Hardware Design and Analysis of Block Cipher Components

Lu Xiao and Howard M. Heys

Electrical and Computer Engineering
Faculty of Engineering and Applied Science, Memorial University of Newfoundland
St. John's, NF, Canada A1B 3X5
{xiao,howard}@engr.mun.ca

Abstract. This paper describes the efficient implementation of Maximum Distance Separable (MDS) mappings and Substitution-boxes (S-boxes) in gate-level hardware for application to Substitution-Permutation Network (SPN) block cipher design. Different implementations of parameterized MDS mappings and S-boxes are evaluated using gate count as the space complexity measure and gate levels traversed as the time complexity measure. On this basis, a method to optimize MDS codes for hardware is introduced by considering the complexity analysis of bit parallel multipliers. We also provide a general architecture to implement any invertible S-box which has low space and time complexities. As an example, two efficient implementations of Rijndael, the Advanced Encryption Standard (AES), are considered to examine the different tradeoffs between speed and time.

1 Introduction

In a product cipher, confusion and diffusion are both important to the security [1]. One architecture to achieve this is the Substitution-Permutation Network (SPN). In such a cipher, a Substitution-box (S-box) achieves confusion by performing substitution on a small sub-block. An $n \times m$ S-box refers to a mapping from an input of n bits to an output of m bits. An S-box is expected to be nonlinear and resistant to cryptanalyses such as differential attacks [2] and linear attacks [3]. In recently proposed SPN-based block ciphers (e.g., Rijndael [4], Hierocrypt [5], Anubis [6], and Khazad [7]), permutations between layers of S-boxes have been replaced by linear transformations in the form of mappings based on Maximum Distance Separable (MDS) codes to achieve diffusion.

During encryption, as Figure 1 illustrates, typically the input data of each round is mixed with round key bits before entering the S-boxes. Key mixing typically consists of the Exclusive-OR (XOR) of key and data bits. The decryption is composed of the inverse S-boxes, the inverse MDS mappings, and the key mixtures in reverse order. To maintain similar dataflow in encryption and decryption, SPNs omit the linear transformation in the last round of encryption. Instead, one additional key mixture is appended at the end of the cipher for security considerations. If the S-box and the MDS mappings are both involutions [8]

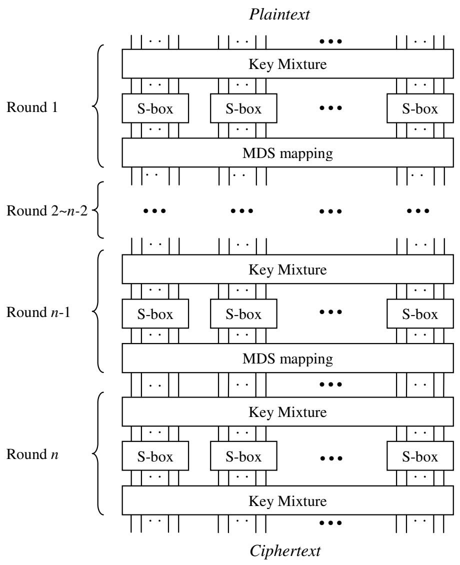


Fig. 1. An SPN with MDS Mappings as Linear Transformation

(i.e., for any input x, $f(f(x)) = x$ where $f(\cdot)$ represents a layer of S-boxes or the MDS layer), both the encryption and decryption operations can be performed by the same SPN except for small changes in the round key schedule in the case of XOR key mixing. We refer to such a cipher as an involution SPN, of which Anubis and Khazad are examples.

An MDS mapping can be performed through multiplications and additions over a finite field. In Galois field arithmetic, additions over a finite field are bit-wise XORs, and multiplications can be calculated as polynomial multiplications modulo an irreducible polynomial. The MDS mapping used in Rijndael is implemented efficiently by several applications of "xtime" [4] (i.e., one-bit left shifting followed by addition with the irreducible polynomial). However, this method only suits the case that all entries in the generation matrix have both low Hamming weights and small magnitudes.

As typically the only nonlinear components in a block cipher, S-boxes must be designed to promote high security. As a result, each bit of an S-box output is a complicated Boolean function of input bits with a high algebraic order, which makes it difficult to optimize or evaluate the complexity of S-boxes generally in hardware[1]. In Section 4, we propose an efficient hardware model of invertible S-boxes through the logic minimization of a decoder-switch-encoder circuit. By use of this model, a good upper bound of the minimum hardware complexity can be deduced for the S-boxes used in SPNs and some Feistel networks (e.g.,

[1] Some special cases with algebraic structure such as the Rijndael S-box can be efficiently optimized.

Camellia [9]). The model can be used as a technique for the construction of S-boxes in hardware so that the space and time complexities are low.

In our work, we take the conventional approach that the space complexity of a hardware implementation is evaluated by the number of 2-input gates and bit-wise inverters; the time complexity is evaluated by the gate delay as measured by the number of traversed layers in the gate network. These measures are not exactly proportional to the real area and delay in a synthesized VLSI design because logic synthesis involves technology-dependent optimization and maps a general design to different sets of cells based on targeted technologies. For example, a 2-input XOR gate is typically larger in area and delay than a 2-input AND gate in most technologies. As well, it is assumed in this paper that the overhead caused by routing after logic minimization can be ignored. Although routing affects the performance in a place-and-routed implementation, it is difficult to estimate its complexity accurately before synthesis into the targeted technology.

From previous FPGA and ASIC implementations of block ciphers such as listed in [10], it is well established that S-boxes normally comprise most of a cipher's area requirement and delay. Although linear components such as MDS mappings are known to be much more efficient than S-boxes, it is important for cipher designers to characterize hardware properties of both S-boxes and MDS mappings on the same basis as is done through the analysis in this paper.

2 Background

2.1 MDS Mappings

A linear code over Galois field GF(2^n) is denoted as an (l, k, d)-code, where l is the symbol length of the encoded message, k is the symbol length of the original message, and d is the minimal symbol distance between any two encoded messages. An (l, k, d)-code is MDS if $d = l - k + 1$. A $(2k, k, k+1)$-code with generation matrix $\mathcal{G} = [\mathcal{I}|\mathcal{C}]$, where $\mathcal{C}$ is a $k \times k$ matrix and $\mathcal{I}$ is an identity matrix, determines an MDS mapping from the input $\mathcal{X}$ to the output $\mathcal{Y}$ through matrix multiplication over a Galois field as follows:

$$f_M : \mathcal{X} \mapsto \mathcal{Y} = \mathcal{C} \cdot \mathcal{X} \tag{1}$$

where

$$\mathcal{X} = \begin{pmatrix} X_{k-1} \\ \vdots \\ X_0 \end{pmatrix}, \quad \mathcal{Y} = \begin{pmatrix} Y_{k-1} \\ \vdots \\ Y_0 \end{pmatrix}, \quad \mathcal{C} = \begin{pmatrix} C_{k-1,k-1} & \dots & C_{k-1,0} \\ \vdots & \ddots & \vdots \\ C_{0,k-1} & \dots & C_{0,0} \end{pmatrix}.$$

Each entry in $\mathcal{X}, \mathcal{Y}$, and $\mathcal{C}$ is an element in GF(2^n).

