk" Proc. of 8th Int. Joint Conf. on Computer Science and Software Engineering (JCSSE-2011), Nakhon Pathom, Thailand, May, 2011.



AN IMPLEMENTATION OF S-BOX FOR A COMPACT AES SYSTEM

Khanob Thongkhome[1] , Chalermwat Thanavijitpun[2], and Somsak Choomchuay[3]

[1]College of Data Storage Technology and Applications, King Mongkut's Institute of Technology Ladkrabang, BKK, Thailand
Tel:/Fax: + 66-3-881-5966, E-mail: raksky.thongkhome@gmail.com
[2]College of Data Storage Technology and Applications, King Mongkut's Institute of Technology Ladkrabang, BKK, Thailand
Tel:/Fax: + 66-2-326-4731, E-mail: chalermwat.thanavijitpun@seagate.com
[3]Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, BKK 10520, Thailand
Tel: +66-2-326-4222 Ext.114, Fax: +66-2-739-2398, E-mail: kchsomsa@kmitl.ac.th

ABSTRACT

This paper presents the implementation of byte substitution (S-Box) layer of an AES system. The implementation is based on composite field technique. To support the designed S-Box, AES core as well as KeyScheduling unit were also designed; however with loose optimization. The S-Box is also designed with a number of pipeline stage options. This enable the system throughput customization. FPGA validation has confirmed to us the promise of the whole AES system design.

Index Terms— AES, S-Box, Composite field, FPGA

1. INTRODUCTION

AES is a cryptographic algorithm, developed by Joan Daemen and Vincent Rijmen and was chosen by NIST from many AES candidates algorithm as an Advanced Encryption Standard for protecting the electronic data to replaced DES (Data Encryption Standard) algorithm [1], [2]. Since then the AES algorithm has been widely implemented. Many implementations are done in software. This approach seems to be too slow for fast applications such as routers, gateways and some wireless communication systems. It is also vulnerable to attacks. In contrast, in the pure hardware implementation, the higher data rate (Gbits/second) could be obtained by parallelization and pipelining [3], [4], [5]. The implementations are physically secure since tempering by an outside attacker is difficult. It's also a cost-effective solution for many application specific systems. The AES IP cores are also available commercially in the ASIC and FPGAs [6], [7].

AES is a 128 bit symmetric data block cipher with 128, 192 or 256 bits key lengths that applied mathematical description over the Galois finite field $GF(2^8)$ .The data block was described by arrays of bytes in 4×4 matrix (Called "State") and it has four basic steps operation; SubBytes, (or S-Box), ShiftRow, MixColumn, and AddRoundKey. These four steps are also known as layers. The four layer steps describe one round of the AES. Number of rounds is made vary according to the key size. The AES with 128-bit key size operates iteratively on those four basic steps for 10 rounds. However, the first and the final rounds are arranged in a slightly different manner compared to others. All four layers have their corresponding inverse operations. The deciphering is, therefore, the reverse order of the ciphering process. Operation steps are similar and at the comparable complexity. Moreover, both processes can share same set of designed hardware. In particular, the S-box computation that occupies the major chip area since it is required in SubBytes transform and in KeyScheduling processes. Many attempts are trying to obtain the compact S-Box [8], [9], [10], [11]. Rather than designing a fast and area consuming AES, many approaches are aiming to low power applications [9], [10], [12].

The rest of this paper is structured as follows: section 2 is delegated to SubBytes transform; Its implementation in the composite field is reviewed in section 3; In section 4 S-Box architecture is discussed whilst section 5 is devoted to the AES system data path. FPGA investigation of the designed S-Box is detailed in section 6 before the work conclusions given in section 7.

2. S-BOX TRANSFORMATION

The S-Box operation is a non-linear byte substitution. It composes of two sub- transformations; multiplicative inverse and affine transformation.

1) Multiplicative inverse of each byte is taken – this stage is to compute $B(x)=A^{-1}(x)$ for an 8-bit input word (in $GF(2^8)$ where $m(x)=x^8+x^4+x^3+x+1$ is taken as a field polynomial; {00} is mapped to itself).

2) Affine Transformation: This sub-step is performed in $GF(2)$ and defined by.

$$D(x)=\delta B(x)_{\mathrm{mod}(x^8+1)}\oplus C(x) \qquad (1)$$

where $\delta=\{1F\}\ =x^4+x^3\ +x^2+x+1$ for the encryption process and $\delta=\{4A\}\ =x^6+x^3+x$ for the decryption. The constant $C(x)$ has been added in order that the S-box has no fixed point (a map to a), and no opposite fixed point (a map to $\overline{a}$). $C(x)=\{63\}\ =x^6+x^5+x+1$ for encryption, and $C(x)=\{05\}=x^2+1$ for S-box inversion. Let $B(x)=A^{-1}(x)$ then eqn. (1) can also be rewritten in the matrix form as given in eqn. (2).

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (2)$$

The required addition of {63} can be made vanish by incorporating it into a modified KeyScheduling [13]. In many AES implementations two sub-steps required in the BytesSub transformation are typically combined into a single lookup table. The table size is 16 by 16 with the content 8 bits in length. The ROM size of 256×8 bit is not big for current technology and can be implemented in a fairly simple manner with modern design tools. However when the area is restricted or a ROM cannot be incorporated, the inversion hardware becomes necessary. Within this scenario, the efficient S-box implementation is the major concern. The affine transform, however requires small number of gates and introduces small delays.

3. S-BOX IMPLEMENTATION

Defined by Rijndael, AES has adopted $m(x)=x^8+x^4+x^3+x+1$ as its field polynomial. Although such a polynomial is an irreducible one but it is not a primitive polynomial. The computation of $B(x)=A^{-1}(x)$ is considered to be intricate by most authors and design engineers. Several techniques for S-box computation have been developed. These are, for instances; (1) The mentioned above table look up where step 2 is usually combined to be a single table, (2) Synthesis and optimized logic function of S-box using CAD tools, and (3) Compute the inversion of element in $GF(2^8)$ and optimize the logic functions. In the computation of element inversion in $GF(2^k)$ one can use either extended Euclid algorithm [14], [15] or composite field technique [16], [17], [18], [19]. Rudra *et al.* [16], [17] mapped all the operation (except ShiftRow) into the composite field of $GF(2^4)^2$. Morioka and Satoh [10] also have exploited the used of composite field in the design of a low power S-box transform. Elements in $GF(2^8)$ are mapped to those defined in $GF(2^4)^2$. Multiplication and inversion are optimized in the ground field. Our approach has drawn many useful ideas reported in [9], [16] and [18]. To reduce the unnecessary overhead, field transform is applied to the S-box computation only. It is not necessary to further breakdown the composite field to the lowest ground field.

Working in the composite field, multiplicative inverse is leisured. However, forth and back, we have to map elements in $GF(2^k)$ into $GF(2^n)^m$ where $k=mn$. Therefore both transform and inverse transform matrices are needed. Elements in $GF(2^8)$ can be mapped to element in $GF(2^4)^2$ by using the polynomial $r(x)=x^2+x+\beta^{14}$ where β^{14} denotes the element in $GF(2^4)$ of which $I(x)=x^4+x+1$ is the primitive irreducible polynomial. The resulted mapping and inverse mapping matrices are given in eqn. (3) and (4) respectively

$$T = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (3)$$

and

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (4)$$

The upper–left element in the above matrices denotes the least significant bit. An advantage of mapping elements form $GF(2^8)$ to $GF((2^4)^2)$ is the simpler multiplicative inverse computation since inversion is performed in $GF(2^4)$. For such a small field size, inversion using either the direct truth-table mapping or table look up consumes small area. Moreover, in Rijndael system data are treated naturally in byte format. Let data (byte) be expressed as $A=\{bc\}=bx+c$, the inversion of A, say $B=A^{-1}=\{pq\}=px+q$. For the field polynomial $r(x)=x^2+Cx+D$, one can have

$$p=b\Delta^{-1} \quad (5)$$

$$q=(Cb\oplus c)\Delta^{-1} \quad (6)$$

where

$$\Delta=c(Cb\oplus c)\oplus b^2D. \quad (7)$$

or

$$\Delta=bcC\oplus c^2\oplus b^2D \quad (8)$$

For $GF((2^n)^2)$, the polynomial in the form of $r(x)=x^2+x+\lambda$ always exists [18]. As such, C and D can be

set to {1} and {9} (in $GF(2^4)$) respectively. Fixed-coefficient multiplication (i.e., b^2D) as well as squaring units are relatively simple according to their small field size. The multiplications required in computing (5), (6) and (7) can be done straight away in $GF(2^4)$ or can be further simplified by making use of composite field $GF((2^2)^2)$ [9].

4. Architecture

For an efficient usage of hardware, the SubByte (or S-Box) module can be shared among three processes, namely; encryption, decryption, and key scheduling. Moreover, the S-box operation seems to be the complex computation in the AES system. Shown in Fig.1, the S-Box sharing can be performed via data multiplexing. KeyScheduking and Encryption process use the same operation – Forward SubByte transform where the inversion is followed by affine mapping. MUX_1 with the control of SEL_1 is taking care of this. When SEL_1=1, KeyScheduling data are fed into the Byte Substitution computation. Alternatively, when SEL_1=0, the ciphering (or deciphering) data are fed into the computation block. Deciphering operates reversely to the ciphering: inverse affine transform comes before inversion. MUX_2, MUX_3, and SEL_2 control data route. SEL_2=0 notes the deciphering operation.

