Optimized System-on-Chip Integration of a Programmable ECC Coprocessor

XU GUO and PATRICK SCHAUMONT
Virginia Tech

Most hardware/software (HW/SW) codesigns of Elliptic Curve Cryptography have focused on the computational aspect of the ECC hardware, and not on the system integration into a System-on-Chip (SoC) architecture. We study the impact of the communication link between CPU and coprocessor hardware for a typical ECC design, and demonstrate that the SoC may become performance-limited due to coprocessor data- and instruction-transfers. A dual strategy is proposed to remove the bottleneck: introduction of control hierarchy as well as local storage. The performance of the ECC coprocessor can be almost independent of the selection of bus protocols. Besides performance, the proposed ECC coprocessor is also optimized for scalability. Using design space exploration of a large number of system configurations of different architectures, our proposed ECC coprocessor architecture enables trade-offs between area, speed, and security.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Design Studies

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Cryptography, elliptic curves, system-on-chip, FPGA

ACM Reference Format:
Guo, X. and Schaumont, P. 2010. Optimized system-on-chip integration of a programmable ECC coprocessor. ACM Trans. Reconfig. Techn. Syst. 4, 1, Article 6 (December 2010), 21 pages. DOI: 10.1145/1857927.1857933. http://doi.acm.org/10.1145/1857927.1857933.

1. INTRODUCTION

Public-key cryptosystems, especially elliptic curve cryptography [Gura et al. 2003; Koblitz 1987; Miller 1986] and recently extensively discussed hyperelliptic curve cryptosystems (HECC) [Koblitz 1990], have become very popular. They have become the preferred public-key cryptosystem for many critical embedded applications. Implementing ECC on an embedded system, including both the hardware and software components, can be a real challenge since

This project was supported in part by the National Science Foundation through grant 0644070.
Authors' address: X. Guo and P. Schaumont, Whittemore Hall 302, Virginia Tech, Blacksburg VA 24061.

one has to deal with ultra-long bit-width data with constrained resources and processing power. A promising approach to deal with this dilemma is the HW/SW codesign, which offers the advantage of flexibility in software with performance in hardware.

The computationally intensive kernel of ECC is well suited for hardware acceleration, so HW/SW codesign is the logic choice to evaluate trade-offs between cost and performance. In the last couple of years, many researchers have used codesign techniques to explore trade-offs between cost, performance, and security in ECC system designs. Typical target platforms include low-end 8 bits platforms (e.g., AVR or 8051) [Aigner et al. 2004; Batina et al. 2005; Gura et al. 2004; Hodjat et al. 2005; Koschuch et al. 2006; Kumar and Paar 2004] as well as 32 bits microprocessors with bus systems (e.g. MicroBlaze with PLB bus). [Orlando and Paar 2000] proposed a scalable elliptic-curve processor architecture which operates over the binary field GF(2^m). [Gura et al. 2003] have introduced a programmable hardware accelerator for ECC over GF(2^m), which can be attached to a 64 bits PCI bus and supports field sizes up to 255. These two papers emphasize the optimization of the coprocessor for speedup and scalability. [Sakiyama et al. 2006] explored architecture optimizations for ECC coprocessors and showed how to exploit parallelism using local control and local coprocessor storage. Their experiments were based on ARM cosimulation. Although each of the above three papers discussed coprocessor implementation results for FPGA, none of them presented a detailed discussion of the system integration effects and the impact of communication bottlenecks when attached to actual processors. [Cheung et al. 2005] implemented a coprocessor with parallel field multiplier attached to OPB bus and identified data transfers over the processor bus as an important system integration problem, but no further optimization for this was mentioned.

Although the design goals in 8 bits platforms and platforms with 32 bits microprocessors and bus systems may differ due to different applications (e.g., low-power sensor nodes versus high performance security systems), both of them have to deal with the same problem of how to minimize the communication overhead resulting from using a single, central controller.

The work described in this article is an extension of [Guo and Schaumont 2009], but with more emphasis on the analysis of architectural scalability and security aspects. Compared with the previous work, this work presents four contributions to codesigns for ECC.

First, we present a complete ECC SoC design and focus on the system integration issues on a real FPGA platform. We use HW/SW cosimulation to do system profiling of the bus bottleneck, and we explore multiple control hierarchies in a typical FPGA based SoC system. We show that, using proper partitioning of control and data, the ECC system execution time can be made almost independent of the selection of bus protocols between the central controller and the coprocessor. Also, we present system performance profiles for an ECC SoC design with MicroBlaze processor and PLB (Processor Local Bus).

Second, we identify two system performance regions. In the first region, the overall system is performance-constrained due to the coprocessor hardware. In the second region, the overall system is communication-constrained due to the

coprocessor-CPU bus. The actual operating point of the system is determined by the choice of coprocessor configurations.

Third, we quantify the impact of control hierarchy and local storage in the coprocessor, and show how the system performance regions are affected. Specifically, we use a small 8 bits microcontroller, PicoBlaze, as a local control unit inside the coprocessor. Moreover, we optimize the control hierarchy by converting a Single-Picoblaze sequencer architecture into a Dual-Picoblaze architecture which runs interleaved instruction sequences. This Dual-PicoBlaze based architecture can achieve the instruction transfer rate of 1 instruction/cycle, while a Single-Picoblaze architecture only provides half that speed, 1 instruction per 2 cycles. The FPGA implementation results show that our proposed ECC SoC architecture with the Dual-PicoBlaze based coprocessor has a better tradeoff between area and speed.

Finally, we optimize the ECC coprocessor to have high flexibility by proposing a novel parallel architecture, which can be used to explore application- and algorithm-level parallelism of ECC. The architecture is scalable and can be adapted to different bus interfaces. Based on our theoretical analysis on its scalability, the parallel ECC SoC design can be used for various high performance applications.

The remainder of this article is as follows. Section 2 gives a brief description of our ECC system configuration and the definition of the ECC design space in our design. In Section 3, the problem statement of HW/SW partitioning will be given. The implementation details will be discussed in Section 4. Section 5 explains the HW/SW codesign flow used in the paper, and performance results of FPGA implementations are analyzed. Section 6 concludes the article.

2. ECC BACKGROUND

Curve-based cryptography, especially ECC, has become very popular in the past several years [Koblitz et al. 2008]. These cryptographic primitives are used for exchanging keys over an insecure channel and for digital signatures. Furthermore, these algorithms show good properties for software and hardware implementations because of the relatively short operand length compared to other public-key schemes, like RSA. However, ECC is still considered as a computationally intensive application due to the complexity of scalar or point multiplications.

2.1 Implementation of ECC over GF(2^m)

Here we present a brief introduction to elliptic curves and more information on elliptic curves can be found in Hankerson et al. [2004]. Let GF(2^m) be a binary field, and let E be an non-supersingular elliptic curve defined with the following equation

$$E: \; y^2 + xy = x^3 + ax^2 + b\,. \tag{1}$$

A basic building block of all elliptic curve cryptosystems is the scalar multiplication, an operation of the form $K \cdot P$ where K is an integer and P is a point on an elliptic curve. A scalar multiplication can be realized through a sequence

ECC Scalar Multiplication (double-and-add-always)	ECC Scalar Multiplication (Montgomery Ladder)
Input: P, $K=\{k_{n-1},..,k_0\}$ **Output**: $Q=K \bullet P$ 1: $Q[0] \leftarrow$ P 2: **for** $i = n\text{-}2$ to 0 **do** 3: $Q[0] \leftarrow 2Q[0]$ $Q[1] \leftarrow Q[0]$ + P $Q[0] \leftarrow Q[k_i]$ **Return** $Q[0]$	**Input**: P, $K=\{k_{n-1},..,k_0\}$ **Output**: $Q = K \bullet P$ 1: $Q[0] \leftarrow$ P, $Q[1] \leftarrow$ 2P; 2: **for** $i = n\text{-}2$ to 0 **do** 3: $Q[1\text{-}k_i] \leftarrow Q[0]+Q[1]$ $Q[k_i] \leftarrow 2Q[k_i]$ **Return** $Q[0]$

Fig. 1. Elliptic curve scalar multiplication (ECSM) algorithms.

