Mobile Netw Appl (2007) 12:259–270
DOI 10.1007/s11036-007-0022-4

A State-of-the-art Elliptic Curve Cryptographic Processor Operating in the Frequency Domain

Selçuk Baktır · Sandeep Kumar · Christof Paar · Berk Sunar

Published online: 30 September 2007

Abstract We propose a novel area/time efficient elliptic curve cryptography (ECC) processor architecture which performs all finite field arithmetic operations in the discrete Fourier domain. The proposed architecture utilizes a class of *optimal extension fields (OEF)* $GF(q^m)$ where the field characteristic is a Mersenne prime $q = 2^n - 1$ and $m = n$. The main advantage of our architecture is that it achieves extension field modular multiplication in the *discrete Fourier domain* with only a *linear* number of base field $GF(q)$ multiplications in addition to a quadratic number of simpler operations such as addition and bitwise rotation. We achieve an area between 25k and 50k equivalent gates for the implementations over OEFs of size 169, 289 and 361 bits. With its low area and high speed, the proposed architecture is well suited for ECC in small device environments such as sensor networks. The work at hand presents the first hardware implementation of a frequency domain multiplier suitable for ECC and the first hardware implementation of ECC in the frequency domain.

Keywords elliptic curve cryptography (ECC) · finite fields · modular multiplication · discrete Fourier domain

1 Introduction

Elliptic curve cryptosystems [9, 13] are favorable choices for asymmetric data encryption compared to other popular algorithms such as RSA [17] mainly due to their requirement for smaller key sizes. The same level of security provided by a 1024-bit key in RSA can be achieved with only a 160-bit key in elliptic curve cryptography (ECC). The key size determines the size of the operands over which finite field arithmetic operations are performed and consequently the efficiency of the cryptosystem [6, 12]. A comprehensive overview for hardware implementations of RSA and ECC are provided in Batina et al. [5].

Efficiency of an elliptic curve cryptosystem is highly dependent on the underlying finite field arithmetic. Multiplication in $GF(q^m)$ can be achieved with a quadratic number of multiplications and additions in the base field $GF(q)$ using the classical polynomial multiplication method. Using the Karatsuba algorithm, this complexity can be reduced significantly, however one still needs to do a subquadratic number of multiplications and additions in the base field $GF(q)$. The multiplication operation is inherently much more

Selçuk Baktır conducted his work in part while he was a visiting researcher at the Communication Security Group at Ruhr-University Bochum, Germany.

Selçuk Baktır and Berk Sunar were supported by the NSF CAREER award ANI-0133297.

S. Baktır (✉) · B. Sunar
Cryptography & Information Security Laboratory,
WPI, Worcester, MA, USA
e-mail: selcuk@wpi.edu

B. Sunar
e-mail: sunar@wpi.edu

S. Kumar · C. Paar
Communication Security Group (COSY),
Ruhr-University Bochum, Bochum, Germany
e-mail: Sandeep.Kumar@crypto.ruhr-uni-bochum.de

C. Paar
e-mail: cpaar@crypto.ruhr-uni-bochum.de

complex than other operations such as addition, therefore it is desirable that one performs as small a number of base field multiplications as possible for achieving an extension field multiplication. *Discrete Fourier transform (DFT) modular multiplication* [3, 4] achieves multiplication in $GF(q^m)$ in the frequency domain with only a linear number of base field $GF(q)$ multiplications in addition to a quadratic number of simpler base field operations such as additions/subtractions and bitwise rotations. An earlier approach for hardware implementation of large integer multiplication in the discrete Fourier domain was proposed in Kalach and David [8]. Although no specific implementation results in terms of timing performance or circuit area are provided in the paper, the authors present analytical results which show that their proposed hardware multiplier architecture is more efficient than multiplication with the classical method for operand sizes of 4096 bits or longer. With our work, we prove with our hardware implementation results that by using the DFT modular multiplication algorithm one can achieve efficient multiplication in the discrete Fourier domain for much smaller operand sizes, e.g. as small as 160 bits, relevant to ECC.

In an ECC processor the multiplier unit usually consumes the most area on the chip, therefore it is crucial that one uses an area/time efficient multiplier, particularly in constrained environments, such as smart cards, wireless sensor nodes or radio frequency identification tags, where resources are precious. In this work we address this issue by proposing an area/time efficient ECC processor architecture utilizing DFT modular multiplication in optimal extension fields (OEF) [1, 2] with the Mersenne prime field characteristic $q = 2^n - 1$ and the extension degree $m = n$. Our ECC processor architecture utilizes an improved and hardware-optimized version of the DFT modular multiplication algorithm originally proposed in Baktır and Sunar [3, 4] and requires an area ranging between 25k to 50k equivalent gates for implementations over OEFs of size 169, 289 and 361 bits.

In Section 2, we provide some background information on OEFs and multiplication in the frequency domain. In Section 3, we overview the DFT modular multiplication algorithm. We then present some optimization ideas for efficient implementation of this algorithm in hardware. In Section 4, we present an efficient elliptic curve cryptographic processor design which utilizes an *optimized DFT modular multiplier architecture* over $GF((2^{13}-1)^{13})$, $GF((2^{17}-1)^{17})$ and $GF((2^{19}-1)^{19})$ for an ASIC implementation using AMI Semiconductor $0.35\mu m$ CMOS technology. Finally, in Section 5 we present our implementation results.

2 Mathematical background

2.1 Optimal extension fields

An extension field $GF(q^m)$ is generated by using an m^{th} degree polynomial irreducible over $GF(q)$ and comprises the residue classes modulo the irreducible field generating polynomial. OEFs are a special class of finite extension fields which use a field generating polynomial of the form $P(x) = x^m - w$ and have a *pseudo-Mersenne prime* field characteristic given in the form $q = 2^n \pm c$ with $\log_2 c < \lfloor \frac{n}{2} \rfloor$. In OEFs the pseudo-Mersenne prime field characteristic allows efficient reduction in the base field $GF(q)$ operations and the binary field generating polynomial allows for efficient reduction in the extension field. OEFs are found to be successful in ECC implementations where resources such as computational power and memory are constrained [10, 20]. In OEFs, the standard basis is utilized for representing finite field elements. In this representation, the elements of $GF(q^m)$ are represented by polynomials of degree $m-1$ with coefficients in $GF(q)$. For instance, an element $A \in GF(q^m)$ is represented as

$$A = \sum_{i=0}^{m-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1}$$

where $a_i \in GF(q)$. Multiplication in OEFs is performed as follows.

