Hardware-Software Co-Design of AES on FPGA

Saambhavi Baskaran
Indian Institute of Technology Hyderabad
Medak, Hyderabad
ee10m09@iith.ac.in

Pachamuthu Rajalakshmi
Indian Institute of Technology Hyderabad
Medak, Hyderabad
raji@iith.ac.in

ABSTRACT

This paper presents a compact hardware-software co-design of Advanced Encryption Standard (AES) on the field programmable gate arrays (FPGA) designed for low-cost embedded systems. The design uses MicroBlaze, a soft-core processor from Xilinx. The computationally intensive operations of the AES are implemented in hardware for better speed. The sub-byte calculation is designed with the help of the processor carrying out the calculations using hardware blocks implemented using FPGA. By incorporating the processor in the AES design, the total number of slices required to implement the AES algorithm on FPGA is proved to be reduced. The entire AES system design is validated using 460 slices in Spartan-3E XC3S500E, which is one of the low-cost FPGAs.

Categories and Subject Descriptors

C [**Computer Systems Organization**]: Miscellaneous

General Terms

Design

Keywords

AES, FPGA

1. INTRODUCTION

The intercommunication of information between several organizations and individuals through the computer networks need to be guarded for its rightfulness. Encipherment is the mechanism of using mathematical algorithms to transform the information to be communicated into an inconceivable form. One of the widely used algorithms is the Advanced Encryption Standard (AES) published by National Institute of Standards and Technology (NIST). It is a symmetric block cipher, which is used for encrypting the data and decrypting the encrypted data.

ICACCI '12, August 03 - 05 2012, CHENNAI, India

There exists extensive types of implementations of the AES algorithm in the literature, using hardware or software or using the combination of both. Mostly the software implementations of the algorithm are not suited for embedded systems because of the high cost associated with high performance processors. The full hardware implementations of AES are faster however at the cost of increased number of gates. Thus an optimal way of implementation would be to combine both software and hardware to suit the needs of the low-cost embedded systems. Integrating both the hardware and software components in the design reduces the area requirement and it further optimizes the speed which is sufficient for embedded systems. In this approach, custom-designed instructions are added to the main processor to implement the computation intensive operations.

One of the designs in the literature by Burke et al [1] discusses about the implementation of custom-designed instructions such as 16-bit modular multiplication, bit-permutation support, etc. The paper by Wu et al [2] discusses the CryptoManiac processor, a Very Long Instruction Word (VLIW) machine. It is designed for better speed and also it is adaptable to various cryptographic algorithms. Cryptonite [3] is yet another VLIW machine, with particular instructions to perform AES operations. A number of other implementations on 32-bit platforms are done by Bertoni et al [4], Ravi et al [5] and Nadehara et al [6]. Most of the literature concentrates on designing custom-designed instructions for high-performance using 32-bit processor. For low-cost embedded devices, which does not need such high performance, a low-cost processor customized with cryptographic instructions will be appropriate. This paper discusses such a low-area implementation of the AES algorithm on an 8-bit soft-core processor from Xilinx. The design can provide the security needed for low-cost embedded systems. The hardware implementation proves to be an optimum design in terms of power-speed ratio.

The implementation is done on Spartan 3E XC3S500E FPGA of Xilinx. The PicoBlaze soft-core processor from Xilinx is used. The processor is used to generate the control signals and also to take care of a part of the sub-byte transformation. The implementation on a Spartan-3E FPGA makes it low-cost. The 8-bit soft-core processor of Xilinx was chosen rather than the 32-bit MicroBlaze because there will be a overhead due to extracting 8-bit values from the 32-bit registers during the byte substitution operation and also due to the fact that all the operations implemented in software are of 8-bits. The use of 8-bit architecture will also lead to a small-area and low-power implementation, since the word