Table I. Basic Configuration for Our ECC Design

Coordinate	L-D Projective coordinates [López and Dahab 1999]
Point Mult.	Montgomery Scalar Multiplication [López and Dahab 1999]
Field	GF(2^{163})
Curve	NIST random elliptic curve B-163
GF Mult.	Bit- /Digit-serial multipliers [Großschädl 2001; Kumar et al. 2006]
GF Addition	Logic XOR operations
GF Square	Dedicated hardware with square and reduction circuits [Hankerson et al. 2004]
GF Inversion	GF Multiplications and Squares based on Fermat's Theorem [Rodríguez-Henríquez et al. 2006]

of point additions and doublings (see Figure 1). This operation dominates the execution time of cryptographic schemes based on ECC, such as signatures (ECDSA).

2.2 Basic Configurations

There are many design options for ECC implementations, including the coordinate system, the field and the type of curve [Hankerson et al. 2004]. We used the configuration in Table I for our ECC design. First, we chose a curve with corresponding parameters from the FIPS 186-2 standard [NIST 2000]. Next, we selected a point multiplication algorithm and a coordinate system. Finally, we selected the lowest level finite filed arithmetic algorithms. For simplicity, the discussion in our work is restricted to the smallest field size m=163 specified in the NIST recommended elliptic curves for federal government use. Since this article mainly focuses on the ECC architecture-level optimizations, not the algorithm-level, the current selection of configurations is only for illustration purpose and does not compromise the generality of our proposed architecture.

2.3 ECC Design Space

We consider design space exploration for ECC system architecture at two abstraction levels.

At the application-level of ECSM, we discuss the scalability of our proposed parallel ECC coprocessor architecture. Multiple scalar multiplications can be

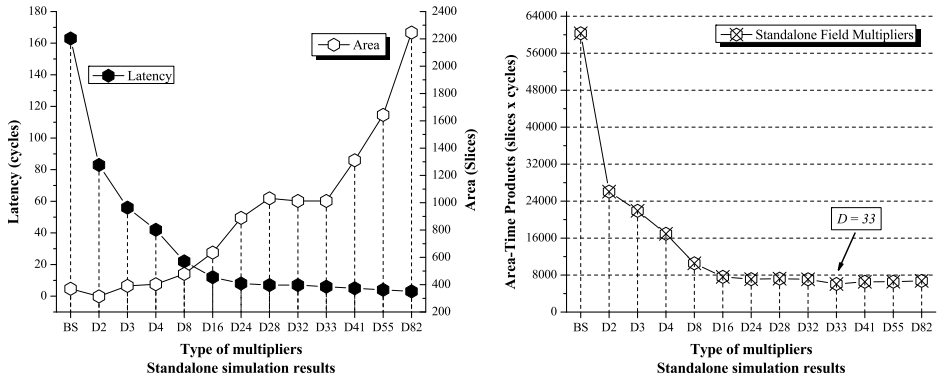

Fig. 2. Area, time and area-time products of different multiplier implementations.

computed simultaneously on one parallel ECC coprocessor, and each of them is executed on independent ECC datapath.

At the algorithm-level of ECSM, we discuss the mapping of existing parallel ECSM algorithms [Järvinen and Skyttä 2008] to our parallel ECC architecture, and one scalar multiplication result can be obtained by summing all the partial ECSM results from parallel datapaths.

Detailed theoretical analysis and FPGA implementation results on design space exploration at the above two levels are provided in Section 4.4 and Section 5.1, respectively.

At the low abstraction level of finite field arithmetic, the design space is defined by the use of different Field Multiplier architectures, including bit-serial as well as digit-serial multipliers of different sizes. A basic bit-serial multiplication in GF(2^m) can be realized through a classic shift-and-XOR based MSB-first bit-serial multiplier with interleaved reduction modulo the irreducible polynomial [Großschädl 2001]. It can finish one GF(2^{163}) multiplication in 163 clock cycles. A digit-serial multiplier on the other hand can process multiple bits of the operands in parallel with a processing time proportional to $\lceil m/D \rceil$ cycles, with digit size $D \leq m - k$, where m is 163 and k is 7 for the B-163 curve. It is obvious that within a certain range of D, when increasing the D, the area will increase accordingly, but the processing time will be the same. For example, for all $D \in [55,81]$, the multiplication time is 3 clock cycles. In this case we only select D size of 55 for our implementations.

Evaluating the multiplication speed and the area-time product for the different architectures leads to the optimum digit size for an implementation on a specific platform. We have evaluated the multipliers required for the NIST B-163 polynomial for different digit sizes to find the optimum values. This leads to Figure 2, which shows the post-place and route results for a Virtex-5 XC5VLX50-1FF676 FPGA with Xilinx ISE10.1 with default settings. For the area metric in Virtex-5 FPGA, we use the unit, *slice*, which is based on the new six-input LUTs. We want to point out that the FPGA implementation results may be affected by different system constraints after place and route and different FPGA platforms, so the results shown in Figure 2 are just for

case study and used as concrete examples to discuss the system integration issues.

Even though a digit-serial multiplier was presented in Kumar et al. [2006], the conclusions of those authors cannot be directly applied here. The differences are due to the use of different technologies (ASICs vs. FPGA) and target application (standalone components vs. system coprocessors). In our case, the system clock frequency is 125MHz, determined by the MicroBlaze system processor. Even though we find that the multipliers can run above 200MHz, we will operate them at the 125MHz system clock. Indeed the coprocessor may run at higher speed, but that would complicate the system implementation by introducing multiple clock regions. Therefore, instead of using absolute execution time and equivalent gate counts, we use cycle counts and total used slices to calculate the area-time products. Then, from Figure 2 we can identify that the best choice, in terms of the area-time product, for a field multiplier in a stand-alone design is a digit-serial multiplier with D size of 33. All the bit-serial and digit-serial multipliers are implemented in logic without using hardware macros because field multiplications on binary fields are mainly composed of shift and XOR operations. Also, the multipliers cannot be pipelined due to the iteration structure and data dependency inside of the bit- and digit-serial multiplier algorithms.

3. CONSIDERATIONS FOR ECC HARDWARE/SOFTWARE PARTITIONING

As shown in Figure 3, a scalar multiplication, $K \cdot P$, with an integer K and a point P on an elliptic curve, needs to be realized through a sequence of point additions and doublings. These group operations can be further decomposed into several types of finite field arithmetic with ultra-long operand word length (e.g., 163 bit above).

