Reversible Logic Implementation of AES Algorithm

Kamalika Datta* Vishal Shrivastav † Indranil Sengupta † Hafizur Rahaman*
*Department of Information Technology, Bengal Engineering & Science University, Shibpur, Howrah 711103, India
Email: kdatta.iitkgp@gmail.com, rahaman_h@it.becs.ac.in
†Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur 721302, India
Email: www.vish.indian@gmail.com, isg@iitkgp.ac.in

***Abstract*—Since the selection of the Rijndael cryptosystem as a new Advanced Encryption Standard (AES) in 2000, many hardware implementations of AES have been reported. Some of these implementations are optimized for speed, some for area, some for reconfigurability, and some for low-power applications. Again, reversible logic synthesis methodologies have drawn the attention of researchers in recent times, mainly with the prospect of quantum computing becoming a reality, and the potential of reversible logic circuits for providing ultra low-power implementations.**

Although many of the cryptographic primitives are inherently reversible by nature, very little work has been done towards reversible logic implementations of the same. The only published works relate to reversible implementations of Montgomery multiplication algorithm, which has applications in cryptography. The present paper, possibly for the first time, presents a reversible logic implementation of a block cipher, namely, 128-bit AES. The various AES functional blocks have been synthesized using reversible gates, using which an overall reversible architecture has been proposed. The pipelined version as suggested can only be used in the Electronic Code Book (ECB) mode. The hardware complexity of the implementation has been evaluated using the number of reversible gates required and the quantum cost.

Keywords: Reversible logic, Advanced Encryption Standard, low-power synthesis

I. Introduction

Reversible logic circuits have attracted the attention of researchers in recent years for mainly two reasons. Firstly, Landauer [8] showed that during logic computation, every bit of information loss generates $KTln2$ joules of heat energy, where K is the Boltzmann's constant and T is the absolute temperature of environment. And, according to Bennet [2], for theoretically zero energy dissipation, computations have to be reversible in nature. Secondly, quantum computions which are the basis of quantum computers are reversible in nature.

In the field of cryptography, there has been many works that propose hardware implementations of cryptographic primitives [13]. Some of these implementations use optimized architectures for high-speed operations, some for area-efficiency targeted to low-cost implementations where speed is not the major concern, some more general-purpose with limited capabilities of reconfigurability, while some optimized for low-power applications. By their very design, some of the cryptographic primitives like encryption and decryption are reversible in nature. However, to the best of the knowledge of the authors, no complete reversible logic implementations of such algorithms have been reported. However, some works on the reversible implementations of specific subsystems of a cryptographic processor, namely the Montgomery multiplier, have been published [11] [14] .

Another motivation for studying reversible logic implementation of cryptographic algorithms results from the fact that side-channels in hardware implementations of such algorithms have been widely studied in recent times [7]. Side-channel attack is considered to be a very cost-effective alternative to attacking traditional cryptographic algorithms, and designers use various countermeasures in this regard. Power analysis attack is one of the easiest attack to mount, and is based on the variations in power dissipation during a computation. Since reversible logic circuits are expected to consume must less energy as compared to traditional CMOS logic, variations in power consumptions will be less and hence side-channel attacks will be more difficult to mount.

With this motivation, this paper reports the results of reversible logic implementation of a state-of-the-art block cipher, the 128-bit Advanced Encryption Standard (AES). The rest of the paper is organized as follows. Section 2 introduces some basic concepts in reversible logic synthesis. Section 3 serves dual purpose; it gives brief introductions to the various steps in AES encryption process, and also discusses the reversible logic implementations of the same. Section 4 discusses the synthesis framework, and presents the experimental results. Section 5 summarizes the paper and identifies a few areas for future work.

II. Reversible Logic and Reversible Gates

A. Preliminaries

A Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}^n$ is said to be reversible if it is bijective. In other words every input vector is uniquely mapped to an output vector. The problem of synthesis is to determine a reversible circuit that realizes a given function f.

