This is a repository copy of *High Speed and Low Latency ECC Implementation over GF(2m) on FPGA*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/99476/

Version: Accepted Version

Article:
Khan, Z.U.A. and Benaissa, M. (2017) High Speed and Low Latency ECC Implementation over GF(2m) on FPGA. IEEE Transactions on Very Large Scale Integration Systems, 25 (1). pp. 165-176. ISSN 1557-9999

https://doi.org/10.1109/TVLSI.2016.2574620

© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.

Reuse
Unless indicated otherwise, fulltext items are protected by copyright with all rights reserved. The copyright exception in section 29 of the Copyright, Designs and Patents Act 1988 allows the making of a single copy solely for the purpose of non-commercial research or private study within the limits of fair dealing. The publisher or other rights-holder may allow further reproduction and re-use of this version - refer to the White Rose Research Online record for this item. Where records identify the publisher as the copyright holder, users can verify any specific terms of use on the publisher's website.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

High Speed and Low Latency ECC Processor Implementation over $GF(2^m)$ on FPGA

Zia U. A. Khan, Student Member, IEEE, and Mohammed Benaissa, Senior, Member, IEEE

Abstract—In this paper, a novel high speed ECC processor implementation for point multiplication on Field Programmable Gate Array (FPGA) is proposed. A new segmented pipelined full-precision multiplier is used to reduce the latency and the Lopez-Dahab (LD) Montgomery point multiplication algorithm is modified for careful scheduling to avoid data dependency resulting in a drastic reduction in the number of clock cycles required. The proposed ECC architecture has been implemented on Xilinx FPGAs Virtex4, Virtex5 and Virtex7 families. To our knowledge, our single multiplier and three multipliers based designs show the fastest performance to date when compared to reported works individually. Our one multiplier based ECC processor also achieves the highest reported speed together with the best reported area-time performance on Virtex4 (5.32 µs at 210 MHz), on Virtex5 (4.91 µs at 228 MHz), and on the more advanced Virtex7, (3.18 µs at 352 MHz). Finally, the proposed three multiplier based ECC implementation is the first work reporting the lowest number of clock cycles and the fastest ECC processor design on FPGA (450 clock cycles to get 2.83 µs on Virtex7).

Index Terms— High Speed ECC, Point Multiplication, Low Latency, Pipelined Bit Parallel Multiplier, Field Programmable Gate Array

I. Introduction

Elliptic curve cryptography (ECC) was proposed by Koblitz [1] and Miller [2] in 1985 individually. Public key cryptography based on ECC provides higher security per bit than its RSA counterpart [3]. ECC has some additional advantages such as a more compact structure, a lower bandwidth, and faster computation that all make ECC usable in both high speed and low resource applications. The National Institute of Standards and Technology (NIST), US has proposed a number of standard Elliptic curves over binary fields $GF(2^m)$ [5]. Binary field curves are suitable for hardware implementation as field arithmetic operations are carry free. FPGA based ECC hardware design is increasingly popular because of its flexibility, shorter development time scale, easy debugging and continual improvement of the technology (lower power and higher performance FPGAs).

Many high performance ECC processor implementations on FPGA have been reported in the literature; the most relevant are presented in [10], [11], [12], [13], [14], [15], [16], [17], [20], [21], [22], and [23]. The common optimizing technique of high speed designs is the reduction of latency (number of clock cycles) of a point multiplication (PM). To achieve low latency for a PM, these works adopted either parallel multipliers or large size multipliers at the expense of additional area complexity; pipelining stages are also often used to increase clock frequency at the expense of few extra clock cycles and area overheads [10, 12]. In addition, the pipelining stages in the multipliers create idle cycles at the PM level if there is data dependency in the instructions. As a result, careful scheduling is required to take full advantage of pipelining. Indeed, recently we have reported the highest throughput and highest speed ECC designs on FPGA in [24, 25] by using novel digit-serial and bit parallel multipliers together with efficient scheduling and pipelining techniques.

In this paper we extend our work in [24, 25] to yield two important contributions to the state of the art. First is the fastest and also crucially with the best area-time metric ECC design on FPGA to date to the best of our knowledge. And secondly, we report an even faster ECC processor design with the lowest ever latency (clock cycles) that achieves the performance of the theoretical limit. These are achieved via a novel pipelining technique that enables high clock frequencies to be attained and via a thorough investigation of the different combinations of the field multipliers to evaluate the performance limits for high speed applications. Below are listed the key contributions to the results:

- A full precision $GF(2^m)$ multiplier with segmented pipelining to reduce both latency and area.
- A one multiplier-based architecture for the ECC processor design targeted at high performance but with low area (fastest ECC processor with best area and time complexities).
- A three multipliers-based architecture for the ECC processor design aimed at the highest possible speed.
- A modified Montgomery point multiplication algorithm to avoid extra latency due to our two-stage pipelining in the field multiplier and use of careful point multiplication scheduling to reduce latency.
- A pipelined Moore finite state machine (*FSM*) based control unit is designed to avoid data dependency in the arithmetic operations by introducing an extra cycle delay.
- Data is tapped from different pipeline stages to localize some arithmetic operations and avoid memory input-output

Z. U. A. Khan and M. Benaissa are with the Electronic and Electrical Engineering, University of Sheffield, Mappin Street, Sheffield S1 3JD, UK (e-mail: elp10zuk@sheffield.ac.uk; m.benaissa@sheffield.ac.uk).

Algorithm 1: LD Montgomery Point Multiplication over GF(2^m) [6]

INPUT: $k = (k_{t-1}, \ldots, k_1, k_0)_2$ with $k_{t-1} = 1, P = (x, y) \in E(F_{2^m})$
OUTPUT: kp

Initial Step: $P(X_1, Z_1) \leftarrow (x, 1), 2P = Q(X_2, Z_2) \leftarrow (x^4 + b, x^2)$
For i from $t-2$ downto 0 do

If $k_i = 1$ then

Point addition: $P(X_1, Z_1) = P(X_1, Z_1) + Q(X_2, Z_2)$	Point Doubling: $Q(X_2, Z_2) = 2Q(X_2, Z_2)$
1. $Z_1 \leftarrow X_2 . Z_1$	1. $Z_2 \leftarrow Z_2^2$
2. $X_1 \leftarrow X_1 . Z_2$	2. $T \leftarrow Z_2^2$
3. $T \leftarrow X_1 + Z_1$	3. $T \leftarrow b.T$
4. $X_1 \leftarrow X_1 . Z_1$	4. $X_2 \leftarrow X_2^2$
5. $Z_1 \leftarrow T^2$	5. $Z_2 \leftarrow X_2 . Z_2$
6. $T \leftarrow x . Z_1$	6. $X_2 \leftarrow X_2^2$
7. $X_1 \leftarrow X_1 + T$	7. $X_2 \leftarrow X_2 + T$
8. Return $P(X_1, Z_1)$	Return $Q(X_2, Z_2)$

Conversion Step: $x_3 \leftarrow X_1/Z_1 \; ; y_3 \leftarrow ((x + X_1)/Z_1)[(X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1 Z_2)](\mathrm{x} Z_1 Z_2)^{-1} + y.$

operations.

- A repeated square over square circuit (capable to perform a 4-square or quad square operation in a single clock cycle) to reduce latency for the multiplicative inversion operation based on Itoh-Tsujii algorithm [9].
- Finally, we use Xilinx ISE timing closure techniques to achieve the best possible high performance results

The rest of this paper is organized as follows: Section II presents the background of ECC and associated arithmetic operations over GF(2^m); full-precision multiplication is also discussed in this section. Our proposed full-precision GF(2^m) multiplier is presented in Section III. Sections IV, and V cover the proposed ECC processor architectures. In Section VI,. the implementation results are presented, and compared to the state of the art. Finally, this paper is concluded in Section VII.

II. ECC Background and its Arithmetic over GF(2^M)

A. Scalar Point Multiplication

The main operation in ECC is scalar point multiplication, $Q = kP$,, where k is a private key (integer), Q is a public key and P is a base point on an elliptic curve, E. The public key, Q is computed by k times point addition operation:

$$Q = kP = P + \cdots + P + P \quad (1)$$

If the public, Q and P are known then the private k is difficult to retrieve. An elliptic curve over $GF(2^m)$, E can be defined as:

$$y^2 + xy = x^3 + ax^2 + b \quad (2)$$

Where $a, b \in GF(2^m)$, $b \neq 0$ and a point at infinity is θ such that $P_1 + \theta = P_1$ where $P_1 = (x_1, y_1)$ and $(x_1, y_1) \in GF(2^m)$.

The point multiplication kP is achieved by using scalar point multiplication algorithms utilizing point addition and point doubling depending on the ith value of k, k_i [4].

Scalar point multiplication can be affine coordinates based or projective coordinates based. Because of the expensive inversion operation involved in affine coordinates based algorithms, projective coordinates based point multiplication is a more common choice for ECC hardware implementation. In this paper, the Lopez–Dahab Montgomery (*LD*) point multiplication is considered. This algorithm, requires 6 field multiplications, 5 field squaring operations and 4 addition operations as shown in algorithm 1 [6]. The *LD* algorithm is generally faster to implement, and leads to improved parallelism and resistance to side channel power attack

B. Field Arithmetic over GF(2^m)

Field multiplication, field squaring, field addition and field inversion operations are involved in a point operation. Addition and subtraction are equivalent over *GF(2^m),* which are very simple bitwise *xor* operations.

Field inversion is very costly in term of hardware and delay. In projective coordinates, an inversion operation is used for the projective to affine coordinates conversion that can be achieved with multiplicative inversion. The Itoh-Tsujii [9] algorithm is selected as it requires only $\log_2(m)$ multiplications and *(m-1)* repeated squaring operations. In projective coordinates based implementations, overall performance depends on the performance of the field multipliers.