Several researchers [Koschuch et al. 2006; Sakiyama et al. 2006] have proposed implementing the field multiplication in hardware and the upper-level point multiplication in software. This typical partitioning is a trade-off between flexibility, cost, and speed, which are the cost factors of most importance in embedded ECC implementations. However, this partitioning may result in a HW/SW communication bottleneck since the lower-level field multiplication function will always be called by upper-level point operations, including many instruction and data transfers. For a baseline ECC system in our design (Scheme A in Figure 3), with all parameters and intermediate results stored in processor local memory, the data/instruction transfer time may take up to 98.3% of the total time required to perform one GF(2^{163}) scalar multiplication when using digit-serial multiplier with D size of 82 and a typical bus communication latency of 9 clock cycles [Guo and Schaumont 2009]. So, the overall ECC system speedup brought by increasing the D size of digit-serial multipliers would be buried if we cannot optimize the HW/SW communication bottleneck.

From the given analysis on the HW/SW partitioning, we already know where the system bottleneck will be, so before starting the system-level design we

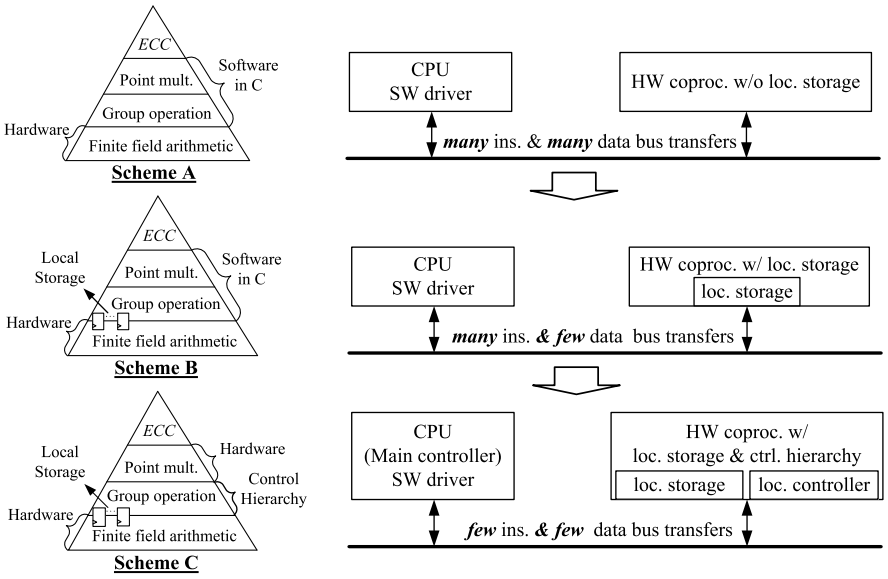

Fig. 3. System architecture modeling of different schemes.

Table II. System Profiling from GEZEL Cosimulation

# Access 163 bit local reg.	Baseline Design bus transactions		Uni-PicoBlaze bus transactions		Dual-PicoBlaze bus transactions	
	# Ins.	# Data	# Ins.	# Data	# Ins.	# Data
2,788	26,791	1,294	481	489	468	476

should first quantitatively measure the bus transactions in the baseline design, and estimate the optimization room left for us. Table II shows the system profiling from GEZEL cosimulation [Schaumont et al. 2006]. Since our system bus interface uses the memory-mapped registers, the way we collect the number of instruction and data transfers over the bus is to measure how many write and read on these registers for instruction or data transfers.

4. ECC SOC DESIGN

4.1 Proposed Optimizations

Targeting the above communication bottleneck problem, we tried to optimize the HW/SW boundary in two steps: reducing data transfers and accelerating instruction transfers.

Apart from architecture-level optimizations (Scheme B and C in Figure 3), we also evaluated the impact of algorithm-level optimizations to further

improve the original Montgomery Scalar Multiplication algorithm [López and Dahab 1999]. These algorithmic optimizations focus on the data and instruction transfers. First, by adding I/O local registers into the coprocessor, data can be transferred between the software and the coprocessor while an ECC field multiplication is in progress. Second, the local registers used to store operands also enable data reuse in the coprocessor. For example, a field multiplication is always followed by a field addition, so the field multiplication results can be directly used as one operand for the following addition. By exploiting the data dependencies in the point multiplication and coordinate conversion operations, we can avoid additional data transfers between the CPU and the coprocessor.

4.2 Impact of Local Storage

Scheme A: using Processor Local Memory as main storage. In a straightforward design, like Scheme A in Figure 3, the ECC system will implement the point operations on the main processor. This requires storing all parameters and intermediate results in Processor Local Memory, so that the main processor can access and manipulate them. This scheme is also called the Baseline ECC SoC System.

From Table II, it is observed that for a full 163 bit scalar multiplication, there are 2,788 times read/write on eight 163 bits coprocessor local registers, so if all parameters and intermediate results are stored in main memory, this may result in 16,728 times data transfers over the 32 bits bus. This represents a significant amount of time (around 60% of the total execution time of a point multiplication, assuming the typical PLB bus HW-SW latency of 9 clock cycles). Hence, a simple optimization can be achieved by adding local storage to the coprocessor, like Scheme B in Figure 3, so that the amount of data transfers over the processor-to-coprocessor bus can be minimized.

Scheme B: using Coprocessor Local Registers as local storage. We optimize the ECC baseline design by adding local storage to the coprocessor, so that the amount of data transfers over the processor-to-coprocessor bus may be reduced. In total, five 163 bits registers were added to the coprocessor. It is also possible to use SRAM blocks instead of registers to save slices.

The comparison of FPGA implementation results between Scheme A and B can be found in Figure 4.

Comparing Scheme A with B, Scheme B provides an average speedup of 2.5 times at the expense of a 1.3 times larger area. The figure also shows that beyond digit-size D = 16, the latency of the overall point multiplication does no longer decrease. Thus, the digit-size D = 16 splits the design space into two. The left half of the figure is a computation constrained area. In that part, the system is constrained by the efficiency of the coprocessor hardware. The right half of the figure is communication constrained. In that part, the system is limited by the processor. For example, if a multiplication in hardware can finish in 9 clock cycles (e.g., 8 clock cycles for digit-serial multiplier of D-size of 24), the speedup of standalone field multiplication brought by D-sizes beyond 24 become invisible. From this point of view, we can conclude that the best

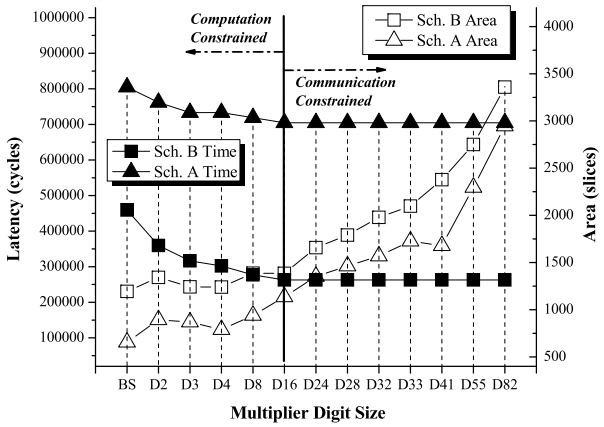

Fig. 4. Time to complete one multiplication and area for each type of coprocessors.

hardware design may not result into the best system solution when system integration overhead is considered.

4.3 Impact of Control Hierarchy

4.3.1 *Embedded Processor Cores.* A wide range of bit widths from 8 to 32 bits processor cores are available to be used as control units in typical SoC designs. Small bit width has the advantage of a small memory footprint for simple applications, but also implies a limited complexity. Wide instructions allow for much more complex applications, but will also require a large amount of memory even for small applications [Hempel and Hochberger 2007].

