IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 21, 2018, accepted April 5, 2018, date of publication April 18, 2018, date of current version May 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2828319

ACryp-Proc: Flexible Asymmetric Crypto Processor for Point Multiplication

MALIK IMRAN[1], MUHAMMAD RASHID[2], ATIF RAZA JAFRI[1], AND MUHAMMAD NAJAM-UL-ISLAM[1]

[1]Department of Electrical Engineering, Bahria University, Islamabad 44000, Pakistan
[2]Computer Engineering Department, Umm Al-Qura University, Makkah 21955, Saudi Arabia

Corresponding author: Muhammad Rashid (mfelahi@uqu.edu.sa)

ABSTRACT Flexibility is one of the driving agents for 5G architecture development to incorporate enhanced mobile broad band, machine type communication (MTC), and ultra-reliable MTC. While considering flexibility in the domain of security/reliability, we present a unified and flexible hardware architecture, implementing point multiplication algorithm for both elliptic curve cryptography (ECC) and Binary Huff Curves (BHC). The proposed architecture can be used in the scenarios where the users can tradeoff between the algorithmic execution time and different reliability/security levels. To establish an area overhead for the unified design, dedicated architectures for ECC and BHC over the $GF(2^m)$ field are implemented in the first step and compared with state-of-the-art. Then, a unified architecture for ECC and BHC is implemented. The performance results of proposed unified architecture illustrate the trade-off between execution time and security level.

INDEX TERMS Asymmetric crypto processor, binary huff curves, elliptic curve cryptography, point multiplication, unified architecture.

I. INTRODUCTION

Cryptography ensures that the data can be transmitted securely over an unsecured public channel. At present, there are two types of cryptography i-e., symmetric and asymmetric. Symmetric cryptography requires a single key to protect the data from eavesdroppers. However, the use of symmetric cryptography in highly secured applications, such as for military communication, is not helpful [1]. To achieve this high standard security, a group of three scientists i-e., Rivest, Shamir, and Adleman introduced asymmetric/public key cryptography, which is now commonly known as Rivest-Shamir-Adleman (RSA) algorithm [2].

The RSA algorithm (asymmetric cryptography) uses two separate keys for information protection. Out of these two keys, the first one is known as secret key while the other is called as public key. At sender side, the secret key is used for encryption process while the public key is used to decrypt the data/information at receiver end. Furthermore, RSA algorithm is based on prime factorizing. If the used primes are small, then searching their factors are quite easy for the computer, however, if the used primes are large, then searching their factors is difficult for computer in a reasonable time.

In order to overcome this issue, elliptic curve cryptography (ECC) was proposed in the mid 1980's [1]. In real-time embedded systems and area-limited portable electronic products, the use of ECC is continuously increasing [3], as it provides equivalent level of security with much shorter key lengths as compared to other commonly used asymmetric cryptosystem such as RSA algorithm [2]. Moreover, for the same level of security, ECC requires only 160 bit key length whereas RSA requires 1024 bits [1]. Due to shorter key lengths, ECC provide low computing power and higher throughput [3]. With the aforementioned facts, ECC has shown real attraction to portable devices and security of their systems.

Although, ECC provides better throughput than other existing asymmetric crypto systems, conventional ECC-based crypto systems are vulnerable to side channel attacks (SCAs) [3]. In order to provide resistance against SCAs, different asymmetric curves, such as Binary Edward Curves (BEC) [4] and Binary Huff Curves (BHC) [5], have been proposed. It has been observed in [4] and [5] that the required number of computations in BEC and BHC are much higher as compared to the conventional ECC. However, the comparison between BEC and BHC shows that the

later (BHC) requires less number of computations than former (BEC) [6]. A brief summary of comparison between BEC and BHC has been provided in Section III.III of this article. Consequently, ECC and BHC are more attractive choices for higher throughput and security level respectively. Therefore, one can trade-off between throughput and security level by using ECC and BHC simultaneously.

A. RELATED WORK

The point multiplication (PM) operation is normally implemented in hardware as it is the most computational intensive part of any asymmetric curve based crypto system (such as ECC and BHC). The hardware implementations of PM can be found in [7]–[13] and [14]–[16] for ECC and BHC respectively. These hardware implementations for ECC and BHC generally provide higher performance with larger area utilization.

1) HARDWARE IMPLEMENTATIONS OF PM FOR ECC

Generally speaking, the hardware implementations of PM for ECC are considered to achieve higher performance/ throughput [7]–[13]. Consequently, various techniques have been implemented. Examples of performance enhancement techniques include the use of multiple multipliers [7], parallelism either at design level [8] or algorithmic level [9] and pipelining [12]. In addition to the aforementioned techniques (use of multiple multipliers, parallelism and pipelining), some other implementations are also considered in [10], [11] and [13].

The work presented in [7] achieves higher performance with low latency (number of clock cycles) by using 3 multipliers. Similarly, the work in [8] obtains higher performance by implementing parallelism at design level using multiple cores. At algorithmic level, the parallelism for point addition (PA) and point doubling (PD) is considered in [9]. Moreover, higher performance is also achieved in [10] by considering lower area (FPGA slices) for constrained applications. Generally speaking, the performance of PM operation directly depends on the finite field (FF) multiplier [1]. Therefore, the work presented in [11] used a segmented FF multiplier whereas a pipelined FF multiplier with different digit sizes is implemented in [12]. Similarly, a bit parallel multiplier is used in [13] to obtain higher performance for PM computations.

2) HARDWARE IMPLEMENTATIONS OF PM FOR BHC

A comparative architectural review on different FF multipliers (bit serial, bit parallel and digit level) for multiple asymmetric curves (ECC, BHC and BEC) is presented in [17]. The results in [17] reveal that the use of bit parallel multiplier is a common trend to implement PM for BHC [14]–[16]. For example, by considering SCAs at algorithmic level, an FPGA based hardware architecture for PM is proposed in [14]. Furthermore, countermeasures against SCAs and Simple Power Analysis attacks (SPAs) at algorithmic level are provided in [15]. Recently, an FPGA based pipelined hardware architecture of BHC for PM is proposed in [16]. While individual architectures have been proposed for ECC and BHC, a unified/flexible hardware architecture for both ECC and BHC is required by considering the lower hardware resources.

3) APPLICATION DEMANDING FLEXIBLE ARCHITECTURES

From applications point of view, the major domains for asymmetric cryptography include internet, identity-based encryption [18], wireless sensor network nodes [19] and radio frequency identification networks (RFID) [20]. The newer application is in 5G, where sensor nodes will be used for machine type communication (MTC) and ultra-reliable MTC. Use-cases related to MTC include low-cost battery-powered sensors and actuators as well as remote controlled and remote-read utility meters to achieve Internet of Things (IoT) [21].

On the other hand, ultra-reliable MTC relates to the capability of providing a given service level with very high probability. The applications of ultra-reliable MTC include remote industrial control [22] and haptic communication [23]. These diverse applications demand flexibility of the architecture so that the system can be configured as per requirements. Considering a 5G base station who is responsible for providing connectivity for diverse 5G services. In such system the proposed flexible architecture can meet the varying security requirements associated with a particular service.

4) STATE-OF-THE-ART ON FLEXIBLE ARCHITECTURES

Broadly speaking, architectures connecting multiple devices with different configurations are known as flexible architectures. Internet of things (IoTs) is a typical example of this trend where multiple devices can communicate with each other [24]. In other words, flexibility is obtained in terms of multi algorithmic support at the same time. For example, a flexible architecture supporting advance encryption standard (AES) algorithm and elliptic curve digital signature algorithm (ECDSA) for security-enabled near field communication tags is presented in [25]. Similarly, a scalable architecture, without the need to reconfigure the hardware, is also considered as a flexible architecture [26]. For example, a crypto processor with multiple key length support for ECC is presented is known as scalable architecture.

Moreover, due to the re-configurability feature of FPGA, implementation of asymmetric crypto system on an FPGA platform is also considered as a flexible architecture [27]. Re-configurability refers to the custom hardware, which can be adapted at the run time, by loading new instructions/program. Recently, support for dual field (prime and binary) elliptic curve cryptographic processor is also considered as a flexible architecture in [27]. In this article, we achieve flexibility by implementing multi curves (ECC and BHC) as well as multi algorithms (Montgomery [28] for ECC and Double and Add [29] for BHC), on a reconfigurable FPGA platform.

B. OUR WORK

This article presents a flexible asymmetric crypto processor for PM operation, named as ''ACryp-Proc''. The proposed processor implements PM for ECC and BHC at the same time. The objective is to trade-off between execution time and the level of security/reliability. For both of the selected curves (ECC and BHC), the following selections have been adopted:

- A polynomial basis (PB) representation with the projective coordinate system (Lopez and Dahab) has been selected due to its simplicity in terms of FF multiplications and inversions [30].
- Montgomery algorithm [28] has been targeted due to its resistance against side channel attacks for ECC while simple Double and Add algorithm [29] has been chosen for BHC. A good comparative analysis over flexible implementations of these PM algorithms (Montgomery and Double and Add) can be found in [31].

With the above stated selections, we have modeled three hardware designs in Verilog (HDL), where the first one is an individual design for ECC (Design I), the second one is a dedicated design for only BHC (Design II) and the last one is a unified/flexible architecture for both ECC and BHC (Design III). The proposed designs (Design I, Design II and Design III) contain a memory unit, an arithmetic and logic unit and two routing networks. Arithmetic and logic unit for all the three designs contain a single FF adder, FF multiplier and FF squarer.

The novelty of proposed work can be described in the following points:

- Our earlier optimized architecture for unified BHC is available in [16]. However, the two important contributions in this work are: 1) a dedicated hardware architecture for ECC and 2) according to author's best knowledge, first unified hardware architecture for both ECC and BHC. Using unified architecture, one can trade-off between the execution time (i-e., throughput) and security level in terms of multi curve (ECC and BHC) as well as multi algorithms (Montgomery and Double and Add).
- To perform FF multiplication in a single clock cycle, a digit parallel multiplier is proposed in this work with a digit size of 32 bits. The proposed digit parallel multiplier uses least significant bit (LSB) order to perform FF multiplication and consumes less hardware resources than bit parallel multipliers such as hybrid Karatsuba [17].
- By considering lower hardware resources, a new arithmetic level scheduling of PA and PD operations to compute PM for conventional ECC and BHC is proposed. The proposed scheduling is performed by considering pipelining hazard such as read after write (RAW). Finally, to optimize the operational clock frequency, pipeline registers are placed at the input of arithmetic and logic unit.

The proposed flexible Design III (ACryp-Proc) is synthesized for Virtex 7 device whereas to perform exact comparison with state-of-the-art, Design I and Design II are synthesized for different FPGA technologies, i.e., Virtex 4, Virtex 5 and Virtex 7. Finally, the performance of proposed flexible and dedicated designs over $GF(2^{163})$ and $GF(2^{233})$ are estimated in terms of a throughput/slices metric. The achieved throughput/slices figures for our proposed dedicated and flexible designs on Virtex 7 are 28.53 (Design I-ECC for $GF(2^{163})$), 10.17 (Design II-BHC for $GF(2^{163})$), 10.73 (Design I-ECC for $GF(2^{233})$), 4.46 (Design II-BHC for $GF(2^{233})$), 15.64 (Design III-ECC for $GF(2^{163})$), 7.05 (Design III-BHC for $GF(2^{163})$), 5.42 (Design III-ECC for $GF(2^{233})$) and 2.43 (Design III-BHC for $GF(2^{233})$) and are comparable with relevant FPGA based designs.

The remainder of this article is organized as follows: In Section II, preliminaries of ECC and BH curves over $GF(2^m)$ are presented. In Section III, the selected point multiplication algorithms over $GF(2^m)$ for respective curves are explained. The proposed hardware architectures for our designs are presented in Section IV. Section V presents the FPGA synthesis results, performance of the architecture and results comparison. Finally, Section VI concludes the article.

