Smashing the Implementation Records of AES S-box

Arash Reyhani-Masoleh, Mostafa Taha and Doaa Ashmawy

Department of Electrical and Computer Engineering
Western University, London, Ontario, Canada
{areyhani,mtaha9,dashmawy}@uwo.ca

Abstract. Canright S-box has been known as the most compact S-box design since its introduction back in CHES'05. Boyar-Peralta proposed logic-minimization heuristics that could reduce the gate count of Canright S-box from 120 gates to 113 gates, however synthesis results did not reflect much improvement. In CHES'15, Ueno et al. proposed an S-box that has a slightly higher area, but significantly faster than the previous designs, hence it was the most efficient (measured by area×delay) S-box implementation to date. In this paper, we propose two new designs for the AES S-box. One design has a smaller implementation area than both Canright and the 113-gate S-boxes. Hence, our first design is the smallest AES S-box to date, breaking the 13 years implementation record of Canright. The second design is faster *and* smaller than the Ueno S-box. Hence, our second design is both the fastest and the most efficient S-box design to date. While doing so, we also propose new logic-minimization heuristics that outperform the previous algorithms of Boyar-Peralta. Finally, we conduct an exhaustive evaluation of each and every block in the S-box circuit, using both structural and behavioral HDL modeling, to reach the optimum synergy between theoretical algorithms and technology-supported optimization tools. We show that involving the technology-supported CAD tools in the analysis results in several counter-intuitive results.

Keywords: AES S-box · Composite/Tower Field Arithmetic · Logic-Minimization Heuristics

1 Introduction

The Advanced Encryption Standard (AES) [FIP01] is a block cipher algorithm that was adopted by the National Institute of Standards and Technology (NIST) as a replacement of the Data Encryption Standard (DES) algorithm back in 2001. AES is essentially a subset of the Rijndael [DR02] algorithm which was the winner of a five-year competition among fifteen block cipher algorithms. AES works as a substitution-permutation network with four main operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. The SubBytes operation uses the Rijndael S-box which is the main non-linear substitution step of AES.

The S-box depends on performing an inversion over the $GF(2^8)$ field of AES, as defined over the irreducible polynomial $(x^8+x^4+x^3+x+1)$, followed by an affine transformation and addition with a constant. The S-box circuit can be implemented using look-up tables, or using field arithmetic. The focus of this paper is to implement the AES S-box circuit using field arithmetic.

Satoh et al. proposed the first tower field inversion using polynomial basis over $GF(((2^2)^2)^2)$ [SMTM01]. This means that an input element from the AES field in $GF(2^8)$ is first converted to two elements in polynomial basis (PB) over $GF(((2^2)^2)^2)$, where

IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925, Vol. 2018, No. 2, pp. 298–336
DOI:10.13154/tches.v2018.i2.298-336

the actual inversion happens, then the result is converted back to the AES field $GF(2^8)$. Canright implemented tower field inversion using normal basis, instead of polynomial basis [Can05b]. In Addition, he conducted an exhaustive search through all the subfield representations (a total of 432 fields) to reduce the overall implementation area. Since then, the research community has proposed many S-box circuits targeting lower-area, lower-delay, and/or higher efficiency (measured by area×delay) than Canright S-box.

In the track of lightweight S-boxes, Boyar, Peralta, along with others proposed several logic-minimization heuristics to reduce the gate count of Canright S-box [BP10, BMP13, VSP17]. Here, they did not explore substantially different subfields, but they focused on reducing the number of gates that are needed to implement the Canright circuit itself. They proposed a reduction in the gate count of the S-box circuit from 120 gates in Canright circuit to 115 gates in [BP10, BMP13], to 114 gates in [VSP17], and to 113 gates in [Boy16]. This 113-gate design, which was recently used by the CHES'17 'bit-sliding' paper [JMPS17], is the currently smallest design, as expressed in terms of gate count.

The design of Canright S-box did not target fast application [Can05b]. Hence, the track of high-speed/higher-efficiency S-boxes received more research focus. [RDJ+01, JKL10, NNT+10, NNI12, UHS+15] are some papers that fall nicely in this latter category. Boyar et al. also proposed low-depth circuits for Canright S-box using delay-controlled logic-minimization heuristics [BP12, BFP17]. It is worth mentioning that the CHES'15 design by Ueno et al. [UHS+15] is the currently fastest and most-efficient S-box design, to the best of our knowledge.

Canright also proposed a combined S-box/inverse S-box architecture, where one slightly bigger circuit can be used to compute the S-box output for the AES encryption-path or the inverse S-box output for the AES decryption-path. This line of research received little research attention, except for [FWR05, JKL10, AH13] where no significant improvements in the area or delay were proposed. Recently, Reyhani-Masoleh et al. proposed a combined S-box/inverse S-box architecture over the tower field architecture of Canright (i.e., $GF(((2^2)^2)^2)$) while using different irreducible polynomials constructing the tower fields than Canright, and hence redesigning the corresponding internal blocks [RTA18]. The proposed architecture in [RTA18] has a lower space and time complexities than the Canright combined S-box/inverse S-box scheme. Note that, in this paper, we use the composite field architecture over $GF((2^4)^2)$ not the tower field over $GF(((2^2)^2)^2)$.

The current trend in designing S-box circuits separates between the theoretical analysis and the technology-supported CAD tools. In other words, the common design goal of theoretical analysis is to reduce the gate-count and/or the circuit-depth, both expressed in indistinctive number of gates, neglecting the fact that logic gates have different implementation area/delay. In addition, theoretical analysis ignores the availability of compound gates in almost any technology library (e.g., the OR-AND-Invert gate), which may perform better in a given circuit. This separation led to many pitfalls as we explain below.

While testing the supposedly most compact S-box circuit (the 113-gate S-box in [Boy16]), we found that the actual implementation area is, in fact, higher than the original Canright design, as we detail in Table 12. This was not readily obvious as Canright provided only behavioral code of the combined S-box/inverse S-box core. We used the equations to write an S-box-only core, in structural modeling, supported by the space complexity provided in the report [Can05a] (also in Table 10), and the circuit was indeed smaller than the 113-gate circuit in [Boy16]. The reason for this counter-intuitive result is that the optimization goal of Boyar-Peralta is to minimize the multiplicative complexity, expressed as the number of $GF(2)$ multiplication operations in the circuit. Then, they directly generated the circuit using the most-commonly used gate for $GF(2)$ multiplication; the AND gate. However, most of the current CMOS technology libraries (e.g., STM, TSMC and NanGate) implement an AND gate as a NAND followed by a NOT gate. Hence, AND gates require higher implementation area and longer delay than the NAND gates that were

used in the original circuit by Canright. Therefore, the improvement brought by reducing the number of gates was lost by using more expensive (in terms of area and delay) gates. In other words, logic gates should not be treated equally in comparing different circuits.

This pitfall was also introduced in the CHES'15 S-box design by Ueno et al. [UHS+15] (and used in the CHES'16 paper [UMHA16]), and by the low-depth S-box circuits in [BP12] and [BFP17], where AND gates were extensively used throughout the circuit instead of the more efficient NAND gates.

Note that the typical use of CAD tools cannot solve this problem as the logic-minimization algorithms implemented within the CAD tool are lacking what is available in research. Hence, although the CAD tools will use more efficient gates, the overall circuit may require a much higher number of gates. In Sec. 8.1, we improve the 113-gate S-box of [Boy16] to use NAND gates instead of AND gates, resulting in the first S-box circuit to have an *actual* lower implementation area than the one proposed by Canright (not just the gate count), breaking a 13-years implementation record. We also improve the CHES'15 S-box [UHS+15] and the low-depth circuits of [BP12, BFP17].

In order to avoid the aforementioned pitfall, we follow a unique design approach throughout this paper. We use all the available logic-minimization methods, supported with Boolean algebra, Karnough maps and De Morgan's laws to derive optimum circuits for each block of the S-box circuit. Then, we invoke the CAD tools with behavioral modeling, under different optimization goals and different levels of circuit-segmentation, to investigate if the CAD tools can bring any smaller and/or faster circuits. At the end, we hand-pick the best circuits in each block of the S-box under the two design criteria (lightweight and fast). While doing so, we also propose several improvements to the underlying logic-minimization algorithms.

As a result, we propose two new S-box circuits. Our first S-box circuit achieves an implementation area that is smaller than Canright S-box, the 113-gate S-box, and our own improved version of these circuits. This lightweight circuit sets a new record for the implementation area of the AES S-box. Our second circuit achieves higher-speed *and* higher-efficiency than the CHES'15 design [UHS+15], the low-depth circuits of [BP12, BFP17], and our improved versions of [BP12, BFP17]. Our fast circuit has a very similar delay to the improved version of [UHS+15] (as proposed in this paper), while having a much higher efficiency.

The rest of the paper is organized as follows. Sec. 2 reviews some mathematical background about multiplicative inversion using composite field arithmetics. Sec. 3 introduces the overall architecture of the proposed S-box designs. The following two sections provide details on the underlying blocks within the S-box circuit. The input and output transformation blocks are discussed in Sec. 4, along with the new logic-minimization heuristics. The design blocks of the composite field inversion (exponentiation, subfield inversion, and multipliers) are discussed in Sec. 5. Other plausible design options that we considered, but did not lead to better results, are discussed in Sec. 6. Further optimization brought by the technology-supported CAD tools are discussed in Sec. 7. The hardware results of our designs are introduced in Sec. 8. In this section, we also propose improvements to the previous contributions and comparison against our designs. We conclude the paper in Sec. 9.

2 Preliminaries

In this section, we briefly discuss the preliminaries regarding the used finite fields. Specifically, we talk about different representations used in this paper, namely the polynomial basis (PB) representation used in the binary field $GF(2^8)$ and the normal basis (NB) representation in the composite field $GF((2^4)^2)$. Let us assume that the input to the S-box is g, where g is an element in $GF(2^8)$ generated by the irreducible polynomial

$q(x) = x^8 + x^4 + x^3 + x + 1$. Let α be a root of $q(x)$, then $g = (g_7, \cdots, g_1, g_0) \in GF(2^8)$ is represented in the PB representation as $g = \sum_{i=0}^{7} g_i \alpha^i$, $g_i \in GF(2)$, $0 \le i \le 7$, where g_i is the coordinates of $g \in GF(2^8)$. For convenience, these coordinates will be denoted in vector notation as $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$, where tr denotes the transposition.

2.1 S-box Computation

Let $f = (f_7, \cdots, f_1, f_0) \in GF(2^8)$ be the multiplicative inverse (or inverse in short) of the non-zero S-box input, i.e., $f \times g = 1$, where $g \neq 0$. Then, the first step in the S-box computation is to find the inverse $f = g^{-1}$ for $g \neq 0$. For $g = 0$, $f = 0$. Let $s = (s_7, \cdots, s_1, s_0) \in GF(2^8)$ be the S-box output. Then $\mathbf{s} = [s_7, \cdots, s_1, s_0]^{tr} = \mathbf{M}\mathbf{f} \oplus \mathbf{h}$, i.e.,

$$\mathbf{s} = \begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_7 \\ f_6 \\ f_5 \\ f_4 \\ f_3 \\ f_2 \\ f_1 \\ f_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \tag{1}$$

where $\mathbf{M}$ is an affine transformation, $\mathbf{h}$ is a constant, and $\oplus$ is the modulo 2 addition (XOR).

2.2 Composite Field Inverse in Normal Basis

The implementation of AES S-boxes using composite fields reduces the chip area for the S-box computation. Two types of composite fields, namely $GF(((2^2)^2)^2)$ and $GF((2^4)^2)$ are typically used. In this paper, we use the composite field $GF((2^4)^2)$ which is referred to as the composite field. We also refer to the $GF(((2^2)^2)^2)$ field as the tower field in order to distinguish it from the composite field $GF((2^4)^2)$ used in this paper.

In the proposed scheme, we convert the elements to an isomorphic composite field $GF((2^4)^2)$. The irreducible polynomial over $GF(2^4)$, namely

$$p(y) = y^2 + \mu y + \nu = (y + \gamma)(y + \gamma^{16}), \tag{2}$$

is used to construct the composite field. In (2), γ is its root and the subfield elements $\mu, \nu \in GF(2^4)$ should be chosen so that this polynomial is irreducible over $GF(2^4)$. Then, $\{\gamma, \gamma^{16}\}$ is the normal basis and every element g in $GF(2^8)$ can be mapped to its composite field representation over $GF(2^4)$ as $g = A\gamma + B\gamma^{16}$, where $A = (a_0a_1a_2a_3)$ and $B = (b_0b_1b_2b_3)$ are subfield elements in $GF(2^4)$ and a_i and b_i are their binary coordinates, respectively. In this paper, the subfield $GF(2^4)$ is generated using the irreducible all-one-polynomial (AOP) with degree four, i.e.,

$$r(t) = t^4 + t^3 + t^2 + t + 1. \tag{3}$$

Let β be a root of $r(t)$, i.e., $r(\beta) = 0$. Then, we use the type-I optimal normal basis (ONB-I) $\{\beta, \beta^2, \beta^{2^2}, \beta^{2^3}\}$ to represent field elements and their efficient computations over $GF(2^4)$ [RH03]. Then, any field element $A = (a_0a_1a_2a_3) \in GF(2^4)$ can be represented as $A = a_0\beta + a_1\beta^2 + a_2\beta^{2^2} + a_3\beta^{2^3}$.

The inverse of $g = A\gamma + B\gamma^{16}$ in the composite field $GF((2^4)^2)$ can be written as [IT88], [Paa94], $g^{-1} = (g^r)^{-1}g^{r-1}$, where $r = \frac{2^{4\times2}-1}{2^4-1} = 17$. Then, $g^{-1} = (g \times g^{16})^{-1}g^{16}$

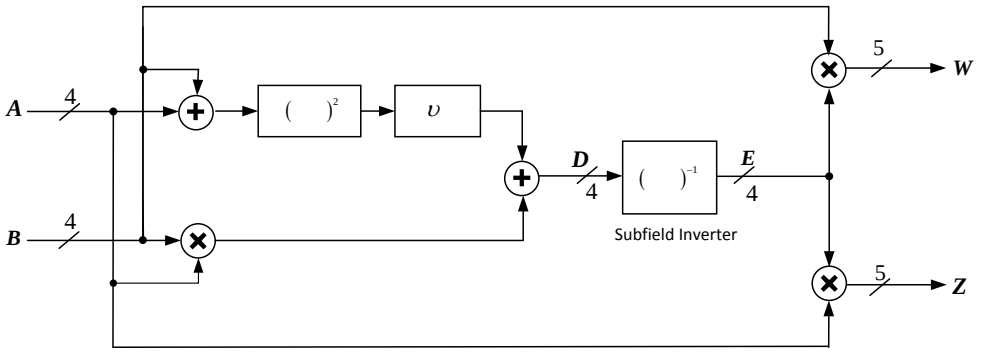

Figure 1: The architecture for $GF\left((2^4)^2\right)$ inversion using the NB representation at the inputs and the redundant normal basis (RNB) representation at the outputs.

can be computed as

$$\begin{aligned} g^{-1} &= [(A\gamma + B\gamma^{16})(B\gamma + A\gamma^{16})]^{-1}(B\gamma + A\gamma^{16}) \\ &= [AB(\gamma + \gamma^{16})^2 + (A+B)^2\gamma\gamma^{16}]^{-1}(B\gamma + A\gamma^{16}) \\ &= EB\gamma + EA\gamma^{16} = W\gamma + Z\gamma^{16}, \end{aligned} \tag{4}$$

where

$$E = D^{-1} = [g^{17}]^{-1} = [AB(\gamma + \gamma^{16})^2 + (A+B)^2\gamma\gamma^{16}]^{-1}, \tag{5}$$

$W = EB \in GF(2^4)$ and $Z = EA \in GF(2^4)$. Since γ and γ^{16} are roots of (2), one can obtain $\gamma + \gamma^{16} = \mu$ and $\gamma\gamma^{16} = \nu$. Thus, $D = g^{17} \in GF(2^4)$ in (5) is simplified to

$$D = AB\mu^2 + (A+B)^2\nu. \tag{6}$$

For simplicity and low complexity implementations, we choose $\mu = 1 \in GF(2^4)$ and then (2) and (6) would respectively become

$$p(y) = y^2 + y + \nu = (y + \gamma)(y + \gamma^{16}), \tag{7}$$

and

$$D = AB + (A+B)^2\nu. \tag{8}$$

Figure 1 shows an architecture to implement (8) and (4). Note that the underlying arithmetic operations in Figure 1, i.e., an addition, a squaring, three multiplications and multiplication by the constant ν (scaling), are implemented over subfield $GF(2^4)$. In [Can05b, Can05a], Canright used the tower field $GF((2^2)^2)$ and chose $\mu = 1$ for the normal basis scheme. Then, the same architecture as shown in Figure 1 was derived using a similar approach.

2.3 Subfield Multiplier

In this paper, we use type-I optimal normal basis (ONB-I) multiplication scheme proposed in [RH03] over $GF(2^m)$ for $m = 4$. A number of generic multiplication schemes are proposed in [RH03]. Specifically, we use equation (36) and the corresponding Algorithm 3 from [RH03] for the multiplication over $GF(2^4)$ generated by the AOP.

Lemma 1. *[RH03] Let $A = (a_0a_1a_2a_3) \in GF(2^4)$ and $B = (b_0b_1b_2b_3) \in GF(2^4)$ be represented in the ONB-I $\{\beta, \beta^2, \beta^{2^2}, \beta^{2^3}\}$. Then, the coordinates of their multiplication,*

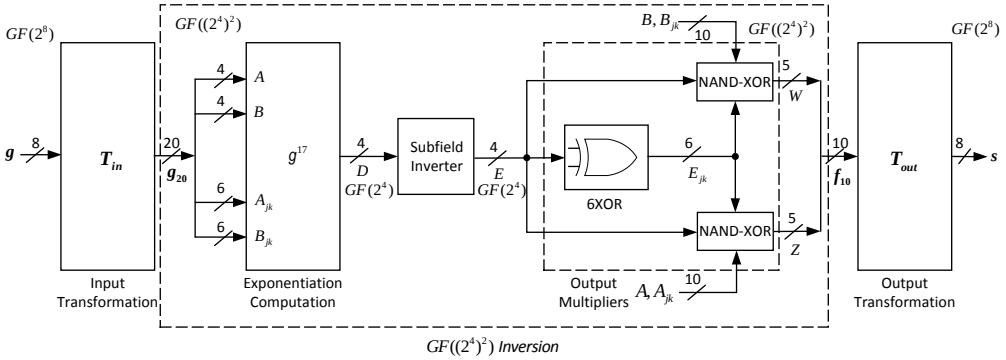

Figure 2: The proposed S-box architecture.

i.e., $C = AB = \sum_{i=0}^{3} c_i \beta^{2^i} + c_4 \in GF(2^4)$, *represented in the redundant normal basis (RNB)* $\{\beta, \beta^2, \beta^{2^2}, \beta^{2^3}, 1\}$, *can be calculated as follows:*

$$\begin{aligned} c_0 &= a_0 b_0 \oplus a_{12} b_{12} \\ c_1 &= a_1 b_1 \oplus a_{23} b_{23} \\ c_2 &= a_2 b_2 \oplus a_{30} b_{30} \\ c_3 &= a_3 b_3 \oplus a_{01} b_{01} \\ c_4 &= a_{02} b_{02} \oplus a_{13} b_{13}, \end{aligned} \tag{9}$$

where $a_{jk} = a_j \oplus a_k$ *and* $b_{jk} = b_j \oplus b_k$ *for* $0 \le j, k \le 3$, *and* $j \ne k$.