Multiplication

For $A, B \in GF(q^m)$, the product $C = A \cdot B$ is computed in two steps:

1) Polynomial multiplication:

$$C' = A \cdot B = \sum_{i=0}^{2m-2} c'_i x^i$$

2) Modular reduction:

$$C = C' \bmod P(x)$$

In this paper we propose an efficient hardware architecture which performs the above modular multiplication operation in the frequency domain. For this, we need to first represent the operands in the frequency domain. To convert an element in $GF(q^m)$ into the frequency domain representation, the *number theoretic transform* is used, which is explained next.

2.2 Number theoretic transform

The number theoretic transform over a ring, also known as *the DFT over a finite field*, was introduced by

Pollard [15]. For a finite field $GF(q)$ and a sequence (a) of length d whose entries are from $GF(q)$, the forward DFT of (a) over $GF(q)$, denoted by (A), can be computed as

$$A_j = \sum_{i=0}^{d-1} a_i r^{ij} \ , \ 0 \le j \le d-1 \ . \tag{1}$$

Here we refer to the elements of (a) and (A) by a_i and A_i, respectively, for $0 \le i \le d-1$. Likewise, the inverse DFT of (A) over $GF(q)$ can be computed as

$$a_i = \frac{1}{d} \cdot \sum_{j=0}^{d-1} A_j r^{-ij} \ , \ 0 \le i \le d-1 \ . \tag{2}$$

We will refer to the sequences (a) and (A) as the *time and frequency domain representations*, respectively, of the same sequence. The above DFT computations over the finite field $GF(q)$ are defined by utilizing a d^{th} primitive root of unity, denoted by r, from $GF(q)$ or a finite extension of $GF(q)$. In this work we will use $r = -2 \in GF(q)$ as it enables efficient implementation in hardware which will be further discussed in Section 3.3. We will consider only finite fields $GF(q^m)$ with a Mersenne prime characteristic $q = 2^n - 1$ and odd extension degree $m = n$. This makes the sequence length $d = 2m$, since $r = -2$ is a $(2m)^{th}$ primitive root of unity in the base field $GF(2^n - 1)$. In this case, when $r = -2$ and $q = 2^n - 1$, a modular multiplication in $GF(q)$ with a power of r can be achieved very efficiently with a simple bitwise rotation in addition to a negation if the power is odd. The DFT computed modulo a Mersenne prime, as in our case, is called the *Mersenne transform* [16].

2.3 Convolution theorem and polynomial multiplication in the frequency domain

A significant application of the Fourier transform is convolution. Convolution of two d-element sequences (a) and (b) in the time domain results in another d-element sequence (c) and can be computed as follows:

$$c_i = \sum_{j=0}^{d-1} a_j b_{i-j \bmod d} \ , \ \ 0 \le i \le d-1 \ . \tag{3}$$

According to the convolution theorem, the above convolution operation in the time domain is equivalent to the following computation in the frequency domain:

$$C_i = A_i \cdot B_i \ , \ \ 0 \le i \le d-1 \ , \tag{4}$$

where (A), (B) and (C) denote the DFTs of (a), (b) and (c), respectively. Hence, convolution of two d-element sequences in the time domain, with complexity $O(d^2)$, is equivalent to simple pairwise multiplication of the DFTs of these sequences and has a surprisingly low $O(d)$ complexity. The DFT and its applications are described in detail in Tolimieri et al. [19] and Burrus and Parks [7].

Note that the summation in (3) is the *cyclic convolution* of the sequences (a) and (b). We have seen that this cyclic convolution can be computed very efficiently in the Fourier domain by pairwise coefficient multiplications. Multiplication of two polynomials on the other hand is equivalent to the *acyclic (linear) convolution* of the polynomial coefficients. However, if we represent elements of $GF(q^m)$, which are $(m-1)^{st}$ degree polynomials with coefficients in $GF(q)$, with at least $d = (2m-1)$ element sequences by appending zeros at the end, then the cyclic convolution of two such sequences will be equivalent to their acyclic convolution and hence give us their polynomial multiplication.

One can form sequences by taking the ordered coefficients of polynomials. For instance,

$$a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1} \ ,$$

an element of $GF(q^m)$ in polynomial representation, can be interpreted as the following sequence after appending $d - m$ zeros to the right:

$$(a) = (a_0, a_1, a_2, \ldots, a_{m-1}, 0, 0, \ldots, 0) \ . \tag{5}$$

For $a(x), b(x) \in GF(q^m)$, and for $d \ge 2m-1$, the cyclic convolution of (a) and (b) yields a sequence (c) whose first $2m-1$ entries can be interpreted as the coefficients of a polynomial $c(x)$ such that $c(x) = a(x) \cdot b(x)$. The computation of this cyclic convolution can be performed by simple pairwise coefficient multiplications in the discrete Fourier domain.

Note that, using the convolution property the polynomial product $c(x) = a(x) \cdot b(x)$ can be computed very efficiently in the frequency domain but the final reduction by the field generating polynomial is not performed. For further multiplications to be performed on the product $c(x)$ in the frequency domain, it needs to be first reduced modulo the field generating polynomial. The DFT modular multiplication algorithm, which will be mentioned briefly in the following section, performs both polynomial multiplication and modular reduction in the frequency domain and thus makes it possible to perform consecutive modular multiplications in the frequency domain.

3 Modular multiplication in the frequency domain

To the best of our knowledge, the *DFT modular multiplication* algorithm [3, 4], which performs Montgomery multiplication in $GF(q^m)$ in the frequency domain, is

the only existing frequency domain multiplication algorithm to achieve efficient modular multiplication for operand sizes relevant to ECC. In this section, we give a brief overview of this algorithm and the notation used.

