Programmable and Parallel ECC Coprocessor Architecture: Tradeoffs between Area, Speed and Security

Xu Guo[1], Junfeng Fan[2], Patrick Schaumont[1], and Ingrid Verbauwhede[2]

[1] Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061, USA
[2] ESAT/SCD-COSIC, Katholieke Universiteit Leuven and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{xuguo,schaum}@vt.edu,{Junfeng.Fan,Ingrid.Verbauwhede}@esat.kuleuven.be

Abstract. Elliptic Curve Cryptography implementations are known to be vulnerable to various side-channel attacks and fault injection attacks, and many countermeasures have been proposed. However, selecting and integrating a set of countermeasures targeting multiple attacks into an ECC design is far from trivial. Security, performance and cost need to be considered together. In this paper, we describe a generic ECC coprocessor architecture, which is scalable and programmable. We demonstrate the coprocessor architecture with a set of countermeasures to address a collection of side-channel attacks and fault attacks. The programmable design of the coprocessor enables tradeoffs between area, speed, and security.

1 Introduction

Elliptic-curve cryptography (ECC) is the algorithm-of-choice for public-key cryptography in embedded systems. Performance, security and cost are the three important dimensions of ECC implementations. ECC accelerators should support multiple security and performance levels, allowing the system to adjust its security-performance to application-specific needs [1]. To achieve these goals, much research has been conducted targeting different aspects, and most research topics fall into two categories: efficient implementations and security analysis.

The computational intensive kernel of ECC is well suited for hardware acceleration, and many Hardware/Software (HW/SW) codesigns have been proposed to evaluate the tradeoffs between cost and performance. The challenge is how to perform optimizations at multiple abstraction levels (e.g. how to devise more efficient scalar multiplication algorithms or how to minimize the communication overhead for the HW/SW interface) and how to map the ECC system architecture to various platforms (e.g. resource-constrained 8-bit platforms or more powerful 32-bit microprocessor with bus systems).

For security analysis, ECC implementations are known to be vulnerable to various side-channel attacks (SCA), including power analysis (PA) attacks, electromagnetic attacks (EMA) and fault attacks (FA). Since Kocher *et al.* [2] showed

C. Clavier and K. Gaj (Eds.): CHES 2009, LNCS 5747, pp. 289–303, 2009.

the first successful PA attacks, there have been dozens of proposals for new SCA attacks and countermeasures. These attacks and countermeasures all tend to concentrate on a single abstraction level at a time [7]. For example, the Smart Card software is developed on fixed hardware platforms, so the results in that area are software-based solutions. At the same time, many special circuit styles [32,33] have been developed to address PA at the hardware level. Such circuit-level solutions are treated independently from the software-level solutions.

From the above descriptions, we have found two gaps in current ECC research. First, security has been generally treated as a separate dimension in designs and few researchers have proposed countermeasures targeting at system integration. For example, some of the fault attack countermeasures or fault detection methods are just like software patches applied to the original algorithms (e.g. perform Point Verification (PV) [23] or Coherence Check [3,34] during computation). The fault model is hypothesized without considering how to introduce faults in an actual platform. Further, the impact of circuit-level PA countermeasures on area, performance and power consumption in large designs remains unclear. Second, most published papers proposed their own attacks with corresponding countermeasures and very few researchers discussed countermeasures targeting multiple attacks. Since a cryptosystem will fail at its weakest link [4] it is not surprising to see how a given countermeasure can actually introduce a new weakness, and thus enable another attack [5]. Although there has been some effort to connect the PA and FA countermeasures [6], solutions at system integration level are unexplored.

Therefore the question now becomes how to fill both of the gaps in one system design. Specifically, we try to move these two research topics to the next step by building a flexible ECC coprocessor architecture with the ability to consider efficiency and security simultaneously and provide a unified countermeasure which can be easily integrated into system designs.

The contributions of this research are three-fold. First, we propose a generic programmable and parallel ECC coprocessor architecture. The architecture is scalable and can be adapted to different bus interfaces. Since it is programmable, both of the efficient ECC scalar multiplication algorithms and algorithmic level countermeasures can be uploaded to the coprocessor without hardware modifications. Second, after review of the security risks for ECC implementations, we suggest a set of countermeasure to protect the the coprocessor against different types of passive attacks and fault injection attacks. Finally, we implement a programmable and parallel ECC coprocessor on an FPGA to show the feasibility of the method. The implementation is scalable over area, cost, and security. The resulting platform allows us to quantify and compare the performance overhead of various algorithmic-level countermeasures.

The remainder of this paper is as follows. Section 2 gives a brief description of ECC implementation and related attacks with corresponding countermeasures. Our proposed generic programmable and parallel coprocessor architecture will be discussed in section 3. The unified countermeasure is analyzed in section 4. The

FPGA implementation of our proposed architecture with unified countermeasure is described in section 5. Section 6 concludes the paper.