We use this multiplier for the two output multipliers and the one at the beginning of the $GF\left((2^4)^2\right)$ inversion in Figure 1.

3 Proposed AES S-box Architecture

The overall architecture of the proposed S-box circuits is highlighted in Figure 2. The input and output transformation blocks are responsible for converting elements between the AES binary field $GF(2^8)$ and the corresponding elements in the composite field $GF((2^4)^2)$. The input transformation block accepts an 8-bit element $\mathbf{g}$ from the $GF(2^8)$ field, and generates two 4-bit field elements A and B, and the mod-2 addition between every two bits in each of the two elements of A and B, i.e., A_{jk} and B_{jk}, where A_{jk} is a set that contains $a_j \oplus a_k$, and B_{jk} contains $b_j \oplus b_k$, both for $0 \le j, k \le 3$, and $j \ne k$. The output transformation block accepts the result of the two subfield multipliers; W and Z (5 bits each, following (9)), and generates the corresponding element $\mathbf{s}$, i.e., the S-box output, in the AES $GF(2^8)$ field. More details about generating, optimizing, and implementing the transformation blocks, targeting lightweight and fast applications, are discussed in Sec. 4.

The composite field inversion is composed of three sub-blocks; the exponentiation block, the subfield inverter, and the output multipliers. The exponentiation block is responsible for computing $D = g^{17}$ as expressed in (8). The subfield inverter accepts g^{17} and generates $E = (g^{17})^{-1}$. The output multipliers perform the multiplication between $(g^{17})^{-1}$ and g^{16} to represent the output in the redundant normal basis (RNB). The exact circuits that we use in the implementations of each of these blocks are introduced in Sec. 5.

4 Input and Output Transformations

The conversion between corresponding elements in two different field representations is realized by a binary transformation matrix. We denote the matrix that is used to convert

from $GF(2^8)$ to $GF\left((2^4)^2\right)$ as $\mathbf{X}^{-1}$, and the matrix that is used to convert in the other way around, after its product with $\mathbf{M}$, as $\mathbf{MX}$, where $\mathbf{M}$ is the affine transformation in the S-box. We use the same notation that was used by Canright [Can05b] for less confusion.

The mapping realized by the transformation matrices should preserve additive, as well as, multiplicative isomorphisms. In other words, addition (or multiplication) between two elements in one field should be equivalent to mapping the two elements to the new field, performing the addition (or multiplication) under the laws of that new field, and mapping the result back to the original field.

In this section, we briefly review how to generate these matrices, along with the existing logic-minimization algorithms which are used to minimize the number of logic gates in their implementations. Thereafter, we highlight possible improvements in the existing logic-minimization algorithms and propose new ones. We conclude this section with lightweight and fast circuits for the selected input and the corresponding output transformation matrices.

4.1 Generating Transformation Matrices

A generator ϕ, as the name implies, can be used to generate all the non-zero field elements by raising it to a positive integer power $\phi^i, i \in [0, n-2]$, where n is the size of the field. Let ϕ_1 be a generator in the $GF(2^8)$ field of AES, and ϕ_2 be a generator in the new $GF((2^4)^2)$ field that we need to use. An input transformation matrix works if it can map all the elements $\phi_1^i, i \in [0,\ 2^8-2]$ in $GF(2^8)$ to $\phi_2^i, i \in [0,\ 2^8-2]$ in $GF\left((2^4)^2\right)$. We select ϕ_1 to be $\phi_1 = 00000011 = (03)_h$ [Can05b]. However, in order to select ϕ_2, we need first to specify $\nu \in GF(2^4)$ in (7) and (8), which affects the multiplication rule under this field.

ν is a 4-bit field element in $GF(2^4)$ chosen so that $p(y) = y^2 + y + \nu$ is irreducible over $GF(2^4)$. Among the 16 possible combinations of ν, only 8 field elements namely $\nu \in \{\beta,\ \beta^2,\ \beta^{2^2},\ \beta^{2^3},\ 1+\beta,\ 1+\beta^2,\ 1+\beta^{2^2},\ 1+\beta^{2^3}\}$ can be used. Representing these 8 field elements in the NB, one can obtain $\nu \in \{(1000), (0100), (0010), (0001), (0111), (1011), (1101), (1110)\}$. Other values cannot be used as the polynomial $p(y) = y^2 + y + \nu$ over $GF(2^4)$ will not be irreducible. Each value of ν results in a different, but still isomorphic field that we can use.

Each of these fields has several generators. Any generator can be used to build a transformation matrix that preserves the additive homomorphism. However, in order to preserve the multiplicative homomorphism, the selected generator must also be a root of the $GF(2^8)$ irreducible polynomial $q(x) = x^8 + x^4 + x^3 + x + 1$, as evaluated under the arithmetic of the composite field $GF\left((2^4)^2\right)$ [Paa94, RDJ+01]. Any generator that preserves additive and multiplicative homomorphisms can be used to build a transformation matrix between the AES $GF(2^8)$ field and the composite field $GF((2^4)^2)$.

Once such a generator (ϕ_2) is found, the transformation matrix $\mathbf{X}^{-1}$ should fulfill

$$\phi_2^i = \mathbf{X}^{-1} \times \phi_1^i, i \in [0, 254].$$

After the input transformation matrix is computed, the output transformation matrix can be found by switching the roles of ϕ_1 and ϕ_2 in the equations above, or by computing the inverse of the input matrix.

Note that if ϕ_2 is found, its conjugate elements $\phi_2^{2^k}, k \in [1, 7]$ can also be used to build other plausible transformation matrices. However, under the composite field arithmetic, if $\sigma = \sigma_h\gamma^{16} + \sigma_l\gamma$, then $\sigma^{16} = \sigma_l\gamma^{16} + \sigma_h\gamma$, where the high part is simply exchanged with the lower part. Hence, the higher and lower four rows of the transformation matrices generated using ϕ_2^{16} will be identical to the lower and higher four rows of matrices generated using ϕ_2, respectively. This means that the circuits used to implement these two cases will be identical, only the output notations will be different. Therefore, whenever a valid generator is found, we study its input and output transformation matrices along with the matrices of the first three conjugate generators $\phi_2^{2^k}$, $k \in [1, 3]$. Hence, there are a total of 8 (values

of ν) $\times 4$ (generators per field) = 32 unique transformation matrices that could be used. Then, logic-minimization algorithms should be used to compare the hardware cost in terms of number of XOR gates of these 32 cases in order to select the smallest one.

4.2 Generating Extended Transformation Matrices

Instead of trying to solve the original 8×8 matrices, we follow the route used by Boyar and Peralta [BP10] and add some linear equations from the composite field inverter circuit into the input and output transformation matrices. This gives more options to consider for the logic-minimization algorithm and improves the overall design. However, instead of collecting the linear gates from the implemented circuit, as used by [BP10], we select the candidate linear equations at the mathematical level. This allows us to run the logic-minimization algorithm over the extended matrices of all the 32 considered cases, without having to design the inner circuit at different values of ν.

Since we use the multiplication formulations presented in Lemma 1, one can add the linear equations for the computation of $A_{jk} = \{a_{01}, a_{02}, a_{03}, a_{12}, a_{13}, a_{23}\}$, and $B_{jk} = \{b_{01}, b_{02}, b_{03}, b_{12}, b_{13}, b_{23}\}$ into the input transformation matrix $\mathbf{X}^{-1}$, resulting in what we denote as the $\mathbf{T}_{in}$ matrix. Hence, we include 12 extra signals in the logic-minimization problem to compute A_{jk} and B_{jk}. This is an interesting choice as the A_{jk} and B_{jk} signals are also used for the output multipliers, as detailed in Sec. 5.3. Other plausible choices are discussed in Sec. 6.2 and Sec. 6.3.

Note that the generic mathematical formulations and the corresponding S-box architecture to generate these 20 bits directly at the input transformation matrix are proposed for the first time in this paper.

In other words, the $\mathbf{T}_{in}$ of interest computes the 20 bits of:

$$\mathbf{g}_{20} = [a_0a_1a_2a_3b_0b_1b_2b_3a_{01}a_{02}a_{03}a_{12}a_{13}a_{23}b_{01}b_{02}b_{03}b_{12}b_{13}b_{23}]^{tr}$$

from the S-box input vector $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$ as follows

$$\mathbf{g}_{20} = \mathbf{T}_{in} \times \mathbf{g}, \tag{10}$$

where $\mathbf{T}_{in}$ is a 20×8 binary input transformation matrix whose entries are either 0 or 1 depending on the chosen composite field. In fact, the first 8 rows of $\mathbf{T}_{in}$ consists of $\mathbf{X}^{-1}$. Let us denote the first and the last 4 rows of $\mathbf{X}^{-1}$ as $\mathbf{a}_i^{tr}$ and $\mathbf{b}_i^{tr}$, $0 \le i \le 3$, respectively. The rows of $\mathbf{X}^{-1}$ generate the coordinates of A and B as $a_i = \mathbf{a}_i^{tr}\mathbf{g}$ and $b_i = \mathbf{b}_i^{tr}\mathbf{g}$, respectively. The next 12 rows of $\mathbf{T}_{in}$ generate $a_{ij} = a_i \oplus a_j = (\mathbf{a}_i^{tr} \oplus \mathbf{a}_j^{tr})\mathbf{g} = \mathbf{a}_{ij}^{tr}\mathbf{g}$ and $b_{ij} = b_i \oplus b_j = (\mathbf{b}_i^{tr} \oplus \mathbf{b}_j^{tr})\mathbf{g} = \mathbf{b}_{ij}^{tr}\mathbf{g}$.

Therefore, one can write $\mathbf{T}_{in}$ as

$$\mathbf{T}_{in} = \begin{bmatrix} \mathbf{X}^{-1} \\ \mathbf{a}_{ij}^{tr} \\ \mathbf{b}_{ij}^{tr} \end{bmatrix} \text{ for } 0 \le i, j \le 3, i \ne j. \tag{11}$$

Similarly, at the output, instead of trying to solve the original 8×8 output transformation matrix $\mathbf{MX}$, we include the equations that are required to convert from the RNB back to the NB (converting the 5 bits of (9) back to 4 bits) into $\mathbf{MX}$, resulting in what we denote as the $\mathbf{T}_{out}$ matrix. Hence, the output of the S-box $\mathbf{s}$ is

$$\mathbf{s} = \mathbf{T}_{out} \times \mathbf{f}_{10} \oplus \mathbf{h}, \tag{12}$$

where $\mathbf{f}_{10} = [w_0w_1w_2w_3w_4z_0z_1z_2z_3z_4]^{tr}$ is the output of composite field inversion, represented in the RNB and $\mathbf{h} = [01100011]^{tr}$ as presented in (1).

Here, we are interested in the logic-minimization of

$$\mathbf{T}_{out} = \mathbf{MX} \times \mathbf{T1}, \tag{13}$$

where **T1** is the 8×10 matrix that converts from the RNB representations of $W = \sum_{i=0}^{3} w_i \beta^{2^i} + w_4$ and $Z = \sum_{i=0}^{3} z_i \beta^{2^i} + z_4$ at the outputs of the $GF((2^4)^2)$ inverter in Figure 2 back to the non-redundant NB representations, i.e.,

$$\begin{bmatrix} w_0 \oplus w_4 \\ w_1 \oplus w_4 \\ w_2 \oplus w_4 \\ w_3 \oplus w_4 \\ z_0 \oplus z_4 \\ z_1 \oplus z_4 \\ z_2 \oplus z_4 \\ z_3 \oplus z_4 \end{bmatrix} = \mathbf{T1} \times \mathbf{f}_{10} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}. \tag{14}$$

$\mathbf{MX} = \mathbf{M} \times \mathbf{X}$ is the 8×8 output transformation matrix, in which $\mathbf{M}$ is the S-box affine transformation as presented in (1) and $\mathbf{X}$ is the inverse of the input transformation matrix $\mathbf{X}^{-1}$.

$\mathbf{T}_{out}$ is an 8×10 binary matrix whose entries are either 0 or 1 based on the chosen ν.

The next subsection reviews some of the known logic-minimization algorithms that could be used to implement these transformation matrices.

4.3 Logic-Minimization Algorithms

The problem of finding the smallest number of gates that is required to implement a transformation matrix is known as the Shortest Linear Program (SLP) problem. The SLP problem is defined as: Given a set of linear target equations E over a specific field $\mathbb{F}$, find the shortest linear program to compute E. Our interest is to find the shortest linear program that can compute $\mathbb{A} = \mathbf{Tr} \times \mathbb{B}$, where $\mathbb{A}$ and $\mathbb{B}$ are elements in two different isomorphic fields and **Tr** is a generic transformation matrix. This problem is known to be NP-hard [BMP08, BMP13], and the best solution could only be found by an exhaustive search over all possible circuits, which quickly becomes infeasible at large input matrices.

4.3.1 Previous Work

Paar proposed a heuristic that works by iteratively selecting the XOR gate which is the most common in the target equations [Paa94]. The matrix is then appended by adding one column that represents this gate, and the target equations are updated accordingly. This gate-selection and matrix-updating continues until there is a single '1' in each row of the matrix.

Canright used an exhaustive search over all possible solutions [Can05b], as the target matrices were relatively small (8×8). In order to manage this time-consuming search, he searched within a small subset of the tower fields that he studied (only 27 cases out of the 432 fields). The smallest circuit that was found required 13 XOR gates at the input matrix $\mathbf{X}^{-1}$ and 11 XOR gates at the output matrix **MX** of the AES S-box.

Boyar and Peralta showed that the algorithms used by both Canright and Paar are cancellation-free, where the selected gates must always result in an output with a Hamming weight that is the addition of the Hamming weights of the inputs [BP10]. Hence, in these solutions, XOR gates are never used to cancel-out common terms. This means that the exhaustive search conducted by Canright did not, in principle, exhaust all the actually possible solutions. Hence, they proposed a new heuristic algorithm that is not cancellation-free. We denote this algorithm as the *Normal-BP* heuristic, referring to the author names. More details about this algorithm is discussed in the next clause.

Using the *Normal-BP* heuristic, Boyar and Peralta did not find any smaller implementation for the matrices used by Canright. As a work around, and being supported by fast searching algorithms, they included all the linear equations in the inner circuit of Canright S-box (parts of the tower field inversion circuit) that directly followed the input matrix $\mathbf{X}^{-1}$ and those that proceeded the output matrix $\mathbf{MX}$ into the input and output transformation matrices, respectively. The new extended-input matrix became 22×8 and the new extended-output matrix became 8×18. These new matrices are computationally intractable for the exhaustive search algorithms, and interesting targets for the fast heuristic-based algorithms. The extended-input matrix directly computes 22 equations at a cost of 23 XOR gates. For comparison, the original Canright circuit would need 27 gates to solve these equations (13 for the 8×8 input matrix $\mathbf{X}^{-1}$ followed by 14 extra gates to solve the 14 extra equations). Similarly, the extended-output matrix required 30 XOR gates.

Visconti et al. proposed a tweaked algorithm that sometimes works better for dense matrices [VSP17]. It depends on computing the common path of the target matrix using its boolean complement, which has a lower density, before applying the regular *Normal-BP* heuristic algorithm.

4.3.2 *Normal-BP* [BP10]

Here, we review the *Normal-BP* heuristic before proposing several improvements. *Normal-BP* heuristic makes use of a distance function δ, which computes the number of XOR operations that are required to reach from a known set of signals (called the 'base' $\mathbf{S}$) to an output target (one row in the generic $\mathbf{Tr}$ matrix). The function δ is computed using exhaustive search within all signals of the base $\mathbf{S}$. The algorithm maintains a vector $Dist$ of the distance δ from the known base $\mathbf{S}$ to each of the output equations (each row of the $\mathbf{Tr}$ matrix). Assuming the size of $\mathbf{Tr}$ matrix is $r \times c$, the number of input signals is c, while the number of target equations is r which is equivalent to the initial size of $Dist$.

In the beginning, the base set $\mathbf{S}$ includes only the input signals, $S_i, i \in [0, c-1]$, where S_i is one input signal in the base $\mathbf{S}$. The $Dist$ vector is initialized to the Hamming weight of each row minus one, which is indeed the required number of gates if there was no sharing of gates. Sharing here means that one gate is used for generating two or more different equations. Then, in each iteration, the algorithm prepares the candidate next gate by XORing every two signals S_i and S_j from the known base $\mathbf{S}$, with $i, j \in [0, c-1]$ and $i \neq j$. One of these gates is selected such that if it was added to the base, the new base would minimize the sum of $Dist$. If there is a tie between two or more candidate gates, the program selects the gate that maximizes the Euclidean norm of $Dist$ ($\sqrt{\sum Dist_i^2}$ for $i \in [0, r_d - 1]$), with r_d being the size of $Dist$. If there is a further tie after this selection criteria, the program just selects the first gate in the set of ties. The authors studied different tie breaking methodologies including: Norm-largest (maximizing the square of the Euclidean norm minus the largest distance), Norm-diff (maximizing the square of the Euclidean norm minus the difference of the largest two distances) and Random selection (try three gates and select the best one). They reported that the tie-breaking methodologies perform fairly similar.

The algorithm makes use of 'pre-emptive' choices, which allows directly picking new gates as follows: if any two signals in the base S_i, S_j are such that $S_i \oplus S_j$ is a target row in $\mathbf{Tr}$, then the program directly picks this gate as the new base element.

In the next clause, we propose an improved logic-minimization algorithm that we denote as *Improved-BP*. We also propose two new logic-minimization methodologies denoted as *Shortest-Dist-First* and *Focused-Search* algorithms.

4.3.3 *Improved-BP* (Proposed)

Our first algorithm (*Improved-BP*) improves the *Normal-BP* algorithm in three aspects:

1- In each iteration, *Normal-BP* adds a candidate gate to the base **S** and evaluates the new distance *Dist*, regardless if this new gate leads to 'pre-emptive' gates in the next iteration or not. In fact, if adding a candidate gate would result in 'pre-emptive' gates, these gates will be automatically added in the next iteration. Hence, whenever a candidate gate leads to 'pre-emptive' gates, these gates should also be added to the base **S** (along with the candidate gate) before evaluating the *Dist* vector.

In the proposed *Improved-BP* algorithm, after adding a candidate gate, we check the *Dist* entries. If any of the *Dist* entries has been changed from 2 to 1, which means that one target will be found using a 'pre-emptive' gate in the next iteration, we add all the targets that will be resolved with 'pre-emptive' gates to the base **S** and re-evaluate the *Dist* vector.

2- The tie-breaking methodologies in the *Normal-BP* algorithm may not lead to the best results. In fact, the authors reported that the random selection (best of three random trials) slightly outperforms all the other tie-breaking methodologies. Meanwhile, it is computationally intensive to exhaustively test all the ties in different algorithm-runs. In the proposed *Improved-BP* algorithm, if there is a tie in one iteration, we test the results of all the ties in the next iteration and keep only the best sequence of gates. In this way, we perform only one algorithm-run while monitoring only the best set of gates.

