Towards Efficient Verification of Arithmetic Algorithms over Galois Fields $GF(2^m)$

Sumio Morioka, Yasunao Katayama, and Toshiyuki Yamane

IBM Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan
e02716@jp.ibm.com
Tel: +81-46-215-5736 Fax: +81-46-273-7413

Abstract. The Galois field $GF(2^m)$ is an important number system that is widely used in applications such as error correction codes (ECC), and complicated combinations of arithmetic operations are performed in those applications. However, few practical formal methods for algorithm verification at the word-level have ever been developed. We have defined a logic system, GF_{2^m} -arithmetic, that can treat non-linear and non-convex constraints, for describing specifications and implementations of arithmetic algorithms over $GF(2^m)$. We have investigated various decision techniques for the GF_{2^m} -arithmetic and its subclasses, and have performed an automatic correctness proof of a $(n, n-4)$ Reed-Solomon ECC decoding algorithm. Because the correctness criterion is in an efficient subclass of the GF_{2^m} -arithmetic (k -field-size independent), the proof is completed in significantly reduced time, less than one second for any $m \geq 3$ and $n \geq 5$, by using a combination of polynomial division and variable elimination over $GF(2^m)$, without using any costly techniques such as factoring or a decision over $GF(2)$ that can easily increase the verification time to more than a day.

1 Introduction

Due to the exponential growth of scale and speed of networks and computer systems, the importance and use of error correction codes (ECC) and crypto systems have been increasing rapidly. In the majority of these algorithms, the Galois field $GF(2^m)$ [1], or finite field, is used as a number system.

Because complicated combinations of arithmetic operations are performed in these algorithms, the necessity of applying formal verification to the entire algorithms at the word level (i.e., checking if the entire combination of operators is correct or not) is very high. The domain space of their inputs can be extremely wide, and therefore, ensuring the correctness of the entire algorithm is almost impossible by any testing-based method.

However, little research has ever been reported on the verification of arithmetic algorithms over Galois fields, although much research has been done for the other number systems such as integers (Presburger arithmetic etc.) [2,3], rational numbers [4], floating point numbers [5] and so on. A decision diagram for Galois fields, based on the decomposition of multiple-valued functions, has been proposed recently [6], but treating practical fields such as $GF(2^8)$, $GF(2^{16})$, $GF(2^{32})$ and larger and/or

treating practical fields such as $GF(2^8)$, $GF(2^{16})$, $GF(2^{32})$ and larger and/or algorithms (formulas) having many operators is still difficult.

In this paper, we investigate how to efficiently prove the correctness of practical algorithms over Galois fields. For describing algorithm specifications and implementations at the word level, we have defined a logic system, GF_{2^m} -arithmetic, that is a subclass of first-order logic and which can describe non-linear and non-convex constraints. This logic can treat the bit level descriptions, too.

We have performed a correctness proof of a key part of a practical $(n, n-4)$ Reed-Solomon ECC decoding algorithm [7,8,9], over $GF(2^8)$ ($m=8$) or larger. We have examined various decision techniques for the GF_{2^m} -arithmetic or its subclasses. Our first experimental result showed that even a small portion of the entire proof required too much CPU time (more than a day), if proof methods over $GF(2)$ such as decision diagrams (DDs) [10,11] were used. By using a special decision procedure based on a combination of polynomial division and variable elimination over $GF(2^m)$, without using any costly techniques such as factoring [1] and proof methods over $GF(2)$, the CPU time for the proof was significantly reduced to less than one second (Pentium III 800MHz), for any $m \geq 3$ and $n \geq 5$. One of the reasons why we could shorten the proof time significantly is that the correctness criterion for the RS-ECC verification is an efficient subclass of the GF_{2^m} -arithmetic: 2^p -field-size independent.

This paper is organized as follows. In Section 2, we will define a logic system, GF_{2^m} -arithmetic. In Section 3, we will investigate various decision techniques for the GF_{2^m} -arithmetic or its subclasses. In Section 4, we will show how we achieved a short verification time for a practical RS decoding algorithm.

2 A Logic System for Describing Specifications and Implementations of Arithmetic Algorithms over Galois Fields

2.1 The Language and Interpretation of GF_{2^m} -Arithmetic

We have defined a GF_{2^m} -arithmetic that is a subclass of the first-order logic. In any instance (sentence) of the GF_{2^m} -arithmetic, only arithmetic operators over Galois fields $GF(2^m)$ ($+$, $-$, $\times$, $\div$), equality ($=$) and logical operators ($\wedge$, $\vee$, $\neg$) can be used as functions or predicates. Please note that m is a given constant value and is not a variable. Symbols used in expressions in the GF_{2^m} -sentence are

$\wedge, \vee, \neg, \forall, \exists, =, +, \times, \div, -, 1, 0, x, y, z, \dots, P, Q, R, \dots, (,), "$.