2 ECC Background

2.1 Implementation of ECC over $GF(2^m)$

A non-supersingular elliptic curve E over $GF(2^m)$ is defined as the set of solutions $(x, y) \in GF(2^m) \times GF(2^m)$ of the equation:

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

where $a, b \in GF(2^m)$, $b \neq 0$, together with the point at infinity.

A basic building block of ECC is the Elliptic Curve Scalar Multiplication (ECSM), 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 (see Fig.1). This operation dominates the execution time of cryptographic schemes based on ECC.

2.2 Types of Attacks

Several kinds of attacks on cryptographic devices have been published. They can be categorized into two types: passive attacks and active attacks [15]. Passive attacks are based on the observation of side-channel information such as the power consumption of the chip or its electromagnetic emanations. Examples of passive attacks include Simple Power Analysis (SPA), Differential Power Analysis (DPA), Simple Electromagnetic Analysis (SEMA) and Differential Electromagnetic Analysis (DEMA). On the other hand, active attacks, including fault injection attacks, deliberately introduce abnormal behavior in the chip in order to recover internal secret data.

As mentioned earlier, a cryptosystem will fail at its weakest link and one countermeasure against one SCA attack may benefit another attack. Therefore, in this paper we want to consider active as well as passive attacks, and define a unified countermeasure that resists a collection of published attacks. Before

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** $i=n-2$ 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** $i=n-2$ to 0 **do** 3: $Q[1-k_i] \leftarrow Q[0] + Q[1]$ $Q[k_i] \leftarrow 2Q[k_i]$ **Return** $Q[0]$

Fig. 1. Elliptic Curve Scalar Multiplication (ECSM) algorithms

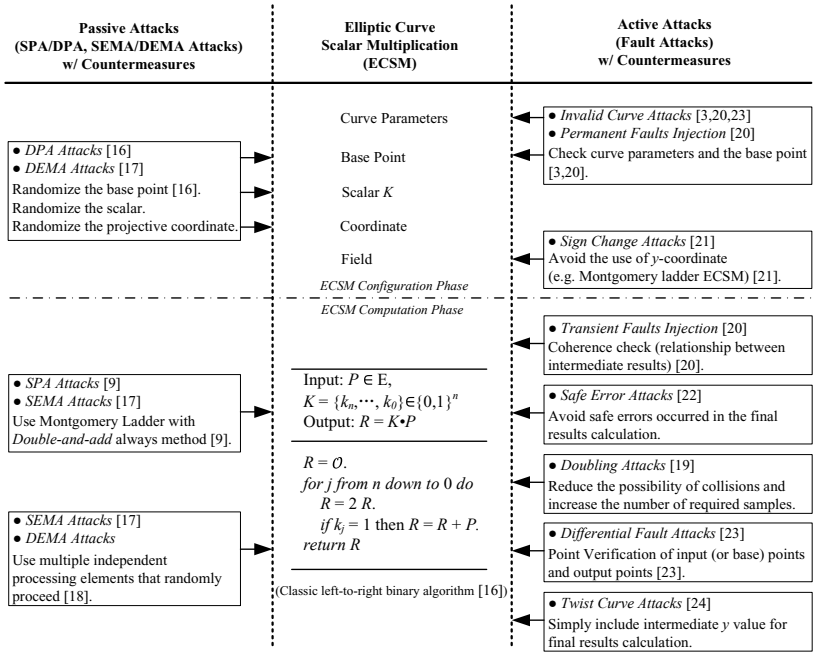

Fig. 2. Summary of attacks and corresponding countermeasures on ECC

proposing our unified countermeasure we first review the known security risks of ECC, as well as the corresponding countermeasures. A brief discussion of each attack and corresponding countermeasure will be provided in section 4.2.

As shown in Fig. 2, we divide all the countermeasures into two categories: protection at the ECSM configuration phase and computation phase. Most fault injection attacks are specific to the ECC algorithm and most of them are combined with passive attacks. Besides, some of the active attacks are very powerful. Even if countermeasures of standard passive attack are used, attackers can still easily retrieve the secret scalar K with only a few power traces (e.g. two power traces for the doubling attacks [19]).

3 Proposed Programmable and Parallel Coprocessor Architecture

Integration of various countermeasures into an existing ECC coprocessors can be very challenging. First, for many proposed ECC coprocessors with single datapath, the added countermeasures will sometimes largely sacrifice its efficiency. Second, the research of side-channel attacks keeps on evolving. Thus, how to devise a flexible ECC coprocessor which can support security updates is also very important. Therefore, a novel generic ECC coprocessor architecture design is proposed to solve the above problems. The architecture of this coprocessor is shown in Fig. 3 and all the design considerations will be discussed below.