3- We implemented the *Improved-BP* algorithm while monitoring the delay of each signal in the base **S**. In the beginning, the input signals in the base **S** are associated with a delay vector, denoted **Delay**, of all zeros. Adding a new gate $S_i \oplus S_j$ to the base, its delay is evaluated as $max(Delay_i, Delay_j) + 1$, where $Delay_i$ is the delay of S_i. This improvement helps in two situations. First, the delay of each target is readily available at the output, which is an important factor in designing the following circuits, as we detail in Sec. 5.1. Second, we can find the best circuit at a specified maximum delay. Assuming that the maximum allowed delay is D_{max}, if the delay of any signal in the base **S** reaches D_{max}, this signal gets removed from the base and no longer used to build any further target. For this delay-controlled option to work properly, we also modified the δ function to report the distance, as a number of gates, only if the delay associated with adding these gates together will not exceed the D_{max}. The delay associated with the addition of several signals is evaluated using the binary tree method where, iteratively, the input signals are sorted and the two signals with the least delays are added together. Otherwise, the δ function returns infinity, reporting that there is no path to the target function under the specified maximum delay.

4.3.4 *Shortest-Dist-First* (Proposed)

The *Shortest-Dist-First* algorithm works similarly to the *Improved-BP*, however, the methodology of selecting the next gate is different. The *Shortest-Dist-First* algorithm selects the gate that maximizes the number $Z = sum(Dist_i = j)$ at $i \in [0, r_d - 1]$, with r_d being the size of *Dist*, for the smallest value of $j > 0$ such that $Z > 0$.

In other words, the algorithm selects the gate that results in as many 'pre-emptive' gates (with $Dist_i = 1$ for $i \in [0, r_d - 1]$) as possible. If no such gate was found ($Z = 0$ at $j = 1$), the algorithm searches for the gate that results in as many $Dist_i = 2$ (at $j = 2$) as possible, and so on. Hence, the algorithm promotes the gates that lead to the *shortest distance*.

4.3.5 *Focused-Search* (Proposed)

This algorithm is similar to the *Shortest-Dist-First* algorithm, however, it tests *all* the gates that result in $Z = sum(Dist_i = j)$, with $i \in [0, r_d - 1]$, with r_d being the size of

$Dist$, for the smallest value of $j > 0$, being greater than zero. In other words, it tests all the gates that lead to $Dist_i = 1$ for $i \in [0, r_d - 1]$. Only if no gates fulfills this criteria, the algorithm tests all the gates that lead to $Dist_i = 2$, and so on.

Another difference is that since this algorithm tests all the candidate gates anyway, we do not re-evaluate the $Dist$ vector whenever 'pre-emptive' gates are found.

This algorithm is the slowest of all. It slightly moves toward being an exhaustive search.

4.4 Searching for Optimum Transformation Matrices

For each of the 32 unique transformation matrices, we applied the logic minimization algorithms in order to find the set of matrices that results in the minimum overall gate-count. We solved for the $\mathbf{X}^{-1}$, $\mathbf{T}_{in}$, and $\mathbf{T}_{out}$. The results are highlighted in Table 1.

Table 1 serves two main goals. First, it helps in comparing the proposed logic-minimization algorithms against the current ones. Second, it helps in selecting the best set of transformation matrices.

Comparing logic-minimization algorithms: First, the table examines the proposed logic-minimization algorithms against the current state-of-the-art (*Normal-BP* [BP10]) in 96 different test matrices, ranging in size between 8×8, 20×8 and 8×10. The improvement in analyzing the 8×8 $\mathbf{X}^{-1}$ matrices were observable in 8 out of the 32 cases. Our algorithms outperformed the previous ones by one and sometimes two gates (7% and 13% respectively). Similarly, analyzing the 20×8 $\mathbf{T}_{in}$ matrices were improved in 12 cases. In addition, the 8×10 $\mathbf{T}_{out}$ matrices were improved in 28 cases, by sometimes over 3 gates (15%).

The MATLAB® computation time in seconds, as measured on a workstation with Intel Xeon CPU (6 cores at 3.2 GHz) equipped with 16 GB of RAM, is shown in Table 2. The *Improved-BP* and *Shortest-Dist-First* algorithms require higher computation time than the *Normal-BP* algorithm because they test all the gates involved in a tie. Counter-intuitively, the *Focused-Search* algorithm, which is supposed to be the slowest algorithm, shows the fastest performance with the 20×8 matrices, with a slim standard deviation. The reason is that the search space for the 20×8 is not large, while the algorithm is relieved from re-evaluating the $Dist$ vector after adding the 'pre-emptive' gates.

The optimum transformation matrices: Table 1 shows that the matrix $\mathbf{T}_{in}$, which is used in the input transformation block of Figure 2, always results in smaller circuits, as compared to the number of gates required for $\mathbf{X}^{-1}$ plus 12 gates. It is noted that these 12 gates are required to compute the 12 extra equations of A_{jk} and B_{jk} in Figure 2 if they were removed from the $\mathbf{T}_{in}$. Solving $\mathbf{T}_{in}$ was better by 3 to 5 gates.

Table 1 also shows that four of out of 32 possibilities require the optimum value of 35 XOR gates for both the input (19 XOR gates) and the output (16 XOR gates) transformation blocks. Among the four optimum results, we chose the one corresponding to $\nu = \beta = (1000)$. This particular selection will further reduce the complexity of the exponentiation computation block as explained in the next section.

4.5 The Used Transformation Matrices

Throughout the rest of this paper, we adopt $\nu = \beta = (1000)$ and $Gen = (1 + \beta^{2^2})\gamma + (1 + \beta^2)\gamma^{16} = (11011011) = (\mathrm{DB})_h$ so that the input ($\mathbf{X}^{-1}$) and output ($\mathbf{MX}$) transformation matrices are as follows.

Table 1: Logic minimization of the input matrix $\mathbf{X}^{-1}$, the extended-input matrix $\mathbf{T}_{in}$ and the extended-output $\mathbf{T}_{out}$ transformation matrices.

ν=	*Gen* #1	*Gen* #2	*Gen* #3	*Gen* #4
1000	[a]$Gen = (03)_h$ [b]13/13/13/- [c]22/21/21/- [e]39 [d]19/18/18/-	$Gen = (6F)_h$ 13/13/13/- 20/20/20/20 36 16/16/16/16	$Gen = (6A)_h$ 14/14/14/- 21/21/21/- 39 19/19/18/-	$Gen = (DB)_h$ 11/11/11/- 19/19/19/**19** **35** 19/17/17/**16**
0100	$Gen = (06)_h$ 13/13/13/- 22/21/21/- 39 19/18/18/-	$Gen = (74)_h$ 13/13/13/- 20/20/20/20 36 16/16/16/16	$Gen = (5C)_h$ 14/14/14/- 21/21/21/- 39 19/19/18/-	$Gen = (95)_h$ 11/11/11/- 19/19/19/**19** **35** 19/17/17/**16**
0010	$Gen = (03)_h$ 13/13/13/- 22/21/21/- 39 19/18/18/-	$Gen = (B2)_h$ 13/13/13/- 20/20/20/20 36 16/16/16/16	$Gen = (A6)_h$ 14/14/14/- 21/21/21/- 39 19/19/18/-	$Gen = (CA)_h$ 11/11/11/- 19/19/19/**19** **35** 19/17/17/**16**
0001	$Gen = (06)_h$ 13/13/13/- 22/21/21/- 39 19/18/18/-	$Gen = (CF)_h$ 13/13/13/- 20/20/20/20 36 16/16/16/16	$Gen = (C5)_h$ 14/14/14/- 21/21/21/- 39 19/19/18/-	$Gen = (B7)_h$ 11/11/11/- 19/19/19/**19** **35** 19/17/17/**16**
0111	$Gen = (12)_h$ 14/14/14/- 22/22/22/- 39 19/18/17/-	$Gen = (7E)_h$ 13/13/13/- 22/21/21/- 38 18/18/17/-	$Gen = (2E)_h$ 15/13/13/- 21/21/21/- 39 19/19/18/-	$Gen = (9F)_h$ 14/14/13/- 21/20/20/20 36 18/17/16/16
1011	$Gen = (17)_h$ 15/13/13/- 21/21/21/- 39 19/19/18/-	$Gen = (CF)_h$ 14/14/13/- 21/20/20/20 36 18/17/16/16	$Gen = (18)_h$ 14/14/14/- 22/22/22/- 39 19/18/17/-	$Gen = (7B)_h$ 13/13/13/- 22/21/21/- 38 18/18/17/-
1101	$Gen = (12)_h$ 14/14/14/- 22/22/22/- 39 19/18/17/-	$Gen = (A3)_h$ 13/13/13/- 22/21/21/- 38 18/18/17/-	$Gen = (E2)_h$ 15/13/13/- 21/21/21/- 39 19/19/18/-	$Gen = (8E)_h$ 14/14/13/- 21/20/20/20 36 18/17/16/16
1110	$Gen = (17)_h$ 15/13/13/- 21/21/21/- 39 19/19/18/-	$Gen = (74)_h$ 14/14/13/- 21/20/20/20 36 18/17/16/16	$Gen = (81)_h$ 14/14/14/- 22/22/22/- 39 19/18/17/-	$Gen = (59)_h$ 13/13/13/- 22/21/21/- 38 18/18/17/-

Each cell contains the following:
[a]- $Gen =$ is the hexadecimal of the used generator.
[b]- Gate-count of the input $\mathbf{X}^{-1}$ matrix,
[c]- Gate-count of the extended input $\mathbf{T}_{in}$ matrix in Figure 2,
[d]- Gate-count of the extended output $\mathbf{T}_{out}$ matrix in Figure 2,
[bcd] are minimized using Normal-BP [BP10]/ Improved-BP (Proposed) / Shortest-Dist-First (Proposed) / Focused-Search (Proposed).
[e]- The best overall gate count is found by adding the smaller gate-count of $(\mathbf{X}^{-1}) + 12$ or $(\mathbf{T}_{in})$ to the gate-count of $\mathbf{T}_{out}$.
The best case is highlighted in **bold**. The result '-' means that we did not attempt to solve this case.

Table 2: The mean and standard deviation of computation time (in seconds) for the Normal-BP [BP10] against the three proposed logic-minimization algorithms.

Target Matrix	$\mathbf{X}^{-1}(8\times 8)$	$\mathbf{T}_{in}(20\times 8)$	$\mathbf{T}_{out}(8\times 10)$
Normal-BP [BP10]	(0.215, 0.066)	(0.312, 0.115)	(2.188, 0.797)
Improved-BP	(5.664, 12.096)	(0.401, 0.2)	(820.587, 1.776e+03)
Shortest-Dist-First	(7.05, 6.99)	(0.567, 0.438)	(636.671, 1.119e+03)
Focused-Search	-	(0.289, 0.062)	(9.563e+03, 1.459e+04)

$$\mathbf{X}^{-1}=\begin{bmatrix}0&0&0&0&1&1&0&1\\0&0&0&0&0&1&0&1\\1&1&1&0&0&0&1&1\\0&0&1&0&0&0&0&1\\0&1&1&1&1&1&0&1\\1&0&1&0&0&1&1&1\\0&1&0&0&1&1&0&1\\1&1&1&1&1&1&0&1\end{bmatrix},\quad \mathbf{MX}=\begin{bmatrix}0&1&0&0&0&0&0&1\\1&1&1&0&1&1&1&0\\1&1&0&1&0&1&1&1\\1&1&0&0&0&1&1&1\\0&0&1&1&0&0&1&0\\0&1&1&1&0&0&0&0\\0&0&1&0&0&0&0&0\\0&1&1&0&0&1&1&1\end{bmatrix}.$$

Therefore, the extended-input ($\mathbf{T}_{in}$) and extended-output ($\mathbf{T}_{out}$) transformation matrices can be found using (11) and (13) as follows.

$$\mathbf{g}_{20}=\begin{bmatrix}a_0\\a_1\\a_2\\a_3\\b_0\\b_1\\b_2\\b_3\\a_{01}\\a_{02}\\a_{03}\\a_{12}\\a_{13}\\a_{23}\\b_{01}\\b_{02}\\b_{03}\\b_{12}\\b_{13}\\b_{23}\end{bmatrix}=\mathbf{T}_{in}\times\mathbf{g}=\begin{bmatrix}0&0&0&0&1&1&0&1\\0&0&0&0&0&1&0&1\\1&1&1&0&0&0&1&1\\0&0&1&0&0&0&0&1\\0&1&1&1&1&1&0&1\\1&0&1&0&0&1&1&1\\0&1&0&0&1&1&0&1\\1&1&1&1&1&1&0&1\\0&0&0&0&1&0&0&0\\1&1&1&0&1&1&1&0\\0&0&1&0&1&1&0&0\\1&1&1&0&0&1&1&0\\0&0&1&0&0&1&0&0\\1&1&0&0&0&0&1&0\\1&1&0&1&1&0&1&0\\0&0&1&1&0&0&0&0\\1&0&0&0&0&0&0&0\\1&1&1&0&1&0&1&0\\0&1&0&1&1&0&1&0\\1&0&1&1&0&0&0&0\end{bmatrix}\begin{bmatrix}g_7\\g_6\\g_5\\g_4\\g_3\\g_2\\g_1\\g_0\end{bmatrix},\tag{15}$$

$$\mathbf{s}=\begin{bmatrix}s_7\\s_6\\s_5\\s_4\\s_3\\s_2\\s_1\\s_0\end{bmatrix}=(\mathbf{T}_{out}\times\mathbf{f}_{10})\oplus\mathbf{h}=\begin{bmatrix}0&1&0&0&1&0&0&0&1&1\\1&1&1&0&1&1&1&1&0&1\\1&1&0&1&1&0&1&1&1&1\\1&1&0&0&0&0&1&1&1&1\\0&0&1&1&0&0&0&1&0&1\\0&1&1&1&1&0&0&0&0&0\\0&0&1&0&1&0&0&0&0&0\\0&1&1&0&0&0&1&1&1&1\end{bmatrix}\begin{bmatrix}w_0\\w_1\\w_2\\w_3\\w_4\\z_0\\z_1\\z_2\\z_3\\z_4\end{bmatrix}\oplus\begin{bmatrix}0\\1\\1\\0\\0\\0\\1\\1\end{bmatrix}.\tag{16}$$

In the following, we propose two sets of circuits to implement these matrices, one for lightweight applications, and one for fast applications.

4.6 Lightweight Implementations

For the lightweight implementations, our first goal is to minimize the implementation area, expressed as the number of used XOR2 (2-input XOR) gates for the transformation matrices. Whenever two circuits are equivalent in the area, we select the circuit that results in the lower overall delay.

The lightweight implementation of $\mathbf{T}_{in}$ in (15) for the input transformation block is presented in Table 3(a). These formulations require 19 XOR2 gates (denoted by $\oplus$ in the table). The propagation delay, in terms of number of the XOR2 gate delay (D_X), for all signals are presented (in parentheses). Therefore, the longest delay from the inputs to the outputs and hence the propagation delay of the input transformation block is $5D_X$.

The lightweight implementation of the $\mathbf{T}_{out}$ in (16) for the output transformation block requires 16 gates with a delay of 6 gates ($6D_X$) as presented in Table 3(b). The computations presented in Table 3(b) perform the operation of $(\mathbf{T}_{out} \times \mathbf{f}_{10}) \oplus \mathbf{h}$ altogether, where XNOR2 gates are sometimes incorporated instead of XOR2 gates to include the effect of adding the constant $\mathbf{h}$. XNOR2 gates are denoted by $\odot$ in the table. Note that the use of XNOR2 gates does not exactly align with the non-zero entries of $\mathbf{h}$, as the outputs s_i are reused in other equations. We coded these equations in VHDL and verified the output with Modelsim® against the legitimate output of the AES S-box.

Table 3: Lightweight implementation of transformations in Figure 2: (a) Computation of $\mathbf{g}_{20} = \mathbf{T}_{in} \times \mathbf{g}$ in (15) for the input transformation. (b) Computation of $\mathbf{s} = (\mathbf{T}_{out} \times \mathbf{f}_{10}) \oplus \mathbf{h}$ in (16) for the output transformation.

$a_0 = a_1 \oplus g_3$ $(2D_X)$	$a_1 = g_2 \oplus g_0$ $(1D_X)$	$a_2 = a_3 \oplus a_{23}$ $(3D_X)$
$a_3 = g_5 \oplus g_0$ $(1D_X)$	$b_0 = b_{02} \oplus b_2$ $(4D_X)$	$b_1 = b_2 \oplus b_{12}$ $(5D_X)$
$b_2 = a_0 \oplus g_6$ $(3D_X)$	$b_3 = b_2 \oplus b_{23}$ $(4D_X)$	$a_{01} = g_3$ $(0D_X)$
$a_{02} = a_{03} \oplus a_{23}$ $(3D_X)$	$a_{03} = a_{13} \oplus g_3$ $(2D_X)$	$a_{12} = a_{13} \oplus a_{23}$ $(3D_X)$
$a_{13} = g_5 \oplus g_2$ $(1D_X)$	$a_{23} = u_0 \oplus g_1$ $(2D_X)$	$b_{01} = b_{02} \oplus b_{12}$ $(5D_X)$
$b_{02} = g_5 \oplus g_4$ $(1D_X)$	$b_{03} = g_7$ $(0D_X)$	$b_{12} = a_{02} \oplus g_2$ $(4D_X)$
$b_{13} = b_{23} \oplus b_{12}$ $(5D_X)$	$b_{23} = b_{02} \oplus g_7$ $(2D_X)$	$u_0 = g_7 \oplus g_6$ $(1D_X)$

(a)

$s_7 = t_1 \oplus t_2$ $(2D_X)$	$s_6 = t_7 \oplus t_6$ $(6D_X)$	$s_5 = t_0 \odot t_6$ $(6D_X)$
$s_4 = t_6 \odot w_2$ $(6D_X)$	$s_3 = t_4 \oplus z_3$ $(5D_X)$	$s_2 = t_0 \odot w_1$ $(3D_X)$
$s_1 = w_2 \odot w_4$ $(1D_X)$	$s_0 = t_5 \oplus t_3$ $(4D_X)$	$t_0 = s_1 \oplus w_3$ $(2D_X)$
$t_1 = w_1 \oplus z_4$ $(1D_X)$	$t_2 = w_4 \oplus z_3$ $(1D_X)$	$t_3 = s_7 \oplus z_2$ $(3D_X)$
$t_4 = s_2 \oplus t_3$ $(4D_X)$	$t_5 = s_1 \oplus z_1$ $(2D_X)$	$t_6 = s_0 \oplus w_0$ $(5D_X)$
$t_7 = t_2 \oplus z_0$ $(2D_X)$		

(b)

4.7 Fast Implementations

Our design goal for the fast implementations is to find a circuit with the smallest critical path delay. Note that the theoretically minimum critical path delay of the transformation matrices can be known upfront by computing $max(\lceil log_2(HW(\mathbf{Tr}_i)) \rceil)$, for $i \in [0, r-1]$ with HW denoting the Hamming weight function, and $\mathbf{Tr}_i$ is row number i of a generic transformation matrix ($\mathbf{T}_{in}$ or $\mathbf{T}_{out}$), and r is the number of rows in the $\mathbf{Tr}$ matrix ($r = 20$ for $\mathbf{T}_{in}$ and $r = 8$ for $\mathbf{T}_{out}$). The theoretically minimum critical path delay of both $\mathbf{T}_{in}$ and $\mathbf{T}_{out}$ is $3D_X$, as the maximum number of non-zero entries in $\mathbf{T}_{in}$ and $\mathbf{T}_{out}$ is 8.