In this paper, for the purpose of synthesis we consider the gate library consisting of multiple-control Toffoli (MCT) gates. An n-input MCT gate with inputs $(x_1, x_2, \ldots, x_n)$ pass the first $(n-1)$ inputs unchanged, and complements the last input if all the remaining $(n-1)$ inputs are at 1. Figure 1 shows an n-input MCT gate. A simple NOT ($n = 1$) and controlled-NOT or CNOT ($n = 2$) are special cases of the MCT gate.

Any reversible function can be implemented as a cascade of reversible gates, without any fanout or feedback. To estimate the cost of an implementation, several metrics are used,

978-1-4673-6040-1/13/$31.00

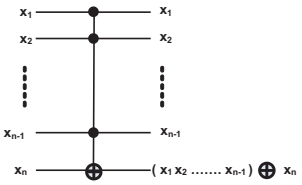

Fig. 1. n-input MCT gate

namely, number of gates, number of equivalent MOS transistors, and number of equivalent basic quantum operations called the quantum cost [1]. There are standard ways of computing the quantum cost from a given gate netlist [3]. Some works also try to reduce the number of *Garbage Outputs*, which are the outputs that are don't cares for all possible input conditions.

B. Synthesis of reversible logic circuits

Various approaches to the synthesis of reversible logic circuits have been reported in the literature, which can be briefly categorized as follows.

- Exact approaches which give the minimum gate realizations, but are very computation intensive and can be used for very small number of inputs only (4 or 5) [5][6]. These approaches typically accept input function specifications as truth table or equivalent input/output permutation.
- Transformation based approaches, which can handle relatively larger functions but the solutions obtained are often sub-optimal [9]. These methods accept functions in either truth table or PLA formats.
- Another class of methods rely on high-level representations like Binary Decision Diagram (BDD) and Exclusive-or-Sum-Of-Products (ESOP), and can handle large functions with more than 100 inputs [4] [15]. Inputs are typically specified in the PLA format. The solution obtained is, however, not optimal, and garbage lines are added in the gate netlist. We have used an ESOP-based synthesis method [12] along with an improved reordering method for reporting the results.

III. AES ALGORITHM AND ITS IMPLEMENTATION

The top-level structure of the AES encryption process is shown in Figure 2, which takes as input an 128-bit plaintext P and an 128-bit key K, and produces as output an 128-bit ciphertext C. The basic steps in the encryption process are shown in Figure 3, which is divided into ten iterations or *rounds*. There are four distinct operations that are carried out in a specific order: *AddRoundKey (ARK)*, *ByteSubstitution (BS)*, *ShiftRow (SR)* and *MixColumns (MC)*. The 128-bit data blocks are divided into groups of 16 bytes each, and organized in the form of a 4×4 *State Matrix*. After an initial ARK step, nine *rounds* are performed, each consisting of a sequence of four operations $\{$BS, SR, MC, ARK$\}$. In the tenth round, only three steps $\{$BS, SR, ARK$\}$ are carried out. The ARK step also takes another 128-bit input, the (transformed) key, which is generated by a separate *Key Scheduler* module as shown in Figure 4. The first ARK step takes the *User Key*, while the following ten rounds use transformed keys.

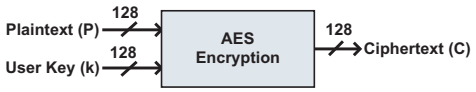

Fig. 2. Top-level schematic of AES encryption

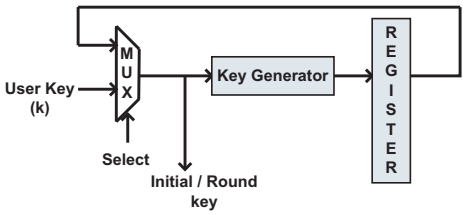

Fig. 4. Iterative key generation

A. Implementation of ByteSubstitution

This step in the AES algorithm carries out a non-linear bijective transformation on each byte of the *State* matrix independently. The transformation is carried out using the S-box, which basically implements a permutation of 8-bit integers (in the range 0 to 255).