Definition of the language is as follows:

1. GF_{2^m} variable and GF_{2^m} constant: $x, y, z, \dots$ are GF_{2^m} variables. Unlike integers, every constant value in $GF(2^m)$ has two representations [1]. The first one is the *exponentiation representation* where an element of $GF(2^m)$ is represented as one of $\{0, 1, \alpha^i\}$. Here, α is a *generator element* of a given field and i is an integer constant ($1 \leq i \leq 2^m - 2$). The second one is the *vector representation* where an element of $GF(2^m)$ is represented as a vector " $c_{m-1}, c_{m-2}, \dots, c_0$ " over $GF(2)$.
2. Term: Only variables, constants, $T_1 + T_2$, $T_1 - T_2$, $T_1 \times T_2$, and $T_1 \div T_2$ are terms, if T_1 and T_2 are terms.
3. Atom: An expression of the form $T_1 = T_2$ is an atom, where T_1 and T_2 are terms.

4. GF_2 variable and GF_2 constant: 1 and 0 are GF_2 constants (they are also elements of GF_{2^m} constants). $P, Q, R, \dots$ are GF_2 variables.
5. Formula: Only atoms, GF_2 variables, GF_2 constants, $F_1 \wedge F_2, F_1 \vee F_2, \neg F_1, \forall x F_1$ and $\exists x F_1$ are *formulas*, where F_1 and F_2 are formulas and x is a GF_{2^m} or GF_2 variable.
6. Sentence: A formula which has no free variable is a *sentence*. GF_{2^m} -arithmetic is the set of all sentences.

The interpretation of operators is defined once the field size m , irreducible polynomial and basis (polynomial basis, normal basis, etc. [1]) are fixed. The domain of GF_{2^m} variables is the entire field and the domain of GF_2 variables is $\{1, 0\}$. All of the above functions and predicates have their natural interpretations. In the following, we will use various standard abbreviations for simplicity: xy denotes $x \times y$, t^p denotes $\prod_{i=1}^p t$, $F_1 \Rightarrow F_2$ denotes $\neg F_1 \vee F_2$, $F_1 \Leftrightarrow F_2$ denotes $(F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2)$, if F_1 then F_2 else F_3 denotes $(F_1 \Rightarrow F_2) \wedge (\neg F_1 \Rightarrow F_3)$, and so on.

Examples: $\exists x \forall y (x^3 = x^2 y + x \Rightarrow \exists z (x + y = z))$ is a sentence. Neither $\exists x \exists y (x^y = 0)$ (a variable is used as the power for exponentiation) nor $\exists x (x = y)$ (y is a free variable) is a sentence.

2.2 Various Useful Subclasses of the GF_{2^m} -Arithmetic

In most of the actual verification applications, as will be described in Section 4, only limited sentences in the following subclasses of the GF_{2^m} -arithmetic appear. The relationships of the subclasses are shown in Fig.1.

A. Basis Independent Sentences.

These are sentences whose truth is independent of the representation basis. If a sentence contains no vector-constant (constant value in vector representation), then the sentence is basis independent. Usually, if a sentence contains multiplication by a vector-constant, it is basis dependent.

B. Irreducible Polynomial Independent Sentences.

These are sentences whose truth is independent of the irreducible polynomial. Usually, if a sentence contains addition by a constant value in an exponentiation representation, it is irreducible polynomial dependent.

Regarding A and B above, the following interesting theorem holds.

Theorem 1: If a sentence of the GF_{2^m} -arithmetic contains no GF_{2^m} -constant other than 0 or 1, then the sentence is basis and irreducible polynomial independent. $\square$

(Proof) For any fields A and B whose sizes are the same, there is an isomorphism function δ from the elements in A to the elements in B where $\delta(x+y) = \delta(x) + \delta(y)$, $\delta(xy) = \delta(x)\delta(y)$, $\delta(1) = 1$ and $\delta(0) = 0$. Consider a sentence S over the field A . Then, for any atom $term(x, y, z, \dots) = 0$ in S , a relation $term(x, y, z, \dots) = 0 \Leftrightarrow term(\delta(x), \delta(y), \delta(z), \dots) = 0$ holds, if S contains no GF_{2^m} -constant other than 0 and 1. Therefore, the truth of S is the same over the field B , by substituting $\delta(x), \delta(y), \delta(z), \dots$ into $x, y, z, \dots$. $\square$

Fig. 1. Relationships of subclasses A-E of the GF_{2^m} -Arithmetic

C. 2^p -Extension-Field-Size Independent Sentences.

A sentence is 2^p -extension-field-size independent, if there exists a constant $p \geq 1$ such that the truth of the sentence is the same over any extension fields of $GF(2^p)$, including $GF(2^p)$. For example, a sentence $\exists x(x^{16} = x \wedge x^4 \neq x)$ is 2^4 -extension-field-size independent. This sentence is true over $GF(2^4)$, $GF(2^{2 \times 4})$, $GF(2^{3 \times 4})$ and so on. Clearly, any 2^p -extension-field-size independent sentence is both basis and irreducible polynomial independent.

