Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2016, Article ID 6371403, 10 pages
http://dx.doi.org/10.1155/2016/6371403

Research Article

FPGA Based High Speed SPA Resistant Elliptic Curve Scalar Multiplier Architecture

Khalid Javeed[1,2] and Xiaojun Wang[2,3]

[1]*Electrical Engineering Department, COMSATS Institute of Information Technology, Abbottabad, Pakistan*
[2]*School of Electronic Engineering, Dublin City University, Dublin, Ireland*
[3]*School of Computer & Software, Nanjing University of Information Science and Technology, Nanjing, Jiangsu, China*

Correspondence should be addressed to Khalid Javeed; khalidjaveed@ciit.net.pk

Received 16 December 2015; Revised 30 March 2016; Accepted 3 May 2016

Academic Editor: Michael Hübner

The higher computational complexity of an elliptic curve scalar point multiplication operation limits its implementation on general purpose processors. Dedicated hardware architectures are essential to reduce the computational time, which results in a substantial increase in the performance of associated cryptographic protocols. This paper presents a unified architecture to compute modular addition, subtraction, and multiplication operations over a finite field of large prime characteristic GF(p). Subsequently, dual instances of the unified architecture are utilized in the design of high speed elliptic curve scalar multiplier architecture. The proposed architecture is synthesized and implemented on several different Xilinx FPGA platforms for different field sizes. The proposed design computes a 192-bit elliptic curve scalar multiplication in 2.3 ms on Virtex-4 FPGA platform. It is 34% faster and requires 40% fewer clock cycles for elliptic curve scalar multiplication and consumes considerable fewer FPGA slices as compared to the other existing designs. The proposed design is also resistant to the timing and simple power analysis (SPA) attacks; therefore it is a good choice in the construction of fast and secure elliptic curve based cryptographic protocols.

1. Introduction

Elliptic curve based cryptography (ECC) proposed independently by Miller [1] and Koblitz [2] has established itself as a proper alternative to the traditional systems such as Ron Rivest, Adi Shamir, and Leonard Adleman (RSA) [3]. The National Institute of Standards and Technology (NIST) recommended 256 bits of key lengths for ECC to achieve the same level of security as 3072 bits of RSA.

Due to the fact that ECC offers similar security with considerable smaller key sizes than RSA, it has been standardized by IEEE and NIST [4]. Thus, as the result of smaller key sizes, its implementation led to substantial reduction in power consumption and storage requirements and offers potentially higher data rates. These inherent properties rank it as a strong candidate for providing security in resource-constrained devices. Unfortunately, due to the underlying complex mathematical structure, its implementation on general-purpose processors (GPP) struggles to meet the speed requirements of many real-time applications.

Thus, several new implementation platforms have been explored during the last years. Field programmable gate array (FPGA) has been established as a proper platform for implementation of security algorithms such as ECC and RSA. Its shorter design cycle time, lower design cost, and its reconfigurability make it more attractive than other platforms, such as Application Specific Integrated Circuits (ASICs).

Elliptic curve scalar point multiplication is the central and most time consuming operation in all ECC based schemes. Its efficient implementation on various platforms is very critical. It is achieved by manipulating points on a properly chosen elliptic curve over a finite field. Mathematically, it is expressed as $Q = sP$, where P is a base point, s is an integer value, and Q is the resultant point of multiplication of s and P. For example, it can be achieved by adding P to itself $(s - 1)$ times. The strength of any ECC schemes is based on the computational

hardness of finding s given P and Q known as Elliptic Curve Discrete Logarithm Problem (ECDLP).

There are several elliptic curve representations satisfying different performance and security requirements. A flexible design capable of supporting different values for elliptic curve parameters and a prime p is more demanding. The ECDLP is not the only way of finding scalar s; it can also be revealed by monitoring the timing [5] and power consumption of cryptographic devices known as side channel attacks (SCAs) [6]. The simplest SCAs are based on the timing and simple power consumption analysis (SPA). Detailed surveys on known SCAs, countermeasures, and secure ECC implementations are reported previously in [7, 8].

Elliptic curve scalar point multiplication involves many basic modular arithmetic operations such as addition, subtraction, multiplication, inversion, and division. Hence, optimization of these operations can significantly improve the performance of ECC schemes.

Elliptic curve cryptosystems can be designed on a finite field either with prime characteristics GF(p) or with binary characteristics GF(2^m). The GF(2^m) arithmetic is easier to implement in hardware than GF(p) because of carry-free arithmetic. However, field parameters in GF(2^m) are mostly fixed and are not very flexible. Some efficient ECC implementations over GF(2^m) are presented in [9–14]. A very good survey of high speed hardware implementations of ECC has been reported in [15].

Several hardware based elliptic curve processors over GF(p) have also been proposed in the literature [5, 16–26]. The design reported in [21] proposed two architectures to speed up the EC point multiplication operation. Both these architectures are based on incorporating parallel dedicated hardware units to compute arithmetic operations such as addition, subtraction, multiplication, and division over GF(p). The GF(p) multiplication unit [21] is based on a bit-serial interleaved multiplication while, for a division over GF(p), a dedicated hardware unit based on a binary version of the extended Euclidean algorithm is used. Ghosh et al. proposed a speed and area optimized architecture for EC point multiplication by exploiting a concept of shared hardware arithmetic over GF(p) [20]. The saving in area is achieved by sharing hardware resources among different GF(p) arithmetic operations, while multiple copies of the arithmetic units are used to speed up EC point multiplication.

1.1. Contribution. Modern FPGAs have dedicated built-in arithmetic components (dedicated multipliers, block RAMs, etc.) to perform different signal processing tasks efficiently. However, in this work these components are not used due to the limitations of the adopted technique to perform a modular multiplication, that is, Interleaved Multiplication (IM) algorithm [27], which interleaved the reduction step by reducing each partial product. To the best of authors knowledge, no work has been reported targeting a digit-wise implementation of the IM technique. However, available small-sized dedicated multipliers inside an FPGA can be very effective in case of the Montgomery multiplication [28] and the NIST recommended primes [29]. A modular multiplication using these methods can be performed by integers multiplication followed by a modular reduction.

