This is a repository copy of *Throughput/Area-efficient ECC Processor Using Montgomery Point Multiplication on FPGA*.

White Rose Research Online URL for this paper: http://eprints.whiterose.ac.uk/131696/

Version: Accepted Version

**Article:**

https://doi.org/10.1109/TCSII.2015.2455992

**Reuse**
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

**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.
Throughput/Area Efficient ECC Processor on FPGA

Zia-Uddin-Ahamed Khan, Student Member, IEEE, Mohammed Benaissa, Senior Member, IEEE

Abstract—High throughput while maintaining low resource is a key issue for Elliptic Curve Cryptography (ECC) hardware implementations in many applications. In this paper, an ECC processor architecture over Galois Fields is presented that achieves the best reported throughput/area performance on FPGA to date. A novel segmented pipelining digit serial multiplier is developed to speed up ECC point multiplication. To achieve low latency, a new combined algorithm is developed for point addition and point doubling with careful scheduling. A compact and flexible distributed RAM based memory unit design is developed to increase speed while keeping area low. Further optimisations were made via timing constraints and logic level modifications at the implementation level. The proposed architecture is implemented on Virtex4 (V4), Virtex5 (V5) and Virtex7 (V7) FPGA technologies and respectively achieved throughout/slice figures of 19.65, 65.30 and 64.48 (10⁶/ (Seconds x Slices)).

Index Terms—Elliptic Curve Cryptography (ECC), Point Multiplication (PM), Field Programmable Gate Array (FPGA), Throughput per Area (throughput/area), Efficiency.

I. INTRODUCTION

PUBLIC key based information security networks use cryptography algorithms such as Elliptic Curve Cryptography (ECC) and RSA. ECC has emerged recently as an attractive replacement to the established RSA due to its superior strength-per-bit and reduced cost for equivalent security [1]. High speed ECC is a requirement for matching real-time information security, however, in many applications the hardware resource implications may be prohibitive and the required high speed performance would need to be achieved within a restricted resource performance. FPGA based Hardware acceleration of ECC has seen a surge of interest recently. There are several state of the art FPGA implementations aimed at the high speed end of the design space [7 -13]. Most of these however use increased hardware resource to achieve the speed improvements sacrificing overall efficiency in terms of the throughput/area metric; such efficiency is desirable in many emerging low resource applications in particular in wireless communications. Area optimised high speed ECC design is challenging; there are requirements of algorithmic optimisation, careful scheduling to reduce clock cycles, size of multiplier, critical delay of the logic, and pipelining issues [7], [9].

In ECC, scalar point multiplication (PM) is the main operation. The PM can be implemented over either prime fields, GF(p) or binary extension fields, GF(2ᵐ) adopting either projective coordinates or affine coordinates. Binary extension fields called also finite fields (FFs) are more suited to hardware implementation due to their lower complexity FF multipliers, simple FF adder and single clocked FF squaring circuits. Projective coordinates are suited to throughput/area efficient ECC designs, where the costly inversion operation is avoided and the inversion operation required to convert projective into affine coordinates can be achieved by multiplicative inversion [2], [6]. ECC computations in the projective coordinates system are based on large operand finite field operations of which multiplication is the most frequently performed. The high speed performance of ECC designs therefore would depend mainly on the performance of the FF multipliers. Digit serial FF multipliers are often used to reduce latency; popular multipliers here include the direct method based multipliers and Karatsuba [7], [10]. If the field size is m and the digit size is w of a digit serial multiplier, then the number of clock cycles for each FF multiplication is s + c, where s = m/w, and c is for clock cycles due to data read-write operations. Thus, large digit multipliers can reduce clock cycles (latency) with increasing complexities of area and critical path delay. The critical path delay can be reduced using pipelining with some extra latency [9].

In this paper, we present an area-time (throughput/slice) efficient ECC processor over binary fields in projective coordinates on FPGA. We implement the Lopez-Dahab (LD) modified Montgomery algorithm for fast PM. We demonstrate a new "no idle cycle" [7] combined point operations (point addition and point doubling) algorithm to remove idle clock cycles in between two successive point operations. We schedule point operations very carefully to avoid the idle clock cycles due to data dependency, read-write operations, and pipelining. In addition, our efficient arithmetic circuit includes a digit serial multiplier, an adder and a square circuit. The presented arithmetic unit can support on-the-fly addition and square operations while performing FF multiplication. Moreover, we present an improved Most Significant Digit(MSD) serial multiplier utilizing segmented pipelining similar to the Least Significant Digit (LSD) multiplier presented in [2,4]. We develop an optimized distributed RAM based memory unit for flexible data access to support reduced data dependency in the arithmetic operations. We adopt the Itoh- Tsujii inversion algorithm for inversion to save area [5, 6]. Finally, we use a
A. ECC over GF(2^m)

ECC over GF(2^m) is suitable for hardware implementation. The main operation of ECC is scalar point multiplication \( Q = k \cdot P \), where \( k \) is a scalar (integer), \( P \) is a point on the elliptic curve, and \( Q \) is a new point of the curve after \( k \cdot P \) [2].

