A Microcoded Elliptic Curve Processor Using FPGA Technology

Philip H. W. Leong, *Senior Member, IEEE*, and Ivan K. H. Leung

***Abstract*—The implementation of a microcoded elliptic curve processor using field-programmable gate array technology is described. This processor implements optimal normal basis field operations in F_{2^n}. The design is synthesized by a parameterized module generator, which can accommodate arbitrary n and also produce field multipliers with different speed/area tradeoffs. The control part of the processor is microcoded, enabling curve operations to be incorporated into the processor and hence reducing the chip's I/O requirements. The microcoded approach also facilitates rapid development and algorithmic optimization: for example, projective and affine coordinates were supported using different microcode. The design was successfully tested on a Xilinx Virtex XCV1000-6 device and could perform an elliptic curve multiplication over the field F_{2^n} using affine and projective coordinates for $n = 113, 155$, and 173.**

***Index Terms*—Arithmetic, cryptography, Galois fields, microprogramming, public key cryptography, reconfigurable architectures.**

I. Introduction

ELLIPTIC curve cryptography (ECC) was proposed by Koblitz [1] and Miller [2] in 1985. Compared with other commonly used public key cryptosystems such as RSA and discrete logarithm, ECC has the following benefits that make it particularly suitable for embedded applications.

1) ECC offers the highest security per bit of any known public key cryptosystem so a smaller memory can be used.
2) ECC hardware implementations use fewer transistors. As an example, a VLSI implementation of a 155-bit ECC processor has been reported that uses only 11 000 transistors [3], compared with an equivalent strength 512-bit RSA processor that used 50 000 transistors [4].
3) ECC is probably more secure than RSA: the largest RSA and ECC challenges solved are 512-bit and 108-bit, respectively. The solution to the 108-bit ECC challenge is believed to be the largest effort ever expended in a public key cryptography challenge. It took four months and involved approximately 9500 machines. The amount of work required to solve the problem was about 50 times that of the 512-bit RSA.

Previous implementations of ECC processors have been based on VLSI chips that implement a coprocessor for performing the underlying field operations. An optimal normal basis multiplier over $F_{2^{593}}$ was reported in 1988. It was implemented in 2-μm CMOS technology, used 90 000 transistors, and occupied 0.3-in on a side. This chip, together with a Motorola DSP 56 000, was used to implement a complete ECC system that could calculate five points a second on a supersingular curve [3]. In 1993, the same team developed a processor for operations in the Galois field $F_{2^{155}}$ [3], which used 11 000 transistors and could operate at 40 MHz. This implementation was intended to be compact yet secure.

A field-programmable gate array (FPGA)-based processor for elliptic curve cryptography in a composite Galois field $F_{(2^n)^m}$ was developed by Rosner [5]. A compact superserial multiplier for FPGAs that trades off performance for area was reported in 1999, and its performance for field (polynomial basis) and curve multiplications over $F_{2^{167}}$ has also been presented [6].

Previous implementations based on Galois field processors have the disadvantage that a high bandwidth interface is required to supply the coprocessor with its data. Another limitation of previous application-specific integrated circuit (ASIC) designs is that the field operations are restricted to certain groups (i.e., the subfield and extension fields of $F_{2^{155}}$) and these cannot be changed without fabricating a new chip.

The implementation described in this paper differs from previous implementations in the following ways.

1) The higher level curve operations as well as the field operations are implemented on the chip. This makes the I/O bandwidth requirements much lower than for chips that only implement the field operations.
2) The curve operations are implemented as sequences of field operations that are programmed in microcode. This allows algorithmic optimizations to the design to be made without changing the hardware.
3) The design uses projective coordinates, which offers a speed advantage over the affine coordinates since fewer field inversions are required.
4) The entire design is generated by a module generator that can generate ECC systems of arbitrary key size. Thus, ECC systems of arbitrary size over an optimal normal basis can be generated (provided they fit on the FPGA device).
5) The parallelism of the field multiplier can also be controlled by the module generator, greatly improving performance.
6) The initialization of the inputs of the curve multiplication to be computed is performed using bitstream reconfiguration, which results in a savings in hardware and could lead to an improvement in speed.

Manuscript received March 1, 2001; revised October 20, 2001.

The authors are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong, China (e-mail: phwl@cse.cuhk.edu.hk; khleung@cse.cuhk.edu.hk).

Digital Object Identifier 10.1109/TVLSI.2002.801608

An earlier paper presented a parameterized elliptic curve processor with some of these features [7]. This paper describes an improved implementation that uses projective coordinates, a parallel field multiplier, and bitstream reconfiguration to achieve significant performance improvements.

The rest of this paper is organized as follows. Section II is an introduction to some of the mathematical concepts needed to understand elliptic curve cryptography. The architecture and implementation details of the elliptic curve processor are presented in Section III. Results are presented in Section IV and conclusions drawn in Section V.

II. Background Mathematics

In this section, an informal introduction to the abstract algebra and elliptic curves used in this implementation is presented. More rigorous treatments can be found in [8]–[11].

A. Groups and Fields

A group $(G, +)$ consists of a set of numbers G together with an operation $+$ that satisfies the following properties.

1) Associativity: $(a+b)+c = a+(b+c)$ for all $a, b, c \in G$.
2) Identity: there is an element $0 \in G$ such that $a+0 = 0+a$ for all $a \in G$.
3) Inverse: For every $a \in G$, there exists an element $-a \in G$ such that $-a + a = a + -a = 0$.

The group G is said to be abelian (or commutative) if $a+b = b+a$ for all $a, b \in G$. We will use the notation $\#G$ to denote the number of elements in a group.

A field $(F, +, \times)$ is a set of numbers F together with two operations $+$ and $\times$ that satisfies the following properties.

1) $(F, +)$ is an abelian group with identity 0.
2) $\times$ is associative.
3) There exists an identity $1 \in F$ with $1 \neq 0$ such that $1 \times a = a \times 1 = a$ for all $a \in F$.
4) The operation $\times$ is distributive over $+$, i.e., $a \times (b+c) = (a \times b) + (a \times c)$ and $(b+c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in F$.
5) $a \times b = b \times a$ for all $a, b \in F$.
6) For every $a \neq 0, a \in F$ there exists an element $a^{-1} \in F$ such that $a^{-1} \times a = a \times a^{-1} = 1$.

If the field has a finite set of elements, it is called a finite (or Galois) field. Let F_p be the finite field with p elements. Numbers in the field F_2 can be represented by {0, 1}. If $p = 2^n$, numbers in F_{2^n} can be represented as n-bit binary numbers.

