A Versatile Multi-Input Multiplier over Finite Fields

Haibo Yi, *Student Member, IEEE,* and Shaohua Tang*, *Member, IEEE* and Lingling Xu, *Member, IEEE*

Abstract—Multiplication of three elements over finite fields is used extensively in multivariate public key cryptography and solving system of linear equations over finite fields. This contribution shows the enhancements of multiplication of three elements over finite fields by using specific architecture. We firstly propose a versatile multi-input multiplier over finite fields. The parameters of this multiplier can be changed according to the requirement of the users which makes it reusable in different applications. Our evaluation of this multiplier gives optimum choices for multiplication of three elements over finite fields. Implemented results show that we takes 22.062 ns and 16.354 ns to execute each multiplication of three elements over $GF((2^4)^2)$ based on table look-up and polynomial basis on a FPGA respectively. Experimental results and mathematical proofs clearly demonstrate the improvement of the proposed versatile multiplier over finite fields.

Index Terms—versatile multiplier, multi-input multiplier, composite field, finite field, table look-up, polynomial basis, Field-Programmable Gate Array (FPGA).

1 INTRODUCTION

Multiplication over finite fields is one of the fundamental and important arithmetical operations in many engineering fields, especially in the area of cryptography.

Most of multipliers over finite fields can be grouped into three families: polynomial basis [1], [2], normal basis [3], [4] and dual basis [5], [6].

Besides, there has been a growing interest to implement the finite field arithmetic using composite field representations [7], [8], [9], which has been employed in cryptographic applications [10], [11] and coding technique [12]. Composite field is one of the specific form of finite fields. To some extent, composite field is a better choice for efficient implementation of many applications.

To our knowledge, all these multipliers are designed for multiplication of two elements. However, multiplication of three elements is playing a key role in cryptographic implementations and other mathematical problems.

Applications. One of the applications is to evaluate multivariate polynomials over finite fields. Generally, multivariate polynomial consists of multiplication of two elements, multiplication of three elements and addition over finite fields. For example, during the implementation of multivariate public key cryptography [13], evaluating multivariate polynomials over finite fields is one of the most time-consuming operations. The multivariate polynomial in Rainbow signature scheme [14], which is one of multivariate signature schemes, can be represented by the form

$$\sum_{i\in O_l, j\in S_l} \alpha_{ij}x_ix_j + \sum_{i,j\in S_l} \beta_{ij}x_ix_j + \sum_{i\in S_{l+1}} \gamma_i x_i + \eta. \quad (1)$$

Here are some definitions of the coefficients in (1). O_i is a set of Oil variables in the the $i-th$ layer; S_i is a set of Vinegar variables in the the $i-th$ layer. α, β and γ are coefficients; η is a constant. It can be observed from (1), computing multiplication of three elements is one of the important parts in the multivariate polynomials. Besides, evaluating multivariate polynomial is playing a key role in graphic computation [15], [16] and decoding of cyclic codes, Reed-Solomon codes [17].

Another application is to solve system of linear equations over finite fields. Gaussian elimination method [18] or Gauss-Jordan elimination method [19] can be employed to solve system of linear equations. These methods consist of pivoting, normalization and elimination. Let a_{ij} be the $i-th$ row and $j-th$ column in a matrix. a_{ii} is the pivot element. a_{kt} is eliminated via $a_{kt} = a_{kt} + a_{ii}{}^{-1}a_{it}a_{ki}$. It can be observed that since elimination includes a multiplication of three elements and an addition, three-input multipliers can be employed. Besides, since solving system of linear equations has a variety of applications, e.g. [20], [21], [22] and [23], three-input multipliers can be adopted in these applications.

However, to the best of our knowledge, since multipliers with three inputs have not been designed, mul-

- *Corresponding Author. Shaohua Tang is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, 510641.*
  E-mail: shtang@IEEE.org
- *Haibo Yi is with the South China University of Technology and University of Cincinnati.*
- *Lingling Xu is with the School of Computer Science and Engineering, South China University of Technology.*

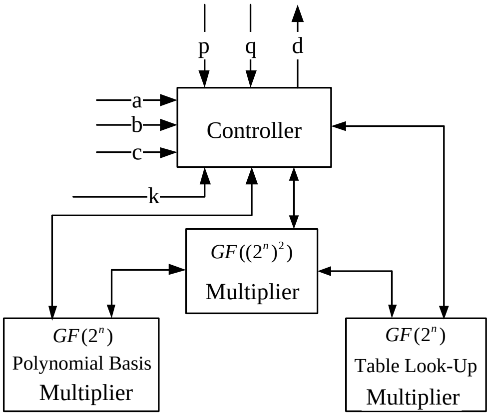

Fig. 1. The architecture of proposed versatile multiplier.

tiplication of three elements is computed by invoking multipliers with two inputs.

