Implementation of the Fast Pipelined AES Algorithm on Xilinx FPGA

1GOWTHAMI REDDY K, 2SOWMYA G
1MTECH, Department of ECE, Bharat institute of technology and Sciences for women
2Assistant Professor, Department of ECE, Bharat institute of technology and Sciences for women
Gowthamireddy413@gmail.com, sowyagudepu@gmail.com

Abstract— Advanced Encryption Standard (AES), a Federal Information Processing Standard (FIPS), is an approved cryptographic algorithm that can be used to protect electronic data. The AES can be programmed in software or built with pure hardware. However Field Programmable Gate Arrays (FPGAs) offer a quicker and more customizable solution. This paper presents the AES algorithm with regard to FPGA and the Very High Speed Integrated Circuit Hardware Description language (VHDL). ModelSim SE PLUS 5.7g software is used for simulation and optimization of synthesizable VHDL code. Synthesizing and implementation (i.e. Translate, Map and Place and Route) of the code is carried out on Xilinx - Project Navigator, ISE 8.2i suite. All the transformations of both Encryption and Decryption are simulated using an iterative design approach in order to minimize the hardware consumptions. Xilinx XC3S400 device of Spartan Family is used for hardware evaluation. This paper proposes a method to integrate the AES encrypter and the AES decrypter. This method can make it a very low-complexity architecture, especially in saving the hardware resource in implementing the AES (Inv) Sub Bytes module and (Inv) Mix columns module etc. Most designed modules can be used for both AES encryption and decryption. Besides, the architecture can still deliver a high data rate in both encryption/decryption operations. The proposed architecture is suited for hardware-critical applications, such as smart card, PDA, and mobile phone, etc.

Index Terms—AES, FPGA, VHDL, cryptography, encryption, decryption.

1. Introduction

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data also called Rijndael. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data. Hardware-based cryptography is used for authentication of users and of software updates and installations. Software implementations can generally not be used for this, as the cryptographic keys are stored in the PC memory during execution, and are vulnerable to malicious codes. Hardware-based encryption products can also vary in the level of protection they provide against brute force rewind attacks, Offline parallel attacks, or other cryptanalysis attacks. An Efficient cryptography hardware implementation and its improvement using pipelines. The algorithm was implemented in FPGA due to its flexibility and reconfiguration capability. A reconfigurable device is very convenient for a cryptography algorithm since it allows cheap and quick alterations. The implementation of pipelined cryptography hardware was used to improve performance in order to achieve higher throughput and greater parallelism. The AES hardware was implemented in three modules contains of the encryption, the decryption and the key expansion module. The VHDL description implemented on both FPGAs is exactly the same, and no change was made in the VHDL description to fit any of the FPGAs. Modelsim Xilinx Edition will be used for functional simulation and verification of results. Xilinx ISE will be used for synthesis. The Xilinx’s chipscope tool will be used for verifying the results on Spartan 3E FPGA.

1.1 Motivation

Cryptography is the science of information and communication security. Cryptography is the science of secret codes, enabling the confidentiality of communication through an insecure channel. It protects against unauthorized parties by preventing unauthorized alteration of use. It uses an cryptographic system to transform a plaintext into a cipher text, using most of the time a key. There exists certain cipher that doesn’t need a key at all. An example is a simple Caesar-cipher that obscures text by replacing each letter with the letter thirteen places down in the alphabet. Since our alphabet has 26 characters, it is enough to encrypt the cipher text again to retrieve the original message.

1.2. Background of the Algorithm

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 uninterpretable 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. The Rijndael algorithm was developed by Joan Daemen of Proton.

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.
1.3. Notation and Conventions
1.3.1. 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 ≤ i < 128, 0 ≤ i < 192 or 0 ≤ i < 256 depending on the block length or key length specified.

2. AES ALGORITHM
2.1 Introduction to AES Algorithm

Serge Vaudenay, in his book "A classical introduction to cryptography", writes: Cryptography is the science of information and communication security. Cryptography is the science of secret codes, enabling the confidentiality of communication through an insecure channel. It protects against unauthorized parties by preventing unauthorized alteration of use. Generally speaking, it uses a cryptographic system to transform a plaintext into a ciphertext, using most of the time a key. One has to notice that there exist certain ciphers that don't need a key at all.