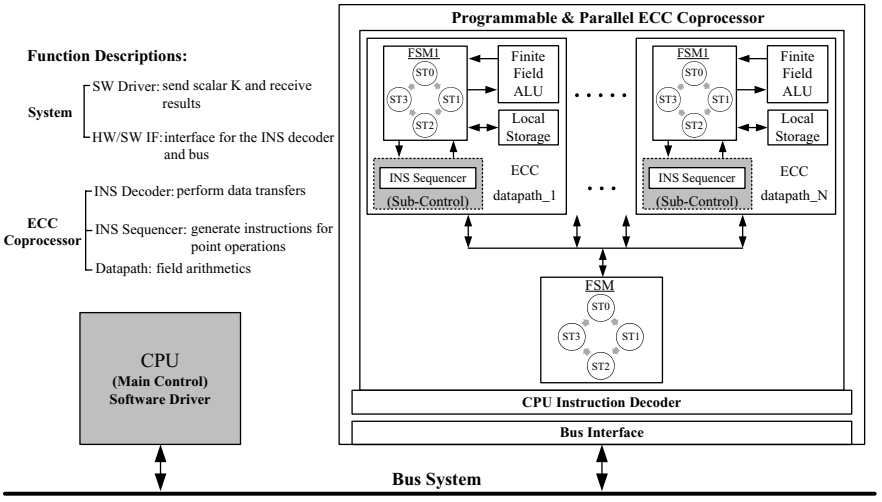

Fig. 3. The structure of generic programmable and parallel ECC coprocessor

The hardware/software partitioning method adopted in this design is trying to offload the field arithmetic operations from the CPU and execute them in a dedicated ECC coprocessor [13,14]. For traditional ECC coprocessor designs, all other higher level point operations, such as point addition/doubling, are implemented in software running on CPU. 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.

Targeting the above communication bottleneck problem we tried to optimize the HW/SW boundary in two steps: reducing data transfers as well as accelerating instruction transfers. As a result, the CPU is only in charge of sending initialization parameters and receiving final results, and the instruction sequencer will issue all the required instructions for a full ECSM. A further optimization has been made to make the ECC coprocessor programmable, which is out of two concerns. First, in general the field operations can already be very fast (a digit-serial multiplier with digit size of 82 can finish one multiplication in $GF(2^{163})$ in 2 clock cycles [11]) and big performance gain of the whole 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 the instruction sequencer with the new point operation algorithms without the need to rewrite the HDLs and synthesize the whole design. Second, this method can also enable the integration of the latest countermeasures against side-channel attacks for security updates.

3.1 Programmability

For our proposed structure, the CPU (main control) is not only able to send data/instructions through the bus, like the controller in most of the HW/SW codesigns, but also to program the instruction sequencer as a sub-controller in the coprocessor. The coprocessor consists of a CPU instruction decoder and single/multiple ECC datapaths, and each ECC datapath is composed of an instruction sequencer, a dedicated instruction decoder, ALU and local memory. Each ECC datapath can be programmed to carry out field operations independently.

However, the design of an instruction sequencer in the ECC datapath can be tricky. Since we have defined it to support the programmable feature, the direct use of hardware FSMs does not work. Another option is using a microcoded controller. However, the design of a dedicated controller with FSMs to dispatch instructions from microcoded controller itself can still be complex and inflexible. Finally, we come to a solution by customizing an existing low-end microcontroller to meet our requirements.

This programmable architecture gives us the freedom to efficiently utilize various countermeasures against different side-channel attacks. For example, we can program the sub-controller component so that it performs Montgomery ladder in order to thwart SPA attacks. We can easily add base point randomization to it in order to thwart DPA attacks. Finally, if the implementation requires resistance to fault attacks, we can update the program in the sub-controller to add coherence check [3] and so on. In short, the flexibility of programmable sub-controller makes the coprocessor able to update with the latest countermeasures.

3.2 Scalability

As shown in Fig.3, our proposed generic coprocessor architecture is scalable for parallel implementations because the ECC datapath can execute the scalar multiplication almost independent of the bus selections and CPU. Once the CPU sends the scalar K to each ECC datapath to initialize the computation, the datapath will work independently. The scalability here means the maximum number of independent ECC datapaths attached to the CPU instruction decoder is purely dependent on the bus latency. As shown in Fig.4, the 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 Fig. 4, 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. Here, 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

