High Performance FPGA based Elliptic Curve Cryptographic Co-Processor

Jonathan Lutz* and Anwarul Hasan
Department of Electrical and Computer Engineering, University of Waterloo
jonathan.lutz@motorola.com, ahasan@ece.uwaterloo.ca

Abstract

In this paper a high performance elliptic curve co-processor is developed which is optimized for a binary field recommended by NIST. The architecture uses a field multiplier capable of performing a field multiplication over the extension field with degree 163 in 0.060 μsec. The co-processor uses López and Dahab's projective coordinate system and is optimized specifically for Koblitz curves. An efficient implementation of Itoh and Tsujii's method for inversion with performance comparable to the extended Euclidean algorithm is used. A prototype of the processor has been implemented for the binary extension field with degree 163 on a Xilinx XCV2000E FPGA. The prototype runs at 66 MHz and performs an elliptic curve scalar multiplication in 0.233 msec on a generic curve and 0.075 msec on a Koblitz curve.

1. Introduction

Elliptic curve-based cryptosystems are most closely related to algorithms like the Digital Signature Algorithm (DSA) which are based on the discrete logarithm problem. In the DSA, the parameters can be chosen to provide efficient implementations of the algorithm. In the same way, the parameters of ECC based cryptosystems can be selected to optimize the efficiency of the implementation. Unfortunately, the selection of the ECC parameters is not a trivial process and, if chosen incorrectly, may lead to an insecure system. In response to this issue both NIST and ANSI have published recommended finite fields for use in the ECDSA [14, 18]. NIST recommends ten finite fields, five of which are binary fields. The field defining polynomials are $F(x) = x^{163}+x^7+x^6+x^3+1$, $F(x) = x^{233}+x^{74}+1$, $F(x) = x^{283}+x^{12}+x^7+x^5+1$, $F(x) = x^{409}+x^{87}+1$, and $F(x) = x^{571}+x^{10}+x^5+x^2+1$. For each field a Koblitz curve, along with a method for generating a pseudo-random curve, are supplied. These curves were intentionally selected for both cryptographic strength and efficient implementation.

In the recent past, several articles have proposed various hardware architectures/accelerators for ECC. The architectures proposed in [15] and [16] were the first reported cryptographic strength elliptic curve co-processors. Montgomery scalar multiplication with an LSD multiplier is used in [16]. In [15] a new field multiplier is developed and demonstrated in an elliptic curve scalar multiplier. In both [9] and [1], parameterized module generation is discussed. To the best of the authors' knowledge the architecture proposed in [4] offers the fastest scalar multiplication using FPGA technology at 0.144 milliseconds. This architecture uses Montgomery scalar multiplication with López and Dahab's projective coordinates. They use a shift and add field multiplier.

In this paper an architecture which performs efficient elliptic curve scalar multiplication for curves over a specific binary field is presented. The scalar multiplier uses a hardware implementation, described in [11], of the look-up table-based multiplier proposed in [5]. The efficiency and low area requirements of this field multiplier allow for an overall architecture which provides high performance at reduced cost. Furthermore, the architecture can be implemented for any field and reduction polynomial. In order to demonstrate the advantages of the architecture given here, prototypes for the smallest NIST curve, GF(2^{163}), were developed. Though the prototypes were implemented using FPGA technology they could also be implemented as an ASIC design.

2. Preliminaries

Interest here is in the elliptic curve defined by the non-supersingular Weierstrass equation for binary fields, which is defined by the equation

$$y^2 + xy = x^3 + \alpha x^2 + \beta. \quad (1)$$

The variables x and y are elements of the field GF(2^m) as are the curve parameters α and β. Points on the curve, defined by the solutions, (x, y), to (1) form an additive

*This work was supported in part by the Security Technology Center in the Semiconductor Products Sector of Motorola.

group when combined with the "point at infinity". This extra point is the group identity and is denoted by the symbol $\mathcal{O}$. By definition, the addition of two elements in a group results in another element of the group. Therefore, any point on the curve, say P, can be added to itself an arbitrary number of times and the result will also be a point on the curve. So, point P added to itself $k-1$ times and will result in the point kP. Given the binary expansion of $k = 2^{l-1}k_{l-1} + \cdots + 2k_1 + k_0$ the scalar multiple kP can be computed by

$$\begin{aligned} Q &= 2^{l-1}k_{l-1}P + \cdots + 2k_1P + k_0P \\ &= (\cdots(k_{l-1}P)2 + \cdots + k_1P)2 + k_0P. \end{aligned}$$

which can be computed by the well known (left-to-right) double and add method for scalar multiplication.

