AES and ECC Cryptography Processor with Runtime Configuration

Samuel Antão, Ricardo Chaves, Leonel Sousa
Instituto Superior Técnico/INESC-ID
Technical University of Lisbon
Email: {sfan,rjfc,las}@sips.inesc-id.pt

***Abstract*—In nowadays society, the amount of applications that require cryptographic support keeps growing. The functionality and security of these applications rely on the capability of cryptographic accelerators in providing both adequate performance metrics while maintaining flexibility. In this paper a programmable cryptographic processor prototype, supporting AES and EC (Elliptic Curve) ciphering is presented. This processor consists of up to 12 programmable processing units. We explore and present results for the dynamic reconfiguration of these processing units, allowing the runtime replacement of AES by EC units (or vice-versa) according to the application needs. Combining programmability and runtime reconfiguration, both flexibility and performance can be improved. Moreover, the reconfiguration capability allows to further reduce the required hardware area, since these functionalities are multiplexed in time. The presented prototype is supported by a Xilinx XC4VSX35 FPGA, consisting of 6 static EC units and 6 reconfigurable AES/EC units, running simultaneously. This processor is able to cipher an 128 bit AES block in 22.9 μs and perform an EC point multiplication in 2.02 ms. The full reconfiguration of a processing unit can be achieved in less time than an EC multiplication.**

I. Introduction

Currently, most applications require security and authentication services. Several protocols have been designed to provide such requirements to these applications, being used in a variety of devices: from smart cards, wireless sensors, cell phones, and laptops, that usually need a small amount of connections, to high-end servers that have to establish thousands of connections. For such a wide variety of devices, there is also a wide range of different demands. The following highlights the key features that have to be considered:

- performance, supporting high-throughput and low latency;
- low cost, using cheap platforms and massive production computing elements;
- compactness, allowing the coexistence of different applications in a small pool of resources;
- flexibility, allowing adjustment to different needs;
- low power, enhancing the battery savings, reducing the costs in energy and heat sinks, and increasing autonomy.

The security and authentication protocols are often supported by two main types of cryptographic functions: symmetric and asymmetric. The latter allows to establish a secure and confidential communication between two entities that share a secret, while the former allows two entities to create a distributed secret without any previous agreed information. Several algorithms have been proposed to implement these cryptographic functions, and the most successful ones have been adopted regarding their strength against attacks, and their compatibility with the performance-and-compact demand [1], [2].

Regarding the asymmetric algorithms, the Elliptic Curve (EC) cryptosystem has emerged as a reliable and effective alternative for the widely used Rivest-Shamir-Adleman (RSA) algorithm. The EC cryptosystems have the advantage of providing a greater security per bit of the secret key. Therefore, smaller keys need to be used/stored. Consequently, more compact and bandwidth efficient, since smaller keys need to be transmitted, cryptosystems can be developed.

Although symmetric algorithms do not offer the same properties as asymmetric ones in terms of the secret key construction, they are simpler, more compact, and more efficiently computed, allowing for better area and throughput metrics. Thus, its usage is mandatory for some applications. Currently, one of the most widely used algorithms for symmetric cryptography is the Advanced Encryption Standard (AES) [2].

Although both symmetric and asymmetric algorithms have shown to provide good performance metrics, their complexity is still considerable. To overcome this problem, hardware accelerators are employed. Several accelerators have been proposed supported on Application Specific Integrated Circuit (ASIC) solutions [3], Field Programmable Gate Array (FPGA) [4], Graphical Processing Unit (GPU) [5], [6], and Instruction Set Architecture (ISA) extensions for general purpose processors [7]. While the flexibility of the solutions increase when we move from the ASIC to the general purpose solutions, the performance decreases. The ASIC approach allows for fast and low power solutions, but with limited adaptability and higher design costs. General purpose processors solutions allow for optimal programmability, but achieve relatively low performances and higher power costs. The GPU solutions allow for the utilization of a large amount of parallel hardware structures with a reduced cost, because of the massive production due to the gaming market. However, the GPU's datapath is not optimized for cryptographic procedures and the parallelism extraction for cryptography is limited, allied with the significant power consumption. The FPGA solutions are a compromise between the high performance/low power of the ASIC and the flexibility/low cost of the general purpose

processors. Moreover, FPGAs allow to combine programmable solutions with reconfiguration capabilities, providing adaptable datapaths. FPGAs can be considered as an advised option to efficiently support a wider range of cryptographic algorithms and procedures.

