

AES IMPLEMENTATION USING FPGA WITH ENHANCED SECURITY FEATURES

Submitted in partial fulfilment of the requirements for the award of

**Bachelor of Engineering degree in
Electronics and Communication Engineering**

By

MOHAN VEERA SHARAN KONA (38130140)

RATNASAI MORLA (38130142)



**SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING
DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

**SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade “A” by NAAC JEPPIAAR NAGAR,
RAJIV GANDHI SALAI, CHENNAI - 600 119**

March - 2022



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of

MOHAN VEERA SHARAN KONA (38130140)

RATNASAI MORLA (38130142)

who carried out the project entitled **AES Implementation Using FPGA With
Enhanced Security Features** under our supervision from December 2021 to
March 2022

Dr.V.Balamurugan, M.Tech., Ph.D.

Internal guide

Dr.T.Ravi, M.E, Ph.D.
Head of the Department

Submitted for Viva voce Examination held on 07/05/2022

Internal Examiner

Dr.V.Balamurugan

External Examiner

Dr.Nirmal Raj

DECLARATION

We Mr. M.V.Sharan.K (38130140) and Mr. RatnaSai.M (38130142) hereby declare that the Project Report entitled **AES Implementation Using FPGA With Enhanced Security Features** done by me under the guidance of **Dr.Balamurugan** is submitted in partial fulfilment of the requirements for the award of Bachelors of Engineering degree in Electronics and communications.

DATE:11-03-2022

PLACE: Chennai

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. N.M. Nandhitha, Dean, School of Electrical and Electronics Engineering** and **Dr. T. Ravi, Head of the Department, Dept. of Electronics and Communication Engineering** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. Balamurugan** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Department of Electronics and Communication Engineering who were helpful in many ways for the completion of the project.

ABSTRACT

In Networks the data is vulnerable to attacks that may cause Misuse of data or data manipulation. Security of data is playing important role in communication & electronics field. Cryptography algorithms provide security for transmitting electronic data like images, videos. With the expansion of data communications and its applications, there is a greater demand for increasing security systems and devices to guard individual information sent over the transmission channel. One of the most important techniques for securing the information is data encryption. Cryptography algorithms provide security for transmitting electronic data over the channel. Symmetric cryptography techniques, such as the Data Encryption Standard (DES), Triple DES and Advanced Encryption Standard (AES), utilize a key that is identical to the transmitter as well as a receiver, for encrypting and decrypting the data transferred.

IBM developed a security standard named Data Encryption Standard (DES). It was a symmetric block cipher with input 64 bits & key size 56 bits. Due to small key length, the algorithm suffered many attacks. The AES algorithm performs operations on 128-bit plaintext and uses identical key for encryption as well as decryption. The AES algorithm processes Plane text of 128-bit and performs 10, 12 and 14 rounds of operations employing a cipher text of 128-bits, 192-bits and 256-bits respectively.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
CHAPTER 1 - INTRODUCTION		9
1.1	Introduction to VLSI	10
1.2	Why VLSI	13
1.3	Applications Of VLSI	15
1.3.1	Electronic System In Cars	16
1.3.2	Digital Electronics Control VCRS	16
1.3.3	Transaction Processing System, ATM	16
1.3.4	Personal Computers And Workstations	16
1.3.5	Medical Electronic Systems	16
1.4	Cryptography	17
1.4.1	Plain Text	17
1.4.2	Cipher Text	17
1.4.3	Encryption Algorithm	17
1.4.4	Decryption Algorithm	17
CHAPTER 2 - LITERATURE SURVEY		18
2.1	Literature Survey	18
CHAPTER 3 - EXPLANATION		24
3.1	Existing System	24
3.2	Proposed System	24
3.3	AES Algorithm	24
3.4	Encryption Process	27
3.4.1	Byte Substitution	28
3.4.2	Row-Wise Permutation	29

3.4.3 Column-Wise Mixing	30
3.4.4 Addition Of Round Key	31
3.4.5 AES Key Expansion	31
3.5 Decryption Process	34
3.5.1 Inverse Row-Wise Permutation	34
3.5.2 Inverse Column-Wise Mixing	35
3.5.3 Inverse Substitution of Bytes	35
3.6 Software Requirements	36
CHAPTER 4 - SOFTWARE DETAILS	37
4.1 Introduction To Xilinx	37
4.1.2 Xilinx Ise 13.1	38
4.1.3 Simulation	38
4.1.4 Synthesis	38
4.1.5 Procedure	38
4.2 Language	39
4.2.1 History Of The IEEE 1364 Verilog Standard	41
4.2.2 Modelling Enhancements	42
4.2.3 Verilog	43
4.2.4 Definition Of Constants	44
4.2.5 Synthesizable Constructs	45
4.2.6 Initial And Always	45
4.2.7 Fork/Join	45
4.2.8 Four-Valued Logic	45
4.2.9 System Tasks	46

4.2.10 Program Language Interface	46
4.3 Flow Chart	46
CHAPTER 5 - RESULT AND ANALYSIS	47
5.1 Input	47
5.2 Console Output	48
5.3 RTL Schematic	49
5.4 Time Analysis	50
CHAPTER 6 - CONCLUSION	51
REFERENCES	52

CHAPTER 1

INTRODUCTION

World has become digital nowadays and networking is a key feature of this changing globe. Key information of official transactions to social life of an individual, all lie on a network which has attracted researchers towards network security as an important issue to deal with. Cryptography is a technique related to aspects of information security such as data confidentiality, data integrity and entity authentication. In data and telecommunication systems, Security is the most important part for an effective communication, where to increase the security as well as complexity to crack the information. Network technology advances, information security issues increase the need for developing low-area and low power high performance real-time processing of cryptographic algorithms. A number of encryption and decryption techniques have developed since years, Advanced Encryption Standard (AES) being the most popular and highly efficient.

The proposed AES Algorithm system aims to a new approach for generation of S-box values using PN Sequence Generator is presented. A PN sequence generator generates a distinct sequence of random numbers based on the initial seed value and feedback taps. This property of a PN Sequence Generator is used for generating dynamic S-box which enhances the strength of the cryptosystem. This work presents a PN Sequence Generator based design of a Key Generation Block for generating initial key internally. A new secure initial key generation technique eliminates the need to apply the key externally and hence strengthens the modified AES algorithm against attack as compared to traditional AES algorithm. The techniques proposed in this paper are synchronized at both encryption and decryption ends for a truly secure AES algorithm. The AES algorithm with modified S-box values and improved key generation technique is tested on Strict Avalanche Criterion for key sensitivity and an avalanche effect of 60% is achieved. Also, the proposed design is

optimized for speed and area and compared with existing FPGA implementations. The implementation of AES algorithm with modified S-box values using Spartan6 XC6SLX150-3FGG900 FPGA device achieves a throughput of 3.039 Gbps with latency of 10 clock cycles.

1.1 INTRODUCTION TO VLSI :

The development of microelectronics spans a time which is even lesser than the average life expectancy of a human, and yet it has seen as many as four generations. Early 60's saw the low density fabrication processes classified under Small Scale Integration (SSI) in which transistor count was limited to about 10. This rapidly gave way to Medium Scale Integration in the late 60's when around 100 transistors could be placed on a single chip.

It was the time when the cost of research began to decline and private firms started entering the competition in contrast to the earlier years where the main burden was borne by the military. Transistor-Transistor logic (TTL) offering higher integration densities outlasted other IC families like ECL and became the basis of the first integrated circuit revolution. It was the production of this family that gave impetus to semiconductor giants like Texas Instruments, Fairchild and National Semiconductors. Early seventies marked the growth of transistor count to about 1000 per chip called the Large-Scale Integration.

By mid-eighties, the transistor count on a single chip had already exceeded 1000 and hence came the age of Very Large-Scale Integration or VLSI. Though many improvements have been made and the transistor count is still rising, further names of generations like ULSI are generally avoided. It was during this time when TTL lost the battle to MOS family owing to the same problems that had pushed vacuum tubes into negligence, power dissipation and the limit it imposed on the number of gates that could be placed on a single die.

The second age of Integrated Circuits revolution started with the introduction of the first microprocessor, the 4004 by Intel in 1972 and the 8080 in 1974. Today many companies like Texas Instruments, Infineon, Alliance Semiconductors, Cadence, Synopsys, Celox Networks, Cisco, Micron Tech, National Semiconductors, ST Microelectronics, Qualcomm, Lucent, Mentor Graphics, Analog Devices, Intel, Philips, Motorola and many other firms have been established and are dedicated to the various fields in "VLSI" like Programmable Logic Devices, Hardware Descriptive Languages, Design tools, Embedded Systems etc.

In 1980s, hold-over from outdated taxonomy for integration levels obviously influenced from frequency bands, i.e. HF, VHF, and UHF. Sources disagree on what is measured (gates or transistors)

SSI – Small-Scale Integration (0-102)

MSI – Medium-Scale Integration (102 -103)

LSI – Large-Scale Integration (103 -105)

VLSI – Very Large-Scale Integration (105 - 107)

ULSI – Ultra Large-Scale Integration (≥ 107)

VLSI Technology, Inc. was a company which designed and manufactured custom and semi-custom ICs. The company was based in Silicon Valley, with headquarters at 1109 McKay Drive in San Jose, California. Along with LSI Logic, VLSI Technology defined the leading edge of the application-specific integrated circuit (ASIC) business, which accelerated the push of powerful embedded systems into affordable products. The company was founded in 1979 by a trio from Fairchild Semiconductor by way of Synertek - Jack Balletto, Dan Floyd, and Gunnar Wetlesen - and by Doug Fairbairn of Xerox PARC and Lambda (later VLSI Design) magazine.

Alfred J. Stein became the CEO of the company in 1982. Subsequently VLSI built its first fab in San Jose; eventually a second fab was built in San Antonio, Texas. The company was later acquired by Philips and survives to this day as part of NXP Semiconductors.

The first semiconductor chips held two transistors each. Subsequent advances added more and more transistors, and, as a consequence, more individual functions or systems were integrated over time. The first integrated circuits held only a few devices, perhaps as many as ten diodes, transistors, resistors and capacitors, making it possible to fabricate one or more logic gates on a single device. Now retrospectively as small-scale integration (SSI), improvements in technique led to devices with hundreds of logic gates, known as medium-scale integration (MSI). Further improvements led to large-scale integration (LSI), i.e. systems with at least a thousand logic gates. Current technology has moved far past this mark and today's microprocessors have many millions of gates and billions of individual transistors.