A famous example is ROT13 (abbreviation from Rotation 13), a simple Caesar-cipher that obscures text by replacing each letter with the letter thirteen places down in the alphabet. Since our alphabet has 26 characters, it is enough to encrypt the ciphertext again to retrieve the original message. Let me just mention briefly that there are secure public-key ciphers, like the famous and very secure Rivest-Shamir-Adleman (commonly called RSA) that uses a public key to Encrypt a message and a secret key to decrypt it.

Cryptography is very important in computer science with many applications. The most famous example of cryptography is certainly the Enigma machine, the legendary cipher machine used by the German Third Reich to encrypt their messages, whose security breach ultimately led to the defeat of their submarine force. Before continuing, please read carefully the legal issues involving cryptography in several countries even the domestic use of cryptography is prohibited: Cryptography has long been of interest to intelligence gathering agencies and law enforcement agencies. Because of its facilitation of privacy, and the diminution of privacy attendant on its prohibition, cryptography is also of considerable interest to civil rights supporters. Accordingly, there has been a history of controversial legal issues surrounding cryptography, especially since the advent of inexpensive computers has made possible widespread access to high quality cryptography. In some countries, even the domestic use of cryptography is, or has been, restricted. Until 1999, France significantly restricted the use of cryptography domestically. In China, a license is still required to use cryptography. Many countries have tight restrictions on the use of cryptography. Among the more restrictive are laws in Belarus, Kazakhstan, Mongolia, Pakistan, Russia, Singapore, Tunisia, Venezuela, and Vietnam. In the United States, cryptography is legal for domestic use, but there has been much conflict over legal issues related to cryptography. One particularly important issue has been the export of cryptography and cryptographic software and hardware. Because of the importance of cryptanalysis in World War II and an expectation that cryptography would continue to be important for national security, many western governments have, at some point, strictly regulated export of cryptography.

After World War II, it was illegal in the US to sell or distribute encryption technology overseas; in fact, encryption was classified as munitions, like tanks and nuclear weapons. Until the advent of the personal computer and The Internet, this was not especially problematic. Good cryptography is Indistinguishable from bad cryptography for nearly all users, and in any case most of the cryptographic techniques generally available were slow and error prone whether good or bad. However, as the Internet grew and computers became more widely available, high quality encryption techniques became well-known around the globe. As a result, export controls came to be seen to be an impediment to commerce and to research.

This standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths; however they are not adopted in this standard.

Throughout the remainder of this standard, the algorithm specified herein will be referred to as “the AES algorithm.” The algorithm may be used with the three different key lengths indicated above, and therefore these different “flavors” may be referred to as “AES-128”, “AES-192”, and “AES-256”.

The Advanced Encryption Standard, in the following referenced as AES, is the winner of the contest, held in 1997 by the US Government, after the Data Encryption Standard was found too weak because of its small key size and the technological advancements in processor power. Fifteen candidates were accepted in 1998 and based on public comments the pool was reduced to five finalists in 1999. In October 2000, one of these five algorithms was selected as the forthcoming standard: a slightly modified version of the Rijndael.

The Rijndael, whose name is based on the names of its two Belgian inventors, Joan Daemen and Vincent Rijmen, is a Block cipher, which means that it works on fixed-length group of bits, which are called blocks. It takes an input block of a certain size, usually 128, and produces a corresponding output block of the same size. The transformation requires a second input, which is the secret key. It is important to know that the secret key can be of any size (depending on the cipher used) and that AES uses three different key sizes: 128, 192 and 256 bits.

To encrypt messages longer than the block size, a mode of operation is chosen, which I will explain at the very end of this tutorial, after the implementation of AES. While AES supports only block sizes of 128 bits and key sizes of 128, 192 and 256 bits, the original Rijndael supports key and block sizes in any multiple of 32, with a minimum of 128 and a maximum of 256 bits. Unlike DES, which is based on a Feistel network, AES is a substitution-permutation network, which is a series of mathematical operations that use substitutions (also called S-Box) and permutations (P-Boxes).