5. Datapath Structure

AES can be design in either loop structure or loop-unrolled structure according to the data rate needed. Pipelined/sub-pipeline loop-unrolled can offer very high throughput by at a cost of very large hardware size. The loop structure with 32 bit data path can be the compact one but at consider low throughput. With these regards, a loop structure with 128 bit data path seems to be the compromised design. This paper elaborates this approach with some extension in number of pipeline state options for the S-Box design. Shown in Fig.2, the 128 bit data path is made up of 4 of 32 bit datapaths. Since one S-Box computation block takes only 8 bit data set, we do need 16 parallel units.

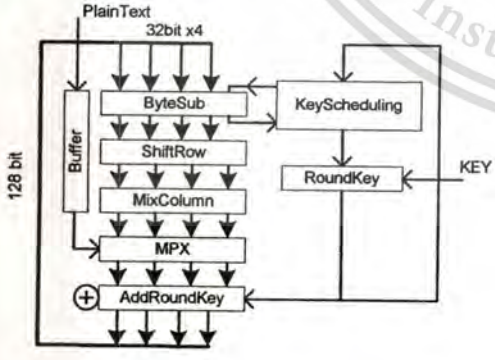


Figure 2 the designed AES data path (128 bit)

The byte substitution computation is also required by the KeyScheduling process in computing its round key. An 128 bit round key is need in every round of AES operation. However only 32 bit: – 4 of 8 bit needs byte substitution. Therefore only 4 of 16 S-Box are shared to key update computation.

6. FPGA Implementation

For the validation of previous described S-Box, the S-Box itself as well as other necessary circuit were designed using VHDL. The code is pure VHDL that could easily be implemented on other FPGA devices, without changing the design. Synthesis and Place & Route were achieved on Xilinx ISE 8.1i with Xilinx's Spartan-3E XC3S400. This is used for writing, debugging and optimizing efforts, and also for fitting, simulating.

A single module S-Box can be implemented directly by referring to eqn.(3) through eqn.(8). Most subcircuits were taken from [4]. As the S-Box will be massively paralleled to process 128 bytes data in few clock cycles, this part of the system must be carefully designed. The composite filed inversion used in this paper composes of few $GF(2^4)$ adders, multipliers, and a field inversion circuit. Complexity detail is given in Table I below. Resource utilization as well as the obtained performances are given in Table II.

TABLE I: Hardware complexity of a $GF(2^4)^2$ inversion

Component List	Complexity*			Unit required
	AND	XOR	τ	
Squaring	-	2	1	2
Fixed-Coeff. Mul.	-	1	1	1
Inversion	11	11	3	1
Multiplication	16	10	3	3

* 1) 2 input gate and 3 input gate are equivalent
2) Gate delay of 1, 2 and 3 input are assumed to be equal

Affine and inverse affine transform circuits are fairly simple since they contain only small number of XOR gates. The forward affine transform can be implanted directly using eqn.(2). Similarly the inverse affine transform can be done. The delay time of those circuits are 3τ and 2τ respectively. Here τ is a unit delay of a single gate (less than 3 input).

For a 32 bit data path, 4 parallel S-Box are required. In total, 16 S-Box circuits (including affine transforms) are needed. To facilitate the AES system to work, we also need other parts of the system; mainly the KeyScheduling circuit. The AES core performs standard AES routines – such as SubByte, AddRoundKey, ShiftRow, and MixColumn. Resource utilization of AES core (S-Box included) is given in Table III. Be noted that the core is not yet nicely turned to match the S-Box performance, the throughput is therefore not yet maximized. S-box function verification is shown in Fig. 3.

TABLE III: AES Core Hardware FPGA Utilization

Utilization Summary	Designed S-Box + AES Core system		
	Used	Available	Utilization
Number of Slices	1040	3584	29%
Number of Slice Flip Flops	914	7618	12%
Number of 4 input LUTs	1910	7618	25%
Number of IOs	302		
Number of bonded IOBs	302	178	170%
Number of BRAMs	2	20	10%
Maximun Priod	26.933ns (Maximum Frequency: 64.283 MHz)		
Throughput	976 Mbit/s		

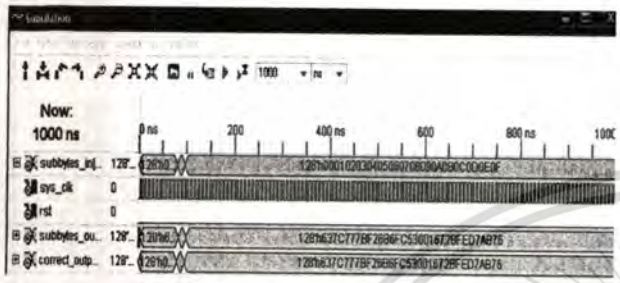


Figure 3 S-Box function verification

For higher throughput the S-box can be sub-pipelined. Illustrated in Fig. 4, the S-box can be sub-pipelined to accommodate the throughput requirement of the system. However the maximum delay between stages can be made down to as the lowest. This is limited by the delay of the $GF(2^4)$ inversion circuit as well as a multiplier. Pipeline option is given in Table IV. Upon the preliminary investigation, the obtained throughput is given in the last column. Obviously increasing the number of pipeline state boosts the throughput; however at the penalty of increasing the circuit size as well as power consumption.

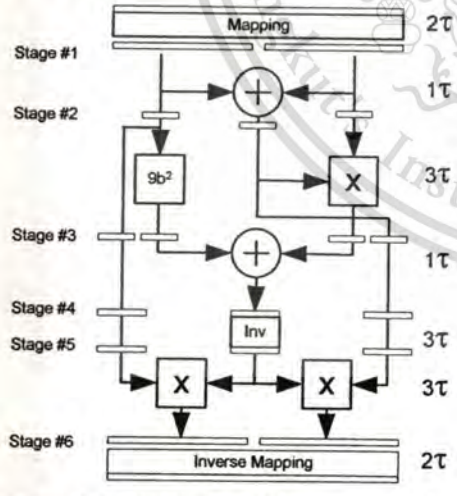


Figure 4 Sub-pipelining of the inversion circuit

TABLE IV: PIPELINING SCHEME OF THE INVERSION CIRCUIT

No. of Pipeline stage	Stages used	Max. delay between stages	Throughput (Gbits/sec)
-	-	15τ	1.12
2	#4	8τ	1.87
3	#2, #4, #5	5τ	2.31
4	#2, #4, #5, #6	4τ	2.76
5	#2 - #5	3τ	2.93

7. CONCLUSIONS

Data privacy and security is of more interest and became vital in this decade of information. AES is one of the widely accept encryption and decryption standard. In this paper, we presented a S-box implementation based on the mathematics of composite field. The proposed sub-pipeline options can be chosen upon speed or throughput requirement. A non-pipelined S-Box can give the throughput of more than 1 Gbits/sec as high. The designed S-Box is also shared by a KeyScheduling process. The internal data path is 128 bit wide. At this width, 16 S-Box are required. A synthesizable VHDL code is developed for the implementation of both encryption and decryption process. The design is verified via the FPGA implementation; Xilinx Spartan 3E. The high level languages (C and Matlab) have confirmed the validity and correctness of the algorithm.

8. REFERENCES

[1] J. Daemen and V. Rijmen, AES Proposal: Rijndael (Version 2). NIST AES Website; http://csrc.nist.gov/publications/ and http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf

[2] NIST Federal Information Processing Standards (FIPS PUB 197) Advanced Encryption Standard (2001, Nov.). [Online]. Available:http://www.nist.gov/aes

[3] I.M. Verbauwhede, P.R. Schaumont, and, H. Kuo, "Deign and Performance Testing of A 2.29 Gb/s Rijndael Processor," *IEEE J. of Solid State-Circuit*, Vol. 38, No. 3, March 2003, pp. 569 – 572.

[4] S. Chantarawong, P. Noo-intara, and S. Choomchuay, "An Architecture for S-Box Computation in the AES," Proc of Information and Computer Engineering Workshop 2004 (ICEP2004), January 2004, pp.157-162.

[5] Alireza Hodjat, and Ingrid Verbauwhede, "Minimum area cost for a 30 to 70 Gbits/s AES processor," in Proc. of the 2004 IEEE Computer society Annual Symposium on VLSI, pp. 83-88, Feb. 2004.

[6] CAST, Advanced Encryption Standard Core, available at; http://www.cast-inc.com/cores/aes/index.shtml.

[7] IP Cores, Ultra-Compact, Advanced Encryption Standard Core, available at; http://www.ipcores.com/AES1.pdf.

[8] Akashi Satoh et.al., "A Compact Rijndael Hardware Architecture with S-Box Optimization", Tokyo Research Laboratory, IBM Japan Ltd.,2001

[9] S. Morioka and A. Satoh, "An Optimized S-box Circuit Architecture for Low Power AES Design," *Proc. CHES 2002*, LNCS Vol. 2523, pp. 172-186, 2003.

[10] Sumio Morioka, Akashi Satoh," An Optimized S-Box Circuit Architecture for Low Power AES Design "Tokyo Research Laboratory, IBM Japan Ltd.,2003.

[11] D.Canright, "A Very Compact Rijndael S-Box," Naval Postgraduate School Monterey CA Dept. of Mathematics,May 2005.

[12] Namn Yu et.al.," Investigation of Compact Hardware Implementation of the Advanced Encryption Standard", IEEE CCECE/CCGEI, Saskatoon, May 2005.

[13] S. Murphy and M.J.B. Robshaw, "Essential Algebraic Structure Within the AES," Proc. CRYPTO 2002, Vol. 2442, Springer-Verlag, pp. 1-16, 2002.

