A PARALLEL PROCESSING HARDWARE ARCHITECTURE FOR ELLIPTIC CURVE CRYPTOSYSTEMS

Kazuo Sakiyama, Elke De Mulder, Bart Preneel and Ingrid Verbauwhede

Katholieke Universiteit Leuven
ESAT-COSIC
B-3001 Leuven-Heverlee, Belgium

ABSTRACT

We propose a parallel processing crypto-processor for Elliptic Curve Cryptography (ECC) to speed up EC point multiplication. The processor consists of a controller that dynamically checks instruction-level parallelism (ILP) and multiple sets of modular arithmetic logic units accelerating modular operations. A case study of HW design with the proposed architecture shows that EC point multiplication over GF(p) and GF(2^m) can be improved by a factor of 1.6 compared to the case of using single processing element.

1. INTRODUCTION

The implementation of a fast Public-Key Cryptography (PKC) is a challenge in embedded systems. Among PKCs, the best well-known and most widely used PKCs are RSA [1] and ECC [2] [3]. In embedded systems, ECC is regarded more suitable than RSA because ECC operates with higher performance, lower power consumption, and smaller area of hardware. Nevertheless, the performance is much slower than private-key cryptography such as AES.

A lot of work has been reported on speeding up the performance of ECC. The work can be classified by the following optimization levels: First of all, a mathematical optimization of ECC has been investigated in order to reduce the number of modular multiplications in a point multiplication. Secondly, a high speed modular multiplier has been researched for both hardware and software implementations. Among the proposed algorithms, Montgomery's algorithm [4] is considered as one of the fastest algorithms especially in terms of hardware. Thirdly, an architecture-level improvement can be considered. Our interest in this paper lies in this level.

The originality of our design is that an EC point multiplication can still be accelerated by using multiple modular arithmetic logic units (MALUs) in parallel. Some previous work reported parallel use of two modular arithmetic units for accelerating the EC point multiplication. In that work, point doubling and point addition are reformulated so that they can be executed by two processing elements. On the contrary, our proposed architecture is flexible in the following sense: 1) it can deal with two or more MALUs. 2) parallelism can be found on the fly by dynamically checking the instructions. 3) therefore, we can deal with any type of algorithm of point multiplication. Another research interest is to clarify the appropriate number of MALUs for ECC.

The remainder of this paper is as follows. Sect. 2 gives a survey of relevant previous work for ECC implementations. In Sect. 3, our proposed system architecture is described. In Sect. 4 the architecture of the MALU is explained. The proposed MALU supports the finite field operation over GF(p) and GF(2^m). Sect 5 explains how to check the ILP in our design. Sect. 6 shows the performance evaluation result of example cases of 160/256-bit ECC over GF(p) and GF(2^m). The implementation result on an FPGA is reported and compared with other work in Sect. 7. Sect. 8 concludes the paper.

2. RELATED WORK

The idea of a unified multiplier for PKC was first introduced by Savaş, Tenca, and Koç in [5]. The most relevant published work is the one of Satoh and Takano [6]. They present a dual field multiplier with the best performance in both types of fields so-far. The throughput of an EC point multiplication is maximized by use of a Montgomery multiplier and an on-the-fly redundant binary converter. The biggest advantage of their design is scalability in operand size and also flexibility between speed and hardware area. The same idea is the basis of the work of Großschädl in [7]. The bit-serial multiplier which is introduced is performing multiplications in both types of fields. Batina, *et.al.,* reported parallel use of two modular arithmetic units for accelerating the EC point multiplication [8]. In general, there are two possible methods to find a parallelism; one is by recompilation of SW (or a controlling logic in an ASIC), another is by HW that can check the parallelism on the fly. The later method is known as a super-scalar architecture that is often used in current high-end

Kazuo Sakiyama is funded by FWO project G.0450.04. Elke De Mulder is funded by Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). This research has been supported by the EU IST FP6 projects SCARD and ECRYPT.

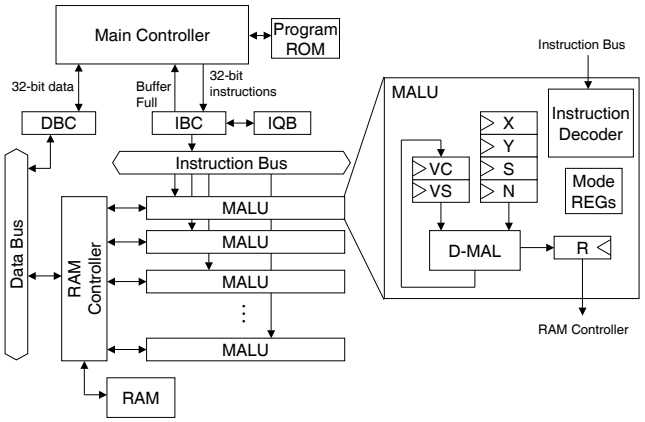

Fig. 1. System architecture of parallel processing ECC processor.