This paper proposes a general cryptographic processor supported on FPGA. This programmable processor was designed to take advantage of the reconfigurable capabilities of an FPGA for achieving good performance metrics and enhanced flexibility. The processor proposed in this work aims to provide support for the majority of the security and authentication protocols, introducing microcoded AES and EC cores, and a true Random Number Generator (RNG) supported on oscillator rings to generate secrets. Very few attempts have been made in the related art to combine AES and EC arithmetic into a single arithmetic body. The efficiency of such approaches is compromised by the difference in the size of the datapath ($m \geq 163$ for the EC versus $m = 8$ for the AES), requiring the use of different irreducible polynomials, thus different reduction structures. Our approach is different: instead of sharing the datapath for the AES and EC arithmetic, we create individual, compact and high-performance AES and EC cores that share the same microcoded control unit. With this approach and using the reconfiguration capabilities of the FPGA, it becomes very easy and efficient to dynamically trade AES and EC cores, depending on the requirements. A compact and flexible cryptographic processor with good performance metrics is obtained. With a RNG associated to the processing units the secret keys of the protocols can be locally computed and directly stored in the processing units' memory. Avoiding the communication of secret keys makes the system more secure and resistant to external attacks.

The paper is organized as follows. In Section II we provide a brief introduction on the AES and EC arithmetic. In Section III we present the details of the reconfigurable architecture used. In Section IV we describe the system layout in order to handle the runtime configuration of processing units. Section V presents results for the developed prototype, and Section VI draw some conclusions about the developed work.

II. AES and EC Cryptography

In this section we briefly introduce the arithmetic that the proposed processor supports.

A. AES arithmetic

The AES algorithm is composed by three main operations: the key expansion, the ciphering, and the deciphering. In the key expansion operation, the used key, with 16, 24 or 32 bytes, is expanded in order to obtain 176, 208, or 240 bytes, depending on the initial size. This expanded key is divided in sets of 16 bytes and each set is used in each round of the ciphering/deciphering operation. The number of rounds depends on the used key size. The key and data used in the ciphering/deciphering rounds are organized in a common way: in a 4×4 bytes matrix. Each AES round affects each of these matrices' elements using the following elementary operations:

- byte additions over $GF(2^8)$, which correspond to a 8-bit bitwise exclusive OR (XOR) operation;
- non-linear function $S(.)$ often called an SBox and its inverse; this function can be computed with multiplications and inversions over $GF(2^8)$ with the irreducible polynomial $I(x) = x^8 + x^4 + x^3 + x + 1$;
- data matrix multiplication with constant matrices, with the irreducible polynomial $I(x)$;
- matrix row rotating shift operation.

Further details about these operations and how they are applied can be found in [2].

B. EC arithmetic

An EC over $GF(2^m)$ is a set composed by a point at infinity $\mathcal{O}$ and the points $P_i = (x_i, y_i) \in GF(2^m) \times GF(2^m)$ that comply the following equation:

$$y_i^2 + x_i y_i = x_i^3 + a x_i^2 + b,\ a, b \in GF(2^m). \quad (1)$$

By establishing the addition operation over the EC points and by applying it recursively, it is possible to obtain the multiplication by a scalar operation. It is known to be computationally hard to invert this operation, since it is difficult to determine, from the recursive addition result of an EC point, how many times this point was added. This is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP), which supports the security of EC cryptosystems.

The EC point addition and doubling (addition to itself) are performed with operations over the underlying field $GF(2^m)$ applied to the points' coordinates. These $GF(2^m)$ operations are the addition, multiplication, squaring and the inversion, modulo an irreducible polynomial with degree m. Details about how these operations can be efficiently performed can be found in [8].