3.1 Mathematical notation

Since the DFT modular multiplication algorithm runs in the frequency domain, the parameters used in the algorithm are in their frequency domain sequence representations. These parameters are the input operands $a(x), b(x) \in GF(q^m)$, the result $c(x) = a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod f(x) \in GF(q^m)$, irreducible field generating polynomial $f(x)$, normalized irreducible field generating polynomial $f'(x) = f(x)/f(0)$, and the indeterminate x. The time domain sequence representations of these parameters are $(a), (b), (c), (f)$, (f') and (x), respectively, and their frequency domain sequence representations, i.e. the DFTs of the time domain sequence representations, are $(A), (B), (C)$, $(F), (F')$ and (X). We will denote elements of a sequence with the name of the sequence and a subscript for showing the location of the particular element in the sequence, e.g. for the indeterminate x represented as the following d-element sequence in the time domain

$$(x) = (0, 1, 0, 0, \cdots, 0),$$

the DFT of (x) is computed as the following d-element sequence

$$(X) = (1, r, r^2, r^3, r^4, r^5, \ldots, r^{d-1}),$$

whose first and last elements are denoted as $X_0 = 1$ and $X_{d-1} = r^{d-1}$, respectively.

3.2 DFT modular multiplication algorithm

DFT Modular Multiplication shown in Algorithm 1, consists of two parts: Multiplication (Steps 1 to 3) and Montgomery reduction (Steps 4 through 13). Multiplication is performed simply by pairwise multiplication of the two input sequences (A) and (B). This multiplication results in (C) which corresponds to $c(x) = a(x) \cdot b(x)$, a polynomial of degree at most $2m - 2$. For performing further multiplications over $c(x)$ using the same method in the *frequency domain*, one needs to first reduce it modulo $f(x)$ so that its time domain representation is of degree at most $m - 1$.

The DFT modular multiplication algorithm performs reduction in the frequency domain by *DFT Montgomery reduction* (Steps 4 to 13). The input to

Algorithm 1 DFT modular multiplication algorithm for $GF(q^m)$

Input: $(A) \equiv a(x) \in GF(q^m)$, $(B) \equiv b(x) \in GF(q^m)$
Output: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod f(x) \in GF(q^m)$

1: **for** $i = 0$ to $d - 1$ **do**
2: $\quad C_i \leftarrow A_i \cdot B_i$
3: **end for**
4: **for** $j = 0$ to $m - 2$ **do**
5: $\quad S \leftarrow 0$
6: $\quad$ **for** $i = 0$ to $d - 1$ **do**
7: $\quad\quad S \leftarrow S + C_i$
8: $\quad$ **end for**
9: $\quad S \leftarrow -S/d$
10: $\quad$ **for** $i = 0$ to $d - 1$ **do**
11: $\quad\quad C_i \leftarrow (C_i + F'_i \cdot S) \cdot X_i^{-1}$
12: $\quad$ **end for**
13: **end for**
14: Return (C)

DFT Montgomery reduction is the frequency domain sequence representation (C) of $c(x) = a(x) \cdot b(x)$ and its output is the sequence (C) corresponding to $c(x) = a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod f(x) \in GF(q^m)$. DFT Montgomery reduction is a direct adaptation of Montgomery reduction. In the frequency domain, the value S is computed such that $(c(x) + S \cdot (f'(x)))$ is a multiple of x. Note that $(c(x) + S \cdot (f'(x)))$ is still equivalent to $c(x) \bmod f(x)$. The algorithm then divides $(c(x) + S \cdot f'(x))$ by x and obtains a result which is congruent to $c(x) \cdot x^{-1} \bmod f(x)$. By repeating this $m - 1$ times (Steps 4 to 13) the initial input which is the $(2m - 2)^{nd}$ degree input polynomial $c(x) = a(x) \cdot b(x)$ is reduced to the final $(m - 1)^{st}$ degree result which is congruent to $a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod f(x)$. Hence, for the inputs $a(x) \cdot x^{m-1}$ and $b(x) \cdot x^{m-1}$, both in $GF(q^m)$, the DFT modular multiplication algorithm computes $a(x) \cdot b(x) \cdot x^{m-1} \in GF(q^m)$ and thus the Montgomery residue representation is kept intact and further computations can be performed in the frequency domain using the same algorithm.

3.3 Optimization

In this work, we show that for the case of $r = -2$, odd m and $n = m$, i.e. when the bit length of the field characteristic $q = 2^n - 1$ is equal to the field extension degree, DFT modular multiplication can be optimized by precomputing some intermediary values in the algorithm. Our optimization takes advantage of the fact that when $r = -2$, $q = 2^n - 1$, the field generating polynomial is

Algorithm 2 Optimized DFT modular multiplication in $GF(q^m)$ for $r = -2$, $q = 2^n - 1$, m odd, $m = n$ and $f(x) = x^m - 2$

Input: $(A) \equiv a(x) \in GF(q^m)$, $(B) \equiv b(x) \in GF(q^m)$
Output: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod f(x) \in GF(q^m)$

1: **for** $i = 0$ to $d - 1$ **do**
2: $C_i \leftarrow A_i \cdot B_i$
3: **end for**
4: **for** $j = 0$ to $m - 2$ **do**
5: $S \leftarrow 0$
6: **for** $i = 0$ to $d - 1$ **do**
7: $S \leftarrow S + C_i$
8: **end for**
9: $S \leftarrow -S/d$
10: $S_{half} \leftarrow S/2$
11: $S_{even} \leftarrow S_{half}$
12: $S_{odd} \leftarrow S + S_{half}$
13: **for** $i = 0$ to $d - 1$ **do**
14: **if** $i \bmod 2 = 0$ **then**
15: $C_i \leftarrow C_i + S_{even}$
16: **else**
17: $C_i \leftarrow -(C_i + S_{odd})$
18: **end if**
19: $C_i \leftarrow C_i/2^i$
20: **end for**
21: **end for**
22: Return (C)