We used the proposed logic-minimization algorithms while setting the maximum allowed delay D_{max} to $3D_X$. Therefore, the target of the logic-minimization is to find a plausible circuit under this delay constraint.

The corresponding formulations for fast implementation are presented in Table 4. For the input transformation block $\mathbf{T}_{in}$, the presented implementation in Table 4(a) requires 24 XOR2 gates with a propagation delay of 3 XOR2 gates (as required). Table 4(b) shows the formulations to compute the output transformation block $\mathbf{T}_{out}$ in (16) using 21 gates at a delay of $3D_X$ (as required).

Similarly, some XOR2 gates have been replaced with XNOR2 gates in order to incorporate the effect of adding the constant $\mathbf{h}$. The correctness of these formulations were checked by our codes as will be explained later.

Table 4: Fast implementation of transformations in Figure 2: (a) Computation of $\mathbf{g}_{20} = \mathbf{T}_{in} \times \mathbf{g}$ in (15) for the input transformation. (b) Computation of $\mathbf{s} = (\mathbf{T}_{out} \times \mathbf{f}_{10}) \oplus \mathbf{h}$ in (16) for the output transformation.

$a_0 = a_1 \oplus g_3$	$(2D_X)$	$a_1 = g_2 \oplus g_0$	$(1D_X)$	$a_2 = a_3 \oplus a_{23}$	$(3D_X)$
$a_3 = g_5 \oplus g_0$	$(1D_X)$	$b_0 = u_2 \oplus u_4$	$(3D_X)$	$b_1 = a_1 \oplus u_1$	$(3D_X)$
$b_2 = a_0 \oplus g_6$	$(3D_X)$	$b_3 = u_5 \oplus u_4$	$(3D_X)$	$a_{01} = g_3$	$(0D_X)$
$a_{02} = a_{03} \oplus a_{23}$	$(3D_X)$	$a_{03} = a_{13} \oplus g_3$	$(2D_X)$	$a_{12} = a_{13} \oplus a_{23}$	$(3D_X)$
$a_{13} = g_5 \oplus g_2$	$(1D_X)$	$a_{23} = u_0 \oplus g_6$	$(2D_X)$	$b_{01} = u_3 \oplus u_5$	$(3D_X)$
$b_{02} = g_5 \oplus g_4$	$(1D_X)$	$b_{03} = g_7$	$(0D_X)$	$b_{12} = u_2 \oplus u_1$	$(3D_X)$
$b_{13} = u_2 \oplus u_3$	$(2D_X)$	$b_{23} = b_{02} \oplus g_7$	$(2D_X)$	$u_0 = g_7 \oplus g_1$	$(1D_X)$
$u_1 = u_0 \oplus g_5$	$(2D_X)$	$u_2 = g_6 \oplus g_3$	$(1D_X)$	$u_3 = g_4 \oplus g_1$	$(1D_X)$
$u_4 = a_1 \oplus b_{02}$	$(2D_X)$	$u_5 = u_2 \oplus g_7$	$(2D_X)$		

(a)

$s_7 = t_0 \oplus t_6$	$(2D_X)$	$s_6 = t_2 \oplus t_{10}$	$(3D_X)$	$s_5 = s_7 \odot t_8$	$(3D_X)$
$s_4 = t_3 \oplus t_2$	$(3D_X)$	$s_3 = t_{11} \oplus t_{12}$	$(2D_X)$	$s_2 = s_1 \odot t_4$	$(2D_X)$
$s_1 = w_2 \odot w_4$	$(1D_X)$	$s_0 = t_5 \odot t_2$	$(3D_X)$	$t_0 = w_1 \oplus z_4$	$(1D_X)$
$t_1 = z_1 \oplus z_2$	$(1D_X)$	$t_2 = t_0 \oplus t_1$	$(2D_X)$	$t_3 = w_0 \oplus z_3$	$(1D_X)$
$t_4 = w_1 \oplus w_3$	$(1D_X)$	$t_5 = w_2 \oplus z_3$	$(1D_X)$	$t_6 = w_4 \oplus z_3$	$(1D_X)$
$t_7 = w_0 \oplus w_3$	$(1D_X)$	$t_8 = t_1 \oplus t_7$	$(2D_X)$	$t_9 = w_0 \oplus z_0$	$(1D_X)$
$t_{10} = s_1 \oplus t_9$	$(2D_X)$	$t_{11} = w_2 \oplus w_3$	$(1D_X)$	$t_{12} = z_2 \oplus z_4$	$(1D_X)$

(b)

5 Proposed $GF((2^4)^2)$ Inversion

The $GF((2^4)^2)$ inversion is the core operation of the proposed S-box, as shown in Figure 2. The proposed $GF((2^4)^2)$ inversion consists of three main blocks, namely exponentiation computation, subfield inverter, and output multipliers. These are explained in this section.

5.1 Exponentiation Computation

Based on the proposed logic-minimization algorithms, we use $\nu = \beta = (1000)$ as discussed in Sec. 4.4. Thus, the exponentiation computation block in Figure 2 generates

$$D = g^{17} = AB + (A+B)^2\beta. \tag{17}$$

Let $A = \sum_{i=0}^{3} a_i \beta^{2^i} = (a_0 a_1 a_2 a_3) \in GF(2^4)$ and $B = \sum_{i=0}^{3} b_i \beta^{2^i} = (b_0 b_1 b_2 b_3) \in GF(2^4)$ be represented in the ONB-I $\{\beta,\ \beta^2,\ \beta^{2^2},\ \beta^{2^3}\}$. Then,

$$(A+B)^2\beta = \sum_{i=0}^{3} (a_{i-1} \oplus b_{i-1}) \beta^{2^i+1}. \tag{18}$$

Since $r(\beta) = 0$ in (3), one can find $\beta^5 = 1$ and so $\beta^3 = \beta^{2^3}$, $\beta^9 = \beta^{2^2}$ which simplify (18) to

$$(A+B)^2\beta = (a_3 \oplus b_3)\beta^2 + (a_2 \oplus b_2)\beta^{2^2} + (a_0 \oplus b_0)\beta^{2^3} + (a_1 \oplus b_1) \times 1. \quad (19)$$

Since $1 = (1111) \in GF(2^4)$, one can change the RNB representation of $(A+B)^2\beta$ in (19) to the corresponding NB as follows:

$$(A+B)^2\beta = (a_1 \oplus b_1)\beta + (a_{13} \oplus b_{13})\beta^2 + (a_{12} \oplus b_{12})\beta^{2^2} + (a_{01} \oplus b_{01})\beta^{2^3}. \quad (20)$$

Substituting (20) and the coordinates of $C = AB$ from Lemma 1 into (17), one obtains

$$\begin{aligned} D &= d_0\beta + d_1\beta^2 + d_2\beta^{2^2} + d_3\beta^{2^3} = c_4\beta + c_4\beta^2 + c_4\beta^{2^2} + c_4\beta^{2^3} + \\ &\underbrace{(c_0 \oplus a_1 \oplus b_1)}_{\tilde{d_0}}\ \beta + \underbrace{(c_1 \oplus a_{13} \oplus b_{13})}_{\tilde{d_1}}\ \beta^2 + \underbrace{(c_2 \oplus a_{12} \oplus b_{12})}_{\tilde{d_2}}\ \beta^{2^2} + \underbrace{(c_3 \oplus a_{01} \oplus b_{01})}_{\tilde{d_3}}\ \beta^{2^3}. \end{aligned} \quad (21)$$

Let us denote $\tilde{D} = \sum_{i=0}^{3} \tilde{d_i}\beta^{2^i} \in GF(2^4)$ such that $D = \tilde{D} + c_4$, where $\tilde{d_i} \in GF(2)$ for $0 \le i \le 3$, is defined in (21). From (21), one can obtain

$$d_i = c_4 \oplus \tilde{d_i},\ 0 \le i \le 3. \quad (22)$$

To simplify $\tilde{d_i}$ for $0 \le i \le 3$, one can substitute c_i, $0 \le i \le 3$, from (9) into (21) and simplify them as follows:

$$\begin{aligned} \tilde{d_0} &= (a_0b_0) \oplus (a_{12}b_{12}) \oplus a_1 \oplus b_1 \\ \tilde{d_1} &= (a_1 \vee b_1) \oplus a_3 \oplus b_3 \oplus (a_{23}b_{23}) \\ \tilde{d_2} &= (a_2 \vee b_2) \oplus a_1 \oplus b_1 \oplus (a_{03}b_{03}) \\ \tilde{d_3} &= (a_3b_3) \oplus (a_{01} \vee b_{01}). \end{aligned} \quad (23)$$

For the simplification of $\tilde{d_1}$ from (21) to (23), we use $a_{13} \oplus b_{13} = a_1 \oplus b_1 \oplus a_3 \oplus b_3$ and then $(a_1b_1 \oplus a_1 \oplus b_1) = (a_1 \vee b_1)$, where $\vee$ represents an OR operation. Similarly, for the simplification of $\tilde{d_2}$ from (21) to (23), we use $a_{12} \oplus b_{12} = a_2 \oplus b_2 \oplus a_1 \oplus b_1$ and then $(a_2b_2 \oplus a_2 \oplus b_2) = (a_2 \vee b_2)$. Similar property, i.e., $(a_{01}b_{01} \oplus a_{01} \oplus b_{01}) = (a_{01} \vee b_{01})$, is used for $\tilde{d_3}$.

Our goal is to minimize the chip area in the ASIC implementations. Therefore, we use NAND and NOR gates instead of AND and OR gates, respectively. This is because NAND and NOR gates have lower chip area and delay as compared with AND and OR gates, respectively [WH15]. The coordinate c_4 is needed in all four formulations for d_i, $0 \le i \le 3$, in (22) and so one can implement c_4 using two NAND gates and one XOR gate as $c_4 = a_{02}b_{02} \oplus a_{13}b_{13} = (a_{02}b_{02})' \oplus (a_{13}b_{13})'$, and share it for the four d_i's. As a result of replacing OR gates to NOR gates and AND gates to NAND gates, we may need to change XOR gates to XNOR in order to keep the same function at the outputs. Therefore, the following formulations can be obtained from c_4 and (23)

$$\begin{aligned} \tilde{d_0} &= (a_1 \odot b_1) \oplus (a_0b_0)' \odot (a_{12}b_{12})' \\ \tilde{d_1} &= (a_1 \vee b_1)' \oplus (a_3 \odot b_3) \odot (a_{23}b_{23})' \\ \tilde{d_2} &= (a_1 \odot b_1) \oplus (a_2 \vee b_2)' \odot (a_{03}b_{03})' \\ \tilde{d_3} &= (a_3b_3)' \oplus (a_{01} \vee b_{01})' \\ c_4 &= (a_{02}b_{02})' \oplus (a_{13}b_{13})', \end{aligned} \quad (24)$$

where an XNOR operation of two inputs (XNOR2) is denoted by $a_1 \odot b_1 = a_1 \oplus b_1 \oplus 1 = (a_1 \oplus b_1)'$. In (24), having two XNOR operations for $\tilde{d_0}$, $\tilde{d_1}$ and $\tilde{d_2}$ do not change their functions as $x \odot y \odot z = x \oplus y \oplus 1 \oplus z \oplus 1 = x \oplus y \oplus z$.

Note that each of the equations that generate $\tilde{d_0}$, $\tilde{d_1}$ and $\tilde{d_2}$ consists of mod-2 additions between three terms (using XOR and XNOR). The order of implementing these three

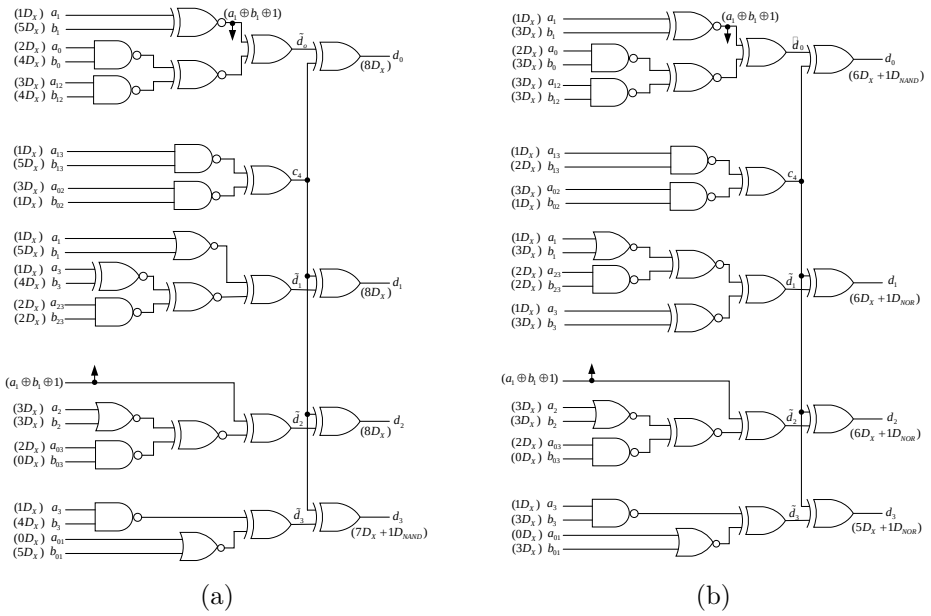

Figure 3: The proposed exponentiation computation block for the S-box architecture. (a) For the proposed lightweight S-box implementation (b) For the proposed fast S-box implementation. Note that the order of additions (XOR/XNOR) that generate d_1 is different between two circuits.

mod-2 additions affects the overall delay of the circuit. The proper order of additions should be selected based on the delay associated with the input terms, which eventually depends on the delay of the $\mathbf{T}_{in}$ block. Here, we use the delay associated with each signal of the $\mathbf{T}_{in}$ block, as detailed in Table 3 for the lightweight implementation and in Table 4 for the fast implementation.

Therefore, we propose two implementations of the exponentiation computation block, as shown in Figure 3, one for the lightweight implementation highlighted in Figure 3(a) and one for the fast implementation highlighted in Figure 3(b). Both implementations have the same space complexity. In each figure, we highlight the delay associated with all the input and output signals in parentheses.

Figure 3(a) shows that the optimum circuit for the lightweight implementation generates all the d_i output signals with a maximum total delay of $8D_X$, where D_X is the delay of one XOR2/XNOR2 gate. This result takes into account that the delay of XOR gate is slightly higher than the delay of a NAND gate. Similarly, Figure 3(b) shows that the highest delay of the fast implementation is $6D_X + 1D_{NOR}$, taking into account that the delay of NOR gate is slightly higher than NAND gate.

Based on Figure 3 and equations (22) and (24), one can find the space and time complexities of the proposed exponentiation computation blocks as follows. Note that the XNOR gate that generates $(a_1 \odot b_1)$ can be shared.

Proposition 1. *The exponentiation computation block consists of 14 XOR2/XNOR2 (2-input XOR/XNOR), 7 NAND2 (2-input NAND), and 3 NOR2 (2-input NOR) gates with the critical path delay of $4D_X$, where D_X is the delay of one XOR2/XNOR2 gate. This exponentiation block adds $3D_X$ and $3D_X + 1D_{NOR}$ to the critical path delay of the lightweight and fast input transformation matrices, respectively, where D_{NOR} is the delay of one NOR2 gate.*

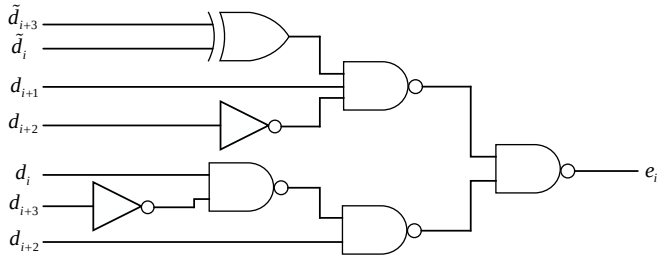

Figure 4: The proposed subfield inverter block over $GF(2^4)$ to generate e_i, $0 \le i \le 3$, for $E = \sum_{i=0}^{3} e_i \beta^{2^i} = D^{-1}$, where $D = \sum_{i=0}^{3} d_i \beta^{2^i}$ and the additions in the input indices are performed modulo 4. Note that $\tilde{d}_i$, $0 \le i \le 3$ are the signals from Figure 3 that generate $d_i = c_4 + \tilde{d}_i$.

5.2 Subfield Inverter

A large number of multiplicative inverse architectures over binary fields are available in the literature. In [WTS+85] and [Fen89], sequential architectures for computing inverse over $GF(2^m)$ are proposed. The inverse architectures proposed in those papers are costly as they require a NB multiplier and compute the inverse operation in a number of clock cycles. On the other hands, for a fast computation operation, one can compute the inverse as a bit-parallel architecture in composite fields [Paa94]. In [Can05b, Can05a, NNT+10], efficient subfield inverter over tower fields $GF((2^2)^2)$ are proposed. However, similar to the schemes proposed in [UHS+15, NNI12], our inverter is over $GF(2^4)$. The subfield inverter schemes presented in [UHS+15, NNI12] use the RNB and the polynomial ring representation (PRR) with 5-bit coordinates, whereas our inverter scheme uses the NB (with 4-bit coordinates).

The subfield inverter block of Figure 2 generates the inverse of its input $D = (d_0 d_1 d_2 d_3) = \sum_{i=0}^{3} d_i \beta^{2^i}$ over $GF(2^4)$. Let us denote $E = D^{-1} \in GF(2^4)$ as the output of this block. Let $e_i \in GF(2)$, be the ith, $0 \le i \le 3$, binary coordinate of E represented with respect to the ONB-I, i.e., $E = (e_0 e_1 e_2 e_3) = \sum_{i=0}^{3} e_i \beta^{2^i}$. Then, one can obtain e_i as follows.

Lemma 2.

$$e_i = d_{i+1} d'_{i+2} (d_i \oplus d_{i+3}) \vee d_{i+2} (d'_i \vee d_{i+3}), \quad 0 \le i \le 3, \tag{25}$$

where the additions in the indices are performed modulo 4.

The proof of Lemma 2 is provided in Appendix A.

In order to reduce the delay of (25), we use $(\tilde{d}_i \oplus \tilde{d}_{i+3})$ instead of $(d_i \oplus d_{i+3})$ as $d_i \oplus d_{i+3} = c_4 \oplus \tilde{d}_i \oplus c_4 \oplus \tilde{d}_{i+3} = (\tilde{d}_i \oplus \tilde{d}_{i+3})$. Also, in order to reduce the area and improve the speed of the ASIC implementations of (25), we use NAND and NOR gates instead of AND and OR gates, respectively. So, we use De Morgan's law for $(d'_i \vee d_{i+3}) = (d_i d'_{i+3})'$ and change two levels of AND-OR to its equivalent NAND-NAND implementation. As a result, we conclude the formulation presented in (25) to the following:

Corollary 1. *Let $D = (d_0 d_1 d_2 d_3) = \sum_{i=0}^{3} d_i \beta^{2^i}$ be the input of the subfield inverter. Then, the coordinates of the inverse output $E = (e_0 e_1 e_2 e_3) = D^{-1}$ can be found as*

$$e_i = ((d_{i+1} d'_{i+2} (\tilde{d}_i \oplus \tilde{d}_{i+3}))' ((d_i d'_{i+3})' d_{i+2})')', \quad 0 \le i \le 3, \tag{26}$$

where the additions in the indices are performed modulo 4.

