IJERMT

International Journal Of Engineering Research & Management Technology

ISSN: 2348-4039

Email: editor@ijermt.org March 2014 Volume-1, Issue-2 Website: ijermt.org

A VHDL Implementation of the Advanced Encryption Standard-Rijndael Algorithm

Himanshu Rajput, Prashant Mani Mohd. Suhaib Abbasi

Department of Electronics and Communication Engineering, S.R.M. University, Delhi NCR Campus, India

Abstract —
The importance of cryptography applied to security in electronic data transactions has acquired an essential relevance during the last few years. A VHDL-based implementation of the Advanced Encryption Standard (AES) algorithm is presented in this paper. The design has been coded by Very high speed integrated circuit Hardware Descriptive Language. All the results are synthesized using Xilinx ISE and simulated by using ModelSim software. The Advanced Encryption Standard can be programmed in software or built with pure hardware. This implementation is compared with other works to show the efficiency. All the transformations of both Encryptions and Decryption are simulated using an iterative design approach in order to minimize the hardware consumption. This research investigates the AES algorithm with regard to FPGA and the Very High Speed Integrated Circuit Hardware Description language (VHDL). Simulation results, performance results are presented and compared with previous reported designs.
The National Institute of Standards and Technology (NIST) has initiated a process to develop a Federal information Processing Standard (FIPS) for the Advanced Encryption Standard (AES), specifying an Advanced Encryption Algorithm to replace the Data Encryption standard (DES) the Expired in 1998. NIST has solicited candidate algorithms for inclusion in AES, resulting in fifteen official candidate algorithms of which Rijndael was chosen as the Advanced Encryption Standard.
The Advanced Encryption Standard can be programmed in software or built with pure hardware. However Field Programmable Gate Arrays (FPGAs) offer a quicker, more customizable solution.

Keywords - AES, FPGA, Encryption/Decryption, Block Cipher and VHDL

I. INTRODUCTION

The National Institute of Standards and Technology, (NIST), solicited proposals for the Advanced Encryption Standard, (AES). The AES is a Federal Information Processing Standard, (FIPS), which is a cryptographic algorithm that is used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt, (encipher), and decrypt, (decipher), information. Encryption converts data to an unintelligible form called cipher-text. Decryption of the cipher-text converts the data back into its original form, which is called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.
Many algorithms were originally presented by researchers from twelve different nations. Fifteen, (15), algorithms were selected from the first set of submittals. After a study and selection process five, (5), were chosen as finalists. The five algorithms selected were MARS, RC6, RIJNDAEL,

SERPENT and TWOFISH. The conclusion was that the five Competitors showed similar characteristics. On October 2nd 2000, NIST announced that the Rijndael Algorithm was the winner of the contest. The Rijndael Algorithm was chosen since it had the best overall scores in security, performance, efficiency, implementation ability and flexibility, [NIS00b]. The Rijndael algorithm was developed by Joan Daemen of Proton World International and Vincent Fijmen of Katholieke University at Leuven.[6]

The Rijndael algorithm is a symmetric block cipher that can process data blocks of 128 bits through the use of cipher keys with lengths of 128, 192, and 256 bits. The Rijndael algorithm was also designed to handle additional block sizes and key lengths. However, the additional features were not adopted in the AES. The hardware implementation of the Rijndael algorithm can provide either high performance or low cost for specific applications.

At backbone communication channels or heavily loaded servers it is not possible to lose processing speed, which drops the efficiency of the overall system while running cryptography algorithms in software. On the other side, a low cost and small design can be used in smart card applications, which allows a wide range of equipment to operate securely.[6]

A. Notation and Conventions

Inputs and Outputs

The input and output for the AES algorithm consists of sequences of 128 bits. These sequences are referred to as blocks and the numbers of bits they contain are referred to as their length. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and Cipher Key lengths are not permitted by this standard. The bits within such sequences are numbered starting at zero and ending at one less than the sequence length, which is also termed the block length or key length. The number "i" attached to a bit is known as its index and will be in one of the ranges $0 \leq i < 128$, $0 \leq i < 192$ or $0 \leq i < 256$ depending on the block length or key length specified.