C. Full-precision Multiplier for ECC Application

For high speed ECC application, the field multiplier is the main part of the arithmetic unit compared to the field squaring and field addition circuits due to the high area and time complexities of the multiplier. The performance of the multiplier affects the overall performance. The performance of the multiplier depends mainly on the size of the multiplier. A larger size multiplier reduces latency to speed up the point operation; however, the critical path delay is increased. Thus, pipelining is often adopted to shorten the critical path delay. Moreover, some multiplication algorithms (such as Karatsuba) are used to improve area and time complexity [10, 11, 23]. For the high-speed end of the design space, large digit serial multipliers or bit parallel multipliers (such as school book and Mastrovito) are often used. The bit parallel multiplier takes one clock cycle latency, which can be an attractive option to speed up the point multiplication.

The field multiplication for ECC over *GF(2^m)*is divided into two parts: the *GF2* multiplication part (*GF2MUL*) and the reduction part. For a large size multiplier, the *GF2MUL* part is costly compared to the reduction part [18]. Thus, the main optimization of the large multiplier is concentrated on the *GF2MUL* part. There are several high performance bit parallel multipliers in the literature [11], [19], [20], [26], and [27]. The complexity of a bit parallel multiplier can be quadratic or subquadratic [18]. A quadratic multiplier achieves higher speed by consuming higher area than that of a subquadratic multiplier. Subquadratic multipliers are mostly based on the Karatsuba algorithm to reduce the area complexity at the expense of a lower clock frequency. The performance of the Karatsuba based bit parallel multiplier is improved by adopting pipelining techniques [11]. In the next section, we present a novel high performance full-precision *GF(2^m)* multiplier with segmented pipelining.

III. Proposed GF2^M Multiplier with Segmented Pipelining

The proposed full-precision *GF*(2^m) field multiplier (including reduction) with segmented pipelining is shown in Fig. 1 and consists of two stages pipelining to improve clock frequency.The first stage pipelining is the proposed segmented pipelining to break the critical path delay of the *GF2MUL* part, which is similar to the presented work in [7]. In the segmented pipelining, we divide the *m* bit multiplier operand into *n* number

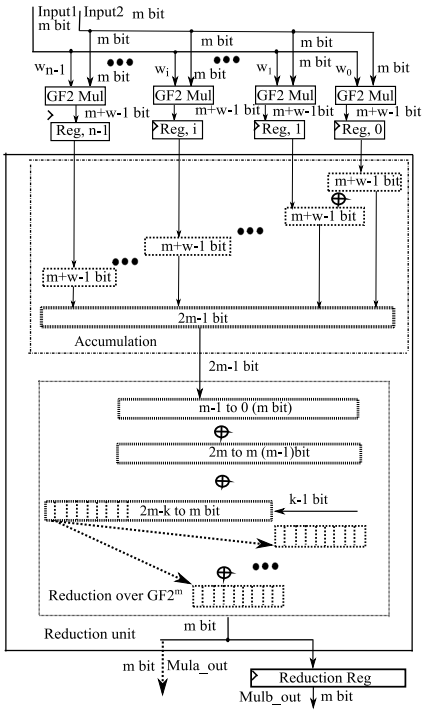

Fig. 1. Proposed Segmented Full-precision Multiplier over *GF(2ᵐ)*.

of *w* bit long digit multiplier operands. Then, we multiply the *m* bit multiplicand by each of the *w* bit multipliers. The results of the *w* digit size multiplier is *m+w-1* bit long. We save each of the results in the *m+w-1* size pipelining register. Here, we save *n* multiplications results in the *n* number of *m+w-1* size registers. The outputs of the *m+w-1* sizes registers are aligned by shifting (logically) *w* bits from each other followed by xor operations (addition). The result of the addition is *2m-1* bit long is then reduced to *m* bit in the reduction unit. In the reduction unit, we reduce *2m-1* bit to *m* bit multiplier output using a fast irreducible reduction polynomial [4][5]. The output of the reduction unit is applied to the second stage pipelining register. Thus, there are two pipelining stages and hence, the proposed multiplier consumes only 2 clock cycles as an initial delay to perform multiplication. The pipelining of the multiplier divides the total critical path delay into two parts: the critical path delay of *GF2MUL*, $T_A + (log_2(^m/_n))\ T_X$ and the critical path delay of the reduction part using the fast NIST reduction polynomial (r-nomial), $(\log_2((n+2r))\ T_X$ as shown in Table 1[4,7]. Both critical path delays depend on the size of the segment, *w*. Thus, any one of the two critical paths can be the critical path of the multiplier. The optimum critical path can be defined by the optimum size of *w* which can be determined by a trial and error method.

A one stage pipelining (segmented pipelining) achieves 1 clock cycle delay. The critical path delay of the multiplier is the combination of the *MULGF2* and the reduction part, which is $T_A + (\log_2(^m/_n + n + 2r))\ T_X$. Again, the critical path delay can be modulated by changing the size of the segment of the multiplier. The optimum size of the segment of the multiplier can be also achieved by using a trial and error method. In Table 1, we present space and time complexities of our proposed multipliers and we compare these with quadratic and subquadratic bit parallel multipliers reported in [19], [20], [26], and [27]. In the theoretical analysis of the quadratic and subquadratic multipliers, the quadratic bit parallel multiplier has twice as high speed as the speed of the subquadratic, but, the quadratic multiplier consumes 2.56 times more area [19]. Moreover, the authors in [19] compare the implementation results of the two bit parallel multipliers where they show the ratio (Quadratic/Subquadratic) is 1.5 in terms of area and 0.625 in terms of delay. Their implementation results show that the quadratic bit parallel multiplier can achieve higher speed, and the area-time product of the subquadratic multiplier outperforms the quadratic multiplier by only 6.65 %. Therefore, a quadratic multiplier is considered a better option for high speed ECC implementation when area is not a constraint; for example, the quadratic multiplier in [26] and its improved speed version in [27] both based on a matrix-vector method (Mastrovito) can achieve improved speed on a subquadratic multiplier [19] but with larger area.

An analytical complexity analysis for the multipliers is shown in Table 1. Our proposed multiplier consumes similar area to the multipliers in [19], [20], [26] and [27] ($m^2 >> ((n-1)m + (r-1)m)$. But, its regular structure makes it more suitable for pipelining, hence offers more scope for higher speed performance. Our proposed multiplier has a very short critical path compared to the reported parallel multipliers; hence, can show better area-time performance due to its high speed advantage. For the area complexity, our proposed multiplier consumes the same resources of *XOR* and *AND* gates as that of the quadratic bit parallel multiplier and uses flip flops (FFs) to reduce the critical path delay. For illustration, an

TABLE I
LATENCY, CRITICAL PATH DELAY (T_{mul}) AND RESOURCES OF PROPOSED FULL-PRECISION MULTIPLIER OVER $GF(2^m)$

Ref	Type	CCs	#XOR	#AND	#FFs	Critical path delay(T_{mul})
[19]	Quadratic	1	$m^2 - 1$	m^2	-	$T_A + (1 + \log_2(m))T_X$
[19]	Subquadratic	1	$5.5m^{\log_2(3)} - 3m + 0.5$	$m^{\log_2(3)}$	-	$T_A + (2\log_2(m) + 3)\ T_X$
[20]	Pipelined-Quadratic	1	$m^2 + 3m$	m^2	40m	$T_A + ((m/p) + \log_2(m))T_X$
[26]	Mastrovito	1	m(m-1)+3(m-1)	m^2	-	$T_A + (log_2(4m + 12u - 21)\ T_X$
[27]	Mastrovito with SPB	1	$m^2 + 3m - 7$	m^2	-	$T_A + (1 + log_2(2m + u - 1)\ T_X$
OUR MUL. Fig. 2	Full-precision (Segmented pipelined)	1	$m(m-1) + (n-1)m + (r-1)m$	m^2	n(m+w-1)+m	$T_A + (\log_2(^m/_n))\ T_X$ or $(\log_2((n+2r))\ T_X$
OUR MUL. Fig. 3	Full-precision (Segmented pipelined)	1	$m(m-1) + (n-1)m + (r-1)m$	m^2	n(m+w-1)	$T_A + (\log_2(^m/_n + n + 2r))\ T_X$

w = segment size, n, #Segments = $^m/_w$, $f(x) = x^m + x^l + x^h + x^g + 1$ *or* $f(x) = x^m + x^l + 1$ *or* $f(x) = x^m + x^{l(u+1)} + x^{lu} + x + 1$. $T_A$and T_X are AND and XOR gates delays respectively, p = # pipelining stages, CCs= Clock Cycles, SPB= shifted polynomial basis, r= 5 (for penta) or 3 (for tri) nomial.

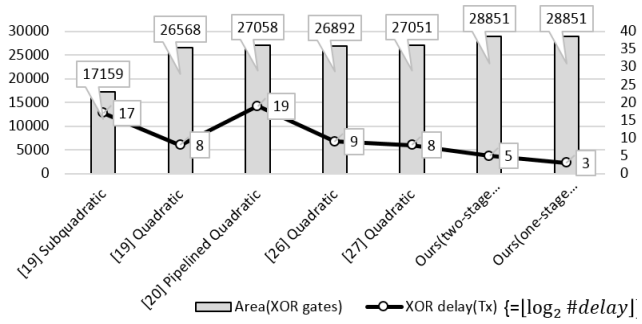

Fig. 2. Comparative Area and Delay Performance of Bit-Parallel GF(2^m) Multiplication (m=163, n=12, r=5 and u=3)

approximate[1] area-time complexity analysis is quantified over $GF(2^{163})$ for the various multipliers and sketched in Fig. 2. The results show that the proposed multiplier outperforms the reported multipliers in [19], [20], [26] and [27] in terms of area-time performance.