Based on the architecture shown in Figure 4 for one e_i, the space and time complexities of the entire subfield inverter (e_i, $0 \leq i \leq 3$,) are as follows. Note that the four NOT gates can be shared among the four e_i outputs.

Proposition 2. *The space complexity of the proposed subfield inverter block over $GF(2^4)$ includes 12 NAND2, 4 NAND3, 4 XOR2 and 4 NOT gates, where NAND2, and NAND3 are, respectively, a 2-input and a 3-input NAND gates and XOR2 is a 2-input XOR gate. The time complexity due to gates for the proposed subfield inverter is $3D_{ND} + D_{NT}$, where D_{ND} and D_{NT} are the delays of a NAND2 and a NOT gate, respectively.*

5.3 Output Multipliers

The two output multipliers that are shown in Figure 1 generate $W = EB \in GF(2^4)$ and $Z = EA \in GF(2^4)$ (see (4)). The output of these multipliers are represented in the RNB $\{\beta, \beta^2, \beta^{2^2}, \beta^{2^3}, 1\}$, whereas their inputs $A = \sum_{i=0}^{3} a_i\beta^{2^i} = (a_0a_1a_2a_3) \in GF(2^4)$, $B = \sum_{i=0}^{3} b_i\beta^{2^i} = (b_0b_1b_2b_3) \in GF(2^4)$, and $E = \sum_{i=0}^{3} e_i\beta^{2^i} = (e_0e_1e_2e_3)$ are represented in the ONB-I $\{\beta, \beta^2, \beta^{2^2}, \beta^{2^3}\}$. Let $W = (w_0w_1w_2w_3w_4) = \sum_{i=0}^{3} w_i\beta^{2^i} + w_4$ and $Z = (z_0z_1z_2z_3z_4) = \sum_{i=0}^{3} z_i\beta^{2^i} + z_4$ be the outputs of these multipliers. The formulations to generate w_i and z_i can be obtained from (9). They can be realized using two levels of NAND-XOR gates as follows.

$$\begin{aligned} w_0 &= (e_0b_0)' \oplus (e_{12}b_{12})' \\ w_1 &= (e_1b_1)' \oplus (e_{23}b_{23})' \\ w_2 &= (e_2b_2)' \oplus (e_{30}b_{30})' \\ w_3 &= (e_3b_3)' \oplus (e_{01}b_{01})' \\ w_4 &= (e_{02}b_{02})' \oplus (e_{13}b_{13})', \end{aligned} \tag{27}$$

$$\begin{aligned} z_0 &= (e_0a_0)' \oplus (e_{12}a_{12})' \\ z_1 &= (e_1a_1)' \oplus (e_{23}a_{23})' \\ z_2 &= (e_2a_2)' \oplus (e_{30}a_{30})' \\ z_3 &= (e_3a_3)' \oplus (e_{01}a_{01})' \\ z_4 &= (e_{02}a_{02})' \oplus (e_{13}a_{13})', \end{aligned} \tag{28}$$

where $e_{jk} = e_j \oplus e_k$ for $0 \leq j, k \leq 3$, $j \neq k$ and $E = (e_0e_1e_2e_3)$ is the shared input. Note that $E_{jk} = \{e_{01}, e_{02}, e_{03}, e_{12}, e_{13}, e_{23}\}$ are shared between the two multipliers and are generated using six 2-input XOR gates, denoted as the 6XOR block in Figure 2. These signals are used in (27) and (28) which are implemented by 10 NAND-XOR modules. These modules are shown by the two blocks of NAND-XOR in Figure 2 to implement (27) and (28). The details of the NAND-XOR blocks in Figure 2 are shown in Figure 5.

Based on the architecture shown in Figure 2 and Figure 5, the space and time complexities of the output multipliers are as follows.

Proposition 3. *The output multipliers in Figure 2 consists of 16 XOR2 and 20 NAND2 gates with longest propagation delay of $D_{ND} + 2D_X$.*

5.4 Complexity Analysis

The space and time complexities of the input and output transformation blocks, along with all the blocks presented in Propositions 1, 2, and 3 are summarized in Table 5. This table also shows the space and time complexities of the $GF((2^4)^2)$ inversion and the entire AES S-box architecture shown in Figure 2 for lightweight and fast implementations. The corresponding gate equivalent (GE) of all the blocks are also presented. GE is the chip areas in terms of an equivalent 2-input NAND gates. The provided GEs are based on 65 nm CMOS technology in which the areas of XOR2/XNOR2 gate=2 GEs, 2-input NAND gate

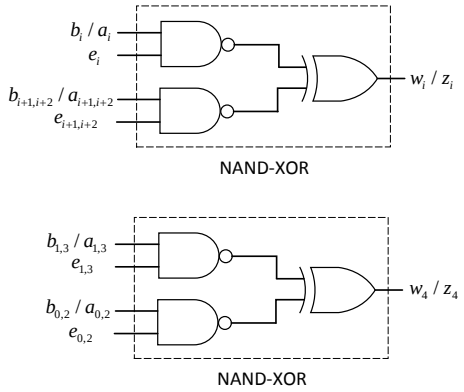

Figure 5: The proposed output multipliers over $GF(2^4)$ to generate w_i/z_i, $0 \leq i \leq 3$, and w_4/z_4. Note that the additions in the input indices are performed modulo 4.

(NAND2)=1 GE, 3-input NAND gate (NAND3)=1.25 GEs, 2-input NOR gate (NOR2)=1 GE, and a NOT gate=0.75 GEs.

Table 5: Space and time complexities for different blocks and the $GF((2^4)^2)$ inversion of the entire proposed S-box architecture shown in Figure 2 (X=XOR2/XNOR2, ND=NAND2, N3=NAND3, NR=NOR2, NT=NOT, 1X= 2GEs, 1ND=1NR= 1GE, 1N3= 1.25GEs, 1NT= 0.75GEs).

Block/ Target	Space Complexity						Time Complexity
	X	ND	N3	NR	NT	GE	
Input Transformation Block							
Lightweight	19					38	$5D_X$
Fast	24					48	$3D_X$
$GF((2^4)^2)$ Inversion							
Exp. Light.	14	7		3		38	$4D_X$
Exp. Fast	14	7		3		38	$3D_X + 1D_{NR}$
Inverter	4	12	4		4	28	$3D_{ND} + 1D_{NT}$
Multipliers	16	20				52	$2D_X + 1D_{ND}$
Total Light.	34	39	4	3	4	118	$6D_X + 4D_{ND} + 1D_{NT}$
Total Fast	34	39	4	3	4	118	$5D_X + 4D_{ND} + 1D_{NT} + 1D_{NR}$
Input Transformation Block cascaded with the $GF((2^4)^2)$ Inversion							
Lightweight	53	39	4	3	4	156	$10D_X$[1] $+ 4D_{ND} + 1D_{NT}$
Fast	58	39	4	3	4	166	$8D_X + 4D_{ND} + 1D_{NT} + 1D_{NR}$
Output Transformation Block							
Lightweight	16					32	$6D_X$
Fast	21					42	$3D_X$
Total Complexity of Proposed S-box (Figure 2)							
Lightweight	69	39	4	3	4	188	$16D_X + 4D_{ND} + 1D_{NT}$
Fast	79	39	4	3	4	208	$11D_X + 4D_{ND} + 1D_{NT} + 1D_{NR}$

[1] One D_X can be saved by incorporating the delay of the input transformation block in selecting the order of mod-2 additions within the exponentiation block.

6 Our Investigations for the Best Choices

We highlight that the aforementioned schemes are proposed as a result of extensive investigation considering many other design options. In this section, we discuss some of the design choices that we explored and how the proposed schemes were selected.

6.1 Different Bases and the Corresponding Multipliers

There are three multipliers that are used within the architecture of composite field inversion. Therefore, choosing an appropriate multiplier plays an important role in overall cost of the circuit. The three multipliers, as seen in Figure 1, process three subfield elements, namely A, B, and E as their inputs. In other words, each input field element activates two multipliers. Here, we are interested in pre-processing the input elements in a way that can minimize some of the internal circuits of the two activated multipliers. In other words, instead of minimizing the cost of individual multipliers, we try to minimize the combined cost of the three multipliers.

Table 6 shows details of the multipliers used in a number of S-boxes. We derived the space and time complexities of these multipliers based on the circuits and information provided in these papers. As shown in the table, the multipliers can be designed upon three different types of bases; polynomial basis (PB), normal basis (NB) and mixed basis (MB), and are built using a set of AND/NAND and XOR/XNOR gates.

Table 6: Multipliers used in different AES S-boxes with their sharing capability between different multipliers (X=XOR, ND=NAND, AD=AND).

S-box	Multiplier	# inputs/ # outputs	Space Complexity	Time Complexity	Sharing Inputs
Subfield $GF((2^2)^2)$					
Satoh et al. [SMTM01]	PB [Paa96]	$4 \times 4/4$	21X+9AD	$4D_X + D_{AD}$	both
Canright [Can05b]	NB	$4 \times 4/4$	20X+9ND	$4D_X + D_{ND}$	both
Nogami et al. [NNT+10]	MB	$4 \times 4/4$	21X+9AD	$4D_X + D_{AD}$	both
Subfield $GF(2^4)$					
Rudra et al. [RDJ+01]	PB [Mas91]	$4 \times 4/4$	15X+16AD	$3D_X + D_{AD}$	one
Gueron et al. [GM16]	PB	$4 \times 4/4$	15X+16ND	$3D_X + D_{ND}$	none
Nekado et al.[NNI12]	RNB	$5 \times 5/5$	25X+10AD	$2D_X + D_{AD}$	both
Ueno et al. [UHS+15]	RNB [NNI12]	$4 \times 5/5$	21X+10AD	$2D_X + D_{AD}$	both
This work	NB [RH03]	$4 \times 4/5$	17X+10ND	$2D_X + D_{ND}$	both

Assume that the input E with coordinates of e_j, $0 \leq j \leq 3$, is connected to two multipliers, generating the outputs AE and BE (see for example, the output multipliers in Figure 1). If the two inputs of each multiplier are first processed through a set of AND/NAND gates, no gate sharing can happen between the multipliers. This is because, the set of gates in the first multiplier (with inputs A and E) will generate $a_i e_j$, with $0 \leq i, j \leq 3$, while the set in the second multiplier (with inputs B and E) will generate $b_i e_j$, with $0 \leq i, j \leq 3$. It is clear that the signals in these two sets do not match, hence no sharing can happen. Such a multiplier was used in [RH04], [KS98], [RH02], [MO86], and recently in [GM16] while preventing sub-expression sharing between the multipliers. Hence, although the hardware cost of an individual multiplier may be small, the combined cost of the three multipliers will be high. As a result, these multipliers are not good candidates for our objective.

The Mastrovito multiplier [Mas91], which is used in the scheme proposed by Rudra et al. [RDJ+01], uses a set of AND gates between the bits of one input and a set of XOR gates between the bits of the other input. As mentioned in [RDJ+01], no optimization

has been performed in their scheme. However, sharing can only happen if the architecture uses the input connected to the set of XOR gates as the common input between the two multipliers, i.e., E in the multipliers with outputs AE and BE. Therefore, no sharing in the other inputs (A or B) can be used.

On the other hand, the maximum sharing can be utilized if each input is first processed through a set of XOR/XNOR gates between its bits, independently, as a pre-processing step. For example, the set in the first multiplier (with inputs A and E) will generate $a_i \oplus a_j$, with $0 \leq i, j \leq 3, i \neq j$, and $e_i \oplus e_j$, with $0 \leq i, j \leq 3, i \neq j$. The set in the second multiplier (with inputs B and E) will generate $b_i \oplus b_j$, with $0 \leq i, j \leq 3, i \neq j$, and $e_i \oplus e_j$, with $0 \leq i, j \leq 3, i \neq j$. Here, the six gates $e_i \oplus e_j$, with $0 \leq i, j \leq 3, i \neq j$ are common in the two multipliers, and can be shared. Such multipliers are used in the schemes proposed by Canright [Can05a, Can05b] and Nogami et al. [NNT+10]. These multipliers are similar to the Karatsuba multiplier [KO63] and are good candidates in terms of sharing. However, using them results in a long delay which is not favorable for our fast objective.

In [NNI12], Nekado et al. presented a fast RNB multiplier with subexpresion sharing in both multiplier inputs. The multiplier uses the RNB representation for the ONE-I over $GF(2^4)$ generated by the AOP. Both inputs and the output of this multiplier have 5 bits (denoted by $5 \times 5/5$ in Table 6). This multiplier requires 25 XOR2 and 10 AND2 gates with delay of $2D_X + D_{AD}$. Recently, Ueno et al. [UHS+15] used the RNB multiplier proposed in [NNI12] for the RNB multiplications, where one input has a 4-bit representation while the other input and the output are represented in the RNB with 5 bits (denoted by $4 \times 5/5$ in Table 6). This multiplier reduces the number of XOR gates to 21 with the same number of 10 AND2 gates and the delay of $2D_X + D_{AD}$.

In this paper, we use the NB multiplier proposed in [RH03] for the ONE-I over $GF(2^4)$ generated by the AOP. Both inputs of this multiplier are represented in the NB, and we chose to represent the output in the 5-bit RNB representation. The original formulations in [RH03] for this multiplier were based on AND-XOR implementations. However, we improved this multiplier by using NAND gates instead of AND gates. This modification improves both the lightweight and fast implementations as the chip area and delay of NAND gates are lower than those of AND gates. As seen from Table 6, the multiplier that we use in this paper has the lowest gate count and the smallest delay available in the literature to the best of our knowledge.

6.2 Different Input Transformation Matrices

In our proposed scheme for the lightweight S-box architecture (shown in Figure 2), the proposed input transformation matrix $\mathbf{T}_{in}$ is selected to be 20×8 (see the input transformation matrix $\mathbf{T}_{in}$ presented in (15)). We investigated additional choices for the input transformation matrices with a corresponding slight modification in the exponentiation block in order to obtain smaller overall space/time complexity. In the following items, we detail our investigations for different input transformation matrices in which 2, 3, and 4 new rows are appended to the existing 20×8 binary matrix $\mathbf{T}_{in}$. Note that we already considered the traditional 8×8 binary input matrix in Table 1.

- *An appended* 22×8 *input matrix*: Since our exponentiation computation block (Figure 3) uses $a_1 \oplus b_1$ and $a_3 \oplus b_3$ as inputs, we considered moving the computation of one (or both) of these two formulations into the $\mathbf{T}_{in}$ matrix. The results of our code showed that we need an extra one gate (or respectively two gates) in the input transformation block. This means that no saving was obtained, while increasing the computation time of the logic-minimization algorithms.

- *An appended* 23×8 *input matrix*: One can derive another set of formulations for $D = (A + B)^2\beta + AB$ using (19) for $(A + B)^2\beta$ with the addition to $C = AB$

presented in Lemma 1. The resultant new formulations for D require $(a_0 \oplus b_0)$, $(a_1 \oplus b_1)$ and $(a_3 \oplus b_3)$ in the exponentiation computation block. To consider this, we added 3 additional rows to compute $(a_0 \oplus b_0)$, $(a_1 \oplus b_1)$ and $(a_3 \oplus b_3)$ to the $\mathbf{T}_{in}$ matrix. This added three more XOR gates to the space complexity and hence, no improvement was found.

- *An appended* 24×8 *input matrix*: Similarly, using (20), we found another set of formulations for D in which $(a_1 \oplus b_1)$, $(a_{13} \oplus b_{13})$, $(a_{12} \oplus b_{12})$ and $(a_{01} \oplus b_{01})$ are used. Similarly, we added 4 new rows (in order to compute these signals) to the existing $\mathbf{T}_{in}$ matrix and no improvement in terms of space complexity was obtained.
- *A faster* 20×8 *input matrix*: We searched for a faster input transformation block to reduce its propagation delay from $3D_X$ to $2D_X$. In fact, we searched for an input transformation matrix $\mathbf{T}_{in}$ with at most four 1s in all the 20 rows of the matrix and we could not find any such a matrix.

6.3 Different Output Transformation Matrices

Similarly, we considered several other revised output transformation matrices with slight modifications to the output multipliers in Figure 2. The content and the size of the existing 8×10 binary matrix $\mathbf{T}_{out}$ were revised. The size of the first output matrix was reduced to 8×8, whereas the size of the second and the third output matrices were increased to 8×20 and 8×10, respectively. These investigations are detailed as follows.

- *A revised* 8×8 *output matrix*: The output subfield elements $W = (w_0 w_1 w_2 w_3 w_4)$ and $Z = (z_0 z_1 z_2 z_3 z_4)$ of the proposed S-box are represented in the 5-bit RNB of $W = \sum_{i=0}^{3} w_i \beta^{2^i} + w_4$ and $Z = \sum_{i=0}^{3} z_i \beta^{2^i} + z_4$. By adding 8 XOR gates at the outputs of the $GF((2^4)^2)$ inversion block, one can obtain the representation of the outputs with respect to the NB as $W = \sum_{i=0}^{3} (w_i \oplus w_4) \beta^{2^i}$ and $Z = \sum_{i=0}^{3} (z_i \oplus z_4) \beta^{2^i}$ using **T1** in (14). Adding these 8 XOR gates to the result of optimizing the 8×8 output binary matrix **MX**, we found that our 8×10 $\mathbf{T}_{out} = \mathbf{MXT1}$ has a smaller area.
- *A revised* 8×20 *output matrix*: The very last stage in the output multipliers before generating W and Z consists of 10 XOR gates. We moved these gates to the output transformation matrix $\mathbf{T}_{out}$ so that the new output matrix has 20 columns (double the original one). The result of optimizing this matrix was worse than the 8×10 matrix while significantly increasing the computation time.
- *A faster* 8×10 *output matrix*: The current output transformation matrix $\mathbf{T}_{out}$ in (16) has at most eight 1s in each row. Our current fast implementation of the output transformation block has $3D_X$ and to make it faster, we have to find a matrix with at most four 1s in all rows. We searched for such a matrix across all composite field representations and we could not find any matrix with this criteria.

6.4 Different Subfield Inverters

In addition to the subfield inverter that was selected in Subsection 5.2, we derived 11 other equivalent functions using Boolean algebra and Karnough maps. In order to select the best circuits in the two design goals (area and speed), we coded all the 12 functions in VHDL and evaluated their ASIC implementation results. The functions and the corresponding ASIC implementation results are provided in Appendix B. Based on the hardware results, the circuit presented in Sec. 5.2 was the best in both area *and* delay.

