A Parallel Hardware Architecture for the Solution of Linear Equation Systems Implemented over $GF(2^n)$

Haibo Yi, Shaohua Tang, Huan Chen, Guomin Chen
School of Computer Science & Engineering
South China University of Technology
Guangzhou, China
haibo.yi87@gmail.com, shtang@IEEE.org, huangege@qq.com, sarlmolapple@gmail.com

***Abstract*—A parallel hardware architecture for the solution of linear equation systems implemented over finite fields is presented in this paper. This proposed hardware architecture could be efficiently employed in the Multivariate Public Key Cryptosystems for hardware implementation. A hardware-paralleled variant of the Gaussian elimination is adopted and is expressed as a series of multiplications with three inputs over finite fields, where the basic common computations can be shared to reduce computational complexity. Its average running time for solving a linear system $Ax = b$ equals n clock cycles, where A is a $n \times n$ matrix with uniformly distributed entries. In other words, it reduces time complexity to $O(n)$. Besides, its worst-case time is equal to its best-case one. The proposed architecture is implemented on a Field Programmable Gate Array. This implementation for solving a 12×12 linear system can be clocked with a frequency of up to 50 MHz and computed in 0. 24 us.**

***Index Terms*—Gaussian Elimination, Linear System of Equations (LSE), Forward Elimination, Field Programmable Gate Array ($FPGA$), Finite Field.**

I. Introduction

Solving linear equation systems is one of the fundamental operations in computational linear algebra. One of the several methods employed to solve linear system of equations (LSE) is known as Gaussian elimination [1]. Any LSE can be solved in at most cubic time using Gaussian elimination method. In other words, it has a worst-case time complexity of $O(n^3)$. As a result, it is not tolerable because of its time and space complexity.

The efficiency of solving linear equation systems is one of the focuses of many researchers. For instance, an efficient implementation of Gaussian elimination method is proposed to recover generator polynomials of convolutional codes in [2]. [3] analyzes the Gaussian elimination from a circuit point of view. Besides, a method solving linear systems through nested dissection is proposed in [4].

The design of novel hardware architectures for the solution of linear equation systems is an efficiently computational way in that $FPGAs$ have become capable of efficiently implementing large scale scientific applications. LU decomposition [5] is basically a modified form of Gaussian elimination. This decomposition is also used in numerical analysis to solve systems of linear equations. Some $FPGA$-based designs for floating-point LU decomposition have been proposed in [6], [7], [8], [9], [10], [11]. Matrix decomposition [12] is also used in solving $LSEs$. For instance, energy-efficient [13], systolic-array-based [14], [15], and circular-linear-array-based [16] architectures for matrix decomposition are also proposed. The parallel technology is adopted because it permit that methods for the solution of equations systems can be executed faster. For example, the parallel architectures applied to lineal algebra problems are proposed in [17].

We note that solving linear systems over finite fields has important applications in cryptography [18]. For instance, solving $LSEs$ over finite fields is the most computationally intensive operation in Multivariate Public Key Cryptosystems ($MPKCs$) [19], which are the cryptosystems that can resist the attacks by quantum computation. Since these $LSEs$ are not sparse but large dense ones over finite fields, efficient accomplishment of solving $LSEs$ would ensure the security level of these cryptosystems. In [20], it presents a hardware-optimized variant of the Gaussian elimination over $GF(2)$. Its average running time for $n \times n$ binary matrices equals $2n$ clock cycles and has a worst-case time complexity of O(n^2).

In this paper, we propose a parallel hardware architecture for the solution of linear equation systems over $GF(2^n)$. The proposed architecture is for the $MPKCs$ and efficiently employed in the Rainbow signature scheme [19]. Since that $MPKCs$ could efficiently work on $GF(2^n)$ and our propose architecture could accelerate the $MPKCs$ for hardware implementation. Brief introduction of $MPKCs$ is presented in the following section and it shows that solving $LSEs$ over finite fields is the most computationally intensive operation in the $MPKCs$.

Compared to the architecture in [20], our solving time for the same matrices equals n clock cycles and the time complexity is reduced to $O(n)$. Besides, the worst-case time complexity, as well as the best-case one, is $O(n)$. Moreover, the coefficient matrices of $LSEs$ are not required to have any special properties like being sparse.

The main contributions of this work are summarized as follows: 1) A novel parallel hardware architecture for the solution of linear equation systems over finite fields is proposed. The focus of this architecture is the idea that mixing the pivoting operations, the normalizing operations and the eliminating operations. Then we substitute a series of multiplications with three inputs for these operations. 2) Multiplications with three inputs over $GF(2^n)$ is first proposed and used in constructing an architecture for solving $LSEs$. 3) A new idea of using

forward elimination and backward substitution in a parallel way is proposed and used in building our architecture.

The rest of this paper is organized as follows: In Section 2, the basic notation used in this paper is presented. In Section 3, brief introduction of Rainbow signature scheme is presented and it shows that solving $LSEs$ over finite fields is the most computationally intensive operation in the $MPKCs$. In Section 4, it provides an overview of the native cubic implementation of Gaussian elimination over $GF(2^n)$. In Section 5, a brief introduction of our parallel architecture is presented. In Section 6, a new method for forward elimination is proposed and applied in building our architecture. In Section 7, a design of multiplication with three inputs is presented and applied in the forward elimination. In Section 8, a parallel method for backward substitution is proposed to accelerate the process of solving $LSEs$. The implementation is evaluated in the $QuartusII$ and compared with other hardware architectures in Section 9. Conclusions and discussions are summarized in Section 10.