Bytes

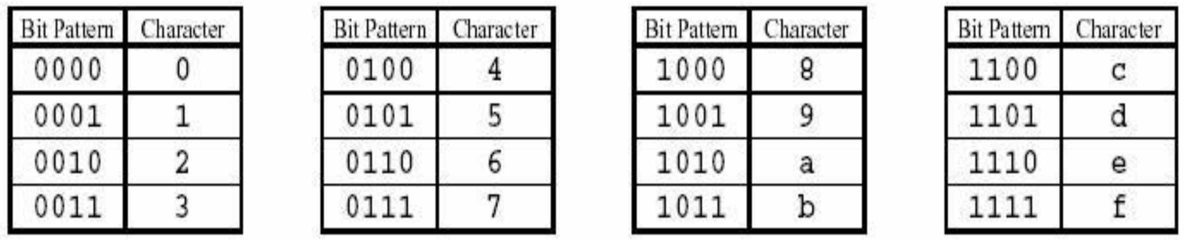
The basic unit of processing in the AES algorithm is a byte, which is a sequence of eight bits treated as a single entity. The input, output and Cipher Key bit sequences described in Section previously are processed as arrays of bytes that are formed by dividing these sequences into groups of eight contiguous bits to form arrays of bytes. For an input, output or Cipher Key denoted by a, the bytes in the resulting array are referenced using one of the two forms, a_n or a[n], where n will be in a range that depends on the key length. For a key length of 128 bits, n lies in the range $0 \leq n < 16$. For a key length of 192 bits, n lies in the range $0 \leq n < 24$. For a key length of 256 bits, n lies in the range $0 \leq n < 32$.

All byte values in the AES algorithm are presented as the concatenation of the individual bit values, (0 or 1), between braces in the order $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$.

These bytes are interpreted as finite field elements using a polynomial representation

$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0 = \sum b_i x^i$

For example, {01100011} identifies the specific finite field element $x^6 + x^5 + x + 1$. It is also convenient to denote byte values using hexadecimal notation with each of two groups of four bits being denoted by a single hexadecimal character. The hexadecimal notation scheme is depicted in Figure.1.



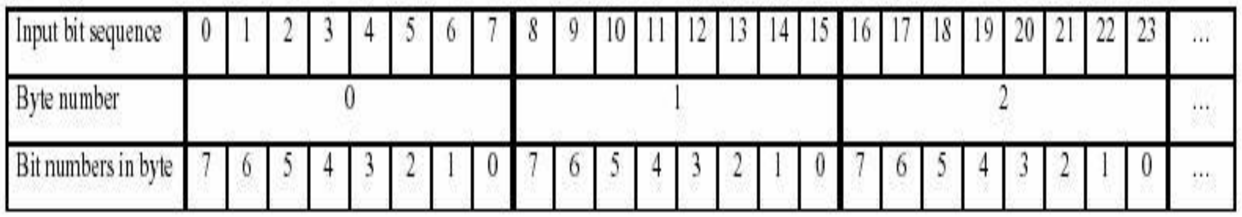
Bit Pattern	Character	Bit Pattern	Character	Bit Pattern	Character	Bit Pattern	Character
0000	0	0100	4	1000	8	1100	c
0001	1	0101	5	1001	9	1101	d
0010	2	0110	6	1010	a	1110	e
0011	3	0111	7	1011	b	1111	f

Figure 1.Hexadecimal Representation of Bit Patterns [1]

Hence the element {01100011} can be represented as {63}, where the character denoting the four-bit group containing the higher numbered bits is again to the left. Some finite field operations involve one additional bit $\{b_8\}$ to the left of an 8-bit byte. When the b_8 bit is present, it appears as {01} immediately preceding the 8-bit byte. For example, a 9-bit sequence is presented as {01} {1b}.