[14] K. Araki, I. Fujita and M. Morisue, "Fast Inverter Over Finite Fields Based on Euclid's Algorithm," *Trans. IEICE*, Vol. E-72, No. 11, pp. 1230–1234, Nov. 1989.

[15] H. Brunner, A. Curiger, and M. Hofstetter, "On Computing Multiplicative Inverse in GF(2^m) ," *IEEE Trans. on Computer*, Vol. 42., No. 8, pp. 1010–1015, Aug. 1993.

[16] A. Rudra et. al., "Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic," *Proc. CHES 2001*, LNCS Vol. 2162, pp. 175-188, 2001.

[17] C. Jutla, V. Kumar and A. Rudra, "On the Complexity of Isomorphic Galois Field Transforms," *IBM Research Report*, Vol. RC22652 (W0211-243), November 2002.

[18] C. Paar, "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields," *IEEE Trans. on Comp.*, Vol. 45, No. 7, pp 856–861, 1996.

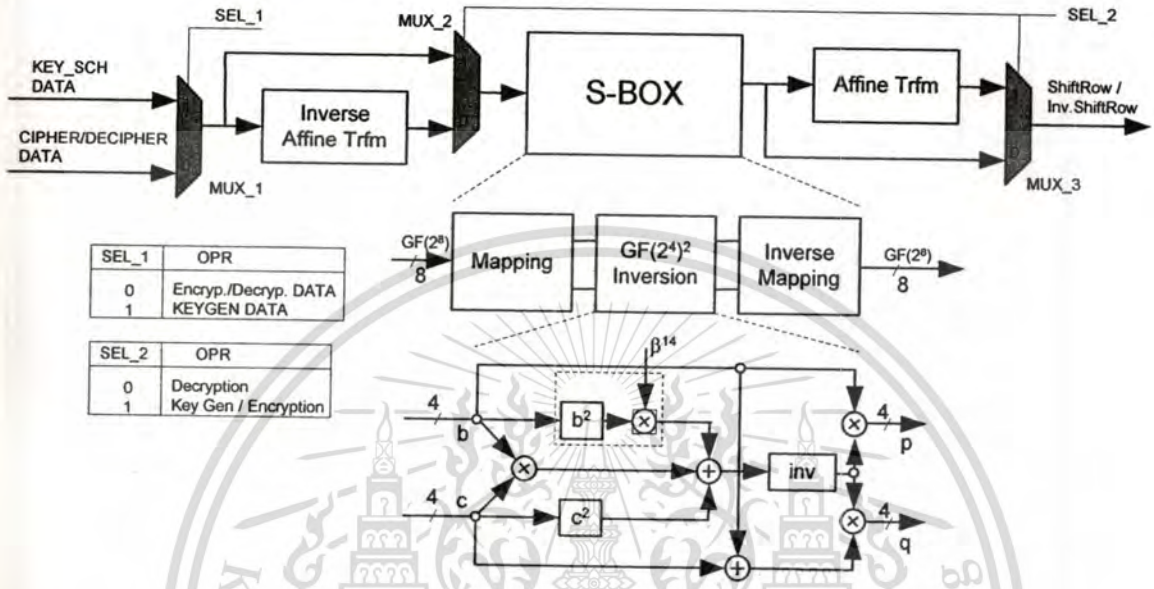
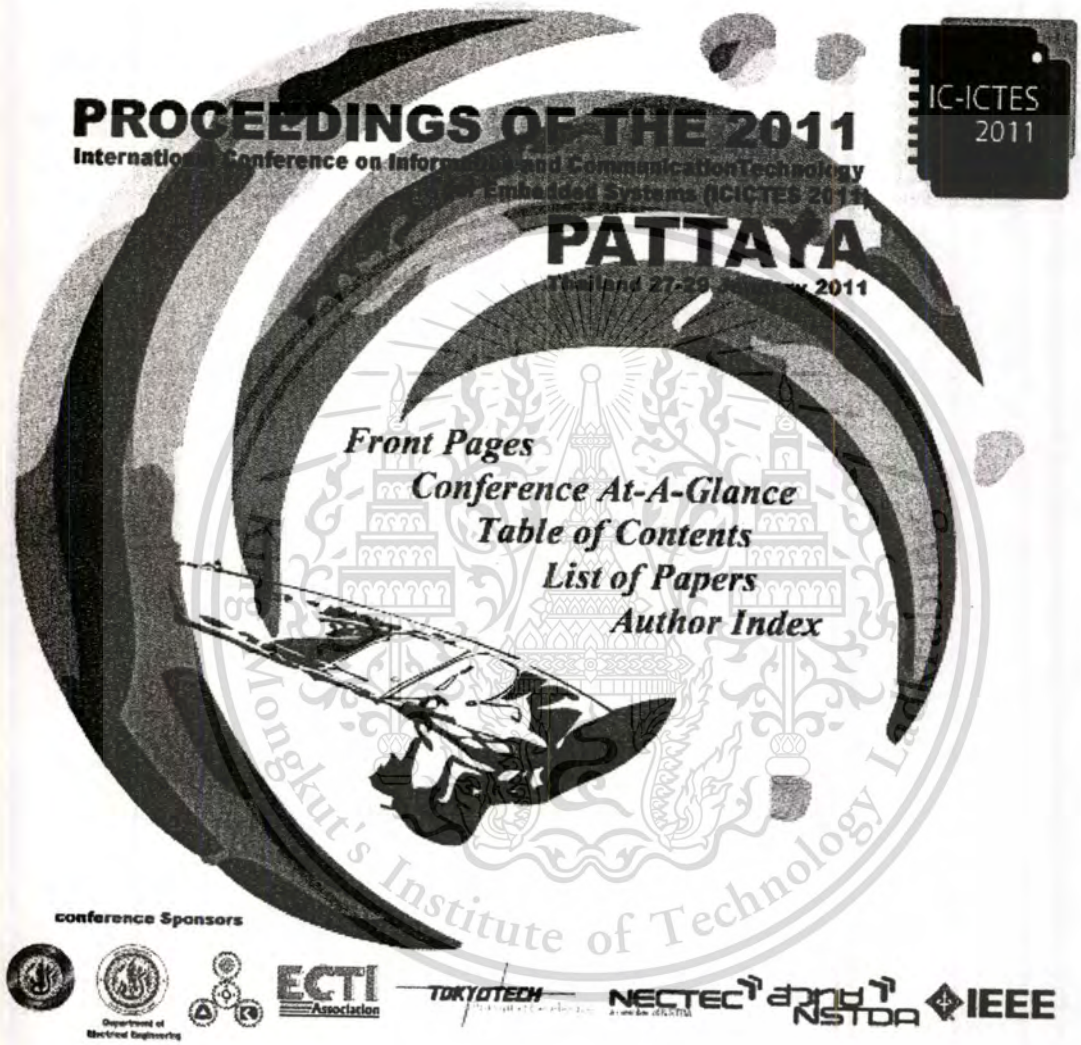


Fig.1 An architecture of S-Box (SubByte Transform)

TABLE II UTILIZATION SUMMARY OF A DESIGNED SINGLE S-BOX (NO SUB-PIPELINE).

Utilization Summary	Designed - No Pipeline S-Box		
	Used	Available	Utilization
Number of Slices	55	3584	1%
Number of Slice Flip Flops	16	7168	0%
Number of 4 input LUTs	103	7168	1%
Number of bonded IOBs	18	97	18%
Number of GCLKs	1	8	12%
Minimum period	13.637ns (Maximum Frequency73.330 MHz)		
Minimum input arrival time before clock	4.454ns		
Maximum output required time after clock	7.165ns		
Maximum combinational path delay	No path found		
Throughput	1.12 Gbit/s		



FPGA IMPLEMENTATION OF FDE-PORTABLE HARD DISK SYSTEM

Chalermwat Thanavijitpun[1], Khanob Thongkhome[2], and Somsak Choomchuay[3]

[1,2] College of Data Storage Technology and Applications, King Mongkut's Institute of Technology Ladkrabang, BKK, Thailand
Tel: + 66-2-329-8263, [1] E-mail: chalermwat.thanavijitpun@seagate.com, [2] Email: raksky.thongkhome@gmail.com
[3]Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, BKK 10520, Thailand
Tel: +66-2-329-8344-5 Ext.114, Fax: +66-2-329-8346, E-mail: kchsomsa@kmitl.ac.th

ABSTRACT

Regarding the well growing of data security and privacy, the total-secured data storages are drawing more interest. This paper presents the implementation of FDE portable hard drive that includes AES in its data encryption/decryption process. Rather than using ordinary password in the authentication process, our design incorporates a technique of fingerprint authentication. It is simple and user friendly but offers good security. The FDE hardware is inserted between the USB port and an ATA hard drive. It complies with USB 2.0 data rate (480 Mbit/sec). ATA protocol command decoder and AES core are implemented with FPGA whilst USB interface controller and fingerprint scanner are served by custom chips. The obtained simulate result of the completed AES core has confirmed us the functionality of the whole system.

Index Terms— Full Disk Encryption, FDE, AES, FPGA

1. INTRODUCTION

As data privacy issue is more aware, the importance of organization of confidential data stored on hard drives in PCs and/or servers cannot be no more ignored. Unfortunately, only few portable hard disk drives nowadays do include data security system/function to protect information that may be critical for individuals and mandatory for business. There could be various security threats, such as the malicious data modification, data leaks, and lost hard-disk. Events may cause inestimable loss to some organizations, such as military, governments, and enterprises. Regarding the secured data importance, several risk protection schemes have been implemented. Those may vary in terms of security strength, user friendliness, and cost of implementation.