This paper presents a novel architecture to speed up the EC point multiplication in affine coordinates. The proposed design is based on a unified GF(p) adder, subtractor, and multiplier (Add/Sub/Mul) unit. The unified Add/Sub/Mul unit is an extension of our previous GF(p) multiplier design reported in [30]. The proposed unified unit in this work performs modular addition and subtraction in a single clock cycle, while modular multiplication is performed in $\lceil K/3 \rceil + 2$ clock cycles, where $K = \log_2 p$. The careful FPGA implementation of the proposed EC point multiplication architecture outperforms the other existing designs. The main advantages of the proposed design are as follows.

(i) It reduces the number of required clock cycles and computation time of EC point multiplication to almost 40% and 35%, respectively, with considerably smaller FPGA area consumption. The reduction in clock cycles and computation time is mainly due to the proposed GF(p) multiplier [30].

(ii) Furthermore, the adopted algorithm for EC point multiplication with careful implementation of GF(p) arithmetic primitives is capable of resisting the timing and SPA attacks [5].

(iii) It is flexible; all parameters (curve parameter a, EC point P, scalar value s, and the prime value p) can be easily changed without FPGA reconfiguration.

This paper is organized as follows. Section 2 briefly explains EC group operations such as EC point addition and EC point doubling. In addition, this section also describes the Montgomery ladder structure for the EC point multiplication algorithm. The unified Add/Sub/Mul unit over GF(p) is presented in Section 3. Section 4 proposes a novel architecture for EC point multiplication based on the GF(p) unified Add/Sub/Mul unit. Implementation results and performance evaluation are presented in Section 5, and finally the paper is concluded in Section 6.

2. Elliptic Curve Group Operations

In this paper, we consider an elliptic curve $\mathbb{E}$, defined over a prime field GF(p), where p is a large prime characteristic number. Field elements are represented as integers in the range $[0, p-1]$. An elliptic curve $\mathbb{E}$ over GF(p) in short Weierstrass form is represented as

$$\mathbb{E} : y^2 = x^3 + ax + b, \tag{1}$$

where, a, b, x, and $y \in$ GF(p) and $4a^3 + 27b^2 \neq 0$ (modulo p). The set of all points (x, y) that satisfies (1), plus the point at infinity, makes an abelian group. EC point addition and EC point doubling operations over such groups are used to construct many elliptic curve cryptosystems. The EC point addition and EC point doubling operations in affine coordinates can be represented as follows: let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on the elliptic curve. The group

Input: An integer s and a point P on elliptic curve
Output: sP
(1) $S_1 \leftarrow P$
(2) $S_2 \leftarrow 2P$
(3) **for** $i = K - 2$ ***downto*** 0 **do** // K is the bit length of s //
(4) **if** $s_i = 1$ **then**
(5) $S_1 \leftarrow S_1 + S_2$ ‖ EC Point addition
(6) $S_2 \leftarrow 2S_2$ ‖ EC Point doubling
(7) **else**
(8) $S_2 \leftarrow S_1 + S_2$
(9) $S_1 \leftarrow 2S_1$
(10) **return** S_1

Algorithm 1: Montgomery ladder for EC point multiplication [20].

operation is the point addition, $P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_2)$, which is defined by the group law and is given as

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \tag{2}$$

where

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{if } P_1 \neq P_2 \\ \dfrac{3x_1^2 + a}{2y_1} & \text{if } P_1 = P_2. \end{cases} \tag{3}$$

If $P_1 = P_2$, then a special case of adding a point to itself is called EC point doubling operation. In affine coordinates the EC point addition requires one division, two multiplications, and six addition or subtraction operations, whereas the EC point doubling can be performed by using one division, three multiplications, and seven addition or subtraction operations. Therefore, optimization of these operations impacts significantly on the overall performance of the EC point multiplication operation.

2.1. Elliptic Curve Scalar Multiplication. EC cryptosystems are mostly based on the EC point multiplication operation. This operation can be performed as a sequence of EC point addition and EC point doubling operations given in Algorithm 1, which is known as the Montgomery ladder for EC point multiplication. Algorithm 1 works on the binary representation of s and it is assumed that the most significant bit is equal to 1. The EC point addition and EC point doubling operations are not dependant on the bit pattern of s, so these operations can be performed in parallel. As these can be executed concurrently, therefore Algorithm 1 gives an extra feature of protection against the timing and simple power analysis (SPA) attacks.

3. Unified Add/Sub/Mul Unit

In this section we present a unified modular adder, subtractor, and multiplier (unified Add/Sub/Mul) unit. This unit is capable of performing modular addition, subtraction, and multiplication operations and supports any prime p; therefore it is able to provide hardware support for ECC over a variety of elliptic curves. Normally, to achieve a better performance of EC point multiplication on dedicated hardware, multiple copies of GF(p) adder, subtractor, multiplier, and divider units are integrated. These multiple copies can help to execute several operations in parallel at the expense of area and cost, which can also result in more power consumption. Our objective is to accelerate the computation of EC point multiplication operation with minimum number of dedicated arithmetic units. Modular multiplication is a critical component in the architecture of EC point multiplication operation. In this regard, several modular multipliers have been proposed. The design reported in [19] is based on an iterative addition and reduction algorithm. In every iteration addition and reduction modulo p of partial products are performed. It computes a K-bit modular multiplication in $K + 1$ clock cycles. Two novel architectures based on radix-4 and radix-8 Booth encoding techniques are reported in [30, 31].