B. F_{2^n} Operations (Normal Basis)

The field F_{2^n} has particular importance in cryptography since it leads to particularly efficient hardware implementations. Elements of the field are represented in terms of a basis. Most implementations either use a polynomial basis or a normal basis. For the implementation described in this paper, a normal basis was chosen since it is believed that it leads to a more efficient hardware implementation.

In a normal basis, an element A can be uniquely represented in the form

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i}$$

where $a_i \in F_2$ and $\beta \in F_{2^n}$.

1) Addition and Squaring: The addition operation over F_{2^n} is simply a bit-wise exclusive OR (XOR) operation. Furthermore, in a normal basis, squaring is simply a rotate left operation.

2) Multiplication: Let

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i}$$
$$B = \sum_{i=0}^{n-1} b_i \beta^{2^i}$$

and

$$C = A \times B = \sum_{i=0}^{n-1} c_i \beta^{2^i}$$

then multiplication is defined in terms of a multiplication table $\lambda_{ij} \in \{0, 1\}$

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij} a_{i+k} b_{j+k}. \tag{1}$$

An optimal normal basis (ONB) [12] is one with the minimum number of terms in (1), or equivalently, the minimum possible number of nonzero λ_{ij}. This value is $2n-1$, and since it allows multiplication with minimum complexity, such a basis would normally lead to a more efficient hardware implementation.

Derivation of the λ_{ij} values in (1) is dependent on n. There exists an optimal normal basis in F_{2^n} if:

1) two is a primitive in F_{n+1};
2) two is a primitive in F_{2n+1};
3) n is odd and two generates the quadratic residues in Z_{2n+1} (where Z are the integers).

An ONB exists in F_{2^n} for 23% of all possible values of $n < 1000$ [12]. The design presented in this paper assumes an n that has an ONB.

The multiplication table is derived differently for the three different types of ONB described above. As an example, for the second type of ONB, $\lambda_{ij} = 1$ iff i and j satisfy one of the four congruences $2^i \pm 2^j \equiv \pm 1$ [12].

3) Inversion: The algorithm used for inversion is derived from Fermat's Little Theorem

$$a^{-1} = a^{2^n-2} = \left(a^{2^{n-1}-1}\right)^2$$

for all $a \neq 0$ in F_{2^n}. The method used was proposed by Itoh and Tsujii [13], based on the following decomposition, which minimizes the number of multiplications (squarings are much cheaper in a normal basis):

$$a^{2^{n-1}-1} = \begin{cases} \left(a^{2^{((n-1)/2)}-1}\right)^{2^{((n-1)/2)}} a^{2^{((n-1)/2)}-1}, & n \text{ odd} \\ a\left(a^{2^{((n-1)/2)}-1}\right)^{2}, & n \text{ even.} \end{cases}$$

The field inversion is computed using the following algorithm:

```
INPUT: $k \in F_{2^n}$
OUTPUT: $l = k^{-1}$
1. Set $s \leftarrow \log_2(n-1) - 1$
2. Set $p \leftarrow k$
3. For $i$ from $s$ downto 0
     Set $r \leftarrow$ shift $(n-1)$ to right by
     $s$ bit(s)
     Set $q \leftarrow p$
     Rotate $q$ to left by $r$ bit(s)
     Set $t \leftarrow$ multiply $p$ by $q$
     If last bit of $r$ is set then
     Rotate $t$ to left by 1-bit
     $p \leftarrow$ multiply $t$ by $k$
     else
     $p \leftarrow t$
     $s \leftarrow s - 1$
4. Rotate $p$ to left by 1-bit
5. Set $l \leftarrow p$
6. Return $l$
```

The total number of multiplies M required to perform an inversion in F_{2^n} using the above algorithm is

$$M(n) = \log_2(n-1) + \nu(n-1) - 1$$

where $\nu(x)$ is the number of nonzero bits in the binary representation of x.

C. Elliptic Curves Over F_{2^n}

Section II-B described the implementations of operations in the underlying field F_{2^n}. In this section, a group constructed from points on elliptic curves over F_{2^n} is defined and the efficient implementation of operations in this group described.

A nonsupersingular elliptic curve E over F_{2^n}, $E(F_{2^n})$ is the set of all solutions to the following equation with coordinates in the algebraic closure of E [8]

$$y^2 + xy = x^3 + a_2x^2 + a_6 \tag{2}$$

where $a_2, a_6 \in F_{2^n}$, and $a_6 \neq 0$. Such an elliptic curve is a finite abelian group [8]. The number of points in this group is denoted by $\#E(F_{2^n})$.

1) Curve Addition: If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are points on the elliptic curve [i.e., satisfy (2)] and $P \neq -Q$ then $(x_3, y_3) = R = P+Q$ can be defined geometrically. In the case that $P \neq Q$ (i.e., point addition), a line intersecting the curve at points P and Q must also intersect the curve at a third point $-R = (x_3, -y_3)$ (in this paper, this operation will be referred to as ESUM). If $P = Q$ (point doubling), the tangent line is used; this will be referred to as EDBL.

For E given in affine coordinates, if $P \neq Q$

$$\begin{aligned} \lambda &= \frac{y_1 + y_2}{x_1 + x_2} \\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a_2 \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1. \end{aligned}$$

If $P = Q$

$$\begin{aligned} \lambda &= \frac{y_1}{x_1} + x_1 \\ x_3 &= \lambda^2 + \lambda + a_2 \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1. \end{aligned}$$

Therefore, in affine coordinates, both point addition and point doubling require two multiplications and one field inversion. Note that field inversion is far more expensive than a field multiplication.

A nonsupersingular curve $E(F_{2^n})$ can be equivalently viewed as the set of all points in the projective plane $P^2(F_{2^n})$ that satisfy [18]

$$y^2z + xyz = x^3 + a_2x^2z^2 + a_6z^3. \tag{3}$$

Let $P = (x_1 : y_1 : z_1) \in E$, $Q = (x_2 : y_2 : 1) \in E$ and $P \neq -Q$. Since $P = (x_1/z_1 : y_1/z_1 : 1)$, we can use the addition formula for E in affine coordinates to find $P + Q = (x_3' : y_3' : 1)$. We have

$$\begin{aligned} x_3' &= \frac{B^2}{A^2} + \frac{B}{A} + \frac{A}{z_1} + a_2 \\ y_3' &= \frac{B}{A}\left(\frac{x_1}{z_1} + x_3'\right) + x_3' + \frac{y_1}{z_1} \end{aligned}$$

where $A = (x_2z_1 + x_1)$ and $B = (y_2z_1 + y_1)$. To eliminate the denominators of the expressions for x_3' and y_3', we set $z_3 = A^3z_1$, $x_3 = x_3'z_3$, and $y_3 = y_3'z_3$; therefore $P + Q = (x_3 : y_3 : z_3)$

$$\begin{aligned} x_3 &= AD \\ y_3 &= CD + A^2(Bx_1 + Ay_1) \\ z_3 &= A^3z_1 \end{aligned}$$

where $C = A + B$ and $D = A^2(A + a_2z_1) + z_1BC$.

