A Formal Approach to Designing Arithmetic Circuits over Galois Fields Using Symbolic Computer Algebra

Kazuya Saito, Naofumi Homma, and Takafumi Aoki

Graduate School of Information Sciences, Tohoku University
Aramaki Aza Aoba 6-6-5, Sendai 980-8579, Japan
Email: {saito, homma}@aoki.ecei.tohoku.ac.jp

Abstract— This paper proposes a formal approach to designing arithmetic circuits over Galois Fields (GFs). Our method represents a GF arithmetic circuit by a hierarchical graph structure specified by variables and arithmetic formulae over GFs. The proposed circuit description is applicable to any $GF(p^m)$ $(p \geq 2)$ arithmetic and is formally verified by symbolic computation techniques such as polynomial reduction using Gröbner basis. In this paper, we propose the graph representation and show some examples of its description and verification. The advantageous effect of the proposed approach is demonstrated through experimental designs of parallel multipliers over Galois field $GF(2^m)$ for different word-lengths and irreducible polynomials. An inversion circuit consisting of some multipliers is also designed and verified as a further application. The result shows that the proposed approach has a definite possibility of verifying practical GF arithmetic circuits where the conventional simulation and verification techniques failed.

I. Introduction

The use of arithmetic algorithms over Galois fields (GFs) have been rapidly increasing due to the high demand of error correction codes and cryptographic systems in recent dependable and secure devices. On the other hand, most of such arithmetic algorithms are devised by researchers who had trained in a particular way to understand GF arithmetic. Even the state-of-the-art Hardware Description Languages (HDLs) and high-level languages (e.g., SystemC and System Verilog) do not handle high-level arithmetic data structures, arithmetic operations and formulae over Galois fields. Such conventional design environments sometimes require us to describe and verify structural details of arithmetic circuits at the lowest level of abstraction.

Previous researches on arithmetic circuit verification are primarily based on Decision Diagrams (DDs) or Binary Moment Diagrams (BMDs) [1, 2, 3, 4]. Reference [3], for example, presents a hardware description language, called ACV language, to verify arithmetic circuits in a hierarchical fashion using *BMDs. However, the conventional approaches are basically limited to binary arithmetic over integer since they are inherently based on bit-level integer operations. Binary Decision Diagrams (BDDs) can be applied to GF arithmetic, but BDDs are not effective for verifying arithmetic circuits. There is another decision diagram for Galois fields based on the decomposition of multiple-valued functions[5], but it is still difficult to handle practical fields such as $GF(2^{16})$, $GF(2^{32})$ and larger algorithms including many operators. Some GF arithmetic circuits were verified effectively in [6][7], but their applications are limited to specific $GF(2^m)$s which are given as a form of $GF((2^n)^p)$s.

Addressing the above problem, this paper proposes a formal approach to describing and verifying arithmetic algorithms over Galois fields using symbolic computer algebra. Our method describes arithmetic circuits directly by high-level mathematical graphs associated with variables and arithmetic formulae over GFs. The graph can represent any $GF(p^m)$ arithmetic circuit (where $p \geq 2$) in a hierarchical manner, where each component (i.e. sub-circuit) has a function and an internal structure defined by Galois-field equations. (A preliminary study presented in [8] was limited to $GF(2^m)$ arithmetic.) 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 Gröbner basis and a polynomial reduction technique [9], which makes it possible to verify arithmetic circuits over practical GFs such as $GF(2^{128})$.

In this paper, we also demonstrate the advantageous effect of the proposed verification method in comparison with conventional simulation and verification methods through experimental designs of parallel multipliers over $GF(2^m)$ for different word-lengths and irreducible polynomials. The result shows that the proposed approach has a definite possibility of verifying GF arithmetic circuits where the conventional techniques failed.

II. Arithmetic circuit representation

The function of arithmetic circuits is usually represented by logic functions or lookup tables defining all the input-output combinations uniquely. Such representations, however, are not always suitable for representing large arithmetic circuits with many input variables. Assuming that arithmetic circuits implement arithmetic functions which should be dealt in the arithmetic domain rather than the Boolean logic domain, we intro-

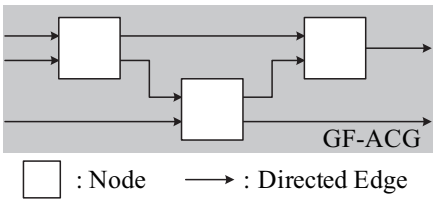


Fig. 1. Galois-field Arithmetic circuit graph.

duce a graph-based representation of arithmetic circuits over Galois fields in a hierarchical manner based on *Galois-field (GF) equations*. This representation can be considered as an extension of arithmetic circuit graphs (ACGs) presented in [10] for integer arithmetic circuits.

