Low-Cost Advanced Encryption Standard (AES) VLSI Architecture: A Minimalist Bit-Serial Approach

Orlando J. Hernandez, Thomas Sodon, and Michael Adel
Department of Electrical and Computer Engineering
The College of New Jersey
hernande@tcnj.edu

Abstract

This paper presents a novel minimum cost architecture for the Advanced Encryption Standard (AES) algorithm. This architecture uses a bit-serial approach, and it is suitable for VLSI implementations. By utilizing a true bit-serial design, this architecture can be used for cost sensitive applications that require high security, such as security system human interfaces, point of sale terminals, and infotainment kiosks. This AES architecture can be used as a coprocessor integrated with an inexpensive microcontroller in a system-on-a-chip (SoC) platform. The prototyping of the architecture is presented as well.

1. Introduction

The need for privacy has become a high priority for both governments and civilians desiring protection from signal and data interception. Widespread use of personal communications devices has only increased demand for a level of security on previously insecure communications.

Both DES (Data Encryption Standard) and AES are defined as symmetric key block ciphers, with the main difference being the bit length of the key (56 bit for DES). These symmetric-key encryption schemes use the same key for both the sender and receiver, and as a result eliminate the need for the verification server needed in public keying. Symmetric keying lends itself to work independently of an open network and in turn a higher level of system interoperability.

Ever since DES was phased out in 2001 and its successor, the Advanced Encryption Standard (also known as Rijndael) took its place, various AES implementations have been proposed both in software and hardware. This paper presents a low cost and low power hardware architecture for the Advanced Encryption Standard (AES). In 1997, the National Institute of Standards and Technology promoted worldwide research into a replacement for DES, or the widely accepted Data Encryption Standard. In this brief, we present an efficient and cost-effective AES co-processor design. To minimize cost, focusing on efficiency reduced overall hardware complexity. By incorporating most of the algorithm complexity into the controller, components are reused and efficiency increased. A Verilog® hardware implementation is also presented, utilizing a field programmable gate array (FPGA) as a prototyping platform. Thus, the design can be easily migrated to an ASIC implementation in an SoC. In this architecture, the main priority was not to increase throughput or decrease processing time but to balance these factors in order to minimize cost. A focus on low power and cost allows for scaling of the architecture towards vulnerable portable communications devices in consumer and military applications such as cellular phones, PDAs, digital radios, pagers, and similar lower speed communication embedded systems.

2. Recent related work

Recent AES implementations have focused on speed gains obtained by manipulations in the SubBytes and MixColumns, two of the more time-consuming functions in the algorithm. One such software technique, called the T-Box algorithm, merges SubBytes and MixColumns in encryption and Inverse SubBytes and Inverse MixColumns in decryption [1]. Another widespread technique used was a BDD architecture and two-level logic to simplify the S-Box. The use of such techniques increases throughput to above 10-Gbps. However, such implementations are fabricated using 0.13 μm technology with clock rates approaching 900 MHz. Even in 0.18 μm CMOS, only 1.6 Gbps is achieved [2]. It is clear that out of all the functions, manipulating SubBytes is the key to increasing performance. Although, custom implementations of modifications to the SubBytes and Mix Columns functions will often result in increased sensitivity to noise and operating temperature. As well as extremely large fan-outs, adding higher propagation delays.

There has been little to no research done regarding lowering power requirements and cost by deemphasizing processing speed. Relatively high throughput (2.381

Gbps) for 128-bit key mode was achieved in one FPGA implementation with a cost of only 58.5K gates [3]. This was done by introducing a 4-stage pipeline for the main functions and performing a basis transformation on SubBytes. The alternative S-Box design would be to use a lookup table (LUT).

3. AES algorithm specification

The AES encryption process [4] for a 128-bit plain text block is shown in Figure 1. The AES algorithm specifies 128-bit, 192-bit, and 256-bit modes. Each bit mode has a corresponding number of rounds (Nr) based on the key length (Nk words). Nb (state block size) is constant for all bit modes. This 128-bit block is termed the state. Each state is comprised of 4 words. A word is subsequently defined as 4 bytes. Table 1 shows the possible key/state block/round combinations.