Let \( E \) be an elliptic curve in the binary extension field. \( E \) is defined by a set of points \((x, y)\), and a point at infinity \( \infty \), which satisfy the equation below:

\[
y^2 + a_1 y = x^3 + a_2 x^2 + b_1
\]

Where \( a \) and \( b \) are elements of the finite field, GF(2^m) and \( b \neq 0 \) [2]. The point multiplication \((k \cdot P)\) is accomplished by point addition and point doubling depending on \( k_i \), the \( i \)th value of \( k \). The projective coordinates based Lopez-Dahab (LD) modified Montgomery point multiplication algorithm, as shown in Algorithm 1, has been adopted by many designs in high performance ECC design [7-13] space due to its speed, side-channel attack resistance, suitability of parallelisation and is low resource friendly.

III. RESOURCE CONSTRAINED HIGH THROUGHPUT ECC

For a high throughput ECC implementation in the low area end dedicated finite state machine based control unit to speed up the control operations. The proposed architecture is implemented on different FPGA technologies, Virtex4 (V4), Virtex5 (V5) and Virtex7 (V7), and compared to state of the art in terms of a throughput/slices metric. The throughput/area performance in (1x10^6/s)/(slices) of our proposed design (19.65 on V4, 65.30 on V5 and 64.70 on V7) outperforms state of art designs on FPGA to date.

The rest of the paper is organized as follows. Section II discusses preliminaries of PM, and the Lopez-Dahab modified Montgomery point multiplication in projective coordinates. Section III reviews resource constraints in high throughput ECC. Section IV illustrates the proposed design. Section V presents the results of the FPGA implementation and a comparison with recently published state of art designs on FPGAs, followed by conclusions in section VI.

II. PRELIMINARIES

A. ECC over GF(2^m)

ECC over GF(2^m) is suitable for hardware implementation. The main operation of ECC is scalar point multiplication \( Q = k \cdot P \), where \( k \) is a scalar (integer), \( P \) is a point on the elliptic curve, and \( Q \) is a new point of the curve after \( k \cdot P \) [2].

Let \( E \) be an elliptic curve in the binary extension field. \( E \) is defined by a set of points \((x, y)\), and a point at infinity \( \infty \), which satisfy the equation below:

\[
y^2 + a_1 y = x^3 + a_2 x^2 + b_1
\]

Where \( a \) and \( b \) are elements of the finite field, GF(2^m) and \( b \neq 0 \) [2]. The point multiplication \((k \cdot P)\) is accomplished by point addition and point doubling depending on \( k_i \), the \( i \)th value of \( k \). The projective coordinates based Lopez-Dahab (LD) modified Montgomery point multiplication algorithm, as shown in Algorithm 1, has been adopted by many designs in the high performance ECC design [7-13] space due to its speed, side-channel attack resistance, suitability of parallelisation and is low resource friendly.

III. RESOURCE CONSTRAINED HIGH THROUGHPUT ECC

For a high throughput ECC implementation in the low area end

Algorithm 1 LD Montgomery Point multiplication over GF(2^m) [3]

INPUT: \( k = (k_1, ..., k_m, k_m) \) with \( k_{m-1} = 1 \), \( P = (x, y) \in E(F_{2^m}) \)
OUTPUT: \( kp \)

Initial Step: \( p(X, Z) \leftarrow (x, 1), 2p = Q(X, Z) \leftarrow (x^2 + b, x^2) \)

For \( i \) from \( t \) to \( 2 \) down to \( 0 \) do

If \( k_i = 1 \) then

Point addition \( p(X, Z) = p(X, Z) + Q(X, Z) \)

Point Doubling: \( Q(X, Z) = 2Q(X, Z) \)

1. \( Z_1 = X_1 \cdot Z_1 \)
2. \( X_1 = X_1 + Z_1 \)
3. \( T = X_1 + Z_1 \)
4. \( X_2 = X_1 \cdot T \)
5. \( Z_2 = X_1 \cdot T \)
6. \( T = x \cdot Z_1 \)
7. \( X_1 = X_1 + T \)
8. \( Return P(X, Z) \)

Conversion Step: \( x = X_1 / Z_1 ; y = \left( \frac{x + z}{Z_1} \right) (X_1 + x Z_1) (x^2 + Z_1) + (x^2 + y) (Z_1 Z_2) (x Z_1 Z_2)^2 + y \)

IV. PROPOSED THROUGHPUT/AREA EFFICIENT ECC PROCESSOR

Our proposed area optimized high throughput architecture is presented in Fig.1. The design consists of an efficient arithmetic unit, an optimised memory unit and a dedicated control unit.

