**CHAPTER 1**

* 1. **INTRODUCTION TO AES:**

AES (Advanced Encryption Standard) is a United States encryption standard defined in Federal Information Processing Standard (FIPS) 197, published in November 2001. It was ratified as a federal standard in May 2002. AES is the most recent of the four current algorithms approved for federal us in the United States. One should not compare AES with RSA, another standard algorithm, as RSA is a different category of algorithm. Bulk encryption of information itself is seldom performed with RSA.RSA is used to transfer other encryption keys for use by AES for example, and for digital signatures. AES is a symmetric encryption algorithm processing data in block of 128 bits. 128-bit block is encrypted by transforming it in a unique way into a new block of the same size. AES is symmetric since the same key is used for encryption and the reverse transformation, decryption. The only secret necessary to keep for security is the key. AES may configured to use different key-lengths, the standard defines 3 lengths and the resulting algorithms are named AES-128, AES-192 and AES-256 respectively to indicate the length in bits of the key. Each additional bit in the key effectively doubles the strength of the algorithm, when defined as the time necessary for an attacker to stage a brute force attack, i.e. an exhaustive search of all possible key combinations in order to find the right one.

**1.2 SOME BACKGROUND ON AES:**

In 1997 the US National Institute of Standards and Technology put out a call for candidates for a replacement for the ageing Data Encryption Standard, DES. 15 candidates were accepted for further consideration, and after a fully public process and three open international conferences, the number of candidates was reduced to five. In February 2001, the final candidate was announced and comments were solicited. 21 organizations and individuals submitted comments. None had any reservations about the suggested algorithm. AES is founded on solid and well-published mathematical ground, and appears to resist all known attacks well. There’s a strong indication that in fact no back-door or known weakness exists since it has been published for a long time, has been the subject of intense scrutiny by researchers all over the world, and such enormous amounts of economic value and information is already successfully protected by AES. A strong encryption algorithm need only meet only single main criteria:

* There must be no way to find the unencrypted clear text if the key is unknown, except brute force, i.e. to try all possible keys until the right one is found.

A secondary criterion must also be met:

* The number of possible keys must be so large that it is computationally infeasible to actually stage a successful brute force attack in short enough a time.

The older standard, DES or Data Encryption Standard, meets the first criterion, but no longer the secondary one – computer speeds have caught up with it, or soon will. AES meets both criteria in all of its variants: AES-128, AES-192 and AES-256.

**1.3 ENCRYPTION MUST BE DONE PROPERLY:**

AES may, as all algorithms, be used in different ways to perform encryption. Different methods are suitable for different situations. It is very easy to implement a system using AES as its encryption algorithm, but much more skill and experience is required to do it in the right way for a given situation. To describe exactly how to apply AES for varying purposes is very much out of scope for this short introduction.

**1.4 STRONG KEYS:**

Encryption with AES is based on a secret key with 128, 192 or 256 bits. But if the key is easy to guess it doesn’t matter if AES is secure, so it is as critically vital to use good and strong keys as it is to apply AES properly. Creating a good and strong key is a surprisingly difficult problem and requires careful design when done with a computer. Keys derived into a fixed length suitable for the encryption algorithm from passwords or pass phrases typed by a human will seldom correspond to 128 bits much less 256. To even approach 128 bit equivalence in a pass phrase, at least 10 typical passwords of the kind frequently used in day-to-day work are needed. The risks of incorrect usage, implementation and weak keys are in no way unique for AES; these are shared by all encryption algorithms. Provided that the implementation is correct, the security provided reduces to a relatively simple question about how many bits the chosen key, password or pass phrase really corresponds to.

**1.5 SECURITY IS RELATIVE:**

Security is not an absolute. it’s a relation between time and cost. Currently, there are speculations that military intelligence services possibly have the technical and economic means to attack keys equivalent to about 90 bits, although no civilian researcher has actually seen or reported of such a capability. Actual and demonstrated systems today, within the bounds of a commercial budget of about 1 million dollars can handle key lengths of about 70 bits. An aggressive estimate on the rate of technological progress is to assume that technologies will double the speed of computing devices every year at an unchanged cost. An illustration of the current status for AES is given by the following example, where we assume an attacker with the capability to build or purchase a system that tries keys at the rate of one billion keys per second.

This is at least 1000 times faster than the fasted personal computer in 2004. In some military circumstances a few hours or days security is sufficient – after that the war or the mission is completed and the information uninteresting and without value. In other cases a lifetime may not be long enough.

**CHAPTER 2**

**2.1 ADVANCED ENCRYPTION STANDARD**:

**Advanced Encryption Standard** (**AES**) is a specification for the [encryption](http://en.wikipedia.org/wiki/Encryption" \o "Encryption) of electronic data. It has been adopted by the [U.S. government](http://en.wikipedia.org/wiki/Federal_government_of_the_United_States" \o "Federal government of the United States) and is now used worldwide. The algorithm described by AES is a [symmetric-key algorithm](http://en.wikipedia.org/wiki/Symmetric-key_algorithm" \o "Symmetric-key algorithm), meaning the same key is used for both encrypting and decrypting the data. It is available in many different encryption packages. AES is the first publicly accessible and open [cipher](http://en.wikipedia.org/wiki/Cipher" \o "Cipher) approved by the [National Security Agency](http://en.wikipedia.org/wiki/National_Security_Agency" \o "National Security Agency) (NSA) for [top secret](http://en.wikipedia.org/wiki/Classified_information" \o "Classified information) information. Originally called Rijndael, the [cipher](http://en.wikipedia.org/wiki/Cipher" \o "Cipher) was developed by two [Belgian](http://en.wikipedia.org/wiki/Belgium" \o "Belgium) cryptographers, Joan Daemen and Vincent Rijmen, and submitted by them to the AES selection process by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted by them to the AES selection process. The name Rijndael is a play on the names of the two inventors. Strictly speaking, AES is the name of the standard, and the algorithm described is a variant of Rijndael. However, in practice the algorithm is also referred to as "AES" (Advanced Encryption Standard).

**2.2 DESCRIPTION OF THE CIPHER:**

AES is based on a design principle known as a [substitution-permutation network](http://en.wikipedia.org/wiki/Substitution-permutation_network" \o "Substitution-permutation network). It is fast in both [software](http://en.wikipedia.org/wiki/Software" \o "Software) and [hardware](http://en.wikipedia.org/wiki/Hardware" \o "Hardware). Unlike its predecessor, DES, AES does not use a Feistel network.

AES has a fixed [block size](http://en.wikipedia.org/wiki/Block_size_(cryptography)" \o "Block size (cryptography)) of 128 [bits](http://en.wikipedia.org/wiki/Bit" \o "Bit) and a [key size](http://en.wikipedia.org/wiki/Key_size" \o "Key size) of 128, 192, or 256 bits, whereas Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits. The block size has a maximum of 256 bits, but the key size has no theoretical maximum. AES operates on a 4×4 [column-major order](http://en.wikipedia.org/wiki/Column-major_order" \o "Column-major order) matrix of bytes, termed the state. Most AES calculations are done in a special [finite field](http://en.wikipedia.org/wiki/Finite_field_arithmetic" \o "Finite field arithmetic).

The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plain text into the final output of cipher text. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform cipher text back into the original plain text using the same encryption key.

### 2.3 HIGH LEVEL DESCRIPTION OF THE ALGORITHM:

1. Key Expansion round keys are derived from the cipher key using [Rijndael's key schedule](http://en.wikipedia.org/wiki/Rijndael_key_schedule" \o "Rijndael key schedule)
2. Initial Round

* Add Round Key: Each byte of the state is combined with the round key using bit wise XOR.