Moreover, instead of designing the subfield inverter as a 4 inputs/4 outputs circuit, we tested the following choices:

1. 4 inputs/10 outputs: The 4 inputs are d_0, d_1, d_2, d_3 and the 10 outputs are e_0, e_1, e_2, e_3, e_{01}, e_{02}, e_{03}, e_{12}, e_{13}, e_{23} . In this design, the six 2-input XOR gates used in the output multipliers to derive $e_{01}, e_{02}, e_{03}, e_{12}, e_{13}, e_{23}$ can be shared with the subfield inverter, suggesting a saving of 12 GEs with reducing $1D_X$ delay.

2. 5 inputs/4 outputs: The 5 inputs are $\widetilde{d}_0, \widetilde{d}_1, \widetilde{d}_2, \widetilde{d}_3, c_4$ and the 4 outputs are e_0, e_1, e_2, e_3. Based on this design, the four 2-input XOR gates used to derive d_0, d_1, d_2, d_3 from $\widetilde{d}_0, \widetilde{d}_1, \widetilde{d}_2, \widetilde{d}_3, c_4$ are no longer needed, suggesting a saving of 8 GEs and reducing $1D_X$ delay.

3. 5 inputs/10 outputs: The 5 inputs are $\widetilde{d}_0, \widetilde{d}_1, \widetilde{d}_2, \widetilde{d}_3, c_4$ and the 10 outputs are $e_0, e_1, e_2, e_3, e_{01}, e_{02}, e_{03}, e_{12}, e_{13}, e_{23}$. This design suggests a saving of ten 2-input XOR gates, a total of 20 GEs with reducing $2D_X$ delay.

The truth table for each new design is used as the design entry method to Logic Friday® [Ric12]. Logic Friday is then used to minimize the logic expressions for the outputs and VHDL is used for coding. The ASIC implementation results show no improvement in the speed of these cases compared to the 4 inputs/4 outputs circuit. The overall gate count of the $GF((2^4)^2)$ inversion circuit using choices #1, #2 and #3 turns out to be 12, 8 and 20 GEs, respectively, more than the gate count of the subfield inverter using 4 inputs/4 outputs. The delay did not improve as a result of using a new number of inputs and/or outputs for the subfield inverter, suggesting the use of the 4 inputs/4 outputs circuit in the final design.

7 Further Optimizations Using CAD Tools

This section complements the extensive investigation started in the previous section, while using technology-supported CAD tools. In order to avoid the pitfall discussed in the introduction, we invoke CAD tools with behavioral modeling to see if the design block could be further optimized using compound/complex gates, in terms of both area and delay.

To achieve this goal, we coded every block within the S-box architecture, separately and combined, using both structural and behavioral modeling of VHDL. In behavioral modeling, we define the circuits based on their input-output relationship (their behavior), and let the CAD tool selects the best implementation based on the available gates in a target library. In the previous section, we used structural modeling where we define the exact gates and circuit structure based on formulations, with no optimization in the gate selection by the CAD tool. Then, we compared the ASIC implementation results of the behavioral modeling against the structural modeling. The results are shown in Table 7 and Table 8 in terms of area (in both μm^2 and GE values), delay (in ns) and power (in μW). For all the ASIC synthesis results presented in this paper, we used the relaxed constraints at a clock frequency of 100 MHz. The gate equivalent (GE) values are calculated by dividing the corresponding area by the area of a 2-input NAND gate, i.e., 2.08 μm^2. Table 7 shows references to the figures (for structural modeling) and the equations (for behavioral modeling) that are used in the comparison.

7.1 Behavioral Modeling of the Composite Field Inversion

Here, we study the blocks used within the composite field inversion; the exponentiation block, the subfield inverter block, and the output multipliers block. Then, we conclude with the overall architecture of the composite field inversion.

For the structural modeling of the exponentiation computation block, we evaluated two circuits; Figure 3(a), denoted as Code#1 and Figure 3(b), denoted as Code#2.

Table 7: ASIC synthesis results for three blocks of the $GF((2^4)^2)$ inversion and the entire $GF((2^4)^2)$ inversion.

Block	Code #	Imp.	Fig. Equ. (#)		Area μm^2	Area GE	Delay ns	Power μW
Expon.[1]	1	Str. Light.	Fig. 3(a)		62.4	30	0.10250	2.600
	2	Str. Fast	Fig. 3(b)		62.4	30	**0.09072**	2.479
	3	Beh.[2]	(24)		60.84	**29.25**	0.10037	1.851
Subfield Inverter[1]	4	Str.	Fig. 4		74.88	36	0.12131	3.836
	5	Beh.[3]	(22), (26)		64.48	**31**	**0.10244**	3.257
Output Mults.	6	Str.	Fig. 5		108.16	**52**	**0.09892**	4.096
	7	Beh.	(27), (28)		111.28	53.5	0.12088	4.327
$GF((2^4)^2)$ inversion	Behavioral coding of three blocks							
			Module	Codes				
	8	All-Str. Light.	1 and 3	1, 4, 6	245.44	118	0.41705	25.132
	9	All-Str. Fast	1 and 3	2, 4, 6	245.44	118	**0.40527**	25.012
	10	All-Beh.	3	3, 5, 7	236.6	113.75	0.49576	23.996
	11	All-Beh.	1		249.6	120	0.46065	24.274
	12	Comb. Light.	3	3, 5, 6	233.48	**112.25**	0.49863	23.554
	13	Comb. Light.	1		240.24	115.5	0.47017	23.868
	14	Comb. Fast	1 and 3	2, 5, 6	235.04	113	0.43682	23.955

[1] The four XOR gates at the output of exponentiation computation block in Figure 3 are moved to the subfield inverter in Figure 4 for accurate delay comparison.

[2] See Figure 6(a) for the behavioral-modeling architecture of the exponentiation block.

[3] See Figure 6(b) for the behavioral-modeling architecture of the subfield inverter block.

While evaluating these two circuits, we moved the four XOR gates at the output of the exponentiation computation block in Figure 3 to the input of the subfield inverter block in Figure 4 in order to achieve accurate delay comparison. For the behavioral coding of the exponentiation block, we coded the formulations provided in (24), and denoted the circuit as Code#3. The results show that our fast circuit (Code#2 of Figure 3(b)) is truly the fastest circuit. However, the behavioral modeling brought a slightly smaller circuit by using 3-input XOR gates instead of two 2-input XOR gates. Figure 6(a) shows the circuit generated using behavioral modeling of the exponentiation block.

Similarly, we evaluated the subfield inverter using structural modeling of Figure 4 (denoted Code#4), and behavioral modeling of (22) and (26) (denoted Code#5). The behavioral modeling presented a circuit that is smaller and faster than Figure 4 by using the non-obvious OAI32 (OR-AND-Invert) compound gate as a replacement of the NAND-NAND at the output of Figure 4. Figure 6(b) shows the circuit generated from the behavioral modeling of the subfield inverter. Using some Boolean algebra and De Morgan's laws, one can show that Figure 4 is equivalent to Figure 6(b).

For the output multipliers, we evaluated the structural modeling of Figure 5 (denoted Code#6), and the behavioral modeling of (27) and (28) (denoted Code#7). Here, the proposed circuit in Figure 5 (Code#6) outperforms the behavioral modeling in both area and delay.

Table 7 also shows the options we investigated in building the overall composite field inversion. We chose 5 different combinations of the evaluated codes from the three blocks of the $GF((2^4)^2)$ inversion. Codes#(1,4,6) combines all the structural modeling for a lightweight design. Codes#(2,4,6) combines all the structural modeling for a fast design. Codes#(3,5,7) combines all the behavioral modeling. Then, we combined the most lightweight codes; Codes#(3,5,6), and the fastest codes; Codes#(2,5,6). In addition,

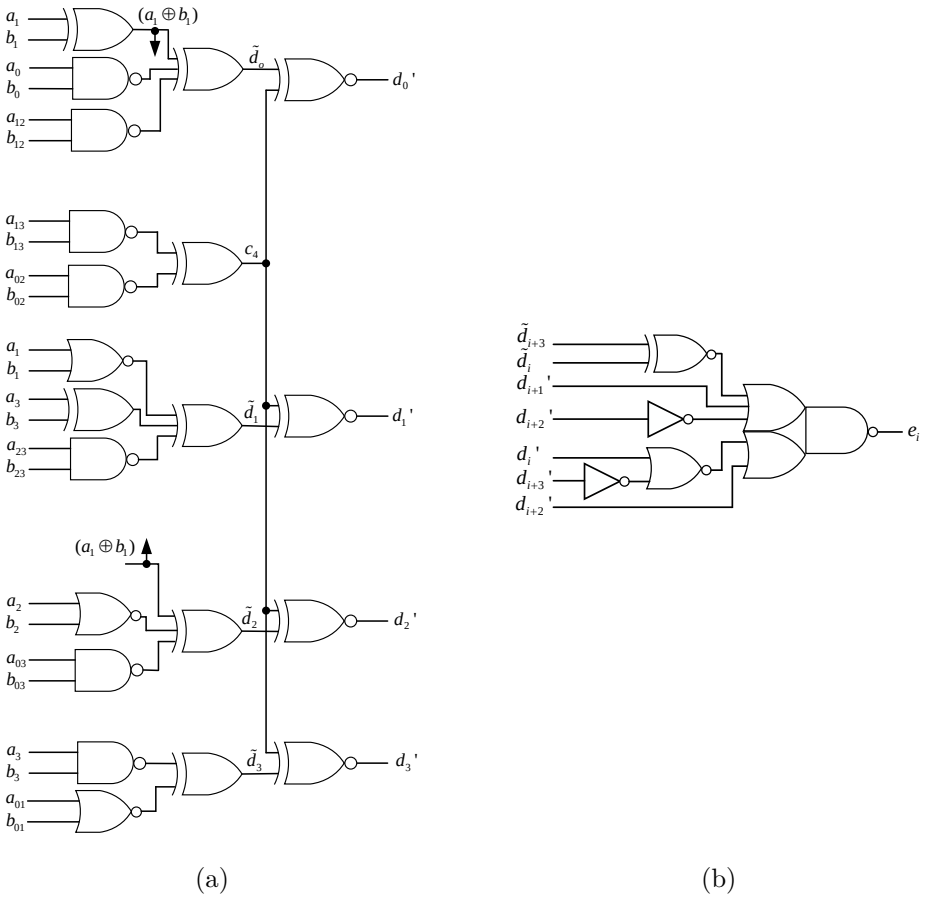

Figure 6: The least-area synthesized architectures from the behavioral modeling of (a) The exponentiation computation block using 3-input XOR gares. (b) The subfield inverter block using an OAI32 (OR-AND-Invert) compound gate.

whenever codes from behavioral modeling is used, we test the circuit as a single module versus three separate modules. Notation for the resultant codes, along with the ASIC implementation results are listed in Table 7.

The results show that the fastest circuit for the composite field inversion is built using structural modeling in each block of the S-box (Code#9). The most lightweight circuit is found by using the behavioral modeling of both the exponentiation block and the subfield inverter, along with the structural modeling of the output multipliers (Code#12).

7.2 Behavioral Modeling of the Input/Output Transformation Blocks

Here, we apply the same evaluation methodology against the input and output transformation blocks, and list the results in Table 8.

For structural coding of the input transformation block, we used the formulations presented in Table 3a and Table 4a (Code#15 and Code#16, respectively). In addition, we evaluated the behavioral modeling based on (15) (Code#17). It is interesting to see that our lightweight structural design has the least area and our fast design has the lowest delay. In fact, our fast design achieves a smaller area than the behavioral modeling. These results validate our logic-minimization algorithms (presented in Sec. 4).

Analysis of the output transformation block goes along the same line. We evaluated the structural modeling following Table 3b and Table 4b, along with the behavioral modeling using (16). Similarly, our structural modeling significantly outperforms behavioral modeling.

Table 8: ASIC synthesis results for the input and transformation matrix blocks.

Trans. Block	Code #	Implementation	Table Equ. (#)	Area μm^2	Area GE	Delay ns	Power μW
Input $\mathbf{T}_{in}$	15	Str. Light.	Table 3a	79.04	**38**	0.266117	6.1824
	16	Str. Fast	Table 4a	99.84	48	**0.166328**	6.9038
	17	Behavioral	(15)	120.64	58	0.205561	6.9460
Output $\mathbf{T}_{out}$	18	Str. Light.	Table 3b	66.56	**32**	0.307335	5.779
	19	Str. Fast	Table 4b	87.36	42	**0.155989**	5.5278
	20	Behavioral	(16)	95.68	46	0.190978	5.6164

8 Implementation Results and Comparison

The overall implementation results are proposed in Table 9. We propose two lightweight cores: the all-structural and the combined cores. The combined core is preferred if the underlying technology library supports the XOR3 and OAI32 used in its design. Otherwise, the all-structural core should be used. The effect of target technology library will be detailed in Sec. 8.3. In addition, we propose the all-structural fast core. We include the all-behavioral code for reference.

Logic synthesis was done using VHDL as a design entry to the Synopsys Design Vision®. The technology library used was STM 65-nm CMOS standard library and the CORE65LPSVT standard cell library which is optimized for low power applications. The area, delay and power for all the considered S-boxes are generated by the CAD tool with relaxed constraints at a clock frequency of 100 MHz.

Table 9: ASIC synthesis results for the proposed S-box architectures.

Architecture	Code #	Using codes	Area μm^2	Area GE	Delay ns	Power μW
All-Str. Light. (Prop._a)	21	8,15,18	391.04	188	1.080835	39.930
Comb. Light. (Prop._b)	24	12,15,18	379.08	**182.25**	1.197698	38.085
All-Str. Fast (Prop.)	22	9,16,19	432.64	208	**0.779697**	42.750
All-Behavioral	23	10,17,20	452.92	217.75	1.004363	42.259

8.1 Comparisons of the Space and Time Complexities

In this section, we compare the space and time complexities of the proposed lightweight and fast S-box architectures against the fastest and most compact S-boxes available in the literature. In addition, we propose structural improvements to these architectures.

Tables 10 and 11 provide the gate count and time delay of five S-boxes and compare them with the proposed architectures. We chose two of the most lightweight schemes, namely Canright [Can05a, Can05b], and the 113-gate design in [Boy16], which was found using the Boyar-Peralta heuristic [BP10] while exhaustively searching through all the ties. In addition, we chose three of the fastest schemes, namely the two Boyar et al. designs in [BP12] and [BFP17], along with the Ueno et al. design [UHS+15]. The space complexity comparison of fast designs helps in detecting where the speed improvements came from

Table 10: Space complexity comparison of different S-boxes (X=XOR2/XNOR2, X3=XOR3, A=AND2, O=OR2, ND=NAND2, N3=NAND3, OAI=OAI32, NR=NOR2, NT=NOT).

	S-boxes	Gate count									GE
		X	X3	A	O	ND	N3	OAI	NR	NT	
Lightweight	Canright [Can05b]	80	-	-	-	34	-	-	6	-	200
	Boyar-Op113 [Boy16]	81	-	32	-	-	-	-	-	-	202
	Imp. Boyar-Op113	81	-	-	-	32	-	-	-	-	194
	All-Str. Light. (Prop._a)[1]	69	-	-	-	39	4	-	3	4	188
	Comb. Light. (Prop._b)	63	3	-	-	27	-	4	7	4	182.25
Fast	Boyar-Dp16-1 [BP12]	94	-	34	-	-	-	-	-	-	230.5
	Imp. Boyar-Dp16-1	94	-	-	-	30	-	-	4	-	222
	Boyar-Dp16-2 [BFP17]	91	-	34	-	-	-	-	-	-	224.5
	Imp. Boyar-Dp16-2[2]	91	-	-	-	30	-	-	4	-	216
	Ueno et al. [UHS+15]	91	-	38	16	-	-	-	-	4	256.5
	Imp. Ueno[3]	91	-	-	-	35	4	-	13	4	238
	All-Str. Fast (Proposed)[1]	79	-	-	-	39	4	-	3	4	208

[1]See Table 5 for the details.

[2]See Appendix C (Table 16) for the improved formulations.

[3]See Appendix C (Table 15) for the improved formulations.

and also in validating the CAD tool results. To the best of our knowledge, these schemes are the most lightweight and the fastest S-boxes available in the literature to date.

The gate count for Canright S-box is obtained from [Can05a] and [Can05b]. Let us denote the 113-gate design in [Boy16] as Boyar-Op113, the 16-depth circuit, with 128 gates in [BP10] as Boyar-Dp16-1, and the new 16-depth circuit, with 125 gates in [BFP17] (which is available in [Boy16]) as Boyar-Dp16-2.

The Boyar-Op113 scheme requires 113 gates (81 XOR with 32 AND operations). The Boyar-Dp16-1 scheme uses 94 XOR2/XNOR2 with 34 AND operations, while the Boyar-Dp16-2 scheme uses 91 XOR2/XNOR2 with 34 AND operations. As mentioned earlier, it is cheaper and faster to use NAND gates instead of AND gates in the ASIC implementations. Therefore, we improved these schemes by changing the AND gates to NAND gates and tracing the results of such changes by replacing some XOR to XNOR and some XNOR to XOR. We also used Boolean algebra whenever the input of an AND gate is complemented to change the AND gate to a NOR gate. To adjust the complement of the other AND input, we moved its complement to the gate that generates it and again trace the complement of this gate to all the other gates. At the end, we verified correctness of the improved schemes by our test benches and using the CAD tools. The exact formulations that we used for the improved S-boxes are included in Appendix C. Formulations listed in Table 16 of the Improved Boyar-Dp16-2 can be used to derive the formulations of the Improved Boyar-Dp16-1, as they share the same nonlinear circuit.

The gate count of the original Boyar-Op113 [Boy16], the Boyar-Dp16-1 [BP12], and the Boyar-Dp16-2 [BFP17] schemes as well as those of the improved schemes are presented in Table 10. In Table 10, we also provided the GE for all S-box architectures using the corresponding GE of the gates from the CMOS 65 nm technology. In our GE calculations, we used XOR2/XNOR2 (2 GEs), AND2 (1.25 GEs), OR2 (1.25 GEs), NAND2 (1 GE), NAND3 (1.25 GEs), NOR2 (1 GE), and NOT (0.75 GEs).

The gate count of the $GF((2^4)^2)$ inversion of the original scheme proposed by Ueno et al. is provided in [UHS+15] as [51 XOR2 + 38 AND2 + 16 OR2 + 4 NOT] gates. However, no information regarding the gate counts of the input matrix (denoted by Δ_f in their paper) and output matrix (denoted by Δ_l in their paper) is provided in the paper. We

Table 11: Time complexity comparison of different S-boxes (X=XOR2/XNOR2, X3=XOR3, A=AND2, O=OR2, ND=NAND2, N3=NAND3, OAI=OAI32, NR=NOR2, NT=NOT).

Design	S-boxes	CPD
Lightweight	Canright [Can05b]	$19D_X + 3D_{ND} + 1D_{NR}$
	Boyar-Op113 [Boy16]	$21D_X + 6D_A$
	Improved Boyar-Op113	$21D_X + 6D_{ND}$
	All-Str. Light. (Prop._a)[1]	$16D_X + 4D_{ND} + 1D_{NT}$
	Comb. Light. (Prop._b)	$15D_X + 1D_{X3} + 1D_{ND} + 1D_{OAI} + 1D_{NT}$
Fast	Boyar-Dp16-1 [BP12]	$14D_X + 2D_A$
	Improved Boyar-Dp16-1	$14D_X + 1D_{ND} + 1D_{NR}$
	Boyar-Dp16-2 [BFP17]	$13D_X + 3D_A$
	Improved Boyar-Dp16-2[2]	$14D_X + 2D_A$
	Ueno et al. [UHS+15]	$11D_X + 3D_A + 1D_O$
	Improved Ueno[3]	$10D_X + 4D_{ND} + 1D_{N3} + 1D_{NT}$
	All-Str. Fast (Proposed)[1]	$11D_X + 5D_{ND} + 1D_{NT}$

[1]See Table 5 for the details.

