Available online at www.sciencedirect.com

ScienceDirect

Computers and Electrical Engineering xxx (2007) xxx–xxx

Computers and Electrical Engineering

www.elsevier.com/locate/compeleceng

HW/SW co-design for public-key cryptosystems on the 8051 micro-controller

K. Sakiyama [*], L. Batina, B. Preneel, I. Verbauwhede

Katholieke Universiteit Leuven, Department Electrical Engineering – ESAT/SCD-COSIC Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium

Abstract

It is a challenge to implement large word length public-key algorithms on embedded systems. Examples are smartcards, RF-ID tags and mobile terminals. This paper presents a HW/SW co-design solution for RSA and Elliptic Curve Cryptography (ECC) over GF(p) on a 12 MHz 8-bit 8051 micro-controller. The hardware coprocessor has a Modular Arithmetic Logic Unit (MALU) of which the digit size (d) is variable. It can be adapted to the speed and bandwidth of the micro-controller to which it is connected. The HW/SW co-design space exploration is based on the GEZEL system-level design environment. It allows the designer to find the best performance-area combination for the digit size. As a case study of an FPGA prototyping, 160-bit ECC over GF(p) (ECC-160p) was implemented on Xilinx Virtex-II PRO (XC2VP30). The results show that one point multiplication takes only 130 ms including all communications between the 8051 and the coprocessor. The performance is 40 times faster than the most optimized SW implementation on a small CPU in literature. This is achieved by the HW/SW co-design exploration in order to find the optimized digit size of the MALU. On the other hand, the design of ECC-160p maintains a high level of flexibility by using coprocessor instructions. Our proposed architecture proves that HW/SW co-design provides a high performance close to ASIC solutions with a flexible feature of SW even on a small CPU.

Keywords: HW/SW co-design; RSA; ECC; FPGA implementation; GF(p) operations; GEZEL

1. Introduction

Public-key cryptosystems form an essential building block for digital communication. Unlike private-key algorithms that allow for a fast encryption of a large bulk of data, the importance of Public-Key Cryptography (PKC) is to have secure communications over insecure channels without prior exchange of a secret key. In addition, PKC enables digital signatures as an important cryptographic service. Diffie and Hellman introduced the idea of PKC in the mid 1970s [1].

[*] Corresponding author.
E-mail address: Kazuo.Sakiyama@esat.kuleuven.be (K. Sakiyama).

0045-7906/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2007.05.005

Implementation of Public-Key Cryptography (PKC) is a challenge in embedded systems. Examples are smartcards, RF-ID tags, and mobile terminals. They have limited silicon resources and a limited power budget. Although software implementations are flexible, the performance of PKC is very slow on the small 8-bit or 16-bit CPUs that are often used for power-constrained embedded systems. On the other hand, ASIC implementations of PKC show better performance than software ones. This is mainly due to the following reasons. (1) PKC systems need repeated modular operations with much longer bit integers than CPU's operation bit widths provide (e.g. RSA needs 1024 bits or larger and ECC needs 160 bits or larger) [2–4]. ASICs do not have this bit-width limitation. (2) In an ASIC, the memory organization and the memory access latency can be tuned to the application. Allocating Flip-Flops (F/Fs) or RAMs dedicated to ASICs enable the best use of clock cycles for transferring intermediate data. (3) A dedicated controller in ASIC can handle operations in a datapath without any stalls.

Such an ASIC is the best for mass production considering trade-off between cost and performance. However, a fixed hardware solution has only a limited application range and cannot be easily reused. One approach to make it reusable is to use re-configurable logic such as FPGAs. The main drawback of using FPGAs is that their power consumption is still unacceptable for most small embedded systems. Hence, HW/SW co-designs are attractive since they offer the advantage of both SW flexibility and HW performance by a proper partition of HW/SW. In this paper, a 12 MHz 8051 was chosen as a controller for our proposed HW/SW co-design. Iterative modular multiplications that are the most critical computation components are implemented on the coprocessor part. The originality of our approach is that we do a simultaneous HW and SW optimization. This is facilitated by GEZEL, the design methodology [13]. GEZEL is a hardware description language that can interface with an Instruction Set Simulator (ISS) of a CPU. The combination of GEZEL and ISS makes it possible to simulate the system-level functionality fast, distribute the computations between HW and SW and even optimize the HW coprocessor for a given embedded core. This shortens the design time for both HW and SW.