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

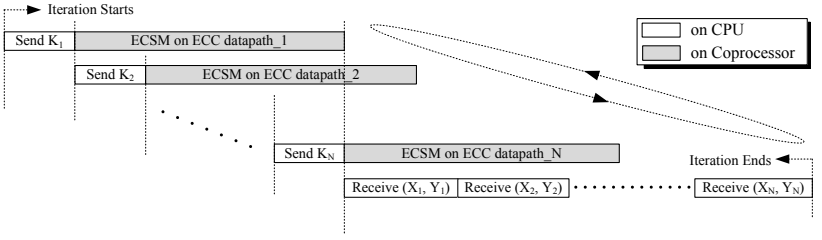

Fig. 4. Exploration of the parallelism within the proposed generic coprocessor architecture

Due to this massive parallel architecture, we can get the fastest implementation with $T_{avg\_min} = 3MT_{delay}$ cycles. This means for the fastest ECC coprocessor configuration with maximum number of parallel ECC datapath, the minimum computation time in average is only related to the bus latency. The system configuration for meeting this upper bound is to make the ECSM computation time be exactly overlapped with the communication time. For actual operations shown in Fig. 4, let the CPU keep sending scalar K and initiating ECSM until the first ECSM computation ends and then start to receive results. Also, we can have tradeoff designs between area and speed with different number of parallel ECC datapath to fit for different embedded applications, and then we can get the computation time in average, T_{avg} as

$$T_{avg} = \frac{(2N+1)MT_{delay} + T_{comp}}{N}. \tag{3}$$

4 Selection of Countermeasures

Indeed, one can not simply integrate all the countermeasures targeting different attacks, as shown in Fig. 2, to thwart multiple attacks due to complexity, cost and the fact that a countermeasure against one attack may benefit another one [5]. For example, the double-and-add-always method to prevent SPA can be used for Safe Error attacks [22]. Unified countermeasures to tackle both the passive and active attacks are attractive. Kim *et al*. proposed a unified countermeasure for RSA-CRT [25]. Baek *et al*. extended Shamir's trick, which was proposed for RSA-CRT, to secure ECC from DPA and FA [6]. However, Joye showed in [26] non-negligible portion of faults was undetected with the unified countermeasure and settings in [6].

In this paper, we suggest a set of existing countermeasures to thwart both passive and active attacks on ECC. Especially, we focus on ECC over binary extension field. We try to take into account as many attacks/countermeasures as possible. Three countermeasures are selected.

1. ***Montgomery Ladder Scalar Multiplication*** [9]. The Montgomery Powering Ladder, shown in Fig. 1, performs both point addition and poing doubling for each key bit. In theory, it is able to prevent SPA and TA.

2. ***Random Splitting of Scalar*** [27]. This method randomly splits $K = K_1 + K_2$, and performs $Q_1 = K_1 \cdot P$ and $Q_2 = K_2 \cdot P$. Eventually, $K \cdot P$ is calculated as $Q_1 + Q_2$.

3. ***Point Verification*** [3,23]. PV checks if a point lies on an curve or not. Let E: $y^2 + xy = x^3 + ax^2 + b$ be an elliptic curve defined over $GF(2^m)$, one can check the validity of a point $P(x_p, y_p)$ by simply verifying the equation $y_p^2 + x_p y_p = x_p^3 + ax_p^2 + b$.

4.1 Security Analysis of the Proposed Unified Countermeasure

SPA/SEMA attacks [9,17] use a single measurement to guess the secret bits. The use of Montgomery Scalar Multiplication computes point addition and doubling each time without depending on the value of each bit of scalar K. Therefore, it is secure against SPA attacks. For SEMA, though in [28] the authors indicated that this can also resist SEMA, we think it is too early to conclude that since in [17] the capability of multi-channel EMA attacks has not been comprehensively investigated.

DPA/DEMA attacks [16,17] use statistical models to analyze multiple measurements. The random scalar splitting was proposed in [27] and the idea behind it is very similar to Coron's [16] first countermeasure against DPA. As for DEMA, the random splitting of K can be considered as a signal information reduction countermeasure [17] against statistical attacks, including DEMA.

Doubling attacks [19] explore the operand reuse in scalar multiplication for two ECSMs with different base point. Borrowing the authors' analysis on Coron's first countermeasure in [19], the random splitting of 163 bit scalar K can simply extend the search space to 2^{81} (birthday paradox applies here), which is enough to resist the doubling attack.

Safe Error attacks [22] introduce safe errors when redundant point operations are used (e.g. double-and-add-always ECSM). No safe errors can be introduced based on our proposed scheme. In order to pass the PV, the outputs from the coordinate conversion must be correct, which means both intermediate results P_1 and P_2 must be used in order to calculate the y value at the last step of coordinate conversion. So, no safe errors can be introduced in either P_1 or P_2.

Invalid Curve attacks [20,23] need to compute the ECSM on a cryptographically weaker group. When using López-Dahab coorindates and Montgomery Scalar Multiplication, all the curve parameters will be used for calculating the final results in affine coordinates. So, if any of the curve parameters is changed in order to force the computation on a weaker elliptic curve, it cannot pass the final point verification.

