# VLSI Algorithms, Architectures and Implementation of a Versatile GF($2^m$) Processor

M.A. Hasan, *Senior Member, IEEE,* and A.G. Wassal, *Member, IEEE*

**Abstract**

With the explosive growth of electronic commerce, dedicated cryptographic processors are becoming essential since general-purpose processors can not provide the performance and functionality direly needed. This paper proposes an architecture for a versatile Galois field GF($2^m$) processor for cryptographic applications. This processor uses both canonical and triangular bases for field elements representation and manipulation. The variable dimension datapath of the processor is versatile enough to meet the varying requirements for different applications and environments. To provide flexibility for different cryptographic applications, an instruction set architecture is designed. Finally, a prototype VLSI implementation of the Galois field processor is presented and discussed.

**Index Terms**

Galois (or finite) field processor, cryptography, canonical (or polynomial) basis, triangular basis, datapath, VLSI implementation.

## I. Introduction

In the next few years electronic commerce (e-commerce) is expected to grow at an exponential rate to facilitate electronic financial transactions among a wide spectrum of users including banks, businesses and individuals over open networks. The use of the open networks, however, poses a variety of security threats concerning authentication, data integrity, confidentiality, etc. Security breeches in e-commerce may not only cause direct financial losses to the parties engaged in financial transactions, but also long term damages to the users' confidence in the underlying infrastructure. As a result, it is important to use appropriate security schemes to ensure adequate level of security in e-commerce [1, 2].

Many of the cryptographic security schemes that can be used in e-commerce or other applications rely on computations in the finite (or Galois) field GF($2^m$) which has $2^m$ elements and supports basic arithmetic operations under the closure condition [3]. These operations are different from the conventional integer and floating point operations and are not directly supported by today's general-purpose processors. As a result, GF($2^m$) based cryptographic schemes are implemented by emulating field operations on the general-purpose processors. This does not always give the desired performance in terms of the response time, especially when a very large finite field is to be used to achieve an increased level of security [4].

To speed up the computations in finite fields, a few dedicated processors and logic units were developed in the past [5–7]. Most of these processors operate over a fixed field. In other words, if there is a change in the field parameters (e.g., the field size or the irreducible polynomial defining the representation of the field elements), a new processor is needed. There are certain situations where different sets of parameters are to be used, for example, cryptographic schemes of different strengths are to be used based on domestic and international transactions. If the field parameters can be changed without switching to a new processor, it would increase the users' flexibility to use the same processor/device in a number of security environments and reduce the cost. The few publications that deal with varying field parameters include [8–10]. However, they mostly address field multiplication only and do not present a complete processor.

In this paper, we present a hardware architecture and the related VLSI implementation of a versatile processor for computations in the field GF($2^m$). This GF processor supports user specified field dimensions

Appeared in *IEEE Trans. Computers,* pp. 1064-1073, October 2000. This work was supported in part by NSERC, MICRONET and the Canadian Microelectronics Corporation.

M.A. Hasan is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada. A.G. Wassal was with the University of Waterloo and is now with PMC-Sierra, Burnaby, BC V5A 4V7, Canada. E-mail: {ahasan, wassal}@vlsi.uwaterloo.ca.

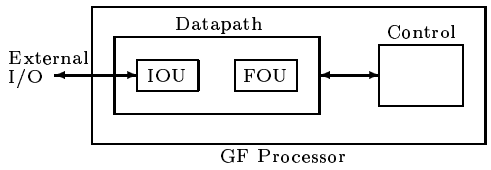



Fig. 1. Overall architecture for the GF processor

and field defining polynomials, and is capable of *directly* computing various field operations including inversion, multiplication, accumulation, and basis transformations. It can be used either as a stand alone arithmetic processor for computations in finite fields or as a coprocessor with other general-purpose or control processors for developing high speed encryption-decryption devices.

The outline of the paper is as follows. In the next section, the representation of the field elements in the GF processor is discussed and the processor core architecture is presented. In section III, the computational algorithms that led to the development of the processor and their operation are discussed. Section IV presents the GF processor datapath optimization techniques and its interface to external devices. An instruction set architecture supporting various finite field operations and a prototype VLSI implementation of the GF processor with some application examples are given in section V while some concluding remarks are given in section VI.

## II. Data Representation and Processor Core Architecture

The main building blocks of the GF processor are its datapath and control unit. The datapath is further divided into two units, namely, field operation unit (FOU) and input/output unit (IOU) as shown in figure 1. The FOU is to realize various arithmetic operations over a range of field dimensions and irreducible polynomials defining the representation of the field elements. The IOU is to provide a convenient mechanism for loading and unloading of data to and from the FOU. The IOU also brings in processor instructions which are decoded by the control unit.

### *A. Data Representation*

The area and time complexities of the FOU units are significantly affected by the way the field elements are represented. In the logarithmic representation, field elements are expressed as exponents of a primitive element. While this representation requires simple integer addition/subtraction modulo $2^m - 1$ for the field multiplication/inversion, it requires complicated discrete log and anti-log operations or special purpose hardware for the GF($2^m$) addition/subtraction. A log/anti-log module designed for a certain field cannot be directly used when the field parameters change. On the other hand, in the conventional representation the field elements are expressed as algebraic sums of a set of $m$ linearly independent elements of the field. The set is referred to as the *basis* of representation. For example, if all $\gamma_i \in$ GF($2^m$), for $0 \leq i \leq m-1$, are linearly independent, then the set $\Gamma = \{\gamma_0,\ \gamma_1,\ \cdots, \gamma_{m-1}\}$ forms a basis of GF($2^m$) over the ground field GF(2), and any element $\alpha$ of GF($2^m$) can be expressed as $\alpha = \sum_{i=0}^{m-1} a_i\gamma_i$ where $a_i \in$ GF(2). The term $a_i$ is referred to as the $i$th coordinate of $a$ with respect to basis $\Gamma$.

The bases which were assumed in the various realizations in the past are *normal*, *canonical* (also known as polynomial or standard), *triangular* and *dual* bases. Although normal bases offer an almost-cost-free squaring operation, they are not suitable for the design of a flexible processor because of their complex multiplication and inversion circuits for generic field parameters. Triangular and dual bases are quite similar in carrying out field operations and under certain conditions, these two types of bases are equal [11]. Basis transformations between the canonical and its triangular bases can be efficiently realized in hardware. In this paper, the canonical basis ($\Omega$) and the triangular basis ($\Lambda$) are used together. In the past, the combined use of two bases was found to yield efficient hardware for finite field arithmetic units [8, 11].