Various synthesizable processor cores are available for FPGA designs. On one hand we have adaptations of some well known embedded processors like AVR8 and Leon2 processor. These cores are not tailored to the specific resources available in FPGAs. On the other hand there are specialized processor cores for FPGAs, such as Altera NIOS II, Xilinx MicroBlaze and PicoBlaze, and Lattice Mico8. These cores are specifically developed for FPGAs, and they can be classified into two categories: 8 bits cores like the Xilinx PicoBlaze or the LatticeMico8 and 32 bits cores like the NIOS II and the Microblaze. The use of 32 bits cores targets at high performance applications and sometimes requires external memory. Peripherals can be attached to these cores through bus systems. Also, there exist FPGAs that have processor cores implemented as hard IP blocks (e.g., PowerPC embedded in Xilinx FPGA).

Compared with the 32 bits cores, the 8 bits cores have very simple I/O interfaces and are very limited in computation power. The on-chip program memory is always very small, typically less than 1K instruction store with simplified instruction set.

As we have indicated, either type of core has its own advantages, which may be complementary to each other in one SoC design. However, most current research only considers them as separate control units and rarely combines them in a single system-level design.

4.3.2 *Single-PicoBlaze Based ECC SoC Design.*

Scheme C: using PicoBlaze (PB) as control hierarchy. From the above analysis of Scheme A and B, these two schemes mitigate the overall bus communication overhead by optimizing the data transfer side; however, instruction transfers still dominate the entire scalar multiplication time. From the cosimulation profiling (see Table II), we can see that for a full 163 bits scalar multiplication, there are 26,791 times instruction transfers though the data transfers have been reduced to 1,294 (mostly composed of reading status registers) with a typical PLB bus communication latency of 9 clock cycles. One area for further optimization is that of coprocessor control. Indeed, for each operation performed by the coprocessor, the processor needs to perform a command transfer over the PLB bus. These command transfers are still needed, even after local registers are added to the coprocessor. In order to reduce the amount of command transfers, we must change the way to control the coprocessor.

The PicoBlaze microcontroller is a compact, capable and cost-effective fully embedded 8 bits RISC microcontroller core optimized for Xilinx FPGAs. It has predictable performance, always two clock cycles per instruction, and 1K instructions of programmable on-chip program store, automatically loaded during FPGA configuration. It only costs 53 slices and 1 block RAM on Virtex-5 XC5VLX50 FPGA and can run at the max frequency of 180MHz.

By introducing a local control hierarchy, the PicoBlaze takes the charge of sending out the point addition and doubling instructions. The main controller, MicroBlaze, only needs to start a scalar multiplication once, after which the detailed sequencing will be completed by the PicoBlaze (like Scheme C in Figure 3). As shown in Figure 5, the coprocessor has two separate FSMs to decode the instructions sent from the CPU and PicoBlaze, respectively. The CPU instruction set only contains instructions for data transfers, while the PicoBlaze instruction set includes all the instructions for controlling the finite field arithmetic ALUs and exchanging data with coprocessor local register array.

The PicoBlaze has additional advantage of having a fixed instruction rate (2 clock cycles per operation). This means that the local instruction decoder in the coprocessor can be simplified: no additional synchronization is needed between the PicoBlaze and the local instruction decoder FSM. From the cosimulation system profiling (see Table II), we can see that for the Single-PicoBlaze design the instruction and data transfers have been greatly reduced to 481 and 489, respectively. The lower-bound on the amount of instruction and data transfers is around 18 and 6, respectively. Most of the current measured instruction and data transfers are devoted to polling the status registers with associated instructions during the coprocessor execution time. Therefore, a simple further optimization can be conducted by using 'interrupt' control, and this improvement may not only reduce the number of bus transactions but also save the task load of ECC on MicroBlaze and PLB bus, which then may be used for other peripherals in a large system.

The results for Scheme C with a Single-PicoBlaze local controller will be discussed in Section 5.2 and compared with other design strategies. We

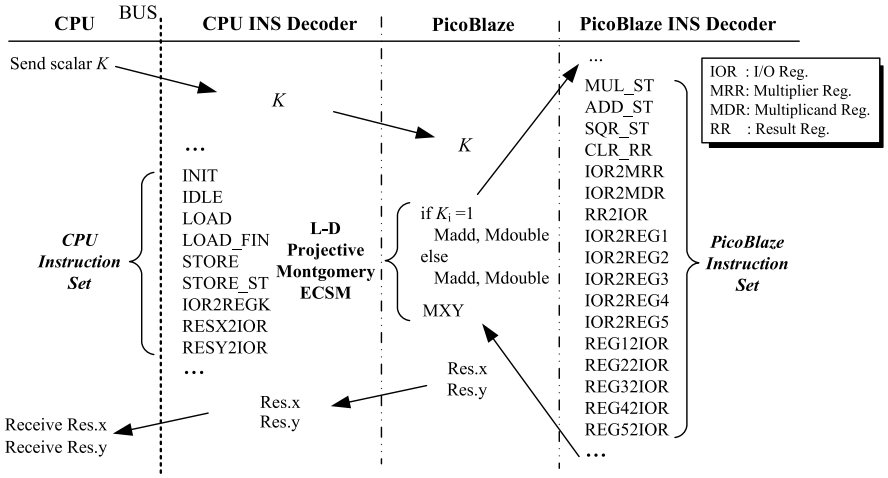

Fig. 5. Dataflow of ECSM with CPU and PicoBlaze instruction sets.

have demonstrated that by introducing the Picoblaze as control hierarchy the communication constrained area disappears as we can observe a continuous speedup when the coprocessor uses a faster multiplier, from a bit-serial multiplier to a digit-serial multiplier of D-size 82.

4.3.3 *Dual-PicoBlaze Based ECC SoC Design.* After analyzing the previous design we find two characteristics of Single-PicoBlaze based ECC coprocessor design, which can help us to further refine the design. First, the local control unit, PicoBlaze, in the current design is acting as an instruction generator. It has no data processing inside, which means it is feasible to split the Picoblaze program into several subsets as long as the sequence of the instructions is guaranteed. Second, the instruction rate of Single-PicoBlaze is fixed 2 clock cycles per operation, which means that there is still one cycle per operation wasted in the ECC datapath. So, from the above two observations we propose the idea of optimizing the local control unit by converting a Single-Picoblaze sequencer architecture into a Dual-Picoblaze architecture which runs interleaved instruction sequences. Hence, this novel Dual-PicoBlaze based architecture can achieve the maximum instruction transfer rate of 1 instruction/cycle. To illustrate the conversion from Single-PicoBlaze to the Dual-PicoBlaze, a simple example of PicoBlaze assembly codes executing one field multiplication followed by an addition is shown in Figure 6. Compared with the Single-PicoBlaze design our proposed Dual-PicoBlaze design can save additional 18% of total clock cycles in average, at the expense of very small hardware overhead (81 slices in average from FPGA implementation results on XC5VLX50). Detailed comparison between cosimulation and FPGA implementation results are presented in Section 5.2.

Traditional ways can also achieve the maximum instruction rate of 1 instruction/cycle, such as implementing the point operations in FSM or using a microcoded controller with preset microcodes [Sakiyama et al. 2006].

