An Area Aware Accelerator for Elliptic Curve Point Multiplication

Malik Imran  
Dept. of Computer Systems  
Tallinn University of Technology (TalTech)  
Tallinn, Estonia  
malik.imran@taltech.ee

Samuel Pagliarini  
Dept. of Computer Systems  
Tallinn University of Technology (TalTech)  
Tallinn, Estonia  
samuel.pagliarini@taltech.ee

Muhammad Rashid  
Dept. of Computer Engineering  
Umm Al-Qura University (UQU)  
Makkah, Saudi Arabia  
mfelahi@uqu.edu.sa

Abstract—This work presents a hardware accelerator, for the optimization of latency and area at the same time, to improve the performance of point multiplication process in Elliptic Curve Cryptography. In order to reduce the overall computation time in the proposed 2-stage pipelined architecture, a rescheduling of point addition and point doubling instructions is performed along with an efficient use of required memory locations. Furthermore, a 41-bit multiplier is also proposed. Consequently, the FPGA and ASIC implementation results have been provided. The performance comparison with state-of-the-art implementations, in terms of latency and area, proves the significance of the proposed accelerator.

Keywords—elliptic curve cryptography, point multiplication, Montgomery algorithm, FPGA, ASIC

I. INTRODUCTION

The main advantages of Elliptic Curve Cryptography (ECC), as compared to the commonplace RSA algorithm, are shorter key lengths, lesser power consumption, and lower hardware cost for an equivalent security [1]–[2]. Therefore, the ECC is implemented frequently in software or hardware [3]. While the software implementation is convenient and flexible, hardware-based solutions display higher throughputs and low latency [4]–[16]. However, low-latency hardware solutions typically demand precious hardware resources; architectures providing low latency and small area are challenging to devise.

The most important operation in ECC is the point multiplication (PM) [2]. Two types of fields are generally utilized to implement PM: the prime field, i.e., $GF(p)$ and the binary extension field, i.e., $GF(2^m)$. For each of the aforementioned fields, the National Institute of Standards and Technology (NIST) provides various key length recommendations [17]. Furthermore, implementations can adopt affine or projective coordinates [5]. Another important issue in ECC is to choose between polynomial and normal basis [10]. The binary field is generally preferred over the primary filed for hardware implementations [4, 5], while projective coordinates are more suitable than the general affine coordinates to attain effective latency/area architectures [3]. Similarly, the normal basis is valuable where recurrent squarings have to be computed, while the polynomial basis is more convenient where frequent multiplications are involved [2]. Consequently, in this paper, we have selected the binary field along with projective coordinates to achieve latency/area goals, whereas the polynomial basis has been selected to execute the finite field (FF) multiplications efficiently.

In order to minimize the latency of ECC, different FF multipliers have been implemented. However, the three most frequently used multipliers are: digit level multipliers [4–8, 12, 15, 16], bit parallel multipliers [3, 9] and parallel Karatsuba multiplier [10, 11, 13, 14]. For each FF multiplication, digit level multipliers require $D=m/n$ cycles, where ‘$D$’ determines the total number of digits, ‘$m$’ defines the length of key and ‘$n$’ is the digit size. There are two possibilities to implement the digit level multipliers, either by using the digit serial [4–8], or digit parallel multiplier [12, 16]. Digit serial multipliers require ‘$D$’ clock cycles for one FF multiplication, while digit parallel multipliers utilize one clock cycle for one FF multiplication, albeit with the higher requirements of required resources [12]. For both digit serial and digit parallel multipliers, larger digit sizes reduce the number of clock cycles while smaller digit sizes reduce the critical path [12, 15]. Apart from the FF multipliers, additional techniques related to the latency optimizations are pipelining [4, 5, 10, 12, 16] and instruction level parallelism [7].

