Optimizing the HW/SW Boundary of an ECC SoC Design Using Control Hierarchy and Distributed Storage

Xu Guo
Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061, USA
Email: xuguo@vt.edu

Patrick Schaumont
Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061, USA
Email: schaum@vt.edu

***Abstract*—Hardware/Software codesign of Elliptic Curve Cryptography has been extensively studied in recent years. However, most of these designs have focused on the computational aspect of the ECC hardware, and not on the system integration into a 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 local control as well as local storage in the coprocessor. We quantify the impact of this strategy on a prototype implementation for Field Programmable Gate Arrays (FPGA) and measured an average speed-up in the resulting design of 9.4 times over the baseline ECC system, while the resulting system area increases by a factor of 1.6. The optimal area-time product improvement of our ECC coprocessor is 4.3 times compared to that of the baseline ECC coprocessor. Using design space exploration of a large number of system configurations using the latest FPGA technology and tools, we show that the optimal choice of ECC coprocessor parameters is strongly dependent on the efficiency of system-level communication.**

I. Introduction

Public-key cryptosystems, especially elliptic curve cryptography [1,2,18] and recently extensively discussed hyper-elliptic curve cryptosystems (HECC) [3], 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 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 HW/SW codesign which offers the advantage of flexibility in software with performance in hardware. The challenge of codesign is to perform proper partitioning of system functionality in hardware and software, and to map the resulting partitions onto the system architecture.

In the past five 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) [1,5,6,7,16,17,19] as well as 32 bits microprocessors and bus systems. Orlando and Paar [8] proposed a scalable elliptic-curve processor architecture which operates over the binary field GF(2^m). Gura et al [1] have introduced a programmable hardware accelerator for ECC over GF(2^m), which can be attached to a 64 bits PCI bus and support field sizes up to 255. These two papers emphasize on the optimization of the coprocessor for speedup and scalability. Sakiyama et al [9] explored architecture optimizations for ECC coprocessors and showed how to exploit parallelism using local control and local coprocessor storage. His 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 [10] 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.

Compared with the previous work, this work presents three contributions to codesign for ECC. First, we present a complete ECC SoC (system-on-chip) design and focus on the system integration issues on a real FPGA platform. We present system performance-profiles for a MicroBlaze processor with 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 parameters. Third, we quantify the impact of local control and additional local storage in the coprocessor, and we show how the system performance regions are affected.

The remainder of this paper 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 hardware/software partitioning will be given. The implementation details will be discussed in Section 4 and comparison of the results for various implementation options are shown in Section 5. Section 6 concludes the paper.

978-3-9810801-5-5/DATE09 © 2009 EDAA

TABLE I
BASIC CONFIGURATION FOR OUR ECC DESIGN

Coordinate System	L-D Projective coordinates [11]
Point Mult.	Montgomery Scalar Mult. [11]
Field	GF(2^{163})
Curve	Koblitz curve (B-163)
GF Mult.	Bit- /Digit-serial multipliers [12,13]
GF Addition	Logic XOR operations
GF Square	Dedicated hardware with square and reduction circuits [4]
GF Inversion	GF Multiplications and Squares based on Fermat's Theorem [14]

II. ECC SCALAR MULTIPLICATIONS

Curve-based cryptography, especially ECC, has become very popular in the past several years [20]. 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 schemes, like RSA. However, ECC is still considered as a computational intensive application due to the complexity of scalar or point multiplications.

A. Basic Configurations

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 of point additions and doublings. This operation dominates the execution time of cryptographic schemes based on ECC, such as signatures (ECDSA). There are many design options for ECC implementations, including the coordinate system, the field and the type of curve used [4]. In this paper, we started from a basic configuration as shown in Table 1.

B. ECC Design Space

Starting from the basic configuration in Table 1, we consider a design space for ECC system architecture 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 [12]. 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.