Similarly, the formulas for $2P = (x_3 : y_3 : z_3)$ are

$$\begin{aligned} x_3 &= AB \\ y_3 &= x_1^4A + B\left(x_1^2 + y_1z_1 + A\right) \\ z_3 &= A^3 \end{aligned}$$

where $A = x_1z_1$ and $B = a_6z_1^4 + x_1^4$. The result can be converted back to affine coordinates by multiplying each coordinate by z_3^{-1}. Therefore, in projective coordinates, we can perform curve multiplication by using only one inversion after a series of additions and doublings. The number of field multiplications and field inversions for curve point addition and doubling are shown in Table I. As an example, for $n = 155$, a field inversion takes ten multiplications. Therefore, the total number of multiplications for an affine point addition or doubling is $2+10 = 12$

TABLE I
NUMBER OF FIELD MULTIPLICATIONS AND INVERSIONS FOR AFFINE AND PROJECTIVE POINT ADDITION AND DOUBLING

	Affine		Projective	
Operation	ESUM	EDBL	ESUM	EDBL
Field Multiplication	2	2	13	7
Field Inversion	1	1	0	0

multiplications, whereas a projective point addition and doubling takes thirteen and seven multiplications, respectively.

2) Curve Multiplication: Multiplication (referred to in this paper as EMUL) is defined by repeated addition, i.e.,

$$Q = cP \tag{4}$$

$$= \underbrace{P + P + \cdots + P}_{c \text{ times}}. \tag{5}$$

This can be computed using the following "double and add" algorithm.

```
For affine coordinates,
INPUT: $P \in E(F_{2^n})$ and $c \in F_{2^n}$
OUTPUT: $Q = cP$
1. $c = \Sigma_{i=0}^{m} b_i 2^i$, $b_i \in 0, 1, b_m = 1$
2. $Q = P$
3. For $i$ from $m-1$ downto 0
     $Q = Q + Q$ (Affine EDBL)
     If $b_i = 1$ then
     $Q = Q + P$ (Affine ESUM)
4. Return $Q$
```

This requires $n + \nu(c) - 2$ point additions, where $\nu(c)$ is the number of nonzero bits in the binary representation of c.

In the case of projective coordinates, the same algorithm can be used except that the input P is first converted to projective coordinates, the projective formulas for EDBL and ESUM are used instead of the affine formulas, and the result Q is converted back to an affine representation. Note that there is no inversion required to perform EDBL and ESUM in projective coordinates; the only inversion is used in the conversion from projective to affine coordinates.

D. Discrete Logarithm Problem

Elliptic curve cryptography is based on the discrete logarithm problem applied to elliptic curves over a finite field. In particular, for an elliptic curve E, it relies on the fact that it is easy to compute

$$Q = cP$$

for $c \in \{1, \ldots, \#G - 1\}$ and $P, Q \in E$. However, there is currently no known subexponential time algorithm to compute c given P and Q.

In fact, the discrete logarithm problem can be used to build cryptosystems with any finite abelian group. Indeed, multiplicative groups in a finite field were originally proposed. However, the difficulty of the problem depends on the group, and at present, the problem in elliptic curve groups is orders of magnitude harder than the same problem in a multiplicative group of a finite field. This feature is the main strength of elliptic curve cryptosystems.

E. Elliptic Curve Cryptography

The discrete logarithm problem can be used as the basis of various public key cryptographic protocols for key exchange, encryption, and digital signatures. It is beyond the scope of this paper to review all of the cryptographic protocols for public key cryptography, but an example of its use in the Diffie–Hellman key exchange is given in this section.

Suppose that Alice and Bob wish to agree on a common key to be used for encryption using a traditional secret key algorithm such as DES, but need to do so over an insecure channel such as the Internet. Then the following Diffie–Hellman procedure can be used with a public elliptic curve E and point $P \in E$.

1) Alice generates a secret random integer $c_A \in 1, \ldots \#G - 1$ and sends the point $c_A \times P$ to Bob.
2) Bob generates a secret random integer $c_B \in 1, \ldots \#G - 1$ and sends the point $c_B \times P$ to Alice.
3) Alice and Bob can both compute the key $c = k_A \times (c_B \times P) = c_B \times (c_A \times P)$.

An adversary, Carol, eavesdropping on the channel, can only gain the information E, P, $c_A \times P$, and $c_B \times P$. For Carol to be able to compute c, she must solve the discrete logarithm problem, and the best known algorithm takes fully exponential time. Alice and Bob, however, need only compute elliptic curve multiplications, which are comparatively easy.

III. AN ELLIPTIC CURVE PROCESSOR

A block diagram of the elliptic curve processor is shown in Fig. 1. The organization is similar to a traditional microcoded central processing unit (CPU) in that it consists of an arithmetic logic unit (ALU), register file, microcode sequencer, and microcode storage. Major differences between this architecture and a conventional CPU are that the datapath is n bits wide and the ALU performs operations based on F_{2^n} arithmetic instead of integer arithmetic.

A. Arithmetic Logic Unit

The ALU is simpler and faster than an integer ALU since no carry propagation is required. The addition operation is implemented simply as an XOR function, and the squaring function is implemented as a rotate left operation. The complexity of the ALU is determined by the F_{2^n} multiplier.

Fig. 2 shows the circuit used for calculating c_k of (1) [14]. In each cycle t, $(0 \le t < n)$, the c_kth cell computes

$$F_k(t) = b_{2k+t} \sum_{i=0}^{n-1} \lambda_{ik} a_{i+k+t}$$

(where all subscripts are reduced modulo n).

