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.
The proposed full-precision \(GF(2^m)\) field multiplier (including reduction) with segmented pipelining is shown in Fig. 1 and consists of two stages of 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
operations (addition). The result of the addition is registers. The outputs of the unit, we reduce the results in the critical path delays depend on the size of the segment, the total critical path delay into two parts: the critical path delay performing multiplication. The pipelining of the multiplier divides the multiplications results in the 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.

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 \( T_T + (\log_2 (m/n) + n + 2r) \) \( T_X \). 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] \( m \times m > (n - 1)m + (r - 1)m \). 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

![Fig. 1. Proposed Segmented Full-precision Multiplier over GF(2^m).](image-url)
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 \( t \) from \( t=2 \) down to \( 0 \) do

If \( k_t = 1 \) then

If \( k_{t+1} = 0 \) then

Point addition: \( P(X_i, Z_i) = P(X, Z) + Q(X, Z) \) and Point Doubling: \( Q(X, Z) = 2Q(X, Z) \)

St1: \( X_1 = X_1 \) \( Z_1 \); \( A \leftarrow Z_1 \)
St2: \( X_2 = X_1, Z_2 \); \( A \leftarrow Z_2 \)
St3: \( X_2 = X_1, Z_2 \); \( A \leftarrow Z_2 \)
St4: \( Z_2 = R_1 \); \( A \leftarrow Z_2 \)
St5: \( X_1 = X_1 \) \( Z_1 \); \( A \leftarrow Z_1 \)
St6: \( X_1 \leftarrow Z \) \( Z_1 \) + \( X_1 \)

If \( k_{t+1} = 1 \) then

Point addition: \( P(X_i, Z_i) = P(X, Z) + Q(X, Z) \) and Point Doubling: \( Q(X, Z) = 2Q(X, Z) \)

St1: \( Z_1 = X_1 \) \( Z_1 \); \( A \leftarrow Z_1 \)
St2: \( Z_2 = X_2 \) \( Z_2 \); \( A \leftarrow Z_2 \)
St3: \( Z_2 = X_2 \) \( Z_2 \); \( A \leftarrow Z_2 \)
St4: \( Z_2 = R_1 \); \( A \leftarrow Z_2 \)
St5: \( X_1 = X_1 \) \( Z_1 \); \( Z_1 \leftarrow A \)
St6: \( X_1 \leftarrow Z_1 \) \( Z_1 \) + \( X_1 \)

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.
intermediate results of field operations in the local registers (R₁, R₂, 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 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 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 \( k_i \), \( k_{i+1} = 0 \) 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
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 $Z_i$. In the same time, the calculated $X_2$ is required and saved in the accumulator for squaring as $k_1 = 1$.

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 $GF(2^{163})$ and $GF(2^{571})$ to evaluate the ECC performance. In the implementation over $GF(2^{163})$, 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 $GF(2^{571})$, 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 processor 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

<table>
<thead>
<tr>
<th>Ref</th>
<th>Critical path delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPECC</td>
<td>$T_{max}$ or $(\log_2(n + 2r))T_{max} + T_{add} + 2T_{max}$</td>
</tr>
<tr>
<td>LLECC</td>
<td>$T_{max} + T_{add} + T_{max} + 3T_{max}$</td>
</tr>
</tbody>
</table>

$n = \#\text{Segments}, r$ is the r-nomial irreducible polynomial, $T_{max} = 2x1$ mux delay, $T_{max} = \log_2(k)$, $T_{add} = T_{x}$

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 tapped 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 $GF2^{163}$ are $5 + (6x162) + 139(= (81+27+3) +28) + 3 = 1119$ cycles. Similarly, the latency of HPECC processor over $GF2^{257}$ 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].

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, theadder 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 $R_s$). For a particular value of $k$, $k_1 = 1$, the multipliers $Mul1$, $Mul2$ and $Mul3$ as shown in Fig. 6 calculate $X_2 \rightarrow X_2$, $R_1 = Z_1$, $Z_2 \rightarrow X_1$, $R_3 = Z_2$, $Z_1 \rightarrow X_1^2$, $R_5 = Z_2^2$. In the same step, a cascaded squaring of $X_2$ is performed to obtain the 4-square operation ($R_5 \leftarrow X^2_2$) followed by save in the $R_7$ register. In step 2, one input of $Mul1$, $X_1 + Z_2^2$ (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 $R_7$ register as $Z_2$. In the next loop. The output of $Mul1$ 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 inputs $Mul1$ ($Z_2$) and $Mul2$ ($X_1$). The $Mul3$ output ($Z_2$) of step1 is saved in the register $R_7$ 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 $R_1$ and $R_s$. Again, the inputs of $Mul3$ of step2 are $b$ from the memory unit and $Z_2^4$ from register, $R_s$ and the multiplication output is added with the content of $R_s$. 