1. Rounds

* Sub Bytes: a non-linear substitution step where each byte is replaced with another according to a [lookup table](http://en.wikipedia.org/wiki/Rijndael_S-box" \o "Rijndael S-box).
* Shift Rows: a transposition step where each row of the state is shifted cyclically a certain number of steps.
* Mix Columns: a mixing operation which operates on the columns of the state, combining the four bytes in each column.
* Add Round Key

1. Final Round (no Mix Columns)

* Sub Bytes
* Shift Rows
* Add Round Key

### 2.4 THE SUB BYTES STEP:

In the Sub Bytes step, each byte in the matrix is updated using an 8-bit [substitution box](http://en.wikipedia.org/wiki/Substitution_box" \o "Substitution box), the [Rijndael S-box](http://en.wikipedia.org/wiki/Rijndael_S-box" \o "Rijndael S-box). This operation provides the non-linearity in the [cipher](http://en.wikipedia.org/wiki/Cipher" \o "Cipher). The S-box used is derived from the [multiplicative inverse](http://en.wikipedia.org/wiki/Multiplicative_inverse" \o "Multiplicative inverse) over **[GF](http://en.wikipedia.org/wiki/Finite_field" \o "Finite field)**(*28*), known to have 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](http://en.wikipedia.org/wiki/Affine_transformation" \o "Affine transformation).  The S-box is also chosen The S-box is also chosen to avoid any fixed points (and so is a [derangement](http://en.wikipedia.org/wiki/Derangement" \o "Derangement)), and also any opposite fixed points. 1

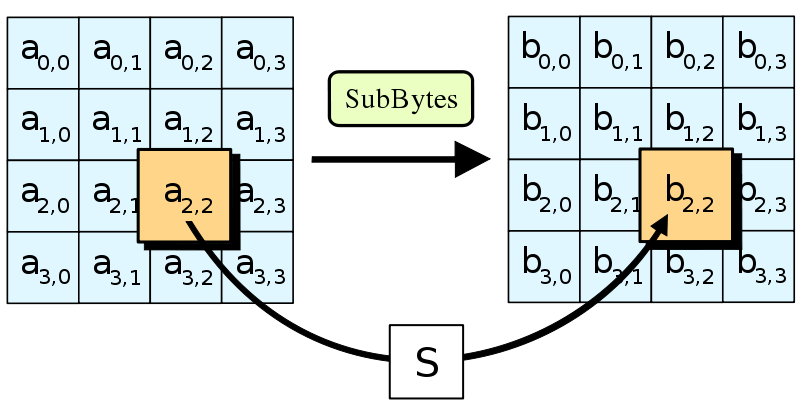


Fig 1: Sub bytes

### 2.5 THE SHIFTROWS STEP:

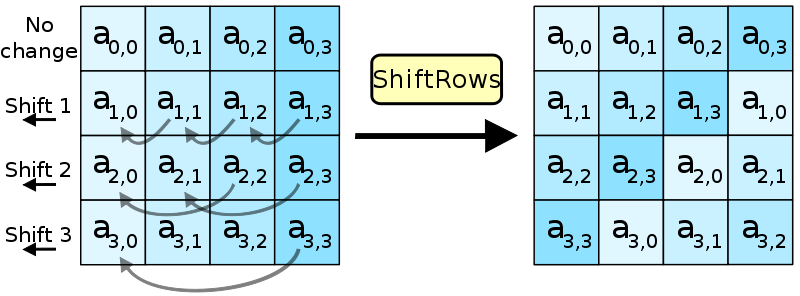
****

Fig 2: Shift rows

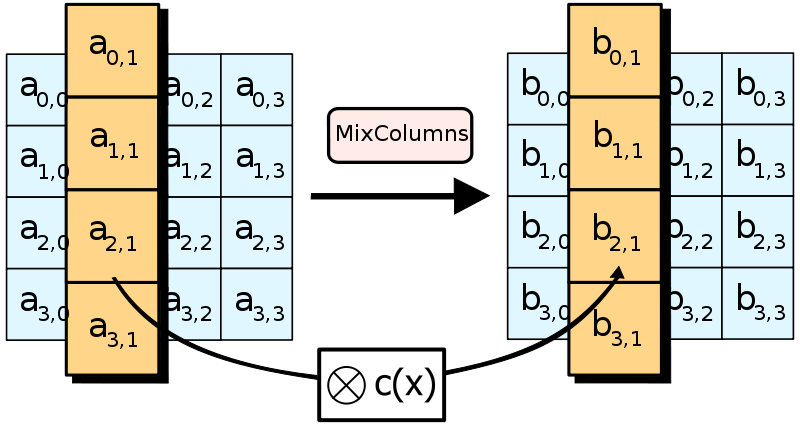
The Shift Rows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain [offset](http://en.wikipedia.org/wiki/Offset_(computer_science)" \o "Offset (computer science)). For AES, the first ro33w is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. 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 Shift Rows 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-bit 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-bit block, as AES does not use 256-bit blocks.

**2.6 THE MIX COLUMNS STEP:**

In the Mix Columns step, the four bytes of each column of the state are combined using an invertible [linear transformation](http://en.wikipedia.org/wiki/Linear_transformation" \o "Linear transformation). The Mix Columns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with Shift Rows, Mix Columns provides [diffusion](http://en.wikipedia.org/wiki/Diffusion_(cryptography)" \o "Diffusion (cryptography)) in the cipher. During this operation, each column is multiplied by the known matrix that for the 128 bit key is


\begin{bmatrix}
2 & 3 & 1 & 1 \\
1 & 2 & 3 & 1 \\
1 & 1 & 2 & 3 \\
3 & 1 & 1 & 2
\end{bmatrix}.


The multiplication operation is defined as: multiplication by 1 means leaving unchanged, multiplication by 2 means shifting byte to the left and multiplication by 3 means shifting to the left and then performing XOR with the initial unshifted value. After shifting, a conditional XOR with 0x1B should be performed if the shifted value is larger than 0xFF.

In more general sense, each column is treated as a polynomial over **GF**(*28*) and is then multiplied modulo x4+1 with a fixed polynomial c(x) = 0x03 · x3 + x2 + x + 0x02. The coefficients are displayed in their [hexadecimal](http://en.wikipedia.org/wiki/Hexadecimal" \o "Hexadecimal) equivalent of the binary representation of bit polynomials from **GF**(2)[x]. The Mix Columns step can also be

### Fig 3: Mix coloumns

viewed as a multiplication by a particular [MDS matrix](http://en.wikipedia.org/wiki/MDS_matrix" \o "MDS matrix) in a [finite field](http://en.wikipedia.org/wiki/Finite_field" \o "Finite field). This process is described further in the article [Rijndael mix columns](http://en.wikipedia.org/wiki/Rijndael_mix_columns" \o "Rijndael mix columns).

### 2.7 THE ROUNDKEY STEP:

In the AddRoundKey step, the sub key is combined with the state. For each round, a sub key is derived from the main [key](http://en.wikipedia.org/wiki/Key_(cryptography)" \o "Key (cryptography)) using [Rijndael's key schedule](http://en.wikipedia.org/wiki/Rijndael_key_schedule" \o "Rijndael key schedule); each sub key is the same size as the state. The sub key is added by combining each byte of the state with the corresponding byte of the sub key using bit wise [XOR](http://en.wikipedia.org/wiki/Exclusive_or" \o "Exclusive or).

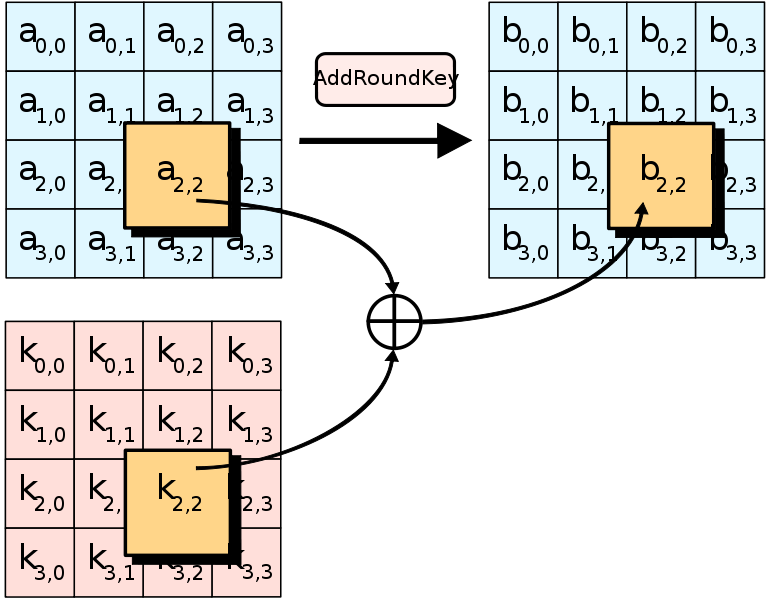


Fig 4: round key

### 2.8 OPTIMIZATION OF THE CIPHER:

On systems with 32-bit or larger words, it is possible to speed up execution of this cipher by combining Sub Bytes and Downshift with Mix Columns, and transforming them into a sequence of table lookups. This requires four 256-entry 32-bit tables, which utilizes a total of four kilobytes (4096 bytes) of memory-one kilobyte for each table. A round can now be done with 16 table lookups and 12 32-bit exclusive-or operations, followed by four 32-bit exclusive-or operations in the AddRoundKey step.

If the resulting four kilobyte table size is too large for a given target platform, the table lookup operation can be performed with a single 256-entry 32-bit (i.e. 1 kilobyte) table by the use of circular rotates.

