IEICE TRANS. INF. & SYST., VOL.E97–D, NO.9 SEPTEMBER 2014

PAPER Special Section on Multiple-Valued Logic and VLSI Computing

Formal Design of Arithmetic Circuits over Galois Fields Based on Normal Basis Representations*

Kotaro OKAMOTO[†a)], ***Nonmember***, **Naofumi HOMMA**[†], ***and*** **Takafumi AOKI**[†], ***Members***

SUMMARY This paper presents a graph-based approach to designing arithmetic circuits over Galois fields (GFs) using normal basis representations. The proposed method is based on a graph-based circuit description called *Galois-field Arithmetic Circuit Graph* (*GF-ACG*). First, we extend GF-ACG representation to describe GFs defined by normal basis in addition to polynomial basis. We then apply the extended design method to Massey-Omura parallel multipliers which are well known as typical multipliers based on normal basis. We present the formal description of the multipliers in a hierarchical manner and show that the verification time can be greatly reduced in comparison with those of the conventional techniques. In addition, we design GF exponentiation circuits consisting of the Massey-Omura parallel multipliers and an inversion circuit over composite field $GF(((2^2)^2)^2)$ in order to demonstrate the advantages of normal-basis circuits over polynomial-basis ones.

key words: *arithmetic circuits, formal verification, normal basis, computer algebra*

1. Introduction

Applications of arithmetic operations over Galois fields (GFs) have been rapidly increasing owing to the high demands of reliable/secure communications and transactions using ECC (error correction code) and cryptographic operations [2]. These operations are often implemented on hardware in recent embedded devices, such as smart cards and cell phones, and the performance and dependability of arithmetic circuits have a significant impact on the entire processors. Currently, many hardware algorithms on GF arithmetic have been devised and some of such algorithms can be implemented by a multiple-valued logic more efficiently than by the binary logic.

On the other hand, most of such arithmetic circuits are designed at the logic level by researchers who had trained in a particular way to understand GF arithmetic. The conventional Hardware Description Languages (HDLs) do not have high-level arithmetic data structures, arithmetic operations and formulae over GFs. This sometimes requires us to describe structural details of arithmetic circuits by hand at the lowest level of abstraction (i.e., AND-XOR expressions) in a flattened manner. In addition, the functional verification using the conventional logic simulation is quite time-consuming since these operations are usually performed with more-than 64-bit operands. The test pattern generation is also difficult since it varies with the irreducible polynomial even for the same operation (e.g., multiplication). In earlier related research, the formal verification of arithmetic circuits was primarily performed based on Decision Diagrams (DDs) and Binary Moment Diagrams (BMDs) [3]–[5]. However, conventional approaches are basically limited not only to binary arithmetic over integers, but also to rather small circuits. Although Binary Decision Diagrams (BDDs) can also be applied to GF arithmetic, BDDs are known to be ineffective for XOR-based logic circuits**. There is a decision diagram specified for Galois fields based on the decomposition of multiple-valued functions [6], but it is difficult to handle practical fields such as $GF(2^{16})$ and $GF(2^{32})$ and apply it to the formal verification. $GF(2^m)$ arithmetic circuits were successfully verified in a few previous studies [7], [8]; however, the application of the verification method appears to be limited to the specific $GF(2^m)$ circuits whose reference (i.e., equivalent) circuits can be prepared in advance.

To address the above problems, a formal design and verification method of arithmetic circuits over GFs was proposed [9] and [10]. The proposed idea is to use a high-level mathematical graph associated with variables and arithmetic formulae over GFs, which is called *Galois-field Arithmetic Circuit Graph*: *GF-ACG*. Using GF-ACGs, we can describe any GF arithmetic circuit in a hierarchical manner as a combination of arithmetic sub-circuits (graphs). Such description is formally verified by checking for every sub-circuit whether the function is obtained from the internal structure. The equivalence checking can be performed by formula manipulations based on a polynomial reduction algorithm using Gröbner Basis [11], which makes it possible to verify practical arithmetic circuits in a short time. On the other hand, the previous works in [9] and [10] were limited to GF arithmetic represented by polynomial basis.

This paper presents an extension of GF-ACGs to designing arithmetic circuits over GFs represented by normal basis (NB). The space and time complexities of GF arithmetic operations heavily depend on how the field elements are represented. The NB representation is useful for designing GF arithmetic circuits such as inversion circuits and exponentiation circuits since the squaring operation based on NB representation is performed only by wiring. In this paper, we first present the extension of GF-ACGs to design and

Manuscript received November 29, 2013.

[†]The authors are with the Graduate School of Information Sciences, Tohoku University, Sendai-shi, 980–8579 Japan.

*A preliminary version of this paper appeared in the IEEE 43rd International Symposium on Multiple-Valued Logic (ISMVL 2013) [1].

a) E-mail: okamoto@aoki.ecei.tohoku.ac.jp

DOI: 10.1587/transinf.2013LOP0012

**GF arithmetic operations mostly consist of XOR and AND gates.

verify GFs represented by NB in addition to PB, and apply the extended GF-ACG to the formal description of Massey-Omura multipliers. The advantage of the proposed method is evaluated through the experimental verification of the designed multipliers. We also design a set of exponentiation circuits using the designed multipliers and a multiplicative inversion circuit over $GF(((2^2)^2)^2)$ in order to evaluate the performance of NB-based circuits in comparison with that of PB-based ones. In addition, we further extend GF-ACG to composite fields based on NB and apply it to the formal design and verification of a multiplicative inversion circuit. Note that the preliminary version [1] studied only for prime and extension fields.

2. Galois-Field Arithmetic Circuit Graph

This section briefly describes the graph-based representation of GF arithmetic circuits, where the graphs are referred to as GF Arithmetic Circuit Graphs (GF-ACGs).

Figure 1 shows an overview of a GF-ACG. A GF-ACG G is defined as $(\boldsymbol{N}, \boldsymbol{E})$, where $\boldsymbol{N}$ is a set of nodes, and $\boldsymbol{E}$ is a set of directed edges. The node represents an arithmetic circuit by its functional assertion and internal structure. The directed edge represents the flow of data between nodes, and defines the data dependency. We assume that every node has at least one edge connection.

A node n ($\in \boldsymbol{N}$) is defined by $(\boldsymbol{F}, G')$, where $\boldsymbol{F}$ is the functional assertion given as a set of equations over GFs (*GF equations*) and G' is the internal structure given as a smaller GF-ACG. A node at the lowest level of abstraction, which does not have its internal structure, is described as $(\boldsymbol{F}, nil)$. A functional assertion is represented as a relation $E_l = E_r$, where E_l and E_r are the output and input expressions, respectively, and each expression is given by variables, constants or combinations of the two or more expressions connected by arithmetic operations $+$, $-$, $\times$, and $/$.