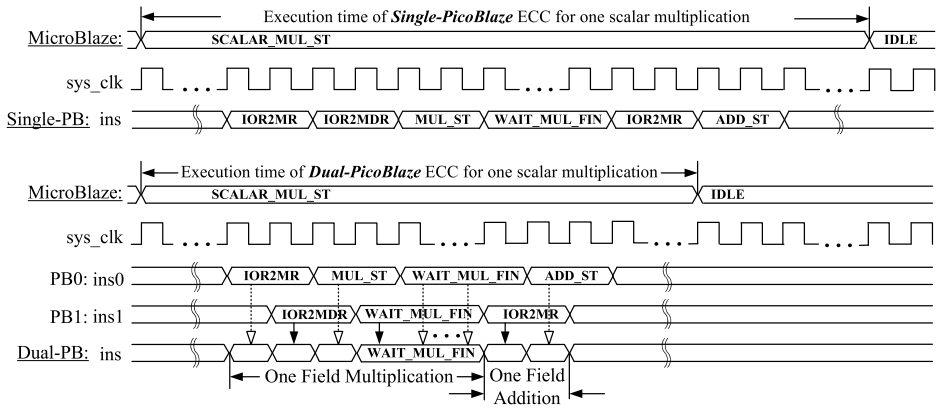

Fig. 6. An example of interleaving PicoBlaze instructions.

Compared with the first approach using FSMs, our Dual-PicoBlaze architecture is more flexible and efficient. In general, the field operations can be very fast (a digit-serial multiplier with D size of 82 can finish one 163 bit field multiplication in 2 clock cycles) and a big performance gain of the whole underlying ECC system can only be obtained if new point operation algorithms with faster point operations are proposed. In this case, by fixing the lowest level field operations in hardware, updating an ECC system is just replacing the software assembly codes in PicoBlaze with the new point operation algorithms without the need to rewrite the HDLs. In Hankerson et al. [2004], more than 10 point multiplication algorithms are compared and the research in this direction is still very active. In addition, this method can also enable the integration of the latest countermeasures against side-channel attacks into the algorithm for scalar multiplication. In Guo et al. [2009], we have already shown a feasible way to resist most existing passive and active attacks on ECC by using a collection of algorithm-level countermeasures based on this programmable architecture.

Compared with the second approach using microcoded controller, the Dual-PicoBlaze architecture is much easier to be programmed. The microcoded controller needs sometimes complex dedicated controller with FSMs to dispatch instructions. Based on the Dual-PicoBlaze architecture we can simply use several PicoBlaze instructions to achieve efficient communication and synchronization with the hardware decoder without additional logic.

4.4 Scalability

For scalability, we mainly discuss about the parallelism inside of our proposed ECC coprocessor architecture. Two levels of parallelism can be explored so that two operation modes have been defined:

1. Application-Level Parallel ECSM. Different parallel ECC datapaths can calculate different $K \cdot P$ and return separate scalar multiplication results.

2. Algorithm-Level Parallel ECSM. The scalar is firstly split into several parts using the fixed window scalar splitting algorithm [Järvinen and Skyttä

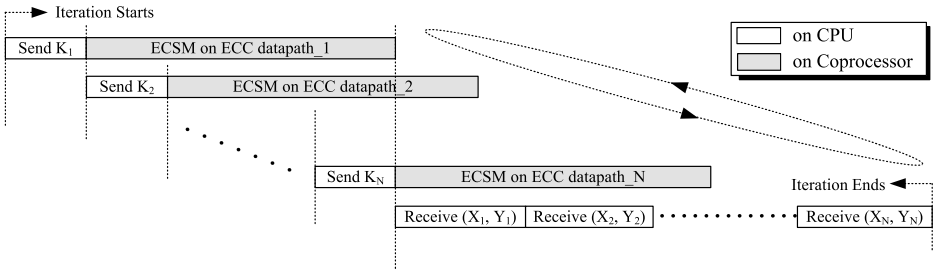

Fig. 7. Exploration of the application-level parallelism within the proposed generic coprocessor architecture.

2008]; then different parallel ECC datapaths calculate the partial scalar multiplication with final point additions to sum all the partial scalar multiplication results from each ECC datapath.

Given the flexibility of the PicoBlaze local control hierarchy, switching the computation mode from Application-Level Parallel ECSM to Algorithm-Level Parallel ECSM, we can just modify the PicoBlaze assembly codes in each datapath without any hardware change.

Application-Level Parallel ECSM. Our proposed ECC coprocessor architecture is scalable for parallel implementations because of three design considerations: 1. distributed data processing makes each ECC datapath be independent to each other; 2. local storage makes all the initialization parameters and intermediate results be stored locally without data transfers through bus; 3. additional hierarchy of control makes point operation instructions be sent from the local control, instruction sequencer. In summary, the ECC datapath can execute scalar multiplication almost independent of the bus selections and CPU, and once the CPU send the scalar K to each ECC datapath to initialize the computation, the datapath will work automatically and turn out the right results. The maximum number of independent ECC datapath which can be attached to the CPU instruction decoder is dependent on the bus latency. Therefore, CPU can control one ECC coprocessor with N datapaths, and N point multiplications can be performed at the same time.

According to the iteration structure shown in Figure 7, we can derive an equation to express the relation between the maximum number of parallel ECC datapaths and bus latency. The basic idea is to overlap the communication time with the computation time. We assume the bus latency is T_{delay} cycles per transfer, and scalar K and results (X, Y) each needs the same M times bus transfers (including both instruction and data transfers), and the ECSM on one ECC datapath requires T_{comp} cycles to complete, so the effective maximum number, N_{max}, of parallel ECC datapath can be expressed as

$$N_{max} = (T_{comp}/MT_{delay}) + 1. \tag{2}$$

From Figure 7, we can observe that the results from the first datapath are ready to be sent back just after the datapath_N_{max} receives the K. Due to

this parallel architecture, we can get the fastest implementation with T_{avg_min} cycles, where

$$T_{avg\_min} = 3MT_{delay}. \tag{3}$$

From Equation (3), we can observe that for the fastest ECC coprocessor configuration with maximum number of parallel ECC datapaths, the minimum computation time in average is only related to the bus latency. Also, we can have trade-off designs between area and speed with different number of parallel ECC datapaths to fit for different embedded applications, and then we can get the computation time in average, T_{avg}, as

$$T_{avg} = \frac{(2N+1)MT_{delay} + T_{comp}}{N}. \tag{4}$$

Algorithm-Level Parallel ECSM. If we assume that a fixed polynomial is used with fixed base point, the algorithm-level parallelism inside of one scalar multiplication can also be explored. We can apply a fixed window scalar splitting algorithm [Järvinen and Skyttä 2008] based on our parallel ECC architecture with multiple datapaths.

If the scalar K has m bits, it can be split into N blocks using predefined windows with a size of w. Here N can be the number of parallel datapath and w is equal to $\lceil m/N \rceil$. Then, the scalar K is decomposed into several parts with $K1$ consists of the w least significant bits (LSB) of K, $K2$ contains the next w bits, etc. The base point Pj for each segment of K can be pre-computed because the window sizes are fixed,

$$Pj = 2^{(j-1)w} P. \tag{5}$$

The overhead of precomputing Pj can be neglected because the base point P is assumed to be fixed in most ECC applications. After the computation of each parallel ECC datapath for $Kj \cdot Pj$, point additions will be followed to obtain the final results of $K \cdot P$. As a case study of two datapaths shown in Figure 8, the fixed window scalar splitting algorithm is used to split the scalar K into two parts, $K1$ and $K2$, and compute them on two datapaths in parallel with precomputed base points, $P1$ and $P2$. Only one final point addition is performed to calculate the sum of the two intermediate scalar multiplication results, $K1 \cdot P1$ and $K2 \cdot P2$. We can also clearly see that when switching the mode from Application-Level Parallel ECSM, we may just add PicoBlaze instructions for one point addition in Affine Coordinate on the first datapath and two instructions on the second datapath for moving the partial ECSM results to the first datapath.

