Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2021) 29: 2127 – 2139
© TÜBİTAK
doi:10.3906/elk-2008-8

# Field-programmable gate array (FPGA) hardware design and implementation of a new area efficient elliptic curve crypto-processor

**Muhammad KASHIF**[1], **İhsan ÇİÇEK**[2,*]
[1]Department of Electrical-Electronics Engineering, İstanbul Şehir University, İstanbul, Turkey
[2]Department of Electrical-Electronics Engineering, İstinye University, İstanbul, Turkey

**Received:** 04.08.2020 • **Accepted/Published Online:** 29.03.2021 • **Final Version:** 26.07.2021

**Abstract:** Elliptic curve cryptography provides a widely recognized secure environment for information exchange in resource-constrained embedded system applications, such as Internet-of-Things, wireless sensor networks, and radio frequency identification. As the elliptic-curve cryptography (ECC) arithmetic is computationally very complex, there is a need for dedicated hardware for efficient computation of the ECC algorithm in which scalar point multiplication is the performance bottleneck. In this work, we present an ECC accelerator that computes the scalar point multiplication for the NIST recommended elliptic curves over Galois binary fields by using a polynomial basis. We used the Montgomery algorithm with projective coordinates for the scalar point multiplication. We designed a hybrid finite field multiplier based on the standard Karatsuba and shift-and-add multiplication algorithms that achieve one finite field multiplication in $\frac{m}{2}$ clock cycles for a key-length of m. The proposed design has been modeled in Verilog hardware description language (HDL), functionally verified with simulations, and implemented for field-programmable gate array (FPGA) devices using vendor tools to demonstrate hardware efficiency. Finally, we have integrated the ECC accelerator as an AXI4 peripheral with a synthesizable microprocessor on an FPGA device to create an elliptic curve crypto-processor.

**Key words:** Elliptic curve cryptography, Karatsuba multiplier, crypto-accelerator, crypto-processor, scalar multiplication, field-programmable gate array

## 1. Introduction

Modern security provisions require and mandate the use of cryptographic algorithms as a result of elevated risks associated with the ever increasing connectivity of the embedded devices. In practical, cryptographic applications, both symmetric and asymmetric cryptographic algorithms are used in communication protocols to establish a secure channel for information interchange. Designers often follow a hybrid approach to utilize the best of both worlds. Asymmetric algorithms are usually used in the management and exchange of keys in a secure communication protocol while the symmetric cryptographic algorithms are utilized for high-throughput secure data exchange. However, the computational requirements still pose a problem for the cost-sensitive lightweight embedded systems, such as Internet-of-things (IoT) or wireless sensor networks (WSN). The processors used in such systems typically have limited computational power, and thus cryptographic software implementations have limited performance. Moreover, cryptographic software consumes the precious code space allocated for the main application. The most efficient approach adopted to address these issues is the use of dedicated cryptographic hardware peripherals integrated with the central processing unit (CPU). Because of higher speed,

*Correspondence: ihsan.cicek@istinye.edu.tr lower power consumption, and smaller area advantages, elliptic curve cryptography (ECC) has become a very convenient choice for lightweight and resource-constrained embedded systems. It is frequently used in various ECC applications such as banking [1], cloud computing [2], and IoT [3]. It is practically impossible to extract the decryption key through brute force breaking attempts due to the computational complexity of the elliptic curve discrete logarithm problem (ECDLP) [4, 5]. ECC can provide an equivalent level of security by using shorter key-lengths when compared to other popular asymmetric cryptography algorithms such as RSA. For example, the security level provided by 2048-bit RSA is equivalent to using 233-bit ECC, which uses much fewer hardware resources.

From a hardware architecture point of view, the ECC algorithm can be hierarchically partitioned into four layers of computation [6]. The first layer consists of finite field arithmetic operations, such as addition, multiplication, squaring, and inversion [7]. Point addition (PA) and point doubling (PD) operations are computed at the second layer above the first one. Scalar point multiplication (SPM) is the most computationally intensive part of the ECC, and it is performed in the third layer. Finally, the encryption and decryption protocols are completed at the outermost fourth layer. Several algorithms are available to perform single point mooring (SPM) operation for different applications. Among all SPM algorithms, Montgomery algorithm is the most popular one due to its inherent resistance against the timing attacks and simple power analysis based side-channel attacks [8]. The overall performance of SPM depends on the type of the utilized finite field multiplier [3]. The bit-serial, bit-parallel, digit serial, and digit parallel multipliers are the most popular types of multipliers used for the computation of SPM operation in the literature [9]. Bit parallel and digit parallel multipliers use single clock cycle for the computation of one finite field multiplication, while the bit-serial and digit serial multipliers require multiple clock cycles [9]. Various technologies such as application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs) are available for hardware implementation of the ECC. Field-programmable gate arrays (FPGAs) have the design portability, reconfigurability, and scalability advantages over application specific integrated circuits (ASICs), which is the common-sense choice for high volume and low-cost applications with limited or no flexibility.

In this work, we propose a new ECC accelerator architecture that offers high-performance in a compact footprint for addressing the ECC IP core requirements of the emerging resource-constrained embedded systems. We can summarize our contributions in this work as the following:

1. We present a new hardware efficient ECC accelerator design for use with National Institute for Standards in Technology (NIST) recommended curves over $GF(2^m)$ with $m = 233$ [10]. We introduced a new hybrid multiplier architecture, which trades time with space to yield a high speed and low area multiplier composed of the high-speed standard Karatsuba [11], and compact shift-and-add multipliers with a timing cost of $m/2$ clock cycles [12].

2. SPM in the proposed ECC accelerator is based on the Montgomery ladder algorithm, which provides a trade-off between speed and area [8, 13]. We used a 2-stage pipeline to keep the overall throughput with a carefully designed schedule for the Montgomery algorithm to avoid read-after-write hazards.

3. Finally, our proposed ECC processor design has been integrated with a microblaze synthesizable central processing unit (CPU) on a Xilinx Artix-7 FPGA as a peripheral to enable fourth layer ECC applications.

