Journal

From the SelectedWorks of Kirat Pal Singh

Winter February 27, 2016

An Efficient Hardware design and Implementation of Advanced Encryption Standard (AES) Algorithm

Kirat Pal Singh
Shiwani Dod

Available at: https://works.bepress.com/kiratpalsingh/2/
An Efficient Hardware design and Implementation of Advanced Encryption Standard (AES) Algorithm

Kirat Pal Singh, Shiwani Dod

Abstract - We propose an efficient hardware architecture design & implementation of Advanced Encryption Standard (AES). The AES algorithm defined by the National Institute of Standard and Technology (NIST) of United States has been widely accepted. The cryptographic algorithms can be implemented with software or built with pure hardware. However Field Programmable Gate Arrays (FPGA) implementation offers quicker solution and can be easily upgraded to incorporate any protocol changes. This contribution investigates the AES encryption cryptosystem with regard to FPGA and Very High Speed Integrated Circuit Hardware Description language (VHDL). Optimized and Synthesizable VHDL code is developed for the implementation of 128- bit data encryption process. AES encryption is designed and implemented in FPGA, which is shown to be more efficient than published approaches. Xilinx ISE 12.3i software is used for simulation. Each program is tested with some of the sample vectors provided by NIST and output results are perfect with minimal delay. The throughput reaches the value of 1609Mbit/sec for encryption process with Device XC6vlx240t of Xilinx Virtex Family.

Keywords- Advanced Encryption Standard (AES), Rijndael, Cryptography, FPGA, Throughput.

I. INTRODUCTION

To protect the data transmission over insecure channels two types of cryptographic systems are used: Symmetric and Asymmetric cryptosystems. Symmetric cryptosystems such as Data Encryption Standard (DES) [1], 3 DES, and Advanced Encryption Standard (AES) [4], uses an identical key for the sender and receiver; both to encrypt the message text and decrypt the cipher text. Asymmetric cryptosystems such as Rivest-Shamir-Adleman (RSA) & Elliptic Curve Cryptosystem (ECC) uses different keys for encryption. Symmetric cryptosystem is more suitable to encrypt large amount of data with high speed. To replace the old Data Encryption Standard, in Sept 12 of 19997, the National Institute of Standard and Technology (NIST) required proposals to what was called Advanced Encryption Standard (AES). Many algorithms were presented originally with researches from 12 different nations. Fifteen algorithms were selected to the Round one. Next five were chosen to the Round two. Five algorithms finalized by NIST are MARS, RC6, Rijndael [2], SERPENT and Twofish [3]. On October 2nd 2000, NIST [4] has announced the Rijndael algorithm is the best in security, performance, efficiency, implement ability, & flexibility. The Rijndael algorithm was developed by Joan Daemen of Proton World International and Vincent Rijmen of Katholieke University at Leuven. AES encryption is an efficient scheme for both hardware and software implementation. As compare to software implementation, hardware implementation provides greater physical security and higher speed. Hardware implementation is useful in wireless security like military communication and mobile telephony where there is a greater emphasis on the speed of communication. Most of the work has been presented on hardware implementation of AES using FPGA [5-9]. This paper presents efficient hardware architecture design & implementation of AES using FPGA and describes performance testing of Rijndael algorithm.

II. PREVIOUS DESIGN

An encryption algorithm converts a plain text message into cipher text message which can be recovered only by authorized receiver using a decryption technique. The AES-Rijndael algorithm [4] is an iterative private key symmetric block cipher. The input and output for the AES algorithm each consist of sequences of 128 bits (block length). Hence Nb = Block length/32 = 4. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits (Key length). In this implementation we set the key length to 128. Hence Nk = Key length/32 = 4.

Fig. 1 AES Encryption process
A. Encryption Process:

The Encryption process 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. For key length of 128 bits, the number of iteration required are 10. \((N_r = 10)\). As shown in Fig. 1, each of the first \(N_r-1\) rounds consists of 4 transformations: SubBytes(), ShiftRows(), MixColumns() & AddRoundKey(). The four different transformations are described in detail below.