In each cycle, the A, B and C registers are rotated as shown in Fig. 3. The result is that after n cycles, the contents of register C are the desired product of the A and B inputs [14]. It should be noted that an optimal normal basis reduces

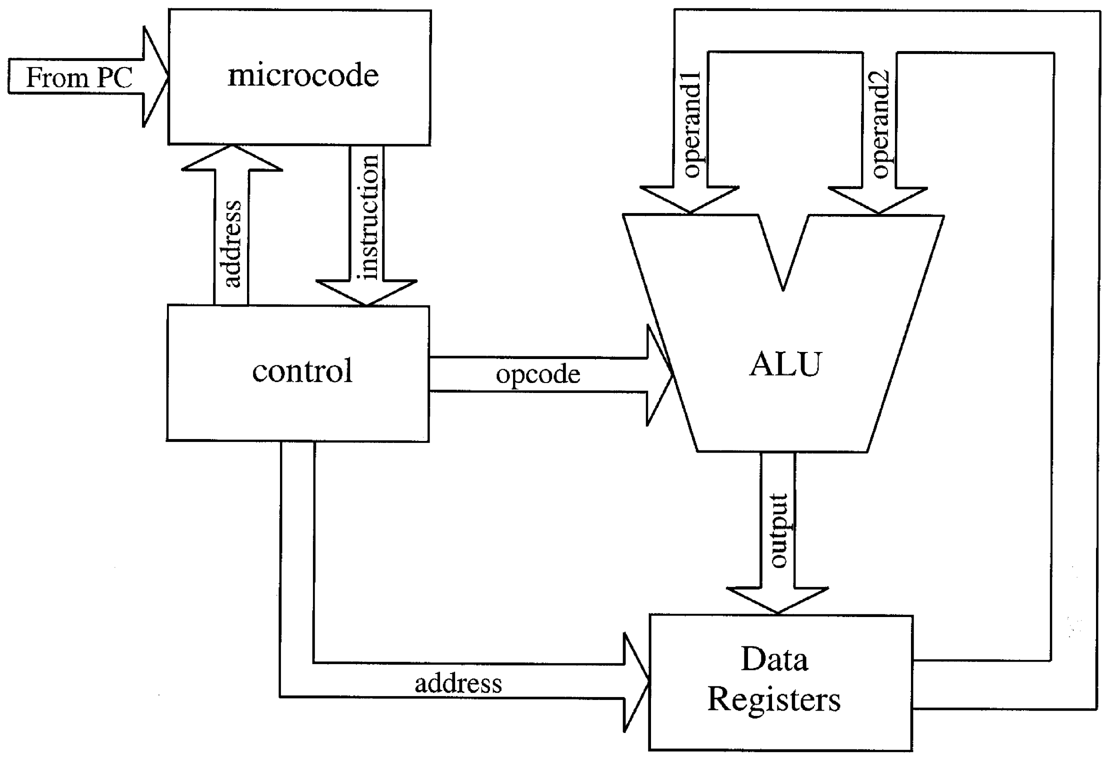

Fig. 1. EC processor architecture.

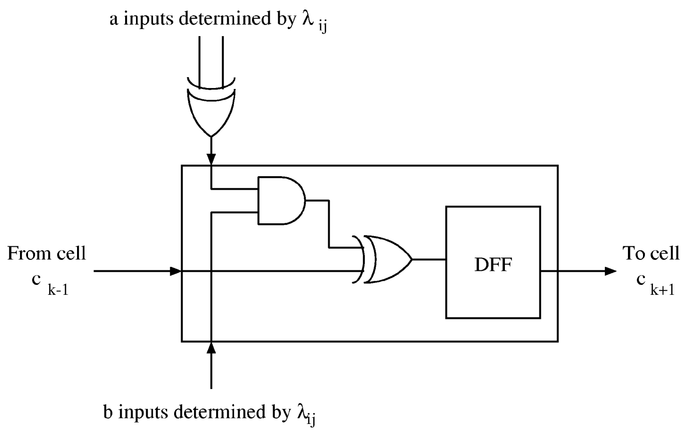

Fig. 2. F_{2^n} multiplier element of c_k.

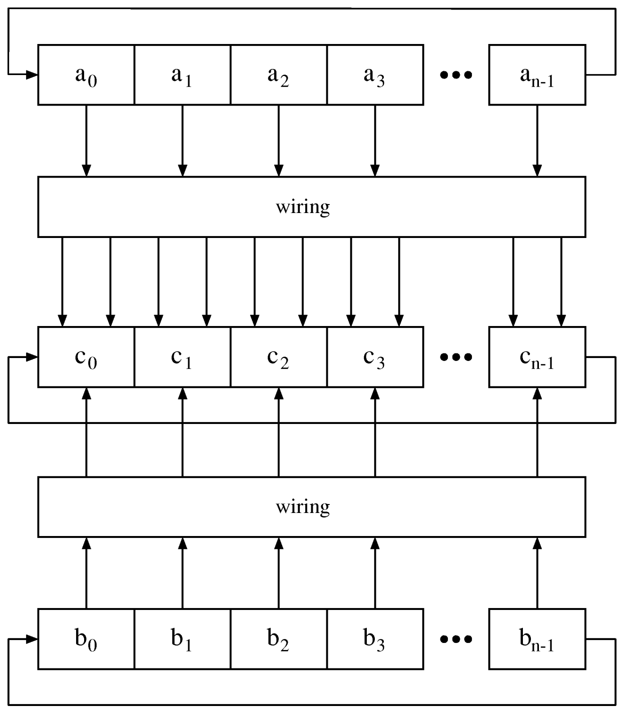

Fig. 3. F_{2^n} multiplier circuit.

the number of interconnections and fanout of signals in the multiplier to the minimum possible, resulting in reduced area and increased speed over a nonoptimal normal basis. In fact, the maximum fanout for a_i in Fig. 3 is four [14].

The field multiplier can be easily parallelized. To increase the parallelism by a factor k, the multipler logic can be duplicated k times and the number of cycles required for a multiplication is reduced to $\lfloor n/k \rfloor + 2$. As an example, a parallel version $(k = 2)$ of Fig. 2 is shown in Fig. 4.

B. Register File

A $16 \times n$-bit dual-port synchronous register file is constructed from the 16×1-bit distributed RAM feature of the Xilinx Virtex series logic cell (see Section III-F). This gives an eightfold reduction in resources over RAMs based on latches.

