Customizable Elliptic Curve Cryptosystems

Ray C. C. Cheung, *Student Member, IEEE*, Nicolas Jean-baptiste Telle, Wayne Luk, *Member, IEEE*, and Peter Y. K. Cheung, *Senior Member, IEEE*

***Abstract*—This paper presents a method for producing hardware designs for elliptic curve cryptography (ECC) systems over the finite field $\mathrm{GF}(2^m)$, using the optimal normal basis for the representation of numbers. Our field multiplier design is based on a parallel architecture containing multiple m-bit serial multipliers; by changing the number of such serial multipliers, designers can obtain implementations with different tradeoffs in speed, size and level of security. A design generator has been developed which can automatically produce a customised ECC hardware design that meets user-defined requirements. To facilitate performance characterization, we have developed a parametric model for estimating the number of cycles for our generic ECC architecture. The resulting hardware implementations are among the fastest reported: for a key size of 270 bits, a point multiplication in a Xilinx XC2V6000 FPGA at 35 MHz can run over 1000 times faster than a software implementation on a Xeon computer at 2.6 GHz.**

***Index Terms*—Field-programmable gate arrays (FPGAs), parallel architectures, public key cryptography, security.**

I. Introduction

ELLIPTIC curve cryptography (ECC) is a public key cryptography system superior to the well-known RSA cryptography: for the same key size, it gives a higher security level than RSA [3]. ECC has been adopted to a wide spectrum of applications from digital certificates in webserver authentication [36] to embedded processors [51] in wearable devices.

This paper presents a scalable method for producing hardware designs for ECC systems over binary field $\mathrm{GF}(2^m)$ [26], using the optimal normal basis (ONB) [34] for the representation of numbers. Our design is based on a field multiplier architecture with multiple m-bit serial multipliers operating in parallel. An unique feature of our approach is a design generator which can automatically produce a customised ECC hardware design that satisfies specific user-defined requirements targeting to different applications. This method enables designers to rapidly explore and implement a design with the best tradeoffs in speed, size and level of security.

When optimized for speed, our design generator produces ECC designs with extensive parallelization and pipelining. These designs do not involve instructions, to avoid overhead associated with instruction fetch and decode. Our architecture is generic: for instance, a user-defined parameter controls the amount of parallelism in evaluating field multiplication. Note that the value of this parallelism does not affect the level of security, but the time to complete a field multiplication. Once this and other parameters such as the number of parallel field multipliers are decided, various design-specific constants, wiring patterns and data widths are generated automatically. Since our approach applies two levels of parallelisms in field and point multiplications and does not involve instruction fetch and decode, significant speedup is achieved when compared with previous customised processor designs.

To facilitate performance characterization, we have developed a parametric model for estimating the number of cycles for our generic ECC architecture. This model is expressed in terms of various customization parameters, such as the key size, the amount of parallelism in field operations, and the number of cycles for basic operations, such as point addition and point multiplication.

As an example of our implemented designs, for a key size of 270 bits, a point multiplication, which is the slowest operation in the ECC method, can be computed in 0.17 ms with our hardware design implemented in an XC2V6000 field-programmable gate array (FPGA) at 35 MHz. In contrast, a software implementation requires 196.71 ms on a Xeon computer at 2.6 GHz; so our FPGA design is more than 1150 times faster, while its clock speed is almost 74 times slower than the Xeon processors.

To summarize, our major achievements include:

- a fully parametric parallel and pipelined design for field multiplication operation (Section IV);
- an optimized point multiplication using parallel field multipliers (Section V);
- a parametric model for calculating the number of cycles for our generic ECC system and for estimating the security/size/speed tradeoffs (Section VI);
- a design generator that takes the key size and degree of parallelism of the design and generate efficient "hard-core" control and data path (Section VII).

The rest of the paper is organized as follows. Section II describes related work in ECC cryptosystems. Section III covers the mathematics behind ECC designs. Section IV focuses on a parallel field multiplier. Section V presents the architecture of our ECC cryptosystem and its components. Section VI provides the mapping from the architecture to reconfigurable hardware, and the parametric model for estimating design tradeoffs. Section VII addresses design automation and customization by a design generator. Section VIII evaluates our results and compares them against existing hardware ECC implementations. Finally, Section IX summarizes our approach and outlines current and future research.

Manuscript received October 10, 2004; revised March 7, 2005. This work was supported by Celoxica, Xilinx Inc., the Croucher Foundation, and by the Engineering and Physical Sciences Research Council (EPSRC), U.K., under Grant GR/R 31409 and Grant GR/R 55931, GR/N 66599.

R. C. C. Cheung, N. J. Telle, and W. Luk are with the Department of Computing, Imperial College London, London W5 4R5, U.K. (e-mail: rcheung@doc.ic.ac.uk; njt02@doc.ic.ac.uk; wl@doc.ic.ac.uk).

P. Y. K. Cheung is with the Department of Electrical & Electronic Engineering, Imperial College London, London W5 4R5, U.K. (e-mail: p.cheung@imperial.ac.uk).

Digital Object Identifier 10.1109/TVLSI.2005.857179

II. Related Work

The difficulty of the underlying elliptic curve discrete logarithm problem (ECDLP) makes ECC cryptosystems suitable for applications that need long-term security and low bandwidth measurements. In 2002, the U.S. Government adopted ECC for protecting mission-critical information [9]. For instance, the NIST has recommended specific curves for implementations [37] and the IEEE has provided detailed specification for the choices of private key length and fields [42].

ECC research has been extensively conducted [19], [32] and it can be first divided into two groups depending on the underlying field representation: prime field, $\mathrm{GF}(p)$ and binary field, $\mathrm{GF}(2^m)$. Two bases, Optimal Normal Basis and Polynomial Basis are commonly used for manipulating binary fields. With PB representation, field elements are represented as polynomials, while with NB the irreducible polynomial used in PB is not required. It is known that binary field is more suitable for hardware implementation, and ONB is said to dominate fast squaring. A recent study [13] reports that ONB ECC hardware design can outperform the same design using PB by 27 times. Fast point multiplication requires efficient field multiplication, inversion [20], squaring and an efficient coordinate system [27]. In recent years, there has been much research in software [8], [18] and hardware [39], [40] for both $\mathrm{GF}(p)$ and $\mathrm{GF}(2^m)$ focusing on the performance of point multiplication, and it is obvious that a fast point multiplication design is necessary and crucial.

The first ECC hardware implementation [1] is presented in 1989. Previous hardware work includes: the first ASIC implementation with Motorola M68008 microcontroller [1], reconfigurable finite-field multiplier [22], ASIC designs for field operations over specific $\mathrm{GF}(2^m)$ fields [29], an ECC implementation on the 8051 microprocessor in smart cards [53], an ECC processor on a smart card device [43], and recent FPGA implementations for ECC designs including [5], [6], [17], [24], [25], [28], [36], [39]–[41]. In this paper, we aim at optimising the field multiplier for normal basis and adopting state-of-the-art algorithm for point multiplication, to build a customisable and efficient ECC cryptosystem.

III. Mathematical Background

The field and group theories relevant to ECC designs have been extensively studied [26]. In this section, we briefly describe the mathematical basics that are necessary for building the field multiplier. The polynomial basis (PB) is formed by the set of $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ where α is a root of the nonreducible prime polynomial $P(x)$ of degree m, while the Normal Basis (NB) uses another set $\{\alpha, \alpha^2, \ldots, \alpha^{2^{m-1}}\}$ [26]. There is always a tradeoff using different bases for both software and hardware implementation. For instance, squaring is easier in NB than in PB, while inversion in NB is slower than in PB. The underlying arithmetic of both PB and NB has been extensively studied [31] and the corresponding high-performance multipliers [44], [49] are also reported.

We first introduce field multiplication for normal basis in this section, and describe the architecture of our field multiplier in Section IV. Given that we multiply two elements A, B of

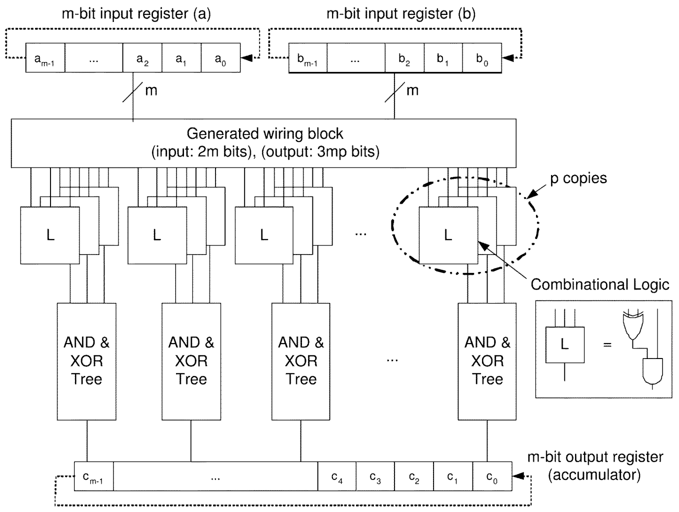


Fig. 1. Parametized, pipelined, and parallel field multiplier design.