Two alternate schemes for implementing an S-box using reversible logic gates is depicted in Figure 5. Figure 5(a) shows the block diagram of an implementation that does not require any garbage output lines, while Figure 5(b) shows the block diagram that uses eight garbage lines. The former approach requires far greater number of gates (and also quantum cost).

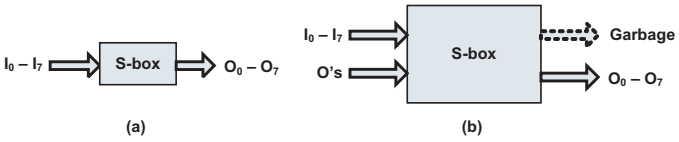

Fig. 5. Alternate reversible designs of the S-box

Since there are 16 bytes of the *State* matrix, to carry out the transformation in parallel, we need 16 S-boxes.

B. Implementation of ShiftRows

This step basically implements fixed cyclic shift operations on the rows of the *State* matrix, and as such can be implemented by permuting the input bits to get the output bits. No gates or hardware components are required for this step.

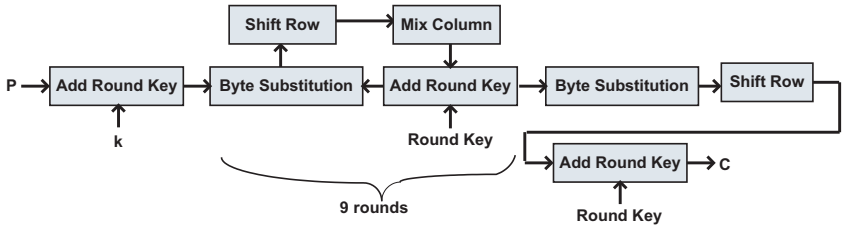

Fig. 3. Steps of AES encryption

C. Implementation of MixColumns

In this step, each column of the *State* matrix is treated as a polynomial over GF(2^8), and is multiplied by a predefined polynomial $03.x^3 + 01; x^2 + 01.x + 01$ modulo $(x^4 + 1)$. This can be formulated using matrix multiplication as follows:

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} O_1 \\ O_2 \\ O_3 \\ O_4 \end{bmatrix}$$

There are four identical *MixColumn* boxes in every round. The inputs and outputs to these boxes are 32-bits long. We represent the inputs as I_1-I_4 and the outputs as O_1-O_4, where each I_i and O_i is 8-bits long, The block level schematic for the reversible implementation of *MixColumns* is shown in Figure 6.

D. Implementation of AddRoundKeys

In this step, the output from the *MixColumns* step is XOR-ed with the corresponding round key. The step can be efficiently implemented in reversible logic using a CNOT gate for every bit. The i^{th} CNOT gate will have the i^{th} bit of key as control input, and i^{th} bit of *MixColumns* output as target. A total of 128 CNOT gates are required for realizing this step. This is shown in Figure 7.

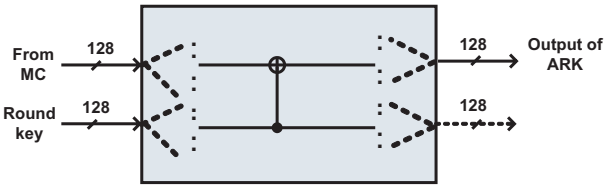

Fig. 7. Reversible implementation of *AddRoundKey*

E. Implementation of Key Scheduler

The *Key Scheduler* is responsible for generating the round keys to be used in the *AddRoundKey* steps. As illustrated in Figure 4, it uses a *KeyGenerator* module in a repetitive fashion to generate the successive round keys. The *KeyGenerator* module has three essential steps:

a) A *rotate left one word* step, that can be implemented by wiring alone.
b) The *ByteSubstitution* step, where the same S-boxes as used in the main encryption flow are used.
c) An 128-bit XOR step, which can again be easily implemented using 128 CNOT gates.