Differential Fault attacks [23] try to make the point leave the cryptographically strong curve. The use of PV at the final step is the standard countermeasure to resist this kind of attacks.

Permanent Faults Injection attacks [20] target the non-volatile memory for storing the system parameters. All the curve parameters and base points in our design can be hardwired to prevent malicious modifications. If needed, the curve

parameters can also be made flexible as well, but then their integrity will need to be verified. This can be done, for example, using a hash operation.

Transient Faults Injection attacks [20] try to modify the system parameters when they are transferred into working memory. By using the random scalar splitting scheme ($K = K_1 + K_2$) the final results have to be obtained through two steps of point operations. For the first step, we use López-Dahab projective coordinates and Montgomery Scalar Multiplication to get two results for $K_1 \cdot P$ and $K_2 \cdot P$. If transient faults are inserted during this step, before coordinate conversion the invariant relation of intermediate point $Q[0]$ and $Q[1]$ (see Fig. 1) will be destroyed. As a result, $K \cdot P = (K_1 \cdot P + K_2 \cdot P)$ cannot pass the final PV since the errors will propagate in the affine point addition.

Twist Curve attacks [24] apply to the case when performing Montgomery Scalar Multiplication, the y-coordinate is not used. However, for our case, to obtain the final results both x and y are needed for final affine point addition.

Sign Change attacks [21] change the sign of a point when the scalar is encoded in Non-adjacent form (NAF). The point sign change implies only a change of sign of its y-coordinate. The use of Montgomery Scalar Multiplication and López-Dahab coordinates can resist this attack.

4.2 Map the Unified Countermeasure to the Coprocessor

In Fig. 5, the dataflow of the ECSM using our proposed unified countermeasures is illustrated. After randomly splitting the scalar K into K_1 and K_2 on the CPU,

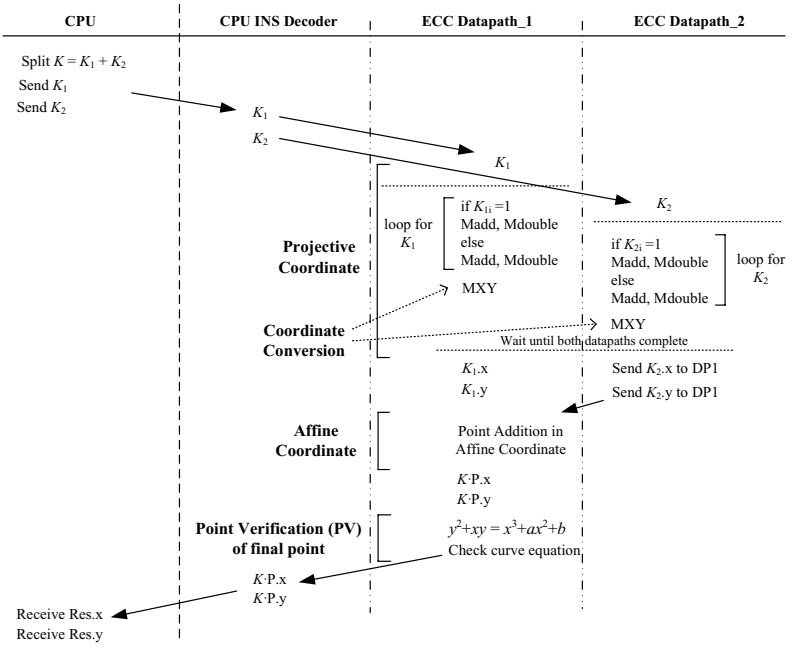

Fig. 5. Dataflow of ECSM with splitting K and PV countermeasures

K_1 and K_2 will be sent to the ECC coprocessor. Then, the calculation of $K_1 \cdot P$ and $K_2 \cdot P$ are processed concurrently, and after that the two intermediate resulting points in Affine Coordinate from both datapaths will be added, generating $K \cdot P$. The point verification will be performed before the output is generated.

4.3 Operation Modes

In order to support the countermeasures suggested above, it is obvious that two point multiplications have to be computed with an extra final affine point addition. Both introduces performance overhead. Therefore, we can make the ECC coprocessor work under different operation modes. Below, four modes have been defined with different security requirements:

1. *Normal Operation Mode.* Two ECC datapaths can calculate different K and return two separate point multiplication results. Note that this normal mode can also implement the basic duplication [31] or concurrent processing comparison [3] to detect faults by sending the same K to the two datapaths.
2. *Safe Mode with the splitting K countermeasure.* Two ECC datapaths compute the split K values in projective coordinate and then perform one affine point addition in the end. PV is not used here.
3. *Safe Mode with the PV countermeasure.* Based on the normal operation mode, we can add the PV to both datapaths before outputting the final results.
4. *Safe Mode with both splitting K and PV countermeasures.* This is the operation mode with the highest security level described in Fig. 5.