Arrays of Bytes

Arrays of bytes are represented in the form $a_0a_1a_2\cdots a_{15}$. The bytes and the bit ordering within bytes are derived from the 128-bit input sequence, $input_0input_1input_2\cdots input_{126}input_{127}$ as $a_0 = \{input_0, input_1, \cdots, input_7\}$, $a_1 = \{input_8, input_9, \cdots, input_{15}\}$ with the pattern continuing up to $a_{15} = \{input_{120}, input_{121}, \cdots, input_{127}\}$. The pattern can be extended to longer sequences associated with 192 and 256 bit keys. In general, $a_n = \{input_{8n}, input_{8n+1}, \cdots, input_{8n+7}\}$.
An example of byte designation and numbering within bytes for a given input sequence is presented in Figure 2.

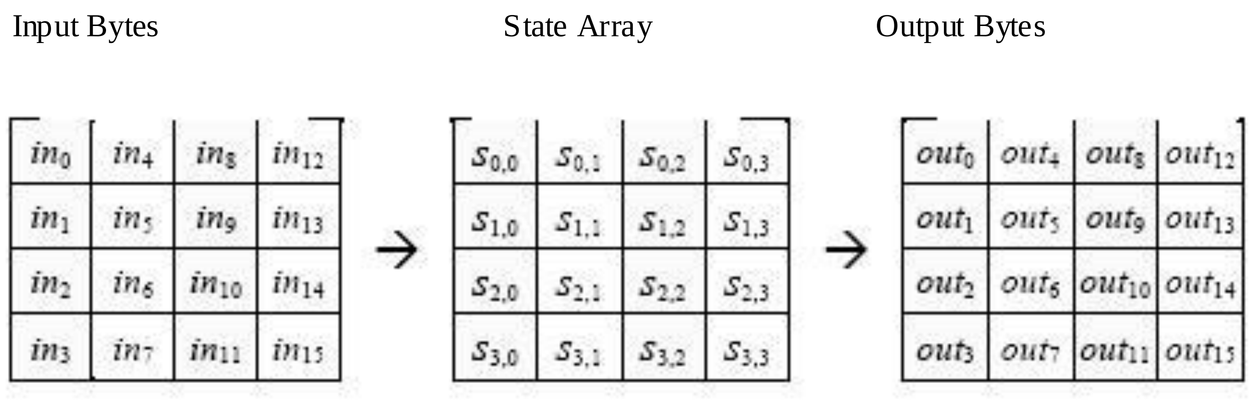


Input bit sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	…
Byte number	0								1								2								…
Bit numbers in byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…

Figure 2.Indices for Bytes and Bits [1]

The State

Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the State. The State consists of four rows of bytes. Each row of a state contains N_b numbers of bytes, where N_b is the block length divided by 32. In the State array, which is denoted by the symbol **S**, each individual byte has two indices. The first byte index is the row number **r**, which lies in the range $0 \leq r \leq 3$ and the second byte
index is the column number **c**, which lies in the range $0 \leq c \leq N_{b-1}$. Such indexing allows an individual byte of the State to be referred to as $S_{r,c}$ or $S_{[r,c]}$. For the AES $N_b = 4$, which means that $0 \leq c \leq 3$. At the beginning of the Encryption and Decryption the input, which is the array of bytes symbolized by $in_0in_1\cdots in_{15}$ is copied into the State array. This activity is illustrated in Figure 3.



in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

Figure 3.State Array Input and Output [1]

The State as an Array of Columns
The four bytes in each column of the State form 32-bit words, where the row number "r" provides an index for the four bytes within each word. Therefore, the state can be interpreted as a one-dimensional array of 32 bit words, which is symbolized by $w_0...w_3$. The column number **c** provides an index into this linear State array. Considering the State depicted in Figure3, the State can be considered as an array of four words where
w0 = S0,0 S1,0 S2,0 S3,0,
w1 = S0,1 S1,1 S2,1 S3,1,
w2 = S0,2 S1,2 S2,2 S3,2
and
w3 = S0,3 S1,3 S2,3 S3,3.