Two basic operations required for elliptic curve scalar multiplication are point ADD and point DOUBLE. Mathematical definitions for these operations are derived from the curve equation in (1) and both require one field inversion each. Field inversion is quite time consuming and is usually avoided by using projective coordinate systems as discussed in Section 3.1.

3. Efficient Elliptic Curve Scalar Multiplication

Efficient implementation of scalar multiplication requires optimization of both the underlying field arithmetic and curve arithmetic. Optimization of the field arithmetic is discussed in [11] and [12]. In this section, optimizations which relate to the curve arithmetic are discussed. Specific discussion will focus on the use of projective coordinates along with two possible techniques for recoding the scalar.

3.1. Projective Coordinates

Projective coordinates allow the inversion required by each DOUBLE and ADD to be eliminated at the expense of a few extra field multiplications. The benefit is measured by the ratio

$$\frac{\text{Time to Complete Inversion}}{\text{Time to Complete Multiplication}}. \tag{2}$$

Several flavors of projective coordinates have been proposed over the last few years. The prominent ones are *Standard* [13], *Jacobian* [6] and López & Dahab [10] projective coordinates. Used here is the projective coordinate system defined by López and Dahab as it offers the best performance for both point addition and point doubling. The scheme is defined as follows: let the affine representation of P be denoted as (x, y) and the projective representation of P be denoted as (X, Y, Z). The affine and projective coordinates are then related by

$$x = \frac{X}{Z} \quad \text{and} \quad y = \frac{Y}{Z^2}.$$

It is important to note that when using the left-to-right double and add method for scalar multiplication all point additions are of the form ADD(P, Q). The base point P is never modified and as a result will maintain its affine representation, i.e. $P = (x, y, 1)$. The constant Z coordinate significantly reduces the cost of point addition (from 14 field multiplications down to 9). The addition of two distinct points $(X_1, Y_1, Z_1) + (X_2, Y_2, 1) = (X_a, Y_a, Z_a)$ using *mixed* coordinates (one projective point and one affine point) is then computed by

$$\begin{array}{ll} A = Y_2Z_1^2 + Y_1 & E = AC \\ B = X_2Z_1 + X_1 & X_a = A^2 + D + E \\ C = Z_1B & F = X_a + X_2Z_a \\ D = B^2(C + \alpha Z_1^2) & G = X_a + Y_2Z_a \\ Z_a = C^2 & Y_a = EF + Z_aG \end{array} \tag{3}$$

Similarly, the double of a point $(X_1, Y_1, Z_1) + (X_1, Y_1, Z_1) = (X_d, Y_d, Z_d)$ is computed by

$$\begin{aligned} Z_d &= Z_1^2X_1^2 \qquad X_d = X_1^4 + \beta Z_1^4 \\ Y_d &= \beta Z_1^4 Z_d + X_d(\alpha Z_d + Y_1^2 + \beta Z_1^4) \end{aligned} \tag{4}$$

By using the inversion algorithm proposed by Itoh and Tsujii [7], as discussed later in Section 4.2, the ratio in (2) is guaranteed to be larger than $\lfloor \log_2(m-1) \rfloor$ and could be larger depending on the efficiency of the squaring operations. Taking into consideration (i) the cost of a field multiplication, (ii) the cost of a field inversion, and (iii) the number of field multiplications required for affine and projective coordinate systems, López & Dahab's projective coordinates provide the best performance for NIST curves.

3.2. Integer Recoding

The binary expansion of an integer k is written as $k = \sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{0, 1\}$. For the case of elliptic curve scalar multiplication the length l is approximately equal to m, the degree of the extension field. Assuming an average Hamming weight, a scalar multiplication will require approximately $l/2$ point additions and $l-1$ point doubles. Several recoding methods, notably NAF and τ-adic NAF [17], have been proposed which in effect reduce these two values.