II. Basic Notation

In order to maintain consistency, the following basic notations are used in this paper.

1) An element in the ith row and j^{th} column of a matrix A is denoted by a_{ij}.

2) The ith row of a matrix A is denoted by a'_i.

3) The ith column of a matrix A is denoted by a_i.

4) The proposed hardware architecture for the forward elimination is denoted by FE.

5) The proposed hardware architecture for the backward substitution is denoted by BS.

6) The proposed parallel hardware architecture for the solution of linear equation systems over finite fields is denoted by FB.

7) Gaussian elimination method is denoted by GE.

III. Overview of Rainbow Signature Scheme

A. General Construction of Rainbow

Rainbow belongs to the class of mixed schemes under MQ constructions. The scheme consists of a quadratic system of equations involving Oil variables and Vinegar variables that is solved iteratively. Such an Oil-Vinegar polynomial over these variables can be represented by the form,

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

O_i is a set of Oil variables in the the ith layer and S_i is a set of Vinegar variables in the the ith layer. The number of Vinegar variables used in the ith layer of signature construction is denoted by v_i. The number of Oil variables used in the ith layer of signature construction is denoted by $o_i = v_{i+1} - v_i$. The number of layers is denoted by u. The message is denoted by Y. The signature of Rainbow is denoted by X'.

Rainbow consists of four parts, private key, public key, signature generation and signature verification. Private key is used in signature generation and public key is used in signature verification.

B. Private Key

The private key consists of the maps L_1, L_2 and F, and is held by the signer. L_1: $k^{n-v_1} \rightarrow k^{n-v_1}$ and L_2: $k^n \rightarrow k^n$ are two randomly chosen invertible affine linear maps. F: $k^{n-v_1} \rightarrow k^{n-v_1}$ is a map consists of $n - v_1$ Oil-Vinegar polynomials. F has $u - 1$ layers of Oil-Vinegar construction. The first layer consists of o_1 polynomials where $\{x_i | i \in O_1\}$ are the Oil variables and $\{x_j | j \in S_1\}$ are the Vinegar variables. The lth layer consists of o_l polynomials where $\{x_i | i \in O_l\}$ are the Oil variables and $\{x_j | j \in S_l\}$ are the Vinegar variables.

C. Public Key

The public key consists of the field k and the $n - v_1$ polynomial components of $\bar{F}$. $\bar{F}$ is defined by the equation $\bar{F} = L_1 \circ F \circ L_2$.

D. Signature Generation

The message is defined by $Y = (y_1, ..., y_{n-v_1}) \in k^{n-v_1}$. The message size is fixed. Thus we use a hash function to produce a fixed-size one-way hash value of the document. After that, it is crucial to find a solution of the equation $\bar{F} = L_1 \circ F \circ L_2 = Y$.

In fact, ${L_1}^{-1}$ and ${L_2}^{-1}$ are used during the signature generation. Firstly, we should compute $\bar{Y'} = {L_1}^{-1}(Y)$.

Next the equation $\bar{Y'} = F$ is needed to be solved. In the first layer, the v_1 Vinegar variables in the Oil-Vinegar polynomials are randomly chosen. Therefore, these polynomials are substituted into a set of linear equations of Oil variables. If these equations have no solution, a new set of Vinegar variables should be chosen. The Vinegar variables in the second layer of Rainbow are all the variables in the first layer. The equations in the first layer are solved to find the values of all the variables. Thus the values of Vinegar variables are knew now. This procedure for each successive layer is repeated until the solutions of all the equations are found. If these equations have no solution, we have to return to the first layer and choose a new set of Vinegar variables. In this step, the values $\bar{X} = (\bar{x_1}, ..., \bar{x_n})$ of all the variables are found.

Finally, compute $X' = {L_2}^{-1}(\bar{X}) = ({x_1}', ..., {x_n}')$. Then X' is the signature of Y.

E. Signature Verification

To verify the authenticity of a signature X', we need to compute $\bar{F}(X') = Y'$. If $Y' = Y$ holds, the signature is accepted, otherwise rejected.

F. Gaussian Elimination in Rainbow

Gaussian elimination over the finite field is a key operation using in the signature generation and verification. It is used during finding an inverse matrix and center mapping. In the next section, an overview of native Gaussian elimination over finite fields is presented.

IV. Overview of Gaussian Elimination over Finite Fields

The purpose of our hardware architecture is to solve linear systems $Ax = b$ over finite fields. The first task is to transform $Ax = b$ into the equivalent system $A'x = b'$, where A' is an upper triangular matrix, by applying the forward elimination. The second task is to solve $A'x = b'$ by applying the backward substitution. In this paper, we propose a parallel hardware architecture to efficiently accomplish these two tasks.

Since a hardware-paralleled variant of the Gaussian elimination over $GF(2^n)$ is adopted in our design, in the following, we review the native Gaussian elimination method over $GF(2^n)$ at first.

A. Standard Gaussian Elimination over $GF(2^n)$

Generally, to solve a linear system $Ax = b$ where A is a $n \times n$ matrix and each element in this matrix is in $GF(2^n)$, Gaussian elimination over the finite field is adopted. It consists of four steps.