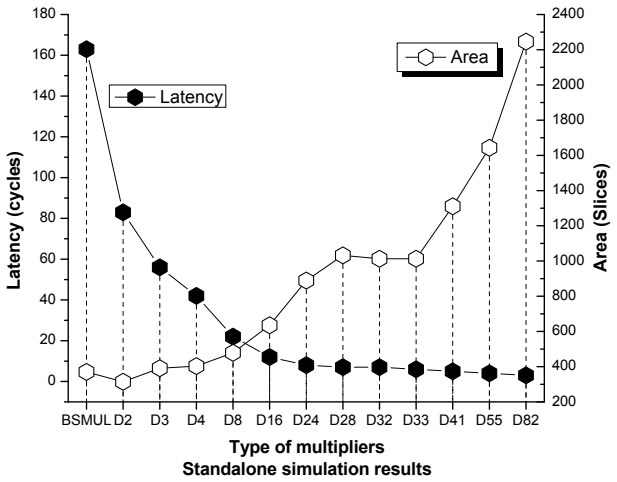

Fig. 1. Time to complete one multiplication and area for each type of multipliers

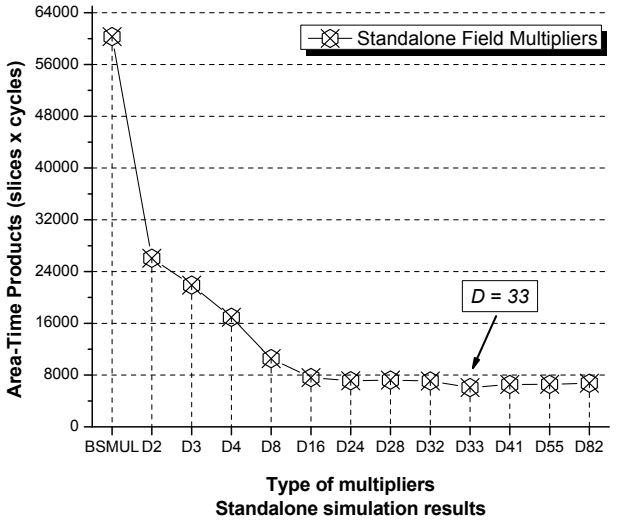

Fig. 2. Area-Time products of the different multiplier implementations

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 1 and Figure 2, which show the results for a Virtex-5 XC5VLX50 FPGA with Xilinx ISE9.2i. For the area metric in Virtex-5 FPGA, we use the unit, slice, which is based on the new six-input LUTs.

Notice that even though a digit-serial multiplier was presented in [13], 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 Fig. 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.

III. Considerations for Hardware/Software Partitioning

Several researchers [5,9] have proposed to implement 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, 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. 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.

IV. Design Flow and Implementation

A. Proposed Optimizations

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

Apart from architecture-level optimizations (Scheme B and C), we also evaluated the impact of algorithmic-level optimizations to further improve the original Montgomery Scalar Multiplication algorithm [11]. These algorithmic optimizations focus on the data and instruction transfers. First, by adding an I/O local register 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.

B. Impact of Distributed Storage

Scheme A: using Processor Local Memory (PLM) as main storage. In a straightforward design, the ECC system will implement the point operations on the main processor.

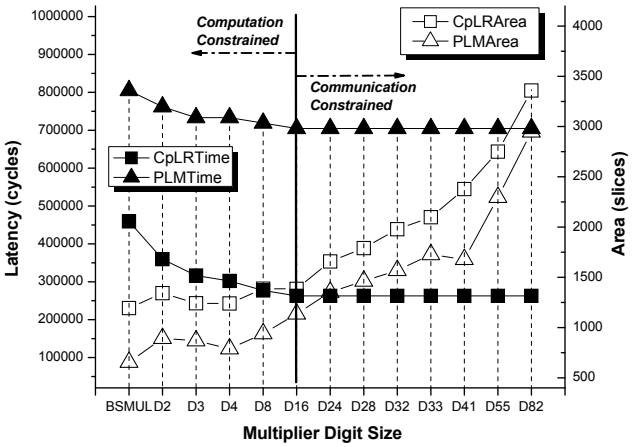

Fig. 4. Time to complete one multiplication and area for each type of coprocessors

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 shown in Fig. 3.

Scheme B: using Coprocessor Local Registers (CpLR) as distributed 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, eight 163 bits registers were added to the coprocessor. It is also possible to use SRAM blocks instead of registers to save slices with a little timing penalty introduced by the SRAM read/write latency.