C. Microcode

The ALU plus register file form a F_{2^n} processor similar to previous designs [3]. However, for performing elliptic curve cryptography, higher level elliptic curve multiplications of Section II-C2 are required. This could be implemented as a finite-state machine (FSM) [5] or in microcode. The implementation

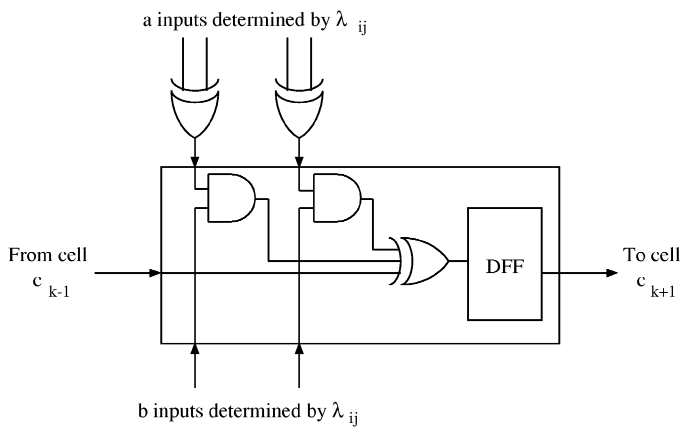

Fig. 4. Multiplier element of a parallel multiplier ($k = 2$).

TABLE II
CLOCK CYCLES REQUIRED FOR EACH INSTRUCTION USING A k-WAY PARALLEL ALU

Operation	Clock Cycles
NOP	1
XOR	1
Rotate left, ROTL	1
Shift right, SHFR	1
Field Multiplication, MUL	$\lfloor \frac{n}{k} \rfloor$+2
Transfer register value, TFR	1
Jump Instructions JKZ, JCZ, JMP	1

described in this paper opted for a microcoded approach, which has the following advantages in a FPGA implementation.

1) The microcode is stored in Xilinx Virtex BlockRAMs and does not use logic resources of the FPGA (as explained in Section III-F). The microcode sequencer in this design is very simple and has a small overhead.
2) The microcode can be changed without requiring recompilation of the elliptic curve processor.
3) Algorithmic optimizations to the processor can be performed entirely in microcode.
4) A microcoded description is a higher level abstraction than a finite-state machine and hence easier to develop and debug.

The instruction set of the processor is shown in Table II. Apart from instructions that directly control the ALU, there are three types of jump instructions: JMP—jump unconditionally, JKZ—jump if the least significant bit of K counter is zero, and JCZ—jump if the C register is zero.

Each instruction is 16 bits in width, and the format of instructions is shown in Fig. 5. Most instructions accept a source register and a destination register in `operand1` and `operand2`, respectively. The jump instructions have a 12-bit jump address.

A two-pass symbolic assembler was developed that takes symbolic input and produces the binary microcode, which can be downloaded to the processor's microcode store. A microcode simulator was also written to facilitate microcode development.

D. Parameterized Module Generator

An important feature of the elliptic curve processor (ECP) is that it is parameterized via a module generation program. In contrast to previous designs [3], this scheme advantageously uses the reconfigurable nature of the FPGA, so an elliptic curve processor for any n with an optimal normal basis can be produced.

The module generator is a program written in the Perl programming language [15], which takes n and k (described in Section III-A) as input parameters and produces the VHDL code of the ECP as output. Perl is a language that supports long integer arithmetic, which was helpful in performing the calculations required to generate the field multiplier. The module generator first computes the the field multiplication table [i.e., the λ matrix in (1)]. With the information from the λ matrix and k, the module generator produces a VHDL description of the circuit shown schematically in Figs. 2 and 3. In actuality, the only part of the processor that needs to be customized by the module generator is the field multiplier. Other parts of the ECP (including the rest of the ALU) are written in standard VHDL and do not require customization by the module generator for different n or k. The same microcode is also used for different n and k.

E. Bitstream Reconfiguration

To perform an elliptic curve multiplication, the values of c and P in (4) as well as the parameters of the curve must be downloaded to the processor. This was done using a bitstream modification technique to modify the contents of ROMs in which the parameters were stored. The circuit is designed in the normal fashion and the ROMs can be placed at arbitrary locations. After synthesis, technology mapping, place, and routing, a circuit description file (for the Xilinx tools this has an extension .ncd) is generated. Using tools provided by Xilinx, the contents of the circuit can be converted into a human readable format, and information regarding the physical location of the ROMs can be extracted. A software program was written that takes as input the bitstream, .ncd file, and initialization parameters, modifies the ROM values in the bitstream accordingly, and recomputes the CRC of the bitstream (it would also be possible to use the Xilinx JBits software developer's kit to manipulate the Xilinx Virtex bitstream). The resulting bitstream can be downloaded to a Virtex FPGA.

The advantage of bitstream reconfiguration is that circuitry to download the parameters is avoided, hence reducing the overall area and increasing the speed of the processor. The disadvantage is that in our current implementation, the entire bitstream must be downloaded in order to change a few parameters and is hence inefficient. It should be possible to improve efficiency by using the partial reconfiguration feature of the Virtex architecture [16] to modify only a small portion of the chip when parameters are changed.

Bitstream reconfiguration could be avoided by using a standard interface where input parameters are downloaded to the chip via registers. From a practical point of view, this would be

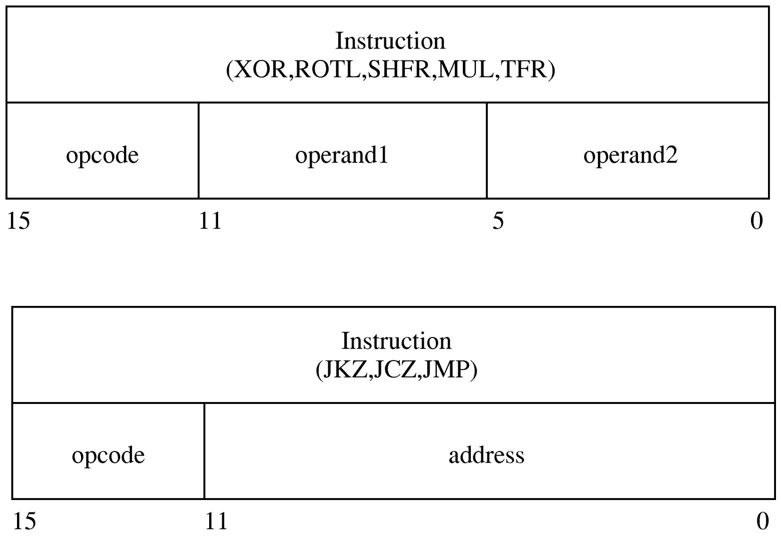

Fig. 5. Instruction format.