In [30] the radix-4 Booth encoded version computes a modular multiplication operation in $K/2 + 1$ clock cycles, whereas the radix-8 Booth encoded multiplier takes $\lceil K/3 \rceil + 2$ clock cycles. The radix-8 Booth encoded multiplier given in Algorithm 2 is based on an iterative addition and reduction modulo p of partial products technique proposed by Blakley reported in [27]. The two main components in the design are as follows:

(i) Three-bit left shift modulo p unit (Step (3)).

(ii) Addition and subtraction modulo p unit (Steps (7), (9), (11), and (13)).

There is also a logic circuit for Booth encoding in addition to these two core components. The presented unified Add/Sub/Mul unit is based on the same design. The radix-8 Booth encoded modular multiplier design has a modular adder/subtractor unit. Hence this paper modified the radix-8 Booth encoded modular multiplier design in such a way that it becomes capable of performing modular addition and subtraction operations in addition to its main task, that is, a modular multiplication operation. Due to the proposed modification dedicated hardware units for modular addition and subtraction operations are not needed.

The top-level block of unified Add/Sub/Mul unit is shown in Figure 1. The whole logic components of the radix-8 Booth encoded modular multiplier are mainly divided into shared and unshared logic parts. The shared logic components can be shared to perform modular addition, subtraction, and multiplication operations, whereas the unshared logic components are only dedicated to a modular multiplication operation. A control unit is responsible for decoding instructions on the basis of two bits of operational code (opcode) and generates appropriate signals for the shared and unshared logic parts.

The shared logic is comprised of a modular adder/subtractor unit while the unshared logic consists of three-bit left shift modulo p unit and Booth encoding logic. The adder/subtractor and three-bit left shift modulo p units are shown in Figure 2. The three-bit left shift modulo p unit is comprised of three identical D1 units cascaded in series. Each

Input: $p, a, b : 0 \le a, b < p$
Output: $z = a \times b \bmod p$
(1) $z = 0$, $R_1 = 2a \bmod p$, $R_2 = 3a \bmod p$, $R_3 = 4a \bmod p$.
(2) **for** $i = K$ *downto* 0; $i = i - 3$ **do** // K is the bit length of p //
(3) $z \coloneqq 8z \bmod p$
(4) **if** $(b_{(i,i-1,i-2,i-3)}) = (\{0000\} \mid \{1111\})$ **then**
(5) $z \coloneqq z$
(6) **else if** $(b_{(i,i-1,i-2,i-3)}) = (\{0001\} \mid \{0010\}\{1101\} \mid \{1110\})$ **then**
(7) $z \coloneqq z \pm a$
(8) **else if** $(b_{(i,i-1,i-2,i-3)}) = (\{0011\} \mid \{0100\}\{1011\} \mid \{1100\})$ **then**
(9) $z \coloneqq z \pm R_1$
(10) **else if** $(b_{(i,i-1,i-2,i-3)}) = (\{0101\} \mid \{0110\}\{1001\} \mid \{1010\})$ **then**
(11) $z \coloneqq z \pm R_2$
(12) **else**
(13) $z \coloneqq z \pm R_3$
(14) **return** z

Algorithm 2: Radix-8 BE IM algorithm [30].

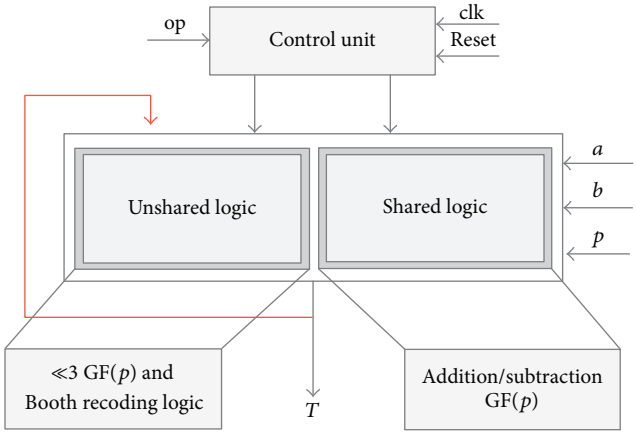

Figure 1: Unified Add/Sub/Mul unit.

Table 1: Operation codes for unified Add/Sub/Mul unit.

Logic operation	Opcode	Shared logic	Unshared logic
GF(p) addition	00	$(a + b) \bmod p$	—
GF(p) subtraction	01	$(a - b) \bmod p$	—
GF(p) multiplication	10	$(d1 \pm d2) \bmod p$	$8d \bmod p$

D1 unit performs a single bit left shift modulo p operation and it consists of one K-bit adder and a multiplexer. Hence, in total, the unshared logic consists of three K-bit adders, three multiplexers, and an additional logic for Booth recoding. The adder/subtractor unit consists of two K-bit adders and five multiplexers.

In the proposed unified Add/Sub/Mul unit, these hardware logic resources are shared with other resources, so two K-bit adders and five multiplexers are saved. This unit is not capable of performing modular addition, subtraction, and multiplication operations in parallel. However, EC point representation in affine coordinates has a very limited scope of parallelism. Therefore, the proposed unified Add/Sub/Mul unit can increase the performance of EC point multiplication in affine coordinates with a lower area overhead. The proposed unified Add/Sub/Mul unit performs modular addition, subtraction, and multiplication operations as given in Table 1 in the following manner.