BIOS and operating system passwords are commonly used but the efforts can offer very limited security. They can be easily removed even by the un-expert attackers. Hard drive protection password is more difficult to crack or remove but the security strength is still not so strong. Full disk encryption (FDE or whole disk encryption) uses disk encryption software or hardware to encrypt every bit of data that goes through a disk or disk volume. The FDE is significantly stronger than the first two methods mentioned above. The security strength depends mainly on the strength of the cryptographic algorithm used.

The software-based FDE method uses the computer's CPU to perform encryption/decryption tasks. This method has shown some disadvantages: (1) The encryption/decryption software can be easily monitored by a Trojan program. (2) The instructions of the encryption/decryption are executed by the CPU. Obviously the operations consume more computer resources. (3) It is difficult to transfer the encryption/decryption software among different operating systems.

The hardware-based method uses specific designed chips to accelerate the encryption/decryption process. Despite the hardware complexity, this method needs less computer resources. The security of hardware-based hard disk encryption/decryption is higher than that of the software-based hard disk encryption/decryption.

Advance Encryption System (AES) accepted by NIST as FIP-197 in the year 2001 is a popular algorithm used in existing FDE systems. In both of software-based and the hardware-based encryption/decryption system, the key is basically stored in special sectors of the hard disk.

In this paper, we partially outline the hardware-base FDE implementation. According to the implementability and the corresponding feasibility validation, the design is targeted at FPGA realization. Despite our own know-how started from the scratch pad, we believe that our design is not much different from others'. We combine fingerprint verification with AES. Fingerprint method is used in authentication process. That process also generates key for AES encryption/decryption module. We employ AES with 128 bit length in both data and key. The use of fingerprint instead of normal type-in password is expected to be better in security strength since fingerprint is unique and individual biometric. However, the detail of fingerprint verification is omitted in this paper according to its lengthy details as well as its off-track of interest.

The rest of this paper is structured as follows: section 2 is devoted to the AES core system. Section 3 is the total overview of design or architecture of the portable hard disk encryption/decryption system. The FPGA implementation is given in section 4 together with evaluated results obtained partially. The work is finally concluded in section 5.

2. AES CORE

AES is a symmetric 128 bit data block cipher with 128, 192 or 256 bits key lengths. The algorithm is developed by

Joan Daemen and Vincent Rijmen and was adopted by NIST among several AES candidate algorithms as an Advanced Encryption Standard for protecting the electronic data to replace DES (Data Encryption Standard) algorithm [1], [2]. Of its widely implemented of that Rideal algorithm, many of those are done in software. Such an approach seems to be too slow for fast applications such as routers, gateways and some wireless communication systems. The uses of AES in FDE application do exist, however not widely published. Software-based AES is also vulnerable to attacks. In contrast, in the pure hardware implementation, the higher data rate (Gbits/second) could be obtained by parallelization and pipelining [3], [4], [5], [8]. The implementations are physically secure since tempering by an outside attacker is naturally difficult. It's also a cost-effective solution for many application specific systems. AES IP cores are also available commercially in the form of both ASIC and FPGAs [6], [7]. Although our design is individual, it has drawn many useful ideas reported in [3], [9] and [10].

The 128 bit AES (with 128 bit key length) requires 10 rounds operation. Each round consists of S-box transformation, shift row, mix column and add round key. The first and final rounds are organized slightly different of other rounds. Each round key is derived from its previous round key. The original key is 128 bit in length. This key is used for both encryption and decryption processes. Round key processing also requires S-box transformation. In practical implementation of the AES, loop structure and loop-unrolled architectures are of preference depends on hardware size or data rate requirement.

3. Architectural Design

The system architecture of the FDE portable hard drive is demonstrated in Fig. 1. The embedded designed system is to be installed in the external hard drive enclosure where the normal ATA drive resides. The system includes a USB interface part, AES core, and a fingerprint verification module. The USB interface made use of a custom chip available commercially. Besides the USB interfacing, the main feature of General Programmable Interface (GPIF) is also controlled by the C-51 microcontroller core. The USB interface communicates with a hard drive via ATA command decoder. At that stage we have inserted the data encryption/decryption capability to the data. The AES core relies on the key sequence provided by the authentication module where the fingerprint is scanned and matched to the stored fingerprint database. The fingerprint scanner and matching is implemented on a custom chip. The 128 bit length cipher/decipher key is generated using authentication-passed user data together with other specific data set. ATA command decoder, AES core, and key control and key receiver modules are designed and implemented with FPGA. From the perspective of the hard drive unit the FPGA designed portion is regarded as the ATA protocol host controller. For the sage of validation of the designed FDE core together with some other necessary circuits, the FPGA were designed using VHDL. The pure VHDL code could be easily transferred among different FPGA devices, without changing the design. Synthesis and Place & Route were carried out on Xilinx ISE 8.1i with Xilinx's Spartan-3E XC3S400 device.

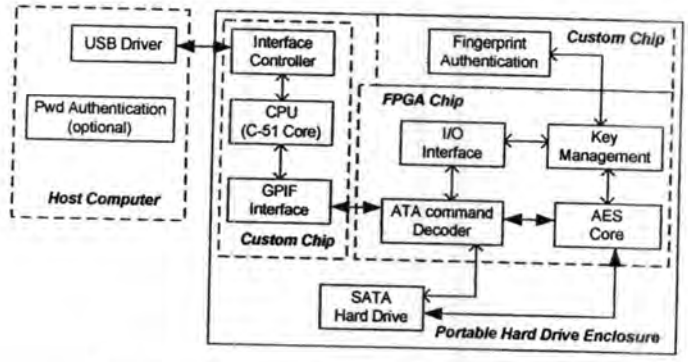


Fig. 1: FDE system for a portable hard drive

3.1 USB Interface Controller

USB interface controller bridges the FPGA and the host computer. Cypress EZ-USB FX2 chip has been adopted. GPIF functions for ATA Protocols such as PIO and UDMA are embedded. When the designed system is plugged into the host computer, EZ-USB FX2 enumerates automatically and downloads both firmware and USB descriptor tables. The host computer will identify EZ-USB FX2 as the development board of EZ-USB FX2. Then EZ-USB FX2 enumerates again as EZ-USB FX2 sample device. If the user passes the authentication check, EZ-USB FX2 enumerates again as the hard drive. If not, the hard disk cannot be remunerated and the hard drive is not recognized.

3.2 Data Encryption/Decryption Module (AES Core)

The data flow line drawn between host computer and hard drive is shown in Fig. 2 below. Down streamed data delivered by a host computer are encrypted before storing. Host computer sees this data path as a virtual SATA device. In contrast, up streamed data delivered by the drive unit must be decrypted before feeding to the computer. The hard drive sees this data path as a virtual host device.

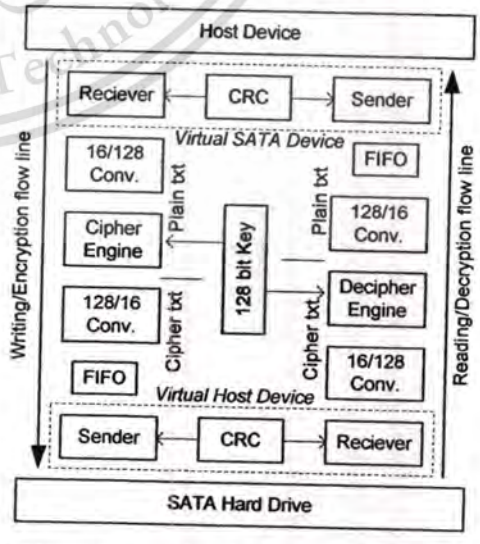


Fig. 2: Down streamed and up streamed data flow line

3.2.1 The Writing/Encryption Flow Line

The encryption flow line consists of a data receiver module, a 16-bit to 128-bit width conversion module, an AES encryption module, a 128-bit to 16-bit width conversion module, a FIFO module, and a data sender module.

The plain text data delivered by a host computer is encrypted before storing on a hard drive. In details; (1) The plain text data from the host device is received by the data receiver module; (2) The 16-bit width plain text is converted to the 128-bit width plain text; (3) The 128-bit width plain text is encrypted by the AES encryption module; (4) The 128-bit encrypted text is converted to the 16-bit width encrypted text; (5) The 16-bit width encrypted text is sent to a 16-bit width FIFO module; (6) The data sender module receives the encrypted text from the FIFO module and sends this encrypted text to store on the hard drive.

After the data receiver module receives the data from the host device, the plain text CRC checkout module will compare the calculated CRC value obtained from the host device with its own calculated value and will affirm that whether the data is transferred correctly or not. After the data is sent by the data sender module, the plain text CRC module will calculate the CRC value of the transferred data. The hard disk then will compare the calculated CRC value in the data sender module with its own calculated CRC value and will verify whether the data is transferred correctly or not. If both of the CRC Checksums are right, the data transfer is correct.

3.2.2 The Reading/Decryption Flow Line

The decryption flow line is basically operated in reverse direction to the encryption flow line. Similarly, the cryptograph (ciphered text or encrypted text) CRC checkout module will compute the CRC value of the data from the hard disk. Then the hard disk will compare the calculated CRC value in the cryptograph CRC checkout module with the calculated value in its own CRC calculation, and will affirm that whether the data is transferred correctly or not. After the data is sent by the data sender module, the plain text CRC checkout module will compare the calculated CRC value in the host device with that in the plain text CRC checkout module, and will verify whether the data is transferred correctly or not. If both of the CRC Checksums are right, the data transfer is correct.