IV. Proposed High Performance ECC (HPECC) for Point Multiplication

In this section, we present careful scheduling in the point addition and point doubling operations, a novel pipelined full-precision multiplier and other supporting units to achieve high speed, low latency while optimizing area complexity.

A. Point Multiplication without Pipelining Delay

In general, the Montgomery point addition and point doubling in the projective coordinates requires a total of six field multiplication, five field squaring and four field addition operations equivalent latency if implemented serially according to Algorithm 1 [6]. If the field squaring and field addition operations can be operated concurrently with multiplication then the point operations latency will be equivalent to the latency of the six field multiplications. The six multiplications can, for example, be computed in two steps using three multipliers or in three steps using two multipliers or in six steps by serial multiplications using one multiplier [17], [13] and [10]. Again, the digit size can affect the performance of ECC; for example, a bit serial implementation takes m cycles, a digit

Algorithm 2: Proposed Combined LD Montgomery Point Multiplication (for 6 clock cycles)

For i from $t-2$ down to 0 do If $k_i = 1$ then	
If $k_{i+1} = 1$ then	If $k_{i+1} = 0$ then
Point addition:$P(X_1,Z_1) = P(X_1,Z_1) + Q(X_2,Z_2)$ and Point Doubling: $Q(X_2,Z_2) = 2Q(X_2,Z_2)$	
St1: $Z_1 \leftarrow X_2.Z_1$; $A \leftarrow Z_2$ St2: $X_1 \leftarrow X_1.Z_2$, $Z_2 \leftarrow A^2$; $R_2 \leftarrow A^4$; $A \leftarrow X_2$	St1: $Z_2 \leftarrow X_1.Z_2$; $A \leftarrow Z_2$ St2: $X_2 \leftarrow X_2.Z_1$; $Z_2 \leftarrow A^2$; $R_2 \leftarrow A^4$; $A \leftarrow X_2$
St3: $X_2 \leftarrow b.R_2 + A^4$; $R_1 \leftarrow A^2$ St4: $Z_2 \leftarrow R_1.Z_2$, $A \leftarrow X_1 + Z_1$ St5: $X_1 \leftarrow X_1.Z_1$, $Z_1 \leftarrow A^2$ St6: $X_1 \leftarrow x.Z_1 + X_1$.	
Conversion Step: Same as Algorithm 1.	

[1] Based on Xors only as this is also done by references [26,27]; ANDs complexity is the same for all.

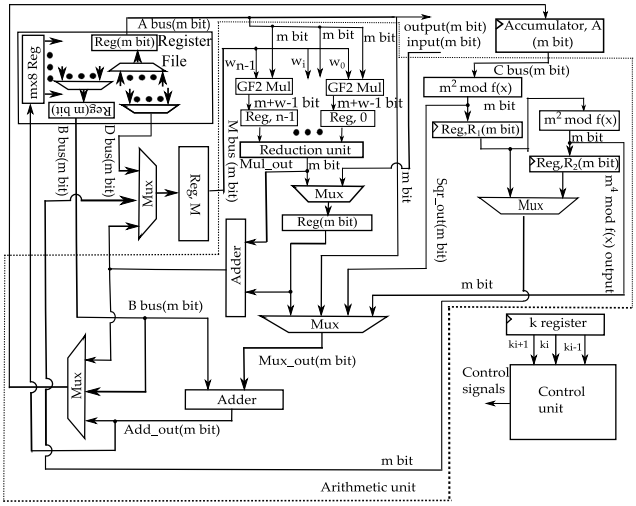

Fig. 3. Proposed High Performance ECC Processor Architecture

(w bits) serial one takes (m / w) cycles and a bit parallel implementation takes a single clock cycle [8], [12] and [11]. In the case of high speed design, digit serial multipliers are considered to reduce latency. The disadvantage of large digit serial multipliers is lower clock frequency. Thus, pipelining stages are applied to improve clock frequency [12]. The clock frequency can be improved with the increase of the number of pipelining stages in breaking the critical path delay. The main disadvantages of increasing the number of pipelining stages in the high-speed end of the design space are the increase in the number of clock cycles per multiplication and overcoming data dependency [12]. To avoid pipelining delay, optimal scheduling of the field operations of the point multiplication is necessary

Our first proposed ECC processor architecture is shown in Fig. 3. It comprises a full-precision m bit multiplier with two stages pipelining, one squaring circuit, one quad squaring circuit and two addition circuits in order to accomplish point operations (point addition and point doubling) within six clock cycles. To achieve six clock cycles based point operations, we include some strategies in the point operations of the Montgomery point multiplication algorithm as shown in Algorithm 2 [24]. In the proposed algorithm, we combine point addition and point doubling to avoid data dependency. In the point multiplication, a particular loop is overlapped with its next loop by 2 clock cycles due to two stages pipelining. Thus, state1 (st1) and state2 (st2) depend on the previous key bit, k_{i+1}. For example, if previous bit, $k_{i+1} = 1$ then the last output will be X_1 otherwise X_2. The last output of a loop decides the sequence of st1 and st2 in the next loop. The rest of the states depend on the current bit of k, k_i. To support a six clock cycle based algorithm, we apply a squarer or double square (Quad Square) or both operations in parallel along with the multiplication. Again, one of the field adders is placed in the common data path to add on the fly. The second adder is used to add the two outputs of the multiplier as shown in Fig. 3. Both adder circuits can add two of their inputs or can transfer either of the inputs, if we need either. Moreover, we can save some

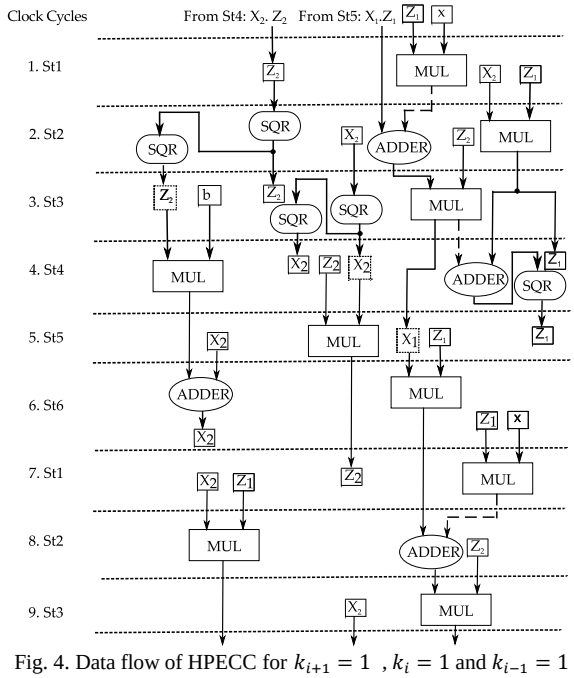

Fig. 4. Data flow of HPECC for $k_{i+1} = 1$, $k_i = 1$ and $k_{i-1} = 1$

intermediate results of field operations in the local registers (R_1, R_2, M and accumulator, A.) to avoid loading/unloading to the main memory. As a result, we can avoid idle clock cycles due to the memory input-output operations. A data flow diagram is shown in Fig. 4 to demonstrate the proposed combined point operations. In this diagram, we explain point operations for $k_{i+1} = 1$, $k_i = 1$ and $k_{i-1} = 1$ where k_i is the current bit, k_{i+1} is the previous bit and k_{i-1} is the next bit of key (k). In this data flow diagram, we show the loop operation of the point multiplication in projective coordinates. In our implementation, a multiplication takes three clock cycles due to two stages pipelining and a square operation takes two clock cycles where one clock cycle is used to load in the accumulator (A) register. The addition operation is realized in the common data path and accomplished in the same clock cycles. As we used two stage pipelining and there is a data dependency in between two loops, we use careful scheduling. In this scheduling, the present loop operation of point multiplication is overlapped with the next loop operations.

- We see, the starting state, st1 of a particular loop depends on the value of previous bit, k_{i+1}. If the previous bit, $k_{i+1} = 1$ means X_1 is not ready. Then, we start from st1 with the multiplication between X_2 and Z_1 instead of X_1 and Z_2 . In this case, The st2 is the multiplication between X_1 and Z_2.
- The X_1 operand of the st2 is calculated by addition of two outputs(Mula_out and Mulb_out in Fig. 1) of the multipliers where one output(from Mula_out) is tapped after the reduction unit (dotted arrow) and the other one from the multiplier output(Mulb_out). The other operand of st2 is Z_2 which is already saved in the memory in st1 to use in st2. Here, the delay of the memory operation (accessing Z_2) is utilized to calculate X_1; again, as $k_i = 1$, we need the square and quad square of Z_2. Thus, we save Z_2 in the memory and accumulator simultaneously in st1 to achieve the squaring operations of Z_2 in the st2. The output of the square circuit ($A^2 = Z_2^2$) is saved in the memory and the output of quad square ($A^4 = Z_2^4$) is saved in the local register, R_2 (dotted box). We can use data from the local register (dotted box) immediately without doing any memory operations to save clock cycles.
- Similarly, during st2, st3 and st4, the squaring operations of X_2 is realized by saving in the accumulator through B-bus; in this case, the square output, $A^2 = X_2^2$ is saved in the local register, R_1 whereas the quad square output, $A^4 = X_2^4$ is saved in the memory. In st3 and st4, one of the multiplication operands is used from the memory and the other operand from the local registers.
- In st4, Z_1 (result of $X_2.Z_1$) is ready to save in the memory to use in st5. Again in st4, the available output, Z_1 is required to add with the multiplication result of X_1 on the fly. At this time, we access (tapping) X_1 from the output of the reduction unit (dotted arrow, one cycle earlier than the normal output) to add with Z_1 followed by saving in the accumulator to do the square operation to get a new Z_1.
- The new Z_1 is ready in st5 to save in the memory and is required in the st6 and the next loop. In st5, the old Z_1 (saved in st4) is used for multiplication with X_1 where X_1 is directly collected from the multiplier output followed by saving in the local register, M. We can manage X_1 to use immediately for multiplication by using the instruction delay (pipelined Moore machine based control unit) of accessing the old Z_1 from memory.
- In st6, we add X_2 (from memory) on the fly with the multiplier output to get new X_2 followed by saving in the memory. Again, the multiplication in st6 is in between the base point, x and new Z_1 is completed after two clock cycles. But, a new loop is started after st6.