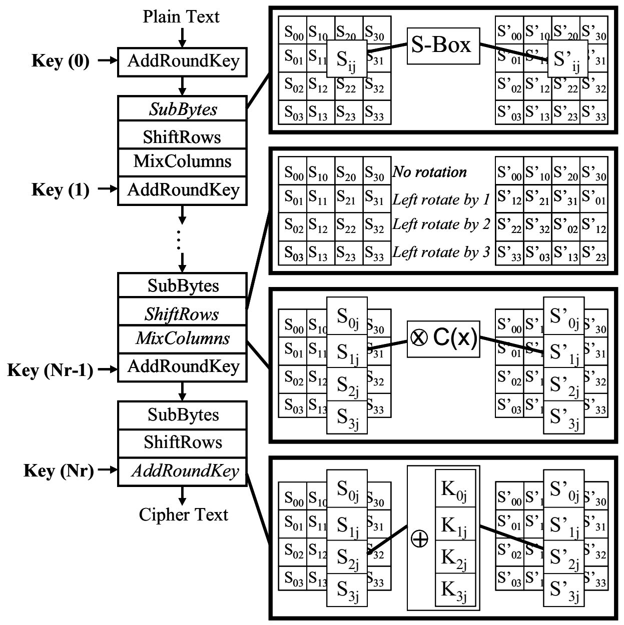


Figure 1. AES encryption algorithm

Table 1. AES bit mode specifications

Bit Mode	Key Length (Nk words)	State Size (Nb words)	Number of Rounds (Nr)
128	4	4	10
192	6	4	12
256	8	4	14

The encrypt/decrypt process involves a sequence of four primitive functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey. AddRoundKey is the same for both encrypt and decrypt. The three other functions have inverses used in the decrypt process: Inverse SubBytes, Inverse ShiftRows, and Inverse MixColumns.

Either encryption or decryption begins with the round key expansion created by the key schedule function. Using the RCON values in combination with a series of XOR, SubBytes, and RotWord (rotate word) operations an expanded round key is generated with a size of $(Nr+1)\times Nb$ bytes. For the 256-bit key expansion, the SubBytes operation is reapplied 4 words after each use of the RCON.

The primitive functions are called Nr times in a loop (called a round). Nr is initialized to 10, 12 or 14 as a function of key length. SubBytes is a nonlinear transformation in which one byte is substituted for another by means of an affine transformation over the Galois Field $GF(2)$, as seen in Equation 1.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (1)$$

ShiftRows is a shift operation performed on the last three rows of the state. The last three rows are rotated to the left by: 1, 2, or 3 bytes shown more specifically in Figure 1. MixColumns is finite field matrix multiplication applied every round except the last. Each column is multiplied as a four-term polynomial in $GF(2^8)\bmod(x^4+1)$ using the array shown in Equation 2.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02&03&01&01 \\ 01&02&03&01 \\ 01&01&02&03 \\ 03&01&01&02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad (2)$$

AddRoundKey performs a bitwise XOR operation with the current state and the round (expanded) key every round including an initial round and the last round. The round key is read from round 0 to Nr for encrypt and vice verse for decrypt. During regeneration of the round key, the decrypt process must halt and wait for generation of the last round key causing a significant delay in initialization.

The decryption process calls the inverse of each function. Inverse SubBytes involves taking an inverse affine transformation. Inverse ShiftRows rotates the bytes to the right by: 3, 2, or 1 byte(s). Inverse MixColumns uses the same operations as MixColumns but uses the inverse matrix shown in Equation 3.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad (3)$$

4. Architecture and hardware design

Before designing the architecture, a preliminary software implementation was developed in order to better understand the algorithm, develop test benches, and anticipate any obstacles in the transition to hardware. The software implementation was written in C, due to its ease of use and similarity to Verilog. A C++ I/O was utilized because of ease of use and interoperability.