Mathematical Background
Every byte in the AES algorithm is interpreted as a finite field element using the notation introduced. All Finite field elements can be added and multiplied. However, these operations differ from those used for numbers and their use requires investigation.

Addition
The addition of two elements in a finite field is achieved by "adding" the coefficients for the corresponding powers in the polynomials for the two elements. The addition is performed through use of the XOR operation, which is denoted by the operator symbol $\oplus$. Such addition is performed modulo-2. In modulo-2 addition
$1 \oplus 1 = 0$,
$1 \oplus 0 = 1$,

Multiplication
In the polynomial representation, multiplication in Galois Field GF (2^8) (denoted by •) corresponds with the multiplication of polynomials modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. For the AES algorithm, this irreducible polynomial is given by the below equation.
$m(x) = x^8 + x^4 + x^3 + x + 1$

II. Encryption

A. Encryption Process

The Encryption process of Advanced Encryption Standard algorithm is presented below, in Figure 4.

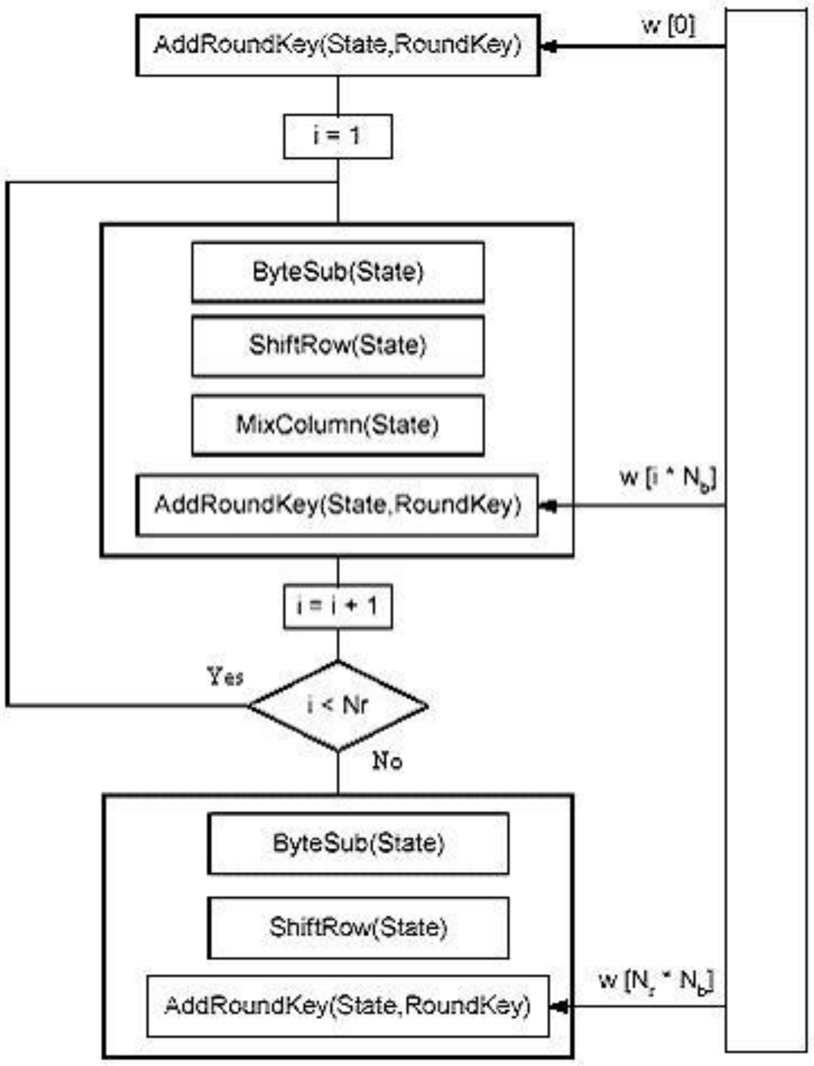


Figure 4

Figure 4.Encryption Process [6]