**2.9 Security:**

Until May 2009, the only successful published attacks against the full AES were [side-channel attacks](http://en.wikipedia.org/wiki/Side-channel_attack" \o "Side-channel attack) on some specific implementations. The [National Security Agency](http://en.wikipedia.org/wiki/National_Security_Agency" \o "National Security Agency) (NSA) reviewed all the AES finalists, including Rijndael, and stated that all of them were secure enough for U.S. Government non-classified data. In June 2003, the U.S. Government announced that AES may be used to protect [classified information](http://en.wikipedia.org/wiki/Classified_information" \o "Classified information). The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use."

AES has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. By 2006, the best known attacks were on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys.

**2.10 NIST/CSEC VALIDATION:**

The [Cryptographic Module Validation Program](http://en.wikipedia.org/wiki/CMVP" \o "CMVP) (CMVP) is operated jointly by the United States Government's [National Institute of Standards and Technology](http://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology" \o "National Institute of Standards and Technology) (NIST) Computer Security Division and the [Communications Security Establishment](http://en.wikipedia.org/wiki/Communications_Security_Establishment" \o "Communications Security Establishment) (CSE) of the Government of Canada. The use of validated cryptographic modules is not required by the United States Government for unclassified uses of cryptography. The Government of Canada also recommends the use of [FIPS 140](http://en.wikipedia.org/wiki/FIPS_140" \o "FIPS 140) validated cryptographic modules in unclassified applications of its departments.

Although NIST publication 197 ("FIPS 197") is the unique document that covers the AES algorithm, vendors typically approach the CMVP under FIPS 140 and ask to have several algorithms (such as Triple DES or [SHA1](http://en.wikipedia.org/wiki/SHA1" \o "SHA1)) validated at the same time.

Therefore, it is rare to find cryptographic modules that are uniquely FIPS 197 validated and NIST itself does not generally take the time to list FIPS 197 validated modules separately on its public web site. Instead, FIPS 197 validation is typically just listed as an "FIPS approved: AES" notation (with a specific FIPS 197 certificate number) in the current list of FIPS 140 validated cryptographic modules.

The Cryptographic Algorithm Validation Program (CAVP) allows for independent validation of the correct implementation of the AES algorithm at a reasonable cost. Successful validation results in being listed on the NIST validations page. This testing is a pre-requisite for the FIPS 140-2 module validation described below.

**CHAPTER 3**

**Area-Optimized AES-128**

**3.1 BRIEF DESCRIPTION OF RIJNDAEL ALGORITHM:**

****Rijndael algorithm consists of encryption, decryption and key schedule algorithm. The main operations of the encryption algorithm among the three parts of Rijndael algorithm include: bytes substitution (Sub Bytes), the row shift (Shift Rows), column mixing (MixColumns), and the round key adding (AddRoundKey). It is shown as Fig5.

Figure 5: The structure of Rijndael encryption algorithm

Encryption algorithm processes Nr+1 rounds of transformation of the plain text for the cipher text. The value of Nr in AES algorithm whose packet length is 128 bits should be 10, 12, or 14 respectively, corresponding to the key length of 128,192,256 bits. In this paper, only the (AES-128) encryption scheme with 128-bit keys is considered.

**3.2 THE DESIGN OF IMPROVED AES-128 ENCRYPTION ALGORITHM:**

**1) Two main processes of AES encryption algorithm:**

The AES encryption algorithm can be divided into two parts, the key schedule and round transformation. Key schedule consists of two modules: key expansion and round key selection. Key expansion means mapping Nk bits initial key to the so-called expanded key, while the round key selection selects Nb bits of round key from the expanded key module.

Round Transformation involves four modules by Byte Substitution, Byte Rotation, Mix Column and AddRoundKey.

2)**Key points for the design:**

In the AES-128, the data in the main process mentioned above is mapped to a 4×4 two-dimensional matrix. The matrix is also called state matrix. In the four transformation modules of round transformation, the Byte Rotation, Mix Column and AddRoundKey are all linear transformations except the Byte Sub.

****Fig 6: The state matrix

**3)Take analysis of the AES algorithm principle and we can find:**

Byte Substitution operation simply replaces the element of 128-bit input plain text with the inverse element corresponding to the Galois field GF (28), whose smallest unit of operation is 8 bits/ group.

Byte Rotation operation takes cyclic shifts of the 128-bit state matrix, in which one row (32 bits) is taken as the smallest operand.

****Mix Columns operation takes multiplication and addition operations of the results of Byte Rotation with the corresponding irreducible polynomial x8 + x4 + x3 + x + 1 in GF(28), whose minimum operating unit is 32 bits.

Addroundkey operation takes a simple XOR operation with 8-bit units.

The inputs of plain text and initial key, intermediate inputs and outputs of round transformation, as well as the output of cipher text in the AES algorithm are all stored in the state matrices, which are processed in one byte or one word. Thus, in order to take operations at least bits, the original 128-bit data should be segmented. We design

Fig 7: Bytes segmentation and replacement processing

some external controllers in the new algorithm, so that the data transmission and processing can be implemented on each column of the state matrix (32bit).

**B. That means the data should be packed and put into further operations:**

Take the independent and reversible bytes substitution operation of S-box as example. Firstly, the state matrix is divided into four columns. And then byte replacement is achieved by the operation of look-up table shown as Fig. 3.

Therefore, the original 128-bit input of plain text and key will be replaced with four consecutive 32-bit input sequences respectively. In order to decrease the output ports, four continuous 32-bit cipher text sequences have taken place of the original 128-bit output by adding a clock controller. The 128- bit data in the round transformation is also split into four groups of 32-bit data before the operation of pipelining.

**The process of new algorithms:**

****From the above analysis, we can find that the process of AES encryption can be mainly divided into two parts: key schedule and round transformation. The improved structure is also divided into these two major processes. The initial key will be sent to the two modules: Key expansion and Key selection, while the plain text is to be sent to the round transformation after the round key is selected. But

Figure 8: The new improved structure of AES algorithm

the operand of data transmission is turned into a 32-bit unit.

The functions of various parts of the structure shown above are described as follow

* **The initial round of encryption**:

The four packets of consecutive 32-bit plain text (128 bits) have been put into the corresponding registers. Meanwhile, another four packets of consecutive 32-bit initial key (128 bits) have been put into other registers by the control of the enable clock signal. Furthermore, this module should combine the plain text and initial key by using the XOR operators.

* **Round Transformation in the intermediate steps:**

A round transformation mainly realizes the function of Sub Bytes and MixColumns with 32-bit columns. Four packets of round transformation are processed independently. Then the results of MixColumns and the 32-bit keys sourced from Key expansion are combined by using XOR operators.

****The implementation of Mix Column is mainly based on the mathematical analysis in the Galois field GF(28). Only the multiplication module and the 32-bit

Figure 9: The round processing with pipeline technology

XOR module of each processing unit (one column) are needed to design, because the elements of the multiplication and addition in Galois field are commutative and associative. Then the function of Mix Column can be achieved. In the process of pipelining, the 128-bit data is divided into four consecutive 32-bit packets that take round transformation independently.

The operation of the above four groups of data can be realized in pipelining technology. In brief, it can be described as follow: store the unprocessed data in the 128-bit register, and control the clock for re-starting the 128-bit register to read the new data when the four groups’ operations have been overcome. Thus the 128-bit round-operating unit has been transformed into four 32-bit round-operating elements.

The internal pipelining processing should be implemented during the whole nine intermediate Round Transformations of the four packets before achieving the 128-bit cipher text.

* **The process of the last round:**

The final round is a 128-bit processor. After nine rounds of operations included Shift rows, Sub Byte and Mix columns, the 128-bit intermediate encrypted data will be used in XOR operation with the final expanded key (4\*32bit), which is provided by the key expansion module. The output of final round in the processor is the desired 128-bit cipher text. Similarly, the cipher text is divided into four packets of 32- bit data by an external enable signal.

* **Key expansion and Key extraction:**

This module is implemented basically the same with the traditional way as another part of the AES encryption algorithm. The only difference lies on the mode of data transmission. The initial key and expanded keys are divided into four 32-bit data before being extracted.

All of the above modules can be decomposed into basic operations of seeking and XOR if the AES algorithm is implemented on FPGA. So the basic processing unit (look-up table) of FPGA can be used. The operation of AddRoundKey is taken

**CHAPTER 4**

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.

**4.1 OVERVIEW:**

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 common place 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.

**4.2 WHAT IS VLSI?**

VLSI stands for "Very Large Scale Integration". This is the field which involves packing more and more logic devices into smaller and smaller areas.

