HIGH THROUGHPUT-LESS AREA EFFICIENT FPGA IMPLEMENTATION OF BLOCK CIPHER AES ALGORITHM

M.SIRIN KUMARI
(M.TECH) IIYEAR
JITS-KNR
sirin.kumari@gmail.com

D.MAHESH KUMAR
ASSOC.PROF
JITS-KNR

Y.RAMA DEVI
M.TECH(DC).
JBIT,HYD
rama_404@rediffmail.com

Abstract

This paper addresses design, hardware implementation and performance testing of AES algorithm. An optimized code for the Rijndael algorithm with 128-bit keys has been developed. The area and throughput are carefully trading off to make it suitable for wireless military communication and mobile telephony where emphasis is on the speed as well as on area of implementation.

Keywords: Cryptography, Rijndael, Encryption,Advanced Encryption Standard (AES), pipelining,security, very-large-scale integration (VLSI), VHDL.

1. Introduction

Several techniques, such as cryptography, steganography, watermarking, and scrambling, have been developed to keep data secure, private, and copyright protected. However, the need for secure transactions in ecommerce, private networks, and secure messaging has moved encryption into the commercial realm. Advanced encryption standard (AES) was issued as Federal Information Processing Standards (FIPS) by National Institute of Standards and Technology (NIST) as a successor to data encryption standard (DES) algorithms. The hardwarebased implementation of AES Rijndael Algorithm [1] is required because it is secure and consumes much less power than a software implementation. In recent literature, a number of architectures for the VLSI implementation of AES Rijndael algorithm are reported [2] [3] [4] [5] [6]. Some of these are of low performance and low throughput architectures. Further, many of the architectures are not area efficient and can result in higher cost when implemented in silicon. Su et al. [15] presented an implementation with reduced hardware overhead. Satoh et. al. [6] presented a low performance implementation with less hardware resources. Verbauwhede et al. [2] presented an ASIC implementation having a throughput of 2.29 Gb/s. In 2001, Elbirt et al. [10] compared five algorithms for AES block cipher FPGA implementations and the throughputs of Rijndael algorithm were found in the range of 188 Mb/s to 1.94Gb/s. McLoone and McCanny [4] utilized look-up tables to implement the entire Rijndael round function in FPGAs and demonstrated a throughput of 12 Gb/s.A high throughput implementation is required to support security for current and future high bandwidth applications. A low silicon area implementation is also, desirable to make it embeddable not only in high-end servers but also in low-end consumer products such as mobile terminals. This paper addresses design,hardware implementation and performance testing of AES algorithm. An optimized code for the Rijndael algorithm with 128-bit keys has been developed. The area and throughput are carefully trading off to make it suitable for wireless military communication and mobile telephony where emphasis is on the speed as well as on area of implementation. The proposed architecture is optimized for high throughput in terms of the encryption and decryption data rates by keeping the combinational paths balanced so that every clock cycle is fully utilized. The paper is organized as follows. In section 2, a brief overview of the AES Algorithm is provided. Section 3, focuses hardware

architecture of the proposed design. Performance analysis and measurement results are reported in section 4. Finally, conclusions are made in section 5.

2. RIJNDAEL Algorithm: A Brief Overview

The AES is a round based symmetric block cipher. It takes a 128 bit data block as input and performs several transformations on this block [4][6][12]. The AES-Rijndael algorithms operations are performed in the *state.*

The State is a two-dimensional array of bytes, consisting of four rows and Nb columns, where Nb is the block length divided by 32. In AES Algorithm, all bytes are represented as elements of the finite field GF(28) Using the polynomial representation, the byte{01100011} is represented as x6+x5+x+1 or {63} in hexadecimal notation. Mathematical operations with finite field elements are different from those used for numbers. The addition is achieved by adding the corresponding powers of the two polynomials. This operation is a modulo 2, i.e. it is an XOR operation. Consequently, subtraction of finite field elements isidentical to addition. ultiplication in GF is a polynomial multiplication of degree 8. AES algorithm specifies an irreducible polynomial as: m(x) = x8+x4+x3+x+1 or {01}{0b}. There is no simple operation available at byte level that mplements this multiplication. The number of rounds depends on the key size and blocks size and is summarized in Table 1. The number of rounds (Nr) is generated by the formula: Nr = (Key Length (in bits))/32 + 6. In the proposed design, we have used 128-bit key and cipher size and therefore 10 rounds are needed.