A directed edge e ($\in \boldsymbol{E}$) is defined as $(src, dest, x)$, where *src* and *dest* represent the start and end node, respectively, and x represents the variable indicating an element of GF. If either *src* or *dest* is *nil*, its directed edge represents an external input or output for the given GF-ACG. Each variable is associated with a Galois field. A Galois field GF based on polynomial basis (PB) is defined as $(\boldsymbol{B}, \boldsymbol{C}, IP)$, where $\boldsymbol{B}$ is the basis, $\boldsymbol{C}$ is the coefficient vector, and IP is the irreducible polynomial. More precisely, $\boldsymbol{B}$, $\boldsymbol{C}$, and IP are given as

$$\boldsymbol{B} = \left(\beta^{m-1}, \beta^{m-2}, \cdots, \beta^0\right), \tag{1}$$

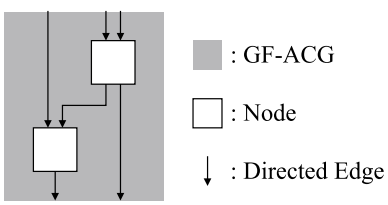

Fig. 1 Galois-field arithmetic circuit graph.

$$\boldsymbol{C} = (C_{m-1}, C_{m-2}, \cdots, C_0), \tag{2}$$

$$IP = \beta^m + c_{m-1}\beta^{m-1} + \cdots + c_0\beta^0, \tag{3}$$

where β is the indeterminate element, C_i is the coefficient set of degree i, m is the degree of field extension, and c_i is the element of the coefficient set C_i. $IP = nil$ if the GF is a prime field. Thus, the above description can handle both prime and extension fields. Let h ($0 \le h \le m-1$) and l ($0 \le l \le h$) be the most and least significant degrees, respectively. A variable is represented as $x = (GF, (h, l))$, where the tuple (h, l) is called the degree range. Using the above notation, we can handle a specific variable x_i of degree i.

A variable is represented as an expression at a lower level of abstraction. Let x be a variable and x_i ($l \le i \le h$) be a lower-level variable. We have two types of decomposition nodes whose functions are given as

$$x_h^{(e)} + x_{h-1}^{(e)} + \cdots + x_l^{(e)} = x, \tag{4}$$

$$x_h^{(p)}\beta^h + x_{h-1}^{(p)}\beta^{h-1} + \cdots + x_l^{(p)}\beta^l = x. \tag{5}$$

Equation (4) indicates that $x \in GF(p^m)$ is divided into a number of variables of degree i (i.e., $x_i(l \le i \le h) \in GF(p^m)$). On the other hand, Eq. (5) indicates that $x \in GF(p^m)$ is divided into a number of variables over the prime field (i.e., $x_i(l \le i \le h) \in GF(p)$). We also have two types of composition nodes given as inverse relations between the above inputs and outputs. Using the decomposition/composition nodes, we can change the level of abstraction in edge representation. Note here that these nodes are implemented by wiring and have no internal structures.

The above GF-ACG can be used also for representing any logic circuit. A logic variable is considered as a variable over the GF whose coefficient set is limited to the zero element "0" and the unit element "1". Any logical operation can be represented with pseudo logic equations. For example, the functions of AND and XOR circuits are given as

$$and(a, b) = ab, \tag{6}$$

$$xor(a, b) = a + b - 2ab, \tag{7}$$

respectively. Note that the idempotent law is considered as one of functional assertions in the corresponding node (i.e., $a = a^2$ and $b = b^2$).

Thus, GF-ACG can represent any arithmetic circuit over GF represented by PB and any logic circuit. The arithmetic circuits given by GF-ACGs are verified by a formal verification method using Gröbner Basis and a polynomial reduction technique. (See [9] for the detailed verification procedure.)

3. Extension to Normal Basis Presentation

This section presents an extension of GF-ACGs to arithmetic circuits over GFs represented by normal basis (NB).

Let α be the indeterminate element β raised to the n-th power (i.e., $\alpha = \beta^n$), where the elements $\alpha^{q^{m-1}}, \alpha^{q^{m-2}}, \cdots, \alpha^{q^0}$ are linearly independent over $GF(q)$ [12], [13]. A normal

basis of $GF(q^m)$ is given by $(\alpha^{q^{m-1}}, \alpha^{q^{m-2}}, \cdots, \alpha^{q^0})$, where q is a power of prime number. It is well known that there is a normal basis for any positive integer m. Any field element is represented as a linear combination of the elements in a normal basis. For example, consider the finite field $GF(2^3)$ generated by the irreducible polynomial $\beta^3 + \beta + 1$. If we choose $\alpha = \beta^3$, we can say that $(\alpha^4, \alpha^2, \alpha)$ is a normal basis.

In order to handle NB representation, we introduce the expression of basis $\boldsymbol{B}$ by α instead of β. More precisely, a Galois field $GF(= (\boldsymbol{B}, \boldsymbol{C}, IP))$ based on NB is defined by

$$\boldsymbol{B} = \left(\alpha^{q^{m-1}}, \alpha^{q^{m-2}}, \cdots, \alpha^{q^0}\right), \tag{8}$$

$$\boldsymbol{C} = (C_{m-1}, C_{m-2}, \cdots, C_0), \tag{9}$$

$$IP = \beta^m + c_{m-1}\beta^{m-1} + \cdots + c_0\beta^0. \tag{10}$$

According to the extension, the expression of the second decomposition node given by Eq. (5) is also extended to

$$x_h^{(p)}\alpha^{q^h} + x_{h-1}^{(p)}\alpha^{q^{h-1}} + \cdots + x_l^{(p)}\alpha^{q^l} = x. \tag{11}$$

The corresponding composition node, as is the case for PB, is given as the inverse relation between the above input and output. Using the decomposition and composition nodes, we can also change the level of abstraction in any edge representation based on NB. Note here that we do not need to change the representation of any logic circuit even if we use the extended GF-ACG. As a result, we can apply the extended GF-ACG to any arithmetic circuit over GFs represented by NB.

The formal verification method in [9] is also extended due to the extended description. Figure 2 shows the extended algorithm, where *GroebnerBasis*($\boldsymbol{P}$) indicates Buchberger's algorithm to obtain a Gröbner Basis $\boldsymbol{GB}$ from a set of polynomials $\boldsymbol{P}$. Given a functional assertion f and internal structure G, $\boldsymbol{P}$ is generated from functional assertions (i.e., $\boldsymbol{F}$) in the internal structure. In the extended algorithm, we minimize the degree of $\boldsymbol{F}$ by *Minimization*($\boldsymbol{F}$) if the $\boldsymbol{F}$ includes the terms of the indeterminate elements. $\boldsymbol{GB}$ is then obtained from *GroebnerBasis*($\boldsymbol{P}$).

Buchberger's algorithm sometimes takes a long time and requires large memory space. The degree of $\boldsymbol{F}$ is a major factor to increase its computation time since the number of polynomial reductions in the algorithm is dependent on the degree. As a result, the above minimization significantly reduces the computation time to generate $\boldsymbol{GB}$. If the normal form of f with respect to $\boldsymbol{GB}$ is equal to zero, f is a member of the ideal from $\boldsymbol{P}$. This means that the functional assertion can be realized with the internal structure. Therefore, this verification algorithm returns *true*.