Our contributions. Therefore, we firstly propose a multi-input versatile multiplier in this paper. We enhance the multiplication of multiple elements in three directions. First, the proposed multiplier can execute multiplications over finite fields $GF(2^n)$ and composite fields $GF((2^n)^2)$. Second, the proposed multiplier can compute multiplications of two elements and three elements. Third, the proposed multiplier can perform multiplications based on table look-up and polynomial basis. By integrating these designs, the proposed versatile multiplier is very efficient for FPGA and VLSI implementation.

We test and verify our design on a FPGA and the experimental results show that the proposed versatile multiplier has a remarkable performance on computing multiplication of multiple elements. We also demonstrate the improvement of the multiplier mathematically.

The rest of this paper is organized as follows: In Section 2, a versatile multi-input multiplier is proposed. In Section 3, we evaluate the performance of the proposed multiplier by mathematical analysis. In Section 4, the proposed multiplier is implemented on a FPGA and the experimental results are analyzed. In Section 5, conclusions are summarized.

2 Versatile Multi-Input Multiplier

2.1 Overview of the Proposed Multiplier

The hardware architecture of proposed versatile multi-input multiplier is depicted in Fig. 1, which consists of four parts, i.e. controller, multiplier over $GF((2^n)^2)$, multiplier with polynomial basis over $GF(2^n)$ and multiplier with table look-up over $GF(2^n)$.

In the proposed versatile multiplier, multiplication of two elements $a \times b = d$ and multiplication of three elements $a \times b \times c = d$ over $GF(2^n)$ and $GF((2^n)^2)$ are performed respectively. $p(x) = x^n + p_{n-1}x^{n-1} + ... + p_1x + 1$ and $q(x) = x^2 + x + e$ are irreducible polynomials over $GF(2^n)$ and $GF((2^n)^2)$ respectively, where $p_{n-1}, p_{n-2}, ..., p_1$ are the elements over $GF(2)$ and e is an element over $GF(2^n)$.

According to different value of the signal k, the versatile multiplier can be reused for different applications, which is illustrated in Table 1. For example, when $k = (111)_2$, the proposed multiplier executes multiplication of three elements with table look-up over $GF((2^n)^2)$.

TABLE 1
The Versatile Multiplier for Different Parameters

k	Field	Operand	Method
$(000)_2$	$GF(2^n)$	Two	Polynomial basis
$(001)_2$	$GF(2^n)$	Two	Table look-up
$(010)_2$	$GF(2^n)$	Three	Polynomial basis
$(011)_2$	$GF(2^n)$	Three	Table look-up
$(100)_2$	$GF((2^n)^2)$	Two	Polynomial basis
$(101)_2$	$GF((2^n)^2)$	Two	Table look-up
$(110)_2$	$GF((2^n)^2)$	Three	Polynomial basis
$(111)_2$	$GF((2^n)^2)$	Three	Table look-up

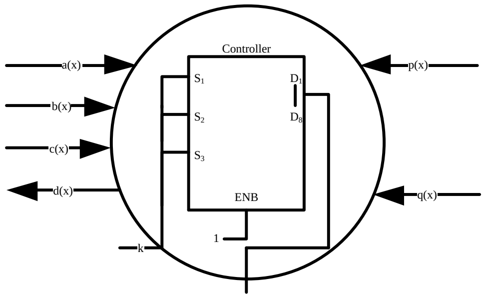

Fig. 2. Controller of the proposed versatile multi-input multiplier.

2.2 Controller

Controller of the proposed versatile multiplier is depicted in Fig. 2. According to different value of k, controller invokes different computational components for different applications.

Signal k is decoded by a 38-line decoder. Three bits of k are connected with S_1, S_2 and S_3 of the decoder respectively. D_1, D_2, ... D_8 are connected with the other computational components.

$a(x)$, $b(x)$, $c(x)$, $p(x)$ and $q(x)$ are directly sent to the computational components. If there exists inputs from the computational components, the controller sends these inputs to its output port $d(x)$.

2.3 Multiplier over $GF(2^n)$ with Polynomial Bases

In the multiplier over $GF(2^n)$ with polynomial basis, the design is based on [1]. Field element is expressed in the polynomial form and field multiplication can be performed in two steps. The first step is to perform the polynomial multiplication. The second step is to reduce modulo the irreducible polynomial $p(x)$.

Let $a(x)=\sum_{i=0}^{n-1} a_i x^i$ and $b(x)=\sum_{i=0}^{n-1} b_i x^i$ be elements in $GF(2^n)$, and

$$c(x)=a(x)\times b(x)(\mathrm{mod}(p(x)))=\sum_{i=0}^{n-1} c_i x^i \tag{2}$$

is the expected multiplication result, where $p(x)$ is the irreducible polynomial over $GF(2^n)$.

First, we compute v_{ij} for $i=0,1,...,2(n-1)$ and $j=0,1,...,n-1$ according to

$$x^i \bmod p(x)=\sum_{j=0}^{n-1} v_{ij} x^j. \tag{3}$$

This step can be pre-computed and v_{ij} is pre-stored in a look-up table.