At one time, there was an effort to name and calibrate various levels of large-scale integration above VLSI. Terms like ultra-large-scale integration (ULSI) were used. But the huge number of gates and transistors available on common devices has rendered such fine distinctions moot. Terms suggesting greater than VLSI levels of integration are no longer in widespread use.

As of early 2008, billion-transistor processors are commercially available. This is expected to become more commonplace as semiconductor fabrication moves from the current generation of 65 nm processes to the next 45 nm generations (while experiencing new challenges such as increased variation across process corners). A notable example is NVidia's 280 series GPU. This GPU is unique in the fact that almost all of its 1.4 billion transistors are used for logic, in contrast to the Itanium, whose large transistor count is largely due to its 24 MB L3 cache. Current designs, as opposed to the earliest devices, use extensive design

automation and automated logic synthesis to lay out the transistors, enabling higher levels of complexity in the resulting logic functionality. Certain high-performance logic blocks like the SRAM (Static Random Access Memory) cell, however, are still designed by hand to ensure the highest efficiency (sometimes by bending or breaking established design rules to obtain the last bit of performance by trading stability)[citation needed]. VLSI technology is moving towards radical level miniaturization with introduction of NEMS technology. A lot of problems need to be sorted out before the transition is actually made.

1.2 WHY VLSI :

Integration improves the design, lowers the parasites, which means higher speed and lower power consumption and physically smaller. The Integration reduces manufacturing cost - (almost) no manual assembly.

The course will cover basic theory and techniques of digital VLSI design in CMOS technology. Topics include: CMOS devices and circuits, fabrication processes, static and dynamic logic structures, chip layout, simulation and testing, low power techniques, design tools and methodologies, VLSI architecture. We use full-custom techniques to design basic cells and regular structures such as data-path and memory. There is an emphasis on modern design issues in interconnect and clocking. We will also use several case-studies to explore recent real-world VLSI designs (e.g. Pentium, Alpha, PowerPC Strong ARM, etc.) and papers from the recent research literature. On-campus students will design small test circuits using various CAD tools. Circuits will be verified and analyzed for performance with various simulators. Some final project designs will be fabricated and returned to students the following semester for testing.

Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistor-based circuits into a single chip. VLSI began in the 1970s when complex semiconductor and communication

technologies were being developed. The microprocessor is a VLSI device. The term is no longer as common as it once was, as chips have increased in complexity into the hundreds of millions of transistors.

The first semiconductor chips held one transistor each. Subsequent advances added more and more transistors, and, as a consequence, more individual functions or systems were integrated over time. The first integrated circuits held only a few devices, perhaps as many as ten diodes, transistors, resistors and capacitors, making it possible to fabricate one or more logic gates on a single device. Now known retrospectively as "small-scale integration" (SSI), improvements in technique led to devices with hundreds of logic gates, known as large-scale integration (LSI), i.e. systems with at least a thousand logic gates. Current technology has moved far past this mark and today's microprocessors have many millions of gates and hundreds of millions of individual transistors.

At one time, there was an effort to name and calibrate various levels of large-scale integration above VLSI. Terms like Ultra-large-scale Integration (ULSI) were used. But the huge number of gates and transistors available on common devices has rendered such fine distinctions moot. Terms suggesting greater than VLSI levels of integration are no longer in widespread use. Even VLSI is now somewhat quaint, given the common assumption that all microprocessors are VLSI or better.

As of early 2008, billion-transistor processors are commercially available, an example of which is Intel's Montecito Itanium chip. This is expected to become more commonplace as semiconductor fabrication moves from the current generation of 65 nm processes to the next 45 nm generations (while experiencing new challenges such as increased variation across process corners). Another notable example is NVIDIA's 280 series GPU. This microprocessor is unique in the fact that its 1.4 Billion transistor count, capable of a teraflop of performance, is almost entirely dedicated to logic (Itanium's transistor count is largely due to

the 24MB L3 cache). Current designs, as opposed to the earliest devices, use extensive design automation and automated logic synthesis to lay out the transistors, enabling higher levels of complexity in the resulting logic functionality. Certain high-performance logic blocks like the SRAM cell, however, are still designed by hand to ensure the highest efficiency (sometimes by bending or breaking established design rules to obtain the last bit of performance by trading stability).

Only in PC chipsets, did VLSI dominate in the early 90s. This product was developed by five engineers using the 'Mega cells' in the VLSI library that led to a business unit at VLSI that almost equaled its ASIC business in revenue. VLSI eventually ceded the market to Intel because Intel was able to package-sell its processors, chipsets, and even board level products together. VLSI also had an early partnership with PMC, a design group that had been nurtured of British Columbia Bell. When PMC wanted to divest its semiconductor intellectual property venture, VLSI's bid was beaten by a creative deal by Sierra Semiconductor. The telecom business unit management at VLSI opted to go it alone. PMC Sierra became one of the most important telecom ASSP vendors.

Scientists and innovations from the 'design technology' part of VLSI found their way to Cadence Design Systems (by way of Redwood Design Automation). Compass Design Automation (VLSI's CAD and Library spin-off) was sold to Avant! Corporation, which itself was acquired by Synopsys.

1.3 APPLICATIONS OF VLSI

VLSI having applications in various domains such as electronic systems, medical, communication, digital signal processing. Some of them are listed as below

1.3.1 ELECTRONIC SYSTEM IN CARS:

Electronic systems now perform a wide variety of tasks in daily life. Electronic systems in some cases have replaced mechanisms that operated mechanically, hydraulically, or by other means; electronics are usually smaller, more flexible, and easier to service. In other cases electronic systems have created totally new applications. Electronic systems perform a variety of tasks, some of them are visible while some are hidden. Personal entertainment systems such as portable MP3 players and DVD players perform sophisticated algorithms with remarkably little energy.

1.3.2 DIGITAL ELECTRONICS CONTROL VCRS:

Digital electronics compress and decompress video, even at high-definition data rates, on-the-fly in consumer electronics. Low-cost terminals for Web browsing still require sophisticated electronics, despite their dedicated function.

1.3.3 TRANSACTION PROCESSING SYSTEM, ATM

Electronic systems in cars operate stereo systems and displays; they also control fuel injection systems, adjust suspensions to varying terrain, and perform the control functions required for anti-lock braking systems.

1.3.4 PERSONAL COMPUTERS AND WORKSTATIONS

Personal computers and workstations provide word-processing, financial analysis, and games. Computers include both central processing units and special-purpose hardware for disk access, faster screen display, etc.

1.3.5 MEDICAL ELECTRONIC SYSTEMS:

Medical electronic systems measure bodily functions and perform complex processing algorithms to warn about unusual conditions. The availability of these

complex systems, far from overwhelming consumers, only creates demand for even more complex systems.

1.4 CRYPTOGRAPHY:

Cryptography is technique of securing information and communications through use of codes so that only those people for whom the information is intended can understand it and process it. Thus, preventing unauthorized access to information. The prefix “crypt” means “hidden” and suffix graphy means “writing”. In Cryptography the techniques which are used to protect information are obtained from mathematical concepts and a set of rule-based calculations known as algorithms to convert messages in ways that make it hard to decode it. These algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on internet and to protect confidential transactions such as credit card and debit card transactions.

1.4.1 PLAIN TEXT:

This is the message which is readable or understandable. This message is given to the Encryption algorithm as an input.

1.4.2 CIPHER TEXT:

The cipher text is produced as an output of Encryption algorithm. We cannot simply understand this message.

1.4.3 ENCRYPTION ALGORITHM:

The encryption algorithm is used to convert plain text into cipher text.

1.4.4 DECRYPTION ALGORITHM:

It accepts the cipher text as input and the matching key (Private Key or Publickey) and produces the original plain text. There are two types of key: private and public key.

CHAPTER 2

LITERATURE SURVEY

2.1 ENHANCED AES ARCHITECTURE USING EXTENDED SET ALU AT 28NM FPGA

In This Project, The Author proposed an extended implementation of 128-bit AES block along with four blocks of 32 bit Arithmetic and logic units (ALU). The extended set ALU with Manchester encoder decoder, Parity Generator, Parity checker, Binary to Gray and Gray to Binary convertor is used for enhanced security. The proposed model in this paper provides flexibility at user level to switch between various security options. Complete design is implemented on 28nm zynq 7000 FPGA board using Xilinx Vivado tool. Increasing CPU speeds and enhanced data rates are posing a challenge to security engineers to further improve the algorithm. Here in this paper a novel way to implement

AES is proposed where it is implemented in two blocks – primary and secondary. Primary block is 128-bit AES block and secondary block consist of extended set ALU blocks. An Arithmetic and Logic Block Unit is an important part of Datapath design. An ALU block has been designed in various ways in literature with basic arithmetic and logical operations like AND, OR, XOR, NOT, Addition, Multiplication, Subtraction and Division. The new network security model depicted in this paper has wide range of applications in the field of network security and communication security. With a little increase in area, additional low power network security architecture is presented which can provide an enhanced security for data transmission. The design could further be optimized for area and delay.

2.2 FPGA IMPLEMENTATION OF ENHANCED KEY EXPANSION ALGORITHM FOR ADVANCED ENCRYPTION STANDARD

In traditional AES, even though the round keys have high security, Power analysis attack and Saturation attack are effective to the key expansion algorithm of AES due to the deducible key rounds and it leads to security problems. As a result, a new algorithm for generation of round keys is developed for AES. On hardware platform, these algorithms are realizing with enormous memory spaces and large execution time. An alternative hardware platform scenario is provided by Field programmable gate arrays (FPGAs) due to its reconfiguration nature, marketing speed and low price. Accordingly, a hardware implementation of the AES-128 encryption and decryption algorithm with the new algorithm of round key expansion is proposed to improve the security against such attacks. This structure will experimentally simulate using Xilinx software with Verilog HDL and hardware implementation on FPGA. The aim of this proposed algorithm is to increase the level of security for Advanced Encryption Standard and providing a faster processing time. In addition, the difficulty of attacking the key is increased which powers the ability to resist major attacks due to the high randomization of key expansion in the new algorithm. The algorithm was synthesized using XILINX 14.3 and implemented on FPGA. The simulation result for AES-128 is obtained by simulating the proposed key expansion algorithm using Verilog Hardware Description Language

2.3 FPGA IMPLEMENTATION OF AN 8-BIT AES ARCHITECTURE: A ROLLED AND MASKED S-BOX APPROACH

In this paper, they have presented a novel 8-bit architecture for Advanced Encryption Standard (AES) encryption which supports keys of 128-bit in length. The proposed architecture consists of a single round of Shift Rows, Byte

Substitution, Mix Columns and Add Round Key operations through which the data is iterated for ten rounds, which results in substantial reduction in terms of area and power consumption. They have proposed a new architecture for Byte Substitution and AddRoundKey operations by employing high order masking and a different key expansion algorithm respectively, hence making the proposed architecture less vulnerable to Differential Power Analysis (DPA) and saturation attacks. Moreover, they have also utilized a new architecture for Shift Rows operation for further minimizing the area on chip. The proposed architecture was implemented on Virtex-7 FPGA using two different implementation strategies- Performance Explore and Area Explore using Vivado Design Suite. Using performance explore strategy, the proposed architecture worked at the maximum frequency of 200.32 MHz with a throughput of 160.26 Mbps, whereas, with area explore strategy, the proposed architecture utilized 662 slices, 796 LUTs and 0.303 Watt in power.

