Optimizing the Control Hierarchy of an ECC Coprocessor Design on an FPGA Based SoC Platform

Xu Guo and Patrick Schaumont

Virginia Tech, Blacksburg VA 24061, USA
{xuguo,schaum}@vt.edu

Abstract. Most hardware/software codesigns of Elliptic Curve Cryptography only have one central control unit, typically a 32 bit or 8 bit processor core. With the ability of integrating several soft processor cores into one FPGA fabric, we can have a hierarchy of controllers in one SoC design. Compared to the previous codesigns trying to optimize the communication overhead between the central control unit and coprocessor over bus by using different bus protocols (e.g. OPB, PLB and FSL) or advanced techniques (e.g. DMA), our approach prevents overhead in bus transactions by introducing a local 8 bit microcontroller, PicoBlaze, in the coprocessor. As a result, the performance of the ECC coprocessor can be almost independent of the selection of bus protocols. To further accelerate the Uni-PicoBlaze based ECC SoC design, a Dual-PicoBlaze based architecture is proposed, which can achieve the maximum instruction rate of 1 instruction/cycle to the ECC datapath. Using design space exploration of a large number of system configurations of different architectures discussed in this paper, our proposed Dual-PicoBlaze based design also shows best trade-off between area and speed.

1 Introduction

FPGAs are ideal for System-on-Chip (SoC) or Hardware/Software (HW/SW) codesign. Embedded processor soft-cores enable integration of various peripherals or coprocessors. Computationally intensive kernels are well suited for hardware acceleration. In this work we investigate the acceleration of the point operations from Elliptic Curve Cryptography (ECC). This task remains a challenge because of the implementation complexity of ECC, which requires optimizations at multiple abstraction levels. We will demonstrate that there is a strong interaction between the target architecture and the most suitable HW/SW partitioning.

Many HW/SW codesigns have been proposed to evaluate trade-offs between cost, performance and security in ECC system designs. Most of the target platforms fall into two categories: 8 bit platforms (e.g. AVR or 8051) [1,2,3,4,5,6] and 32 bit microprocessors and bus systems (e.g. MicroBlaze with PLB bus) [7,8,9]. Although the design goals in these two platforms may differ due to different applications (e.g. low power sensor nodes *vs*. high performance security

J. Becker et al. (Eds.): ARC 2009, LNCS 5453, pp. 169–180, 2009.
© Springer-Verlag Berlin Heidelberg 2009

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 discussed in this paper differs from previous research by realizing the fact that both of the 8 bit microcontrollers and 32 bit microprocessors can co-exist in modern FPGAs and the combination of them in one SoC design may result in a better trade-off between area and speed. In this work, we propose an SoC architecture for ECC Point Multiplication that uses a central 32 bit microprocessor, an 8 bit local control processor, and a dedicated field arithmetic datapath. The contribution of our work is two-fold. First, 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. Second, we optimize the local control unit by converting a Uni-Picoblaze sequencer architecture into a Dual-Picoblaze architecture which runs interleaved instruction sequences. This novel Dual-PicoBlaze based architecture achieves the instruction transfer rate of 1 instruction/cycle, while a Uni-Picoblaze architecture only provides half that speed, 1 instruction per 2 cycles. Moreover, the FPGA implementation results show that our proposed ECC SoC architecture with the Dual-PicoBlaze based coprocessor has a better trade-off between area and speed.

The remainder of this paper is organized as follows. Section 2 gives a brief description of current available processor cores in FPGAs and their different usages. Our complete ECC SoC system design will be presented in Section 3 and detailed discussion on our proposed Uni-PicoBlaze and Dual-PicoBlaze based coprocessor designs is given. Section 4 explains the system-level design flow used in the paper, and performance results are analyzed and compared between cosimulation and FPGA implementation. Section 5 concludes the paper.