D. 2^p -Field-Size Independent Sentences.

This subclass is important because the correctness criteria for most of the arithmetic algorithms defined over Galois Fields are in this subclass.

A sentence is 2^p -field-size independent, if there exists a constant $p \geq 1$ such that the truth of the sentence is the same over all of $GF(2^i)$ ($i \geq p$). For example, a sentence $\forall x \forall y (y = 1 \Rightarrow x^2 = xy)$ is 2^2 -field-size independent. This sentence is false over $GF(2^2)$, $GF(2^3)$, $GF(2^4)$ and so on.

E. Field-Size Independent Sentences.

A 2^1 -field-size independent sentence is field-size independent. For example, a sentence $\forall x \forall y (y = x \Rightarrow x^2 = xy)$ is field-size independent.

F. $\exists$ -only (or $\forall$ -only) Prenex Normal Form Sentences.

One of the most important subclasses of the GF_{2^m} -arithmetic is the prenex normal form sentences where all of quantifiers are $\forall$. Any $\forall$ -only prenex normal form sentence can be transformed into $\exists$ -only sentences, by using the relation $\forall x p(x) \Leftrightarrow \neg \exists x \neg p(x)$.

3 Deciding the Truth Value of a Sentence of the GF_{2^m} -Arithmetic

3.1 Decidability of the GF_{2^m} -Arithmetic and the Problem of Deciding the Truth Value

Unlike integers or rational numbers, the truth value of any sentences S of the GF_{2^m} -arithmetic are decidable, even if S contains multiplications. This is because the domain of all the variables is finite. The truth of S can be determined by substituting all possible 2^m values into each variable in S . Clearly, the upper bound of computational complexity of this direct substitution method is $O(2^{mv_1} \cdot 2^{v_2})$, where v_1 is the number of GF_{2^m} variables and v_2 is the number of GF_2 variables (please note that the lower bound is not yet known). Therefore, the development of some decision heuristics are necessary.

However, in considering heuristics, we have found that interpreting both addition and multiplication at the same abstraction level can be a difficult problem. The reason is that interpreting multiplication over $GF(2^m)$ can be performed efficiently at the $GF(2^m)$ -level, which corresponds to the word level, using the exponential representation by adding the values of the exponents of the elements as natural numbers. On the other hand, interpreting addition over $GF(2^m)$ can be performed efficiently at the $GF(2)$ -level, which corresponds to the bit level, using the vector representation by adding corresponding vector elements over $GF(2)$ in parallel. If a sentence contains both addition and multiplication, converting the representation is necessary for evaluating the sentence, but conversion from the vector representation to the exponential representation is a discrete logarithm problem. The *Zech logarithm* is a known method to reduce the size of the addition-table, but only the reduction from $m2^{2m}$ to $m2^m$ is possible.

3.2 Basic Structure of Decision Procedures for the GF_{2^m} -Arithmetic

Following the discussions in Section 3.1, the basic structure of the decision procedure for the GF_{2^m} -arithmetic is as follows:

Step 1 (Decision over $GF(2^m)$): Simplify a given sentence using the well-known mathematical theorems of operators over $GF(2^m)$ (Table 1). Standard techniques of theorem proving (term rewriting, case analysis, etc.) and standard expression transformation rules over propositional logic and first-order logic can be used, too. How to apply those techniques and rules is highly application specific, and an example will be discussed in Section 3.3. In many cases, the truth can not be completely determined in this **Step 1** and if so, go to **Step 2**.

If a given sentence is basis or irreducible polynomial dependent, the truth of the sentence cannot be determined using only the rules in Table 1, because all of the rules in Table 1 are both basis and irreducible polynomial independent transformations. More concretely, sentences that contain constant values other than 0 and 1 usually require **Step 2**.

Step 2 (Decision over $GF(2)$): Apply the techniques in Section 3.4, or substitute all possible values into each variable and evaluate the truth of the sentence. When substitution is performed and m is large, avoiding conversion from vector

Table 1. Mathematical Theorems over $GF(2^m)$
(Basis and irreducible polynomials independent rules).