For $\beta \in$ GF($2^m$), a canonical basis has the form $\{1,\ \beta,\ \cdots, \beta^{m-1}\}$. If $F(x) = \sum_{i=0}^{m} f_i x^i$ is an irreducible binary polynomial of degree $m$ (i.e., $f_m = f_0 = 1$ and $f_i \in \{0,\ 1\}$ for $0 < i < m$) and has a root $\omega$ in GF($2^m$), then it is well known that the elements of the set $\{1,\ \omega, \cdots,\ \omega^{m-1}\}$ are linearly independent and hence form

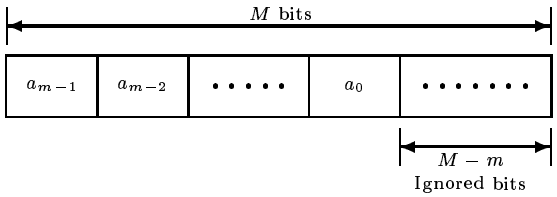



Fig. 2. Representation of the $\Omega$ coordinates of $a \in \mathrm{GF}(2^m)$ in an $M$-bit register. The $\Lambda$ coordinates have the same representation.

a canonical basis. Using this canonical basis as the primal basis of representation, a triangular basis, hereafter denoted as $\{\lambda_0,\ \lambda_1,\ \cdots,\ \lambda_{m-1}\}$, is derived as follows:

$$\lambda_i = \sum_{j=i}^{m-1} f_{j+1}\omega^{j-i} \qquad 0 \leq i \leq m-1 \tag{1}$$

where $f_i$'s are the coefficients of the irreducible polynomial $F(x)$. For an element $a \in \mathrm{GF}(2^m)$, its coordinates with respect to (w.r.t.) $\Omega$ and $\Lambda$ are denoted as $(a_{m-1}, a_{m-2}, \ldots, a_0)$ and $(\tilde{a}_{m-1}, \tilde{a}_{m-2}, \ldots, \tilde{a}_0)$, respectively. Given the $\Lambda$ coordinates of an element $a \in \mathrm{GF}(2^m)$, the forward transformation is referred to as the conversion of the coordinates to the corresponding $\Omega$ coordinates. The backward transformation is simply the reverse process. With regards to these transformations, we have the following [11]

$$a_{m-1-i} = \begin{cases} \tilde{a}_0 & i = 0 \\ \sum_{j=0}^{i} \tilde{a}_{i-j}\ f_{m-j} & 1 \leqslant i \leqslant m-1. \end{cases} \tag{2}$$

and

$$\tilde{a}_i = \begin{cases} a_{m-1} & i = 0 \\ a_{m-1-i} + \sum_{j=0}^{i-1} \tilde{a}_{i-1-j}\ f_{m-1-j} & \\ & 1 \leqslant i \leqslant m-1. \end{cases} \tag{3}$$

Appendix I shows the representation of the elements of an example field w.r.t. $\Omega$ and $\Lambda$.

While storing these coordinates in an $M$-bit register ($M \geqslant m$), it is advantageous to align them with the left or the right boundaries of the register. In this paper, these coordinates are assumed to be aligned to the left boundary and the rightmost $M - m$ bits are ignored as shown in figure 2. This kind of alignment enables us to serially load the coordinates into the register in $m$ clock cycles independent of the register size $M$.

### *B. Processor Core Architecture*

The core of the GF processor is the FOU. The FOU architecture is presented here while the algorithms and operations which led to the development of this FOU are given in section III. The FOU is the most space consuming unit in the GF processor and consists of a number of registers and logic blocks. The details of these blocks depend on the algorithms used to realize field arithmetic operations. An organization of the FOU using an $M+1$ bit bus is shown in figure 3. The leftmost 1-bit bus ($\mathrm{bus}_M$) is for bit serial operations while the $M$-bit ($\mathrm{bus}_{M-1,\ldots,0}$) is for parallel operations. The value of $M$ is the largest dimension the GF processor can support.

As shown in figure 3, the FOU has seven registers; namely, A1, A2, B1, B2, C, D and T. All registers, except for D, are $M$ bits long. Register D is $\lceil \log_2 M \rceil$ bits long and holds the binary representation of the current field dimension. Register C is the configuration register and holds the current irreducible polynomial which defines the representation of the field elements. Registers C and D are loaded with appropriate values only when the field parameters change. For $F(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$, the C register contents are as shown in figure 4, where the right-most $M - m$ bits are zero. These zero bits nullify the corresponding bits in other registers during certain arithmetic operations, which will be explained later in this article.

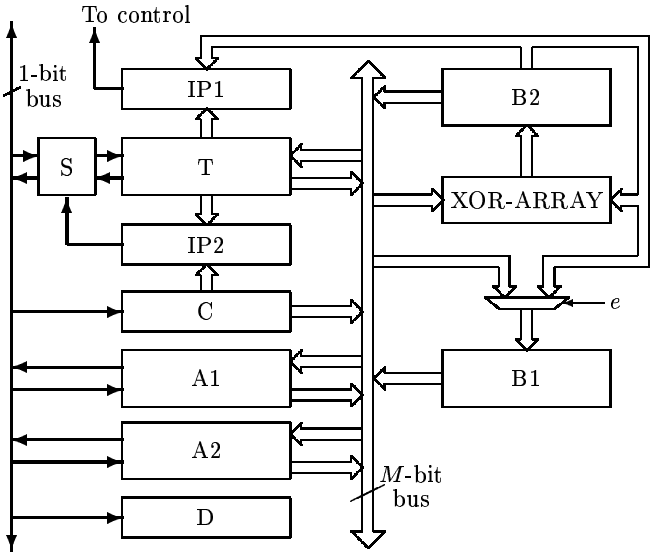



Fig. 3. Organization of the FOU in the GF processor.

Data can be loaded and unloaded in parallel to and from registers A1, A2, B1, B2 and T. Additionally, registers A1, A2 and T can be shifted in both directions. The FOU also has four logic blocks, namely, XOR-ARRAY, IP1, IP2 and S. The XOR-ARRAY block takes two $M$-bit inputs and produces an $M$-bit output which is the bit-wise ex-or of the inputs. Both IP1 and IP2 are inner product units, each of which takes two $M$-bit vectors and produces a one-bit output since the inner product is over GF(2). Depending on the direction of the data flow in the T register, IP1 produces the following outputs

$$IP1_{out} = \begin{cases} \sum_{i=0}^{M-1} T_i \; B2_i & \text{T in right shift mode,} \\ \sum_{i=0}^{M-1} T_{i-1} \; B2_i & \text{T in left shift mode.} \end{cases} \tag{4}$$

where $T_{-1} = 0$ and $T_i$ and $B2_i$ correspond to the $i$-th bit of registers T and B2 respectively. The output of IP2 is similar to the above equation for IP1 except that B2 is replaced by C.

The S block consists of a two-input XOR gate and a two-input multiplexer. It can be configured to provide input $T_{in}$ at the left end of register T and to place a single bit on the 1-bit bus ($\text{bus}_M$). The configuration is controlled by two signals; namely, $T_{modeCb}$ and $T_{inMUX}$, generated by the control unit. The operations of the S block, based on these signals, are as follows: If $T_{modeCb} = 0$, then

$$T_{in} = \begin{cases} \text{bus}_M + IP2_{out} & \text{if } T_{inMUX} = 0, \\ T_{special} + IP2_{out} & \text{if } T_{inMUX} = 1. \end{cases} \tag{5}$$

else if $T_{modeCb} = 1$,

$$\text{bus}_M = T_{out} + IP2_{out} \tag{6}$$

where $T_{special}$ is a control signal and $T_{out}$ is the bit serial output of T at the left end.

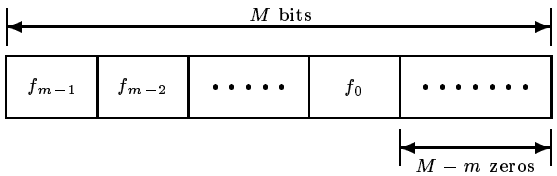



Fig. 4. Representation of the irreducible polynomial in the C register.