Fig. 6. Proposed Low Latency ECC Processor Architecture
The field multiplication takes 1 clock cycle delay due to one square and addition) for a point-multiplication loop operation. In Fig. 7, the data diagram shows the utilization of three full-precision multipliers called for the values of \( k \) in Fig. 7 and Fig. 8.

The data flow diagram shown in Fig. 7 is for the values of \( k_{i+1} = 1, k_1 = 1 \) 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. We show a data flow diagrams to illustrate the point operations for the different combinations of the previous, current and next values of \( k \) in Fig. 7 and Fig. 8.

The data flow diagram shown in Fig. 7 is for the values of \( k_{i+1} = 1, k_1 = 1 \) 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 \( \text{Mul}1 \), \( \text{Mul}2 \) and \( \text{Mul}3 \) 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, \( \text{Mul}1 \) and \( \text{Mul}2 \) outputs (i.e., \( Z_2 \) and \( X_2 \)) 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 \( \text{Mul}1 \) and also \( Z_1 \) is saved in Register 1 (\( R_1 \)) to use in the next loop. The output of \( \text{Mul}3 \) 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 \( X_2 \) by adding the outputs of \( \text{Mul}1 \) and \( \text{Mul}2 \) and we also get \( X_2 \) by adding the output of \( \text{Mul}3 \) and the content of \( R_2 \) (\( X_2^2 \)). The \( X_2 \) and its square, \( X_2^2 \) are directly applied as an input of \( \text{Mul}1 \) and \( \text{Mul}3 \) 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.

Thus, all inputs that are required to begin the next loop are ready. The dataflow diagram is the same for the combination of values \( k_{i+1} = 0, k_1 = 0 \) and \( k_{i-1} = 0 \) except that the variables are changed as shown in Algorithm 3.

In Fig 7, a data flow diagram of the loop of point multiplication is presented for the values of \( k_{i+1} = 1, k_1 = 0 \) 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 \( k_1 = 1 \) to the loop of \( k_1 = 0 \).

- In clock cycles 1 and 2, the point operations for the value of \( k_1 = 1 \) is performed. As the next loop for \( k_1 = 0 \), the squared outputs of the loop (\( k_1 = 1 \)) should be \( Z_2^2, Z_2^4 \) and \( X_2^2 \) instead of \( Z_2^4, Z_2^8, Z_2^{16} \) and \( X_2^8 \). In the loop, \( Z_2^2 \) is calculated and saved in \( R_5 \) in the st2. Again, the output \( X_2 \) of the loop will be squared and 4-squared to get \( X_2^2 \) and \( X_2^4 \) in the st1 of the next loop (\( k_1 = 0 \)).

- In st1 of the loop of \( k_1 = 0 \) (at clock cycle 3), the \( X_2^4 \) is used as \( \text{Mul}3 \) input, the \( X_2^8 \) is saved in \( R_2 \). In the same state, the content of \( R_2 \) (\( Z_2^2 \)) is squared to get \( Z_2^4 \) and saved in \( R_4 \). Thus, the second loop for \( k_1 = 0 \) can be started with three multipliers inputs \( X_2 \). \( Z_1 \), \( X_2 \) and \( Z_2^4 \) \( X_2^8 \) after the previous loop (\( k_1 = 1 \)). In this case, the loop (\( k_1 = 0 \)) inputs of \( \text{Mul}1 \) and \( \text{Mul}2 \) are the same as the inputs of the previous loop (\( k_1 = 1 \)) due to the last output (the addition of \( R_2 \) and \( \text{Mul}3 \)) 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 \( k_1 = 0 \) (clock cycles 5 and 6) is similar to Fig. 6, (no transition) except that the variables are changed as shown in Algorithm 3.
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 $MULGF2 + \text{the reduction part} + \text{adder} + \text{square} + 3x1 \text{mux}$ 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.

\[ \text{TABLE III:} \]

<table>
<thead>
<tr>
<th>ECC</th>
<th>Initial + point operations + Conversion</th>
<th>$GF(2^{63})$</th>
<th>$GF(2^{73})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPECC</td>
<td>$5 + (6M_M)(m-1)$</td>
<td>1099</td>
<td>3783</td>
</tr>
<tr>
<td></td>
<td>$+ (7M_M + \text{Inv} + 3M_M + 8)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLECC</td>
<td>$5 + (4 + 2M_M)(m-1) + 4$</td>
<td>450</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$+ (7M_M + \text{Inv} + 3M_M + 3)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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(CC)s for point multiplications of the LLECC = 8 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 + #mul81) 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 $\log_2(m-1) + h(m-1)-1$, 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 $GF(2^{163}) = 5+4+162x2+4+113(= (81 + 9) +23) = 450$ 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 (XC7VX330T) for HPECC and again, Virtex5 (XC5VLX110) and Virtex7 (XC7VX980T) for LLECC over $GF(2^{163})$, and Virtex7 (XC7VX980T) for HPECC over $GF(2^{255})$ 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 the use of a single multiplier (HPECC_1M) on Virtex4, Virtex5 and Virtex7 consumes 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


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 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.