The remainder of this paper is as follows. Section 2 gives a survey of relevant previous work. In Section 3 the architecture for our proposed coprocessor is explained. The software implementation on the 8051 is explained in Section 4. The performance evaluation is done with a system-level simulation and the results are reported in Section 5. The results of our FPGA implementation are introduced in Section 6, and Section 7 concludes the paper.

2. Related work

The most recent published work is the one of Satoh and Takano [8]. They present a dual field multiplier with a high performance in both types of fields. 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 in scalability in operand size and also flexibility between speed and hardware area. They show that ECC-160p point multiplication can be calculated in 1.21 ms with 0.13-μm CMOS ASIC running at 137.7 MHz. As this is a full-custom ASIC solution, its fast performance cannot be compared directly with our proposed architecture. However, it can be a good index to evaluate the efficiency of the proposed HW/SW co-design. Örs et al. introduce an FPGA implementation of ECC over GF(p) with systolic array type of Montgomery Modular Multiplier (MMM) [9]. Batina et al. optimize the architecture of [9] and obtain 3.9 ms for ECC-160p on an FPGA operating at 53 MHz [10]. Considering software design on an embedded CPU, the only relevant work was reported by Gura et al. [11]. They compared ECC and RSA on 8-bit CPUs and show that PKC is viable on small devices. One of their results shows an ECC-160p point multiplication computed in 4.58 s with a 14.7 MHz 8051-based CPU.

3. Coprocessor architecture

In this section, the proposed coprocessor architecture is described in the details.

3.1. Modular arithmetic logic unit (MALU)

For the datapath of the coprocessor, we designed a configurable Modular Arithmetic Logic Unit (MALU) based on Montgomery's algorithm [5]. The MALU consists of regularly allocated Carry-Save Adders (CSAs).

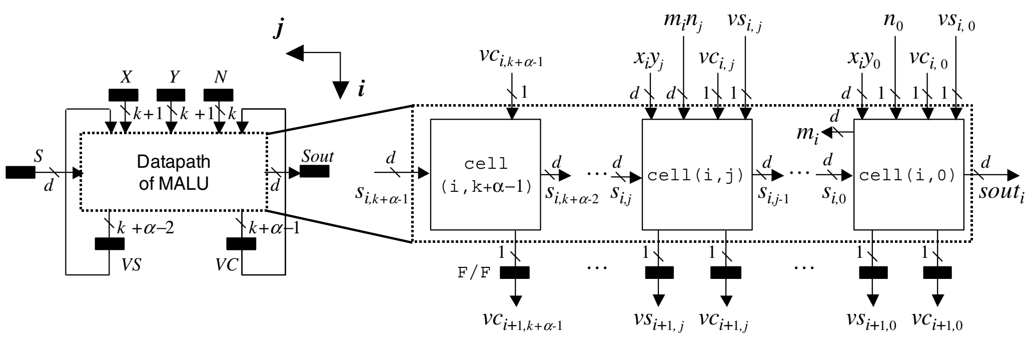


Fig. 1. Schematics of the MALU.

As shown on the left side of Fig. 1, it has four input vectors, $X = (x_g \ldots x_1 x_0)_{2^d}$, $Y = (y_{k+1} \ldots y_1 y_0)_2$, $N = (n_{k-1} \ldots n_1 n_0)_2$, $S = (s_{h,k+\alpha-1} \ldots s_{1,k+\alpha-1} s_{0,k+\alpha-1})_{2^d}$ where $g = \lceil (k+1)/d \rceil$ and $h = \lceil k/d \rceil$. X and Y are the multiplicand and the multiplier, and N is the modulus. The addend vector S is provided to the MALU by d bits in every cycle and eventually added to the result of modular multiplication of X and Y (modulo N). The intermediate results, the so-called virtual sums and carries, are stored in $V_S = (vs_{i,k+\alpha-1} \ldots vs_{i,1} vs_{i,0})_2$ and $V_C = (vc_{i,k+\alpha-1} \ldots vc_{i,1} vc_{i,0})_2$. They are reset to zero when a modular multiplication starts to execute ($i = 0$). After finishing Montgomery multiplication, the multiplication result is output from the right-most cell by d bits in every cycle as $S_{out} = (sout_g \ldots sout_1\ sout_0)_{2^d}$. As will be explained later, V_S and V_C are fed back as inputs in a modular addition. In order to make $sout_i$ zero, the right most cell, `cell(i,0)` determines m_i vector and provides it for the rest of cells.