This block diagram is generic for AES specifications. It consists of a number of different transformations applied consecutively over the data block bits, in a fixed number of iterations, called rounds. The number of rounds depends on the length of the key used for the encryption process.[11]

SubBytes Transformation:

The SubBytes transformation is a non-linear byte substitution, operating on each of the state bytes independently. The SubBytes transformation is done using a once-precalculated substitution table called S-box. That S-box table contains 256 numbers (from 0 to 255) and their corresponding resulting values. This approach has the significant advantage of performing the S-box computation in a single clock cycle, thus reducing the latency and avoids complexity of hardware implementation.

ShiftRows Transformation:
In ShiftRows transformation, the rows of the state are cyclically left shifted over different offsets. Row 0 is not shifted; row 1 is shifted one byte to the left; row 2 is shifted two bytes to the left and row 3 is shifted three bytes to the left.[12]

MixColumns Transformation:
In MixColumns type of transformation, the columns of the state are considered as polynomials over GF (28) and multiplied by modulo x4 + 1 with a fixed polynomial c(x), given by: c(x)={03}x3 + {01}x2 + {01}x + {02}.

AddRoundKey Transformation:
In the AddRoundKey type of transformation, a Round Key is added to the State - resulted from the operation of the MixColumns transformation - by a simple bitwise XOR operation. The RoundKey of each round is derived from the main key using the KeyExpansion algorithm. The encryption/ decryption algorithm needs eleven 128-bit RoundKey, which are denoted RoundKey[0] RoundKey[10].

Key Schedule Generation
The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and Cipher Key lengths are not permitted by this standard. The Each round key is a 4-word (128-bit) array generated as a product of the previous round key, a constant that changes each round, and a series of S-Box lookups for each 32-bit word of the key. The first round key is the same as the original user input. Each byte (w_0 - w_3) of initial key is XOR'd with a constant that depends on the current round, and the result of the S-Box lookup for w_i, to form the next round key. The number of rounds required for three different key lengths is presented in Table 1.

AES	Key Lenght (NkWords)	Block Size (NbWords)	Number of Rounds (Nr)
AES 128	4	4	10
AES 192	6	4	12
AES 256	8	4	14

Table 1.Key-Block- Round Combinations [1]

The Key schedule Expansion generates a total of Nb(Nr + 1) words: the algorithm requires an initial set of Nb words, and each of the Nr rounds requires Nb words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted [wi], with i in the range $0 \leq i <$ Nb(Nr + 1).

III. Decryption

A. Decryption Process

The Decryption process of Advanced Encryption Standard algorithm is presented below, in figure 6.