II. PRELIMINARIES

This section provides the necessary background for the selected curves (ECC and BHC) and PM operation.

A. ELLIPTIC CURVE CRYPTOGRAPHY (ECC) OVER GF(2^m)

ECC provides two different fields for computations: prime field $GF(p)$ and binary field $GF(2^m)$. ECC over $GF(2^m)$ is selected as it provides the efficient hardware implementations of FF operations [30]. For $GF(2^m)$, a projective (Lopez and Dahab) form of elliptic curve is defined as a set of points $P(X : Y : Z)$, satisfying the following expression:

$$E : Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \tag{1}$$

In Eq. (1), the variables 'X', 'Y' and 'Z' are the Lopez and Dahab projective elements of point $P(X : Y : Z)$, $Z \neq 0$, 'a' and 'b' are the curve constants with $b \neq 0$. Furthermore, the points on elliptic curve form a group, called as an additive group, when they are combined with the '*point at infinity*'. By definition of additive group, the addition of two elements in the group defines another element in the group. For example, consider two distinct points i.e., point 'P' and 'Q' on the defined elliptic curve, then a point addition will be $R = P + Q$. Where 'R' will be the resultant (addition) of the two points on the curve. Moreover, when both the points on the curve are same ($P + P = 2P$) then this is called a point doubling.

Point addition and doublings are completely rely on the FF arithmetic operations [30]. Consider a base point 'P' and a large integer 'k' of the size of underlying field, the PM will be the additions of 'k' copies of point 'P' on the defined elliptic curve. In other words, PM is defined by repeating the additions, as illustrated in Eq. 2. It is normally a basic

equation for PM, where 'Q' is the resultant point.

$$Q = k.(P + P + \ldots + P) = k.P \tag{2}$$

B. BINARY HUFF CURVES (BHC) OVER GF(2^m)

Huff model was first introduced in 1963 [32]. Later on, the Huff model was revisited in 2010 [33], where the description and formulation for the odd characteristic fields were provided. Thereafter, in 2011, Devigne and Joye developed the formal construction of a Huff model for binary field [5]. Consequently, the formal construction in [5] provides the unified PA and PD formula in the binary field. A binary Huff curve is defined as a set of projective point $P(X : Y : Z)$ over $GF(2^m)$, satisfying Eq. (3):

$$E : aX(Y^2 + YZ + Z^2) = bY(X^2 + XZ + Z^2) \tag{3}$$

In Eq. (3), the variables 'a' and 'b' are the curve parameters and they ϵ $GF(2^m)$ while considering $a \neq b$. Interested readers can consult [14] and [15] for further mathematical formulations.

III. POINT MULTIPLICATION ALGORITHMS

We have described in the introductory part of this article that Montgomery algorithm is used for ECC while Double and Add algorithm is selected for BHC in the proposed designs. Consequently, this section briefly describes the Montgomery as well as the Double and Add algorithm.

A. MONTGOMERY ALGORITHM FOR ECC

The implemented point multiplication algorithm for ECC is presented in Algorithm 1. It requires scalar multiplier 'k' along with (x_p, y_p) coordinates of the initial point 'P' as an input and produces (x_q, y_q) coordinates of the final point 'Q' on the defined elliptic curve as an output. The Algorithm 1 contains three steps:

- Step 1 consists of initializations, where affine to projective conversions are performed.
- Step 2 is the PM, where PA ($P = P + Q$) and PD ($P = 2P$) instructions are performed, based on the inspected value of key (k_i).
- Step 3 consists of reconversion, where projective to affine conversions are performed.

B. DOUBLE AND ADD ALGORITHM FOR BHC

The implemented Double and Add algorithm for BHC is presented in Algorithm 2. It requires scalar multiplier 'k' along with (x_p, y_p) coordinates of the initial point 'P' as an input and produces (x_q, y_q) coordinates of the final point 'Q' on the defined binary huff curve as an output. The Algorithm 2 contains three steps:

- In Step 1, initializations are performed to convert affine co-ordinates into projective co-ordinates.
- Step 2 is the point multiplication, where Unif_Add instructions are performed, based on the inspected value of key (k_i).
- Step 3 consists of reconversion, where projective to affine conversions are performed.

Algorithm 1 Montgomery Algorithm [28] Over $GF(2^m)$

Input: $k = (k_{n-1}, \ldots, k_1, k_0)$ with $k_{n-1} = 1$, $P = (x_p, y_p)$ ϵ $GF(2^m)$

Output: $Q(x_q, y_q) = k.P$

Step 1 (Initializations): $X_1 = x_p$, $Z_1 = 1$, $X_2 = xp^4 + b$, $Z_2 = x_p^2$

Step 2 (Point Multiplication): for (i *from* $n-2$ *down to* 0) do

if ($k_i = 1$)		Else	
P = P + Q *Return* $P(X_1, Z_1)$	**P = 2P** ***Return*** $Q(X_2, Z_2)$	**P = P + Q** ***Return*** $P(X_2, Z_2)$	**P = 2P** ***Return*** $Q(X_2, Z_2)$
$Z_1 = X_2.Z_1$	$Z_2 = Z_2^2$	$Z_2 = X_1.Z_2$	$Z_1 = Z_1^2$
$X_1 = X_1.Z_2$	$T = Z_2^2$	$X_2 = X_2.Z_1$	$T = Z_1^2$
$T = X_1 + Z_1$	$T = b.T$	$T = X_2 + Z_2$	$T = b.T$
$X_1 = X_1.Z_1$	$X_2 = X_2^2$	$X_2 = X_2.Z_2$	$X_1 = X_1^2$
$Z_1 = T^2$	$Z_2 = X_2.Z_2$	$Z_2 = T^2$	$Z_1 = X_1.Z_1$
$T = x_p.Z_1$	$X_2 = X_2^2$	$T = x_p.Z_2$	$X_1 = X_1^2$
$X_1 = X_1 + T$	$X_2 = X_2 + T$	$X_2 = X_2 + T$	$X_1 = X_1 + T$

end if
end for

Step 3 (Reconversion):

$$x_q = \frac{X_1}{Z_1},\ y_q = (x_p + \frac{X_1}{Z_1})[(X_1 + x_p.Z_1)(X_2 + x_p.Z_2) + (x_p^2 + y_p)(Z_1.Z_2)](x_p.Z_1.Z_2) - 1 + y_p$$

The Unif_Add in step 2 of Algorithm 2 represents the following set of equations for PA and PD [15]. Moreover, X_3, Y_3 and Z_3 are the projective points on the defined curve, whereas 'α' and 'β' are the curve constants and can be computed as: $\alpha = (a + b)/b$ and $\beta = (a + b/a$.

C. COMPUTATIONAL COST OF PA AND PD

As explained in the introductory part of this article, PM is the most computational intensive part of any asymmetric curve based crypto system. However, the cost of PA and PD to compute PM in Lopez and Dahab projective coordinates for conventional ECC is $6M + 5S + 3A$ [30]. Similarly, the projective Lopez and Dahab cost of unified addition law (Unif_Add) for PM in BHC is $15M + 3S + 12A + 2D$ [5] and in BEC is $16M + 1S + 21A + 4D$ [4]. In aforementioned expressions, 'M' is the cost of a field multiplication, 'S' is the cost of a field squaring, 'A' is the cost of a field addition and 'D' is the cost of a field multiplication by a constant curve parameter.

By using single adder, multiplier and squarer units, the computations involved in the BEC (total of 42) and BHC (total of 32) are much higher as compared to the conventional ECC (total of 14) [14]. However, unified addition law of BHC requires less number of computations (total of 32) as compared to BEC (total of 42), while providing the same level of security [14]. Consequently, this article has established a trade-off between the time of execution and the level of security by using ECC and BHC at the same time. Furthermore,

Algorithm 2 Double and Add Algorithm [29] Over $GF(2^m)$

Input: $k = (k_{n-1}, \ldots, k_1, k_0)$ with $k_{n-1} = 1$, $P = (x_p, y_p)$ ϵ $GF(2^m)$
Output: $Q(x_q, y_q) = k.P$

Step 1 (Initializations): $X_1 = x_p$, $Z_1 = 1$, $X_2 = xp^4 + b$, $Z_2 = x_p^2$

Step 2 (Point Multiplication):
for (i *from* $n-2$ *down to* 0) do
$Q = Unif\_Add(Q, Q)$
if ($k_i = 1$) then
$Q = Unif\_Add\ (P, Q)$
end if
end for

Step 3 (Reconversion):
$x_q = \frac{X_2}{Z_2}$, $y_q = \frac{Y_2}{Z_2^2}$

Unif_Add [15]

$m_1 = X_1.X_2$, $m_2 = Y_1.Y_2$, $m_3 = Z_1.Z_2$, $m_4 = (X_1 + Z_1)(X_2 + Z_2)$, $m_5 = (Y_1 + Z_1)(Y_2 + Z_2)$
$m_6 = m_1.m_3$, $m_7 = m_2.m_3$, $m_8 = m_1.m_2 + m_3^2$, $m_9 = m_6\,(m_2 + m_3)^2$, $m_{10} = m_7\,(m_1 + m_3)^2$
$m_{11} = m_8(m_2 + m_3)$, $X_3 = \alpha m_9 + m_4.m_{11} + Z_3$, $Y_3 = \beta m_{10} + m_5.m_8(m_1 + m_3) + Z_3$
$Z_3 = m_{11}(m_1 + m_3)$

for exact mathematical based comparisons among BEC, BHC and ECC, interested readers can consult [4], [5], and [30] respectively.

IV. PROPOSED HARDWARE ARCHITECTURES

In this article, we are presenting three different 2-stage-pipelined architectures. The first pipeline stage is ''instruction fetch'' while the second pipeline stage is ''Execute and Write back''. Consequently, the three proposed designs are:

- Design I is architecture for point multiplication in ECC only.
- Design II is dedicated for point multiplication in BHC only.

ACryp-Proc (Design III) is the flexible design for point multiplication in ECC as well as in BHC. For all the three aforementioned designs, the hardware architectures are capable to perform PM over $GF(2^{163})$ and $GF(2^{233})$. The proposed pipelined architectures of Design I, II and III consist of a memory unit (MU), Routing Networks (RNs), Arithmetic and Logic Unit (ALU) and a dedicated Control Unit (CU), as shown in Figure 1, 2 and 3 respectively. In order to get the best frequency versus algorithmic execution clock cycle count compromise, the pipelined registers are placed at the input of ALU.

A. MEMORY UNIT (MU)

The memory unit of Design I is a register file of size '$8 \times m$', as shown in Figure 1. Whereas the size of register file in Design II and III is extended up to '$16 \times m$', due to the complexity of BHC, as shown in Figure 2 and Figure 3. Moreover, the value of 'm' (key length) in all the three designs (Design I, Design II and Design III) can be either 163 or 233. This unit is used to store the intermediate results, while implementing Algorithm 1 and Algorithm 2 for respective ECC and BHC curves. Furthermore, it contains two multiplexers (Mux M1 and Mux M2), which are used to fetch the operands (OP1 and OP_2) from MU and a single de-multiplexer (Demux) to modify the MU contents (Mplex_out).

B. ROUTING NETWORKS (RNs)

The proposed designs constitute two RNs as shown in Figure 1, Figure 2 and Figure 3: Mux M3 (RN 1) and Mux M4 (RN 2):

- Inputs to the Mux M3 are curve parameters and an operand from MU (OP1). The output of Mux M3 is an operand (OP_1) to the Arithmetic Logic Unit (ALU). The input curve parameters are different for different designs, according to the requirements for a particular curve. In Design III, the curve parameters for ECC and BHC are muxed through M5 and M6, as shown in Figure 3.
- Inputs to the Mux M4 are from the ALU and its output goes into MU. The corresponding input and output parameters for respective ECC and BHC designs are shown in Figure 1, Figure 2 and Figure 3 respectively.