Input:	Functional assertion f Internal structure $G = (\boldsymbol{N}, \boldsymbol{E})$
Output:	Verification result $r \in \{true, false\}$

```
1:  Function FormulaEvaluation(f, G)
2:      P := ∅
3:      for each (F, G') ∈ N
4:          if F includes α^k
5:              F = Minimization(F)
6:          end if
7:          P := P ∪ F
8:      end for
9:      GB := GroebnerBasis(P)
10:     if NF_GB(f) = 0
11:         r := true
12:     else
13:         r := false
14:     end if
15:     return r
16: end
```

Fig. 2 Extended verification algorithm.

4. Design and Verification of Massey-Omura Parallel Multipliers

This section presents the application of the extended GF-ACG to the design and verification of parallel multipliers based on NB representation.

The Massey-Omura parallel multiplier [14] is a 2-input 1-output parallel multiplier over $GF(2^m)$ represented by NB, which has an efficient structure reducing the redundancy of a well-known Massey-Omura multiplier [15]. Let a and $b \in GF(2^m)$ be the inputs and let $c \in GF(2^m)$ be the output. Let $a_i^{(p)}$ and $b_i^{(p)}$ be the i-th elements of decomposed inputs (i.e., $a = \sum_{i=0}^{m-1} a_i^{(p)}\alpha^{2^i}$ and $b = \sum_{i=0}^{m-1} b_i^{(p)}\alpha^{2^i}$). The operation of Massey-Omura parallel multiplier, whose function is given as $c = a \times b$, is originally represented by

$$c = \begin{cases} \displaystyle\sum_{i=0}^{m-1} a_i^{(p)} b_i^{(p)} \alpha^{2^{i+1}} + \sum_{i=0}^{m-1}\sum_{j=1}^{v} x_{i,j}\gamma_j^{2^i}, & \textit{for } m \textit{ odd} \\ \displaystyle\sum_{i=0}^{m-1} a_i^{(p)} b_i^{(p)} \alpha^{2^{i+1}} + \sum_{i=0}^{m-1}\sum_{j=1}^{v-1} x_{i,j}\gamma_j^{2^i} + \sum_{i=0}^{v-1} x_{i,v}\gamma_v^{2^i}, & \textit{for } m \textit{ even,} \end{cases} \tag{12}$$

where $x_{i,j} = a_i^{(p)} b_{i+j}^{(p)} + a_{i+j}^{(p)} b_i^{(p)}$ $(0 \le i \le m-1,\ 1 \le j \le v)$, $\gamma_j = \alpha^{1+2^j}$ and $v = \lfloor \frac{m}{2} \rfloor$.

For the GF-ACG design, we derive a hierarchical description from the above flattened description. First, Eq. (12) is simplified as follows:

$$\begin{aligned} c &= \sum_{i=0}^{m-1} a_i^{(p)} b_i^{(p)} \alpha^{2^{i+1}} + \sum_{i=0}^{m-1}\sum_{j=1}^{m-1} a_i^{(p)} b_{i+v}^{(p)} \gamma_j^{2^i} \\ &= \sum_{i=0}^{m-1} \left(a_i^{(p)} b_{i+0}^{(p)} \alpha^{2^i + 2^{i+0}} + \sum_{j=1}^{m-1} a_i^{(p)} b_{i+j}^{(p)} \alpha^{2^i + 2^{i+j}} \right) \\ &= \sum_{i=0}^{m-1} \left(\sum_{j=0}^{m-1} \left(a_i^{(p)} \times b_{i+j}^{(p)} \right) \alpha^{2^i + 2^{i+j}} \right) \\ &= \sum_{i=0}^{m-1} \left(\sum_{j=0}^{m-1} \left(a_i^{(p)} \times b_j^{(p)} \right) \alpha^{2^i + 2^j} \right). \end{aligned} \tag{13}$$

Here, the terms in the parenthesis are given as

$$\sum_{j=0}^{m-1}\left(a_i^{(p)} \times b_j^{(p)}\right)\alpha^{2^i+2^j} = w_i. \tag{14}$$

The operation of Massey-Omura parallel multiplier is finally represented by the following two equations:

$$\sum_{i=0}^{m-1} w_i = a \times b, \tag{15}$$

$$c = \sum_{i=0}^{m-1} w_i. \tag{16}$$

This suggests that at the 2^{nd}-level of the hierarchy, a Massey-Omura parallel multiplier is represented by a GF-ACG with two nodes performing the operations corresponding to Eqs. (15) and (16), respectively.

Equation (15) is then represented by

$$\begin{aligned} w_i &= a_i^{(p)}\alpha^{2^i} \times \sum_{j=0}^{m-1} b_j^{(p)}\alpha^{2^j} \\ &= a_i^{(e)} \times b, \end{aligned} \tag{17}$$

where $a_i^{(e)}$ is the i-th element obtained by the other decomposition of a ($a = \sum_{i=0}^{m-1} a_i^{(e)}$) and $b \in GF(2^m)$. This means that the 3^{rd}-level node of Eq. (15) is represented by m nodes performing the operations of Eq. (17). The node of Eq. (16) is represented by m-1 2-input 1-output adders over $GF(2^m)$, and these adders are given by 2-input 1-output adders over $GF(2)$.

For the 4^{th}-level description, let $a_i^{(p)} \times b_j^{(p)} = s_{i,j}$ and $\alpha^{2^i+2^j} = \delta_{i,j}$ in Eq. (14). Using $w_i = \sum_{k=0}^{m-1} w_{i,k}^{(p)}\alpha^{2^k}$ and $\delta_{i,j} = \sum_{k=0}^{m-1} \delta_{i,j,k}\alpha^{2^k}$, we can divide the operation of Eq. (14) (i.e., Eq. (17)) into the following two operations:

$$s_{i,j} = a_i^{(p)} \times b_j^{(p)},\ 0 \le j \le m-1, \tag{18}$$

$$w_{i,k}^{(p)} = \sum_{j=0}^{m-1} s_{i,j}\delta_{i,j,k}. \tag{19}$$

Thus, the node of Eq. (17) can be given by the nodes performing two operations of Eqs. (18) and (19). If the number of δ satisfying $\delta_{i,j,k} = 1$ is one, Eq. (19) is given as $w_{i,k}^{(p)} = s_{i,j}$. Finally, the node of Eq. (18) can be given by m multipliers over $GF(2)$, and the node of Eq. (19) is given by some 2-input 1-output adders over $GF(2)$.

Figure 3 shows the GF-ACGs for the Massey-Omura parallel multiplier over $GF(2^3)$, where the GF-ACGs are represented by five levels of abstraction. The nodes in Figs. 3 (a), (b), (c) and (d) correspond to the shaded parts in Figs. 3 (b), (c), (d) and (e), respectively. Here, "GFA0" and "GFA1" in Figs. 3 (c), (d) correspond to G_{16} and G_{17} in Fig. 3 (e), respectively. Table 1 shows the details of nodes, GFs and variables used in Fig. 3. In this example, α is β raised to the cube (i.e., $\alpha = \beta^3$). Note that the decomposition/composition nodes are not shown in Table 1.