The remainder of this paper is organized as follows. We discuss the preliminaries to compute SPM on ECC over $GF(2^m)$ in Section 2. We present the hardware architecture of the proposed ECC accelerator in

Section 3. A performance-optimized architecture for the SPM for ECC is explained in Section 4. Section 5 provides the details on FPGA hardware implementations, and in Section 6, we discuss performance results of the proposed ECC processor along with a comparison to the literature.

## 2. Elliptic curve cryptography on $GF(2^m)$

In cryptography, binary fields $GF(2^m)$ are usually considered more convenient for hardware implementation of the elliptic curves (ECs) [7]. Singular and nonsingular ECs can be used to implement the field arithmetic. Nonsingular ECs are considered to be more secure and, hence, more suitable for cryptography [14]. In the affine coordinate system, a nonsingular EC over $GF(2^m)$ is defined as a set of points ($x$ and $y$), which satisfy the following:

$$y^2 + xy = x^3 + ax^2 + b\, mod(F(x)) \tag{1}$$

in which, $a$ and $b$ are EC parameters with $b \neq 0$. The variables $x$ and $y$ are the base point elements, and $F(x)$ is the irreducible polynomial. The point at infinity is $\phi$. Therefore, when a point $P1$ on an EC is added to point at infinity $\phi$, the resulting point also lies on the EC, i.e., $P1 + \phi = P1$. Finite field arithmetic operations are used for point addition (PA) and point doubling (PD) when two points $P1$ and $P2$ over $GF(2^m)$ on the EC are considered. If the two points of EC are not the same ($P1 \neq P2$), then the addition of these points gives a new point Q on the EC as a result of PA; whereas, if the points are the same ($P1 = P2$) then adding these two points gives a new point $Q$ which lies on EC and is the result of PD. The following arithmetic operations are used in ECC over $GF(2^m)$:

Finite field addition and squaring: Finite field addition is performed usually using bitwise exclusive OR (XOR) operation without any carry propagation. The technique of interleaving zeros is normally used for finite field squaring. It takes one clock cycle to compute finite field squaring and finite field addition.

Finite field inversion and reduction: Field squaring and field multiplication are performed repeatedly to compute the field inversion. The reduction is needed to return to the original bit-length after multiplication and squaring.

Finite field multiplication: Finite field multiplication is the most important arithmetic operation in ECC. Numerous efforts have been spent on developing new ECC multiplier architectures and optimizing their performance. In [15], the authors have used a simple Karatsuba multiplier and use partial product technique at final recursion of the Karatsuba multiplication algorithm. Bit-parallel and bit-serial multipliers are the most frequently used multipliers in cryptographic applications [16]. The compact digit-serial multiplier takes $u/v$ clock cycles to perform multiplication [17] where $u/v$ is the total digits, $u$ and $v$ are the key length and the digit size, respectively. Bit-parallel multiplier multiplies two m-bit numbers in one clock cycle, but it requires a large area [15]. A modified multiplier architecture is proposed in [18] using an encoding algorithm. Authors of [19] chose projective coordinates and used a digit-serial multiplier for efficient EC multiplication. In [20], the authors have designed four different multipliers. Since they focused more on pipelining, their multipliers improved the speed but at the cost of increased area. A comprehensive survey on finite field multipliers is available in [21]. In this work, we have proposed a novel multiplier employing the Karatsuba multiplier along with the ordinary shift-and-add algorithm as discussed in Section 4.

The most important operation in the ECC is the scalar point multiplication (SPM) because it directly affects the computation time and the overall ECC performance [22]. SPM requires an initial point $P$ on the EC and an integer $k$ whose size is equivalent to the size of the field under consideration [23]. The SPM is simply the addition of $k$ copies of the point $P$, $(Q = kP = P + P + P...)$. Alternatively, $k$ is the discrete logarithm of $Q$ to the base $P$ [24]. The SPM of $kP$ is computed by performing the PA and PD repeatedly. if the *ith* bit of $k$ is 1, point addition is performed and if it is 0, either PA or PD can be done. Usually for $k[i] = 0$, PD is performed because it needs comparatively less arithmetic operations than the PA [14]. In SPM, inversion needs to be performed for each PA and PD operation, which creates a performance bottleneck [14]. To avoid this inversion cost, a conversion from affine (x,y) to projective coordinates $(X : Y : Z)$ can be computed, and then SPM is performed [23]. Afterward, a projective to affine coordinates conversion is performed back again to get the EC points result. For EC over $GF(2^m)$ as represented by the Equation 1, the corresponding conversion to projective coordinates is provided in Equation 2 and the reconversion to general affine coordinates is shown by the Equation 3.

$$(X, Y, Z) = (\lambda^c x, \lambda^d y, \lambda) \tag{2}$$

$$(x, y) = (X/Z^c, Y/Z^d) \tag{3}$$

Different kinds of projective coordinates can be achieved using different values of $c$ and $d$. For example, $c = 1$ and $d = 2$ results in Lopez Dahab projective coordinates, while $c = 2$ and $d = 3$ result in Jacobean coordinates. Since Lopez Dahab coordinate system requires less finite field multipliers for one point multiplication, we have chosen it in hardware implementation [22]. Montgomery algorithm, presented by Algorithm 1, has been used for the computation of scalar point multiplication.

**Algorithm 1** Montgomery algorithm over $GF(2^m)$

```
    Input :   $P = (x_p, y_p) \in GF(2^m)$, $k = (k_{n-1}, ...., k_1, k_0)$ with $k_{n-1} = 1$
    Output:   $Q(x_q, y_q) = k.P$
 1: //Affine to projective conversion
 2: $A_1 = x_p,\ B_1 = 1,\ A_2 = x_p^4 + b,\ B_2 = x_p^2$
 3: for ($i$ from 0 to $n-2$) do
 4: if $k_i = 1$
 5:     $T \leftarrow A_1,\ A_1 \leftarrow (A_1A_2 + A_2B_1)^2,\ A_1 \leftarrow x_pB_1 + A_1A_2TB_2$
 6:     $T \leftarrow A_2,\ A_2 \leftarrow A_2^4 + bB_2^4,\ B_2 \leftarrow T^2B_2^2$
 7: else
 8:     $T \leftarrow B_2, B_2 \leftarrow (A_1B_2 + A_2B_1)^2, B_2 \leftarrow x_pB_2 + A_1A_2TB_1$
 9: end if
10: end for
11:     $T \leftarrow A_1, A_1 \leftarrow A_1^4 + bB_1^4, B_1 \leftarrow T^2B_1^2$
12: //Projective to affine conversion
13: $x_q \leftarrow A_1/B_1$,
14: $y_q \leftarrow (x_p + \frac{A_1}{B_1})[(A_1 + x_pB_1)(A_2 + x_pB_2) + (x_p^2 + y_p)(B_1B_2)]/(x_pB_1B_2) + y_p$
15: return $Q(x_q, y_q)$
```