$\mathrm{GF}(2^m)$ using normal basis and the product is another element C in the same field, then

$$A = \sum_{i=0}^{m-1} a_i \alpha^{2^i}, \quad B = \sum_{j=0}^{m-1} b_j \alpha^{2^j}, \quad C = \sum_{k=0}^{m-1} c_k \alpha^{2^k}$$

where $C = A \times B$, and the coefficients of element C, c_k $(k \in [0, m-1])$ can be expressed as follows [45]:

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

The λ_{ij}'s are the elements of the *multiplication matrix* Λ, which are either 0 or 1 [45]. An optimal normal basis means that the number of nonzero λ_{ij} is minimized. Since most of the λ values are zero, the sum of the inner products in the above equation can be written in the form:

$$\sum_{i=0}^{m-1} a_{i+k} b_{j+k} \lambda_{ij} = b_r(a_p + a_q)$$

for some $r, p, q \in [0, m-1]$, except for $j = 0$ where (for some $r, p \in [0, m-1]$):

$$\sum_{i=0}^{m-1} a_{i+k} b_{j+k} \lambda_{ij} = b_r(a_p).$$

There are two types of ONB, Types I and II, which are determined by the m value and the values in the λ matrix. In this paper, we handle both cases.

IV. Parallelized Field Multiplier

One of the contribution of this paper is the design of a parallel field multiplier for normal basis. This multiplier is used in many different parts of our customisable cryptosystem which is described in Section V.

Our field multiplier architecture is based on having p copies of an m-bit multiplier operating in parallel. Fig. 1 shows the datapath of our field multiplier, which is inspired by a nonparameterised architecture [30], [50]. The wiring block is auto-

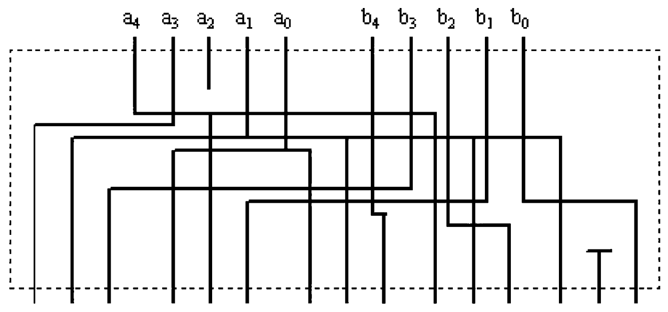


Fig. 2. Example of a wiring block when $m = 5$ and $p = 1$.

matically generated for different m values, an example of wiring pattern is shown in Fig. 2 in which the degree of parallelism p is one. The details of design generation is described in Section V.

The field multiplication operation is presented in pseudo-code as follows with the following properties.

- $\texttt{parallel}\{\texttt{do(a)};\texttt{do(b)}\}$ means that both operations $\texttt{do(a)}$ and $\texttt{do(b)}$ are executed in parallel.
- In a given clock cycle, the value of a field element x is the one it had at the previous clock cycle. This property results that the program $\texttt{parallel}\{\texttt{a} = \texttt{b}; \texttt{b} = \texttt{a}\}$ swaps the values of the field element a and b because the following two operations are performed simultaneously at time $\texttt{t} : \texttt{a(t)} = \texttt{b(t}-\texttt{1)}$ and $\texttt{b(t)} = \texttt{a(t}-\texttt{1)}$.
- x_k denotes the kth bit of the field element x.

We first describe our serial field multiplier in the pseudo-code format, given that the wiring pattern has been first generated.

```
1 field_mult(a, b)
  {
2   for i = 0...m - 1
    {
      //compute the wiring pattern
3     inputs = wiring(a, b);
      //compute the outputs of the L functions
4     parallel for k = 0...m - 1
      {
5       temp_k <- L(inputs);
      }
      //left rotate all registers
      {
6       a <- left_rotate(a);
7       b <- left_rotate(b);
8       c <- left_rotate(c);
      }
      //XOR with the output register
9     c <- c XOR temp;
    }
    //final rotation : every c_k is at the right
      position
10  c <- left_rotate(c);
11  return c;
  }
```

The algorithm shows that such a computation takes $4m + 1$ cycles, as we can see that steps 3, 5–8, and 9 are repeated m times, plus one final rotation in step 10. We pipeline this field multiplier since the computation of the L functions, the wiring pattern operations and the input/output rotations can be performed concurrently in different pipeline stages.

The time complexity of the original m-bit serial algorithm is $O(m)$. The key element of our design is to parallelise it to achieve the complexity of $O(m/p)$ where p is the degree of parallelism. Thus, the pipelined field multiplier is mapped into a parallel design such that for each k value, instead of computing only $L(k, a, b)$ at one specific cycle, it will also compute

$$L(k, l\text{rotate}(1, a), lrotate(1, b)),\\ L(k, l\text{rotate}(2, a), l\text{rotate}(2, b)), \ldots,\\ L(k, l\text{rotate}(p-1, a), l\text{rotate}(p-1, b)),$$

where $lrotate(n, x)$ returns x rotated left by n bits.

In order to compute these functions, a new wiring pattern which outputs p layers of m groups of three values is developed. These layers are derived from the wiring pattern that we generate for the serial multiplier as shown in Fig. 1. If a layer (number 0) outputs for some k (for the kth group of three values) the values a_q, a_r and b_t, then the i^{th} layer will output the values a_{q-i}, a_{r-i} and b_{t-i} at position $k - i$. Besides the wiring modification, in each step the registers are rotated by p bits instead of only one bit that is used in the serial design. Next, we define

$$m = \mu \times p + \nu$$

where μ is the quotient and ν is the remainder, when m is not divisible by p. If $\nu \neq 0$, an extra step that calculates the last ν elements of each sum is added.

The parallel and pipelined algorithm is shown below. Note that if $m = p$, this design only takes $(1 + m/p + 1 + 1 = 4)$ cycles to compute the result.

```
1 field_mult_par(a, b, p)
  {
2   parallel
    {
3     inputs = wiring(a, b); //inputs is
        a m*p*3 table
4     a <- left_rotate(a, p); //left rotate
        by p bits
5     b <- left_rotate(b, p);
6     clear(output[0]...output[p - 1]);
    }
7   for i = 0...($\mu$) - 1
    {
8     parallel
      {
        //pipeline stage 1 : compute wiring
9       inputs = wiring(a, b);
        //pipeline stage 2: compute L
10      parallel for j = 0...p
        {
11        parallel for k = 0...m − 1
          {
12          output[j]$_{(k)}$ <− L(wiring_inputs);
          }
        }
        //pipeline stage 3: rotating
13      a <− left_rotate(a, p);
14      b <− left_rotate(b, p);
15      c <− left_rotate(c, p) XOR
          output[0] XOR...XOR output[p − 1];
      }
    }
16  c <− left_rotate(c, p) XOR
        output[0] XOR...XOR output[p − 1];
17  if(($\nu$)! = 0)
    {
18    parallel
      {
19      a <− left_rotate(a, r); //realign a
20      b <− left_rotate(b, r); //realign b
21      c <− left_rotate(c, p);
22      parallel for j = 0...($\nu$){
23        parallel for k = 0...m − 1{
24          output[j]$_{(k)}$ <− L(wiring_inputs);
          }
        }
      }
25    c <− c XOR output[0] XOR...XOR output[($\nu$) − 1];
26    c <− left_rotate(c, p); //final rotation
    }
27  else //only do the final rotation
28    c <− left_rotate(c, p)
29  return c;
}
```

The number of cycles is reduced if a larger p value is used. A field multiply operation takes $\lfloor m/p \rfloor + 5$ cycles if $r \neq 0$, and $(m/p) + 3$ cycles otherwise. The two remainder cycles are from step 18–25 in the above pseudocode. In contrast, the non-pipelined serial version takes $4m + 1$ cycles.