The AES algorithm specification FIPS-197 [4] was followed as closely as possible, while minimizing redundancy. Processes shared between the encryption and decryption processes were reused as often as possible. From this software design, it was observed that the MixColumns and SubBytes functions took up significant CPU time. These functions require GF operations that are difficult to negotiate in software. MixColumns specifically required sequential left shifts, each followed by a conditional XOR operation. The condition of the XOR operation depends on there being a 1 in the most significant bit of the current byte before it is shifted. If the condition is true, the shifted byte is XOR-ed with byte {1b}, the irreducible GF polynomial is shown in Equation 4 below [4].

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (4)$$

The affine transformation used by SubBytes can be implemented as a 16x16 lookup table. Every time SubBytes/Inverse SubBytes is called; the table is parsed for the appropriate byte substitution.

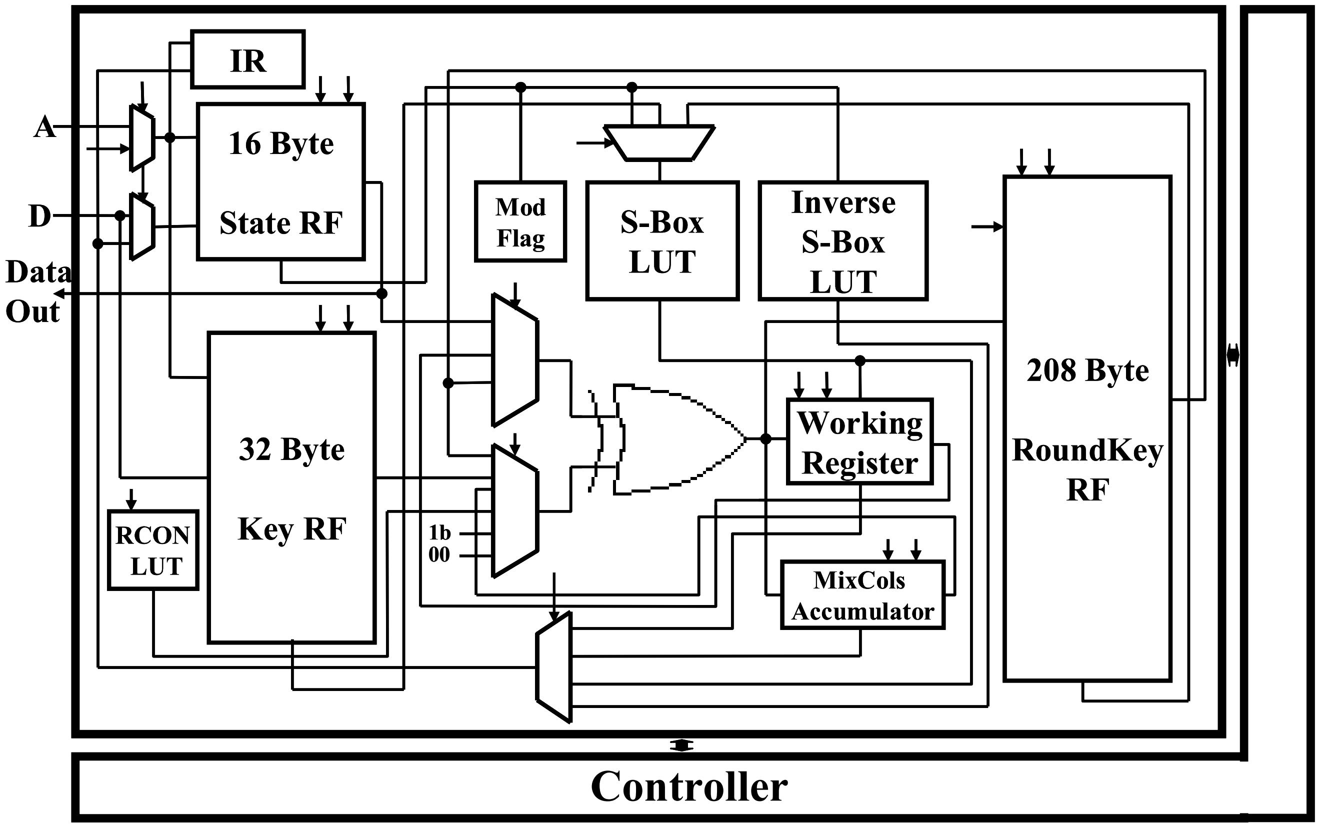


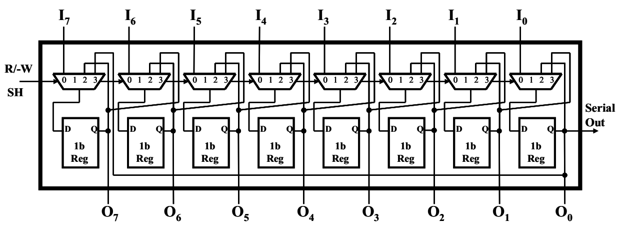
Figure 2. Overall hardware design

There is no simple way to getting around the use of a lookup table, as research into other implementations has shown. An attempt at splitting the search process into sixteen smaller comparisons did not significantly increase efficiency. The LUT used a great deal of memory in software, and similarly a large number of gates in the hardware design.

Various methods for reducing GF circuit size exist, such as composite (or tower field inversion), Fermat's little theorem or extended Euclidean algorithms [2]. However, these methods introduce large propagation delays and increased power consumption. The low-cost and low-power approach minimizes the complexity of GF operations by sacrificing speed.

The basis for the hardware implementation is a result of the work done on the C++ reference code. Using the code, the design intent was to develop a loosely coupled AES co-processor which operates independently of the main processor. The resulting data-path design is shown in Figure 2.