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

A node n ($\in \boldsymbol{N}$) is given by $n = (\boldsymbol{F}, G')$, where $\boldsymbol{F}$ is the functional assertion given as a set of GF equations and G' is the internal structure given as a lower-level GF-ACG.

A node that does not have its internal structure is said to be *lowest level*, and is represented as $n = (\boldsymbol{F}, nil)$. Let E_l and E_r be expressions given by variables, constants or combinations of the two or more expressions connected by arithmetic operations $+$, $-$, $\times$, and $/$. A GF equation is defined as a relation $E_l = E_r$, where E_l and E_r indicate the output and input expressions, respectively.

A directed edge e ($\in \boldsymbol{E}$) is defined as $e = (src, dest, x)$, where src indicates the start node, $dest$ indicates the end node, and x indicates the variable. A directed edge is said to be a half edge if one endpoint of the directed edge is not connected to any node. The half edge represents an external input or output for the given GF-ACG.

Each variable is associated with a Galois field. A Galois field GF is represented as $GF = (\boldsymbol{B}, \boldsymbol{C}, IP)$, 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

$$\boldsymbol{B} = \left(\beta^{m-1}, \cdots, \beta^{i+1}, \beta^i, \beta^{i-1}, \cdots, \beta^0\right), \quad (1)$$

$$\boldsymbol{C} = (C_{m-1}, \cdots, C_{i+1}, C_i, C_{i-1}, \cdots, C_0), \quad (2)$$

$$IP = \beta^m + \alpha_{m-1}\beta^{m-1} + \cdots + \alpha_1\beta^1 + \alpha_0\beta^0, \quad (3)$$

where β is the indeterminate element, C_i is the coefficient set of degree i, m is the degree of field extension, and α_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 ($h \leq m-1$) and l ($0 \leq l \leq h$) be the most and least significant degrees, respectively. A variable is given as $x = (GF, (h, l))$, where the tuple (h, l) is called the degree range. Using the above notation, we easily 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 ($l \leq i \leq h$) be a lower-level variable. We have two types of decomposition nodes whose functions are given as

$$x_h + x_{h-1} + \cdots + x_i + \cdots + x_{l+1} + x_l = x, \quad (4)$$

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

Eq. (4) indicates that the variable $x \in GF(p^m)$ is divided into a number of variables of degree i (i.e. $x_i (l \leq i \leq h) \in GF(p^m)$). On the other hand, Eq. (5) indicates that the variable $x \in GF(p^m)$ is divided into a number of variables over the prime field (i.e. $x_i (l \leq i \leq h) \in GF(p)$). 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.

For example, a variable $x \in GF(2^2)$ is represented as

$$GF(2^2) = ((\beta^1, \beta^0), (\{0,1\}, \{0,1\}), \beta^2 + \beta^1 + \beta^0), \quad (6)$$

$$x = (GF(2^2), (1, 0)). \quad (7)$$

The variable x can be decomposed into two lower-level variables x_i ($0 \leq i \leq 1$), such as $x_1 + x_0 = x$, by the decomposition node of Eq. (4).

The above GF-ACG can be used also for representing logic circuits. A logic variable is considered as a variable over a Galois field 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 NOT, OR, AND, and XOR circuits are given as

$$NOT(u) = 1 - u, \quad (8)$$

$$OR(u, v) = u + v - uv, \quad (9)$$

$$AND(u, v) = uv, \quad (10)$$

$$XOR(u, v) = u + v - 2uv, \quad (11)$$

respectively. Each logic variable is associated with a Galois field $Logic$ defined as

$$Logic = ((\beta^0), (\{0,1\}), nil). \quad (12)$$

Therefore $u = (Logic, (0, 0))$ and $v = (Logic, (0, 0))$. Note that the idempotent law is considered as a functional assertion in the corresponding node (i.e. $u = u^2$ and $v = v^2$). Thus, GF-ACG can represent any GF arithmetic circuit from the logic level.