2 Embedded Processor Cores in FPGAs

Modern FPGAs are considered as configurable systems on chip (CSoC) [21]. A primary component for such CSoC systems is the processor core. A wide range of bit widths from 8 to 32 bit are used in SoC designs. Small bit widths have the advantage of a small memory footprint for simple applications, but also imply a limited complexity. Wide instructions allow for much more complex applications, but will also require a large amount of memory even for small applications [10].

Various synthesizable processor cores are available for FPGAs. On one hand we have some popular embedded processors like AVR8[11] and Leon2 [12]. 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. They can be classified into two categories: 8 bit cores like the Xilinx PicoBlaze or the Lattice Mico8 and 32 bit cores like the NIOS II and the Microblaze. The use of 32 bit cores

targets at high performance applications and sometimes requires external memory. Compared with the 32 bit cores, the 8 bit cores have very simple IO 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. Hence, they consume very few hardware resources on FPGAs, which makes them sometimes be an ideal alternative for complicated FSMs.

As indicated above, 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.

3 ECC SoC Design

Curve-based cryptography, especially ECC, has become very popular in the past several years [22]. 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 implementation because of the relatively short operand length compared to other public-key scheme, like RSA. However, ECC is still considered as a computational intensive application due to the complexity of scalar or point multiplications. As shown in Fig. 1, a scalar multiplication, $k \cdot P$, with k is an integer and P is a point on an elliptic curve, needs to be realized through a sequence of point additions and doublings. These group operations can be further decomposed in several types of finite field arithmetic with ultra-long operand word length (e.g. 163 bit or above).

There are many design options for ECC designs, and different approaches differ in the selection of coordinate system, field and type of curve[13]. Since in our design the main focus lies on the architectural optimization of the ECC SoC system, we start with a baseline design using Montgomery Scalar Multiplication on GF(2^{163}) based on L-D projective coordinates [14]. For hardware implementations of the lowest level field arithmetic, the GF multiplication is implemented both as bit-serial [15] and digit-serial multipliers [16] with different digit sizes; the GF addition is simply logic XORs; the GF square is implemented by dedicated hardware with square and reduction circuits [13]; the GF inversion consists of a sequence of GF multiplications and squares based on Fermat's Theorem [17].

Since the core component of our ECC coprocessor is the field multiplier, here we define a design space based on the use of different field multipliers to discuss the design trade-offs between area and speed. A basic bit-serial multiplication in GF(2^m) can be realized through an classic 'shift-and-XOR' based MSB-first bit-serial multiplier with interleaved reduction modulo the irreducible polynomial [15]. It can finish one GF(2^{163}) multiplication in 163 clock cycles. A digit-serial multiplier [16] 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

Table 1. 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

multiplication time is 3 clock cycles. In this case we only select D size of 55 for our implementations.

The HW/SW partitioning adopted in this design is trying to offload the field arithmetic operations from the microprocessor and execute them in a dedicated coprocessor [2,7]. For our baseline design, all other operations, such as point addition/doubling, are implemented in software running on microprocessor. 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 a large amount of instruction and data transfers.

From the above analysis on the HW/SW partitioning, we already know where the system bottleneck will be, so before starting the system-level design we should first quantitatively measure the bus transactions in the baseline design, and estimate the optimization room left for us. Table 1 shows the system profiling from GEZEL cosimulation [18,19], and more details can be found in section 4.1. 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.

3.1 Baseline ECC SoC Design with Coprocessor Local Storage

In a straightforward design (see Fig.1 (a)), 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. From Table 1, it is observed that for a 163 bit scalar multiplication, there are 2,788 times read/write on eight 163 bit coprocessor local registers, so if all parameters and intermediate results are stored in main memory, this may result in 16,728 times 32 bit data transfers over the 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 Fig. 1, so that the amount of data transfers over the processor-to-coprocessor bus can be minimized.