3. Design and Implementation

The Encryption and Decryption flow of the AES algorithm implemented is represented in Fig. 1 The decryption structure has exactly the same sequence of transformation as that in the encryption structure. This feature enables more efficient implementation of joint Encryptor/ Decryptor. In a standard AES algorithm, there are four steps i.e. SubByte, ShiftRows, MixColumns and AddRoundKey in normal rounds for both the Cipher and its Inverse (a) SubBytes - The bytes substitution transformation is a non-linear substitution of bytes that operates independently on each byte of the State using a substitution table (Sbox). This S-box is also invertible. (b) ShiftRows – In the Shift Rows transformation ShiftRows, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row is not shifted. (c) MixColumns - a mixing operation which operates on the columns of the state, combining the four bytes in each column using a linear transformation. (d) AddRoundKey - each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule.The largest component is Key Schedule. A wide variety of architectures could be used to implement a given algorithm [2,3]. In the proposed design, an iterative architecture is chosen to suit entire deign in the available single chip FPGA.

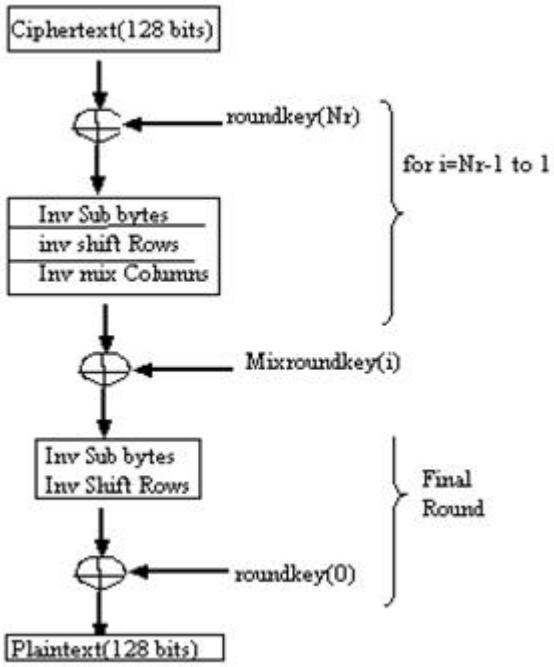

Fig. 1 (a) AES algorithm for Encryption

There are several variations on these approaches, including multiple copies of an iterative implementation for parallel processing, a partially pipelined implementation, or a combination of these hybrids (multiple copies of a partially pipelined implementation). We have followed an iterative approach followed by efficient utilization of a number of operations in each clock cycles. Despite the large amount of symmetry in encryption and decryption, care has been taken to eliminate symmetry in the behavior of the cipher. This is achieved by the round constants that are different for each round. In order to protect against chosen plaintext and chosen cipher text attacks, before the first round itself, a key addition layer is applied. The motivation for this initial key addition is the following. Any layer after the last key addition in the cipher (or before the first in the context of known-plaintext attacks) can be simply peeled off without knowledge of the key and therefore does not contribute to the security of the cipher.

The Linear mixing of layer by shifting rows and mixing columns guarantee high diffusion as the transformations take place over multiple rounds. The non-linear S boxes help to protect against linear and differential cryptanalysis The Rijndael block executes either encrypt or decrypt algorithm, according to the case. As seen in Fig. 2, the other processes only provide support to read and write bus operation and to round keys generation. The Data In process gives support to Rijndael. It is used to take the data from the bus. It is controlled by the WrD and Clk signals. When the bus puts a data to be read, this signal is selected and the data is taken. Mainly three modifications are proposed in our design keeping in mind the tradeoff between silicon area and throughput. They are, (i) Merger of Sub Bytes and Shift Rows (ii) Generating the Sub Bytes for Encryption and using Look up table for Decryption (iii) Each clock cycle is efficiently assigned to complete a set of operations.