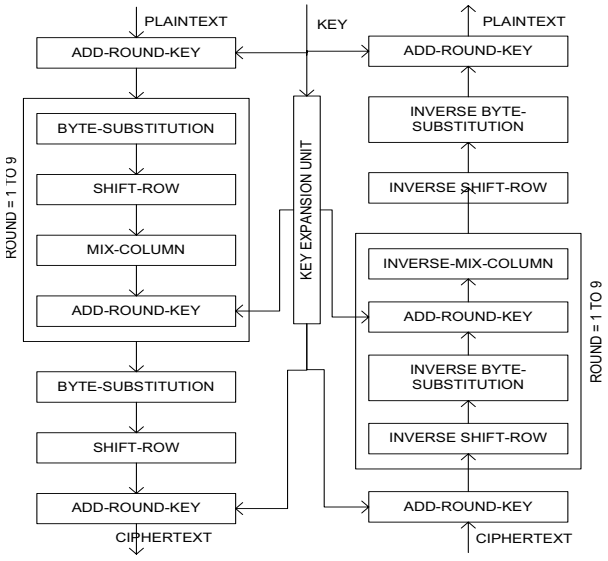

Figure 1: Block diagram of AES encryption and decryption

size controls the area of the circuit and the computation power is reduced in comparison to a 32-bit processor.
The paper is organized as follows. The overview and the basic structure of AES algorithm is discussed in Section II. Section III describes the implementation of the proposed AES design. The implementation results are discussed in Section IV. Finally, Section V concludes the paper.

2. OVERVIEW OF AES

The overall architecture of AES encryption and decryption is shown in Fig. 1 [7]. The input to the encryption block is a 128-bit plaintext, which is the user data. The data and the key are depicted in the form of 4x4 square matrices with each element considered to be of 8-bits size. The input data square matrix is referred as the State matrix, which undergoes the ten rounds of encryption to produce the final ciphertext. Apart from the AES encryption and decryption blocks, the key expansion operates on the initial 128-bit key to give nine more sets of 128-bit key, which is used for each of the rounds from 2 to 10. The four functional blocks of AES encryption and decryption are described as follows.

2.1 Byte Substitution (Forward and Inverse)

In the forward byte substitution [7], each 8-bit element of the state matrix is mapped to a new byte. This is implemented by finding the multiplicative inverse in the Galois Field (2^8), followed by affine transformation (which is a matrix multiplication with a column matrix followed by XOR operation). The inverse byte substitution involves the inverse affine transformation, followed by multiplicative inverse in Galois Field (2^8).

2.2 Shift-Row Transformation

The Shift-Row transform of encryption and decryption operations, is a permutation operation [7], where the rows 1, 2, 3 and 4 of the State matrix undergo a left-shift of 0, 1, 2 and 3 bytes respectively. The inverse shift-row involves the right-shift of the rows by 0, 1, 2 and 3 bytes respectively.

2.3 Mix-Column Transformation (Forward and Inverse)

The forward Mix-Column transformation operates on each column individually. Each byte of a column is mapped to a new value that is a function of all the four bytes from that column. It can be implemented as a matrix multiplication [7] with another 4x4 matrix. The inverse mix-column matrix is found such that the inverse matrix times the forward matrix equals the identity matrix.

2.4 Add-Round-Key transformation

In this step, the State matrix is XORed with the 128-bits of the round key. The forward and inverse Add-Round-Key transformations are identical.
The following sections present the proposed architecture for the implementation of the AES algorithm.

3. DESIGN ARCHITECTURE OF AES IN FPGA

This section discusses the hardware-software co-design approach implemented in this paper in detail. The implemented hardware units are explained first, followed by the discussion of the detailed data-flow operations. The register-transfer-level/block diagram schematics are given along with the explanation.

3.1 Design of Hardware Blocks

3.1.1 Add-Round-Key

It is implemented in hardware as a simple exclusive-or operation of the 128 bit data and key, as shown in Fig. 2.

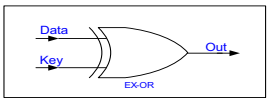

Figure 2: Add-Round-Key.

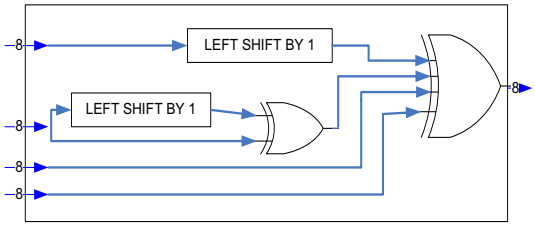

Figure 3: Mix-Column Transformation.