In the lowest-level node, we usually represent the functional assertion and the internal structure by GF equations and pseudo-logic equations (i.e.,(8)- (11)), respectively. In order to evaluate the relationship between the GF equations and the

TABLE I
TRANSLATION OF GF VALUE TO LOGIC VALUE

(a) Example of $GF(2)$

GF(2) val.	Logic val.
0	0
1	1

(b) Example of $GF(3)$

GF(3) val.	Logic val.
0	00
1	01
2	10

pseud-logic equations, we need to include an encoding function from GF variable to logic variables in the internal structure.

Each GF variable in C_i (the coefficient set of degree i) is encoded by at least $\lceil \log |C_i| \rceil$ logic variable(s). For example, GF values used in $GF(2)$ ($\in \{0,1\}$) are encoded as shown in Table I(a). Any encoding including non-minimum-length encodings is possible also for larger characteristic p (≥ 2). For example, GF values used in $GF(3)$ ($\in \{0,1,2\}$) are encoded as shown in Table I(b), which corresponds to a standard binary encoding.

An encoding from GF variable and logic variables is specified by a specific equation called *encoding equation*. Let x and $L_j (0 \leq j \leq k-1)$ be a GF variable over $GF(p)$ and a logic variable used for encoding. Also, let $\boldsymbol{\alpha} (= (\alpha_0, \alpha_1, \cdots, \alpha_{k-1}) \in \{0,1\}^k)$ be a k-bit logic value. The general form of the encoding equation is given as

$$x = \sum_{\boldsymbol{\alpha} \in \{0,1\}^k} \left(f(\boldsymbol{\alpha}) \times \Pi_{j=0}^{k-1} L_j^{\alpha_j} \right), \tag{13}$$

where $f(\boldsymbol{\alpha})$ is the GF value corresponding to $\boldsymbol{\alpha}$, and $L_j^{\alpha_j}$ is the jth literal defined as

$$L_j^{\alpha_j} = \begin{cases} 1 - L_j & (\alpha_j = 0) \\ L_j & (\alpha_j = 1) \end{cases}. \tag{14}$$

For example, the encoding equations for Table I (a) and (b) are given as

$$x = L_0, \tag{15}$$

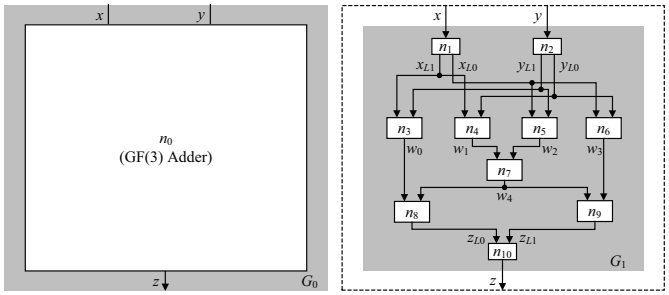
and

$$x = (1 - L_1)L_0 + 2 \times L_1(1 - L_0), \tag{16}$$

respectively. Thus, we describe any internal structure of logic circuit at the lowest level of abstraction with the above encoding equations in addition to pseudo-logic equations. As a result, GF-ACG can represent any $GF(p^m)$ arithmetic circuit including logical operations in a uniform manner.

Fig. 2 shows the GF-ACGs for 2-input adder over $GF(3)$ [11]. Tables II and III show the functional assertions and GF variables, respectively. The functions of n_1 and n_2 are to translate a GF variables to logic variables. In contrast, the function of n_{10} is to translate logic variables to a GF variable. Note here that the functional assertions also declare unused inputs by an equation such as $x_{L0}x_{L1} = 0$, which means that $(x_{L0}, x_{L1}) = (1,1)$ is not used.

As another example, Fig. 3 shows the GF-ACGs for 2-input parallel multiplier over $GF(2^2)$ at various levels of abstraction, where the square blocks indicate the nodes. The "Multiplier" block in (a) is in the highest level of hierarchy. The



(a) (b)

Fig. 2. GF-ACG for adder over $GF(3)$.

