A High-Speed FPGA Implementation of an RSD-Based ECC Processor

Hamad Marzouqi, *Member, IEEE*, Mahmoud Al-Qutayri, *Senior Member, IEEE*, Khaled Salah, *Senior Member, IEEE*, Dimitrios Schinianakis, *Member, IEEE*, and Thanos Stouraitis, *Fellow, IEEE*

Abstract— **In this paper, an exportable application-specific instruction-set elliptic curve cryptography processor based on redundant signed digit representation is proposed. The processor employs extensive pipelining techniques for Karatsuba–Ofman method to achieve high throughput multiplication. Furthermore, an efficient modular adder without comparison and a high-throughput modular divider, which results in a short datapath for maximized frequency, are implemented. The processor supports the recommended NIST curve P256 and is based on an extended NIST reduction scheme. The proposed processor performs single-point multiplication employing points in affine coordinates in 2.26 ms and runs at a maximum frequency of 160 MHz in Xilinx Virtex 5 (XC5VLX110T) field-programmable gate array.**

Index Terms— **Application-specific instruction-set processor (ASIP), elliptic curve cryptography (ECC), field-programmable gate array (FPGA), Karatsuba–Ofman multiplication, redundant signed digit (RSD).**

I. Introduction

Elliptic curve cryptography (ECC) [1] is an asymmetric cryptographic system that provides an equivalent security to the well-known Rivest, Shamir and Adleman system with much smaller key sizes [2]. The basic operation in ECC is scalar point multiplication, where a point on the curve is multiplied by a scalar. A scalar point multiplication is performed by calculating series of point additions and point doublings. Using their geometrical properties, points are added or doubled through series of additions, subtractions, multiplications, and divisions of their respective coordinates. Point coordinates are the elements of finite fields closed under a prime or an irreducible polynomial. Various ECC processors have been proposed in the literature that either target binary fields [3], [4], prime fields [5]–[7], or dual field operations [8], [9].

In prime field ECC processors, carry free arithmetic is necessary to avoid lengthy datapaths caused by carry propagation. Redundant schemes, such as carry save arithmetic (CSA) [10], [11], redundant signed digits (RSDs) [12], or residue number systems (RNSs) [7], [13], have been utilized in various designs. Carry logic or embedded digital signal processing (DSP) blocks within field-programmable gate arrays (FPGAs) are also utilized in some designs to address the carry propagation problem [14], [15]. It is necessary to build an efficient addition datapath since it is a fundamental operation employed in other modular arithmetic operations.

Modular multiplication is an essential operation in ECC. Two main approaches may be employed. The first is known as interleaved modular multiplication using Montgomery's method [16]. Montgomery multiplication is widely used in implementations where arbitrary curves are desired [17], [18]. Another approach is known as multiply-then-reduce and is used in elliptic curves built over finite fields of Merssene primes [19]. Merssene primes are the special type of primes which allow for efficient modular reduction through series of additions and subtractions [5], [20]. In order to optimize the multiplication process, some ECC processors use the divide and conquer approach of Karatsuba–Ofman multiplications [21], where others use embedded multipliers and DSP blocks within FPGA fabrics [22]–[24].

Since modular division in affine coordinates is a costly process, numerous coordinate representation systems have been proposed to compensate this cost by means of extra multiplications and additions (e.g., Jacobian coordinates) [6], [24]. Conversion back to affine representation can be mechanized using Fermat's little theorem [11], [25]. Such processors may implement a dedicated squarer to speed up the inversion process [5]. On the other hand, binary GCD modular division algorithm [26] is utilized in many ECC processors where affine coordinate system is used. Binary GCD algorithm is based on simple add and shift operations, while the same operations are used by Montgomery multiplication. Hence, many ECC processors with combined modular division and multiplication blocks have been proposed [27], [28]. The complexity of modular division algorithms is approximately $\mathcal{O}(2n)$, where n is the size of operands and the running time is variable and depends directly on the inputs.

This paper proposes a new RSD-based prime field ECC processor with high-speed operating frequency. The processor is an application-specific instruction-set processor (ASIP) type to provide programmability and configurability. In this paper, we demonstrate the performance of left-to-right scalar point multiplication algorithm; however, the ASIP feature of the processor allows different algorithms to be performed by the through

Manuscript received July 6, 2014; revised October 13, 2014 and December 28, 2014; accepted January 2, 2015. Date of publication January 29, 2015; date of current version December 24, 2015.

H. Marzouqi, M. Al-Qutayri, and K. Salah are with the Department of Electrical and Computer Engineering, Khalifa University, Abu Dhabi 52799, United Arab Emirates (e-mail: hamad.marzouqi@kustar.ac.ae; mqutayri@kustar.ac.ae; khaled.salah@kustar.ac.ae).

D. Schinianakis and T. Stouraitis are with Department of Electrica and Computer Engineering, University of Patras, Patras 265 04, Greece (e-mail: dsxoiniana@upatras.gr; thanos@upatras.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2015.2391274

read-only memory (ROM) programming. The overall processor architecture is of regular cross bar type with 256 digit wide data buses. The design strategy and optimization techniques are focused toward efficient individual modular arithmetic modules rather than the overall architecture. Such architecture allows for easy replacement of individual blocks if different algorithms or modular arithmetic techniques are desired. Different efficient architectures of individual modular arithmetic blocks for various algorithms are proposed. The novelty of our processor evolves around the following.

1) We introduce the first FPGA implementation of RSD-based ECC processor.
2) Extensive pipelining and optimization strategies are used to obtain a high-throughput iterative Karatsuba multiplier which lead to a performance improvement of almost 100% over the processor proposed in [29].
3) To the best of our knowledge, the proposed modular division/inversion is the fastest to be performed on FPGA device. This is done through a new efficient binary GCD divider architecture based on simple logical operations.
4) A modular addition and subtraction is proposed without comparison.
5) Most importantly, exportable design is proposed with specifically designed multipliers and carry free adders that provided in competitive results against DSPs and embedded multipliers-based designs.

The remaining of this paper is organized as follows. Section II provides background information on ECC systems as well as other algorithms and approaches for modular arithmetic. Section III presents the overall architecture of the proposed processor. The architecture of the modular arithmetic unit (AU) is presented in Section IV. Control unit and instruction set are provided in Section V, while implementation and results analysis are presented in Section VI. Finally, the conclusion is drawn in Section VII.

II. Background

A. Elliptic Curve Cryptography

Elliptic curves [30] over a field K are defined by the reduced Weierstrass equation in (1) when the characteristic of the field is two or three. The set of solutions along with a point at infinity $\mathcal{O}$ defines the algebraic structure as a group with point addition as the basic operation

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

The smoothness of the curve and distinct roots are guaranteed by $4a^3 + 27b^2 \neq 0$. Points on the curve are defined by their affine coordinates (x, y). Point coordinates are of type integers for an elliptic curve defined by (1) and are the elements of an underlying finite field with operations performed modulo a prime number. Such elliptic curves are known as prime field elliptic curves.

For prime field elliptic curves defined by (1), the coordinates of the point addition result is calculated as follows, assuming $P = (x_1, y_1)$, $Q = (x_2, y_2)$, and $R = P + Q = (x_3, y_3)$:

Algorithm 1 Left-to-Right Point Multiplication Binary Method

```
Input: A scalar $k = (k_{t-1}, ..., k_1, k_0)$ point $P$
Output: $kP$
1: $Q \leftarrow \emptyset$
2: for $i = t - 1$ downto $0$ do
3:     $Q \leftarrow 2Q$; If $k_i = 1$ then $Q \leftarrow Q + P$
4: end for
5: return $Q$
```

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

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1. \tag{3}$$

Whereas the point doubling operation is calculated as follows:

$$x_3 = \left(\frac{3{x_1}^2 + a}{2y_1}\right)^2 - 2x_1 \tag{4}$$

$$y_3 = \left(\frac{3{x_1}^2 + a}{2y_1}\right)(x_1 - x_3) - y_1. \tag{5}$$

1) Point Scalar Multiplication: Point scalar multiplication is the operation of multiplying a point P on the elliptic curve by an integer scalar k within the underlying field. The operation is performed as k-times addition of the point P to itself. A discrete logarithm problem is formulated based on the scalar point multiplication and several cryptographic protocols and algorithms have been established accordingly [31].

Several algorithms have been proposed to perform the scalar point multiplication that are either based on the direction of the scalar scanning or on the representation of the scalar [30]. Algorithm 1 is based on the square-and-multiply method for the exponentiation, where the exponent is scanned from left-to-right and the operations of squaring and/or multiplication are performed according to the binary value of the scanned bit. Similarly, a right-to-left point multiplication algorithm exists that differs in the direction of exponent scanning. The operations of squaring and multiplication are replaced by point doubling and point addition, respectively.

B. Redundant Signed Digits

The RSD representation, first introduced by Avizienis [32], is a carry free arithmetic where integers are represented by the difference of two other integers. An integer X is represented by the difference of its x^+ and x^- components, where x^+ is the positive component and x^- is the negative component. The nature of the RSD representation has the advantage of performing addition and subtraction without the need of the two's complement representation. On the other hand, an overhead is introduced due to the redundancy in the integer representation, since an integer in RSD representation requires double wordlength compared with typical two's complement representation. In radix-2 balanced RSD represented integers, digits of such integers are either 1, 0, or -1.