A GF(p) addition is performed by the shared logic unit, if the two-bit input opcode = 00. The control unit decodes the opcode and activates the shared logic block; that is, the adder/subtractor unit and sets $c_{\text{in}} = 0$. The adder/subtractor unit consists of two K-bit adders and logic for input output multiplexing shown in Figure 2. The first K-bit adder performs addition of operands $(a + b)$ and the result is fed into the second K-bit adder where a modulus p is subtracted from it. Similarly, a GF(p) subtraction is performed by the same unit by setting opcode = 01; the first K-bit adder performs subtraction $(a - b)$ followed by the addition of a modulus p. The result of modular addition and subtraction becomes available at port T after a single clock cycle. In the case of GF(p) multiplication indicated by opcode = 10, the control unit generates appropriate signals for the shared and unshared logic components. Partial products addition or subtraction $(d1 \pm d2 \bmod p)$ is computed by the shared logic components depending on cin signal generated by the Booth recoding logic, while three-bit left shift modulo p $(8d \bmod p)$ operation is computed by the unshared logic components. The detailed execution procedure and control signals for both shared and unshared logic components are given in [30]. The unified Add/Sub/Mul architecture takes $\lceil K/3 \rceil + 2$ clock cycles to produce a GF(p) multiplication result at port T. The main advantages of the proposed unified Add/Sub/Mul units are a single unit that can handle GF(p) addition, subtraction, and multiplication instructions. It eliminates a need for dedicated hardware units for GF(p) addition and subtraction operations, which consumes two K-bit adders in addition to I/O multiplexers. The proposed unit is not only optimized for hardware resources and required

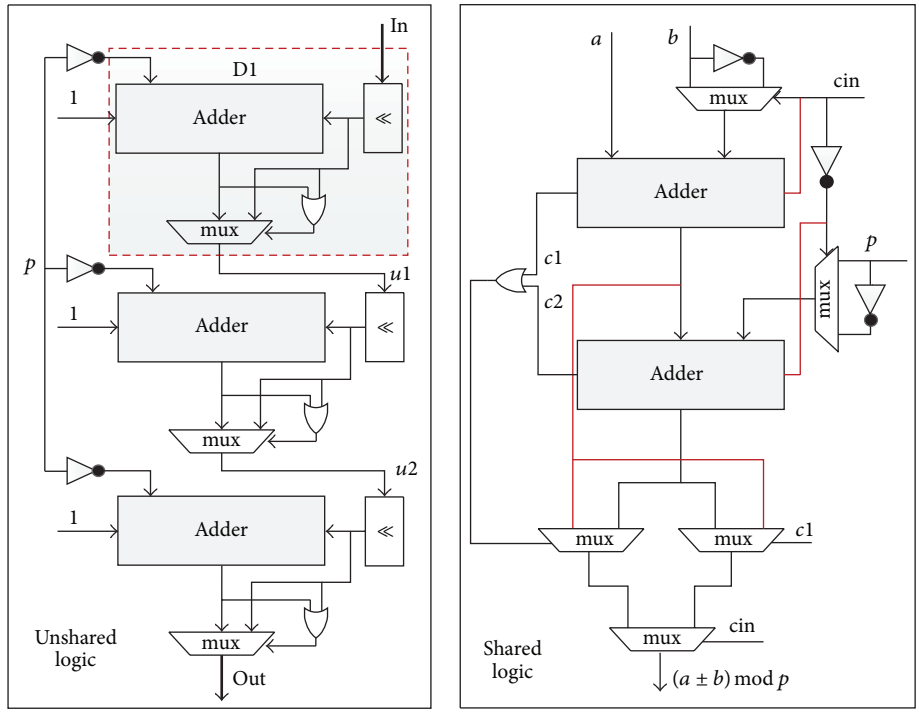

Figure 2: Shared and unshared logic components.

number of clock cycles for GF(p) multiplication operation, but it is also programmable and supports any value for a modulus p.

4. Elliptic Curve Scalar Multiplier Architecture

ECC based schemes heavily rely on the EC scalar multiplication operation; therefore, its efficient implementation can greatly improve the performance of associated ECC based protocols.

The EC scalar multiplication is the computation of sP operation, where s is an integer and P is a base point of a chosen elliptic curve. Several algorithms have been proposed to compute the EC scalar multiplication operation [29]. Standard double-and-add, nonadjacent form (NAF), and a Montgomery ladder for EC point multiplication are mostly used. EC point addition and EC point doubling operation can be executed in parallel using a Montgomery ladder method given in Algorithm 1. As these EC point operations are not dependant on the respective scalar bit s_i, hence, power consumptions of these operations are symmetric and it is not possible for an attacker to extract any information regarding a secret value s. Therefore, this technique provides a protection against simple power analysis attacks. This section presents an efficient architecture for EC scalar multiplication in affine coordinates based on the proposed unified Add/Sub/Mul unit in Section 3. The proposed EC scalar multiplier architecture executes a scalar multiplication as a sequence of EC point addition and EC point doubling operations. These EC point operations can be achieved as a sequence of GF(p) arithmetic operations as given in Table 2.

The EC point addition operation requires six GF(p) subtraction, two GF(p) multiplication, and one GF(p) division operations. On the other hand, three GF(p) addition, four GF(p) subtraction, two GF(p) multiplication, and single GF(p) division operations are required to perform EC point doubling operation. As depicted in Table 2, the EC point operations in affine coordinates also require GF(p) division operation in addition to GF(p) addition, subtraction, and multiplication operations. A GF(p) division and inversion can be performed either by Fermat little theorem or by Extended Euclidean algorithm (EEA). The binary version of EEA given in [29] is the mostly adopted algorithm for GF(p) division. The EEA implementation in this work is based on the guidelines presented in [34]. It takes $2K$ clock cycles to perform a K-bit GF(p) division or inversion operation.

It is evident from Table 2 that, in the computation of EC point operations, a scope of parallelism among GF(p) arithmetic operations is very limited. Therefore, a semiparallel architecture for EC scalar multiplication is shown in Figure 3.