For a linear transformation, the branch number is defined as the minimum number of nonzero elements in the input and output when the input elements are not all zero [11]. It is desirable that a linear transformation has a high branch number when it is used after a layer of S-boxes in a block cipher, in order for there to be low probabilities for differential and linear characteristics [2, 3]. A mapping based on a $(2k, k, k+1)$-code has an optimal branch number of $k+1$.

2.2 Bit-Parallel Multipliers

An MDS mapping can be regarded as matrix multiplication in a Galois field. Since the generation matrix is constant, each element in the encoded message is the XOR of several outputs of constant multipliers. As basic operators, bit-parallel multipliers given in standard base [12, 13] are selected in this paper. A constant multiplier can be written as a function from element A to element B over GF(2^n) as follows:

$$f_C : A \mapsto B = C \cdot A \tag{2}$$

where C is the constant element in GF(2^n). The expression in binary polynomial form is given as

$$b_{n-1}x^{n-1} + \cdots + b_0 = (c_{n-1}x^{n-1} + \cdots + c_0)(a_{n-1}x^{n-1} + \cdots + a_0) \bmod P(x) \tag{3}$$

where $P(x)$ is denoted as the irreducible polynomial of degree n. An $n \times n$ binary matrix $\mathcal{F}_C$ is associated with this constant multiplier such that:

$$\begin{pmatrix} b_{n-1} \\ b_{n-2} \\ \vdots \\ b_0 \end{pmatrix} = \mathcal{F}_C \times \begin{pmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_0 \end{pmatrix} \tag{4}$$

where

$$\mathcal{F}_C = \begin{pmatrix} f_{n-1,n-1} & \cdots & f_{n-1,0} \\ \vdots & \ddots & \vdots \\ f_{0,n-1} & \cdots & f_{0,0} \end{pmatrix}$$

and $f_{i,j} \in \{0, 1\}, 0 \leq i, j \leq n$–1. The entries in each column of $\mathcal{F}_C$ are determined by

$$f_{n-1,j}x^{n-1} + \cdots + f_{0,j} = x^j(c_{n-1}x^{n-1} + \cdots + c_0) \bmod P(x). \tag{5}$$

Since $\mathcal{F}_C$ is constant, it is trivial to implement a constant bit-parallel multiplier by bit-wise XOR operations. For example, considering a constant multiplier to perform $B = 19\text{H} \times A$ over GF(2^8) where "H" indicates hexadecimal format and $P(x) = x^8 + x^4 + x^3 + x + 1$, we get the binary product matrix $\mathcal{F}_{19\text{H}}$ and the corresponding Boolean expressions for all bit outputs as the following:

$$\mathcal{F}_{19\text{H}} = \begin{pmatrix} 0\,0\,0\,1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,1\,0\,0 \\ 1\,0\,0\,0\,0\,1\,1\,0 \\ 1\,1\,0\,0\,0\,0\,1\,1 \\ 0\,1\,1\,1\,1\,0\,0\,1 \\ 1\,0\,1\,0\,0\,1\,0\,0 \\ 0\,1\,0\,1\,0\,0\,1\,0 \\ 0\,0\,1\,1\,0\,0\,0\,1 \end{pmatrix} \Rightarrow \begin{cases} b_7 = a_4 \bigoplus a_3 \\ b_6 = a_3 \bigoplus a_2 \\ b_5 = a_7 \bigoplus a_2 \bigoplus a_1 \\ b_4 = a_7 \bigoplus a_6 \bigoplus a_1 \bigoplus a_0 \\ b_3 = a_6 \bigoplus a_5 \bigoplus a_4 \bigoplus a_3 \bigoplus a_0 \\ b_2 = a_7 \bigoplus a_5 \bigoplus a_2 \\ b_1 = a_6 \bigoplus a_4 \bigoplus a_1 \\ b_0 = a_5 \bigoplus a_4 \bigoplus a_0 \end{cases}.$$

If we define $w(\mathcal{F}_C)$ as the count of nonzero entries in $\mathcal{F}_C$ and $w_i(\mathcal{F}_C)$ as the count of nonzero entries in the i-th row of $\mathcal{F}_C$, the number of 2-input XOR gates used for the multiplier is upper bounded by $w(\mathcal{F}_C) - n$ and the delay of gate levels is $\max\{\lceil \log_2 w_i(\mathcal{F}_C) \rceil\}$.

2.3 Three Types of Matrices

In the search of optimized MDS mappings in the next section, we will use three types of matrices which suit different applications. When an exhaustive matrix search is impractical, we will limit the search scope to one of the following three matrix types.

- *Circulant matrices*: Given k elements $\alpha_0, \ldots, \alpha_{k-1}$, a circulant matrix $\mathcal{A}$ is constructed with each entry $A_{i,j} = \alpha_{(i+j) \bmod k}$. The probability that a circulant matrix is suitable for an MDS mapping $\mathcal{C}$ is much higher than that of a normal square matrix [14].
- *Hadamard matrices*: Given k elements $\alpha_0, \ldots, \alpha_{k-1}$, a Hadamard matrix $\mathcal{A}$ is constructed with each entry $A_{i,j} = \alpha_{i \oplus j}$. Each Hadamard matrix $\mathcal{A}$ over a finite field has the following properties: $\mathcal{A}^2 = \gamma \cdot \mathcal{I}$ where γ is a constant. When $\gamma = 1$, $\mathcal{A}$ is an involution matrix. An involution MDS mapping is required by an involution SPN.
- *Cauchy matrices*: Given $2k$ elements $\alpha_0, \ldots, \alpha_{k-1}$, $\beta_0, \ldots, \beta_{k-1}$, a Cauchy matrix $\mathcal{A}$ is constructed with each entry $A_{i,j} = 1/(\alpha_i \oplus \beta_j)$. Any Cauchy matrix is MDS when $\alpha_0, \ldots, \alpha_{k-1}$ are distinct, $\beta_0, \ldots, \beta_{k-1}$ are distinct, and $\alpha_i \neq \beta_j$ for all i, j [15]. Although a Cauchy matrix can be conveniently used as matrix $\mathcal{C}$ for an MDS mapping, the relation between selected coefficients (i.e., $\alpha_0, \ldots, \alpha_{k-1}, \beta_0, \ldots, \beta_{k-1}$) and corresponding MDS complexity is not as straightforward as in the former two matrix types. Hence, it is difficult to select coefficients to construct a Cauchy matrix that can be efficiently implemented in hardware.