Field size independent rules			Field size dependent rule
$A + B = B + A$	$A \times B = B \times A$	$A \times (B + C) = (A \times B) + (A \times C)$	$A^{2^m} = A$
$A + (B + C) = (A + B) + C$	$A \times (B \times C) = (A \times B) \times C$	$A^2 + B^2 + \dots = (A + B + \dots)^2$	
$A + 0 = A$	$A \times 1 = A$	if $B \neq 0, A \div B = C \Leftrightarrow A = B \times C$	
$A + A = 0$	$A \times 0 = 0$	$A - B = A + B$	

representation to exponential representation by using the following steps will be good, even though each step may sometimes increase the size of the sentence rapidly:

- **Step 2a:** Transform all of the atoms in the sentence into sum-of-product form.
- **Step 2b:** Evaluate all multiplications over $GF(2^m)$ using the exponential representation.
- **Step 2c:** Convert all of the elements in the sentence from exponential representation to vector representation.
- **Step 2d:** Evaluate all addition over $GF(2)$ using the vector representation.

3.3 A Step 1 Implementation for ECC Algorithm Verification

In **Step 1** in Section 3.2, the efficiency of deciding the truth value is very dependent on how a given sentence is evaluated. In the following, we show an example implementation of **Step 1** as it will be used in Section 4.3.3. This example is suitable for word-level verification of an Reed-Solomon ECC, and is constructed so that **Step 2** is unnecessary when a complete (correct) ECC algorithm implementation is given.

As will be shown in Section 4.2, in general, the correctness criteria of a key part of ECC algorithms have the following characteristics:

- Described as $\forall$ -only prenex normal sentences, because the correctness criteria mean that output of the algorithm is correct for *any* input value.
- Often transformed into the form $\exists \text{ or } \forall (term = variable \wedge formula)$ or $\exists \text{ or } \forall (term \neq variable \wedge formula)$.
- Basis and irreducible polynomial independent because the correctness criteria satisfy Theorem 1 in Section 2.2. Therefore, **Step 2** may be unnecessary.
- 2^p -field-size-independent (actual value of p is different for different ECC codes). Therefore, no field-size dependent rule from Table 1 may be required.

Based on the above characteristics, we have implemented **Step 1** as follows:

Step 1a (Preparation): Transform the given sentence S into prenex normal form (usually unnecessary because the given sentence should already be in prenex normal form). Eliminate all $\neg$ by using De Morgan's Law and transform all of the atoms into the form $term = (\neq) 0$, where $term$ is in the sum-of-product form. Eliminate all division and unnecessary terms by using rules such as $T_1 = T_2 \div T_3 \Leftrightarrow T_1 \times T_3 = T_2$ (when $T_3 \neq 0$) and $T + T = 0$.

Step 1b (Eliminate Variables): Try to transform all of the atoms into the form $term = variable \ V$, where no V appears in T . If any atom is in the form $(T_1^2 + T_2^2 + \dots + T_k^2) = 0$, replace it by $T_1 + T_2 + \dots + T_k = 0$. If the form of the entire sentence S becomes $\exists or \forall (T = V \wedge formula)$, eliminate the variable V by substituting T into all occurrences of V in the sentence. After that, again transform all of the atoms into the form $term = (\neq) 0$ and then, the truth value of any atom that contains no variable can be determined.

Step 1c (Factoring and Construct Zero/Non-zero Set): Try factoring the left side of all atoms $term \neq 0$ (this factoring can be sometimes omitted, as shown in Section 4.3.3) [1]. If the entire sentence is of the form $\exists or \forall \{T_1 T_2 \dots T_k \neq 0 \wedge formula\}$, construct a set of non-zero terms $\Phi = \{T_1, T_2, \dots, T_k, \dots\}$ (otherwise, $\Phi = \emptyset$). Similarly, if the sentence is of the form $\exists or \forall \{U = 0 \wedge formula\}$, construct a set of zero terms $\Psi = \{U, \dots\}$.

Step 1d (Division by Polynomials in Zero/Non-zero Set): If the non-zero set Φ is not empty, test if the left side term of each atom can be factored by an element in Φ . If a term is factored by an element T_i , then replace the term by the quotient and append a new atom $T_i \neq 0$ to the entire sentence, connected by $\wedge$. In the same manner, test by the zero-set Ψ and if a term is factored by an element in Ψ , replace the term by 0.

Step 1e (Transform Expression to Make a New Zero/Non-zero Set): Iterate Steps 1b-1d until no change occurs. If the truth value of the entire sentence has been determined, terminate the procedure. Otherwise, apply the techniques (i) and (ii) below. If the form of the entire sentence becomes $\exists or \forall (term = variable \wedge formula)$ or $\exists or \forall (term \neq variable \wedge formula)$ as a result, then return to Step 1b. If not, go to Step 2 in Section 3.2.

(i) If S is a $\forall$ -only (or $\exists$ -only) prenex normal form sentence $\forall x \forall y \forall z \dots f(x, y, z, \dots)$, make a corresponding new $\exists$ -only (or $\forall$ -only) sentence $\exists x \exists y \exists z \dots \neg f(x, y, z, \dots)$.

