Hardware/Software Co-design Implementations of Elliptic Curve Cryptosystems

Turki F. Al-Somani¹, Esam A. Khan², Ahmad M. Qamar-ul-Islam¹ and Hilal Houssain³

¹Computer Engineering Department, Umm Al-Qura University, Makkah, Saudi Arabia

{tfsomani, amqamarulislam}@uqu.edu.sa

²The Custodian of the Two Holy Mosques Institute for Hajj Research, Umm Al-Qura University,

Makkah, Saudi Arabia

eakhan@uqu.edu.sa

³LIMOS, CNRS, University Blaise Pascal, Clermont-Ferrand II, France

hilal.houssain@isima.fr

Abstract

This paper presents a survey of hardware/software co-design implementations of elliptic curve cryptosystems. A critical study of the underlying finite field, the representation basis, and the partitioning schemes of these implementations is conducted. The study shows that all implementations are implemented over binary fields $GF(2^m)$ and the implementations that use polynomial basis are more than implementations that use normal basis for finite field arithmetic. The study also shows that the best partitioning scheme, among the surveyed implementations,

implements the finite field arithmetic on hardware and the remaining operations of the Elliptic Curve Cryptosystem (ECC) on software.

Keywords: Elliptic Curve Cryptosystems (ECCs), Hardware/software co-design, Normal Basis, $GF(2^m)$.

1. Introduction:

Elliptic Curve Cryptosystems (ECCs) ([Cohen, 2005](#); [Hankerson, 2004](#)) have been recently attracting increased attention. The ability to use smaller key sizes and the computationally more efficient ECC algorithms are two main reasons why elliptic curve cryptosystems are becoming more popular. They are considered to be particularly suitable for implementation on platforms with constrained storage and/or battery specifications, e.g. smart cards or mobile devices.

Hardware/software co-design was first proposed by Franke and Purvis ([Franke and Purvis, 1991](#)) as a new design approach that combines the hardware and software perspectives from the earliest stages of the design process and exploits the design flexibility and efficient allocation of functions that such an approach offers. The main goal of hardware/software co-design is to achieve better designs and meet system-level objectives by concurrently designing both hardware and software ([Micheli and Gupta, 1997](#)). This paper presents a survey of hardware/software co-design implementations of elliptic curve cryptosystems.

2. Finite Field Arithmetic

In abstract algebra, a *finite field* is a field that contains only finitely many elements. Finite fields are important in number theory, algebraic geometry, Galois theory, coding theory, and cryptography (Biggs, 1985) (McEliece, 1987) (Lidl, and Niederreiter, 1994).

A *group* is a set of elements G together with one binary operation, $\diamond$, which have the following properties:

1. **Closure:** $\forall a, b \in G, a \diamond b \in G$.
2. **Associativity:** $\forall a, b, c \in G, (a \diamond b) \diamond c = a \diamond (b \diamond c)$.
3. **Identity:** The group contains an identity element $e \in G$ such that $\forall a \in G, a \diamond e = e \diamond a = a$.
4. **Inverse:** Every element $a \in G$ has an inverse $a^{-1} \in G$ such that $a \diamond a^{-1} = a^{-1} \diamond a = e$.

Abelian groups are groups with commutative group operation; i.e., $a \diamond b = b \diamond a \quad \forall a, b \in G$.

Cyclic groups are groups that have a generator element. An element $g \in G$, is a generator of the group if each element $a \in G$ can be generated by repeated application of the group operation on g . Thus, $\forall a \in G$,

$$a = \underbrace{g \diamond g \diamond g \diamond \dots \diamond g}_{i \text{ times}}. \quad (1)$$

Additive groups, are groups with the "+" group operator, denoted as:

$$ig = \underbrace{g + g + g + \dots + g}_{i \text{ times}}. \quad (2)$$

Similarly, *multiplicative groups*, are groups with the "*" group operator, denoted as:

$$g^i = \underbrace{g * g * g * \dots * g}_{i \text{ times}} \quad (3)$$

The order of a group G , represented by the symbol $|G|$, is the number of elements in the group. A *field* is a set of elements F with two binary operations, represented here as addition $++$ and multiplication $**$, which have the following properties:

1. F is an abelian group with respect to the $++$ operation.
2. The elements of the set F^* form an abelian group under the $**$ operation. The set F^* is a set that contains all the elements in F except the additive identity.
3. The distribution law applies to the two binary operations; as follows:

$$\forall a, b, c \in F, a * (b + c) = (a * b) + (a * c).$$

Finite fields or Galois field, so named in honor of Evariste Galois, are represented by the symbol $GF(q)$. For any prime p and positive integer m , there always exists a Galois field of order $q = p^m$. The prime p is called the characteristic of the finite field $GF(p^m)$.