2.4 FPGA IMPLEMENTATION OF AN 8-BIT AES ARCHITECTURE: A PIPELINED AND MASKED APPROACH

In accordance with the past trend of technological advancements in hardware implementation of security mechanisms, there is an ongoing decrease in size of cryptographic systems with increase in low power and high throughput constraints. In this paper, we present a novel 8-bit pipelined architecture for Advanced Encryption Standard (AES) which ensures high throughput with low area and power consumption. The proposed architecture supports 10 rounds of encryption, where each round consists Shift Rows, Byte Substitution, Mix Columns and AddRoundKey operations. We have employed Boolean masking for all AES operations to increase the security of the intermediate data between the operations and the rounds. To increase the resistance against Differential Power Analysis (DPA) and saturation attacks, high order masking and a different key expansion algorithm in Byte Substitution and for computing round keys in

AddRoundKey operation has been employed respectively. The proposed architecture was implemented on Virtex-7 FPGA using two different implementation strategies: Performance Explore and Area Explore using Vivado Design Suite. Using performance explore strategy, the proposed architecture worked at the maximum frequency of 175.1 MHz with a throughput of 1400.8 Mbps, whereas, while using the area explore strategy, the proposed architecture utilized 7227 slices, 8709 LUTs and 0.717 Watt in power.

2.5 AN EFFICIENT AES IMPLEMENTATION USING FPGA WITH ENHANCED SECURITY FEATURES

With an aim to protect data for secure communication, a new Hybrid non pipelined Advanced Encryption Standard (AES) algorithm based on traditional AES algorithm with enhanced security features is proposed in this work. Abysmal analysis of the AES algorithm implies that the security of AES lies in the S-box operations. This paper presents a new approach for generating S-box values (modified S-box) and initial key required for encryption/decryption (improved key generation) using PN Sequence Generator. The AES algorithm with proposed modifications shows significant improvement in the encryption quality as compared to traditional AES algorithm. The traditional AES algorithm equipped with proposed novel modified S-box technique and improved key generation technique gives an avalanche effect of 60% making it invulnerable to attacks. The proposed design is synthesized on various Field Programmable Gate Array (FPGA) devices and compared to the existing designs resulting in significant improvement in throughput. The proposed design is implemented on Spartan6 FPGA device. Encryption using hardware platforms is widely being used in order to secure data and enhance throughput. In this paper, techniques to enhance the encryption quality of AES algorithm and its implementation on FPGA are proposed. First, the S-box values in the modified AES algorithm are generated

using PN Sequence Generator. Second, the initial key required for encryption/decryption is also based on the output of PN Sequence Generator. The result of encryption for modified AES algorithm is tested on the Strict Avalanche Criterion for 2048 variations and the average percentage avalanche effect of 60% is achieved for modified AES algorithm as compared to tradition AES algorithm. Thus, the modifications suggested, results in improved quality of encryption.

FPGAs are used for efficient hardware implementation of the modified AES algorithm. The results for throughput and area are compared with existing non-pipelined and pipelined designs and are found to achieve better performance.

The proposed design achieves a throughput of 59.3Gbps when synthesized on XC6VLX240T device with a maximum frequency of 463.42 MHz and 30.39 Gbps when implemented on XC6SLX150 with a maximum frequency of 237.45 MHz

2.6 EFFICIENT ARCHITECTURE DESIGN FOR THE AES-128 ALGORITHM ON EMBEDDED SYSTEMS

Many applications make use of the edge devices in wireless sensor networks (WSNs), including video surveillance, traffic monitoring and enforcement, personal and health care, gaming, habitat monitoring, and industrial process control. However, these edge devices are resource-limited embedded systems that require a low-cost, low-power, and high-performance encryption/decryption solution to prevent attacks such as eavesdropping, message modification, and impersonation. This paper proposes a field-programmable gate array (FPGA) based design and implementation of the Advanced Encryption Standard (AES) algorithm for encryption and decryption using a parallel-pipeline architecture with a data forwarding mechanism that efficiently utilizes on-chip memory modules and massive parallel processing units to support a high throughput rate.

Hardware designs that optimize the implementation of the AES algorithm are proposed to minimize resource allocation and maximize throughput. These designs are shown to outperform existing solutions in the literature. Additionally, a rapid prototype of a complete system-on-chip (SoC) solution that employs the proposed design on a configurable platform has been developed and proven to be suitable for real-time applications. The design of a resource-aware, high-performance architecture for the AES encryption/decryption algorithm was presented in this paper. The proposed design and implementation require minimal computational resources, making them ideal for embedded systems that have limited resources and power. The proposed parallel and pipeline architecture achieves the high throughput and low latency that is desirable for real-time applications. Employing a high speed encryption unit in an embedded system of an edge device enhances its ability to protect data against security threats such as electronic eavesdropping, impersonation, and message modification.

CHAPTER 3

EXPLANATION

3.1 EXISTING SYSTEM:

- In previous cryptographic algorithms key size is less and block size is less as compared to the latest algorithm. It also vulnerable to attacks which causes loss or corruption of data.
- In conventional methods they had fixed number of rotations and comparatively slow.

3.2 PROPOSED SYSTEM:

- A new approach for generation of S-box values using PN Sequence Generator is proposed and designing a round key circuit for generating Key using Xilinx ISE.
- New secure initial key generation technique eliminates the need to apply the key externally and hence strengthens the modified AES algorithm against attack as compared to traditional AES algorithm.
- In this algorithm key length is more and the number of rounds depends upon the key length which provides fast operation and more efficient in encryption and providing security.

3.3 AES ALGORITHM:

The Advanced Encryption Standard (AES) was published by the National Institute of Standards and Technology (NIST) in 2001. AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. Compared to public-key ciphers such as RSA, the structure of AES and most symmetric ciphers is quite complex and cannot be explained as easily as many other cryptographic algorithms. Accordingly, the reader may wish

to begin with a simplified version of AES. This version allows the reader to perform encryption and decryption by hand and gain a good understanding of the working of the algorithm details. Classroom experience indicates that a study of this simplified version enhances understanding of AES. The evaluation criteria used by NIST to select from among the candidates for AES, plus the rationale for picking Rijndael, which was the winning candidate. This material is useful in understanding not just the AES design but the criteria by which to judge any symmetric encryption algorithm.

The AES algorithm is a symmetric block cipher used to encrypt and decrypt digital data. The original data is called plaintext, and the encrypted data is called ciphertext. A block of data and a cipher key are 128 bits each for the AES-128 standard. AES-128 is secure enough to protect information up to the secret level. Information is classified as secret by the U.S. Government if its unauthorized disclosure can cause serious damage to national security. A block is represented in an array of 16 bytes known as the State. In this paper, each byte in an input block is referenced using the form in_n , where n is a value between 0 and 15. An input block is presented in the following form: $in_{15}in_{14}in_{13} \dots in_0$. Similarly, each byte in an output block is referenced using the form out_n , and an output block is presented in the following form: $out_{15}out_{14}out_{13} \dots out_0$. Each byte in a key is referenced using the form k_n , and a key is presented in the following form: $k_{15}k_{14}k_{13} \dots k_0$. Figure 1.1 shows the overall structure of the AES algorithm. AES-128 performs 10 rounds of encryption/decryption. Each round uses a different key called a Round Key. The 10 Round Keys are generated by a Key Scheduler using the provided cipher key. In the Add Round Key (AddRoundKey) operation, Galois Field of order 256 addition, or GF (256) addition, is performed to add the input block to the Round Key. A bitwise XOR is used to perform GF (256) addition. A bitwise XOR takes two-bit patterns of equal length and performs the logical exclusive OR operation on each pair of

corresponding bits. In the Substitute Bytes (SubBytes) operation, each input byte is substituted by a different value defined by a substitution table (Sbox). The input byte value is used as the index to look up the output byte value in the S-box. The Inverse Substitute Bytes (InvSubBytes) operation is the same as the SubBytes operation with the exception that it uses the inverse S-box instead. In the Shift Rows (ShiftRows) operation, each row of the State array is cyclically shifted left by a set amount. The 1st row is unchanged, the 2nd row is cyclically shifted left by one position, the 3rd row is cyclically shifted left by two positions, and the 4th row is cyclically shifted left by three positions. The Inverse Shift Rows (InvShiftRow) operation is the same as the ShiftRows operation with the exception that each row is cyclically shifted right instead.

AES does not use a Feistel structure. Instead, each full round consists of four separate functions: byte substitution, permutation, arithmetic operations over a finite field, and XOR with a key.