## VLSI:

## Simply we say Integrated circuit is many transistors on one chip.

## Design/manufacturing of extremely small, complex circuitry using modified

## semiconductor material

## Integrated circuit (IC) may contain millions of transistors, each a few mm in size

## Applications wide ranging: most electronic logic devices

# 4.3 HISTORY OF SCALE INTEGRATION:

* late 40s Transistor invented at Bell Labs
* late 50s First IC (JK-FF by Jack Kilby at TI)
* early 60s Small Scale Integration (SSI) 10s of transistors on a chip
* late 60s Medium Scale Integration (MSI) 100s of transistors on a chip
* early 70s Large Scale Integration (LSI) 1000s of transistor on a chip
* early 80s VLSI 10,000s of transistors on a chip (later 100,000s & now 1,000,000s)
* Ultra LSI is sometimes used for 1,000,000s
* 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)

### 4.4 ADVANTAGES OF ICS OVER DISCRETE COMPONENTS:

While we will concentrate on integrated circuits, the properties of integrated circuits-what we can and cannot efficiently put in an integrated circuit-largely determine the architecture of the entire system. Integrated circuits improve system characteristics in several critical ways. ICs have three key advantages over digital circuits built from discrete components:

* **Size:** Integrated circuits are much smaller-both transistors and wires are shrunk

micrometer sizes, compared to the millimeter or centimeter scales of discrete components. Small size leads to advantages in speed and power consumption, since smaller components have smaller parasitic resistances, capacitances, and inductances.

* **Speed:** Signals can be switched between logic 0 and logic 1 much quicker within

chip than they can between chips. Communication within a chip can occur hundreds of times faster than communication between chips on a printed circuit board. The high speed of circuits on-chip is due to their small size-smaller components and wires have smaller parasitic capacitances to slow down the signal.

* **Power consumption:** Logic operations within a chip also take much less power.

Once again, lower power consumption is largely due to the small size of circuits on the chip-smaller parasitic capacitances and resistances require less power to drive them.

#### 4.5 VLSI AND SYSTEMS:

These advantages of integrated circuits translate into advantages at the system level:

* **Smaller physical size:** Smallness is often an advantage in itself-consider portable televisions or handheld cellular telephones.
* **Lower power consumption:** Replacing a handful of standard parts with a single

chip reduces total power consumption. Reducing power consumption has a ripple effect on the rest of the system: a smaller, cheaper power supply can be used; since less power consumption means less heat, a fan may no longer be necessary; a simpler cabinet with less shielding for electromagnetic shielding may be feasible, too.

* **Reduced cost:** Reducing the number of components, the power supply and the

requirements, cabinet costs, and so on, will inevitably reduce system cost. The ripple effect of integration is such that the cost of a system built from custom ICs can be less, even though the individual ICs cost more than the standard parts they replace.

##### **Applications:**

* Electronic system in cars.
* Digital electronics control VCRs
* Transaction processing system, ATM
* Personal computers and Workstations
* Medical electronic systems.

### 4.6 APPLICATIONS OF VLSI:

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 visible, some more hidden:

* Personal entertainment systems such as portable MP3 players and DVD players perform sophisticated algorithms with remarkably little energy.
* 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 (ABS) systems.
* 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.
* Personal computers and workstations provide word-processing, financial analysis, and games. Computers include both central processing units (CPUs) and special-purpose hardware for disk access, faster screen display, *etc*.
* 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.

### 4.7 ASIC:

An Application-Specific Integrated Circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed solely to run a cell phone is an ASIC. Intermediate between ASICs and industry standard integrated circuits, like the 7400 or the 4000 series, are application specific standard products (ASSPs).

As feature sizes have shrunk and design tools improved over the years, the maximum complexity (and hence functionality) possible in an ASIC has grown from 5,000 gates to over 100 million. Modern ASICs often include entire 32-bit processors, memory blocks including ROM, RAM, EEPROM, Flash and other large building blocks. Such an ASIC is often termed a SoC (system-on-a-chip). Designers of digital ASICs use a hardware description language (HDL), such as Verilog or VHDL, to describe the functionality of ASICs.

Field-programmable gate arrays (FPGA) are the modern-day technology for building a breadboard or prototype from standard parts; programmable logic blocks and programmable interconnects allow the same FPGA to be used in many different applications. For smaller designs and/or lower production volumes, FPGAs may be more cost effective than an ASIC design even in production.

* An application-specific integrated circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use.
* A Structured ASIC falls between an FPGA and a Standard Cell-based ASIC.
* Structured ASIC’s are used mainly for mid-volume level design. The design task for structured ASIC’s is to map the circuit into a fixed arrangement of known cells.

## CHAPTER 5

## XILINX

**5.1 Migrating Projects from Previous ISE Software Releases:**

When you open a project file from a previous release, the ISE® software prompts you to migrate your project. If you click Backup and Migrate or Migrate only, the software automatically converts your project file to the current release. If you click Cancel, the software does not convert your project and, instead, opens Project Navigator with no project loaded.

**Note:**After you convert your project, you cannot open it in previous versions of the ISE software, such as the ISE 11 software. However, you can optionally create a backup of the original project as part of project migration, as described below.

**To Migrate a Project**

1. In the ISE 12 Project Navigator, select **File > Open Project**.
2. In the Open Project dialog box, select the .xise file to migrate.

**Note:**You may need to change the extension in the Files of type field to display .npl (ISE 5 and ISE 6 software) or .ise (ISE 7 through ISE 10 software) project files.

1. In the dialog box that appears, select **Backup and Migrate** or **Migrate**

**Only**.

1. The ISE software automatically converts your project to an ISE 12 project.

**Note:** If you chose to Backup and Migrate, a backup of the original project is created at project\_name\_ise12migration.zip.

1. Implement the design using the new version of the software.

**Note:**Implementation status is not maintained after migration.

**5.2 Properties:**

For information on properties that have changed in the ISE 12 software, see [ISE 11 to ISE 12 Properties Conversion](ise_r_properties_conversion.htm).

**5.3 IP Modules:**

If your design includes IP modules that were created using CORE Generator™ software or Xilinx® Platform Studio (XPS) and you need to modify these modules, you may be required to update the core. However, if the core netlist is present and you do not need to modify the core, updates are not required and the existing netlist is used during implementation.

**5.4 Obsolete Source File Types:**

The ISE 12 software supports all of the source types that were supported in the ISE 11 software. If you are working with projects from previous releases, state diagram source files (.dia), ABEL source files (.abl), and test bench waveform source files (.tbw) are no longer supported. For state diagram and ABEL source files, the software finds an associated HDL file and adds it to the project, if possible.

**5.5 Using ISE Example Projects:**

To help familiarize you with the ISE® software and with FPGA and CPLD designs, a set of example designs is provided with Project Navigator. The examples show different design techniques and source types, such as VHDL, Verilog, schematic, or EDIF, and include different constraints and IP.

**To Open an Example**

1. Select **File > Open Example**.
2. In the [Open Example dialog box](pn_db_open_example_project.htm), select the Sample Project Name.

**Note**To help you choose an example project, the Project Description field describes each project. In addition, you can scroll to the right to see additional fields, which provide details about the project.

1. In the Destination Directory field, enter a directory name or browse to the directory.
2. Click **OK**.

**Note:** If you modified an example project and want to overwrite it with the original example project, select **File > Open Example**, select the Sample Project Name, and specify the same Destination Directory you originally used. In the dialog box that appears, select **Overwrite the existing project** and click **OK**.

**5.6 Creating a Project:**

Project Navigator allows you to manage your FPGA and CPLD designs using an ISE® project, which contains all the source files and settings specific to your design. First, you must create a project and then, add source files, and set process properties. After you create a project, you can run processes to implement, constrain, and analyze your design. Project Navigator provides a wizard to help you create a project as follows.

**Note:**If you prefer, you can create a project using the **[New Project dialog box](pn_db_new_project.htm)** instead of the New Project Wizard. To use the New Project dialog box, deselect the **Use New Project wizard** option in the **[ISE General page](pn_db_ise_general_options.htm)** of the Preferences dialog box.

**To Create a Project**

1. Select **File > New Project** to launch the New Project Wizard.
2. In the **[Create New Project page](pn_db_npw_create_new_project.htm),** set the name, location, and project type, and click **Next**.
3. For EDIF or NGC/NGO projects only: In the **[Import EDIF/NGC Project page](pn_db_npw_import_edif_ngc_project.htm)**, select the input and constraint file for the project, and click **Next**.
4. In the **[Project Settings page](pn_db_npw_device_properties.htm)**, set the device and project properties, and click **Next**.
5. In the **[Project Summary page](pn_db_npw_project_summary.htm)**, review the information, and click **Finish** to create the project.