III. Cryptographic Processor Architecture and Details

In this work we developed a prototype of a cryptographic accelerator supported on reconfigurable hardware, namely a prototyping board powered by a Xilinx Virtex 4 FPGA [9]. In this prototype the aim is to support the majority of protocols that need asymmetric and symmetric cryptographic schemes, and also the secure generation of secret keys for these protocols. A schematic overview of the proposed processor organization is presented in Figure 1. The processor is composed by several processing units (PUs), responsible for computing the cryptographic procedures. An RNG is also included in order to generate the secret data (such as the private keys). The processor has an I/O interface to communicate and receive the data (public keys, plain texts, ciphered texts) to/from the main controller, which we herein call host of the processor. This interface is also used to provide commands, such as start commands for the processing units (PUs) or write/read commands of data and instructions. Through this interface the host can query the processor for, e.g., availability of PUs or check if the required tasks were already done. When the host sends a write command to any PUs, it also defines the origin

of the data to be written, namely external data or internal data read from the RNG. Thus, the host can use the secret information without having to touch or to know it.

All the PUs are responsible to run according to the microcode stored in a centralized instruction memory. For this, each PU has its own microprogram counter (μPC) and startup addresses to run and control the flow of the correct program. An arbiter controls the access to the instructions memory according to a priority policy, and signals any PU when the memory retrieves a valid microinstruction for it.

Each PU contains its local data memory, which is addressed according to the received microinstructions. Input data and temporary data, as well as the final results are stored in this local memory. This memory can be accessed by the host, when the PU is set to the idle state through specific microinstructions directly provided by the host, in order to be possible to read and write data from/to the PUs. The width of the data memory as well as the details of the arithmetic units available is customizable according to the type of the PU. Different types of PUs support different cryptographic procedures.

With this modular architecture, the PUs share the same control through the instruction memory while facilitating the replacement of a given PU by another one. This allows to extract full advantage of the reconfiguration capabilities of the electronic devices.

A. PU for AES

The architecture of the AES PU is presented in Figure 2a. This architecture is composed by a data RAM of 512 positions, a ROM and two adders. The ROM implements a look up table for the non-linear function $S(.)$ and its inverse $S^{-1}(.)$ (see Section II-A). We also include in this ROM the operations $2S(x)$, $3S(x)$, $9x$, $11x$, $13x$, and $14x$, where x represents the ROM address. With these operations, we are able to perform the multiplications with the constants matrices operations.

Since the computation of the AES is performed over $GF(2^8)$ the used datapath and memory width is of 8 bits. Regarding that x has 8 bits, the ROM has 2048 entries of 8 bits. This amount of data fits a single BRAM present in the Xilinx Virtex 4 technology. Furthermore, since these BRAMs are dual

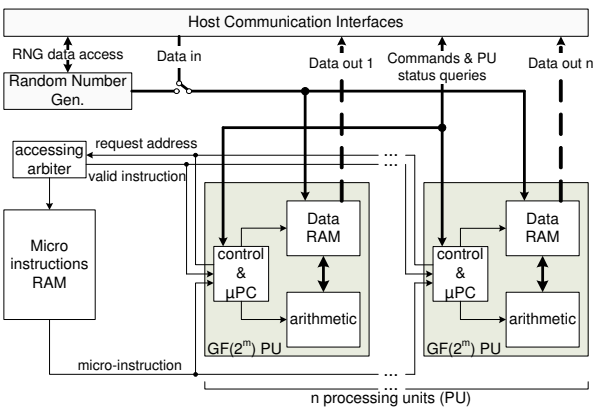
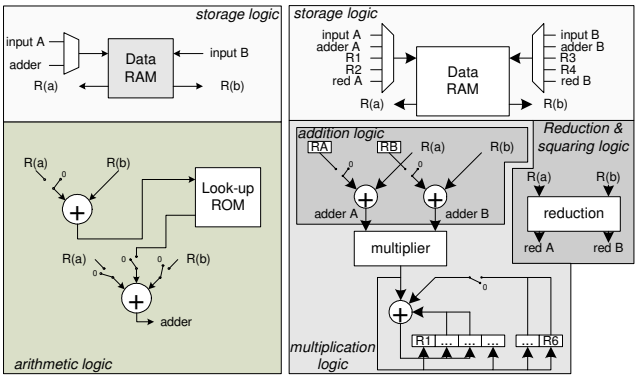


Fig. 1: Processor Organization Overview.