3.2 Uni-PicoBlaze Based ECC SoC Design

The above design mitigates the overall bus communication overhead by optimizing the data transfer side; however, instruction transfers still dominate the

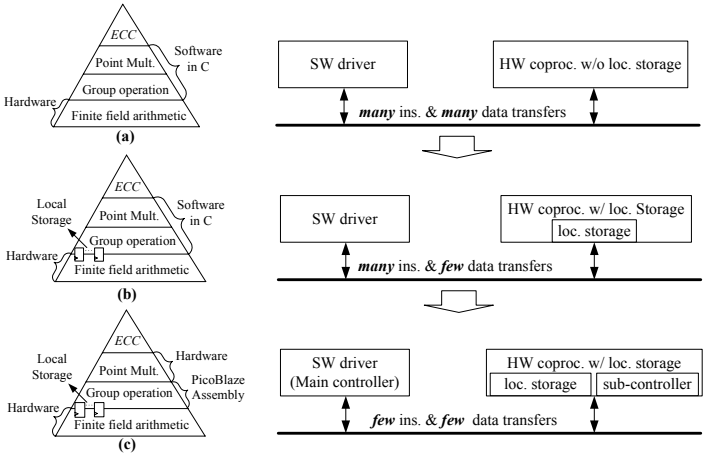

Fig. 1. System architecture modeling of different schemes

entire scalar multiplication time. From the cosimulation profiling (see Table.1), we can see for a 163 bit 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. 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 and cost-effective embedded 8 bit RISC microcontroller core optimized for Xilinx FPGAs. It has predictable performance, always 2 clock cycles per instruction, and 1K instructions of programmable on-chip program store, automatically loaded during FPGA configuration. It only costs 63 slices and 1 block RAM on Virtex-5 XC5VLX50 FPGA.

By introducing PicoBlaze as a sub-control hierarchy to be in 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 Fig. 1). The PicoBlaze has the 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. 1), we can see that for the Uni-PicoBlaze design the instruction and data transfers have been greatly reduced to 481 and 489, respectively. Here, we want to further point out that 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 the continuous read on 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 ECC task's occupation time on MicroBlaze and PLB bus, which then may be used for other peripherals in a large system.

3.3 Dual-PicoBlaze Based ECC SoC Design

After analyzing the above design we find two characteristics of Uni-PicoBlaze based ECC coprocessor design, which can help us to further refine the design. First, the sub-control unit, PicoBlaze, in 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 Uni-PicoBlaze is fixed 2 clock cycles per operation, which means that there is still one cycle per operation wasted. So, from the above two observations we propose the idea of optimizing the local control unit by converting an Uni-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 Uni-PicoBlaze to the Dual-PicoBlaze, a simple example of PicoBlaze assembly codes executing one field multiplication followed by an addition is shown in Fig.2. Compared with the Uni-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 4.3.

As we know there are some traditional ways which can achieve the maximum instruction rate of 1 instruction/cycle, such as implementing the point operations in FSM or using microcoded controller with pre-set micro codes [7].

Compared with the first approach using FSMs, our Dual-PicoBlaze architecture is more flexible and efficient. In general, the field operations can already be very fast (a digit-serial multiplier with D size of 82 can finish one 163 bit field multiplication in 2 clock cycles) and big performance gain of the whole underlying ECC system can only be obtained if new point operation algorithms 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 addition, this method can also enable the integration of the latest countermeasures against side-channel attack into the algorithm for scalar multiplication. Moreover, many people regard the 8 bit microcontroller, like PicoBlaze, as a replacement for large and complicated FSMs since they are always hard to write and debug.