C. Karatsuba–Ofman Multiplication

The complexity of the regular multiplication using the schoolbook method is $\mathcal{O}(n^2)$. Karatsuba and Ofman [33] proposed a methodology to perform a multiplication with complexity $\mathcal{O}(n^{1.58})$ by dividing the operands of the multiplication into smaller and equal segments. Having two operands of length n to be multiplied, the Karatsuba–Ofman methodology suggests to split the two operands into high-(H) and low-(L) segments as follows [33]:

$$\begin{aligned} a_H &= (a_{n-1}, \ldots, a_{\lceil n/2 \rceil}), \quad a_L = (a_{\lceil n/2 \rceil - 1}, \ldots, a_0) \\ b_H &= (b_{n-1}, \ldots, b_{\lceil n/2 \rceil}), \quad b_L = (b_{\lceil n/2 \rceil - 1}, \ldots, b_0). \end{aligned}$$

Consider β as the base for the operands, where β is 2 in case of integers and β is x in case of polynomials. Then, the multiplication of both operands is performed as follows: considering $a = a_L + a_H \beta^{\lceil n/2 \rceil}$ and $b = b_L + b_H \beta^{\lceil n/2 \rceil}$ then

$$\begin{aligned} C = AB &= (a_L + a_H \beta^{\lceil n/2 \rceil})(b_L + b_H \beta^{\lceil n/2 \rceil}) \\ &= a_L b_L + (a_L b_H + a_H b_L)\beta^{\lceil n/2 \rceil} + a_H b_H \beta^n. \end{aligned} \quad (6)$$

Hence, four half-sized multiplications are needed, where Karatsuba methodology reformulate (6) to

$$\begin{aligned} C = AB &= (a_L + a_H \beta^{\lceil n/2 \rceil})(b_L + b_H \beta^{\lceil n/2 \rceil}) \\ &= a_L b_L \\ &\quad + ((a_L + a_H)(b_L + b_H) - a_H b_H - a_L b_L)\beta^{\lceil n/2 \rceil} \\ &\quad + a_H b_H \beta^n. \end{aligned} \quad (7)$$

Therefore, only three half-sized multiplications are needed. The original Karatsuba algorithm is performed recursively, where the operands are segmented into smaller parts until a reasonable size is reached, and then regular multiplications of the smaller segments are performed recursively.

D. Binary GCD Modular Division

A modular division algorithm is proposed in [26] based on the extended Euclidean algorithm. This algorithm is considered as the basis for several hardware implementations of modular division [28], [34], [35]. Algorithm 2 computes the modular division $Z \equiv X/Y \pmod{M}$ based on the plus–minus version of the original binary GCD algorithm. The algorithm instantiates the four registers A, B, U, and V that are initialized with Y, M, X, and 0, respectively. Then, it constantly reduces the values of Y and M in order to calculate the GCD(Y, M) which is equal to 1 in well formed elliptic curves where the modulo is prime. The registers U and V are used to calculate the quotient and the operations performed on these registers are similar to the operations performed on the A and B registers. The operations on the registers A and B are performed by repetitively reducing the contents of both registers by simple shift or add/subtract-shift operations based on the conditions whether the intermediate contents are even or not. In the case where both registers contents are odd, the content of both registers are added if $A+B$ is divisible by 4 or subtracted, $(A-B)$, otherwise. Two variables ρ and δ are used to control the iterations of the algorithm based on the bounds of the registers contents, where $\delta = \alpha - \beta$, 2^α and 2^β are the upper bounds of A and B, respectively, and $\rho = \min(\alpha, \beta)$.

Algorithm 2 Radix-4 Binary GCD Modular Division Algorithm

```
Input: $M : 2^{n-1} < M < 2^n$ and prime; $X, Y : 0 \le X, Y < M$
Output: $Z \equiv X/Y \bmod M$
 1: $A \leftarrow Y$; $B \leftarrow M$; $U \leftarrow X$; $V \leftarrow 0$; $\rho \leftarrow n$; $\delta \leftarrow 0$
 2: while $\rho \ne 0$ do
 3:     while $A \bmod 2 = 0$ do
 4:         if $A \bmod 4 = 0$ then
 5:             $A \leftarrow A/4$; $U \leftarrow U/4 \bmod M$
 6:             $\rho \leftarrow \rho - 2$; $\delta \leftarrow \delta - 2$
 7:         else
 8:             $A \leftarrow A/2$; $U \leftarrow U/2 \bmod M$
 9:             $\rho \leftarrow \rho - 1$; $\delta \leftarrow \delta - 1$
10:         end if
11:     end while
12:     if $\delta < 0$ then
13:         $T \leftarrow A$; $A \leftarrow B$; $B \leftarrow T$
14:         $T \leftarrow U$; $U \leftarrow V$; $V \leftarrow T$
15:         $\delta \leftarrow -\delta$
16:     end if
17:     if $(A + B) \bmod 4 = 0$ then
18:         $q \leftarrow 1$
19:     else
20:         $q \leftarrow -1$
21:     end if
22:     $A \leftarrow (A + qB)/4$; $U \leftarrow (U + qV)/4 \bmod M$
23:     $\rho \leftarrow \rho - 1$; $\delta \leftarrow \delta - 1$
24: end while
25: if $B = 1$ then $Z \leftarrow V$ else $Z \leftarrow M - V$
26: return $Z$
```

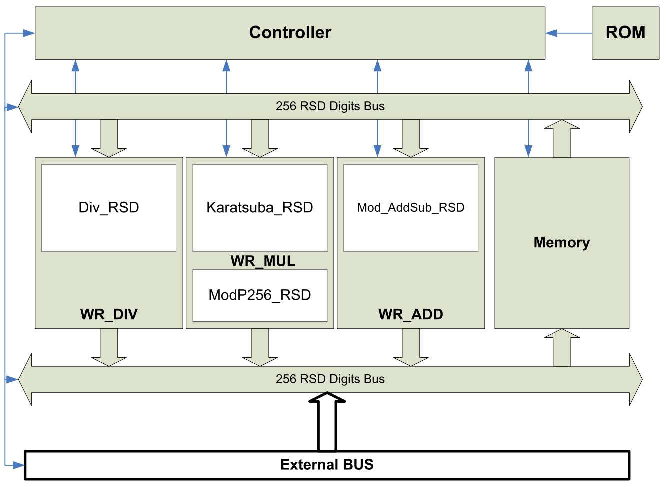


Fig. 1. Overall processor architecture.

III. Overall Processor Architecture

The proposed P256 ECC processor consists of an AU of 256 RSD digits wide, an finite-state machine (FSM), memory, and two data buses. The processor can be configured in the presynthesis phase to support the P192 or P224 NIST recommended prime curves [36]. Fig. 1 shows the overall

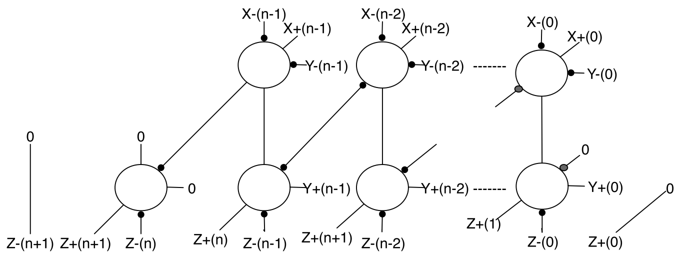


Fig. 2. RSD adder.

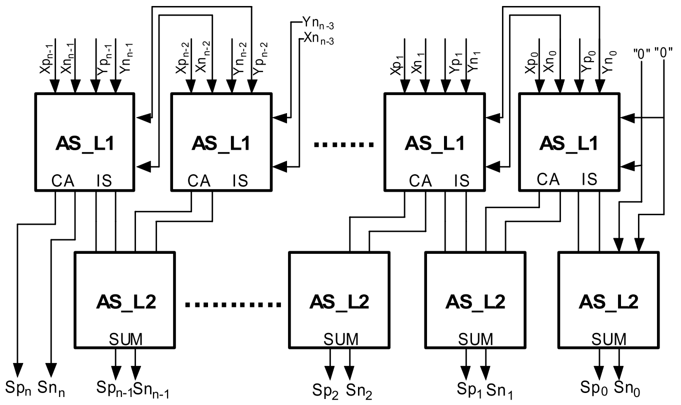


Fig. 3. RSD adder/subtracter.

processor architecture. Two subcontrol units are attached to the main control unit as add-on blocks. These two subcontrol units work as FSMs for point addition and point doubling, respectively. Different coordinate systems are easily supported by adding corresponding subcontrol blocks that operate according to the formulas of the coordinate system.

External data enter the processor through the external bus to the 256 RSD digits input bus. Data are sent in binary format and a binary to RSD converter stuffs zeros in between the binary bits in order to create the RSD representation. Hence, 256-bits binary represented integers are converted to 512-bits RSD represented integers. To convert RSD digits to binary format, one needs to subtract the negative component from the positive component of the RSD digit.

IV. Arithmetic Unit

The AU is the core unit of the processor that includes the following blocks: 1) modular addition/subtraction block; 2) modular multiplication block; and 3) modular division block.

A. Modular Addition and Subtraction

Addition is used in the accumulation process during the multiplication, as well as, in the binary GCD modular divider algorithm.

In the proposed implementation, radix-2 RSD representation system as carry free representation is used. In RSD with radix-2, digits are represented by 0, 1, and -1, where digit 0 is coded with 00, digit 1 is coded with 10, and digit -1 is coded with 01. In Fig. 2 [37], an RSD adder is presented that is built from generalized full adders. The problem with this adder is that it tends to expand the addition result even if there is no overflow, since it restricts the least significant digit (LSD) to be digit -1 only. This unnecessary overflow affects the reduction process later and produces some control complexities in the overall processor architecture. However, the overflow is easily managed when the adder is instantiated as a subblock within a multiplier or a divider as is the case in the proposed implementation.

TABLE I
Addition Rules for the RSD Adder [38]

$a_i b_i$	$a_{i-1} b_{i-1}$	**Carry**	**Interim Sum**
ZZ	–	Z	Z
ZP/PZ	neither is N	P	N
ZP/PZ	at least one is N	Z	P
ZN/NZ	neither is N	Z	N
ZN/NZ	at least one is N	N	P
PP	–	1	Z
NN	–	N	Z
PN/NP	–	Z	Z

In order to overcome the problem of overflow introduced in the adder proposed in [37], a new adder is proposed based on the work proposed in [38]. The proposed adder consists of two layers, where layer 1 generates the carry and the interim sum, and layer 2 generates the sum, as shown in Fig. 3. Table I shows the addition rules that are performed by layer 1 of the RSD adder, where RSD digits 0, $+1$, and -1 are represented by Z, P, and N, respectively. It works by assuring that layer 2 does not generate overflow through the use of previous digits in layer 1. The proposed adder is used as the main block in the modular addition component to take advantage of the reduced overflow feature. However, overflow is not an issue in both the multiplier and the divider when an RSD adder is used as an internal block. Hence, the reduced area is taken as an advantage in instantiating adders within the multiplier and the divider.