Registers T and C along with logic blocks IP2 and S form two special structures, namely, linear feedback shift register (LFSR) and feed-forward shift register (FFSR). When $T_{modeCb} = 0$ and the T register is shifted right, an LFSR is formed. On the other hand, when $T_{modeCb} = 1$ and the register T is shifted left, an FFSR is formed and the one bit output of the feed-forward operation is placed on the 1-bit bus given by

$$\text{bus}_M = T_{M-1} + \sum_{i=0}^{M-1} T_{i-1}C_i = T_{out} + IP2_{out} \tag{7}$$

where $T_i$ and $C_i$ correspond to the $i$-th bit of registers T and C respectively.

Another special structure in the FOU consists of register B1 and its multiplexing like input logic. Depending on the value of $e \in \{0, 1\}$, this structure operates as a buffer or a shift-right-with-one-and-store unit, i.e.,

$$\text{B1} = \begin{cases} \text{bus}_{M-1}, \ldots, \text{bus}_0 & \text{if } e = 0, \\ 1(B2_{M-1}, \ldots, B2_1) & \text{if } e = 1. \end{cases} \tag{8}$$

## III. Computing Algorithms and Their Operations

In this section, the algorithms which have led to the development of the FOU organization along with their operations are presented. Addition/subtraction over GF($2^m$) is simple bit-wise XOR of the field elements. In the FOU, an addition operation is performed using the XOR-ARRAY, register B2 (which acts as an accumulator) and another register (excluding D) in bit parallel fashion. On the other hand, GF($2^m$) inversion, multiplication and basis transformations are performed in bit serial fashion. The latter is essential to obtain an area efficient realization of the FOU.

### *A. Inversion*

Using the extended Euclidean algorithm, which is a well known method for area efficient hardware realization of an inverter over GF($2^m$) for an arbitrary $m$, one needs four or more $m$-bit registers. In a large field, the number of registers needed plays an important role in determining the size of the processor as well as the overall performance of a system which relies on the processor. If the algorithms take fewer registers, then we can either reduce the area of the processor or utilize the unused registers to store intermediate results and hence reduce loading and unloading operations. The latter can yield a considerable improvement in the performance of system level computations (e.g., to establish a shared secret key on an elliptic curve defined over a large finite field).

#### A.1 Algorithm

To compute the inverse of $a \in$ GF($2^m$) assume that

$$h_i = \begin{cases} a_{m-1} & i = 0 \\ a_{m-1-i} + \sum_{j=0}^{i-1} h_{i-1-j} f_{m-1-j} & \\ & 1 \le i \le m-1 \\ \sum_{j=0}^{m-1} h_{i-1-j} f_{m-1-j} & m \le i \le 2m-1 \end{cases} \tag{9}$$

Also assume that $\omega \in \mathrm{GF}(2^m)$ is a root of the irreducible polynomial $F(x)$, $b$ is the inverse of $a$ and $g_i$ is the $i$th coordinate of $g = b + \omega^m$ with respect to the canonical basis. Then from [12] one can write

$$\begin{bmatrix} s_0 & \cdots & s_{m-1} \\ s_1 & \cdots & s_m \\ \vdots & \ddots & \vdots \\ s_{m-2} & \cdots & s_{2m-3} \\ s_{m-1} & \cdots & s_{2m-2} \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{m-2} \\ g_{m-1} \end{bmatrix} = \begin{bmatrix} s_m \\ s_{m+1} \\ \vdots \\ s_{2m-2} \\ s_{2m-1} \end{bmatrix} \tag{10}$$

where

$$s_i = \begin{cases} h_i & 0 \le i \le 2m-2, \\ h_i + 1 & i = 2m-1. \end{cases} \tag{10a}$$

Equation (10) has a special structure and can be efficiently solved to obtain $g_i$'s using the Berlekamp-Massey algorithm [13]. We thus have the following algorithm to compute $a^{-1} = b = g + \omega^m$.

*Algorithm 1: (GF($2^m$) Inversion)*
*Input: a w.r.t. $\Omega$  Output: b w.r.t. $\Omega$*

*Step 1.1.*

$$\begin{aligned} P(x) &\triangleq \sum_{i=0}^{m} p_i x^i = 1, \\ Q(x) &\triangleq \sum_{i=0}^{m} q_i x^i = 1, \quad L = 0, \quad s_0 = a_{m-1} \end{aligned}$$

*Step 1.2. For $i = 1$ to $2m$ {*

$$\begin{aligned} d &= \sum_{j=0}^{L} p_j s_{i-1-j} \\ e &= \begin{cases} d & \text{if } i - 1 - 2L \ge 0 \\ 0 & \text{otherwise} \end{cases} \\ \begin{bmatrix} P(x) \\ Q(x) \end{bmatrix} &= \begin{bmatrix} 1 & d \\ e & 1-e \end{bmatrix} \begin{bmatrix} P(x) \\ xQ(x) \end{bmatrix} \\ L &= e(i-L) + (1-e)L \\ s_i &= \begin{cases} a_{m-1-i} + \sum_{j=0}^{i-1} s_{i-1-j} f_{m-1-j} \\ \qquad 1 \le i \le m-1 \\ \sum_{j=0}^{m-1} s_{i-1-j} f_{m-1-j} \\ \qquad m \le i \le 2m-2 \\ 1 + \sum_{j=0}^{m-1} s_{i-1-j} f_{m-1-j} \\ \qquad i = 2m-1 \\ \textit{Do not care} \\ \qquad i = 2m. \end{cases} \end{aligned}$$

}

*Step 1.3.*

$$\begin{aligned} g &= (p_1,\ p_2,\ \cdots,\ p_{m-1},\ p_m) \\ b &= (p_1 + f_{m-1},\ p_2 + f_{m-2},\ \cdots, \\ &\qquad p_{m-1} + f_1,\ p_m + f_0) \end{aligned} \qquad \square$$

An example showing the inversion operation using the above algorithm is given in appendix II.

A.2 Operation

In the above algorithm, $s_i, d$ and $e$ are either 0 or 1. Also, $P(x)$ and $Q(x)$ are polynomials over GF(2), while, $L$ is an integer. Thus, $P(x)$ and $Q(x)$ are updated using modulo 2 arithmetic and $L$ is updated using integer arithmetic. These two polynomials are stored in registers B2 and B1, respectively, in the FOU while $L$ is stored in the control unit.

Based on the properties of the Berlekamp-Massey algorithm [13], we have $\deg P(x) \leqslant m$. Thus, an $m+1$-bit register is needed to store $P(x)$ implying that register B2 should be $M+1$ bits long for the maximum value of $m = M$. This however causes an inconsistency in the lengths of the registers connected to the $M$-bit bus of the FOU. To eliminate this inconsistency, coefficient $p_0$ (i.e., the constant term of $P(x)$) is not stored in B2. This however does not result in any information loss since in step 1.2 $p_0$ is always '1'. However when $e = 1$, polynomial $Q(x)$ is updated to $P(x)$. In this updating process, coefficient $p_0 = 1$ is restored by the multiplexing-like-logic at the input of register B1.

At the end of the iteration process in step 1.2, the coefficients of $P(x)$ are essentially the coordinates of $g \in$ GF($2^m$) in some reverse order, specifically,

$$g_j = p_{m-j}, \qquad 0 \leqslant j \leqslant m-1. \tag{11}$$

thus, $P(x)$ and $Q(x)$ are stored left-adjusted in registers B2 and B1 respectively starting from their lowest order coefficients (i.e., $p_1$ for $P(x)$ and $q_0$ for $Q(x)$).