$$\begin{aligned} \mathrm{MALU}_N(XR, YR, SR) &= (XY + S)R \bmod N \\ \mathrm{C}P_N(V_S R, V_C R, SR) &= (V_S + 2V_C + S)R \bmod N \end{aligned} \tag{1}$$

The proposed array is flexible as for the size of d. Performance and cost trade-offs, and the type of interface to the μ-controller determine the value of d. For a slow μ-controller with a slow interface it is beneficial to take smaller d since the coprocessor can utilize cycles between coprocessor instructions. The MALU has two independent stages. One is the Carry-Save (CS)-stage that implements the Montgomery's algorithm in a CS-form. Another converts the CS-form integer into a normal integer by propagating carries, namely the Carry-Propagate (CP)-stage. Moreover the CP-stage is capable of adding S to the result of the CS-stage. For reducing the hardware cost, the CP calculations are executed in the same cell that is used for the CS-stage. This operation is described by Eq. (1).

Here R is selected as $R^{k+\alpha}$ where k is the bit-length of the secret key and α is a value determined so that the final reductions can be avoided [12]. In this paper, we chose α as $\alpha = 4$. Thus, the reduction step is not required while it is needed in the original notation of Montgomery's algorithm. For the convenience of iterative use of Eq. (1), the so-called Montgomery form is applied to keep the output in the Montgomery form as well. The clock latency for the CP-stage and the MALU (CS-stage + CP-stage) are $\lceil (k+\alpha)/d \rceil$ and $2\lceil (k+\alpha)/d \rceil$ cycles, respectively.

3.2. Coprocessor memory

The MALU operations needs seven sets of $(k+\gamma)$-bit F/Fs that store inputs, outputs, and intermediate variables, where $\gamma \in [0,2]$. The MALU coprocessor will be used repetitively by the 8051. The values used and generated should be stored somewhere in the system. One possible location for them is the memory attached to the 8051 (XRAM). This is the simplest and cheapest solution. Another solution is realized by allocating more sets of F/Fs or SRAMs in the coprocessor itself, called CP-RAM. This is an expensive solution in a cost point of view, especially for large k. But it offers a much higher performance since the overhead time to access the F/Fs in the coprocessor is much less than accessing XRAM.

3.3. Finite state machine (FSM)

The coprocessor is controlled by opcodes sent via the port, P0 of the 8051 (coprocessor instructions). The operands are transferred through P1, P2, and P3. The FSM block in the coprocessor decodes the coprocessor instructions and executes the required operations. A 12 MHz 8051 can issue one CPU instruction only once every 12 clock cycles and the port accesses take several CPU instruction cycles. This results in huge intervals between consecutive coprocessor instructions with a coprocessor running at 12 MHz. Therefore, the FSM should keep the datapath busy computing until the next coprocessor instruction is available. The details are given in Section 5.2. This is another system optimization problem.

4. Software implementation on the 8051

The performance of PKCs is primarily determined by the efficient realization of the arithmetic operations (addition, multiplication and inversion) in the underlying finite field. If projective coordinates are used for an elliptic curve, the inversion can be neglected because it is needed only when converting the projective point back to the affine point at the last step of point multiplication. Therefore, the system architecture for PKCs is normally designed to accelerate the field multiplication and addition. Fig. 2 shows the design hierarchy using the 8051. An 8051 is an 8-bit micro-controller originally designed by Intel that consists of several components: a controller and instruction decoder, an ALU, 128 bytes of internal memory (IRAM), up to 64 Kbytes of external RAM (XRAM) and up to 64 Kbytes of external program memory or 4 Kbytes of internal program memory (PROM). Thus, the targeted PKC has a high flexibility in programming a large variation of public-key operations as well as the high performance by the coprocessor. In the following sections, the operations handled by SW are explained.