The n-digits modular addition is performed by three levels of RSD addition. Level 1 performs the basic addition of the operands which produces $n + 1$ digits as a result. If the most significant digit (MSD) of level 1 output has a value of 1/-1, then level 2 adds/subtracts the modulo P256 from the level 1 output correspondingly. The result of level 2 RSD addition has $n + 2$ digits; however, only the $n + 1$th digit may have a value of 1/-1. This assertion is backed up by the fact that the operation of level 2 is a reversed operation with the modulo P256, and most importantly, the proposed adder assures that no unnecessary overflow is produced. If the $n + 1$th digit of level 2 result has a value 1 or -1, then level 3 is used to reduce the output to the n-digit range. Algorithm 3 shows the sequence of operations performed by the modular addition block. Notice that one modular addition is performed within one, two, or three clock cycles.

Fig. 4 shows the block diagram of the RSD modular addition block. The advantage of the proposed modular addition scheme is that only the MSD digits of the intermediate results are checked for the reduction process, as shown in Fig. 4. Our modular adder/subtracter consists of one full word RSD adder, two full word multiplexers, and one register with some control signals. One modular addition/subtraction is performed within one, two, or three clock cycles as per the value of the

Algorithm 3 RSD Modular Addition/Subtraction

```
Input: $A = a_{n-1}.....a_0$ and $B = b_{n-1}.....b_0$ and $M = m_{n-1}.....m_0$ and $AddSub$ 1-bit
Output: $S = s_{n-1}.....s_0 = X + Y \bmod M$
1: if $AddSub = 0$ then
2:     $T_1 \leftarrow A + B$
3: else
4:     $T_1 \leftarrow A - B$
5: end if
6: $T_2 = \begin{cases} T_1 & T_1[n] = 0 \\ T_1 - M & T_1[n] = 1 \\ T_1 + M & T_1[n] = -1 \end{cases}$
7: $T_3 = \begin{cases} T_2 & T_2[n] = 0 \\ T_2 - M & T_2[n] = 1 \\ T_2 + M & T_2[n] = -1 \end{cases}$
8: return $S \leftarrow T_3$
```

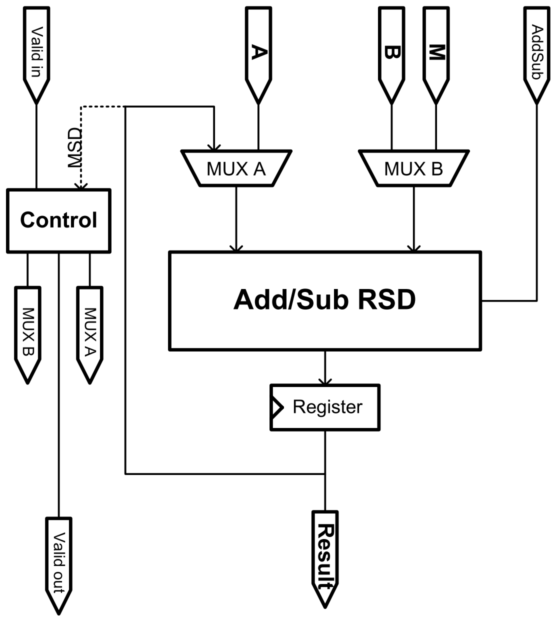


Fig. 4. Modular addition subtraction block diagram.

MSD that is retrieved after every addition. Whenever MSD becomes zero, the modular addition/subtraction module stops the operation and the valid out signal is activated. An $n + 1$ RSD digit does not necessarily yield a value larger than the n-digit P256 modulo. Hence, the output of level 1 is in the range $-2 * \text{P256} < \text{L1} < 2 * \text{P256}$. Level 2 reduces the result to the range $-\text{P256} < \text{L2} < \text{P256}$. Level 3 assures that the output is represented by n-RSD digits in the range $-\text{P256} < \text{L3} < \text{P256}$. Notice that the subtraction is performed by negating operand B which is simply performed by swapping the negative and positive components of the RSD representation of the operand.

B. Modular Multiplication

Karatsuba's multiplier recursive nature is considered a major drawback when implemented in hardware [39]. Hardware complexity increases exponentially with the size of the operands to be multiplied. To overcome this drawback, Karatsuba method is applied at two levels. A recursive Karatsuba block that works depthwise, and an iterative Karatsuba that works widthwise. The proposed method consists of two phases: 1) in phase 1, a regular recursive Karatsuba is built through recursive construction down to 1-digit level and 2) the recursive Karatsuba block is used to perform Karatsuba multiplications iteratively. Hence, three recursive Karatsuba blocks are used to perform single widthwise Karatsuba iteration.

TABLE II
MIDDLE PRODUCT FOR THE RECURSIVE KARATSUBA CONSTRUCTION

C_x	C_y	**Middle Multiplication**
0	0	S_xS_y
0	1	$S_xS_y + 2^nS_x$
0	-1	$S_xS_y - 2^nS_x$
1	0	$S_xS_y + 2^nS_y$
1	1	$S_xS_y + 2^n(S_x + S_y) + 2^{2n}$
1	-1	$S_xS_y + 2^n(S_y - S_x) - 2^{2n}$
-1	0	$S_xS_y - 2^nS_y$
-1	1	$S_xS_y + 2^n(S_x - S_y) - 2^{2n}$
-1	-1	$S_xS_y - 2^n(S_x - S_y) + 2^{2n}$

1) Recursive Construction of Karatsuba Multiplier: In general, the reduced complexity of Karatsuba multiplication comes from the fact that four half-word multiplications are replaced by three half-word multiplications with some additions and subtractions. However, the complexity impact increases with the increase of the recursive depth of the multiplier. Hence, it is not sufficient to divide the operands into halves and apply the Karatsuba method at this level only.

Operands of size n-RSD digits are divided into two (low and high) equal sized $n/2$-RSD digits branches. The low branches are multiplied through an $n/2$ Karatsuba multiplier and the high branches are multiplied through another $n/2$ Karatsuba multiplier. Implementation difficulties arise with the middle Karatsuba multiplier when multiplying the results of addition of the low and high branches of each operand by itself. The results of the addition are of size $n/2 + 1$-RSD digits so that an unbalanced Karatsuba multiplier of size $n/2 + 1$ is required. Hence, the carry generated by the middle addition operation needs to be addressed to avoid implementation complexities of the unbalanced Karatsuba multiplier.

In [40], a method to handle the carry produced by the middle addition is proposed. This is adapted to RSD arithmetic as follows. The $n/2$-digit Karatsuba block is used to multiply the middle summations, excluding the carry. A 1-digit RSD multiplier is used to multiply the carry digits. The cross multiplication is simply performed by checking the carry in the other middle summation. In general, the middle multiplication is calculated as Table II indicates, where S_x and S_y represent the summation without the carry digits and C_x and C_y are the carry digits.

Algorithm 4 represents the recursive construction of the Karatsuba multiplication method at the n-digits level. A recursive call of the Karatsuba multiplication module is performed three times for K_{low}, K_{high}, and K_1. These three multiplications are performed in parallel through three Karatsuba blocks. Each Karatsuba block performs recursively the same

Algorithm 4 Karatsuba (X, Y, n)

```
Input: $X = X_L + X_H 2^{n/2}$ and $Y = Y_L + Y_H 2^{n/2}$
Output: $Z = XY$
 1: $K_{low} = Karatsuba(X_L, Y_L, n/2)$,
 2: $K_{high} = Karatsuba(X_H, Y_H, n/2)$
 3: $S_x = \text{sum}(X_L + X_H)$, $C_x = \text{carry}(X_L + X_H)$
 4: $S_y = \text{sum}(Y_L + Y_H)$, $C_y = \text{carry}(Y_L + Y_H)$
 5: $K_1 = Karatsuba((S_x - C_x 2^{n/2}) * (S_y - C_y 2^{n/2}), n/2)$
 6: $K_2 = C_x * C_y$
 7: $K_{3A} = \begin{cases} (S_x - C_x 2^{n/2}) & C_y = 1 \\ -(S_x - C_x 2^{n/2}) & C_y = -1 \\ 0 & C_y = 0 \end{cases}$
 8: $K_{3B} = \begin{cases} (S_y - C_y 2^{n/2}) & C_x = 1 \\ -(S_y - C_y 2^{n/2}) & C_x = -1 \\ 0 & C_x = 0 \end{cases}$
 9: $K_3 = K_{3A} + K_{3B}$
10: $K_{middle} = K_1 + K_3 2^{n/2} + K_2 2^n$
11: $Z = K_{low} + K_{middle} 2^{n/2} + K_{high} 2^n$
12: return $Z$
```

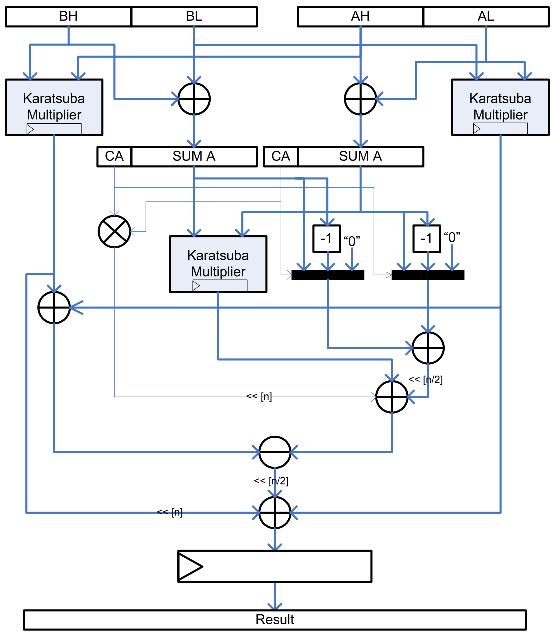


Fig. 5. Karatsuba recursive block.

operations for $n/2$, then for $n/4$, and so forth. The recursive Karatsuba is constructed by recursive generation down to 4-digit of RSD schoolbook multiplier. The use of schoolbook multiplier at the lower level of the multiplier is due to the fact that Karatsuba method produces delays that cannot be compensated at small operand sizes. Hence, the recursive construction of the Karatsuba multiplier should stop at a level where the critical path delay (CPD)-performance tradeoff is advantageous.