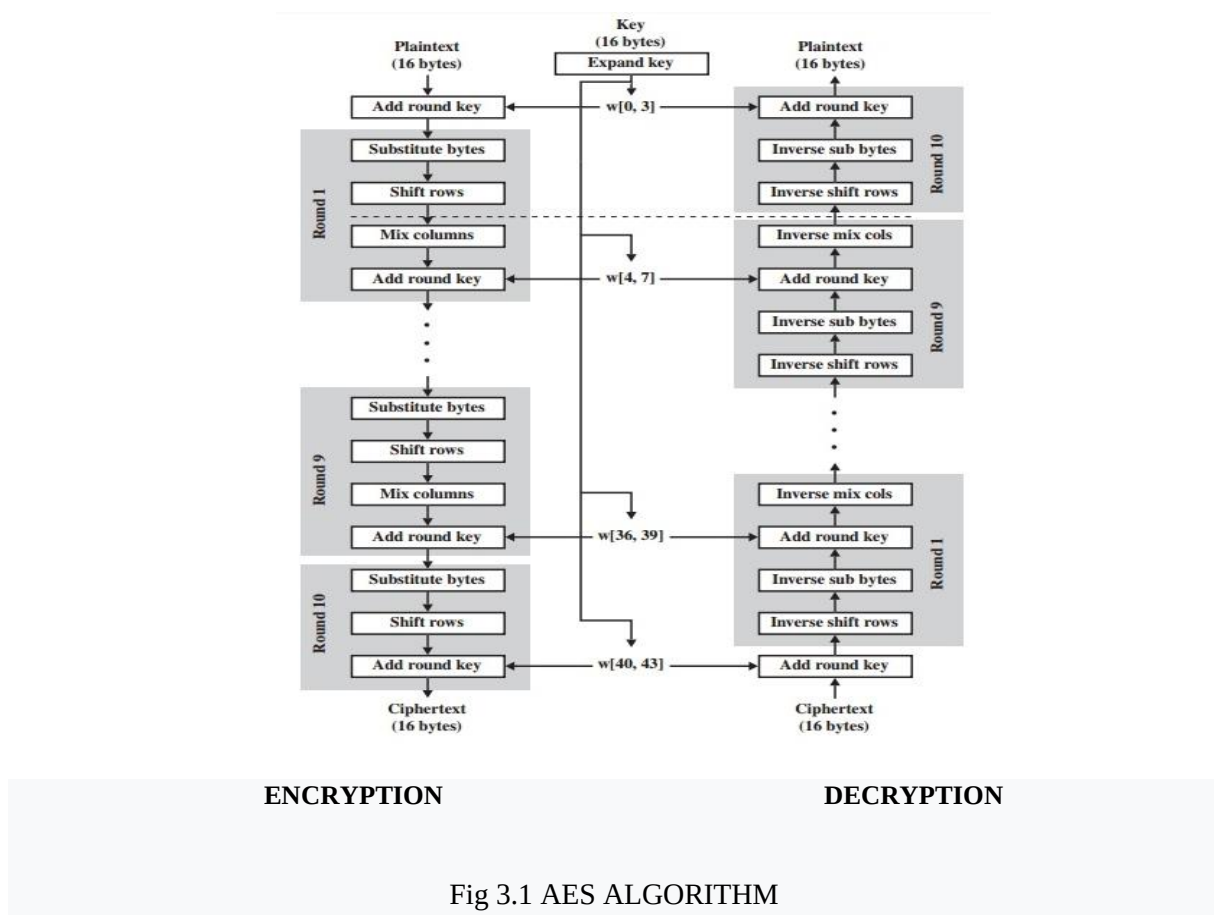


Fig 3.1 AES ALGORITHM

Figure 1.1 shows the AES cipher in more detail, indicating the sequence of transformations in each round and showing the corresponding decryption function.

One noteworthy feature of this structure is that it is not a Feistel structure. In the classic Feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. AES instead processes the entire data block as a single matrix during each round using substitutions and permutation.

The key that is provided as input is expanded into an array of forty-four 32-bit words, $w[i]$. Four distinct words (128 bits) serve as a round key for each round; these are indicated in Figure 1.1.

Four different stages are used, one of permutation and three of substitution

Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block

Shift Rows: A simple permutation

Mix Columns: A substitution that makes use of arithmetic over $GF(28)$

AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key

3.4 ENCRYPTION PROCESS :

The initial process in AES encryption is the addition (xoring) of original key to the input data, which is called an initial round. This is followed by nine iterations of a normal round and ends with a modified final round. During each normal round the following operations are performed in the following order: Byte substitution, Row-wise permutation, Column-wise mixing, Addition of the round key. The final round is also a normal round without the Column-wise mixing process.

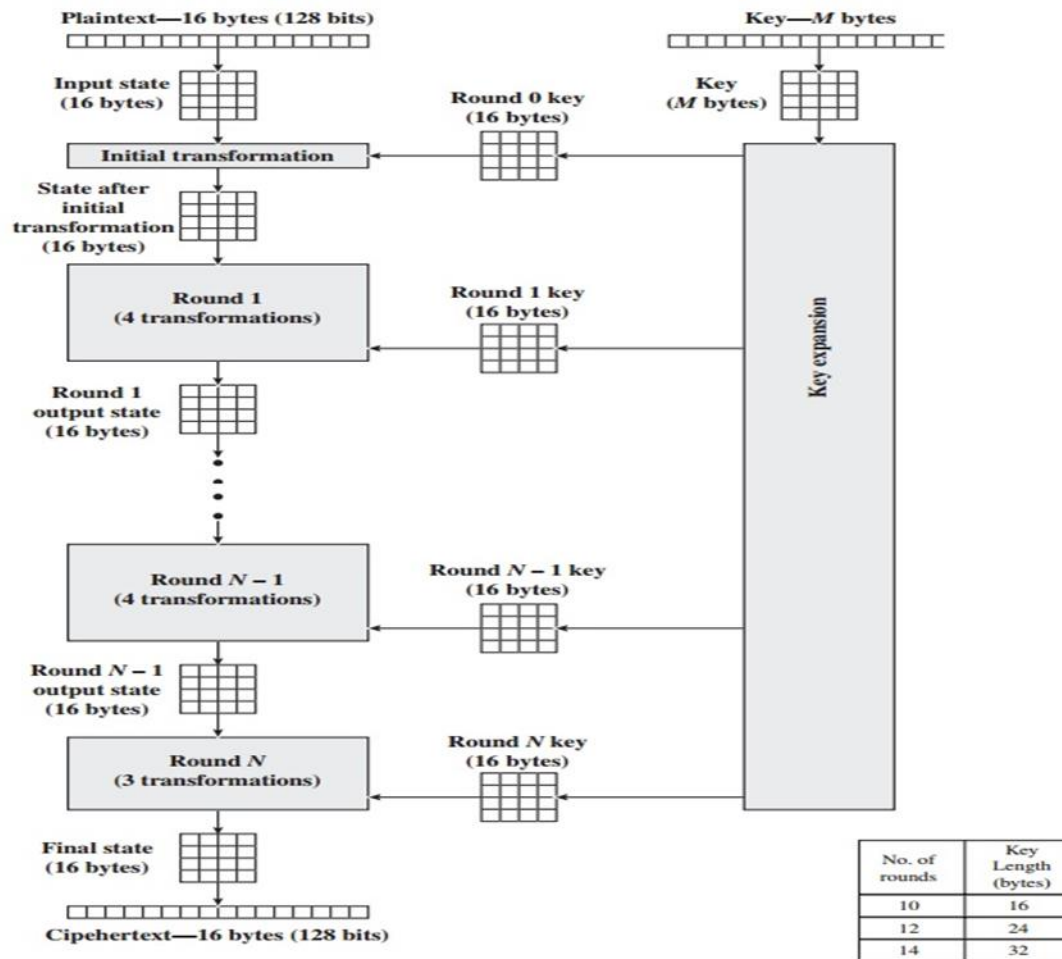


Fig 3.2 AES ENCRYPTION BLOCK DIAGRAM

3.4.1 BYTE SUBSTITUTION:

This is a byte-by-byte substitution process shown in Fig. 1.2. which processes substitution of bytes. The substitution byte for each input byte is found by using the S-Box lookup table. The size of the lookup table is 16×16.

S-box for Encryption.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	1d	0e	07	03	81	c0	60	30	98	4c	26	93	49	24	92	c9
1	64	b2	d9	ec	76	3b	9d	4e	27	13	09	04	82	41	a0	50
2	a8	d4	6a	b5	da	00	6d	b6	5b	ad	d6	6b	35	9a	4d	a6
3	d3	69	34	1a	0d	86	c3	e1	f0	f8	7c	be	df	6f	b7	db
4	ed	f6	7b	bd	5e	af	d7	eb	75	ba	5d	2e	17	8b	45	22
5	11	08	84	c2	61	b0	d8	6c	36	1b	8d	c6	e3	f1	78	3c
6	9e	cf	e7	73	39	9c	ce	67	33	19	8c	46	a3	d1	68	b4
7	5a	2d	96	4b	25	12	89	44	a2	51	28	94	4a	a5	52	a9
8	54	2a	95	ca	e5	72	b9	dc	ee	77	bb	dd	6e	37	9b	cd
9	e6	f3	79	bc	de	ef	f7	fb	fd	7e	bf	5f	2f	97	cb	65
a	32	99	cc	66	b3	59	ac	56	2b	15	8a	c5	62	31	18	0c
b	06	83	c1	e0	70	b8	5c	ae	57	ab	55	aa	d5	ea	f5	fa
c	7d	3e	9f	4f	a7	53	29	14	0a	85	42	21	90	c8	e4	f2
d	f9	fc	fe	ff	7f	3f	1f	0f	87	43	a1	d0	e8	f4	7a	3d
e	1e	8f	c7	63	b1	58	2c	16	0b	05	02	01	80	40	20	10
f	88	c4	e2	71	38	1c	8e	47	23	91	48	a4	d2	e9	74	3a

Fig 3.3 SUBSTITUTION BYTES TABLE

To find the substitute byte for a given input byte, we divide the input byte into two 4-bit patterns, each yielding an integer value between 0 and 15 which can represent these by hex values 0 through F. One of the hex values is used as a row index and the other as a column index for reaching into the 16×16 lookup table.

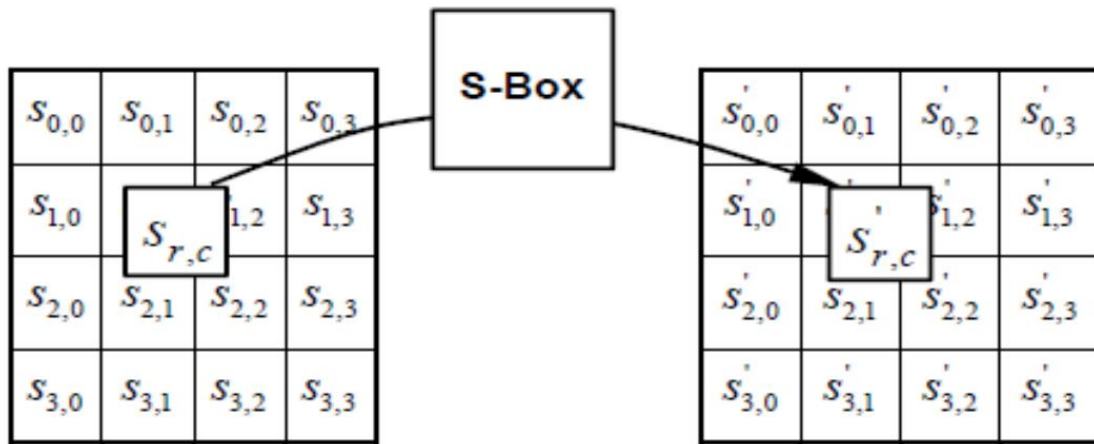


Fig 3.4 BYTE SUBSTITUTION

The entries in the lookup table are constructed by a combination of GF (28) arithmetic and bit scrambling. The goal of the substitution step is to reduce the correlation between the input bits and the output bits. The bit scrambling part of the substitution step ensures that the substitution cannot be described in the form of evaluating a simple mathematical function.