2.4 A Method to Simplify S-box Circuits

In [16], a method of generating a Boolean function through nested multiplexing is introduced to optimize gate circuits for the 6×4 S-boxes in DES implementations. Consider that a Boolean function $f(a, b, c)$ with three input bits a, b, and c can be written as

$$f(a, b, c) = f_1(a, b) \cdot c + f_2(a, b) \cdot \overline{c}$$

where $f_1(a, b)$ and $f_2(a, b)$ are two Boolean functions and "+" denotes OR. If $f_3(a, b) = f_1(a, b) \oplus f_2(a, b)$, then

$$f(a, b, c) = f_2(a, b) \oplus (f_3(a, b) \cdot c) \ .$$

Similarly, a Boolean function with an input of 4 bits can be regarded as a multiplexor using one input bit to select two boolean functions determined by the other three input bits. This procedure is repeated until a Boolean function has

6 input bits. A 6×4 DES S-box contains four of these 6-bit Boolean functions. This general approach can be taken for any size S-box and works well for optimization of small S-boxes such as the 4×4 S-boxes in Serpent [17]. However, in the case of general invertible 8×8 S-boxes used by many ciphers, this method can be improved upon, as we shall see.

3 Optimized MDS Mappings for Hardware

3.1 Complexity of MDS Mappings

An MDS mapping has been defined in (1) where each entry $C_{i,j}$ of matrix $\mathcal{C}$ is associated with a product matrix $\mathcal{F}_{C_{i,j}}$. Replacing each $C_{i,j}$ in matrix $\mathcal{C}$ with $\mathcal{F}_{C_{i,j}}$ as a submatrix, we get an $nk \times nk$ binary matrix $\mathcal{F}_{\mathcal{C}}$ as the following:

$$\mathcal{F}_{\mathcal{C}} = \begin{pmatrix} \mathcal{F}_{C_{k-1,k-1}} & \dots & \mathcal{F}_{C_{k-1,0}} \\ \vdots & \ddots & \vdots \\ \mathcal{F}_{C_{0,k-1}} & \dots & \mathcal{F}_{C_{0,0}} \end{pmatrix}.$$

Because $\mathcal{Y}$ is the matrix product of $\mathcal{F}_{\mathcal{C}}$ and $\mathcal{X}$, the MDS mapping can be straightforwardly implemented by a number of XOR gates. The gate count of 2-input XORs is upper bounded by

$$G_{MDS} = w(\mathcal{F}_{\mathcal{C}}) - nk \tag{6}$$

and the delay is upper bounded by

$$D_{MDS} = \max\{\lceil \log_2 w_i(\mathcal{F}_{\mathcal{C}}) \rceil\} \tag{7}$$

where $0 \leq i \leq n-1$.

3.2 The Optimization Method

The hardware complexity of an MDS mapping is determined directly by matrix $\mathcal{C}$. In order to improve hardware performance, matrix $\mathcal{C}$ should be designed to produce low hardware complexity. However, not every matrix with low complexity is suitable as an MDS mapping. The mapping associated with matrix $\mathcal{C}$ can be tested using the following theorem:

Theorem 1. [15]: *An (l, k, d)-code with generation matrix $\mathcal{G} = [\mathcal{I}|\mathcal{C}]$ is MDS if, and only if, every square submatrix of $\mathcal{C}$ is nonsingular.*

To minimize gate count and delay in hardware, we want to find an MDS mapping based on a $(2k, k, k+1)$-code over $\mathrm{GF}(2^n)$ with low Hamming weights of $w(\mathcal{F}_{\mathcal{C}})$ and $w_i(\mathcal{F}_{\mathcal{C}})$. Theorem 1 provides us a way to determine whether a matrix candidate is MDS. Theoretically, the optimal MDS mapping can always be determined through an exhaustive search of all matrix candidates of $\mathcal{C}$. However, such a search is computationally impractical when k and n get large. In this

Table 1. Four Choices for MDS Search

Search Options	# of Candidates	Applicable Cases
Exhaustive	$2^{k^2 n}$	small k, n
Circulant Matrices	2^{kn}	large k, n
Hadamard Matrices	2^{kn}	large k, n as well as involution
Cauchy Matrices	2^{2kn}	no MDS mappings found in other matrix categories

case, it is reasonable to focus the search on some subsets of candidates which are likely to yield MDS mappings. The search scope can thus be limited to circulant, Hadamard, and Cauchy matrices.

Table 1 describes four choices for the MDS search. We adopt an appropriate searching method based on the number of candidates to be tested and the required MDS features (involution or not). If computation permits, exhaustive search is preferred. When an exhaustive search is impractical, a search in circulant matrices may be performed for non-involution MDS mappings or a search in Hadamard matrices may be performed for MDS mappings which are involutions. Since only a subset of MDS mappings are derived from circulant, Hadamard, or Cauchy matrices, only exhaustive search over all possible matrices (and therefore all MDS mappings) is guaranteed to find a truly optimized MDS mapping. However for large k and n, searching over a subset of MDS mappings is the best that can be achieved. The objective is to find the candidate with the MDS property and a low hardware cost. The hardware "cost" could be gate count, delay, or both. Sometimes, no candidates in the sets of circulant and Hadamard matrices pass the MDS test. In this case, the optimal mapping will be determined through a search of Cauchy matrices, where each candidate is deterministically MDS.

Once a candidate is proved to be MDS (or involution MDS), those remaining candidates with higher hardware cost can be ignored narrowing the search space. The results generated in this searching method can be used for the hardware characterization of ciphers with MDS mappings of a specified size.

It is noted that $w(\mathcal{F}_\mathcal{C}) - nk$ just indicates the upper bound of XORs in the circuit. Two greedy methods introduced in [13] can be applied to the MDS matrix multiplication in order to further reduce redundancy in the circuit. However, the improvement of using greedy methods is not significant when $w(\mathcal{F}_\mathcal{C})$ is already low.

3.3 MDS Search Results

We have implemented a search for the best MDS mappings of various sizes. During the search, gate reduction is given higher priority than delay reduction because the delay difference among mappings is generally not evident. The optimal[2] non-involution MDS mappings for bit-parallel implementations for various

[2] Here "optimal" means "locally optimal" when the MDS mapping is constrained to a particular matrix category.