5. DESIGN FLOW AND IMPLEMENTATION

5.1 FPGA Implementation

Using the GEZEL cosimulation environment we can translate the GEZEL description of the ECC datapath and control wrappers into synthesizable VHDL, which can be then added as coprocessors in the Xilinx Platform Studio (XPS) 10.1.

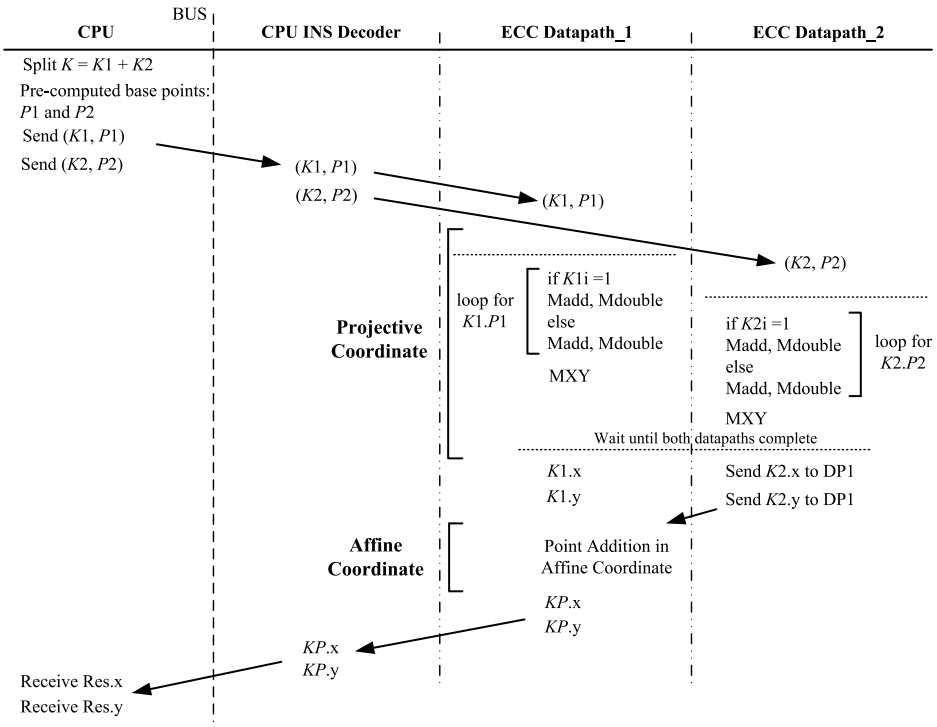

Fig. 8. Parallel scalar multiplication with fixed window scalar splitting method.

The SoC system shown in Figure 9 is built on the Xilinx Virtex-5 XC5VLX50 ML501 development board. A hardware timer is added for measuring the cycle counts for each design configuration.

Based on the discussion in Section 4.4, the timing profiling of parallel implementations of multiple ECC datapaths based on Processor Local Bus (PLB) on Xilinx FPGA platform can be found in Table III.

The Processor Local Bus (PLB) interface is a memory-mapped interface for peripheral components with typical bus transfer latency, T_{delay}, of 9 clock cycles. For the above timing profiling, we just consider the ideal case for data/instruction transfers without any software overhead (e.g., function calls), so the estimated value is from the theoretical point of view. For instance, to finish one 163 bit scalar multiplication and transfer results back to CPU, in the minimum average time, T_{avg_min}, of 540 clock cycles is very appealing; however, the maximum parallel implementation of 139 ECC datapaths with digit-serial multiplier of $D82$ is impractical. Still, we can find reasonable trade-offs through Equation (4).

5.2 Discussion of Experimental Results

To make fair comparisons with other designs we only use the results of ECC coprocessor design with single datapath. From the deterministic and cycle-accurate GEZEL cosimulation, we can obtain both of the stand-alone hardware

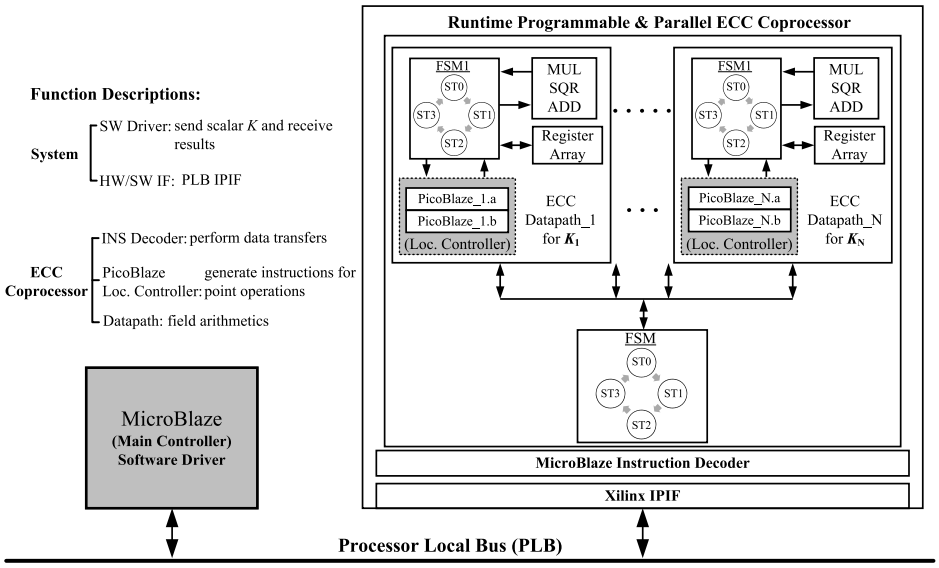

Fig. 9. The structure of our proposed parallel ECC coprocessor.

Table III. Timin Profiling of Parallel Implementations of the Proposed ECC Coprocessor

	T_{delay}	M	T_{comp} $w/D82$	N_{max} $w/D82$	T_{avg_min}
$MicroBlaze w/PLB$	9	20	24,698	139	540

design profiles and the system profiles (e.g., cycle counts and toggle counts). This is very helpful for system designers to get performance evaluation at a very early design stage and accelerate their design space exploration.

As indicated in Figure 10, for the Scheme B with only local storage, the two systems are limited by the throughput of the PLB bus. For example, for the FPGA implementation with the PLB bus latency of 9 clock cycles, if a field multiplication in hardware can finish in 9 clock cycles (e.g., 8 clock cycles for digit-serial multiplier of D size of 24), the speedup of standalone field multiplication brought by D-sizes beyond 24 become invisible. For the results for Scheme C with a Single-PicoBlaze local controller, we can observe a continuous speedup when the coprocessor uses a faster multiplier, from a bit-serial multiplier to a digit-serial multiplier of D-size 82. Compared to the results of Scheme B, the communication bottleneck has disappeared as we can observe a speedup of 1.2 for the coprocessor with D-size 82 over the one with D-size of 16. Since the control hierarchy optimization by introducing PicoBlaze also features small hardware overhead, from Figure 11 we can see our proposed Dual-PicoBlaze based design can achieve the best trade-off design with D size of 28.