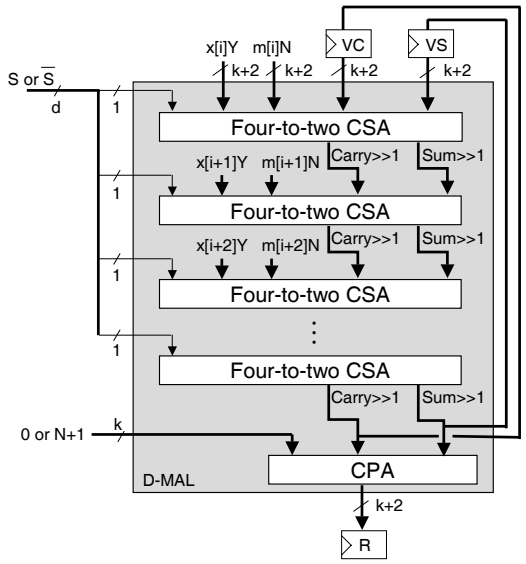

Fig. 2. Datapath of the MALU

RISC CPUs.

3. SYSTEM ARCHITECTURE

The proposed parallel computing ECC cryptosystem is composed of the main controller, several MALUs and RAM which is shared between the MALUs, *i.e.*, this RAM is the so-called single-port shared SRAM. The configuration of MALUs and RAM is assumed flexible by setting interconnections among them. The block diagram for the system is illustrated in Fig. 1.

The main controller has three in- and outputs, one of them is a signal which tells the controller to stop sending instructions when the instruction buffer is full. A 32-bit output is used to send instructions and a 32-bit input/output passes data back and forward between the controller and the datapath. A dedicated controller is chosen instead of a normal CPU because of two reasons. Firstly, instructions can be sent at a constant timing interval. Secondly, it is more compact and faster, it is possible to send one instruction per cycle. The datapath is made with a Harvard architecture, it has a separate data bus and instruction bus. The data transfer between the main controller and the MALUs is controlled by a Data Bus Controller (DBC). This path is only activated when an initial point and the parameters of an elliptic curve are sent to the RAM, or when the result is retrieved. During a point multiplication, the controller keeps on sending instructions to the instruction decoder, they are stored in an Instruction Queue Buffer (IQB) via the Instruction Bus Controller (IBC). The IQB has two main roles. First, the IBC checks if there is an instruction-level parallelism (ILP) by checking the data-dependency in the IQB and forwards them to the different MALU(s). More information about the ILP can be found in Sect. 5, for the MALUs see Sect. 4. Another purpose of the IQB is buffering the difference of the speed of issuing instructions and the processing speed. The program ROM stores the instructions which are sent by the main controller. To be able to store all instructions for an ECC point multiplication of both GF(p) and GF(2^m) (which can be chosen by setting the Mode register), only 596 bytes are necessary.

4. MODULAR ARITHMETIC LOGIC UNIT

The data path of the MALU (D-MAL) is a Carry-Save Adder (CSA) based GF(p) Montgomery modular multiplier (Fig.2). The four-to-two (4-2) CSAs are used to sum up four types of inputs which are xy, mn, vs and vc, and outputs the redundant CS-form that has a value of $(2vc + vs)$, where vs and vc are the virtual sum and the virtual carry respectively. The bit multiplications xy and mn are the main inputs for computing a bit-serial Montgomery multiplication, *i.e.* $(xy + mn + 2vc + vs)/2$ and hence the result of each multiplication is right-shifted by one bit.

Input/output vectors for such bit-level variables are described as X, Y, M, N, S, VS, and VC, where X and Y are the multiplier and multiplicand, N is the modulus, and VS and VC are intermediate variables representing a redundant CS-form. The values in vector M are calculated on the fly so that the least significant bit of VS is zero. In addition, the augend vector S is provided to the MALU by d bits in every cycle and eventually added to the result of a modular multiplication. When subtracting S, $\bar{S}$ is provided instead of S and $(N + 1)$ is added in a later stage.

The MALU has two independent stages. One is the Carry-Save(CS)-stage that operates on Montgomery's algorithm in a redundant CS-form. Another stage converts the CS-form integer into a normal integer by propagating carries, namely the Carry-Propagate(CP)-stage. Although it is possible to execute both stages with CSAs, we allocated a separate Carry-Propagate Adder (CPA) to accelerate the CP-stage. The CP-

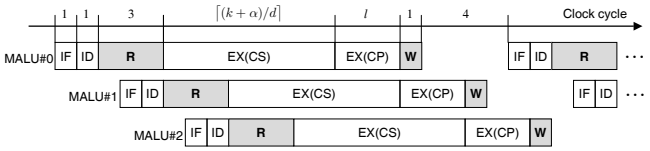

Fig. 3. Example of multiple issue of instructions over GF(p). (IF: Instruction Fetch, ID: Instruction Decode, EX(CS): Execution of CS-stage, EX(CP):Execution of CP-stage, R/W: Read/Write from/to RAM)

stage is skipped for GF(2^m) operations.