3.1.2 Mix column

Mix-column Transform can be written mathematically as shown in equation 1 [7]. The 8-bit elements of the state matrix are denoted as $s_{x,y}$, where x denote the row and y denote the column of the state matrix. The modified elements are represented as $s'_{x,y}$. The numeric values are written in hexadecimal.

$$\begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} * \begin{pmatrix} s_{0,0}\ s_{0,1}\ s_{0,3}\ s_{0,4} \\ s_{1,0}\ s_{1,1}\ s_{1,2}\ s_{1,3} \\ s_{2,0}\ s_{2,1}\ s_{2,2}\ s_{2,3} \\ s_{3,0}\ s_{3,1}\ s_{3,2}\ s_{3,3} \end{pmatrix} = \begin{pmatrix} s'_{0,0}\ s'_{0,1}\ s'_{0,3}\ s'_{0,4} \\ s'_{1,0}\ s'_{1,1}\ s'_{1,2}\ s'_{1,3} \\ s'_{2,0}\ s'_{2,1}\ s'_{2,2}\ s'_{2,3} \\ s'_{3,0}\ s'_{3,1}\ s'_{3,2}\ s'_{3,3} \end{pmatrix}$$

Multiplication by $(02)_H$ is equivalent to shifting the byte left by 1 bit. Multiplication by $(03)_H$ is equivalent to the XOR operation between the data itself and the data shifted left by one. Thus, each 8-bit element in the modified State matrix will have the operation as depicted in the Fig. 3.
The synthesized RTL schematic result is as shown in the Fig. 4, which is the equivalent to that of Fig. 3.

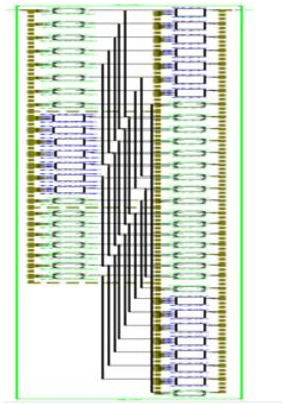

Figure 4: RTL Schematic of Mix-Column Transformation

3.1.3 *Shift row*

Shift row can be merged with the substitute byte transform. That is, the result of the byte substitution is substituted in the 128-bit data register as per the Shifter-Row order with appropriate addressing and control signals generated by the processor through software.

3.1.4 *Sub-Byte Transformation*

The Extended Euclidean algorithm is used to calculate the multiplicative inverse [7][8]. Here, A(x) is the 8-bit element input to the Byte Substitution block. M(x) is the irreducible polynomial in Galois Field (2^8).

3.1.5 *Extended Euclidean Algorithm (A(x)):*

The Extended Euclidean Algorithm can be identified as follows [7]:

R[1]=M(x);R[2]=A(x);
A[1]=0;A[2]=1;
i=2;
While R[i]>1
    i=i+1 ;
    R[i]=Remainder [R[i-2] / R[i-1]];
    Q[i]=Quotient [R[i-2] / R[i-1]];
    A[i] = Q[i] * A[i-1] + A[i-2];
Inverse = A[i];

3.1.6 *Division in GF (2^8)*

The division is implemented in hardware as a combinational circuit consisting of shifting and XOR-operations as shown in Fig. 5. Since it is one of the computationally intensive functions, a combinatorial circuit is designed for better

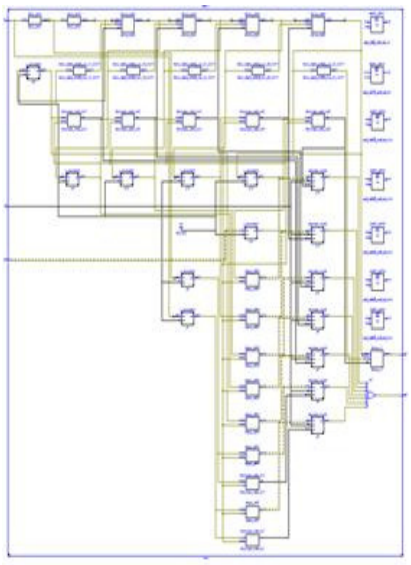

Figure 5: RTL Schematic of Division in GF(2^8) generated by the Synthesizer.

speed. This block is called for execution by the processor whenever sub-byte transformation needs to be done.