Next, we compute S_i by AND logic gates for $i=0,1,...,2(n-1)$ via

$$S_i=\sum_{j+k=i} a_j b_k. \tag{4}$$

After that, we compute c_i by XOR logic gates for $i=0,1,...,n-1$ via

$$c_i=\sum_{j=0}^{2(n-1)} v_{ji} S_j. \tag{5}$$

Finally, the multiplication result is $c(x)=\sum_{i=0}^{n-1} c_i x^i$.

2.4 Multiplier over $GF(2^n)$ with Table Look-up

Multiplier over $GF(2^n)$ with table look-up is depicted in Fig. 3.

Table look-up is always adopted to accelerate computing multiplication. The number of elements in $GF(2^n)$ is 2^n. If α is chosen as the primitive element, all non-zero elements can be represented as a power of α. Hence these elements are stored sequentially in the look-up table. We store $(i,k_i(x))$ in the look-up table when $\alpha^i \ mod\ p(x)=k_i(x)$, where $p(x)$ is the irreducible polynomial over $GF(2^n)$.

We define a look-up table by Table 2, where $n=4$ and the irreducible polynomial $p(x)$ is x^4+x+1. Here are three examples for multiplications of two elements and three elements using table look-up respectively. Suppose the element in $GF(2^4)$ is represented as $(xxxx)_2$, where $x\in GF(2)$, i.e. $x=0$ or $x=1$.

Example 1. *We are going to compute* $(1000)_2\times(1100)_2$. *Since* $(1000)_2=\alpha^3$ *and* $(1100)_2=\alpha^6$, $(1000)_2\times(1100)_2=\alpha^3\times\alpha^6=\alpha^9$. *By looking up Table 2, we have* $\alpha^9=(1010)_2$. *Then,* $(1000)_2\times(1100)_2=(1010)_2$.

Example 2. *The second example is to compute* $(1011)_2\times(1101)_2$. *Since* $(1011)_2=\alpha^7$ *and* $(1101)_2=\alpha^{13}$, $(1011)_2\times(1101)_2=\alpha^7\times\alpha^{13}=\alpha^{20}=\alpha^{20\ mod\ 15}=\alpha^5$. *Looking up Table 2, we have* $\alpha^5=(0110)_2$. *Therefore,* $(1011)_2\times(1101)_2=(0110)_2$.

TABLE 2
Table Look-Up over $GF(2^4)$

Power Representation	Binary Representation
α^0	0001
α^1	0010
α^2	0100
α^3	1000
α^4	0011
α^5	0110
α^6	1100
α^7	1011
α^8	0101
α^9	1010
α^{10}	0111
α^{11}	1110
α^{12}	1111
α^{13}	1101
α^{14}	1001

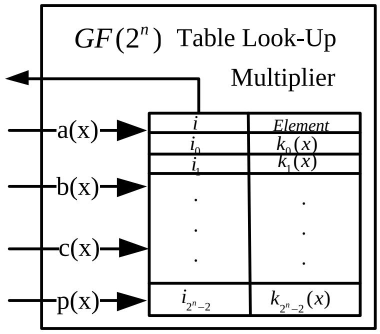

Fig. 3. Multiplier over $GF(2^n)$ with table look-up.

Example 3. *The third example is to compute* $(1000)_2\times(1011)_2\times(1101)_2$. *Since* $(1000)_2=\alpha^3$, $(1011)_2=\alpha^7$ *and* $(1101)_2=\alpha^{13}$, $(1000)_2\times(1011)_2\times(1101)_2=\alpha^3\times\alpha^7\times\alpha^{13}=\alpha^{23}=\alpha^{23\ mod\ 15}=\alpha^8$. *Looking up Table 2, we have* $\alpha^8=(0101)_2$. *Therefore,* $(1000)_2\times(1011)_2\times(1101)_2=(0101)_2$.

It can be observed from these examples that using table look-up in multiplication over $GF(2^n)$ is very efficient when executing multiplication of multiple elements simultaneously.

2.5 Multiplier over $GF((2^n)^2)$

In the multiplier over $GF((2^n)^2)$, there exists multiplication over subfields $GF(2^n)$, where multiplication with polynomial basis is computed by invoking the multiplier with polynomial basis over $GF(2^n)$ and multiplication with table look-up is computed by invoking the multiplier with table look-up over $GF(2^n)$.

Multiplication of Two Elements. Let $a(x)=a_h x+a_l$ and $b(x)=b_h x+b_l$ be the elements in $GF((2^n)^2)$, where a_h, a_l, b_h and b_l are elements in $GF(2^n)$.