An initial point $P(x_p, y_p)$, along with a scalar multiplier $k$ is required as an input to implement the Montgomery algorithm. The algorithm computes the coordinates of the $Q(x_q, y_q)$ point as the final output. The Montgomery algorithm operates in a three-phase process. The first phase is the conversion from affine coordinates to projective coordinates (Lopez-Dahab) to avoid the inversion cost of the PA and PD computation. Then, in the second phase, the scalar multiplication is performed where PA, $(P_{i+1} = P_i + Q_i)$, and PD, $(P_{i+1} = 2P_i)$, instructions are computed depending on the value of $(k_i)$ according to Algorithm 1. Finally, reconversion to affine coordinates is done to get the final point Q of EC after PA and PD computations.

## 3. Hardware design of the elliptic curve cryptography (ECC) accelerator

We have chosen a binary $GF(2^m)$ field for the SPM computation of the ECC by using a polynomial basis representation with a projective coordinate system to perform efficient finite field multiplications and to reduce the cost of finite field inversion in hardware. Our ECC hardware accelerator design has a 2-stage pipeline architecture as shown in Figure 1. It is composed of a memory unit (MU) for storing the results of point multiplication, multiplexers, and demultiplexers for routing purposes, and an arithmetic logic unit (ALU) for the computation of finite field arithmetic. We also designed a finite state machine (FSM) based dedicated controller unit (DCU) and pipeline registers to manage the functions and to optimize the critical delay path. The initial EC parameters $(x_p, y_p, b)$, were chosen from NIST recommended ECs [10].

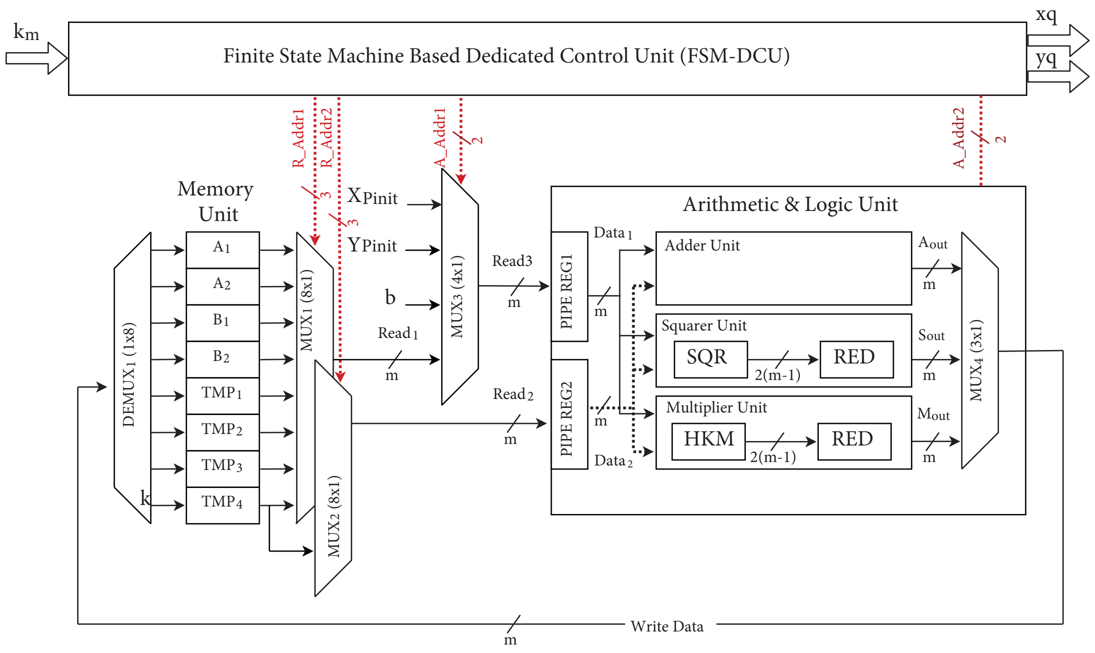

**Figure 1**. Hardware architecture of the proposed elliptic curve processor.

Memory unit: MU contains $8 \times$m size of an array as shown in Figure 1, where m identifies the size of each memory array and is equal to the size of the underlying field, which is 233-bits. The main purpose of the MU unit is to store the intermediate results (A1-2, B1-2, Tmp1-4) during the computation of the SPM. Also, it contains two multiplexers labeled as (Multiplexer_1 and Multiplexer_2) as presented in Figure 1. They are used for reading the operands (Read_1 and Read_2) as inputs to the ALU from MU and a single demultiplexer is used to update the MU registers using the Write_Data control signal.

Multiplexer units: Two additional multiplexers (Multiplexer_3 and Multiplexer_4) are employed for routing purposes, as presented in Figure 1. Inputs to the Multiplexer_3 are the initial EC parameters and an operand (Read_1) from the MU. The output of the Multiplexer_3 is an operand (Read_3) that is the next input to the ALU. The inputs for the Multiplexer_4 are the adder's, multiplier's, and squarer's outputs, and they are written back to MU using the demultiplexer.