Table 2. MDS Search Results

MDS	Galois Field	$P(x)$	Average $w(\mathcal{F}_\mathcal{C})$	Optimal Non-involution MDS $w(\mathcal{F}_\mathcal{C})$	Optimal Non-involution MDS Delay (Gate levels)	Optimal Non-involution MDS Matrix Type	Optimal Involution MDS $w(\mathcal{F}_\mathcal{C})$	Optimal Involution MDS Delay (Gate levels)	Optimal Involution MDS Matrix Type
$(4,2,3)$	GF(2^2)	7 H	8	9	2	exhaustive	11	2	exhaustive
$(4,2,3)$	GF(2^4)	13 H	32	17	2	exhaustive	21	2	exhaustive
$(4,2,3)$	GF(2^8)	11D H	128	35	3	exhaustive	48	3	exhaustive
$(8,4,5)$	GF(2^4)	13 H	128	76	3	circulant	88	3	Hadamard
$(8,4,5)$	GF(2^8)	11D H	512	164	3	circulant	200	4	Hadamard
$(16,8,9)$	GF(2^4)	13 H	512	464	4	Cauchy	544	5	Cauchy
$(16,8,9)$	GF(2^8)	11D H	2048	784	4	circulant	928	5	Hadamard

sizes of MDS mappings are given in Table 2. As in Rijndael, SPNs using these optimal MDS mappings are more efficient in encryption than decryption. In Table 2, the average $w(\mathcal{F}_\mathcal{C})$ is determined by computing the number of matrix entries and dividing by two. These average $w(\mathcal{F}_\mathcal{C})$ values are included to show how effective the optimization work is for each MDS category.

The optimal involution MDS mappings in terms of our complexity analysis are also given in Table 2. Since the MDS test of Theorem 1 is computationally intensive, an involution test will be performed first to eliminate wrong candidates. In [8], an algebraic construction of an involution MDS mapping based on Cauchy matrices is described. This known MDS mapping is used to prune remaining candidates that produce higher complexity before a better mapping is found. These two steps reduce the candidate space dynamically.

The categories in Table 2 correspond to many MDS mappings in real ciphers (although there are minor differences in Galois field selection). For example, Square, Rijndael, and Hierocrypt at the lower level have non-involution MDS mappings based on $(8,4,5)$-codes over GF(2^8) [14, 4, 5]. SHARK has an non-involution MDS mapping based on $(16,8,9)$-codes over GF(2^8) [11]. Hierocrypt at the higher level has two choices of non-involution MDS mappings, based on $(8,4,5)$-codes over GF(2^4) and GF(2^{32}), respectively [5]. Anubis has an involution MDS mapping based on an $(8,4,5)$-code over GF(2^8) [6]. Khazad has an involution MDS mapping based on a $(16,8,9)$-code over GF(2^8) [7]. None these ciphers have MDS mappings with complexity as low as their corresponding cases listed in the tables. The mappings of Rijndael, Anubis, and Khazad have MDS mappings that are close to the optimal cases in terms of gate counts (i.e., $w(\mathcal{F}_\mathcal{C}) = 184, 216,$ and 1296, respectively), while Hierocrypt's MDS mappings have high complexity, similar to the average gate counts.

As Table 2 indicates, the involution MDS mappings are not as efficient as non-involution MDS mappings after optimization. However, the performance difference between them is quite small. When used in an SPN, the involution MDS mapping produces equally optimized performance for both encryption and decryption. When an SPN uses a non-involution MDS mapping optimized only for encryption, the inverse MDS mapping used in decryption has a higher complexity. For example, the MDS mapping used in Rijndael decryption has $w(\mathcal{F}_\mathcal{C}) = 472$

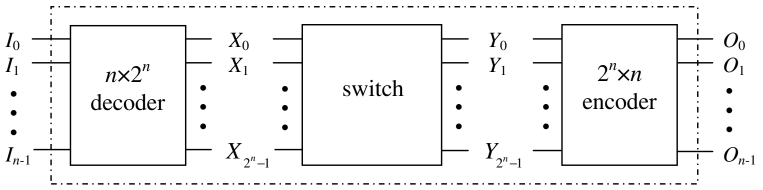


Fig. 2. A General Hardware Structure of Invertible S-boxes

and, hence, needs more gates in hardware than the MDS mapping used for encryption which has $w(\mathcal{F}_\mathcal{C}) = 184$. When a non-involution MDS mapping is optimized for both encryption and decryption, the overall hardware cost is similar to an optimized involution MDS mapping.

The real hardware circuits of these MDS mappings produce complexities with the same trends as shown in Table 2. For example, using Synopsys Design Compiler (with default optimization strategy) and TSMC's 0.18 μm CMOS cell library, we get the area sizes of the optimal non-involution MDS mappings of the bottom four rows of Table 2 as 1549.0, 3659.0, 8863.0, and 17376.4 μm^2, respectively. Their critical time delays are 1.30, 1.33, 2.01, and 2.01 ns, respectively.

4 General Hardware Model of Invertible S-boxes

4.1 Decoder-Switch-Encoder Structure

In this section, we derive a general hardware model of $n \times n$ invertible S-boxes by simplification of a decoder-switch-encoder structure. Using this model, the upper bounds of optimized gate counts and delay for S-boxes can be deduced.

As shown in Figure 2, the $n \times 2^n$ decoder outputs 2^n distinct minterms from the n-bit S-box input. The switch is a wiring area composed of 2^n wires. Each wire connects an input port X_i to an output port Y_j, $0 \leq i, j \leq 2^n - 1$. Since the S-box is invertible, only one input port is connected to an output port. Although the wiring scheme embodies the S-box mapping, the switch does not cost any gates. The output of the switch is encoded through a $2^n \times n$ encoder, which produces the n-bit output of the S-box.

4.2 Decoder

The $n \times 2^n$ decoder is implemented by n NOT gates and a number of AND gates. The NOT gates generate complementary variables of n inputs. The AND gates produce all 2^n minterms from n binary inputs and their complements.

The most straightforward approach is to generate every minterm separately, which costs $2^n \cdot (n-1)$ 2-input AND gates plus n bit-wise NOT gates, and a delay of $\lceil \log_2 n \rceil + 1$ gate levels. This approach can be improved by eliminating redundant AND gates in the circuit. The optimized circuit can be generated using a dynamic programming method.

```
**for** $i \leftarrow 0$ **to** $n-1$ **do**
    $D(i,i) \leftarrow 0$
**for** $step \leftarrow 1$ **to** $n-1$ **do**
    **for** $i \leftarrow 0$ **to** $n-1-step$ **do**
        $j = i + step$
        $D(i,j) \leftarrow \infty$
        **for** $k \leftarrow i$ **to** $j-1$ **do**
            $temp = D(i,k) + D(k+1,j) + 2^{j-i+1}$
        **if** $temp < D(i,j)$ **then** $D(i,j) \leftarrow temp$
**return** $D(0, n-1)$
```