Combining the aforementioned programmable feature with the above defined four operation modes, we can customize a *protocol* of how to efficiently select modes to reduce the performance overhead.

5 FPGA Implementation

In order to prove the feasibility of the proposed generic ECC coprocessor architecture and give a concrete example with quantitative experimental results, we have also built the generic coprocessor on an FPGA based SoC platform.

When comparing the actual implementation in Fig. 6 with the generic architecture in Fig. 3, the CPU becomes the 32-bit MicroBlaze, PLB bus is used, the instruction sequencer is replaced with programmable Dual-PicoBlaze microcontrollers, the finite field ALU is implemented with addition, multiplication and square function units, and the 163-bit register array is used as local storage.

There are many design options for ECC implementations, and different approaches differ in the selection of coordinate system, field and type of curves [8]. In our design we will use Montgomery Scalar Multiplication and López-Dahab projective coordinates [9]. For hardware implementations of the lowest level field arithmetic, the field multiplication is implemented both as bit-serial [10] and digit-serial multipliers [11] with different digit sizes; the field addition is simply logic XORs; the field square is implemented by dedicated hardware with

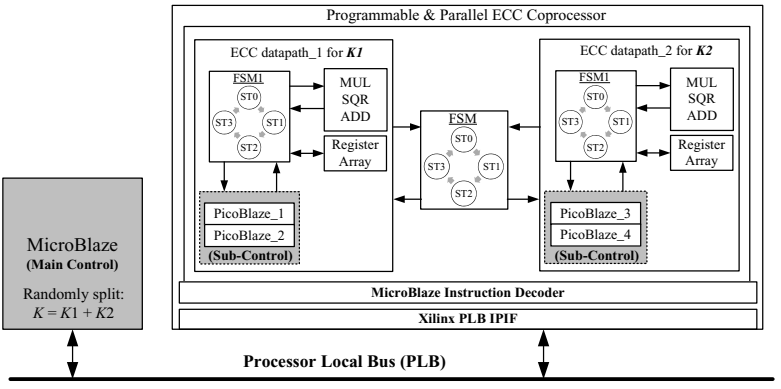

Fig. 6. Block diagram of the proposed ECC SoC architecture on FPGA

square and reduction circuits [12]; the field inversion consists of a sequence of field multiplications and squares based on Fermat's Theorem [12].

For the implementation of the instruction sequencer, 8-bit PicoBlaze microcontrollers are used. The PicoBlaze has a predictable performance. It takes always two clock cycles per instruction. It costs 53 slices and 1 block RAM on the Virtex-5 XC5VLX50 FPGA. The use of PicoBlaze as a new control hierarchy was first proposed in [29] and based on that we in [30] proposed a Dual-PicoBlaze based design to achieve a high instruction rate of 1 instruction per cycle by interleaving instructions from two PicoBlazes. However, by applying the Dual-Picoblaze architecture we can only get performance enhancement. Therefore, we decided to customize the Picoblaze by replacing the instruction ROM with a dual-port RAM to incorporate the programmable feature.

Based on the discussion on Section 3.2, the timing profile of parallel implementations of multiple ECC datapaths based on the Processor Local Bus (PLB) on a Xilinx FPGA platform can be obtained. The PLB interface is a memory-mapped interface for peripheral components with typical bus transfer latency, T_{delay}, of 9 clock cycles. Instruction and data transfers for one operand require $M = 20$ times bus transfers. As we use digit size 82 (denoted as $D82$), one scalar multiplication on ECC defined on $GF(2^{163})$ takes 24,689 clock cycles, i.e. $T_{comp} = 24,689$. If we just consider the ideal case for data/instruction transfers without any software overhead (e.g. function calls), the minimum average time, T_{avg_min}, is about of 540 clock cycles. However, 139 ECC datapaths in parallel are required to achieve this minimum delay, which makes it impractical. Still, we can find reasonable tradeoffs through Equation 3.

Following the discussion in Section 4.2, we also implement a ECC coprocessor with two datapath to integrate our proposed unified countermeasure. It is shown in Fig. 6. Moreover, we designed the coprocessor to support different operation modes as defined in section 4.3.

Since the PLB bus is 32-bit wide and our targeting implementation is based on $GF(2^{163})$, we need 6 times bus transfers to send one scalar K to the coprocessor, which means we still have 29 bits left when transfer the last word. Therefore, we

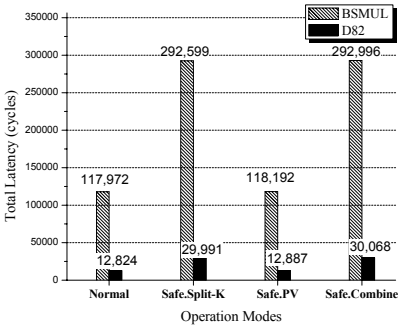

Fig. 7. Comparison of timing profiling between different operation modes

encode these bits as function bits and they can be interpreted by the MicroBlaze Instruction Decoder to make the coprocessor work under different modes.