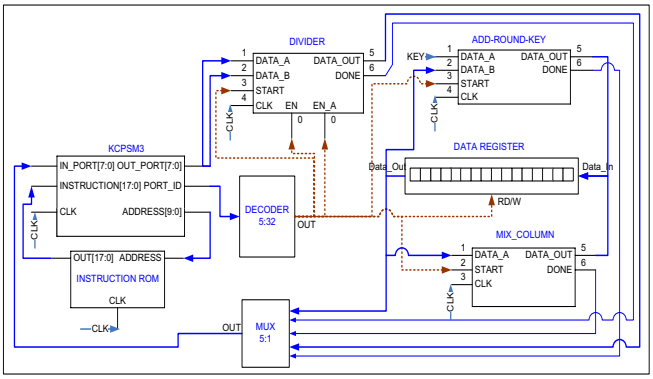

Figure 6: Block Diagram of the AES System.

3.2 Proposed Hardware-Software Co-Design of AES

The block diagram of the whole system is shown in Fig.6. The processor takes care of the sequential steps required to execute the ten rounds. It calls the hardware block functions by giving appropriate 'enable and control signals' to the units.
The custom-designed instructions are implemented for the operations of Add-Round-Key, Mix-Column Transformation, Shift-Row Transformation, the division of two numbers in Galois Field (2^8), and Multiplication. The flow of the execution is explained as follows. As the program execution starts from the instruction ROM, the processor enables the 128-bit Data Register and the input data is clocked in with control signals from the processor (RD/W=0). The Add-Round-Key block starts the execution when Start signal becomes

Table 1: Synthesis Results of the Blocks in the Aes System

Functional Units	Number of Gates or Slices	Number of Slices	Delay (ns)	Number of Clock Cycles
`Add-Round-Key`	128 XOR	74	0.25	1
`Mix-Column`	240 XOR	120	0.504	4
`Divider`	55 XOR, 1 shifter, 1 subtractor, 1 comparator	116	9.652	1
`Multiplier`	36 XOR	19	1.247	1
`PicoBlaze`	-	96	-	-

logic 1 and RD/W=1 for reading the data from the register. On receiving the proper control signals from the processor, the key is XORed with the input data. The output data is written to the 128-bit register with RD/W=0. The processor next progresses to the subsequent instruction, where the bytes from the data register are read into the PicoBlaze one by one.

For the byte substitution, the extended Euclidean Algorithm is implemented as a five stage finite state machine, with each stage having separate control circuits to produce appropriate outputs upon the enabling from the processor. The division operation is implemented as hardware for gaining faster operation. For division of the irreducible polynomial which is of 9 bits, an additional control signal A_*en* is provided, so that only the divisor can be clocked in as input. The result of the byte substitution is written back into the memory in the appropriately shifted position, so that the Shift-Row operation is integrated with Sub-Byte execution. The Mix-Column operation is executed in hardware as an instruction. Upon its finish, the processor instructs the Add-Round-Key block to function, and the whole operation repeats again 8 more times. The key for the remaining rounds are taken from the external hardware implemented key expansion unit. The key expansion unit is designed for on-the-fly-key generation similar to [10]. The final round executes in similar manner, though without the execution of the Mix-Column operation. This information is integrated in the software program. The final Ciphertext is written back into the 128-bit Data Register.

The decryption process is implemented in the same manner, with appropriate modifications in the program along with the hardware of Mix-Column for the implementation of its inverse transform.

4. ANALYSIS OF THE IMPLEMENTATION

A major advantage of the current design is the realization of reduction in the number of S boxes from 4 to 1. This reduces the number of slices needed, as S-box forms the major part of the complete design. Also the Mix-column instruction has four instances of the 32-bit Mix-column module and it finishes the operation for a single byte within 1 clock cycle. The same 4 instances are used for the remaining 3 rows of the state matrix instead of using another 12 instances of the Mix-column hardware.

The design uses an 8-bit processor and 128-bit hardware registers. This 8/128-bit mixed architecture helps in conserving the area, even while achieving increased performance.

The design is coded in VHDL. It is functionally verified and synthesized. Xilinx-XST tool is used for synthesis. The numerical results of the synthesis are shown in Table I. As seen from the table, the total design takes 460 slices. The design is targeted for Spartan 3E FPGA. The picoblaze is operated at 100 MHz. The maximum worst-case delay of the byte substitution block is 80 ns with a clock of period 10 ns. The delay varies depending upon the data as the extended Euclidean algorithm is used.