2.2. AES Evaluation: Criteria for Initial Selection

- **Security**: randomness, soundness, results of cryptanalysis during evaluation
- **Cost**: royalty-free, computational efficiency, memory requirement
- **Flexibility**: Hardware & software suitability
3. AES ALGORITHM IMPLEMENTATION

For the AES algorithm, the length of the input block, the output block and the State is 128 bits. This is represented by Nb = 4, which reflects the number of 32-bit words (number of columns) in the State.

For the AES algorithm, the length of the Cipher Key, K, is 128, 192, or 256 bits. The key length is represented by Nk = 4, 6, or 8, which reflects the number of 32-bit words (number of columns) in the Cipher Key.

For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key size. The number of rounds is represented by Nr, where Nr = 10 when Nk = 4, Nr = 12 when Nk = 6, and Nr = 14 when Nk = 8.

The only Key-Block-Round combinations that conform to this standard are given in the table below.

<table>
<thead>
<tr>
<th>Key Length (Nk words)</th>
<th>Block Size (Nb words)</th>
<th>Number of Rounds (Nr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>AES-192</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>AES-256</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.1. Key-Block-Round Combinations

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte-oriented transformations:
1) Byte substitution using a substitution table (S-box)
2) Shifting rows of the State array by different offsets
3) Mixing the data within each column of the State array
4) Adding a Round Key to the State.

3.1 CIPHER

At the start of the Cipher, the input is copied to the State array. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first Nr -1 rounds. The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words. The individual transformations -SubBytes (), ShiftRows (), MixColumns (), and AddRoundKey () - process the State and are described in the following subsections.

3.1.1 SubBytes () Transformation

The SubBytes () transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box).

Each state byte is replaced by another in the S-box (replacement Box). The replacement follows a matrix, where the first hexadecimal value corresponds to the line positioning, and the second hexadecimal value corresponds to the column positioning. The inverse operation (decryption) is called InvSubBytes, and uses an inverse S-Box.

The SubBytes operation is a non-linear byte substitution, operating on each byte of the state independently. The substitution table (S-Box) is invertible and is constructed by the composition of two transformations:

1. Take the multiplicative inverse in Rijndael’s finite field.
2. Apply an affine transformation which is documented in the Rijndael documentation.

A graphical representation of this operation can be found below:

![Figure 3.1 subbyte substitution](image-url)

Since the S-Box is independent of any input, pre-calculated forms are used; if enough memory (256 bytes for one S-Box) is available. Each byte of the state is then substituted by the value in the S-Box whose index corresponds to the value in the state: $a(i,j) = Sbox[a(i,j)]$.

![Figure 3.2 Matrix form](image-url)
Figure 3.3 SubBytes () applies the S-box to each byte of the State.

The S-box used in the SubBytes () transformation is presented in hexadecimal form in Fig. 3.3

For example, if \( \text{in} = 1,1 \text{s} \{53\} \), then the substitution value would be determined by the intersection of the row with index ‘5’ and the column with index ‘3’ in Fig. 3.4. This would result as an “ed”.

Figure 3.4. S-box: substitution values for the byte xy (in hexadecimal format).

3.1.2 ShiftRows () Transformation

In the ShiftRows () transformation, the bytes in the last three rows of the state are shifted. It consists of a left shift on the state lines, replacing therefore their byte position. Line 0 suffers 0 shifting. Line 1 is shifted by one position and line 2 by two positions to the left. The 4th row is shifted 3 positions to the left.

This has the effect of moving bytes to “lower” positions in the row (i.e., lower values of \( C \) in a given row), while the “lowest” bytes wrap around into the “top” of the row (i.e., higher values of \( C \) in a given row).

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

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

A graphical representation of this operation can be found below:

Figure 3.5: shift operation

3.1.3 MixColumns () Transformation

The MixColumns () transformation operates on the state column-by-column, treating each column as a four-term polynomial:

\[
\begin{align*}
\text{MixColumns} & : \\
& = \begin{pmatrix}
2 & 3 & 1 & 1 \\
1 & 2 & 3 & 1 \\
1 & 1 & 2 & 3 \\
3 & 1 & 1 & 2 \\
\end{pmatrix}
\end{align*}
\]

A graphical representation of this operation can be found below:

Figure 3.8: mixColumn operation

And that the addition and multiplication operations are a little different from the normal ones.