Sub Bytes Transformation: It is a non-linear substitution of bytes that operates independently on each byte of the State using a substitution table (S box). This S-box which is invertible is constructed by first taking the multiplicative inverse in the finite field \(GF(2^8)\) with irreducible polynomial \(m(x) = x^8 + x^4 + x^3 + x + 1\). The element \{00\} is mapped to itself. Then affine transformation is applied (over \(GF(2)\)).

Shift Rows Transformation: Cyclically shifts the rows of the State over different offsets. The operation is almost the same in the decryption process except for the fact that the shifting offsets have different values.

Mix Columns Transformation: This transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over \(GF(2^8)\) and multiplied by modulo \(x^4 + 1\) with a fixed polynomial \(a(x) = \{03\} x^3 + \{01\} x^2 + \{02\} x\). The function \(\text{xtime}\) is used to represent the multiplication with \(‘02’\), modulo the irreducible polynomial \(m(x) = x^8 + x^4 + x^3 + x + 1\). Fig. 3 illustrates the implementation of function \(B = \text{xtime}(A)\), in which output bits 0, 2, 5, 6, 7 just correspond to input bits shifted and only 3 bits are modified by the XOR operation. Applying this concept, we can easily realize the 4-byte output of MixColumn as shown in Fig. 4. This direct implementation takes 2 \(\text{xtime}\) and 4 additions in calculating each byte output using the module Byte_MixC whose operation is \(2a \oplus b \oplus c \oplus d\). In our design, we express the operation of Byte_MixC as \(2(a \oplus b) \oplus b \oplus (c \oplus d)\). So, an efficient design of MixColumn transformation is shown in Fig. 5. The new architecture needs only 1 \(\text{xtime}\) and 4 additions operations for each Byte_MixC module.

AddRoundKey transformation: This transformation is simply performed by XOR the state with the round key. In [2], the Key Expansion (KE) is realized as Fig. 6, where \(K_{i+1,2}\) must wait until the result of \(K_{i+1,3}\) is calculated and it needs 4 calculated steps to obtain the complete outputs of KE. Such design directly follows from:

\[
\begin{align*}
K_{i+1,0} &= K_{i,0} \oplus F(K_i,3) \\
K_{i+1,1} &= K_{i+1,1} \oplus K_{i,1} \\
K_{i+1,2} &= K_{i+1,2} \oplus K_{i,2} \\
K_{i+1,3} &= K_{i+1,3} \oplus K_{i,3}
\end{align*}
\]

and we can reexpress above equation as:

\[
\begin{align*}
K_{i+1,1} &= (K_{i,0} \oplus K_{i,1}) \oplus F(K_i,3) \\
K_{i+1,2} &= (K_{i,0} \oplus K_{i,1} \oplus K_{i,2}) \oplus F(K_i,3) \\
K_{i+1,3} &= (K_{i,0} \oplus K_{i,1} \oplus (K_{i,2} \oplus K_{i,3})) \oplus F(K_i,3)
\end{align*}
\]

Based on this expression, we can develop a new key expansion module in which the output byte don’t need to
wait the other output bytes. The proposed key expansion module is constructed and shown in Fig. 7.

Fig. 6 KE module in [2]

Fig. 7 Proposed KE module

It is known that the normal round includes the operations of SubByte, ShiftRow, Mixcolumn and AddRoundKey, and the final round is equal to the normal round without the MixColumn. Our architecture of AES encryptor is shown in Fig. 8. The final round in our design is just a operation of XOR with round key because SubByte and ShiftRow are the same as the normal round. Fig 9 shows the detailed design of AES encryptor where the control signals are described in Table 1 and Fig 10 depicts its entity diagram. It needs 10 cycles to finish AES encryption. Based on the same methodology of AES encryptor, an AES decryptor is also designed and integrated with the AES encryptor to yield a full functional AES en/decryptor.

Fig. 8 Architecture of AES Encryptor

Fig. 9 Proposed AES Encryptor design

1. EXPERIMENTAL RESULTS:

All the results are based on simulations from the Xilinx ISE tools, using Test Bench Waveform Generator. All the individual transformation of encryption are simulated and synthesized using FPGA Vertex family and XC6vx1x240t device. Pin configurations of AES Entity are shown in Table 2. Each program is tested with some of the sample vectors provided by NIST [4].