The first step is to choose a column which should be the current pivot column. Each nonzero element in this column can be chosen as the pivot element. The corresponding row is known as the pivot row. If the pivot element in this row is zero, this row could be interchanged with one in which the pivot one is nonzero. The worst case of this part is needed to search all the elements in this pivot column. This operation is known as the pivoting operation.

The next step is the normalizing operation. First we should find the multiplicative inverse of the pivot element. Then each element should be multiplied by the multiplicative inverse. This operation is known as the normalization.

The operation in this step is known as the eliminating operation. We choose a row a_i' in the matrix at first. If the current pivot column is a_j, we should compute $a_{ij} \otimes a_{j(j+1)} \oplus a_{i(j+1)} = a_{i(j+1)}$, where $\otimes$ is multiplication over finite fields while $\oplus$ is $XOR - operation$. Then choose the following element in this row, and do the same operation until the last element is chosen. After that, choose the next row to accomplish all the tasks in this step until the last row is chosen. Each element except the pivot element in the pivot column should be assigned zero. Then we choose the next column to repeat the above three steps to do all jobs until the last column is chosen.

The last step is to solve $A'x = b'$ using backward substitution. In the Section 7, a simple example for backward substitution is presented.

When these four steps are carried out, the linear systems $Ax = b$ has been solved. Therefore, the native Gaussian elimination has a worst-case running time of O(n^3). It is not tolerant because Gaussian elimination is often used.

B. Implementation and Evaluation

This Gaussian elimination is different form the normal one in that it is implemented over $GF(2^n)$. For the purpose of comparison, we use the software $QuartusII$ to implement the native Gaussian elimination algorithm. We choose $GF(2^8)$ as the finite field used in the following text. Experimental results show that the worst-case time for a 12×12 matrix equals 774 clock cycles. Also, in [20], its average running time for the same one equals 24 clock cycles and has a worst-case time complexity of $O(n^2)$. However, in the following section, we will present a hardware architecture FB and its solving time for the same matrix equals 12 clock cycles and has the time complexity of $O(n)$.

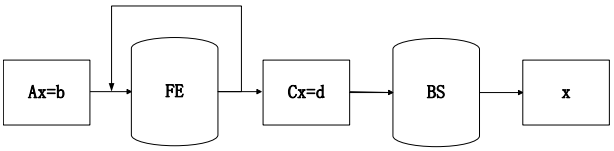

Fig. 1. Proposed parallel hardware architecture for solving $LSEs$

V. The Parallel Hardware Architecture for Solving $LSEs$

If we want to solve a linear system $Ax = b$, it should be transformed into the equivalent systems $Cx = d$ by means of forward elimination, where C is an upper triangular matrix. And then $Cx = d$ could be solved by applying backward substitution. The process is shown in Fig. 1.

Fig. 1 shows how we solve a linear system of equations $Ax = b$ in parallel, where A is a $n \times n$ matrix. At first, the input of FE is $Ax = b$. The output of FE is $A_1x = b_1$ after one clock cycle. A_1 is an equivalent matrix which the first row is normalized and each row except the first one is eliminated. Then $A_1x = b_1$ is the input of FE in the second clock cycle. Before the BS is applied, we iterate over all columns of the matrix A. After n iterations of this matrix, the $Ax = b$ is transformed into the equivalent systems $Cx = d$, where C is an upper triangular matrix. The running time of these n iterations equals n clock cycles. Since BS is a parallel hardware architecture, we could solve the linear system $Cx = d$ within only one clock cycle. FE is needed to be reused n times while BS is only used one time. As a result, all running time for solving a $n \times n$ matrix, equals to n clock cycles.

But there is some room for improving in the proposed hardware architecture. In the following section, a idea of using FE and BS in a parallel way is adopted so that the total running time could reduce to n clock cycles. The corresponding hardware structure applying this idea is shown in Fig. 2. The performance of hardware architectures could be measured by clock cycle and clock frequency. The minimum number of clock cycles and the maximum clock frequency are the design goal of our hardware architecture. We do not show how FE and BS work until now. Therefore, they will be specified in the following sections. FE is specified in Section 5 while BS is specified in Section 6.

VI. The Proposed Parallel Hardware Architecture for Forward Elimination

FE is used for forward elimination. In the hardware implementation, the output of last clock cycle is written in the matrix

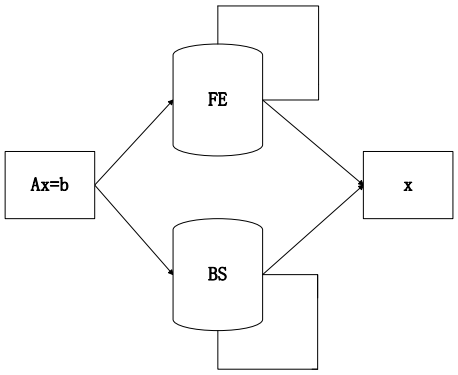

Fig. 2. Modified parallel hardware architecture for solving $LSEs$

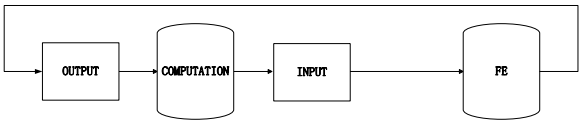

Fig. 3. Proposed parallel hardware architecture of FE

at the first half of this clock cycle. Since data dependency exists at each clock cycle, the new input is computed by the new matrix being updated. In the following time of this clock cycle, FE is performed with this input. At the end of this clock cycle, the new output is presented. According to this design, one iteration could be performed within only one clock. The process is shown in Fig. 3.