Thus, the st1 of the new loop depends on the last coordinate of the previous loop, X_1 (in this case of $k_{i+1} = 1$, $k_i = 1$ and $k_{i-1} = 1$) which is calculated by adding the results of the multiplications started in st5 and st6.

In Fig. 5, we demonstrate the loop of point multiplication for $k_{i+1} = 0$, $k_i = 1$ and $k_{i-1} = 1$. The previous bit of k, is $k_{i+1} = 0$ means coordinate X_2 of the last loop is not ready to start with.

- In this case, the first state (st1) is started with multiplication between X_1 and Z_2. In this state, the multiplier output (Z_1) started from st4 of the previous loop is saved in the memory to using in the next state (st2). In the same state, we need to start the squaring operation on Z_2. Thus Z_2 is accessed from memory through the A_bus for multiplication and through the B_bus into the accumulator for squaring.
- In st2, the multiplication is $X_2.Z_1$; where X_2 is

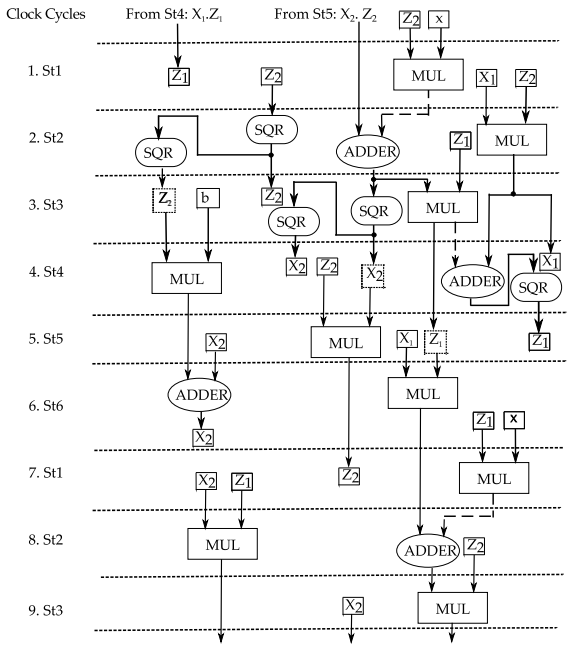

Fig. 5. Data flow of HPECC for $k_{i+1} = 0$, $k_i = 1$ and $k_{i-1} = 1$

calculated by adding two outputs of the multiplier and then is saved in the M register for use in the next cycles to multiply with Z_1. In the same time, the calculated X_2 is required and saved in the accumulator for squaring as $k_i = 1$.

The rest of the states of Fig. 5 are similar to Fig. 4.

B. Multiplier with Segmented Pipelining for HPECC

We consider the two extreme field sizes in the NIST standard [5] i.e $GF2^{163}$ and $GF2^{571}$ to evaluate the ECC performance. In the implementation over $GF2^{163}$, we select $w = 14$ bit to get 12 of the 14 digit serial multiplication results. The results then are loaded in the twelve 177 bit long registers. Thus the critical path of *MULGF2* depends on one two input *AND* gate and 13 layers of two input XOR gates to achieve a 14x163 multiplication. Again, the 12 pipelining register outputs are shifted and *xored* (for accumulation) to get the full-precision multiplication result *(2m-1)* without reduction. The result is then reduced into 163 bit in the reduction unit using the fast irreducible reduction polynomial [5]. The reduced result is saved in the second stage pipelining register. Thus, the architecture works like 12 (14-bit) digit serial multipliers are operating in parallel followed by a full precision reduction operation. The reduction unit consists two parts: the accumulation part and the reduction part. The accumulation part has 11 layers of 2 inputs *XORs* and the reduction part has *2r* (*r*-nomial irreducible polynomial) layers of 2 input *XORs*. Thus, the critical path delay is balanced theoretically. Again, in the ECC processor implementation over $GF2^{571}$, we also consider the segment size of 14 bit.

C. Square Circuit, Memory Unit and Control Unit of HPECC

Our proposed high speed ECC processor design operates by using six clock cycles for each loop of the point multiplication. To achieve the six cycles point multiplication loop, we need a quad square (4-square) circuit to do a one clock quad square operation. The quad squaring is used in the st2 and st3 along with field multiplication as shown in our proposed Algorithm 2. Again, the latency of the conversion step contributes a significant amount to the total latency of the proposed ECC processr as the latency of the loop operation is comparable with that of the conversion step. In the conversion step, the inversion operation consumes the major part of the latency in our projective based ECC processor implementation, a multiplicative inversion is applied for the projective to affine coordinate conversion. Several multiplications and m steps repeated squaring operations are required. Thus, we can utilize the quad square circuit for speeding up the inversion by reducing the number of the repeated square operations. In our proposed architecture, we use a register (accumulator) in the arithmetic data path to achieve a repeated quad square operation without loading to the main memory. Thus, we need 1 clock cycle for a 4-square, 2 clock cycles for an 8-square and so on.

We design a friendly memory unit that is developed in a single behavioral entity which comprises an accumulator and 8xm register file. The register file is based on distributed RAM to give high performance and flexibility. There are five input-output buses in the memory unit. Particularly, our register file consists of three output buses (A, B, D) and one input bus. Data through A-bus and B-bus takes one more cycle delay than data through D-bus as shown in Fig. 3. Data from D-bus is dedicated to the multiplier input through the M register. Hence, the two outputs of the memory through A-bus and B-bus, and the output of M (through D-bus) are synchronized. The M register acts as a pipelining register between the input and the output of the multiplier and also saves local data for the multiplier. The memory unit offers flexibility to access any data from any location of the memory through each of the output buses independently. The memory unit takes one cycle for a write operation and one cycle for a read operation. The accumulator is designed in the same entity of the memory unit and utilizes unused resources (flip-flops) of the memory unit. Apart from our memory unit, we deploy local registers R_1 and R_2 ; R_1 and R_2 are used to save outputs of square and quad square respectively. Thus, the local registers (R_1 and R_2) and M save outputs of concurrent operations to avoid the idle state that is due to the common input bus of the memory unit, and also avoid the data dependency in the successive point operations loop.

A pipelined Moore finite state machine based control unit is developed in the single behavioral entity. The Moore machine takes one clock cycle delay to address the memory unit. The advantage of this initial instruction delay is a more flexible data control that allows for some intermediate operations to be carried out during this cycle delay with the help of the local registers. Again, the control unit consists of very few states to complete a point multiplication due to the full-precision multiplier and concurrent operations. As a result, the control unit consumes very low area while helps keeping speed very high.

TABLE II
CRITICAL PATH DELAY (T_{ECC}) OF THE PROPOSED ECC

Ref	Critical path delay
HPECC	T_{mul} or $(\log_2(n+2r))\ T_X + T_{adder} + 2T_{mux}$
LLECC	$T_{mul} + T_{adder} + T_{sqr} + 3\ T_{mux}$

$n =$ #Segments, r is the r-nomial irreducible polynomial, T_{mux}= 2x1 mux delay. T_{sqr}=$\log_2(k)$, $T_{adder} = T_x$.

D. Critical Path Delay and Clock Cycles of the HPECC

Our proposed high speed ECC (*HPECC*) processor design uses a segmented pipelining based full-precision multiplier to achieve six clock cycles for each loop of the point multiplication. The critical path delay of the ECC mainly depend on the critical path of the multipliers. Again, the proposed multipliers critical path delay can be the critical path delay of the *GF2MUL* part or the reduction part depending on the size of the segment. As the multiplier output (*Mula_out*) is taped at end of the reduction part, and passed through the adder and multiplexer followed by saving in the M register, the critical path delay of the ECC can be the delay of the reduction part+adder + mux. The critical path delay of the ECC processor architecture is shown in Table II. The main focus of our proposed ECC processor is the reduction in the number of clock cycles. Particularly, our design can manage to take 6 clock cycles for each loop of the point multiplication in the projective coordinates. The total clock cycles for point multiplications is the sum of three main parts: affine coordinates to projective coordinates initialization, point multiplication in the projective coordinates and finally projective coordinates to affine coordinates conversion. The total number of clock cycles (*CCs*) for point multiplication = 5 *CCs* (required for initialization) + *6x(m-1) CCs* (for point multiplication in the projective coordinates) + *CCs* (for the final coordinates conversion = m/2 *CCs* for square + #Mul for inversion x3 +3 *CCs* for Inversion + *28 CCs* for others) + 3 clock cycles for pipelining as shown in Table III. The others clocks cycles are that are independent of curve sizes included: 10 multiplications, 6 addition and 1 square operations. For example, the total clock cycles for point multiplication over $GF2^{163}$= *5+(6x162)+139(= (81+27+3) +28) +3=1119* cycles. Similarly, the latency of *HPECC* processor over $GF2^{571}$ is 3783 clock cycles.