Another advantage of this proposed architecture is that we can quantify the security cost when different countermeasures are used. Without changing the hardware, we can simply make changes in the software driver running on MicroBlaze to turn the ECC coprocessor into different operation modes. This is similar with the workload characterization in software-only implementations [35]. Hence, the security cost can be expressed in terms of pure performance overhead.

In Fig. 7, we compare the timing profiling of the design working in four operation modes. Here, we select two extreme coprocessor configurations, the smallest (with bit-serial multiplier, BSMUL) and the fastest ($D82$), for detailed comparison. The whole system works at 100MHz, and the maximum frequency of both coprocessors with BSMUL and $D82$ is 165.8MHz. The logic critical path mainly goes through the programmable PicoBlaze. The hardware cost for the bit-serial multiplier based design is 2,179 slices and 6,585 slices for the $D82$-based design when implemented on a Vertex-5 XC5VLX50 FPGA.

Note that the cycle counts for normal mode and safe mode with only PV are the average speed for two ECSMs in parallel. From [31], the authors conclude that if fair performance evaluations are performed, many fault attack countermeasures are not better than the naive solutions, namely duplication or repetition. So, this means it is also important to find a universal platform to quantify the security cost. From Fig. 7, it is easy to see that based on our proposed generic ECC coprocessor architecture we can quantify the overhead of these countermeasures in terms of a single metric – cycle counts.

To compare with other similar ECC codesigns [13,14,29,30,36,37], our proposed ECC coprocessor architecture considers optimizations for performance, flexibility and security at the same time. For performance, the designs described in [29,30], same as the base design of the ECC coprocessor architecture proposed in this paper with single ECC datapath, have already shown good tradeoffs between area and speed. For flexibility, the programmable coprocessor is addressed for its advantages of both performance and security enhancement, and the massively parallel architecture can be explored to meet various application requirements. For

security, unlike [14,36,37] which only consider passive attacks (e.g. power analysis attacks and timing attacks), our design can defend most existing passive and active attacks. Besides, it can be easily updated with new algorithmic-level countermeasures to resist new attacks without hardware changes.

6 Conclusions

In this paper we have presented a generic ECC coprocessor architecture, which can fill the gap between efficient implementation and security integration at the architecture level. For security, a unified countermeasure is proposed by combining the random scalar splitting [27] and Point Verification [3,23]. For performance, the introduction of distributed storage and new control hierarchy into the ECC coprocessor datapath can greatly reduce the communication overhead faced by a traditional centralized control scheme. Scalable parallelism can also be explored to achieve tradeoff designs between area and speed. The feasibility and efficiency of our proposed generic ECC coprocessor architecture and unified countermeasure are verified and shown from an actual implementation on FPGA. Experimental results show that the proposed programmable and parallel ECC coprocessor architecture can be suitable for a wide range of embedded applications with different user defined security requriments.

Acknowledgments. This project was supported in part by the US National Science Foundation through grant 0644070 and 0541472, by Virginia Tech Pratt Fund, by IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by FWO projects G.0300.07, by the European Comission through the IST Programme under Contract IST-2002-507932 ECRYPT NoE, and by the K.U. Leuven-BOF.

References

1. Alrimeih, H., Rakhmatov, D.: Security-Performance Trade-offs in Embedded Systems Using Flexible ECC Hardware. IEEE Design & Test of Computers 24(6), 556–569 (2007)
2. Kocher, C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
3. Dominguez-Oviedo, A.: On Fault-based Attacks and Countermeasures for Elliptic Curve Cryptosystems. PhD Thesis, University of Waterloo (2008)
4. Verbauwhede, I., Schaumont, P.: Design Methods for Security and Trust. In: Proceedings of the conference on Design, automation and test in Europe –DATE 2007, pp. 1–6 (2007)
5. Yen, S.-M., Kim, S., Lim, S., Moon, S.-J.: A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 414–427. Springer, Heidelberg (2002)
6. Baek, Y.-J., Vasyltsov, I.: How to prevent DPA and fault attack in a unified way for ECC scalar multiplication – ring extension method. In: Dawson, E., Wong, D.S. (eds.) ISPEC 2007. LNCS, vol. 4464, pp. 225–237. Springer, Heidelberg (2007)