Then the multiplication of two elements $a(x)$ and

$b(x)$ over $GF((2^n)^2)$ can be expressed as

$$\begin{aligned} a(x) \times b(x) &= (a_h x + a_l)(b_h x + b_l) \\ &= (a_h b_h x^2 + (a_h b_l + a_l b_h)x + a_l b_l) \mod q(x). \end{aligned} \tag{6}$$

We perform the polynomial multiplication and reduction module $q(x)$, where $q(x) = x^2 + x + e$ is an irreducible polynomial over $GF((2^n)^2)$ and e is a constant in $GF(2^n)$. Then we have

$$\begin{aligned} c_h &= a_h b_l + a_l b_h + a_h b_h, \\ c_l &= a_l b_l + a_h b_h e. \end{aligned} \tag{7}$$

Or

$$\begin{aligned} c_h &= (a_h + a_l)(b_h + b_l) + a_l b_l, \\ c_l &= a_l b_l + a_h b_h e. \end{aligned} \tag{8}$$

According to (7), the computational operations include five multiplications and three additions over $GF(2^n)$. (8) is an equivalent form of (7) and only four multiplications and four additions over $GF(2^n)$ are computed in (8). Usually one multiplication is more complex than one addition. Thus the latter is more efficient than the former. Hence, we adopt (8) to implement the multiplication of two elements over $GF((2^n)^2)$.

By observing (6) and (8), the critical path of multiplication of two elements over $GF((2^n)^2)$ includes one multiplication, one constant multiplication and one addition over $GF(2^n)$.

Multiplication of Three Elements. Suppose $a(x) = a_h x + a_l$, $b(x) = b_h x + b_l$ and $c(x) = c_h x + c_l$ are elements in $GF((2^n)^2)$.

$d(x) = a(x) \times b(x) \times c(x) \mod q(x)$ is computed via

$$\begin{aligned} d(x) &= (d_h x + d_l) \\ &= (a_h b_h c_h x^3 + (a_h b_l c_h + a_l b_h c_h + a_h b_h c_l)x^2 \\ &\quad + (a_l b_l c_h + a_h b_l c_l + a_l b_h c_l)x + a_l b_l c_l) \mod q(x), \end{aligned} \tag{9}$$

$$\begin{aligned} d_l &= e(a_h b_h c_h + a_h b_l c_h + a_l b_h c_h + a_h b_h c_l) + a_l b_l c_l, \\ d_h &= e a_h b_h c_h + a_h b_h c_h + a_h b_l c_h \\ &\quad + a_l b_h c_h + a_h b_h c_l + a_l b_l c_h + a_h b_l c_l + a_l b_h c_l. \end{aligned} \tag{10}$$

$q(x) = x^2 + x + e$ is the irreducible polynomial over $GF((2^n)^2)$, where e is a constant in $GF(2^n)$. Therefore, it can be observed that the critical path of multiplication of three elements over $GF((2^n)^2)$ includes one constant multiplication, one multiplication, and three additions over $GF(2^n)$.

3 Evaluation of Performance

In the proposed versatile multiplier, multiplications of two elements and three elements with polynomial basis over $GF((2^n)^2)$ are computed by invoking the two-input multiplier ($PB2$) and three-input multiplier ($PB3$) respectively. We can also compute multiplication of three elements over $GF((2^n)^2)$ by invoking these two multipliers respectively.

For presenting a mathematical analysis of the performance of the proposed multiplier, we give some definitions at first. M_2 stands for one multiplication of two elements over $GF(2^n)$, M_C stands for one constant multiplication over $GF(2^n)$, A stands for one addition over $GF(2^n)$ and M_3 stands for one multiplication of three elements over $GF(2^n)$.

In order to prove invoking three-input multiplier $PB3$ over $GF((2^n)^2)$ is faster than invoking two-input multiplier $PB2$ over $GF((2^n)^2)$ twice when computing multiplication of three elements over $GF((2^n)^2)$, we give Lemma 1 from the view of critical path.

Lemma 1. *When computing multiplication of three elements by invoking $PB2$, the critical path includes $2(M_2 + M_C + A)$. In the same condition, if we adopt $PB3$, the critical path includes $M_3 + M_C + 3A$. The critical path by invoking $PB3$ is shorter than the critical path by invoking $PB2$ when computing multiplication of three elements.*

Proof: First, the critical path of computing multiplication of two elements by invoking $PB2$ can be evaluated as follows,

$$T_2 = M_2 + M_C + A. \tag{11}$$

Second, the critical path of computing multiplication of three elements by invoking $PB2$ is double,

$$2T_2 = 2(M_2 + M_C + A). \tag{12}$$

Third, the critical path of computing multiplication of three elements by invoking $PB3$ can be evaluated as follows,

$$T_3 = M_3 + M_C + 3A. \tag{13}$$

The difference between $2T_2$ and T_3 is,

$$D = 2T_2 - T_3 = 2M_2 - M_3 + M_C - A. \tag{14}$$

Since the critical path of two multiplications of two elements is longer than the critical path of one multiplication of three elements, we have $2M_2 > M_3$. Moreover, since the critical path of one constant multiplication is much longer than the critical path of one addition, we have $M_C >> A$.