It consists of two GF(p) unified Add/Sub/Mul units, two GF(p) divider units, two register files (each comprised of 3 K-bit registers), I/O multiplexers, and a main controller. The GF(p) unified Add/Sub/Mul unit executes a GF(p) addition, subtraction, or multiplication operations at a time, while GF(p) division unit executes GF(p) division (a/b modulo p) operation in $2K$ clock cycles. Therefore, the proposed design can execute two GF(p) addition, subtraction, or multiplication operations in parallel to two GF(p) division operations. We grouped these GF(p) arithmetic units into SAU1 and SAU2 units. Each SAU1 and SAU2 consists of one GF(p) unified Add/Sub/Mul unit, one GF(p) divider

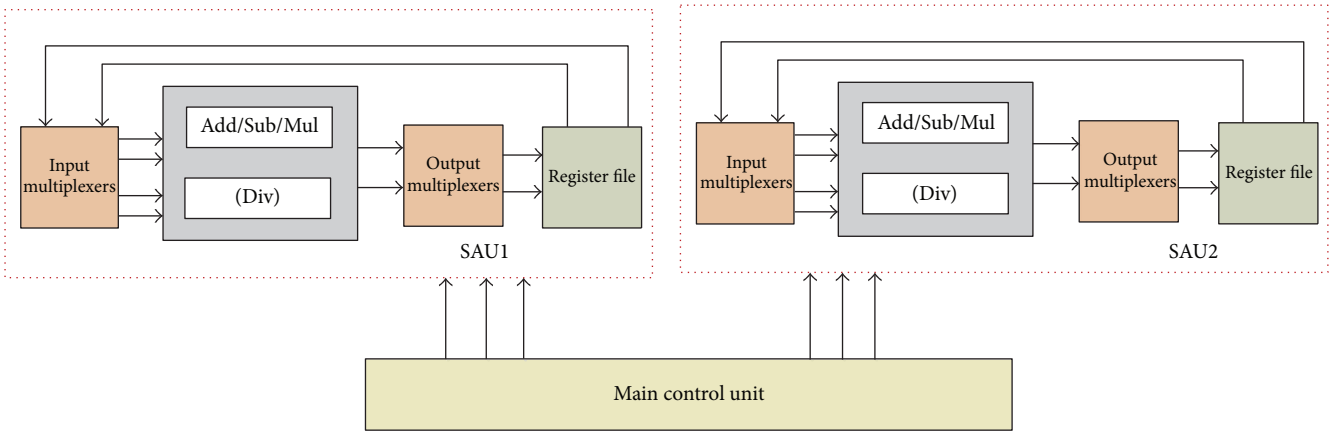

Figure 3: Proposed EC scalar multiplier architecture.

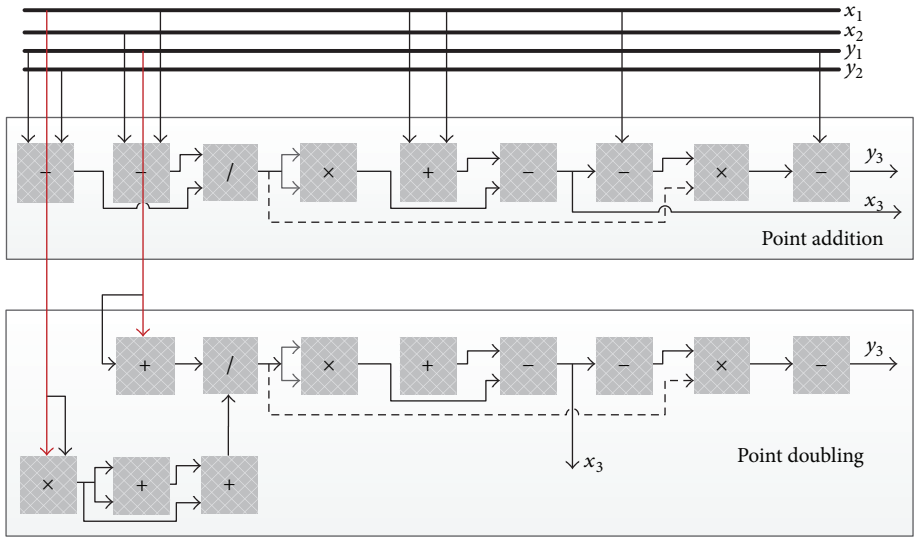

Figure 4: Scheduling of point operations.

unit, and one register file. The EC point addition operation and EC point doubling operation in Algorithm 1 can be performed in parallel. Therefore, the proposed architecture performs these EC point operations in parallel; however, on the unified Add/Sub/Mul unit, GF(p) addition, subtraction, and multiplication operations can only be performed in a serial fashion. The SAU1 unit is dedicated to perform the EC point addition operation, while the EC point doubling operation is executed by SAU2 unit. The register files store intermediate results during execution of EC point addition and EC point doubling operations based on control signals generated and managed by the main controller.

4.1. Scheduling Strategy. A scheduling policy to compute EC point addition and EC point doubling operations on the proposed SAU1 and SAU2 units is shown in Figure 4, where GF(p) addition, subtraction, multiplication, and division operations are denoted as +, −, ×, and /, respectively. Coordinates of two input points P_1, P_2 are denoted by x_1, x_2, y_1, y_2, while resultant point coordinates are shown as x_3, y_3. The results of + and − operations are available after one clock cycle, whereas × and / operations are completed in $\lceil K/3 \rceil + 2$ and $2K$ clock cycles, respectively. The register transfer logic of EC point addition and EC point doubling operations on SAU1 and SAU2 units can be analyzed using Figure 4 and Table 2. Initially registers R_{x_1}, R_{y_1}, R_{x_2}, and R_{y_2} are loaded with coordinates of EC input points P and $2P$, while register R_a is initialized with the EC parameter a. The computation of EC point addition on the proposed SAU1 unit is completed in $(11 + 8\lceil K/3 \rceil)$ clock cycles, whereas SAU2 unit takes $(15 + 3K)$ number of clock cycles to execute EC point doubling operation. Therefore, a single iteration of Algorithm 1 is completed in $(15 + 3K)$ clock cycles and registers R_{x_1}, R_{y_1}, R_{x_2}, and R_{y_2} are updated with new values of EC point addition and EC point doubling. Let K_n be the total number of required clock cycles to compute the EC point multiplication operation; then on the proposed architecture it can be estimated as