In order to make a fair comparison with other published results, we also synthesize our ECC coprocessor design with single datapath based on Virtex-2 Pro XC2VP30 FPGA. As shown in Table IV, our Dual-PicoBlaze based ECC (with maximum frequency around 136MHz on XC2VP30-FF896-7C) shows a better

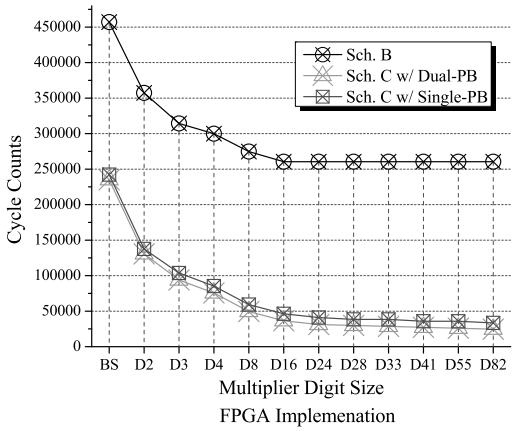

Fig. 10. Cycle counts of FPGA implementations of each configuration of coprocessors for one full scalar multiplication.

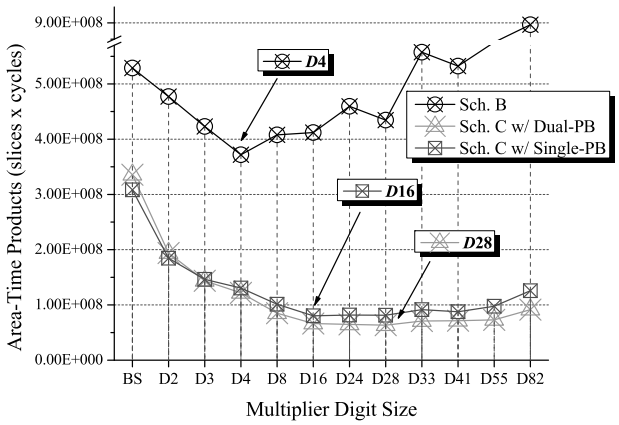

Fig. 11. Comparison of time-area products for each configuration of coprocessors.

trade-off between cost and performance: compare our fastest design with ref. 2, it gains 13.4% speedup; compare our best trade-off design with ref.1 and ref.2, its area-time product is 60.4% and 63.7% smaller. Although the current version of Dual-PicoBlaze design does not support arbitrary field size and superscalar as Sakiyama et al. [2006], it still offers an ideal alternative since in most cases the arbitrary field size is not required. This is especially true for reconfigurable computing since the hardware complexity resulted from supporting arbitrary field size in traditional ASICs can be replaced with multiple configuration bitstreams and dynamic reconfigurations. The optimizations of ECC SoC design can be done in several levels (e.g., architecture, algorithm, and circuit) and the results shown here might not be the optimal ones even in terms of the area-time product since the performance optimization focus in this paper only lies on the architectural-level.

Table IV. Comparison of ECC Coprocessor Implementations on Xilinx XC2VP30 FPGA

	Field	Platform	Slices	Cycle Counts	Field Size	Comments
Dual-PB w/ D28	$GF(2^{163})$	*V2Pro*	5, 158	29, 897	Fixed	Best trade-off
Dual-PB w/ D82	$GF(2^{163})$	*V2Pro*	8, 944	24, 689	Fixed	Fastest
ref.1 in [Sakiyama et al. 2006]	$GF(2^{163})$	*V2Pro*	4, 749	48, 800	Arbitrary	1xMALU163
ref.2 in [Sakiyama et al. 2006]	$GF(2^{163})$	*V2Pro*	8, 450	28, 000	Arbitrary	2xMALU163

Table V. Parallel ECC Coprocessor Implementations on Xilinx XC5VLX50 FPGA

	Field	Slices	Cycle Counts	Field Size	Modes
Dual-PB Single-DP w/ BS	$GF(2^{163})$	1, 261	234, 985	Fixed	Normal Operation
Dual-PB Single-DP w/ D82	$GF(2^{163})$	3, 522	24, 689	Fixed	Normal Operation
Quad-PB Dual-DP w/ BS	$GF(2^{163})$	2, 179	117, 972	Fixed	Application-Level Parallel
Quad-PB Dual-DP w/ D82	$GF(2^{163})$	6, 585	12, 824	Fixed	Application-Level Parallel
Quad-PB Dual-DP w/ BS	$GF(2^{163})$	2, 179	204, 390	Fixed	Algorithm-Level Parallel
Quad-PB Dual-DP w/ D82	$GF(2^{163})$	6, 585	19, 461	Fixed	Algorithm-Level Parallel

In order to show the scalability of our architecture, we also compare the design with Dual-PicoBlaze Single-Datapath ECC design with the Quad-PicoBlaze Dual-Datapath ECC designs under different operation modes. We select two extreme coprocessor configurations, the smallest (with bit-serial multiplier, BSMUL) and the fastest (with D82), for detailed comparison. For the results shown in Table V, the cycle counts for Application-Level Parallel ECSM mode are the average speed for two ECSMs in parallel. For the results of algorithm-level Parallel ECSM, the speedup over the Dual-PB Single-DP based design is not as good as in [Järvinen and Skyttä 2008] because different finite field inversion algorithms are used (Fermat's little theorem vs. Itoh-Tsujii [Itoh and Tsujii 1988]). However, these experimental results still effectively demonstrate the capability of our proposed architecture to switch operation modes by just revising the PicoBlaze assembly codes.

Besides performance, cost efficiency and flexibility, using PicoBlaze as control hierarchy can also enhance the side-channel attack resistance. The use of L-D Montgomery scalar multiplication is already useful as a countermeasure since it performs exactly one Madd and one Mdouble operation for each bit of the scalar K. Consequently, the total number of Madd/Mdouble operations depends only on the bit length of K, but not on its Hamming weight. This property helps to prevent certain side-channel attacks like simple power analysis (SPA) attacks and timing attacks [Coron 1999]. By using PicoBlaze to implement point operation we can add dummy instructions or field computations with predictable timings to further adjust the balance of the power or timing of either

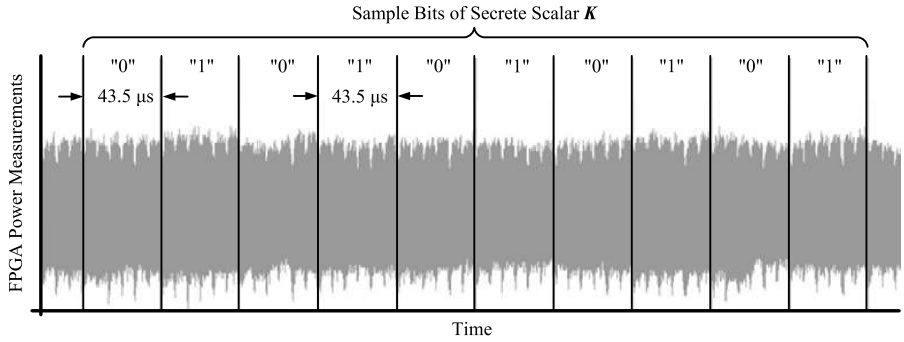

Fig. 12. SPA attacks on FPGA implementation of our proposed ECC coprocessor.

branch of Madd/Mdouble operations. Figure 12 shows the real power measurement on the implementation of our proposed Dual-PicoBlaze based ECC coprocessor with single datapath and bit-serial multiplier on the Xilinx Virtex2Pro XC2VP7 FPGA based on a SASEBO-G board. From the power traces, it is observed that it is impossible to distinguish the 1 bit from 0 bit of scalar K, which can prove that our design also shows good SPA attack resistance. Since this ECC coprocessor is programmable, new algorithm-level countermeasures can be uploaded to the coprocessor without hardware modifications. In Guo et al. [2009], we have already demonstrated that based on the programmable and parallel ECC coprocessor architecture described in this work we can resist a number of passive and active attacks by just changing the PicoBlaze assembly codes with novel algorithm-level countermeasures.