In the inversion algorithm, the sequence $\{s_i\}$ is generated from the coordinates of $a$ using the LFSR structure consisting of T, C, S and IP2 in the FOU. Assuming that the canonical basis coordinates of $a$ are stored in register A$i$, $i = 1, 2$, if we right-shift these coordinates into logic S via the 1-bit bus, then $s_0, s_1, \ldots, s_{m-1}$ are generated at the output of S and enter T in bit serial fashion. In the next $m-1$ clock cycles, $s_m, s_{m+1}, \ldots, s_{2m-2}$ are generated simply by right-shifting T with $T_{special} = 0$ and $T_{inMUX} = 1$ at the inputs of S. The term $s_{2m-1}$ is generated with one more right-shift of T with $T_{special} = 1$ and $T_{inMUX} = 1$ at the inputs of S. Since B2 contains the coefficients of $P(x)$ and T contains the sequence $\{s_i\}$, the inner product $d$ of the inversion algorithm is obtained by combining the output of IP1 of the FOU to $T_{in}$.

A.3 Remarks

• The first $m$ elements of the $\{s_i\}$ sequence in the above inversion algorithm are essentially the $\Lambda$ coordinates of $a$, i.e.,

$$s_i = \tilde{a}_i \qquad 0 \leqslant i \leqslant m-1. \tag{12}$$

Thus, the inversion algorithm and its operation on the FOU are also applicable to the $\Lambda$ representation of $a$ with only minor modifications. In either basis representation, the inverse of $a$ is however obtained with respect to $\Omega$.

• The element $a$ whose inverse is to be determined does not need to be in one of the registers of the FOU. Element $a$ can be directly brought into the LFSR structure in bit serial fashion to form the $\{s_i\}$ sequence and to apply the inversion algorithm. In some applications, this will save $m$ clock cycles needed to buffer $a$.

• As it can be seen from the operation of the inversion algorithm, the main components for the inversion operation are three registers[1] (viz., T, B1 and B2) and three logic blocks[2] (viz., XOR-ARRAY, IP1 and IP2). Instead of using the above Berlekamp-Massey algorithm based inversion, if the extended Euclidean algorithm based inversion (or its variation such as the almost inverse algorithm [14]) is used then one will need four

[1] We have not counted C since it is in the FOU anyway for configuration purposes.

[2] We have ignored the logic block S since it is a simple structure and does not depend on the value of $m$.

(or more) registers and two XOR-ARRAYs. Although, both algorithms take $2m$ clock cycles per inversion, the former has a longer critical path of $\lceil \log_2 M \rceil D_X + D_A$ because of the inner product unit ($D_X$ and $D_A$ correspond to the delays due to an XOR and AND gates respectively). Variations of the Berlekamp-Massey exist where the inner product can be avoided at the expense of other circuits [15]. The inner product units, however, can be useful in implementing other finite field arithmetic operations such as multiplication and basis transformations which are discussed below.

*B. Multiplication*

B.1 Algorithm

Let $u$ be the product of $a$ and $b$ where $a, b, u \in \mathrm{GF}(2^m)$. Conventional finite field multipliers, where both the multiplicand ($a$) and the *multiplier* ($b$) are represented w.r.t. $\Omega$, have logic circuit, consisting of XOR and AND gates, at the input of each stage of the register which generates the partial products [16]. The presence of the logic circuits limits the usage of the register as a general purpose buffer whose contents could otherwise be shifted in both directions, a feature a designer would like to have in the processor on which larger systems may be built. Keeping this in mind, if one applies equation (9) and (10$a$) to Proposition 1 of [11], the following is obtained.

$$\begin{bmatrix} s_0 & \cdots & s_{m-1} \\ s_1 & \cdots & s_m \\ \vdots & \ddots & \vdots \\ s_{m-2} & \cdots & s_{2m-3} \\ s_{m-1} & \cdots & s_{2m-2} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-2} \\ b_{m-1} \end{bmatrix} = \begin{bmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_{m-2} \\ \tilde{u}_{m-1} \end{bmatrix} \tag{13}$$

Using this type of matrix equation, the coordinates of the product can be generated in two ways – One using two inner product units where the coordinates are obtained in bit serial fashion and the other using only one inner product unit where the coordinates are obtained in bit parallel fashion after $m$ clock cycles. This leads to the following algorithm that can be implemented using the FOU.

*Algorithm 2: (GF($2^m$) multiplication)*
*Input: $a$ in $\Lambda$ and $b$ in $\Omega$*
*Output: $u$ in $\Lambda$*

*Step 2.1.* $s_j = \tilde{a}_j \qquad 0 \leqslant j \leqslant m-1$
$\tilde{u}_j = 0 \qquad 0 \leqslant j \leqslant m-1$

*Step 2.2. For* $i = 0$ *to* $m-1$ {
*If* $b_i \neq 0$ {
$\tilde{u}_j = \tilde{u}_j + s_{i+j} \qquad 0 \leqslant j \leqslant m-1$
}
$s_{i+m} = \sum_{j=0}^{m-1} s_{i+j}\ f_j$
} □

B.2 Operation

The $\Lambda$ coordinates $\tilde{a}_j$, $0 \leqslant j \leqslant m-1$, are loaded into register T of the FOU, which are the first $m$ elements of the $\{s_i\}$ sequence. The remaining elements are generated in T in the LFSR mode. The coordinates of $b$ are assumed to be in A$i$, $i = 1, 2$, in reverse order. In each clock cycle of the multiplication operation, these coordinates are left-shifted one position and if the leftmost bit of A$i$ is 1, the contents of T are accumulated in B2 to yield the $\Lambda$ coordinates of $u$ in B2 after $m$ clock cycles.

B.3 Remarks

• If the coordinates of $b$ follow those of $a$ in entering the FOU, then the iterations of step 2.2 of algorithm 2 can proceed with the arrival of each $b$ coordinate and the product $u$ is fully available in B2 right after the

last coordinate of $b$ enters the FOU. This implies that after the first coordinate of $a$ has entered the FOU, algorithm 2 is able to generate the product $u$ in $2m$ clock cycles

• As stated before, both $a$ and $u$ in algorithm 2 are given w.r.t. $\Lambda$. In applications where $a$ is available w.r.t. $\Omega$ and/or $u$ is needed w.r.t. $\Omega$, the basis transformations can be performed on-the-fly as the coordinates of $a$ and $u$ enter and leave, respectively, the FOU. This is discussed below.

### *C. Basis Transformation*

In this subsection, the transformations of the coordinates of an element $a \in \mathrm{GF}(2^m)$ from $\Lambda$ to $\Omega$ and vice versa are discussed. Toward this effect, if equation (2) is used for the forward transformation (i.e., $\Lambda$ to $\Omega$) in the FOU, then the $\Lambda$ coordinates are to be shifted, in reverse order, in the T register operating in the FFSR mode. The desired $\Omega$ coordinates would appear at the output of IP2 in bit serial fashion. If these $\Omega$ coordinates are to be directly taken to a register (either A1 or A2) for the purpose of storage, then we would have these coordinates stored in reverse order. More importantly, a separate path (perhaps an additional 1-bit bus) would be needed from the IP2 output to the various possible destination registers.

As an alternative approach, after a few steps of algebraic manipulation, equation (2) can be written as follows

$$a_i = \sum_{j=0}^{m-1-i} \tilde{a}_j \; f_{j+1} \qquad 0 \leqslant i \leqslant m-1 \tag{14}$$

In order to apply equation (14) to implement the forward transformation in the FOU, the $\Lambda$ coordinates are first loaded into T in correct order in bit parallel fashion. Then, T is left-shifted in the FFSR mode and the $\Omega$ coordinates appear on the 1-bit bus in bit serial fashion. If these coordinates are to be stored in A1 or A2, they can be directly taken from the bus in correct order.