3. Elliptic Curve Cryptosystems

Elliptic Curve Cryptosystem (ECC), which was originally proposed by Niel Koblitz and Victor Miller in 1985 (Koblitz, 1987; Miller, 1986), is seen as a serious alternative to RSA (Rivest *et al.*, 1978) with much shorter key size. ECC with key size of 128-256 bits is shown to offer equal security to that of RSA with key size of 1-2K bits. To date, no significant breakthroughs have been made in determining weaknesses in the ECC algorithm, which is based on the discrete logarithm problem over points on an elliptic curve. The fact that the problem appears so difficult to crack means that key sizes can be reduced in size considerably, even exponentially (Hankerson et al., 2004). This made ECC become a serious challenge to RSA. The advantage of ECC is being recognized recently where it is being incorporated in many standards. ECCs have gained popularity for cryptographic applications because of the short key compared

with earlier public key cryptosystems such as RSA (Rivest *et al.*, 1978) and ElGamal (ElGamal, 1985). They are considered particularly suitable for implementations on smart cards or mobile devices.

Extensive research has been done on the underlying math, security strength and efficient implementations of elliptic curve cryptosystems. Among the different fields that can underlie elliptic curves, prime fields $GF(p)$ and binary fields $GF(2^m)$ have shown to be best suited for cryptographic applications. An elliptic curve E over the finite field $GF(p)$ defined by the parameters $a, b \in GF(p)$ with $p > 3$, consists of the set of points $p = (x, y)$, where $x, y \in GF(p)$, that satisfy the elliptic curve equation (Equation 4) together with the additive identity of the group point O known as the “point at infinity” (Koblitz, 1987).

$$y^2 = x^3 + ax + b$$

(4)

where $a, b \in GF(p)$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

The number of points n on an elliptic curve over a finite field $GF(q)$ is defined by Hasse’s theorem (McEliece, 1987). The set of discrete points on an elliptic curve form an abelian group, whose group operation is known as point addition. Elliptic curve point addition is defined according to the “chord-tangent process”. Point addition over $GF(p)$ is described as follows: Let P and Q be two distinct points on E defined over $GF(p)$ with $Q \neq P$ (Q is not the additive inverse of P). The addition of the two points P and Q is the point R ($R = P + Q$), where R is the additive inverse of S , and S is a third point on E intercepted by the straight line through points P and Q . The additive inverse of a point $P = (x, y) \in E$, over $GF(p)$, is the point $-P = (x, -y)$ which is the reflection of the point P with respect to the x -axis on E .

When $P = Q$ and $P \neq -P$ the addition of P and Q is the point R ($R = 2P$), where R is the additive inverse of S and S is the third point on E intercepted by the straight line tangent to the curve at point P . This operation is referred to as point doubling.

Equation (5) defines the non-supersingular elliptic curve equation for $GF(2^m)$ fields. Only non-supersingular curves over $GF(2^m)$ are considered since supersingular curves are not secure. Supersingular elliptic curves define a special class of curves with some special properties that make them unstable for cryptography (Menezes, 1993).

$$y^2 + xy = x^3 + ax^2 + b$$

(5)

where $a, b \in GF(2^m)$ and $b \neq 0$.

For a non-supersingular elliptic curve E defined over $GF(2^m)$, point addition and point doubling operations are generally computed using the algebraic formulae as follows:

1. **Identity:** $P + O = O + P = P$ for all $P \in E$.
2. **Negatives:** If $P = (x, y) \in E$, then $(x, y) + (x, x + y) = O$. The point $(x, x + y)$ is called the negative of P , denoted as $-P$.
3. **Point Addition:** Let $P = (x_1, y_1), Q = (x_2, y_2) \in E$, $P \neq Q$ and $Q \neq -P$ then

$P + Q = (x_3, y_3)$ where

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + \left(\frac{y_1 + y_2}{x_1 + x_2}\right) + x_1 + x_2 + a$$

$$y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right) \cdot (x_1 + x_3) + x_3 + y_1$$

4. **Point Doubling:** If $P = Q = (x_1, y_1)$ then $2P = P + P = (x_3, y_3)$ where

$$x_3 = x_1^2 + \frac{b}{x_1^2}$$

$$y_3 = x_1^2 + (x_1 + \frac{y_1}{x_1})x_3 + x_3$$

A major operation required by elliptic curve cryptosystems is the point scalar multiplication. The scalar multiplication operation, denoted as kP , where k is an integer and P is a point on the elliptic curve represents the addition of k copies of point P as given by Equation 6.

$$kP = P + P + \dots + P \text{ (} k \text{ times)}$$

(6)

Elliptic curve cryptosystems are built over cyclic groups. Each group contains a finite number of n points that can be represented as scalar multiples of a generator point: iP for $i = 0, 1, \dots, n-1$, where P is a generator of the group. The order of point P is n , which implies that $nP = O$ and $iP \neq O$ for $1 < i \leq n-1$. The order of each point on the group must be dividable by n . Consequently, a point multiplication kQ for $k > n$ can be computed as $(k \bmod n)Q$. A good survey has been conducted by Hankerson *et. al.* in ([Hankerson, 2004](#)).