Therefore, $D >> 0$ has been proved. In other words, when computing multiplication of three elements, the critical path of using $PB3$ once is shorter than the one of using $PB2$ twice. □

4 Implementation and Experimental Analysis

4.1 Overview of Implementation

The proposed versatile multiplier can compute multiplication over $GF(2^n)$ and $GF((2^n)^2)$ respectively. We implement the proposed multiplier for different parameters, where the parameter $n = 2, 3, 4, 5, 6, 7, 9, 11, 13$ and 15, respectively. Our design is programmed in VHDL by using Quartus II, and implemented on a EP2S130F1020I4 FPGA device, which

TABLE 3
Executing Time of the Proposed Multiplications over $GF(2^n)$

n	$p(x)$	$GLUT2$(ns)	$GPB2$(ns)	$GLUT3$(ns)	$GPB3$(ns)
2	x^2+x+1	8.505	8.620	8.591	17.238
3	x^3+x+1	8.631	8.791	10.616	17.580
4	x^4+x+1	10.440	9.569	11.106	19.133
5	x^5+x^2+1	11.128	9.634	11.139	19.263
6	x^6+x+1	10.575	18.102	11.229	36.198
7	x^7+x+1	11.187	10.276	15.359	20.547
9	x^9+x^4+1	17.854	13.298	24.749	26.595
11	$x^{11}+x^2+1$	14.879	11.681	28.117	23.360
13	$x^{13}+x^4+x^3+x+1$	23.145	12.643	37.737	25.282
15	$x^{15}+x^4+x^2+x+1$	27.275	15.322	68.881	30.640

is a member of ALTERA Stratix II family. Furthermore, this design can be generalized to cover other devices of FPGA.

4.2 Implementation of Versatile Multiplier for Computing Multiplication over $GF(2^n)$

Table 3 depicts the performance of the proposed versatile multiplier when computing multiplication of two elements with table look-up over $GF(2^n)(GLUT2)$, multiplication of two elements with polynomial basis over $GF(2^n)(GPB2)$, multiplication of three elements with table look-up over $GF(2^n)(GLUT3)$ and multiplication of three elements with polynomial basis over $GF(2^n)(GPB3)$ respectively. An optimized irreducible polynomial $p(x)$ over $GF(2^n)$ is given for each field.

The comparison on the proposed multiplications over $GF(2^n)$ is given in Fig. 4, where (a) is the comparison of multiplication of two elements by invoking $GLUT2$ and $GPB2$ and (b) is the comparison of multiplication of three elements by invoking $GLUT3$ and $GPB3$.

In Fig. 4, the horizontal axis is the value of n and the vertical axis is the value of the executing time (ns) of multiplication. The curves of the figure of the multiplication over $GF(2^n)$ show that multiplication with table look-up is faster than multiplication with polynomial basis when field size is small and multiplication with polynomial basis is faster than multiplication with table look-up when field size is large.

4.3 Implementation of Versatile Multiplier for Computing Multiplication over $GF((2^n)^2)$

Table 4 depicts the performance of the proposed versatile multiplier when computing multiplication of two elements with table look-up over $GF((2^n)^2)(LUT2)$, multiplication of two elements with polynomial basis over $GF((2^n)^2)(PB2)$, multiplication of three elements with table look-up over $GF((2^n)^2)(LUT3)$ and multiplication of three elements with polynomial basis over $GF((2^n)^2)(PB3)$ respectively. An optimized irreducible polynomial $q(x)$ over $GF((2^n)^2)$ and an optimized irreducible polynomial $p(x)$ over $GF(2^n)$ are given for each field.

Example 4. *By observing Table 4, if we choose $LUT2$ over $GF((2^2)^2)$ to compute multiplication of three elements, 2×9.456 ns is required. However, if we choose $LUT3$ over $GF((2^2)^2)$ to compute multiplication of three elements, 10.229 ns is required. Since $2\times 9.456 > 10.229$, $LUT3$ is faster than $LUT2$ when computing multiplication over $GF((2^2)^2)$ of three elements.*

Example 5. *When computing multiple multiplications of three elements and multiple multiplications of two elements, e.g.*

$$\sum_{i=0}^{m-1}(\sum_{j=0}^{m-1}\alpha_{ij}x_ix_j)+\sum_{i=0}^{m-1}\beta_ix_i,$$

we can use $LUT3$ and $LUT2$ respectively to perform the computation.

If we only use $LUT2$, $2m^2+m$ multiplications of two elements are required. If we only adopt $LUT3$, m^2+m multiplications are required. According to experimental results, if the computation is performed over $GF((2^2)^2)$, the executing time by only using $LUT2$ is $9(2m^2+m)$ and the executing time by adopting $LUT3$ is $10(m^2+m)$.