V. PROPOSED LOW LATENCY ECC (LLECC) PROCESSOR FOR POINT MULTIPLICATION

The speed of ECC can be improved for high speed applications by reducing latency of the point multiplication. Parallel full-precision multipliers can reduce latency to speed up the point operations. We proposed a high speed ECC processor for point multiplication utilizing three full-precision multipliers to achieve the lowest latency high speed ECC as shown in Fig. 6.

A. Low Latency Montgomery Point Multiplication

Montgomery Point multiplication offers flexibility of parallel field operations; there are six field multiplications in the projective coordinates based Montgomery point multiplication, as shown in Algorithm 1, all which can be carried out in parallel based on data dependency. In addition, the Montgomery algorithm exhibits the low data dependency as it employs only *x* coordinates [4].

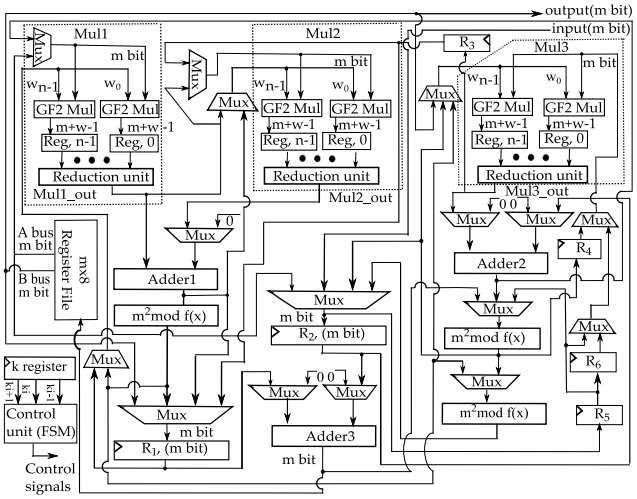

Fig. 6. Proposed Low Latency ECC Processor Architecture

The six multiplications can be achieved in two steps by using three full-precision multipliers as shown in Algorithm 3. To achieve the theoretical limit of the loop operation, an ECC processor architecture needs single clocked field multipliers along with concurrent square and addition operations, all with careful scheduling. In our implementation here we target and achieve this limit which to our knowledge, no previously reported implementation has achieved to date due to the hitherto restrictive performance of the field multiplier. We propose a modified Montgomery point multiplication loop based on two steps using three full precision multipliers (*Mul1*, *Mul2* (highlighted) and *Mul3*) as shown in Algorithm 3. In each state of the proposed algorithm, three multiplications outputs are concurrently used for additions, square and square over square (*4-square*) to generate the required output for the next states as shown in Fig 6. *Mul1*, *Mul2* and *Mul3* are the three multipliers that multiply the three different multiplications involved in each step of Algorithm 3 in a single clock cycle. Again, the adder and cascaded square circuits are in the same data path of the multiplier output to perform addition, square and 4-square operations using the multipliers outputs.

For the initialisation of Algorithm 3, we save the required variables to start the loop operation in local registers (R_1, R_2, R_3, R_4, R_5, and R_6). For a particular value of k, $k_i = 1$, the multipliers M*ul1* , *Mul2* and *Mul3* as shown in Fig. 6 calculate $X_2 \leftarrow X_2.\ R_1\ \{R_1 = Z_1\},\ Z_2 \leftarrow X_1.R_3\ \{R_3 = Z_2\},\quad Z_1 \leftarrow X_1^2.R_5\ \{R_5 = {Z_2}^2\}$. In the same step, a cascaded squaring of X_2 is performed to obtain the 4-square operation ($R_2 \leftarrow X_2^4$) followed by save in the R_2 register. In step 2, one input of *Mul1*, $(X_1 + Z_1)^2$ (and the other input, x from memory unit) is processed by adding the outputs of *Mul1* and *Mul2* using *adder1* followed by squaring. The output of the squaring is also saved in the R_1 register as Z_1 for the next loop. The *Mul1* output and *Mul2* are added by *adder1* to get X_1, an input of step1 of the *Mul2* in the next loop. In the step2, the inputs of *Mul2* are the outputs *Mul1* (Z_1) and *Mul2* (X_1). The *Mul3* output (Z_2) of step1 is saved in the register R_3 in the step2 to use as an input of mul2 in the next loop and the *Mul3* output, Z_2 is squared (${Z_2}^2$) and 4-squared (${Z_2}^4$) using the cascaded square circuits then saved in the registers R_4 and R_5 . Again, the inputs of *Mul3* of step2 are *b* from the memory unit and ${Z_2}^4$ from register, R_5 and the multiplication output is added with the content of R_2

Algorithm 3: Proposed Low Latency Montgomery Point Multiplication (2 clock cycles based loop operation is shown)

For i from $t-2$ down to 0 do		
If $k_i = 1, k_{i+1} = 1$ and $k_{i-1} = 1$ then{ No transition}		
Point addition:$P(X_1, Z_1) = P(X_1, Z_1) + Q(X_2, Z_2)$ and Point Doubling: $Q(X_2, Z_2) = 2Q(X_2, Z_2)$		
Mul1	Mul2	Mul3
St1: $Z_1 \leftarrow X_2.\ R_1$; $\{R_1 = Z_1\}$	$X_1 \leftarrow X_1.R_3$;	$Z_2 \leftarrow X_2^2.R_4$; $R_2 \leftarrow X_2^4$;
St2: $X_1 \leftarrow (x.(X_1 + Z_1)^2$ $R_1 \leftarrow (X_1 + Z_1)^2$	+ $X_1 \leftarrow X_1.Z_1)$; $R_3 \leftarrow Z_2$	$X_2 \leftarrow b.R_5 + R_2$; $R_5 \leftarrow {Z_2}^4$; $R_4 \leftarrow {Z_2}^2$;
else If $k_i = 1, k_{i+1} = 1$ and $k_{i-1} = 0$ then{Transition :$k_i = 1$ to $k_i = 0$}		
St1: $Z_1 \leftarrow X_2.\ R_1$; $\{R_1 = Z_1\}$	$X_1 \leftarrow X_1.R_3$;	$Z_2 \leftarrow X_2^2.R_4$; $R_2 \leftarrow X_2^4$;
St2: $X_1 \leftarrow (x.(X_1 + Z_1)^2$ $R_1 \leftarrow (X_1 + Z_1)^2$	+ $X_1 \leftarrow X_1.Z_1)$; $R_3 \leftarrow Z_2$	$X_2 \leftarrow b.R_5 + R_2$; $R_5 \leftarrow ((X_1 + Z_1)^2)^2$;{ $R_5 = {Z_1}^2$}
If $k_i = 0$, $k_{i+1} = 1$ and $k_{i-1} = 0$ then {Transition : $k_i = 1$ to $k_i = 0$}		
St1: $X_2 \leftarrow X_2.\ R_1$; $\{R_1 = Z_1\}$	$Z_2 \leftarrow X_1.R_3$;	$Z_1 \leftarrow X_1^2.R_5$; $R_2 \leftarrow X_1^4$; $R_4 \leftarrow {R_5}^2$; $\{R_4 = {Z_1}^4\}$
St2: $X_2 \leftarrow (x.(X_2 + Z_2)^2$ $R_1 \leftarrow (X_2 + Z_2)^2$	+ $X_2 \leftarrow X_2.Z_2)$; $R_3 \leftarrow Z_1$	$X_1 \leftarrow b.R_4 + R_2$; $R_5 \leftarrow {Z_1}^4$; $R_4 \leftarrow {Z_1}^2$;
If $k_i = 0$, $k_{i+1} = 0$ and $k_{i-1} = 0$ then{ No transition}		
St1: $Z_2 \leftarrow X_1.\ R_1$;$\{R_1 = Z_2\}$	$X_2 \leftarrow X_2.R_3$;	$Z_1 \leftarrow X_1^2.R_4$; $R_2 \leftarrow X_1^4$;
St2: $X_2 \leftarrow (x.(X_2 + Z_2)^2$ $R_1 \leftarrow (X_2 + Z_2)^2$	+ $X_2 \leftarrow X_2.Z_2)$; $R_3 \leftarrow Z_1$	$X_1 \leftarrow b.R_5 + R_2$; $R_5 \leftarrow {Z_1}^4$; $R_4 \leftarrow {Z_1}^2$;
Conversion Step: As shown in the Algorithm 1.		

(X_2^4) using *adder2* then inputted as X_2, an input of mul1 in the next loop. Thus, the proposed architecture supports the calculation of all of the new inputs for the next loop such as $X_1, X_2, Z_1,$ and, Z_2 using the two steps of algorithm 3. Apart from this, we utilize a smart scheduling to avoid data dependency in the successive loops. We a show data flow diagrams to illustrate the point operations for the different combinations of the previous, current and next values of k_i in Fig.7 and Fig 8.

The data flow diagram shown in Fig. 7 is for the values of $k_{i+1} = 1, k_i = 1$ and $k_{i-1} = 1$. In this case, the point operations of the previous loop, current loop and next loop are the same, hence, there is no transition of the point operations in the successive loops. There are only two states (*st1* and *st2*) for each loop to accomplish the field operations (i.e. multiplication, square and addition) for a point-multiplication loop operation. The field multiplication takes 1 clock cycle delay due to one stage pipelining; however, the field square and field adder have only combinational circuit delay and can be performed in the same clock cycle. In Fig. 7, the data diagram shows the utilization of three full-precision multipliers called *Mul1*, *Mul2* and *Mul3* in each state to accomplish three multiplications. As the multiplier, adder and square circuits are cascaded, we can achieve different field operations in the same clock cycle by tapping the results respectively.