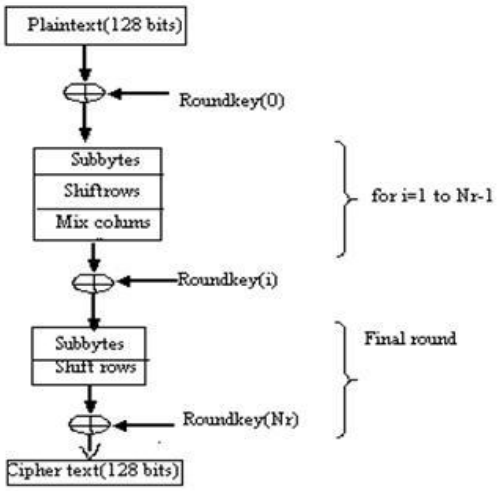

Fig.1(b) AES algorithm for Decryption Flow

3.1 Merger of Sub Bytes and Shift Rows

This merging is performed by calling required shifted element from the data matrix, instead of calling element one by one sequentially from the data matrix. Thereby SUB-BYTE and SHIFT ROW operations are carried out in one-step instead of two. Fig. 3 shows how the merger is performed. The 16 elements are stored sequentially after each round in a register file. Using Mux selection, required

shifted data elements are called (instead of calling sequentially) from the register file and put into the State. This merging process would increase throughput since elements are not called sequentially and a balance between throughput and area is maintained.

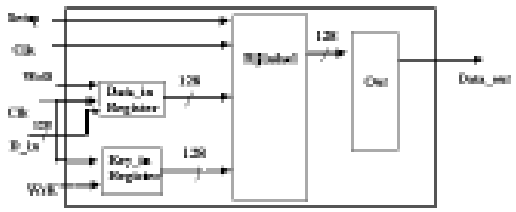

Fig. 2. Encrypt/decrypt architecture

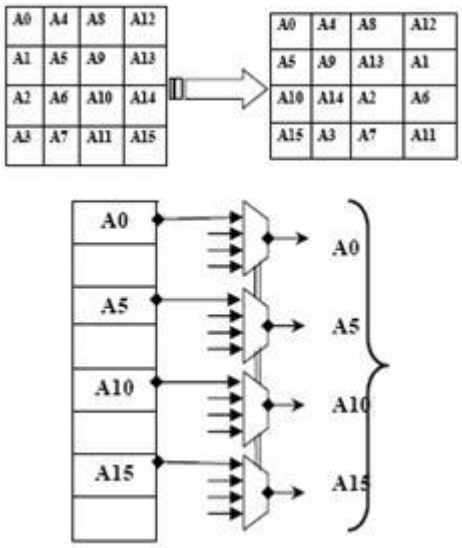

Fig. 3. Merging of Sub Bytes and Shift Rows

3.2 Generating the Sub Key for Encryption and using Look up table for Decryption

The SubBytes(S-box) transformation, which consists of a multiplicative inversion over GF (28) and an affine

transform is the most critical part in the AES Algorithm, as far as computational complexity is concerned. The S-box operation is required both for encryption and for key expansion. The S-box dominates the hardware complexity of the AES circuit. Conventionally the coefficients of the S-Box and inverse S-box are stored in LUT's, or a hard-wired multiplicative inverter over GF (28) can be used together with the affine transformation. We propose a multiplicative inverter together with the affine transformation be applied to the encryption unit. The LUT's be used for storing the coefficients of the inverse S-Box for the Decryption Unit is shown in Fig.4. (a) (b) Most approaches use a ROM/RAM-based lookup table (LUT) to implement the most critical transformation step in the AES algorithm, the *SubBytes* transformation as described in Fig. 2(a). This approach is cost effective for SRAM-based FPGAs, The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

integrated circuit (ASIC) implementation. Results from several other projects indicates that the implementation of an arithmetic circuit in a composite field to compute the multiplicative inverse and affine transformation of the S-Box provides an excellent trade-off between silicon area and performance. The composite field implementation was first recommended by the inventor of Rijndael [9]. Other implementations based on this idea can be found in [5-8]. However, no approach has been made so far on using both the LUT-based approach as well the composite field implementation. Since the Encryption Unit and the Key Scheduling Component require the S-box, we propose that the S-box be generated using composite field implementation in this case and that LUT's be used for storing the coefficients of the inverse S-box for the Decryption Unit. This would ensure high throughput as well as less area resources.The Key Scheduling Component performs Round Key generation. This round key could be generated dynamically each round based on the previous rounds key or they can be generated at the start and stored in the RAM. The former case however, would offer much less buffering.

3.3 Optimization of each clock cycle to incorporate maximum number of operations

Various types of hardware architectures for AES Algorithm are possible. As mentioned earlier, architecture that offers the best tradeoff between data throughput and silicon area is targeted in this work.Each clock cycle is efficiently assigned to complete a set of operations and the entire Encryption/decryption round can be completed with less clock cycles.