Fig. 10 Entity diagram of AES

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O Port</th>
<th>Pin Number (bit)</th>
<th>Pin Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>I</td>
<td>1</td>
<td>Chip clock</td>
</tr>
<tr>
<td>Reset</td>
<td>I</td>
<td>1</td>
<td>Clear all signal and data</td>
</tr>
<tr>
<td>Load</td>
<td>I</td>
<td>1</td>
<td>Load key and plaintext</td>
</tr>
<tr>
<td>Start</td>
<td>I</td>
<td>1</td>
<td>Start encryption process</td>
</tr>
<tr>
<td>Key_in</td>
<td>I</td>
<td>128</td>
<td>Key data bus</td>
</tr>
<tr>
<td>Data_in</td>
<td>I</td>
<td>128</td>
<td>Plaintext data</td>
</tr>
<tr>
<td>Kstat</td>
<td>O</td>
<td>1</td>
<td>Kstat becomes high before output comes encryption is completed</td>
</tr>
<tr>
<td>Qstrb</td>
<td>O</td>
<td>1</td>
<td>Dstat becomes high when plaintext is loaded, and low when Qstrb is high</td>
</tr>
<tr>
<td>Data_out</td>
<td>O</td>
<td>128</td>
<td>Cipher data bus</td>
</tr>
</tbody>
</table>
An Efficient Hardware design and Implementation of Advanced Encryption Standard (AES) Algorithm

AES block length/Plane Text = 128 bits (Nb=4)

Key length = 128 bits (Nk=4)

No. of Rounds = 10 (Nr=10)

Input/plain text – 0x0000000000000000000000038

Key – 0x00000000000000004b49524154504114c

Output/Cipher – ef3577ad6e455fcdad863df695572ed0

Fig. 11 represents the waveforms generated by the 128-bit complete encryption Process. The inputs are clock1 & clock2, Active High reset, 4-bit round, and 128-bit state & key as a standard logic vectors, whose output is the 128-bit cipher (encrypted) data.

<table>
<thead>
<tr>
<th>Device Parameter</th>
<th>Synthesis Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target FPGA Device</td>
<td>Virtex6-XXV6t1x240tff1156-3</td>
</tr>
<tr>
<td>Optimization Goal</td>
<td>Speed</td>
</tr>
<tr>
<td>Maximum Operating Frequency</td>
<td>515.38MHz</td>
</tr>
<tr>
<td>Number of Slices Registers</td>
<td>954 out of 301440(0%)</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>632 out of 150720(0%)</td>
</tr>
<tr>
<td>Number of fully used LUT-FF pairs</td>
<td>447 out of 1139(39%)</td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>391 out of 600(65%)</td>
</tr>
<tr>
<td>Number of bonded BUFG/BUFGCTRLs</td>
<td>1 out of 32(3%)</td>
</tr>
<tr>
<td>Number of BRAM/FIFO</td>
<td>5 out of 416(1%)</td>
</tr>
<tr>
<td>Throughput</td>
<td>1609MHz</td>
</tr>
</tbody>
</table>

The parameter that compares AES candidates from the point of view of their hardware efficiency is Throughput [12].

Encryption Throughput = block size frequency/total clock cycles. Thus, Throughput = 128 x 515.38MHz/41 = 1609Mbits/sec.

CONCLUSION

The Advanced Encryption Standard-Rijndael algorithm is an iterative private key 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. An efficient FPGA implementation of 128 bit block and 128 bit key AES-Rijndael cryptosystem has been presented in this paper. Optimized and Synthesizable VHDL code is developed for the implementation of 128 bit data encryption process & description is verified using ISE 12.3i functional simulator from Xilinx. All the transformations of algorithm are simulated using an iterative design approach in order to minimize the hardware consumption. Each program is tested with some of the sample vectors provided by NIST. The proposed implementation is efficient and suitable for hardware-critical applications.

REFERENCES


[9] Piotr Mroczkowski, Military University of Technology, Poland, “Implementation of the block cipher Rijndael using Altera FPGA.”