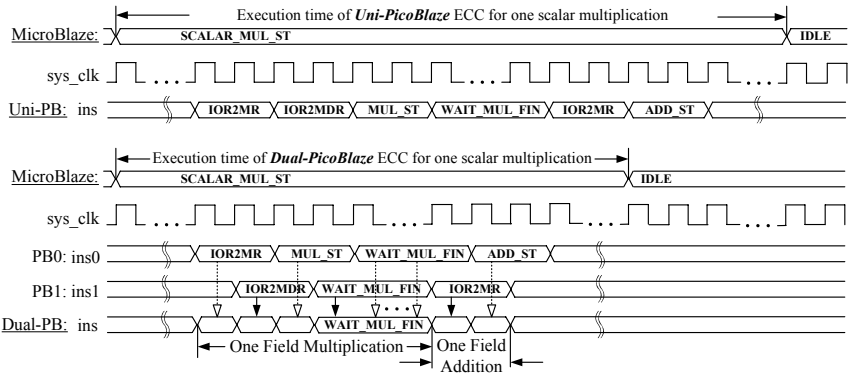

Fig. 2. An example of interleaving instructions

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

4 Design Flow and Implementation

4.1 System-Level Design and Co-simulation Using GEZEL

In order to narrow the gap between performance and flexibility, to reduce the time required to complete a design and to reduce the risk of errors that might result from translating a high-level prototype (e.g. C model) into HDLs, we use GEZEL to perform system-level design [18,19,20]. GEZEL is especially suitable for the exploration of domain-specific coprocessor and multiprocessor micro architectures as it can provide cycle-true hardware/software co-simulation with various embedded core instruction set simulators. This shortens the design time for both HW and SW. After finishing cosimulation the GEZEL file can be automatically translated into synthesizable VHDL files.

As shown in Fig. 3, the cosimulation is based ARM and PicoBlaze with PLB IPIF and all of them are instantiated as 'ipblock' in GEZEL. The implementation of the whole ECC coprocessor in GEZEL is based on Finite State Machine with Datapah (FSMD) model. The ARM communicates with the coprocessor through three 32 bit memory-mapped registers. An instruction decoder with a FSM and PicoBlazes are added on the top of the hardware field multiplier to dispatch instructions. Then, it will be attached to an interface module, 'user_ logic', to be connected with ARM through PLB IPIF. The last step is to develop software drivers in C. It should be pointed out that the GEZEL cosimulation can not only verify the correctness of the coprocessor and generate the corresponding VHDLs of the function unit, but also generate the bus interface module in VHDL.

Therefore, by following the GEZEL design flow, system designers do not need to make any change in hardware when doing the FPGA SoC integration of coprocessors. However, minor changes have to be made in the software driver when shifting from GEZEL cosimulation to the FPGA implementation because the ARM ISS is replaced with the actual microprocessor, a MicroBlaze core. The comparison between GEZEL cosimulation and FPGA implementation, in terms of the software driver and bus interface implementations, are illustrated below.