For backward (i.e., $\Omega$ to $\Lambda$) transformation in the FOU, equation (3) can be applied. In this case, the $\Omega$ coordinates are simply shifted into the T register operating in the LFSR mode and after $m$ clock cycles, the $\Lambda$ coordinates are formed in the T register.

### *D. Comments*

• From the algorithms and operations presented above, one can see that the field inversion operation essentially dominates the FOU organization. The important blocks of the FOU, viz., T, B1, B2, IP1 and IP2, are employed during the inversion operation. Subsets of these blocks satisfy the needs of field multiplication and basis transformations. The two registers A1 and A2 are more like general purpose registers and are used to reduce the number of input/output operations by storing intermediate results. Although only two such registers are shown to be in the FOU, this number can be increased for improved performance at the application level where this type of processor is used. This is mainly because additional registers can be used to hold intermediate results and hence reduce I/O operations which may constitute a significant portion of the total computation time at the application level.

• When an elliptic curve cryptosystem is to be implemented on the processor, one might attempt to represent the elliptic curve points with respect to various projective coordinate systems. Such representation systems reduce the number of inversions at the expense of an increased number of multiplications and are considered to be advantageous if the inverse is at least three times slower than the multiplication and enough registers are available to hold intermediate results. This is however not the case for the processor being presented here. Additionally, the main cost of having an inverter in the FOU, given that the latter already has a multiplier, is only an inner product unit (IP1) and a register (B1).

## IV. FOU Optimization and the I/O Unit

Based on the algorithms and operations discussed in the previous section, we will give a number of approaches to reduce space and time complexities of the FOU. Then, we will present the IOU which interfaces the FOU to external devices.

*A. FOU Optimization*

A.1 Bus Reduction

The $M$-bit bus provides fast data transfers among various FOU blocks. For large values of $M$, this bus may constitute a real design challenge at the physical layout level and if not designed properly, it can account for a significant increase in silicon area. A trade-off can be made by reducing the width of the bus to $t$ bits ($t < M$). This reduction would increase the number of bus transfers to $\lceil \frac{m}{t} \rceil$ for a single $m$-bit operand. The effect of this on the field inversion is as follows. Using algorithm 1, this $t$-bit bus would be used to update $P(x)$ as follows

$$P(x) := P(x) + d\ x\ Q(x) \tag{15}$$

Assuming that the probability of $d$ being non-zero is 0.5, the total number of bus transfers for step 1.2 is $m\lceil \frac{m}{t} \rceil$, on average. Another $\lceil \frac{m}{t} \rceil$ bus transfers are needed for step 1.3. Similarly, for field multiplication operation, algorithm 2 would require $\frac{m}{2}\lceil \frac{m}{t} \rceil$ bus transfers on average. The other multiplication algorithm is however not affected by the bus size reduction.

A.2 Inner Product Unit Optimization

The two inner product units of the FOU provide the longest critical path. While IP2 is used by inversion, multiplication and basis transformations, IP1 is used only by inversion to compute $d$ in step 1.2. Variations of the Berlekamp-Massey algorithms, which do not require the inner product $d$, employ other circuits such as registers and XOR gates [16]. For large $M$, when these circuits can not be added for space limitations, one can proceed as follows to shorten the critical path of IP1.

Assuming that the two-input XOR gates are in binary tree form, 1-bit buffers are placed at a level of $\frac{1}{2}\lceil \log_2 M \rceil$. The total number of such buffers is only $2^{\frac{1}{2}\lceil \log_2 M \rceil} \approx \sqrt{M}$. (In most elliptic curve cryptosystems, $M < 256$, this implies that only sixteen 1-bit buffers would be needed). Although these buffers can potentially reduce the critical path by half (and hence double the clock frequency), the value of $d$ is available one clock cycle later. In order to proceed with the iterations of step 1.2, the value of $d$ is predicted to be '0' and the triplet $P(x)$, $Q(x)$ and $L$ are updated accordingly. In the next clock cycle, if the prediction turns out to be correct, the next iteration starts. For a wrong prediction, $P(x)$, $Q(x)$ and $L$ are corrected as follows.

Assume that at iteration $i$, the correct $P(x)$, $Q(x)$ and $L$ values in step 1.2 are $P^{(i)}(x)$, $Q^{(i)}(x)$ and $L^{(i)}$ respectively. In the next clock cycle, with the prediction of $d = 0$, the triplet is updated as follows

$$P'^{(i+1)}(x) = P^{(i)}(x) \tag{16}$$

$$Q'^{(i+1)}(x) = x\ Q^{(i)}(x) \tag{17}$$

$$L'^{(i+1)} = L^{(i)} \tag{18}$$

where the primes in the superscripts indicate predicted values of the triplet. The correct values of the triplet, however, at iteration $i + 1$ are

$$\begin{aligned} P^{(i+1)}(x) &= P^{(i)}(x) + x\ Q^{(i)}(x) \\ &= P'^{(i+1)}(x) + Q'^{(i+1)}(x) \end{aligned} \tag{19}$$

$$Q^{(i+1)}(x) = \begin{cases} P^{(i)}(x) = P'^{(i+1)}(x) & \text{if } e = 1 \\ x\ Q^{(i)}(x) = Q'^{(i+1)}(x) & \text{if } e = 0 \end{cases} \tag{20}$$

$$L^{(i+1)} = \begin{cases} i - L^{(i)} = i - L'^{(i+1)} & \text{if } e = 1 \\ L^{(i)} = L'^{(i+1)} & \text{if } e = 0. \end{cases} \tag{21}$$

Thus, the correct values of the triplets can be completely recovered from the predicted values.

The realization of the correction process in the FOU, however, requires a minor change in the input logic of register B1, which holds $Q(x)$, since

$$Q^{(i+1)}(x) = Q'^{(i+1)}(x) \quad \text{if } e = 0. \tag{22}$$

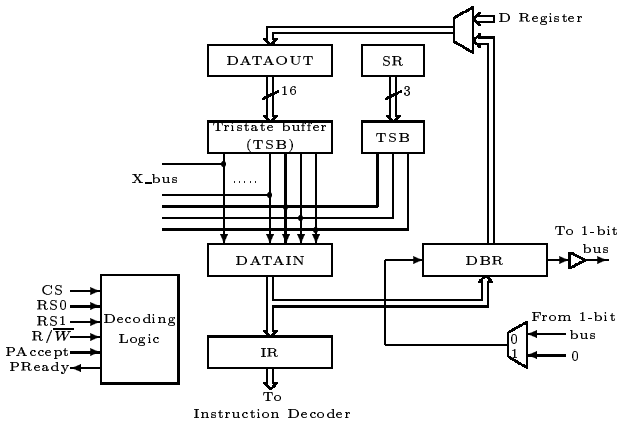



Fig. 5. Block diagram of the input/output unit (IOU).

The correction process also requires one extra clock cycle as a penalty for the wrong prediction. Assuming that 50% predictions are wrong and that the reduction of the critical path doubles the clock frequency, the average speed-up of the inversion operation is approximately $2 \times \frac{2m}{(2+1)m} - 1 = 33\%$. Since the other operations like multiplication and basis transformations do not use IP1, their speed-ups are expected to be close to the speed-up of the clock.