Project Navigator creates the project file (project\_name.xise) in the directory you specified. After you add source files to the project, the files appear in the Hierarchy pane of the

**5.7 [Design panel](pn_r_design_panel.htm)**:

Project Navigator manages your project based on the design properties (top-level module type, device type, synthesis tool, and language) you selected when you created the project. It organizes all the parts of your design and keeps track of the processes necessary to move the design from design entry through implementation to programming the targeted Xilinx® device.

**Note:**For information on changing design properties, see **[Changing Design Properties](pn_p_changing_design_properties.htm).**

**You can now perform any of the following:**

* Create new source files for your project.
* Add existing source files to your project.
* Run processes on your source files.

**5.8 Creating a Copy of a Project:**

You can create a copy of a project to experiment with different source options and implementations. Depending on your needs, the design source files for the copied project and their location can vary as follows:

* Design source files are left in their existing location and the copied project points to these files.
* Design source files, including generated files, are copied and placed in a specified directory.
* Design source files, excluding generated files, are copied and placed in a specified directory.

**Copied projects are the same as other projects in both form and function. For example, you can do the following with copied projects:**

* Open the copied project using the File > Open Project menu command.
* View, modify, and implement the copied project.
* Use the Project Browser to view key summary data for the copied project and then, open the copied project for further analysis and implementation.

5.9 **[Using the Project Browser](ise_c_project_browser.htm)**:

**Note:** Alternatively, you can create an archive of your project, which puts all of the project contents into a ZIP file. Archived projects must be unzipped before being opened in Project Navigator. For information on archiving, see **[Creating a Project Archive](ise_c_project_archive.htm).**

**To Create a Copy of a Project**

1. Select **File > Copy Project**.
2. In the Copy Project dialog box, enter the **Name** for the copy.

**Note**The name for the copy can be the same as the name for the project, as long as you specify a different location.

1. Enter a directory **Location** to store the copied project.
2. Optionally, enter a **Working directory**.
3. Optionally, enter a **Description** for the copy.
4. In the Source options area, do the following:

Select one of the following options:

* **Keep sources in their current locations -** to leave the design source files in their existing location. If you select this option, the copied project points to the files in their existing location.
* **Copy sources to the new location -** to make a copy of all the design source files and place them in the specified Location directory. If you select this option, the copied project points to the files in the specified directory. If you edit the files in the copied project, the changes do not appear in the original project, because the source files are not shared between the two projects.

Optionally, select **Copy files from Macro Search Path directories** to copy files from the directories you specify in the Macro Search Path property in the **[Translate Properties](pp_db_translate_properties.htm)** dialog box. All files from the specified directories are copied, not just the files used by the design.

**Note:**If you added a netlist source file directly to the project as described in **[Working with Netlist-Based IP](ise_c_using_fixed_netlist_ip.htm)**, the file is automatically copied as part of Copy Project because it is a project source file.

Optionally, click **Copy Additional Files** to copy files that were not included in the original project. In the Copy Additional Files dialog box, use the **Add Files** and **Remove Files** buttons to update the list of additional files to copy. Additional files are copied to the copied project location after all other files are copied.

To exclude generated files from the copy, such as implementation results and reports, select.

**5.10 Exclude generated files from the copy**:

When you select this option, the copied project opens in a state in which processes have not yet been run.

1. To automatically open the copy after creating it, select **Open the copied project**.

**Note:**By default, this option is disabled. If you leave this option disabled, the original project remains open after the copy is made.

* Click **OK**.

**5.11 Creating a Project Archive:**

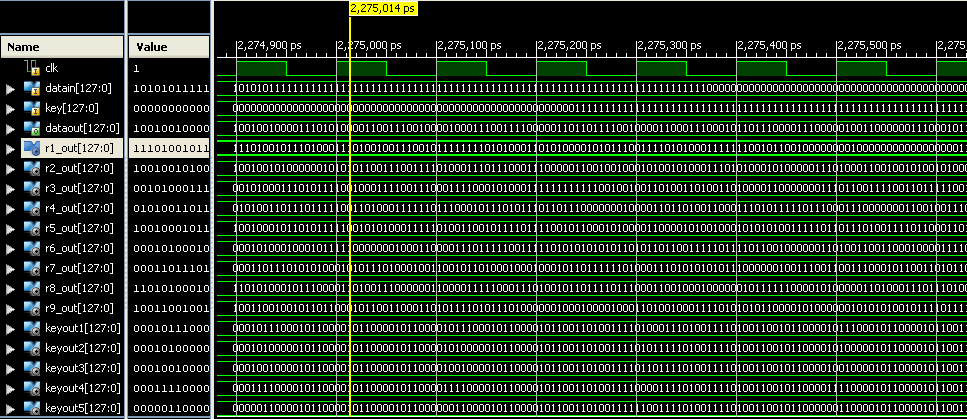
A project archive is a single, compressed ZIP file with a .zip extension. By default, it contains all project files, source files, and generated files, including the following:

* User-added sources and associated files
* Remote sources
* Verilog `include files
* Files in the macro search path
* Generated files
* Non-project files

**5.12 To Archive a Project:**

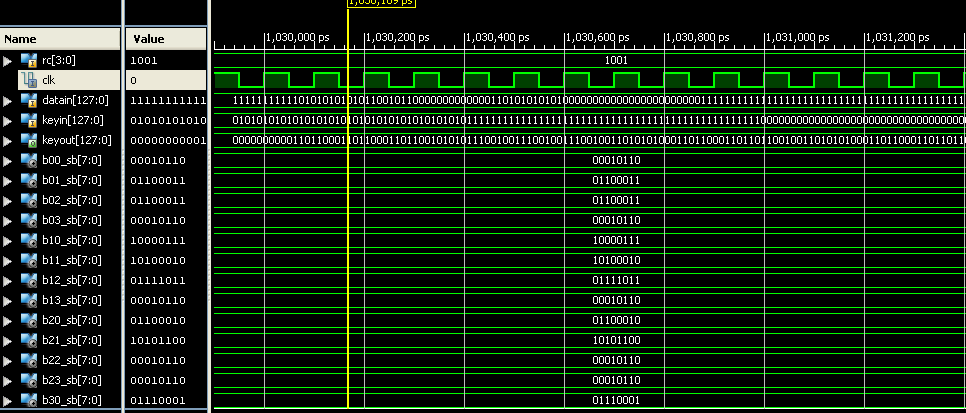
1. Select **Project > Archive**.
2. In the Project Archive dialog box, specify a file name and directory for the ZIP file.
3. Optionally, select **Exclude generated files from the archive** to exclude generated files and non-project files from the archive.
4. Click **OK**.

A ZIP file is created in the specified directory. To open the archived project, you must first unzip the ZIP file, and then, you can open the project.