6. CONCLUSIONS

ECC SoC designs may become performance limited due to coprocessor data- and instruction-transfer bottleneck. The introduction of local storage and control hierarchy into the ECC coprocessor datapath can greatly reduce the communication overhead faced by traditional centralized control scheme. Starting from the system profiling of ECC codesigns using cosimulation, we tried to not repeat the conventional optimization techniques on bus communication, but instead explore new system architectures with multiple control hierarchies. This results in the Single-PicoBlaze based ECC coprocessor design and further into the Dual-PicoBlaze based design with the maximum instruction rate of 1 instruction/cycle. For flexibility, the PicoBlaze controller allows us to configure its instruction RAM and update the coprocessor with the newly developed scalar multiplication algorithms and security countermeasures. Scalable application-level parallelism and algorithm-level parallelism can also be explored to achieve tradeoff designs between area and speed. With flexibility, ease of integration of multiple PicoBlazes into current FPGA systems and predictable performance, the proposed parallel ECC coprocessor architecture can not only be extended to other curve-based cryptography systems, but also to some other similar computationally intensive embedded applications.

ACKNOWLEDGMENTS

The authors would like to thank the Xilinx University Program and National Institute of Advanced Industrial Science and Technology (AIST) of Japan for their hardware support.

REFERENCES

AIGNER, H., BOCK, H., HTTER, M., AND WOLKERSTORFER, J. 2004. A low-cost ecc coprocessor for smartcards. In *Proceedings of the Conference on Cryptographic Hardware and Embedded Systems*. Springer, Berlin, 107–118.

BATINA, L., HWANG, D., HODJAT, A., PRENEEL, B., AND VERBAUWHEDE, I. 2005. Hardware/software co-design for hyperelliptic curve cryptography (hecc) on the 8051 p. In *Proceedings of the Conference on Cryptographic Hardware and Embedded Systems*. Springer, Berlin, 106–118.

CHEUNG, R. C. C., LUK, W., AND CHEUNG, P. Y. K. 2005. Reconfigurable elliptic curve cryptosystems on a chip. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'05)*. IEEE Computer Society, Los Alamitos, CA, 24–29.

CORON, J.-S. 1999. Resistance against differential power analysis for elliptic curve cryptosystems. In *Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*. Springer-Verlag, 292–302.

GROßSCHÄDL, J. 2001. A low-power bit-serial multiplier for finite fields gf(2^m). In *Proceedings of the 34th IEEE International Symposium on Circuits and Systems*. IEEE, 37–40.

GUO, X. AND SCHAUMONT, P. 2009. Optimizing the control hierarchy of an ecc coprocessor design on an fpga based soc platform. In *Proceedings of 5th International Workshop on Applied Reconfigurable Computing*. Springer-Verlag, Berlin, 169–180.

GUO, X., FAN, J., SCHAUMONT, P., AND VERBAUWHEDE, I. 2009. Programmable and parallel ecc coprocessor architecture: Tradeoffs between area, speed and security. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES'09)*. Springer, 289–303.

GURA, N., SHANTZ, S. C., EBERLE, H., GUPTA, S., GUPTA, V., FINCHELSTEIN, D., GOUPY, E., AND STEBILA, D. 2003. An end-to-end systems approach to elliptic curve cryptography. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'02)*. Springer-Verlag, 349–365.

GURA, N., PATEL, A., WANDER, A., EBERLE, H., AND SHANTZ, S. C. 2004. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *Proceedings of the Conference on Cryptographic Hardware and Embedded Systems*. Springer, Berlin, 925–943.

HANKERSON, D., MENEZES, A., AND VANSTONE, S. 2004. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, Berlin.

HEMPEL, G. AND HOCHBERGER, C. 2007. A resource optimized processor core for fpga based socs. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD'07)*. IEEE Computer Society, Los Alamitos, CA, 51–58.

HODJAT, A., HWANG, D., BATINA, L., AND VERBAUWHEDE, I. 2005. A hyperelliptic curve crypto coprocessor for an 8051 microcontroller. In *Proceedings of the 19th IEEE Workshop on Signal Processing Systems*. IEEE, 93–98.

ITOH, T. AND TSUJII, S. 1988. A fast algorithm for computing multiplicative inverses in gf(2^m) using normal bases. In *Information and Computation*. Academic Press, Inc., 171–177.

JÄRVINEN, K. AND SKYTTÄ, J. 2008. On parallelization of high-speed processors for elliptic curve cryptography. *IEEE Trans. VLSI Syst*. 1162–1175.

KOBLITZ, A. H., KOBLITZ, N., AND MENEZES, A. 2008. Elliptic curve cryptography: The serpentine course of a paradigm shift. http://eprint.iacr.org/2008/390.

KOBLITZ, N. 1987. Elliptic curve cryptosystems. Mathematics of computation. *Math. Comput. 48,* 177, 203–209.

KOBLITZ, N. 1990. A family of jacobians suitable for discrete log cryptosystems. In *Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'88)*. Springer-Verlag, 94–99.

KOSCHUCH, M., LECHNER, J., WEITZER, A., GROßSCHÄDL, J., SZEKELY, A., TILLICH, S., AND WOLKERSTORFER, J. 2006. Hardware/software co-design of elliptic curve cryptography on an 8051 microcontroller. In *Proceedings of the Conference on Cryptographic Hardware and Embedded Systems*. Springer, Berlin, 430–444.

KUMAR, S. AND PAAR, C. 2004. Reconfigurable instruction set extension for enabling ecc on an 8-bit processor. In *Proceedings of the Conference on Field Programmable Logic and Application*. Springer, Berlin, 586–585.

KUMAR, S., WOLLINGER, T., AND PAAR, C. 2006. Optimum digit serial gf(2^m) multipliers for curve-based cryptography. *IEEE Trans. Comput. 55,* 10, 1306–1311.

LÓPEZ, J. AND DAHAB, R. 1999. Fast multiplication on elliptic curves over gf(2^m) without precomputation. In *Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*. Springer-Verlag, 316–327.

MILLER, V. S. 1986. Use of elliptic curves in cryptography. In *CRYPTO'85: Advances in Cryptology*. Springer-Verlag, 417–426.

NIST. 2000. Digital signature standard. FIPS PUB 186-2 Federal Information Processing Standard. NIST.

ORLANDO, G. AND PAAR, C. 2000. A high performance reconfigurable elliptic curve processor for gf(2^m). In *Proceedings of the 2nd International Workshop on Cryptographic Hardware and Embedded Systems (CHES'00)*. Springer-Verlag, 41–56.

RODRÍGUEZ-HENRÍQUEZ, F., SAQIB, N. A., DÍAZ-PÈREZ, A., AND KOC, C. K. 2006. *Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology)*. Springer-Verlag.

SAKIYAMA, K., BATINA, L., PRENEEL, B., AND VERBAUWHEDE, I. 2006. Superscalar coprocessor for high-speed curve-based cryptography. In *Proceedings of the Conference on Cryptographic Hardware and Embedded Systems*. Springer, Berlin, 415–429.

SCHAUMONT, P., CHING, D., AND VERBAUWHEDE, I. 2006. An interactive codesign environment for domain-specific coprocessors. *ACM Trans. Des. Autom. Electron. Syst. 11,* 1, 70–87.

Received April 2009; revised November 2009; accepted January 2010