$$K_n = \left(\log_2 (s-1)\right)(15 + 3K). \quad (4)$$

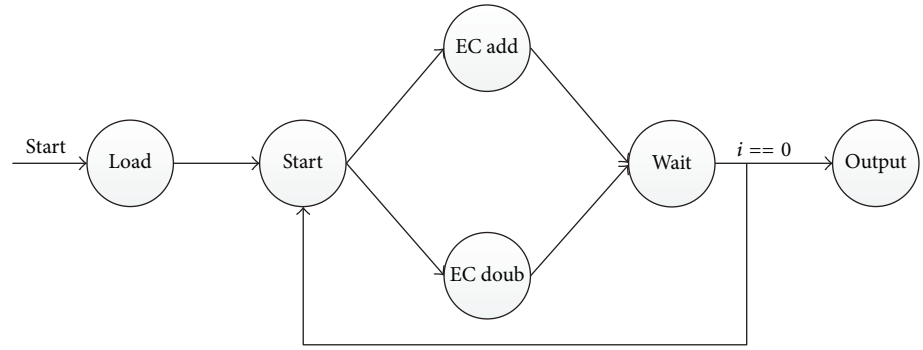

Figure 5: State diagram for main controller.

Table 2: Sequences of GF(p) operations for EC point operations.

GF(p) operation	EC point addition	#cc	GF(p) operation	EC point doubling	#cc
Subtraction	$f_1 = y_2 - y_1$	1	Multiplication	$f_1 = x_1 \times x_1$	1
Subtraction	$f_2 = x_2 - x_1$	2	Addition	$f_2 = f_1 + f_1$	$\lceil K/3 \rceil + 3$
Division	$f_1 = f_1/f_2$	3	Addition	$f_1 = f_1 + f_2$	$\lceil K/3 \rceil + 4$
Multiplication	$f_2 = f_1 \times f_1$	$2K + 4$	Addition	$f_1 = f_1 + a$	$\lceil K/3 \rceil + 5$
Subtraction	$f_3 = x_1 - x_2$	$7\lceil K/3 \rceil + 6$	Addition	$f_2 = y_1 + y_1$	$\lceil K/3 \rceil + 6$
Subtraction	$f_3 = f_2 - f_3$	$7\lceil K/3 \rceil + 7$	Division	$f_1 = f_1/f_2$	$\lceil K/3 \rceil + 7$
Subtraction	$f_2 = x_1 - f_3$	$7\lceil K/3 \rceil + 8$	Multiplication	$f_2 = f_1 \times f_1$	$7\lceil K/3 \rceil + 8$
Multiplication	$f_1 = f_1 \times f_2$	$7\lceil K/3 \rceil + 9$	Addition	$f_3 = x_1 + x_1$	$8\lceil K/3 \rceil + 10$
Subtraction	$f_2 = f_1 - y_1$	$8\lceil K/3 \rceil + 11$	Subtraction	$f_3 = f_2 - f_3$	$8\lceil K/3 \rceil + 11$
			Subtraction	$f_2 = x_1 - f_3$	$8\lceil K/3 \rceil + 12$
			Multiplication	$f_1 = f_1 \times f_2$	$8\lceil K/3 \rceil + 13$
			Subtraction	$f_1 = f_1 - y_1$	$3K + 15$

4.2. Main Controller Logic. The main controller is shown in Figure 5; it is based on a finite state machine (FSM) logic comprised of six states. The control unit is responsible for generating appropriate control signals required to execute EC point addition and EC point doubling on the proposed SAU1 and SAU2 units according to the scheduling strategy shown in Figure 4. It waits for the respective done signals, checks the ith bit of scalar s, and either decides to update the register files with new values or outputs the result and stops execution.

5. Implementation Results and Discussion

The elliptic curve scalar multiplier architecture presented in the previous section has been implemented in Verilog HDL. For simulation, synthesis, mapping, and routing purposes Xilinx ISE 9.1 design suite has been used.

Table 3, shows the performance of the proposed architecture for 160, 192, 224, and 256 bits field sizes on several different FPGA platforms. It takes 3.2 ms, 2.3 ms, and 1.4 ms while running at a maximum frequency of 35 MHz, 48 MHz, and 81 MHz for 192-bit implementation on Virtex-II pro, Virtex-4, and Virtex-6 FPGA platforms, respectively. As, ISE 9.1 design suit does not have a support for Virtex-6 FPGA, so implementation on Virtex-6 FPGA has been done using Xilinx ISE 14.7.

For 192-bit field size our implementation on Virtex-4 computes a single EC scalar multiplication in 2.3 ms in 113,472 clock cycles running at a maximum frequency of 48 MHz. The 192-bit implementation consumes 8,500 slices of Virtex-4 FPGA and has a throughput of 83.5 Kbps. The same design on Virtex-II pro takes 3.2 ms at a maximum frequency of 35 MHz and it uses 7,930 slices. Performance comparison among the proposed architecture and other FPGA implementations is analyzed on the basis of clock cycles, computation time, frequency, occupied FPGA slices, and throughput (TP).

Table 4 shows the required number of clock cycles to compute the EC scalar multiplication operation. The proposed design computes EC point addition and EC point doubling operations in $(11 + (8\lceil K/3 \rceil))$ and $(15 + 3K)$ clock cycles, respectively. As in the proposed design EC point operations are executed concurrently; therefore a single iteration of Algorithm 1 is completed in $(15+3K)$ clock cycles. The designs reported in [21] take $(13+5K)$ clock cycles, which is almost 40% more than the proposed design. Similarly, [18, 24–26] require 48%, 179%, 85%, and 62% more clock cycles to perform the EC scalar multiplication, respectively.

Table 5 demonstrates performance analysis of the several existing FPGA based implementations of EC scalar multiplier. The design reported in [21] is based on parallel dedicated hardware units for GF(p) addition, subtraction,

Table 3: Performance evaluation on different FPGA platforms.