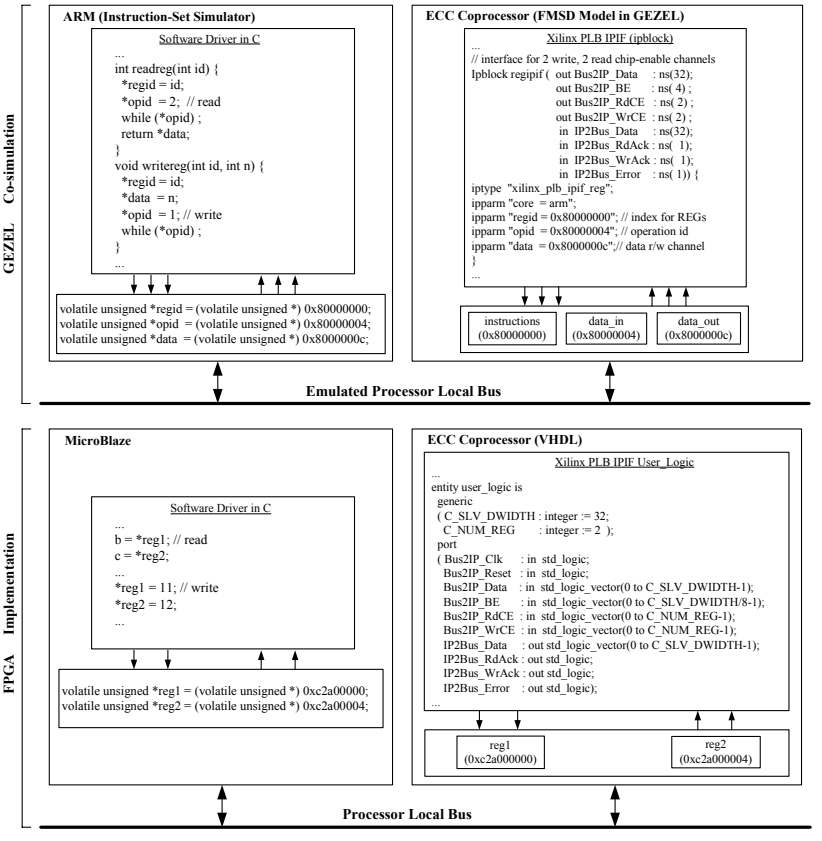

Fig. 3. Comparison between the GEZEL cosimulation and FPGA implementation

4.2 FPGA Implementation

After GEZEL cosimulation we can translate the GEZEL description of the ECC datapath and control wrappers into synthesizable VHDLs, which can be added as coprocessors in the Xilinx Platform Studio (XPS) 9.2.02i. The system shown in Fig. 4 is built on Xilinx Virtex-5 XC5VLX50 ML501 development board. A hardware timer is added for measuring cycle counts for each design configuration.

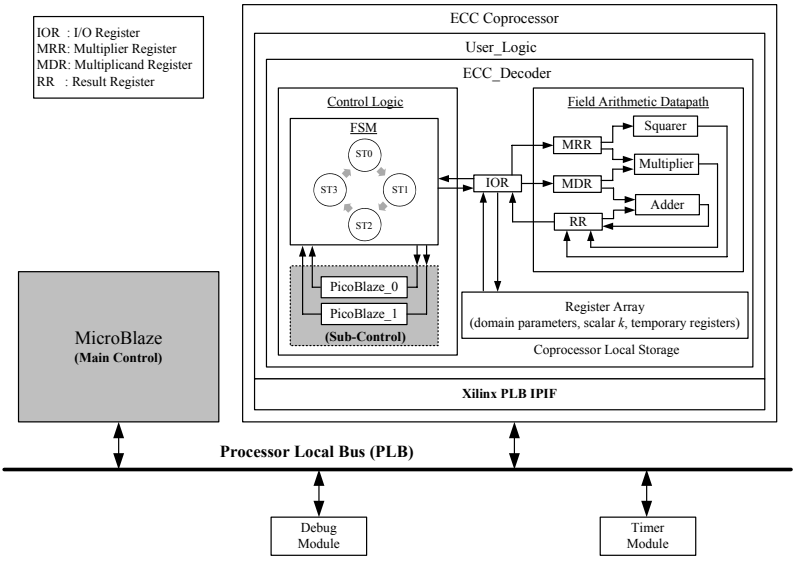

Fig. 4. FPGA implementation block diagram of ECC SoC system

4.3 Discussion of Experimental Results

From the deterministic and cycle-accurate GEZEL cosimulation, we can obtain both of the standalone hardware design profiles and the system profiles (e.g. cycle and toggle counts) [20]. This is very helpful for system designers to evaluate performance at a very early design stage and accelerate their design space exploration. To verify the correctness of our cosimulation, the FPGA implementation performance results measured by hardware timer are also added for comparison.

As indicated in Fig. 5, for the baseline design without PicoBlaze, 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. Similarly, due to overhead introduced in the ARM ISS model, the bus latency is much higher, around 44 clock cycles, which results in no overall system speedup observed after D size of 2. Whereas, for the other PicoBlaze based designs, since the bus transactions are no longer the bottleneck, we can observe comparable results between the FPGA implementation and GEZEL cosimulation. Since the control hierarchy optimization by introducing PicoBlaze also features small hardware overhead, from Fig. 6 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 design based on Virtex-2 Pro XC2VP30 FPGA. As shown in Table 2, our Dual-PicoBlaze based ECC (with maximum frequency around 200MHz on