The design of FE architecture is shown in Fig. 4. Before sending elements as the new input, we should consider pivoting operation at first. It means that we should judge whether the pivot element is nonzero. We have to exchange rows to move the entry with the nonzero element to the pivot position in another clock cycle. In this paper, we propose a method to avoid such a knotty problem as follows.

Since the matrix after being updated in last iteration is needed, the normal pivoting operation is performed after the output is written in the matrix. But we start pivoting operation before updating the matrix in that we use the computing results instead of the updated matrix. Therefore it is convenient for the exchanging operation. Then the new input is computed according to the result of the pivoting operation.

After the pivoting operation is performed, the normalizing operation and eliminating operation could be considered. The input of normalization and elimination consists of the pivot row and the pivot column except the successive zero in the first half. The normalizing operation consists of one inversion and one multiplication. The eliminating operation consists of one multiplication and one addition. The native method is to perform normalization and elimination serially. Therefore, the running time of the Gaussian elimination is defined by the critical path including one inversion, two multiplications, and one addition. It means that the critical path must be reduced if we want to increase the efficiency of solving $LSEs$.

The idea proposed in this paper to reduce the critical path includes two parts. The first is to decompose the inversion into components. Another one is to perform normalizing operation and eliminating operation in parallel. A parallel design of inversion over finite fields based on Fermat's theorem [21] is presented in the following section.

A. A Parallel Design of Inversion over Finite Fields

Inversion over finite fields is used in Gaussian elimination to find the multiplicative inverse of a pivot element. Suppose $n(x)$ is the multiplicative inverse of $m(x)$, then $m(x)$ GF_mult $n(x)$ mod $f(x)$=1, where GF_mult is defined as multiplications with two inputs and $f(x)$ is the primitive polynomial over finite fields. The efficiency of inversion over finite fields has been one of the focuses of many researchers. For instance, some of the recent researches have been proposed in [22], [23], [24], and [25].

To achieve better performance, A parallel design of inversion over finite fields based on Fermat's theorem is proposed. Due to $GF(2^8)$ chosen as the finite field in the above test, we suppose β is an element in $GF(2^8)$.

$$\beta = a_7x^7+a_6x^6+a_5x^5+a_4x^4+a_3x^3+a_2x^2+a_1x^1+a_0x^0.$$

According to Fermat's theorem, we have

$$\beta^{2^8} = \beta,$$

$$\beta^{-1} = \beta^{2^8-2} = \beta^{254},$$

$$2^8 - 2 = 2 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7,$$

$$\beta^{-1} = \beta^2 * \beta^4 * \beta^8 * \beta^{16} * \beta^{32} * \beta^{64} * \beta^{128}.$$

Therefore, the computation of β^{-1} can be converted to the multiplication of $\beta^2, \beta^4, \beta^8, \beta^{16}, \beta^{32}, \beta^{64}$ and β^{128}. We have

$$\begin{aligned}\beta^{2^n} = {} & a_7x^{2^n*7} + a_6x^{2^n*6} + a_5x^{2^n*5} + a_4x^{2^n*4} + \\ & a_3x^{2^n*3} + a_2x^{2^n*2} + a_1x^{2^n*1} + a_0x^{2^n*0}.\end{aligned}$$

As a result, we can directly construct logic expressions of these seven items. The following expressions of β^{2^n}, where $n = 1, 2, 3, 4, 5, 6, 7$, are presented.

$$\begin{aligned}\beta^2 = {} & a_7x^{14} + a_6x^{12} + a_5x^{10} + a_4x^8 + \\ & a_3x^6 + a_2x^4 + a_1x^2 + a_0x^0.\end{aligned}$$

$$\begin{aligned}\beta^4 = {} & a_7x^{28} + a_6x^{24} + a_5x^{20} + a_4x^{16} + \\ & a_3x^{12} + a_2x^8 + a_1x^4 + a_0x^0.\end{aligned}$$

$$\begin{aligned}\beta^8 = {} & a_7x^{56} + a_6x^{48} + a_5x^{40} + a_4x^{32} + \\ & a_2x^{16} + a_1x^8 + a_0x^0.\end{aligned}$$

$$\begin{aligned}\beta^{16} = {} & a_7x^{112} + a_6x^{96} + a_5x^{80} + a_4x^{64} + \\ & a_3x^{48} + a_2x^{32} + a_1x^{16} + a_0x^0.\end{aligned}$$

$$\begin{aligned}\beta^{32} = {} & a_7x^{224} + a_6x^{192} + a_5x^{160} + a_4x^{128} + \\ & a_3x^{96} + a_2x^{64} + a_1x^{32} + a_0x^0.\end{aligned}$$

$$\begin{aligned}\beta^{64} = {} & a_7x^{448} + a_6x^{384} + a_5x^{320} + a_4x^{256} + \\ & a_3x^{192} + a_2x^{128} + a_1x^{64} + a_0x^0.\end{aligned}$$

$$\begin{aligned}\beta^{128} = {} & a_7x^{896} + a_6x^{768} + a_5x^{640} + a_4x^{512} + \\ & a_3x^{384} + a_2x^{256} + a_1x^{128} + a_0x^0.\end{aligned}$$