Arithmetic logic unit: The ALU component consists of an adder, a multiplier, and a squarer unit as shown in Figure 1. The adder used for the computation of addition operation, and it is implemented by the bitwise XOR gates, whereas squaring is performed by inserting 0s after the each successive input data bit. Addition and squaring are performed in a single clock cycle. To multiply two m-bit polynomials, we have utilized a hybrid approach using the simple Karatsuba multiplier with the classic shift-and-add multiplication algorithm. When two m-bit polynomials are multiplied or a single $m$-bit polynomial is squared, the result will be $2m$-bits. Therefore, a finite field reduction (Red) needs to be performed whenever a finite field multiplier or squarer of the ALU is used [25]. We have used the reduction algorithm over $GF(2^m)$ recommended by NIST as described in [14]. Finally, the Itoh–Tsujii algorithm is used to perform an inversion over the field square [26].

**Pipeline registers:** we have used the Montgomery algorithm for the SPM. Two pipeline registers have been placed at the inputs of the ALU as shown in Figure 1. The addition of pipeline registers allows the parallel execution of the Montgomery algorithm's PA and PD instructions, therefore, read-after-write hazards (RAW) may occur. The PA and PD instructions of the Montgomery algorithm have been carefully scheduled as shown in Table 1 to avoid potential RAW hazards. The sequence for the PA and PD instructions of the Montgomery algorithm for both no-pipeline and 2-stage pipeline variants are presented in Table 1. The first column shows the required clock cycles, while the second one lists the sequence of instructions for the no-pipeline variant. The corresponding RAW hazards are tabulated in the third column, and instruction scheduling used for the 2-stage pipeline variant is shown in the fifth column. According to the Table 1, the total number of required instructions for each PA and PD computation for the no-pipeline architecture is 14, and it takes 710 clock cycles for a single SPM computation. Out of these 710 cycles, 6 multiplications are required, and each finite field multiplication requires $\frac{m}{2}$ clock cycles. However, for $GF(2^{233})$, a total of 6 $\times \frac{m}{2} = 702$ clock cycles are needed for the computation of 6 finite field multiplications. Finite field addition and squaring instructions consume the remaining 8 clock cycles. In the context of pipelining, PA and PD instructions have a total of 7 RAW hazards as shown in Table 1. The hazard term means that the execution of current instruction is stalled until the result of previous instruction has been written back in the memory unit [13]. Due to the occurrence of several RAW hazards (Third column of the Table 1), the instructions shown in the second column of Table 1 requires a total of 717 cycles when using the same sequence for one PA and PD computation. Moreover, the no-pipeline variant has a longer critical path delay and consequently operates at a lower clock frequency [13]. To optimize the clock frequency and reduce the critical path, we used the proposed sequence of instructions shown in the fourth column of Table 1. The proposed sequence of instructions require only 713 clock cycles for the execution of PA and PD as shown in Table 1.

Dedicated controller unit: We designed a finite state machine (FSM) based dedicated controller unit (DCU) to execute control functions such as the control signals of the multiplexers and read-write addresses of the memory unit. The DCU generated signals are shown as dotted lines with red color in Figure 1. Each finite field adder, squarer, and reduction modules generate results in one clock cycle to compute the Montgomery algorithm for scalar multiplication. Each finite field multiplication requires $\frac{m}{2}$ clock cycles. The total clock cycles for the no-pipeline and 2-stage pipeline variants are calculated using the Equation 4 and 5 respectively, and the yielding

**Table 1**. Execution scheduling of PA and PD instructions in the 2-stage pipeline.

| Clock cycles | No-pipeline schedule ($In_i$) | Pipeline hazard | Clock cycles | 2-stage pipeline schedule ($In_i$) |
|---|---|---|---|---|
| 1-117 | $In_1 \rightarrow B_1 = A_2 \times B_1$ | – | 1-1 | $In_1[R]$ |
| 118-234 | $In_2 \rightarrow A_1 = A_1 \times B_2$ | – | 2-118 | $In_1[E, WB], In_2[R]$ |
| 235-235 | $In_3 \rightarrow T_1 = A_1 + Z_1$ | $RAW : A_1$ | 119-235 | $In_2[E, WB], In_8[R]$ |
| 236-352 | $In_4 \rightarrow A_1 = A_1 \times B_1$ | – | 236-236 | $In_8[E, WB], In_3[R]$ |
| 353-353 | $In_5 \rightarrow B_1 = T^2$ | – | 237-237 | $In_3[E, WB], In_4[R]$ |
| 354-470 | $In_6 \rightarrow T_1 = x_p \times B_1$ | $RAW : B_1$ | 238-354 | $In_4[E, WB], In_5[R]$ |
| 471-471 | $In_7 \rightarrow A_1 = A_1 + T_1$ | $RAW : T_1$ | 355-355 | $In_5[E, WB], In_{11}[R]$ |
| 472-472 | $In_8 \rightarrow B_2 = B_2^2$ | – | 356-356 | $In_{11}[E, WB], In_6[R]$ |
| 473-473 | $In_9 \rightarrow T_1 = B_2^2$ | $RAW : B_2$ | 357-473 | $In_6[E, WB]$ |
| 474-590 | $In_{10} \rightarrow T_1 = b \times T_1$ | $RAW : T_1$ | 474-474 | $In_7[R]$ |
| 591-591 | $In_{11} \rightarrow A_2 = A_2^2$ | – | 475-475 | $In_7[E, WB], In_9[R]$ |
| 592-708 | $In_{12} \rightarrow Z_2 = A_2 \times Z_2$ | $RAW : A_2$ | 476-476 | $In_9[E, WB], In_{12}[R]$ |
| 709-709 | $In_{13} \rightarrow A_2 = A_2^2$ | – | 477-593 | $In_{12}[E, WB], In_{10}[R]$ |
| 710-710 | $In_{14} \rightarrow A_2 = A_2 + T_1$ | $RAW : A_2$ | 594-710 | $In_{10}[E, WB], In_{13}[R]$ |
| - | – | – | 711-711 | $In_{13}[E, WB]$ |
| - | – | – | 712-712 | $In_{14}[R]$ |
| - | – | – | 713-713 | $In_{14}[E, WB]$ |

clock cycles are provided in Table 2.