$f(x) = x^m - 2$ and hence $f'(x) = -\frac{1}{2} \cdot x^m + 1$, m is odd and $m = n$, the following equalities hold in $GF(q)$:

$$F'_i = -\frac{1}{2} \cdot (-2)^{mi} + 1 = \begin{cases} -\frac{1}{2} + 1 = \frac{1}{2}\,, & i \text{ even} \\ \frac{1}{2} + 1\,, & i \text{ odd} \end{cases}$$

This equality holds since

$$(-2)^{mi} \equiv (-2)^{ni} \equiv (-1)^{ni}(2^n)^i \equiv (-1)^{ni} \pmod{q}.$$

Note that in this case F'_i has only two distinct values, namely $-\frac{1}{2} + 1 = \frac{1}{2}$ and $\frac{1}{2} + 1$ for the irreducible field generating polynomial $f(x) = x^m - 2$. Hence, $F'_i \cdot S$ in Step 11 of Algorithm 1 can attain only two values for any distinct value of S and these values can be precomputed outside the loop avoiding all such computations inside the loop. The precomputations can be achieved very efficiently with only one bitwise rotation and one addition. With the suggested optimization, both the number of base field additions/subtractions and the number of base field bitwise rotations required to perform an extension field multiplication are reduced by $d(m-1) = 2m(m-1)$.

Table 1 List of parameters suitable for optimized DFT modular multiplication

n	$q = 2^n - 1$	m	d	r	Equivalent binary field size
13	8, 191	13	26	-2	$\sim 2^{169}$
17	131, 071	17	34	-2	$\sim 2^{289}$
19	524, 287	19	38	-2	$\sim 2^{361}$

In Algorithm 2, we present the optimized algorithm for the irreducible polynomial $f(x) = x^m - 2$. In Table 1, we suggest a list of parameters for implementation of the optimized algorithm over finite fields of different sizes. Note that these parameters are perfectly suited for ECC. In Section 5, we provide the implementation results for all the finite fields listed in Table 1 and thus show the relevance of the optimized algorithm for area/time efficient hardware implementation of ECC in constrained environments.

4 Implementation of an ECC processor utilizing DFT modular multiplication

In this section we present a hardware implementation of an ECC processor using the DFT modular multiplication algorithm. The DFT modular multiplication algorithm trades off computationally expensive modular multiplication operations for simple bitwise rotations which can be achieved practically for free in hardware by proper rewiring. We exemplarily use the field $GF((2^{13} - 1)^{13})$ to explain our design, although the design is easily extendable for the other parameter sizes mentioned in Table 1 and the implementation results for all the parameter sizes are given in Section 5.

We first describe the implementation of the base field arithmetic in $GF(2^{13} - 1)$ and then make parameter decisions based on Algorithm 2 to implement an efficient DFT modular multiplier. Then we present the overall processor design to compute the ECC scalar point multiplication operation.

4.1 Base field arithmetic

Base field arithmetic consists of addition, subtraction (addition with a negation) and multiplication in $GF(2^{13} - 1)$. The arithmetic architectures are designed to ensure area/time efficiency.

Base Field Addition
Addition in the base field is implemented using a ripple carry adder. Reduction with the Mersenne prime $q = 2^{13} - 1$ is just an extra addition of the carry generated.

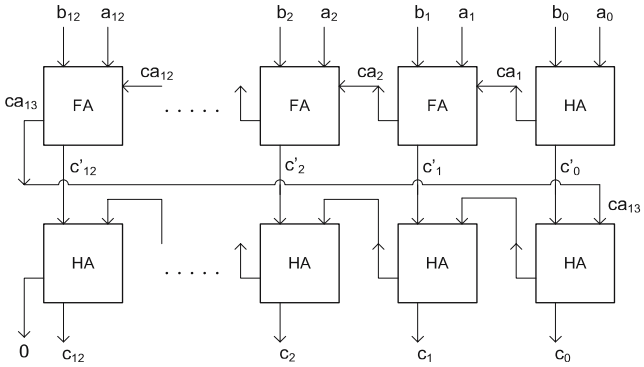

Fig. 1 Base field addition architecture

This additional addition is always hard wired, independent of the value of the carry, to avoid any timing related attacks. Figure 1 shows the design of the base field adder built using half adders and full adders.

Base Field Negation

Negation in the base field is extremely simple to implement with a Mersenne prime q as the field characteristic. Negation of $B \in GF(q)$ is normally computed as $B' = q - B$. However, when q is a Mersenne prime, it is easy to see from the binary representation of $q = 2^{13} - 1$ (which is all 1's) that this subtraction is equivalent to flipping (NOT) of the bits of B. Hence, subtraction in this architecture can be implemented by using the adder architecture with an additional bitwise NOT operation on the subtrahend.

Base Field Multiplication

Base field multiplication is a 13×13-bit integer multiplication followed by a modular reduction with $q = 2^{13} - 1$. Since q is a Mersenne prime, an efficient way to implement this operation is to do an integer multiplication with interleaved reduction. Figure 2 shows the design of our base field multiplier architecture. It consists of the multiplication core (which performs the integer multiplication with interleaved reduction of the carry) and a reduction unit for the final reduction of the result which is performed using a ripple carry adder.

The processing cell of the multiplier core, which is shown in Fig. 3, is built with a full adder. Here, a_i and b_i represent the inputs, ca_i represents the carry, and i and k are the column and row numbers, respectively, in Fig. 2.