C. ARITHMETIC AND LOGIC UNIT (ALU)

For all the three designs (Design I, II and III), ALU consists of adder, squarer and multiplier units. The adder is implemented through exclusive OR operations, whereas the FF polynomial squarer is implemented through inserting '0' after every bit of input [8], as illustrated in Figure 4. The index terms $a[0]$, $a[1]$, $a[m-2]$ and $a[m-1]$ represent the data bits of input polynomial A(x) and these bits are directly mapped on an output polynomial D(x) with placement of '0' between two successive data bits.

In order to perform multiplication of two 'm' bit polynomials ($A(x)$ *and* $B(x)$) over GF(2^m), different FF multipliers are discussed in [34]–[37]. For example, a serial digit level multiplier with digit size of 41 bits is implemented in [34], where each multiplication requires 4 clock cycles (CC). On the other hand, the work in [36] presents a bit level pipelined digit serial multiplier. In order to further improve the speed of PM operation, three bit level pipelined digit serial multipliers are used in [36]. Moreover, an efficient architecture to perform FF multiplication over both GF(p) and GF(2^m) fields is discussed in [37]. In this paper, we have implemented digit parallel least significant bit order multiplier with digit size of $s = 32$ bits, as shown in Figure 5. The digits with $s = 32$ bits of polynomial $B(x)$ is created (i.e., $B1$ *to* $B8$) and then parallel multiplication of each 's' bit digit with 'm' bit polynomial ($A(x)$) is performed by generating partial products. To compute FF multiplication operation over $GF(2^{163})$, a total of six digits are required. Out of these 6 digits, 5 digits are with 32 bit size whereas 1 digit is with 3 bit size. Similarly, for $GF(2^{233})$, a total of eight digits are

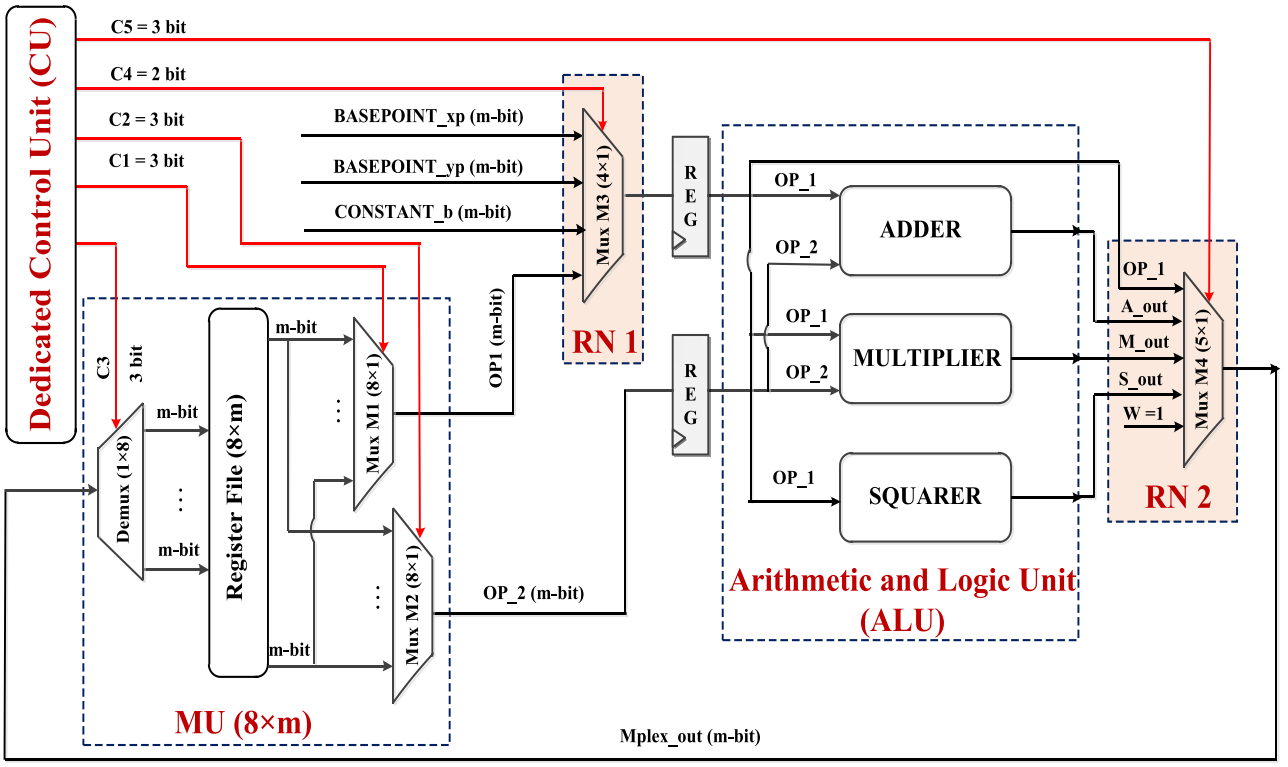

FIGURE 1. ECC design for point multiplication (Design I).

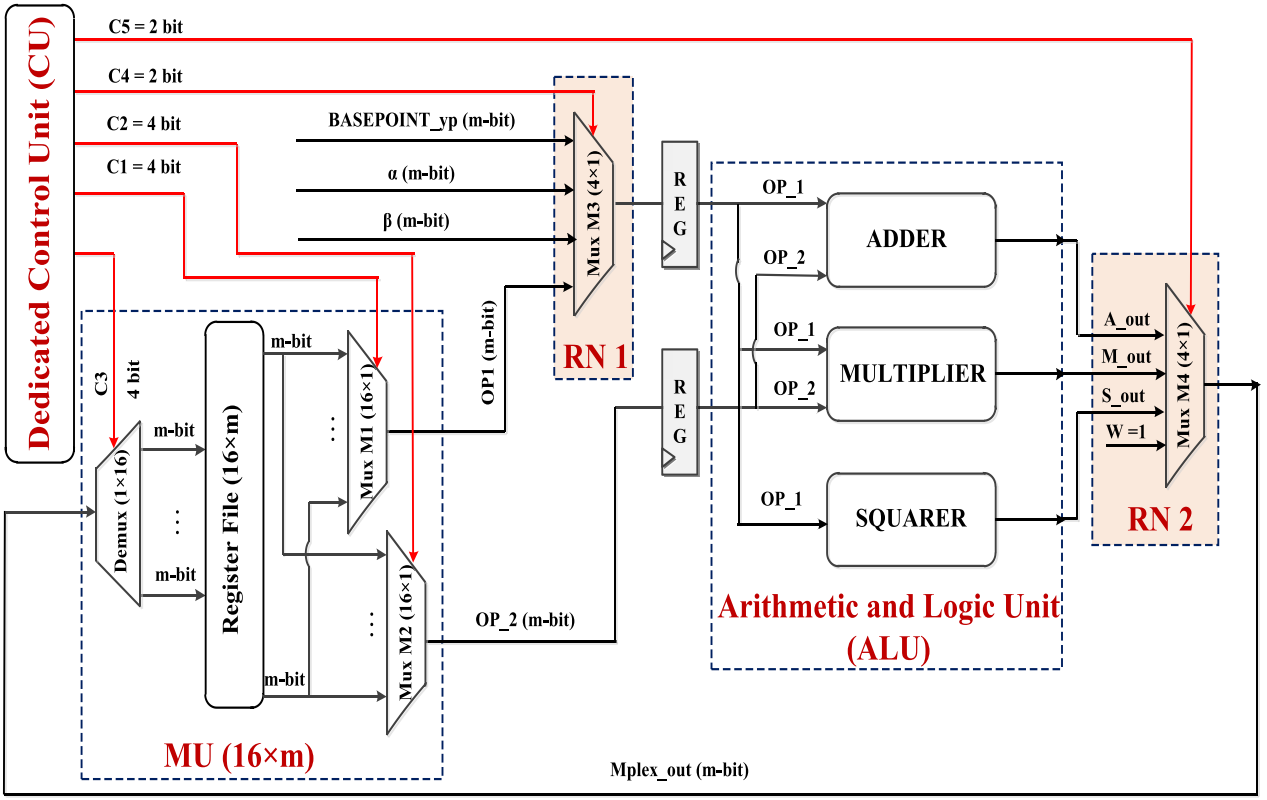

FIGURE 2. BHC design for point multiplication (Design II).

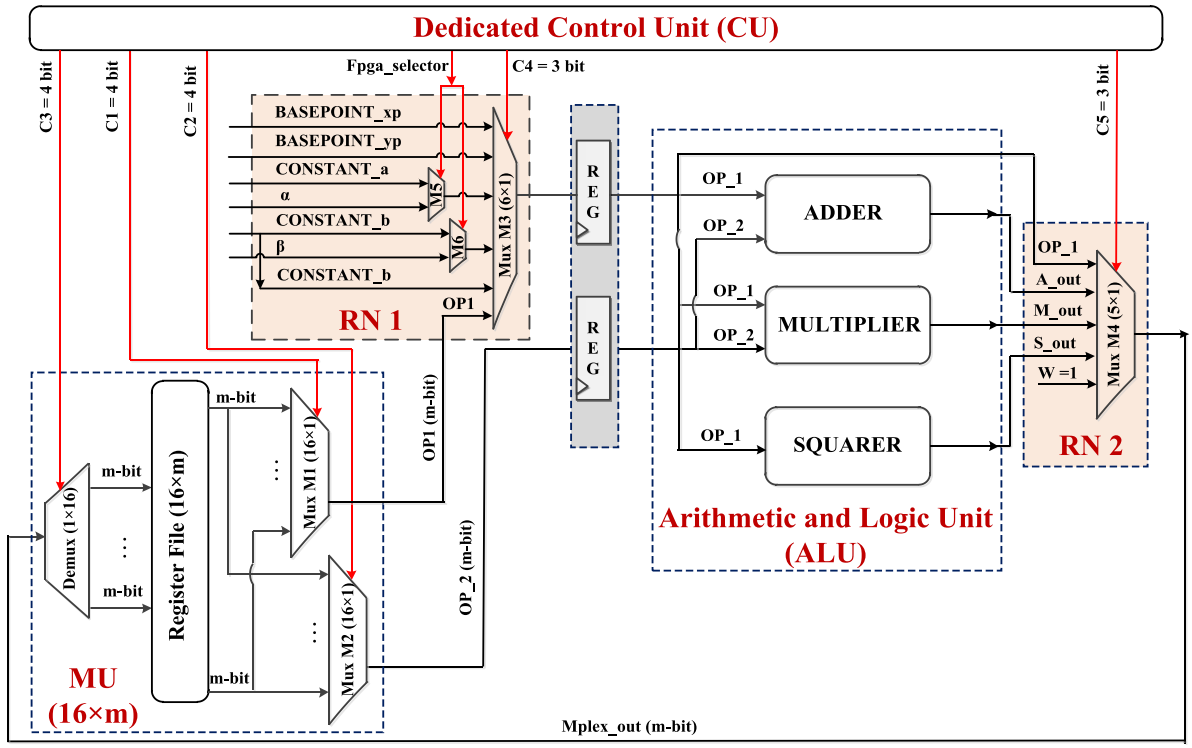

FIGURE 3. ACryp-Proc for point multiplication (Design III).