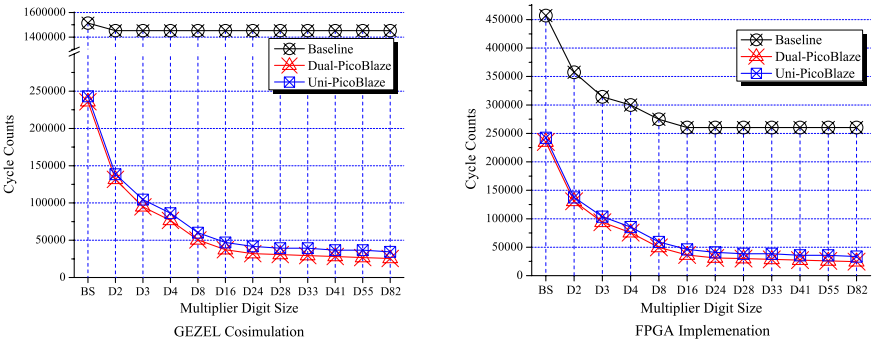

Fig. 5. Comparison of cycle counts of GEZEL cosimulation and FPGA implementation for one full scalar multiplication

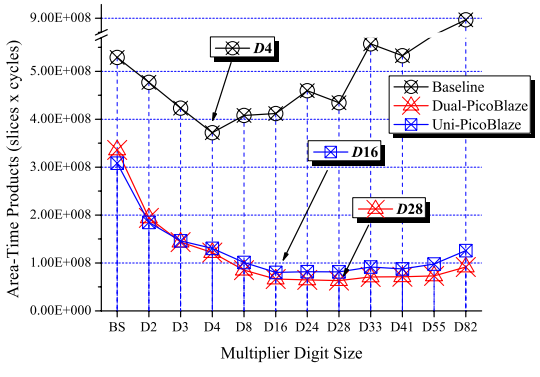

Fig. 6. Comparison of time-area products for each configuration of coprocessors

XC2VP30) shows a better 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 50.3% and 53.4% smaller. Although the current version of Dual-PicoBlaze design does not support arbitrary field size as [7], it still offers an ideal alternative since in most cases the arbitrary field size is not required. We want to also point out that 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. However, we believe our proposed ECC SoC architecture can be easily adapted to other system requirements, such as the integration of high-level (e.g. algorithmic level) countermeasures against simple power analysis (SPA) and differential power analysis (DPA) attacks [23].

Table 2. Comparison of ECC coprocessor implementations on FPGAs

	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 [7]	GF(2^{163})	V2Pro	4,749	48,800	Arbitrary	1xMALU163
ref.2 in [7]	GF(2^{163})	V2Pro	8,450	28,000	Arbitrary	2xMALU163

5 Conclusions

This paper introduced an ECC coprocessor design using PicoBlaze as sub-control hierarchy. Starting from the system profiling of a baseline ECC design 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 Uni-PicoBlaze based ECC coprocessor design. Since with this architecture, the impact of the bus latency is almost negligible, we then focus on improving the computational performance of the design to achieve the maximum instruction rate. This leads to a novel Dual-PicoBlaze based ECC architecture is proposed, which can achieve a better trade-off between area and speed with proper choosing the digit size of field multiplier. With flexibility, ease of integration of multiple PicoBlazes into current FPGA systems and predictable performance, this Dual-PicoBlaze based architecture can not only be extended to other curve-based cryptography systems, but also to some other computational intensive applications.

Acknowledgments. This project was supported in part by the National Science Foundation through grant 0644070. The authors would like to thank the Xilinx University Program for their hardware support.