The NAF of an Integer: The symbols in the binary expansion are selected from the set $\{0, 1\}$. If this set is increased to $\{0, 1, -1\}$ the expansion is referred to as *signed binary* representation. Interest is in a particular form of this signed representation called NAF or nonadjacent form. A signed binary integer is said to be in NAF if there are no adjacent non-zero symbols. The average Hamming weight of

a NAF is approximately $l/3$ compared to that of the binary expansion which is $l/2$.

When an integer is represented in NAF, the double and add scalar multiplication method must be slightly modified to handle the -1 symbol as shown in Algorithm 1. The negative of the point (x, y) is $(x, x + y)$ and can be computed with a single field addition. For this reason NAF is a particularly attractive recoding method when performing elliptic curve scalar multiplication. The average running time of elliptic curve scalar multiplication when using binary NAF is $(l + 1)/3$ point additions and l point doubles, representing a significant reduction in run time.

Algorithm 1 Scalar multiplication for binary NAF

Input: Integer $k = (k'_l, \ldots, k'_1, k'_0)_{NAF}$, Point P
Output: Point $Q = kP$

```
Q ← O
if (k'_{l-1} ≠ 0) then
    Q ← k'_{l-1} P
for i = l − 2 downto 0 do
    Q ← DOUBLE(Q)
    if (k'_i ≠ 0) then
        Q ← ADD(Q, k'_i P)
```

The τ-NAF of an Integer: Anomalous Binary Curves (ABC's), first proposed for cryptographic use in [8], provide an efficient implementation when the scalar is represented as a complex algebraic number. ABC's, often referred to as the Koblitz curves, are defined by (1) with $\beta = 1$ and either $\alpha = 0$ or $\alpha = 1$. The advantage provided by the Koblitz curves is that the DOUBLE operation in Algorithm 1 can be replaced with a second operation, namely Frobenius mapping, which is easier to perform.

Consider the equation $\tau^2 + 2 = \mu\tau$ where $\mu = (-1)^{1-\alpha}$ and α is the curve parameter in (1) and is 0 or 1 for the Koblitz curves. The solution to this equation is $\tau = \frac{\mu + \sqrt{-7}}{2}$. If point $P = (x, y)$ is on the Koblitz curve then $\tau P = (x^2, y^2)$ which is also on the same curve. If k is represented with radix τ i.e. $k = \sum_{i=0}^{l-1} \kappa_i \tau^i$, then the operation DOUBLE(Q) required for scalar multiplication can be replaced by the Frobenius map of Q or τQ. The modified algorithm is shown in Algorithm 2.

Solinas provides an algorithm in [17] which generates the τ-adic NAF of an integer where the length of the result is approximately the same length as binary expansion of the integer. Like radix 2 NAF, the τ-adic NAF also has an average Hamming weight of approximately $l/3$ for an l-bit integer. So Algorithm 2 has a running time of $l/3$ point additions and $l - 1$ Frobenius mappings.

Algorithm 2 Scalar multiplication for τ-adic NAF

Input: Integer $k = (\kappa_{l-1}, \ldots, \kappa_1, \kappa_0)_{\tau-NAF}$, Point P
Output: Point $Q = kP$

```
Q ← O;
if (κ_{l-1} ≠ 0) then
    Q ← κ_{l-1} P;
for i = l − 2 downto 0 do
    Q ← τQ;
    if (κ_i ≠ 0) then
        Q ← ADD(Q, κ_i P);
```

3.3. Summary and Analysis

A point addition using López & Dahab's projective coordinates requires ten field multiplications, four field squarings and eight field additions. A point double requires five field multiplications, five field squarings and four field additions. Using this information, the run time for scalar multiplication can be written in terms of field operations. Typically scalar multiplication is measured in terms of field multiplications, inversions and squarings, ignoring the cost of addition. In the case of this architecture, field multiplication and squaring are completed quickly enough that the cost of field addition becomes significant. The run times using binary, binary NAF and τ-adic NAF representations are shown in Table 1. These values are based on the curve addition and doubling equations defined in (3) and (4) assuming arbitrary curve parameters α and β and the average Hamming weights discussed in the previous sections. For the case of τ-NAF, a Frobenius mapping is assumed to require three squaring operations. The symbols $\mathcal{M}$, $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{I}$ correspond to field multiplication, squaring, addition and inversion respectively. In each case it is assumed that the length of the integer is approximately equal to m.