(a) AES PU (b) EC PU

Fig. 2: Architecture of the processing units.

port, the same ROM can be used for two PUs. The two adders at the input and output of the ROM perform the required additions for the AES arithmetic. With this architecture the following basic operations are implemented, where $L(.)$ is a look-up result: $R(c) = R(a) + R(b)$, $R(c) = R(a) + L(R(b))$, $R(c) = L(R(a) + R(b))$, and $R(c) = L(R(b))$, where a, b, and c are the addresses provided by the microinstruction. An operation to load a constant directly to the memory is also implemented. The byte shift operations can be overcome with the appropriate addressing of data, since each address correspond to one byte. Regarding the flow control of the program, three jump related operations are implemented:

- `jmpset`: set the value of an indexing counter;
- `jmpinc`: jump if the value associated to this jump instruction match the value in the indexing counter; the indexing counter is incremented;
- `jmpdec`: similar to `jmpinc`, but decrements the indexing counter;
- `end`: determines the end of the program, and thus the PU becomes idle.

It is also possible to sum the value in the indexing counter multiplied by 16 to the data addresses. This allows to easily browse through the 16 byte matrices where the AES data is organized in the data BRAM depending on the indexing counter. All these functionalities, including the choice of the ROM look-up and the usage of the indexing counter data in the addresses, are mapped in microinstructions of 36 bit width. Each microinstruction run in 3 clock cycles: one cycle to read the data, one cycle to read the ROM, and another cycle to write the result.

B. PU for EC

We support the EC PU in our previous work presented in [8] for polynomial basis field arithmetic. The architecture of this compact and flexible PU is similar to the one of the AES PU, and is depicted in Figure 2b. There is also a data BRAM where the field elements (of size $m \geq 163$) are split and stored in 21 bit words. The arithmetic logic supports two-

word with two-word operand additions and Karatsuba-Offman multiplications.

The microcode adopted for the EC PU can be classified into two main microinstruction types. The complex microinstructions (type I) are performed over field elements, while the lower complexity microinstructions (type II) operate over words. There is a type I reserved microinstruction that corresponds to a customizable sequence of type II operations.

Type I instructions are used to access the memory (`read` and `write`) and the key register (`key`) to compute the m bit add, squaring and reduction operations (`add`, `sqr` and `red`), and to control the flow by conditionally jumping to a microinstruction address depending on the key register or by turning the Processing Unit (PU) to an idle state (`jmp` and `end`). The type II instructions allow for adding and multiplying 2-word operands (`eadd` and `emult`). An instruction determines the end of a type II sequence (`eret`) and, consequently, the end of the `pers` type I instruction. A customizable instruction (`pers`) is also reserved, corresponding to a user-defined sequence of type II instructions.

Another jump instruction is also introduced. When a PU is placed in the architecture, an ID is assigned to it. This jump instruction, called `jumpid`, is an unconditional jump operation, and is only executed if the ID in the microinstruction match the ID assigned to the PU. If the IDs do not match, this instruction is ignored. Introducing this instruction allows a program to use microcode segments of other programs, since this instruction works as a return instruction that is only considered by a PU running a specific program. This is useful to shrink the program sizes by running a single routine needed in different programs, e.g. the inversion in the scalar multiplication and in the point addition. This unit also supports an instruction that signals the end of the program.

The functionality provided by the EC PU can be controlled by microinstructions of 32 bit width. Since the AES core need 36 bit width instructions, the EC PU also uses 36 bit coded instructions, by ignoring the 4 most significant bits. Regarding the clock cycles required for the instructions, the jump and word size addition needs 3 cycles, the word size multiplication needs 5 cycles, the field size addition needs 13 cycles, and the reduction and squaring operations need 14 clock cycles.

C. Arbiter

The arbiter controls the access to the microinstruction memory when there are simultaneous and pending requests. The arbiter considers a static priority for each PU, where all the EC PUs have higher priority than the AES PUs. This is because the EC programs possess microinstructions that take a larger amount of clock cycles comparing with the AES microinstructions. Thus, it is more likely the AES microinstructions to efficiently fill the clock cycles between the EC requests, than the opposite, resulting in a better efficiency of the whole system.

D. True Random Number Generator

A true RNG is also included to generate the secrets that lead to the private keys. Hence, since the private keys are not communicated by the device, there is no entity, other than the host, external to the device capable of achieving them, at least without implementing sophisticated attacks, such as Differential Power Attacks [10].