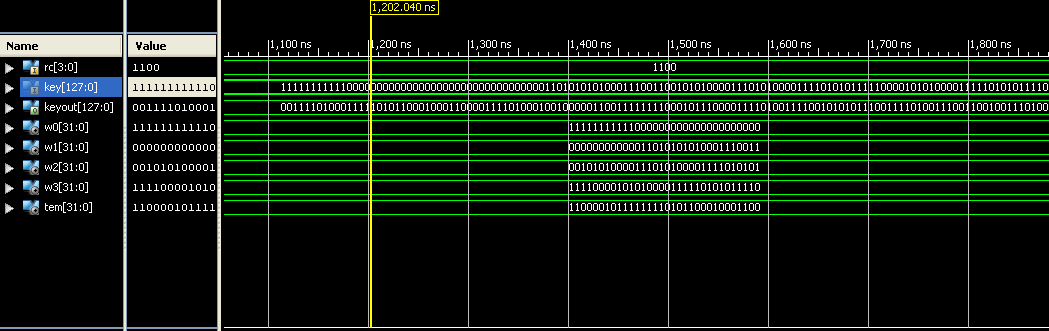
**CHAPTER 6**

**6.1 SIMULATION FOR AES:**

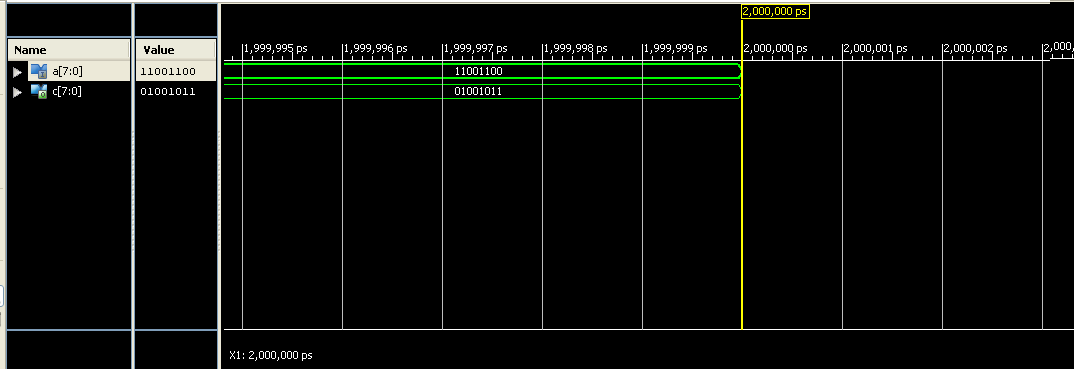
**6.2 SIMULATION FOR ROUNDS:**

****

**6.3 SIMULATION FOR KEY EXPANSION:**

****

**6.4 SIMULATION FOR AES\_SBOX:**

****

**SOURCE CODE**

**module aes(clk,datain,key,dataout);**

**input clk;**

**input [127:0] datain;**

**input [127:0] key;**

**output [127:0] dataout;**

**reg [127:0]r0\_out;**

**wire [127:0] r1\_out,r2\_out,r3\_out,r4\_out,r5\_out,r6\_out,r7\_out,r8\_out,r9\_out;**

**wire[127:0]keyout1,keyout2,keyout3,keyout4,keyout5,keyout6,keyout7,keyout8,keyout9,keyout1,keyout0;**

**always@(posedge clk) r0\_out=(datain^key);**

**rounds r1(.clk(clk),.rc(4'b0000),.datain(r0\_out),.keyin(key),.keyout(keyout1),.rndout(r1\_out));**

**rounds r2(.clk(clk),.rc(4'b0001),.datain(r1\_out),.keyin(key),.keyout(keyout2),.rndout(r2\_out));**

**rounds r3(.clk(clk),.rc(4'b0010),.datain(r2\_out),.keyin(key),.keyout(keyout3),.rndout(r3\_out));**

**rounds r4(.clk(clk),.rc(4'b0011),.datain(r3\_out),.keyin(key),.keyout(keyout4),.rndout(r4\_out));**

**rounds r5(.clk(clk),.rc(4'b0100),.datain(r4\_out),.keyin(key),.keyout(keyout5),.rndout(r5\_out));**

**rounds r6(.clk(clk),.rc(4'b0101),.datain(r5\_out),.keyin(key),.keyout(keyout6),.rndout(r6\_out));**

**rounds r7(.clk(clk),.rc(4'b0110),.datain(r6\_out),.keyin(key),.keyout(keyout7),.rndout(r7\_out));**

**rounds r8(.clk(clk),.rc(4'b0111),.datain(r7\_out),.keyin(key),.keyout(keyout8),.rndout(r8\_out));**

**rounds r9(.clk(clk),.rc(4'b1000),.datain(r8\_out),.keyin(key),.keyout(keyout9),.rndout(r9\_out));**

**roundlast r10(.clk(clk),.rc(4'b1001),.datain(r9\_out),.keyin(keyout9),.keyout(keyout10),.rlastout(dataout));**

**endmodule**

**Round program:**

**module rounds(clk,rc,datain,keyin,keyout,rndout);**

**input [3:0] rc;**

**input clk;**

**input [127:0] datain;**

**input [127:0] keyin;**

**output [127:0] keyout;**

**output[127:0] rndout;**

**wire [127:0] datain,keyin;**

**reg [127:0] rndout;**

**wire [7:0] b00\_sb,b01\_sb,b02\_sb,b03\_sb;**

**wire [7:0] b10\_sb,b11\_sb,b12\_sb,b13\_sb;**

**wire [7:0] b20\_sb,b21\_sb,b22\_sb,b23\_sb;**

**wire [7:0] b30\_sb,b31\_sb,b32\_sb,b33\_sb;**

**reg [7:0] b00\_sr,b01\_sr,b02\_sr,b03\_sr;**

**reg [7:0] b10\_sr,b11\_sr,b12\_sr,b13\_sr;**

**reg [7:0] b20\_sr,b21\_sr,b22\_sr,b23\_sr;**

**reg [7:0] b30\_sr,b31\_sr,b32\_sr,b33\_sr;**

**reg [7:0] b00\_mcl,b01\_mcl,b02\_mcl,b03\_mcl;**

**reg [7:0] b10\_mcl,b11\_mcl,b12\_mcl,b13\_mcl;**

**reg [7:0] b20\_mcl,b21\_mcl,b22\_mcl,b23\_mcl;**

**reg [7:0] b30\_mcl,b31\_mcl,b32\_mcl,b33\_mcl;**

**key\_exp k1(.rc(rc),.key(keyin),.keyout(keyout));**

**always@(posedge clk)**

**begin**

**b00\_sr = b00\_sb;**

**b01\_sr = b01\_sb;**

**b02\_sr = b02\_sb;**

**b03\_sr = b03\_sb;**

**b10\_sr = b11\_sb;**

**b11\_sr = b12\_sb;**

**b12\_sr = b13\_sb;**

**b13\_sr = b10\_sb;**

**b20\_sr = b22\_sb;**

**b21\_sr = b23\_sb;**

**b22\_sr = b20\_sb;**

**b23\_sr = b21\_sb;**

**b30\_sr = b33\_sb;**

**b31\_sr = b30\_sb;**

**b32\_sr = b31\_sb;**

**b33\_sr = b32\_sb;**

**b00\_mcl = xtime(b00\_sr)^xtime(b10\_sr)^(b10\_sr)^(b20\_sr)^(b30\_sr);**

**b10\_mcl = (b00\_sr)^xtime(b10\_sr)^xtime(b20\_sr)^(b20\_sr)^(b30\_sr);**

**b20\_mcl = (b00\_sr)^(b10\_sr)^xtime(b20\_sr)^xtime(b30\_sr)^(b30\_sr);**

**b30\_mcl = xtime(b00\_sr)^(b00\_sr)^(b10\_sr)^(b20\_sr)^xtime(b30\_sr);**

**b01\_mcl = xtime(b01\_sr)^xtime(b11\_sr)^(b11\_sr)^(b21\_sr)^(b31\_sr);**

**b11\_mcl = (b01\_sr)^xtime(b11\_sr)^xtime(b21\_sr)^(b21\_sr)^(b31\_sr);**

**b21\_mcl = (b01\_sr)^(b11\_sr)^xtime(b21\_sr)^xtime(b31\_sr)^(b31\_sr);**

**b31\_mcl = xtime(b01\_sr)^(b01\_sr)^(b11\_sr)^(b21\_sr)^xtime(b31\_sr);**

**b02\_mcl = xtime(b02\_sr)^xtime(b12\_sr)^(b12\_sr)^(b22\_sr)^(b32\_sr);**

**b12\_mcl = (b02\_sr)^xtime(b12\_sr)^xtime(b22\_sr)^(b22\_sr)^(b32\_sr);**

**b22\_mcl = (b02\_sr)^(b12\_sr)^xtime(b22\_sr)^xtime(b32\_sr)^(b32\_sr);**

**b32\_mcl = xtime(b02\_sr)^(b02\_sr)^(b12\_sr)^(b22\_sr)^xtime(b32\_sr);**

**b03\_mcl = xtime(b03\_sr)^xtime(b13\_sr)^(b13\_sr)^(b23\_sr)^(b33\_sr);**

**b13\_mcl = (b03\_sr)^xtime(b13\_sr)^xtime(b23\_sr)^(b23\_sr)^(b33\_sr);**

**b23\_mcl = (b03\_sr)^(b13\_sr)^xtime(b23\_sr)^xtime(b33\_sr)^(b33\_sr);**

**b33\_mcl = xtime(b03\_sr)^(b03\_sr)^(b13\_sr)^(b23\_sr)^xtime(b33\_sr);**

**rndout[127:120] = b00\_mcl ^ keyout[127:120];**

**rndout[119:112] = b10\_mcl ^ keyout[119:112];**

**rndout[111:104] = b20\_mcl ^ keyout[111:104];**

**rndout[103:96] = b30\_mcl ^ keyout[103:96];**

**rndout[95:88] = b01\_mcl ^ keyout[95:88];**

**rndout[87:80] = b11\_mcl ^ keyout[87:80];**

**rndout[79:72] = b21\_mcl ^ keyout[79:72];**

**rndout[71:64] = b31\_mcl ^ keyout[71:64];**

**rndout[63:56] = b02\_mcl ^ keyout[63:56];**

**rndout[55:48] = b12\_mcl ^ keyout[55:48];**

**rndout[47:40] = b22\_mcl ^ keyout[47:40];**

**rndout[39:32] = b32\_mcl ^ keyout[39:32];**

**rndout[31:24] = b03\_mcl ^ keyout[31:24];**

**rndout[23:16] = b13\_mcl ^ keyout[23:16];**

**rndout[15:08] = b23\_mcl ^ keyout[15:08];**

**rndout[7:0] = b33\_mcl ^ keyout[7:0];**

**end**

**aes\_sbox q0( .a(datain[127:120]),.c(b00\_sb) );**

**aes\_sbox q1( .a(datain[119:112]),.c(b10\_sb) );**

**aes\_sbox