TABLE III
RESOURCE UTILIZATION AND MAXIMUM CLOCK RATE FOR DIFFERENT n ON A XILINX XCV1000-6. THE XILINX XCV1000-6 CONTAINS 12 288 SLICES (6144 CLBS)

n	# of slices	Reported freq (MHz)
113	1410	31
155	1868	30
173	2148	28
281	3315	26
371	4247	22
473	5264	18

a better choice since a register-based interface does not require that the entire bitstream be downloaded every time the input parameters are changed.

F. Implementation Platform

The ECP was implemented on an Annapolis Micro Systems Wildstar board [17]. It is a PCI board consisting of three processing elements (PEs) (Xilinx Virtex XCV1000-6 [18]), of which only one was used. The XCV1000-6 has 128 Kbits of dedicated dual-ported synchronous 4096-bit RAM block SelectRAM (arranged as 32×4-Kbit blocks) and 6144 configurable logic blocks (CLBs), which are arranged as 12 288 slices.

The basic building block of the Virtex FPGA is the logic cell (LC). An LC includes a four-input function generator, carry logic, and a storage element. Each Virtex CLB contains four LCs, organized in two slices. The four-input function generator are implemented as four-input lookup tables (LUTs). Each of them can provide the functions of one four-input LUT or a 16×1-bit synchronous RAM (called "distributed RAM"). Furthermore, two LUTs in a slice can be combined to create a 16×2-bit or 32×1-bit synchronous RAM or a 16×1-bit dual-port synchronous RAM.

Apart from the use of bitstream reconfiguration, the other parts of the processor specific to the Virtex FPGA architecture were the use of Block SelectRAM for storage of the microcode

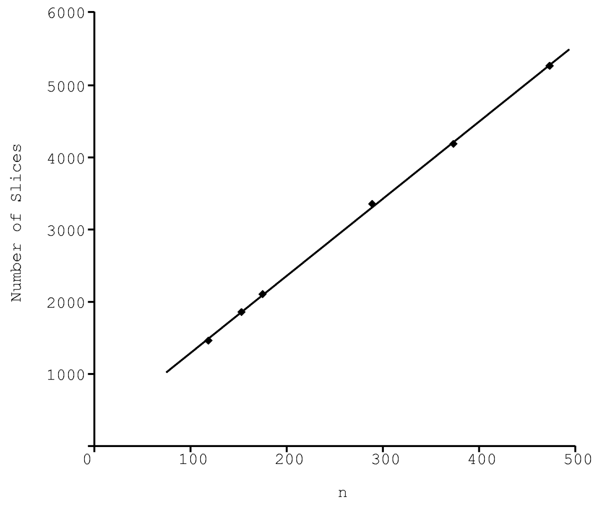

Fig. 6. Number of slices used for different n.

and for communications between the host and the PE, and the use of the LUT dual-port synchronous RAM for the ECP's registers. Thus the design should be easy to port to other FPGA and ASIC libraries. Dual-port LUT RAM and dual-port Block SelectRAMs make efficient use of FPGA resources and leads to a faster and smaller design.

IV. RESULTS

The EC processor was successfully tested on a Wildstar board (see Section III-F), the microcode being downloaded to the BlockRAMs by the host PC, and the parameters being downloaded to ROMs by bitstream reconfiguration.

A. EC Processor With Serial Multiplier $(k = 1)$

VHDL code for the elliptic curve processor was generated using the parameterized module generator for different values of n with an optimal normal basis. Synthesis and implementation were performed using Synopsys FPGA Express 3.4 and Xilinx Foundation 3.2i, respectively. Table III shows the resource utilization and maximum clock rate reported by the Xilinx tools

TABLE IV
EXECUTION TIME FOR ELLIPTIC CURVE MULTIPLICATION (PROJECTIVE COORDINATES) AND COMPARISON WITH A SOFTWARE IMPLEMENTATION

n	SW time (ms)	HW time (ms)	Speed-up
113	27.6	4.3	6
155	63.2	8.3	8
173	86.6	11.1	8

TABLE V
EXECUTION TIME FOR PROJECTIVE AND AFFINE COORDINATE IMPLEMENTATIONS OF ELLIPTIC CURVE MULTIPLICATION

n	Cycles (affine)	Cycles (proj.)	HW time affine (ms)	HW time proj.(ms)	$P : A$
113	148581	134484	4.8	4.3	0.9
155	324717	249879	10.8	8.3	0.77
173	402926	310043	14.4	11.1	0.77

TABLE VI
DYNAMIC INSTRUCTION COUNT (DYNAMIC INSTRUCTION FREQUENCIES IN PARENTHESES) FOR AN ELLIPTIC CURVE MULTIPLICATION USING DIFFERENT n. THE "JUMP" ENTRY IS THE SUM OF JKZ, JCZ, AND JMP FREQUENCIES

	Projective			Affine		
n	113	155	173	113	155	173
NOP	291 (0.22%)	391 (0.16%)	444 (0.14%)	291 (0.2%)	391 (0.12%)	444 (0.11%)
XOR	616 (0.46%)	847 (0.34%)	946 (0.31%)	784 (0.53%)	1078 (0.33%)	1204 (0.3%)
MUL	128820 (95.79%)	242112 (96.89%)	301368 (97.2%)	127680 (85.93%)	288288 (88.78%)	359136 (89.13%)
ROTL	673 (0.5%)	925 (0.37%)	1033 (0.33%)	12825 (8.63%)	24102 (7.42%)	30015 (7.45%)
TFR	3625 (2.7%)	4972 (1.99%)	5548 (1.79%)	5432 (3.66%)	7931 (2.44%)	8858 (2.2%)
SHFR	123 (0.09%)	170 (0.07%)	188 (0.06%)	1233 (0.83%)	2465 (0.76%)	2753 (0.68%)
Jump	335 (0.24%)	461 (0.18%)	515 (0.18%)	335 (0.24%)	461 (0.15%)	515 (0.12%)
Total	134484 (100%)	249879 (100%)	310043 (100%)	148581 (100%)	324717 (100%)	402926 (100%)

for designs with different n. As can be seen in Fig. 6, resource requirements are linear with n. The size of the microcode is less than 512 16-bit words and does not change for different n or k.