IP2, the other inner product unit of the FOU, has $F(x)$, the field defining irreducible polynomial stored in register C, as one of its inputs. Assuming that the second leading term of $F(x)$ is $x^k$, i.e., $F(x) = x^m + x^k + \sum_{i=1}^{k-1} f_i x^i + 1$, the C register in the FOU has zero in its leftmost $m - 1 - k$ positions. If the FOU is designed to be used for a set of predetermined irreducible polynomials (e.g., those specified by certain standardization committees), one can find $\min\{m - 1 - k\}$ for these polynomials and hence can reduce the number of AND and XOR gates by $\min\{m - 1 - k\}$ each.

Additionally, for efficient hardware realization, $F(x)$ is usually chosen to be an irreducible trinomial, if exists. Where there is no such trinomials, irreducible pentanomials are known to exist for all values of practical interest of $m$ [17]. Thus, if the FOU is designed to support a set of $n$ such low weight irreducible polynomials, then the maximum number of AND and XOR gates in IP2 would be $4n$ and $4n - 1$ respectively. The value of $n$ is expected to be reasonably small for most applications. For example, the value of $n$ can be as small as 4, yet satisfying the GF($2^m$) computational requirements for elliptic curve cryptosystems with $m \in \{113, 163, 193, 239\}$ which are of practical interest [18].

### *B. Input/Output Unit*

Compared to general purpose processors, a finite field processor for cryptographic applications needs to handle operands of much larger size. For example, using elliptic curve cryptography, which appears to use smaller operand (i.e., key) sizes, a GF($2^m$) processor needs to deal with 113-bits long operands for the minimum level of security being recommended by various standard committees. The operands can be of 1000 bits or more if the cryptosystem is based on the Diffie-Hellman discrete logarithm problem.

True bit-parallel input/output operations for such large operands are difficult even with today's advanced VLSI technologies. On the other hand, bit-serial input/output appears to be conservative and may cause an unacceptable amount of time delays for loading and unloading the operands to and from the GF processor. A more practical approach to these input/output operations is to split the operand into several blocks. The block size can be 8, 16, 32 or 64 bits to make the processor chip compatible with other devices.

To this effort, figure 5 shows the structure of the external input/output interface for the GF processor. This interface is based on asynchronous handshaking between the GF processor and other devices. The external bus (X_bus) moves the data and instructions to and from the processor in 16-bit blocks to speed up the data transfer. Three asynchronous registers (namely, DATAIN, DATAOUT and SR) connect to the external bus while two synchronous ones (viz., IR and DBR) are used for decoding and buffering.

The DATAIN register buffers data and instructions from the external bus. Instructions are copied later

to the instruction register (IR) for further decoding and execution while data are copied to the data buffer register (DBR) for further transfer to the FOU via the 1-bit bus. The DATAOUT register latches data from the D register or from the DBR and sends it out on the external bus via tristate buffers (TSB).

## V. Instruction Set Architecture and VLSI Implementation

In this section, we first present a set of instructions to be executed on the datapath developed in the previous sections. Then, we present a prototype VLSI implementation of the entire processor.

### *A. Instruction Set*

The instruction set consists of four logical groups. The first group contains those instructions used to clear, load and unload the registers. They also support transforming the data representation to and from the triangular basis during loading and unloading respectively. The second group contains shift left and right instructions. Arithmetic operations are carried out by the third group instructions. These include accumulation, multiplication and inversion. Moving the data between different registers in various fashions, even with basis transformation, is the responsibility of the last group. Table I lists the instructions supported along with their valid operands. To support these instructions, a control unit has been designed in [19]. Using the instruction set, a sample program for computing an inverse over GF($2^4$) is given below.

```
Ld        D,$0004
Ld        C,$000C
Ld        A1,$0005
Inv       A1
MvPr      A1,B2
MvAr      A2,A1
UnLd      A2

; Result  =  $000B
```

Assuming that it is a 'cold' start, the program sets up the D and C registers with the field dimension ($m$) and the configuration polynomial ($F(x)$) respectively. The contents of D and C correspond to the field given in appendix I. The element whose inverse is to be computed is loaded in A1. The 'Inv A1' instruction computes the inverse and stores the result in register B2 which is then loaded into A1 in bit parallel fashion. The 'MvAr A2,A1' instruction reverses the bit ordering and 'UnLd A2' unloads the result.

### *B. VLSI Implementation*

The GF processor architecture was modeled in VHDL and simulated to verify its functionality. After complete verification of the design functionality, it was then synthesized using appropriate time and area constraints. Time constraints in this case were alleviated by the pads, package, bonding and current limitations of the available prototyping technology, because these limitations are already forcing us to operate the chip at lower frequencies [20]. Area constraints were of higher priority since we target an area efficient GF processor. Both simulation and synthesis steps were carried out using Synopsys® tools [21] and a CMOS 0.5$\mu$m technology optimized for a 3.3V supply voltage. The prototype can support operations over finite fields up to GF($2^{64}$). The silicon area used was approximately 3.445mm×3.827mm for the whole chip packaged in a 68 pin PGA package.

The prototype chip was successfully tested at a clock frequency of 50MHz. This frequency limitation is due to the packaging technology used, however, the implemented chip core can run at a frequency of more than 80MHz while an implementation with $M = 256$ is estimated to run at a frequency of more than 75MHz. Approximate times required for some of the important finite field operations as well as those for Elliptic Curve point addition and point doubling operations at such frequency are compared to a recent fast software implementation [22] in table II. The speed-up ratios are also shown and are at least 8 which verifies the importance of the hardware implementation. The Elliptic Curve operations, however, require as much as 6 general registers for the operations to be implemented without much loading and unloading of intermediate

TABLE I

THE GF PROCESSOR INSTRUCTION SET.

| Mnemonic | Operation performed |
|---|---|
| Clr Des<br>Des ∈ {T, A1, A2, B1, B2, C} | Des ← 0 |
| Ld Des<br>Des ∈ {D, T, A1, A2, C} | Des ← $DBR_{in}$ |
| LdTb Des<br>Des ∈ {T, A1, A2} | Des ← TrBk($DBR_{in}$) |
| UnLd Src<br>Src ∈ {D, A1, A2} | $DBR_{out}$ ← Src |
| UnLdTb Src<br>Src ∈ {T, A1, A2} | $DBR_{out}$ ← TrFd(Src) |
| ShLf Src<br>Src ∈ {A1, A2} | Src ← ShiftLeft(Src) |
| ShRt Src<br>Src ∈ {A1, A2} | Src ← ShiftRight(Src) |
| AccB2 Src<br>Src ∈ {T, A1, A2, B1, B2} | B2 ← B2 + Src |
| MulT Src<br>Src ∈ {A1, A2} | B2 ← Src × T<br>(Src & T contents destroyed; result in Λ) |
| Inv Src<br>Src ∈ {A1, A2} | B2 ← $(\text{Src})^{-1}$<br>(Src & T contents destroyed; result in Ω) |
| NOp | no operation |
| MvAr Des, Src<br>Des ∈ {T, A1, A2, B1, B2},<br>Src ∈ {A1, A2}, Des ≠ Src | Des ← Src<br>(Src & T contents destroyed) |
| MvTb Des, Src<br>Des ∈ {T, A1, A2, B1, B2},<br>Src ∈ {A1, A2} | Des ← Src<br>(Src & T contents destroyed) |
| MvCb Des, Src<br>Des ∈ {A1, A2},<br>Src ∈ {T, A1, A2, B1, B2}, Des ≠ Src | Des ← Src<br>(Src & T contents destroyed) |
| MvPr Des, Src<br>Des, Src ∈ {T, A1, A2, B1, B2} | Des ← Src |