q2( .a(datain[111:104]),.c(b20\_sb) );**

**aes\_sbox q3( .a(datain[103:96]),.c(b30\_sb) );**

**aes\_sbox q4( .a(datain[95:88]),.c(b01\_sb) );**

**aes\_sbox q5( .a(datain[87:80]),.c(b11\_sb) );**

**aes\_sbox q6( .a(datain[79:72]),.c(b21\_sb) );**

**aes\_sbox q7( .a(datain[71:64]),.c(b31\_sb) );**

**aes\_sbox q8( .a(datain[63:56]),.c(b02\_sb) );**

**aes\_sbox q9( .a(datain[55:48]),.c(b12\_sb) );**

**aes\_sbox q10(.a(datain[47:40]),.c(b22\_sb) );**

**aes\_sbox q11(.a(datain[39:32]),.c(b32\_sb) );**

**aes\_sbox q12(.a(datain[31:24]),.c(b03\_sb) );**

**aes\_sbox q13(.a(datain[23:16]),.c(b13\_sb) );**

**aes\_sbox q14(.a(datain[15:08]),.c(b23\_sb) );**

**aes\_sbox q16(.a(datain[7:0]),.c(b33\_sb) );**

**function [7:0] xtime;**

**input [7:0] b;**

**begin**

**if ( b[7] == 0 )**

**xtime = {b[6:0],(1'b0)};**

**else**

**xtime = ({b[6:0],(1'b0)} ^ (8'b00011011));**

**end**

**endfunction**

**endmodule**

**Keyexpansion program :**

**module key\_exp (rc,key,keyout);**

**input [3:0] rc;**

**input [127:0]key;**

**output [127:0] keyout;**

**wire [31:0] w0,w1,w2,w3,tem;**

**assign w0 = key[127:96];**

**assign w1 = key[95:64];**

**assign w2 = key[63:32];**

**assign w3 = key[31:0];**

**assign keyout[127:96]= w0 ^ tem ^ rcon(rc);**

**assign keyout[95:64] = w0 ^ tem ^ rcon(rc)^ w1;**

**assign keyout[63:32] = w0 ^ tem ^ rcon(rc)^ w1 ^ w2;**

**assign keyout[31:0] = w0 ^ tem ^ rcon(rc)^ w1 ^ w2 ^ w3;**

**aes\_sbox a1(.a(w3[23:16]),.c(tem[31:24]));**

**aes\_sbox a2(.a(w3[15:8]),.c(tem[23:16]));**

**aes\_sbox a3(.a(w3[7:0]),.c(tem[15:8]));**

**aes\_sbox a4(.a(w3[31:24]),.c(tem[7:0]));**

**function [31:0] rcon;**

**input [3:0] rc;**

**case(rc)**

**4'h0: rcon=32'h01\_00\_00\_00;**

**4'h1: rcon=32'h02\_00\_00\_00;**

**4'h2: rcon=32'h04\_00\_00\_00;**

**4'h3: rcon=32'h08\_00\_00\_00;**

**4'h4: rcon=32'h10\_00\_00\_00;**

**4'h5: rcon=32'h20\_00\_00\_00;**

**4'h6: rcon=32'h40\_00\_00\_00;**

**4'h7: rcon=32'h80\_00\_00\_00;**

**4'h8: rcon=32'h1b\_00\_00\_00;**

**4'h9: rcon=32'h36\_00\_00\_00;**

**default: rcon=32'h00\_00\_00\_00;**

**endcase**

**endfunction**

**endmodule**

**AES\_SBOX:**

**module aes\_sbox(a,c);**

**input [7:0] a;**

**output [7:0] c;**

**reg [7:0] c;**

**always @(a)**

**case (a)**

**8'h00: c=8'h63;**

**8'h01: c=8'h7c;**

**8'h02: c=8'h77;**

**8'h03: c=8'h7b;**

**8'h04: c=8'hf2;**

**8'h05: c=8'h6b;**

**8'h06: c=8'h6f;**

**8'h07: c=8'hc5;**

**8'h08: c=8'h30;**

**8'h09: c=8'h01;**

**8'h0a: c=8'h67;**

**8'h0b: c=8'h2b;**

**8'h0c: c=8'hfe;**

**8'h0d: c=8'hd7;**

**8'h0e: c=8'hab;**

**8'h0f: c=8'h76;**

**8'h10: c=8'hca;**

**8'h11: c=8'h82;**

**8'h12: c=8'hc9;**

**8'h13: c=8'h7d;**

**8'h14: c=8'hfa;**

**8'h15: c=8'h59;**

**8'h16: c=8'h47;**

**8'h17: c=8'hf0;**

**8'h18: c=8'had;**

**8'h19: c=8'hd4;**

**8'h1a: c=8'ha2;**

**8'h1b: c=8'haf;**

**8'h1c: c=8'h9c;**

**8'h1d: c=8'ha4;**

**8'h1e: c=8'h72;**

**8'h1f: c=8'hc0;**

**8'h20: c=8'hb7;**

**8'h21: c=8'hfd;**

**8'h22: c=8'h93;**

**8'h23: c=8'h26;**

**8'h24: c=8'h36;**

**8'h25: c=8'h3f;**

**8'h26: c=8'hf7;**

**8'h27: c=8'hcc;**

**8'h28: c=8'h34;**

**8'h29: c=8'ha5;**

**8'h2a: c=8'he5;**

**8'h2b: c=8'hf1;**

**8'h2c: c=8'h71;**

**8'h2d: c=8'hd8;**

**8'h2e: c=8'h31;**

**8'h2f: c=8'h15;**

**8'h30: c=8'h04;**

**8'h31: c=8'hc7;**

**8'h32: c=8'h23;**

**8'h33: c=8'hc3;**

**8'h34: c=8'h18;**

**8'h35: c=8'h96;**

**8'h36: c=8'h05;**

**8'h37: c=8'h9a;**

**8'h38: c=8'h07;**

**8'h39: c=8'h12;**

**8'h3a: c=8'h80;**

**8'h3b: c=8'he2;**

**8'h3c: c=8'heb;**

**8'h3d: c=8'h27;**

**8'h3e: c=8'hb2;**

**8'h3f: c=8'h75;**

**8'h40: c=8'h09;**

**8'h41: c=8'h83;**

**8'h42: c=8'h2c;**

**8'h43: c=8'h1a;**

**8'h44: c=8'h1b;**

**8'h45: c=8'h6e;**

**8'h46: c=8'h5a;**

**8'h47: c=8'ha0;**

**8'h48: c=8'h52;**

**8'h49: c=8'h3b;**

**8'h4a: c=8'hd6;**

**8'h4b: c=8'hb3;**

**8'h4c: c=8'h29;**

**8'h4d: c=8'he3;**

**8'h4e: c=8'h2f;**

**8'h4f: c=8'h84;**

**8'h50: c=8'h53;**

**8'h51: c=8'hd1;**

**8'h52: c=8'h00;**

**8'h53: c=8'hed;**

**8'h54: c=8'h20;**

**8'h55: c=8'hfc;**

**8'h56: c=8'hb1;**

**8'h57: c=8'h5b;**

**8'h58: c=8'h6a;**

**8'h59: c=8'hcb;**

**8'h5a: c=8'hbe;**

**8'h5b: c=8'h39;**

**8'h5c: c=8'h4a;**

**8'h5d: c=8'h4c;**

**8'h5e: c=8'h58;**

**8'h5f: c=8'hcf;**

**8'h60: c=8'hd0;**

**8'h61: c=8'hef;**

**8'h62: c=8'haa;**

**8'h63: c=8'hfb;**

**8'h64: c=8'h43;**

**8'h65: c=8'h4d;**

**8'h66: c=8'h33;**

**8'h67: c=8'h85;**

**8'h68: c=8'h45;**

**8'h69: c=8'hf9;**

**8'h6a: c=8'h02;**

**8'h6b: c=8'h7f;**

**8'h6c: c=8'h50;**

**8'h6d: c=8'h3c;**

**8'h6e: c=8'h9f;**

**8'h6f: c=8'ha8;**

**8'h70: c=8'h51;**

**8'h71: c=8'ha3;**

**8'h72: c=8'h40;**

**8'h73: c=8'h8f;**

**8'h74: c=8'h92;**

**8'h75: c=8'h9d;**

**8'h76: c=8'h38;**

**8'h77: c=8'hf5;**

**8'h78: c=8'hbc;**

**8'h79: c=8'hb6;**

**8'h7a: c=8'hda;**

**8'h7b: c=8'h21;**

**8'h7c: c=8'h10;**

**8'h7d: c=8'hff;**

**8'h7e: c=8'hf3;**

**8'h7f: c=8'hd2;**

**8'h80: c=8'hcd;**

**8'h81: c=8'h0c;**

**8'h82: c=8'h13;**

**8'h83: c=8'hec;**

**8'h84: c=8'h5f;**

**8'h85: c=8'h97;**

**8'h86: c=8'h44;**

**8'h87: c=8'h17;**

**8'h88: c=8'hc4;**

**8'h89: c=8'ha7;**

**8'h8a: c=8'h7e;**

**8'h8b: c=8'h3d;**

**8'h8c: c=8'h64;**