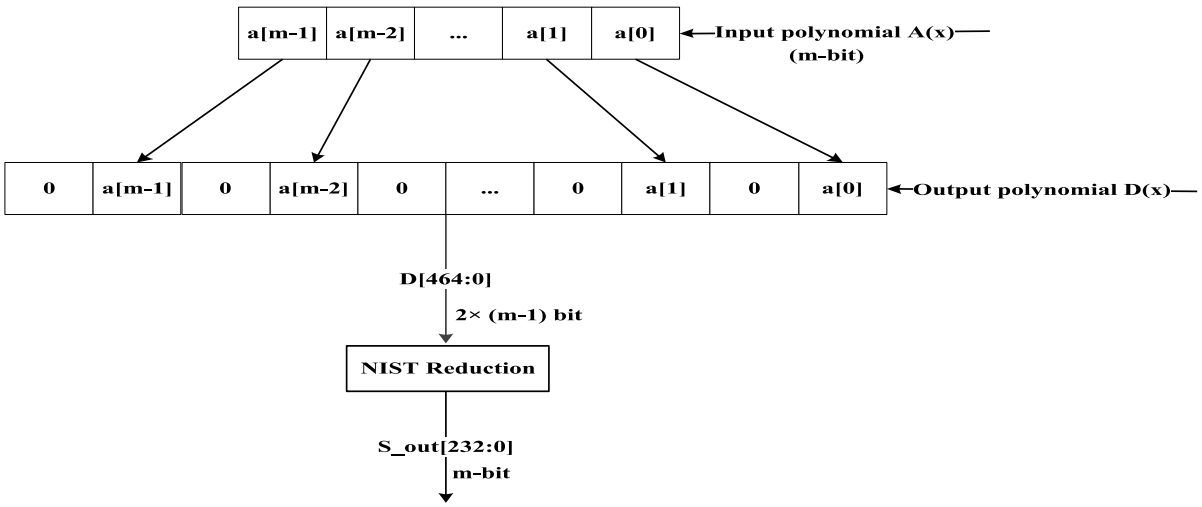

FIGURE 4. FF squarer.

required. Out of these 8 digits, 7 digits ($B1$ *to* $B7$) are with 32 bit size whereas 1 digit ($B8$) is with 9 bit size, as shown in Figure 5. Parallel multiplication of each $B1$ to $B8$ digit with an 'm' bit polynomial $A(x)$ results in '$s + m - 1$' bits of polynomials and these resultant polynomials are represented as $D1$ *to* $D8$, as shown in Figure 5. Once multiplication of each 's' bit digit with an 'm' bit polynomial is completed, the final resultant polynomial of size $2 \times m - 1$ bit is created by XOR and shift operations of $D1$ *to* $D8$.

As shown in Figure 4 and Figure 5, after each 'm' bit polynomial squaring and multiplication, the resulting polynomial will be $2 \times m - 1$, hence field reduction is necessary to perform [30]. In order to perform polynomial reduction, NIST recommended field reduction algorithms over $GF(2^{163})$ and $GF(2^{233})$ are implemented in this work and these algorithms are described as Algorithm 2.41 and 2.42 in [30]. For inversion operation, a scalable and unified architecture is proposed in [38] and [39], where GF(p) and GF(2^m) field

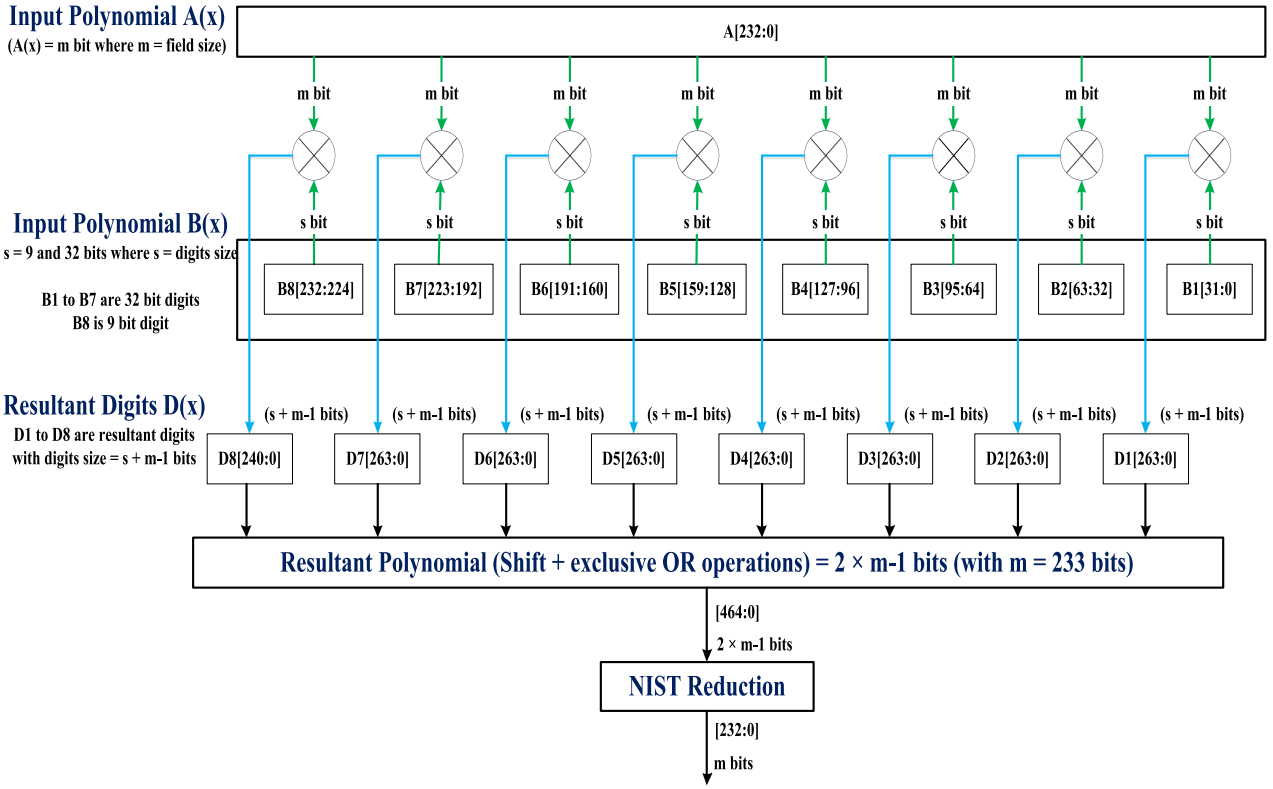

FIGURE 5. Parallel FF LSD multiplier.

inversions are computed using Montgomery modular inversion algorithm. However, Montgomery inversion requires higher clock cycles to perform a single inversion than Itoh-Tsujii algorithm [40]. Therefore, to perform FF inversion, a square Itoh-Tsujii algorithm has been implemented in this work by using same hardware resources of multiplier and squarer units. For $GF(2^m)$, it requires 'm' squarer operations whereas 9 and 10 field multiplications are required, when implementing from $GF(2^{163})$ to $GF(2^{233})$ respectively [41].

D. CONTROL UNIT (CU)

The control unit generates control signals in the design and shown as red color lines in the corresponding figures (Figure 1, Figure 2 and Figure 3). A 2-stage-pipelined data path has been used in this work to reduce the data hazards and minimize the no-operations (NOPs). A careful scheduling of PA and PD for Algorithm 1 and Algorithm 2 is presented in TABLE 1 and TABLE 2 respectively.

First column of TABLE 1 and TABLE 2 represents the clock cycle sequence. Instructions/operations performed during each cycle are represented in column 2 where X_1, X_2, Y_1, Y_2, Z_1, Z_2, m_1 to m_6 and T_1 to T_4 represent the intermediate projective points. The x-coordinate of initial point on the curve is represented by 'x_p' whereas 'b', 'α' and 'β' represents the curve constants. Column 3 shows the corresponding RAW hazard during different clock cycles. Finally, the last column shows the scheduling of instructions in two pipeline stages, where instruction read (fetch) is represented as [R] while execute and write back are represented as [E, WB].

TABLE 1. Proposed scheduling of PA and PD for algorithm 1.

CC	$Inst_i$ / Operation	hazard	Proposed instruction scheduling
1	$\boldsymbol{Inst_1/Z_1 = X_2.Z_1}$	-	$Inst_1$[R]
2	$\boldsymbol{Inst_2/X_1 = X_1.Z_2}$	-	$Inst_1$[E,WB], $Inst_2$[R]
3	$\boldsymbol{Inst_3/T_1 = X_1 + Z_1}$	RAW: X_1	$Inst_2$[E,WB], $Inst_8$[R]
4	$\boldsymbol{Inst_4/X_1 = X_1.Z_1}$	-	$Inst_8$[E,WB], $Inst_3$[R]
5	$\boldsymbol{Inst_5/Z_1 = T_1^2}$	-	$Inst_3$[E,WB], $Inst_4$[R]
6	$\boldsymbol{Inst_6/T_1 = x_p.Z_1}$	RAW: Z_1	$Inst_4$[E,WB], $Inst_5$[R]
7	$\boldsymbol{Inst_7/X_1 = X_1 + T_1}$	RAW: T_1	$Inst_5$[E,WB], $Inst_{11}$[R]
8	$\boldsymbol{Inst_8/Z_2 = Z_2^2}$	-	$Inst_{11}$[E,WB], $Inst_6$[R]
9	$\boldsymbol{Inst_9/T_1 = Z_2^2}$	RAW: Z_2	$Inst_6$[E,WB]
10	$\boldsymbol{Inst_{10}/T_1 = b.T_1}$	RAW: T_1	$Inst_7$[R]
11	$\boldsymbol{Inst_{11}/X_2 = X_2^2}$	-	$Inst_7$[E,WB], $Inst_9$[R]
12	$\boldsymbol{Inst_{12}/Z_2 = X_2.Z_2}$	RAW: X_2	$Inst_9$[E,WB], $Inst_{12}$[R]
13	$\boldsymbol{Inst_{13}/X_2 = X_2^2}$	-	$Inst_{12}$[E,WB], $Inst_{10}$[R]
14	$\boldsymbol{Inst_{14}/X_2 = X_2 + T_1}$	RAW: X_2	$Inst_{10}$[E,WB], $Inst_{13}$[R]
15	-	-	$Inst_{13}$[E,WB]
16	-	-	$Inst_{14}$[R]
17	-	-	$Inst_{14}$[E,WB]

TABLE 2. Proposed scheduling of PA and PD for algorithm 2.