(ii) Case analysis by variable: select the innermost quantified variable V (or any variable, when the sentence is $\forall$ -only or $\exists$ -only prenex normal form) and partition the entire proof into a case $V = 0$ and another case $V \neq 0$.

3.4 General Verification-Cost Reduction Techniques for Step 2

3.4.1. Field Change for Basis or Irreducible Polynomial Independent Sentences.

If a sentence is basis or irreducible polynomial independent, we can select an appropriate basis or irreducible polynomial at the verification stage, even if a different field is actually used for the algorithm implementation. The cost of evaluating sentences can be reduced by changing the basis or irreducible polynomial, because the cost of a multiplication can be reduced from $O(m^3)$ to $2m^2 - 1$ or fewer operations over $GF(2)$ [12].

3.4.2. Field Change for 2^p -(Extension)-Field-Size Independent Sentences.

If a sentence is 2^p -extension-field-size, 2^p -field-size or field-size independent, we can select a small field at the verification stage, for significant reduction of the variable domain space.

Fig. 2. Block Diagram of the One Shot Reed-Solomon Decoding Algorithm

3.4.3. Use of DDs for $\exists$ -only (or $\forall$ -only) Prenex Normal Form Sentences.

For $\exists$ -only (or $\forall$ -only) prenex normal form sentences S over $GF(2^m)$, the truth value can be determined as follows: (i) extract the entire S into an equivalent sentence T over $GF(2)$, by replacing all of the variables and operators by their bit-level implementations, (ii) transform T into a canonical form U over $GF(2)$ and (iii) check if U is a constant 0 (or 1). This is the same method used in model checking over Boolean variables [13] and is usually much faster than the direct substitution method.

4 Verification Example of a Reed-Solomon Decoding Algorithm

4.1 An Ultrafast Reed-Solomon Decoding Algorithm

The systematic (n, k) Reed-Solomon codes (RS codes), where n is the code word length and k is the information word (or message word) length, have a maximum error-correcting capability of $\lfloor (n - k)/2 \rfloor$ symbols. They are used in many areas, such as communication, storage, and fault-tolerant memory systems [7].

In [8,9], an ultrafast decoding algorithm for RS codes (one shot RS algorithm) was described. This one shot RS algorithm can exceed more than a Gbps of throughput when implemented in hardware. It is designed for combinational circuit implementation and does not use any iterative execution (loops).

We have selected this algorithm for our verification example, because applying formal verification is highly desired. The algorithm contains many operators and many mistakes could happen during algorithm implementation. In addition, avoiding loops is preferable for applying a formal verification procedure, although almost the same correctness criterion and decision procedure can be applied to other RS algorithms by adding some proof mechanisms that can handle loops [14].

Table 2. Correctness Criterion for $(n, n-4)$ RS Code (when # of errors = 2).

```

On a given field (given field size  $m$ , irreducible poly and basis),
 $\forall s_0 \forall e_0 \forall e_1 \forall s_1 \forall a_0 \dots$  (for any value of variables)
(
  -- input constraints (mathematical def. of syndrome  $s_0$ - $s_3$ )
   $s_0 = e_0 + e_1$  and  $s_1 = e_0 \cdot a_0 + e_1 \cdot a_1$ 
  and  $s_2 = e_0 \cdot a_0^2 + e_1 \cdot a_1^2$  and  $s_3 = e_0 \cdot a_0^3 + e_1 \cdot a_1^3$ 
  and  $e_0 \neq 0$  and  $e_1 \neq 0$  -- error values are not 0
  and  $a_0 \neq 0$  and  $a_1 \neq 0$  -- error locations are not 0
  and  $a_0 \neq a_1$  -- error locations are different

  -- algorithm formula (compute  $l_0$ - $l_2$  and  $er_0$ - $er_1$ , from  $s_0$ - $s_3$ )
  and  $l_0e_2 = s_0 \cdot s_2 + s_1^2$  and  $l_1e_2 = s_0 \cdot s_3 + s_1 \cdot s_2$ 
  and  $l_2e_2 = s_1 \cdot s_3 + s_2^2$  and  $l_1e_1 = s_0$ 
  and  $l_2e_1 = s_1$  and  $l_0 = l_0e_2$ 
  and if  $l_0e_2 = 0$  -- select an error locator polynomial by # of errors
    then  $l_1 = l_1e_1$  and  $l_2 = l_2e_1$ 
    else  $l_1 = l_1e_2$  and  $l_2 = l_2e_2$ 
  endif
  and  $er_0 = ((s_1 \cdot l_0e_2) / l_1e_2) + s_0$  and  $er_1 = ((s_0 \cdot l_0e_2) / l_1e_2)$ 
) imply (
  -- output specification (correct polynomials are obtained)
   $er_1 \cdot a_1 + er_0 = e_1$  and  $er_1 \cdot a_0 + er_0 = e_0$  -- error value poly.
  and  $l_0 \neq 0$  and  $l_1/l_0 = a_0 + a_1$  and  $l_2/l_0 = a_0 \cdot a_1$  -- error loc poly.
)