V. System Architecture

This section describes the operations supported by our ECC architecture. These operations include field multiplication, field inversion and point multiplication, and their interaction is summarized in Fig. 3. In the following, m is the key size for our ECC architecture, which is a characteristic of the field. The subsections below cover field multiplication, field inversion, field squaring, point multiplication, point addition and data embedding respectively.

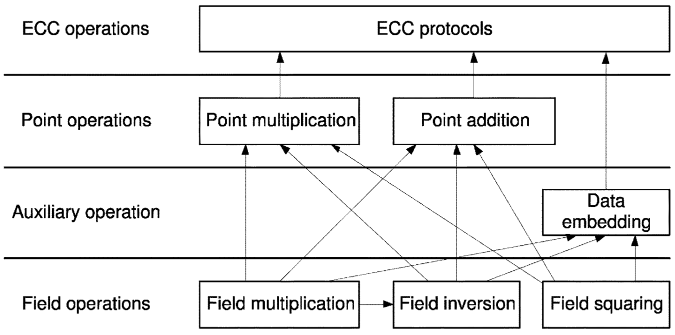


Fig. 3. Interactions between different operations.

A. Field Multiplication

In the proposed ECC customizable cryptosystem, the field multiplier presented in Section IV is the crucial component that is repeatedly used by other operations, as shown in Fig. 3. The datapath of our system for various operations is shown in Fig. 4. Users are able to select the second level parallelism in term of the number of executable field multipliers to the maximum four parallel field multipliers. Different scheduling optimizations have been applied to various designs which are summarized in the parametric model as shown in Section VI. Additional field multipliers do not bring further speed gains but result in an area penalty. There is a tradeoff between efficiency and area.

B. Field Inversion

The algorithm we use for field inversion is based on *Fermat's theorem* which states that in a normal basis

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

for all $a \neq 0$ in GF_{2^m}. This method was proposed by Itoh and Tsujii [20] and has been widely used for computing inversion. Using this formula to compute an inverse, it would require m multiplications. The following reduces the complexity of the inversion, as it is easy to calculate $x^{2^{(m-1)/2}+1}$ where x is shifted $(m-1)/2$ times and then multiplied by x.

$m - 1$ is even:

$$\begin{aligned} a^{(2^{m-1}-1)} &= a^{(2^{(m-1)/2}+1)\cdot(2^{(m-1)/2}-1)} \\ &= \left(a^{2^{(m-1)/2}-1}\right)^{2^{(m-1)/2+1}}. \end{aligned}$$

$m - 1$ is odd:

$$\begin{aligned} a^{(2^{m-1}-1)} &= a^{(2^{(m-1)/2}+1)\cdot(2^{(m-1)/2}-1)} \\ &= \left(a^{2^{(m-1)/2}-1}\right)^{2^{(m-1)/2+1}}. \end{aligned}$$

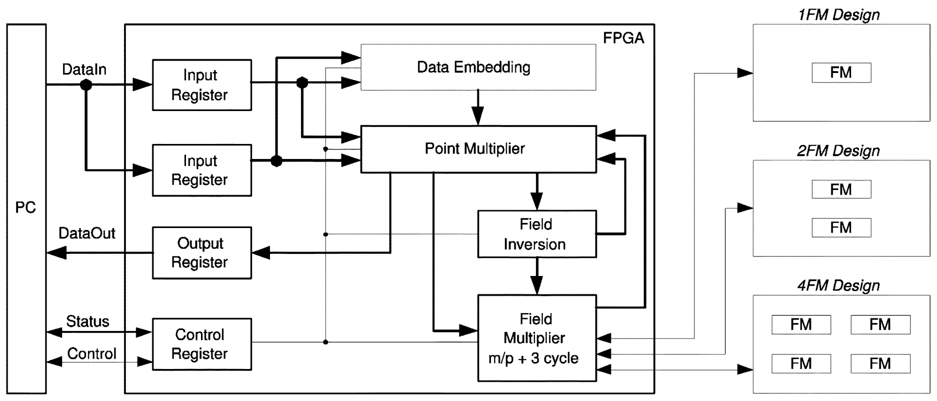


Fig. 4. Datapath of the customizable ECC system (FM denotes a field multiplier).

The algorithm is described below.

```
Input: $a \in \mathrm{GF}(2^m)$ to be inverted
Output: $x = a^{-1}$
 • $x \leftarrow a$;  $s \leftarrow \log_2(m) - 1$
 • while $s \geq 0$
   –$r \leftarrow$ right shift $m$ by $s$ bits
   –$y \leftarrow$ left shift $x$ by $\lfloor r/2 \rfloor$
   –$y \leftarrow$ multiply $x$ by $y$
   –if $x$ is odd
     $y \leftarrow$ left shift $y$ by 1 bit
     $y \leftarrow$ multiply $x$ by $y$
   –$x \leftarrow y$
   –$s \leftarrow s - 1$
 • $x \leftarrow$ left shift $x$ by 1 bit
 • return $x$
```

Since the time to perform a multiplication is usually much longer than the time to perform a shift operation, the number of cycles for this inversion algorithm can be approximated by the following equation [45] where T_{mult} represents the number of cycles to perform a field multiplication:

$$(\log_2(m-1) + \mathrm{number\_of\_bits\_set\_in}(m-1) - 1) \times T_{\mathrm{mult}}.$$

C. Field Squaring and Field Addition

The key benefit of using ONB representation is the simplicity of the squaring operation. Each element in the field $\mathrm{GF}(2^m)$ is represented by m binary digits. Field squaring is simply a cyclic shift [50] while field addition is a Boolean XOR operation. Therefore these two functions take up little space and are efficient in hardware.

D. Point Multiplication

This part presents the point operations based on the following elliptic curve:

$$y^2 + xy = x^3 + a_2x^2 + a_6.$$

A point is described by its x and y value in the above equation. In our point multiplication, we use projective coordinates such that each point on the curve is described by (X, Y, Z) on the following curve:

$$Y^2Z + XYZ = X^3 + a_2X^2Z^2 + a_6Z^3.$$

Instead of using the "add-and-double" (binary) method [7], we adopt the improved Montgomery Scalar Multiplication [27] in our design for point multiplication. Let P, P_1 and P_2 be three points of an elliptic curve E, and assume that the property $P_2 - P_1 = P$ holds. Let the (affine) x-coordinate of P_i $(i \in \{1, 2\})$ be X_i/Z_i where X_i and Z_i are the first and third projective coordinate of P.

It can be shown that the y-coordinate of any of the three points are not required in computing the x-coordinate (X/Z) of $P_1 + P_2$ or of $2P_i$. Hence, in the main loop of the algorithm, we can get rid of these y-coordinates and use the above formula to compute the x-coordinate of point doubling, $2P_i$ and point addition, $P_1 + P_2$ at each step. When the main computation is over, we can get the y-coordinate of the result point by using the following formula which is proved in [27]:

$$y_1 = (x_1 + x) \cdot \left(\frac{(x_1 + x) \cdot ((x_2 + x) + x^2 + y)}{x} \right) + y.$$

The Montgomery algorithm [27] for point multiplication we used is shown below:

```
Input: $k \in \mathrm{GF}(2^m)$ $(k = (k_{l-1}, \ldots, k_1, k_0))$, $P(x, y)$
a point of a certain curve $E$ defined by
$a_2$ and $a_6$
Output: $Q = k \cdot P$
 • $X_1 \leftarrow X$;  $Z_1 \leftarrow 1$;  $X_2 \leftarrow x^4 + a_6$;  $Z_2 \leftarrow x^2$
 • for $i$ from $l-1$ downto 0 do
   if $k_i = 1$ then
     $(Z_1, X_1) \leftarrow Madd(X_1, Z_1, X_2, Z_2)$;
     $(Z_2, X_2) \leftarrow Mdouble(X_2, Z_2)$
   else
     $(Z_2, X_2) \leftarrow Madd(X_2, Z_2, X_1, Z_1)$;
     $(Z_1, X_1) \leftarrow Mdouble(X_1, Z_1)$
 • return $Q = Mxy(X_1, Z_1, X_2, Z_2)$
where:
 • $Madd(X_1, Z_1, X_2, Z_2)$ returns $(Z_3, X_3)$ where
   $Z_3 = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2$
   $X_3 = x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1)$
 • $Mdouble(X_1, Z_1)$ returns $(Z_3, X_3)$ where
   $Z_3 = {Z_i}^2 \cdot {X_i}^2$
   $X_3 = {X_i}^4 + a_6 \cdot {Z_i}^4$
 • $Mxy(X_1, Z_1, X_2, Z_2)$ returns $Q(x_1, y_1)$ where
   $x_1 = X_1/Z_1$
   $y_1 = (x_1 + x) \cdot ((x_1 + x) \cdot (x_2 + x) + x^2 + y)/x + y$.
```