Field size	Virtex-II pro		Virtex-4		Virtex-6	
	Freq (MHz)	Time (ms)	Freq (MHz)	Time (ms)	Freq (MHz)	Time (ms)
160	40	1.9	53	1.4	86	0.9
192	35	3.2	48	2.3	81	1.4
224	31	4.9	43	3.5	76	2
256	27	7.4	40	5	70	2.8

Table 4: Clock cycles requirements for different designs.

Design	Field size	Point addition	Point doubling	EC point multiplication
This work	160	437	495	79,200
	192	523	591	113,472
[21]	160		814	130,240
	192		974	187,008
[25]	160	868	668	153,000
[18]	160	809	972	283,000
[24]	167	2120	2540	545,040
[26]	192	—	—	300,000
[22]	192	—	—	120,000

Table 5: Performance comparison with exiting FPGA implementations.

Design	Field size	Platform	Area (slices)	Freq (MHz)	Time (ms)	TP (Kbps)
This work	160	Virtex-4	7,088	53	1.4	114
	192		8,590	48	2.3	83.5
	224		10,800	43	3.5	64
	256		13,158	40	5	51
This work	160	Virtex-II pro	6,492	40	1.9	84
	192		7,930	35	3.2	60
	224		9,308	31	4.9	45
	256		11,104	27	7.4	34
[21]	160	Virtex-4	12,415	60	2.2	72
	192		14,858	53	3.5	55
	224		17,331	47	5.4	41
	256		20,123	43	7.7	33
[18]	160	Virtex-II	3,433	40	7.1	22
	192		4,135	35	11.6	16.5
	224		4,808	33	16.8	13.3
[16]	192	Virtex-II pro	20,793	49	7.24	26
[32]	192	Virtex-II pro	3,173	93	9.90	19.3
[33]	256	Virtex-II pro	15,775	39	5.99	42.7
[20]	192	Virtex-II pro	8,972	43	4.47	42
	224		10,386	40	6.50	34
	256		11,953	36	9.38	27.2

multiplication, and division. It computes a 192-bit EC scalar multiplication in 3.5 ms at a maximum frequency of 53 MHz on the Virtex-4 platform. On the same platform the proposed design is 34% faster and requires 39% fewer clock cycles with 40% lower FPGA slice consumption as compared to [21]. The proposed design completes a 160-bit EC point multiplication in 79,200 clock cycles at a maximum frequency of 40 MHz. It consumes 6,492 Virtex-II pro FPGA slices. Embedded multicore design reported in [32] computes 192-bit EC scalar multiplication in 9.9 ms running at a maximum frequency of 93 MHz and consumed 3,173 Virtex-II pro FPGA slices. It also uses 6 block BRAMs (BRAM) and sixteen 18×18-bit embedded multipliers. Compared to our design, it is 210% slower, but it consumes 149% fewer FPGA slices if we ignore

the slices for BRAM and dedicated embedded multipliers. The design presented in [33] consumes 15,775 slices and takes 5.99 ms to compute one EC scalar multiplication. On the same platform it is 25% faster but it consumes 28% more FPGA slices. The design proposed by Daly et al. in [18] is 262% slower but it consumes 47% lower slices. The design reported in [20] is 40% slower and consumes 13% more FPGA slices as compared to the proposed design.

Performance comparison on the basis of throughput rate is depicted in the last column of Table 5. The proposed design has 0.5 times, 2.64 times, 1.30 times, 2.1 times, and 0.42 times higher throughput rate as compared to the designs [21], [18], [16], [32], and [20], respectively. The design [33] has 1.25 times higher throughput rate as compared to our design; however, it consumes 1.42 times more FPGA slices. Therefore, our design is better in terms of the computation time, slice area, and throughput rate as compared to all the designs listed in Table 5. As the proposed design executed EC point addition and EC point doubling operations concurrently in a fixed amount of time $(15 + 3K)$, therefore, it provides a protection against the timing and simple power analysis attacks, which is an important feature in modern day security applications. Due to the lower computation time and high throughput rate it is suitable for network applications like *SSL* and *IPsec*. It is also suitable in the low power resource-constrained environments because of the smaller area and reduced clock cycles.

6. Conclusion

This paper first introduces a unified arithmetic architecture for GF(p) addition, subtraction, and multiplication operations. Then, a high speed elliptic curve scalar multiplier is developed on the basis of the unified arithmetic architecture. The proposed design has been synthesized using Xilinx ISE 9.1 and 14.2 Design Suites targeting various Xilinx FPGA devices. Performance is shown for 160-, 192-, 224-, and 256-bit elliptic curve scalar multiplication operation. Compared with other contemporary designs, it gives 34% and 40% better performance in terms of computation time and number of required clock cycles, respectively. It is programmable for any value of prime p and is also resilient to timing and simple power analysis attacks. Therefore, it is a good choice in ECC based cryptosystems.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgments

This work is supported by the Telecommunication Graduate Initiative (TGI) scheme funded by the Higher Education Authority (HEA), Ireland.

References

[1] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology—CRYPTO '85 Proceedings*, pp. 417–426, Springer, 1986.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the Association for Computing Machinery*, vol. 21, no. 2, pp. 120–126, 1978.

[4] IEEE standard specifications for Public, "Key cryptography," IEEE Standards 1363-2000, 2000.

[5] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology—CRYPTO '96*, Lecture Notes in Computer Science, pp. 104–113, Springer, Berlin, Germany, 1996.

[6] F.-X. Standaert, "Introduction to side-channel attacks," in *Secure Integrated Circuits and Systems*, I. M. R. Verbauwhede, Ed., Integrated Circuits and Systems, pp. 27–42, Springer, Berlin, Germany, 2010.

[7] J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, "State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures," in *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST '10)*, pp. 76–87, Anaheim, Calif, USA, June 2010.