7. Schaumont, P., Hwang, D., Yang, S., Verbauwhede, I.: Multilevel Design Validation in a Secure Embedded System. IEEE Transactions on Computers 55(11), 1380–1390 (2006)
8. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)
9. 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)
10. 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)
11. 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)
12. Rodríguez-Henríquez, F., Saqib, N.A., Díaz-Pérez, A., Koç, Ç.K.: Cryptographic Algorithms on Reconfigurable Hardware. Springer, Heidelberg (2006)
13. Koschuch, M., Lechner, J., Weitzer, A., Großschädl, J., Szekely, A., Tillich, S., Wolkerstorfer, J.: 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)
14. 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)
15. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis. In: FDTC 2007, pp. 92–102. IEEE, Los Alamitos (2007)
16. Coron, J.-S.: Resistance against differential power analysis for elliptic curve. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
17. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003)
18. Ciet, M., Neve, M., Peeters, E., Quisquater, J.: Parallel FPGA implementation of RSA with residue number systems - can side-channel threats be avoided? In: IEEE International Symposium on Micro-NanoMechatronics and Human Science, vol. 2, pp. 806–810. IEEE Computer Society Press, Los Alamitos (2003)
19. Fouque, P.-A., Valette, F.: The Doubling Attack - Why Upwards Is Better than Downwards. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 269–280. Springer, Heidelberg (2003)
20. Ciet, M., Joye, M.: Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. Design, Codes and Cryptography 36, 33–43 (2005)
21. Blömer, J., Otto, M., Seifert, J.-P.: Sign change fault attacks on elliptic curve cryptosystems. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 36–52. Springer, Heidelberg (2006)
22. Yen, S.-M., Joye, M.: Checking before output not be enough against fault-based cryptanalysis. IEEE Trans. on Computers 49(9), 967–970 (2000)
23. Biehl, I., Meyer, B., Müller, V.: Differential Fault Attacks on Elliptic Curve Cryptosystems. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 131–146. Springer, Heidelberg (2000)
24. Fouque, P.-A., Lercier, R., Real, D., Valette, F.: Fault Attack on Elliptic Curve with Montgomery Ladder Implementation. In: FDTC2008, pp. 92–98. IEEE, Los Alamitos (2008)

25. Kim, C.H., Quisquater, J.-J.: How can we overcome both side channel analysis and fault attacks on RSA-CRT? In: FDTC 2007, pp. 21–29. IEEE, Los Alamitos (2007)
26. Joye, M.: On the Security of a Unified Countermeasure. In: FDTC 2008, pp. 87–91. IEEE, Los Alamitos (2008)
27. Joye, M., Ciet, M. (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 348–359. Springer, Heidelberg (2003)
28. De Mulder, E., Ors, S.B., Preneel, B., Verbauwhede, I.: Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem. In: EUROCON 2005, vol. 2, pp. 1879–1882. IEEE, Los Alamitos (2005)
29. Guo, X., Schaumont, P.: Optimizing the HW/SW Boundary of an ECC SoC Design Using Control Hierarchy and Distributed Storage. In: DATE 2009, pp. 454–459. EDAA (2009)
30. Guo, X., Schaumont, P.: Optimizing the Control Hierarchy of an ECC Coprocessor Design on an FPGA based SoC Platform. In: Becker, J., Woods, R., Athanas, P., Morgan, F. (eds.) ARC 2009. LNCS, vol. 5453, pp. 169–180. Springer, Heidelberg (2009)
31. Malkin, T.G., Standaert, F.-X., Yung, M.: A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 159–172. Springer, Heidelberg (2006)
32. Hwang, D., Tiri, K., Hodjat, A., Lai, B.C., Yang, S., Schaumont, P., Verbauwhede, I.: AES-Based Security Coprocessor IC in 0.18um CMOS with resistance to differential power analysis side-channel attacks. IEEE Journal of Solid-State Circuits 41(4), 781–791 (2006)
33. Chen, Z., Zhou, Y.: Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side-Channel Leakage. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 242–254. Springer, Heidelberg (2006)
34. Giraud, C.: An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. IEEE Trans. on Computers 55(9), 1116–1120 (2006)
35. Koschuch, M., Großschädl, J., Payer, U., Hudler, M., Krüger, M.: Workload Characterization of a Lightweight SSL Implementation Resistant to Side-Channel Attacks. In: Franklin, M.K., Hui, L.C.K., Wong, D.S. (eds.) CANS 2008. LNCS, vol. 5339, pp. 349–365. Springer, Heidelberg (2008)
36. Sakiyama, K., Batina, L., Schaumont, P., Verbauwhede, I.: HW/SW Co-design for TA/SPA-resistant Public-Key Cryptosystems. In: ECRYPT Workshop on Cryptographic Advances in Secure Hardware (2005)
37. Batina, L., Mentens, N., Preneel, B., Verbauwhede, I.: Balanced point operations for side-channel protection of elliptic curve cryptography. IEE Proceedings of Information Security 152(1), 57–65 (2005)