All the units are verified for correct functionality individually. The Byte Substitution block is verified for all the bit combinations. The waveform for encryption of arbitrary inputs of data and key is shown in Fig. 7. Here, a[0:127] denotes the plaintext in hexadecimal values, key[0:127] denotes the initial 128-bit key, b[0:127] denotes the final ciphertext and w[0:1407] denotes the entire key generated by the external key expansion unit. Similarly the output waveform for

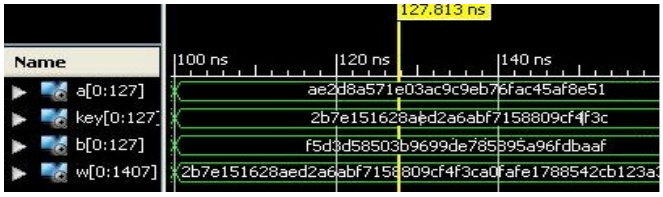

Figure 7: Output Waveform for Encryption with the input plaintext, key, ciphertext and the expanded key shown.

decryption is as shown in Fig. 8, where the input a[0:127] is the ciphertext, and b[0:127] is the resulting plaintext.

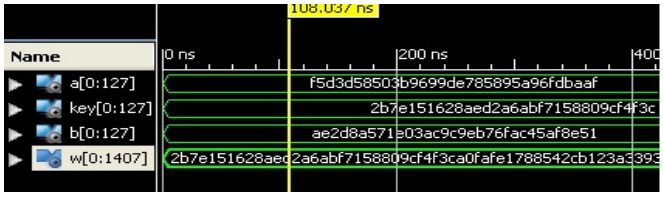

Figure 8: Output Waveform for Decryption with the input ciphertext, key, plaintext and the expanded key shown.

5. CONCLUSIONS

The main objective of this work is to identify and realize a compact design for AES algorithm that involves reduced number of slices, which could be implemented in a low-cost FPGA. The design is suitable for incorporating in low-cost embedded systems where the speed needed is moderate. The synthesized design proves to be compact with respect to the number of slices needed. The unit occupies about 9.9% of the total number of slices (4656) in Spartan 3E FPGA XC3S500E.

The proposed design addresses the area to speed ratio for the AES implementation. As a future scope, the implementation of the byte substitution operation can be implemented

using composite field mapping [9] to decrease the delay incurred by the byte substitution operation.

6. REFERENCES

[1] Burke et al, Architectural Support for fast symmetric-key cryptography, *ASPLOS-IX Proceedings of the ninth International conference on architectural support for programming languages and operating systems*, 2000.

[2] Wu et al, CryptoManiac: a fast flexible architecture for secure communication, *Proceedings of the 28th International Symposium on Computer Architecture*, ISCA 2001.

[3] Dino Oliva, Rainer Buchty and Nevin Heintze, AES and the cryptonite crypto processor, *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*, 2003.

[4] Guido Marco Bertoni, Speeding Up AES By Extending a 32 bit Processor Instruction Set, *Proceedings of the IEEE 17th International Conference on Application-specific Systems, Architectures and Processors*, IEEE Computer Society Washington, DC, USA, 2006.

[5] Fei Sun, Ravi Raghunathan.A and Jha N.K, Custom-instruction synthesis for extensible-processor platforms, *Computer-Aided Design of Integrated Circuits and Systems*, IEEE Transactions on , vol.23, no.2, pp. 216- 228, Feb. 2004.

[6] Nadehara.K, Ikekawa.M and Kuroda.I, Extended instructions for the AES cryptography and their efficient implementation, *Signal Processing Systems* SIPS 2004.

[7] William Stallings, *Cryptography and Network Security,* 4th ed., Pearson, 2009.

[8] Hannes Brunner, Andreas Curiger and Max Hofstetter, On Computing Multiplicative Inverses in GF(2m), *IEEE Transactions on Computers*, Vol. 42, No. 8, August 1993.

[9] Mao-Yin Wang, Chih-Pin Su, Chia-Lung Horng, Cheng-Wen Wu, and Chih-Tsun Huang, Single- and Multi-core Configurable AES Architectures for Flexible Security, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 18, No. 4, April 2010.

[10] H Li, Efficient and flexible architecture for AES, *Circuits, Devices and Systems, IEE Proceedings*, Vol. 153, no. 6, Dec. 2006.