4.1. Point multiplication

Point multiplication as shown on top of Fig. 2, can be implemented as a combination of point additions and point doublings. Although other faster algorithms (e.g. sliding window method, windowed NAF, or

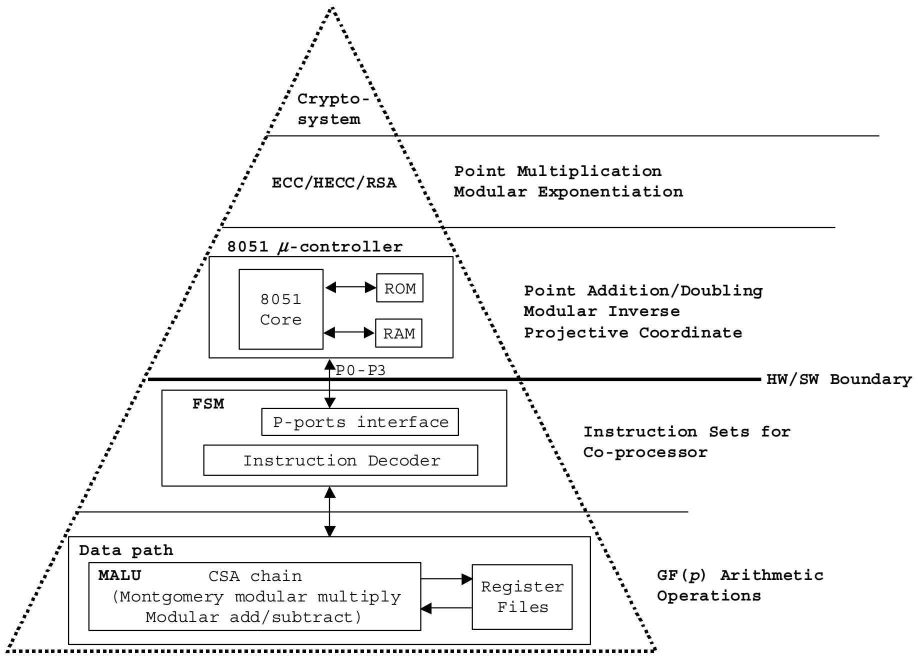


Fig. 2. System architecture with the 8051.

signed m-ary [6]) also can be applied for the proposed architecture, the simplest algorithm, binary-method [7] is implemented in this work.

4.2. Point addition/doubling

Point addition and doubling can be performed according to the algorithm given in [6]. Here we assume that the two points that will be added, i.e. $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ are the points on the weighted projective coordinates (*Jacobian representation*) and Montgomery representation, where (X, Y, Z) corresponds to the affine coordinates $(X/Z^2, Y/Z^3)$. The resultant point is stored as Q, i.e. $Q \Leftarrow P + Q$. The following scheduling as shown in Table 1 can be used for point addition.

Point doubling is considered as a special case of point addition, i.e. $Q \Leftarrow 2Q = Q + Q$. In Table 1, a possible schedule for point doubling is also given. For efficient computing of point addition and doubling, four additional registers (t_1–t_4) storing intermediate variables are provided. These are allocated in CP-RAM.

5. System-level design and simulation

5.1. GEZEL system design environment

The specified architecture was designed and simulated with GEZEL. GEZEL provides cycle-accurate co-simulation with the 8051 ISS. The coprocessor is designed in an FSMD-manner that GEZEL uses for a hardware description. The GEZEL codes are automatically translated into VHDL codes that are synthesizable with existing synthesis tools. GEZEL enables a fast system-level estimation for a target system architecture and quick prototyping on an FPGA.