4.2 Polynomial multiplier

Finite field multiplication of polynomials in an extension field, with coefficients in the base field as described in Section 2.1, is computed using a polynomial multiplier. Using an extension field $GF(q^m)$, we can reduce the area of a finite field multiplier in a very natural way,

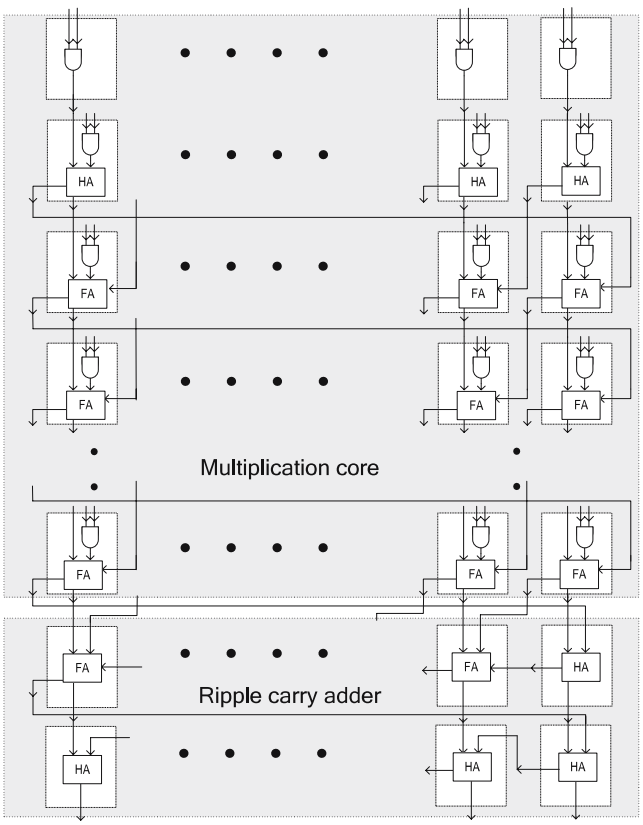

Fig. 2 Base field multiplication with interleaved reduction

since in this case only a smaller base field multiplier is required. For instance, for performing multiplication in $GF((2^{13} - 1)^{13})$ we would only need a 13×13-bit base field multiplier. However, an implementation in the time domain has the disadvantage of having a quadratic time complexity, because a total number of $13 \times 13 = 169$ base field multiplications need to be computed. In our design, we save most of these base field multiplications by utilizing *DFT modular multiplication* which requires performing only a linear number of 26 base field multiplications (Steps 1 – 3, Algorithm 2).

DFT Modular Multiplication

For the application of DFT modular multiplication, Algorithm 2 is modified for an optimized hardware

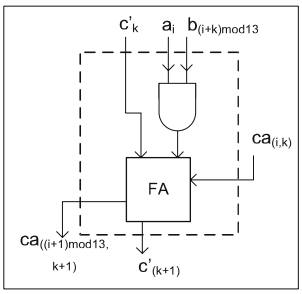

Fig. 3 Processing cell for the base field multiplier core

implementation. The main design decision here is to use a single base field multiplier to perform Step 2 (for pairwise coefficient multiplications) and Step 9 (for multiplications with the constant $-1/d$). Next, the two loops Steps $6-8$ (for accumulating C_i's) and Steps $13-20$ (for computing C_i's) were decided to be performed in parallel simultaneously. The final design that emerged is as shown in Fig. 4 and the functionality is represented by the pseudo-code in Algorithm 3. During Steps $2-5$ (Algorithm 3) the multiplexer select signal *red* is set to 0 and later it is set to 1 for the remaining steps. This allows *MUL* (the base field multiplier) to be used for the initial multiplication, with the proper results accumulated in the register S. The registers S_{even} and S_{odd} cyclically rotate their contents every clock cycle performing Steps 17 and 18 (Algorithm 3), respectively.

Step 19 in Algorithm 2, which involves different amounts of cyclic rotations for different C_i, would normally be inefficient to implement in hardware. In this work, this problem is solved in a *unique* way with the *first-in first-out (FIFO) cyclic register block*. This temporary memory location for storing C_i values pushes in values till it is completely full. In the next loop, as the values are moved out of the *FIFO*, each of them is cyclically rotated at each clock cycle as they move up. Hence the different C_i values are cyclically rotated by different number of bits with no extra cost. The pseudo-code which shows the functionality of the memory block is given in Steps $19-21$ of Algorithm 3. Steps $14-18$ (Algorithm 2) are implemented using the two multiplexers with the select signal *o_e*.

Thus, based on an iterative study of different architectures we investigated, in this work we show the steps of the original DFT modular multiplication algorithm (Algorithm 1), reordered so as to fine tune it to generate a hardware efficient architecture. Due to its regular design, the DFT modular multiplier architecture is easy to layout. The area is optimized by reusing the various components. Also, since all the bus signals are 13-bits wide, signal routing is made extremely easy in the design.

Algorithm 3 Pseudo-code for hardware implementation of DFT modular multiplication

Input: $(A), (B)$
Output: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(m-1)} \bmod f(x)$

1: $S \leftarrow 0$
2: **for** $i = 0$ to $d-1$ **do**
3: $\quad C_i \leftarrow A_i \cdot B_i$
4: $\quad S \leftarrow S + C_i$
5: **end for**
6: **for** $j = 0$ to $m-2$ **do**
7: $\quad S \leftarrow -S/d$
8: $\quad S_{even} \leftarrow S/2$
9: $\quad S_{odd} \leftarrow S + S/2$
10: $\quad S \leftarrow 0$
11: $\quad$ **for** $i = 0$ to $d-1$ **do**
12: $\quad\quad$ **if** $i \bmod 2 = 0$ **then**
13: $\quad\quad\quad C_i \leftarrow C_i + S_{even}$
14: $\quad\quad$ **else**
15: $\quad\quad\quad C_i \leftarrow -(C_i + S_{odd})$
16: $\quad\quad$ **end if**
17: $\quad\quad S_{even} \leftarrow S_{even}/2$
18: $\quad\quad S_{odd} \leftarrow S_{odd}/2$
19: $\quad\quad$ **for** $k = i+1$ to $d-1$ **do**
20: $\quad\quad\quad C_k \leftarrow C_k/2$
21: $\quad\quad$ **end for**
22: $\quad\quad S \leftarrow S + C_i$
23: $\quad$ **end for**
24: **end for**
25: Return (C)

4.3 Point arithmetic

The overall architecture of the ECC processor is shown in Fig. 5. The *Arithmetic Unit* consists of the *DFT modular multiplier*, and the base field adder which has a negation unit on one of its inputs for performing also the subtraction. All the necessary point variables are stored in the *Memory* component. We use FIFO registers here, because DFT modular multiplication and addition/subtraction operate only on 13 bits of the data at each clock cycle. This enables our processor to use 13-bit wide buses throughout the design, resulting in easy routing with reduced power consumption. To avoid losing the contents of the memory when being read out, they are looped back in the FIFO block, if new values are not being written in.