References

1. Gura, N., et al.: An End-to-End Systems Approach to Elliptic Curve Cryptography. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 349–365. Springer, Heidelberg (2003)
2. Koschuch, M., et al.: Hardware/Software Co-design of Elliptic Curve Cryptography on an 8051 Microcontroller. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 430–444. Springer, Heidelberg (2006)
3. Gura, N., et al.: Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 119–132. Springer, Heidelberg (2004)
4. Aigner, H., Bock, H., Hütter, M., Wolkerstorfer, J.: A low-cost ECC coprocessor for smartcards. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 107–118. Springer, Heidelberg (2004)

5. Batina, L., et al.: Hardware/software co-design for hyperelliptic curve cryptography (HECC) on the 8051 μP. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 106–118. Springer, Heidelberg (2005)
6. Hodjat, A., Hwang, D., Batina, L., Verbauwhede, I.: A hyperelliptic curve crypto coprocessor for an 8051 microcontroller. In: SIPS 2005, pp. 93–98. IEEE, Los Alamitos (2005)
7. Sakiyama, K., Batina, L., Preneel, B., Verbauwhede, I.: Superscalar Coprocessor for High-Speed Curve-Based Cryptography. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 415–429. Springer, Heidelberg (2006)
8. Cheung, R.C.C., Luk, W., Cheung, P.Y.K.: Reconfigurable Elliptic Curve Cryptosystems on a Chip. In: DATE 2005, vol. 1, pp. 24–29. IEEE, Los Alamitos (2005)
9. Klimm, A., Sander, O., Becker, J., Subileau, S.: A Hardware/Software Codesign of a Co-processor for Real-Time Hyperelliptic Curve Cryptography on a Spartan3 FPGA. In: Brinkschulte, U., Ungerer, T., Hochberger, C., Spallek, R.G. (eds.) ARCS 2008. LNCS, vol. 4934, pp. 188–201. Springer, Heidelberg (2008)
10. Hemple, G., Hochberger, C.: A resource optimized Processor Core for FPGA based SoCs. In: DSD 2007, pp. 51–58. IEEE, Los Alamitos (2007)
11. AVR Core at opencores.org (2008), `http://www.opencores.com/projects/avr_core/`
12. Gaisler Research: LEON2 Processor User's Manual (2005)
13. Hankerson, D., Menezes, A.J., Vanston, S.A.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)
14. López, J., Dahab, R.: Fast multiplication on elliptic curves over GF(2^m). In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316–327. Springer, Heidelberg (1999)
15. Großschädl, J.: A low-power bit-serial multiplier for finite fields GF(2^m). In: ISCAS 2001, vol. IV, pp. 37–40. IEEE, Los Alamitos (2001)
16. Kumar, S., Wollinger, T., Paar, C.: Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography. IEEE Transactions on Computers 55(10), 1306–1311 (2006)
17. Rodríguez-Henríquez, F., Saqib, N.A., Díaz-Pérez, A., Koç, Ç.K.: Cryptographic Algorithms on Reconfigurable Hardware. Springer, Heidelberg (2006)
18. Schaumont, P., Ching, D., Verbauwhede, I.: An Interactive Codesign Environment for Domain-specific Coprocessors. ACM Transactions on Design Automation of Electronic Systems 11(1), 70–87 (2006)
19. Schaumont, P., Verbauwhede, I.: A Component-based Design Environment for Electronic System-level Design. IEEE Design and Test of Computers Magazine, special issue on Electronic System-Level Design 23(5), 338–347 (2006)
20. Guo, X., Chen, Z., Schaumont, P.: Energy and Performance Evaluation of an FPGA-Based SoC Platform with AES and PRESENT Coprocessors. In: Bereković, M., Dimopoulos, N., Wong, S. (eds.) SAMOS 2008. LNCS, vol. 5114, pp. 106–115. Springer, Heidelberg (2008)
21. Becker, J.: Configurable systems-on-chip (CSoC). In: SBCCI 2002, pp. 379–384. IEEE, Los Alamitos (2002)
22. Koblitz, A. H., Koblitz, N., Menezes, A.: Elliptic Curve Cryptography: The Serpentine Course of a Paradigm Shift (2008), `http://eprint.iacr.org/2008/390`
23. Coron, J.-S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)