5.2. Instruction sets for coprocessor

First, the registers in the coprocessor are initialized via ports in unit of 8 bits. Setting CP-RAM for modulo N requires 20 executions of `SETN(din)` instructions for ECC-160p. After initializing all parameters and initial values, MALU$_N$ and CP_N operations are executed according to the sequence of Table 1. The MALU$_N$ operation consists of several instructions. For instance, the second step of point addition in Table 1 starts from `CS(din,din2)` instruction. Here, `din` and `din2` are the addresses of CP-RAM where the values X_2 and t_1 are stored. The coprocessor loads the two operands and executes the CS-stage. The succeeding instructions are `CS( )` and `CP( )` to continue the CS-stage and the CP-stage. The number of `CS( )` and `CP( )` instructions are determined by the value of d. At the last step, `CS(din)` instruction is sent from the 8051 and the coprocessor stores the resultant value of the CP-stage, t_2 at the address of din of CP-RAM.

In our SW case, the 8051 needs four port accesses to issue a coprocessor instruction whose type is `INST(din,din2,din3)` for instance. This is illustrated in Fig. 3. The four port accesses correspond to 96 clock cycles or even more for the coprocessor. Therefore, we need to define a new instruction that is a series of `CS( )` and `CP( )` instructions so that the coprocessor could iteratively execute the CS-stage and the CP-stage. When d is large enough to complete the CS-stage and the CP-stage within 96 clock cycles, we need only one instruction for executing the MALU$_N$ operation. This fact infers that use of a large digit size does not always improve performance in our case because the 8051 cannot dispatch instructions to keep the coprocessor busy computing. On the other hand, in case of using a relatively small digit size, we need to send several instructions. More precisely, when $2\lceil(160 + 4)/d\rceil \geqslant 96$ or $d \leqslant 3$, we need to send two or more instructions for the MALU$_N$ operation in the case of ECC-160p. For the 1024-bit RSA case, $2\lceil(1024 + 4)/d\rceil \geqslant 96$ or $d \leqslant 21$ is derived. Those calculated values of d indicate that the maximum area size of the coprocessor (determined by the digit size) is limited by the communication speed between the 8051 and the coprocessor. In other words, performance improvement cannot be expected even if setting the digit size $d \geqslant 4$ for ECC-160p as shown in Fig. 4.

However, as far as the value of d satisfies $d \leqslant 3$ for ECC-160p or $d \leqslant 21$ for 1024-bit RSA, a coprocessor with a larger d provides better performance. Moreover, those calculated values of d do not guarantee the best performance because of additional communication between the coprocessor and the 8051, i.e. as d becomes