F. Pipelined implementation of AES encryption

In order to have high throughput, we can overlap successive block encryption processes by implementing the AES encryptor as a pipeline. Since the overlapped encryption processes may use different keys, the *Key Scheduler* also needs to be pipelined to generate the round keys for successive encryption processes in an overlapped fashion. A block level diagram of the pipelined implementation is shown in Figure 8.

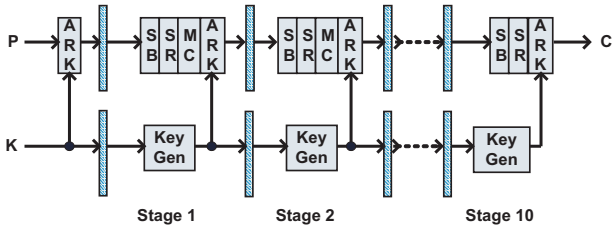

Fig. 8. Reversible pipelined implementation of AES

In Figure 8, each of the blocks *SB*, *SR*, *MC*, *ARK* and *KeyGen* are implemented using reversible logic gates. The registers that serve to isolate the pipeline stages are all 128-bits wide, and are implemented in a reversible way as shown in Figure 9 [14]. Each of the flip-flops is implemented using a Fredkin gate and a CNOT gate, with a quantum cost of $5 + 1 = 6$. An 128-bit register therefore requires 256 gates, with a quantum cost of $128 * 6 = 768$.

Fanouts are not allowed in reversible gate cascades. However, there exists a fanout connection from the output of every *KeyGen* module (see Figure 8). Every individual 1×2 fanout connection can be implemented using a CNOT gate, as shown in Figure 10. A similar idea is followed to construct the flip-flops for the registers [14]. Every 128-bit fanout therefore requires 128 CNOT gates, with a quantum cost of 128.

IV. Results of Synthesis

In this section we present the results of synthesizing various AES subsystems and also the overall encryption process using reversible multiple-control Toffoli (MCT) gates.