3.4.2 ROW-WISE PERMUTATION:

The Row-wise permutation transformation consists of (i) not shifting the first row of the state array at all (ii) circularly shifting the second row by one byte to the left (iii) circularly shifting the third row by two bytes to the left and (iv) circularly shifting the last row by three bytes to the left. This operation on the state array can be represented by Fig 1.3

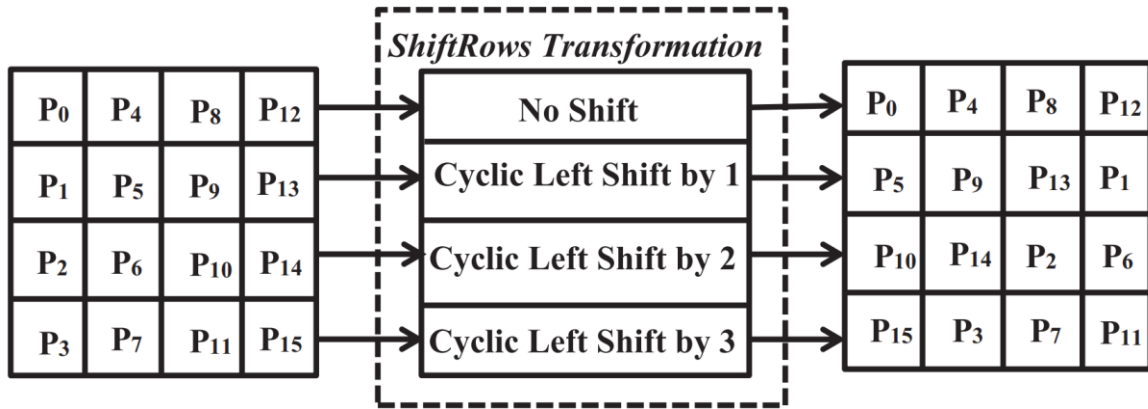


Fig 3.5 SHIFT ROWS OPERATION

3.4.3 COLUMN-WISE MIXING:

This step replaces each byte of a column by a function of all the bytes in the same column. In this Arithmetic Operation, Each column is now multiplied with a constant matrix to mix the columns in state matrix. The MixColumns transformation performs operations on each column of the state matrix one at a time. It is a linear diffusion process.

Where, a row of the leftmost matrix multiplies a column of the state array matrix, additions involved are meant to be XOR operation.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} \dot{S}_{0,0} & \dot{S}_{0,1} & \dot{S}_{0,2} & \dot{S}_{0,3} \\ \dot{S}_{1,0} & \dot{S}_{1,1} & \dot{S}_{1,2} & \dot{S}_{1,3} \\ \dot{S}_{2,0} & \dot{S}_{2,1} & \dot{S}_{2,2} & \dot{S}_{2,3} \\ \dot{S}_{3,0} & \dot{S}_{3,1} & \dot{S}_{3,2} & \dot{S}_{3,3} \end{bmatrix}$$

Fig 3.6 MIX COLUMNS OPERATION

3.4.4 ADDITION OF ROUND KEY:

The key words are generated by key expansion process. Round Key is added to every State by a XOR operation where each Round Key consists of Nb words. Those Nb words are each added into the columns of the State, such that $[w_i]$ are the key schedule words and round is a value in the range 0 round Nr. In the Cipher, the initial Round Key addition occurs when round = 0, prior to the first application of the round function. Add Round Key transformation to the Nr rounds of the Cipher occurs when $1 < \text{round} < \text{Nr}$. A simple xor operation with the state to key word is shown in Fig. 1.5.

The key words are generated by key expansion process. The round keys are calculated for each round are added to state matrix. In this transformation the round key obtained is XORed with the state by bitwise operation. The Nibble words from key matrix of each Round Key are XORed with the columns of the State matrix.

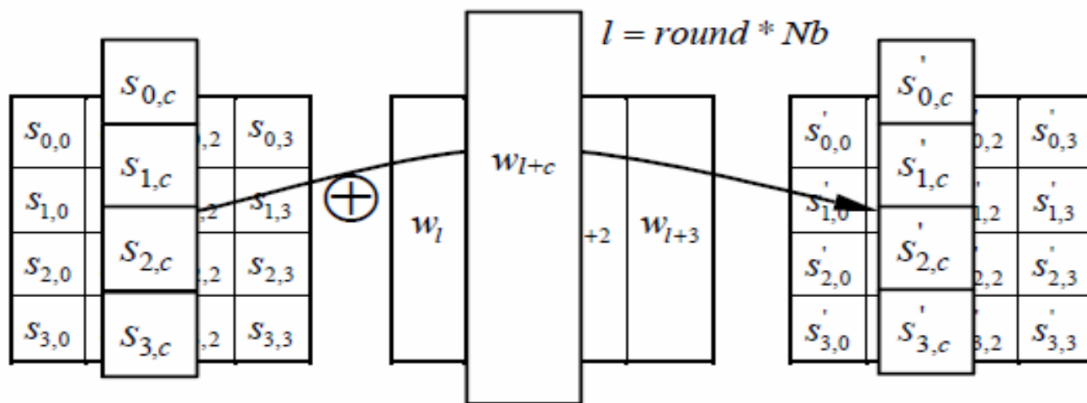


Fig 3.7 ADDING ROUND KEY OPERATION

3.4.5 AES Key Expansion:

The AES key expansion algorithm takes input 128-bits as 16-bytes which is represented by $k_0, k_1, \dots, k_{15}$. The first 4-key words w_0, w_1, w_2, w_3 are obtained

from those columns which is shown in Fig. 1.6. This original key words are sufficient to generate add round keys for each of the 10 rounds of the cipher.

In the traditional AES key expansion algorithm, supposing that the original key words are (w_0, w_1, w_2, w_3) , it generates the sub-key words $(w_4, w_5, \dots, w_{43})$ for 10 rounds. (i.e) Original key: w_0, w_1, w_2, w_3 1st round key: w_4, w_5, w_6, w_7 2nd round key: w_8, w_9, w_{10}, w_{11} ... 10th round key: $w_{40}, w_{41}, w_{42}, w_{43}$ The key expansion process of the first round is

$$w_4 = w_0 + \text{Subword}(\text{Rotword}(w_3)) + \text{Rcon}(1)$$

$$w_5 = w_1 + w_4$$

$$w_6 = w_2 + w_5$$

$$w_7 = w_3 + w_6$$

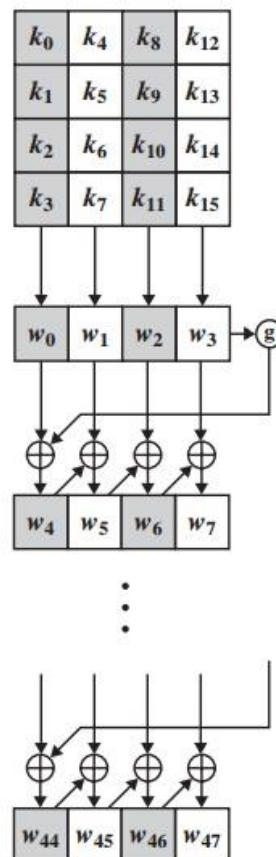


Fig 3.8 AES KEY EXPANSION

From the second round, the key expansion process will be same upto 10th round as follows.

$$w_8 = w_4 + \text{Subword}(\text{Rotword}(w_7)) + \text{Rcon}(2)$$

$$w_9 = w_5 + w_8$$

$$w_{10} = w_6 + w_9$$

$$w_{11} = w_7 + w_{10}$$

Root Word performs a one-byte circular left shift on a word. This means that an input word $[B_0, B_1, B_2, B_3]$ is transformed into $[B_1, B_2, B_3, B_0]$. Sub Word performs a byte substitution on each byte of its input word, using the S-box. The result is XORed with a round constant. The Fig. 1.7. shown the process of function g.

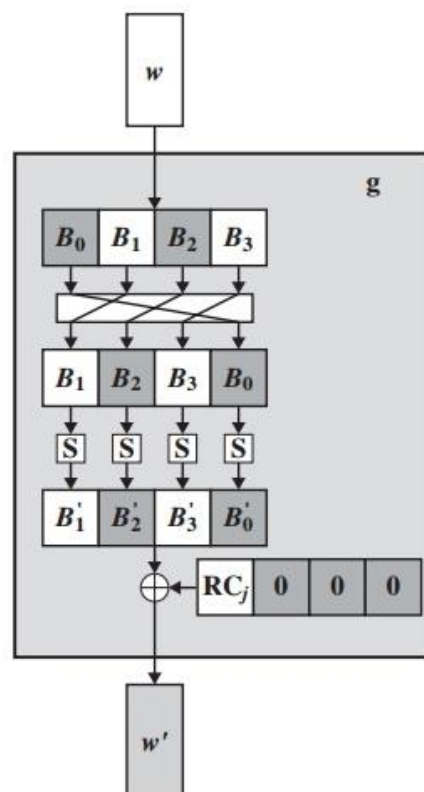


Fig 3.9 G FUNCTION

The first four words of the expanded key are defined from the original keys. From the second round each added word $w[i]$ depends on the immediately preceding word $w[i-1]$ and the word four positions back $w[i-4]$ from the previous round. In three out of four cases, a simple XOR is used. A complex function g is used in generation of the expanded key to make higher security.

3.5 DECRYPTION PROCESS:

In AES decryption, the operations are in reverse order. It starts with an initial round followed by nine iterations of an inverse round and ends with an Addition of Round Key. An inverse round consists of the following operations in this order: Addition of Round Key, Inverse column-wise mixing, Inverse Row-wise permutation and Inverse Substitution of Bytes. An initial round is an inverse round without the Inverse Column-wise mixing and addition of round key has its own inverse for both the encryption and decryption process.