```

As shown in Fig.2, the one shot RS algorithm consists of four major blocks: (i) syndrome calculation, (ii) polynomials coefficients calculation, (iii) polynomial evaluation and (iv) error modification. The major difference between the one shot RS and the others is the computation sequence in the shaded block (ii). The most important and complicated part is also block (ii) and this is our verification target. The other blocks perform simple constant matrix multiplications [9], and checking if those matrices are correct or not is neither a difficult nor critical problem.

4.2 Correctness Criterion of the Reed-Solomon Decoding Algorithm

The verified second block computes, from 4 syndrome values S_0, S_1, S_2 and S_3 , the coefficients of the error locator polynomial (l_0, l_1 and l_2) and those of the error polynomial (er_0 and er_1). In the block, multiple error locator polynomials and error value polynomials that correspond to different numbers of errors (from 1 to $\lfloor (n-k)/2 \rfloor$) are computed, and one of them is selected as a result of evaluating the number of errors that actually occurred.

In Table 2, the correctness criterion for the second block for $(n, n-4)$ RS code is shown. What we have proved is that, assuming the multipliers and adders are correctly implemented in the $GF(2)$ -level (bit-level), the formula in the one shot decoding algorithm computes a correct output, if mathematically appropriate input is given. The criterion shown in Table 2 corresponds to the case when the number of errors is two, and it is necessary to independently prove a different case when the number of errors is one (the mathematical definitions of syndrome values are different between these cases).

The criterion is of the form

$$\forall \{ \{ input\ constraints \} \wedge \{ algorithm\ formula \} \Rightarrow \{ output\ specification \} \}$$

Table 3. Cost For proving Entire Correctness Criterion, over $GF(2^8)$ (Pentium III 800MHz).

Proof Method	CPU time
Approach 1: DD (OBDD or OFDD)	> 1 week
Approach 2: Variable elimination and DD	> 1 day (mostly spent by DD)
Approach 3: Polynomial division and variable elimination	0.2 second

and this statement has to be true under a given field size m , irreducible polynomial and basis. All of identifiers between operators are GF_{2^m} variables. All of variables are bounded by $\forall$ and the correctness criterion is a $\forall$ -only prenex normal form sentence.

The input constraints describe the standard mathematical definition of syndromes using the variables a_0, a_1 (error locations in the error word), e_0 and e_1 (error values) [7]. The output specification can be easily extracted from standard descriptions of the RS ECC algorithms. This part means that the obtained error locator polynomial $l_0x^2 + l_1x + l_2$ should be equivalent to $l_0(x - a_0)(x - a_1)$ and the obtained error value polynomial $Er(x) = er_1x + er_0$ should satisfy $Er(a_i) = e_i$ ($i = 0, 1$).

The same criterion can be used for any $n \geq 5, m \geq 3$, irreducible polynomial and basis.

4.3 Experimental Results

4.3.1 Approach 1: Direct Proof by DDs over $GF(2)$.

Because the correctness criterion is a $\forall$ -only prenex normal form sentence, we first tried to decide the truth of the entire sentence by DDs (see Section 3.4.3). We examined shared-OBDD [10] and shared-OFDD [11]. We thought that FDD would be efficient, because multiplication over $GF(2^m)$ (Mastrovito multiplier [12]) is usually implemented as a positive polarity Reed-Muller formula (PPRM) over $GF(2)$ [9,15]. However, too much proof time was necessary for those DDs (more than a week, when $m = 8$) because too many variables appeared in the correctness criterion.

4.3.2 Approach 2: Variable Elimination over $GF(2^m)$ and DDs over $GF(2)$.

To reduce the number of variables in the sentence, we tried a decision procedure that performs only **Steps 1a, 1b** and **1e** in Section 3.3. Using this procedure, the correctness criterion was transformed into a $\exists$ -only sentence $\exists(\text{term} = \text{variable} \wedge \text{formula})$, which should be false, and the variables $s_0, s_1, s_2, s_3, l_0e_2, l_1e_2, l_2e_2, l_1e_1, l_2e_1, er_0$ and er_1 were replaced by terms that consist of e_0, e_1, a_0 and a_1 . However, because the if-sentence in Table 2 still remains as well as the variables l_1 and l_2 (in the then/else clause of the if-sentence), it was not yet possible to determine the truth value of the entire sentence.