The proposed datapath is flexible in the digit size, d which can be decided by exploring the best combination of performance and cost, and the field size, k that is determined by the key-length of ECC.

Here, we define an instruction, MALU$_N$ that proceeds through the CS-stage and CP-stage consecutively and computes a field operation shown in Eq. 1. Here R is selected as $R = 2^{k+\alpha}$ where k is the bit-length of the modulus and α is a value to be determined so that the final reductions can be avoided. In our case, we chose $\alpha = 4$ and $X, Y < 4N$. For convenience of repeated use of Eq. 1, the so called Montgomery form is applied because the output is in the Montgomery form as well.

$$\text{MALU}_N(XR, YR, SR) = (XY \pm S)R \bmod N \qquad (1)$$

The whole procedure to execute MALU$_N$ starts from an instruction fetch and decode (IF and ID). Then, X, Y and S are loaded via RAM (R) for executing the succeeding CS-stage. The number of cycles necessary to execute the CS-stage varies from the configuration of the MALU, *i.e.*, $\lceil (k + \alpha)/d \rceil$ cycles are required for a modular multiplication over GF(p). After the CS-stage, an l-cycle CP-stage is executed for GF(p). The number of cycles, l is decided by considering the critical path delay of the CPA and the frequency of the provided clock. The result is stored to the RAM (W) in the last step. When using multiple MALUs, succeeding instructions should wait for three cycles to escape any memory-access conflicts. This is illustrated in Fig 3.

5. EXPLORING INSTRUCTION-LEVEL PARALLELISM

ILP is checked for all instructions as long as two or more instructions are buffered in the IQB. Here, we introduce our strategy to find ILP. A MALU$_N$ instruction has three source operands and outputs the result to the RAM, *i.e.,* MALU$_N$ deals with four types of addresses, X, Y, S, and R that are expressed as follows:

$$\text{MALU}_N : R = X, Y, S \qquad (2)$$

Table 1. Cycle counts [*cycle*] of EC point multiplication executed in different number of MALUs.

Field Type	1MALU	2MALU	3MALU	4MALU
160-bit GF(p)	165,326	111,046	103,594	103,594
256-bit GF(p)	399,785	269,886	251,826	251,826
GF(2^{163})	135,689	86,277	83,901	83,901
GF(2^{257})	326,688	210,672	205,110	205,110

With out-of-order execution, the following dependencies are possible between two instructions, MALU^i_N and MALU^j_N (i and j are labels indicating issued order; $i < j$).

1) Read-After-Write (RAW) Dependency in order execution: $R^i = X^j$, $R^i = Y^j$, or $R^i = S^j$. If the preceding result of R^i is necessary for following instructions, the instruction cannot be issued until the preceding instruction finishes.

2) RAW Dependency in out-of-order execution: $R^j = X^i$, $R^j = Y^i$, or $R^j = S^i$. In this case, Instruction MALU^j_N cannot be issued until the instruction MALU^i_N finishes.

The proposed architecture does not need to check Write-After-Read and Write-After-Write dependencies although they should be checked in a general super-scalar machine. This is because MALU$_N$ is a fixed-length instruction. Suppose we deal with D sets of instructions to search ILP, the number of conditions to check become $3(D-1)^2$. The hardware complexity for ILP expands with a large D.

6. PERFORMANCE EVALUATION OF EC POINT MULTIPLICATION

The system performance is evaluated with GEZEL [9] which can make a fast cycle-accurate simulations. The detailed hardware configuration is set as follows:

- Number of MALUs (#MALU) = 1 to 4
- MALU Configuration: $k = 160/256, d = 4$
- Depth of ILP exploration: $D = 4$

We use projective coordinates in EC point multiplication. Three sets of points, $P, 2P$, and $3P$ are pre-computed and used for the left-to-right binary algorithm [10], *i.e.*, one point addition is always executed after two point doublings are executed. A modular inversion which is necessary for a coordinate conversion is executed with Fermat's Little Theorem. In our simulation, all necessary computations for EC point multiplication are included. The performance of a 160/256-bit EC point multiplication (ECC-160p / 256p and ECC-163b / 257b) for different number of MALUs is summarized in Table 1. Considering allocation of two or three MALUs in the system, the performance improvement by a factor of 1.5 ~ 1.6 is observed compared to the case of using one MALU. Any more speed-up is not observed even when increasing the number of MALUs to more than three. Therefore, it is useless to allocate four or more MALUs in our ECC implementation

Table 2. Performance Comparison of EC point multiplication.