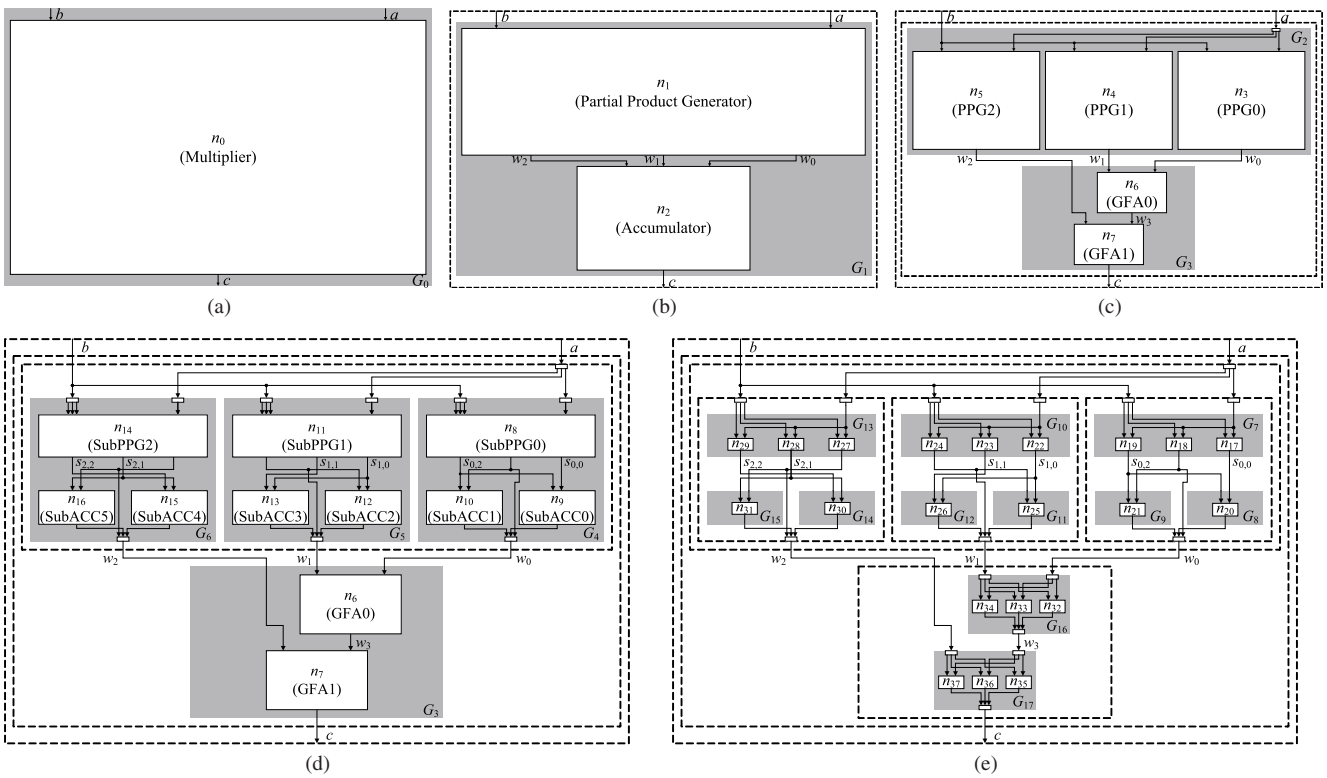

Fig. 3 GF-ACGs for $GF(2^3)$ Massey-Omura parallel multiplier: (a)–(e) GF-ACGs at five levels of abstraction.

Table 1 Nodes, Galois fields, and variables for $GF(2^3)$ Massey-Omura parallel multiplier in Fig. 3.

Nodes

[Multiplier] $n_0 = (\{c = a \times b\}, G_1)$
[Partial Product Generator] $n_1 = (\{\sum_{i=0}^{2} w_i = a \times b\}, G_2)$
[PPG0] $n_3 = (\{w_0 = a_0^{(e)} \times b\}, G_4)$
[SubPPG0]
$n_8 = (\{s_{0,0} = a_0^{(p)} \times b_0^{(p)}, w_{0,0}^{(p)} = a_0^{(p)} \times b_1^{(p)}, s_{0,2} = a_0^{(p)} \times b_2^{(p)}\}, G_7)$
$n_{17} = (\{s_{0,0} = a_0^{(p)} \times b_0^{(p)}\}, nil)$
$n_{18} = (\{w_{0,0}^{(p)} = a_0^{(p)} \times b_1^{(p)}\}, nil)$
$n_{19} = (\{s_{0,2} = a_0^{(p)} \times b_2^{(p)}\}, nil)$
[SubACC0] $n_9 = (\{w_{0,1}^{(p)} = s_{0,0} + s_{0,2}\}, G_8)$
$n_{20} = (\{w_{0,1}^{(p)} = s_{0,0} + s_{0,2}\}, nil)$
[SubACC1] $n_{10} = (\{w_{0,2}^{(p)} = w_{0,0}^{(p)} + s_{0,2}\}, G_9)$
$n_{21} = (\{w_{0,2}^{(p)} = w_{0,0}^{(p)} + s_{0,2}\}, nil)$
[PPG1] $n_4 = (\{w_1 = a_1^{(e)} \times b\}, G_5)$
[SubPPG1]
$n_{11} = (\{s_{1,0} = a_1^{(p)} \times b_0^{(p)}, s_{1,1} = a_1^{(p)} \times b_1^{(p)}, w_{1,1}^{(p)} = a_1^{(p)} \times b_2^{(p)}\}, G_{10})$
$n_{22} = (\{s_{1,0} = a_1^{(p)} \times b_0^{(p)}\}, nil)$
$n_{23} = (\{s_{1,1} = a_1^{(p)} \times b_1^{(p)}\}, nil)$
$n_{24} = (\{w_{1,1}^{(p)} = a_1^{(p)} \times b_2^{(p)}\}, nil)$
[SubACC2] $n_{12} = (\{w_{1,0}^{(p)} = s_{1,0} + w_{1,1}^{(p)}\}, G_{11})$
$n_{25} = (\{w_{1,0}^{(p)} = s_{1,0} + w_{1,1}^{(p)}\}, nil)$
[SubACC3] $n_{13} = (\{w_{1,2}^{(p)} = s_{1,0} + s_{1,1}\}, G_{12})$
$n_{26} = (\{w_{1,2}^{(p)} = s_{1,0} + s_{1,1}\}, nil)$
[PPG2] $n_5 = (\{w_2 = a_2^{(e)} \times b\}, G_6)$
[SubPPG2]
$n_{14} = (\{w_{2,2}^{(p)} = a_2^{(p)} \times b_0^{(p)}, s_{2,1} = a_2^{(p)} \times b_1^{(p)}, s_{2,2} = a_2^{(p)} \times b_2^{(p)}\}, G_{13})$
$n_{27} = (\{w_{2,2}^{(p)} = a_2^{(p)} \times b_0^{(p)}\}, nil)$
$n_{28} = (\{s_{2,1} = a_2^{(p)} \times b_1^{(p)}\}, nil)$
$n_{29} = (\{s_{2,2} = a_2^{(p)} \times b_2^{(p)}\}, nil)$
[SubACC4] $n_{15} = (\{w_{2,0}^{(p)} = s_{2,1} + s_{2,2}\}, G_{14})$
$n_{30} = (\{w_{2,0}^{(p)} = s_{2,1} + s_{2,2}\}, nil)$
[SubACC5] $n_{16} = (\{w_{2,1}^{(p)} = w_{2,2}^{(p)} + s_{2,1}\}, G_{15})$
$n_{31} = (\{w_{2,1}^{(p)} = w_{2,2}^{(p)} + s_{2,1}\}, nil)$
[Accumulator] $n_2 = (\{c = \sum_{i=0}^{2} w_i\}, G_3)$
[GFA0] $n_6 = (\{w_3 = w_0 + w_1\}, G_{16})$
$n_{32} = (\{w_{3,0}^{(p)} = w_{0,0}^{(p)} + w_{1,0}^{(p)}\}, nil)$
$n_{33} = (\{w_{3,1}^{(p)} = w_{0,1}^{(p)} + w_{1,1}^{(p)}\}, nil)$
$n_{34} = (\{w_{3,2}^{(p)} = w_{0,2}^{(p)} + w_{1,2}^{(p)}\}, nil)$
[GFA1] $n_7 = (\{c = w_2 + w_3\}, G_{17})$
$n_{35} = (\{c_0^{(p)} = w_{2,0}^{(p)} + w_{3,0}^{(p)}\}, nil)$
$n_{36} = (\{c_1^{(p)} = w_{2,1}^{(p)} + w_{3,1}^{(p)}\}, nil)$
$n_{37} = (\{c_2^{(p)} = w_{2,2}^{(p)} + w_{3,2}^{(p)}\}, nil)$