Fig. 3. Algorithm to Determine Decoder AND-Gate Count

Consider the dynamic programming algorithm in Figure 3, used to compute the minimum number of AND gates in the decoder. Let $D(i,j)$ be the minimal number of 2-input AND gates used for generating all possible minterms composed of literals $I_i, \cdots, I_j$ and their complements. Thus, $D(i,j) = 0$ when $i = j$. If we know two optimal results of subproblems, say $D(i,k)$ and $D(k+1,j)$ where $i \leq k < j$, all minterms for $I_i, \cdots, I_j$ can be obtained by using AND gates to connect two different minterms in the subproblems, respectively. Since the number of these pairs is 2^{j-i+1}, this solution needs $D(i,k) + D(k+1,j) + 2^{j-i+1}$ AND gates in total. The algorithm of Figure 3 can be easily modified to determine the actual gate network used for the decoder. When $n = 2^k$, it can be shown that the number of 2-input AND gates and bit-wise NOT gates in the decoder is given by

$$G_{Dec}(n) = n \sum_{i=1}^{k} 2^{2^i - i} + n \ . \tag{8}$$

The delay, in terms of the number of gate levels, of the decoder is

$$D_{Dec}(n) = \lceil \log_2 n \rceil + 1 \ .$$

4.3 Encoder

The $2^n \times n$ binary encoder can be implemented using a number of 2-input OR gates. Table 3 gives the truth table of a 16×4 binary encoder. Each output signal O_i is the OR of the 2^{n-1} input signals that produce "1" in column O_i in the truth table; this is denoted as $O_i = \sum Y_k$. If we separately construct circuits for these output signals, it would cost $n \cdot (2^{n-1} - 1)$ 2-input OR gates and a delay of $n-1$ gate levels. Fortunately, most OR gates can be saved if the same intermediate ORed signals are reused.

Considering that the OR is done in a dynamic programming method, some subproblems used in calculating O_i are also used in calculating O_j if $i > j > 0$. For example, as shown in Table 3, the task of calculating O_{n-1} includes the subproblems of calculating the OR from $Y_{5 \cdot 2^{n-3}}$ to $Y_{6 \cdot 2^{n-3} - 1}$ and calculating the OR from $Y_{6 \cdot 2^{n-3}}$ to $Y_{2^n - 1}$. These two subproblems are also included in the

Table 3. Truth Table of a $2^n \times n$ Encoder

Input	Output			
Y_k	O_3	O_2	O_1	O_0
Y_0	0	0	0	0
Y_1	0	0	0	1
Y_2	0	0	1	0
Y_3	0	0	1	1
Y_4	0	1	0	0
Y_5	0	1	0	1
Y_6	0	1	1	0
Y_7	0	1	1	1
Y_8	1	0	0	0
Y_9	1	0	0	1
Y_{10}	1	0	1	0
Y_{11}	1	0	1	1
Y_{12}	1	1	0	0
Y_{13}	1	1	0	1
Y_{14}	1	1	1	0
Y_{15}	1	1	1	1

(a) $n = 4$

Input	Output			
Y_k	O_{n-1}	O_{n-2}	O_{n-3}	$\cdots$
$Y_0, \cdots, Y_{2^{n-3}-1}$	0	0	0	$\cdots$
$Y_{2^{n-3}}, \cdots, Y_{2^{n-2}-1}$	0	0	1	$\cdots$
$Y_{2^{n-2}}, \cdots, Y_{3\cdot 2^{n-3}-1}$	0	1	0	$\cdots$
$Y_{3\cdot 2^{n-3}}, \cdots, Y_{2^{n-1}-1}$	0	1	1	$\cdots$
$Y_{2^{n-1}}, \cdots, Y_{5\cdot 2^{n-3}-1}$	1	0	0	$\cdots$
$Y_{5\cdot 2^{n-3}}, \cdots, Y_{6\cdot 2^{n-3}-1}$	1	0	1	$\cdots$
$Y_{6\cdot 2^{n-3}}, \cdots, Y_{7\cdot 2^{n-3}-1}$	1	1	0	$\cdots$
$Y_{7\cdot 2^{n-3}}, \cdots, Y_{2^n-1}$	1	1	1	$\cdots$

(b) $n \geq 4$

calculation of O_{n-3} and O_{n-2}, respectively. As a result, the OR gates needed to solve the recurrent subproblems can be saved. Actually, in the procedure of calculating O_i, only the subproblem of calculating the OR from Y_{2^i} to $Y_{2^{i+1}-1}$ has to be solved because all other $2^{n-i-1}-1$ subproblems have been solved in the procedures of calculating $O_{n-1}, \cdots, O_{i+1}$. In this sense, we need 2^i-1 OR gates for the subproblem that has not been solved and $2^{n-i-1}-1$ OR gates to OR the results of all 2^{n-i-1} subproblems. In total, the count of OR gates for the encoder is

$$G_{Enc}(n) = \sum_{i=0}^{n-1}[(2^i-1)+(2^{n-i-1}-1)] = 2^{n+1}-2n-2 \tag{9}$$

and the gate delay is

$$D_{Enc}(n) = n-1.$$

4.4 S-box Complexity

Based on the analysis of the decoder-switch-encoder structure, the hardware complexity of invertible S-boxes is estimated. Since 8×8 S-boxes are very popular in current block ciphers (e.g., Rijndael [4], Hierocrypt [5], and Camellia [9]), let us examine the usability of this model in this case. According to (8) and (9), the upper bound of the optimal gate count for an 8×8 invertible S-box is 806, while the gate count before logic minimization is 2816. Through experimental simplifications using the Synopsys logic synthesis tool [18], we can realize 8×8

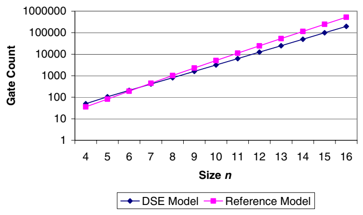


Fig. 4. Gate Count Upper Bounds of S-boxes

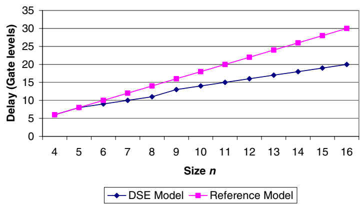


Fig. 5. Delay Upper Bounds of S-boxes

invertible S-boxes with a count of area units close to 800 when the target library is lsi_10k.db. Since a small part of cells in the library have more than 2 inputs, the cell count is around 550. Such a result is quite close to the upper bound when $n = 8$.