Projective coordinate systems define points over the projective plane as triplets (X, Y, Z) . Projective coordinate systems are used to eliminate the number of inversions ([Menezes, 1993](#)). For an elliptic curve defined over $GF(2^m)$, many different forms of formulas may be used for point addition and doubling. For the Homogeneous coordinate system, an elliptic curve point (x, y) takes the form $(x, y) = (X/Z, Y/Z)$ ([Koyama and Tsutsumi, 1993](#)), while for the Jacobian coordinate system; a point takes the form $(x, y) = (X/Z^2, Y/Z^3)$ ([Cohen et al., 1997](#)). The

Lopez-Dahab coordinate system takes the form $(x, y) = (X/Z, Y/Z^2)$ (Lopez and Dahab, 1999).

The Mixed coordinate system, on the other hand, adds two points where one is given in a certain coordinate system while the other is given in another coordinate system. The coordinate system of the resulting point may be in a third coordinate system (Cohen et al., 1998).

The finite $GF(2^m)$ field, with characteristic 2, has particular importance in cryptography since it leads to efficient hardware. Elements of the $GF(2^m)$ field are represented in terms of a basis. Most implementations use either a *Polynomial Basis* or a *Normal Basis*. Normal basis is more suitable for hardware implementations than polynomial basis since operations are mainly comprised of rotation, shifting and exclusive-ORing which can be efficiently implemented in hardware.

In Elliptic Curve Diffie-Hellman Protocol, the base point P and the elliptic curve equation are public. User's A private and public keys are k_A and P_A respectively. User's A public key is equal to k_AP . User's B , on the other hand, private and public keys are k_B and P_B respectively. Similarly, User's A public key is equal to k_BP . The message to be encrypted is embedded into the x -coordinate of a point on the elliptic curve ($P_m = (x_m, y_m)$) (Rosing, 1999). The shared secret key S between two parties A and B is easily calculated by

$$S = k_A(k_B P) = k_B(k_A P)$$

Whenever one of the users need to send a message to the other party, he/she needs to add the shared secret key to the message to produce the ciphertext point P_C which is

$$P_C = P_m + S$$

To decrypt the ciphertext point, the secret key is subtracted from the ciphertext point to give the plaintext point P_m as follows

$$P_m = P_C - S$$

In elliptic curve ElGamal protocol, on the other hand, for some user to encrypt and send the message point P_m to user A , he/she chooses a random integer “ l ” and generates the ciphertext C_m which consists of the following pair of points:

$$C_m = (lP, P_m + lP_A)$$

The ciphertext pair of points uses A 's public key, where only user A can decrypt the plaintext using his/her private key. To decrypt the ciphertext C_m , the first point in the pair of C_m , lP is multiplied by A 's private key to get the point $k_A(lP)$. This point is subtracted from the second point of C_m to produce the plaintext point P_m

The complete decryption operations can be summarized in the following equation

$$P_m = (P_m + lP_A) - k_A(lP) = P_m + l(k_A P) - k_A(lP)$$

4. Hardware/Software Co-Design

In digital system design, hardware and software were two distinct parts that are designed relatively independently. In general, the role of hardware engineers was to supply general-purpose computing systems. Then, it comes the role of software engineers to program these systems without the need to worry about the details of the low-level architecture of the hardware (Franke and Purvis, 1991).

However, due to the evolution of very high scale integration, new chips are large enough to include complete systems. Nowadays, it is possible to have CPUs, memories, and other digital systems in a single chip. The existence of embedded CPUs and complex digital systems raised two classes of problems: 1) the design of the embedded CPU and 2) the design of the software running on these CPUs. In this case, software design becomes a first-class component in chip

design. The need for designing both software and hardware in the early stage formed the root of what is called today *hardware/software co-design* (Wolf, 2003).