3.1.4 AddRoundKey () Transformation

In the AddRoundKey () transformation, a Round Key is added to the state by a simple bitwise XOR operation. Each Round Key consists of \( \text{Nb} \) words from the key schedule.
3.2 KEY EXPANSION

The AES algorithm takes the Cipher Key, $K$, and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total of $N_f \times (N_r + 1)$ words: the algorithm requires an initial set of $N_b$ words, and each of the $N_r$ rounds requires $N_b$ words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted $[w_i]$, with $i$ in the range $0 \leq i < N_f \times (N_r + 1)$. The expansion of the input key into the key schedule proceeds according to the pseudo code.

SubWord ($w$) is a function that takes a four-byte input word and applies the S-box to each of the four bytes to produce an output word. The function RotWord ($a$) takes a word $[a_0, a_1, a_2, a_3]$ as input, performs a cyclic permutation, and returns the word $[a_1, a_2, a_3, a_0]$.

The round constant word array, $Rcon[i]$, contains the values given by $[x^i, [00],[00],[00]]$, with $x$ being a primitive element of the field GF ($2^8$). From Fig. 13, it can be seen that the first $N_k$ words of the expanded key are filled with the Cipher Key. Every following word, $w[i]$, is equal to the XOR of the previous word, $w[i-1]$, and the word $N_k$ positions earlier, $w[i-N_k]$. For words in positions that are a multiple of $N_k$, a transformation is applied to $w[i-1]$ prior to the XOR, followed by an XOR with a round constant, $Rcon[i]$. This transformation consists of a cyclic shift of the bytes in a word (RotWord ($a$)), followed by the application of a table lookup to all four bytes of the word (SubWord ($w$)).

It is important to note that the Key Expansion routine for 256-bit Cipher Keys ($N_k = 8$) is slightly different than for 128- and 192-bit Cipher Keys. If $N_k = 8$ and $i$ is a multiple of $N_k$, then SubWord ($w$) is applied to $w[i-1]$ prior to the XOR.

3.3 INVERSE CIPHER

The Cipher transformations can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher - InvShiftRows ($s$), InvSubBytes ($s$), InvMixColumns ($s$), and AddRoundKey ($s$) - process the State and are described in the following subsections.

3.3.1 InvShiftRows ($s$) Transformation

InvShiftRows ($s$) is the inverse of the ShiftRows ($s$) transformation. The bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, $r = 0$, is not shifted. The bottom three rows are cyclically shifted by $N_b - shift(r, N_b)$ bytes, where the shift value $shift(r, N_b)$ depends on the row number, and is given in below equation.

Specifically, the InvShiftRows ($s$) transformation proceeds as follows:

Below figure illustrates the InvShiftRows ($s$) transformation.

3.3.2 InvSubBytes ($s$) Transformation

InvSubBytes ($s$) is the inverse of the byte substitution transformation, in which the inverse Sbox is applied to each byte of the State. This is obtained by applying the inverse of the affine transformation followed by the multiplicative inverse in GF ($2^8$). The inverse S-box used in the InvSubBytes ($s$) transformation is presented in Fig. 12.

3.3.3 InvMixColumns ($s$) Transformation

InvMixColumns ($s$) is the inverse of the MixColumns ($s$) transformation. InvMixColumns ($s$) operates on the state column-by-column, treating each column as a four-term polynomial.

3.3.4 Inverse of the AddRoundKey ($s$) Transformation

AddRoundKey ($s$), which was described in Sec. 3.1.4, is its own inverse, since it only involves an application of the XOR operation.

3.3.5 Equivalent Inverse Cipher

In the straightforward Inverse Cipher, the sequence of the transformations differs from that of the Cipher, while the form of the key schedules for encryption and decryption remains the same. However, several properties of the AES algorithm allow for an Equivalent Inverse Cipher that has the same sequence of transformations as the Cipher (with the transformations replaced by their inverses). This is accomplished with a change in the key schedule.

The two properties that allow for this Equivalent Inverse Cipher are as follows:

1. The SubBytes ($s$) and ShiftRows ($s$) transformations commute; that is, a SubBytes ($s$) transformation immediately followed by a ShiftRows ($s$) transformation is equivalent to a ShiftRows ($s$) transformation immediately Followed by a SubBytes ($s$) transformation.

Figure 3.9 Add round key operation

Figure 3.10 InvShiftRows ($s$) cyclically shifts the last three rows in the state.