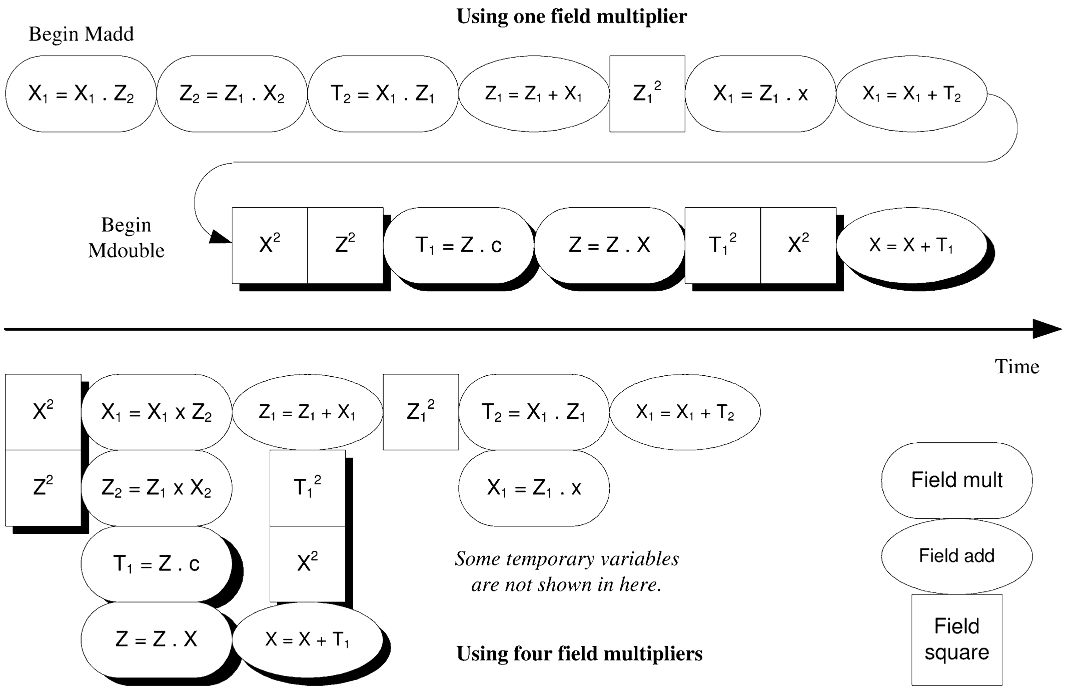


Fig. 5. Applying multiple parallel field multipliers in point multiplication.

This algorithm requires $6l + 10$ multiplications and only one inversion, where l is the number of bits needed to represent k in base 2. The value of l is usually close to the value of m. The algorithm used for point multiplication is mainly sequential, since each step of the loop needs the results of the previous step to start.

Since the two functions $Madd$ and $Mdouble$ can be computed independently in parallel as illustrated in Fig. 5, the number of cycles for point multiplication can be reduced. The reason is that the $Madd$ function requires four field multiplications, and the $Mdouble$ function requires two field multiplications; the sequential execution of $Madd$ and $Mdouble$ takes the time of six field multiplications, while a design using four parallel field multipliers will significantly reduce the time to that for one field multiplier. Due to data dependency and area overhead, there is no gain in using more than four field multipliers. This improvement is significant, since each step in the loop is executed m times, where m is typically between 100 and 500 for a secure system [37]. This flexible field multiplication function can be made large or small, fast or slow depending on the performance required. Since other functions involve extensive use of the field multiplier, an optimized design is crucial and has a large impact on the final ECC system.

E. Point Addition and Point Subtraction

Although point addition is not as common as point multiplication, some security protocols require both. The algorithm used to compute this operation is simple. Adding $P(x_1, y_1)$ and $Q(x_2, y_2)$ $(P \neq Q)$ gives $R(x_3, y_3)$ where

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

The details of the algorithm can be found in [45]. Our system will also support point subtraction. To compute subtraction, we develop an algorithm that can invert a point. The negation of a point $P(x, y)$ is $-P(x, x + y)$. The addition algorithm requires one field inversion and two field multiplications. The point inversion algorithm takes negligible time, as it only requires an XOR operation.

F. Data Embedding

Data embedding embeds data onto a point of an elliptic curve. Not all elements of $\mathrm{GF}(2^m)$ are the x-coordinate of a point of a given elliptic curve, and the ECC technique only allows the encryption of a point on an elliptic curve. It is therefore necessary to embed data into a point in order to encrypt them.

It has been shown [45] that, given a specific elliptic curve, if 5 *don't care* bits are appended to $m - 5$ bits of data and forms data $embed\_data$, there always exists at least one combination of the don't care bits such that the value x of $embed\_data$ stays on the curve. The outline of the embedding algorithm is as follows.

```
Input: $d$ data written in base 2 (the
data must be of length $m-5$ bits)
Output: $M(x,y)$ point in which the data
to be encrypted are stored
 • $x \leftarrow append(d, 00000_2)$
 • while not $on\_curve(x)$
   increment $x$
 • $compute\_y(x)$
 • return $M(x,y)$
```

where $on\_curve(x)$ checks if $x \in \mathrm{GF}(2^m)$ is the x-coordinate of a point of the curve that we are working on. $compute\_y(x)$

returns the y-coordinate of one of the two points whose x-coordinate is x. The equation of the curve is

$$y^2 + xy = x^3 + a_2x^2 + a_6 \Rightarrow y^2 + xy + f(x) = 0$$

where $f(x) = -\left(x^3 + a_2x^2 + a_6\right)$. Let $y = zx$. Given $x \neq 0$, the equation becomes

$$(xz)^2 + x^2z + f(x) = 0 \Rightarrow z^2 + z + c = 0$$

where $c = f(x) \cdot x^{-2}$. The function $on\_curve(x)$ is described below:

```
Input: $x \in \mathrm{GF}(2^m)$ bits
Output: true if $x$ is the $x$-coordinate of
a point, false otherwise
 • $c \leftarrow \left(x^3 + a_2x^2 + a_6\right) \cdot x^{-2}$
 • $trace \leftarrow$ XOR of all bits in $c$
 • if $trace = 1$ then
   return false
 • else
   return true
```

We compute $-f(x) \cdot x^{-2}$ instead of $f(x) \cdot x^{-2}$ since for all $u \in \mathrm{GF}(2^m)$, $-u$ is defined by $u + (-u) = 0$, that is u XOR $(-u) = 0$ and $-u = u$. To compute the y-coordinate given the x-coordinate, we can rewrite the equation that we are working on as ($i \in [0, m-1]$):

$$z = z^{1/2} + c^{1/2} \Rightarrow z_i = z_{i-1} + c_{i-1}.$$

Moreover, if z is a solution to our equation, then the other solution is $z+1$. It is easy to prove this by assuming $z^2+z+c = 0$ and by calculating

$$\begin{aligned}(z+1)^2 + (z+1) + c &= z^2 + 2z + 1 + z + 1 + c\\ &= z^2 + z + c + 2(z+1)\\ &= z^2 + z + c = 0\end{aligned}$$

In this proof, an addition is just an XOR operation in $\mathrm{GF}(2^m)$, then for all $u \in \mathrm{GF}(2^m)$, $2u = 0$. Since $z+1$ is actually $\overline{z}$ in a normal basis, in one of the two solutions the least significant bit will be 0 and the other one will be 1. As a result, the least significant bit of the solution that we are looking for is equal to 0. We then further compute all the other bits one by one. To compute the y value, we simply multiply z by x.