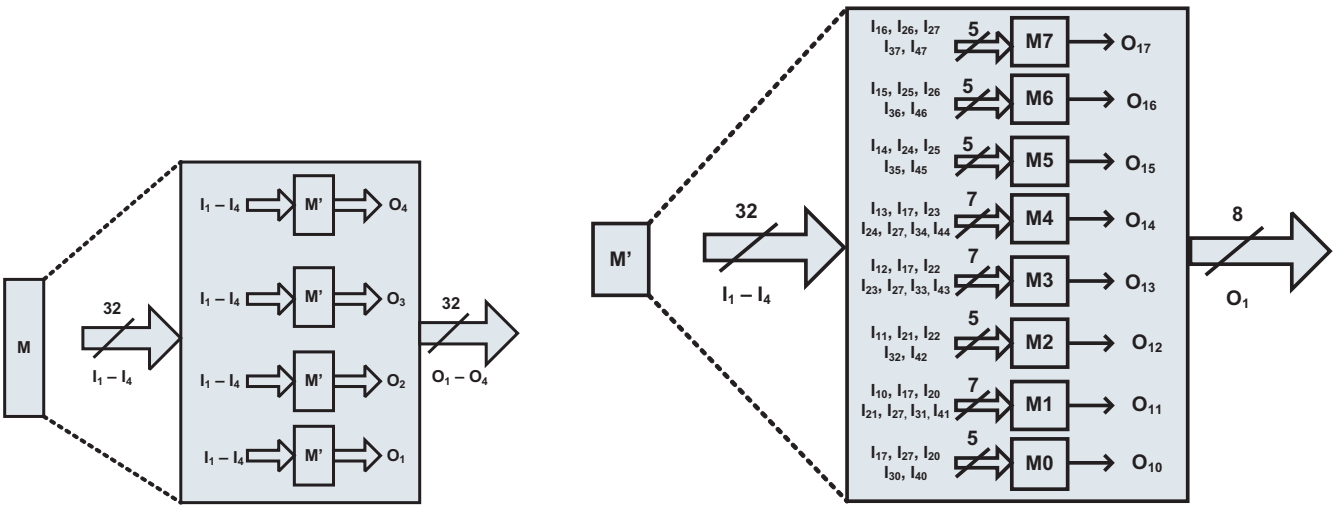

Fig. 6. Reversible schematic of *MixColumns*

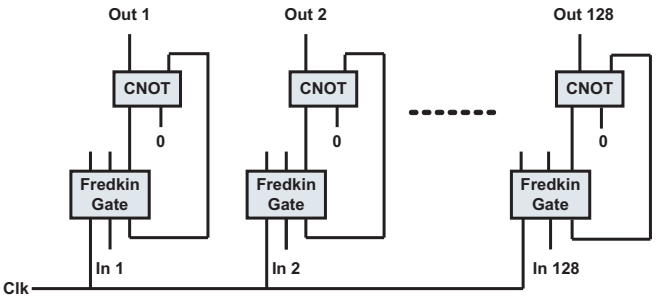

Fig. 9. 128-bit reversible register [14]

Fig. 10. 128-bit reversible register [14]

Each of the AES sub-blocks discussed in the previous section has been synthesized into reversible gate cascades using the MCT gate library. The input specification of each sub-block is first coded in PLA format, and transformed into a set of ESOP cubes using the Exorcism-4 tool [10]. The ESOP cubes are next transformed using a set of optimizing rules and then mapped to reversible gates [12]. It may be noted that not much effort has been put in the present work to optimize the reversible gate netlist that is being generated. The primary objective is to demonstrate that it is possible to have a reversible implementation of the AES encryption process.

To evaluate the cost of implementation, two metrics have been used: (i) gate count (GC), and (ii) cost in terms of equivalent quantum operations called *quantum cost* (QC). Standard methods for calculating quantum cost [1] has been used.

The costs for the different sub-blocks are shown in the following.

a) **ByteSubstitution**: Each 8×8 S-box has been synthesized as 8 reversible circuit blocks implementing the 8 independent functions. It may be noted that the cost of implementation can be reduced if synthesis is carried out treating it as a multi-output function. The costs are summarized in Table I.

TABLE I
IMPLEMENTATION COST OF S-BOX

Subfunction	GC	QC
S1	34	1279
S2	37	1380
S3	37	1285
S4	38	1537
S5	36	1537
S6	39	1756
S7	39	1473
S8	34	1355
TOTAL	294	11602

Since there are 16 S-boxes in the *ByteSubstitution* step, total cost is

$$GC_{BS} = 16 \times 294 = 4704$$
$$QC_{BS} = 16 \times 11602 = 185632$$

b) **AddRoundKey**: This step simply contains 128 XOR operations, that can be implemented using 128 CNOT gates. The total cost is thus

$GC_{BS} = 128$
$QC_{BS} = 128$ (CNOT gate has quantum cost of 1)

c) **MixColumns**: The cost of this step can be computed in a bottom-up fashion following Figure 6. The cost of a single M' block can be computed as shown in Table II.

TABLE II
IMPLEMENTATION COST OF M' BLOCK

Subfunction	GC	QC
M0	5	13
M1	7	19
M2	5	13
M3	7	19
M4	7	19
M5	5	13
M6	5	13
M7	5	13
TOTAL	46	122

Each M block consists of four M' blocks, and hence

$$\begin{aligned} GC_{M\text{-}block} &= 4 \times 46 = 184 \\ QC_{M\text{-}block} &= 4 \times 122 = 488 \end{aligned}$$

The *MixColumn* block consists of four M blocks, and hence

$$\begin{aligned} GC_{MC} &= 4 \times 184 = 736 \\ QC_{MC} &= 4 \times 488 = 1952 \end{aligned}$$

d) **Key Scheduler**: This module consists of 10 *KeyGen* blocks, and each *KeyGen* block requires 16 S-boxes and an 128-bit XOR operation. Thus,

$$\begin{aligned} GC_{KeyGen} &= 4704 + 128 = 4832 \\ QC_{KeyGen} &= 185632 + 128 = 185760 \end{aligned}$$

For implementing the fanout connections from the output of *KeyGen*, 128 CNOT gates are required, with cost

$$\begin{aligned} GC_{fanout} &= 128 \\ QC_{fanout} &= 128 \end{aligned}$$

e) **Register**: Each 128-bit register that is required between the pipeline stages will require 128 Fredkin gates and 128 CNOT gates. The total cost is thus

$$\begin{aligned} GC_{reg128} &= 256 \\ QC_{reg128} &= 128 \times 6 = 768 \end{aligned}$$

The total cost of the full pipelined implementation of AES as shown in Figure 8 is shown in Table III below.