When considering the implementation of an S-box in hardware, the upper bound of the gate count increases exponentially with the S-box size n, as shown in Figure 4. Simultaneously, the upper bound of delay increases linearly, as shown in Figure 5. In these two figures, the S-box optimization model described in [16] and presented in Section 2 is used as a reference and the decoder-switch-encoder model is labelled DSE. When the size of an S-box is less than 6, the delay of the two models are similar and the gate count of the reference model is slightly lower. As the size of the S-box increases, the decoder-switch-encoder model costs less in both gate count and delay. The details of gate counts and delays are listed in Table 4 and Table 5. Given the fact that about half the gates used in the reference model are XOR gates which are typically more expensive in hardware than NOT, AND, and OR gates, the decoder-switch-encoder model would appear to be more useful for hardware design, both as an indication of the upper bound on the optimal S-box complexity and as a general methodology for implementing an invertible S-box.

Table 4. Gate Counts of Invertible S-boxes in the Decoder-Switch-Encoder Model

S-box Size	4×4	6×6	8×8	10×10	12×12	14×14	16×16
NOT #	4	6	8	10	12	14	16
AND #	24	88	304	1120	4272	16712	66144
OR #	22	114	494	2026	8166	32738	131038
Gate Count	50	208	806	3156	12450	49464	197198
Reference Count	36	192	1020	5112	24564	114672	524268

Table 5. Gate Delays of Invertible S-boxes in the Decoder-Switch-Encoder Model

S-box Size	4×4	6×6	8×8	10×10	12×12	14×14	16×16
NOT	1	1	1	1	1	1	1
AND	2	3	3	4	4	4	4
OR	3	5	7	9	11	13	15
Delay	6	9	11	14	16	18	20
Reference Delay	6	10	14	18	22	26	30

5 Efficient Rijndael Encryption Implementations

Since Rijndael was selected as AES, it is of great significance to characterize the implementation of Rijndael in hardware. Each round of Rijndael contains the following operations to the state (i.e., the intermediate data stored in a two dimensional array) [4]: (1) a layer of 8×8 S-boxes called ByteSub, (2) a byte-wise cyclic shift per row called ShiftRow, (3) an MDS mapping based on an (8, 4, 5)-code per column called MixColumn, and (4) the round key mixing through XORs. The MDS mapping is defined over GF(2^8) and the S-box performs multiplicative inverse over GF(2^8) followed by a bitwise affine operation.

With parallel S-boxes implemented through table lookups, a hardware design is proposed in [19]. Adhering to the structure of the algorithm specification of [4] as in Figure 6(a), this design achieves a throughput of 1.82 Gbits/sec in 0.18 μm CMOS technology, where each S-box costs about 2200 gates. Since some operations over the composite field GF($(2^4)^2$) are more compact than over GF(2^8), an efficient Rijndael design in composite field arithmetic is proposed in [20]. A cryptographic core (i.e., essentially one round mainly consisting of 16 S-boxes and the MDS mapping layer) in [20] only costs about 4000 gates and a delay of 240 gate levels [21] is expected in theory.

Following the normal encryption dataflow, labelled as Design I in Figure 6(a), we apply the discussed S-box model and MDS bit-parallel implementation method to ByteSub and MixColumn, respectively. After the first round key K_0 is added to the plaintext, the state goes through an iterative round structure. Regardless of its mathematical definition, ByteSub is implemented as a layer of 16 parallel 8×8 S-boxes using the decoder-switch-encoder model. Then, the state iteratively proceeds through ShiftRow, MixColumn, and the addition with round key K_r. ShiftRow is implemented through wiring without any gates needed. Four bit-parallel MDS mappings perform MixColumn for the 4 columns. As listed in Table 6, we get an iterative core circuit of one round which costs 13456 gates and produces a delay of 15 gate levels per round. Because the MDS mappings are omitted in the last round, the Rijndael encryption of 10 rounds produces a delay of 148 gate levels, a significant improvement over the delay of 240 gates levels in the design of [20]. The design needs far fewer gates than in [19].

As shown in Figure 6(b), labelled as Design II, we get a more compact circuit through hybrid operations over GF(2^8) and its equivalent composite field

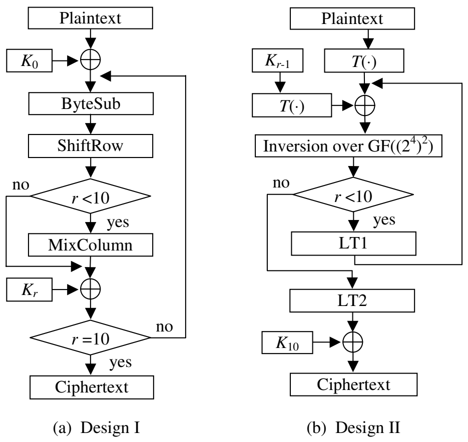


Fig. 6. Rijndael Encryption Implementations

$GF((2^4)^2)$. The polynomial $P_1(y) = y^4 + y + 1$ is used to define $GF(2^4)$ and the polynomial $P_2(x) = x^2 + x + 09H$ is used to define $GF((2^4)^2)$. Such a composite field is the same as in the implementation proposed in [20] for ease of comparison. The conversion from $GF(2^8)$ to $GF((2^4)^2)$ is denoted as $T(\cdot)$, and its inverse is $T^{-1}(\cdot)$.

It has been recognized that the multiplicative inverse over $GF((2^m)^n)$ can have a much lower complexity than the equivalent inverse over $GF(2^{mn})$ [13, 22]. As an example, the equivalent ByteSub over $GF((2^4)^2)$ costs less than one fifth of the gate count of a general invertible S-box based on the upper bound of 806 in the decoder-switch-encoder S-box model. However, the subfield-based operation is normally slow. In the implementation of Figure 6(b), the inverse over the composite field costs a gate delay of 14 (as deduced from [12, 13, 20, 21]). Given additional overhead for field conversion and ByteSub's affine function, the ByteSub instance has a much longer delay path than in the implementation of Design I. To mitigate this problem, we can incorporate all linear operations into LT1 in the first nine rounds and LT2 in the last round as shown in Figure 7, resulting in a delay of 202 gate levels for encryption. The number of gates used in the iterative core circuit is slightly (about 3%) less than in [20]. The detailed gate counts and delays for Design II components are listed in Table 7. The Appendix describes the detailed implementation of LT1 and LT2.

Table 6. Gate Counts and Delays of Operations in Design I

Operations	ByteSub	MixColumn	Key Addition	Total Per Round
Gate Count	12896	432	128	13456
Delay (gate levels)	11	3	1	15

Table 7. Gate Counts and Delays of Operations in Design II

Operations	16×Inversion over GF($(2^4)^2$) [12, 13, 20, 21]	LT1	LT2	$T(\cdot)$	Key Addition	Total Per Round
Gate Count	2384	792	304	208	128	3816
Delay (gate levels)	14	5	3	3	1	20

Figure 8 compares the estimated performance of the two designs of Figure 6. Design I uses the MDS mapping implementation method and S-box model discussed in Sections 2 and 4 directly (while "Design I (Ref.)" uses the reference model in [16] for the S-boxes). In Design II, the method discussed in the Appendix is used to deduce the linear transformations LT1 and LT2. As Figure 8 shows, Design II gains a delay reduction of 16% and a slight reduction in the number of gates compared with the implementation of [20]. Design I is a much faster implementation with about three times as many gates.