The *Control Unit* is the most important component which performs the point arithmetic by writing the required variables onto the A and B busses, performing the required operations on them and storing the result back to the proper memory register. The *Control Unit* is also responsible for interacting with the external world by reading in inputs and writing out the results. The instruction set of the *Control Unit* is given in Table 2.

The ECC point arithmetic is performed using mixed Jacobian-affine coordinates [12] to save area on inversion. Here, we assume the elliptic curve is of the form $y^2 = x^3 - 3x + b$. We use the binary NAF method (Algorithm 3.31 in Menezes et al. [12]) with mixed coordinates to perform the point multiplication. The point arithmetic is performed in such a way that the

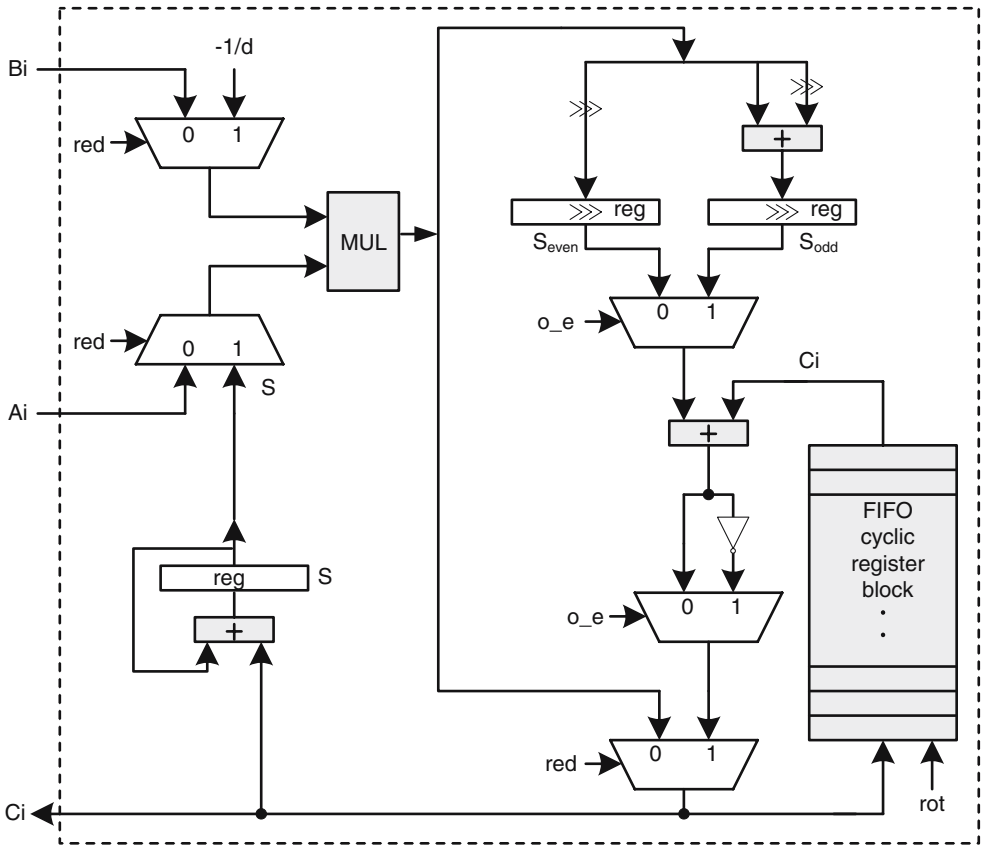

Fig. 4 DFT modular multiplier architecture

least amount of temporary storage is required. Since the point multiplication algorithm allows overwriting of the inputs while performing point doubling and addition, it requires only three extra temporary memory locations. The point doubling operation is performed in Jacobian coordinates and requires 8 DFT modular multiplications and 12 sequence additions (or subtractions). Point addition is performed in mixed Jacobian-affine coordinates and requires 11 DFT modular multiplications and 7 additions. Point subtraction (for the binary NAF method) is easily implemented in exactly the same way as point addition with the exception of flipping the bits of y_2, the y-coordinate of the point to be subtracted. The *Memory* unit therefore consists of eight FIFO register blocks, one for each sequence, and has the total size of $26 \times 13 \times 8 = 2704$ bits.

The inversion operation required for the final conversion from projective to affine coordinates is performed using Fermat's Little Theorem. The conversion from the time to the frequency domain, and vice-versa, is achieved by simple rotations which are performed