3.5.1 INVERSE ROW-WISE PERMUTATION:

For decryption, the corresponding step shifts the rows in opposite manner. The first row is left unchanged, the second row is shifted to the right by one byte, the third row to the right by two bytes, and the last row to the right by three bytes, all are circularly shifted.

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \implies \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,3} & S_{1,0} & S_{1,1} & S_{1,2} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,0} \end{bmatrix}$$

Fig 3.10 INVERSE ROW SHIFTING OPERATION

3.5.2 INVERSE COLUMN-WISE MIXING:

It is the inverse process of column-wise mixing that operates on the State column-by-column and taking each column as a four-term polynomial.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} \dot{S}_{0,0} & \dot{S}_{0,1} & \dot{S}_{0,2} & \dot{S}_{0,3} \\ \dot{S}_{1,0} & \dot{S}_{1,1} & \dot{S}_{1,2} & \dot{S}_{1,3} \\ \dot{S}_{2,0} & \dot{S}_{2,1} & \dot{S}_{2,2} & \dot{S}_{2,3} \\ \dot{S}_{3,0} & \dot{S}_{3,1} & \dot{S}_{3,2} & \dot{S}_{3,3} \end{bmatrix}$$

Fig 3.11 INVERSE COLUMN SHIFTING OPERATION

The columns are considered as polynomials over GF (28) and multiplied modulo $x^4 + 1$ with a fixed polynomial $a^{-1}(x)$, given by $a^{-1}(x) = \{0b\} x^3 + \{0d\} x^2 + \{09\} x + \{0e\}$, this can be written as a matrix multiplication.

3.5.3 INVERSE SUBSTITUTION OF BYTES:

The inverse substitution byte for each input byte is found by using the inverse s-box lookup table. The size of the lookup table is 16×16. To find the inverse substitution byte for a given input byte, we divide the input byte into two 4-bit patterns, each yielding an integer value between 0 and 15 which can represent these by hex values 0 through F.

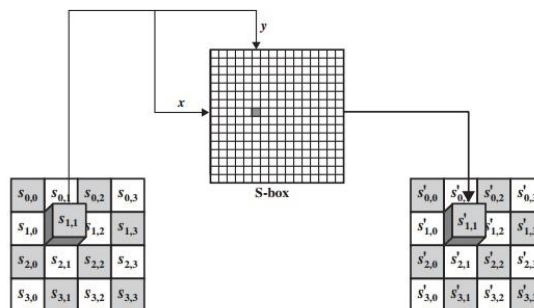


Fig 3.12 INVERSE SUBSTITUTION BYTES OPERATION

One of the hex values is used as a row index and the other as a column index for reaching into the 16×16 lookup table. The entries in the lookup table are constructed by a multiplicative inverse of GF (28).

3.6 SOFTWARE REQUIREMENTS:

- Xilinx ISE
- Language - Verilog

CHAPTER - 4

SOFTWARE REQUIREMENT DETAILS

4.1 INTRODUCTION TO XILINX:

Xilinx, Inc. is the world's largest supplier of programmable logic devices, the inventor of the field programmable gate array (FPGA) and the first semiconductor company with a fabless manufacturing model.

Xilinx designs, develops and markets programmable logic products including integrated circuits (ICs), software design tools, predefined system functions delivered as intellectual property (IP) cores, design services, customer training, field engineering and technical support. Xilinx sells both FPGAs and CPLDs programmable logic devices for electronic equipment manufacturers in end markets such as communications, industrial, consumer, automotive and data processing.

Xilinx's FPGAs have even been used for the ALICE (A Large Ion Collider Experiment) at the CERN European laboratory on the French-Swiss border to map and disentangle the trajectories of thousands of subatomic particles.

The Vertex-II Pro, Virtex-4, Virtex-5, and Virtex-6 FPGA families are particularly focused on system-on-chip (SOC) designers because they include up to two embedded IBM PowerPC cores. The ISE Design Suite is the central electronic design automation (EDA) product family sold by Xilinx. The ISE Design Suite features include design entry and synthesis supporting Verilog or VHDL, place-and-route (PAR), completed verification and debug using Chip Scope Pro tools, and creation of the bit files that are used to configure the chip. XST-Xilinx Synthesis Technology performs device specific synthesis for Cool Runner XPLA3/-II and XC9500/XL/XV families and generates an NGC file ready for the CPLD fitter.

4.1.2 XILINX ISE 13.1:

Xilinx is the most important tool and in this tool we can perform both simulation and synthesis.

4.1.3 SIMULATION:

In this process we are going to verify our required output to get the simulation process firstly we have to implement a top module (combination of all modules) and then in the simulation behavior we can simulate the results.

4.1.4 SYNTHESIS:

Synthesis process defines converting Verilog code into gate level which creates a net list.

4.1.5 PROCEDURE:

- Click project navigator
- Create new project
- Selection of FPGA

Create new source

- Select source type (Verilog module)
- Coding
- Declaration of inputs and output
- Sources for implementation

Synthesize – XST

- Check syntax
- View design summary
- View RTL schematic
- View technology schematic

- Sources for behavioral simulation

Create new source

- Select source type(Verilog text fixture)
- Write test bench code
- Xilinx ISE simulator
- Behavioral check syntax
- Simulate behavioral model

4.2 LANGUAGE

Both VHDL and Verilog are well established hardware description languages. They have the advantage that the user can define high-level algorithms and low-level optimizations (gate-level and switch-level) in the same language. A basic example of VHDL code, the evaluation of the Fibonacci series, is shown below, and it is a good example of the points made above. The code itself is reasonably straightforward for a software programmer to understand, provided that he/she understands that this is a truly parallel language and all lines are executing “at once”. It is also straightforward to simulate a simple design of this nature. However it is surprisingly difficult to implement it in hardware and this difficulty is a direct result of I/O issues. As noted above for a design to work in hardware access is required to resources that are external to the FPGA, such as memory, and an FPGA is, by its very nature, unaware of the components to which it is connected. If you want to retrieve a value from main memory and use it on the FPGA then you need to instantiate a memory controller. While systems such as the Cray XD1 provide cores for communicating with memory, such cores are still complex and unfamiliar to software programmers. Our early experiences with VHDL have indicated that it should only be used for FPGA development if

you are in a position to work closely with experienced hardware designers throughout the development process.

Verilog HDL is one of the two most common Hardware Description Languages (HDL) used by integrated circuit (IC) designers. The other one is VHDL. HDL's allows the design to be simulated earlier in the design cycle in order to correct errors or experiment with different architectures. Designs described in HDL are technology-independent, easy to design and debug, and are usually more readable than schematics, particularly for large circuits. Verilog can be used to describe designs at four levels of abstraction:

- Algorithmic level (much like c code with if, case and loop statements).
- Register transfer level (RTL uses registers connected by Boolean equations).
- Gate level (interconnected AND, NOR etc.)
- Switch level (the switches are MOS transistors inside gates).

The language also defines constructs that can be used to control the input and output of simulation. More recently Verilog is used as an input for synthesis programs which will generate a gate-level description (a netlist) for the circuit. Some Verilog constructs are not synthesizable. Also the way the code is written will greatly affect the size and speed of the synthesized circuit. Most readers will want to synthesize their circuits, so no synthesizable constructs should be used only for test benches. These are program modules used to generate I/O needed to simulate the rest of the design. The words "not synthesizable" will be used for examples and constructs as needed that do not synthesize.

Verilog (released as IEEE 1364-2001) adds many significant enhancements to the Verilog language, which add greater support for configurable IP modeling and deep-submicron accuracy, and development of design management. Other enhancements make Verilog easier to use. These

changes will affect everyone who uses the Verilog language, as well as those who implement Verilog software tools. This paper will review and highlight the main features added to the Verilog standard for the IEEE 1364-2001 update. The focus will be on new simulation and synthesis constructs. Where possible, status regarding Synopsys support for the new features will also be noted.

4.2.1 History of the IEEE 1364 Verilog standard:

The Verilog Hardware Description Language was first introduced in 1984, as a proprietary language from Gateway Design Automation. The original Verilog language was designed to be used with a single product, the Gateway Verilog-XL digital logic simulator. In 1989, Gateway Design Automation was acquired by Cadence Design Systems. In 1990, Cadence released the Verilog Hardware Description Language and the Verilog Programming Language Interface (PLI) to the public domain. Open Verilog International (OVI) was formed to control the public domain Verilog, and to promote its usage. Cadence turned over to OVI the Frame Maker source files containing most, but not all, of the Cadence Verilog-XL user's manual. This document became OVI's Verilog 1.0 Reference Manual. In 1993, OVI released its Verilog 2.0 Reference Manual, which contained a few enhancements to the Verilog language, such as array of instances, and major enhancements to the Verilog PLI. OVI then submitted a request to the IEEE to formally standardize Verilog 2.0. The IEEE formed a standards working group to create the standard, and, in 1995, IEEE 1364-1995 became the official Verilog standard. It is important to note that for Verilog-1995, the IEEE standards working group did not consider any enhancements to the Verilog language. The goal was to standardize the Verilog language the way it was being used at that time. The IEEE working group also decided not to create an entirely new document for the IEEE 1364 standard. Instead, the OVI Frame Maker files were used to create the IEEE standard. Since the origin of the OVI manual was Gateway's Verilog-XL user's manual, the IEEE 1364-1995 and

IEEE 1364-2001 Verilog language reference manuals are still organized somewhat like a user's guide.

Goals for Verilog standard Work on the IEEE 1364-2001 Verilog standard began in January 1997. Three major goals were established:

- Enhance the Verilog language to help with today's deep-submicron and intellectual property modeling issues.
- Ensure that all enhancements were both useful and practical, and that simulator and synthesis
- Vendors would implement Verilog-2000 in their products. Correct any errata or ambiguities in the IEEE 1364-1995 Verilog Language Reference Manual.

4.2.2 MODELING ENHANCEMENTS:

Many enhancements improve the ease and accuracy of writing synthesizable RTL models. Other enhancements allow models to be more scalable and re-usable. With the exception of the following paragraph, only changes which add new functionality or syntax are listed here. Verilog also contains many clarifications to Verilog-1995, which do not add new functionality. Notes are added to the sub-sections indicating Synopsys support with Presto and VCS at the time this paper was completed. Since the inception of Verilog in 1984, the term "register" has been used to describe the group of variable data types in the Verilog language. "Register" is not a keyword, it is simply a name for a class of data types, namely: reg, integer, time, real, and realtime. The use of term "register" is often a source of confusion for new users of Verilog, who sometimes assume that the term implies a hardware register (flip-flops).