CC	$Inst_i$ / Operation	Hazard	Proposed instruction scheduling
1	$Inst_1/m_1 = X_1 \times X_2$	-	$Inst_1$[R]
2	$Inst_2/m_2 = Y_1 \times Y_2$	-	$Inst_1$[E,WB], $Inst_2$[R]
3	$Inst_3/m_3 = Z_1 \times Z_2$	-	$Inst_2$[E,WB], $Inst_3$[R]
4	$Inst_4/T_1 = (X_1 + Z_1)$	-	$Inst_3$[E,WB], $Inst_4$[R]
5	$Inst_5/T_2 = (X_2 + Z_2)$	-	$Inst_4$[E,WB], $Inst_5$[R]
6	$Inst_6/m_6 = m_1 \times m_3$	-	$Inst_5$[E,WB], $Inst_6$[R]
7	$Inst_7/m_4 = T_1 \times T_2$	-	$Inst_6$[E,WB], $Inst_7$[R]
8	$Inst_8/T_1 = (Y_1 + Z_1)$	-	$Inst_7$[E,WB], $Inst_8$[R]
9	$Inst_9/T_2 = (Y_2 + Z_2)$	-	$Inst_8$[E,WB], $Inst_9$[R]
10	$Inst_{10}/x_q = m_2 + m_3$	-	$Inst_9$[E,WB], $Inst_{10}$[R]
11	$Inst_{11}/m_5 = T_1 \times T_2$	-	$Inst_{10}$[E,WB], $Inst_{11}$[R]
12	$Inst_{12}/T_1 = m_2 \times m_3$	-	$Inst_{11}$[E,WB], $Inst_{12}$[R]
13	$Inst_{13}/T_2 = m_1 + m_3$	-	$Inst_{12}$[E,WB], $Inst_{13}$[R]
14	$Inst_{14}/T_4 = x_q^2$	-	$Inst_{13}$[E,WB], $Inst_{14}$[R]
15	$Inst_{15}/T_2 = T_2^2$	-	$Inst_{14}$[E,WB], $Inst_{15}$[R]
16	$Inst_{16}/T_3 = m_3^2$	-	$Inst_{15}$[E,WB], $Inst_{16}$[R]
17	$Inst_{17}/T_1 = T_1 \times T_2$	-	$Inst_{16}$[E,WB], $Inst_{17}$[R]
18	$Inst_{18}/T_2 = m_1 \times m_2$	-	$Inst_{17}$[E,WB], $Inst_{18}$[R]
19	$Inst_{19}/X_2 = \beta \times T_1$	-	$Inst_{18}$[E,WB], $Inst_{19}$[R]
20	$Inst_{20}/T_2 = T_2 + T_3$	-	$Inst_{19}$[E,WB], $Inst_{20}$[R]
21	$Inst_{21}/T_3 = m_6 \times T_4$	-	$Inst_{20}$[E,WB], $Inst_{21}$[R]
22	$Inst_{22}/T_4 = T_2 \times x_q$	-	$Inst_{21}$[E,WB], $Inst_{22}$[R]
23	$Inst_{23}/x_q = \alpha \times T_3$	-	$Inst_{22}$[E,WB], $Inst_{23}$[R]
24	$Inst_{24}/y_q = m_4 \times T_4$	-	$Inst_{23}$[E,WB], $Inst_{24}$[R]
25	$Inst_{25}/m6 = m_5 \times T_2$	-	$Inst_{24}$[E,WB], $Inst_{25}$[R]
26	$Inst_{26}/m4 = x_q + y_q$	-	$Inst_{25}$[E,WB], $Inst_{26}$[R]
27	$Inst_{27}/x_q = m_1 + m_3$	-	$Inst_{26}$[E,WB], $Inst_{27}$[R]
28	$Inst_{28}/X_2 = m_4 + Z_2$	-	$Inst_{27}$[E,WB], $Inst_{28}$[R]
29	$Inst_{29}/m_2 = m_6 \times x_q$	-	$Inst_{28}$[E,WB], $Inst_{29}$[R]
30	$Inst_{30}/Z_2 = x_q \times T_4$	-	$Inst_{29}$[E,WB], $Inst_{30}$[R]
31	$Inst_{31}/m_5 = m_2 + T_1$	-	$Inst_{30}$[E,WB], $Inst_{31}$[R]
32	$Inst_{32}/y_q = m_5 + Z_2$	RAW: m_5	$Inst_{31}$[E,WB]
33	-	-	$Inst_{32}$[R]
34	-	-	$Inst_{32}$[E,WB]

1) CONTROL UNIT FOR DESIGN I (ECC)

To implement the Montgomery algorithm, the CU incorporates a total of 121 states (St), as shown in Figure 6.

- The St: 0 is an idle state whereas St: 1 to St: 6 are responsible to implement the initialization step of Algorithm 1 which requires 6 clock cycles. These initialization states are termed as ''affine to projective conversion states'' in Figure 6.
- The proposed scheduling of PA and PD for PM step of Algorithm 1, shown in TABLE 1, requires a total of 35 states. Out of these 35 states, 17 states are for PA and PD when inspected key bit is '1' (shown in TABLE 1). Similarly, an additional 17 states are required for PA and PD if the inspected key bit is '0'. Furthermore, St: 7 is a conditional state and is used to count the number of points on the specified ECC curve by using count signal and also used to check the inspected key bit. Count has an initial value of '$m-1$'. St: 8 to St 24 (17 states) are used if the inspected key bit is '1' (IF part of Algorithm 1 in Section 3). St: 25 to St: 41 (17 states) are used if the inspected bit for key is '0' (Else part of Algorithm 1 in Section 3).
- Finally, the reconversion step of Algorithm 1 requires two FF inversion (Inv) operations, as shown in Algorithm 1. To check the status of required inversion operations a single bit 'inv_1' signal is used. This signal is checked at last state of inversion operation i-e., at St: 87 to define one of the next state either 88 or 117. When inv_1 signal is '0' then next state after the completion of first inverse will be St: 88 otherwise next state will be St: 117. In addition to inversion operations, additional 34 cycles are required to complete the reconversion step. Each inversion requires St: 42 to St: 87 for implementations, St: 116 to generate the addresses for second inversion and remaining states (St: 88 to St: 115 and St: 117 to St: 120) performs rest of the operations in the reconversion step of Algorithm 1.

TABLE 3. Clock cycles information for design I, II and III.

Curve	Key length (m)	Clock cycles to execute algorithm			
		Initial	PA + PD	Reconversion	Total cycles
Design I - Algorithm 1					
ECC	163	6	2916	1038 (Inv 502)	3960
	233	6	4176	1452 (Inv 709)	5634
Design II - Algorithm 2					
BHC	163	6	8262	508 (Inv 502)	8776
	233	6	11832	715 (Inv 709)	12553
Design III (ACryp-Proc) - Algorithm 1					
ECC	163	6	2916	1038 (Inv 502)	3961
	233	6	4176	1452 (Inv 709)	5635
Design III (ACryp-Proc) - Algorithm 2					
BHC	163	6	8262	508 (Inv 502)	8777
	233	6	11832	715 (Inv 709)	12554

Design I requires a total of 3960 clock cycles when implemented over $GF(2^{163})$, where 6 cycles are needed for initializations, 2916 cycles are required to perform PA and PD. Finally, the reconversion requires 1038 cycles where 502 cycles are required for each inverse operation. Similarly, for $GF(2^{233})$, the Design I covers a total of 5634 clock cycles. Exact information about the clock cycles for Design I is tabulated in TABLE 3, and it can also be calculated by using Eq. (4):

$$Initial + 18\,(m-1) + 2(Inv) + 34 \tag{4}$$

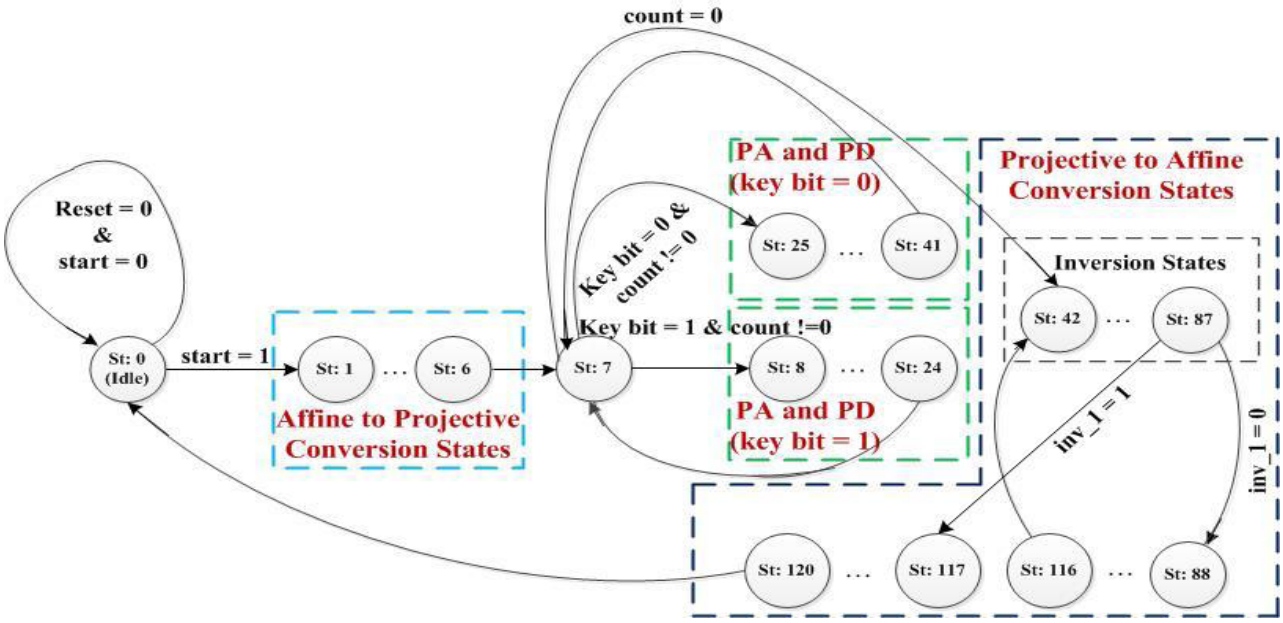

FIGURE 6. CU for Design I.

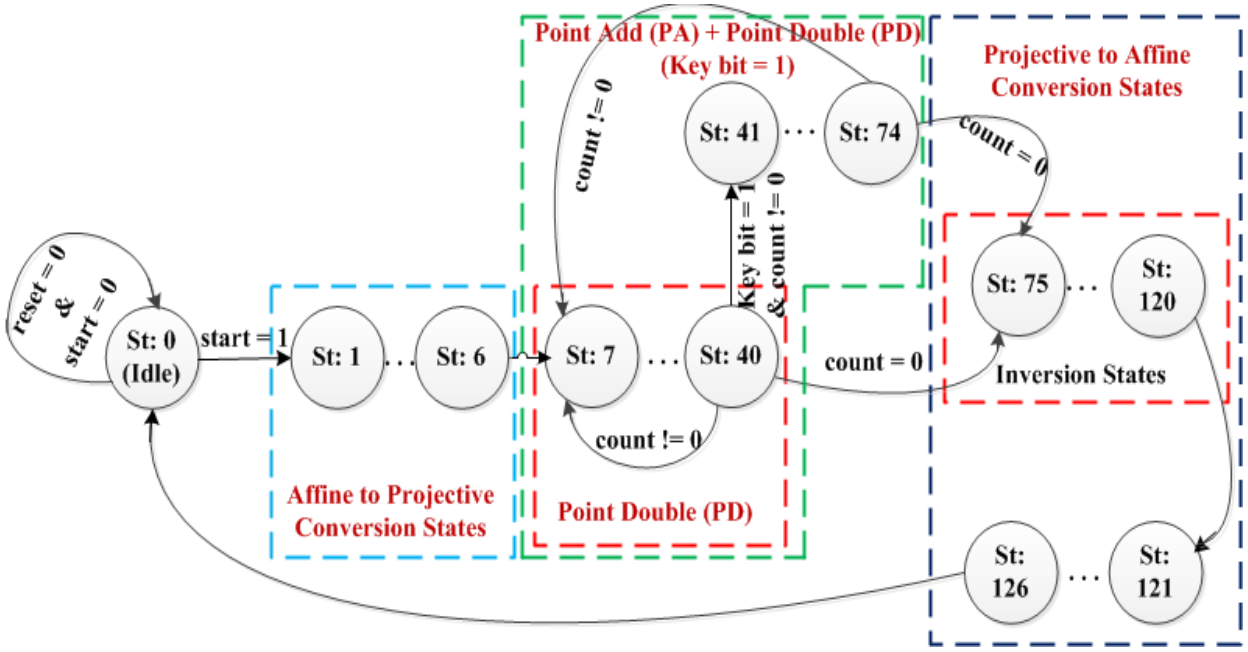

FIGURE 7. CU for Design II.

2) CONTROL UNIT FOR DESIGN II (BHC)

In order to implement Double and Add algorithm (Algorithm 2) for PM, the CU consists of 127 states, as shown in Figure 7.