The round structures of the two Rijndael designs have been coded in VHDL and synthesized by using Synopsys Design Compiler and TSMC's 0.18 μm CMOS cell library. Setting constraints to tradeoff area and delay during synthesis, we get the characteristic curves shown in Figure 9. The two end points of each curve represent the synthesis results with smallest delay and area. In line with our performance evaluation, Design I can lead to an iterative cipher architecture with a throughput up to 4 Gbits/sec (i.e., the smallest round critical path is 3.04 ns). On the other hand, Design II is useful for an area-restricted or pipelined application because of its small area requirement.

6 Conclusions

We have presented a mechanism to select the MDS mappings for optimal hardware implementation of a block cipher. The optimized MDS mapping straightforwardly leads to a compact and fast implementation at the gate level. As well, a general model of invertible S-boxes is proposed and the upper bounds of the

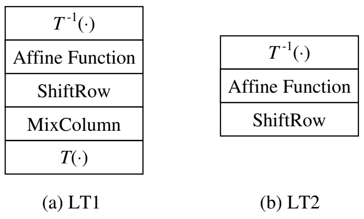
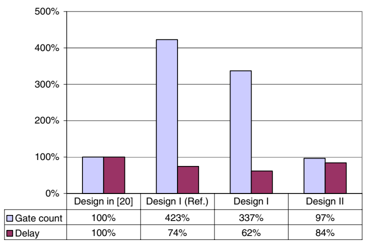


Fig. 7. Linear Transformations in Design II



	Design in [20]	Design I (Ref.)	Design I	Design II
□ Gate count	100%	423%	337%	97%
■ Delay	100%	74%	62%	84%

Fig. 8. Performance Comparison

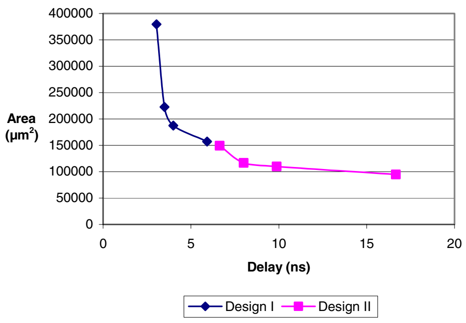


Fig. 9. Synthesis of Round Structure

minimal hardware complexity are deduced through systematic logic minimization. Since S-boxes and MDS mappings are both widely used cipher components, the discussed design, optimization and hardware complexity evaluation provides an analytical basis for studying the hardware performance of block ciphers. As an example, two efficient hardware designs of Rijndael encryption are considered with regards to different tradeoffs between gate count and delay, and their synthesis results are presented.

References

[1] C. E. Shannon, "Communication Theory of Secrecy Systems", *Bell System Technical Journal*, vol. 28, pp. 656-715, 1949. 164

[2] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems", *Advances in Cryptology - CRYPTO '90* , Lecture Notes in Computer Science 537, pp. 2-21. Springer-Verlag, 1991. 164, 167

[3] M. Matsui, "Linear Cryptanalysis Method for DES Cipher", *Advances in Cryptology - Eurocrypt '93*, Lecture Notes in Computer Science 765, Springer-Verlag, pp. 386-397, 1993. 164, 167

[4] J. Daemen and V. Rijmen, "AES Proposal: Rijndael", *Advanced Encryption Standard*, available on: csrc.nist.gov/encryption/aes/rijndael. 164, 165, 171, 174, 176

[5] K. Ohkuma, H. Muratani, F. Sano, and S. Kawamura, "The Block Cipher Hierocrypt", *Workshop on Selected Areas in Cryptography - SAC 2000*, Lecture Notes in Computer Science 2012, Springer-Verlag, pp. 72-88, 2001. 164, 171, 174

[6] P. Barreto and V. Rijmen, "The Anubis Block Cipher", *NESSIE Algorithm Submission*, 2000, available on: `www.cosic.esat.kuleuven.ac.be/nessie`. 164, 171

[7] P. Barreto and V. Rijmen, "The Khazad Legacy-Level Block Cipher", *NESSIE Algorithm Submission*, 2000, available on: `www.cosic.esat.kuleuven.ac.be/nessie`. 164, 171

[8] A. Youssef, S. Mister, and S. Tavares, "On the Design of Linear Transformations for Substitution-Permutation Encryption Networks", *Workshop on Selected Areas in Cryptography - SAC '97*, Ottawa, 1997. 164, 171

[9] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: a 128-bit Block Cipher Suitable for Multiple Platforms", *NESSIE Algorithm Submission*, 2000, available on: `www.cosic.esat.kuleuven.ac.be/nessie`. 166, 174

[10] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, "Report on the Development of the Advanced Encryption Standard (AES)", *Report on the AES Selection from U. S. National Institute of Standards and Technology (NIST)*, available on: `csrc.nist.gov/encryption/aes`. 166
[11] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win, "The Cipher SHARK", *Workshop on Fast Software Encryption - FSE '96*, Lecture Notes in Computer Science 1039, Springer-Verlag, pp. 99-112, 1997. 166, 171
[12] E. D. Mastrovito, "VLSI Design for Multiplication over Finite Fields GF(2^m)", *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes - AAECC-6*, Lecture Notes in Computer Science 357, pp. 297-309, 1989. 167, 177, 178
[13] C. Paar, "Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields", PhD Thesis, Institute for Experimental Mathematics, University of Essen, Germany, 1994. 167, 170, 177, 178, 181
[14] J. Daemen, L. R. Knudsen, and V. Rijmen, "The Block Cipher Square", *Workshop on Fast Software Encryption - FSE '97*, Lecture Notes in Computer Science 1267, Springer-Verlag, pp. 54-68, 1997. 168, 171
[15] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977. 168, 169
[16] E. Biham, "A Fast New DES Implementation in Software", *Workshop on Fast Software Encryption - FSE '97*, Lecture Notes in Computer Science 1267, Springer-Verlag, pp. 260-272, 1997. 168, 175, 178
[17] R. Anderson, E. Biham, and L. Knudsen, "Serpent: a Proposal for the Advanced Encryption Standard", *AES Algorithm Submission*, available on: `www.cl.cam.ac.uk/~rja14/serpent.html`. 169
[18] Synopsys, Online Documentation on Synopsys Design Analyzer, 2000. 174
[19] H. Kuo and I. Verbauwhede, "Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rijndael algorithm", *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2001*, Lecture Notes in Computer Science 2162, Springer-Verlag, pp. 51-64, 2001. 176
[20] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, "Efficient Rijndael Encryption Implementation with Composite Field Arithmetic", *Cryptographic Hardware and Embedded Systems - CHES 2001*, Lecture Notes in Computer Science 2162, Springer-Verlag, pp. 171-184, 2001. 176, 177, 178
[21] A. Rudra, Personal Communication. 176, 177, 178
[22] V. Rijmen, "Efficient Implementation of the Rijndael S-box", available on: `www.esat.kuleuven.ac.be/~rijmen/rijndael`. 177