3.2.3 The AES Encryption/Decryption Module (Cipher/Decipher Engine)

The AES encryption/decryption module work on 128 bit data block and in similar manner but in reverse direction. The module comprises similar operations which are: (1) BytesSub transformation or permutation. The operation is a non-linear byte substitution where multiplicative inverse and affine transformation are involved. (2) ShiftRow transformation is a linear diffusion process operates on individual row of a state. (3) MixColumn transformation is also a linear diffusion process. Each data column is multiplied by fixed matrix. And (4) AddRoundKey is the operation operates in a ground field of GF(2). As the engine must iterate for 10 rounds, the roundkey changes its value from round to round. As such, the key scheduling must be designed to cover this requirement. The addition in GF(2) is simple as XOR operation.

The above round operation appears in 2 folds: loop and loop-unrolled. In contrast to the loop-unrolled, the loop approach needs single block and repeatedly used for 10 rounds. The hardware size is minimized whilst the speed is not that fast compared to its competitor. According to our limited FPGA resources, we are staying on the loop architecture.

3.2.4 The Path width Conversion and the CRC Check

The AES engine works on128 bit data width while SATA devices rely on 16 bit data format. Basically, the back and forth path width conversions are needed. The implementation is obvious and straight forward.

CRC (Cyclic Redundant Check) is an efficient data error check method. It is a kind of sum checking. We used CRC-16 that can detect most error cases. Shift register implementation is fairly simple and very area efficient one. These circuit portions reside in both virtual host device and virtual SATA device.

4. FPGA Implementation and Prelim Results

Once the FDE system had committed its implementation, we designed the core system based on FPGA. Hardware description language (VHDL) is employed according to the flexibility in environment changing. The software used for this work is Xilinx - Project Navigator, ISE 8.1i suite. This is used for writing, debugging and optimizing efforts, and also for fitting, simulating and performance checking.

All the results are based on simulations from the ISE Test Bench. Individual transformation of both encryption and decryption are simulated using FPGA Spartan 3 families and Xilinx XC3S400 device. By far, only portions of the whole work had been done. We have done mostly the AES core. There are 3 main routines designed to realize the AES algorithm: encryption, decryption and key expansion routines. The complete VHDL code is organized with several modules. Basically, those modules implement the corresponding AES functions as described in previous section. Similar to others', our design criteria, are fastest, smallest and lowest power consumption.

The key expansion is calculated from the original 128 bit key supplied by the fingerprint authentication module. The key is supplied only the user has passed the verification process. In optional case, the verification process can be made by pass code typing. As shown in Fig. 3 above, the operational blocks are common for both encryption and decryption process. Fortunately, those processes are not working in the same times. Data paths are then can be managed for each purpose. The most complicate operation is ByteSub transformation. Computation made on the fly was selected for our design. The computation is performed in the composite field as details could be found in [10].

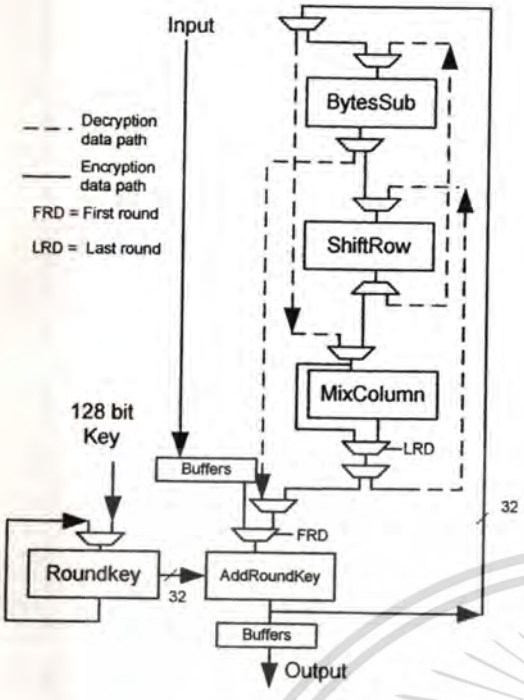


Fig. 3: AES Core (32 bit data path)

The design is synthesized and simulated for the above said Xilinx FPGA. The resource utilization is summarized in Table I below. The simulation to verify the function of encryption and decryption is made and the obtained result is illustrated in Fig. 4. Of the obtained data rate of 976 Mbit/sec, we believe that this is enough for USB 2.0 compliant even when the whole designs are integrated. The data rate can be made vary according to the design architecture. One can lower the data rate by either reducing the gate count or removing some internal pipeline stages.

TABLE I: AES CORE HARDWARE FPGA UTILIZATION

Utilization Summary	Designed S-Box+ AES Core system		
	Used	Available	Utilization
Number of Slices	1040	3584	29%
Number of Slice Flip Flops	914	7618	12%
Number of 4 input LUTs	1910	7618	25%
Number of IOs	302		
Number of bonded IOBs	302	178	170%
Number of BRAMs	2	20	10%
Maximun Priod	26.933ns (Maximum Frequency: 64.283 MHz)		
Throughput	976 Mbit/s		

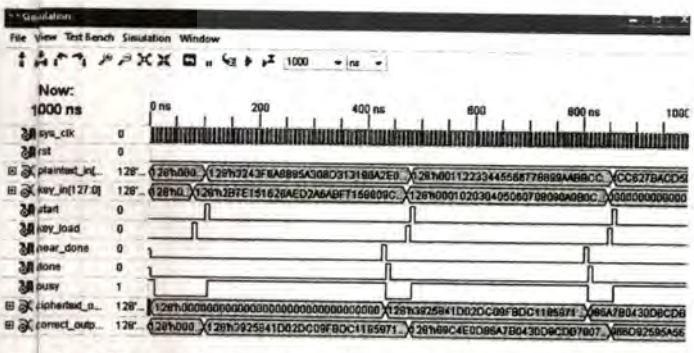


Fig.4: AES core timing simulation and function verification

5. CONCLUSIONS

In this paper, we have addressed the implementation of Full Disk Encryption (FDE) system designed for ATA external hard drive. The FDE is superior in term of data security compared to other methods. The system is enclosed in a tiny box together with a hard drive, connect to the host computer via the USB port. Fingerprint scanner as well as matching software is also integrated to the system for biometric authentication. Both commercial chips and our own FPGA are combined to serve full FDE function. Although the system is not yet fully implemented when this manuscript is prepared, the most completed AES core has convinced well the whole system functionality. The obtained throughout of 976 Mbit/sec is by far twice faster than that of USB 2.0 specification. The data rate can be fine-tuned by re-designing the hardware. As we are looking forward to the USB 3.0 complaint architecture, the current design has shown its fairly good potential. In comparison with the system proposed in [9] where AES was also evoked, our data rate is much faster. We used fingerprint identification rather than a password and MEM code lock. However, our architecture and system verifications are partially done and needed further development.

REFERENCES

[1] J. Daemen and V. Rijmen, AES Proposal: Rijndael (Version 2). NIST AES Website; http://csrc.nist.gov/publications/ and http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf

[2] NIST Federal Information Processing Standards (FIPS PUB 197) Advanced Encryption Standard (2001, Nov.). [Online]. Available:http://www.nist.gov/aes

[3] I.M. Verbauwhede, P.R. Schaumont, and, H. Kuo, "Deign and Performance Testing of A 2.29 Gb/s Rijndael Processor," *IEEE J. of Solid State-Circuit,* Vol. 38, No. 3, March 2003, pp. 569 – 572.

[4] S. Chantarawong, P. Noo-intara, and S. Choomchuay, "An Architecture for S-Box Computation in the AES," *Proc. of Information and Computer Engineering Workshop 2004 (ICEP2004)*, January 2004, pp.157-162.

[5] Alireza Hodjat, and Ingrid Verbauwhede, "Minimum area cost for a 30 to 70 Gbits/s AES processor," *Proc. of the 2004 IEEE Computer society Annual Symposium on VLSI*, pp. 83-88, Feb. 2004.

[6] CAST, Advanced Encryption Standard Core, available at; http://www.cast-inc.com/cores/aes/index.shtml.

[7] IP Cores, Ultra-Compact, Advanced Encryption Standard Core, available at; http://www.ipcores.com/AES1.pdf.

[8] Sumio Morioka, Akashi Satoh," An Optimized S-Box Circuit Architecture for Low Power AES Design," Tokyo Research Laboratory, IBM Japan Ltd.,2003.

[9] Weiping Zhang, Wenyuan Chen, Jian Tang, Peng Xu, Yibin Li and Shengyong Li, "The Development of a Portable Hard Disk Encryption/Decryption System with a MEMS Coded Lock" *Sensor 2009*, Pages 9300-9331.