$$Init + 710 \times (m - 1) + 2 \times (Inv) + 3,556 \quad (4)$$

$$Init + 713 \times (m - 1) + 2 \times (Inv) + 3,571 \quad (5)$$

For the no-pipeline and 2-stage pipeline architectures, the projective to affine coordinate conversions (*Init*) requires only 6 and 12 clock cycles, respectively. It takes a total of 164,720 clock cycles to perform one PA and PD computation for the no-pipeline architecture when the sequence shown in the second column of Table 1 is used for the order of instruction execution. Similarly, for the 2-stage pipeline variant, the total number of clock cycles required for PA and PD computation is 165,416 when the sequence of instructions shown in the fifth column of Table 1 is executed. The projective to affine conversion requires two inverse operations with additional clock cycles of 3556 and 3571 for the no-pipeline and 2-stage pipeline architectures, respectively. Each inverse operation (*Inv*) of the no-pipeline architecture requires 2436 clock cycles, whereas it takes 2524 cycles for the 2-stage pipeline variant. The computation of scalar multiplication operation requires 173,154 for the no-pipeline and 174,047 cycles for the 2-stage pipeline architectures.

## 4. Proposed finite field multiplier

We have used the standard Karatsuba–Offman multiplier algorithm with the shift-and-add multiplication algorithm in our design. Karatsuba multiplier is much faster than the shift-and-add multiplier, whereas the shift-and-add multiplier needs much less area. The Karatsuba multiplier avoids some multiplication steps at the cost of the addition. The input operands are divided into two equal parts. For example, if one input is

**Table 2**. Timing information for ECC over $GF(2^{233})$.

| **Parameters** | **No-pipeline** | **2-stage pipeline** |
|---|---|---|
| $Init$ | 6 | 12 |
| $710 \times m - 1$ | 164,720 | - |
| $713 \times m - 1$ | - | 165,416 |
| $2 \times Inv$ | $2 \times 2436 = 4872$ | $2 \times 2524 = 5048$ |
| $Proj - to - affine$ | 3,556 | 3,571 |
| **Total clock cycles** | **173,154** | **174,047** |

$OP1$ of length $OP1\_length$, then the second input is $OP2$ of length $OP2\_length$. $OP1$ is divided into $a$ and $b$ where $a$ contains upper half of $OP1$ and $b$ contains lower half of $OP1$. Similarly, $c$ and $d$ divides $OP2$ in the same way. Then, $a$ is multiplied with $c$, and $b$ is multiplied with $d$ using the shift-and-add algorithm. The next step in computation is the addition of $a$, $b$, $c$, and $d$. The sum of $a$ and $b$ is multiplied with the sum of $c$ and $d$ in the following step. All of these are performed in parallel, which increases the overall speed. Afterward, we subtract the $bd$ and $ac$ multiplication results from the accumulated multiplication and shift the result left by $OP1\_length/2$ times, and add it to the multiplication result of $a$ and $c$. Finally, we add the result in the previous step to the multiplication result of $b$ and $d$ for computing the final result. The implemented Karatsuba and shift-and-add algorithms are shown in Algorithm 2 and Algorithm 3, respectively. Also, a flowchart representation of our proposed hybrid Karatsuba multiplier (HKM) is shown in Figure 2.

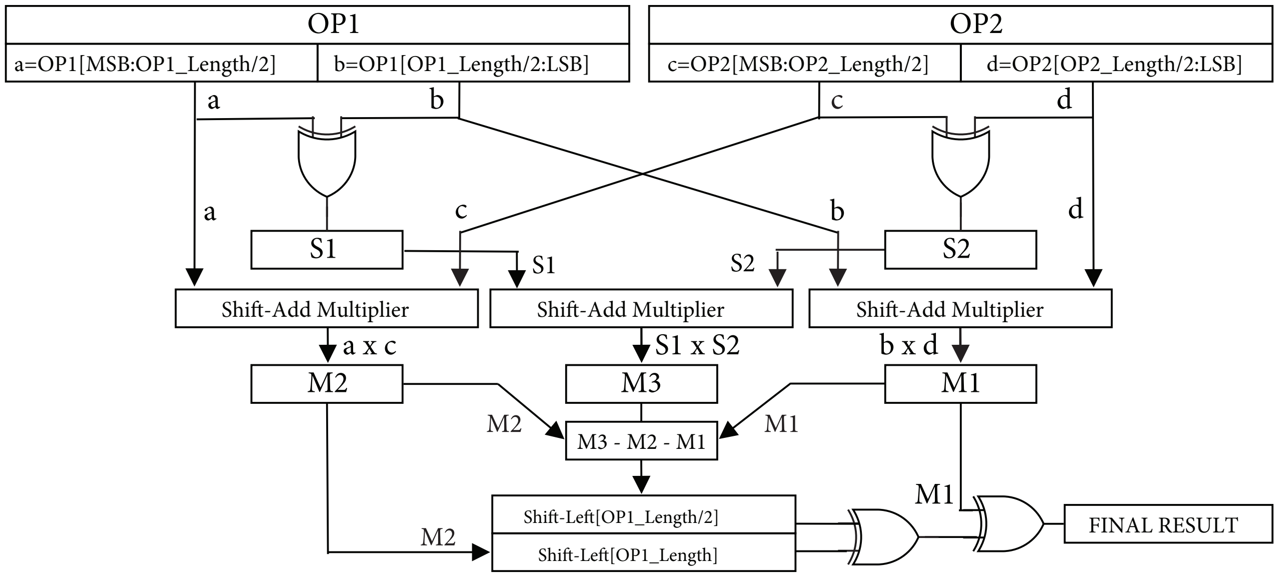

**Figure 2**. The proposed hybrid Karatsuba multiplier architecture.

## 5. Hardware implementation of the elliptic curve crypto-processor

We developed a Verilog model for the ECC accelerator at the RTL level. Then, we simulated the model using the Xilinx Vivado design suite and verified the correct functionality. In the testbench, we set a 233-bit test key