TABLE II
FUNCTIONAL ASSERTIONS IN FIG. 2

[GF(3) Adder] $n_0 = (\{z = x + y\}, G_1)$
$n_1 = (\{(1 - x_{L1})x_{L0} + 2x_{L1}(1 - x_{L0}) = x, x_{L0}x_{L1} = 0\}, nil)$
$n_2 = (\{(1 - y_{L1})y_{L0} + 2y_{L1}(1 - y_{L0}) = y, y_{L0}y_{L1} = 0\}, nil)$
$n_3 = (\{w_0 = OR(x_{L1}, y_{L1})\}, nil)$
$n_4 = (\{w_1 = OR(x_{L1}, y_{L0})\}, nil)$
$n_5 = (\{w_2 = OR(x_{L0}, y_{L1})\}, nil)$
$n_6 = (\{w_3 = OR(x_{L0}, y_{L0})\}, nil)$
$n_7 = (\{w_4 = XOR(w_1, w_2)\}, nil)$
$n_8 = (\{z_{L0} = XOR(w_0, w_4)\}, nil)$
$n_9 = (\{z_{L1} = XOR(w_3, w_4)\}, nil)$
$n_{10} = (\{z = (1 - z_{L1})z_{L0} + 2z_{L1}(1 - z_{L0}), z_{L1}z_{L0} = 0\}, nil)$

blocks in Figs. 3 (a), (b), and (c) correspond to the shaded parts in Figs. 3 (b), (c), and (d), respectively. Each block has its internal structure given by a combination of smaller blocks in the corresponding shaded part. For example, " Partial product generator " block consists of two smaller blocks " PPG0 " and " PPG1. "

For example, an GF-ACG G_1 is represented as

$$\begin{aligned} G_1 =& (\{n_1, n_2\}, \\ & \{(nil, n_1, x), (nil, n_1, y), \\ & (n_1, n_2, t_0), (n_1, n_2, t_1), (n_2, nil, z)\}), \end{aligned} \tag{17}$$

where

$$n_1 = (\{t_0 + t_1 = x \times y\}, G_2), \tag{18}$$

$$n_2 = (\{z = t_0 + t_1\}, G_3). \tag{19}$$

Tables IV and V show the functional assertions and GF variables, respectively. Note that decomposition/composition nodes are not shown in Fig. 3 and Table IV.

III. FUNCTIONAL VERIFICATION USING SYMBOLIC COMPUTER ALGEBRA

Fig. 4 shows an overview of the verification procedure. Given a GF-ACG, the $FormulaEvaluation$ is applied to all the nodes having functional assertions and internal structures. If GF equations of the internal structure are equivalent to the functional assertion(s), $FormulaEvaluation$ returns

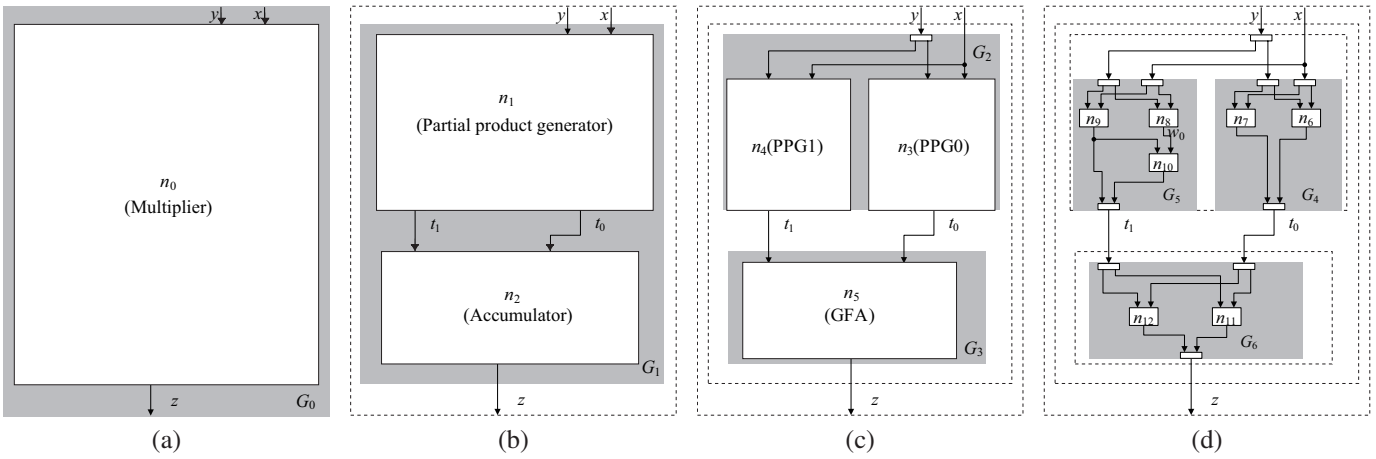


Fig. 3. GF-ACG for parallel multiplier over $GF(2^2)$.