Table 1. Field arithmetic required for scalar multiplication

	Cost in Field Operations
Binary	$(10\mathcal{M} + 7\mathcal{S} + 8\mathcal{A})m + \mathcal{I}$
NAF	$(\frac{25}{3}\mathcal{M} + \frac{19}{3}\mathcal{S} + \frac{20}{3}\mathcal{A})m + \mathcal{I}$
τ-NAF	$(\frac{10}{3}\mathcal{M} + \frac{13}{3}\mathcal{S} + \frac{8}{3}\mathcal{A})m + \mathcal{I}$

4. Co-processor Architecture

The architecture, detailed in this section, consists of several finite field arithmetic units, field element storage, and control logic. All logic related to finite field arithmetic is

optimized for specific field size and reduction polynomial. Internal curve computations are performed using López & Dahab's projective coordinate system. While generic curves are supported, the architecture is optimized specifically for the special Koblitz curves.

4.1. Finite Field Arithmetic Units

The arithmetic units used in this design include a multiplier, adder, and squaring unit. The multiplier uses a digit serial algorithm (presented in [5]) and is optimized for a single finite field and reduction polynomial. The squaring unit uses an optimized derivative of the multiplication algorithm. The implemented for the field GF(2^{163}) using a 66 MHz clock, the squaring unit is capable of computing the square of an element in a single cycle. Under the same conditions the multiplier can compute the product of two field elements in four clock cycles (using a digit size of 41). Detailed architectural descriptions of both the multiplier and squaring unit can be found in [11] and [12]. For comparison purposes, we implemented several versions of the co-processor varying the digit size in the field multiplier. Results related to these implementations are given in Section 5.

4.2. The Data Path

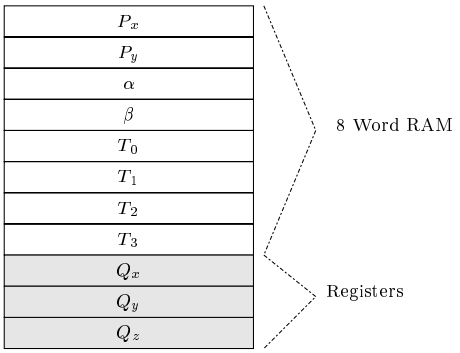
The data path of the co-processor consists of three finite field arithmetic units as well as space for operand storage. In an attempt to minimize time lost to data movement, the adder and multiplier are equipped with dual input ports which allow both operands to be loaded at the same time (the squaring unit requires a single operand and cannot benefit from an extra input bus). Similarly, the field element storage has two output ports used to supply data to the finite field units. This data path requires two cycles to load and one cycle to unload any of the finite field units. This allows the co-processor to complete addition, squaring, and multiplication in three, four, and seven clock cycles respectively (when using a digit size of 41 in the field multiplier).

In addition to providing field element storage, the storage unit provides the connection between the internal m-bit data path and the 32-bit external world. Figure 1 shows how the arithmetic units are connected to the storage unit.

The internal m-bit buses connecting the storage and arithmetic units are controlled to perform sequences of field operations. In this way the underlying curve operations `DOUBLE` and `ADD` as well as field inversion are performed.

Field Element Storage: The field element storage unit provides storage for curve points and parameters as well as temporary values. Parameters required to perform elliptic curve scalar multiplication include the field elements α and β and coordinates of the base point P. Storage will also be required for the coordinates of the scalar multiple Q. The point addition routine developed for this design also requires four temporary storage locations for intermediate values. Figure 2 shows how the storage space is organized.

Figure 2. Field element storage



The top eight field element storage locations are implemented using 32-bit dual-port RAMs generated by the Xilinx Coregen tool and the bottom three storage locations (these locations are shaded gray in Figures 2 and 3) are made of register files with 32-bit register widths. The dual 32-bit/m-bit interface support is achieved by instantiating $\lceil \frac{m}{32} \rceil$ dual-port storage blocks (either memories or register files) with 32-bit word widths as shown in Figure 3. The figure assumes $m = 163$. If the 32-bit storage locations in Figure 3 are viewed as a matrix, then the rows of the matrix hold the m-bit field words. Each 32-bit location is accessible by the 32-bit interface and each m-bit location is accessible by the m-bit interface. For simplicity sake the field elements are aligned at 32 byte boundaries.