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

Comparing Scheme A (PLM) with B (CpLR), Scheme B provides an average speedup of 2.5 times at the expense of a 1.4 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 throughput of the PLB bus between the coprocessor and 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 hardware design may not result into the best system solution when system integration overhead is considered.

Note that the slice count denotes the area of the entire coprocessor, including the multiplier core, wrapper with decoder, the PLB/IPIF user-interface and the PLB bus-decoding. As far as we are aware, all published results on ECC coprocessors focus on the computational core, and they do not include the overhead of the latter two parts (user-interface and bus-decoding).

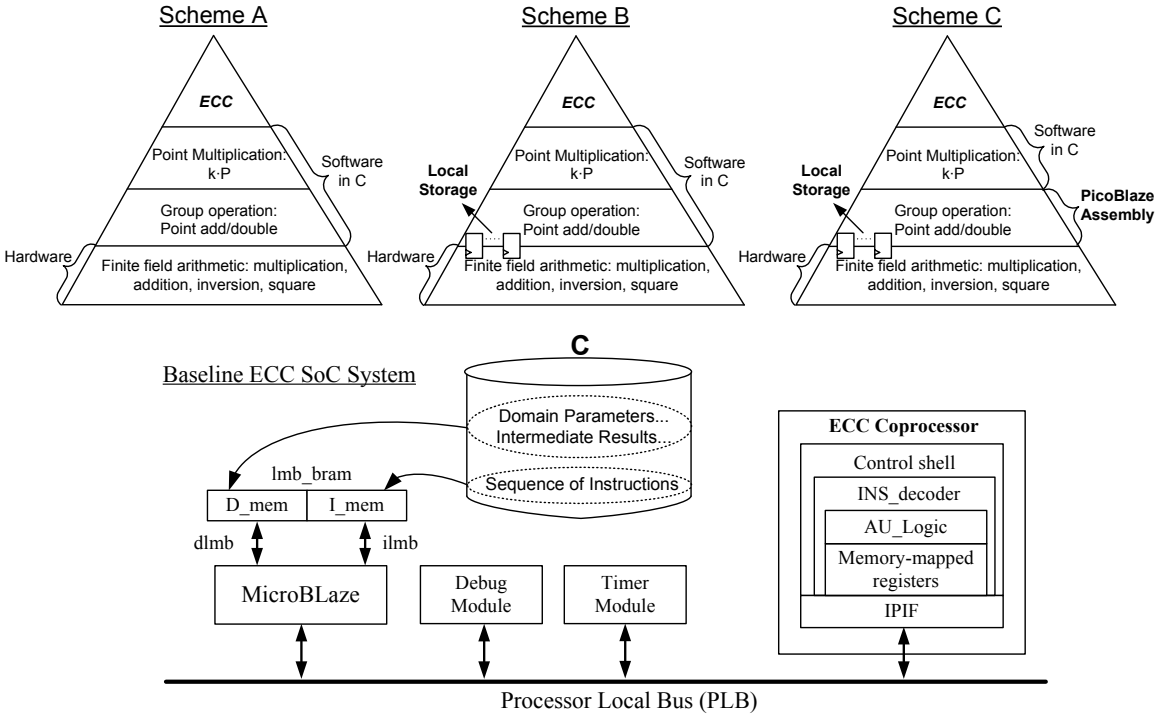

Fig. 3. ECC system schemes and block diagram of baseline ECC SoC system

We choose not to ignore this hardware, since it is essential for the hardware-software communication infrastructure, and an integral part of the coprocessor.

C. Impact of control hierarchy

Scheme C: using PicoBlaze (PB) as control hierarchy. From the above analysis of Scheme A and B, if we want to get further system speedup, we should shift the optimization from coprocessors to the system architecture. Specifically, we should try to eliminate the communication overhead. 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 63 slices and 1 block RAM on Virtex-5 XC5VLX50 FPGA.