Table 1
Scheduling of point addition and point doubling

Point addition	Cost
$t_1 = \mathrm{MALU}_N(Z_1, Z_1, 0)$	1M
$t_2 = \mathrm{MALU}_N(X_2, t_1, 0)$	1M
$t_3 = \mathrm{MALU}_N(Z_2, Z_2, 0)$	1M
$t_4 = \mathrm{MALU}_N(X_1, t_3, t_2)$	1M
$t_2 = \mathrm{CP}_N(2N+1, t_2^*, 0)$	1A
$t_2 = \mathrm{MALU}_N(X_1, t_3, t_2)$	1M
$t_1 = \mathrm{MALU}_N(t_1, Z_1, 0)$	1M
$t_1 = \mathrm{MALU}_N(t_1, Y_2, 0)$	1M
$Y_2 = \mathrm{MALU}_N(t_3, Z_2, 0)$	1M
$t_3 = \mathrm{MALU}_N(Y_2, Y_1, t_1)$	1M
$t_1 = \mathrm{CP}_N(2N+1, t_1^*, 0)$	1A
$Y_2 = \mathrm{MALU}_N(Y_2, Y_1, t_1)$	1M
$t_1 = \mathrm{MALU}_N(t_2, t_2, 0)$	1M
$t_1 = \mathrm{MALU}_N(t_4, t_1, 0)$	1M
$t_4 = \mathrm{CP}_N(2N+1, t_1^*, 0)$	1A
$X_2 = \mathrm{MALU}_N(Y_2, Y_2, t_4)$	1M
$t_4 = \mathrm{MALU}_N(2, X_2, 0)$	1M
$t_4 = \mathrm{CP}_N(2N+1, t_4^*, 0)$	1A
$t_4 = \mathrm{MALU}_N(1, t_1, t_4)$	1M
$t_1 = \mathrm{MALU}_N(t_2, t_2, 0)$	1M
$t_1 = \mathrm{MALU}_N(t_1, t_2, 0)$	1M
$t_1 = \mathrm{MALU}_N(t_3, t_1, 0)$	1M
$t_1 = \mathrm{CP}_N(2N+1, t_1^*, 0)$	1A
$Y_2 = \mathrm{MALU}_N(t_4/2, Y_2, t_1/2)$	1M
$t_1 = \mathrm{MALU}_N(Z_1, Z_2, 0)$	1M
$Z_2 = \mathrm{MALU}_N(t_2, t_1, 0)$	1M
Total	21M + 5A
$t_1 = \mathrm{MALU}_N(X_2, X_2, 0)$	1M
$t_1 = \mathrm{MALU}_N(2X_2, X_2, t_1)$	1M
$t_2 = \mathrm{MALU}_N(Z_2, Z_2, 0)$	1M
$t_2 = \mathrm{MALU}_N(t_2, t_2, 0)$	1M
$t_2 = \mathrm{MALU}_N(a, t_2, 0)$	1M
$t_1 = \mathrm{MALU}_N(1, t_1, t_2)$	1M
$t_2 = \mathrm{MALU}_N(2Y_2, Y_2, 0)$	1M
$t_3 = \mathrm{MALU}_N(2t_2, t_2, 0)$	1M
$t_2 = \mathrm{MALU}_N(2X_2, t_2, 0)$	1M
$t_4 = \mathrm{MALU}_N(2, t_2, 0)$	1M
$t_4 = \mathrm{CP}_N(2N+1, t_4^*, 0)$	1A
$X_2 = \mathrm{MALU}_N(t_1, t_1, t_4)$	1M
$t_4 = \mathrm{MALU}_N(1, X_2, 0)$	1M
$t_4 = \mathrm{CP}_N(2N+1, t_4^*, 0)$	1A
$t_2 = \mathrm{MALU}_N(1, t_2, t_4)$	1M
$Z_2 = \mathrm{MALU}_N(2Z_2, Y_2, 0)$	1M
$t_3 = \mathrm{CP}_N(2N+1, t_3^*, 0)$	1A
$Y_2 = \mathrm{MALU}_N(t_1, t_2, t_3)$	1M
Total	15M + 3A

t_1^* denotes the bit-inversion of t_1.

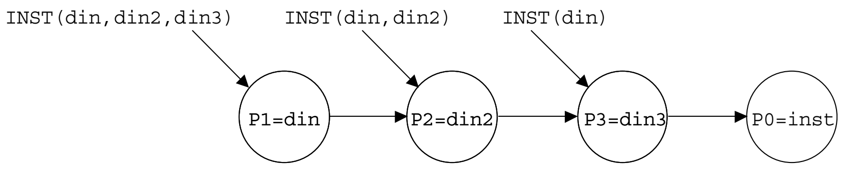


Fig. 3. Port accesses for coprocessor instructions.

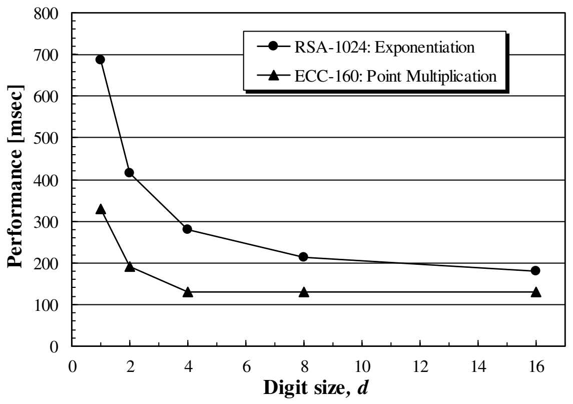


Fig. 4. Performance for point multiplication of ECC-160p and exponentiation of RSA-1024.