Computation of $\tau(Q)$: In addition to providing storage, the registers in the bottom three m-bit locations are capable of squaring the resident field element. This allows the squaring operations required to compute $\tau(Q)$ to be performed in parallel. Furthermore, it eliminates the data movement otherwise required if the squaring unit were to be loaded and unloaded for each coordinate of Q. This provides significant performance improvement when using Koblitz curves.

Fast Inversion: Itoh and Tsujii's method for computing the inverse of an element, described in [7], requires $\lfloor \log_2(m-1) \rfloor + H(m-1) - 1$ multiplications and $m-1$ squarings where $H(\cdot)$ denotes Hamming weight. If C_s and C_m denote the number of clock cycles required to complete a squaring and multiplication respectively, and it takes three

Figure 1. Co-processor data-path

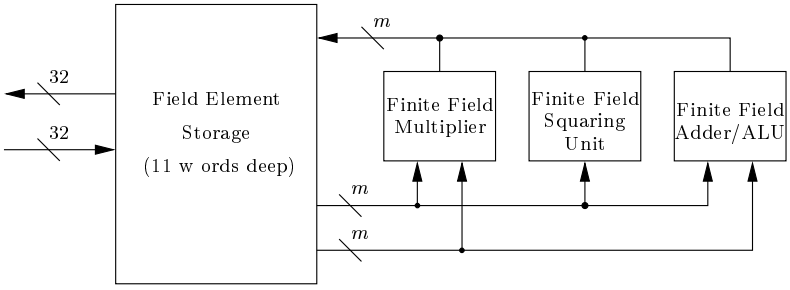
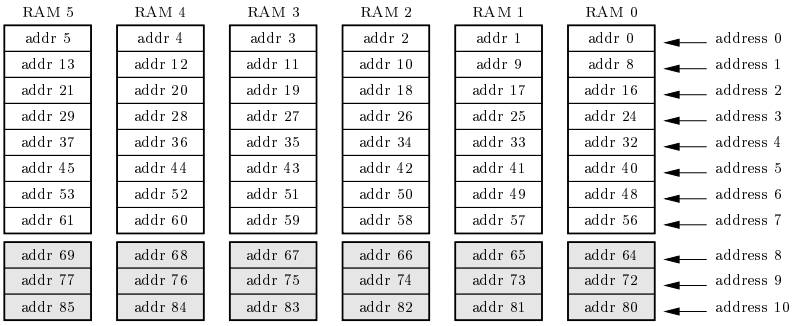


Figure 3. 32**-bit/**163**-bit address map**



clock cycles to load and unload the finite field arithmetic units, then an inversion can be completed in

$$(C_s + 3)(m-1)+ \\ (C_m + 3)(\lfloor \log_2(m-1) \rfloor + H(m-1) - 1)$$

clock cycles. For GF(2^{163}) with $C_s = 1$ and $C_m = 4$ this translates to 711 clock cycles.

Performance can be significantly improved by optimizing the squaring unit to perform computations of the form x^{2^i}. These repeated squarings do not require intermediate values to be stored outside the squaring unit. By modifying the squaring unit to support the *re-square* of an element, most of the memory accesses otherwise required to load and unload the squaring unit are eliminated. Accounting for this optimization, an inversion can then be completed in

$$(C_s(m-1) + 3(\lfloor \log_2(m-1) \rfloor + H(m-1) - 1))+ \\ (C_m + 3)(\lfloor \log_2(m-1) \rfloor + H(m-1) - 1)$$

clock cycles. For the field GF(2^{163}) and the same values for C_s and C_m, this results in 252 clock cycles.

4.3. Control

The processor's architecture consists of the data path and two levels of control. The lower level of control is composed of a micro-sequencer which holds the routines required for curve arithmetic such as `DOUBLE` and `ADD`. The top level control is implemented using a state machine which parses the scalar and invokes the appropriate routines in the lower level control.

The Micro-sequencer: The micro-sequencer controls the data movement between the field element storage and the finite field arithmetic units. In addition to the fundamental load and store operations, it supports control instructions such as jump and branch. The following routines were developed for use in the design.

- `POINT_ADD` (P, Q): Adds the elliptic curve points P and Q where P is represented in affine coordinates and Q is represented using projective coordinates. The result is given in projective coordinates.
- `POINT_SUB` (P, Q): Computes the difference $Q - P$. P is represented using affine coordinates and Q is represented using projective coordinates. The result is given in projective coordinates. This routine calls the `POINT_ADD` routine.
- `POINT_DBL` (Q): Doubles the elliptic curve point Q. Both Q and the result are in projective coordinates.