Several techniques have been adopted to reduce the hardware resources: the execution of Itoh Tsujii inversion algorithm by sharing the hardware resources of squarer and multiplier components [7, 9, 12, 16], the use of fewer temporary storage elements to keep the intermediate results during the computation of PM [12, 14, 16], the use of a single FF adder, multiplier and squarer components in the arithmetic and logic unit (ALU) of the crypto processor [12] and the use of digit serial multipliers [4–8].

In this paper, we propose an area aware latency optimized hardware accelerator architecture with $m=163, 233, 283, 409$ and 571 for the PM computation of ECC. To reduce the latency, we have performed, (a) an exploration for different stages of pipelining, and (b) an efficient rescheduling of PM instructions. Similarly, towards area reduction, we have proposed: (a) an efficient use of required memory locations,
and (b) a digit parallel least significant digit level (DP-LSD) multiplier, with the digit size of 41 bits.

II. POINT MULTIPLICATION ON ECC OVER $GF(2^m)$

For $GF(2^m)$, a Lopez Dahab projective form of the elliptic curve is described as a set of points $P(X:Y:Z)$ satisfying the following:

$$E: Y^2+XYZ= X^3+ aX^2Z^2+bZ^4$$

(1)

In (1), the terms ‘$X$’, ‘$Y$’ and ‘$Z$’ are the Lopez Dahab projective components of initial point $P(X:Y:Z)$ where $Z≠0$, ‘$a$’ and ‘$b$’ are the curve constants, with $b≠0$. Then, PM is the addition of ‘$k$’ copies of ‘$P$’ where ‘$P$’ is an initial point with ‘$x$’ and ‘$y$’ coordinates, ‘$k$’ is an integer (equal to the size of field) and ‘$Q$’ is a final point on the elliptic curve. Due to its simplicity, several algorithms are available for the computation of PM. We have selected the Montgomery algorithm [18], which consists of three steps for the computation of PM. A scalar multiplier ‘$k$’ and an initial point ‘$P$’ with its coordinates ($x_p, y_p$) are the inputs. The coordinates ($x_p, y_p$) of the final point ‘$Q$’ are the outputs. Step one in Montgomery algorithm ensures the conversion from affine to Lopez Dahab projective form, whereas step two calculates point addition (PA) and point doubling (PD) considering the valid value of the scalar multiplier ‘$k$’. At the end, step three (reconversion) reverts the Lopez Dahab projective to affine conversion.

III. ARCHITECTURE OF PROPOSED ACCELERATOR

The proposed 2-stage pipelined accelerator architecture is composed of a register file (RF), an arithmetic and logic unit (ALU), pipeline registers and an FSM-based control unit (CU), as presented in Fig. 1. The parameters for the architecture have been chosen from NIST [17].

The register file contains an ‘$8×m$’ array, as illustrated in Fig. 1. In this paper, ‘$m$’ is the ECC key length, i.e., 163, 233, 283, 409 and 571. The purpose of RF is to keep intermediate results of the Montgomery algorithm. The multiplexers $M1$ ($8×1$) and $M2$ ($8×1$) are used to fetch the operands from RF to the ALU while a single de-multiplexer $Dmux$ ($1×8$) updates the contents of the RF by using the $Mplex_out$ signal.

The ALU of the proposed hardware accelerator contains an adder and a multiplier. Input to both operators are two ‘$m$’ bit polynomials $A(x)$ and $B(x)$, where $A(x)$ is an output of routing multiplexer $M3$ while $B(x)$ comes from the RF. Each operator produces a single output polynomial, i.e., $A_{out}(x)$ or $M_{out}(x)$. Both outputs go through $M4$ for write back on the RF. In order to compute the addition of polynomials $A(x)$ and $B(x)$, bitwise exclusive-OR gates have been utilized.