Since $m\geq 1$, $18m^2+9m > 10m^2+10m$. It can be observed that when performing the computation, adopting $LUT3$ is faster.

Therefore, $LUT3$ and $PB3$ are faster than $LUT2$ and $PB2$ when computing multiplication over $GF((2^n)^2)$ of three elements.

The comparison on the proposed multiplications over $GF((2^n)^2)$ is given in Fig. 5, where (c) is the comparison of multiplication of two elements by invoking $LUT2$ and $PB2$ and (d) is the comparison of multiplication of three elements by invoking $LUT3$ and $PB3$.

In Fig. 5, the horizontal axis is the value of n and the vertical axis is the value of the executing time (ns) of multiplication. The curves of the figure of the multiplication over $GF((2^n)^2)$ show that multiplication with table look-up is faster than multiplication with polynomial basis when field size is small and multiplication with polynomial basis is faster than

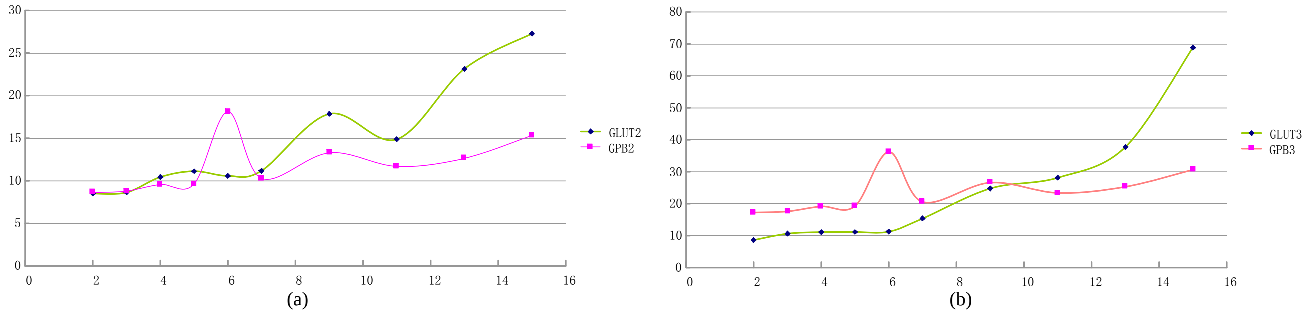

Fig. 4. Comparison on the proposed multiplications over $GF(2^n)$.

TABLE 4
Executing Time of the Proposed Multiplications over $GF((2^n)^2)$

n	$p(x)$	$q(x)$	$LUT2$(ns)	$PB2$(ns)	$LUT3$(ns)	$PB3$(ns)
2	x^2+x+1	x^2+x+3	9.456	12.608	10.229	13.530
3	x^3+x+1	x^2+x+5	10.304	12.226	13.215	14.873
4	x^4+x+1	x^2+x+9	11.422	12.842	22.062	16.354
5	x^5+x^2+1	x^2+x+8	15.072	13.792	16.653	17.706
6	x^6+x+1	x^2+x+33	14.535	14.698	21.234	20.425
7	x^7+x+1	$x^2+x+113$	15.563	14.689	21.698	20.989
9	x^9+x^4+1	x^2+x+32	19.621	16.143	27.215	23.268
11	$x^{11}+x^2+1$	$x^2+x+1367$	27.541	17.558	34.348	24.441
13	$x^{13}+x^4+x^3+x+1$	$x^2+x+3085$	30.133	19.125	36.794	27.642
15	$x^{15}+x^4+x^2+x+1$	$x^2+x+16395$	48.575	19.947	57.344	26.917

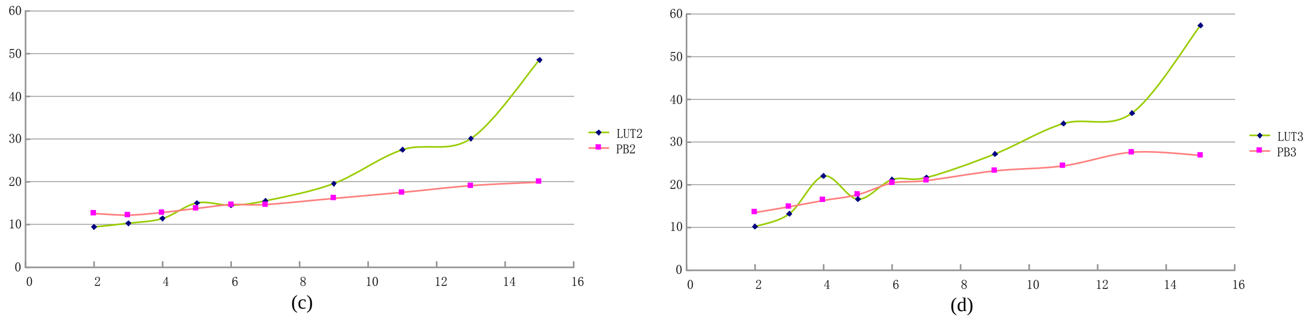

Fig. 5. Comparison on the proposed multiplications over $GF((2^n)^2)$.

multiplication with table look-up when field size is large.