TABLE III
GALOIS FIELDS AND VARIABLES IN FIG. 2

Galois field
$GF(3) = (\langle\beta^0\rangle, (\{0,1,2\}), nil)$
$Logic = (\langle\beta^0\rangle, (\{0,1\}), nil)$
Galois field variables
$x = (GF(3), (0,0))$
$x_{L0} = (Logic, (0,0))$
$x_{L1} = (Logic, (0,0))$
$y = (GF(3), (0,0))$
$y_{L0} = (Logic, (0,0))$
$y_{L1} = (Logic, (0,0))$
$z = (GF(3), (0,0))$
$z_{L0} = (Logic, (0,0))$
$z_{L1} = (Logic, (0,0))$
$w_i = (Logic, (0,0)),\ (0 \le i \le 4,)$

TABLE IV
FUNCTIONAL ASSERTIONS IN FIG. 3

[Multiplier] $n_0 = (\{z = x \times y\}, G_1)$
[Partial product generator]
$n_1 = (\{t_0 + t_1 = x \times y\}, G_2)$
[PPG0] $n_3 = (\{t_0 = x \times y_0\}, G_4)$
$n_6 = (\{t_{0,0} = x_0 \times y_{0,0}\}, nil)$
$n_7 = (\{t_{0,1} = x_1 \times y_{0,0}\}, nil)$
[PPG1] $n_4 = (\{t_1 = x \times y_1\}, G_5)$
$n_8 = (\{w_0 = x_0 \times y_{1,1}\}, nil)$
$n_9 = (\{t_{1,2} = x_1 \times y_{1,1}\}, nil)$
$n_{10} = (\{t_{1,3} = x_2 \times y_{1,1}\}, nil)$
[Accumulator]
$n_2 = (\{z = t_0 + t_1\}, G_3)$
[GFA] $n_5 = (\{z = t_0 + t_1\}, G_6)$
$n_{11} = (\{z_0 = t_{0,0} + t_{1,0}\}, nil)$
$n_{12} = (\{z_1 = t_{0,1} + t_{1,1}\}, nil)$

TABLE V
GALOIS FIELDS AND VARIABLES IN FIG. 3

Galois field
$GF(2^2) = (\langle\beta^1, \beta^0\rangle, (\{0,1\}, \{0,1\}), \beta^2 + \beta^1 + \beta^0)$
$GF(2) = (\langle\beta^0\rangle, (\{0,1\}), nil)$
Galois field variables
$x = (GF(2^2), (1,0))$
$x_i = (GF(2), (0,0)),\ (0 \le i \le 1)$
$y = (GF(2^2), (1,0))$
$y_i = (GF(2^2), (i,i)),\ (0 \le i \le 1)$
$y_{i,i} = (GF(2), (0,0)),\ (0 \le i \le 1)$
$z = (GF(2^2), (1,0))$
$z_i = (GF(2), (0,0)),\ (0 \le i \le 1)$
$t_i = (GF(2^2), (1,0)),\ (0 \le i \le 1)$
$t_{i,j} = (GF(2), (1,0)),\ (0 \le i,j \le 1)$
$w_0 = (GF(2), (0,0))$

$true$. Here, we assume that the lowest-level nodes are given by logical functions, such as NOT and AND, which are predetermined and reliable. The major feature of GF-ACGs is that the formula evaluation can be preformed by symbolic computation, which utilizes polynomial reduction and Gröbner basis techniques [9]. In the following, we briefly describe the fundamentals of polynomial reduction, normal form, and the Gröbner basis, and then explain how the symbolic computation is applied to the formula evaluation.

Given a polynomial p, let $HT(p)$ be the monomial in the maximal term (or head term) among those in p with respect to a total ordering of the variables. Let $HC(p)$ be the coefficient of the maximal term. Given polynomials p and $q \neq 0$, suppose a term M, which can be divided by $HT(q)$, exists in p. The polynomial reduction is defined as

$$p' = p - \frac{C_M M}{HC(q)HT(q)}q, \tag{20}$$

where p' is the resulting polynomial and C_M is the coefficient of M.

For any polynomial p, we have a unique element, which is reduced repeatedly with respect to a set of polynomials $\boldsymbol{Q} = \{q_1, \cdots, q_m\}$. The element is called a normal form, and is denoted by $NF_{\boldsymbol{Q}}(p)$.