**Algorithm 2** Standard Karatsuba multiplier algorithm.

```
Input  : $OP1$ ($bit-length$ = OP1_length)
         $OP2$ ($bit-length$ = OP2_length)
Output: $OP1 \times OP2$
1: Step1: $a \longleftarrow OP1[MSB : (OP1\_length/2)]$
2:        $b \longleftarrow OP1[(OP1\_length/2) : 0]$
3:        $c \longleftarrow OP2[MSB : (OP2\_length/2)]$
4:        $d \longleftarrow OP2[(OP1\_length/2) : 0]$
5: Step2: $S1 \longleftarrow a \bigoplus b$
6:        $S2 \longleftarrow c \bigoplus d$
7: Step3: $M1 \longleftarrow ShiftAdd(d \times b)$
8:        $M2 \longleftarrow ShiftAdd(a \times c)$
9:        $M3 \longleftarrow ShiftAdd(S1 \times S2)$
10: Step4: $R \longleftarrow M3 - M1 - M2$
11:        $R \longleftarrow R << (OP1\_length / 2)$
12:        $R \longleftarrow R \bigoplus (M2 << OP2\_length)$
13:        $R \longleftarrow R \bigoplus M1$
```

**Algorithm 3** Shift-and-add multiplier algorithm.

```
Input  : $OP1$ ($bit-length$ = OP1_length)
         $OP2$ ($bit-length$ = OP2_length)
Output: $OP1 \times OP2$
1: for $OP1\_length$ down to 0 do
2:     if($OP1[i]$) then $Result = Result \bigoplus (OP2 << i)$
3:     $i = i - 1$
4: end for
```

based on alternating 1s and 0s (0x155...55) and used a 50 MHz test clock frequency for functional verification as shown in Figure 3. We have used test vectors from standard NIST recommended standard ECs to verify and confirm the correct operation. We have synthesized both the 2-stage pipeline and the no-pipeline variants and implemented them on various FPGAs to obtain the area and power estimations for comparing with the literature. The synthesis results showed promising improvements in area utilization, maximum achievable clock frequency and energy efficiency. The proposed ECC accelerator consumes 3.09 times fewer hardware resources in contrast to [27], and 1.52 times fewer hardware resources than the design introduced in [28]. Our design achieves an operational clock frequency of 119 MHz on a Xilinx Virtex-5 device that is 1.5 times higher than the implementation in [27], which could achieve only 79 MHz.

Our accelerator design with 2-stage pipeline for ECC over $GF(2^{233})$ has been implemented on a Xilinx Artix-7 FPGA device (XC7A35TICPG238-1L). Table 3 presents the implementation results, in which the no-pipeline architecture occupies only 4001 Slice LUTs, 2933 Slice FFs and achieves an operational clock frequency of 89 MHz. For the computation of one SPM operation, no-pipeline variant requires 1945 $\mu$s with an estimated power consumption of 87 mW. Proposed ECC accelerator with 2-stage pipeline uses 4467 Slice LUTs, 3,399 Slice FFs, and achieves a maximum frequency of 143 MHz with an estimated power consumption of 106 mW. Moreover, it only requires 1217 $\mu$s for one SPM computation. We calculated the energy efficiency per clock cycle using $E = \frac{Power \times Duration}{Clockcycles}$ and the calculated results are promising for both no-pipeline and 2-stage pipeline variants. Although 2-stage pipeline variant uses 466 additional Slice LUTs and Slice FFs in contrast to the no-pipeline counterpart, the energy efficiency is better due to higher clock frequency. In the computation of one scalar point multiplication, the 2-stage pipeline variant achieves 38% higher clock frequency and 24% lower execution time. To summarize, the proposed 2-stage pipeline architecture is 1.31 times faster than the no-pipeline architecture at the expense of 18% higher power consumption.

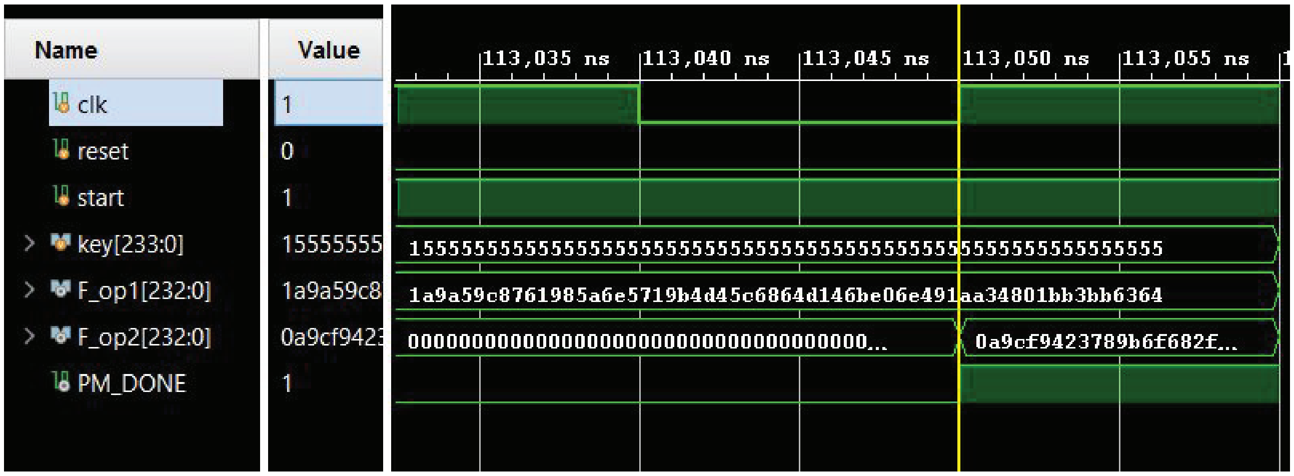

**Figure 3**. RTL Simulation results of the proposed ECC processor.

**Table 3**. Implementation results for Xilinx Artix-7 FPGA

| Architecture | Clock cycles | Slice LUTs | Slice FFs | Freq (MHz) | Time ($\mu s$) | Power (mW) | Energy (pJ/cycle) |
|---|---|---|---|---|---|---|---|
| no-pipeline | 173,154 | 4001 | 2933 | 89 | 1945 | 87.739 | 985 |
| 2-stage pipeline | 174,047 | 4467 | 3399 | 143 | 1217 | 106.852 | 747 |

Proposed ECC accelerator implements the first three layers of the ECC as a high-speed, area, and power-efficient hardware module. The fourth layer is more convenient for software implementations. To enable further testing of the ECC accelerator and also the Layer-4 operation, we have created a customized version of the ECC accelerator IP core and integrated it with the Xilinx microblaze CPU to create an ECC processor. The customized version of the ECC accelerator operates as a peripheral of the microblaze CPU over the AXI4 interface. Data read-write operations are performed through the memory-mapped registers of the AXI4 interface, which are accessible by a custom-developed driver software written in C using the Xilinx software development kit (SDK). Finally, we have also added a UART-Lite peripheral to the ECC processor to enable FPGA-PC communication as shown in Figure 4.