4.2.3 VERILOG:

Hardware description languages such as Verilog differ from software programming languages because they include ways of describing the propagation of time and signal dependencies (sensitivity). There are two assignment operators, a blocking assignment (`=`), and a non-blocking (`<=`) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables. Since these concepts are part of Verilog's language semantics, designers could quickly write descriptions of large circuits in a relatively compact and concise form. At the time of Verilog's introduction (1984), Verilog represented a tremendous productivity improvement for circuit designers who were already using graphical schematic capture software and specially written software programs to document and simulate electronic circuits.

The designers of Verilog wanted a language with syntax similar to the C programming language, which was already widely used in engineering software development. Like C, Verilog is case-sensitive and has a basic preprocessor (though less sophisticated than that of ANSI C/C++). Its control flow keywords (`if/else`, `for`, `while`, `case`, etc.) are equivalent, and its operator precedence is compatible. Syntactic differences include variable declaration (Verilog requires bit-widths on `net/reg` types), demarcation of procedural blocks (`begin/end` instead of curly braces `{ }`), and many other minor differences.

A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: `net/variable` declarations (`wire`, `reg`, `integer`, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies). Sequential statements are placed inside a `begin/end` block and

executed in sequential order within the block. However, the blocks themselves are executed concurrently, making Verilog a dataflow language.

Verilog's concept of 'wire' consists of both signal values (4-state: "1, 0, floating, undefined") and strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive a common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths. Subsets of statements in the Verilog language are synthesizable. Verilog modules that conform to a synthesizable coding style, known as RTL (register-transfer level), can be physically realized by synthesis software. Synthesis software algorithmically transforms the (abstract) Verilog source into a net list, a logically equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific FPGA or VLSI technology. Further manipulations to the net list ultimately lead to a circuit fabrication blueprint (such as a photo mask set for an ASIC or a bit stream file for an FPGA).

4.2.4 DEFINITION OF CONSTANTS:

The definition of constants in Verilog supports the addition of a width parameter. The basic syntax is:

<Width in bits>'<base letter><number>

Examples:

- 12'h123 - Hexadecimal 123 (using 12 bits)
- 20'd44 - Decimal 44 (using 20 bits - 0 extension is automatic)
- 4'b1010 - Binary 1010 (using 4 bits)
- 6'o77 - Octal 77 (using 6 bits)

4.2.5 SYNTHESIZABLE CONSTRUCTS:

There are several statements in Verilog that have no analog in real hardware, e.g. \$display. Consequently, much of the language cannot be used to describe hardware. The examples presented here are the classic subset of the language that has a direct mapping to real gates.

4.2.6 INITIAL AND ALWAYS:

There are two separate ways of declaring a Verilog process. These are the always and the initial keywords. The always keyword indicates a free-running process. The initial keyword indicates a process executes exactly once. Both constructs begin execution at simulator time 0, and both execute until the end of the block. Once an always block has reached its end, it is rescheduled (again). It is a common misconception to believe that an initial block will execute before an always block. In fact, it is better to think of the initial-block as a special-case of the always-block, one which terminates after it completes for the first time.

4.2.7 FORK/JOIN:

The fork/join pair are used by Verilog to create parallel processes. All statements (or blocks) between a fork/join pair begin execution simultaneously upon execution flow hitting the fork. Execution continues after the join upon completion of the longest running statement or block between the fork and join.

4.2.8 FOUR-VALUED LOGIC:

The IEEE 1364 standard defines a four-valued logic with four states: 0, 1, Z (high impedance), and X (unknown logic value). For the competing Verilog HDL, a dedicated standard for multi-valued logic exists as IEEE 1164 with nine levels.

4.2.9 SYSTEM TASKS:

System tasks are available to handle simple I/O, and various design measurement functions. All system tasks are prefixed with \$ to distinguish them from user tasks & functions. It presents a short list of the most often used tasks. It is by no means a comprehensive list.

4.2.10 PROGRAM LANGUAGE INTERFACE:

The PLI provides a programmer with a mechanism to transfer control from Verilog to a program function written in C language. It is officially deprecated by IEEE Std 1364-2005 in favor of the newer Verilog Procedural Interface, which completely replaces the PLI. The PLI enables Verilog to cooperate with other programs written in the C language such as test harnesses, instruction set simulators of a microcontroller, debuggers, and so on. For example, it provides the C functions `tf_putlongp()` and `tf_getlongp()` which are used to write and read the argument of the current Verilog task or function, respectively.

4.3 FLOW CHART:

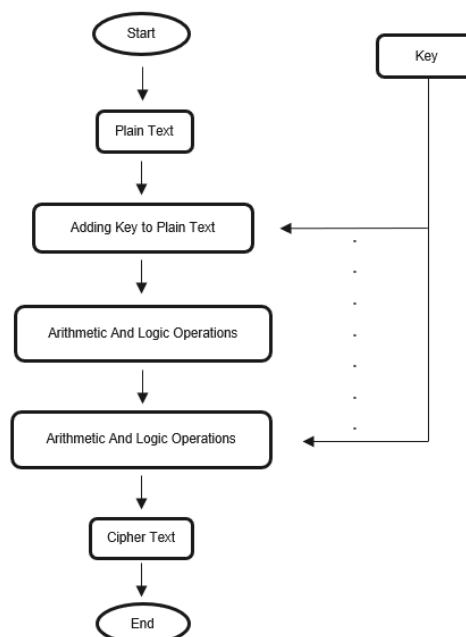


Fig 4.1 FLOW CHARTS

CHAPTER 5

RESULTS AND ANALYSIS

5.1 INPUT:

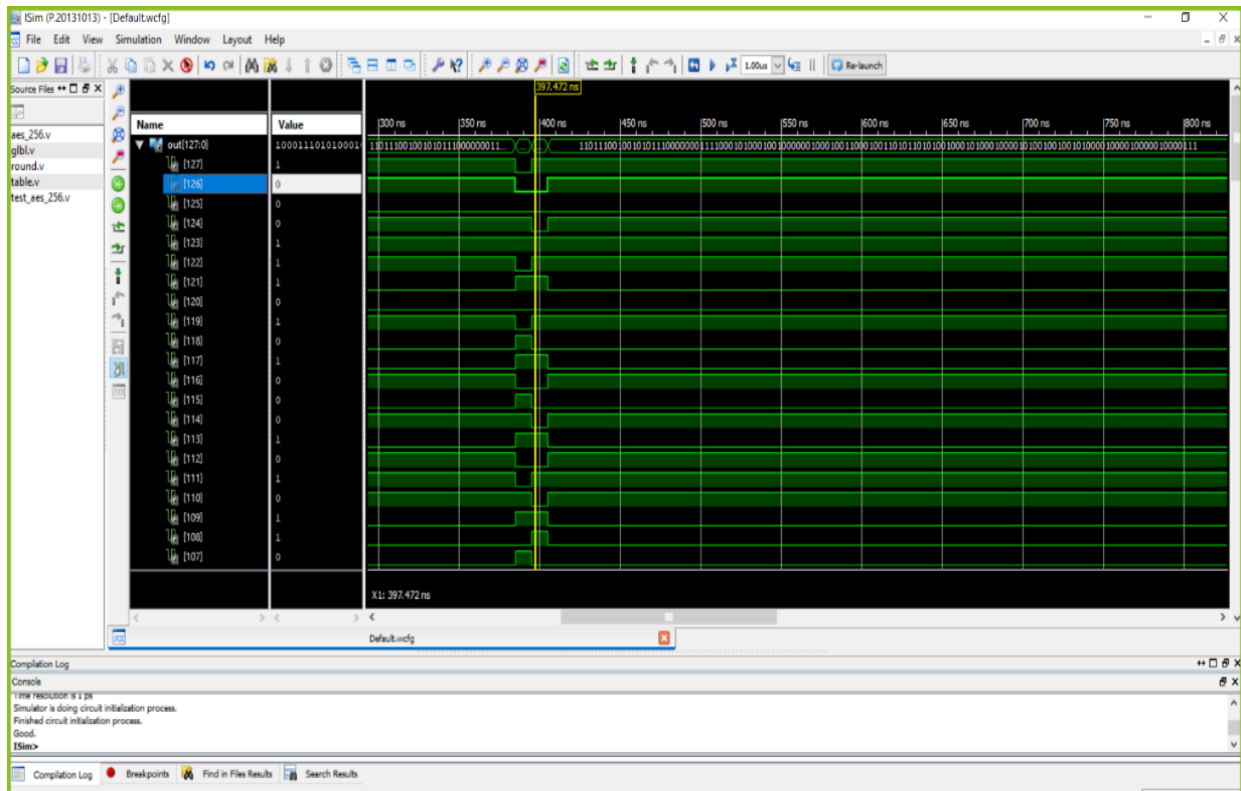


Fig 5.1 128-bit Input

This is the 128-bit Input.

State Input = 128'h3243f6a8885a308d313198a2e0370734;

Key size = 256 bits

Number of Rounds = 14

We used 128 bit and 256 bit Key to get 14 AES algorithm rounds, Which increases randomization of key expansion that in return makes data more securable over the network line.

5.2 CONSOLE OUTPUT :

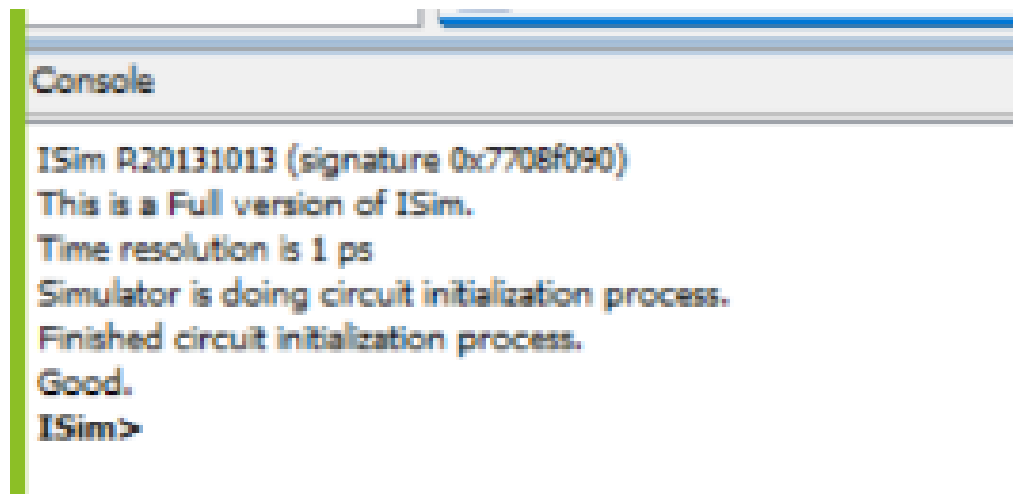


Fig 5.2 Console window

To confirm whether the given plain text and the out cipher text are same, we used if condition to get an output statement **GOOD** or **E**. If the console output is **GOOD** that indicates the cipher and plain texts are same .In case if the output is **E** it means the given plain text doesn't matches with the obtained cipher text.