Galois field

$GF(2^3) = (\left(\alpha^{2^2}, \alpha^{2^1}, \alpha^{2^0}\right), (\{0,1\}, \{0,1\}, \{0,1\}), \beta^3 + \beta^1 + \beta^0)$ $\quad GF(2) = (\left(\beta^0\right), (\{0,1\}), nil)$

Galois field variables

$a, b, c = (GF(2^3), (2, 0))$	$w_i = (GF(2^3), (2, 0)),\ (0 \le i \le 3)$	$s_{1,i} = (GF(2), (0, 0)),\ (i = 0, 1)$
$a_i^{(e)} = (GF(2^3), (i, i)),\ (0 \le i \le 2)$	$w_{i,j}^{(p)} = (GF(2), (0, 0)),\ (0 \le i \le 3,\ 0 \le j \le 2)$	$s_{2,i} = (GF(2), (0, 0)),\ (i = 1, 2)$
$a_i^{(p)}, b_i^{(p)}, c_i^{(p)} = (GF(2), (0, 0)),\ (0 \le i \le 2)$	$s_{0,i} = (GF(2), (0, 0)),\ (i = 0, 2)$	

The 2^{nd}-level nodes "Partial Product Generator" and "Accumulator" in Fig. 3 (b) have functional assertions corresponding to Eqs. (15) and (16), respectively. The 3^{rd}-level nodes "PPGi" in Fig. 3 (c) have functional assertions corresponding to Eq. (17). The nodes "GFA0" and "GFA1" in Figs. 3 (c) and (d) indicate 2-input 1-output adders over $GF(2^3)$ to construct "Accumulator". The 4^{th}-level nodes "SubPPGi" and "SubACCi" in Fig. 3 (d) have functional assertions corresponding to Eqs. (18) and (19), respectively. If the number of δ satisfying $\delta_{i,j,k} = 1$ is one, $s_{i,j}$ becomes $w_{i,k}^{(p)}$ in the functional assertion of "SubPPGi" instead of Eq. (19). It is important to note that we can simply extend the above GF-ACG description to describe any Massey-Omura parallel multiplier over $GF(2^m)$ $(2 \le m)$.

In order to demonstrate the capability of the proposed method, we verify a set of the designed Massey-Omura parallel multipliers over $GF(2^m)$ $(2 \le m \le 64)$. In this experiment, we performed the proposed verification techniques using Risa/Asir on a Linux PC with an Intel Xeon E5450 3.00 GHz processor and 32 GB RAM. Both the original algorithm and the extended algorithm were performed in the same condition. For comparison, we also performed the Verilog-XL simulation using the corresponding HDL descriptions. Table 2 shows the verification results. We were not able to succeed the complete simulation of $GF(2^{16})$ and larger multipliers in this experiment because the verification time increases exponentially as the signal length increases. On the other hand, using our extended method, we were able to succeed the complete verification even for the 64-bit multiplier over $GF(2^{64})$.

Table 2 Verification times of Massey-Omura parallel multipliers [sec].

	$GF(2^4)$	$GF(2^8)$	$GF(2^{16})$	$GF(2^{32})$	$GF(2^{64})$
(a)	0.282	0.436	N/A	N/A	N/A
(b)	2.550	4.021	257.4	N/A	N/A
(c)	2.334	3.618	5.482	16.24	372.5

(a) Verilog-XL simulation, (b) previous work [9], (c) this work

5. Application to Exponentiation Circuits over $GF(2^m)$

This section applies the extended GF-ACG to $GF(2^m)$ exponentiation circuits given by NB representation and shows the performance of them. One major feature of NB representation is that the squaring operation is done by a cyclic shift (i.e., wiring) without any hardware component. A set of GF exponentiation circuits designed here include such squaring operations depending on the exponent.

Let $a \in GF(2^m)$ be the input. Let b $(= \sum_{k=0}^{n-1} b_k 2^k)$ and $c \in GF(2^m)$ be the exponent and the output, respectively. The exponentiation operation (i.e., $c = a^b$) is calculated by a combination of multiplication and squaring operations and is represented as

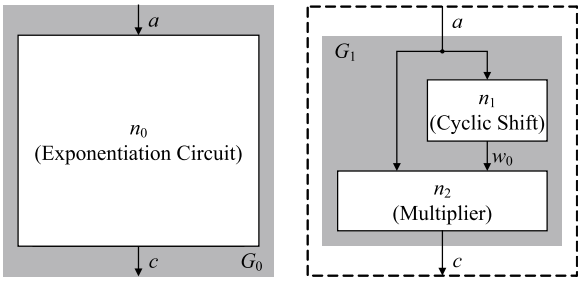

Fig. 4 GF-ACGs for cubic circuit: (a)–(b) GF-ACGs at two levels of abstraction.

Table 3 Nodes, Galois fields, and variables for cubic circuit in Fig. 4.