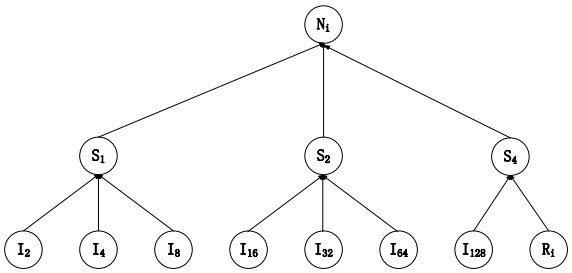

Fig. 4. Proposed parallel hardware architecture for normalization

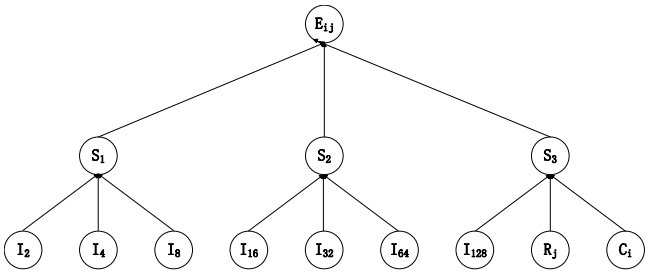

Fig. 5. Proposed parallel hardware architecture for elimination

$x^{2^n * i}$ could be computed modulo the primitive polynomial. Then β^{2^n} could be transformed into the equivalent form. For instance,

$$\beta^{2^n} = a'_7x^7 + a'_6x^6 + a'_5x^5 + a'_4x^4 + a'_3x^3 + a'_2x^2 + a'_1x^1 + a'_0.$$

After that, we could use multipliers over finite fields to get the result, namely the multiplicative inverse.

B. A Parallel Design of Normalization over Finite Fields

Normalization is one part of FE. To perform normalizing operation, the inversion of the pivot element is needed. As presented in the section above, the inversion could be expressed as

$$\beta^{-1} = \beta^2 * \beta^4 * \beta^8 * \beta^{16} * \beta^{32} * \beta^{64} * \beta^{128}.$$

The critical path of the normalization includes one inversion and one multiplication. It is possible to reduce the running time by means of efficient design. We adopt a method by decomposing the inversion into components. The architecture for normalization is showed in Fig. 4.

In Fig. 4, I_2 represents β^2, and I_4 represents β^4, ... , and I_{128} represents β^{128}. R_i represents the ith element in the pivot row. S_1 represents the product of the multiplication with three inputs, β^2, β^4 and β^8. S_2 represents the product of the multiplication with three inputs, β^{16}, β^{32} and β^{64}. S_4 represents the product of the multiplication with two inputs, β^{128} and R_i. N_i represents the result of normalization.

Now we define two multiplications over finite fields. $ThreeMult(v1, v2, v3)$ is multiplication with three inputs while $TwoMult(v1, v2)$ is multiplication with two inputs over finite field. According to this architecture, we have

$$S_1 = ThreeMult(I_2, I_4, I_8),$$

$$S_2 = ThreeMult(I_{16}, I_{32}, I_{64}),$$

$$S_4 = TwoMult(I_{128}, R_i),$$

$$N_i = ThreeMult(S_1, S_2, S_4).$$

Suppose $inver$ is the multiplicative inverse of the pivot element. These computations equal

$$N_i = inver \times R_i.$$

S_1, S_2 and S_4 can be executed simultaneously in parallel. Therefore, its normalizing results could be computed by one multiplication with three inputs as well as another one. As a result, it is safe to say the critical path of normalization only includes two multiplications with three inputs.

C. A Parallel Design of Elimination over Finite Fields

Elimination is another part of FE. Suppose C_i is the ith element in the pivot column, and N_j is the jth element in the pivot row after being normalizing. a_{ij} is the corresponding element in the matrix A and E_{ij} is the same element after being eliminated. To achieve the elimination, we only need to compute

$$E_{ij} = C_i \times N_j + a_{ij}.$$

But we do not use the above method to perform eliminating operations in that the eliminating operations have to be performed after the normalizing operations. We adopt a parallel way to perform normalizing and eliminating operations at the same time. The architecture for elimination is shown in Fig. 5.

In Fig. 5, C_i represents the ith element in the pivot column. S_3 represents the product of the multiplication with three inputs, β^{128}, R_i and C_i. E_{ij} represents the result of elimination.

According to this architecture, we have

$$S_1 = ThreeMult(I_2, I_4, I_8),$$

$$S_2 = ThreeMult(I_{16}, I_{32}, I_{64}),$$

$$S_3 = ThreeMult(I_{128}, R_j, C_i),$$

$$E_{ij} = ThreeMult(S_1, S_2, S_3),$$

$$E_{ij} = E_{ij} + a_{ij}.$$

S_1, S_2 and S_3 are computed in parallel. Therefore, its eliminating results could be computed by one multiplication with three inputs as well as another. As a result, it is safe to say the critical path of elimination only includes two multiplications with three inputs.