[8] J. Fan and I. Verbauwhede, "An updated survey on secure ECC implementations: attacks, countermeasures and cost," in *Cryptography and Security: From Theory to Applications*, pp. 265–282, Springer, 2012.

[9] H. Eberle, N. Gura, C. Sheueling, and V. Gupta, "A cryptographic processor for arbitrary elliptic curves over GF(2m)," *International Journal of Embedded Systems*, vol. 3, no. 4, pp. 241–255, 2008.

[10] J. Lutz and A. Hasan, "High performance FPGA based elliptic curve cryptographic co-processor," in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC '04)*, vol. 2, pp. 486–492, Las Vegas, NV, USA, April 2004.

[11] N. Mentens, S. B. Ors, and B. Preneel, "An FPGA implementation of an elliptic curve processor over GF(2^m)," in *Proceedings of the 14th ACM Great Lakes Symposium on VLSI (GLSVLSI '04)*, pp. 454–457, ACM, Boston, Mass, USA, April 2004.

[12] 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 2000*, Ç. K. Koç and C. Paar, Eds., vol. 1965 of *Lecture Notes in Computer Science*, pp. 25–40, Springer, Berlin, Germany, 2000.

[13] G. Orlando and C. Paar, "A high performance reconfigurable elliptic curve processor for GF (2m)," in *Cryptographic Hardware and Embedded Systems—CHES 2000: Second International Workshop Worcester, MA, USA, August 17-18, 2000 Proceedings*, vol. 1965 of *Lecture Notes in Computer Science*, pp. 41–56, Springer, Berlin, Germany, 2000.

[14] N. A. Saqib, F. Rodríguez-Henriquez, and A. Díaz-Pérez, "A parallel architecture for fast computation of elliptic curve scalar multiplication over GF (2 m)," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 1967–1974, April 2004.

[15] G. Meurice de Dormale and J.-J. Quisquater, "High-speed hardware implementations of elliptic curve cryptography: a survey," *Journal of Systems Architecture*, vol. 53, no. 2-3, pp. 72–84, 2007.

[16] K. Ananyi, H. Alrimeih, and D. Rakhmatov, "Flexible hardware processor for elliptic curve cryptography over NIST prime

fields," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 8, pp. 1099–1112, 2009.

[17] G. Chen, G. Bai, and H. Chen, "A high-performance elliptic curve cryptographic processor for general curves over GF(p) based on a systolic arithmetic unit," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 5, pp. 412–416, 2007.

[18] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "An FPGA implementation of a GF(p) ALU for encryption processors," *Microprocessors and Microsystems*, vol. 28, no. 5-6, pp. 253–260, 2004.

[19] S. Ghosh, M. Alam, I. S. Gupta, and D. R. Chowdhury, "A robust GF(p) parallel arithmetic unit for public key cryptography," in *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD '07)*, pp. 109–115, Lübeck, Germany, August 2007.

[20] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "Petrel: power and timing attack resistant elliptic curve scalar multiplier based on programmable GF(p) arithmetic unit," *IEEE Transactions on Circuits and Systems. I. Regular Papers*, vol. 58, no. 8, pp. 1798–1812, 2011.

[21] S. Ghosh, M. Alam, D. R. Chowdhury, and I. S. Gupta, "Parallel crypto-devices for GF(p) elliptic curve multiplication resistant against side channel attacks," *Computers and Electrical Engineering*, vol. 35, no. 2, pp. 329–338, 2009.

[22] G. Orlando and C. Paar, "A scalable GF(p) elliptic curve processor architecture for programmable hardware," in *Cryptographic Hardware and Embedded Systems—CHES 2001*, pp. 348–363, Springer, Berlin, Germany, 2001.

[23] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of an elliptic curve processor over GF(p)," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP '03)*, pp. 433–443, IEEE, June 2003.

[24] E. Öztürk, B. Sunar, and E. Savaş, "Low-power elliptic curve cryptography using scaled modular arithmetic," in *Cryptographic Hardware and Embedded Systems—CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11–13, 2004. Proceedings*, vol. 3156 of *Lecture Notes in Computer Science*, pp. 92–106, Springer, Berlin, Germany, 2004.

[25] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 449–460, 2003.

[26] W. Shuhua and Z. Yuefei, "A timing-and-area tradeoff GF(p) elliptic curve processor architecture for FPGA," in *Proceedings of the International Conference on Communications, Circuits and Systems*, vol. 2, pp. 1308–1312, May 2005.

[27] G. R. Blakley, "A computer algorithm for calculating the product AB modulo M," *IEEE Transactions on Computers*, vol. 32, no. 5, pp. 497–500, 1983.

[28] K. Javeed and X. Wang, "Efficient montgomery multiplier for pairing and elliptic curve based cryptography," in *Proceedings of the 9th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP '14)*, pp. 255–260, Manchester, UK, July 2014.

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

[30] K. Javeed and X. Wang, "Radix-4 and radix-8 booth encoded interleaved modular multipliers over general Fp," in *Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL '14)*, pp. 1–6, September 2014.

[31] K. Javeed, X. Wang, and M. Scott, "Serial and parallel interleaved modular multipliers on FPGA platform," in *Proceedings of the 25th International Conference on Field Programmable Logic and Applications (FPL '15)*, pp. 1–4, London, UK, September 2015.

[32] J. Fan, K. Sakiyama, and I. Verbauwhede, "Elliptic curve cryptography on embedded multicore systems," *Design Automation for Embedded Systems*, vol. 12, no. 3, pp. 231–242, 2008.

[33] C. J. McIvor, M. McLoone, and J. V. McCanny, "Hardware elliptic curve cryptographic processor over GF(p)," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 9, pp. 1946–1957, 2006.

[34] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "Fast modular division for application in ECC on reconfigurable logic," in *Field Programmable Logic and Application*, pp. 786–795, Springer, Berlin, Germany, 2003.