Input:	Arithmetic circuit graph $G = (\boldsymbol{N}, \boldsymbol{E})$
Output:	Verification result $r \in \{true, false\}$

```
1:  Function Verify(G)
2:    r := true
3:    for each (F, G') ∈ N
4:      if G' ≠ nil
5:        r := r & Verify(G')
6:        for each f ∈ F
7:          r := r & FormulaEvaluation(f, G')
8:        end for
9:      end if
10:   end for
11:   return r
12: end
```

Fig. 4. Proposed verification algorithm.

Here, we denote $\boldsymbol{R}[\boldsymbol{x}] = \boldsymbol{R}[x_0, x_1, \cdots, x_{n-1}]$ as the ring of all polynomials obtained from variables $\boldsymbol{x} = (x_0, x_1, \cdots, x_{n-1})$. Every finite set of polynomials $\boldsymbol{P} = \{p_0, p_1, \cdots, p_{k-1}\} \subset \boldsymbol{R}[\boldsymbol{x}]$ generates a *polynomial ideal* (or simply, *ideal*) $\boldsymbol{I}$ as follows:

$$\boldsymbol{I} = \{a_0p_0 + a_1p_1 + \cdots + a_{k-1}p_{k-1} \mid a_0, a_1, \cdots, a_{k-1} \in \boldsymbol{R}[\boldsymbol{x}]\}. \tag{21}$$

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

Function $FormulaEvaluation(f, G)$
  $\boldsymbol{P} := \emptyset$
  for each $(\boldsymbol{F}, G') \in \boldsymbol{N}$
    $\boldsymbol{P} := \boldsymbol{P} \cup \boldsymbol{F}$
  end for
  $\boldsymbol{GB} := GroebnerBasis(\boldsymbol{P})$
  if $NF_{\boldsymbol{GB}}(f) = 0$
    $r := true$
  else
    $r := false$
  end if
  return r
end

Fig. 5. Formula evaluation algorithm.

The set $\boldsymbol{P}$ is called generator or *basis* of $\boldsymbol{I}$. A basis is a Gröbner basis with respect to a ordering of the variables if $NF_{\boldsymbol{Q}}(p) = 0$ for any polynomial p in an ideal $\boldsymbol{I}$.

Buchberger [12] has shown that an arbitrary basis can be transformed into the Gröbner basis. A reduced Gröbner basis forms a canonical representation for a polynomial ideal, which enables us to check whether the given polynomial is in the ideal.

Fig. 5 illustrates the formula evaluation procedure using Gröbner basis, 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 in the internal structure. $\boldsymbol{GB}$ is then obtained from $GroebnerBasis(\boldsymbol{P})$. If the normal form of f with respect to $\boldsymbol{GB}$ is equal to zero, f is a member of the ideal generated from $\boldsymbol{P}$. This means that the functional assertion can be realized with the internal structure. Therefore, $FormulaEvaluation(f, G)$ returns $true$.

Example 1 *Consider a formula evaluation for the highest-level node n_0 of $GF(2^2)$ multiplier in Fig. 3, where the irreducible polynomial is $IP = \beta^2 + \beta^1 + \beta^0$. We first obtain a set of polynomials $\boldsymbol{P}$ from the internal structure including two nodes n_1 and n_2 in G_1.*

$$\boldsymbol{P} = \{t_0 + t_1 - x \times y, \\ z - (t_0 + t_1)\}. \quad (22)$$

Then, we derive the Gröbner basis $\boldsymbol{GB}$ from $\boldsymbol{P}$ as follows:

$$\boldsymbol{GB} = \{x \times y + z, t_0 + t_1 + z\}. \quad (23)$$

The normal form of the function with respect to $\boldsymbol{GB}$ is given as $NF_{\boldsymbol{GB}}(z - x \times y) = 0 \left(\mathrm{mod}\ \beta^2 + \beta^1 + \beta^0\right)$. Therefore, the formula evaluation returns true.

Buchberger's algorithm sometimes takes a long time and requires large memory space. If the set of polynomials consists of linear polynomials, however, the Gröbner basis calculation is equivalent to Gaussian Elimination [9]. In this case, the computation cost of the proposed method becomes $O(k^3)$, where k is the number of equations. For many arithmetic circuits, word-level structures are commonly represented by linear equations, and thus the proposed verification method can be effective for verifying such word-level functions.