The algorithm that performs the operation $compute\_y(x)$ is described below:

```
Input: $x \in \mathrm{GF}(2^m)$ (and $c$ calculated in
function $on\_curve(x)$)
Output: $M(x, y)$ a point of the curve we
are working on
 • $z_0 \leftarrow 0$
 • for $i$ from 1 to $m-1$
   • $z_i \leftarrow z_{i-1} + c_{i-1}$
 • $y \leftarrow x \cdot z$
 • return $y$
```

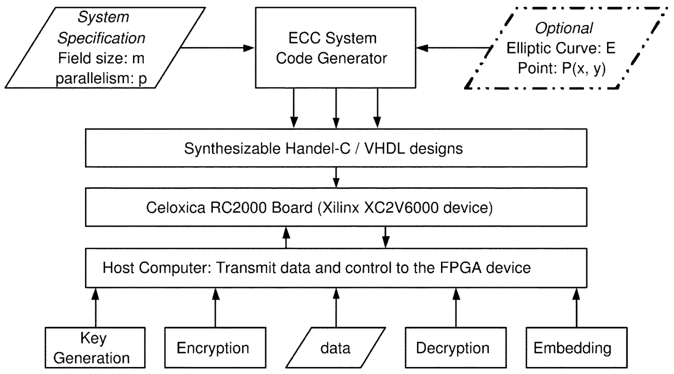

System Specification
Field size: m
parallelism: p
ECC System Code Generator
Optional
Elliptic Curve: E
Point: P(x, y)
Synthesizable Handel-C / VHDL designs
Celoxica RC2000 Board (Xilinx XC2V6000 device)
Host Computer: Transmit data and control to the FPGA device
Key Generation
Encryption
data
Decryption
Embedding

Fig. 6. Overview of the customizable ECC system.

VI. FPGA IMPLEMENTATION

This section presents the implemention of our design to produce a customisable encryption/decryption system. We have implemented our designs in the Handel-C language [10]. The key components in our design are: field operations (multiplication, inversion, squaring, addition), point operations (multiplication, addition), and data embedding. Fig. 6 provides an overview of our system.

Functions are implemented as shared logic. They will only be mapped once on the hardware. We then generate the routing logic and the control logic to send the appropriate data to these functions and fetch the result when necessary. The system performance has been optimized by exploring the maximum possible parallelism between operations at both the field-operation level and the point-operation level (Fig. 3). We also notice that because these functions called each other many times and have to send large values (usually two or three m bits values) to each other every time, the design's speed is limited by parameter passing. To tackle this problem, all function calls are pipelined.

The estimation of the number of clock cycles required by each function is presented in Table I. Notations for this table include: $T_{\mathrm{mult}}, T_{\mathrm{inv}}$ and $T_{\mathrm{point\_add}}$ which represent the number of cycles for field multiplication, field inversion and point addition functions respectively, and $s(x)$ and $ns(x)$ represent the number of set bits and of clear bits in the binary representation of x. The $nb\_attempts$ in the data embedding formula represents the number of times the data need to be incremented to find the x-coordinate of a point.

VII. AUTOMATIC GENERATION AND CUSTOMISATION

This section presents a design generator which can automatically produce implementations with optimized speed, size and level of security. The major customisable elements of a cryptosystem are: the key size m, the degree of parallelism p, and the protocols of the system.

We develop a program that takes a valid ONB m-value (type I or type II optimal normal basis) and the degree of parallelism in the field multiplication function, and produces synthesizeable Handel-C code. This design generator first computes the Λ table for the given m using the algorithm presented in [45]. This algorithm generates an $m \times 2$ matrix (Lambda) where the jth row contains two values of i for which:

TABLE I
NUMBER OF CLOCK CYCLES REQUIRED FOR EACH FUNCTION TO EXECUTE. FM MEANS THE NUMBER OF FIELD MULTIPLIER, T_{mult}, T_{inv}, T_{point_add} MEANS THE NUMBER OF CYCLES FOR FIELD MULTIPLICATION, FIELD INVERSION, AND POINT ADDITION

Operation	Number of cycles
	Field Operations
Addition	1
Squaring	1
Multiplication	$5+\lfloor m/p \rfloor$ or $3+(m/p)$ if m is divisible by p
Inversion	$2+(\lceil log_2(m)\rceil-2)\times(3+T_{mult})+s(m-1)\times(3+T_{mult})+ns(m-1)$
	Point Operations
Addition	$12+2T_{mult}+T_{inv}$
Subtraction	$1+T_{point\_add}$
	Point Multiplication
Unoptimised	$28+10T_{mult}+T_{inv}+(m-\lceil log_2(k)\rceil)+(\lceil log_2(k)\rceil-1)\times(14+6T_{mult})$
1 FM	$14+10T_{mult}+T_{inv}+(m-\lceil log_2(k)\rceil)+(\lceil log_2(k)\rceil-1)\times(8+6T_{mult})$
2 FM	$10+6T_{mult}+T_{inv}+(m-\lceil log_2(k)\rceil)+(\lceil log_2(k)\rceil-1)\times(9+4T_{mult})$
4 FM	$9+5T_{mult}+T_{inv}+(m-\lceil log_2(k)\rceil)+(\lceil log_2(k)\rceil-1)\times(4+2T_{mult})$
Data embedding	$6+3T_{mult}+nb\_attempts(8+T_{inv}+T_{mult})$

TABLE II
COMPARISON BETWEEN OUR DESIGN (VIRTEX-II XC2V6000 DEVICE USING 0.15 MICRON) AND THE REFERENCE DESIGNS [25], [45] (VIRTEX XCV1000 DEVICE USING 0.25 MICRON). THE SYMBOL (*) DENOTES EXTRAPOLATED RESULTS BASED ON PUBLISHED DATA FOR DIFFERENT m VALUES

Parallelism	P&R results		Measured results		Software [45]			Hardware (serial) [25]		
p	clock	time(ms)	clock	time(ms)	clock	time(ms)	speedup	clock	time(ms)	speedup
					$m=113$					
2	42MHz	0.36	56MHz	0.27	2.6GHz	15.99	59.22	31MHz	4.3	15.93
8	42MHz	0.13	56MHz	0.09	2.6GHz	15.99	177.67	31MHz	4.3	47.78
16	42MHz	0.08	56MHz	0.06	2.6GHz	15.99	266.50	31MHz	4.3	71.67
32	42MHz	0.07	56MHz	0.04	2.6GHz	15.99	399.75	31MHz	4.3	107.5
56	42MHz	0.05	56MHz	0.04	2.6GHz	15.99	399.75	31MHz	4.3	107.5
					$m=162$					
2	40MHz	0.83	54MHz	0.55	2.6GHz	45.67	83.04	29MHz*	9.39*	17.07
8	40MHz	0.27	54MHz	0.17	2.6GHz	45.67	268.65	29MHz*	9.39*	55.24
16	40MHz	0.18	54MHz	0.11	2.6GHz	45.67	415.18	29MHz*	9.39*	85.36
32	40MHz	0.13	54MHz	0.07	2.6GHz	45.67	652.43	29MHz*	9.39*	134.14
56	40MHz	0.10	54MHz	0.06	2.6GHz	45.67	761.17	29MHz*	9.39*	156.5
					$m=270$					
2	24MHz	3.28	35MHz	2.24	2.6GHz	196.71	87.82	26MHz*	27.99*	12.50
8	24MHz	0.92	35MHz	0.63	2.6GHz	196.71	312.24	26MHz*	27.99*	44.43
16	24MHz	0.53	35MHz	0.36	2.6GHz	196.71	546.42	26MHz*	27.99*	77.75
32	24MHz	0.35	35MHz	0.24	2.6GHz	196.71	819.63	26MHz*	27.99*	116.63
56	24MHz	0.25	35MHz	0.17	2.6GHz	196.71	1157.12	26MHz*	27.99*	164.65

if GF(2^m) has an ONB of type I: 2^i+2^j equals 1 or 0 in $\bmod(m+1)$,
if GF(2^m) has an ONB of type II: $2^i \pm 2^j$ equals ± 1 in $\bmod(2m+1)$

When $j=0$, there is only one value of i that satisfies the equations. From the Λ table, we generate the wiring pattern required by our field multiplication design. This wiring rearranges the m-bit inputs `a` and `b` into $3p$ m-bit variables

$$\texttt{inputa1}[0],\ldots,\texttt{inputa1}[p-1],$$
$$\texttt{inputa2}[0],\ldots,\texttt{inputa2}[p-1],$$
$$\texttt{inputb}[0],\ldots,\texttt{inputb}[p-1],$$

where

$$\texttt{inputb}[i]\_k\ (\texttt{the } k^{th}\texttt{bit})\ \texttt{is } b\_(2k-i),$$
$$\texttt{inputa1}[i]\_k\ \texttt{is } a\_(\texttt{Lambda}[k,0]+k-i),$$
$$\texttt{inputa2}[i]\_k\ \texttt{is } a\_(\texttt{Lambda}[k,1]+k-i).$$

This design generator computes all the constants that are used by various functions. In particular, it computes $s=\lfloor m/p \rfloor$ and $r=m \bmod p$ used in field multiplication. It also calculates the size of those variables and various values, such as the constant 1 in GF(2^m) which is an m-bit variable with all bits set.