Appendix: Implementation of LT1 and LT2 in Rijndael Design II

In order to mathematically represent LT1 and LT2, we denote the input state as $\{U_{i,j}\}$ and the output state as $\{V_{i,j}\}$, where i denotes the row index and j denotes the column index of an element in the state. The binary coefficients of $U_{i,j}$ and $V_{i,j}$ in their polynomial expressions can be written as two tuples $\mathcal{U}_{i,j}$ and $\mathcal{V}_{i,j}$, respectively. LT1 can be expressed as

$$\begin{pmatrix} \mathcal{V}_{0,j} \\ \mathcal{V}_{1,j} \\ \mathcal{V}_{2,j} \\ \mathcal{V}_{3,j} \end{pmatrix} = \begin{pmatrix} \mathcal{F}_{L02} & \mathcal{F}_{L03} & \mathcal{F}_{L01} & \mathcal{F}_{L01} \\ \mathcal{F}_{L01} & \mathcal{F}_{L02} & \mathcal{F}_{L03} & \mathcal{F}_{L01} \\ \mathcal{F}_{L01} & \mathcal{F}_{L01} & \mathcal{F}_{L02} & \mathcal{F}_{L03} \\ \mathcal{F}_{L03} & \mathcal{F}_{L01} & \mathcal{F}_{L01} & \mathcal{F}_{L02} \end{pmatrix} \begin{pmatrix} \mathcal{U}_{0,j} \\ \mathcal{U}_{1,j-1} \\ \mathcal{U}_{2,j-2} \\ \mathcal{U}_{3,j-3} \end{pmatrix} + \begin{pmatrix} T(63\text{H}) \\ T(63\text{H}) \\ T(63\text{H}) \\ T(63\text{H}) \end{pmatrix}. \quad (10)$$

In above equation, $\mathcal{F}_{L01}$, $\mathcal{F}_{L02}$, and $\mathcal{F}_{L03}$ are 8×8 submatrices derived from the following expression:

$$\mathcal{F}_{L0i} = \mathcal{F}_T \cdot \mathcal{F}_{0i} \cdot \mathcal{F}_A \cdot \mathcal{F}_T^{-1},\ i = 1, 2, 3 \qquad (11)$$

where $\mathcal{F}_{0i}$ is the product matrix associated with 01H, 02H, or 03H in GF(2^8) and matrix $\mathcal{F}_A$ is associated with the affine function $A(\cdot)$ inside ByteSub (i.e., $A(\mathcal{X}) = \mathcal{F}_A \cdot \mathcal{X} + 63\text{H}$). $\mathcal{F}_T$ is the 8×8 transformation matrix associated with $T(\cdot)$(i.e., $T(\mathcal{U}_{i,j}) = \mathcal{F}_T \cdot \mathcal{U}_{i,j}$). Its inverse is $\mathcal{F}_T^{-1}$.

Similarly, LT2 is a function defined as

$$\begin{pmatrix} \mathcal{V}_{0,j} \\ \mathcal{V}_{1,j} \\ \mathcal{V}_{2,j} \\ \mathcal{V}_{3,j} \end{pmatrix} = (\mathcal{F}_A \cdot \mathcal{F}_T^{-1}) \begin{pmatrix} \mathcal{U}_{0,j} \\ \mathcal{U}_{1,j-1} \\ \mathcal{U}_{2,j-2} \\ \mathcal{U}_{3,j-3} \end{pmatrix} + \begin{pmatrix} 63\text{H} \\ 63\text{H} \\ 63\text{H} \\ 63\text{H} \end{pmatrix}. \qquad (12)$$

Once we know the matrices $\mathcal{F}_T$, $\mathcal{F}_{L0i}$, and the result of $\mathcal{F}_A \cdot \mathcal{F}_T^{-1}$ (as listed in the following), the gate networks consisting of XORs can be straightforwardly derived for LT1 and LT2. The greedy method I described in [13] is used to reduce redundancy in the gate network, where small modifications are made in order to avoid the increase of delay.

$$\mathcal{F}_T = \begin{pmatrix} 1&0&1&0&0&0&0&0 \\ 1&0&1&0&1&1&0&0 \\ 1&1&0&1&0&0&1&0 \\ 0&1&1&1&0&0&0&0 \\ 1&1&0&0&0&1&1&0 \\ 0&1&0&1&0&0&1&0 \\ 0&0&0&0&1&0&1&0 \\ 1&1&0&1&1&1&0&1 \end{pmatrix} \qquad \mathcal{F}_{L01} = \begin{pmatrix} 0&0&0&0&1&0&0&0 \\ 0&1&0&1&0&1&0&0 \\ 1&0&1&0&0&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&1&0&0 \\ 1&0&0&0&1&0&1&0 \\ 0&0&0&1&0&1&0&0 \end{pmatrix}$$

$$\mathcal{F}_{L02} = \begin{pmatrix} 1&0&1&0&1&0&1&1 \\ 1&1&1&1&1&0&1&1 \\ 0&0&1&1&1&1&1&0 \\ 0&1&1&1&0&1&1&0 \\ 0&0&1&1&0&0&1&0 \\ 1&1&1&0&1&1&1&0 \\ 0&0&1&1&1&1&0&0 \\ 0&0&0&0&1&0&1&1 \end{pmatrix} \qquad \mathcal{F}_{L03} = \begin{pmatrix} 1&0&1&0&0&0&1&1 \\ 1&0&1&0&1&1&1&1 \\ 1&0&0&1&1&1&0&0 \\ 0&1&0&1&0&0&1&1 \\ 1&0&1&1&0&0&1&0 \\ 1&1&0&0&1&0&1&0 \\ 1&0&1&1&0&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{pmatrix}$$

$$\mathcal{F}_A \cdot \mathcal{F}_T^{-1} = \begin{pmatrix} 1&0&0&0&0&1&1&0 \\ 1&1&0&1&0&0&0&0 \\ 1&0&0&0&1&1&1&0 \\ 0&1&1&1&1&0&1&1 \\ 0&0&0&0&0&1&0&1 \\ 0&1&0&1&1&0&0&1 \\ 1&0&0&0&1&1&1&1 \\ 0&1&1&0&0&1&0&1 \end{pmatrix}$$