Proposed multiplier: A DP-LSD multiplier with a digit size of 41 bits is utilized. The digits with ‘$d=41$’ bits of input polynomial $B(x)$ are created ($B1$–$B14$) by generating simple partial products. Parallel execution of each created digit ($B1$–$B14$) for the polynomial multiplication is then performed with the input polynomial $A(x)$. For the computation of finite field multiplication over $GF(2^{57})$, a total of 14 digits are required. Out of these 14 digits, ($B1$–$B13$) are 41 bits wide, whereas the remaining digit ($B14$) is 38 bits wide. The parallel multiplication of each ‘$B1$–$B14$’ digit with an ‘$m$’ bit polynomial $A(x)$ results ‘$d+m-1$’ bits of polynomials.

We have provided identical inputs to the DP-LSD multiplier for the computation of FF squaring. Both multiplication and squaring produce resultant polynomial of degree almost ‘$2×m$–1’ bits. Consequently, reduction is essential to execute after each FF multiplication and squaring. For this purpose, NIST reduction algorithms over $GF(2^{163})$ to $GF(2^{57})$ are implemented as described in [3]. Finally, to perform inversion over $GF(2^m)$, a square Itoh Tsujii algorithm [19] is implemented by repeatedly squaring along with FF multiplication operation, using only the multiplier unit.

For each PA and PD computation, Montgomery algorithm requires a total of 14 instructions (inst1 to inst14), as shown in column 2 of Table I. Out of these 14 instructions, seven instructions are for PA (inst1 to inst7) and remaining instructions are for PD. However, in its presented form, the algorithm can present read after write (RAW) hazards (see column 3 of Table I). In order to reduce RAW hazards, we reschedule the instructions by considering the following:

- Parallel execution of PA and PD instructions for PM computation. For example, inst1 cannot be read until the result of inst2 is available, resulting in one cycle delay. We have instead scheduled inst11 when inst2 is in the write back stage. In the next cycle, inst3 is scheduled when inst11 is in the write back stage.

- Efficient replacement of temporary storage elements ‘$T_i$’ for PA instructions and ‘$T_j$’ for PD instructions. This results in decrease of one clock cycle for one PA and PD computation. Therefore, for ‘$m$’ bit key length, ‘$n$’ number of clock cycles are reduced.

The rescheduled instructions for non-pipelined case (read – [R], execute – [E], write back – [WB]) are presented in column 4 of Table I, resulting in a single RAW hazard, shown in column 5. The corresponding sequences carried out in 2- and 3-stage pipelining architectures are shown in the right hand side of Table I. For each PA and PD computation, the last instruction for non-pipelined, 2-stage, and 3-stage pipelined cases are written back in 14, 16 and 22 clock cycles, respectively. Moreover, moving from 2- to 3-stage brings only a slight increase in clock frequency that is not beneficial.
Consequently, in the remainder of this paper, we discuss only 2-stage pipeline case.

The proposed hardware accelerator also contains an FSM-based control unit. In order to implement Montgomery algorithm, step one requires only 33 cycles. The proposed rescheduling of PA and PD requires a total of 33 cycles. Out of these 33 cycles, 32 cycles are required for the computation of PA and PD instructions, while the remaining cycle is required to inspect the key bit. Finally, the reconversion step requires two FF inversions (\(\text{inv}\)) for an additional 34 cycles. Clock cycles for each \(\text{inv}\) operation are computed by implementing \(m-1\) times repeated squares followed by 9 (for 163), 10 (for 233), 10 (for 283), 11 (for 409) and 12 (for 571) FF multiplications. The total number of required cycles for the proposed accelerator are 3798, 5402, 6568, 9454 and 12329 when moving from 163 to 571 and these can be calculated by using \(6 + 17 \times (m-1) + 2 \times (\text{inv}) + 34\).

### IV. RESULTS AND COMPARISONS

This section describes the implementation results for the proposed accelerator on ASIC and FPGA platforms. We have created a Verilog HDL model for each field size over \(GF(2^{163})\) to \(GF(2^{571})\). The HDL models are then synthesized for XC7VX690T FPGA. For ASIC results, the HDL model for \(GF(2^{571})\) is synthesized targeting a 16nm FinFET technology, using Cadence Genus and a commercial library. Synthesis results on ASIC and FPGA are given in Table II and Table III, respectively. The numerical figures of achieved latency for the proposed hardware accelerator on both FPGA and ASIC are 36.26\(\mu\)s and 5.55\(\mu\)s over \(GF(2^{571})\) respectively. Consequently, and as expected, the ECC computation in the ASIC implementation takes 6.53 times less time than in the FPGA implementation. The achieved frequency is 2.2GHz.