- The St: 0 is an idle state, while during St: 1 to St: 6, CU generates control signals for initialization (Initial) step of Algorithm 2. These initialization states are termed as "affine to projective conversion states" in Figure 7.
- The proposed scheduling of PA and PD (Unif_Add) for PM step of Algorithm 2 requires a total of 68 states. Out of these 68 states, 34 states (St: 7 to St: 40) are used for PD. Similarly, additional 34 states (St: 41 to St: 74) are required for PA (shown in TABLE 2) operation. In order to perform either "PD" or "PA with PD", St: 40 is used to check the inspected key bit and the number of points on the curve by using count signal (initially $count = m - 1$). For each inspected key bit (either '0' or '1'), PD states must be computed. PA is

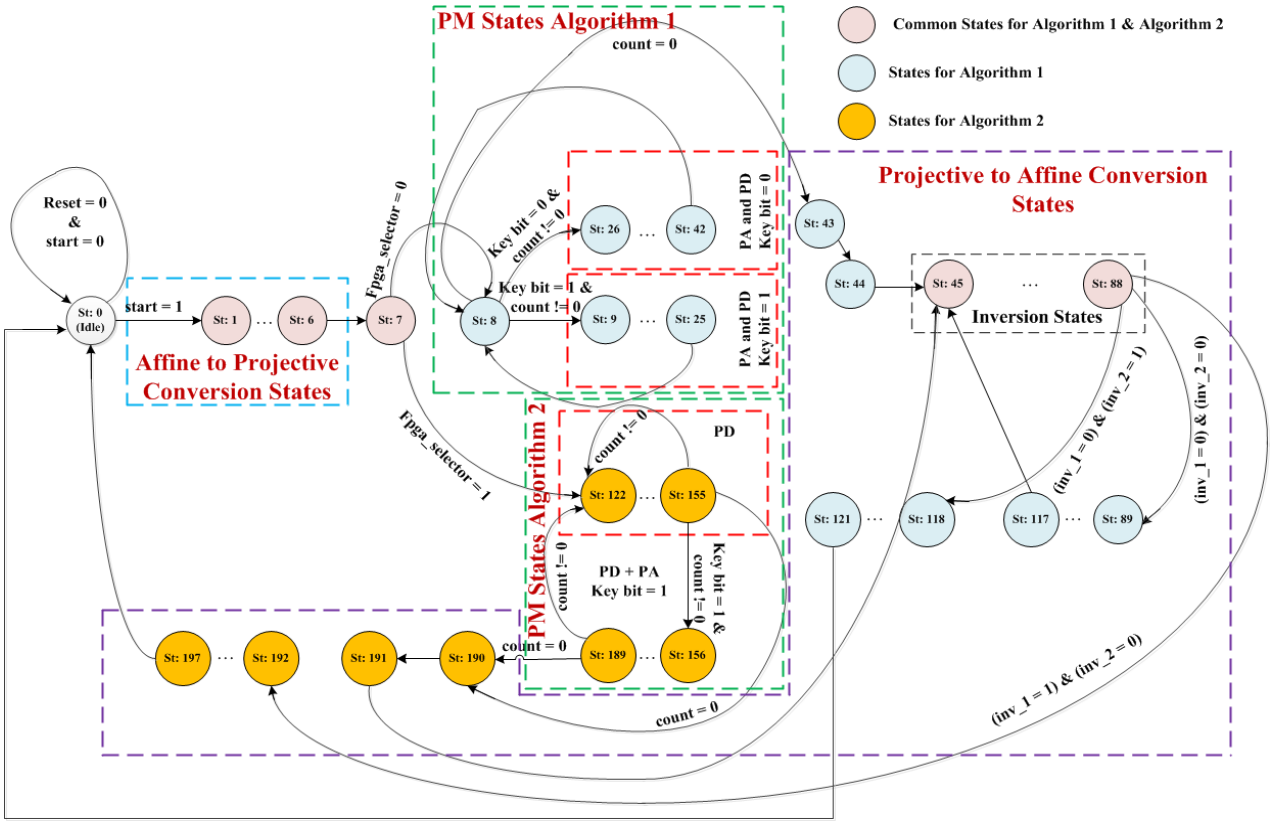

FIGURE 8. Control Unit for ACryp-Proc (Design-III).

only performed when the inspected key bit is '1' and count is not equal to '0' (Key bit = '1' & $count$! = '0'). In addition, the last state of PD and PA (St: 40 and St: 74) are also responsible to check the status of count signal. If count becomes '0' then next state from St: 40 and St: 74 will be St: 75 otherwise it will be St: 7.

- Finally, during St: 75 to St: 126 coordinate reconversion step of Algorithm 2 including finite field inversion (Inv) is performed, as shown in Algorithm 2. Each inversion requires St: 75 to St: 120 and additional 6 states (St: 121 to St: 126) are used to implement the rest of the reconversion step of Algorithm 2.

To compute PM, Algorithm 2 requires a total of 8776 clock cycles when implemented over $GF(2^{163})$. In total 6 cycles are required for initializations step of Algorithm 2 whereas for most time consuming part i-e., for PA or PD the proposed design requires 8262 clock cycles. Finally, the reconversion step requires a total of 508 cycles where 502 cycles are required to compute FF inversion while rest of the 6 cycles perform other instructions as presented in Algorithm 2. Furthermore, Design II requires a total of 12553 cycles for implementation over $GF(2^{233})$. Total clock cycles for the proposed Design II are illustrated in TABLE 3 and it can also be calculated by using Eq. (5):

$$Initial + 34\,(m-1) + 34\left(\frac{m}{2} - 1\right) + inv + 6 \qquad (5)$$

In order to perform point multiplication, the CU of flexible ACryp-Proc (Design III) contains a total of 198 states to generate the control signals, either for ECC (Algorithm 1) or BHC (Algorithm 2), as illustrated in Figure 8. Out of these 198 states, there are some common states (St: 1 to St: 7 and St: 45 to St: 88) for both Algorithms (Algorithm 1 and Algorithm 2). In addition to these common states, St: 8 to St: 44 and St: 89 to St: 121 are used to implement Algorithm 1, whereas St: 122 to St: 197 are responsible to implement Algorithm 2.

3) CONTROL UNIT FOR DESIGN III (ACryp-Proc)

- The St: 0 is an idle state, whereas St: 1 to St: 6 are responsible to generate control signals for initialization step of either Algorithm 1 or Algorithm 2. These initialization states are termed as "affine to projective conversion states" in Figure 8.
- In order to select an appropriate curve (ECC or BHC), State 7 is a conditional state, which is used to make the decision for scheduling of PA and PD instructions, either from TABLE 1 (for ECC) or from TABLE 2 (for BHC), based on the value of Fpga_selector signal (user defined input). If Fpga_selector signal is '0' then PA and PD instructions are scheduled from TABLE 1 for ECC (Algorithm 1). Otherwise, instructions are scheduled from TABLE 2 for BHC (Algorithm 2).

- To perform PM, the PA and PD states from St: 8 to St: 42 (35 states) are performed for Algorithm 1. Out of these 35 states, 17 states (St: 9 to St: 25) are responsible for PA and PD once the inspected key bit is '1' and with count not equal to '0' (count is used to check the number of points on the specified curve and initially it contains a value of '$m-1$') whereas rest of the 17 states (St: 26 to St: 42) are used for PA and PD when inspected bit for key is '0' with count not equal to '0'. Similarly, Algorithm 2 contains St: 122 to St: 189 (68 states) for PA and PD operations. Out of these 68 states, 34 states (St: 122 to St: 155) are responsible to compute PD operation for each inspected key bit (either '0' or '1'). If inspected key bit is '1' then PA states (St: 156 to St: 189) are also performed. A two conditional states St: 8 (for Algorithm 1) and St: 155 (for Algorithm 2) are also responsible to check the inspected key bit and count signals.
- Finally, to perform coordinate reconversion step of both algorithms, 3 inverse operations are required to compute. Out of these 3 inverse operations, 2 inverse operations are required for Algorithm 1 (Section 3.1) whereas a single inverse is required for Algorithm 2 (Section 3.2). For each inverse operation the common states are St: 45 to St: 88, whereas with these common states two additional states are required to generate addresses, for Algorithm 1 St: 43 to St: 44 (for first inverse computation), St: 116 to St: 117 (for second inverse operation) and for Algorithm 2 St: 190 to St: 191. To check the status of the required inversion operations, CU incorporates a two additional control signals named as, inv_1 and inv_2. These signals are checked in the last inversion state (St: 88) and defines one of the next state either St: 89, St: 118 or St: 192. When both inv_1 and inv_2 are '00' then next state after first inverse completion for Algorithm 1 will be St: 89 whereas when these signals are '01' (inv_1 $=$ 0 and inv_2 $=$ 1) then next state after second inverse completion for Algorithm 1 will be St: 118. Moreover, when inv_1 and inv_2 are '10' then CU defines the next state (St: 192) for Algorithm 2 after inverse completion. Finally, St: 89 to St: 115 and St: 118 to St: 121 are responsible to perform rest of the operations required for reconversion step of Algorithm 1 whereas St: 192 to St: 197 are responsible to perform other operations, which are required for reconversion step of Algorithm 2.

To achieve flexibility, Design III requires a total of 3961 and 5635 clock cycles when implemented Algorithm 1 for ECC over GF(2^{163}) and GF(2^{233}) respectively, as shown in TABLE 5. Similarly, Algorithm 2 requires a total of 8777 and 12554 clock cycles for BHC implementation over GF(2^{163}) and GF(2^{233}) respectively. Design III has only 1 cycle penalty for flexibility and is available in the last column of TABLE 5, as compared with dedicated designs (Design I and Design II). The 1 cycle penalty is due to St: 7 where selection between ECC (Algorithm 1) and BHC (Algorithm 2) is required. The total clock cycles for the ACryp-Proc are tabulated in TABLE 3 and it can be calculated by using Eq. 6 (ECC) and Eq. 7 (BHC).

$$Initial + 18\,(m-1) + 1 + 2(Inv) + 34 \tag{6}$$

$$Initial + 34\,(m-1) + 34\left(\frac{m}{2} - 1\right) + 1 + Inv + 6 \tag{7}$$

V. PERFORMANCE RESULTS AND COMPARISON WITH STATE OF THE ART IMPLEMENTATIOS

A. PERFORMANCE RESULTS

In order to evaluate the performance of a hardware architecture, parameters such as number of clock cycles to execute the targeted algorithm, the highest achievable frequency and the hardware resource utilization are required. Hence, in totality, 6 Verilog HDL models have been created to evaluate the performance of our proposed architecture. The first two models belong to Design I and implement ECC over GF(2^{163}) and GF(2^{233}). The next two designs are part of Design II to implement BHC over GF(2^{163}) and GF(2^{233}). Finally, the last two models (Design III) for ACryp-Proc have been created by allowing flexibility in terms of distinct asymmetric curves and algorithms for both ECC and BHC over GF(2^{163}) and GF(2^{233}).

The initial curve parameters for respective ECC and BHC curves have been selected from NIST [42]. In order to perform PM using BHC, pre-computed values for 'α' and 'β' are used in this work. The proposed dedicated (Design I and II) and flexible (Design III) models were synthesized for Xilinx Virtex 7 (xc7v585t-3ffg1157) device using ISE design suite.

The execution time in terms of number of clock cycles for all the three designs (Design I, II and III), executing two curves with two different key lengths are arranged in TABLE 3 using expression (4), (5), (6) and (7). In TABLE 3, first and second columns show the implemented curve and key length (m). Third column is showing clock cycle for execution of an algorithms which is further divided into information related to clock cycles required for Initial, PA + PD, Reconversion and overall processing.