The term “co-design” was first proposed in 1991 by Franke and Purvis (Franke and Purvis, 1991) as a new design approach that combines the hardware and software perspectives from the earliest stages of the design process and exploits the design flexibility and efficient allocation of functions that such an approach offers. The idea of concurrently designing both hardware and software is older than this date. In (Sommerville, 1989), it is stated that “It is becoming increasingly cost-effective to delay decisions about which functions should be implemented in hardware and which functions should be software components”, which is a main task of hardware /software co-design.

Co-design can be thought of as a special case of an older approach called *concurrent engineering*, which is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. In this context, co-design attempts to better integrate two concurrent activities, namely the design of the hardware and software components of a system (Franke and Purvis, 1991).

The main goal of co-design is to achieve better designs and meet system-level objectives by concurrently designing both hardware and software (Micheli and Gupta, 1997). However, there are many design factors that make the design space exploration of a digital system a difficult task. One factor is the level of programmability. A system can be designed at the application level using general purpose processors programmed using high level languages. It can also be designed at the instruction level. In this case, special processors are designed with instruction set architectures (ISAs) that are optimized for a specific application domain. These kinds of processors are called *Application Specific Instruction Processors* (ASIPs) (some papers

call them semi-custom processors to differentiate them from general-purpose processors (GPPs) and custom processors, which are called *Application Specific Integrated Circuits* (ASICs)) (Vahid, 2003). The software designers then need to use this ISA to program the ASIP. A third option is to program the design in the hardware level, in which a software is used to configure the hardware after it is manufactured. Hence, programmability and reprogrammability can be done for both hardware and software (Micheli and Gupta, 1997).

Another design factor that increases the design space of a system is the implementation feature. This includes the design style (e.g., clocking strategy and operation mode), the manufacturing technology (e.g, CMOS or bipolar), and the integration level (e.g., system components in a single chip (SoC) or different chips) (Micheli and Gupta, 1997). Considering all these factors in the design of a system increases the possibility of different optimal solution (with respect to different design factors).

The evolution of new integrated circuit technologies (FPGA, ASIC, complex systems, embedded systems) also motivates using co-design (Micheli and Gupta, 1997). Co-design can be done by human designers. However, it is unlikely that human designers can optimize all objectives and consider all design factors. This may lead to designs that are lower than the optimal ones. Moreover, detailed-level design performed by humans is often a time-consuming and error-prone task. Therefore, it is better to have some automatic approaches for co-design supported by computer-aided design (CAD) tools (Micheli and Gupta, 1997).

Co-design can be applied to different types of systems and platforms. Examples include stand-alone digital systems, embedded systems, and reconfigurable hardware (Micheli and Gupta, 1997). Co-design involves several processes and phases. We can classify them into three main phases:

1. **Modeling:** The modeling phase includes identifying system requirements and specification (Franke and Purvis, 1991), refining the specifications, and producing a hardware and software model (Micheli and Gupta, 1997). Identifying system requirements and specification of both hardware and software is called *co-specification* (Ernst, 1998). This means to start with an abstract notation such that each component or module is independent of its final realization in hardware or software (Franke and Purvis, 1991). Capturing all system requirements at the earliest stage restricts the space of possible design options and hence makes specification, design, implementation, and verification efforts more straightforward (Franke and Purvis, 1991). One important process of the modeling phase is hardware/software partitioning (or co-partitioning), whose goal is to find those parts of the model best implemented in hardware and those best implemented in software (Micheli and Gupta, 1997). Hardware is targeted for higher performance, and software is better for flexibility (Schaumont, 2003). Hardware/software partitioning has a first order impact on the cost/performance characteristics of the final design (Micheli and Gupta, 1997). It should be noted here that co-partitioning should be made in the most appropriate manner and not according to conventional wisdom (Franke and Purvis, 1991). After partitioning, scheduling can be loosely defined as assigning an execution start time to each task in a set, where tasks are linked by some relations. The tasks can be elementary (like hardware operations or computer instructions) or can be a group of elementary operations (like software programs) (Micheli and Gupta, 1997). For embedded systems, The modeling style can be *homogeneous* or *heterogeneous* (Micheli and Gupta, 1997). In homogeneous modeling, a modeling language (e.g., the C programming language) or a graphical formalism (e.g.,

state charts) is used to represent both the hardware and software portions. Then, hardware/software partitioning is applied to the model. Partitioning can be decided by the designer, with a successive refinement and annotation of the initial model, or determined by a CAD tool. In heterogeneous modeling, the hardware/ software partition is often outlined by the model itself, because hardware and software components may be expressed in the corresponding languages. Nevertheless, system designers may want to explore alternative implementations of some components.