Nodes
[Exponentiation Circuit] $n_0 = (\{c = a^3\}, G_1)$
[Cyclic Shift] $n_1 = (\{w_0 = a^2\}, G_2)$
[Multiplier] $n_2 = (\{c = a \times w_0\}, G_2)$
Galois field
$GF(2^m) = (\left(\alpha^{2^m}, \alpha^{2^{m-1}}, \cdots, \alpha^{2^0}\right), (\{0,1\}, \{0,1\}, \cdots, \{0,1\}), \beta^m + \beta^{m-1} + \cdots + \beta^0)$
$GF(2) = (\left(\beta^0\right), (\{0,1\}), nil)$
Galois field variables
$a, c, w_0 = (GF(2^m), (m-1, 0))$

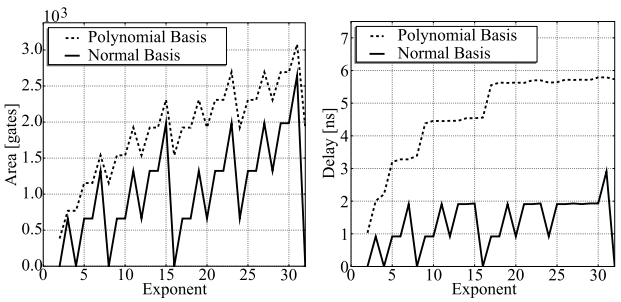

Fig. 5 Performance of $GF(2^8)$ exponentiation circuits.

$$c = a^{b_{n-1}2^{n-1}} \times a^{b_{n-2}2^{n-2}} \times \cdots \times a^{b_0 2^0}. \quad (20)$$

We design such exponentiation circuits based on NB by the GF-ACGs. The Massey-Omura parallel multipliers described in the above section are used for the multiplication, and the graphs performing the cyclic shift are added for the squaring. Figure 4 shows an example of the GF-ACGs for a cubic circuit given as $c = a^3$. Table 3 shows the details of nodes, GFs and variables used in Fig. 4. Note here that Cyclic Shift is implemented by wiring and have no internal structures.

The area and delay of the exponentiation circuits were evaluated using Synopsys Design Compiler with a TSMC 65-nm cell library. The extension degree used in this experiment was 8 (i.e., $GF(2^8)$). For comparison, we also designed the corresponding exponentiation circuits based on PB representation presented in [9]. Figure 5 shows the area and delay of the exponentiation circuits, respectively. We confirmed here that as the exponent b increased, the area and the delay of PB-based exponentiation circuits increased by $O(\log b)$, because they were constructed by a tree structure of some multipliers. On the other hand, the NB-based exponentiation circuits showed better performance the PB-based ones for both area and delay because squaring operations were free of cost in the NB-based circuits.

6. Application to Inversion Circuit over $GF(((2^2)^2)^2)$

This section presents a further extension of GF-ACGs to composite fields based on NB and shows an application of the extended GF-ACG to a multiplicative inversion circuit over composite field $GF(((2^2)^2)^2)$ that can be implemented more compactly than the counterpart based on PB [16].

In order to describe a composite field based on NB, the representation of coefficient sets is extended in such a way as to include all the elements of its basic field. In the following, we present the $GF((2^2)^2)$ description as an example. Let $GF(2^2)$ be the basic field, given as

$$GF(2^2) = (\left(\beta_0^{2^1}, \beta_0^{2^0}\right), (\{0,1\}, \{0,1\}), \beta_0^2 + \beta_0^1 + \beta_0^0). \quad (21)$$

The composite field $GF((2^2)^2)$ is then given as

$$GF((2^2)^2) = (\left(\beta_1^{(2^2)^1}, \beta_1^{(2^2)^0}\right), \left(\{0, 1, \gamma_0, \gamma_0^2\}, \{0, 1, \gamma_0, \gamma_0^2\}\right), \beta_1^2 + \beta_1^1 + \beta_0), \quad (22)$$

where the elements of $GF(2^2)$ are included with the primitive element γ_0 in the exponential representation.

Figure 6 shows a GF-ACG for the inversion circuit at three levels of abstraction, and Table 4 shows the nodes, GFs and variables in Fig. 6. The "Inversion" in Fig. 6 (a) is the highest-level node. Each node exhibits an internal structure given as a combination of lower-level nodes in the corresponding shaded part. Note again that decomposition/composition nodes are not shown in Table 4.

The functional assertion of the "Inversion" is given as $y = x^{254}$ according to the definition of multiplicative inversion. The circuit outputs a value of zero when the input is zero. As shown in Fig. 6 (b), the internal structure consists of three multipliers, two adders, one squaring coefficient multiplier and one inverter over $GF((2^2)^2)$. Each circuit over $GF((2^2)^2)$ is recursively described with lower-level $GF(2^2)$ circuits, which are shown in Fig. 6 (c). The lower-level nodes in Fig. 6 (c) are also described with the lowest-level nodes over $GF(2)$. The "Inversion" was verified with the proposed verification technique in about 2.5 s on the PC mentioned in Sect. 4.

The area and delay of the inversion circuit described in Fig. 6 and the corresponding inversion circuit based on PB in [10] were evaluated under the same condition mentioned in Sect. 4. Table 5 shows the comparison result. We confirmed that the NB-based inversion circuit showed better performance than the PB-based inversion circuit. This suggests the feasibility and advantage of the extended design and verification method.