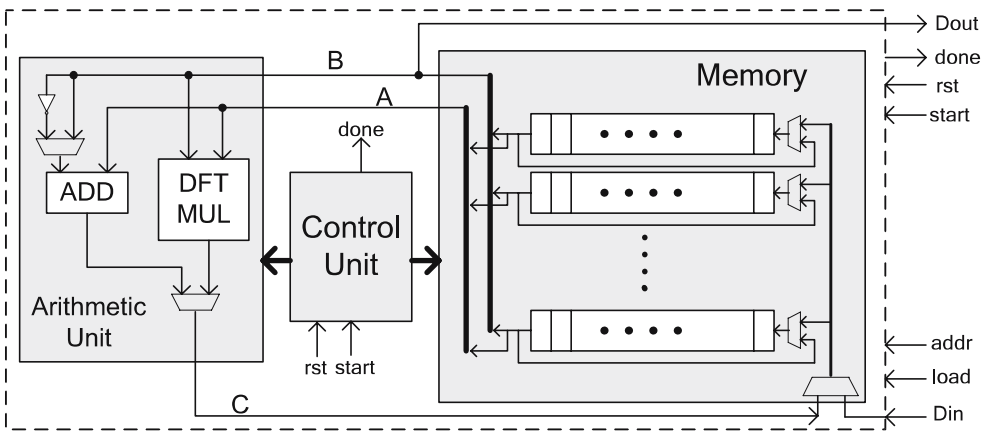

Fig. 5 Top level ECC processor architecture

Table 2 Controller commands of ECC processor

Command	Action
LOAD [$addr$]	Load data into the register [$addr$]
READ [$addr$]	Read data from the register [$addr$]
DFT_MULT [$addr_A$] [$addr_B$] [$addr_C$]	Perform DFT modular multiplication on the sequences in [$addr_A$] and [$addr_B$], and store the result in [$addr_C$]
ADD_SEQ [$addr_A$] [$addr_B$] [$addr_C$]	Perform base field addition on the sequences in [$addr_A$] and [$addr_B$], and store the result in [$addr_C$]
SUB_SEQ [$addr_A$] [$addr_B$] [$addr_C$]	Perform base field subtraction on the sequences in [$addr_A$] and [$addr_B$], and store the result in [$addr_C$]
MOVE_SEQ [$addr_A$] [$addr_B$]	Move data from the register [$addr_A$] to the register [$addr_B$]
DFT_SEQ [$addr$]	Convert the data sequence at [$addr$] to the frequency domain
TIME_SEQ [$addr$]	Convert the data sequence at [$addr$] to the time domain

Table 3 Equivalent gate count areas for the ECC processor

Field	Arithmetic unit	Control unit	Memory	Total area
$GF((2^{13}-1)^{13})$	5, 537.06	351.26	18, 768.66	24, 754.62
$GF((2^{17}-1)^{17})$	6, 978.95	362.56	31, 794.52	39, 243.00
$GF((2^{19}-1)^{19})$	10, 898.82	362.89	39, 586.72	50, 959.02

Table 4 Timing measurements (in clock cycles) for the ECC processor

Field	DFT multiplication	Point double	Point addition	Max. frequency (MHz)	Point multiplication (avg. ms)
$GF((2^{13}-1)^{13})$	354	3, 180	4, 097	238.7	3.47
$GF((2^{17}-1)^{17})$	598	5, 228	6, 837	226.8	10.33
$GF((2^{19}-1)^{19})$	744	6, 444	8, 471	221.7	16.34

Table 5 Comparisons with other ECC processors for similar application scenarios

Implementation	Field size (equiv. binary)	Area (equiv. kgates)	Max. frequency (MHz)	Point multiplication (avg. ms)
Lee et. al. [11]	$\sim 2^{217}$	228	26	11
Öztürk et. al. [14]	$\sim 2^{165}$	30	100	6.3
Satoh and Takano [18]	$\sim 2^{160}$	28	364	7.5
Ours	$\sim 2^{169}$	24.8	238.7	3.47
	$\sim 2^{289}$	39.2	226.8	10.33
	$\sim 2^{361}$	50.9	221.7	16.34

using the *FIFO cyclic register block* inside the DFT modular multiplier unit.

5 Performance analysis

In this section, we present the implementation results for the ECC processor design for three different finite fields: $GF((2^{13}-1)^{13})$, $GF((2^{17}-1)^{17})$ and $GF((2^{19}-1)^{19})$. For our performance measurements, we synthesized for a custom ASIC design using AMI Semiconductor $0.35\mu m$ CMOS technology using the Synopsys Design Compiler tools. Timing measurements were performed using the Modelsim simulator against test vectors generated with Maple.

Table 3 shows the area requirements for the ECC processor in terms of the equivalent number of NAND gates. The areas required for each of the three main components of the processor are also shown individually.

Table 4 presents the number of clock cycles required for DFT modular multiplication and ECC point arithmetic. It also shows the maximum clock frequency of the processor and the total time required to perform a point multiplication.

Although there are numerous ECC hardware implementations which are openly available in the literature, in Table 5 we attempt to compare our results only to VLSI implementations of ECC oriented towards similar application scenarios requiring small area with moderate speed. To the best of our knowledge, the only OEF implementation in hardware is presented by Lee et al. [11]. The authors present an FPGA implementation of ECC over $GF(q^m)$, where $q = 2^{31}-1$ and $m = 7$, which is comparable to our implementation over $GF((2^{17}-1)^{17})$. The best design mentioned here has a gate count of 228k and performs a scalar point multiplication in 11 ms at a maximum possible clock frequency of 26 MHz. The huge area is due to the inversion unit that is used as an alternative to the projective coordinates used in our design. This leads to our design being a factor of 5.8 smaller than the only known OEF hardware implementation but still being able to provide the same timing performance.

We would like to compare our results also to ECC implementations over other fields for similar applications. An implementation of ECC over the prime field $GF((2^{167}+1)/3)$, which has a comparable key length with our implementation over $GF((2^{13}-1)^{13})$, is presented in Öztürk et al. [14]. This design occupies an area of around 30k gates and achieves a point multiplication in 6.3 ms at 100 MHz clock frequency. Our design is 20% smaller than this design and still efficient in performance. Finally, we compare our design to the scalable dual-field based implementation presented by Satoh and Takano [18]. For the 160-bit field size, this implementation has an area of 28k gates and achieves a point multiplication in 7.5 ms at the maximum clock frequency of 364 MHz. Our implementation over $GF((2^{13}-1)^{13})$, with the same field size, is more efficient in terms of both area and time. Based on these observations, we can easily confirm that the proposed implementation is area efficient without compromising on speed.

6 Conclusion

We have proposed a novel hardware architecture for DFT modular multiplication, and also presented an ECC processor architecture to perform point multiplication in the frequency domain using this multiplier. This is the first ever hardware implementation of ECC in the frequency domain, and we hope further research using other point multiplication algorithms applied to this architecture would lead to new results such as side channel resistance. We have synthesized our architecture for custom VLSI CMOS technology to estimate the area and time performance, and shown that the proposed ECC processor is time/area efficient and useful in resource constrained environments such as sensor networks.