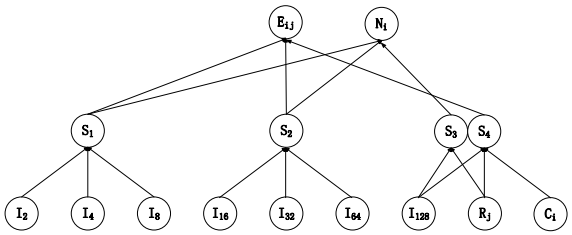

Fig. 6. Proposed parallel hardware architecture of FE

D. Overview of FE

The whole architecture of FE is shown in Fig. 6. It includes the normalizing and eliminating architectures. Those are performed in parallel. Therefore the critical path of FE includes two multiplications with three inputs. As a result, compared to the native design, the critical path of our design is shorter.

Now the efficiency of FE basically depends on the design of multiplications with three inputs over finite fields based on our architecture. In the next section, a new design of this multiplier is proposed.

VII. The Design of Multiplications with Three Inputs over Finite Fields

Actually the efficiency of the multiplication over finite fields has been one of the focuses of many researchers focus of many researchers. For instance, some of the recent researches have been proposed in [26], [27], and [28]. Since the most time consuming operation in implementing FE is multiplications with three inputs over finite fields, it is crucial to improve the efficiency of these multiplications. We name our design of multiplications with three inputs over finite fields $ThreeMult$. It is presented in the following section.

A. Proposed Algorithm

The multiplication over finite fields is different from the normal one, in that the product should be computed modulo the primitive polynomial over this field. Suppose $f(x)$ is the primitive polynomial over this field. Now we consider three elements in $GF(2^8)$ as the inputs of multiplications.

$$A(x) = a(7)x^7 + a(6)x^6 + a(5)x^5 + a(4)x^4 + a(3)x^3 + a(2)x^2 + a(1)x^1 + a(0)x^0.$$

$$B(x) = b(7)x^7 + b(6)x^6 + b(5)x^5 + b(4)x^4 + b(3)x^3 + b(2)x^2 + b(1)x^1 + b(0)x^0.$$

$$C(x) = c(7)x^7 + c(6)x^6 + c(5)x^5 + c(4)x^4 + c(3)x^3 + c(2)x^2 + c(1)x^1 + c(0)x^0.$$

$$D(x) = A(x) \otimes B(x) \otimes C(x) \bmod f(x).$$

$D(x)$ is the result of multiplication over the field. We can express $D(x)$ in the following form.

$$D(x) = d(7)x^7 + d(6)x^6 + d(5)x^5 + d(4)x^4 + d(3)x^3 + d(2)x^2 + d(1)x^1 + d(0)x^0.$$

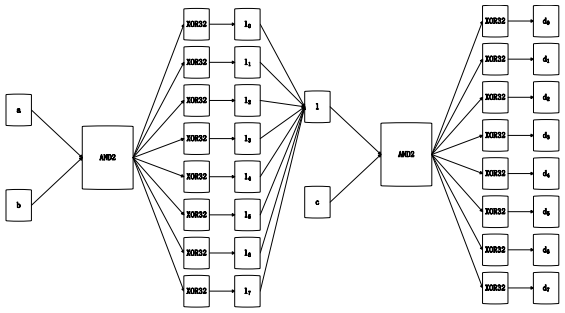

Fig. 7. Proposed parallel hardware architecture of $ThreeMult$

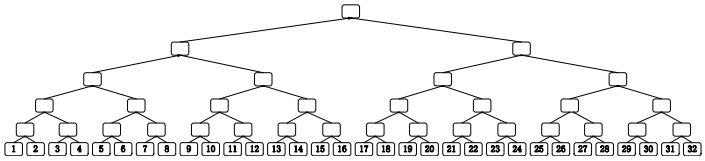

Fig. 8. Proposed parallel hardware architecture of $XOR32$

The design of $ThreeMult$ is shown in Fig. 7. Two basic operations are presented. The first is $AND2$. We need to compute $a_i \times b_j = s_{ij}$, where $i, j = 0, 1, ..., 7$. Since the inputs are bits, actually the multiplication is $AND-operation$. The second is $XOR32$. It is a parallel $XOR-operation$ shown in Fig. 8. Each input of $XOR32$ is one of the outputs of $AND2$. Each bit of the product could be computed by $XOR32$. The inputs of $XOR32$ are determined by the algorithm in [29]. Since the number of the elements needed to be computed is less than 32, $XOR32$ could be efficiently performed.

B. Implementation and Evaluation

Since we not only need a multiplier with three inputs but also a multiplier with two inputs, they are efficiently implemented. The critical path of a two-input multiplier, proposed $TwoMult$, includes $T_A + 5T_X$, where T_A is the delay of AND gate with two inputs while T_X is the delay of XOR gate with two inputs. Compared to other multipliers, our multiplier has a better performance shown in Table 1.

For the purpose of functional simulation and verification, we use the $QuartusII$ Version 8.0. For the $FPGA$ implementation, we choose the $ALTERA\ Stratix\ FPGA$ family. The data in Table 2 is tested on the device of EP2S180F1508C3 in $QuartusII$. A multiplication could be performed within only one clock cycle. According to the data in Table 2, $ThreeMult$ is more efficient and properly applied in FE.