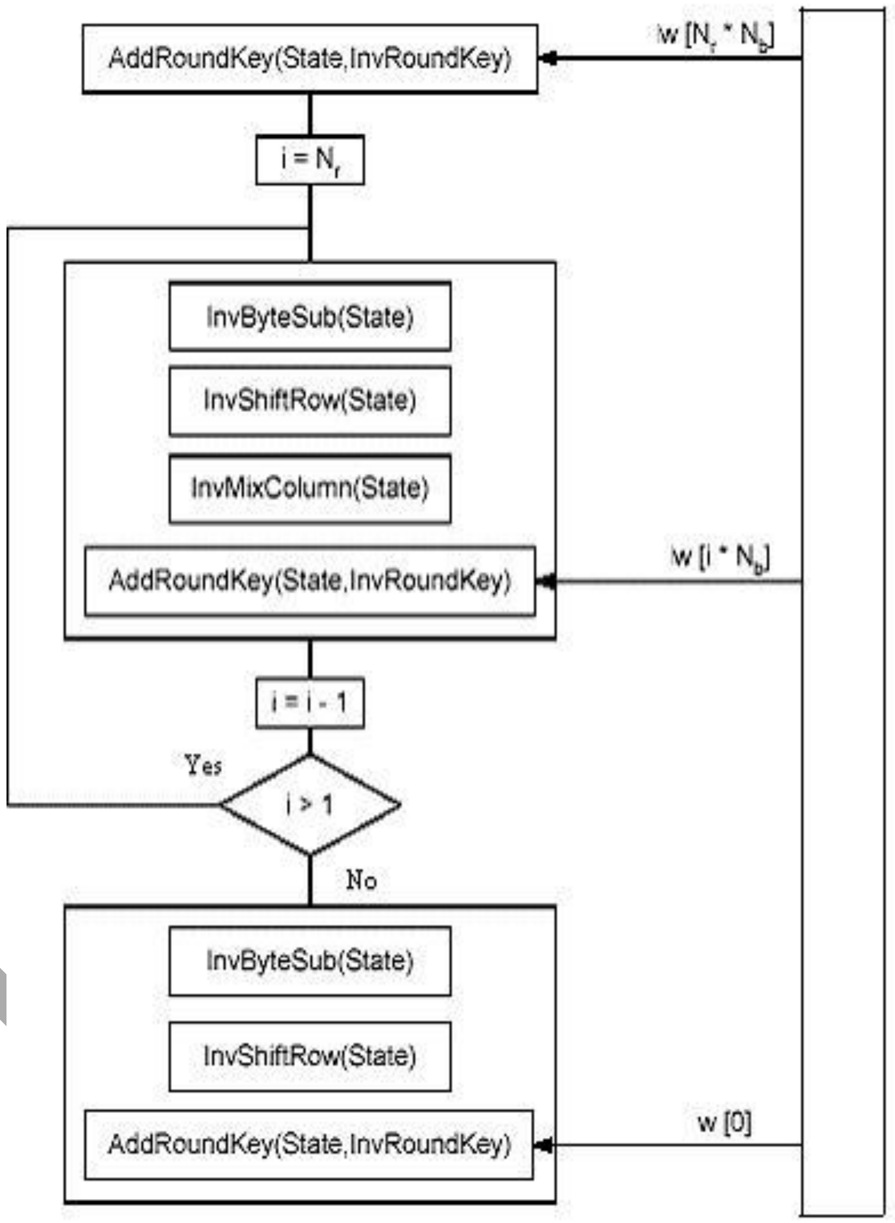


Figure 5 .Flow Chart

Figure 5.Decryption Process [6]

This process is direct inverse of the Encryption process. All the transformations applied in Encryption process are inversely applied to this process. Hence the last round values of both the data and key are first round inputs for the Decryption process and follows in decreasing order.

AddRoundKey:
AddRoundKey is its own inverse function because the XOR function is its own inverse. The round keys have to be selected in reverse order.

InvShiftRows Transformation:
InvShiftRows exactly functions the same as ShiftRows, only in the opposite direction. The first row is not shifted, while the second, third and fourth rows are shifted right by one, two and three bytes respectively.

InvSubBytes transformation:
The InvSubBytes transformation is done using a once recalculated substitution table called InvS-box. That InvS-box table contains 256 numbers (from 0 to 255) and their corresponding values.

InvMixColumns Transformation:
The InvMixColumns transformation is done using polynomials of degree less than 4 over GF(28), which coefficients are the elements in the columns of the state, are multiplied modulo (x4 + 1) by a fixed polynomial d(x) = {0B}x3 + {0D}x2 + {09}x + {0E}, where {0B}, {0D}; {09}, {0E} denote hexadecimal values.

Design Flow Chart
Figure 6. shows design of project flow. From design specification, design will be coded using Very High Speed Integrated Circuit Hardware Descriptive Language. Simulation and verification will be done on ModelSim software. Results are then synthesized on Xilinx ISE. Generated Bitstream file will need to program the FPGA.