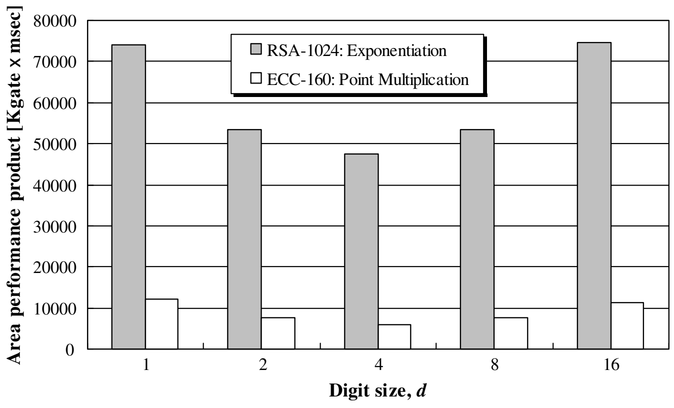


Fig. 5. Estimation of area-performance product for ECC-160p and RSA-1024.

smaller, the coprocessor has to frequently communicate with the 8051 to wait a new instruction by a flag polling or an interrupt which need extra cycles. In the next section, trade-off between cost and performance is discussed by using GEZEL.

5.3. Cost/performance estimation using GEZEL

Before prototyping on an FPGA, the performance for RSA and ECC is estimated using GEZEL system-level co-simulation with the 8051. As illustrated in Fig. 4, ECC-160p shows the same performance for $d = 8$ and 16 as one for $d = 4$. As for RSA-1024, the performance improves as d increases for $d \leqslant 16$. As a result, the best trade-off is obtained in the case of $d = 4$ for both RSA and ECC as shown in Fig. 5. To make a reasonable and quick estimation for the area, the following gate counts are used for the proposed coprocessor: F/F, FA (Full Adder), HA (Half Adder), 2-to-1 MUX (Multiplexer), and 2-bit XOR are counted as 8, 12, 6, 2, and 3, respectively. Here, we use product of area and performance to find the best trade-off between cost and performance.

6. FPGA implementation results of ECC-160p

Based on the performance estimation with GEZEL, we implemented ECC-160p and compared it with the previous work. The software C codes are compiled with μVision2 by Keil Software, Inc. with a target device of Intel 8051AH. The coprocessor block was synthesized with Project Navigator by Xilinx and implemented on

Table 2
Implementation results of ECC-160p on various platform

Reference	Architecture/platform	f_{max} (MHz)	Perform. (ms)		Comments
			@f_{max}	@12 MHz	
[8]	ASIC/0.13-μm CMOS	137.7	1.21	13.9	64-bit multiplier, 117.5 Kgates
		178.6	1.71	25.5	32-bit multiplier, 52.0 Kgates
		253.8	3.33	70.4	16-bit multiplier, 33.2 Kgates
		363.6	7.47	226.3	8-bit multiplier, 28.3 Kgates
[9]	FPGA/Virtex-E	91.3	14.8	112.6	115.5 Kgates[a]
[10]	FPGA	53	3.9	17.2	Based on MMM performance
This work	8051 + coproc./Virtex-II PRO	12(CPU)	129.8	129.8	58.4 Kgates ($d = 4$)[b]
[11]	8051 (Software)	14.7	4580	5628	One instruction cycle is 4

[a] Equivalent gate count reported by the Xilinx tool.
[b] Equivalent gate count is 49.5 Kgates + 6 Block RAMs.

Virtex-II PRO (XC2VP30). The first three references targeting an ASIC solution show better performance at 12 MHz operation than this work except the design of using 8-bit multiplier of [8] as shown in Table 2. Considering the flexibility of this work, the difference of the performance can be regarded as small enough. On the other hand our result is about 40 times faster than an optimized software implementation of [11]. Our architecture proves that HW/SW co-design provides a high performance close to ASIC solutions with a flexible feature of SW.

7. Conclusions

We have presented a MALU coprocessor that is scalable in the digit size d. It allows a fast execution for modular multiplications and additions on CSA chains. As a case study to prove the appropriate HW/SW partitioning, ECC-160p is prototyped on an FPGA. The result shows high performance even with 8-bit 8051. Point multiplication takes only 130 ms including all communications between the 8051 and the coprocessor. This is achieved by the HW/SW co-design exploration with the digit size of the MALU. By changing the coprocessor configuration and adapting SW for a micro-controller, it is ideally suitable for RSA and ECC algorithms.