References

1. Bailey DV, Paar C (1998) Optimal extension fields for fast arithmetic in public-key algorithms. In: Krawczyk H (ed) Advances in cryptology—CRYPTO '98, vol LNCS 1462. Springer, Berlin Heidelberg New York, pp 472–485
2. Bailey DV, Paar C (2001) Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. J Cryptol 14(3):153–176
3. Baktır S, Sunar B (2005) Finite field polynomial multiplication in the frequency domain with application to elliptic curve cryptography. Technical report. Worcester Polytechnic Institute, Worcester

4. Baktır S, Sunar B (2006) Finite field polynomial multiplication in the frequency domain with application to elliptic curve cryptography. In: Proceedings of the 21st international symposium on computer and information sciences (ISCIS 2006). Lecture notes in computer science (LNCS), vol 4263. Springer, Berlin Heidelberg New York, pp 991–1001
5. Batina L, Örs SB, Preneel B, Vandewalle J (2003) Hardware architectures for public key cryptography. Integr VLSI J 34(6):1–64
6. Blake IF, Seroussi G, Smart N (1999) Elliptic curves in cryptography. Mathematical Society Lecture Notes Series 265. Cambridge University Press, London
7. Burrus CS, Parks TW (1985) DFT/FFT and convolution algorithms. Wiley, New York
8. Kalach K, David JP (2005) Hardware implementation of large number multiplication by FFT with modular arithmetic. In: Proceedings of the 3rd international IEEE-NEWCAS conference. IEEE, Piscataway, pp 267–270
9. Koblitz N (1987) Elliptic curve cryptosystems. Math Comput 48:203–209
10. Kumar S, Girimondo M, Weimerskirch A, Paar C, Patel A, Wander AS (2003) Embedded end-to-end wireless security with ECDH key exchange. In: 46th IEEE midwest symposium on circuits and systems, Cairo, pp 27–30 December 2003
11. Lee M-K, Kim KT, Kim H, Kim DK (2006) Efficient hardware implementation of elliptic curve cryptography over GF(p^m). In: Proceedings of the 6th international workshop on information security applications (WISA 2005). Lecture notes in computer science (LNCS), vol 3786. Springer, Berlin Heidelberg New York, pp 207–217
12. Menezes AJ, van Oorschot PC, Vanstone SA (1997) Handbook of applied cryptography. CRC, Boca Raton
13. Miller V (1986) Uses of elliptic curves in cryptography. In: Williams HC (ed) Advances in cryptology, CRYPTO '85, vol LNCS 218. Springer, Berlin Heidelberg New York, pp 417–426
14. Öztürk E, Sunar B, Savas E (2004) Low-power elliptic curve cryptography using scaled modular arithmetic. In: Proceedings of the workshop on cryptographic hardware and embedded systems (CHES 2004). Lecture notes in computer science (LNCS), vol 3156. Springer, Berlin Heidelberg New York, pp 92–106
15. Pollard JM (1971) The fast fourier transform in a finite field. Math Comput 25:365–374
16. Rader CM (1972) Discrete convolutions via mersenne transforms. IEEE Trans Comput C-21(12):1269–1273
17. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–126
18. Satoh A, Takano K (2003) A scalable dual-field elliptic curve cryptographic processor. IEEE Trans Comput 52(4):1–64
19. Tolimieri R, An M, Lu C (1989) Algorithms for discrete fourier transform and convolution. Springer, Berlin Heidelberg New York
20. Woodbury A, Bailey DV, Paar C (2000) Elliptic curve cryptography on smart cards without coprocessors. In: IFIP CARDIS 2000, fourth smart card research and advanced application conference. Kluwer, Bristol, pp 20–22

Selçuk Baktır received the B.Sc. degree in electrical engineering from Bilkent University, Ankara, Turkey, in 2001 and the M.Sc. degree in electrical and computer engineering from Worcester Polytechnic Institute, Worcester, Massachusetts, in 2003. He is currently pursuing the Ph.D. degree. His research interests include applied cryptography and information security, elliptic curve cryptography, coding and information theory, finite fields, number theory, computer arithmetic, and computational algebra. He is a student member of the IEEE, the IEEE Computer Society, the IEEE Communications Society, and the International Association of Cryptologic Research.

Sandeep Kumar works in the area of applied data security in embedded systems. He completed both his Bachelor of Technology (B Tech) and Master of Technology (M Tech) degrees in electrical engineering from the Indian Institute of Technology, Bombay, India in 2002. He received his Ph.D. (Dr.-Ing.) from Ruhr-University Bochum (HGI, eurobits) in the area of applied data security in 2006. His main research area has been in Elliptic Curve Cryptography (ECC) for constrained devices. He has also worked extensively on other areas of cryptography including efficient software and hardware implementations. He is presently working as a research scientist at Philips Research Europe in the Information and System Security Group.

Christof Paar has the chair for communication security and is director of the Horst Görtz Institute for IT Security at Ruhr University in Bochum (Germany). From 1994 to 2001, he was a professor at Worcester Polytechnic Institute (USA), where he headed the Cryptography and Information Security Labs. He co-founded, with Çetin K. Koç, the Cryptographic Hardware and Embedded Systems (CHES) workshop series, which is one of the leading international forums for research in applied cryptography. His research interests cover fast software- and hardware-realizations of cryptographic algorithms, physical security, cryptanalytical hardware, and embedded security in real world applications such as cars or ad-hoc networks. He has over 80 peer-reviewed publications in embedded cryptography, is editor of 10 conference proceedings, special journal issues and edited books, and holds several patents in this area. He was recipient of a CAREER award of the US National Science Foundation.

Berk Sunar received his B.Sc. degree in Electrical and Electronics Engineering from Middle East Technical University in 1995 and his Ph.D. degree in Electrical and Computer Engineering from Oregon State University in December 1998. After briefly working as a member of the research faculty at Oregon State University, Sunar has joined Worcester Polytechnic Institute as an Assistant Professor. Since July 2006, he is serving as an Associate Professor. He is currently heading the Cryptography and Information Security Laboratory. Sunar received the National Science Foundation CAREER award in 2002. He organized the Cryptographic Hardware and Embedded Systems Conference (CHES) in 2004, and is the co-editor of CHES 2005. His research interests include finite fields, elliptic curve cryptography, low-power cryptography, and computer arithmetic. Sunar is a member of the IEEE Computer Society, the Association for Computing Machinery, and the International Association of Cryptologic Research professional societies.