By utilizing PicoBlaze as a local controller to implement the point addition and doubling, the MicroBlaze system processor only needs to start a point multiplication once, after which the detailed sequencing will be completed by the PicoBlaze. The PicoBlaze has the additional advantage of having a fixed instruction rate (2 clock cycles per operation). This means

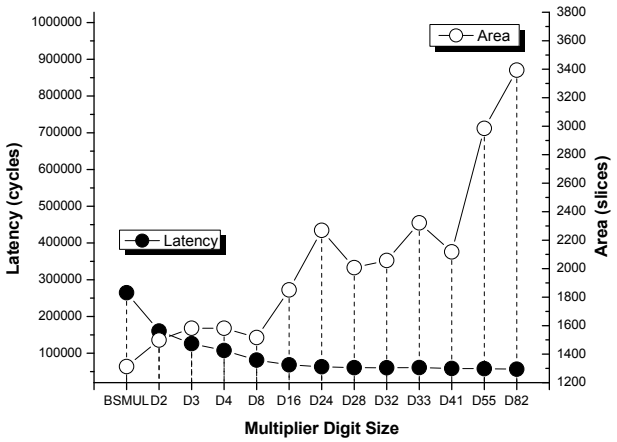

Fig. 7. Time to complete one multiplication and area for each configuration of coprocessors

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. The block diagram of this PicoBlaze based ECC SoC system is shown in Fig. 5, and the detailed structure of the complete ECC coprocessor design can be found in Fig. 6.

The results for Scheme C with a PicoBlaze controller are shown in Figure 7. In this case, 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 Figure 4, the communication bottleneck has disappeared as we can observe a speedup of 1.2 for the

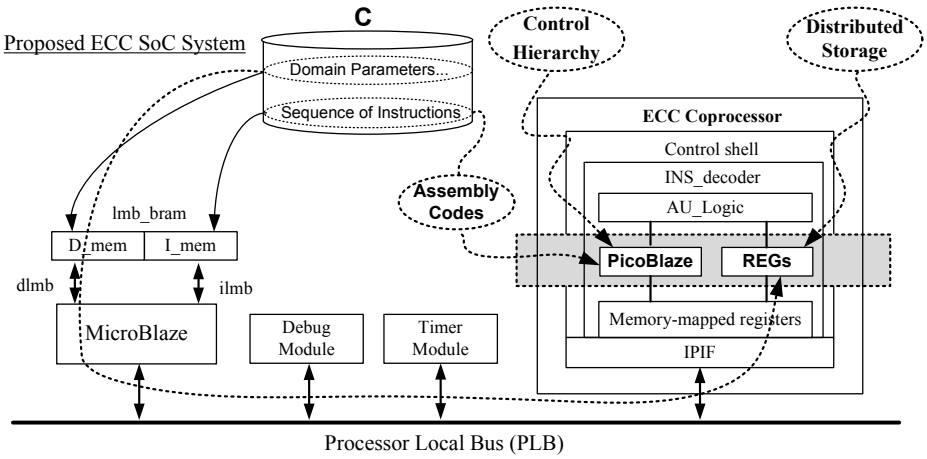

Fig. 5. Block diagram of proposed ECC SoC system

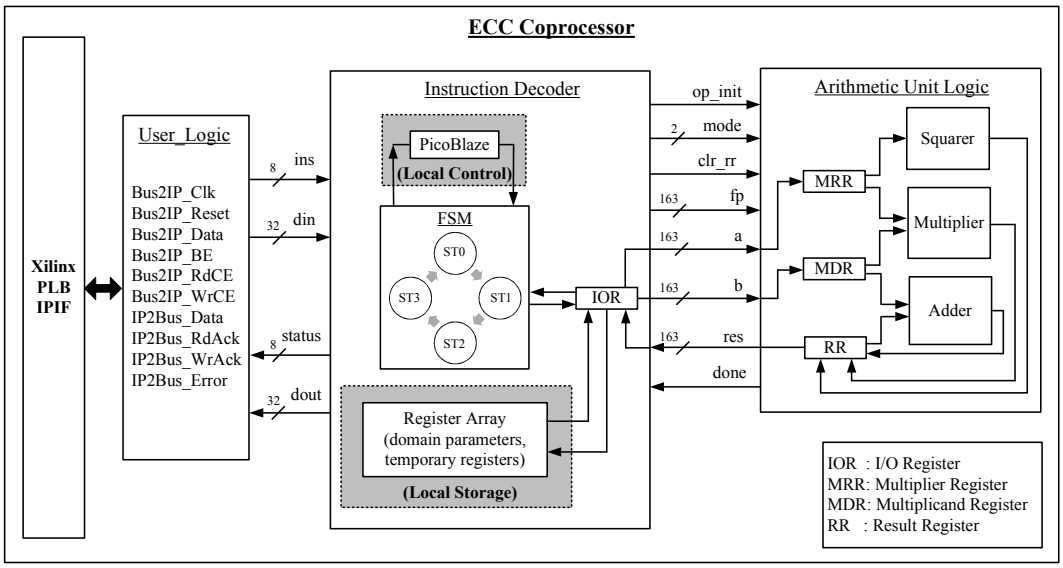

Fig. 6. The structure of proposed ECC coprocessor

coprocessor with D-size 82 over the one with D-size of 16 .