## 6. Performance evaluation

We synthesized the ECC accelerator design with 2-stage pipeline on Virtex-5 (XC5VFX200T) and Virtex-7 (XC7VX690T) FPGA devices using Xilinx ISE design suite for performance comparison with the recent state-of-the-art solutions. Table 4 shows the performance evaluation and comparison results. Comparing our architecture with the other architectures that support 163-bit key lengths, the work in [27] utilizes 16,116 FPGA LUTs and 5,341 FPGA slices and achieves an operational clock frequency of 79 MHz on Virtex-5. We achieved

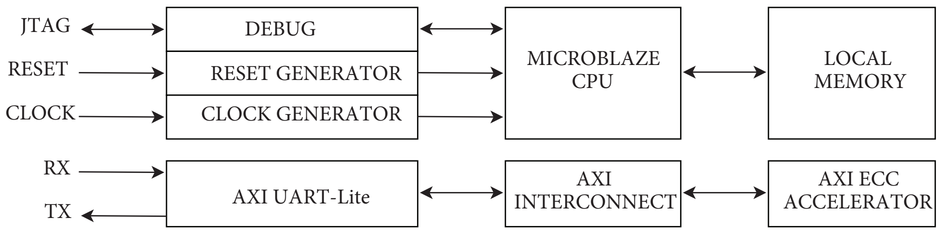

**Figure 4**. Hardware architecture of the ECC crypto-processor system.

better memory utilization as a result of the fewer number of storage elements used. Our ECC accelerator design with 2-stage pipeline needs only 1728 FPGA slices that are 3.09 times smaller than hardware resources consumed by [27]. Also, when compared to the design in [27], which operates at 79 MHz, our design achieved a higher clock frequency of 119 MHz on the same Virtex-5 device, thanks to its 2-stage pipeline architecture. The design in [28], utilizes 10,128 FPGA LUTs and 3,657 FPGA slices on a Virtex-7 FPGA device. In contrast, our ECC accelerator with 2-stage pipeline consumes 1.52 times (ratio of 3,657 with 2,403) fewer hardware resources (FPGA slices) and still achieves higher clock frequencies (148 vs. 135 MHz) when compared to [28] for the same FPGA device. The work in [29] utilizes 10863 FPGA slices on Virtex-5 (XC5VLX50) FPGA for $GF(2^{233})$. Our proposed design requires 6.29 times (ratio of 10,863 to 1,726) fewer hardware resources when compared to [29]. Although the design in [16] is 6% faster than ours, the cost of area utilization is quite high. It uses 11,849 slices on the target FPGAi, whereas our design with 2-stage pipeline consumes only the 2403 FPGA slices, which is 4.93 times smaller and performs almost fast. Although, the latency (computational duration) of the proposed ECC accelerator for one scalar point multiplication is high as an inevitable drawback of its pipeline architecture, our design outperforms existing solutions in every other aspect, especially in terms of area utilization and hardware efficiency. Consequently, proposed ECC accelerator is a high-speed, low power, and small footprint hardware design, which is very promising for use in emerging resource-constrained embedded systems, such as IoT applications, as a standalone IP core, or as a peripheral integrated with a microprocessor, such as Microblaze, ARM, or RISC-V.

**Table 4**. Comparison with ECC accelerator FPGA implementations in the literature.

| Ref #. Device | Key length | FPGA LUTs | FPGA Slices | $F_{clk}$ (MHz) | Time ($\mu$s) |
|---|---|---|---|---|---|
| [27] XC5VLX330T | 163 | 16,116 | 5,341 | 79 | 17.0 |
| [28] XC7VX690T | 163 | 10,128 | 3657 | 135 | 25.0 |
| [29] XC5VLX50 | 233 | 10,710 | 10,863 | - | - |
| [16] XC7VX690T | 233 | 21,453 | 11,849 | 157 | - |
| **This Work** | | | | | |
| XC5VFX200T | 233 | 6,912 | 1,728 | 119 | 1,462 |
| XC7VX690T | 233 | 9,612 | 2,403 | 148 | 1,175 |

## 7. Conclusion

In this work, we presented the design of an ECC accelerator with 2-stage pipeline for hardware efficient computation of the scalar point multiplication (SPM), which is the performance bottleneck. In order to optimize the SPM, we introduced a new hybrid scalar point multiplier based on a balanced combination of the standard Karatsuba multiplier with high-speed advantage and the shift-and-add multiplier that offers smaller area. The proposed Hybrid Karatsuba multiplier is designed and implemented mainly for light-weight embedded applications and consumes fewer hardware resources with a penalty of $(\frac{m}{2}-1)$ clock cycles for an m-bit ECC key-length when compared to the bit-parallel multiplier architectures. To avoid potential read-after-write hazards in the pipeline, careful ordering of the point addition and point doubling instructions has been scheduled to save clock cycles. The proposed ECC accelerator with 2-stage pipeline is 1.31 times faster than the no-pipeline variant and outperforms the other designs in the literature in terms of FPGA resource utilization and maximum achievable clock frequency. We have created an ECC processor by integrating our design with a synthesizable CPU to create a hardware platform for enabling the future development of ECC fourth layer applications that use key exchange protocols. Our keystone contribution is the design of an efficient and light-weight ECC accelerator architecture that unifies high-performance in a compact footprint suitable for FPGA and ASIC implementations.

## References

[1] Wadhe A, Sabhle N. Mobile SMS banking security using elliptic curve cryptosystem in binary field. Journal of Engg Res & App (IJERA) 2019; 3 (3): 413-420.