Our design generator enables users to choose appropriate encryption protocols for their applications. For example, a system can easily implement an encryption/decryption protocol using a particular elliptic curve. Users can store their private keys in the FPGA.

VIII. RESULTS AND EVALUATIONS

In this section, we compare the performance of various software and hardware implementations for point multiplication, which is the bottleneck for ECC systems. We have implemented the software design [45] on a dual-processor Intel Xeon 2.66 GHz (compiled for one processor) with 4 GB of RAM. We also compare our design with existing FPGA ECC cryptosystems over GF(2^m). The comparison for serial and parallel designs on different m and p values, where p refers to the degree of parallelization, is presented in Table II. Note that Place-and-Route (P&R) results mean the results that are obtained from the Celoxica DK3 and Xilinx ISE 6.2 tools, and measured results refer to the measured results from hardware realization. Our

TABLE III
COMPARISON BETWEEN OUR DESIGN AND THE PARALLEL REFERENCE DESIGNS [25]

Parallelism	Hardware (parallel) [25]	Our design		Hardware (parallel) [25]	Our design	
p	time(ms)	time(ms)	speedup	time(ms)	time(ms)	speedup
	$m = 113$			$m = 473$		
1	4.3	0.51	8.43	126.2	20.76	6.08
2	2.6	0.27	9.63	69.2	10.51	6.58
4	1.7	0.15	11.33	35.7	5.41	6.60
8	1.06	0.09	11.78	19.1	2.85	6.70
16	0.81	0.06	13.5	12.7	1.56	8.14
32	0.79	0.04	19.75	-	0.91	-
64	0.77	0.03	25.67	-	0.61	-
Average			14.3			6.82

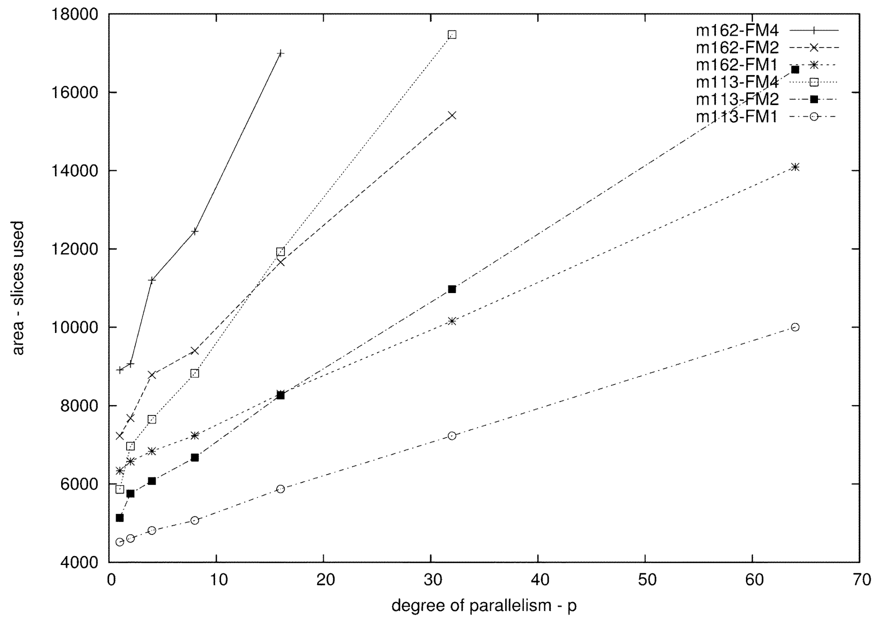


Fig. 7. Area usage for various p and m values.

hardware design has been implemented on an RC2000 board containing an XC2V6000 FPGA chip. We have also verified the correctness of each design using 300 000 consecutive point multiplications using data sent and received from the local PC. Note that from our results, for a given m, the change of p does not obviously change the critical path which shows that our designs are highly scalable and parameterisable. The "Speedup" column in Tables II, III, and VI shows the performance gain of our design over other methods. This gain is due to our highly parallel architecture which does not involve instruction fetch and decode.

As shown in Fig. 7, the area requirements for various designs such as $m = 113$, 162 using one, two, and four parallel field multipliers (1FM, 2FM and 4FM) show a linear growth of resource usage when the degree of parallelism p increases. It is notable to observe the first three points in the curve m162-FM4 which correspond to when $p = 1, 2, 4$. We can see that when $p = 4$, the circuit for the remainder of the parameterisable field multiplier causes additional resource usage. The runtime for performing one point multiplication is shown in Fig. 8 with respect to four different cases: m is not divisible by p using 1 FM, m is divisible by p using 1 FM, 2 FM and 4 FM. In order to test the effect of using different curve and base points in the point multiplication, we have also randomly picked a number of test cases and verified our designs.

TABLE IV
COMPARISON OF SPEED (NUMBER OF POINT MULTIPLICATION IN A SECOND) PER SLICES USED FOR DIFFERENT DESIGNS USING ONE, TWO, AND FOUR PARALLEL FIELD MULTIPLIERS

Designs	m = 113
Leong (parallel) [25]	0.15
FM1	0.77
FM2	0.71
FM4	0.70

TABLE V
COMPARISON OF CRITICAL PATHS FOR DIFFERENT FPGA PLATFORMS USING TWO ECC DESIGNS

Platforms	m = 53	m = 113
Xilinx Virtex : XCV1000-4	27.323ns	33.132ns
Xilinx Virtex : XCV1000-5	23.753ns	28.804ns
Xilinx Virtex : XCV1000-6	21.211ns	25.721ns
Xilinx Virtex-E : XCV2000E-8	18.092ns	23.441ns
Xilinx Virtex-2 : XC2V6000-4	16.139ns	21.404ns
Xilinx Virtex-4 : XC4VFX20-11	9.438ns	15.367ns

Moreover, we compare our results with another hardware implementation [25] for different p-values as shown in Table III. A larger speedup is achieved for the design with a smaller m value; we expect that this effect is due to the longer critical delay

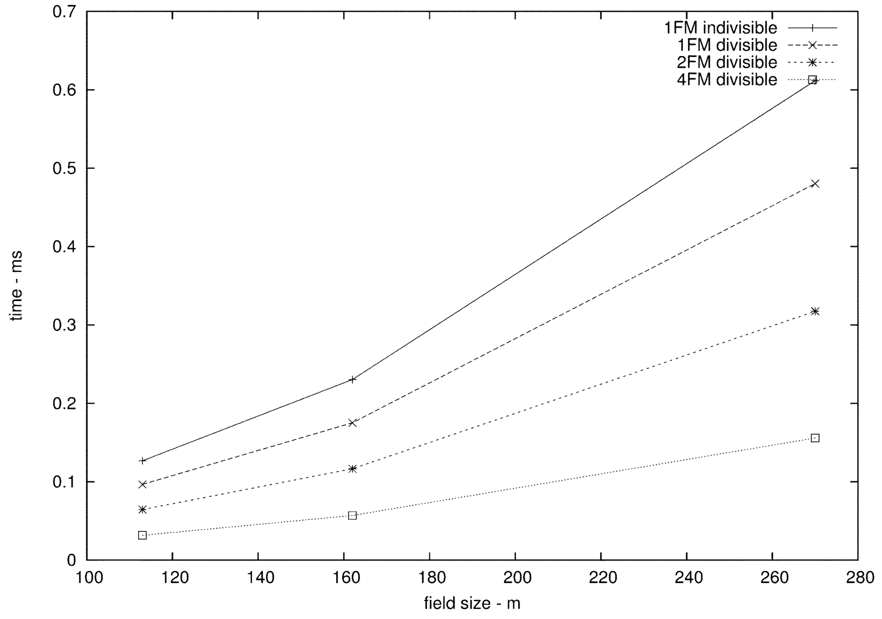


Fig. 8. Field multiplication comparison for various m values using one, two and four parallel field multipliers (divisible means m is divisible by p).