4.4 Comparison of Versatile Multiplier for Computing Multiplication over $GF(2^n)$ and Multiplication over $GF((2^n)^2)$

We compare the versatile multiplier for computing multiplication over $GF(2^n)$ and multiplication over $GF((2^n)^2)$.

Suppose $GLUT2$, $GPB2$, $GLUT3$ and $GPB3$ are performed over $GF(2^m)$ and $LUT2$, $PB2$, $LUT3$ and $PB3$ are performed over $GF((2^{(m/2)})^2)$. Therefore, the size of $GF((2^{(m/2)})^2)$ with composite field expression equals the size of $GF(2^m)$ with finite field expression.

The comparison on multiplications over $GF(2^m)$ and $GF((2^{(m/2)})^2)$ is given in Fig. 6, where (e) is the comparison of the multiplications of two elements by invoking $GLUT2$, $GPB2$, $LUT2$ and $PB2$ respectively and (f) is the comparison of the multiplications of three elements by invoking $GLUT3$, $GPB3$, $LUT3$ and $PB3$ respectively.

In Fig. 6, the horizontal axis is the value of m and the vertical axis is the value of the executing time (ns) of multiplication. The curves of the figure show that $LUT2$ and $LUT3$ are faster than $GLUT2$ and $GLUT3$ and $GPB2$ and $GPB3$ are faster than $PB2$ and $PB3$. In other words, multiplication with table look-up is more efficient for composite fields and multiplication with polynomial basis is more efficient for finite fields.

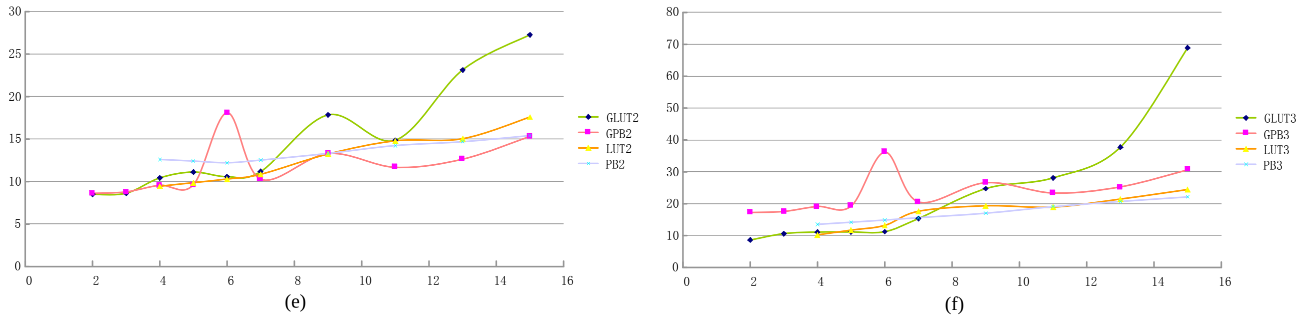

Fig. 6. Comparison on the proposed multiplications of two elements and three elements.

5 Conclusion

We propose a versatile multi-input multiplier, which can compute multiplications of two elements and three elements over finite fields $GF(2^n)$ and composite fields $GF((2^n)^2)$ based on polynomial basis and table look-up respectively. The parameters of the multiplier can be changed according to the requirement of the users which makes the multiplier reusable in different applications. The proposed versatile multiplier is very efficient for FPGA and VLSI implementation.

To the best of our knowledge, this is the first multiplier with multiple inputs. We demonstrate the improvement of our new designs mathematically. Then we implement the proposed multiplier on a FPGA and experimental results confirm our estimate.

The main characteristics of the proposed multiplier are presented as follows. First, when the field size is small, adopting multiplication with table look-up rather than polynomial basis is a better choice. Second, when the field size is large, adopting multiplication with polynomial basis rather than table look-up is more efficient. Third, multiplication with polynomial basis is more efficient for finite fields and multiplication with table look-up is more efficient for composite fields. Fourth, when computing multiplication of three elements, adopting three-input multiplier is more efficient than invoking two-input multiplier twice.

In addition, the proposed multiplier has a variety of applications, especially in cryptographic implementations and solving mathematical problems. Moreover, the proposed designs can be easily extended to $GF((2^n)^m)$.

Today's modern processors, such as Intel's and AMD's, and graphic processors, such as NVIDIA's and AMD/ATI's, provide tremendous abilities for parallel computing. We believe our design can easily carry over to modern processors as well as graphic processors for software implementation and achieve a good performance for software implementation.