[2] Bai T P, Rabara S, Jerald A. Elliptic curve cryptography based securing framework for Internet of Things and cloud computing. In: WSEAS 2015 Conference on Recent Advances on Computer Engineering; Seoul, South Korea; 2015. pp. 65-74.

[3] Marin L, Pawlowski MP, Jara A. Optimized ECC implementation for secure communication between heterogeneous IoT devices. Sensors (Switzerland) 2015; 15 (9): 21478-21499.

[4] Koblitz N. Elliptic curve cryptosystems. Mathematics of Computation 1987; 48 (Jan): 203-209.

[5] Koblitz AH, Koblitz N, Menezes A. Elliptic curve cryptography: the serpentine course of a paradigm shift. Journal of Number Theory 2011; 131 (5): 781-814.

[6] Rashid M, Imran M, Jafri AR. Comparative analysis of flexible cryptographic implementations. In: 2016 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC); Talinn, Estonia; 2016. pp. 1-6.

[7] Trujillo-Olaya V, Sherwood T, Koç Ç K. Analysis of performance versus security in hardware realizations of small elliptic curves for lightweight applications. Journal of Cryptographic Engineering 2012; 2 (3): 179-188.

[8] Montgomery P. Speeding the pollard and elliptic curve methods of factorization. Mathematics of Computation 1987; 48 (Jan): 243-264.

[9] Imran M, Rashid M. Architectural review of polynomial bases finite field multipliers over GF($2^m$). In: 2017 Proceedings of IEEE International Conference on Communication, Computing and Digital Systems; Islamabad, Pakistan; 2017. pp. 331-336. March 2017.

[10] Kerry CF, Secretary A, Director CR. FIPS PUB 186-4 Federal Information Processing Standards Publication Digital Signature Standard (DSS), 2013.

[11] Karatsuba A, Ofman Y. Multiplication of multidigit numbers on automata. Soviet Physics Doklady 1963; 7: 595-596.

[12] Patel Z. Enhancing speed and reducing power of shift and add. International Journal Of Electrical, Electronics And Data Communication 2016; 4(Jun): 13-17.

[13] Rashid M, Imran M, Jafri AR, Kashif M. A Throughput/Area Optimized Pipelined Architecture for Elliptic Curve Crypto Processor. IET Computers & Digital Techniques 2019; 13 (5): 361-368.

[14] Hankerson D, Menezes A, Vanstone S. Guide to Elliptic Curve Cryptography. NY, USA: Springer-Verlag, 2004.

[15] Imran M, Kashif M, Rashid M. Hardware design and implementation of scalar multiplication in elliptic curve cryptography (ECC) over GF(2163) on FPGA. In: ICICT 2015 International Conference on Information and Communication Technologies; Karachi, Pakistan; 2015. pp. 4-7. 2016.

[16] Imran M, Shehzad F. FPGA Based Crypto Processor for Elliptic Curve Point Multiplication ECPM over GF $2^{233}$. International Journal for Information Security Research (IJISR) 2017; 7 (3): 706-713.

[17] Khan Z U A, Benaissa M. High Speed ECC Implementation on FPGA over GF $(2^{m})$ on FPGA. IEEE Transactions on Very Large Scale Integration Systems 2017; 25 (3): 165-176.

[18] Bonifus PL, George D. ECC Encryption Ssystem Uusing Encoded Multiplier and VEDIC Mathematics. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering 2013; 2 (11): 5531-5538.

[19] Sutter G, Deschamps J, Imana J. Efficient elliptic curve point multiplication using digit-serial binary field operations. IEEE Transactions on Industrial Electronics 2013; 60 (Jan): 217-225.

[20] Bednara M, Grabbe C, Teich J, Gathen JVZ, Shokrollahi J. FPGA Designs of Parallel High Performance GF $(2^{233})$ Multipliers. In: ISCAS 2003 Proceedings of the 2003 IEEE International Symposium on Circuits and Systems; Bangkok, Thailand; 2003. pp. II-268. Computer Engineering 2003: 268-271.

[21] Fan H, Hasan M. A survey of some recent bit-parallel GF$(2^{n})$ multipliers. Finite Fields and their Applications 2105; 32: 5-43.

[22] Jafri AR, Islam M, Imran M, Rashid M. Towards an Optimized Architecture for Unified Binary Huff Curves. Journal of Circuits, Systems and Computers 2017; 26 (11): 175-178.

[23] Imran M, Rashid M, Shafi I. Lopez Dahab based elliptic crypto processor (ECP) over GF(2163) for low-area applications on FPGA. In: ICEET 2018 International Conference on Engineering and Emerging Technologies; Lahore, Pakistan; 2018. pp. 1-6.

[24] Khan Z U, Benaissa M. High speed ECC implementation on FPGA over GF(2m). In: FPL 2015 25th International Conference on Field Programmable Logic and Applications; London, UK; 2015. pp. 1-6.

[25] Halak B, Waizi S S, Islam A. A Survey on Hardware Implementations of Elliptic Curve Cryptosystems. IACR Cryptology ePrint Archive 2016; 16 (712): .

[26] Itoh T, Tsujii S. A Fast algorithm for Computing Multiplicative Inverse in GF $(2^{m})$ Using Normal Bases. Information and Computation 1988; 78 (3): 171-177.

[27] Benselama ZA, Bencherif MA, Khorissi N, Bencherchali MA. Low cost reconfigurable Elliptic Crypto-hardware. In: AICCSA 2014 Proceedings of IEEE/ACS International Conference on Computer Systems and Applications; Doha, Qatar; 2014. pp 788-792.

[28] Imran M, Shafi I, Jafri A, Rashid M. Hardware design and implementation of ECC based crypto processor for low-area-applications on FPGA. In: ICOSST 2017 Proceedings of 2017 International Conference on Open Source Systems and Technologies; Lahore, Pakistan; 2018. pp. 54-59.

[29] Panchbhai M, Ghodeswar US. Implementation of point addition & point doubling for Elliptic Curve. In: ICCSP 2015 International Conference on Communication and Signal Processing; Chengdu, China; 2015. pp. 746-749.