TABLE I
Comparisons of Multipliers

Multiplier	Gate delay
Proposed $TwoMult$	$T_A + 5T_X$
$Karatsuba - Ofman$ [30]	$T_A + 6T_X$
$TMVP$ [31]	$T_A + 6T_X$
KOA [32]	$T_A + 8T_X$
CRT [33]	$9T_A + 8T_X$

TABLE II
EFFICIENCY OF MULTIPLIERS

Multiplier	Total Pins	Combinational $ALUTs$	Delay(ns)
$TwoMult$	24	44	10.663
$ThreeMult$	32	87	13.072

VIII. THE PARALLEL HARDWARE ARCHITECTURE FOR BACKWARD SUBSTITUTION

BS architecture is used for backward substitution. The process of solving a linear system of equations that has been transformed into row-echelon form [34]. The last equation is solved first, then the next-to-last, etc. In the following section, a simple example for backward substitution is presented.

A. An Example for backward substitution

Consider a system with the given row-echelon form for its augmented matrix. The equations for the system are

$$x + 2y + z = 16,$$

$$y + 6z = 18,$$

$$z = 2.$$

The last equation says $z = 2$. Substitute this into these equations to get

$$x + 2y + 2 = 16,$$

$$y + 6(2) = 18.$$

Then we have

$$y = 6.$$

This is the first step. And in the second step, we substitute $y = 6$ into these equations to get

$$x + 2(6) = 14.$$

Then we have

$$x = 2.$$

Thus the solution is $x = 2, y = 6, z = 2$.

B. Proposed BS

The focus of BS is parallel technology. From the above example, to solve a a $n \times n$ matrix, $n - 1$ steps are needed. Unfortunately, data dependency exists at each iteration. Thus these $n - 1$ steps are serially performed. But operations in each step could be executed in parallel. Operations in each step include one multiplication and one $XOR - operation$. Since $XOR - operation$ could be added into $XOR32$, the critical path in each step includes $T_A + 5T_X$. The inputs are the row-echelon form for the augmented matrix. After a step, the results of this step is related to the inputs of the next step. Thus these results are not immediately written in the matrix. The outputs are the solution of the linear system of equations.

Since we want to increase the efficiency of backward substitution, the idea of performing backward substitution and forward elimination in a parallel way is adopted. FE is executed as downward elimination. In the same way, BS can also be executed as upward elimination. The operations of downward elimination and upward elimination could be performed in a parallel way. In other words, the normalizing operation, the downward elimination and the upward elimination could be performed within a clock cycle. In this way, to solve a $n \times n$ matrix, n clock cycles are only needed.

TABLE III
$FPGA$ IMPLEMENTATION OF PROPOSED FB

Component	Number used	% Utilized
Combinational $ALUTs$	23773	17
Dedicated logic registers	1638	1
Total pins	98	8

TABLE IV
COMPARISON OF PROPOSED FB WITH OTHER IMPLEMENTATIONS

Scheme	Cycles
Native GE	128
Linear array [8] based real matrix	82
$G - SMITH$ [35]	16
Proposed FB	8

IX. IMPLEMENTATION AND COMPARISON

The construction of $VHDL$ modules for digital systems for the simulation of a parallel implementation is using the $QuartusII$ Version 8. 0. For the $FPGA$ implementation, we choose the $ALTERA\ Stratix$ II $FPGA$ family. EP2S180F1020C3 is chosen as the target device.

The device utilization summary is presented in Table 3. The number of clock cycles required for solving a 12×12 matrix is 12. In this case, its running time for a 12×12 matrix is 0.24 us with a frequency of 50 MHz.

We now compare some methods for solving a 8×8 LSE to our implementation in terms of performance. The comparison results are given in Table 4. It can be observed that our design has a good performance in clock cycles.

X. CONCLUSION AND DISCUSSION

For the purpose of efficient hardware implementation of $MPKCs$, a parallel hardware architecture FB for the solution of linear equation systems has been proposed in this paper. Its running time for solving a $n \times n$ linear system of equations equals n clock cycles. In the process of FE for forward elimination, Gaussian elimination method has been expressed as a series of multiplications with three inputs. Comparisons in this paper show that the implementation of $threemult$ as the multiplier used in our architecture, has a good performance. Besides, the operation of upward elimination is used in this hardware architecture and it could be executed in a parallel way.

Also, the idea presented in this paper could be applied to various applications including matrix factorization, matrix inversion and LU decomposition.

REFERENCES

[1] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[2] M. Atif and A. Rauf. Efficient implementation of gaussian elimination method to recover generator polynomials of convolutional codes. In *Emerging Technologies, 2009. ICET 2009. International Conference on*, pages 153 –156, 2009.

[3] F. Vacca and S. Vergura. The Gauss Elimination from the Circuit Theory Point of View: Triangular Nodal Equivalent. In *Computer as a Tool, 2005. EUROCON 2005. The International Conference on*, volume 1, pages 776–779. IEEE, 2006.

[4] Noga Alon and Raphael Yuster. Solving linear systems through nested dissection. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 225 –234, 2010.

[5] A. Schneider. Estimate of the number of arithmetic operations required in the lu decomposition of a sparse matrix. *Circuit Theory, IEEE Transactions on*, 17(2):269 – 270, May 1970.

[6] Y. Shao, L. Jiang, Q. Zhao, and Y. Wang. High Performance and Parallel Model for LU Decomposition on FPGAs. In *Frontier of Computer Science and Technology, 2009. FCST'09. Fourth International Conference on*, pages 75–79. IEEE, 2010.

[7] S. Choi and V. Prasanna. Time and energy efficient matrix factorization using fpgas. In *Field-Programmable Logic and Applications*, pages 507–519. Springer, 2003.