TABLE VI
COMPARISON BETWEEN OUR DESIGN AND OTHER EXISTING HARDWARE DESIGNS. PB STANDS FOR POLYNOMIAL BASIS AND ONB STANDS FOR OPTIMAL NORMAL BASIS. THE SYMBOL (*) DENOTES ESTIMATED RESULTS BASED ON THE CRITICAL PATH INFORMATION SHOWN IN TABLE V

Name	Year	Platform	Basis	m	Clock MHz	Time ms	FPGA Slices	Our Time (ms)	Speedup	Scaled* Time (ms)	Speedup
Agnew [2]	1989	ASIC	ONB	155	40.0	3.90	-	0.05	78.00	-	-
Rosner [46]	1998	XC4062	PB	168	16.3	4.47	956	0.06	74.50	-	-
Gao [14]	1999	XC4044XL	ONB	53	-	2.40	813	0.02	120.00	-	-
Okada [38]	2000	EPF10K	PB	163	3.0	80.30	-	0.05	1606.00	-	-
Okada [38]	2000	0.25-um ASIC	PB	163	66.0	1.10	-	0.05	22.00	-	-
Orlando [39]	2000	XCV400E	PB	167	76.7	0.21	1512	0.06	3.50	0.07	3.00
Goodman [15]	2000	ASIC	PB	160	50.0	7.00	-	0.05	140.00	-	-
Ernst [11]	2001	XC4085XLA	ONB	155	37.0	1.30	2346	0.05	26.00	-	-
Smart (Hessian form) [48]	2001	XC4000XL	PB	191	-	11.82	-	0.08	147.75	-	-
Leong (serial) [25]	2002	XCV1000	ONB	173	28.0	11.10	2148	0.07	158.57	0.12	92.50
Leong (parallel) [25]	2002	XCV1000	ONB	113	31.0	0.75	8753	0.03	25.00	0.05	15.00
Gura [16], [17]	2002	XCV2000E	PB	163	66.4	0.14	15768	0.05	2.80	0.055	2.55
Ernst [12]	2002	Atmel AT94K40	PB	113	12.0	1.40	-	0.03	46.67	-	-
Jung [22]	2002	Atmel AT94K40	PB	128	12.0	0.15	-	0.04	3.75	-	-
Kerins [24]	2002	XCV2000	PB	176	40.0	6.90	-	0.07	98.57	0.12	57.50
Bednara (LFSR) [5]	2002	XCV1000	PB	191	50.0	2.27	-	0.08	28.38	0.13	17.46
Bednara (parallel) [5]	2002	XCV1000	PB	191	50.0	0.27	-	0.08	3.38	0.13	2.08
Nguyen [36]	2003	XC2V6000	PB	233	100	3.35	-	0.12	27.91	0.12	27.91
Satoh [47]	2003	0.13-um ASIC	PB	160	510.2	0.19	-	0.05	3.80	-	-
Lutz (Koblitz curve) [28]	2004	XCV2000E	PB	163	66.0	0.075	-	0.05	1.50	0.055	1.36
Mentens [33]	2004	XCV800	PB	160	47.0	3.810	-	0.05	76.20	0.08	47.63

path in the design with a larger m value, which can be further optimized.

Additionally, we compare our designs with previous work in terms of speed per unit area (the number of point multiplication per second divided by the number of hardware slices used) as shown in Table IV. It shows that our approach is more efficient than designs involving custom instruction processors [25], even when area is taken into account. We also note that small and fast are the two major goals of cryptographic designs. For example, Orlando *et al.* [39] and Gura *et al.* [17] have about the same speed, and [39] is about ten times smaller.

We also compare our design with other existing hardware cryptosystems. Table VI shows the performance improvement by using our design (with the maximum implementable value for p) for the m-values that have been published. We adopt maximum parallelism since a large speed improvement can be obtained by a small area increase. Since the comparison of different ECC implementations on different FPGAs is not possible, we implement our designs on different FPGA architecture as a reference comparison. As shown in Table V, the timing values for different FPGA architectures have been compared by using two ECC designs. It shows how the advance in process technology brings additional performance gain. For instance, the latest Virtex-4 architecture can even speedup our m113 design by 30%. The last two columns in Table VI show the scaled timings and their speedup when the designs are mapped using the previous process technology.

We find that area varies approximately linearly with m and p. Given $p = 2$, for $m = 113$ the design requires 6961 FPGA slices whereas for $m = 270$, it requires 14 787 slices. So increasing the key by 2.4 times (or increasing the security level radix by $\sqrt{2^{270} - 2^{113}}$), the design needs 2.1 times as many slices.

IX. Concluding Remarks

A customizable pipelined and parallelised ECC design for various field operations has been proposed. This design supports various parameters, such as the key size and the degree of parallelism, to enable tradeoff between level of security, design size and speed. A design generator has been developed to facilitate fast implementation. Performance analysis shows that our design at 35 MHz is the fastest amongst existing hardware implementations. It can compute a point multiplication up to 1150 times faster than a software ECC application on a Xeon 2.66-GHz computer. On-going and future work includes functional extensions and optimizations such as speed improvement, resource minimization, and run-time customization of ECC designs.

Acknowledgment

The authors thank the anonymous reviewers, and also P. Leong, D.-U. Lee, and A. A. Gaffar for their useful remarks and suggestions.

References

[1] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "A fast elliptic curve cryptosystem," in *Adv. in Cryptology: EUROCRYPT'89*, LNCS 434, 1989, pp. 706–708.

[2] ——, "An implementation of elliptic curve cryptosystem over $F_{2^{155}}$," *IEEE J. Select. Areas Commun.*, vol. 11, pp. 804–813, 1993.

[3] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle, "Hardware architectures for public key cryptography," *Integration, the VLSI J.*, vol. 34, pp. 1–64, 2003.

[4] L. Batina, G Bruin-Muurling, and S. B. Örs, "Flexible hardware design for RSA and elliptic curve cryptosystems," in *Proc. Cryptographers Track—RSA Conf. (CT-RSA)*, 2004, pp. 250–263.

[5] M. Bednara, M. Daldrup, J. Gathen, J. Shokrollahi, and J. Teich, "Reconfigurable implementation of elliptic curve crypto algorithms," in *Proc. Reconfigurable Architectures Workshop*, 2002.

[6] M. Bednara, M. Daldrup, J. Teich, J. Gathen, and J. Shokrollahi, "Tradeoff analysis of FPGA based elliptic curve cryptography," in *Proc. IEEE Int. Symp. on Circuits and Systems*, vol. V, 2002, pp. 797–800.

[7] I. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*, ser. London Math. Soc. Lecture Note Series: Cambridge Univ. Press, 1999.

[8] M. Brown, D. Hankerson, J. López, and A. Menezes, "Software implementation of the NIST elliptic curves over prime fields," in *Proc. CT-RSA*, LNCS 2020, 2001, pp. 250–265.

[9] (2002). Certicom. [Online]Available: http://www.certicom.com

[10] "Handel-C Language Reference Manual for DK2.0," Celoxica, 2003.

[11] M. Ernst, S. Klupsch, O. Hauck, and S. A. Huss, "Rapid prototyping for hardware accelerated elliptic curve public-key cryptosystems," in *Proc. Workshop on Rapid System Prototyping*, 2001, pp. 24–29.

[12] M. Ernst, M. Jung, F. Madlener, S. Huss, and R. Blümel, "A reconfigurable system on chip implementation for elliptic curve cryptography over $\mathrm{GF}(2^n)$," *Cryptographic Hardware and Embedded Systems*, pp. 381–399, 2002.

[13] K. Gaj, S. Bajracharya, C. Shu, S. Han, and T. El-Ghazawi, "Elliptic curve cryptography over $\mathrm{GF}(2^m)$ on a reconfigurable computer: Polynomial basis vs. Optimal normal basis representation comparative study," in *Military and Aerospace Applications of Programmable Devices and Technologies Conf.*, 2004.

[14] L. Gao, S. Shrivastava, and G. E. Sobelman, "Elliptic curve scalar multiplier design using FPGAs," *Proc. Cryptographic Hardware and Embedded Systems*, pp. 257–268, 1999.

[15] J. Goodman and A. Chandrakasan, "An energy efficient reconfigurable public-key cryptography processor architecture," *Proc. Cryptographic Hardware and Embedded Systems*, pp. 175–190, 2000.

[16] N. Gura, H. Eberle, and S. C. Shantz, "Generic implementations of elliptic curve cryptography using partial reduction," *Proc.. ACM Computer and Communications Security*, pp. 108–116, 2002.

[17] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila, "An end-to-End systems approach to elliptic curve cryptography," *Proc. Cryptographic Hardware and Embedded Systems*, pp. 349–365, 2002.