[2]See Appendix C (Table 16) for the improved formulations.

[3]See Appendix C (Table 15) for the improved formulations.

drove the gate counts of the input and output matrices (14 and 26) to minimize the total number of XOR2/XNOR2 gates needed in these blocks. Most importantly, we made sure the delays of these transformation matrices are the same as the ones proposed in [UHS+15], namely $2D_X$ and $3D_X$, respectively. As a result, the delay of the entire S-box remains the same as the one proposed in [UHS+15], i.e., $11D_X + 3D_A + 1D_O$. The same critical path delay (CPD) was obtained by the CAD tool after coding the original Ueno et al. [UHS+15] S-box architecture in VHDL. Since the original Ueno et al. used several AND gates, we improved the design using NAND gates by changing the corresponding formulations of all stages and blocks in their architecture. We provide the details of the new formulations in Appendix C, Table 15. As one can see from Table 10, our proposed Lightweight and Fast schemes have the least GE, as compared with their counterparts.

The CPD of those schemes are also compared with the proposed ones in Table 11. These delays are obtained from the corresponding papers and are also verified by the codes. More importantly, we used the CAD tool to provide the CPD in terms of the number and type of gates. The results from the CAD tool match the ones provided in the corresponding papers except for the Boyar-Op113 scheme which is reduced to the depth of 27 from the depth of 28 mentioned in [BFP17]. The CPD and the gate count for the improved schemes are derived by analysis of the improved formulations, and verified using the CAD tool. As seen from Table 11, our proposed lightweight S-box is faster than other lightweight schemes, namely Canright, Boyar-Op113 and the improved Boyar-Op113. Also, our proposed Fast S-box is faster than five other fast schemes and comparable with our improved version of the Ueno S-box.

8.2 ASIC Implementations and Comparisons

We coded all the above-mentioned S-boxes (original and improved ones) in VHDL and present their ASIC results in Table 12. For each and every code, we verified the codes by testing the S-box test benches using Modelsim®.

The original code for the Canright scheme was obtained from [Can05a], where they provide a behavioral modeling (in Verilog) for the combined S-box/inverse S-box core. We

Table 12: ASIC comparisons of the S-box architectures.

S-box	Code #	Area		Delay	Power	Area-Time product
		μm^2	GE	ns	μW	
Canright_Beh. [Can05b]	25	433.68	208.5	1.287395	42.125	268.422
Canright_Str.	26	416	200	1.252811	41.023	250.562
Boyar-Op113 [Boy16]	27	420.16	202	1.522775	39.365	307.601
Improved Boyar-Op113	28	403.52	194	1.34606	40.471	261.136
All-Str. Light. (Prop._a)	21	391.04	188	**1.080835**	39.930	**203.20**
Comb. Light. (Prop._b)	24	**379.08**	**182.25**	**1.197698**	**38.085**	**218.28**
Boyar-Dp16-1 [BP12]	29	479.44	230.5	0.960458	44.020	221.386
Improved Boyar-Dp16-1	30	461.76	222	0.905652	44.797	201.055
Boyar-Dp16-2 [BFP17]	31	466.96	224.5	0.956535	42.724	214.742
Improved Boyar-Dp16-2[1]	32	449.28	216	0.911743	43.645	196.936
Ueno et al. [UHS+15]	33	533.52	256.5	0.831007	48.178	213.153
Improved Ueno et al.[2]	34	495.04	238	**0.772424**	49.609	183.837
All-Str. Fast (Proposed)	22	**432.64**	**208**	**0.779697**	**42.750**	**162.177**

[1]See Appendix C (Table 16) for the improved formulations.

[2]See Appendix C (Table 15) for the improved formulations.

revised it, with minimal changes, to a behavioral modeling of an S-box-only core, leading to a hardware cost of 208.5 GEs (denoted Canright_Beh. in the table). Then, we wrote a structural code following the formulations written in the paper, in order to exactly match the hardware complexity provided in the report, leading to a hardware cost of 200GEs (denoted Canright_Str. in the table). Note that in [KR12], the authors mention that the Boyar-Peralta S-box [BP10] can be implemented in 193.8 GEs by converting AND gates to NAND gates, but they did not detail on the exact equations used.

The results listed in Table 12 show that the proposed combined lightweight design is the smallest, fastest, lowest power consumption, and most efficient (measured by area×delay) S-box design in this category to date. The all-structural lightweight design, as proposed in this paper, has a slightly less delay and better overall efficiency for a slight higher implementation area. Similarly, the proposed all-structural fast design is the smallest, fastest (in almost a tie with our improved version of the Ueno S-box), lowest power consumption, and most efficient S-box design in this category to date.

8.3 Discussion on the Effect of Target Technology Library

Our design concept depends on testing the behavioral modeling on the target technology library against the structural modeling of every block to ensure best usage of the available gates. Following this concept, we tested all the codes under different libraries ([Nan, VTV, NG00]) and found that the results are in-line with Table 12.

The industrial technology libraries, e.g., STM and TSMC, support the XOR3 and OAI32 gates that are used in the proposed combined lightweight design, targeting 182.25 GEs. However, other open-source/educational libraries, e.g., the NanGate [Nan], VTVT [VTV] and WPI [NG00] libraries, may not support the XOR3 and/or the OAI32 gates. As a result, behavioral modeling under these technology libraries may lead to a different combined lightweight design.

As a concrete example, we tested the behavioral modeling of the proposed S-box against the NanGate45nm library. NanGate45nm library does not support the XOR3 and OAI32 gates, but supports AOI12 and OAI12 gates instead. As a result, the CAD tool reported a lightweight design of 186 GEs, instead of 182.25 GEs. The exact hardware complexity of a combined lightweight S-box under the NanGate45nm library is [69 XOR2 + 31 NAND2 +

7 OAI21 + 1 AOI21 + 3 NOR2 + 5 NOT], that can be compared to Table 10.

In any case, the all-structural lightweight and fast S-box designs do not require any compound gate, only simple 2-input gates and NAND3. Hence, these designs are supported by all the technology libraries that we aware of and serve as a fallback choice if the target technology library does not support any compound gate.

If the 4 NAND3 gates in Figure 4 are not allowed in the design, e.g., for theoretical comparison against previous work, we may replace each NAND3 gate (1.25 GEs) by one NAND2 and one NOR2 gates (1+1=2 GEs), where the inputs of the NOR2 should be complemented with no cost. Since we use 4 NAND3 gates in each designs (see Table 10), the all-structural lightweight and all-structural fast designs would require 191 and 211 GEs, respectively. These designs can be implemented using the same type of gates that are used by previous work.

9 Conclusions and Future Work

The contributions proposed in this paper are manyfold. We have proposed two new designs for the AES S-box that break all the current implementation records in the two design criteria of lightweight and fast, to the best of our knowledge. We have also introduced several improvements to the current logic-minimization heuristics and proposed three new ones. More importantly, we have demonstrated that optimum ASIC designs can only be achieved through a perfect synergy between theoretical analysis and technology-assisted CAD tools. Our extensive analysis and exhaustive ASIC implementations show that the proposed lightweight and fast architectures outperform in terms of area, delay and efficiency as compared to the best S-box architectures available in the literature.

As a future research direction, we will investigate if a combined S-box/inverse S-box design using composite field architecture over $GF((2^4)^2)$, as proposed in this paper, can outperform the combined S-box/inverse S-box designs under tower field architecture over $GF(((2^2)^2)^2)$ as proposed in [Can05b] and [RTA18]. In addition, we will investigate designing full AES encryption and/or decryption engines, where the field conversion mappings happen only at the data and key inputs and data output. In this case, we may save the repetitive field conversion mappings that happen at the input and output of each SubBytes operation as proposed in this paper.

Acknowledgments

We would like to thank the reviewers of TCHES for their constructive comments and the Canadian Microelectronics Corporation (CMC) Microsystems for providing the required infrastructure and the CAD tools used in this work. This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under the Discovery and Discovery Accelerate Supplement (DAS) Grants awarded to A. Reyhani-Masoleh.

References

[AH13] Nabihah Ahmad and S.M. Rezaul Hasan. Low-power compact composite field AES S-Box/inv S-Box design in 65nm cmos using novel xor gate. *Integration, the VLSI Journal*, 46(4):333 – 344, 2013.

[BFP17] Joan Boyar, Magnus Find, and René Peralta. Low-depth, low-size circuits for cryptographic applications. In *Boolean Functions and their Applications BFA-The 2nd International Workshop on, Os, Hordaland, Norway, July 3-8, 2017, Proceedings*, 2017.

[BMP08] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2008.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, 2013.

[Boy16] CMT: Circuit minimization team, 2016. http://www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html, last accessed on: 1st April, 2018.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010, Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.

[BP12] Joan Boyar and René Peralta. A small depth-16 circuit for the AES S-box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012, Proceedings*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012.

[Can05a] David Canright. A very compact Rijndael S-box. Technical report, Naval Postgraduate School Technical Report: NPS-MA-05-001, 2005.

[Can05b] David Canright. A very compact S-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndaels: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[Fen89] Gui Liang Feng. A VLSI architecture for fast inversion in GF(2^m). *IEEE Trans. Computers*, 38(10):1383–1386, 1989.

[FIP01] PUB 197 FIPS. Specification for the advanced encryption standard (AES). National Institute of Standards and Technology, US Department of Commerce„ November 2001.

[FWR05] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *IEE Proceedings - Information Security*, 152:13–20(7), October 2005.

[GM16] Shay Gueron and Sanu Mathew. Hardware implementation of AES using area-optimal polynomials for composite-field representation GF($(2^4)^2$) of GF(2^8). In Paolo Montuschi, Michael J. Schulte, Javier Hormigo, Stuart F. Oberman, and Nathalie Revol, editors, *23nd IEEE Symposium on Computer Arithmetic, ARITH 2016, Silicon Valley, CA, USA, July 10-13, 2016, Proceedings*, pages 112–117. IEEE Computer Society, 2016.

[IT88] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and computation*, 78(3):171–177, 1988.

[JKL10] Yong-Sung Jeon, Young-Jin Kim, and Dong-Ho Lee. A compact memory-free architecture for the AES algorithm using resource sharing methods. *Journal of Circuits, Systems, and Computers*, 19(5):1109–1130, 2010.

[JMPS17] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-sliding: A generic technique for bit-serial implementations of SPN-based primitives - applications to AES, PRESENT and SKINNY. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 687–707. Springer, 2017.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Sov. Phys.-Dokl. (English translation)*, 7(7):595–596, 1963.

[KR12] Mehran Mozaffari Kermani and Arash Reyhani-Masoleh. Efficient and high-performance parallel hardware architectures for the AES-GCM. *IEEE Trans. Computers*, 61(8):1165–1178, 2012.

[KS98] Çetin Kaya Koç and Berk Sunar. Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields. *IEEE Trans. Computers*, 47(3):353–356, 1998.

[Mas91] E. D. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Linkoping Univ., Linkoping Sweden, 1991.

[MO86] J. L. Massey and J. K. Omura. Computational method and apparatus for finite field arithmetic, 1986. US Patent 4,587,627.

[Nan] NanGate open cell library. http://www.nangate.com/, last accessed on: 1st April, 2018.

[NG00] Carl F. Nielsen and Samuel R. Girgis. WPI 0.5 mm CMOS standard cell library databook. Technical report, April 2000.

[NNI12] Kenta Nekado, Yasuyuki Nogami, and Kengo Iokibe. Very short critical path implementation of AES with direct logic gates. In Goichiro Hanaoka and Toshihiro Yamauchi, editors, *Advances in Information and Computer Security - 7th International Workshop on Security, IWSEC 2012, Fukuoka, Japan, November 7-9, 2012, Proceedings*, volume 7631 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2012.

[NNT+10] Yasuyuki Nogami, Kenta Nekado, Tetsumi Toyota, Naoto Hongo, and Yoshitaka Morikawa. Mixed bases for efficient inversion in $\mathbb{F}_{((2^2)^2)^2}$ and conversion matrices of subbytes of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010, Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 234–247. Springer, 2010.

[Paa94] Christof Paar. *Efficient VLSI architectures for bit parallel computation in Galios fields*. PhD thesis, University of Duisburg-Essen, Germany, 1994.

[Paa96] Christof Paar. A new architecture for a parallel finite field multiplier with low complexity based on composite fields. *IEEE Trans. Computers*, 45(7):856–861, 1996.

[RDJ+01] Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, Vijay Kumar, Josyula R. Rao, and Pankaj Rohatgi. Efficient Rijndael encryption implementation with composite field arithmetic. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 171–184. Springer, 2001.

[RH02] Arash Reyhani-Masoleh and M. Anwar Hasan. A new construction of massey-omura parallel multiplier over GF(2^m). *IEEE Trans. Computers*, 51(5):511–520, 2002.

[RH03] Arash Reyhani-Masoleh and M. Anwar Hasan. Efficient multiplication beyond optimal normal bases. *IEEE Trans. Computers*, 52(4):428–439, 2003.

[RH04] Arash Reyhani-Masoleh and M. Anwar Hasan. Low complexity bit parallel architectures for polynomial basis multiplication over GF(2^m). *IEEE Trans. Computers*, 53(8):945–959, 2004.

[Ric12] Steve Rickman. Logic friday (version 1.1.4)[computer software], 2012.

[RTA18] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. New area record for the AES combined S-box/inverse S-box. In *25th IEEE Symposium on Computer Arithmetic, ARITH 2018, Amherst, MA, USA, June 25-27, Proceedings*, 2018.

[SMTM01] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact Rijndael hardware architecture with S-box optimization. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.

[UHS+15] Rei Ueno, Naofumi Homma, Yukihiro Sugawara, Yasuyuki Nogami, and Takafumi Aoki. Highly efficient GF(2^8) inversion circuit based on redundant GF arithmetic and its application to AES design. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2015.

[UMHA16] Rei Ueno, Sumio Morioka, Naofumi Homma, and Takafumi Aoki. A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths - toward efficient cbc-mode implementation. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 538–558. Springer, 2016.

[VSP17] Andrea Visconti, Chiara Valentina Schiavo, and René Peralta. Improved upper bounds for the expected circuit complexity of dense systems of linear equations over GF(2). Cryptology ePrint Archive, Report 2017/194, 2017. https://eprint.iacr.org/2017/194.

[VTV] VTVT standard cell library. http://www.vtvt.ece.vt.edu/vlsidesign/cell.php, last accessed on: 1st April, 2018.

[WH15] Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[WTS+85] Charles C. Wang, Trieu-Kien Truong, Howard M. Shao, Leslie J. Deutsch, Jim K. Omura, and Irving S. Reed. VLSI architectures for computing multiplications and inverses in GF(2^m). *IEEE Trans. Computers*, 34(8):709–717, 1985.

Appendix A: Proof of Lemma 2

Proof. The subfield inverter is a combinational circuit which can be explained by its truth table. The truth table of the inverter is shown in Table 13, which was calculated for each subfield element and then verified by our multiplication formulations coding in MATLAB®. Then, from the e_0 column, one can represent e_0 as an OR operation ($\vee$) of the corresponding minterms, i.e.,

$$\begin{aligned} e_0 &= m_2 \vee m_3 \vee m_5 \vee m_6 \vee m_7 \vee m_{11} \vee m_{12} \vee m_{15} \\ &= (m_{12} \vee m_5) \vee (m_2 \vee m_3 \vee m_6 \vee m_7) \vee (m_3 \vee m_7 \vee m_{11} \vee m_{15}). \end{aligned} \tag{29}$$

One can simplify the first term of (29) as $m_{12} \vee m_5 = d_0 d_1 d_2' d_3' \vee d_0' d_1 d_2' d_3 = d_1 d_2' (d_0 d_3' \vee d_0' d_3) = d_1 d_2' (d_0 \oplus d_3)$. Also, the second and third terms of (29) can be simplified as $m_2 \vee m_3 \vee m_6 \vee m_7 = d_0' d_2$ and $m_3 \vee m_7 \vee m_{11} \vee m_{15} = d_2 d_3$, respectively. Therefore, $e_0 = d_1 d_2' (d_0 \oplus d_3) \vee d_0' d_2 \vee d_2 d_3 = d_1 d_2' (d_0 \oplus d_3) \vee d_2 (d_0' \vee d_3)$.

If the input is changed from $D = (d_0 d_1 d_2 d_3) = \sum_{i=0}^{3} d_i \beta^{2^i}$ to its left cyclic shift $D' = D^{2^{-1}} = \sum_{i=0}^{3} d_i \beta^{2^{i-1}} = (d_1 d_2 d_3 d_0)$, then its inverse at the output will be changed from $E = (e_0 e_1 e_2 e_3) = D^{-1}$ to $E' = (D')^{-1} = (D^{2^{-1}})^{-1} = (D^{-1})^{2^{-1}} = (e_1 e_2 e_3 e_0)$. In other words, the cyclic shifts at the inverse input will result in the same cyclic shifts at its output. Let us denote the 0th coordinate of $E = (e_0 e_1 e_2 e_3)$ as the function $e_0 = I(d_0 d_1 d_2 d_3)$ in terms of its input $D = (d_0 d_1 d_2 d_3)$ (see the formulation presented in Lemma 2 for the details of the Boolean function I). Then, the bit 0 of $E' = (e_1 e_2 e_3 e_0)$, i.e., e_1, is obtained when the input is $D' = (d_1 d_2 d_3 d_0)$. Therefore, one can find $e_1 = I(d_1 d_2 d_3 d_0)$ and so other coordinates of $E = D^{-1}$, say e_i, for $1 \leq i \leq 3$, can be found similarly by adding (i modulo 4) with D, i.e., $e_i = I(d_i d_{i+1} d_{i+2} d_{i+3})$. This is similar to to the idea proposed in [MO86, WTS+85, RH02] for multiplication operation.

Replacing $D = (d_0 d_1 d_2 d_3)$ by $D = (d_i d_{i+1} d_{i+2} d_{i+3})$ concludes the proof.

Table 13: The truth table of the inverter over $GF(2^4)$ generated by the ONB-I.

$d_0 d_1 d_2 d_3$	$e_0 e_1 e_2 e_3$	$d_0 d_1 d_2 d_3$	$e_0 e_1 e_2 e_3$
0000	0000	1000	0010
0001	0100	1001	0111
0010	1000	1010	0101
0011	1110	1011	1100
0100	0001	1100	1011
0101	1010	1101	0110
0110	1101	1110	0011
0111	1001	1111	1111

□

Appendix B: Other formulations for the subfield inverter

The 12 formulations that were considered in order to obtain the best subfield inverter circuits are as follows:

Code# 35: $e_i = (d_{i+1}d'_{i+2}(d_i \oplus d_{i+3})) \vee (d_{i+2}(d'_i \vee d_{i+3}))$.

Code# 36: $e_i = ((d_{i+1}d'_{i+2}(d_i \oplus d_{i+3}))'(d_{i+2}(d_i d'_{i+3})')')'$.