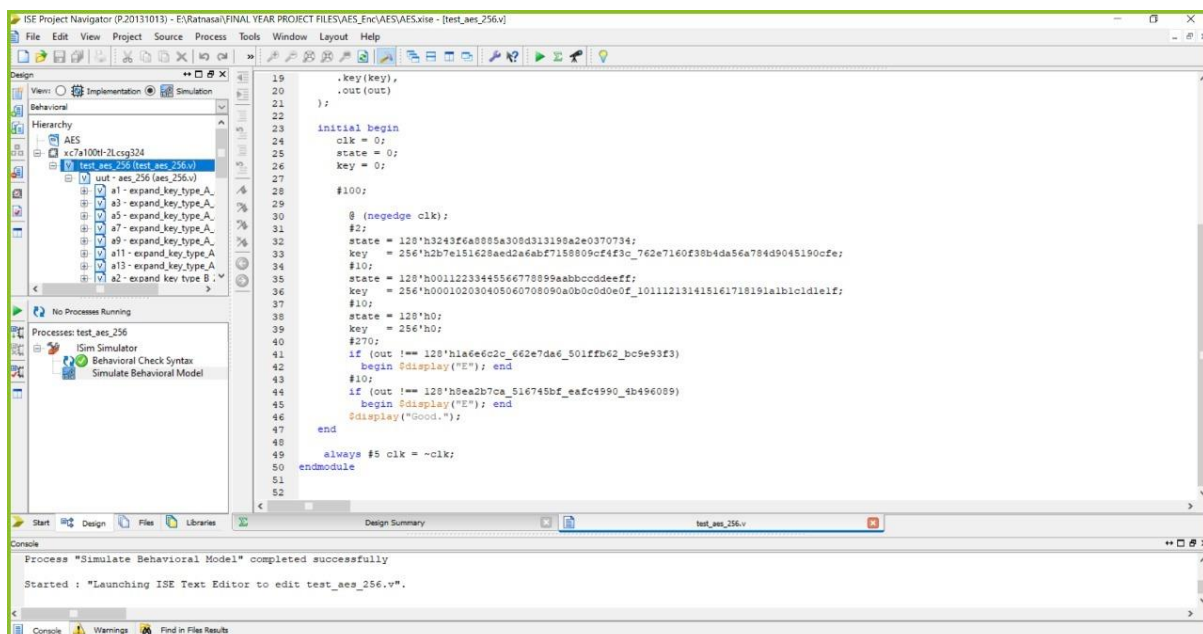


Fig 5.3 Code for Confirmation

5.3 OUTPUT (RTL SCHEMATIC):

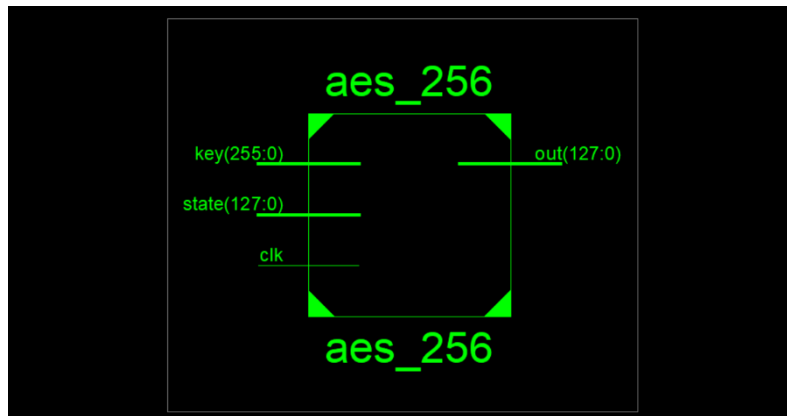


Fig 5.4 RTL Schematic of Implemented AES Algorithm

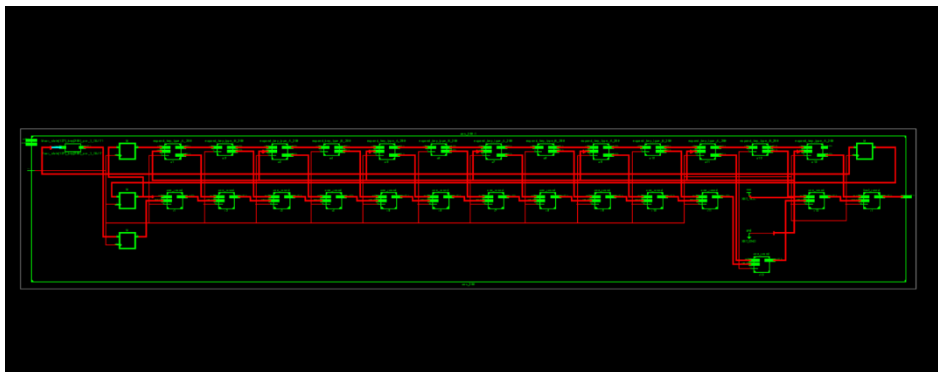


Fig 5.5 14 Rounds of AES Algorithm

5.4 TIME ANALYSIS:

```

=====
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
Total number of paths / destination ports: 128 / 128
=====
Offset:                0.737ns (Levels of Logic = 1)
Source:                rf/state_out_127 (FF)
Destination:          out<127> (PAD)
Source Clock:          clk rising

Data Path: rf/state_out_127 to out<127>

```

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FD:C->Q	1	0.398	0.339	rf/state_out_127 (rf/state_out_127)
OBUF:I->O		0.000		out_127_OBUF (out<127>)
Total		0.737ns (0.398ns logic, 0.339ns route) (54.0% logic, 46.0% route)		

```

=====

Cross Clock Domains Report:
=====

Clock to Setup on destination clock clk

```

Source Clock	Src:Rise	Src:Fall	Dest:Rise	Dest:Fall
clk	3.795			

```

=====

```

Fig 5.6 Time Taken to give Output

Due to the 256-bit key and 14 round operation of the algorithm, key expansion and randomization it takes some time to execute all steps and give the cipher text. But in communication data must be transferred with low latency. Implementation of AES Algorithm in FPGA makes algorithm work faster. It is implemented in 0.737ns.

CHAPTER 6

CONCLUSION

AES has wide range of applications in the field of network security and communication security. Our project is to increase the level of security using Advanced Encryption Standard and providing a faster processing time.

In addition, the difficulty of attacking the key is increased which powers the ability to resist major attacks due to the high randomization of key expansion in the new algorithm. The aim of this proposed algorithm is to increase the level of security for Advanced Encryption Standard and providing a faster processing time.

The algorithm was synthesized using XILINX 14.3. The simulation result for AES-128 is obtained by simulating the proposed key expansion algorithm using Verilog Hardware Description Language.

REFERENCES

- **Deshpande, Hrushikesh S; Karande, Kailash J; Mulani, Altaaf O [IEEE 2014 International Conference on Communications and Signal Processing (ICCSP) efficient implementation of aes algorithm on FPGA.**
- **G. Nidhi, M. Anu, K. Deepika and T. Devyani, “A Novel Implementation of 32 bit Extended ALU Architecture at 28nm FPGA”, ICETCT.**
- **Hoang Trang and Nguyen Van (2012), An efficient FPGA implementation of the Advanced Encryption Standard algorithm**
- **Harshali Zodpe and Ashok Sapkal. 2018. An Efficient AES Implementation using FPGA with Enhanced Security Features. Journal of King Saud University - Engineering Sciences 32 .**
- **L. Khoi-Nguyen, D. Quoc-Minh, B. Trong-Tu and D. Anh-Vu, “RTL Implementation for a Specific ALU of the 32-bit VLIW DSP Processor Core”, International conference on advanced technologies for communications (ATC’14), p.p.387-392.**
- **M. Rao, T. Newe, and I. Grout. 2015. AES implementation on Xilinx FPGAs suitable for FPGA based WBSNs. In 2015 9th International Conference on Sensing Technology (ICST).**
- **Peter Schwabe and Ko Stoffelen. All the AES You Need on Cortex-M3 and M4. 180–194.**
- **P. Prasithsangaree and P. Krishnamurthy. Analysis of energy consumption of RC4 and AES algorithms in wireless LANs. In GLOBECOM ’03. IEEE Global Telecommunications Conference (IEEE Cat. No.03CH37489), Vol. 3. 1445–1449 vol.3. [14] Daniele Puccinelli and Martin Haenggi.**

- **Ritu Pahal Vikas kumar,” Efficient implementation of aes”
International Journal of Advanced Research in Computer Science
and Software Engineering, Volume 3, Issue 7, July 2013 ISSN: 2277
128X.**
- **S. Chen, W. Hu, and Z. Li. 2019. High Performance Data Encryption
with AES Implementation on FPGA. In 2019 IEEE 5th Intl
Conference on Big Data Security on Cloud (BigDataSecurity), IEEE
Intl Conference on High Performance and Smart Computing,
(HPSC) and IEEE Intl Conference on Intelligent Data and Security
(IDS). 149–153.**
- **Understanding AES Mix-Columns Transformation Calculation , Kit
Choy Xintong .**
- **Understanding Cryptography by Christof Paar, Jan Pelzl.**
- **Verilog HDL: A Guide to Digital Design and Synthesis, By Samir
Palnitkar.**
- **Xinmiao Zhang and Keshab K. Parhi, ”Implementation approaches
for the advanced encryption standard algorithm”.**
- **Yang Jun Ding Jun Li Na Guo Yixiong School of Information
Science and Engineering, Yunnan University Kunming, China -
“FPGA based design and implementation of reduced AES
algorithm”.**