- For example, in st1, *Mul1* and *Mul2* outputs $(i.e. Z_1 and\ X_1)$ are added and squared to get new Z_1 on the fly. The Z_1 is immediately used in the next loop as an input to *Mul1* and also Z_1 is saved in Register 1 (R_1) to use in the next loop. Again, the output of *Mul3* is Z_2 is squared and 4-squared in the same clock to get Z_2^2 and Z_2^4. After then, the three outputs (Z_2, Z_2^2 and Z_2^4) are saved in R_3, R_4 and R_5 register respectively to use in the next loop.
- In state st2, we get output X_1 by adding the outputs of *Mul1* and *Mul2* and we also get X_2 by adding the output of *Mul3* and the content of R_2 (X_2^4). The X_2 and its square, X_2^2 are directly applied as an input of *Mul1* and *Mul3* respectively in the st1 of the next loop and also X_2 is squared over squared (*4-square*) to get X_2^4 output in the same clock cycle is saved in the R_2 for the next operation.

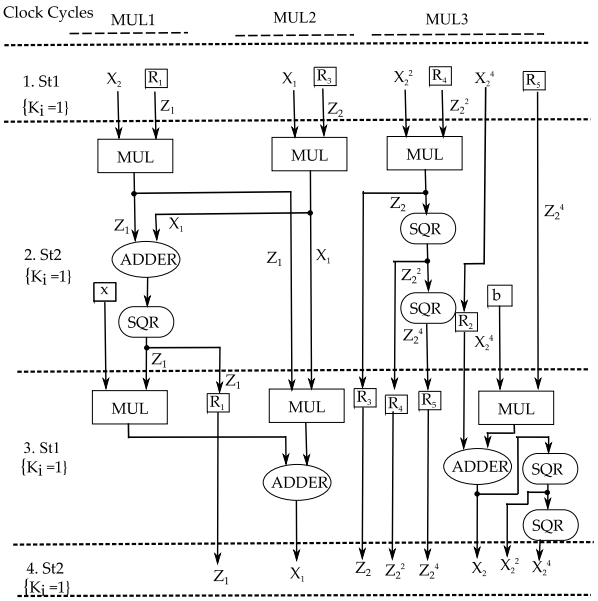

Fig. 7. Data flow of LLECC for $k_{i+1} = 1$, $k_i = 1$ and $k_{i-1} = 1$

Thus, all inputs that are required to begin the next loop are ready. The dataflow diagram is the same for the combination of values $k_{i+1} = 0, k_i = 0$ and $k_{i-1} = 0$ except that the variables are changed as shown in Algorithm 3.

In Fig7, a data flow diagram of the loop of point multiplication is presented for the values of $k_{i+1} = 1, k_i = 0$ and $k_{i-1} = 0$. The diagram shows three consequent loops (for six clock cycles) of data flow to illustrate the transition from the loop of $k_i = 1$ to the loop of the $k_i = 0$.

- In clock cycles 1 and 2, the point operations for the value of $k_i = 1$ is performed. As the next loop for $k_i = 0$, the squared outputs of the loop ($k_i = 1$) should be Z_1^2, Z_1^4, X_1^2, and X_1^4 instead of Z_2^2,Z_2^4, X_2^2, and X_2^4. In the loop, Z_1^2 is calculated and saved in R_5 in the *st2*. Again, the output X_1 of the loop will be squared and 4-squared to get X_1^2 and X_1^4 in the *st1* of the next loop ($k_i = 0$).
- In *st1* of the loop of $k_i = 0$ (at clock cycle 3), the X_1^2 is used as *Mul3* input, the X_1^4 is saved in R_2. In the same state, the content of R_5 (Z_1^2) is squared to get Z_1^4 and saved in R_4.Thus, the second loop for $k_i = 0$ can be started with three multipliers inputs X_2 . Z_1, X_1. Z_2 and Z_1^2. X_1^2 after the previous loop ($k_i = 1$). In this case, the loop ($k_i = 0$) inputs of *Mul1* and *Mul2* are the same as the inputs of the previous loop ($k_i = 1$) due to the last output (the addition of R_2 and *Mul3)* of the previous loop is X_2; however, the outputs of the multipliers are different than that of the previous loop.
- Now, the final loop is for $k_i = 0$ (clock cycles of 5 and 6) is similar to Fig. 6, (no transition) except that the variables are changed as shown in Algorithm 3.

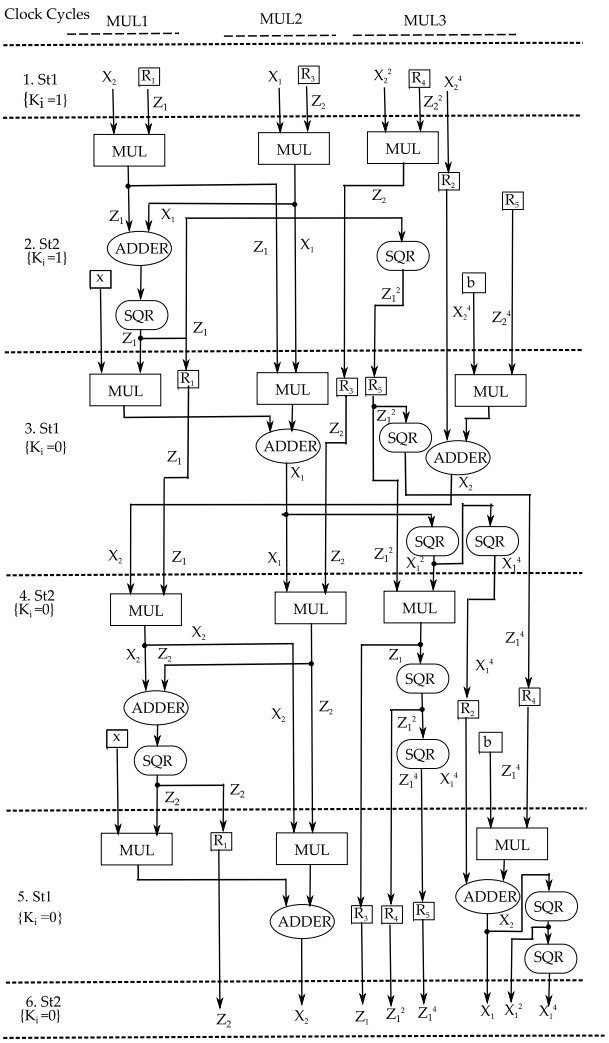

Fig.8. Data flow of LLECC for $k_{i+1} = 1$, $k_i = 0$ and $k_{i-1} = 0$

Thus, the loop of the point operations can be accomplished utilizing only two clock cycles for any set of values of k_{i+1}, k_i and k_{i-1}.

B. Multiplier with Segmented Pipelining for LLECC

Parallel multipliers are used to reduce latency for point multiplication in ECC processor implementations and the majority of reported designs in the literature are based on digit serial multipliers instead of bit parallel multipliers [13 -17]. Bit parallel multipliers take larger area and critical path delay as the size of the multiplier is large due to the large field sizes of the ECC curves [18]. The subquadratic bit parallel multiplier can be suitable for a high speed ECC design, however, pipelining is required to improve speed [11]. The adoption of the pipelining in the proposed 3 multiplier-based ECC processor is limited as the loop operation takes place within two clock cycles only. Thus, only one stage pipelining can be adopted to improve the performance of the multiplier providing a smart scheduling is devised to overcome the data dependency. The limitation of pipelining is a serious bottleneck for the traditional bit parallel and subquadratic multipliers to achieve significant performance. This is overcome in our proposed segmented pipelining technique by implementing *n* pipelines in parallel, achieving an overall single stage only pipelining as shown in Fig. 6. This makes the proposed full-precision multiplier suitable for the very low latency loop while still maintaining a high performance. The high performance can allow high security ECC curves to be deploy in more applications. In our proposed low latency ECC (*LLECC*) processor architecture (as shown in Fig. 6), we consider *LLECC* implementation over *GF(2[163])* where we use three parallel multipliers where each of them is a 163 bit full-precision multiplier with 14 bit segmented pipelining.

C. Square Circuit, Memory Unit and Control Unit of LLECC

Our proposed least latency ECC (*LLECC*) processor takes 2 clock cycles for a loop operation of the Montgomery point multiplications. To accomplish 2 clock cycles based loop operation, we need to process the multiplier output in same clock cycle by cascading the adder and square circuits. Thus, in Fig. 6, there are several extra adder, square circuits and local registers are considered to calculate some instructions of the point operation on the fly as compare to Fig .3. The main memory architecture adopted is the same as that of the distributed based memory of Fig. 3 used to enhance speed. Our main memory saves the initial input and the final outputs, and during a loop operation, the memory supplies the constant values (*x, y, b*) as most of the calculated outputs are saved in the local registers to reduce the delay for memory access.

We also use a separate shift register (*k* register) to save the key of the ECC. The shift register shifts 1 bit in every two cycles to generate a new set of values for k_{i+1}, k_i and k_{i-1} used in the control unit as shown in Fig. 6. The control unit of the *LLECC* is also based on a finite state machine (*FSM*) that controls the two clock cycles based point operations and is simpler than the control unit of the *HPECC* as most of the operation are performed concurrently.

D. Critical Path Delay and Clock Cycles of the LLECC

In the proposed low latency ECC (*LLECC*) architecture, we perform several instructions in the same cycle by cascading the multiplier, adder and square circuits as shown in Fig. 6. The critical path delay of the *LLECC* is the path delay of *MULGF2* + the reduction part + adder + square + *3x1 mux* as shown in Table II. The critical path delay can be optimised by selecting the size of *w* through a trial and error approach.