TABLE VI
VERIFICATION TIME OF PARALLEL MULTIPLIERS OVER $GF(2^m)$ FOR DIFFERENT DEGREES

Galois fields	Verification time [sec]			
	HDL sim.	BDD	This work (all)	This work (minimum)
$GF(2^4)$	0.12	< 0.01	8.69	1.95
$GF(2^8)$	0.54	0.01	44.57	2.79
$GF(2^{16})$	1 day	10.98	183.00	4.91
$GF(2^{32})$	N/A	1 week	747.63	12.67
$GF(2^{64})$	N/A	N/A	6241.30	64.10
$GF(2^{128})$	N/A	N/A	18244.04	615.06

IV. EXPERIMENTAL VERIFICATION

To evaluate the verification times of the proposed method, we designed a set of 2-operand parallel multipliers over $GF(2^m)$ $(4 \leq m \leq 128)$ as shown in Fig. 3. (Each circuit structure in Fig. 3 is simply extended according to the value m.) We describe adders and multipliers over $GF(2)$ with pseudo logic equations at the lowest level of abstraction. In this experiment, we implement the proposed verification method in two ways. One is a straightforward implementation to verify all the graphs included in the multiplier representation. Another is an optimized implementation to verify only a minimum number of the graphs. The GF-ACGs designed here include a number of the same sub-circuits, such as 1-bit adders, which have the same internal structures and are different only in variable names. We can easily extract the same sub-circuits from the functional assertions. Therefore, we verify such sub-circuits only once in the optimized implementation in order to reduce the verification time. This technique is particularly effective for arithmetic circuits.

These proposed verification methods were performed using Mathematica (version 6.0) on Intel Xeon E5450 3.00 GHz and 32GB memory. For comparison, we also performed a Verilog-XL simulation using the corresponding HDL descriptions and a BDD equivalence checking as conventional simulation and formal-verification techniques, respectively. We used an open-source code for the BDD construction [13].

Table VI shows the verification time of 2-operand multipliers over $GF(2^m)$ whose operand lengths are m and the irreducible polynomials are selected from typical ones [14]. The verification times of HDL simulation ("HDL sim.") and BDD equivalent checking ("BDD") are smaller than those of the proposed methods for low extension degrees (i.e., operand lengths) such as $GF(2^8)$. However, the simulation time increased exponentially as the operand length increased. We required one day to finish the complete simulation of $GF(2^{16})$ in this experiment. The verification time using BDDs also in-

creased significantly as the operand length increased. We used one week to verify $GF(2^{32})$ by BDDs in this experiment. This result shows the verification time of GF multipliers is similar to that of integer multipliers [15]. Other DDs such as FDDs and BMDs are defined on integer operations and not directly applicable to GF arithmetic circuits.

On the other hand, the straightforward implementation ("This work (all)") can verify the GF multipliers effectively even if the operand length is 128 bits. The verification time of the proposed methods increase in the order of at most m^3, where m is the extension degree. Furthermore, we significantly reduced the verification times by the optimized implementation ("This work (minimum)") since the GF multipliers include a number of $GF(2)$ adders, $GF(2)$ multipliers and $GF(2^m)$ adders. The verification of $GF(2^{128})$ multiplier was performed about 10 minutes.

Table VII shows the verification times of $GF(2^{31})$ multipliers in both straightforward and optimized implementations for different irreducible polynomials. This suggests that the verification time of GF arithmetic circuits is dependent on the type of irreducible polynomial which changes the circuit structure. But the maximum time is at most 1.5 times the minimum time as shown in Table VII.

Other arithmetic operations, such as squaring and inverse, are usually composed of multipliers. For example, a datapath of a point addition for ECC (Elliptic Curve Cryptography) processors can also be composed of adders, multipliers, squaring and inverse operations. The computational cost of the other operations would be comparable to that of the multiplier over the same GFs. For example, if the inverse function is given as $y = x^{2^m-2}$, the GF-ACG can be used to represent and verify it. When $GF(2^8)$ inversion circuit, which is used in AES hardware, consists of seven squaring circuits and six multipliers, it was verified by our method in about 5 seconds.