Code# 37: $e_i = ((d_i \vee d_{i+3})(d_i d_{i+3})'(d_{i+1}d'_{i+2})) \vee ((d'_i \vee d_{i+3})d_{i+2})$.

Code# 38: $e_i = (((d_i \vee d_{i+3})(d_i d_{i+3})'(d'_{i+1} \vee d_{i+2})')'((d_i d'_{i+3})'d_{i+2})')'$.

Code# 39: $e_i = (d_i d_{i+1}d'_{i+2}d'_{i+3}) \vee (d'_i d_{i+1}d_{i+3}) \vee ((d'_i \vee d_{i+3})d_{i+2})$.

Code# 40: $e_i = ((d_i d_{i+1}d'_{i+2}d'_{i+3})'(d'_i d_{i+1}d_{i+3})'((d_i d'_{i+3})'d_{i+2})')'$.

Code# 41: $e_i = ((d'_{i+1}d'_{i+2}) \vee (d_i d_{i+2}d'_{i+3}) \vee (d_i d'_{i+2}d_{i+3}) \vee (d'_i d'_{i+2}d'_{i+3}))'$.

Code# 42: $e_i = ((d_{i+1} \vee d_{i+2})' \vee (d'_i \vee d'_{i+2} \vee d_{i+3})' \vee (d'_i \vee d_{i+2} \vee d'_{i+3})' \vee (d_i \vee d_{i+2} \vee d_{i+3})')'$.

Code# 43: $e_i = (d_{i+1} \vee d_{i+2})(d'_i \vee d'_{i+2} \vee d_{i+3})(d'_i \vee d_{i+2} \vee d'_{i+3})(d_i \vee d_{i+2} \vee d_{i+3})$.

Code# 44: $e_i = (d_{i+2} \oplus (d_i d_{i+2}d'_{i+3})) \vee ((d_i \oplus d_{i+3})d_{i+1}d'_{i+2})$.

Code# 45: $e_i = ((d_{i+2} \oplus (d_i d_{i+2}d'_{i+3})')((d_i \oplus d_{i+3})d_{i+1}d'_{i+2})')'$.

Code# 46: $e_i = d_{i+2} \oplus (d_i d_{i+2}d'_{i+3})' \oplus ((d_i \oplus d_{i+3})d_{i+1}d'_{i+2})'$.

Table 14: Synthesis results of different formulations for one bit of the subfield inverter.

Code #	Area		Delay	Power
	μm^2	GE	ns	nW
35	19.76	9.5	0.09	550.7
36 (proposed)	16.12	7.75	0.07	493.24
37	23.4	11.25	0.10	581.8
38	19.24	9.25	0.06	515.9
39	22.88	11	0.11	510.15
40	19.24	9.25	0.08	463.88
41	24.96	12	0.10	575.12
42	19.76	9.5	0.09	440.01
43	23.92	11.5	0.08	514.45
44	21.84	10.5	0.10	590.07
45	18.72	9	0.09	493.52
46	20.28	9.75	0.12	472.28

Appendix C: Improved S-box Architectures

Table 15: Formulations of the improved Ueno S-box (the original S-box is in [UHS+15]).

Stage 1
$d_0 = ((h_1 \oplus h_2) \vee (l_1 \oplus l_2))' \oplus ((h_3 \oplus h_4) \vee (l_3 \oplus l_4))' \oplus (h_2 \vee l_2)' \oplus (h_3 l_3)'$
$d_1 = ((h_1 \oplus h_2) \vee (l_1 \oplus l_2))' \oplus ((h_1 \oplus h_3)(l_1 \oplus l_3))' \oplus (h_3 \vee l_3)' \oplus (h_4 \vee l_4)'$
$d_2 = ((h_1 \oplus h_3) \vee (l_1 \oplus l_3))' \oplus ((h_1 \oplus h_4)(l_1 \oplus l_4))' \oplus ((h_2 \oplus h_3) \vee (l_2 \oplus l_3))' \oplus (h_4 \vee l_4)'$
$d_3 = ((h_1 \oplus h_4) \vee (l_1 \oplus l_4))' \oplus ((h_2 \oplus h_3) \vee (l_2 \oplus l_3))' \oplus ((h_2 \oplus h_4)(l_2 \oplus l_4))' \oplus (h_1 \vee l_1)'$
$d_4 = ((h_2 \oplus h_4) \vee (l_2 \oplus l_4))' \oplus ((h_3 \oplus h_4) \vee (l_3 \oplus l_4))' \oplus (h_1 \vee l_1)' \oplus (h_2 l_2)'$
Stage 2
$e_0 = ((d_1 \vee d_4)' \vee (d_2 \vee d_3)')'$
$e_1 = ((d_4'(d_1 \oplus d_2))'(d_0 d_4 (d_2' d_3')')')'$
$e_2 = ((d_3'(d_2 \oplus d_4))'(d_0 d_3 (d_1' d_4')')')'$
$e_3 = ((d_2'(d_1 \oplus d_3))'(d_0 d_2 (d_1' d_4')')')'$
$e_4 = ((d_1'(d_3 \oplus d_4))'(d_0 d_1 (d_2' d_3')')')'$
Stage 3
$h_0^o = ((l_1 \oplus l_4)(e_1 \oplus e_4))' \oplus ((l_2 \oplus l_3)(e_2 \oplus e_3))'$
$h_1^o = (l_1(e_0 \oplus e_1))' \oplus ((l_2 \oplus l_4)(e_2 \oplus e_4))'$
$h_2^o = (l_2(e_0 \oplus e_2))' \oplus ((l_3 \oplus l_4)(e_3 \oplus e_4))'$
$h_3^o = (l_3(e_0 \oplus e_3))' \oplus ((l_1 \oplus l_2)(e_1 \oplus e_2))'$
$h_4^o = (l_4(e_0 \oplus e_4))' \oplus ((l_1 \oplus l_3)(e_1 \oplus e_3))'$
$l_0^o = ((h_1 \oplus h_4)(e_1 \oplus e_4))' \oplus ((h_2 \oplus h_3)(e_2 \oplus e_3))'$
$l_1^o = (h_1(e_0 \oplus e_1))' \oplus ((h_2 \oplus h_4)(e_2 \oplus e_4))'$
$l_2^o = (h_2(e_0 \oplus e_2))' \oplus ((h_3 \oplus h_4)(e_3 \oplus e_4))'$
$l_3^o = (h_3(e_0 \oplus e_3))' \oplus ((h_1 \oplus h_2)(e_1 \oplus e_2))'$
$l_4^o = (h_4(e_0 \oplus e_4))' \oplus ((h_1 \oplus h_3)(e_1 \oplus e_3))'$

Table 16: Formulations of the improved Boyar-Dp16-2 S-box. The original S-box is found from [BFP17] with formulations from [Boy16]. In this table, we use lower case letters, e.g., t_{44}, to represent the complemented signals as compared to the original signals which are represented with capital letters, e.g., $T_{44} = (t_{44})'$.

$T_1 \leftarrow U_6 \oplus U_4$ $1D_X$	$T_{59} \leftarrow T_{46} \odot t_{45}$ $5D_X$	$t_{101} \leftarrow (T_{86}T_7)'$ $10D_X + 2D_{ND}$
$T_2 \leftarrow U_3 \oplus U_0$ $1D_X$	$T_{60} \leftarrow T_{48} \oplus T_{42}$ $5D_X + 1D_{ND}$	$t_{102} \leftarrow (T_{81}T_9)'$ $9D_X + 2D_{ND}$
$T_3 \leftarrow U_1 \oplus U_2$ $1D_X$	$T_{61} \leftarrow T_{51} \odot t_{50}$ $5D_X$	$t_{103} \leftarrow (T_{80}T_{22})'$ D_3
$T_4 \leftarrow U_7 \oplus T_3$ $2D_X$	$T_{62} \leftarrow T_{53} \oplus T_{58}$ $5D_X + 1D_{ND}$	$t_{104} \leftarrow (T_{85}T_2)'$ D_4
$T_5 \leftarrow T_1 \oplus T_2$ $2D_X$	$T_{63} \leftarrow T_{59} \oplus T_{56}$ $6D_X$	$t_{105} \leftarrow (T_{88}T_{41})'$ $11D_X + 2D_{ND}$
$T_6 \leftarrow U_1 \oplus U_5$ $1D_X$	$T_{64} \leftarrow T_{60} \oplus T_{58}$ $6D_X + 1D_{ND}$	$t_{106} \leftarrow (T_{84}T_{16})'$ $10D_X + 2D_{ND}$
$T_7 \leftarrow U_0 \oplus U_6$ $1D_X$	$T_{65} \leftarrow T_{61} \oplus T_{56}$ $6D_X$	$T_{107} \leftarrow t_{104} \oplus t_{105}$ $12D_X + 2D_{ND}$
$T_8 \leftarrow T_1 \oplus T_6$ $2D_X$	$T_{66} \leftarrow T_{62} \oplus T_{43}$ $6D_X + 1D_{ND}$	$T_{108} \leftarrow t_{93} \oplus t_{99}$ $10D_X + 2D_{ND}$
$T_9 \leftarrow U_6 \oplus T_4$ $3D_X$	$t_{67} \leftarrow T_{65} \odot T_{66}$ $7D_X + 1D_{ND}$	$T_{109} \leftarrow t_{96} \odot T_{107}$ $13D_X + 2D_{ND}$
$T_{10} \leftarrow U_3 \oplus T_4$ $3D_X$	$t_{68} \leftarrow (T_{65}T_{63})'$ $6D_X + 1D_{ND}$	$T_{110} \leftarrow t_{98} \odot T_{108}$ $11D_X + 2D_{ND}$
$T_{11} \leftarrow U_7 \oplus T_5$ $3D_X$	$t_{69} \leftarrow T_{64} \oplus t_{68}$ $7D_X + 1D_{ND}$	$T_{111} \leftarrow t_{91} \oplus t_{101}$ $11D_X + 2D_{ND}$
$T_{12} \leftarrow T_5 \oplus T_6$ $3D_X$	$t_{70} \leftarrow T_{63} \odot T_{64}$ $7D_X + 1D_{ND}$	$T_{112} \leftarrow t_{89} \oplus t_{92}$ $11D_X + 2D_{ND}$
$T_{13} \leftarrow U_2 \oplus U_5$ $1D_X$	$t_{71} \leftarrow T_{66} \oplus t_{68}$ $7D_X + 1D_{ND}$	$T_{113} \leftarrow T_{107} \oplus T_{112}$ $13D_X + 2D_{ND}$
$T_{14} \leftarrow T_3 \oplus T_5$ $3D_X$	$T_{72} \leftarrow (t_{71} \vee t_{70})'$ D_0 [a]	$T_{114} \leftarrow t_{90} \odot T_{110}$ $12D_X + 2D_{ND}$
$T_{15} \leftarrow U_5 \oplus T_7$ $2D_X$	$T_{73} \leftarrow (t_{69} \vee t_{67})'$ D_0	$T_{115} \leftarrow t_{89} \oplus t_{95}$ $11D_X + 2D_{ND}$
$T_{16} \leftarrow U_0 \oplus U_5$ $1D_X$	$t_{74} \leftarrow (T_{63}T_{66})'$ $6D_X + 2D_{ND}$	$T_{116} \leftarrow t_{94} \oplus t_{102}$ $10D_X + 2D_{ND}$
$T_{17} \leftarrow U_7 \oplus T_8$ $3D_X$	$T_{75} \leftarrow (t_{70} \vee t_{74})'$ D_0	$T_{117} \leftarrow t_{97} \oplus t_{103}$ $11D_X + 2D_{ND}$
$T_{18} \leftarrow U_6 \oplus U_5$ $1D_X$	$T_{76} \leftarrow t_{70} \oplus t_{68}$ $8D_X + 1D_{ND}$	$T_{118} \leftarrow t_{91} \odot T_{114}$ $13D_X + 2D_{ND}$
$T_{19} \leftarrow T_2 \oplus T_{18}$ $2D_X$	$t_{77} \leftarrow (T_{64}T_{65})'$ $6D_X + 2D_{ND}$	$T_{119} \leftarrow T_{111} \oplus T_{117}$ $12D_X + 2D_{ND}$
$T_{20} \leftarrow T_4 \oplus T_{15}$ $3D_X$	$T_{78} \leftarrow (t_{67} \vee t_{77})'$ D_0	$T_{120} \leftarrow t_{100} \odot T_{108}$ $11D_X + 2D_{ND}$
$T_{21} \leftarrow T_1 \oplus T_{13}$ $2D_X$	$T_{79} \leftarrow t_{67} \oplus t_{68}$ $8D_X + 1D_{ND}$	$T_{121} \leftarrow t_{92} \oplus t_{95}$ $11D_X + 2D_{ND}$
$T_{22} \leftarrow U_0 \oplus T_4$ $3D_X$	$T_{80} \leftarrow T_{64} \oplus T_{72}$ D_1 [b]	$T_{122} \leftarrow T_{110} \oplus T_{121}$ $12D_X + 2D_{ND}$
$T_{39} \leftarrow T_{21} \oplus T_5$ $3D_X$	$T_{81} \leftarrow T_{75} \oplus T_{76}$ $9D_X + 1D_{ND}$	$T_{123} \leftarrow t_{106} \odot T_{119}$ $13D_X + 2D_{ND}$
$T_{40} \leftarrow T_{21} \oplus T_7$ $3D_X$	$T_{82} \leftarrow T_{66} \oplus T_{73}$ D_1	$T_{124} \leftarrow t_{104} \odot T_{115}$ $12D_X + 2D_{ND}$
$T_{41} \leftarrow T_7 \oplus T_{19}$ $3D_X$	$T_{83} \leftarrow T_{78} \oplus T_{79}$ $9D_X + 1D_{ND}$	$T_{125} \leftarrow T_{111} \oplus T_{116}$ $12D_X + 2D_{ND}$
$T_{42} \leftarrow T_{16} \oplus T_{14}$ $4D_X$	$T_{84} \leftarrow T_{83} \oplus T_{81}$ $10D_X + 1D_{ND}$	$S_0 \leftarrow T_{109} \oplus T_{122}$ $14D_X + 2D_{ND}$
$T_{43} \leftarrow T_{22} \oplus T_{17}$ $4D_X$	$T_{85} \leftarrow T_{80} \oplus T_{82}$ D_2 [d]	$S_2 \leftarrow T_{123} \odot T_{124}$ $14D_X + 2D_{ND}$
$t_{44} \leftarrow (T_{19}T_5)'$ $2D_X + 1D_{ND}$	$T_{86} \leftarrow T_{80} \oplus T_{81}$ $10D_X + 1D_{ND}$	$T_{128} \leftarrow t_{94} \odot T_{107}$ $13D_X + 2D_{ND}$
$t_{45} \leftarrow (T_{20}T_{11})'$ $3D_X + 1D_{ND}$	$T_{87} \leftarrow T_{82} \oplus T_{83}$ $10D_X + 1D_{ND}$	$S_3 \leftarrow T_{113} \oplus T_{114}$ $14D_X + 2D_{ND}$
$T_{46} \leftarrow T_{12} \odot t_{44}$ $4D_X$	$T_{88} \leftarrow T_{85} \oplus T_{84}$ $11D_X + 1D_{ND}$	$S_4 \leftarrow T_{118} \oplus T_{128}$ $14D_X + 2D_{ND}$
$t_{47} \leftarrow (T_{10}U_7)'$ $3D_X + 1D_{ND}$	$t_{89} \leftarrow (T_{87}T_5)'$ $10D_X + 2D_{ND}$	$T_{131} \leftarrow t_{93} \oplus t_{101}$ $11D_X + 2D_{ND}$
$T_{48} \leftarrow t_{47} \oplus t_{44}$ $4D_X + 1D_{ND}$	$t_{90} \leftarrow (T_{83}T_{11})'$ $9D_X + 2D_{ND}$	$T_{132} \leftarrow T_{112} \oplus T_{120}$ $12D_X + 2D_{ND}$
$t_{49} \leftarrow (T_7T_{21})'$ $2D_X + 1D_{ND}$	$t_{91} \leftarrow (T_{82}U_7)'$ D_3 [c]	$S_7 \leftarrow T_{113} \odot T_{125}$ $14D_X + 2D_{ND}$
$t_{50} \leftarrow (T_9T_4)'$ $3D_X + 1D_{ND}$	$t_{92} \leftarrow (T_{86}T_{21})'$ $10D_X + 2D_{ND}$	$T_{134} \leftarrow t_{97} \odot T_{116}$ $11D_X + 2D_{ND}$
$T_{51} \leftarrow T_{40} \odot t_{49}$ $4D_X$	$t_{93} \leftarrow (T_{81}T_4)'$ $9D_X + 2D_{ND}$	$T_{135} \leftarrow T_{131} \oplus T_{134}$ $12D_X + 2D_{ND}$
$t_{52} \leftarrow (T_{22}T_{17})'$ $3D_X + 1D_{ND}$	$t_{94} \leftarrow (T_{80}T_{17})'$ D_3	$T_{136} \leftarrow t_{93} \odot T_{115}$ $12D_X + 2D_{ND}$
$T_{53} \leftarrow t_{52} \oplus t_{49}$ $4D_X + 1D_{ND}$	$t_{95} \leftarrow (T_{85}T_8)'$ D_4 [e]	$S_6 \leftarrow T_{109} \odot T_{135}$ $14D_X + 2D_{ND}$
$t_{54} \leftarrow (T_2T_8)'$ $2D_X + 1D_{ND}$	$t_{96} \leftarrow (T_{88}T_{39})'$ $11D_X + 2D_{ND}$	$T_{138} \leftarrow T_{119} \oplus T_{132}$ $13D_X + 2D_{ND}$
$t_{55} \leftarrow (T_{41}T_{39})'$ $3D_X + 1D_{ND}$	$t_{97} \leftarrow (T_{84}T_{14})'$ $10D_X + 2D_{ND}$	$S_5 \leftarrow T_{109} \oplus T_{138}$ $14D_X + 2D_{ND}$
$T_{56} \leftarrow t_{55} \oplus t_{54}$ $4D_X + 1D_{ND}$	$t_{98} \leftarrow (T_{87}T_{19})'$ $10D_X + 2D_{ND}$	$T_{140} \leftarrow T_{114} \oplus T_{136}$ $13D_X + 2D_{ND}$
$t_{57} \leftarrow (T_{16}T_{14})'$ $3D_X + 1D_{ND}$	$t_{99} \leftarrow (T_{83}T_{20})'$ $9D_X + 2D_{ND}$	$S_1 \leftarrow T_{109} \odot T_{140}$ $14D_X + 2D_{ND}$
$T_{58} \leftarrow t_{57} \oplus t_{54}$ $4D_X + 1D_{ND}$	$t_{100} \leftarrow (T_{82}T_{10})'$ D_3	

[a] $D_0 = 7D_X + 1D_{ND} + 1D_{NR}$	[d] D_2=$9D_X + 1D_{ND} + 1D_{NR}$	[e] $D_4 = 9D_X + 2D_{ND} + 1D_{NR}$
[b] $D_1 = 8D_X + 1D_{ND} + 1D_{NR}$	[c] $D_3 = 8D_X + 2D_{ND} + 1D_{NR}$	