4. Performance Analysis

The proposed architecture was implemented using VHDL and implemented on Xilinx's Web pack version 8.2i. The implementations were simulated for the correct encryption and decryption operation using the test vectors provided by the AES submission package [4][11]. The VHDL code of the design is synthesized, placed and routed using target device of Xilinx (Virtex2Pro). The architecture was simulated for verification of the correct functionality. It is important to determine the amount of data that the channel can use with cryptography. The throughput is calculated with the following formula: Throughput =block size * frequency / total clock cycles. The Post Route Simulation analysis is performed. The input and the cipher key are given to the device and the output is displayed at the 27th clock cycle (after 26 cycles),there by achieving a sufficient throughput rate with a significant compromise on area. The standard used is 128-bit key and 128-bit data block size, therefore, the Control Component allows for only 10 rounds of AES encryption. The number of cycles per encryption block is 13. The maximum clock frequency is 142 MHz. The encryption and decryption throughput is given by 1.4Gbps. The parameters used to evaluate the quality of device are summarized in Table II. Two designs are compared. The original design based on simple iterative architecture cost large amount of area with a low throughput. The proposed design with the three modifications discussed in section 3 resulted in a much compact implementation with high throughput. Besides, it gave optimum usage of silicon area as only 50% of the resources are used (XC2VP30ff896). In addition, the speed is also improved four times.A comparison of various AES Designs is given in Table III. Only [14] has used computational arithmetic (multiplication and affine transform) over GF (28) to generate the S-box required. All the other architectures in Table III have made use of a ROM/RAM based Look up Table (LUT) to implement this transformation. Jarvinen et al. and [4] have used much lesser slices than our work but they have compensated
it by utilizing more BRAM's. Most designs have followed a pipeline architecture that has led to their increased throughput values. Recently, Hodjat and Ingrid [16]'s FPGA implementation showed a high throughput of 21.54 Gb/s using a fully pipelined approach with inner-round pipelining and outer-round pipelining. In our approach, we have worked with an iterative structure and with fully pipelined and resource-sharing methods; we hope to achieve better throughputs in future. Though proposed design does not offer a very high throughput and throughput/slice value due to lack of pipelining, it offers a significant decrease in the number of slices required. The motivation behind this work was to achieve a tradeoff between throughput with area and results are satisfactory.

5. Conclusions

The objective of this paper was to present the hardware implementation of Advanced Encryption Standard (AES) algorithm. The importance of the Advanced Encryption Standard and the significance of area-throughput balanced implementations of the Rijindael have examined. We have worked with an iterative structure and modifications such as merging of Subbytes and ShiftRows, Look Up tables for decryption, and optimization of each clock cycle to incorporate maximum number of operations etc. have been successfully implemented. The encryption and decryption process of Rijndael algorithm was captured in VHDL language

and corresponding FPGA implementation resulted in reduced number of slices and achieved a data throughput of 1.4 Gbit/sec. The combination of security, and high-speed implementation and marginal silicon area makes it a very good choice for wireless systems.

References

1. W. Diffle and H.Hellman, “Privacy and authentication: An Introduction to cryptography”, Proceedings of
IEEE, pp 397-427, vol 67, 1979.
2. H. Kuo and I. Verbauwhede, “Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the
AES Rijindael Algorithm”, 3rd international workshop cryptographic Hardware and embedded systems
(CHES 2001), LNCS2162,Paris, pp 51-64, May 2001.
3. I. Verbauwhede and H. Kuo, “Design and performance testing of a 2.29Gbits/sec Rijindael algorithm”,
IEEE Journal of solid state circuits, vol38, No.3, pp 569-572, March 2003.
4. M. McLoone and J. McCanny, “High Performance Single-Chip FPGA Rijndael Algorithm
Implementations,” Proceedings Cryptographic Hardware and Embedded Systems Workshop,CHES, Paris,
May 2001.
5. S. Mangard, M. Aigner, and S. Dominikus, “A Highly Regular and Scalable AES Hardware Architecture,”
IEEETrans. Comp.,vol. 52, no. 4, pp. 483–91, Apr.2003
6. A. Satoh *et al.*, “A Compact Rijndael HARDWARE Architecture with S-Box Optimization,”
ASIACRYPT 2001 LNCS, vol. 2248, pp. 239–54,
7. J. Wolkerstorfer, E. Oswald, and M. Lamberger, “An ASIC Implementation of the AES SBoxes,” CT-RSA
2002, vol. 2271 of LNCS, Springer-Verlag, pp. 67–78, 2002