A. Segmented Pipelining Based Digit Serial Multiplier

The arithmetic unit consists of a novel most significant digit (MSD) serial multiplier, a square and adder circuit as shown in Fig.1. Digit serial multiplication for the high speed ECC implementation end tended to be either in direct form (i.e. MSD serial Multiplier) [10] or in bit parallel form (i.e. Karatsuba multiplier) [8], [9]. There are some advantages of Karatsuba multiplication over MSD multiplication. A Karatsuba FF multiplication takes \( s \)-1 cycles, where \( s = m/w \), and is suitable for pipelining. An MSD FF multiplication takes \( m + 1 \) cycles where the extra clock cycle delay is due to the reduction register [2], [5], [9], and [10]. However, a pipelined Karatsuba
TABLE I

LATENCY, CRITICAL PATH DELAY AND RESOURCES OF DIGIT SERIAL MULTIPLIERS OVER GF(2^n)

<table>
<thead>
<tr>
<th>Ref</th>
<th>Latency, cc</th>
<th>Critical path delay</th>
<th>#XOR</th>
<th>#AND</th>
<th>#FFs</th>
<th>#Maxx</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15]</td>
<td>2C</td>
<td>((2 + \log(d_{2}) T_X))</td>
<td>(m + C(2d + S_1) + d + S_2)</td>
<td>(CS_3)</td>
<td>((C + 2)m + CS_4)</td>
<td></td>
</tr>
<tr>
<td>[16]</td>
<td>(m \log_{2}2)</td>
<td>(T_X + (2 + 3\log(m) T_X))</td>
<td>(\frac{\theta m \log_{2}n - 2m - \frac{1}{2} m \log_{2}n^2 - \frac{1}{2} + 2m - 2}{2})</td>
<td>(-)</td>
<td>(-)</td>
<td></td>
</tr>
<tr>
<td>[17]</td>
<td>2C</td>
<td>(T_X + (\log_{2}d T_X)</td>
<td>(d(2d_{2} + d + 3))</td>
<td>(Cd_m)</td>
<td>(-)</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>(\frac{m}{\sqrt{d}})</td>
<td>(T_X + (\log_{2}d T_X))</td>
<td>(dm + s \cdot m + kd, k)</td>
<td>(d(n))</td>
<td>(s(m+1)m-l, m)</td>
<td></td>
</tr>
</tbody>
</table>

n = #segments, d= digit size, \(S_1 = \left\lceil \frac{m}{d} \right\rceil \left(2.5 d \log_{2}d - 3d + 0.5 \right) + d \log_{2}d - d, S_2 = \left\lceil \frac{m}{d} \right\rceil \left(2 d \log_{2}d - 2d \right), S_3 = \left\lceil \frac{m}{d} \right\rceil \left(2 d \log_{2}d \right), C = \left\lceil \frac{m}{d} \right\rceil.\)

Algorithm 2 Proposed combined loop operation of the LD

Montgomery point multiplication with careful scheduling

For \(t\) from 1 to \(n\) down to 0 do If \(k_{t-1} = 1\) then If \(k_{t-1} = 0\) then

Point addition:\(P(X_{t-1}Z_{t}) = P(X_{t-1}) + Q(X_{t-2}Z_{t})\) and Point Doubling:\(Q(X_{t-2}Z_{t}) = 2Q(X_{t-2}Z_{t})\).

St1: \(X_{t} = X_{t-1}Z_{t}\).
St2: \(X_{t} = X_{t-1} + Z_{t}, Z_{t} = Z_{t-1} + T\) - \(Z_{t}^{2}\).
St3: \(X_{t} = bT + X_{t}^{2}, X_{t}^{2} = X_{t-1}Z_{t}\).
St4: \(X_{t} = X_{t-1}Z_{t}, T = (X_{t-1} + Z_{t})^{2}Z_{t}^{2}, T = T - Z_{t}^{2}\).
St5: \(X_{t} = X_{t-1}Z_{t-1}, T = (X_{t-1} + Z_{t-1})^{2}Z_{t-1}^{2}, T = T - Z_{t-1}^{2}\).
St6: \(X_{t} = -xT + X_{t}\).