[10] Khanob Thongkhome, Chalermwat Thanavijitpun, and Somsak Choomchuay, "An Implementation of S-Box for a Compact AES System," *Proc. of 25th Int. Conf. on*

A FPGA Design of AES Core Architecture for Portable Hard Disk

[1]Khanob Thongkhome, [2]Chalermwat Thanavijitpun
College of Data Storage Technology and Applications
King Mongkut's Institute of Technology Ladkrabang
Bangkok 10520, Thailand
[1]raksky.thongkhome@gmail.com
[2]chalermwat.thanavijitpun@seagate.com

Somsak Choomchuay
Dept. of Electronic Engineering, Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang
Bangkok 10520, Thailand
kchsomsa@kmitl.ac.th

***Abstract*—This paper describes a high effective AES core hardware architecture for implementing it to encrypt/decrypt the data in portable hard disk drive system that apply to effectively in the terms of speed, scale size and power consumption to comply with minimum speed of 5 Gbps (USB3.0). We proposed the 128 bits data path of two different AES architectures design, Basic Iterative AES, which reuses the same hardware for all the ten iterations and , One Stage Sub Pipelined AES, with one stage of outer pipelining in the data blocks that both of them are purely 128 bits data path architecture that different from the previous public paper. The implementation result on the targeted FPGA, the basic iterative AES encryption can offer the throughput of 3.85 Gbps at 300 MHz and one stage sub pipelined AES can offer the throughput to increase the efficiency of 6.2 Gbps at 481 MHz clock speed.**

Index Terms*— *AES, Encrypt/decrypt, USB3.0, FDE, ATM switch

I. INTRODUCTION

The advanced encryption standard(AES), standardized by NIST, National Institute of Standards and Technology, is a cryptographic algorithm replacement to DES (Data Encryption Standard) algorithm [1] ,[2] as the federal standard to protect sensitive information. AES has already received widespread use because of its high security, high performance in both hardware and software. Many implementations are done in software but it seems to be too slow for fast applications such as routers and some wireless communication systems. The various of AES hardware implementation architectures and optimizations have been suggested for different applications. Those to achieve high speed are usually very expensive in hardware complexity such as hard disk, ATM switch, etc . The large area of such architectures may not be suitable for some practical low-end embedded applications and do not require high speed or throughput, but are area critical. Therefore, reducing hardware resources to gain a compact and efficient implementation circuit is an increasing demand.

AES is a 128 bit symmetric data block cipher with 128, 192 or 256 bits key. The data block was described by arrays of bytes in 4×4 matrix (Called "State") and it has four basic steps operation as see in Fig.1 ; SubBytes, (or S-Box), ShiftRow, MixColumn, and AddRoundKey. These four steps are also known as layers. The four layer steps describe one round of the AES. Number of rounds is made vary according to the key size. The AES with 128-bit key size operates iteratively on those four basic steps for 10 rounds. However, the first and the final rounds are arranged in a slightly different manner compared to others. All four layers have their corresponding inverse operations. The deciphering is, therefore, the reverse order of the ciphering process. Operation steps are similar and at the comparable complexity. Moreover, both processes can share same set of designed hardware

The uses of AES in FDE application do exist in both software and hardware form, however they are not so widely published. Software-based AES is also vulnerable to attacks. In contrast, in the pure hardware implementation is more robust. AES IP cores are also available commercially in the form of both ASIC and FPGAs [3,4]. To obtain the higher data rate AES (Gbits/second), a technique of parallelization and pipelining [5] can be combined. The implementations are physically secure since tempering by an outside attacker is naturally difficult. It's also a cost-effective solution for many application specific systems. Although our design is individual, it has drawn many useful ideas and modified some technique reported in [6] to improve system throughput.

The rest of this paper is organized as follows: section 2 is devoted to the AES core system data path architecture for portable hard disk. with delegated to each basic step operations hardware architecture; they're implementation is reviewed in section 3; before the work conclusions given in section 4.

II. AES CORE ARCHITECTURE

AES core of portable hard disk can be design in either Basic iterative AES or One Stage Sub-pipeline AES structure according to the data rate needed. The Basic iterative AES with 128 bit data path is in Fig.1(Data Block#1). The same set of hardware is reused for all the ten iterations. This architecture is entirely based on the iterative approach of design for encryption algorithms. The key expansion block generates the key required for the corresponding iteration on the fly. This design harnesses the parallelism in the AES algorithm and increases the throughput of the design. The One Stage Sub-pipeline structure in Fig.1(Data Block#1 and #2).It is an improvement of the basic iterative architecture with respect to speed. It has just one stage of pipeline within the

data block. The data block is replicated once. In One stage the same work load is shared by two data blocks. The hardware used in the data block is just more than twice the hardware used in Basic iterative design.

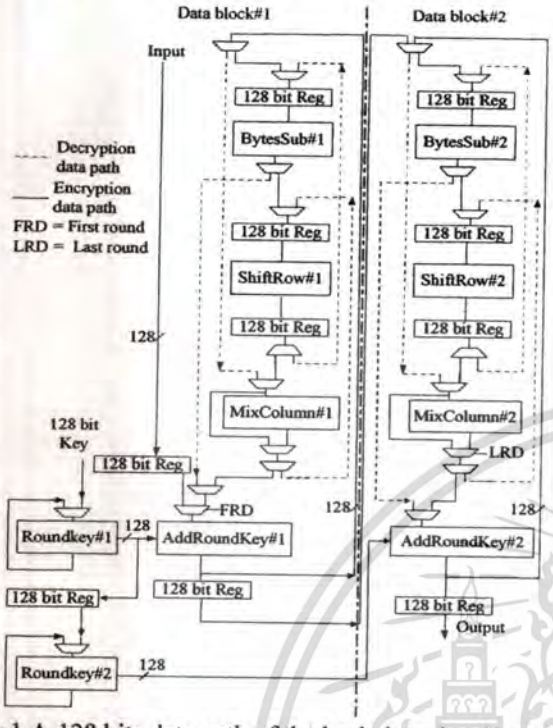


Fig.1 A 128 bits data path of the basic iteration (Data block#1) and one stage sub pipelined AES (Data block #1 and #2)

2.1 S-BOX Transformation

The S-Box operation is a non-linear byte substitution. It composes of two sub- transformations; multiplicative inverse and affine transformation.

1) Multiplicative inverse of each byte is taken – this stage is to compute $B(x)=A^{-1}(x)$ for an 8-bit input word (in $GF(2^8)$ where $m(x)=x^8+x^4+x^3+x+1$ is taken as a field polynomial; {00} is mapped to itself).

2) Affine Transformation: This sub-step is performed in $GF(2)$ and defined by.

$$D(x)=\delta B(x)_{\mathrm{mod}(x^8+1)} \oplus C(x) \qquad (1)$$

where $\delta=\{1F\} = x^4+x^3+x^2+x+1$ for the encryption process and $\delta=\{4A\} = x^6+x^3+x$ for the decryption. The constant $C(x)$ has been added in order that the S-box has no fixed point (a map to a), and no opposite fixed point (a map to $\bar{a}$). $C(x)=\{63\} = x^6+x^5+x+1$ for encryption, and $C(x)=\{05\}=x^2+1$ for S-box inversion. Let $B(x)=A^{-1}(x)$

The required addition of {63} can be made vanish by incorporating it into a modified KeyScheduling [11]. In many AES implementations two sub-steps required in the BytesSub transformation are typically combined into a single lookup table in Fig. 2. The table size is 16 by 16 with the content 8 bits in length. The ROM size of 256 x 8 bit is not big for current technology and can be implemented in a fairly simple manner with modern design tools. However when the area is restricted or a ROM cannot be incorporated, the inversion hardware becomes necessary. Within this scenario, the efficient S-box implementation is the major concern. The affine transform, however requires small number of gates and introduces small delays. Defined by Rijndael, AES has adopted $m(x)=x^8+x^4+x^3+x+1$ as its field polynomial. Although such a polynomial is an irreducible one but it is not a primitive polynomial. The computation of $B(x)=A^{-1}(x)$ is considered to be intricate by most authors and design engineers. Several techniques for S-box computation have been developed. These are, for instances; (1) The mentioned above table look up where step 2 is usually combined to be a single table, (2) Synthesis and optimized logic function of S-box using CAD tools, and (3) Compute the inversion of element in $GF(2^8)$ and optimize the logic functions. In the computation of element inversion in $GF(2^k)$ one can use either extended Euclid algorithm [8], [9] or composite field technique [10], [11], [12]. Rudra *et al.* [10], [11] mapped all the operation (except ShiftRow) into the composite field of $GF(2^4)^2$. Morioka and Satoh [7] also have exploited the used of composite field in the design of a low power S-box transform. Elements in $GF(2^8)$ are mapped to those defined in $GF(2^4)^2$. Multiplication and inversion are optimized in the ground field. Our approach has drawn the techniques of (1) ,(3) and many useful ideas reported in [7], [13] and [14]. To reduce the unnecessary overhead, field transform is applied to the S-box computation only. It is not necessary to further breakdown the composite field to the lowest ground field.

Working in the composite field, multiplicative inverse is leisured. However, forth and back, we have to map elements in $GF(2^k)$ into $GF(2^n)^m$ where $k=mn$. Therefore both transform and inverse transform matrices are needed. Elements in $GF(2^8)$ can be mapped to element in $GF(2^4)^2$ by using the polynomial $r(x)=x^2+x+\beta^{14}$ where β^{14} denotes the element in $GF(2^4)$ of which $I(x)=x^4+x+1$ is the primitive irreducible polynomial. The resulted mapping and inverse mapping matrices are given in eqn. (2) and (3) respectively

$$T=\begin{bmatrix}1&0&1&1&1&0&1&1\\0&1&0&1&0&0&0&0\\0&1&0&0&1&0&1&0\\0&1&1&0&0&0&1&1\\0&0&0&0&1&1&1&0\\0&1&0&0&1&0&1&1\\0&0&1&1&0&1&0&1\\0&0&0&0&0&1&0&1\end{bmatrix}, T^{-1}=\begin{bmatrix}1&0&0&0&1&0&1&0\\0&0&0&0&1&1&0&1\\0&1&0&0&1&1&1&0\\0&1&0&0&1&1&0&1\\0&1&0&1&1&0&1&0\\0&0&1&0&0&1&0&1\\0&1&1&1&0&1&1&1\\0&0&1&0&0&1&0&0\end{bmatrix} \qquad (2,3)$$

The upper–left element in the above matrices denotes the least significant bit. An advantage of mapping elements form $GF(2^8)$ to $GF((2^4)^2)$ is the simpler multiplicative inverse computation since inversion is performed in $GF(2^4)$. For such a small field size, inversion using either the direct truth-table mapping or table look up consumes small area. Moreover, in Rijndael system data are treated naturally in byte format. Let data (byte) be expressed as $A=\{bc\}=bx+c$, the inversion of A, say $B=A^{-1}=\{pq\}=px+q$. For the field polynomial $r(x)=x^2+Cx+D$, one can have

$$p=b\Delta^{-1} \quad (4)$$

$$q=(Cb\oplus c)\Delta^{-1} \quad (5)$$

where

$$\Delta=c(Cb\oplus c)\oplus b^2D. \quad (6)$$

or

$$\Delta=bcC\oplus c^2\oplus b^2D \quad (7)$$

For $GF((2^n)^2)$, the polynomial in the form of $r(x)=x^2+x+\lambda$ always exists [12]. As such, C and D can be set to {1} and {9} (in $GF(2^4)$) respectively. Fixed-coefficient multiplication (i.e., b^2D) as well as squaring units are relatively simple according to their small field size. The multiplications required in computing eqn. (4), (5) and (6) can be done straight away in $GF(2^4)$ or can be further simplified by making use of composite field $GF((2^2)^2)$ [7].