On the other hand, synthesis result of all 6 HDL models using Virtex 7 FPGA are tabulated in TABLE 4. The first and second columns show the corresponding curve and implemented key length (m) respectively. The information about hardware utilizations in terms of FPGA slices (Slices), look up tables (LUTs) and flip flops (FFs) are reported in column 3, 4 and 5 respectively. The achievable frequencies are shown in column 6. Time and power consumptions for one PM ($k.P$) are provided in column 7 and 8 respectively. In order to evaluate the performance of both dedicated and flexible designs, throughput/area ratio is computed for each design, as shown in the second last column. In throughput/area ratio, throughput is computed using the time required (in μsec) to perform one PM ($k.P$) whereas FPGA slices are used for area. Finally, the last column indicates the power used by each hardware model.

TABLE 4. Implementation results for Design I, II and III on Xilinx Virtex 7.

Curve	Key length (m)	Slices	LUTs	FFs	Freq. MHz	K.P (µs)	10^6/s/slice	Power (W)
Design I								
ECC	163	3107	10955	1968	351	11.28	28.53	0.35273
	233	5674	19753	2764	343	16.42	10.73	0.58664
Design II								
BHC	163	3440	11745	3197	307	28.58	10.17	0.37708
	233	6083	22254	4927	341	36.81	4.46	0.62984
Design III (ACryp-Proc)								
ECC	163	4520	12113	3161	280	14.14	15.64	0.46611
BHC						31.34	7.05	
ECC	233	8866	23017	4414	271	20.78	5.42	0.75557
BHC						46.32	2.43	

The main contributions for this work is to, first of all, propose efficient individual architectures to perform PM with ECC and BHC and secondly to realize a flexible architecture (ACryp-Proc). It can be seen from TABLE 4 that our efficient dedicated designs achieve a throughput/area ratio of 28.53 and 10.73 for ECC and 10.17 and 4.46 for BHC over GF(2^{163}) and GF(2^{233}) respectively. On the other hand, the Acryp-Proc gives performance metric equals to 15.64 and 7.05 for ECC and 5.42 and 2.43 for BHC over GF(2^{163}) and GF(2^{233}) respectively.

As far as power consumption is concerned, it can be seen that putting two separate hardware of ECC and BHC will consumes more power than using a unified architecture for two applications e-g. in case of key length of 163, the sum of power consumption of dedicated hardware of ECC and BHC is more than twice the power consumed by ACryp-Proc. This feature also supports the idea of using a unified architecture.

Using the results presented in Table 3 and 4, we can discuss the tradeoff among parameters such as execution time, area utilization and security features of proposed architectures. If we consider the execution speed in terms of clock cycles, as presented in TABLE 3, for the key length $m = 163$, the ratio of clock cycled to execute BHC and ECC is $8776/3960 = 2.2$. Hence, more than twice clock cycles are required to get immunity against SCA using BHC as compared to ECC for key length of 163. Almost same factor of extra clock cycles is required in case of $m = 233$. On the other hand, in Design I and II the ratio of FPGA slices used by BHC and ECC for $m = 163$ is 1.1 whereas for $m = 233$ this ratio is 1.07. Finally if we look into the throughput/area ratios, BHC has 0.35 times throughput/area ratio of ECC (10.17/28.53) for $m = 163$ for provision of immunity against SCA. For $m = 233$ BHCs throughput/area ratio is 0.41 times as that of ECC. In flexible implementation of ACrypProc we get better results i-e. for $m = 163$ and 233 BHC's throughput/area ratio is almost 0.45 times as that of ECC to gain immunity against SCA. With this analysis we can see that using unified architecture we can achieve a compromise between the throughput and the immunity against SCAs.

B. COMPARISON WITH STATE OF THE ART IMPLEMENTATIONS

1) DEDICATED VERSUS UNIFIED DESIGNS

While comparing dedicated and flexible architectures shown in TABLE 4, we have considered two system configurations:

- A system comprised of individual hardware of ECC and BHC (i-e. Design-I and Design-II)
- A system with a single flexible ACryp-Proc.

It can be seen from TABLE 4 that 3107 and 3440 slices are used to realize the hardware in order to compute PM in ECC and BHC for GF(2^{163}) respectively. Hence, in total, 6547 slices are considered to be used in the first system configuration. On the other hand, Design III (ACryp-Proc) only uses 4520 slices. It can be deduced that the flexible design uses 2027 (31%) lesser slices for GF(2^{163}). Similarly, for GF(2^{233}) the ACryp-Proc uses 25% less hardware resources (slices) as compared to the combined resources of dedicated implementations of ECC and BHC.

To compare the throughput/area performances of above-stated two configurations, we need to first find equivalent throughput/area ratio of first system configuration (i-e. system made up of Design-I and II on single FPGA). Consider the case of GF(2^{163}) for first system configuration where total of 6547 slices are used. If PM for ECC is performed, then 3960 clock cycles (TABLE 3) of a clock having a frequency of 351 MHz (TABLE 4) is used. Here the throughput/area ratio will be $10^6/(3960/351)/6457 = 13.54$. Similarly, for BHC with GF(2^{163}) it will be 5.34. For GF(2^{233}), the joint slices will be 11757 and the throughput/area figures will be 5.18 and 2.31 for ECC and BHC respectively. If we compare these four values of throughput/area ratios with four through/area figures achieved through ACryp-Proc, it can be seen that ACryp-Proc gives better throughput/area ratios.

2) DEDICATED DESIGNS VERSUS STATE-OF-THE-ART

In the absence of other flexible implementations, we have compared our dedicated designs i-e. Design-I and Design-II with relevant state of the art solutions. In order to ensure a fair comparison, we have synthesized our designs on the same FPGA device which is used in a design with which we are doing the comparison. This is shown in TABLE 5.

The first column of TABLE 5 presents the reported designs, implemented either for ECC or BHC. The implemented key length 'm' is shown in column 2. FPGA devices for implementations are shown in column 3. Used slice resources, clock cycles and highest achievable clock frequency are presented in column 4, 5 and 6 respectively. The time required for one PM is tabulated in column 7. Finally, the throughput/area ((10^6/s)/slices) ratio (targeted performance metric in this article) is presented in the last column of TABLE 5. At first, we will compare our Design-I for ECC with relevant works presented in [7]–[13]. In our design, we have made efforts to maximize the overall throughput/area ratio.

The best design in terms of (10^6/s)/slices for GF(2^{163}) is presented in [7], which is implemented on Virtex 5

TABLE 5. Comparisons with state-of-the-art under same conditions.

Designs	m	FPGA device	Slices	Clock Cycle	Freq. (MHz)	K.P (µs)	10^6/s/slice
			For ECC				
[7]	163	XC5VLX110	11777	450	113	3.99	21.28
[8]	163	XC4VLX80	20807	1428	185	7.71	6.23
[9]	163	XC4VLX100	12834	3372	196	17.20	4.53
[9]	163	XC5VLX110T	6536	3380	262	12.90	11.86
[10]	163	Virtex 7	4665	52012	800	65.00	3.29
[11]	163	Virtex 5	10363	781	153	5.10	18.92
[12]	163	XC4VLX200low	17929	2751	250	9.60	5.80
[13]	233	XC4VFX140	20917	1870	64	29.00	1.64
Proposed Design I	163	XC5VLX110	2956	3960	247	16.03	**21.10**
	163	XC4VLX80	8019	3960	211	18.76	**6.64**
	163	XC4VLX100	8019	3960	212	18.67	**6.67**
	163	XC5VLX110T	2850	3960	228	17.36	**20.21**
	163	XC7V585T	3107	3960	351	11.28	**28.53**
	163	XC4VLX200	8022	3960	189	20.95	**5.95**
	233	XC4VFX140	16352	5634	162	34.77	**1.75**
			For BHC				
[14]	233	XC4VFX140	20437	5913	81	73.00	0.67
[15]	233	Virtex 7	6032	7370	183	40.27	4.11
Proposed Design II	233	XC4VFX140	17410	12553	162	77.48	**0.74**
	233	XC7VX585T	6083	12553	341	36.81	**4.46**

(XC5VLX110) for ECC. In [7], two hardware architectures were proposed for ECC to implement PM operation. Out of these two architectures, the first one was proposed to achieve high speed with a single pipelined segmented multiplier whereas the second one was proposed to achieve low latency with three pipelined segmented multipliers. The key feature of their architecture is low latency (i.e. 3.99 μs) as well as throughput (i.e. 250 K $k.P$/sec) but requires 4 times FPGA slices as compared to our solution. However, in totality, they achieve a (10^6/s)/slices figure of 21.28 which is quite comparable with Design I having a throughput/area ratio of 21.10 for the same technology. It can therefore be claimed that our design supports scalability and hence multiple instances of our Design-I working concurrently can provide same throughput as presented in [7] at same hardware cost. In all other designs i-e. from [8]–[13] our throughput/area ratio is higher.

On the same FPGA Virtex 5 (XC5VLX110T) device, higher hardware resources are utilized in [9] by considering parallelism at algorithmic level using multiple arithmetic operators i-e., two FF adders, multipliers and square units. On the other hand, the proposed design uses only a single adder, multiplier and squarer units. Due to use of multi arithmetic operators in [9], the utilized FPGA slices is 6536 which is comparably 57% more than Design I (2850) whereas the achieved (10^6/s)/slices ratio is 11.86 which is comparatively 41% lesser than Design I (20.21). Similarly, on the same Virtex 5 device, higher hardware resources are also reported in [11] by implementing a multi stage pipelined bit parallel multiplier whereas a digit parallel multiplier is used in this work. By multi stage pipelined multiplier, the number of required pipeline registers increases, which results in more resource utilizations. The utilized FPGA slices (10363) in [11] is comparably 73% more than Design I (2850). The achieved (10^6/s)/slices ratio in [11] is 18.92 which is almost 6% lesser than Design I (20.21). Parallelism at design level using three FF cores results in higher hardware resources, as shown in [8]. While comparing with [8] under the same conditions on Virtex 4 (XC4VLX80), our design consumes 38% lower FPGA slices and provides (10^6/s)/slices ratio of 6.64 whereas this ratio is reported as 6.23 in [8]. On Virtex 4 (XC4VLX100), Design I achieves 37% higher (10^6/s)/slices ratio (6.67) than the work reported in [9] which is 4.53.

In [12], the digit level multiplier with different digit sizes is used to perform PM. The optimal digit size of 55 bits is obtained in [12]. On the other hand, this work implements the digit parallel multiplier with digit size of 32 bits. Higher digits (55 bits) in [12] results higher hardware resources than the lower digits (32 bits) in the proposed work. However, on Virtex 4 (XC4VLX200), FPGA slices used in [12] is 55% (17929) more than Design I which is only 8022. Furthermore, the Design I achieves 3% higher (10^6/s)/slices ratio of 5.95 as compared to [12] which is 5.80. In order to reduce hardware resources, a bit serial FF multiplier is used in [10] where multiple clock cycles ('m' clock cycles for 'm' bit key length) are required to generate the final result. Using a bit serial multiplier on Virtex 7 in [10], the utilized FPGA resources are 4665 (slices) which is 34% higher than our work (3107) and the (10^6/s)/slices ratio of our work is 28.53 which is 89% higher than the presented work in [10]. Furthermore, for GF(2^{233}), the proposed Design I provides 6.3% higher performance than [13] on Virtex 4 (XC4VFX140).

For BHC, we can compare our Design-II with the work presented in [14] and [15]. Bit parallel multipliers (hybrid karatsuba) are used in [14] and [15] whereas a digit parallel

multiplier is used in our work. Hybrid karatsuba performs FF multiplication in a chronological order (such as multiplication from lower to higher bits). Therefore, multiplications with larger bits in hybrid Karatsuba utilizes higher hardware resources and results longer delays [17]. As a result, the proposed Design II outperforms the solutions presented in [14] and [15] by achieving almost 9.5% and 8% higher (10^6/s)/slices ratio than the work presented in [14] and [15] respectively.