Acknowledgements

Kazuo Sakiyama and Lejla Batina are funded by FWO projects (G.0450.04, G.0475.05). This research has been also supported by IBBT-QoE and the EU IST FP6 projects SESOC, ECRYPT.

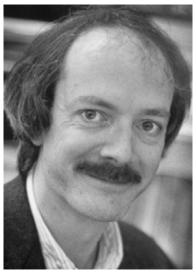
References

[1] Diffie W, Hellman ME. New directions in cryptography. IEEE Trans Inform Theory 1976;22:644–54.
[2] Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Comm ACM 1978;21:120–6.
[3] Koblitz N. Elliptic curve cryptosystems. Math Comput 1987;48:203–9.
[4] Miller VS. Use of elliptic curve in cryptography. Advances in cryptology: proceedings of CRYPTO'85. Lecture note in computer science, vol. 218. Springer-Verlag; 1985, p. 417–26.
[5] Montgomery PL. Modular multiplication without trial division. Math Comput 1985;44:519–21.
[6] Blake I, Seroussi G, Smart N, Elliptic curves in cryptography, Cambridge University Press, London Mathematical Society Lecture Note Series 265, 1999.
[7] Menezes A, van Oorschot P, Vanstone S. Handbook of applied cryptography. CRC Press; 1996.
[8] Satoh A, Takano K. A scalable dual-field elliptic curve cryptographic processor. IEEE Trans Comput 2003;52:449–60.
[9] B Örs S, Batina L, Preneel B, Vandewalle J. Hardware implementation of an elliptic curve processor over GF(p). In: The proceedings of the IEEE 14th international conference on application-specific systems, architectures and processors, 2003. p. 433–43.
[10] Batina L, Bruin-Muurling G, Örs SB. Flexible hardware design for RSA and elliptic curve cryptosystems. Proceedings of topics in cryptology – CT-RSA 2004. Lecture note in computer science, vol. 2271. Springer-Verlag; 2004, p. 250–63.

[11] Gura N, Patel A, Wander A, Eberle H, Shantz SC. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. Cryptographic hardware and embedded systems: proceedings of CHES'04. Lecture note in computer science, vol. 3156. Springer-Verlag; 2004, p. 119–32.
[12] Walter CD. Montgomery exponentiation needs no final subtraction. Electron Lett 1999;35(21):1831–2.
[13] Schaumont P, Verbauwhede I. Interactive cosimulation with partial evaluation. Proc Des Aut Test Eur (DATE 2004) 2004:642–7.

Kazuo Sakiyama obtained the M.Eng. degree from Osaka University, Japan. From 1996 to 2004, he was with Hitachi, Ltd. (now Renesas Technology Corp.). He received the M.Sc. degree from the UCLA in 2003. He is currently working on a Ph.D. program at the K.U. Leuven, Belgium. His research interests include efficient and secure embedded system architectures.

Lejla Batina is a post-doctoral researcher at the COSIC group, Katholieke Universiteit Leuven, Belgium. She received her M.Sc. degree in Mathematics from the University of Zagreb, Croatia in 1995 and Ph.D. degree in engineering from the K.U. Leuven in 2005. Her research interests include efficient arithmetic for cryptographic algorithms, side-channel security, lightweight cryptography, etc.



Bart Preneel received the Electrical Engineering degree and the Doctorate in Applied Sciences from the Katholieke Universiteit Leuven (Belgium). He is currently full Professor at the Katholieke Universiteit Leuven and visiting professor at the T.U. Graz in Austria. He was visiting professor at several universities in Europe. His main research interests are cryptography and information security.

Ingrid Verbauwhede is interested in the implementation aspects of crypto and security applications. It includes circuits, processor architectures, embedded systems and secure design methods. She is currently a professor at the K.U. Leuven and an adjunct associate professor at UCLA. At K.U. Leuven, she is co-director of the COSIC lab.