[8] G. Govindu, S. Choi, V. Prasanna, V. Daga, S. Gangadharpalli, and V. Sridhar. A high-performance and energy-efficient architecture for floating-point based LU decomposition on FPGAs. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 149. IEEE, 2004.

[9] V. Daga, G. Govindu, V. Prasanna, S. Gangadharapalli, and V. Sridhar. Efficient floating-point based block lu decomposition on fpgas. In *International Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas*, pages 21–24. Citeseer, 2004.

[10] Ling Zhuo and Viktor K. Prasanna. High-performance and parameterized matrix factorization on fpgas. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1 – 6, 2006.

[11] Ling Zhuo, Qingbo Wang, and V.K. Prasanna. Matrix computations on heterogeneous reconfigurable systems. In *Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on*, pages 310 –311, 2008.

[12] O.H. Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, 1982.

[13] J.W. Choi, J. Lee, B.G. Min, and J. Park. Energy Efficient Hardware Architecture of LU Triangularization for MIMO Receiver. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 57(8):632–636, 2010.

[14] A. Maltsev, V. Pestretsov, R. Maslennikov, and A. Khoryaev. Triangular systolic array with reduced latency for QR-decomposition of complex matrices. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, page 4. IEEE, 2006.

[15] DaeGon Kim and S.V. Rajopadhye. An improved systolic architecture for lu decomposition. In *Application-specific Systems, Architectures and Processors, 2006. ASAP '06. International Conference on*, pages 231 – 238, 2006.

[16] G. Govindu, S. Choi, V. Prasanna, V. Daga, S. Gangadharpalli, and V. Sridhar. A high-performance and energy-efficient architecture for floating-point based LU decomposition on FPGAs. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 149. IEEE, 2004.

[17] RM Alonso and DT Lucio. Parallel Architecture for the Solution of Linear Equation Systems Implemented in FPGA. In *Electronics, Robotics and Automotive Mechanics Conference, 2009. CERMA'09.*, pages 275–280. IEEE, 2009.

[18] J. Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[19] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in CryptologyłEUROCRYPT99*, pages 223–238. Springer, 1999.

[20] A. Bogdanov, MC Mertens, C. Paar, J. Pelzl, and A. Rupp. A parallel hardware architecture for fast Gaussian elimination over GF (2). In *Field-Programmable Custom Computing Machines, 2006. FCCM'06. 14th Annual IEEE Symposium on*, pages 237–248. IEEE, 2006.

[21] M.R. Schroeder and MR Schroeder. *Number theory in science and communication*. Springer, 1986.

[22] A. Mahboob and N. Ikram. Faster polynomial basis finite field squaring and inversion for gf(2m) with cryptographic software application. In *Biometrics and Security Technologies, 2008. ISBAST 2008. International Symposium on*, pages 1 –6, 2008.

[23] N. Jachimiec, N. Iliev, and J. Stine. Strategies for vlsi implementations of finite field inversion algorithms. In *Circuits and Systems, 2005. 48th Midwest Symposium on*, pages 1589 – 1592 Vol. 2, 2005.

[24] Yang Xiao-hui, Qin Fan, Dai Zi-bin, and Zhang yong fu. Scalable and unified hardware architecture for montgomery inversion computation in gf(p) and gf(2n). In *ASIC, 2009. ASICON '09. IEEE 8th International Conference on*, pages 843 –846, 2009.

[25] J.H. Jeng. Normal basis inversion in some finite fields. In *Signal Processing and Its Applications, 1999. ISSPA '99. Proceedings of the Fifth International Symposium on*, 1999.

[26] S. Talapatra and H. Rahaman. Low complexity montgomery multiplication architecture for elliptic curve cryptography over gf(pm). In *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, pages 219 –224, 2010.

[27] G. Perin, D.G. Mesquita, F.L. Herrmann, and J.B. Martins. Montgomery modular multiplication on reconfigurable hardware: Fully systolic array vs parallel implementation. In *Programmable Logic Conference (SPL), 2010 VI Southern*, pages 61 –66, 2010.

[28] S. Talapatra, H. Rahaman, and S.K. Saha. Unified digit serial systolic montgomery multiplication architecture for special classes of polynomials over gf(2m). In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 427 –432, 2010.

[29] LI Yue qiao and DU Man. A VHDL - based Design of a Fast Multiplier in $GF(2^8)$. *Telecommunication Engineering*, 44(005):148–152, 2004.

[30] H. Fan, J. Sun, M. Gu, and K.Y. Lam. Overlap-free Karatsuba-Ofman polynomial multiplication algorithms. *Information Security, IET*, 4(1):8–14, 2010.

[31] H. Fan and M.A. Hasan. A new approach to subquadratic space complexity parallel multipliers for extended binary fields. *IEEE Transactions on Computers*, pages 224–233, 2007.

[32] J. von zur Gathen and J. Shokrollahi. Fast arithmetic for polynomials over f2in hardware. In *Information Theory Workshop, 2006. ITW '06 Punta del Este. IEEE*, pages 107 –111, 2006.

[33] J.-C. Bajard, L. Imbert, and G.A. Jullien. Parallel montgomery multiplication in gf(2k) using trinomial residue arithmetic. In *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pages 164 – 171, 2005.

[34] E. Kreyszig. *Advanced engineering mathematics*. Wiley-India, 2007.

[35] S.R. Balasubramanian. *A Parallel Hardware Architecture for fast signature generation of Rainbow*. University of Cincinnati, 2007.