TABLE II

TIMING COMPARISON OF A RECENT SOFTWARE IMPLEMENTATION AND THE GF PROCESSOR.

| Operation over $GF(2^{191})$ | Time in $\mu$sec | | Speed-up |
|---|---|---|---|
| | Software | GF Coprocessor | |
| Addition | 0.6 | 0.03 | 20.00 |
| Multiplication | 39.0 | 2.41 | 16.18 |
| Inversion | 126.0 | 4.81 | 26.20 |
| EC Addition | 215.0 | 24.61 | 8.74 |
| EC Doubling | 220.0 | 27.05 | 8.13 |

results. This number can be reduced to 2 general registers to maintain a reasonable chip area by delegating the field addition operations needed to the main processor at almost no speed penalty (e.g., the times needed for EC addition and doubling become 24.27$\mu$sec and 26.79$\mu$sec respectively). Figure 6 shows the annotated layout along with the micro-photograph of the fabricated and tested chip.

Table III shows the area used by each of the building blocks along with the input/output pads ring which is about 0.5 mm wide from each of the four sides. It also shows an estimate for the area required for the FOU that can support finite fields up to a dimension of 256 which is a practical dimension for current cryptographic applications. The core area utilization factors represent the ratio of the core processor area to the area used for buses and interconnects in all of the blocks. These factors do not exceed 0.5 showing the significant area used by the wide parallel interconnects.

## VI. CONCLUSIONS

In this paper, the design and implementation of a Galois field processor, proposed as a flexible and configurable module for cryptographic applications, have been presented. The processor computation algorithms make use of both canonical and triangular basis computations. Algorithms for VLSI realization of basis transformations, multiplication and inversion have been presented and implemented. An extensible instruction set architecture is used to support those computations and some other manipulations. A variable dimension datapath allows the processor to operate over different finite fields in order to accommodate different applications

TABLE III

THE GF PROCESSOR AREA UTILIZATION IN PHYSICAL LAYOUT.

| Block name | # of std. cells | Dim. in $\mu$m $\times$ $\mu$m | Area in mm$^2$ | Core area utilization |
|---|---|---|---|---|
| Control unit | 1285 | 1032 $\times$ 860 | 0.89 | 0.47 |
| IOU | 303 | 630 $\times$ 520 | 0.33 | 0.5 |
| FOU ($M = 32$) | 1424 | 1092 $\times$ 932 | 1.02 | |
| Pads ring | | | 4.58 | |
| FOU ($M = 64$) | 4153 | 2090 $\times$ 1570 | 3.82 | 0.43 |
| Pads ring | | | 5.92 | |
| FOU ($M = 256$) | 22372 | 5800 $\times$ 4200 | 24.36 | |

and different requirements. A VHDL model was used to simulate the processor and verify its functionality. Furthermore, the model was synthesized and a prototype chip was fabricated and tested. The design and the implementation emphasize an area efficient processor to target embedded systems and smart card applications, however, more registers, complex controller circuitry, pipelining, wider internal buses could be used to build a high-performance variation at the cost of increased area. Also, for large composite fields, such as GF($2^{1024}$) used in the conventional discrete logarithm based cryptosystems, one can define the arithmetic operations over the subfield GF($2^{256}$) or GF($2^{128}$) and use the proposed processor for the subfield.

## APPENDICES

### I. GF($2^4$) REPRESENTATION

Let $F(x)$ be $x^4 + x + 1$ and $\omega \in \mathrm{GF}(2^4)$ satisfy $F(\omega) = 0$. Then, it can be shown that $\Omega = \{1, \omega, \omega^2, \omega^3\}$ and $\Lambda = \{\lambda_0, \lambda_1, \lambda_2, \lambda_3\} = \{\omega^{14}, \omega^2, \omega, 1\}$. The representation of all of the non-zero elements of GF($2^4$) w.r.t. $\Omega$ and $\Lambda$ are given below in table IV. For easy reference, the elements are also represented as powers of $\omega$ which happens to be a primitive element of GF($2^4$).

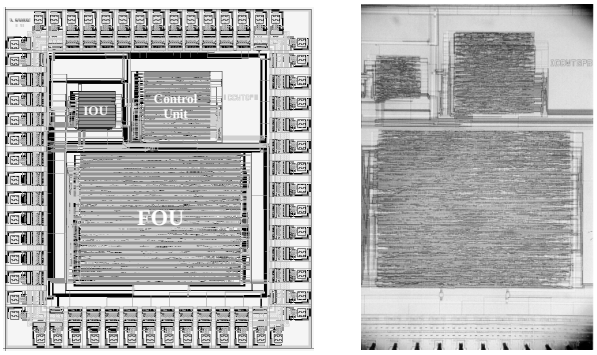
### II. EXAMPLE OF INVERSE OPERATION

Consider the field GF($2^4$) given in appendix I. Let $a = \omega^8 = (a_3, a_2, a_1, a_0) = (0, 1, 0, 1)$. To find the inverse of $a$, below we apply algorithm 1.

Step 1.1.

$$P(x) = 1, \quad Q(x) = 1, \quad L = 0, \quad s_0 = 0$$

Step 1.2. The iterations are shown in table V.




(a) Annotated layout (b) Micro-photograph

Fig. 6. The fabricated GF processor

Step 1.3.

$$g = (1,0,0,0)$$
$$b = (1,0,0,0)+(0,0,1,1) = (1,0,1,1) = \omega^7$$
$$\text{(from appendix I)}$$

which is the inverse of $a = \omega^8$.

## References


[1] M.A. Sirbu, "Credits and debits on the internet," *IEEE Spectrum*, vol. 34, no. 2, pp. 23–29, Feb. 1997.

[2] C.H. Fancher, "In your pocket: Smartcards," *IEEE Spectrum*, vol. 34, no. 2, pp. 47–53, Feb. 1997.

[3] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge, CUP, 1986.

[4] T. Hasegawa, J. Nakajima, and M. Matsui, "A practical implementation of elliptic curve cryptosystem over GF($p$) on a 16-bit microcomputer," *Proceedings of the first international workshop on Practice and Theory in Public Key Cryptography, PKC'98*, Feb. 1998.

[5] D. Naccache and D. M'Raïhi, "Cryptographic smart cards," *IEEE MICRO*, vol. 16, no. 3, pp. 14–24, June 1996.

[6] J.-F. Dhem, D. Veithen, and J.-J. Quisquater, "SCALPS: Smart card for limited payment systems," *IEEE MICRO*, vol. 16, no. 3, pp. 42–51, June 1996.

[7] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An implementation for elliptic curve cryptosystems over $F_{2^{255}}$," *IEEE Journal on Selected Areas in Communications*, vol. 11, pp. 804–813, June 1993.

[8] M.A. Hasan and M. Ebtedaei, "Efiicient architectures for computations over variable dimensional Galois fields," *IEEE Transactions on Circuits and Systems – I. Fundamental Theory and Applications*, vol. 45, no. 11, pp. 1205–1211, Nov. 1998.

[9] B. Green and G. Drolet, "A universal Reed-Solomon decoder chip," in *Proceedings of the 16$^{th}$ Biennial Symposium in Communications, Kingston, ON*, May 1992, pp. 327–330.