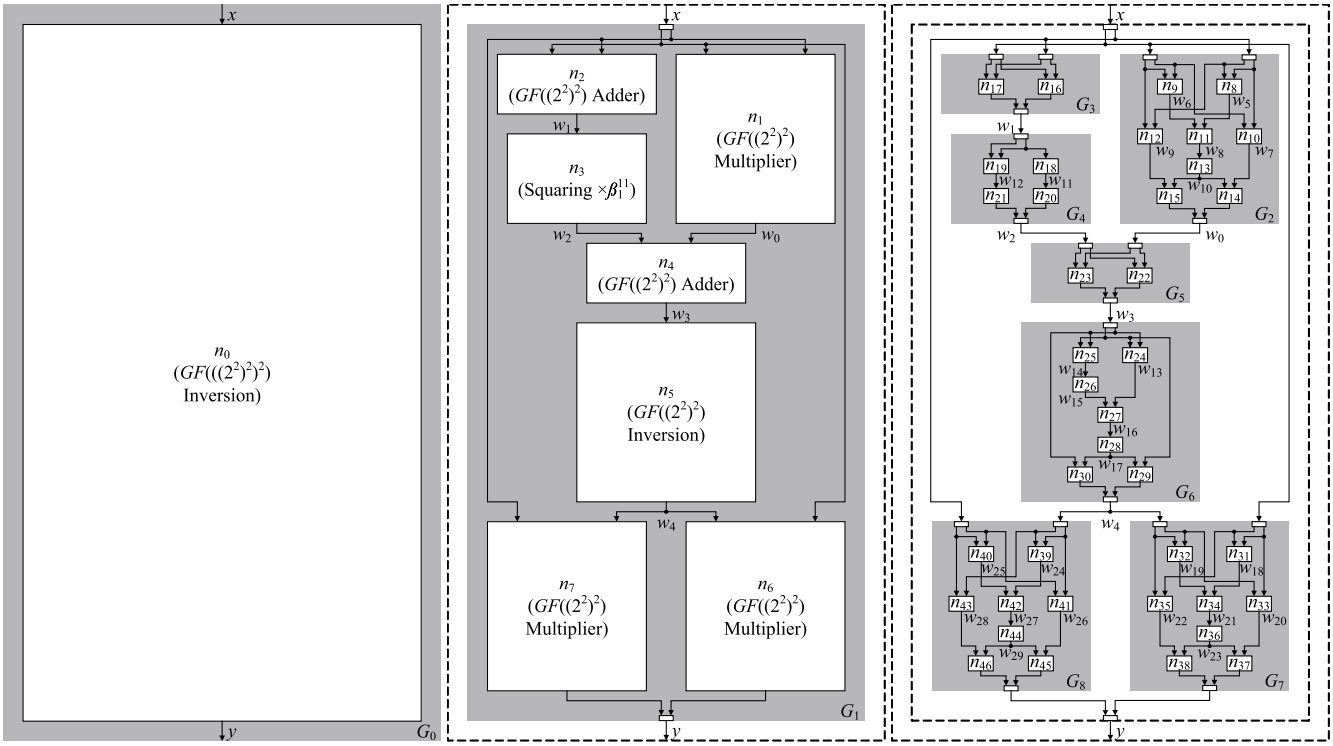

Fig. 6 GF-ACG for a multiplicative inversion circuit over $GF(((2^2)^2)^2)$: (a)–(c) GF-ACGs at three levels of abstraction.

Table 4 Nodes, Galois fields, and variables for the $GF(((2^2)^2)^2)$ inversion circuit in Fig. 6.

Nodes		
[Inversion over $GF(((2^2)^2)^2)$]	$n_{19} = (\{w_{12} = w_{1,0} + w_{0,0}\}, G_{20})$	$n_{31} = (\{w_{18} = x_{1,0} + x_{1,1}\}, G_{32})$
$n_0 = (\{y = x^{254}\}, G_1)$	$n_{20} = (\{w_{2,0} = w_{1,1}^2\}, G_{21})$	$n_{32} = (\{w_{19} = w_{4,0} + w_{4,1}\}, G_{33})$
[Multiplier over $GF((2^2)^2)$]	$n_{21} = (\{w_{2,1} = w_{1,2}^2\}, G_{22})$	$n_{33} = (\{w_{20} = x_{1,0} \times w_{4,0}\}, G_{34})$
$n_1 = (\{w_0 = x_0 \times x_1\}, G_2)$	[Adder over $GF((2^2)^2)$]	$n_{34} = (\{w_{21} = w_{18} \times w_{19}\}, G_{35})$
$n_8 = (\{w_5 = x_{0,0} + x_{0,1}\}, G_9)$	$n_4 = (\{w_3 = w_0 + w_2\}, G_5)$	$n_{35} = (\{w_{22} = x_{1,1} \times w_{4,1}\}, G_{36})$
$n_9 = (\{w_6 = x_{1,0} + x_{1,1}\}, G_{10})$	$n_{22} = (\{w_{3,0} = w_{0,0} + w_{2,0}\}, G_{23})$	$n_{36} = (\{w_{23} = w_{21} \times \beta_0^2\}, G_{37})$
$n_{10} = (\{w_7 = x_{0,0} \times x_{1,0}\}, G_{11})$	$n_{23} = (\{w_{3,1} = w_{0,1} + w_{2,1}\}, G_{24})$	$n_{37} = (\{y_{0,0} = w_{20} + w_{23}\}, G_{38})$
$n_{11} = (\{w_8 = w_5 \times w_6\}, G_{12})$	[Inversion over $GF((2^2)^2)$]	$n_{38} = (\{y_{0,1} = w_{23} + w_{22}\}, G_{39})$
$n_{12} = (\{w_9 = x_{0,1} \times x_{1,1}\}, G_{13})$	$n_5 = (\{w_4 = w_3^{14}\}, G_6)$	[Multiplier over $GF((2^2)^2)$]
$n_{13} = (\{w_{10} = w_8 \times \beta_0^2\}, G_{14})$	$n_{24} = (\{w_{13} = w_{3,0} \times w_{3,1}\}, G_{25})$	$n_7 = (\{y_1 = w_4 \times x_0\}, G_8)$
$n_{14} = (\{w_{0,1} = w_7 + w_{10}\}, G_{15})$	$n_{25} = (\{w_{14} = w_{3,0} + w_{3,1}\}, G_{26})$	$n_{39} = (\{w_{24} = x_{0,0} + x_{0,1}\}, G_{40})$
$n_{15} = (\{w_{0,0} = w_{10} + w_9\}, G_{16})$	$n_{26} = (\{w_{15} = w_{14}^2 \times \beta_0^2\}, G_{27})$	$n_{40} = (\{w_{25} = w_{4,0} + w_{4,1}\}, G_{41})$
[Adder over $GF((2^2)^2)$]	$n_{27} = (\{w_{16} = w_{13} + w_{15}\}, G_{28})$	$n_{41} = (\{w_{26} = w_{4,0} \times x_{0,0}\}, G_{42})$
$n_2 = (\{w_0 = x_0 + x_1\}, G_3)$	$n_{28} = (\{w_{17} = w_{16}^2\}, G_{29})$	$n_{42} = (\{w_{27} = w_{24} \times w_{25}\}, G_{43})$
$n_{16} = (\{w_{1,0} = x_{0,0} + x_{1,0}\}, G_{17})$	$n_{29} = (\{w_{4,0} = w_{3,1} \times w_{17}\}, G_{30})$	$n_{43} = (\{w_{28} = w_{4,1} \times x_{0,1}\}, G_{44})$
$n_{17} = (\{w_{1,1} = x_{0,1} + x_{1,1}\}, G_{18})$	$n_{30} = (\{w_{4,1} = w_{17} \times w_{3,0}\}, G_{31})$	$n_{44} = (\{w_{29} = w_{27} \times \beta_0^2\}, G_{45})$
[Squaring $\times\beta_1^{11}$]	[Multiplier over $GF((2^2)^2)$]	$n_{45} = (\{y_{0,0} = w_{26} + w_{29}\}, G_{46})$
$n_3 = (\{w_2 = w_1^2 \times \beta_1^{11}\}, G_4)$	$n_6 = (\{y_0 = x_1 \times w_4\}, G_7)$	$n_{46} = (\{y_{0,1} = w_{29} + w_{28}\}, G_{47})$
$n_{18} = (\{w_{11} = w_{1,0} \times \beta_0^2\}, G_{19})$		

Galois field	
$GF(((2^2)^2)^2) = (\left(\beta_2^{((2^2)^2)^1}, \beta_2^{((2^2)^2)^0}\right), \left(\{0, 1, \gamma_1^1, \ldots, \gamma_1^{14}\}, \{0, 1, \gamma_1^1, \ldots, \gamma_1^{14}\}\right), \beta_2^2 + \beta_2^1 + \beta_1^{11})$	$GF(2^2) = (\left(\beta_0^{2^1}, \beta_0^{2^0}\right), (\{0, 1\}, \{0, 1\}), \beta_0^2 + \beta_0^1 + \beta_0^0)$
$GF((2^2)^2) = (\left(\beta_1^{(2^2)^1}, \beta_1^{(2^2)^0}\right), \left(\{0, 1, \gamma_0^1, \gamma_0^2\}, \{0, 1, \gamma_0^1, \gamma_0^2\}\right), \beta_1^2 + \beta_1^1 + \beta_0^2)$	$GF(2) = (\left(\beta_0^0\right), (\{0, 1\}), nil)$