The block diagram of the recursive Karatsuba multiplier is shown in Fig. 5, where data dependences are clearly noticed. As shown in Fig. 5, Karatsuba method requires performing

Algorithm 5 RSD Iterative–Recursive Karatsuba Multiplier

```
Input: $X = x_{n-1}.....x_0$ and $Y = y_{n-1}.....y_0$, $k$: recursive block size
Output: $Z = z_{2n-1}.....x_0 = XY$
 1: for $i = 0$ to $n/k$ do
 2:     for $j = 0$ to $n/k$ do
 3:         $K_L = RecursiveKaratsuba_k(X_i, Y_j)$
 4:         $K_H = RecursiveKaratsuba_k(X_{i+1}, Y_{j+1})$
 5:         $K_M = RecursiveKaratsuba_k((X_i + X_{i+1}), (Y_j + Y_{j+1}))$
 6:         $PP = K_L + (K_M - K_L - K_H)2^k + K_H 2^{2k}$
 7:         $Z = Accumulate(Z, PP)$
 8:     end for
 9: end for
10: return $Z$
```

a subtraction at every level, which is an advantage of the proposed implementation since subtraction is performed with no added cost in RSD representation. The block diagram of the recursive Karatsuba module is built from three half-sized recursive Karatsuba blocks and some RSD adders/subtracters. There is one 1-digit RSD multiplier that is used to multiply the carry digits from the middle addition. The cross multiplication of the middle addition is performed through multiplexers with the carry digits as the select signals according to Table II. According to Fig. 5, the critical datapath of the recursive Karatsuba is divided into two paths. The first path goes through the middle half-sized recursive Karatsuba block, and the other goes through the cross product of the middle addition with multiplexers and some adders. Any optimizations attempts should focus at these two datapaths, which is discussed in more details in Section IV-B3.

2) Iterative Karatsuba Multiplier: The recursive construction of the in-depth Karatsuba can be performed to 256 digits; however, the hardware complexity would increase exponentially. Hence, an iterative version is introduced to reduce the complexity at the expense of extra clock cycles. A recursive in-depth Karatsuba block is built of size k, where $k = 2^i$. The value of i is determined according to design tradeoff of hardware versus clock cycles. The full-sized multiplicands (size n) are handled as words of size k. At each iteration, two words of each operands are multiplied using Karatsuba method. The cost of multiplying two $2k$-operands using multiplier of size k requires four full k-multiplications along with the accumulation process, as shown in Algorithm 5. Using Karatsuba at this level, the $4k$-multiplications are compensated for $3k$-multiplications with some additions and subtraction operations. Meanwhile, the k-multiplier by itself is a Karatsuba-based multiplier.

3) Optimization and Pipelining Techniques: The CPD of the processor is dominated by the Karatsuba multiplier datapath. Our 32-digit Karatsuba multiplier datapath consists of cascaded adders along with the recursive datapath of the 16-digit Karatsuba. The number of cascaded adders defines the datapath of the multiplier. There are seven adders in the multiplier datapath as shown in the following equations,

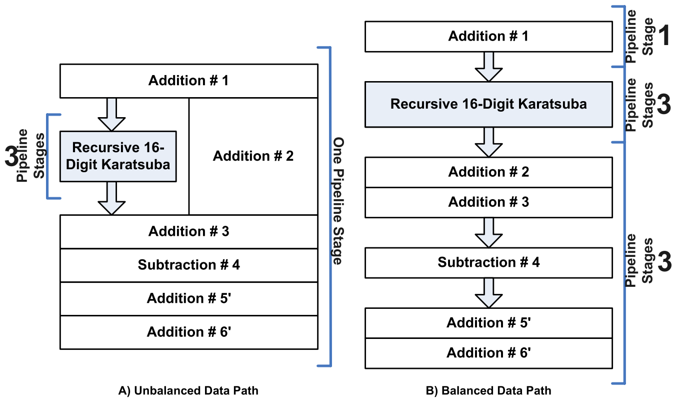


Fig. 6. Karatsuba multiplier datapath optimization. (A) Unbalanced datapath. (B) Balanced datapath.

where A and B are RSD digits of size k:

$$
\begin{aligned}
1)\ A_{\text{sum}} &= A_H 2^{k/2} + A_L\\
B_{\text{sum}} &= B_H 2^{k/2} + B_L\\
C &= A_{\text{sum}} 2^k * B_{\text{sum}} 2^k\\
K_{3A} &= \text{Karatsuba}_{16}(A_{\text{sum}}, B_{\text{sum}})\\
K_1 &= \text{Karatsuba}_{16}(A_L, B_L)\\
K_2 &= \text{Karatsuba}_{16}(A_H, B_H)\\
2)\ K_{3B} &= A'_{\text{sum}} + B'_{\text{sum}}\\
3)\ K_{3C} &= K_{3A} + K_{3B} 2^{k/2}\\
4)\ K_3 &= K_{3C} + C2^k\\
M_1 &= K_1 + K_2\\
5)\ M_2 &= K_3 - M_1\\
6)\ S' &= K_1 + K_2 2^{2k}\\
7)\ S &= S' + M_2 2^k.
\end{aligned}
$$

Numbered equations represent the additions that contribute to the multiplier's datapath. To reduce path delays and increase maximum operating frequency of the system, two approaches have been followed. First, the number of additions within the CPD is reduced. Second, a highly pipelined system is introduced by careful placement of registers within the datapath. By working around addition numbers 4, 6, and 7, the number of RSD additions within the datapath could be reduced to six additions only. The addition of the carry C in (4) is delayed until the end of the process. Also, the addition in (6) is only a cascade operation when the extra digit at position $2k$ of operand K_1 is removed. Hence, the carry C and the extra digit of K_1 from addition (6) are cascaded for a final addition at once. Therefore, additions (4), (6), and (7) are replaced by additions (5′) and (6′) as follows, where, operation number (5) become operation number (4) and || represents concatenation

$$
\begin{aligned}
5')\ S' &= (K_1 - K_1[2k] || K_2 2^{2k}) + M_2 2^k\\
6')\ S &= S' + (C2^{2k} || K_1[2k] 2^k).
\end{aligned}
$$

Fig. 6 shows the datapath of the multiplier. Six adders represent the datapath along with the recursive datapath for the 16-digit Karatsuba multiplier blocks. The 16-digit recursive Karatsuba block is a three stages pipelined datapath and

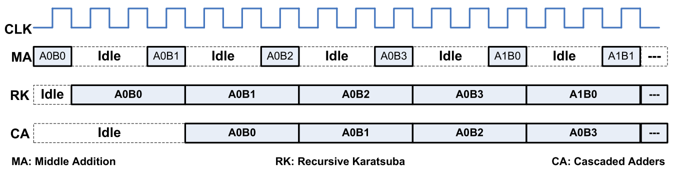


Fig. 7. Occupied and idle clock cycle instants of the proposed multistage Karatsuba multiplier.

it yields an unbalanced datapath, as shown in Fig. 6(A). Hence, an almost balanced datapath with three main stages is introduced, as shown in Fig. 6(B). Two stages consist of three substages datapath as the recursive 16-digit Karatsuba block, and the cascaded adders stage. One initial stage consists of one substage of the middle addition.

An iterative Karatsuba multiplication can be easily adopted to run with minimum idle stages starting from the initial middle addition that runs for one clock cycle. Then, the 16-digit Karatsuba would run for three clock cycles. Finally, the cascaded adders stage would run for another three clock cycles. Fig. 7 shows the run time for the three main stages. Note that idle stages appear only in the initial middle addition stage, where, Karatsuba and cascaded adders stages are fully occupied during the run time of our iterative Karatsuba multiplier.

4) Extended NIST Reduction: Generalized Mersenne primes [19] are the special type prime numbers that allow fast modular reduction. Regular division is replaced by few additions and subtractions. Such primes are represented as $p = f(t)$, where t is a power of 2. The modulus of the P256 curve is Merssene prime $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

Due to the redundancy nature of the RSD representation, the multiplication process may produce results that are represented by more than 512 digits and these results are still in the range $-p^2 < A < p^2$. These one or two extra digits are outside the range of the NIST reduction process. Hence, we derived new formulas to include these extra digits in the reduction process. The new reduction process has one extra 256-digit term, D_5, along with some modification of the previously existed terms. This term is added conditionally, whether the extra digit is set or not. Thus, two additions are the total overhead required to handle the extra digits caused using the RSD representation. The modified reduction formula is $B = T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4 - D_5 \mod p$, where $\mathbf{A_{16}}$ represents the extra digits produced by RSD Karatsuba multiplier

$$
\begin{aligned}
T &= (A_7 \| A_6 \| A_5 \| A_4 \| A_3 \| A_2 \| A_1 \| A_0)\\
S_1 &= (A_{15} \| A_{14} \| A_{13} \| A_{12} \| A_{11} \| 0 \| 0 \| 0)\\
S_2 &= (\mathbf{2 * A_{16}} \| A_{15} \| A_{14} \| A_{13} \| A_{12} \| 0 \| 0 \| \mathbf{A_{16}})\\
S_3 &= (A_{15} \| A_{14} \| 0 \| 0 \| \mathbf{-2 * A_{16}} \| A_{10} \| A_9 \| A_8)\\
S_4 &= (A_8 \| A_{13} \| A_{15} \| A_{14} \| A_{13} \| A_{11} \| A_{10} \| A_9)\\
D_1 &= (A_{10} \| A_8 \| 0 \| 0 \| \mathbf{2 * A_{16}} \| A_{13} \| A_{12} \| A_{11})\\
D_2 &= (A_{11} \| A_9 \| 0 \| \mathbf{A_{16}} \| A_{15} \| A_{14} \| A_{13} \| A_{12})\\
D_3 &= (A_{12} \| \mathbf{2 * A_{16}} \| A_{10} \| A_9 \| A_8 \| A_{15} \| A_{14} \| A_{13})\\
D_4 &= (A_{13} \| 0 \| A_{11} \| A_{10} \| A_9 \| \mathbf{A_{16}} \| A_{15} \| A_{14})\\
\mathbf{D_5} &= (\mathbf{-A_{16}} \| \mathbf{0} \| \mathbf{0} \| \mathbf{0} \| \mathbf{0} \| \mathbf{0} \| \mathbf{0} \| -\mathbf{A_{16}}).
\end{aligned}
$$