[10] Y.R. Shayan, *Versatile Reed-Solomon Decoder*, Ph.D. thesis, Concordia University, Montreal, Canada, 1990.

[11] M.A. Hasan and V.K. Bhargava, "Architecture for a low complexity rate-adaptive Reed-Solomon encoder," *IEEE Transactions on Computers*, vol. 44, no. 7, pp. 938–942, July 1995.

[12] M. A. Hasan, "Shift-register synthesis for multiplicative inversion over GF($2^m$)," *Proceedings of the ISIT, Whistler, B.C.*, p. 49, 1995.

[13] R.E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, MA, 1984.

[14] R. Schroeppel, S. O'Malley, H. Orman, and O. Spatscheck, "A fast software implementation for arithmetic operations in GF($2^n$)," in *Advances in Cryptology–CRYPTO '95, Lecture Notes in Computer Science*. 1995, pp. 43–56, Springer.

[15] C.J. Zarowski, "Parallel implementation of the Schur Berlekamp-Massey algorithm on a linearly connected processor array," *IEEE Transactions on Computers*, vol. 44, no. 7, pp. 930–933, 1995.

[16] E.D. Mastrovito, *VLSI Architectures for Computations in Galois Fields*, Ph.D. thesis, Department of Electrical Engineering, Linköping University, S-581 83 Linköping, Sweden, 1991.


TABLE IV

Field elements of GF($2^4$).

| Power representation | $\Omega$ basis representation $(a_3, a_2, a_1, a_0)$ | $\Lambda$ basis representation $(\tilde{a}_3, \tilde{a}_2, \tilde{a}_1, \tilde{a}_0)$ |
|---|---|---|
| $\omega^0$ | (0, 0, 0, 1) | (1, 0, 0, 0) |
| $\omega^1$ | (0, 0, 1, 0) | (0, 1, 0, 0) |
| $\omega^2$ | (0, 1, 0, 0) | (0, 0, 1, 0) |
| $\omega^3$ | (1, 0, 0, 0) | (1, 0, 0, 1) |
| $\omega^4$ | (0, 0, 1, 1) | (1, 1, 0, 0) |
| $\omega^5$ | (0, 1, 1, 0) | (0, 1, 1, 0) |
| $\omega^6$ | (1, 1, 0, 0) | (1, 0, 1, 1) |
| $\omega^7$ | (1, 0, 1, 1) | (0, 1, 0, 1) |
| $\omega^8$ | (0, 1, 0, 1) | (1, 0, 1, 0) |
| $\omega^9$ | (1, 0, 1, 0) | (1, 1, 0, 1) |
| $\omega^{10}$ | (0, 1, 1, 1) | (1, 1, 1, 0) |
| $\omega^{11}$ | (1, 1, 1, 0) | (1, 1, 1, 1) |
| $\omega^{12}$ | (1, 1, 1, 1) | (0, 1, 1, 1) |
| $\omega^{13}$ | (1, 1, 0, 1) | (0, 0, 1, 1) |
| $\omega^{14}$ | (1, 0, 0, 1) | (0, 0, 0, 1) |

TABLE V

THE ITERATIONS OF STEP 1.2 IN THE INVERSE OPERATION EXAMPLE.

| $i$ | $d$ | $e$ | $P(x),\ \ Q(x)$ | $L$ | $s_i$ |
|---|---|---|---|---|---|
| 1 | 0 | 0 | $P(x) = 1$<br>$Q(x) = x$ | 0 | 1 |
| 2 | 1 | 1 | $P(x) = 1 + x^2$<br>$Q(x) = 1$ | 2 | 0 |
| 3 | 0 | 0 | $P(x) = 1 + x^2$<br>$Q(x) = x$ | 2 | 1 |
| 4 | 0 | 0 | $P(x) = 1 + x^2$<br>$Q(x) = x^2$ | 2 | 1 |
| 5 | 1 | 1 | $P(x) = 1 + x^2 + x^3$<br>$Q(x) = 1 + x^2$ | 3 | 1 |
| 6 | 0 | 0 | $P(x) = 1 + x^2 + x^3$<br>$Q(x) = x + x^3$ | 3 | 1 |
| 7 | 1 | 1 | $P(x) = 1 + x^3 + x^4$<br>$Q(x) = 1 + x^2 + x^3$ | 4 | 1 |
| 8 | 1 | 0 | $P(x) = 1 + x$ | - | - |


[17] G. Seroussi, "Table of low-weight binary irreducible polynomials," Hewlett-Packard Laboratories Technical Reports, HPL-98-135, Palo Alto, CA, August 1998. http://www.hpl.hp.com/techreports/98/HPL-98-135.html.

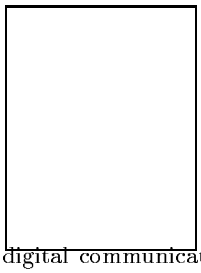
[18] S. Blake-Wilson, "SEC1: Elliptic curve cryptography," Certicom Research, Working Draft Ver.0.4, Toronto, ON, Canada, August 1999. http://www.secg.org/drafts.htm.

[19] M.A. Hasan and A.G. Wassal, "A variable dimension Galois field coprocessor with a double-bases approach," Tech. Rep., University of Waterloo-E&CE#2000-03, 2000.

[20] N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A systems Perspective*, Addison Wesley, 1994.

[21] Synopsys, Inc., Mountain View, CA, *Synopsy Online Documentations v1998.08-1*, 1998.

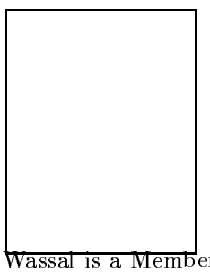
[22] E. De Win and B. Parneel, "Elliptic curve public-key cryptosystems – An introduction," in *State of the Art in Applied Cryptography– Course on Computer Security and Industrial Cryptography, Lecture Notes in Computer Science*. 1998, pp. 131–141, Springer.

**M.A. Hasan** received the B.Sc. degree in electrical and electronic engineering, the M.Sc. degree in computer engineering, both from the Bangladesh University of Engineering and Technology, in 1986 and 1988, respectively, and the Ph.D. degree in electrical engineering from the University of Victoria in 1992.

Since 1993 he has been with the Department of Electrical and Computer Engineering, University of Waterloo, where he is now an Associate Professor. At the university of Waterloo, he is also a member of the Center for Applied Cryptographic Research and the Center for Wireless Communications. His current research interests include algorithms and architectures for computations in Galois fields, data security and reliability, and digital communication networks. From January to December of 1999, he was with the Motorola Labs., Schaumburg, IL, USA on a sabbatical leave from the University of Waterloo.

Dr. Hasan is a recipient of the Raihan Memorial Gold Medal. At the University of Victoria, he was awarded the President's Research Scholarship four times. He served on the program and executive committees of several conferences, and currently, he is an associate editor of the IEEE Transactions of Computers. He is a Senior Member of the IEEE and a licensed professional engineer of Ontario.



**Amr G. Wassal** (S'92-M'00) received the B.Sc. (Hons.) and M.Sc. degrees in electronics and communications engineering, both from Cairo University, Egypt, in 1993 and 1996, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo in 2000. From 1993 to 1995, he was a research engineer with the Technology Group, IBM, Egypt. He is currently with the research and development division of PMC-Sierra Inc.

His current research interests include modeling, analysis and design of VLSI architectures for digital communications particularly in the areas of traffic management and data security for broadband networks. Dr. Wassal is a Member of the IEEE computer and communication societies.