- `INVERT` (X): Computes the inverse of the finite field element X.
- `CONVERT` (Q): Computes the affine coordinates of an elliptic curve point Q given the point's projective coordinates. This routine calls the `INVERT` routine.
- `COPY_P2Q` (P, Q): Copies the x and y coordinates of point P to the x and y coordinates of point Q. The z coordinate of point Q is set to 1.
- `COPY_MP2Q` (P, Q): Computes the x and y coordinates of point $-P$ and copies them to the x and y coordinates of point Q. The z coordinate of point Q is set to 1.

Top Level Control: The routines listed above are invoked by the top level state machine. This state machine parses the scalar k and calls the routines as needed. The functionality of the state machine is based directly on Algorithms 1 and 2. Since the curve arithmetic is performed using projective coordinates, the state machine must convert the result to affine coordinates at the end of computation.

5. Results

We implemented a prototype of the architecture for the field GF(2^{163}) using the NIST recommended field polynomial. The design was coded using Verilog HDL and synthesized using Synopsys FPGA Compiler II. Xilinx' Foundation software was used to place, route and time the netlist. The prototype was designed to run at 66 MHz on a Xilinx' Virtex 2000E FPGA.

The resulting design was verified on the Rapid Prototyping Platform (RPP) [2, 3] provided by Canadian Microelectronics Corporation (CMC). The hardware/software system included an ARM Integrator/LM-XCV600E+ (board with a Virtex 2000E FPGA) and an ARM Integrator/ARM7TDMI (board with an ARM7 core) connected by the ARM Integrator/AP board. The design was connected to an AHB slave interface which made it directly accessible by the ARM7 core. Stimulated by compiled C-code, the core read from and wrote to the design. The Integrator/AP's system clock had a maximum frequency of 50 MHz. In order to run the design at 66 MHz it was necessary to use the oscillator generated clock provided with the Integrator/LM-SCV600E+. The data headed to and coming from the design was passed across the two clock domains.

Table 2 shows the performance in clock cycles of the prototype's curve operations. These numbers assume a field multiplier digit size of 41. Two values in the table deserve attention. First, the inversion algorithm, based on [7], performs an inversion in 252 clock cycles. A hardware implementation of the Extended Euclidean Algorithm is expected to take approximately $2m$ (i.e. $2 \times 163 = 326$) clock cycles. The implementation presented provides a 60 clock cycle, or 20% improvement, without requiring hardware dedicated specifically for inversion. Second, multiple instantiations of the squaring logic allow for the Frobenius mapping of a projective point to be completed in a single cycle. This significantly improves the performance of scalar multiplication when using the Koblitz curves.

Table 2. Performance of curve operations

Operation	# Cycles
Inversion	252
Point Addition	79
Point Subtraction	87
Point Double	68
Frobenius Mapping	1

The prototype of the scalar multiplier has been implemented using several digit sizes in the field multiplier. Table 3 reports the area consumption and resulting performance of the architecture given several different digit sizes. The field multiplier was unable to meet timing in the co-processor for field sizes greater than 41. For this reason, 41 is the maximum digit size.

Table 4 provides a comparison of published performance results for scalar multiplication. The performance of 0.144 ms reported in [4] is the fastest reported scalar multiplication using FPGA technology. The design presented in this paper provides almost double (0.075 ms) the performance for the specific case of Koblitz curves.

The co-processor discussed in this paper requires approximately half the CLBs used in the co-processor of [4] using the same FPGA. It must be noted that the co-processor presented in [4] is flexible in that it supports all fields up to GF(2^{256}). In applications where support for a only single field size is required, it is excessive to support elliptic curves over many fields. In scenarios such as this, this new elliptic curve co-processor offers an improved cost effective solution. Furthermore, total throughput can be increased by including multiple co-processors in a system. In doing this, a system could perform scalar multiplications at a rate of one every 0.038 ms with an area very close to that reported in [4]. This represents an approximate four fold reduction in execution time for scalar multiplication using the same area.