Besides performance, cost efficiency and flexibility, using PicoBlaze 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 [15]. By using PicoBlaze to implement point operation you can add dummy instructions or field computations with predictable timings to further adjust the balance of the power or timing of either branch of Madd/Mdouble operations.

V. Discussions of Experimental Results

The experimental results of all three schemes for FPGA on board implementations are combined below for system analysis and to find the best trade-off between area and speed.

According to Fig. 8, the ECC designs using PicoBlaze (PB) with almost negligible area overhead (less than 63 slices and 1 block RAM on Virtex-5 XC5VLX50) compared to the CpLR system exhibit the best trade-off between time and area among the three systems. From Fig. 8 you can also find the best trade-off design in each type of systems. For the PLM system, since the data/instruction transfer overhead between the MicroBlaze and the coprocessor takes up the majority of the cycles (about 71% of the cycles on average, over the entire ECC design space), the system speedup becomes trivial compared with the increased area overhead, so the most compact bit-serial multiplier (BSMUL) based ECC coprocessor design turns out to be the optimal one. Also, you may notice that the best trade-off design for field multiplier in standalone simulation is digit-multiplier with D size of 33. After system integration, this conclusion needs to be changed to BSMUL, D = 16, or D = 28 for PLM, CpLR and PB design configurations respectively.

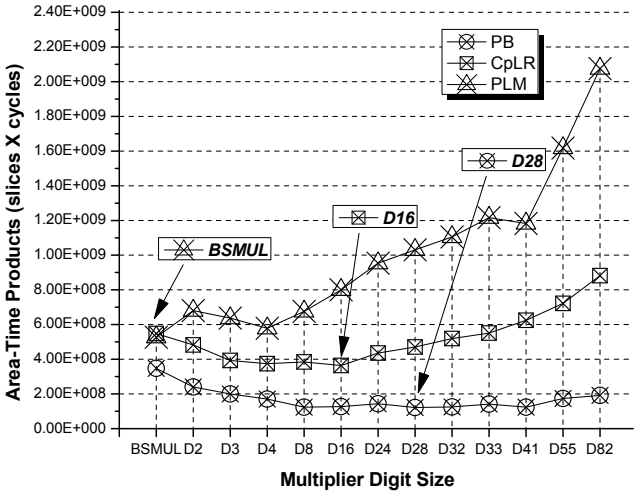

Fig. 8. Comparison of time-area products for each configuration of coprocessors

This demonstrates the large impact of system integration onto the efficiency of an ECC coprocessor. Among the three best trade-off designs, the improvement for CpLR (with D16) and PB (with D28) over PLM (with BSMUL) are 1.4 and 4.3 times, respectively, in terms of area-time products.

VI. Conclusion

In this paper we proposed an ECC coprocessor design using PicoBlaze as control hierarchy and coprocessor local registers as distributed storage to achieve the best trade-off designs between time and area. By performing design exploration we offered three design schemes with different configurations of the multipliers, and all of them have been implemented using the latest FPGA technology and tools, and implementation results and comparisons have been given. Over the whole ECC design space our proposed ECC SoC system is 9.4 times faster than the baseline ECC SoC system with 1.6 times larger area. And, the optimal area-time product improvement of our ECC coprocessor is 4.3 times compared to that of the baseline ECC coprocessor. With flexibility, ease of integration of PicoBlaze into current FPGA systems and predictable performance, this ECC SoC architecture can also be extended to other curve-based cryptography systems.

Acknowledgment

This work reported in this paper was supported in part by the National Science Foundation Grant N0. 0644070.