2. **Validation:** Validation is the process of achieving a reasonable level of confidence that the system will work as designed. It has two main goals: to insure the correct functionality of the designed system and that the required performance levels are achieved in the implementation of a system model (Micheli and Gupta, 1997). Validation (also known as co-verification) could be achieved by *co-simulation*, in which the software component is simulated as running on or communicating with the hardware component (Ernst, 1998).
3. **Implementation:** Implementation means the physical realization of hardware and software. When both components are implemented concurrently, it is called *co-synthesis*, although it may involve hardware synthesis and software compilation (Micheli and Gupta, 1997).

There exists a number of frameworks for hardware/software co-design of FPGA systems. In (Tham and Maskell, 2006), a number of these frameworks are mentioned and compared. Some of these frameworks focus only on the co-simulation step of the co-design process. Other frameworks need manual translation of software codes into HDLs for hardware implementation. A software-oriented methodology is presented in (Tham and Maskell, 2006) that uses SystemC

to model both software and hardware parts and then use a tool called SystemCrafter to translate the hardware part into HDL. Another methodology is presented in (Jussel and Sullivan, 2003) and (Sullivan, 2002) where Handel-C is used as the modeling language. This methodology consists of four main steps: software coding, software profiling, function partitioning and co-simulation, and hardware/software co-synthesis. Due to the experience we have with the tools used in this methodology, we intend to use it in this work.

5. Hardware/Software Co-Design Implementations of Elliptic Curve Cryptosystems

Janssens *et al.* presented the first hardware/software co-design of a $GF(2^m)$ elliptic curve cryptosystem (Janssens et al., 2001). The proposed design in (Janssens et al., 2001) performs scalar multiplication using a software controller written in C. Point operations and field arithmetic, on the other hand, are implemented in hardware. The presented hardware/software co-design in (Janssens et al., 2001) has been implemented on an Atmel FPSLIC FPGA with $m = 8, 16, 72, 192$ bits. The FPSLIC incorporates an AVR micro-controller, memory, peripherals, and a small FPGA on the same chip. The finite field operations are implemented on the FPGA, with the elliptic curve operations on the micro-controller. The data bus between the micro-controller and the FPGA logic is only 8 bits wide, creating a severe bottleneck. A block diagram of the processor can be seen in [Fig. 1](#).

Fig. 1: The block diagram of the processor of (Janssens et al., 2001).

Zeng *et al.* presented in (Zeng et al., 2002) a $GF(2^{209})$ hardware/software co-design of an elliptic curve cryptosystem according to the partitioning schemes proposed in (Janssens et al., 2001). An 8-bit embedded micro-controller IP core is used as a software controller in (Zeng et al., 2002). The hardware part, on the other hand, has been synthesized based on 0.35 micrometer standard cell library. The proposed design in (Zeng et al., 2002) was the first VLSI hardware/software co-design of an elliptic curve cryptosystem. The finite field arithmetic is implemented using normal basis.

Ernst *et al.* presented in (Ernst et al., 2002) three hardware/software co-design implementations of a $GF(2^{113})$ elliptic curve cryptosystem on an Atmel AT94K40 FPGA. These implementations include: a pure software implementation of the whole elliptic curve cryptosystem, an implementation that uses a hardware multiplier, and an implementation that implements the field arithmetic on hardware.

Cheung *et al.* proposed in (Cheung et al., 2005) four partitioning schemes. Two of these schemes were proposed already in (Ernst et al., 2002). The four partitioning schemes presented in (Cheung et al., 2005) include: a pure software implementation scheme, a scheme that uses a hardware multiplier, a scheme that uses a hardware inverter and a scheme that implements point multiplication in hardware. The four proposed partitioning schemes were implemented on the Xilinx ML310 board containing a XC2VP30 FPGA chip with $m = 113, 162, 270$ bits using normal basis. A diagram of the datapath of the system can be seen in Fig. 2.

Fig. 2: The datapath of the ECC of (Cheung et al., 2005).

Koschuch *et al.* in (Koschuch et al., 2006) have co-designed an elliptic curve cryptosystem over binary extension fields using the Dalton 8051 as host controller which executes the software part. The hardware part consists of an elliptic curve acceleration unit (ECAU) and an interface with direct memory access (DMA) to enable fast data transfer between the ECAU and the external RAM (XRAM) attached to the 8051 microcontroller as can be seen in Fig. 3.

Fig. 3: The block diagram of the system of (Koschuch et al., 2006)

Batina et. al. in (Batina et. al., 2006) investigated three methodologies on two different platforms: 8051 micro-controller connected to an FPGA and an AVR micro-controller connected to an FPGA. The first methodology implements only the finite field multiplication on the FPGA. The second implements all finite field arithmetic on the FPGA. The third implements all finite field and point operations on the FPGA. The point operations are accomplished via micro-code. A block diagram of the hardware architecture can be seen in Fig. 4.