The main components of the datapath are the State RF (Register File), Key RF, RoundKey RF, XOR gate, S-Box, Inv S-Box, Working Register, and MixColumns accumulator. Data and instructions are fed into the module by an 8-bit line and assigned to an 8-bit shift register (see Figure 3) by a 5-bit address line. The data is output serially upon completion of the state operations. The Controller generates control signals for data transportation, key expansion, encryption and decryption.



I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0
R/-W
SH
0 1 2 3
D Q
1b Reg
Serial Out
O_7 O_6 O_5 O_4 O_3 O_2 O_1 O_0

Figure 3. 8-bit shift register

Assuming the data and key are first loaded by the main processor into the register files, the key expansion process can begin. The Key RF and RoundKey RF will receive addresses from the controller allowing individual bytes to be chosen for manipulation. Once the round key is generated, the values are held constant until the main processor assigns a new key. The State RF contains the current state values and changes after each function call.

The Key RF is designed to hold the first 16, 24 or 32 bytes of the round key as shown in Figure 4. Similarly, the RoundKey RF is simply a 208 byte extension of the Key RF design. The State RF is broken down into 4 sets of 4 8-bit shift registers (a single set is shown in Figure 5). Each set constitutes a row, and using a pair of control wires, both the ShiftRows and Inverse ShiftRows operation can be performed internally. The data in the State RF is physically moved from one shift register to the next until the each byte reaches its respective register during either ShiftRows operation.

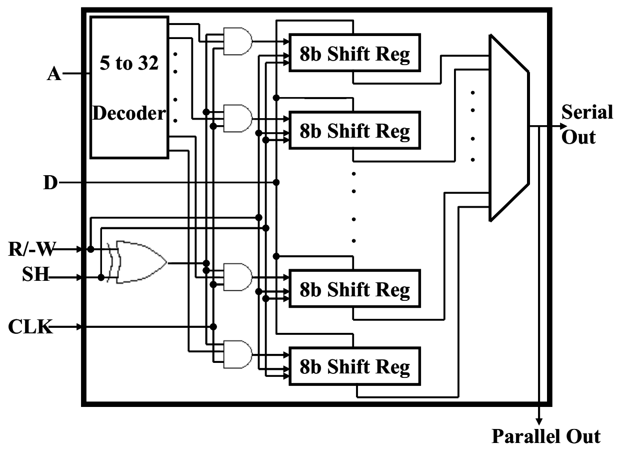


Figure 4. Key register file

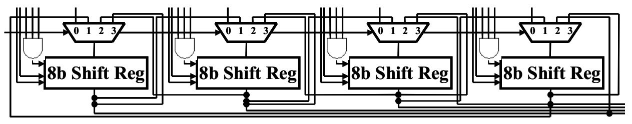


Figure 5. Row shifter in state register file

The S-Box LUT receives an 8-bit input through a multiplexer from either one of the three register files and processes the byte by combinational logic. The Inverse S-Box only receives a connection from the State RF. The S-Box output is tied to the State RF, RoundKey RF, and Working Register; while the Inv S-box ties back to the State RF.