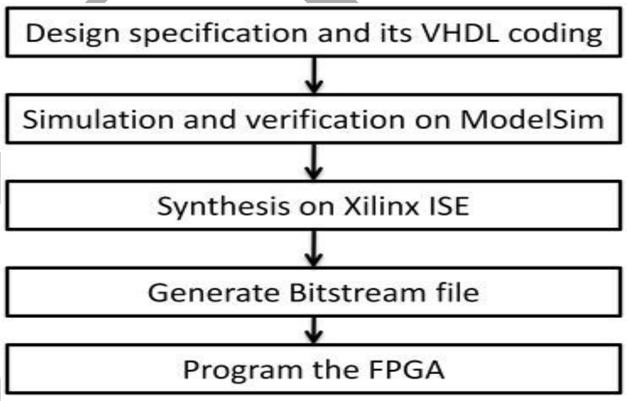

Design specification and its VHDL coding
Simulation and verification on ModelSim
Synthesis on Xilinx ISE
Generate Bitstream file
Program the FPGA

Figure 6.Flow Chart

IV. Conclusion

The AES algorithm can be efficiently implemented by software. Software implementations cost the smallest resources, but they offer a limited physical security and the slowest process. Besides, growing requirements for high speed, high volume secure communications combined with physical security, hardware implementation of cryptography takes place.

REFERENCES

1. FIPS 197, "Advanced Encryption Standard (AES)", November 26, 2001 http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
2. J. Daemen and V. Rijmen, "AES Proposal: Rijndael", AES Algorithm Submission, September 3, 1999 http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaeldocV2.zip
3. ALTERA. Max+plus II VHDL. San Jose. Altera, 1996
4. ALTERA "ACEX1K Embedded Programmable Logic Family Data Sheet", pdf files, http://www.altera.com/literature/ds/acex.pdf (May 2003)
5. ALTERA High-Speed Rijndael Encryption/Decryption Processors, http://www.altera.com/literature/wp/wp_hcores_rijnfast.pdf
6. Marcelo B. de Barcelos Design Case, "Optimized performance and area implementation of Advanced Encryption Standard in Altera Devices, by, http://www.inf.ufrgs.br/~panato/artigos/designcon02.pdf
7. "FPGA Simulations of Round 2 Advanced Encryption Standards" http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/presentations/elbirt.pdf.
8. http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm
9. Tilborg, Henk C. A. van. "Fundamentals of Cryptology: A Professional Reference and Interactive Tutorial", New York Kluwer Academic Publishers, 2002
10. Peter J. Ashenden, "The Designer's Guide to VHDL", 2nd Edition, San Francisco, CA, Morgan Kaufmann, 2002
11. Hoang Trang and Nguyen Van Loi HoChiMinh City, VietNam- "An efficient FPGA implementation of the Advanced Encryption Standard algorithm" (IEEE 2012)
12. 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"(IEEE 2010).
13. Adam J. Elbirt, W. Yip, B. Chetwynd, and C. Paar- "An FPGA Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists" (IEEE 2001).
14. WANG Wei, CHEN Jie & XU Fei, China-"An
15. Implementation of AES Algorithm Based on FPGA" (IEEE2012).
16. Nalini C, Nagaraj, Dr. Anandmohan P.V, &
17. Poornaiah D.V, V.D.kulkarni -"An FPGA Based Performance Analysis Pipelining and Unrolling of AES Algorithm" (IEEE2006).
18. Tessier. R. and Burleson W-"Reconfigurable computing for digital signal processing: a survey", J.VLSI Signal Process, 2001, 28.
19. Ahmad, N.; Hasan, R.; Jubadi, W.M; "Design of AES S-Box using combinational logic optimization", IEEE Symposium on Industrial Electronics & Applications (ISIEA), pp. 696-699, 2010.
20. Pravin B. Ghewari "Efficient Hardware Design and Implementation of AES Cryptosystem" International Journal of Engineering Science and Technology Vol. 2(3), 2010, 213-219.
21. Mg Suresh, Dr.Nataraj.K.R Asst Professor Rgit, Bangalore -"Area Optimized and Pipelined FPGA Implementation of AES Encryption and Decryption" International Journal Of Computational Engineering Research (ijceronline.com) Vol. 2 Issue. 7 (2012).