Fig. 4: The block diagram of the design of (Batina et. al., 2006).

Recently, Ramsey in ([Ramsey, 2008](#)) investigated the implementation of elliptic curve scalar multiplication on Hybrid FPGA. C code executing on a PowerPC processor performed complex elliptic curve arithmetic algorithms. This C code is interfaced with a finite field processor placed in reconfigurable fabric surrounding the processor. The presented hardware/software co-design in ([Ramsey, 2008](#)) has been implemented on a Xilinx Virtex4 Hybrid FPGA with $m = 163, 233, 283, 409, 571$ bits using normal basis.

6. Discussion

The main goal of this survey is to provide a critical study of the underlying finite field, the representation basis, and the partitioning schemes of the hardware/software co-design implementations of ECCs. Table 1 summarizes the underlying finite field, the representation basis, and the number of bits of these implementations. Clearly, Table 1 shows that all the implementations use binary fields $GF(2^m)$ and none of these implementations use prime fields $GF(p)$ ([Janssens et al., 2001](#); [Zeng et al., 2002](#); [Ernst et al., 2002](#); [Cheung et al., 2005](#); [Koschuch et al., 2006](#); [Batina et. al., 2006](#); [Ramsey, 2008](#)). This is because $GF(2^m)$ have shown to be best suited for cryptographic applications ([Cohen, 2005](#)). Despite that normal basis representation provides more efficient hardware, Table 1 shows that the implementations that use polynomial basis ([Janssens et al., 2001](#); [Ernst et al., 2002](#); [Koschuch et al., 2006](#); [Batina et. al., 2006](#)) are more than implementations that use normal basis ([Zeng et al., 2002](#); [Cheung et al., 2005](#); [Ramsey, 2008](#)).

Table 1 : A Summary of hardware/software co-design implementations of ECCs.

Reference	Galois Field	Representation	No. of bits (m)
(Janssens et al., 2001)	$GF(2^m)$	Polynomial Basis	8, 16, 72, 192
(Zeng et al., 2002)	$GF(2^m)$	Normal Basis	209
(Ernst et al., 2002)	$GF(2^m)$	Polynomial Basis	113
(Cheung et al., 2005)	$GF(2^m)$	Normal Basis	113, 162, 270
(Koschuch et al., 2006)	$GF(2^m)$	Polynomial Basis	163, 191
(Batina et. al., 2006)	$GF(2^m)$	Polynomial Basis	83, 163
(Ramsey , 2008)	$GF(2^m)$	Normal Basis	163, 233, 283, 409, 571

The partitioning schemes, on the other hand, starts by a pure software implementation scheme ([Ernst](#) et al., 2002; [Cheung](#) et al., 2005). Then, the partitioning schemes differ in which part of the ECC remains on software and which is implemented by hardware. The survey showed that the best partitioning scheme, among the surveyed implementations, implements the finite field arithmetic on hardware and the remaining operations of the ECC on software ([Janssens](#) et al., 2001; [Zeng](#) et al., 2002; [Ernst](#) et al., 2002; [Cheung](#) et al., 2005; [Koschuch](#) et al., 2006; [Batina](#) et. al., 2006; [Ramsey](#), 2008).

7. Conclusion

In this paper a survey of hardware/software co-design implementations of elliptic curve cryptosystems has been presented. A critical study of the underlying finite field, the representation basis, and the partitioning schemes of these implementations have been conducted. The study showed that all implementations are implemented over binary fields

$GF(2^m)$ and none of them are implemented over prime fields $GF(p)$. The study also showed that the implementations that use polynomial basis are more than implementations that use normal basis for finite field arithmetic. The study also showed that the best partitioning scheme, among the surveyed implementations, implements the finite field arithmetic on hardware and the remaining operations of the Elliptic Curve Cryptosystem (ECC) on software.

Acknowledgments

The authors would like to acknowledge the support of King Abdul Aziz City for Science and Technology (KACST) for the grant of the research No. DRP-2-1. The authors would like also to acknowledge the support of Umm Al-Qura University (UQU).

References

Batina, L., A. Hodjat, D. Hwang, K. Sakiyama and I. Verbauwhede, 2006. Reconfigurable architectures for curve-based cryptography on embedded micro-controllers. In Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL 2006), Aug. 28-30, Madrid, Spain, pp: 1–4, Available from:

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4101048, ISBN: 1-4244-0312-X

DOI: 10.1109/FPL.2006.311286.

Biggs, N., 2003. Discrete Mathematics. 2nd edition, Oxford University Press, New York, ISBN-10: 0198507178, ISBN-13: 978-0198507178.