Conversion Step: same as Algorithm 1.

Critical path delay can be optimised (to achieve the desirable high speed) by choosing an optimum number of segmentations (n). To generalise, from Table 1, the best figure latency for a field multiplication [15,17] is \(2\left\lceil \frac{m}{d} \right\rceil\). Our multiplier’s latency is \(\left\lceil \frac{m}{d} \right\rceil\).

As a rule of thumb, therefore as long as \(m<4d\), our multiplier would achieve comparable or better latency figure. But what is crucial is that for comparable (less or higher) latency say and same digit size, our design can achieves improved critical path delay, \(T_X + (\log_{2}(d)T_X)\) in our case (due to GF2MUL) compared to \(T_X + (\log_{2}(d)T_X)\) in [15,17], using an optimum segment size without increasing the latency of the multiplier. Thus, utilising similar area, our multiplier can achieve higher speed. At the extreme, the use a full precision multiplier \((d=m)\) with an optimised segmentation would thus lead to the highest speed.

B. Optimized Memory Unit.

High speed and flexible design for the memory unit can improve performance. We consider an optimised distributed RAM based memory unit. There is an \(8x8\) size register file in a unit, one \(m\) bit register (accumulator) and one shift register (Shiftreg). The \(8x8\) register file consists of one \(m\) bit input that can load data in any location of the register file, two \(m\) bit output buses (A bus and B bus) that can access data from any location of the register file. The shift register can store data from any location of the register file to provide \(w\) size digit (bit) multiplier for the FF multiplication. The accumulator can save a result from the arithmetic unit or new data from the register file to do a square operation. The accumulator and square circuit are connected such that repeated squaring can be done without saving in the register file. The repeated squaring improves latency of multiplicative inversion as proposed in [6]. The memory unit is smartly accessible to write, read shifting operation in any location. The easy accessibility of the memory reduces the number of temporary registers for the PM. The memory unit consumes very low area to provide high speed data access. LD Montgomery point multiplication as shown in algorithm 2.

To schedule for no idle cycles [7], we combine the point addition and point doubling algorithms for the current value of \(k_t = 1\) as shown in Algorithm 2. We observe that the product of the last multiplication is \(X_1\) if \(k_t = 1\) or \(X_2\) if \(k_t = 0\). Thus, the first multiplication of the loop should be independent of the last multiplication. For example, if the last product is \(X_1\) then
the next operands of multiplication are $X_2$ and $Z_1$. Otherwise, the next operands will be $X_1$ and $Z_2$. Thus, the first multiplication depends on the last $k_i$, which means the $k_{i+1}$ bit as shown in Algorithm 2.

C. Scheduling for Point Operations

In this paper, we propose new scheduling in the combined

Fig 2 illustrates the proposed scheduling for a 41-bit digit size FF multiplier. The 41-bit digit size FF multiplier takes $M = 4$ cycles for actual multiplication, and $c = 4$, with 2 clock cycles for pipelining and 2 clock cycles for unloading from and loading to the memory unit. In a loop, the point operation in the projective coordinates system requires 6 multiplications. To ensure no idle state in the point multiplication, a new multiplication is started at every 4 clock cycles. Thus, two consequent but independent multiplications are overlapping each other as shown in Fig. 2 for $k_i = 1$ and $k_{i-1} = 1$. Again, the adder circuit placed in the common data path is capable of doing addition concurrently. The square operation takes three cycles with 1 cycle to save in the accumulator, 1 clock cycle for squaring, and 1 clock cycle for loading. Repeated squaring can be done without storing in the register file. Thus, double squaring takes 4 clock cycles.

V. IMPLEMENTATION ON FPGA AND RESULTS

Our proposed efficient ECC processor is implemented over $GF(2^{163})$, $GF(2^{233})$, $GF(2^{281})$, $GF(2^{409})$, and $GF(2^{103})$, on different FPGA technologies, namely Virtex4 (LX25_12 for f163, and LX100_12 for f233 to f571), Virtex5 (XC5VLX50_3 for f163), and Virtex7 (Vx550T_3 for f163, and V585_T for f233 to f571) using Xilinx tools versions 13.2 and 14.5 respectively. The design was implemented on Virtex4 and Virtex5 technologies to allow for a fair comparison to most relevant works, and on the Virtex7 to evaluate the performance on the newer technology. The Xilinx tools were used to set high speed properties and put subsequent timing constraints to improve the area-time product. The implementation results after place and route of our ECC designs are summarized in

![Fig. 2. Proposed careful scheduling (4 Clock cycles/multiplication)](image)

Table II. Table IV also includes area-time performance and comparison to state of the art.