The total clock cycles of ECC mainly depends on the latency of the loop operation of the point multiplication. We achieve *2* clock cycles for each loop operation for the Montgomery point multiplication in projective coordinates which is the theoretical limit of the Montgomery point multiplication algorithm under projective coordinates. Again, the coordinates conversion circuit includes the costly inversion operation. We adopt multiplicative inversion to reduce area and time complexities

TABLE III :

LATENCY OF THE PROPOSED ECC (MUL = $M_1 = 1, or\ M_2 = 2, or\ M_3 = 3$, ADD=1, SQR=1 AND 4SQR =1)

ECC	Initial + point operations + Conversion	GF(2^{163})	GF(2^{571})
HPECC	$5 + (6M_1)(m-1)$ $+(7M_2 +$ Inv1$+3M_3 + 8)$	1099	3783
LLECC	$5 + (4 + 2M_1(m-1) + 4)$ $+(7M_2 +$Inv2$+3M_2 + 3)$	450	-

TABLE IV
COMPARISON OF PROPOSED ECC WITH PUBLISHED STATE OF THE ART OVER GF(2^M) AFTER PLACE AND ROUTE ON FPGA

Ref.	Slices (Sls)	FFs	LUTs	Freq. (MHz)	kP Time (µs)	Sls x Time x 10^{-3}	Latency, Clock Cycles	FPGA	Resources: Multipliers(Mul)
ECC over GF2^{163}									
[10]	4080	1502	7719	197	20.56	84	4050	Virtex-4	41 bit Karatsuba Mul
[11]	8095	-	14507	131	10.70	87	1429	Virtex-4	163 bit Karatsuba Mul
[12]	16209	7962	26364	154	19.55	317	3010	Virtex-4	55 bit Mastrovito mul
[14]	20807	-	-	185	7.72	161	1428	Virtex-4	3 Core 82 bit Mul
[15]	24363	-	-	143	10.00	244	1446	Virtex-4	3 GNB 55 bit Mul
[16]	17929	-	33414	250	9.60	172	2751	Virtex-4	3 Digit Serial 55 bit Mul
[17]	12834	6683	22815	196	17.20	221	3372	Virtex-4	2 GNB 55 bit Mul
[21]	8070	-	14265	147	9.70	78	1429	Virtex-4	163 bit Karatsuba Mul
[22]	10417	-	-	121	9.00	94	1091	Virtex-4	163 bit Karatsuba Mul
[23]	-	-	27889	133	16.00	-	2128	Virtex-4	163 bit Karatsuba Mul
[24]	3536	1870	6672	290	14.39	51	4168	Virtex-4	41 bit Digit Serial Mul
HPECC_1M	**12964**	**3077**	**23468**	**210**	**5.32**	**69**	**1119**	**Virtex-4**	**163 bit Mul**
[13]	6150	-	22936	250	5.48	34	1371	Virtex-5	3 Digit Serial 81 bit Mul
[11]	3513	-	10195	147	9.50	33	1429	Virtex-5	163 bit Karatsuba Mul
[17]	6536	4075	17305	262	12.90	84	3379	Virtex-5	2 GNB 55 bit Mul
[21]	3446	-	10176	167	8.60	30	1429	Virtex-5	163 bit Karatsuba Mul
[23]	-	-	18505	199	11.00	-	2189	Virtex-5	163 bit Karatsuba Mul
[24]	1089	1522	3958	296	14.06	15	4168	Virtex-5	41 bit Digit Serial Mul
[25]	10363	6529	29095	153	5.10	53	780	Virtex-5	2x163 bit Mul
HPECC_1M	**4393**	**3090**	**16090**	**228**	**4.91**	**22**	**1119**	**Virtex-5**	**163 bit Mul**
LLECC_3M	**11777**	**3403**	**42192**	**113**	**3.99**	**47**	**450**	**Virtex-5**	**3x163 bit Mul**
[24]	1476	1886	4721	397	10.51	16	4168	Virtex-7	41 bit Digit Serial Mul
[25]	8736	6529	27105	223	3.50	31	780	Virtex-7	2x163 bit Mul
HPECC_1M	**4150**	**3747**	**14202**	**352**	**3.18**	**13**	**1119**	**Virtex-7**	**163 bit Mul**
LLECC_3M	**11657**	**7969**	**41090**	**159**	**2.83**	**33**	**450**	**Virtex-7**	**3x163 bit Mul**
ECC over GF2^{571}									
[10]	34892	6445	66594	107	133.00	4641	14231	Virtex-4	143 bit Karatsuba Mul
[24]	12965	10066	38547	250	57.61	747	14420	Virtex-7	143 bit Digit Serial Mul
HPECC_1M	**50336**	**29217**	**141078**	**111**	**34.05**	**1815**	**3783**	**Virtex-7**	**571 bit Mul**

overheads [9]. As the total latency of the point multiplication in projective coordinates based on the two clocked cycles loop operations is comparable to the latency of the final conversion operation, reducing the clock cycles for the conversion operation is required. The inversion operation involved in the conversion step consumes most of the clock cycles and is thus the focus for optimisation. We use a *4-square* circuit to speed up the multiplicative inversion operation. The total clock cycles(CCs) for point multiplications of the *LLECC = 5 CCs* for initialisation + *4 CCs* to start of the loop+ *(m-1)x2 CCs* for loop operations + *4 CCs* to exit loop+ *CCs* for Coordinates conversion(= (m/2) for square + *#mulx1) CCs* for inversion +23 others) as shown in Table III. The LLECC architecture consumes extra clock cycles at the start of first loop and at the end of the final loop operation due to load/unload of variables to/from the local registers. Again, the latency for inversion depends on the curve size and defined by $\lfloor \log_2 m - 1 \rfloor + h(m-1) - 1$, where $h(m-1)$ is the Hamming weight. The other clock cycles, 23 clock cycles that are independent of curve size include mainly 10 multiplications and 6 addition and 1 square operations. For example, the total clock cycles for *GF2^{163} = 5+4+162x2+4+113(= (81 + 9) +23) = 450* clock cycles.

VI. IMPLEMENTATION RESULTS

The architectures have been implemented (placed and routed) on Xilinx Virtex4, Virtex5 and Virtex7 FPGA technologies to enable fair comparisons to relevant reported designs on the same technologies as well as provide achievable implementation results on more recent technologies. Where feasible the designs have been implemented in each Virtex family. The FPGA size selected was the smallest in the family that could accommodate the design in terms of area and pin count.

The results of our proposed high speed ECC processor implementation on Virtex4 (XC4VLX60), Virtex5 (XC5VLX50) and Virtex7 (XC7V330T) for *HPECC* and again, Virtex5 (XC5VLX110) and Virtex7 (XC7V690T) for *LLECC* over *GF(2^{163}),* and Virtex7 (XC7VX980T) for *HPECC* over *GF(2^{571})* using Xilinx ISE 14.5 tool after place and route are shown in TABLE IV. The presented results are achieved with the use of high speed timing closure techniques. We used repeated place and route for different timing constraints to achieve the best possible result.The high performance ECC implementations over *GF(2^{163})* based on one multiplier (*HPECC_1M*) on Virtex4, Virtex5 and Virtex7 consume 12964 slices, 4393 slices and 4150 slices and can operate at maximum

clock frequencies of 210 MHz, 228 MHz and 352 MHz respectively. The achievement of high frequency is due to the design of the high performance field multiplier. Our Low latency ECC processor based on three parallel multipliers (*LLECC_3M*) improves speed by reducing latency with an area overhead. The proposed *LLECC* on Virtex7 can manage 159 MHz frequency by consuming the same area of the Virtex5 (113 MHz and 11777 Slices).

TABLE IV provides detailed comparison to state of the art using the same technology.

Our previous high throughput design presented [24] is the best reported implementation in terms of area-time metric; our *HPECC* implementation presented here over $GF(2^{163})$ on *Virtex7* achieves a better metric value (area-time metric of 13) even using a full precision multiplier. Our previous high speed ECC implementation presented in [25] is the fastest FPGA design to date on Virtex7. Our proposed design in this paper outperforms [25] in both speed and area-time metrics.

For Virtex4, the previous highest speed implementation is presented in [14] and consumed 20807 slices to achieve 7.72 µs using three 82 bit parallel multiplier cores. Our *HPECC* implementation on Virtex4 consumes 38% less area and shows 31% speed improvement. Again, our work uses less arithmetic (163 bit multiplier) resource to gain 2.33 times improvement in the area-time metric(Slices x Time x 10^{-3}) as compared to the work in [14]. In [16], the authors presented a high speed design that used 17929 slices to attain 9.60 µs for the point multiplication time; meanwhile, our proposed work on Virtex4 is 45% faster than that in [16] and consuming less area. The work presented in [15] uses three 55 bit multipliers consumed two times the area to achieve 10 µs whereas our design can show two times better speed. The most relevant work is presented in [11] where the authors using a 163 bit multiplier with four stage pipelining to achieve maximum clock frequency 131 MHz. Our design is based on 163 bit multiplier with two stages pipelining achieved a clock frequency of 210 MHz that is 60% clock frequency speed up improvement. Again, our ECC processor implementation is twice as fast with only 60% more slices; this translates to 21% improvement in the area-time metric than the reported efficient design in [11]. Our design shows 18% better area-time metric than the previous best optimized design presented in [10]. The work presented in [12] used pipelining technique to achieve high clock frequency. Our proposed ECC processor uses 2 stages pipelining to get 36% improvement in clock frequency speed over [12]. The work in [21] is the previous version of [11]. The work in [22] and [23] are a similar implementation to [11]; however, [11] is a LUTs optimised implementation. In comparison with [21], [22] and [23], our work shows better results than the best results they presented.

For Virtex5, the best reported performance result over $GF(2^{163})$ is 5.48 µs and is presented in [13] with 6150 slices. Our proposed ECC processor consumes only 4393 slices to compute a point multiplication in 4.91 µs is better in both speed (10%) and area (29%) than that in [13]. Our state of art achieves double the speed of [11] but consuming only 25 % more slices. The presented work in [17] consumes 6536 slices to get a speed of 12.9 µs; our area-time metric is 3.81 times better than that in [17].