Cheung, R., W. Luk and P. Cheung, 2005. Reconfigurable elliptic curve cryptosystems on a chip. In Proceedings of Design, Automation and Test in Europe, Mar. 7-11, Munich, Germany, vol.

1, pp: 24–29, Available from: <http://portal.acm.org/citation.cfm?id=1048924.1049067>, DOI: 10.1109/DATE.2005.254.

Cohen, H., A. Miyaji and T. Ono, 1997. Efficient elliptic curve exponentiation. Proceedings of Advances in Cryptology ICICS '97, Dec. 12-15, Zhengzhou, China, Springer-Verlag. LNCS 1334, pp: 282-290, Available from: <http://www.springerlink.com/content/w536l58355350v50/?p=9afa5450617f468f9bb11ffc69911364&pi=1>, DOI: 10.1007/BFb0028484, ISBN 978-3-540-63696-0.

Cohen, H., A. Miyaji and T. Ono, 1998. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In Advances in Cryptology -ASIACRYPT '98, 18-22 October 1998, Kazuo Ohta, Dingyi Pei (Eds.), LNCS 1514, Springer-Verlag, New York, pp. 51-65, ISBN: 3-540-65109-8.

Cohen, H., G. Frey, R.M. Avanzi, C. Doche, T. Lange, K. Nguyen and F. Vercauteren, 2005. Handbook of Elliptic and Hyperelliptic Curve Cryptography. Discrete Mathematics and Its Applications, Vol. 34, Chapman and Hall, CRC, USA, ISBN: 9781584885184.

T. ElGamal, 1985. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. Advances in Cryptology: Proceedings of CRYPTO 84, August 19-22, Springer Verlag, pp. 10-18, DOI: 10.1007/3-540-39568-7_2, Available from: <http://www.springerlink.com/content/jl0mkpm32tn8ve3q/>.

Ernst, R., 1998. Codesign for embedded systems: Status and trends. IEEE Design Test Comput., Vol. 15, Issue 2, pp. 45 – 54, DOI: 10.1109/54.679207, Available from:

<http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel4%2F54%2F14951%2F00679207.pdf%3Farnumber%3D679207&authDecision=-203>

Ernst, M., M. Jung, F. Madlener, S. Huss and R. Blümel, 2002. A reconfigurable system on chip implementation for elliptic curve cryptography over GF (2^m). Proceedings of Cryptographic Hardware and Embedded Systems - CHES 2002, LNCS 2523, pp: 381–399, DOI 10.1007/3-540-36400-5_28, Available from:

<http://www.springerlink.com/content/arh2jdkyqj4djd/fulldtext.pdf>

Franke, D.W. and M.K. Purvis, 1991. Hardware/software co design: A perspective. In Proceedings of the 13th International conference on Software Engineering. Austin, Texas, United States, May 13-16, pp: 344–352, DOI: 10.1109/ICSE.1991.130661, Available from:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=130661&isnumber=3626> .

Hankerson, D., Menezes, A. and S. Vanstone, 2004. Guide to Elliptic Curve Cryptography. 1st edition, Springer, ISBN-13: 978-0387952734.

Janssens, S., J. Thomas, W. Borremans and P. Gijssels, 2001. Hardware/software co-design of an elliptic curve public-key cryptosystem. In Proceedings of IEEE Workshop on of Signal Processing Systems, Sept. 26-28, Antwerp, Belgium, pp: 209–216, DOI: 10.1109/SIPS.2001.957349, Available from:

<http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F7590%2F20700%2F00957349.pdf&authDecision=-203> .

Jussel, J. and C. Sullivan, 2003. Software-Compiled System Design: A Methodology for Field-Programmable System-on-Chip Design. Celoxica Ltd., Abingdon, Oxfordshire, UK, Available from: <http://www.eda.org/edps/edp03/submissions/paperCeloxica.pdf>.

Koblitz, N., 1987. Elliptic curve cryptosystems. Math. Comput., Vol. 48, No. 177, pp. 203-209.

M. Koschuch, J. Lechner, A. Weitzer, J. Großschädl, A. Szekely, S. Tillich and J.

Wolkerstorfer. Hardware/software co-design of elliptic curve cryptography on an 8051 microcontroller. In Proceedings of Cryptographic Hardware and Embedded Systems – CHES 2006, LNCS 4249, pp: 430 – 444, DOI: 10.1007/11894063_34, Available from: https://online.tu-graz.ac.at/tug_online/voe_main2.getvolltext?pDocumentNr=91442.