Figure 3.11. Inverse S-box: substitution values for the byte xy

Figure 3.12.
SubBytes () transformation. The same is true for their inverses, InvSubBytes () and InvShiftRows.
2. The column mixing operations – MixColumns () and InvMixColumns () – are linear with respect to the column input, which means
InvMixColumns (state XOR Round Key) = InvMixColumns (state) XOR InvMixColumns (Round Key).

These properties allow the order of InvSubBytes () and InvShiftRows () Transformations to be reversed. The order of the AddRoundKey () and InvMixColumns () transformations can also be reversed, provided that the columns (words) of the decryption key schedule are modified using the InvMixColumns () transformation.

The equivalent inverse cipher is defined by reversing the order of the InvSubBytes () and InvShiftRows () transformations and by reversing the order of the AddRoundKey () and InvMixColumns () transformations used in the “round loop” after first modifying the decryption key schedule for round = 1 to Nr=1 using the InvMixColumns () transformation. The first and last Nb words of the decryption key schedule shall not be modified in this manner.

Given these changes, the resulting Equivalent Inverse Cipher offers a more efficient structure than the Inverse Cipher . Pseudo code for the Equivalent Inverse Cipher (The word array dw [ ] contains the modified decryption key schedule.

Note that, since InvMixColumns operates on a two-dimensional array of bytes while the Round Keys are held in an array of words, the call to InvMixColumns in this code sequence involves a change of type (i.e., the input to InvMixColumns () is normally the State array, which is considered to be a two-dimensional array of bytes, whereas the input here is a Round Key computed as a one-dimensional array of words).

The following chapter consists of all the software and hardware results observed in the project. The results include snapshots of each and every module individually with all the inputs, outputs and intermediate waveforms.

4. EXPERIMENTAL RESULTS

4.1 SIMULATION RESULTS

DESCRIPTION:
The design has been coded by VHDL. Modelsim Xilinx Editionis used for functional simulation and verification of results. The results of simulating the encryption/decryption algorithm from the ModelSim simulator . The plaintext_encryption, enter_key are the 2 inputs of each 128-bit. The encryption_out will be given to decryption as an input, the plaintext_decryption.

4.2 CHIPSOCPE RESULTS

Figure 3.12 Block diagram

Figure 4.1(a) Simulation Result

Figure 4.1(b) Simulation Result

Figure 5.1(b) Simulation Result

Figure 5.2 AES Simulation

Figure 5.2 AES Simulation
Figure 5.3 verification of basic AES (Encryption)

shows the verification of basic design. The data input to the encryption module is (0000_0000_0000_0000_0000_0000_0000_0000) h and the key is also (0000_0000_0000_0000_0000_0000_0000_0000)h. The above data is encrypted to (66E9_4BD4_EF8A_2C3B_884C_FA59_CA34_2B2B) h. The encrypted data is also verified by the AES algorithm implemented.

Figure 5.4 verification of basic AES (Decryption)

The Xilinx’s chipscope tool will be used for verifying the results on Spartan 3E FPGA (shown in Fig.10). Chipscope is an embedded, software-based logic analyzer. By inserting an “Integrated Controller Core” (ICON) and an “Integrated Logic Analyzer” (ILA) into the design and connecting them properly, the signals in the design can be monitored. ChipScope provides with a convenient software-based interface for controlling the “integrated logic analyzer,” including setting the triggering options and viewing the waveforms.

5. CONCLUSION

Optimized and Synthesizable VHDL code is developed for the implementation of both encryption and decryption process. Each program is tested with some of the sample vectors provided by NIST and output results are perfect with minimal delay. Therefore, AES can indeed be implemented with reasonable efficiency on an FPGA, with the encryption and decryption taking an average of 320 and 340 ns respectively (for every 128 bits). The time varies from chip to chip and the calculated delay time can only be regarded as approximate. Adding data pipelines and some parallel combinational logic in the key scheduler and round calculator can further optimize this design.

- This AES algorithm can be parameterized by selection of cipher key bits (128, 192 or 256).
- For higher throughput, 16 S-Box can be used completing whole processes around 44-50 cycles (at the same time, compromising the silicon area).
- Graphical use Interface(GUI) can also be made which may be interactive with the user.

6. REFERENCES