This paper elaborates this approach with some extension in number of pipeline state options for the S-Box design. Shown in Fig.1 the 128 bit data path is made up of 4 of 32 bit data paths. Since one S-Box computation block takes only 8 bit data set, we do need 16 parallel units. The byte substitution computation is also required by the KeyScheduling process in computing its round key. An 128 bit round key is need in every round of AES operation. However only 32 bit: 4 of 8 bit needs byte substitution. Therefore only 4 of 16 S-Box are shared to key update computation.

2.2 ShiftRow Transformation

The ShiftRow process operates on individual row with individual offset byte of state. In the state arrangement, data are fed into a square matrix in row order. To operate the ShiftRow transformation, we need register#1 to store the whole data before byte swapping. This can result in the unsmooth data flow. However, the implementation is not very difficult. Due to we have designed the ShiftRow transform throughput is 128 bits per clock cycle for support the high throughput of hard disk. We used register#2 to be a pipeline arrangement (see Fig. 3 below) such that the data are arranged in order and ready for the following operation, MixColumn transformation.

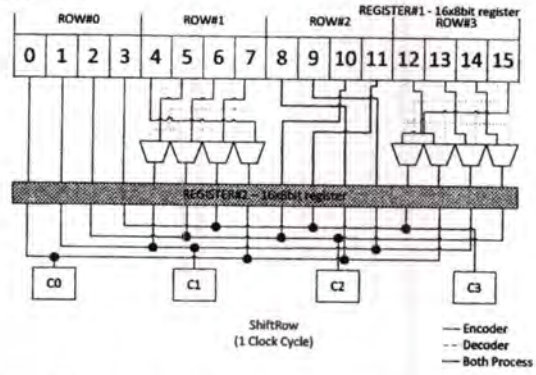


Fig. 3 ShiftRow and Inverse ShiftRow (dash line) Switches with MixColumn-TRANSFORM Ready

2.3 MixColumn Transformation

The mix column transformation operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be expressed by the following matrix multiplication on State.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} c'_{0,0} & c'_{0,1} & c'_{0,2} & c'_{0,3} \\ c'_{1,0} & c'_{1,1} & c'_{1,2} & c'_{1,3} \\ c'_{2,0} & c'_{2,1} & c'_{2,2} & c'_{2,3} \\ c'_{3,0} & c'_{3,1} & c'_{3,2} & c'_{3,3} \end{bmatrix}$$

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF (2^8).The MixColumn transform of an AES can be expressed as $C'(x)=C(x)a(x)_{mod(x4+1)}$ and each column is considered as a polynomial with coefficients Ci,c define in GF(2^8). The multiplication is modulo x^4+1 and a(x) is given by $a(x)=a_0+a_1x+a_2x^2+a_3x^3$ where a0 = {02}, a1 = a2 = {01} and a3 = {03} respectively. The inverse MixColumn matrix can be written in similar way $b(x)=b_0+b_1x+b_2x^2+b_3x^3$ is defined as the inverse transform polynomial where b_0 = {0E} ,b_1 = {09}, b_2 = {0D} and b_3 = {0B} respectively. $C_{i,c}$ is computed in one clock cycle (or one column per one clock cycle) with four parallel fixed-coefficient multipliers, followed by a summing operation as shown in Fig 4. A compact size hardware with one column transform per 4 clock cycles can be arranged similarly.

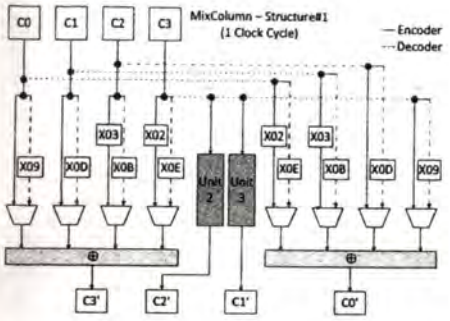


Fig. 4 MixColumn and Inverse MixColumn Transform

2.4 AddRoundKey, KeyScheduling transformation

The KeyScheduling expands the initial 128-bit cipher keys to generate the round keys. The two methods for the key expansion are commonly used, i.e., the round keys can be generated on-the-fly with the data transformation, or they are pre-calculated and stored for later use. In this paper, the round keys applied to the data transformation for encryption or decryption are calculated on-the-fly. The agility of the key expansion deal with the situation that the cipher keys are changed frequently.

The implementation of the key generation for encryption is illustrated in Fig 5. Every word (32 bits) of the next state is the XOR of the current word in the same position with its left neighboring word. For example, the word in C_1 is calculated as $w'[C_1] = w[C_1] \oplus w[C_0]$. For words in the position C_1, its neighboring word is in position C_3. A transformation RotWord is applied to the word in position C_3 prior the XOR, followed by an XOR with the round constant Rcon. So, we need two 128 bit register to store round keys. The S-box can be shared with the S-box as details in section 2. The RotWord register is a circular word shift register. And The RCon is a feedback word shift register. For the one stage sub pipelined AES structure, we have two different KeyScheduling modules which share the load of ten iterations. The KeyScheduling#1 generates the first five keys and the KeyScheduling#2 generates the last five keys. The entire KeyScheduling module generates three set of keys for iteration. Two set of keys from the KeyScheduling#1,#2 and the third, by accumulating the four 32 bits of outputs of the input block.

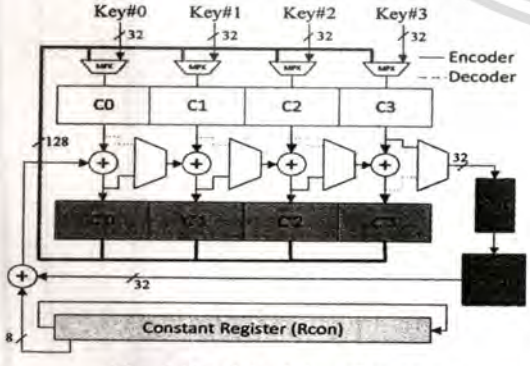


Fig. 5 KeyScheduling Structure

III. FPGA IMPLEMENTATION

For the validation of previous described both AES core structure, the core itself as well as other necessary circuit were designed using VHDL. The code is pure VHDL that could easily be implemented on targeted FPGA devices, without changing the design. Synthesis and Place & Route were achieved on Xilinx ISE 8.1i with Xilinx's device family. This is used for writing, debugging and optimizing efforts, and also for fitting, simulating. The throughput of both AES core structure design based on the data path and processing sub module used.

3.1 A 128 Bit Architecture

The entity of the 128 bits FPGA implementation is shown in Fig. 6. Almost internal data path within AES core entity are 32 bits width. However, some internal register in ShiftRow operation and Key scheduling operation are 128 bits. For the AES structure in Fig. 1 with 10 rounds, The top module provides three separate status signals for interfacing. After START has been asserted and the core is processing, the BUSY pin is asserted and no new plaintext or master key will be accepted. When the core only has one cycle left to process, the NEAR_DONE pin is asserted, allowing for external modules to prepare for the next plaintext/key to be used. When the encryption process is complete and valid cipher text is present on CIPHERTEXT_OUT, the DONE pin is asserted. At this point, both a new key and new plaintext can be loaded, starting on the next rising clock edge. If a key was initially loaded and START is asserted and held high, the core will continuously process the input plaintext with only a single cycle of delay between processing. Both of structure have total latency between asserting START and DONE is 21 clock cycles. But the output of the one stage sub pipelined AES structure is generated after every 11 cycles whereas we have two different data block and KeyScheduling modules which share the load of ten iterations.