The proposed *HPECC* architecture over $GF(2^{571})$ (the highest security NIST curve) is the first reported full precision multiplier based implementation and sets a new time record for point multiplication (37.5 µs on Virtex7).

Our low latency ECC (*LLECC*) requiring only two clock cycles for Montgomery point multiplication is the first implementation in the literature with such schedule. The proposed *LLECC* design has the lowest latency figure (450 clock cycles for the curve over $GF(2^{163})$) reported to date while still achieving a high clock frequency thanks to the novel pipelining technique in the field multiplier and the smart breaking of the long critical path delay by inserting local registers. Furthermore, the *LLECC* over $GF(2^{163})$ implemented on Virtex7 shows the fastest ever figure for point multiplication (2.83 µs) on FPGA at the theoretical limit of performance.

VII. CONCLUSIONS

This paper presented a very high speed elliptic curve cryptography processor for point multiplication on FPGA based on a novel 2 stages pipelined full-precision multiplier in *HPECC*, and a 1 stage pipelined full-precision multiplier in *LLECC* with careful scheduling in both cases for the combined Montgomery point multiplication algorithm.

Our proposed high performance one multiplier based architecture takes six cycles for a loop of the Montgomery point multiplication in the projective coordinates without any pipelining delay whereas our low latency ECC (3-multiplier based) processor takes only two clock cycles. The architectures have been implemented (placed and routed) on Xilinx Virtex4, Virtex5 and Virtex7 FPGA families resulting in the fastest reported implementations to date to the best knowledge of the authors. On the Virtex4 our ECC point multiplication over $GF(2^{163})$ takes 5.32 µs with 13418 slices - is faster than the fastest previously reported Virtex 4 design [14] and also faster than the fastest reported design to date (5.48 µs) which was on a Virtex 5 [13]. On Virtex5, our design over $GF(2^{163})$ is not only even faster at 4.91 µs but also smaller than that of [13]. Our implementation on the new Virtex7 FPGA technology achieves the best area-time performance with the highest speed to date; an ECC implementation takes only 3.18 µs using 4150 slices. To evaluate scalability of our contributions, we also implemented the proposed one multiplier based architecture over $GF(2^{571})$, the highest security curve in the NIST standard [5], on Virtex 7; this is the first reported implementation, which can complete a point multiplication by taking only 37.54 µs. Our parallel multipliers based ECC design is the first reported full-precision parallel architecture which shows the highest speed (2.83 µs) for the point multiplication over $GF(2^{163})$ with the lowest latency (450 clock cycles) on FPGA.

The proposed ECC processor implementations would enable faster deployment of public-key cryptography protocols for example in terms of key agreement (ECDH) and digital signatures (ECDSA) across a range of platforms with improved efficiency in terms of area/power resource.

REFERENCES

[1] N. Koblitz, "Elliptic Curve Cryptosystems," Mathematics of

Computation, vol. 48(177), pp. 203-209, January 1987.
[2] V.Miller, "Use of Elliptic Curves in Cryptography," Advances in Cryptology- *CRYPTO'85,* pp. 417-426, Springer, 1986.
[3] N. Koblitz, A. Menezes, and S. Vanstone, "The State of Elliptic Curve Cryptography," Des. Codes Cryptography, vol. 19, no. 2-3, pp. 173-193, Mar. 2000.
[4] D. Hankerson, A. Menezes, and S. Vanstone, Guide to Elliptic Curve Cryptography. New York, Springer-Verlag, 2004.
[5] U. S. Department of Comerce/NIST, "National Institute of Standards and Technology," Digital Signature Standard, FIPS Publications 186-2,January 2000.
[6] J. Lopez and R. Dahab, "Fast Multiplication on Elliptic Curve Over GF(2^m) without Precomputation," in Proc. 1st Int. Workshop Cryptograph. Hardw. Embedded Syst., 1999, pp. 316-327.
[7] S. Kummar, T. Wollinger, and C. Par, "Optimum digit serial GF(2^m) multiplier for curve based cryptography," IEEE Trans. Comput., vol. 55, no. 10, pp. 1306-1311, Oct. 2006.
[8] Z. Khan and M. Benaissa, "Low area ECC implementation on FPGA," in Proc. IEEE 20th Int. Conf. Electronics, Circuits, and Systems, 2013, pp.581-584.
[9] T.Itoh and S. Tsujii,"A fast algorithm for computing multiplicative inverses in GF (2^m) using normal bases,*" J. Inf. Comput*., vol. 78, no.3, pp. 171-177, Sep. 1988.
[10] B. Ansari, M. Hasan, "High-Performance Architecture of Elliptic Curve Scalar Multiplication," IEEE Trans. Computers, vol.57, no.11, pp.1443-1453, Nov.2008.
[11] S. Roy, C. Rebeiro, and D. Mukhopadhyay, "Theoretical Modeling of Elliptic Curve Scalar Multiplier on LUT-Based FPGAs for Area and Speed," IEEE Trans. VLSI Systems, vol. 21, no. 5, pp. 901–909, May. 2013.
[12] W. Chelton and M. Benaissa, "Fast Elliptic Curve Cryptography on FPGA," IEEE Trans. VLSI Systems, vol. 16, no. 2, pp. 198–205, Feb. 2008.
[13] G. Sutter and J. Deschamps and J. Imana, "Efficient Elliptic Curve Point Multiplication Using Digit Serial Binary Field Operations," IEEE Trans. Ind. Electron., vol. 60, no. 1, pp. 217-225, 2013.
[14] 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, Oct. 2010.
[15] H. M. Choi, C. P. Hong and C. H. Kim "High Performance Elliptic Curve Cryptographic Processor Over GF(2^{163})," in proc. 4th IEEE Intl. Symposium on Electronic Design, Test & Applications, DELTA,2008, pp. 290 – 295.
[16] H. Mahdizadeh, and M. Masoumi, "Novel Architecture for Efficient FPGA Implementation of Elliptic Curve Cryptographic Processor Over GF(2^{163})," IEEE Trans. VLSI Systems, vol. 21, no.12, pp. 2330-2333, Dec. 2013.
[17] 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. VLSI Systems, vol. 20, no. 8, pp. 1453-1466, Aug. 2012.
[18] H. Fan, M. A. Hasan, " A survey of some recent bit-parallel multipliers," Elsevier - Finite Fields and Their Applications, Vol. 32, pp. 5-43, March ,2015.
[19] M. A. Hasan, A.H. Namin, and C. Negre., "Toeplitz Matrix Approach for Binary Field Multiplication Using Quadrinomials," IEEE Transactions on VLSI Systems, vol.20, no.3, pp.449-458, March, 2012.
[20] B. Rashidi, R.R. Farashahi, S.M. Sayedi, "High-speed and pipelined finite field bit-parallel multiplier over GF(2^m) for elliptic curve cryptosystems," in Proc. 11th Int. ISC Conf. on Info. Security and Cryptology (ISCISC), 2014, pp.15-20.
[21] C. Rebeiro, S. Roy, and D. Mukhopadhyay, "Pushing the Limits of High-Speed GF(2^m) Elliptic Curve Scalar Multiplication on FPGAs," *lecture Notes in Comp. Sc. – CHES* 2012 vol. 7428, pp. 496-511.
[22] S. Liu, L. Ju, X. Cai, Z. Jia, Z. Zhang, "High Performance FPGA Implementation of Elliptic Curve Cryptography over Binary Fields," *in proc. 13th IEEE Int. Conf. on Trust, Security and Privacy in Comp. and Communications(TrustCom)*, 2014, pp.148-155.
[23] A.P. Fournaris, J. Zafeirakis, and O. Koufopavlou, "Designing and Evaluating High Speed Elliptic Curve Point Multipliers," in *proc. 17th Euromicro Conf. on, Digital System Design (DSD),* 2014, pp.169-174.
[24] Z. Khan, M. Benaissa, "Throughput/Area Efficient ECC Processor on FPGA," *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol.62, no.11, pp.1078-1082, Nov. 2015.
[25] Z. Khan and M. Benaissa, "High Speed ECC Implementation on FPGA over GF(2m)," in Proc.25th Int. Conf. on Field-programmable Logic and Applications (FPL), 2-4 Sept. 2015, pp.1-6.
[26] N. Petra, D. D. Caro and A. G. M. Strollo, "A Novel Architecture for Galois Fields GF(2^m) Multipliers Based on Mastrovito Scheme," IEEE Transactions on Computers, vol. 56, no. 11, pp. 1470-1483, Nov. 2007.
[27] H. Fan and M. A. Hasan, "Fast Bit Parallel-Shifted Polynomial Basis Multipliers in GF(2^n)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 12, pp. 2606-2615, Dec. 2006.

Zia U. A. Khan received his BSc Engg. in Electrical and Electronic Engineering from Chittagong University of Engineering and Technology, Bangladesh. He received his MSc Engg. in Data Communication Engineering from The University of Sheffield, UK in 2010. He is currently a PhD candidate at the University of Sheffield. His research interests are hardware and hardware/software design and implementation of arithmetic circuits and cryptography processors for high speed, low power and scalable applications. He is a student member of IEEE.

Mohammed Benaissa received the PhD degree in VLSI signal processing from the University of Newcastle, Upon Tyne, UK, in 1990. He has been with the Department of Electronic and Electrical Engineering, University of Sheffield, Sheffield, UK, since 1999. His research interests focus on the design and implementation of innovative electronic circuits and systems and their application to Communications and Healthcare. He has published over 150 papers on contributions to algorithmic, architectural, and circuit issues in these areas. He is a Senior Member of the IEEE, a College and Panel member of the EPSRC, and has served on the Technical Program Committees of numerous conferences.