As shown in Table IV, the main contribution of the segmentations in the multiplier is an increase in the clock frequency while utilizing very small resources (FFs). The clock frequency for 3 segmented (3 Seg.) pipelined multipliers based ECC design is 290 MHz on the Virtex4- that is 38 MHz more than the respective implementation of non-segmented (No Seg.) multiplier based ECC. Again, the 2 segmented (2 Seg.) pipelined multiplier based ECC shows the best throughput per slice (65.30) is implemented on Virtex5 the 3 segmented multiplier based ECC on Virtex7 shows the highest performance (only 10.51 µs for an ECC point multiplication). The optimum size of the segments is subject to a trial-error method to achieve high throughput.

Table IV shows comparisons with relevant high performance ECC designs on FPGAs in terms of efficiency metric throughput/area ((1 x10⁶/s)/slices) over $GF(2^{163})$ and $GF(2^{233})$. For $GF(2^{163})$, the previous best optimised work was reported in [7] where one 41bit pseudo-pipelined Karatsuba multiplier was used with a so-called “no-idle cycles” point multiplication approach to achieve 11.92 throughput/area figure on Virtex4. Our no-segment based ECC design consumes less area (3623 slices) and achieves higher clock frequency (252 MHz) than [7] (4080 slices, 197 MHz) and therefore has a 40% higher throughput/area efficiency. Particularly, our segmented based design shows 65% better efficiency than [7]. Our f571 achieves 180 MHz speed while the work in [7] operates at a max speed of 107 MHz. Another optimised ECC in [8] used full length (164 bit) word serial Karatsuba multiplier with pipelining and implemented on Virtex4 and Virtex5. The work in [9] uses four times bigger multiplier than ours to achieve 11.55 and 29.96 throughput/area figure on Virtex4 and Virtex5, respectively. Our 3 segmented 41 bit multiplier based design on Virtex4 is 70% and the 2 segmented 41 bit multiplier based design on Virtex5 is 118% better than [8]. In [10], the reported best throughput/area efficiency is based on three 33 bit multipliers based ECC on Virtex5 shows 9.86 in throughput/LUTs ((1 x10⁶/s)/LUTs). Our 2 segmented multiplier based ECC shows 17.9 in (1 x10⁶/s)/LUTs is 82% better than the reported most efficient design in [10]. The hardware results presented in [11], [12],

| Table II |
| LATENCY OF ECC FOR [m/s] = 4, MUL-MFp/Mp ADD-1, SQR-2 |
| Algorithm | Initial + point operations + Conversion | GF(2163) |
| [2] | $5 + (6M + 13)(m - 1) + (10M + Inv)$ | 9211 |
| Algorithm2 | $5 + (6M(on - 1)) + (17M + 3M + Inv)$ | 4168 |
| $M_4 = 4$, $M_5 = 7$, Inversion (Inv) = ($\theta$Mul for Inversion $\times M_5 + m$) |

<p>| Table III |
| FPGA IMPLEMENTATION RESULTS AFTER PLACE &amp; ROUTE IN VIRTEX7 |</p>
<table>
<thead>
<tr>
<th>m(segments size)</th>
<th>Slices</th>
<th>LUTs</th>
<th>FFs</th>
<th>Eq.</th>
<th>MHz</th>
<th>Time (10¹⁷s)/Sls</th>
</tr>
</thead>
<tbody>
<tr>
<td>163(3x14)</td>
<td>1476</td>
<td>4721</td>
<td>1886</td>
<td>397</td>
<td>10.51</td>
<td>65</td>
</tr>
<tr>
<td>233(4x14+3)</td>
<td>2647</td>
<td>7895</td>
<td>2832</td>
<td>370</td>
<td>16.01</td>
<td>24</td>
</tr>
<tr>
<td>283(5x14+1)</td>
<td>3728</td>
<td>11593</td>
<td>3973</td>
<td>345</td>
<td>20.96</td>
<td>13</td>
</tr>
<tr>
<td>409(7x14+5)</td>
<td>6888</td>
<td>20881</td>
<td>6038</td>
<td>316</td>
<td>32.72</td>
<td>4.4</td>
</tr>
<tr>
<td>571(10x14+1)</td>
<td>12965</td>
<td>38547</td>
<td>10066</td>
<td>250</td>
<td>57.61</td>
<td>1.3</td>
</tr>
</tbody>
</table>
We proposed a highly efficient FPGA ECC processor design for high speed applications over GF(2^m) without precomputation, in Proc. 1st Int. Workshop Cryptograph. Hardw. Embedded Syst., 1999, pp. 316-327.