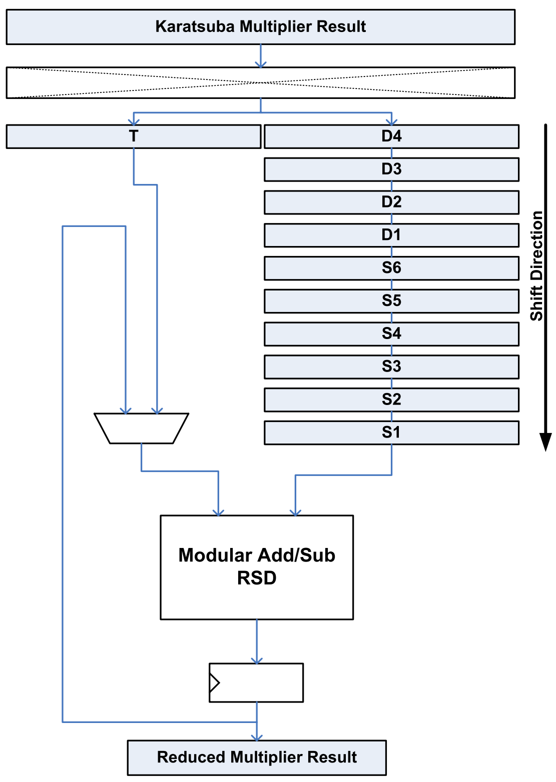


Fig. 8. Mod P256 reduction block.

In order to accommodate the extra digit produced by the RSD Karatsuba multiplier, NIST reduction is reformulated. The resultant reduction scheme consists of three extra additions. However, through reformulation and combining the original terms with the additional terms, the reduction scheme is optimized. Accordingly, the modular multiplier is built with a Karatsuba multiplier, modular RSD adder, and some registers to hold the 256-digit terms. Fig. 8 shows the block diagram of the Mod P256 RSD multiplier. A controller is used to control the flow of the terms to the modular adder and at every turn, the result of the modular addition is accumulated and fed back to the adder. The cross-bar in Fig. 8 shows the wiring of the 32-digit words to their respective locations within the extended NIST reduction registers.

C. High-Radix Modular Division

Binary GCD algorithm is an efficient way of performing modular division since it is based on addition, subtraction, and shifting operations. The complexity of the division operation comes from the fact that the running time of the algorithm is inconsistent and is input dependent. As seen in Algorithm 2, three main states define the flow of the algorithm. In the first state, the divider is checked whether it is even or odd. In the second state, the content of the corresponding registers are swapped according to the flag δ. In the last state, division by 4 modulo M is performed.

In order to efficiently implement Algorithm 2 in hardware, the following list of operations should be adopted to be executed efficiently in hardware. First, division by 2 or by 4 is simply performed by shifting to right 1-digit/2-digits accordingly based on the guarantee that the LSDs are zeros in line 3 and 12 of the algorithm. On the other hand, division by 2 modulo M (division by 4 modulo M) is performed by adding or subtracting the dividend to or from the modulus according to whether the dividend is even or odd and the value of M (mod 4). For both δ and ρ, a comparison with 0 is necessary. However, an efficient alternative is to initialize a vector of size n with all zeros except the least significant byte (LSB) for δ and the most significant byte (MSB) for ρ. Hence, the counting down of ρ is performed by shifting 1 bit to right and only the LSB is checked for the loop termination. On the other hand, a flag is needed to control the shift direction of δ, where the flag and the value of the LSB are used to determine whether it is less than zero or not. The implementation of the algorithm follows the implementation proposed in [38]. The modular divider architecture is shown in Fig. 9. Three RSD adders are used along with three 3×1 multiplexers and one 4×1 multiplexer with some control logic.

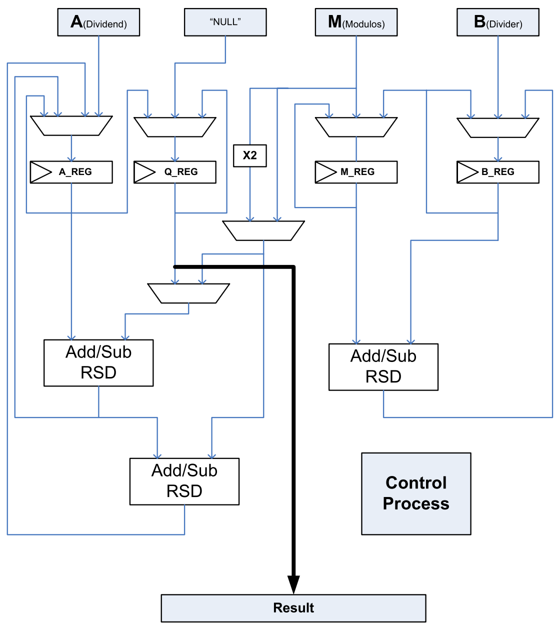


Fig. 9. Modular divider block.

V. Control Unit and Instruction Set

The control unit consists of the main controller of the processor that is an FSM. It also includes two processing units that control the procedures for the point doubling and point addition. In order to process the different coordinate systems, the other processing units can be generated to support them as an add-on feature. Hence, new instructions need to be added to the instruction set to accommodate the new processing units. Different scalar point multiplication algorithms are supported at instruction level. Different projective coordinate systems and point addition/doubling variations are configured at the control level through add-on processing units. On the other hand, elliptic curves with different finite fields are supported as a presynthesis process.

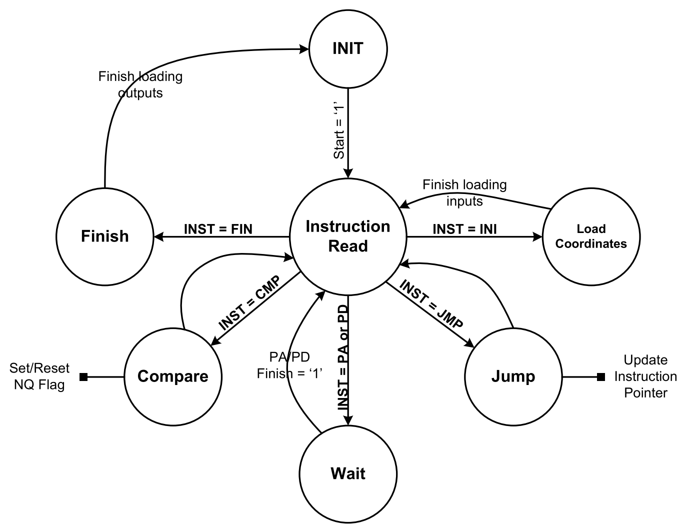


Fig. 10. Main controller FSM.

A. Main Controller

The primary task of the main controller is to fetch, read, and execute instructions from memory. Based on the flow of instructions, the two attached processing units for point doubling and point addition are controlled accordingly. During the running time of the point doubling and point addition, control signals for the AU blocks and the memory are generated. In addition, the procedure flow of operations within the processing units is controlled via flag signals coming from the AU blocks. The main controller is a simple FSM, where the states represent the instructions listed in the instruction set of Fig. 10. Initially, the controller is in the INIT state waiting for the START signal from the outside. When the START signal is set, the controller starts reading and executing instructions. Normally, the first instruction would be INI which leads to coordinates-loading state. After that, the FSM keeps bouncing between the reading instructions state and the other three states: 1) Wait; 2) Jump; and 3) Compare. Finally, the FSM reaches the Finish state where the output coordinates are loaded to output and the processor goes back to its initial state.

B. Point Addition/Doubling Controller

The point addition/doubling processing units are subroutines that control the sequence of operations for affine coordinate system. These subroutines are the part of the main controller and are made as add-on blocks for customization purposes. They mainly control the execution order of the AU blocks in order to obtain the resultant coordinates of adding two points or doubling a point. The coordinates of these two points reside in the first half of the memory, where the second half is used for storing the intermediate results from the AU. The flow and sequence of operations controlled by the point addition or point doubling controllers and the intermediate contents of the memory registers are shown in Fig. 11. It is to be noted that the dependence of operations in these two figures show limitation of parallelism for affine coordinates.

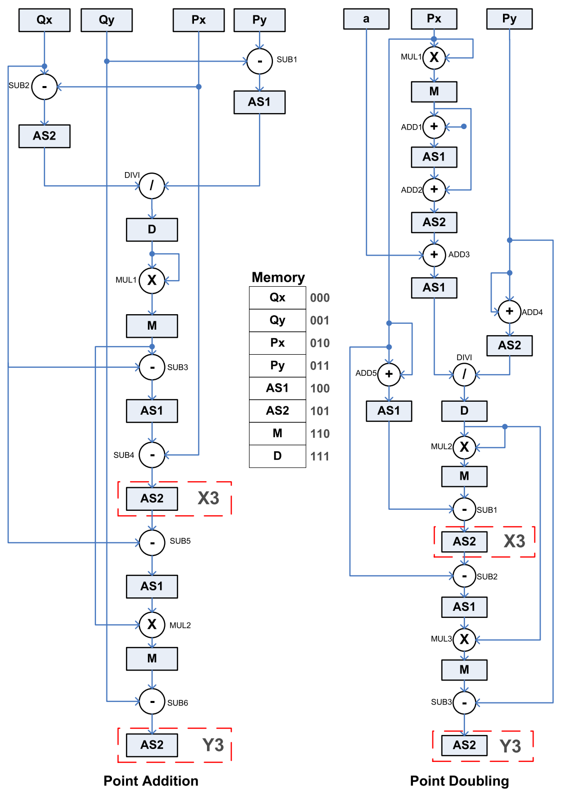


Fig. 11. Point addition and doubling process flow.