Koyama, K. and Y. Tsutsumi, 1993. Speeding up elliptic cryptosystems by using signed binary window method. Proceedings of the Advances in Cryptology Crypto '92, Springer-Verlag, LNCS 740, pp: 345-357, ISBN: 978-3-540-57340-1, DOI: 10.1007/3-540-48071-4_25, Available from: <http://www.springerlink.com/content/bdjjju843tvx6ak9n/?p=b939852b15254c62909220736c018a75&pi=0>.

Lidl, R. and Niederreiter, H. 1994. Introduction to finite fields and their applications'. Cambridge University Press, Cambridge, UK, 2nd edition, ISBN-13: 978-0521460941.

Lopez, J. and R. Dahab, 1998. Improved Algorithms for Elliptic Curve Arithmetic in GF (2^n). Springer-Verlag, pp: 201-212, ISBN: 978-3-540-65894-8.

- Lopez, J. and R. Dahab, 1999. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In Proceedings of the Cryptographic Hardware and Embedded Systems - CHES '99. LNCS 1717, pp: 316-327, DOI : 10.1007/3-540-48059-5_27, Available from: <http://www.springerlink.com/content/0q3wn0y79w1lma0b/fulltext.pdf>.
- R. McEliece, 1987. Finite Fields for Computer Scientists and Engineers. Kluwer Academic Publishers, Boston, 1st edition, ISBN-13: 978-0898381917.
- Menezes, A., 1993. Elliptic Curve Public Key Cryptosystems. 1st edition, Kluwer Academic Publishers, ISBN-13: 978-0792393689.
- Micheli, G.D. and R.K. Gupta, 1997. Hardware/software co-design. Proc. IEEE, Vol. 85, No. 3, pp. 349–365, DOI: 10.1109/5.558708, Available from: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=558708&isnumber=12188>.
- Miller, V.S., 1986. Use of elliptic curves in cryptography. In Proceedings of the Advances in Cryptology CRYPTO '85, Springer-Verlag, LNCS 218, pp: 417–426, DOI : 10.1007/3-540-39799-X_31, ISBN: 978-3-540-16463-0, Available from: <http://www.springerlink.com/content/w475304616327668/fulltext.pdf>
- Ramsey, G.J., 2008. Hardware/software optimizations for elliptic curve scalar multiplication on hybrid FPGAs. M. Sc. Thesis, Rochester Institute of Technology.
- Rosing, M., 1999. Implementing Elliptic Curve Cryptography. Manning Publications Company, ISBN-10: 1884777694, ISBN-13: 978-1884777691.

Rivest, R., A. Shamir and L. Adleman, 1978. A method for obtaining digital signatures and public key cryptosystems. Commun. ACM, Vol. 21, No.2, pp. 120-126, DOI:

10.1145/357980.358017, <http://portal.acm.org/citation.cfm?doid=357980.358017>

Schaumont, P. and I. Verbauwhede, 2003. Domain-specific codesign for embedded security.

IEEE Comput., Vol. 36, No. 4, pp. 68–74, DOI: 10.1109/MC.2003.1193231, Available from:

<http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F2%2F26760%2F01193231.pdf%3Farnumber%3D1193231&authDecision=-203>

Sommerville, I., 2004. Software Engineering. 7th edition, Workingham, England, Addison-Wesley, ISBN-10: 0321210263, ISBN-13: 978-0321210265.

Sullivan, C., 2002. Co design comes to virtex-II pro and microblaze systems, develop your hardware and software in a single, integrated environment. Xcell J., 1: 36 – 39.

Tham, K.S. and D.L. Maskell, 2006. Software-oriented approach to hardware-software co-simulation For FPGA-Based RISC extensible processor. Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'06), Aug. 28-30, Madrid, Spain, pp: 1-6, DOI: 10.1109/FPL.2006.311230, Available from:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&isnumber=&arnumber=4100992>

Vahid, F., 2003. The Softening of hardware. IEEE Comput., Vol. 36, No. 4, pp. 27-34, DOI: 10.1109/MC.2003.1193225, Available from:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1193225&isnumber=26760>

Wolf, W., 2003. A decade of hardware/software co design. IEEE Comput., Vol. 36, No. 4, pp. 38–43, DOI: 10.1109/MC.2003.1193227, available from:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1193227&isnumber=26760>

Zeng, X., X. Zhou and Q. Zhang, 2002. Hardware/software co-design of elliptic curves public-key cryptosystems. In Proceedings of International Conference on Commun. Circuits and System, Jun. 29 – Jul. 1, St. Petersburg, Russia, Vol. 2, pp: 1496–1499, Available from:

<http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F8402%2F26481%2F01179062.pdf&authDecision=-203>