6. Conclusions

In this paper a new elliptic curve co-processor architecture was proposed. Its feasibility and efficiency were demonstrated through a prototype implementation on an FPGA. The prototype provides record performance for el-

Table 3. Performance and cost results for scalar multiplication

Field Mult. Digit Size	# LUTs	# FFs	Binary (ms)	NAF (ms)	τ-NAF (ms)
4	6,144	1,930	1.107	0.939	0.351
14	7,362	1,930	0.446	0.386	0.135
28	8,838	1,930	0.309	0.272	0.090
41	10,017	1,930	0.264	0.233	0.075

Table 4. Comparison of published results

Implementation	Field	FPGA	Scalar Mult. (ms)
M. Bednara et. al. [1]	GF(2^{191})	Xilinx XCV1000	0.27
Orlando & Paar [16]	GF(2^{167})	Xilinx XCV400E	0.210
N. Gura et. al. [4]	GF(2^{163})	Xilinx XCV2000E	0.144
Our design (digit size of 41)	GF(2^{163})	Xilinx XCV2000E	0.075

liptic curve scalar multiplication over the field GF(2^{163}). Key features of the architecture include (i) multiple instantiations of the squaring logic used to speed up Frobenius mappings and (ii) a high performance implementation of Itoh & Tsujii's inversion method.

Possible future work includes extending support to several field sizes. Ideally, the architecture would support all NIST recommended fields simultaneously. Logic would be reused wherever possible. Extra logic would be limited to certain parts of the squaring and multiplication units which are dependent on the field reduction polynomial. Another possible improvement to the design would be to implement a window scheme using point pre-computation. Such a scheme cannot be used to speed up the Montgomery scalar multiplication method, implemented in [16] and [4]. A windowing scheme can increase performance by up to four times or, in our case, approximately 0.02 ms per scalar multiplication.

References

[1] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich. Implementation of elliptic curve cryptographic coprocessor over GF(2^m) on an FPGA. In *International Parallel and Distributed Processing Symposium: IPDPS Workshops*, April 2002.

[2] C. M. Corporation. *CMC Rapic-Prototyping Platform: Design Flow Guide*, 2002.

[3] C. M. Corporation. *CMC Rapic-Prototyping Platform: Installation Guide*, 2002.

[4] 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. In *Cryptographic Hardware and Embedded Systems (CHES)*, 2002.

[5] M. A. Hasan. Look-up table-based large finite field multiplication in memory constrained cryptosystems. *IEEE Transactions on Computers*, 49(7), July 2000.

[6] IEEE. *P1363: Editorial Contribution to Standard for Public Key Cryptography*, February 1998.

[7] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and Computing*, 78(3):171–177, 1988.

[8] N. Koblitz. CM curves with good cryptographic properties. In *Advances in Cryptography, Crypto '91*, pages 279–287. Springer-Verilag, 1991.

[9] P. H. W. Leong and I. K. H. Leung. A microcoded elliptic curve processor using FPGA technology. *IEEE Transactions on VLSI Systems*, 10(5), October 2002.

[10] J. Lopez and R. Dahab. Improved algorithms for elliptic curve arithmetic in GF(2^n). In *Selected Areas in Cryptography*, pages 201–212, 1998.

[11] J. Lutz. High performance elliptic curve cryptogrphic coprocessor. Master's thesis, University of Waterloo, 2003, see http://library.uwaterloo.ca/ETD/etheses.html.

[12] J. Lutz and A. Hasan. High performance finite field multiplier for cryptographic applications. In *SPIE 2003, Optical Science and Technology*, To be published.

[13] A. Menezes. Elliptic curve public key cryptosystems. *Kluwer Academic Publishers*, 1993.

[14] NIST. *FIPS 186-2 draft, Digital Signature Standard (DSS)*, 2000.

[15] S. Okada, N. Torii, K. Itoh, and M. Takenaka. Implementation of elliptic curve cryptographic coprocessor over GF(2^m) on an FPGA. In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 25–40. Springer-Verlag, 2000.

[16] G. Orlando and C. Paar. A high-performance reconfigurable elliptic curve processor for GF (2^m). In *Cryptographic Hardware and Embedded Systems (CHES)*, 2000.

[17] J. A. Solinas. Improved algorithms for arithmetic on anomalous binary curves. In *Advances in Cryptography, Crypto '97*, 1997.

[18] A. X9.62. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1998.