C. ROM and Instruction Set

The instruction set defined for our processor is at point multiplication level, where the main instructions represent point doubling and point addition operations. The other choice was to implement the instruction set at the arithmetic level; however, complexity increases as a cost for increased flexibility. Instructions are 16-bits wide, the MSB is used for the operation, and the LSB is used for operands. Initially, point doubling and point addition instructions are defined with the possibility to expand the instruction set to include other coordinate systems operations. In addition, the instruction set can be extended to include simple power analysis (SPA) countermeasures. For instance, other point addition and point doubling (XPA and XPD) are added to the instruction set that perform dummy operations to provide the balance needed to counteract an SPA attacks. The regular point addition requires 6 additions, 2 multiplications, and 1 division, where point doubling requires 8 additions, 3 multiplications, and 1 division. Hence, 1 dummy multiplications and 2 additions are added to XPA instruction to balance the point doubling with the point addition operation. Table III presents the instruction set with their operations and hexadecimal codes along with the clock cycles and their timings.

A ROM is used to hold the assembly code that represents a certain point multiplication algorithm. The code is loaded into ROM as a presynthesis process. The size of the ROM in our processor is 16×16 and can be expanded easily. Table IV shows the assembly code for the left-to-right point multiplication Algorithm 1. As a key feature of ASIP implementations, and due to the fact that our instruction set is built at curve level, i.e., point addition and point

TABLE III
P256 ECC PROCESSOR INSTRUCTION SET

Instruction	Operand A	Operand B	Hex Code	Clock Cycles	Operation
INI	None	None	01XX	1	Initialize registers
CMP	S: Comparison state	V: Comparison value	02VS	1	Compare the given value with another based on the state and set the not-equal flag
JMP	Next Instruction	None	03NI	1	Jump to the given instruction address in case the no-equal flag is set
WPA	None	None	04XX	790	Perform point addition
WPD	None	None	05XX	1000	Perform Point doubling
XPA	None	None	07XX	1000	SPA resistant point addition
XPD	None	None	08XX	1000	SPA resistant point doubling
FIN	None	None	06XX	1	Finish

TABLE IV
ASSEMBLY CODE FOR LEFT-TO-RIGHT POINT MULTIPLICATION

ROM Address	Assembly Code	Hex Code	Operation
0:0000	INI	0100	Initialize
1:0001	WPD	050F	Point double
2:0010	CMP 0, 1	0290	Check if key bit = 1
3:0011	JMP 5	0365	Jump to address x[5] if NQ flag is set
4:0100	WPA	0402	Point add
5:0101	CMP 1, 1	0291	Check if counter reached the end
6:0110	JMP 1	0351	Jump to address x[1] if NQ flag is set
7:0111	FIN	060F	Finish, reset instruction pointer
8:1000	—	0000	NOP

TABLE V
IMPLEMENTATION RESULTS OF INDIVIDUAL BLOCKS IN VIRTEX 5

	Modular Divider	**Modular Multiplier**	**Modular Adder/ Subtracter**	**Other Peripherals**
Resources (LUTs)	9,219	19,747	3,300	2,340
Flip-flops	4114	12,046	2,054	6,292
Delay (ns)	5.924	6.24	4.79	–
Maximum Frequency (MHz)	169	160	208	–

doubling level, the program size of the left-to-right point multiplication algorithm is highly optimized. According to Table IV, the program is coded by 8 instructions, which yields a program size of 16 bytes. The left-to-right point multiplication is performed within 361 700 clock cycles using the WPA and WPD instructions, where it is performed within 391 834 clock cycles if the XPA and XPD instructions. There is a timing overhead of 8.33% for applying a SPA countermeasure at the point multiplication level.

VI. IMPLEMENTATION RESULTS AND DISCUSSION

The proposed processor was implemented in Xilinx Virtex 5-XC5VLX110T FPGA and a single point multiplication for P256 is achieved within 2.26 ms. Detailed implementation results of individual blocks are listed in Table V. Such detailed results are useful in understanding the main block contributors to the overall hardware resources. It can be noted that the modular multiplier is the largest block within the design due to the three recursively built Karatsuba blocks, which operate in parallel. With the extensive pipelining techniques that are applied to the Karatsuba blocks, the CPD is shortened down to 6.24 ns. Such CPD figure allows the processor to operate at 160 MHz, which is the fastest achieved in the literature in FPGA devices without embedded blocks. Detailed timing performance of operations performed by the processor that is operating at 160 MHz on Virtex 5 device are listed in Table VI.

Table VII lists a comparison of our modular divider implementation results against other FPGA-based designs. Our modular divider performs the fastest timing of prime field dividers and competitive to binary field GF2^{233} modular divider. The performance enhancement is due to the usage of RSD, which leads to short datapath and high operating frequency. Efficient architecture that is based on implementing complex operations through simple shifting single bit checking is another factor that gives our divider such enhancement. Finally, the modular divider operates on higher radix which results in improved throughput.

The exportability feature of the processor comes from the fact that none of the macros or embedded blocks within the FPGA fabric is utilized in the proposed processor. Such feature gives our processor the freedom to be implemented in different FPGA devices from different vendors and, eventually, as an application-specified integrated circuit (ASIC). Hence, our processor is Look-Up Table

TABLE VI
DETAILED TIMING PERFORMANCE OF OPERATIONS PERFORMED BY THE PROPOSED PROCESSOR

	Modular Addition/Subtraction	**Modular Multiplication**	**Modular Division**	**Point Addition**	**Point Doubling**	**Point Multiplication**
CLK	3	212	315	790	1000	361,700
Time (μs)	0.0187	1.33	1.97	4.94	6.26	2262

TABLE VII
COMPARISON OF MODULAR DIVISION/INVERSION IN FPGA DEVICES

Article	**Device**	**Slices**	**Maximum Frequency**	**Time (μs)**	**Field and operand size**
This Work	Virtex 5 (XC5VLX110T)	3140	169 MHz	1.86	Prime 256
McIvor et al. [22]	Virtex-2 Pro XC2VP100-6	14.723	40 MHz	14.46	Prime 256
Sun et al. [41]	Virtex 5 (XC5VFX200T)	1410	300 MHz	8.00	Dual Field 160
Dormale [42]	Virtex 4 (XC4VLX15)	982	324 MHz	0.72 (Calculated)	Binary 233

TABLE VIII
COMPARISON OF PRIME FIELD ECC PROCESSORS

Article	**Device**	**Resources**	**Maximum Frequency**	**PM Time**	**Remarks**
Our processor	Virtex 5 (XC5VLX110T)	34,612 LUTs	160 MHz	2.26 ms	Based on RSD, without embedded blocks
	Virtex 4 (XC4VLX160)	50,589 LUTs	139 MHz	2.6 ms	
	Virtex-2 Pro (XC2VP100-6)	48,833 LUTs	95 MHz	3.81 ms	
	Virtex (XCV1000E-8)	48,500 LUTs	47.3 MHz	7.56 ms	
	Cyclone V (CEFA7F23C6)	29,311 LUTs	99.25 MHz	3.64 ms	
McIvor et al. [22]	Virtex-2 Pro XC2VP100-6	31,755 LUTs + 256 (18x18) Multipliers	39.46 MHz	3.86 ms	Full word Montgomery Multiplier, FPGA embedded multipliers and carry chain
Ananyi et al. [23]	Virtex-2 Pro	41,586 LUTs + 32 (18x18) Multipliers	49.5 MHz	6.41 ms	NIST curves and reduction, based on embedded multipliers, projective coordinates
Guneysu et al. [15]	Virtex 4 XC4VFX12	3,430 LUTs + 32 DSP	490 MHz	0.62 ms	Efficient utilization of FPGA DSP
Guillermin et al. [43]	Stratix II	18,354 LUTs + 96 DSP	157.2 MHz	0.68 ms	RNS and full word Montgomery multipliers, DSP based, projective coordinates
Esmaeildoust et al. [6]	Virtex 2 Pro	28,746 LUTs	50.2 MHz	0.59 ms	RNS and Montgomery multiplier, projective coordinates
Schinianakis et al. [7]	Xilinx XCV1000E-8	32,716 LUTs	39.7 MHz	3.95 ms	RNS based ECC point multiplier and modular multiplier based on Horner's rule
Satoh et al. [44]	CMOS 0.13 μm	107K Gates	137.7 MHz	2.68 ms	Word based W=64, Montgomery Multiplier with WTM, Projective
karakoyunlu et al. [12]	CMOS 0.13 μm	100K Gates	147 MHz	P192 = 1.95 ms	For arbitrary 192 prime curves, two AU's, with and without a divider for different coordinate systems, based on RSD.

(LUT)-based which means that simple logical operations can be mapped easily to both, LUT on FPGA and standard cells on ASIC technologies. We assessed the exportability feature while considering the fairness in comparing our processor with other processors proposed in the literature. Our processor is implemented in four different FPGA devices from Xilinx and one device from Altera. The selection of such devices was carefully taken care of to provide implementation results comparable with their counterparts listed in Table VIII. Note that all implementation results listed in Table VIII are specific to prime fields of 256 size, except the processor proposed in [12] which operates in the 192-bit field. It is important to note that such results listed in Table VIII are not specific to certain scalar point multiplication algorithm, scalar encoding, coordinate system, and target platform. Hence, the performance of different processors will vary according to the parameters listed above.