TABLE VII
VERIFICATION TIMES OF MULTIPLIERS OVER $GF(2^{31})$ FOR DIFFERENT IRREDUCIBLE POLYNOMIALS

Irreducible polynomial	Verification time [sec]	
	This work (all)	This work (minimum)
$\beta^{31}+\beta^3+\beta^0$	617.41	11.06
$\beta^{31}+\beta^6+\beta^0$	620.83	11.12
$\beta^{31}+\beta^7+\beta^0$	623.53	11.13
$\beta^{31}+\beta^{13}+\beta^0$	637.69	11.35
$\beta^{31}+\beta^{23}+\beta^{15}+\beta^7+\beta^0$	796.09	12.31
$\beta^{31}+\beta^{25}+\beta^{19}+\beta^{13}+\beta^0$	802.51	12.86
$\beta^{31}+\beta^3+\beta^2+\beta+\beta^0$	862.06	13.46
$\beta^{31}+\beta^6+\beta^4+\beta^2+\beta^0$	868.86	13.57
$\beta^{31}+\beta^{13}+\beta^8+\beta^3+\beta^0$	920.39	15.14
$\beta^{31}+\beta^{15}+\beta^{14}+\beta^{13}+\beta^0$	942.87	15.50

V. CONCLUSION

This paper proposed a graph-based approach for designing arithmetic circuits over Galois fields. The key idea is to describe arithmetic circuits with high-level graphs based on Galois-field equations in a hierarchical manner. The proposed representation can be formally verified by formula manipulations using polynomial reduction techniques. The experimental result showed that the proposed method can reduce the verification time significantly as compared with the conventional simulation and verification techniques. As a result, we can successfully verify practical GF arithmetic circuits such as parallel multipliers over $GF(2^{128})$ about 10 minutes. The proposed graph-based representation is capable of describing any GF arithmetic circuit including logic operation. However, conventional formal verification techniques based on DD still have advantages in terms of verification time for some circuits over lower-extension fields such as $GF(2^4)$. Further investigations are being conducted to develop the effective combination of the proposed method and DD-based methods in order to reduce the verification time. Also, an automated generator for GF arithmetic circuits would be implemented based on the proposed design method.

REFERENCES

[1] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[2] E. R. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," *Proc. of 32nd Design Automation Conf.*, pp. 535 – 541, 1995.

[3] Y. Chen and R. Bryant, "ACV: an arithmetic circuit verifier," *Proc. of the 1996 IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 361–365, 1997.

[4] R. Drechsler, Ed., *Advanced Formal Verification*. Kluwer Academic Publishers, 2004.

[5] R. Stankovic and R. Drechsler, "Circuit design from kronecker Galois field decision diagrams for multiple-valued functions," *Proc. 27nd IEEE Int. Symp. Multiple-Valued Logic*, pp. 275–280, 1997.

[6] 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*, vol. LNCS 2102, pp. 465–477, 2001.

[7] D. Mukhopadhyay, G. Sengar, and R. D. Chowdhury, "Hierarchical verification of Galois field circuits," *IEEE Trans. CAD.*, vol. 26, no. 10, pp. 1893–1898, 2007.

[8] K. Saito, N. Homma, and T. Aoki, "A graph-based approach to designing multiple-valued arithmetic algorithms," *Proc. 41st IEEE Int. Symp. Multiple-Valued Logic*, pp. 27–32, May 2011.

[9] D. A. Cox, J. B. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, 2nd ed. NY: Springer-Verlag, 1996, 536 pages.

[10] Y. Watanabe, N. Homma, T. Aoki, and T. Higuchi, "Application of symbolic computer algebra to arithmetic circuit verification," *Proc. 25th IEEE Int. Conf. Computer Design*, pp. 25–32, Oct. 2007.

[11] D. Page and N. P. Smart, "Hardware implementation of finite fields of characteristic three," *CHES 2002, Lecture Notes in Computer Science*, vol. 2523, pp. 529–539, Aug. 2002.

[12] B. Buchberger, "Some properties of Gröbner-bases for polynomial ideals," *SIGSAM Bull.*, vol. 10, no. 4, pp. 19–24, 1976.

[13] J. Lind-Nielsen, "Buddy: A bdd package," http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.itu.dk/research/buddy/index.html.

[14] A. M. D. Hankerson and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer Professional Computing, 2009.

[15] R. E. Bryant, "On the complexity of vlsi implementations and graph representations of boolean functions with application to integer multiplication," *IEEE Trans. Computers*, vol. 40, no. 2, pp. 205–213, Feb. 1991.