The execution time of the processor was compared with that of an optimized software implementation of an optimal normal basis elliptic curve package [10] running on a SUN Enterprise E4500 with UltraSPARC-II 400-MHz processors and 8 GB of RAM. The results are presented in Table IV. It can be seen that the raw performance of the elliptic processor is approximately six to eight times faster than the software implementation. Note that the hardware times include interfacing overheads but do not include the time to modify the bitstream (approximately 260 ms) and download the bitstream (approximately 90 ms) via the PCI bus on the Wildstar. This 350-ms overhead could be reduced to a negligible value using the techniques discussed in Section III-E, making the overall speed of the ECP comparable to that of a typical workstation. However, the ECP is a single-chip implementation and has advantages in terms of cost, memory, energy, size weight, and real-time performance.

TABLE VII
k-WAY PARALLEL FIELD MULTIPLIER RESOURCE UTILIZATION, NUMBER OF CYCLES, AND EXECUTION TIME FOR A CURVE MULTIPLICATION USING PROJECTIVE COORDINATES

k	Slices	Cycles	Time (ms)	Slices	Cycles	Time (ms)
	n=113			n=473		
1	1410	134484	4.3	5264	2267501	126.2
2	1860	71204	2.6	6928	1150278	69.2
4	1970	39564	1.7	7396	591666	35.7
6	2076	28264	1.2	7872	402306	24.5
8	2182	23744	1.06	8340	312360	19.1
10	2300	20354	0.93	8799	253246	15.7
12	2434	18094	0.89	9288	217680	13.8
14	2515	16964	0.84	9752	192602	13.5
16	2614	15833	0.81	10229	170340	12.7
24	–	–	–	12160	147354	12.3
32	3524	12188	0.79	–	–	–
64	5572	10375	0.77	–	–	–
113	8753	9187	0.75	–	–	–

B. Projective Verses Affine Coordinates

The projective and affine implementations share the same hardware design with different microcode and hence occupy the same circuit area. The total number of cycles required for an elliptic curve multiplication for various n are given in Table V, where we assume that the c of (4) is an n-bit binary number with half the number of bits set. The execution time required for an elliptic curve multiplication at the maximum frequency is shown in Table V. These figures were obtained by multiplying the number of cycles by the minimum period reported by the Xilinx implementation tools. They do not include the time for host processor interfacing nor the time for downloading the bitstream. The implementation using projective coordinates is always faster than using affine coordinates.

C. Parallel Multiplier $(k > 1)$

The dynamic instruction frequencies for a curve multiplication using different n are shown in Table VI. From the table, it can be clearly seen that the bottleneck is in field multiplication (MUL), which accounts for approximately 90% of the execution time.

The resources used and time taken for a curve multiplication using a parallel multiplier for $n = 113$ and 473 are shown in Table VII. For the $n = 113$ case, a fully parallel multiplier could be used, whereas for $n = 473$, the maximally parallel multiplier that can fit in an XCV1000E corresponded to $k = 24$.

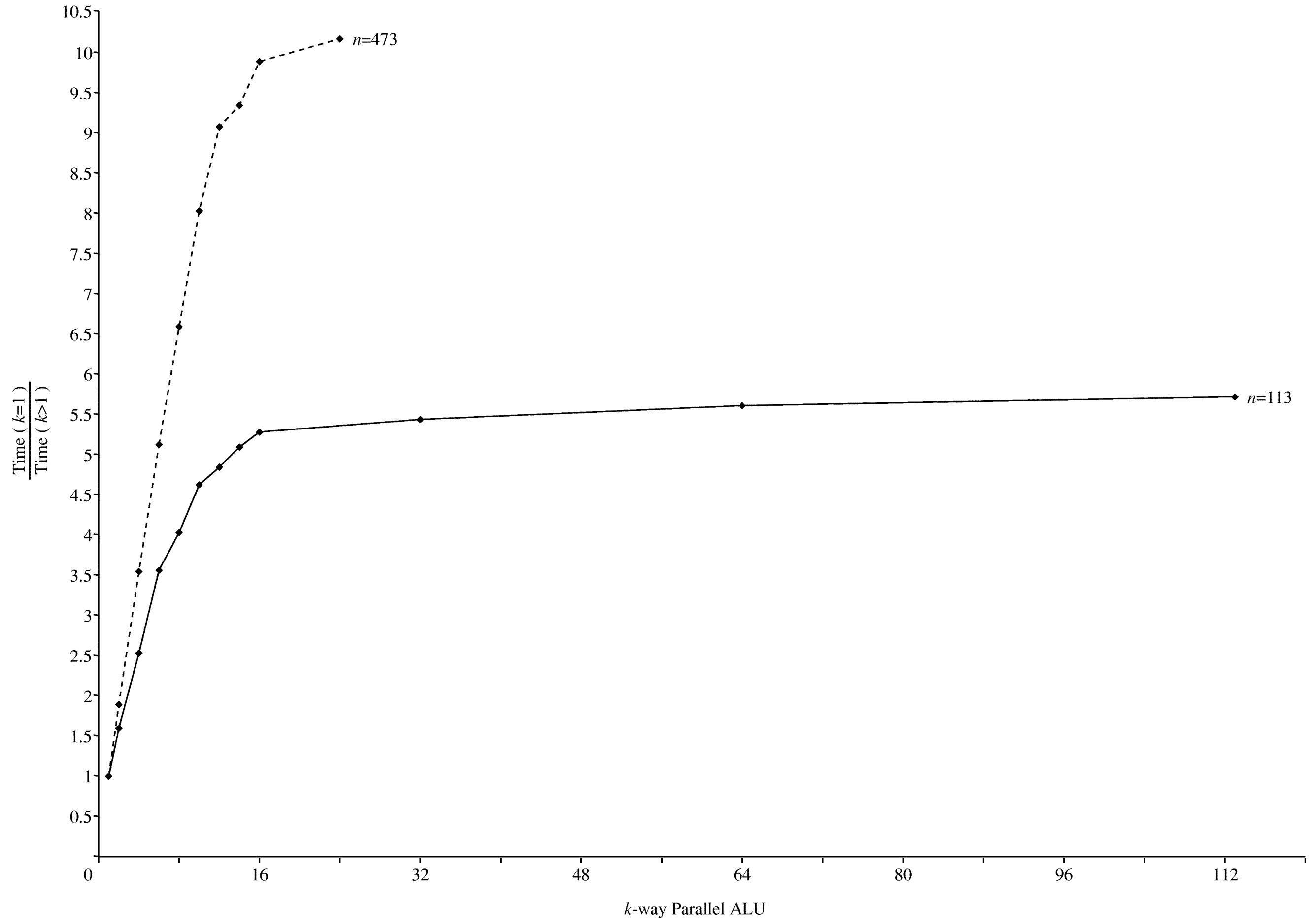

Fig. 7. Normalized execution time for one curve multiplication using a k-way parallel field multiplier.