In order to correctly interpret the achieved results, it is important to understand the structure of the logical elements within different FPGA devices. Since our design is fully based on basic logical operations, then we may classify different devices based on the structure of the LUTs within such devices. We found that Xilinx Virtex E, Virtex 2 Pro, and Virtex 4 devices share the same structure of 4×1 LUTs. On the other hand, Virtex 5 has a LUT configuration of 6×2 which gives the advantage of incorporating larger logical networks compared with the 4×1 configuration. A disadvantage of 6×2 LUTs is that there is a high probability of under utilization of hardware resources since a simple 2×1 logical operation may occupy the same hardware resources as a fully 6×1 logical network. Altera's adaptive LUT (ALUT) in Cyclone and Stratix II devices tries to reduce the under utilization problem of large LUTs. A single ALUT includes two 3×1 and two 4×1 LUTs which are used to provide many configuration options of up to 7×1. Our processor is implemented in five different devices as listed in Table VIII. The processor occupies ~48 K of LUTs in devices with 4×1 LUTs, where it consumes 34 K in Virtex 5 and 29 K in Cyclone FPGA. The maximum operating frequency changes due to different factors, such as the processing technology of the devices, the interconnect architectures, the structure of the configurable blocks, the density of the device, and so on.

Different processors proposed in the literature are listed in Table VIII with different architectures. The processor in [22] is designed for arbitrary curves with projective coordinates. Embedded multipliers within the FPGA fabric are highly utilized for full word Montgomery multiplication along with the carry chain of the chip. An ECC processor that is specific for NIST primes is proposed in [23]. This processor is highly based on embedded multipliers and carry chain of the FPGA chip. An RNS are used with Montgomery multiplication in [43] and based on Horner's rule in [7]. The DSPs of the targeted FPGAs are used to speed up the full word multiplications. Hence, most of the FPGA implementations of ECC processors make use of embedded multipliers and DSP blocks that operate on projective coordinates. Two ASIC-based processors are also listed [12], [44].

The processor proposed in [22] is a full word processor similar to the work presented in this paper. The processor in [22] implements the full word Montgomery multiplier which is build from 256 18×18 embedded multipliers within the Virtex 2 Pro device. Both our processor (Virtex 2 Pro version) and the processor presented in [22] are approximately of the same size if we add the 256 embedded multipliers to the 32-K LUTs. Our processor operates at 95 MHz as opposed to 40 MHz in their processor. Both processors perform a single point multiplication within almost the same time. However, their work gets a major advantage through the use of the embedded multipliers that allow them to perform partial 18×18 multiplication within one clock cycles. This makes up the total number of clock cycles to 32 clock cycles for a single full 256 modular multiplication. If the processor in [22] employs our modular multiplier, then it would perform a single point multiplication in ~10.3 ms. The approximation is calculated with the assumption of a balanced hamming weight scalar and at an operating frequency of 95 MHz.

Similar to our processor and the processor proposed in [22], the processor proposed in [23] is also a full word 256 processor. Our processor occupies less resources within Virtex 2 Pro as opposed to 42-K LUTs and 32 embedded multipliers in [23]. Our processor runs at higher operating frequency and outperforms the processor in [23] in terms of scalar point multiplication by almost a double. On the other hand, the point multiplication accelerator proposed in [15] is a special category on its own. This accelerator employs efficiently the DSP blocks and the their dedicated interconnects to achieve a high frequency of 490 MHz. Note that the reported frequency is achievable if only DSP blocks are used without any control logic, since other FPGA components cannot run at such frequency. Hence, the results reported in [15] do not represent a full functional ECC processor.

Different processors that are built over a RNSs are also included in Table VIII. The processor in [43] carefully chooses base sizes that can be efficiently mapped to the DSP blocks within Altera's Startix II device. Our processor is implemented in Altera's Cyclone V device, which has the same ALUT structure as in Stratix II device. Although the processor in [43] is built on much smaller field sizes and greatly depends on the DSP blocks, our processor could achieve a competitive operating frequency of 99 MHz. Not to mention the fact that Cyclone devices are cost effective and lower end products as opposed to Stratix II devices. It is to be noted that the processors proposed in [6] and [43] are the fastest reported in Table VIII in terms of scalar point multiplication. This high performance capability is mainly inherited from the RNS system. In addition, the processor proposed in [7] is RNS-based and could achieve high performance in a slow device such as the Virtex E.

The design strategy of our processor is similar to many CMOS-based ECC processors reported in the literature. Two examples are added to Table VIII. Most CMOS-based ECC processors adopt carry free arithmetic, such as CSA [44] and RSD [12] to avoid lengthy datapaths. The processor in [44] is a word-based processor of 64-bit size. It operates on arbitrary field sizes using Montgomery multiplication. With the interconnects overhead that exists in FPGA designs, our processor could achieve shorter datapath and higher frequency. A single scalar point multiplication is performed by our processor in shorter time than the processor in [44]. However, the processor in [44] has the advantage of operating on different field types and sizes as opposed to our processor, which operates on dedicated NIST prime fields. Although the processor in [12] is built over RSD arithmetic, our processor could achieve higher throughput due to the extensive pipelining techniques and the efficiency of the implemented logical operations. Comparison in terms of hardware resources between FPGA- and CMOS-based designs is unfeasible since the exact gate equivalent of different LUT configuration is difficult to calculate or estimate. In our design, for instance, a 6×1 LUT in Virtex 5 may represents a logical network of 48 gates in one part of the design, and 12 gates in another part of the design.

The achieved short critical path is due to the improved pipelining strategies used in Karatsuba multiplier and the efficient architecture of the divider. The use of RSD representation is essential in reducing CPD of the processor. The processor in [15] achieved the highest operating frequency by careful utilization and routing of DSP blocks within the FPGA.

In addition, the processor in [43] operates on a maximum frequency of 157 MHz since the length of the datapath is reduced by the RNS representation along with the use of DSP blocks.

The target of achieving short datapath through the use of RSD representation lead to a compromise in terms of hardware resources used, as shown in Table VIII. However, if we accounted for the embedded multipliers and DSP blocks that are utilized by most FPGA designs in addition to the carry logic within the FPGA, the hardware overhead due to RSD can be justified. Also, the addition of such embedded blocks to the number of LUTs consumed by different designs makes our processor occupies competitive hardware resources figures, if not outperforming them. It is important to note that there is no fair metric to convert DSPs and embedded multipliers to equivalent LUT in order to present fair comparison.

The power consumption is estimated for the proposed processor using XPower Analyzer tool in the Xilinx ISE 10.1 suit. The power consumption of the proposed processor running on Virtex 5 and operating at frequency 160 MHz is estimated at 1.755 W with dynamic power at 0.693 W. These power consumption estimations are calculated for default operating parameters of the device, such as, temperature at 25 °C, Vccint at 1 V, and VCCAUX and VCCO at 2.5 V.

VII. Conclusion

In this paper, a NIST 256 prime field ECC processor implementation in FPGA has been presented. An RSD as a carry free representation is utilized which resulted in short datapaths and increased maximum frequency. We introduced enhanced pipelining techniques within Karatsuba multiplier to achieve high throughput performance by a fully LUT-based FPGA implementation. An efficient binary GCD modular divider with three adders and shifting operations is introduced as well. Furthermore, an efficient modular addition/subtraction is introduced based on checking the LSD of the operands only. A control unit with add-on like architecture is proposed as a reconfigurability feature to support different point multiplication algorithms and coordinate systems.

The implementation results of the proposed processor showed the shortest datapath with a maximum frequency of 160 MHz, which is the fastest reported in the literature for ECC processors with fully LUT-based design. A single point multiplication is achieved by the processor within 2.26 ms, which is comparable with ECC processors that are based on embedded multipliers and DSP blocks within the FPGA. The main advantages of our processor include the exportability to other FPGA and ASIC technologies and expandability to support different coordinate systems and point multiplication algorithms.

References

[1] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, Jan. 1987.

[2] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 5th ed. Englewood Cliffs, NJ, USA: Prentice-Hall, Jan. 2010.

[3] C. Rebeiro, S. S. Roy, and D. Mukhopadhyay, "Pushing the limits of high-speed GF(2^m) elliptic curve scalar multiplication on FPGAs," in *Proc. Cryptograph. Hardw. Embedded Syst. (CHES)*, vol. 7428. Jan. 2012, pp. 494–511.

[4] Y. Wang and R. Li, "A unified architecture for supporting operations of AES and ECC," in *Proc. 4th Int. Symp. Parallel Archit., Algorithms Programm. (PAAP)*, Dec. 2011, pp. 185–189.

[5] S. Mane, L. Judge, and P. Schaumont, "An integrated prime-field ECDLP hardware accelerator with high-performance modular arithmetic units," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Nov./Dec. 2011, pp. 198–203.

[6] M. Esmaeildoust, D. Schinianakis, H. Javashi, T. Stouraitis, and K. Navi, "Efficient RNS implementation of elliptic curve point multiplication over GF(p)," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 8, pp. 1545–1549, Aug. 2012.

[7] D. M. Schinianakis, A. P. Fournaris, H. E. Michail, A. P. Kakarountas, and T. Stouraitis, "An RNS implementation of an F_p elliptic curve point multiplier," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 6, pp. 1202–1213, Jun. 2009.

[8] J.-W. Lee, S.-C. Chung, H.-C. Chang, and C.-Y. Lee, "Efficient power-analysis-resistant dual-field elliptic curve cryptographic processor using heterogeneous dual-processing-element architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 1, pp. 49–61, Feb. 2013.

[9] J.-Y. Lai and C.-T. Huang, "Energy-adaptive dual-field processor for high-performance elliptic curve cryptographic applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 8, pp. 1512–1517, Aug. 2011.

[10] S.-C. Chung, J.-W. Lee, H.-C. Chang, and C.-Y. Lee, "A high-performance elliptic curve cryptographic processor over GF(p) with SPA resistance," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 1456–1459.