Therefore, we tried to evaluate the truth of the if-condition part which was rewritten as $e_0e_1a_0 + e_0e_1a_1 = 0$, by using DDs. More precisely, a simple $\exists$ -only sentence

$$\exists e_0 \exists e_1 \exists a_0 \exists a_1 ((e_0, e_1, a_0, a_1 \neq 0 \wedge a_0 \neq a_1) \Rightarrow e_0e_1a_0 + e_0e_1a_1 = 0)$$

was evaluated separately. However, too much time was still necessary over $GF(2^8)$ (Tables 3,4). Assuming the sentence is field-size independent, we tried to change the

Table 4. Cost for Evaluating the If-condition Part Using BDD/FDD (P-III 800MHz).

Field size and irreducible polynomial	# of bits	OBDD		OFDD	
		Max. BDD size	CPU time (sec)	Max. FDD size	CPU time (sec)
$GF(2^2), x^2+x+1$	8	62	< 1	40	< 1
$GF(2^3), x^3+x+1$	12	276	< 1	175	< 1
$GF(2^4), x^4+x+1$	16	1,145	< 1	871	< 1
$GF(2^5), x^5+x^2+1$	20	4,820	8	6,784	726
$GF(2^6), x^6+x^5+1$	24	18,520	359	> 50,000	> 22,000
$GF(2^7), x^7+x+1$	28	75,096	14,277	N/A	N/A

Table 5. Irreducible polynomial vs. max. BDD Size (over $GF(2^6)$) (P-III 800MHz).

Irreducible polynomial	Max. BDD size	CPU time (sec)
x^6+x^5+1	18,520	359
x^6+x+1	18,673	364
$x^6+x^5+x^3+x^2+1$	20,487	439
$x^6+x^5+x^4+x+1$	20,558	439
$x^6+x^5+x^2+x+1$	20,566	459
$x^6+x^4+x^3+x+1$	20,611	469

Table 6. Variable Ordering vs. max. BDD Size (over $GF(2^5), x^5+x^2+1$) (P-III 800MHz).

Variable ordering (from BDD top to leaf)	Max. BDD size	CPU time (sec)
a1(4)-a1(0), a0(4)-a0(0), e1(4)-e1(0), e0(4)-e0(0)	4,820	8
a1(0)-a1(4), a0(0)-a0(4), e1(0)-e1(4), e0(0)-e0(4)	4,775	8
e1(4)-e1(0), e0(4)-e0(0), a1(4)-a1(0), a0(4)-a0(0)	3,320	10
e1(4)-e1(0), a1(4)-a1(0), a0(4)-a0(0), e0(4)-e0(0)	12,280	57
a1(4)-a1(0), e1(4)-e1(0), a0(4)-a0(0), e0(4)-e0(0)	41,191	355

field size and proved over smaller fields, which was pretty effective for verification cost reduction (Table 4). However, we do not yet know any efficient formal method to prove if the sentence is (2^p -)field-size independent or not. Proof over a smaller field at least increases the probability of algorithm correctness.

We also examined changing the polynomial, because this sentence satisfies Theorem 1 in section 2.2, and it is a basis and irreducible polynomial independent sentence. As shown in Table 5, the proof time was slightly better when trinomials were used, because the fewer number of $GF(2)$ operators are used in a multiplier [12]. We examined various variable orderings of DDs, and found it was better to place variables that appeared in the same atoms close together in the ordering (Table 6).

4.3.3. Approach 3: Polynomial Division and Variable Elimination over $GF(2^m)$.

Because proof methods over $GF(2)$ were still too inefficient, we incorporated **Steps 1c** and **1d** into the decision procedure, in order to perform proof only over $GF(2^m)$. As mentioned in Section 3.3, **Step2** in Section 3.2 could be eliminated because the correctness criterion is basis and irreducible polynomial independent.

In these additional steps, a non-zero set $\{a_0, a_1, e_0, e_1, a_0+a_1\}$ was created, the if-condition part was divided (factored) by the elements of this set, an atom $1=0$ was obtained, and finally the truth of the if-condition part was determined. Then, the variables l_1 and l_2 were eliminated, and the entire sentence was transformed into $\exists\{(T \neq 0) \wedge \dots \wedge (T=0)\}$. After appending T to the non-zero set and dividing the atom $T=0$ by T , the truth of the entire sentence was determined.

In addition, we have found that the factoring in **Step 1c**, that is considered to be the most costly step, can be omitted from the verification process of general Reed-Solomon decoders, if their input/output specifications are described as functions of e_i and a_i . In a $(n, n-2t)$ Reed-Solomon code, the mathematical formula that evaluates the number of errors is $\prod_{0 \leq i < t} e_i \prod_{0 \leq i < j < t} (a_i + a_j) = 0$ [7], and this is logically equivalent to checking if the left term is factored by each element in $\{e_0, \dots, e_{t-1}, a_0 + a_1, \dots, a_{t-2} + a_{t-1}\}$. All of these terms already exist in the correctness criterion, and therefore, the factoring stage can be omitted. Although general RS decoding algorithms are implemented as functions of syndrome, the above mathematical formula is obtained from the correctness criteria, after eliminating the syndrome variables that are defined by e_i and a_i .