Fig. 7 is a plot of normalized performance verses the degree of parallelism k. As can be seen from the figure, the execution time improves as parallelism is increased and tradeoffs between area and performance can be easily made. Improvement is initially linear for small k, after which degradation occurs mainly because of increased routing delays. The performance then quickly saturates. As can be seen from the graph, for smaller n, degradation occurs at a smaller value of k. A sensible tradeoff between performance and area could be achieved by limiting k to the linear region of Fig. 7.

V. Conclusion

The feasibility and utility of implementing a microcoded elliptic curve processor over F_{2^n} for arbitrary n was demonstrated. We believe that a microcoded implementation has a shorter development time and is more flexible than an FSM-based implementation. Moreover, the processor's I/O requirements are simply the inputs and outputs of the elliptic curve multiplication and are hence much lower than that of a processor that only implements field operations. Experiments with the processor showed that projective coordinates are always faster than affine and that field multiplication accounts for approximately 90% of the execution time of a curve multiplication. By increasing the parallelism of the field multiplier unit, significant speed improvements can be gained.

References

[1] N. Koblitz, "Elliptic curve cryptosystems," *Math. Computat.*, vol. 48, pp. 203–209, 1987.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in *CRYPTO '85*, 1986, pp. 417–426.

[3] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$," *IEEE Trans. Select. Areas Commun.*, vol. 11, pp. 804–813, 1993.

[4] J. S. P. Ivey, S. Walker, and S. Davidson, "An ultra-high speed public key encryption processor," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1992, pp. 19.6.1–19.6.4.

[5] M. Rosner, "Elliptic curve cryptosystems on reconfigurable hardware," master's thesis, Worcester Polytechnic Inst., Worcester, MA, 1998.

[6] G. Orlando and C. Paar, "A super-serial galois field multiplier for FPGA's and its application to public key algorithms," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM'99)*, 1999, pp. 232–239.

[7] K. H. Leung, K. W. Ma, W. K. Wong, and P. H. W. Leong, "FPGA implementation of a microcoded elliptic curve cryptographic processor," in *Proc. Field-Programmable Custom Computing Machines (FCCM'00)*, 2000, pp. 68–76.

[8] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*. Norwell, MA: Kluwer Academic, 1993.

[9] A. J. Menezes, P. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1999.

[10] M. Rosing, *Implementing Elliptic Curve Cryptography*. Greenwich, CT: Manning, 1998.

[11] I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*. Cambridge, U.K.: Cambridge Univ. Press, 1999.

[12] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, "Optimal normal bases in $GF(p^n)$," *Discrete Appl. Math.*, vol. 22, pp. 149–161, 1988/1989.

[13] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases," *Info. Comput.*, vol. 78, no. 3, pp. 171–177, 1988.

[14] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone, "An implementation for a fast public-key cryptosystem," *J. Cryptol.*, vol. 3, pp. 63–79, 1991.

[15] L. Wall, T. Christianson, and R. L. Schwartz, *Programming Perl*, 2nd ed: O'Reilly, 1996.

[16] *Xilinx Applications Note XAPP 151: Virtex Configuration Architecture Advanced User Guide*, Xilinx, 2000.

[17] *Wildstar Reference Manual Revision 3.3*, Annapolis Micro Systems, Inc, 1999.

[18] *Virtex 2.5 V Field Programmable Gate Arrays*, Xilinx, 2000.

[19] J. A. Solinas, "An improved algorithm for arithmetic in a family of elliptic curves," in *CRYPTO '97*, 1997, pp. 357–371.

[20] R. Schroeppel, H. Orman, S. O'Mally, and O. Spatscheck, "Fast key exchange with elliptic curve systems," in *CRYPTO '95*, 1995, pp. 43–56.

[21] J. Guajardo and C. Paar, "Efficient algorithms for elliptic curve cryptosystems," in *CRYPTO '97*, 1997, pp. 343–356.

[22] J. Lopez and R. Dahab, "An improvement of the guajardo-paar method for multiplication on nonsupersingular elliptic curves," in *Proc. XVIII Int. Conf. Chilean Society of Computer Science (SCCC'98)*, 1998, pp. 91–95.

[23] S. Sutikno and A. Surya, "An architecture of $F_{2^{2N}}$ multiplier for elliptic curves cryptosystem," in *Proc. IEEE 2000 Int. Symp. Circuits and Systems*, 2000, pp. 279–282.

[24] L. Gao, S. Shrivastava, and G. E. Sobelman, "Elliptic curve scalar multiplier design using FPGA's," in *Proc. Cryptographic Hardware and Embedded Systems (CHES'99)*, 1999, pp. 257–268.

[25] J. López and R. Dahad, "Fast multiplication on elliptic curves over $GF(2^m)$ without precompuatation," in *Proc. Cryptographic Hardware and Embedded Systems (CHES'99)*, 1999, pp. 257–268.

[26] *IEEE Standard Specifications for Public-Key Cryptography*, IEEE Standard 1363, 2000.

Philip H. W. Leong (SM'87) received the B.Sc., B.E., and Ph.D. degrees from the University of Sydney, Australia, in 1986, 1988, and 1993, respectively.

In 1989, he was a Research Engineer at AWA Research Laboratory, Sydney. From 1990 to 1993, he was a Postgraduate Student and Research Assistant at the University of Sydney, where he worked on low-power analog VLSI circuits for arrhythmia classification. In 1993, he was a Consultant to SGS Thomson Microelectronics in Milan, Italy. He was a Lecturer at the Department of Electrical Engineering, University of Sydney, from 1994 to 1996. He is currently an Associate Professor in the Department of Computer Science and Engineering, Chinese University of Hong Kong, and Director of the Custom Computing Laboratory. He is the author of more than 50 technical papers and three patents. His research interests include reconfigurable computing, digital systems, parallel computing, cryptography, and signal processing.

Ivan K. H. Leung received the B.Eng. and M.Phil. degrees from the Chinese University of Hong Kong in 1999 and 2001, respectively, where he is currently pursuing the Ph.D. degree.

He is with the Center for Large Scale Computation, Chinese University of Hong Kong. His research interests include reconfigurable computing, cryptography, VLSI, and parallel computing.