[11] J.-Y. Lai and C.-T. Huang, "Elixir: High-throughput cost-effective dual-field processors and the design framework for elliptic curve cryptography," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 11, pp. 1567–1580, Nov. 2008.

[12] D. Karakoyunlu, F. K. Gurkaynak, B. Sunar, and Y. Leblebici, "Efficient and side-channel-aware implementations of elliptic curve cryptosystems over prime fields," *IET Inf. Secur.*, vol. 4, no. 1, pp. 30–43, Mar. 2010.

[13] D. Schinianakis and T. Stouraitis, "Multifunction residue architectures for cryptography," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 4, pp. 1156–1169, Apr. 2014.

[14] J. Vliegen *et al.*, "A compact FPGA-based architecture for elliptic curve cryptography over prime fields," in *Proc. 21st IEEE Int. Conf. Appl.-Specific Syst. Archit. Process. (ASAP)*, Jul. 2010, pp. 313–316.

[15] T. Güneysu and C. Paar, "Ultra high performance ECC over NIST primes on commercial FPGAs," in *Proc. 10th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2008, pp. 62–78.

[16] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, Apr. 1985.

[17] K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems," in *Proc. 2nd Int. Workshop Reconfigurable Comput., Archit. Appl.*, vol. 3985. 2006, pp. 347–357.

[18] A. Byrne, E. Popovici, and W. P. Marnane, "Versatile processor for GF(p^m) arithmetic for use in cryptographic applications," *IET Comput. Digit. Tech.*, vol. 2, no. 4, pp. 253–264, Jul. 2008.

[19] J. Solinas, "Generalized Mersanne number," Univ. Waterloo, Waterloo, ON, Canada, Tech. Rep. CORR 99-39, 1999.

[20] B. Ansari and M. A. Hasan, "High-performance architecture of elliptic curve scalar multiplication," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1443–1453, Nov. 2008.

[21] N. Smyth, M. McLoone, and J. V. McCanny, "An adaptable and scalable asymmetric cryptographic processor," in *Proc. Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Sep. 2006, pp. 341–346.

[22] C. J. McIvor, M. McLoone, and J. V. McCanny, "Hardware elliptic curve cryptographic processor over GF(p)," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 9, pp. 1946–1957, Sep. 2006.

[23] K. Ananyi, H. Alrimeih, and D. Rakhmatov, "Flexible hardware processor for elliptic curve cryptography over NIST prime fields," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 8, pp. 1099–1112, Aug. 2009.

[24] M. Hamilton and W. P. Marnane, "FPGA implementation of an elliptic curve processor using the GLV method," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, Dec. 2009, pp. 249–254.

[25] F. Crowe, A. Daly, and W. Marnane, "A scalable dual mode arithmetic unit for public key cryptosystems," in *Proc. Int. Conf. Inf. Technol., Coding Comput. (ITCC)*, vol. 1. Apr. 2005, pp. 568–573.

[26] N. Takagi, "A VLSI algorithm for modular division based on the binary GCD algorithm," *IEICE Trans. Fundam. Electron., Commun., Comput. Sci.*, vol. E81-A, no. 5, pp. 724–728, May 1998.

[27] 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 Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 5, pp. 412–416, May 2007.

[28] J.-W. Lee, Y.-L. Chen, C.-Y. Tseng, H.-C. Chang, and C.-Y. Lee, "A 521-bit dual-field elliptic curve cryptographic processor with power analysis resistance," in *Proc. ESSCIRC*, Sep. 2010, pp. 206–209.

[29] H. Marzouqi, M. Al-Qutayri, and K. Salah, "An FPGA implementation of NIST 256 prime field ECC processor," in *Proc. IEEE 20th Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2013, pp. 493–496.

[30] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, 1st ed. New York, NY, USA: Springer-Verlag, Jan. 2004.

[31] I. F. Blake, G. Seroussi, and N. P. Smart, *Advances in Elliptic Curve Cryptography* (London Mathematical Society Lecture Note), 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, Apr. 2005.

[32] A. Avizienis, "Signed-digit numbe representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, no. 3, pp. 389–400, Sep. 1961.

[33] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Phys. Doklady*, vol. 7, p. 595, Jan. 1963.

[34] Y.-L. Chen, J.-W. Lee, P.-C. Liu, H.-C. Chang, and C.-Y. Lee, "A dual-field elliptic curve cryptographic processor with a radix-4 unified division unit," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2011, pp. 713–716.

[35] G. Chen, G. Bai, and H. Chen, "A new systolic architecture for modular division," *IEEE Trans. Comput.*, vol. 56, no. 2, pp. 282–286, Feb. 2007.

[36] NIST. (2000). *Recommended Elliptic Curves for Federal Government Use*. [Online]. Available: http://csrc.nist.gov/encryption

[37] Ç. K. Koç, Ed., *Cryptographic Engineering*, 1st ed. New York, NY, USA: Springer-Verlag, Dec. 2008.

[38] M. E. Kaihara and N. Takagi, "A hardware algorithm for modular multiplication/division," *IEEE Trans. Comput.*, vol. 54, no. 1, pp. 12–21, Jan. 2005.

[39] S. Yazaki and K. Abe, "VLSI design of Karatsuba integer multipliers and its evaluation," *Electron. Commun. Jpn.*, vol. 92, no. 4, pp. 9–20, 2009.

[40] N. Nedjah and L. de Macedo Mourelle, "A reconfigurable recursive and efficient hardware for Karatsuba–Ofman's multiplication algorithm," in *Proc. IEEE Conf. Control Appl. (CCA)*, vol. 2. Jun. 2003, pp. 1076–1081.

[41] W. Sun and L. Chen, "Design of scalable hardware architecture for dual-field Montgomery modular inverse computation," in *Proc. Pacific-Asia Conf. Circuits, Commun., Syst. (PACCS)*, May 2009, pp. 409–412.

[42] G. M. de Dormale and J.-J. Quisquater, "Iterative modular division over GF(2^m): Novel algorithm and implementations on FPGA," in *Proc. 2nd Int. Workshop Appl. Reconfigurable Comput. (ARC)*, vol. 3985. Mar. 2006, pp. 370–382.

[43] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$," in *Proc. 12th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, vol. 6225. Jan. 2010, pp. 48–64.

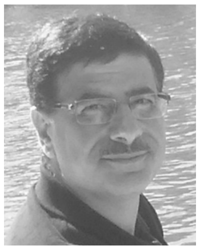
[44] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, Apr. 2003.

Hamad Marzouqi (M'13) received the B.Sc. degree in electrical and electronics engineering from the University of Sharjah, Sharjah, United Arab Emirates, in 2006, and the master's degree in security, cryptology and coding of information system from the Université de Grenoble, Grenoble, France, in 2008. He is currently pursuing the Ph.D. degree with Khalifa University, Abu Dhabi, United Arab Emirates.

He was a Hardware Cryptographic Researcher with the Emirates Advanced Investments Group until 2010. His current research interests include efficient and secure implementation of cryptographic hardware systems.

Mahmoud Al-Qutayri (M'86–SM'–) received the B.Eng. degree from Concordia University, Montreal, QC, Canada, in 1984, the M.Sc. degree from the University of Manchester, Manchester, U.K., in 1987, and the Ph.D. degree from the University of Bath, Bath, U.K., in 1992, all in electrical and electronic engineering.

He is currently a Full Professor with the Department of Electrical and Computer Engineering and the Associate Dean of Graduate Studies with Khalifa University, Abu Dhabi, United Arab Emirates. He has authored numerous technical papers in peer-reviewed international journals and conferences, and co-authored a book. His current research interests include embedded systems design and applications, design and test of mixed-signal integrated circuits, wireless sensor networks, and cognitive radio.



Khaled Salah (M'92–SM'–) received the B.S. degree in computer engineering with a minor in computer science from Iowa State University, Ames, IA, USA, in 1990, and the M.S. degree in computer systems engineering and the Ph.D. degree in computer science from the Illinois Institute of Technology, Chicago, IL, USA, in 1994 and 2000, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Khalifa University, Abu Dhabi, United Arab Emirates. He has over 10 years of industrial experience in the firmware development of embedded systems. He has over 100 publications in the areas of computer and network security, embedded systems, and operating systems.

Dimitrios Schinianakis (M'12) received the Diploma degree in electrical and computer engineering and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Patras, Patras, Greece, in 2005 and 2013, respectively.

He has authored over 15 conference and journal papers. His current research interests include computer arithmetic, cryptography, cryptanalysis, and VLSI hardware design.

Dr. Schinianakis is a member of the Technical Chamber of Greece. He was a recipient of the student paper award from the IEEE in 2006, and an award for a tutorial in the International Conference on Electronics, Circuits, and Systems (ICECS) in 2009. He regularly reviews for conferences like the IEEE International Symposium on Circuits and Systems and ICECS, and for journals like the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS and the *Journal of Systems and Software* (Elsevier). He served as a member of the Program Committee in the International Conference on Management in 2011, and the Publications Chair in the International Symposium on Communications, Control and Signal Processing in 2014.

Thanos Stouraitis (F'07) received the B.S. and M.S. degrees from the University of Athens, Athens, Greece, the M.Sc. degree from the University of Cincinnati, Cincinnati, OH, USA, and the Ph.D. degree in electrical engineering from the University of Florida, Gainesville, FL, USA.

He has served as a Faculty Member with Ohio State University, Columbus, OH, USA, and has visited the University of Florida, New York Polytechnic University, Brooklyn, NY, USA, and the University of British Columbia, Vancouver, BC, Canada. He is currently a Professor with the Department of Electronics and Communication Engineering, University of Patras, Patras, Greece, where he directs the Signal and Image Processing Laboratory. He has authored over 180 technical papers and several books and book chapters, and holds one patent in DSP processor design. His current research interests include signal and image processing systems, application-specific processor technology and design, computer arithmetic, and design and architecture of optimal digital systems with an emphasis on cryptographic systems.