As a result, the proof time was shortened to 0.2 second (PentiumIII 800 MHz). This verification cost is the same for any $n \geq 5$, $m \geq 3$, irreducible polynomial and basis. The proof was fully mechanized.

5 Conclusion

In this paper we have defined a logic system for verification of arithmetic algorithms over Galois fields $GF(2^m)$, and various proof techniques were investigated. We have carried out a correctness proof of a practical $(n, n-4)$ Reed-Solomon decoding algorithm in less than one second for any $n \geq 5$ and $m \geq 3$, by using a decision procedure based on a combination of polynomial division and variable elimination over $GF(2^m)$.

One of the reasons why we could shorten the proof time significantly is that the correctness criterion is 2^p -field-size independent. If any one of these conditions were not satisfied, achieving efficient verification would be much more difficult, because a decision over $GF(2)$ would be necessary even if the sentence is written over $GF(2^m)$.

The verification of general (n, k) RS codes that can contain increasing numbers of operations can be done efficiently by using our approach. To the best of the authors' knowledge, our work is the first investigation of verifying a practical arithmetic algorithm over $GF(2^m)$ within a reasonable proof time.

References

1. I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone and T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic Publishers (1993).
2. W. Pugh "A practical algorithm for exact array dependence analysis," *Communications of the ACM*, 35(8), pp. 102-114 (1992).
3. S. Morioka, N. Shibata, T. Higashino and K. Taniguchi, "Techniques to Reduce Computation Time in Decision Procedure for Prenex Normal Form Presburger Sentences

- Bounded only by Existential Quantifiers," *Journal of IPSJ*, Vol. 38, No. 12, pp. 2419-2426 (1997).
4. N. Shibata, K. Okano, T. Higashino and K. Taniguchi, "A Decision Algorithm for Prenex Normal form Rational Presburger Sentences based on Combinatorial Geometry," *The 2nd Intl. Conf. on Discrete Mathematics and Theoretical Computer Science and the 5th Australian Theory Symposium (DMTCS'99+CATS'99)*, pp. 344-359 (1999).
 5. Y.A. Chen, R.E. Bryant, "Verification of Floating-Point Adders," *The 10th Intl. Conf. on Computer Aided Verification (CAV'98)*, pp. 488-499 (1998).
 6. R. Stankovic and R. Drechsler, "Circuit Design from Kronecker Galois Field Decision Diagrams for Multiple-valued Functions," *The Intl. Symp. on Multiple-Valued Logic*, pp. 275-280 (1997).
 7. S.B. Wicker and V.K. Bhargava (eds.), *Reed-Solomon Codes and Their Applications*, IEEE Press, 1994.
 8. Y. Katayama and S. Morioka, "One shot Reed-Solomon decoder," *33rd Annual Conf. on Information Science and Systems (CISS)*, pp. 700-705 (1999).
 9. S. Morioka and Y. Katayama, "Design Methodology for one-shot Reed-Solomon Decoder," *1999 IEEE Intl. Conf. on Computer Design (ICCD'99)*, pp. 60-67 (1999).
 10. R.E. Bryant, "Graph-based algorithms for Boolean function manipulations," *IEEE Trans. on Computers*, Vol. C-35, No. 8, pp. 677-691 (1986).
 11. U. Kebshull and W. Rosenstiel, "Efficient Graph-Based Computation and Manipulation of Functional Decision Diagrams," *European Design Automation Conf. '93*, pp. 278-282 (1993).
 12. A. Halbutogullar and C.K. Koc, "Mastrovito Multiplier for General Irreducible Polynomials," *IEEE Trans. on Computers*, Vol. 45, No.5, pp. 503-518 (2000).
 13. J.R. Burch, E.M. Clarke, D.E. Long, K.L. McMillan and D.L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 14, pp. 401-423 (1994).
 14. J. Kitamichi, S. Morioka, T. Higashino and K. Taniguchi, "Automatic Correctness Proof of Implementation of Synchronous Sequential Circuits Using Algebraic Approach," *Intl. Conf. on Theorem Provers in Circuit Design (TPCD'94)*, *LNCS Vol. 901*, pp. 165-184 (1994).
 15. T. Sasao, "AND-EXOR expressions and their optimization", in Sasao, editor: *Logic Synthesis and Optimization*, Kluwer Academic Publishers, pp. 287-312 (1993).