VI. CONCLUSION

By considering flexibility in terms of security/reliability, a novel flexible hardware architecture has been presented in this work. For flexibility, the proposed architecture provides multi curve (ECC and BHC) as well as multi algorithmic (Montgomery and Double and Add) support. For exact comparison with state-of-the-art, we have proposed the dedicated designs for both the curves. By utilizing the same hardware resources, flexibility directly influences the cost and performance of the hardware while dedicated hardware provides higher performance. The proposed flexible hardware enables its usage where the users can tradeoff between the algorithmic execution time and different reliability/security levels.

REFERENCES

[1] J. W. Bos, C. Costello, P. Longa, and M. Naehrig, "Selecting elliptic curves for cryptography: An efficiency and security analysis," *J. Cryptogr. Eng.*, vol. 6, no. 4, pp. 259–286, 2016.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[3] H. Marzouqi, M. Al-Qutayri, and K. Salah, "Review of elliptic curve cryptography processor designs," *Microprocess. Microsyst.*, vol. 39, no. 2, pp. 97–112, 2015.

[4] D. J. Bernstein, T. Lange, and R. R. Farashahi, "Binary Edwards curves," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), vol. 5154. Berlin, Germany: Springer, 2008, pp. 244–265.

[5] J. Devigne and M. Joye, "Binary huff curves," in *Topics in Cryptology—CT-RSA* (Lecture Notes in Computer Science), vol. 6558. Berlin, Germany: Springer, 2011, pp. 340–355.

[6] Z. Lijun, W. Kunpeng, and W. Hong, "Unified and complete point addition formula for elliptic curves," *Chin. J. Electron.*, vol. 21, no. 2, pp. 345–349, 2012.

[7] Z. U. A. Khan and M. Benaissa, "High-speed and low-latency ECC processor implementation over GF(2^m) on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 1, pp. 165–176, Jan. 2017.

[8] Y. Zhang, D. Chen, Y. Choi, L. Chen, and S.-B. Ko, "A high performance ECC hardware implementation with instruction-level parallelism over GF(2^{163})," *Microprocess. Microsyst.*, vol. 34, no. 6, pp. 228–236, 2010.

[9] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 8, pp. 1453–1466, Aug. 2012.

[10] T. T. Nguyen and H. Lee, "Efficient algorithm and architecture for elliptic curve cryptographic processor," *J. Semicond. Technol. Sci.*, vol. 16, no. 1, pp. 118–125, 2016.

[11] Z. U. A. Khan and M. Benaissa, "High speed ECC implementation on FPGA over GF(2^m)," in *Proc. 25th Int. Conf. Field Programm. Logic Appl. (FPL)*, 2015, pp. 1–6.

[12] H. Mahdizadeh and M. Masoumi, "Novel architecture for efficient FPGA implementation of elliptic curve cryptographic processor over GF(2^{163})," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 12, pp. 2330–2333, Dec. 2013.

[13] C. Rebeiro and D. Mukhopadhyay, "High speed compact elliptic curve cryptoprocessor for FPGA platforms," in *Proc. 9th Int. Conf. Cryptol.*, 2008, pp. 376–388.

[14] A. Chatterjee and I. Sengupta, "High-speed unified elliptic curve cryptosystem on FPGAs using binary huff curves," in *Progress in VLSI Design and Test* (Lecture Notes in Computer Science), vol. 7373. Berlin, Germany: Springer, 2012, pp. 243–251.

[15] S. Ghosh, A. Kumar, A. Das, and I. Verbauwhede, "On the implementation of unified arithmetic on binary huff curves," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), vol. 8086. Berlin, Germany: Springer, 2013, pp. 349–364.

[16] A. R. Jafri, M. N. Ul Islam, M. Imran, and M. Rashid, "Towards an optimized architecture for unified binary huff curves," *J. Circuits, Syst. Comput.*, vol. 26, no. 11, p. 1750178, Nov. 2017.

[17] M. Imran and M. Rashid, "Architectural review of polynomial bases finite field multipliers over GF(2^m)," in *Proc. Int. Conf. Commun., Comput. Digit. Syst.*, 2017, pp. 331–336.

[18] G. Hanaoka and S. Yamada, "Survey on identity-based encryption from lattices," in *Mathematical Modelling for Next-Generation Cryptography*, vol. 29. Singapore: Springer, 2018, pp. 349–365.

[19] L. Sujihelen and C. Jayakumar, "Inclusive elliptical curve cryptography (IECC) for wireless sensor network efficient operations," *Wireless Pers. Commun.*, vol. 99, no. 2, pp. 893–914, 2018.

[20] K. Leng, L. Jin, W. Shi, and I. Van Nieuwenhuyse, "Research on agricultural products supply chain inspection system based on Internet of Things," *Cluster Comput.*, pp. 1–9, Feb. 2018.

[21] N. Assaf, B. Alkazemi, and A. A.-A. Gutub, "Applicable light-weight cryptography to secure medical data in IoT systems," *J. Res. Eng. Appl. Sci.*, vol. 2, no. 2, pp. 50–58, 2017.

[22] A. F. M. Piedrahita, V. Gaur, J. Giraldo, Á. A. Cárdenas, and S. J. Rueda, "Leveraging software-defined networking for incident response in industrial control systems," *IEEE Softw.*, vol. 35, no. 1, pp. 44–50, Jan./Feb. 2017.

[23] N. Pedemonte, T. Laliberté, and C. Gosselin, "A haptic bilateral system for the remote human–human handshake," *J. Dyn. Syst., Meas., Control*, vol. 139, no. 4, p. 044503, 2017.

[24] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[25] T. Plos, M. Hutter, M. Feldhofer, M. Stiglic, and F. Cavaliere, "Security-enabled near-field communication tag with flexible architecture supporting asymmetric cryptography," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 11, pp. 1965–1974, Nov. 2013.

[26] A. A.-A. Gutub and A. F. Tenca, "Efficient scalable VLSI architecture for Montgomery inversion in GF(p)," *Integr., VLSI J.*, vol. 37, no. 2, pp. 103–120, May 2004.

[27] M. Rashid, M. Imran, and A. R. Jafri, "Comparative analysis of flexible cryptographic implementations," in *Proc. 11th IEEE Int. Symp. Reconfigurable Commun.-Centric Syst. Chip (ReCoSoC)*, Jun. 2016, pp. 1–6.

[28] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, no. 177, pp. 243–264, Jan. 1987. [Online]. Available: http://www.jstor.org/stable/2007888?seq=1#page_scan_tab_contents

[29] *Elliptic Curves-Double and Add Algorithm*. Accessed: Jan. 2018. [Online]. Available: http://www.hyperelliptic.blogspot.com/2009/06/double-and-add-algorithm.html

[30] J. López and R. Dahab, "Fast multiplication on elliptic curves over GF(2^m) without precomputation," in *Cryptographic Hardware and Embedded Systems—CHES)* (Lecture Notes in Computer Science), vol. 1717. Berlin, Germany: Springer, 1999, pp. 316–327.

[31] Z. Liu, D. Liu, and X. Zou, "An efficient and flexible hardware implementation of the dual-field elliptic curve cryptographic processor," *IEEE Trans. Ind. Electron.*, vol. 64, no. 3, pp. 2353–2362, Mar. 2017.

[32] D. L. Huff, "A probabilistic analysis of shopping center trade areas," *Land Econ.*, vol. 39, no. 1, pp. 81–90, 1963.

[33] M. Joye, M. Tibouchi, and D. Vergnaud, "Huff's model for elliptic curve," in *Algorithmic Number Theory* (Lecture Notes in Computer Science), vol. 6197. Berlin, Germany: Springer, 2010, pp. 234–250.

[34] Z.-U.-A. Khan and M. Benaissa, "Throughput/area-efficient ECC processor using Montgomery point multiplication on FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 11, pp. 1078–1082, Nov. 2016.

[35] A. A.-A. Gutub, "GF(2^K) elliptic curve cryptographic processor architecture based n bit level pipelined digit serial multiplication," in *Proc. ACS/IEEE Int. Conf. Comput. Syst. Appl.*, Jul. 2003, p. 12.
[36] A. A.-A. Gutub, "Fast elliptic curve cryptographic processor architecture based on three parallel GF(2^k) bit level pipelined digit serial multipliers," in *Proc. 10th IEEE Int. Conf. Electron., Circuits Syst.*, Jan. 2004, pp. 72–75.
[37] A. A.-A. Gutub, "Area flexible GF(2k) elliptic curve cryptography coprocessor," *Int. Arab J. Inf. Technol.*, vol. 4, no. 1, pp. 1–10, 2007.
[38] A. A.-A. Gutub, A. F. Tenca, E. Savaş, and C. K. Koç, "Scalable and unified hardware to compute Montgomery inverse in GF(p) and GF(2^n)," in *Cryptographic Hardware and Embedded Systems—CHES* (Lecture Notes in Computer Science), vol. 2523. Berlin, Germany: Springer, 2003, pp. 484–499.
[39] A. A.-A. Gutub, "High speed hardware architecture to compute galois fields GF(p) Montgomery inversion with scalability features," *IET Comput. Digit. Techn.*, vol. 1, no. 4, pp. 389–396, Jul. 2007.
[40] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases," *Inf. Comput.*, vol. 78, no. 3, pp. 171–177, 1988.
[41] M. Imran, I. Shafi, A. R. Jafri, and M. Rashid, "Hardware design and implementation of ECC based Crypto processor for low-area-applications on FPGA," in *Proc. 11th IEEE Int. Conf. Open Source Syst. Technol.*, Dec. 2018, pp. 54–59.
[42] (1999). *National Institute of Standards and Technology: Recommended Elliptic Curves for Federal Government Use*. [Online]. Available: http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf

MALIK IMRAN received the bachelor's degree in computer engineering from the COMSATS Institute of Information Technology, Abbottabad, Pakistan, in 2011, and the M.S. degree in telecommunication and networks from Abasyn University, Islamabad, Pakistan, in 2015. He is currently with Bahria University as a Teaching Assistant for assisting in programming fundamentals course and lab work. Apart from teaching activities, he is involved in the research related to efficient hardware solutions for asymmetric cryptography.

MUHAMMAD RASHID received the Ph.D. degree in embedded systems design from the University of Bretagne Occidentale, Brest, France, in 2009. He was with Thomson Research and Development, Paris, France. He is currently with the Computer Engineering Department, Umm Al-Qura University, Makkah, Saudi Arabia. His research is mainly focused on electronic design automation for embedded systems.

ATIF RAZA JAFRI received the B.Sc. degree in electrical engineering from UET Lahore, Pakistan, in 1999, the M.S. degree from the University of Nice Sophia Antipolis, France, in 2007, and the Ph.D. degree from Telecom Bretagne, Brest, France, in 2011. He served with the Research and Development Sector for over 10 year. He has been with the Electrical Engineering Department, Bahria University, Islamabad, Pakistan, since 2015, where he is currently serving as an Associate Professor. His research interests include hardware architectures implementation for wireless communication and cryptographic applications using both HDL and HLS. He is currently engaged in the research related to architectures of physical layer of future wireless communications and ultra-light weight and asymmetric cryptography.

MUHAMMAD NAJAM-UL-ISLAM received the bachelor's degree in electrical engineering from LUMS and the master's degree in computer sciences from UET, Lahore, Pakistan, and the Ph.D. degree in electrical engineering from Telecom ParisTech, France. During the Ph.D. studies, he was with EURECOM, Sophia Antipolis, involved on the research theme of flexible radios for multi-standard wireless communication devices. He has also with the Government of Pakistan in Research and Development projects from 1998 to 2004. He is currently a Professor and the Dean with the Faculty of Engineering and Sciences, Bahria University, Islamabad. He has authored or co-authored over 40 papers in reputed international journals and conferences. His current research interests include information security, flexible radios, and renewable energy.