Galois field variables		
$x, y = (GF(((2^2)^2)^2), (1, 0))$	$x_{i,j}, y_{i,j} = (GF(2^2), (1, 0)),\ (0 \le i, j \le 1)$	$w_{i,j} = (GF(2^2), (1, 0)),\ (0 \le i \le 4, 0 \le j \le 1)$
$x_i, y_i = (GF((2^2)^2), (1, 0)),\ (0 \le i \le 1)$	$w_i = (GF((2^2)^2), (1, 0)),\ (0 \le i \le 4)$	$w_i = (GF(2^2), (1, 0)),\ (5 \le j \le 29)$

7. Conclusion

This paper presented a formal design of GF arithmetic circuits represented by normal basis (NB). First, we extended GF-ACG to describe any GF based on NB in addition to polynomial basis (PB) and presented a formal design of Massey-Omura parallel multipliers with the extended GF-

Table 5 Performance of $GF(((2^2)^2)^2)$ inversion circuits.

Basis	Area [gates]	Delay [ns]
Polynomial Basis	502.0	3.41
Normal Basis	375.3	3.08

ACG. The experimental result showed that the verification time was greatly reduced as compared with that of the conventional methods. For example, a multiplier over $GF(2^{64})$ was verified within 7 minutes. For another application, we also designed a set of NB-based exponentiation circuits and evaluated the performance in comparison with that of the corresponding PB-based circuits. In addition, we presented a further extension of GF-ACG to composite fields based on NB and applied it to an inversion circuit based on $GF((2^2)^2)$. The proposed method is applicable for both binary and multiple-valued implementations since the GF-ACG description is technology-independent except for the lowest-level description. The formal design of GF arithmetic circuits based on both PB and NB would remain in the future study.

Acknowledgments

We sincerely thank Prof. Y. Nogami of Okayama University for his valuable advice about the normal basis theory.

References

[1] K. Okamoto, N. Homma, and T. Aoki, "A graph-based approach to designing parallel multipliers over Galois fields based on normal basis representations," Proc. 43rd IEEE Int. Symp. Multiple-Valued Logic, pp.158–163, May 2013.
[2] E. Savas and K.C. Koc, "Finite field arithmetic for cryptography," IEEE Circuits Syst. Mag., vol.10, no.2, pp.40–56, Aug. 2010.
[3] R. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.
[4] E.R. Bryant and Y.A. Chen, "Verification of arithmetic circuits with binary moment diagrams," Proc. 32nd Design Automation Conf., pp.535–541, 1995.
[5] R. Drechsler, ed., Advanced Formal Verification, Kluwer Academic Publishers, 2004.
[6] R. Stankovic and R. Drechsler, "Circuit design from Kronecker Galois field decision diagrams for multiple-valued functions," Proc. 27th IEEE Int. Symp. Multiple-Valued Logic, pp.275–280, 1997.
[7] S. Morioka, Y. Katayama, and T. Yamane, "Towards efficient verification of arithmetic algorithms over Galois fields $GF(2^m)$," Proc. 13th Conf. on Computer Aided Verification, LNCS, vol.2102, pp.465–477, 2001.
[8] D. Mukhopadhyay, G. Sengar, and R.D. Chowdhury, "Hierarchical verification of Galois field circuits," IEEE Trans. Comput.-Aided Integr. Circuits Syst., vol.26, no.10, pp.1893–1898, 2007.
[9] N. Homma, K. Saito, and T. Aoki, "A formal approach to designing cryptographic processors based on $GF(2^m)$ arithmetic circuits," IEEE Trans. Information Forensics and Security, vol.7, no.1, pp.3–13, Feb. 2012.
[10] N. Homma, K. Saito, and T. Aoki, "Toward formal design of practical cryptographic hardware based on Galois field arithmetic," IEEE Trans. Comput., DOI: http://doi.ieeecomputersociety.org/10.1109/TC.2013.131, 2013.
[11] D.A. Cox, J.B. Little, and D. O'Shea, Ideals, Varieties, and Algorithms, 2nd ed., Springer-Verlag, NY, 1996.
[12] R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, and R.M. Wilson, "Optimal normal bases in $GF(p^n)$," Discrete Applied Mathematics, vol.22, no.2, pp.149–161, 1989.
[13] S. Gao, Normal bases over finite fields, Citeseer, 1993.
[14] A. Reyhani-Masoleh and M. Hasan, "A new construction of Massey-Omura parallel multiplier over $GF(2^m)$," IEEE Trans. Comput., vol.51, no.5, pp.511–520, May 2001.
[15] J. Massey and J. Omura, "Computational method and apparatus for finite field arithmetic," 1986. US Patent.
[16] D. Canright, "A very compact S-box for AES," CHES 2005, pp.441–455, May 2005.

Kotaro Okamoto received the B.E. degree in information engineering and the M.S degree in information sciences from Tohoku University, Sendai, Japan, in 2012 and 2014, respectively. From 2014, he joined Hitachi, Ltd. His research interest includes Galois-field arithmetic and its circuit design.

Naofumi Homma received the B.E. degree in information engineering, and the M.S. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Japan, in 1997, 1999 and 2001, respectively. He is currently an Associate Professor of the Graduate School of Information Sciences at Tohoku University. For 2002–2006, he also joined the Japan Science and Technology Agency (JST) as a researcher for the PRESTO project. His research interests include computer arithmetic, EDA methodology, high-performance/secure VLSI computing, and hardware security. Dr. Homma received the IP Award at the 2005 LSI IP Design Award, and the Best Paper Award at the Workshop on Synthesis And System Integration of Mixed Information Technologies in 2007.

Takafumi Aoki received the B.E., M.E., and D.E. degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1988, 1990, and 1992, respectively. He is currently a Professor of the Graduate School of Information Sciences at Tohoku University. For 1997–1999, he also joined the PRESTO project, Japan Science and Technology Corporation (JST). His research interests include theoretical aspects of computation, digital signal processing, computer vision, image processing, biometric authentication, and security issues in computer systems. Dr. Aoki received the Outstanding Paper Award at the 1990, 2000, 2001 and 2006 IEEE International Symposiums on Multiple-Valued Logic, the Outstanding Transactions Paper Award from the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan in 1989 and 1997, the IEE Ambrose Fleming Premium Award in 1994, the IEICE Inose Award in 1997, the IEE Mountbatten Premium Award in 1999, the Best Paper Award at the 1999 IEEE International Symposium on Intelligent Signal Processing and Communication Systems, the IP Award at the 7th LSI IP Design Award in 2005, and the Best Paper Award at the 14th Workshop on Synthesis And System Integration of Mixed Information technologies in 2007.