Ref.	Target Platform	Field	f_{max} [*MHz*]	Perf. [*msec*]	Comments
This work	Xilinx Virtex-II pro	160-bit GF(p)	100.0	1.04	3 MALUs, 6 BRAMs + 8,954 Slices
		GF(2^{163})	100.0	0.84	
		256-bit GF(p)	100.0	2.70	2 MALUs, 9 BRAMs + 10,847 Slices
		GF(2^{257})	100.0	2.11	
[6]	0.13μm-CMOS	160-bit GF(p)	137.7	1.21	64-bit multiplier 115.5 *Kgates*
		GF(2^{160})	510.2	0.19	
		256-bit GF(p)	137.7	2.68	
		GF(2^{256})	510.2	0.45	
[8]	Virtex-E	160-bit GF(p)	53	3.9	Two sets of mult.
[11]	Virtex-E	192-bit GF(p)	40	3	$p = 2^{192} - 2^{64} - 1$
[12]	Virtex-E	GF(2^{167})	76.7	0.21	Digit size = 16

from performance point of view.

7. IMPLEMENTATION RESULTS

We implemented two designs on a FPGA with different k. In case of $k = 256$, we allocated two MALUs because of resource limitation of our FPGA board. As shown in Table 2, our FPGA implementation shows an equivalent or better performance than other previous work for GF(p). However, regarding GF(2^m), the results of Satoh and Takano [6] and Orlando and Paar [12] are faster than our result for GF(2^m). This is obviously due to the faster clock frequency in case of the design of [6]. We note that the operation counts of their results (98 $Kcycle$ and 230 $Kcycle$ for GF(2^{160}) and GF(2^{256}), respectively) are more than ours. Regarding the design of [12], the architecture is intensively dedicated to GF(2^m). In this term, future work will deal with implementing a fast dual-field ECC which runs at the same clock frequency in both fields.

8. CONCLUSIONS

We introduced a parallel processing hardware architecture for ECC implementation. The proposed system is flexible in the MALU configuration and the number of processing elements. In conclusion, the performance of an EC point multiplication can be improved by using additional processing elements and an IQB that buffers the instructions and explores ILP dynamically. We could find ILP up to three instructions in our ECC program. The prototype implementation shows a significantly fast performance for both types of fields.

9. REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM,* vol. 21, pp. 120-126, 1978.

[2] N. Koblitz, "Elliptic Curve Cryptosystems," *Math. Computation,* vol. 48, pp. 203-209, 1987.

[3] V. S. Miller, "Use of Elliptic Curve in Cryptography," *Advances in Cryptology: Proceedings of CRYPTO'85. Lecture Note in Computer Science, Springer-Verlag* vol. 218, pp. 417-426, 1985.

[4] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation,* vol. 44, pp. 519-21, 1985.

[5] E. Savaş, A. F. Tenca and Ç. K. Koç, "A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m)," *Cryptographic Hardware and Embedded Systems: Proceedings of CHES'00. Lecture Note in Computer Science, Springer-Verlag,* vol. 1965, pp. 281-296, 2000.

[6] A. Satoh and K. Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor," *IEEE Trans. Computers,* vol. 52, pp. 449-460, 2003.

[7] J. Großschädl, "A bit-serial unified multiplier architecture for finite fields GF(p) and GF(2^n)," *Cryptographic Hardware and Embedded Systems: Proceedings of CHES'01. Lecture Note in Computer Science, Springer-Verlag,* vol. 2162, pp. 206-223, 2001.

[8] L. Batina, G. Bruin-Muurling, and S. B. Örs, "Flexible Hardware Design for RSA and Elliptic Curve Cryptosystems, " *Proceedings of Topics in Cryptology - CT-RSA 2004. Lecture Note in Computer Science, Springer-Verlag,* vol. 2271, pp. 250-263, 2004.

[9] P. Schaumont and I. Verbauwhede, "Interactive cosimulation with partial evaluation," *Proc. Design Automation and Test in Europe (DATE 2004),* pp. 642-647, 2004.

[10] A. J. Menezes, "Elliptic Curve Public Key Cryptosystems," *Kluwer Academic Publishers,* 1993.

[11] G. Orlando and C. Paar, "A Scalable $GF(p)$ Elliptic Curve Processor Architecture for Programmable Hardware," *Cryptographic Hardware and Embedded Systems: Proceedings of CHES'01. Lecture Note in Computer Science, Springer-Verlag,* vol. 2162, pp. 118-125, 2001.

[12] G. Orlando and C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m)," *Cryptographic Hardware and Embedded Systems: Proceedings of CHES'00. Lecture Note in Computer Science, Springer-Verlag,* vol. 1965, pp. 48-56, 2000.