6 Acknowledgement

This paper is supported by the National Natural Science Foundation of China under Grant No. U1135004 and 61170080, and Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2011), and Guangzhou Metropolitan Science and Technology Planning Project under grant No. 2011J4300028, and High-level Talents Project of Guangdong Institutions of Higher Education (2012), and the Fundamental Research Funds for the Central Universities under Grant No. 2009ZZ0035 and 2011ZG0015, and Guangdong Provincial Natural Science Foundation of under grant No. 9351064101000003.

References

[1] E. D. Mastrovito, "VLSI designs for multiplication over finite fields $GF(2^m)$," in *Proceedings of the 6th International Conference, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, 1989, pp. 297–309.

[2] G. Orlando and C. Paar, "A super-serial Galois fields multiplier for FPGAs and its application to public-key algorithms," in *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '99, 1999, pp. 232–.

[3] H. Fan and M. A. Hasan, "Subquadratic computational complexity schemes for extended binary field multiplication using optimal normal bases," *IEEE Trans. Comput.*, vol. 56, pp. 1435–1437, October 2007.

[4] c. K. Koç and B. Sunar, "Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields," *IEEE Trans. Comput.*, vol. 47, pp. 353–356, March 1998.

[5] H. Wu, M. A. Hasan, and I. F. Blake, "New low-complexity bit-parallel finite field multipliers using weakly dual bases," *IEEE Trans. Comput.*, vol. 47, pp. 1223–1234, November 1998.

[6] S. T. J. Fenn, M. Benaissa, and D. Taylor, "$GF(2^m)$ multiplication and division over the dual basis," *IEEE Trans. Comput.*, vol. 45, pp. 319–327, March 1996.

[7] C. Paar, "A new architecture for a parallel finite field multiplier with low complexity based on composite fields," *IEEE Trans. Comput.*, vol. 45, pp. 856–861, July 1996.

[8] S. Oh, C. H. Kim, J. Lim, and D. H. Cheon, "Efficient normal basis multipliers in composite fields," *IEEE Trans. Comput.*, vol. 49, pp. 1133–1138, October 2000.

[9] C. Paar, P. Fleischmann, and P. Roelse, "Efficient multiplier architectures for Galois fields $GF((2^4)^n)$," *IEEE Trans. Comput.*, vol. 47, pp. 162–170, February 1998.

[10] R. Schroeppel, H. Orman, S. W. O'Malley, and O. Spatscheck, "Fast key exchange with elliptic curve systems," in *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '95, 1995, pp. 43–56.

[11] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A lightweight high-performance fault detection scheme for the advanced encryption standard using composite fields," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 19, pp. 85–91, January 2011.

[12] C. Paar and M. Rosner, "Comparison of arithmetic architectures for Reed-Solomon decoders in reconfigurable hardware," in *Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines*, 1997, pp. 219–.

[13] J. Ding and D. Schmidt, "Multivariate public key cryptosystems," *Advances in Information Security*, 2006.

[14] ——, "Rainbow, a new multivariable polynomial signature scheme," pp. 317–366, 2005.

[15] M. Allard, P. Grogan, and J. David, "A scalable architecture for multivariate polynomial evaluation on FPGA," in *Proc. 2009 International Conference on Reconfigurable Computing and FPGAs*. IEEE, 2009, pp. 107–112.

[16] J. Rotgé and J. Farret, "Universal solid 3D format for high performance urban simulation," in *Proc. 2007 Urban Remote Sensing Joint Event*. IEEE, 2007, pp. 1–10.

[17] D. Schipani, M. Elia, and J. Rosenthal, "Efficient evaluations of polynomials over finite fields," in *Proc. 2011 Australian Communications Theory Workshop (AusCTW)*, Feb. 2011, pp. 154 –157.

[18] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[19] H. Hsieh, "Fill-in comparisons between Gauss-Jordan and Gaussian eliminations," *IEEE Trans. Circuits and Systems*, vol. 21, no. 2, pp. 230 – 233, Mar. 1974.

[20] S. Kim, K. Ko, and S.-Y. Chung, "Incremental Gaussian elimination decoding of raptor codes over BEC," *IEEE Communications Letters*, vol. 12, no. 4, pp. 307 –309, Apr. 2008.

[21] M. Atif and A. Rauf, "Efficient implementation of Gaussian elimination method to recover generator polynomials of convolutional codes," in *Proc. International Conference on Emerging Technologies (ICET)*, Oct. 2009, pp. 153 –156.

[22] C. Hou, X. Guo, and G. Wang, "Cluster based routing scheme for distributed regression in wireless sensor networks: Gaussian eliminations," in *Proc. 10th IEEE International Conference on High Performance Computing and Communications (HPCC)*, Sep. 2008, pp. 813 –818.

[23] V. Bioglio, M. Grangetto, R. Gaeta, and M. Sereno, "On the fly Gaussian elimination for LT codes," *IEEE Communications Letters*, vol. 13, no. 12, pp. 953 –955, Dec. 2009.