References

[1] N. Gura, et al. *An End-to-End Systems Approach to Elliptic Curve Cryptography*. In Cryptographic Hardware and Embedded Systems - CHES 2002, LNCS2523, pp.349-365, 2002.

[2] N. Koblitz. *Elliptic curve cryptosystems*. Mathematics of Computation, vol. 48, no. 177, pp.203-209, 1987.

[3] N. Koblitz. *A Family of Jacobians Suitable for Discrete Log Cryptosystem*. Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology, LNCS403, pp.94-99, 1988.

[4] D. Hankerson, A.J. Menezes, and S.A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Verlag, 2004.

[5] M. Koschuch, et al. *Hardware/Software Co-design of Elliptic Curve Cryptography on an 8051 Microcontroller*. In Cryptographic Hardware and Embedded Systems - CHES2006, LNCS 4249, pp.430-444, 2006.

[6] N. Gura, et al. *Comparing elliptic curve cryptography and RSA on 8-bit CPUs*. In Cryptographic Hardware and Embedded Systems - CHES 2004, LNCS 3156, pp.119-132, 2004.

[7] H. Aigner, H. Bock, M. Hütter, and J. Wolkerstorfer. *A low-cost ECC coprocessor for smartcards*. In Cryptographic Hardware and Embedded Systems - CHES 2004, LNCS 3156, pp.107-118, 2004.

[8] G. Orlando and C. Paar. *A high-performance reconfigurable elliptic curve processor for GF(2^m)*. In Cryptographic Hardware and Embedded Systems - CHES2000, LNCS1965, pp.41-56, 2000.

[9] K. Sakiyama, L. Batina, B. Preneel and I. Verbauwhede. *Superscalar Coprocessor for High-Speed Curve-Based Cryptography*. In Cryptographic Hardware and Embedded Systems - CHES 2006, LNCS 4249, pp. 415-429, 2006.

[10] R. C. C. Cheung, W. Luk and P. Y. K. Cheung. *Reconfigurable Elliptic Curve Cryptosystems on a Chip*. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05), vol. 1, pp.24-29, 2005

[11] J. López and R. Dahab. *Fast multiplication on elliptic curves over GF(2^m)*. In Cryptographic Hardware and Embedded Systems - CHES1999, LNCS1717, pp.316 - 327, 1999.

[12] J. Großschädl. *A low-power bit-serial multiplier for finite fields GF(2^m)*. In Proceedings of the 34th IEEE International Symposium on Circuits and Systems (ISCAS 2001), vol. IV, pp.37-40. IEEE, 2001.

[13] S. Kumar, T. Wollinger and C. Paar. *Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography*. IEEE Transactions on Computers, vol. 55, no. 10, pp.1306-1311, 2006.

[14] F. Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pérez and Ç. K. Koç. *Cryptographic Algorithms on Reconfigurable Hardware*. Springer, 2006.

[15] J.-S. Coron. *Resistance against differential power analysis for elliptic curve cryptosystems*. In Cryptographic Hardware and Embedded Systems - CHES1999, LNCS1717, pp.292-302, 1999.

[16] L. Batina, D. Hwang, A. Hodjat, B. Preneel, and I. Verbauwhede. *Hardware/software co-design for hyperelliptic curve cryptography (HECC) on the 8051 P*. In Cryptographic Hardware and Embedded Systems - CHES 2005, LNCS 3659, pp. 106-118, 2005.

[17] A. Hodjat, D. Hwang, L. Batina, and I. Verbauwhede. *A hyperelliptic curve crypto coprocessor for an 8051 microcontroller*. In Proceedings of the 19th IEEE Workshop on Signal Processing Systems (SIPS 2005), pp. 93-98. IEEE, 2005.

[18] V. S. Miller. *Use of Elliptic Curves in Cryptography*. In Advances in Cryptology - CRYPTO, LNCS, vol. 218, 1985, pp.417-26, 1986.

[19] S. Kumar, and C. Paar. *Reconfigurable Instruction Set Extension for enabling ECC on an 8-bit Processor*. In FPL 2004, LNCS3203, pp.586-595, 2004.

[20] Koblitz, A. H., Koblitz, N., Menezes, A.. *Elliptic Curve Cryptography: The Serpentine Course of a Paradigm Shift*. Available from http://eprint.iacr.org/2008/390, 2008.