#### Figure-of-Merit (FoM):
To evaluate the performance of our accelerator and to perform a fair comparison with recent state-of-the-art solutions, we have defined a FoM that captures both latency and area (slices) characteristics at the same time. Latency is the time required for the computation of one PM \((k.P\text{ in }\mu\text{s})\) and is calculated by ratio of clock cycles to clock frequency. Therefore, the defined FoM is calculated by using ratio of 1 over latency times slices as given in Fig. 2.

#### Comparison using the individual latency and area:
As far as only the latency is concerned, the proposed accelerator over \(GF(2^{163})\) to \(GF(2^{283})\) is 8\%, 10\% and 15\% faster than [12]. For \(GF(2^{163})\) to \(GF(2^{571})\), this work provides 6\%, 11\%, 17\%, 16\% and 38\% speedup with respect to [15]. As compared to [16] this work shows 66\% improvement in latency for \(GF(2^{163})\), while over \(GF(2^{233})\) the solution described in [16] is 25\% faster than the proposed accelerator. Finally, for \(GF(2^{163})\) to \(GF(2^{571})\), the latency achieved in [9] is 75\%, 72\%, 67\%, 66\% and 49\% better than the proposed one.

On the other hand, when considering only the area for comparison, the work in [12] over \(GF(2^{163})\) to \(GF(2^{283})\) utilizes 69\%, 40\% and 50\% more FPGA slices than this work. As compared to [15] over \(GF(2^{163})\), the proposed accelerator consumes 4\% more hardware slices while for the remaining field sizes, i.e., \(GF(2^{283})\) to \(GF(2^{571})\), the proposed accelerator utilizes 23\%, 30\%, 52\% and 65\% less slices. While the work in [9] shows better results in terms of latency for \(GF(2^{163})\) to \(GF(2^{571})\), the proposed accelerator utilizes 59\%, 64\%, 67\%, 73\% and 78\% lower FPGA slices than [9]. Finally, as compared to [16], the proposed accelerator consumes 51\% and 64\% lower slices over \(GF(2^{163})\) and \(GF(2^{233})\).

#### Comparison using the defined FoM:
As compared to [12], the proposed accelerator achieves 36\%, 65\% and 57\%
Similarly, an efficient rescheduling of PA and PD computations is performed to shorten the critical path(s). Moreover, the pipeline registers are efficiently placed at the digit size of $GF(2^{257})$ to $GF(2^{385})$. The proposed accelerator achieves 39%, 21% and 2% lower values while for the remaining field sizes, i.e., $GF(2^{177})$ to $GF(2^{385})$, it achieves 20% and 56% higher values as compared to [9]. These results strongly suggest that our accelerator is better suited for longer key lengths.

V. CONCLUSIONS

This paper has presented an efficient 2-stage pipelined accelerator, in terms of latency and area, over $GF(2^{177})$ to $GF(2^{385})$. The proposed accelerator provides a best-in-class figure-of-merit of 6 (for $m=571$) as compared to state-of-the-art FPGA implementations. Our ASIC implementation pushes the performance envelope further, as it displays a speedup of 6.53 as compared to the same FPGA implementation.

ACKNOWLEDGEMENTS

This work was supported by the European Regional Development Fund and the European Social Fund in the context of the project “ICT programme”.

REFERENCES

[14] M. Imran, M. Kashif and M. Rashid “Hardware design and implementation of scalar multiplication in elliptic curve cryptography (ECC) over $GF(2^{257})$ on FPGA,” IEEE Int. Conf. on Information and Communication Technologies (ICICT), 2015, pp. 1–4.