TABLE III
TOTAL COST OF AES IMPLEMENTATION

Sub-block	No. of copies	GC	QC
AddRoundKey	11	1408	1408
ByteSubstitution	10	47040	1856320
MixColumns	9	6624	17568
ShiftRows	10	0	0
KeyGen	10	48320	1857600
Fanout	9	1152	1152
128-bit register	20	5120	15360
TOTAL		109664	3749408

V. CONCLUSION

The reversible logic implementation of the 128-bit AES block cipher has been discussed in this paper. The detailed synthesis results for the encryption module has been presented. To the best of knowledge of the authors, this is the first attempt to implement a block cipher using reversible logic gates. Although the number of reversible gates required is significantly greater than that required using traditional CMOS designs, there are several scopes for optimizing the synthesized netlist in the future. For instance, a template based post synthesis optimization method using negative control MCT gates that is currently under development, can be used to significantly improve the netlist. Since the reversible and traditional CMOS design approaches are not comparable, no attempts have been made to provide the same.

REFERENCES

[1] A. Barenco, H. H. Bennett, R. Cleve, D. P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A (Atomic, Molecular, and Optical Physics)*, 52(5):3457–3467, 1995.

[2] C. H. Bennett. Logical reversibility of computation. *Journal of IBM Research and Development*, 17:525–532, 1961.

[3] R. Drechsler, A. Finder, and R. Wille. Improving ESOP-based synthesis of reversible logic using evolutionary algorithms. In *Proceedings of Intl. Conference on Applications of Evolutionary Computation (Part II)*, pages 151–161, 2011.

[4] K. Fazel, M. A. Thornton, and J. Rice. ESOP-based Toffoli gate cascade generation. In *Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 206–209, 2007.

[5] D. Grosse, R. Wille, G. W. Dueck, and R. Drechsler. Exact multiple control Toffoli network synthesis with SAT techniques. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 28(5):703–715, May 2009.

[6] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(9):1652–1663, September 2006.

[7] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of Advances in Cryptology (CRYPTO '99), LNCS Vol. 1666*, pages 388–397, 1999.

[8] R. Landauer. Irreversibility and heat generation in computing process. *Journal of IBM Research and Development*, 5:183–191, 1961.

[9] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of Design Automation Conference*, pages 318–323, 2003.

[10] A. Mishchenko and M. Perkowski. Fast heuristic minimization of exclusive-sums-of-products. In *Proceedings of 6^{th} Reed-Muller Workshop*, pages 242–250, 2001.

[11] N. Nayeem, L. Jamal, and H. Babu. Efficient reversible Montgomery multiplier and its applications to hardware cryptography. *Journal of Computer Science*, 5(1):49–56, January 2009.

[12] N. Nayeem and J. E. Rice. A shared-cube approach to ESOP-based synthesis of reversible logic. *Facta Universitatis of NiÊ, Elec. Energ.*, 24(3):385–402, 2011.

[13] F. Rodriguez-Henriquez, N. Saqib, A. Perez, and C. Koc. *Cryptographic Algorithms on Reconfigurable Hardware*. Springer: Series on Signals and Communication Technology, New York, 2006.

[14] H. Thapliyal and M. Zwolinski. Reversible logic to cryptographic hardware: a new paradigm. In *Proceedings of 49^{th} Midwest Symposium on Circuits and Systems (MWSCAS '06)*, pages 342–346, 2006.

[15] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Proceedings of Design Automation Conference*, pages 270–275, 2009.