[18] D. Hankerson, J. Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," *Cryptographic Hardware and Embedded Syst.*, pp. 1–24, 2000.

[19] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography* New York, 2004.

[20] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $\mathrm{GF}(2^m)$ using normal bases," *Inf. Comput.*, vol. 78, no. 3, pp. 171–177, 1998.

[21] S. Janssens, J. Thomas, W. Borremans, P. Gijsels, I. Verbauwhede, F. Vercauteren, B. Preneel, and J. Vandewalle, "Hardware/software co-design of an elliptic curve public-key cryptosystem," in *IEEE Workshop on Signal Processing Systems (SiPS)*, 2001, pp. 209–216.

[22] M. Jung, F. Madlener, E. Ernst, and S. A. Huss, "A reconfigurable coprocessor for finite field multiplication in $\mathrm{GF}(2^m)$," in *Proc. IEEE Workshop on Heterogeneous Reconfigurable System on Chip*, 2002.

[23] N. Koblitz, A. Menezes, and S. A. Vanstone, "The state of elliptic curve cryptography," *Designs, Codes and Cryptography*, vol. 19, pp. 173–193, 2000.

[24] T. Kerins, E. Popovici, W. Marnane, and P. Fitzpatrick, "Fully parameterizable elliptic curve cryptography processor over $\mathrm{GF}(2^m)$," *Proc. Field-Programmable Logic and Applications*, pp. 750–759, 2002.

[25] P. H. W. Leong and K. H. Leung, "A microcoded elliptic curve processor using FPGA technology," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, pp. 550–559, 2002.

[26] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. Cambridge, U.K.: Cambridge Univ. Press, 1994.

[27] J. López and R. Dahab, "Fast multiplication on elliptic curves over $\mathrm{GF}(2^m)$ without precomputation," *Cryptographic Hardware and Embedded Syst.*, pp. 316–327, 1999.

[28] J. Lutz and A. Hasan, "High performance FPGA based elliptic curve cryptographic co-processor," in *Proc. IEEE Conf. on Information Technology: Coding and Computing*, 2004, pp. 486–492.

[29] E. D. Mastrovito, "VLSI Architectures for Computation in Galois Fields," Ph.D. dissertation, Linköping Univ., 1991.

[30] J. Massey and J. Omura, "Computational Method and Apparatus for Finite Field Arithmetic," U.S. Patent 4 587 627, 1986.

[31] A. J. Menezes, *Applications of Finite Fields*: Kluwer Academic, 1993.

[32] ——, *Elliptic Curve Public Key Cryptosystems*: Kluwer Academic, 1993.

[33] N. Mentens, S. B. Örs, and B. Preneel, "An FPGA implementation of an elliptic curve processor over $\mathrm{GF}(2^m)$," *Proc. GVLSI*, pp. 454–457, 2004.

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

[35] P. Ning and Y. L. Yin, "Efficient software implementation for finite field multiplication in normal basis," in *Proc. Conf. on Information and Communications Security*, LNCS 2229, 2001, pp. 177–188.

[36] N. Nguyen, K. Gaj, D. Caliga, and T. El-Ghazawi, "Implementation of elliptic curve cryptosystems on a reconfigurable computer," *Proc. IEEE Field Programmable Technology*, pp. 60–67, 2003.

[37] FIPS 186-2, Digital Signature Standard (DSS) (2000, Feb.). *Available: http://csrc.nist.gov/encryption/* [Online]

[38] S. Okada, N. Torii, K. Itoh, and M. Takenaka, "Implementation of elliptic cruve cryptographic coprocessor over $\mathrm{GF}(2^m)$ on an FPGA," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, LNCS 1965, 2000, pp. 25–40.

[39] G. Orlando and C. Paar, "A high performance reconfigurable elliptic curve for $\mathrm{GF}(2^m)$," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, LNCS 1965, 2000, pp. 41–56.

[40] ——, "A scalable $\mathrm{GF}(p)$ elliptic curve processor architecture for programmable hardware," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, LNCS 2162, 2003, pp. 356–371.

[41] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of elliptic curve processor over $\mathrm{GF}(p)$," in *Proceedings of the IEEE Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2003, pp. 433–443.

[42] *Standard Specifications for Public Key Cryptography*, IEEE P1363, 2000.

[43] H. Pietiläinen, "Elliptic Curve Cryptography on Smart Cards," M.Sc., Helsinki Univ. of Technology, 2000.

[44] F. Rodríguez-Henríquez and Ç. K. Koç, "Parallel multipliers based on special irreducible pentanomials," *IEEE Trans. Computers*, vol. 52, pp. 1535–1542, 2003.

[45] M. Rosing, *Implementing Elliptic Curve Cryptography*: Manning, 1999.

[46] M. Rosner, "Elliptic Curve Cryptosystems on Reconfigurable Hardware," M.Sc., Worcester Polytechnic Inst., 1998.

[47] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Computers*, vol. 52, pp. 449–460, 2003.

[48] N. P. Smart, "The Hessian form of an elliptic curve," in *Proc. Workshop on Cryptographic Hardware and Embedded Systems*, LNCS 2162, 2001, pp. 118–125.

[49] B. Sunar and Ç.K. Koç, "An efficient optimal normal basis Type II multiplier," *IEEE Trans. Computers*, vol. 50, no. 1, pp. 83–87, 2001.

[50] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in $\mathrm{GF}(2^m)$," *IEEE Trans. Computers*, vol. 34, pp. 709–717, 1985.

[51] T. Wollinger, J. Guajardo, and C. Paar, "Cryptography in embedded systems: An overview," in *Proc. Embedded World Exhibition and Conf.*, 2003, pp. 735–744.

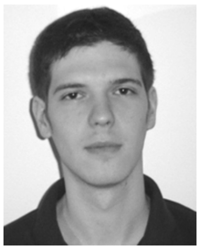
[52] A. Woodbury, "Efficient Algorithms for Elliptic Curve Cryptosystems on Embedded Systems," M.Sc., Worcester Polytechnic Inst., 2001.

[53] A. Woodbury, D. V. Bailey, and C. Paar, "Elliptic curve cryptography on smart cards without coprocessors," in *Smart Card Research and Advanced Application Conf.*, 2000.

Ray C. C. Cheung (S'05) received the B.Eng. degree in Computer Engineering, and the and M.Phil. degree in Computer Science & Engineering from the Chinese University of Hong Kong (CUHK) in 1999 and 2001, respectively. He is currently working toward the Ph.D. degree in the Custom Computing group, Department of Computing, Imperial College London, London, U.K.

In 2001, he worked as a system administrator in the Center of Large-Scale Computation (CLC) of Cluster Technology, Hong Kong. From January 2002 to December 2003, he was an instructor of the Department of Computer Science & Engineering, CUHK. His current research interests include cryptographic hardware designs and design exploration of System-on-Chip (SoC) designs and embedded systems.

Mr. Cheung is a student member of the ACM, and is the newsletter and web editor of the SIGDA UK chapter.



Nicolas Jean-baptiste Telle graduated from the French "GrandeÉcole" Supélec in 2003. In the same year, he received the M.Sc. degree in advanced computing from Imperial College London with distinction, and was awarded the best project prize for his work on optimizing Elliptic Curve Cryptosystems using hardware designs and reconfigurable computing.

After working for a year and a half for a major consulting services company in their business intelligence department on massively parallelized large scale databases, he is now a freelance consultant still in the business intelligence domain.

Wayne Luk (M'85) received the M.A., M.Sc., and D.Phil. degrees in engineering and computing science from the University of Oxford, Oxford, U.K.

He is a Member of the Academic Staff in the Department of Computing, Imperial College London, University of London, U.K. His research interests include theory and practice of customising hardware and software for specific application domains such as graphics and image processing, multimedia, and communications. His current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays.

Peter Y. K. Cheung (M'85–SM'04) graduated from Imperial College of Science and Technology, University of London,in 1973, with first class honors and was awarded the IEE prize.

After working at Hewlett Packard for a few years, he returned to Imperial College as a lecturer in 1980. He runs an active research group in digital design, attracting support from many industrial partners. He was elected as one of the first Imperial College Teaching Fellows in 1994 in recognition of his innovation in teaching. He is currently a professor of digital systems and deputy head of the Department of Electrical & Electronic Engineering at Imperial College. His research interests include VLSI architectures for signal processing, asynchronous systems, reconfigurable computing using FPGA, and architectural synthesis.