The modules are connected serially to the XOR gate. An 8-bit temporary register (Working Register) is placed at the gate's output to assist in byte operations during the key expansion and MixColumns. The Working Register operates in tandem with the MixCols Accumulator to compute both MixColumns functions. In addition, the byte values {1b} and {00} are held constant at the input of the XOR gate to be used as needed.

The controller reads flags from the ModFlag block and instruction register (IR). The ModFlag block signals the controller as to the status of the most significant bit in the current output byte for use in the conditional XOR. The IR contains the command the co-processor will respond to.

5. Discussion and conclusions

This architecture is being developed and prototyped in an FPGA platform using the Verilog® Hardware Description Language (HDL) and the Spartan II XC2S50

FPGA from Xilinx. This FPGA platform has 50K available gates, 1,728 logic cells, 32 Kb RAM, and 64 I/O ports; which are suitable for the design.

The critical paths in the hardware implementation have shown to be SubBytes, MixColumns, and the decrypt key schedule. The hardware testing phase will reveal the actual delays in algorithm processing. Several design modifications are to be evaluated: modified SubBytes design, and a reduced MixColumns algorithm. By focusing on minimization of redundancies within these functions, cost can be reduced while maintaining a reasonable operating speed.

The SubBytes and MixColumns algorithms can also be merged as other implementations have done. With an advanced chip process, the architecture would perform much faster, and the design can be scaled accordingly. Some aspects of the design can also be parallelized selectively, but will not necessarily reach the levels of other implementations; however, the chief objective of this research was to develop an architecture to minimize the cost of the implementation (i.e. gate count).

Table 2. Comparison of some AES architectures

	[5]	[6]	[3]	THIS
Technology	0.35 µm	0.18 µm	0.35 µm	FPGA
Maximum Clock Rate	N/A	125 MHz	200 MHz	510 MHz
Throughput (Gb/S)	1.95	1.14	2.00	0.37
Gate Count	612K	173K	59K	10K
Throughput/ Gate Count (Kb/S/Gate)	3.18	6.59	34.98	36.13

Table 2 shows comparisons of size and speed, among other attributes, of this implementation with others. As Table 2 demonstrates, this implementation uses only a small fraction of the gates used by other designs that have been reported in the literature, while maintaining an acceptable level of throughput. This is evident from the high efficiency of this design, which is given by the Throughput/Gates figure of merit.

By using a true low level bit-serial approach, a minimum cost AES co-processor architecture has been achieved. This architecture can be used in many military, industrial, and commercial applications that require compactness and low cost.

6. References

[1] S. Morioka and A. Satoh "A 10-Gbps Full AES-Crypto Design with a Twisted BDD S-Box Architecture", *IEEE Transactions on VLSI Systems,* Vol. 12, No. 7, July 2004, pp. 686-691.

[2] V. Fischer and M. Drutarovsky, "Two Methods of Rijndael Implementation in Reconfigurable Hardware", *Proc. CHES,* Vol. 2162, 2001, pp.81-96.

[3] C-P. Su, T-F. Lin, C-T. Huang; and C-W. Wu, "A High-Throughput Low-Cost AES Processor", *Communications Magazine,* IEEE, Vol. 41, Issue: 12, December 2003, pp. 86-91.

[4] NIST, "ADVANCED ENCRYPTION STANDARD (AES, Rijndael)", FIPS-197, November 2001, http://csrc.nist.gov/encryption/aes/.

[5] T. Ichikawa, T. Kasuya, and M. Matsui, "Hardware Evaluation of the AES Finalists," *Proc. 3rd AES Candidate Conf.,* 2000.

[6] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and Performance Testing of a 2.29-Gb/s Rijndael Processor," *IEEE J. Solid-State Circuits,* Vol. 38, No. 3, Mar. 2003, pp. 569-572.