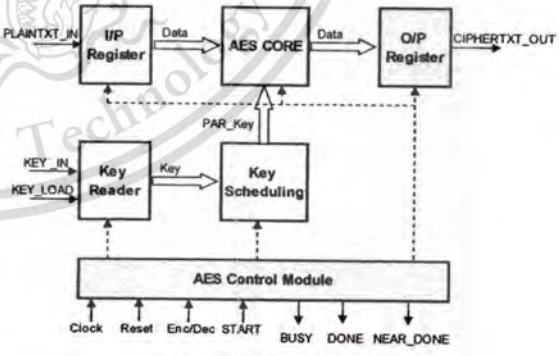


Fig.6 AES's core entity

Clock; Input : AES system clock
Start; Input : AES process starting
Reset; Input : Initial all internal register
BUSY,DONE,NEAR_DONE; Output : AES process status
CIPHERTXT_OUT; Output : Cipher text output
PLAINTXT_IN; Input : Plain text input
KEY_IN; Input : AES master key input
KEY_LOAD; Input : Master key input loading
Enc/Dec; Input : Operation selective(Encrypt/Decrypt)

3.2 Performance Evaluation

The Fig.7 shows the FDE portable hard disk with an designed AES system, which is realized by FPGA. The system includes a USB portable hard disk interface card, a FPGA portable hard disk data encryption/decryption card, an authentication module, a key and a hard disk [14]. The USB portable hard disk interface card is composed of JMS539, SATA II and a USB interface controller. The FPGA portable hard disk data encryption /decryption card consists of an ATA protocol command decoder, a data encryption /decryption module, an I/O external interface, a key signal and a cipher key management module. The FPGA portable hard disk data encryption/decryption card can be regarded as the ATA protocol storage device from the perspective of the host computer. The FPGA portable hard disk data encryption /decryption card can also be regarded as the ATA protocol host controller from the perspective of the hard disk. The SATA interface circuit and encryption /decryption circuit are realized by FPGA method. The throughput and resource required resulted from the synthesis tool are shown in Table I whereas the throughput was calculated from throughput = (block size X Clock Frequency)/Total clock cycle and timing summary example of each sub block in the AES data block is shown in Table II.

IV. CONCLUSIONS

The proposed AES core architecture can be chosen upon speed or throughput requirement for supporting portable hard disk data encryption. A one stage pipelined with LUT S-Box can give the throughput of more than 5 Gbps as high but it consumed more resource than composite field S-Box about 30%.It should be more efficiency than previous paper in the operation speed (the area wasn't significant different) and throughput. A synthesizable VHDL code is developed for the implementation of both encryption and decryption process. The design is verified via the FPGA implementation with Xilinx family.

V. REFERENCES

[1] J. Daemen and V. Rijmen, AES Proposal: Rijndael (Version 2). NIST AES Website; http://csrc.nist.gov/publications/ and http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf

[2] NIST Federal Information Processing Standards (FIPS PUB 197) Advanced Encryption Standard (2001, Nov.). [Online]. Available:http://www.nist.gov/aes

[3] CAST, Advanced Encryption Standard Core, available at; http://www.cast-inc.com/cores/aes/index.shtml.

[4] IP Cores, Ultra-Compact, Advanced Encryption Standard Core, available at; http://www.ipcores.com/AES1.pdf.

[5] S. Chantarawong, P. Noo-intara, and S. Choomchuay, "An Architecture for S-Box Computation in the AES," Proc of Information and Computer Engineering Workshop 2004 (ICEP2004), January 2004, pp.157-162.

[6] Alireza Hodjat, and Ingrid Verbauwhede, "Minimum area cost for a 30 to 70 Gbits/s AES processor," in Proc. of the 2004 IEEE Computer society Annual Symposium on VLSI, pp. 83-88, Feb. 2004.

[7] Sumio Morioka, Akashi Satoh," An Optimized S-Box Circuit Architecture for Low Power AES Design "Tokyo Research Laboratory, IBM Japan Ltd.,2003.

[8] K. Araki, I. Fujita and M. Morisue, "Fast Inverter Over Finite Fields Based on Euclid's Algorithm," *Trans. IEICE*, Vol. E-72, No. 11, pp. 1230–1234, Nov. 1989.

[9] H. Brunner, A. Curiger, and M. Hofstetter, "On Computing Multiplicative Inverse in $GF(2^m)$," *IEEE Trans. on Computer*, Vol. 42., No. 8, pp. 1010–1015, Aug. 1993.

[10] A. Rudra et. al., "Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic," *Proc. CHES 2001*, LNCS Vol. 2162, pp. 175-188, 2001.

[11] C. Jutla, V. Kumar and A. Rudra, "On the Complexity of Isomorphic Galois Field Transforms," *IBM Research Report*, Vol. RC22652 (W0211-243), November 2002.

[12] C. Paar, "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields," *IEEE Trans. on Comp.*, Vol. 45, No. 7, pp 856–861, 1996.

[13] Khanob Thongkhome, Chalermwat Thanavijitpun, and Somsak Choomchuay, "An Implementation of S-Box for a Compact AES System," *Proc. of 25th Int. Conf. on Circuits/Systems, Computers, and Communications (ITC-CSCC2010)*, Pattaya, Thailand, July 2010

[14] Chalermwat Thanavijitpun, Khanob Thongkhome, and Somsak Choomchuay, "FPGA Implementation of FDE-Portable hard diskSystem,"The Int. Conf. on Information and Communication Technology for Embedded Systems, Pattaya, Thailand, January 2011

x \ y	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

Fig.2(a) The LUT S-box substitution values for the byte xy (in hexadecimal format)

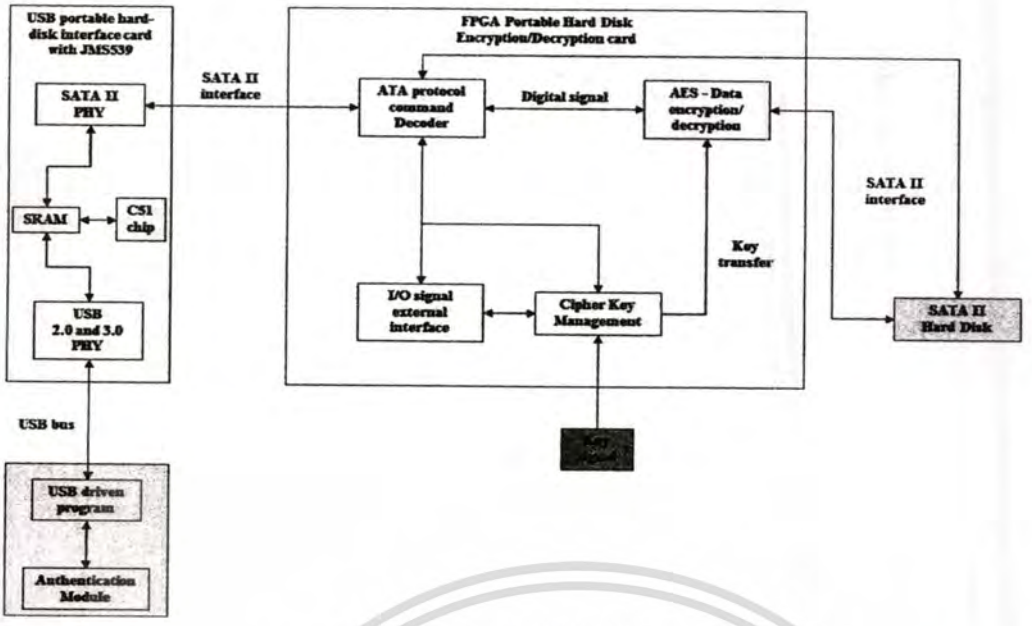


Fig.7 The frame work of the portable hard disk encryption/decryption system with AES core

Table I Utilization Summary of a Designed AES core.

AES Structure	Xilinx Device	CLB Slice	IOs	fmax(MHz)	TP(Gbps)
Basic iteration AES	SPARTAN2E(XC2S600E) with LUT S-Box	2274	391	99.16	1.27
	VERTEX2P(XC2VP7X with LUT S-Box	2599	391	300.76	3.85
	VERTEX4(XC4V2XS) with LUT S-Box	2936	391	247.81	3.78
	SPARTAN2E(XC2S600E) with GF S-Box	1689	391	55.402	0.845
	VERTEX2P(XC2VP7X with GF S-Box	1569	391	141.54	1.81
	VERTEX4(XC4V2XS) with GF S-Box	1569	391	148.85	1.9
One Stage Pipelined	SPARTAN2E(XC2S600E) with LUT S-Box	2729	391	168.67	2.16
	VERTEX2P(XC2VP7X with LUT S-Box	3119	391	481.25	6.16
	VERTEX4(XC4V2XS) with LUT S-Box	3523	391	396.25	5.07
	SPARTAN2E(XC2S600E) with GF S-Box	2027	391	94.29	1.21
	VERTEX2P(XC2VP7X with GF S-Box	1883	391	226.25	2.9
	VERTEX4(XC4V2XS) with GF S-Box	1883	391	238.75	3.06
Previous Paper [4]	VERTEX2(XC21000) with GF S-Box	1724	N/A	264	2.413
Previous Paper [7,8]	SPARTAN3E(XC3S1600E) with GF S-Box	1040	302	64.283	0.976

Table II Timing Summary of a One Stage Pipelined LUT S-Box AES core structure Data Block.

Synthesis result on VERTEX4 - Xilinx (396.25 Mz)	S-Box	ShiftRow	MixColumn	AddRoundKey
Minimum input arrival time before clock(ns)	4.94	3.821	3.821	3.821
Maximum output required time after clock(ns)	4.851	4.851	4.851	4.851

Author's Biography

Name – Surname :	Khanob Thongkhome
Date of Birth :	October 20'1972
Address :	67/80, Moo 5, Soi Srisamitr, Teparuk Road, Bangmuang, Muang, Samutprakarn, 10270
Education :	1994 Bachelor of Science in Technical Education in Computer Technology, King Mongkut's Institute of Technology North Bangkok
	2001 Master of Engineering in Electrical Engineering, Prince of Songkla University
Expertise Field :	1. Digital Image Processing
	2. VHDL programming
	3. Cryptography
Work Experience :	
2010 – Current	Senior FDB Process Engineer, Seagate Technology (Thailand) Ltd.
2002 – 2010	Senior Test Engineer, Seagate Technology (Thailand) Ltd.
1994 – 2002	Test Engineer, Seagate Technology (Thailand) Ltd.