**8'h8d: c=8'h5d;**

**8'h8e: c=8'h19;**

**8'h8f: c=8'h73;**

**8'h90: c=8'h60;**

**8'h91: c=8'h81;**

**8'h92: c=8'h4f;**

**8'h93: c=8'hdc;**

**8'h94: c=8'h22;**

**8'h95: c=8'h2a;**

**8'h96: c=8'h90;**

**8'h97: c=8'h88;**

**8'h98: c=8'h46;**

**8'h99: c=8'hee;**

**8'h9a: c=8'hb8;**

**8'h9b: c=8'h14;**

**8'h9c: c=8'hde;**

**8'h9d: c=8'h5e;**

**8'h9e: c=8'h0b;**

**8'h9f: c=8'hdb;**

**8'ha0: c=8'he0;**

**8'ha1: c=8'h32;**

**8'ha2: c=8'h3a;**

**8'ha3: c=8'h0a;**

**8'ha4: c=8'h49;**

**8'ha5: c=8'h06;**

**8'ha6: c=8'h24;**

**8'ha7: c=8'h5c;**

**8'ha8: c=8'hc2;**

**8'ha9: c=8'hd3;**

**8'haa: c=8'hac;**

**8'hab: c=8'h62;**

**8'hac: c=8'h91;**

**8'had: c=8'h95;**

**8'hae: c=8'he4;**

**8'haf: c=8'h79;**

**8'hb0: c=8'he7;**

**8'hb1: c=8'hc8;**

**8'hb2: c=8'h37;**

**8'hb3: c=8'h6d;**

**8'hb4: c=8'h8d;**

**8'hb5: c=8'hd5;**

**8'hb6: c=8'h4e;**

**8'hb7: c=8'ha9;**

**8'hb8: c=8'h6c;**

**8'hb9: c=8'h56;**

**8'hba: c=8'hf4;**

**8'hbb: c=8'hea;**

**8'hbc: c=8'h65;**

**8'hbd: c=8'h7a;**

**8'hbe: c=8'hae;**

**8'hbf: c=8'h08;**

**8'hc0: c=8'hba;**

**8'hc1: c=8'h78;**

**8'hc2: c=8'h25;**

**8'hc3: c=8'h2e;**

**8'hc4: c=8'h1c;**

**8'hc5: c=8'ha6;**

**8'hc6: c=8'hb4;**

**8'hc7: c=8'hc6;**

**8'hc8: c=8'he8;**

**8'hc9: c=8'hdd;**

**8'hca: c=8'h74;**

**8'hcb: c=8'h1f;**

**8'hcc: c=8'h4b;**

**8'hcd: c=8'hbd;**

**8'hce: c=8'h8b;**

**8'hcf: c=8'h8a;**

**8'hd0: c=8'h70;**

**8'hd1: c=8'h3e;**

**8'hd2: c=8'hb5;**

**8'hd3: c=8'h66;**

**8'hd4: c=8'h48;**

**8'hd5: c=8'h03;**

**8'hd6: c=8'hf6;**

**8'hd7: c=8'h0e;**

**8'hd8: c=8'h61;**

**8'hd9: c=8'h35;**

**8'hda: c=8'h57;**

**8'hdb: c=8'hb9;**

**8'hdc: c=8'h86;**

**8'hdd: c=8'hc1;**

**8'hde: c=8'h1d;**

**8'hdf: c=8'h9e;**

**8'he0: c=8'he1;**

**8'he1: c=8'hf8;**

**8'he2: c=8'h98;**

**8'he3: c=8'h11;**

**8'he4: c=8'h69;**

**8'he5: c=8'hd9;**

**8'he6: c=8'h8e;**

**8'he7: c=8'h94;**

**8'he8: c=8'h9b;**

**8'he9: c=8'h1e;**

**8'hea: c=8'h87;**

**8'heb: c=8'he9;**

**8'hec: c=8'hce;**

**8'hed: c=8'h55;**

**8'hee: c=8'h28;**

**8'hef: c=8'hdf;**

**8'hf0: c=8'h8c;**

**8'hf1: c=8'ha1;**

**8'hf2: c=8'h89;**

**8'hf3: c=8'h0d;**

**8'hf4: c=8'hbf;**

**8'hf5: c=8'he6;**

**8'hf6: c=8'h42;**

**8'hf7: c=8'h68;**

**8'hf8: c=8'h41;**

**8'hf9: c=8'h99;**

**8'hfa: c=8'h2d;**

**8'hfb: c=8'h0f;**

**8'hfc: c=8'hb0;**

**8'hfd: c=8'h54;**

**8'hfe: c=8'hbb;**

**8'hff: c=8'h16;**

**endcase**

**end**

**endmodule**

**APPLICATIONS:**

* Widely used for computer and communication network.

* Information security has aroused high attention.

* Used in military, political and diplomatic fields.

* Also applied to common fields of people’s daily lives.

**MERITS:**

* Requiems low space.

* Speed of operation is high.

* Requires low power consumption.

* Easy to implement.

**FUTURE SCOPE AND CONCLUSION:**

A FPGA implementation of area-optimized AES algorithm which meets the actual application is proposed in this paper.After being coded with Verilog Hardware Description Language, the waveform simulation of the new algorithm was taken in the ModelSim SE PLUS 6.0 and Quartus 7.2 platform. Ultimately, a synthesis simulation of the new algorithm has been done.

The result shows that the design with the pipelining technology and special data transmission mode can optimize the chip area effectively. Meanwhile, this design reduces power consumption to some extent, for the power consumption is directly related to the chip area. Therefore the encryption device implemented in this method can meet some practical applications.

As the S-box is implemented by look-up-table in this design, the chip area and power can still be optimized. So the future work should focus on the implementation mode of S-box. Mathematics in Galois field (28) can accomplish the bytes substitution of the AES algorithm, which could be another idea of further research.

**REFERENCES:**

[1] J.Yang, J.Ding, N.Li and Y.X.Guo, “FPGA-based design and implementation of reduced AES algorithm” IEEE Inter.Conf. Chal Envir

Sci Com Engin(CESCE).,Vol.02, Issue.5-6, pp.67-70, Jun 2010.

[2] A.M.Deshpande, M.S.Deshpande and D.N.Kayatanavar,“FPGA

Implementation of AES Encryption and Decryption”IEEE

Inter.Conf.Cont,Auto,Com,and Ener., vol.01,issue04, pp.1-6,Jun.2009.

[3] Hiremath.S. and Suma.M.S.,“Advanced Encryption Standard

Implemented on FPGA” IEEE Inter.Conf. Comp Elec

Engin.(IECEE),vol.02,issue.28,pp.656-660,Dec.2009.

[4] Abdel-hafeez.S.,Sawalmeh.A. and Bataineh.S.,“High Performance AES

Design using Pipelining Structure over GF(28)” IEEE Inter Conf.Signal

Proc and Com., vol.24-27, pp.716-719, Nov. 2007.

[5] Rizk.M.R.M. and Morsy, M., “Optimized Area and Optimized Speed

Hardware Implementations of AES on FPGA”, IEEE Inter Conf. Desig

Tes Wor., vol.1, issue.16, pp.207-217, Dec. 2007.

1. Liberatori.M.,Otero.F.,Bonadero.J.C. and Castineira.J. “AES-128 Cipher.High Speed, Low Cost FPGA Implementation”, IEEE Conf. SouthernProgrammable Logic (SPL), vol.04, issue.07, pp.195-198, Jun. 2007.

[7] Abdelhalim.M.B., Aslan.H.K. and Farouk.H. “A design for an FPGA based

implementation of Rijndael cipher”,ITICT. Ena TechSoc. (ETNKS), vol.5, issue.6, pp.897-912, Dec. 2005.