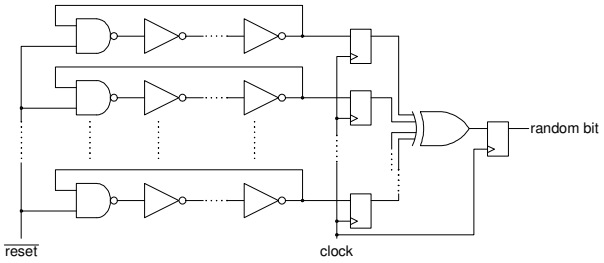


Fig. 3: Random bits generator.

The randomness source of the RNG is the jitter of an oscillator. In a digital device, such as an FPGA, these oscillators can be obtained with combinatorial rings of an odd number of logical inverters. To obtain a random bitstream, we can implement several of these oscillators, obtain the logic exclusive *OR* for all the outputs of each oscillator, and sample the obtained signal with a frequency lower than the frequency of the oscillators [11]. An FPGA implementation of such RNG was reported in [12] for an Altera Cyclone II FPGA. The authors in [12] suggest an improvement to the method presented in [11]. They suggest to sample the output of each oscillator prior the exclusive-*OR* operation. This suggestion is based on the observation that the combinatorial logic responsible for computing the exclusive-*OR* operation may not have enough commutation speed between events at the inputs. For the RNG designed in this paper we followed this suggestion, which resulted in the circuit presented in Figure 3. We also introduced a reset signal in order to halt the oscillators and the random bitstream generation, in order to reduce the power consumption when the RNG is not being used. A shift register was padded to the output of the RNG to store the random data and allow it to be readily read.

IV. Runtime Reconfiguration

The proposed processor is specially designed to efficiently support runtime reconfiguration. The modularity of the processor allows to easily configure different processing units without affecting the behavior of the others. This allows to fulfill the runtime needs of the host by better adapting the computation to the protocols being used. Our design is supported by a Xilinx Virtex 4 FPGA, allowing for the Xilinx dynamic reconfiguration flow for this processor.

The only concern regarding the control of the dynamic reconfiguration is related with the dummy requests placed in the arbiter by the PU under reconfiguration, due to the unexpected behavior of the PUs outputs during reconfiguration. To overcome this issue, the architecture contains an enable register that can be accessed by the host. When the host disables the PU that is going to be reconfigured, the valid requests of that PU are ignored by the arbiter. After the

reconfiguration, when the host enables the PU, a reset pulse is generated for that PU to set it to the idle state.

In order to support both AES and EC processing units, the reconfigurable zones should cover the resources required by the most demanding implementation loaded in that zone. Between the two considered PUs, the most demanding in terms of resources is the EC PU, due to the wider datapath (21-bit instead of 8-bit) and larger complexity. For these reasons, the reconfigurable zones are sized to fit an EC PU.

Since the several PUs compete to access the instruction memory, conflicts can exist, thus some PUs may stall waiting for their request to be fulfilled. These conflicts penalty will increase if the number of PUs appended to one of the instructions memory port increases. This effect has to be taken into account when setting the number of PUs in the design and, consequently, the number of reconfigurable zones. Each of the AES operations requires 3 clock cycles to perform, while an EC operation requires from 3 to 14 cycles to perform. This means that the average of clock cycles per instruction in the AES PUs is less than the EC PUs average. Thus, the AES PUs will generate more conflicts than the EC PUs. Since the arbiter can issue one instruction per clock cycle, only a maximum of 3 AES PUs can ideally operate at the same time without conflicts. Putting a forth PU with less priority than the others will cause this fourth PU to stall until one of the others finishes the ongoing computation. This observation determines the number of the required reconfigurable zones, which is 3 per instruction memory port. Thus, the system can have up to 3 AES PUs per instruction memory port, implemented in the reconfigurable zones. The system can have more static EC PUs according to the conflicts that the user admits or to the available resources. Considering a dual port instructions memory, the number of reconfigurable zones can be increased to the double, 6.

The use of dual port memories also contributes to reduce the resources used in the design of the AES PUs. Considering a dual-port look-up ROM the same memory can be implemented statically outside the PUs and shared by two AES PUs, as Figure 4 suggests. Moreover, this procedure allows for the information inside the RAM not to reside among the configuration data, enhancing the compactness of the bitstream and the configuration speed. Another issue that as to be considered while reconfiguring the PUs is the amount of signals that cross the reconfigurable zone boundary, since the path of these signals through the boundary has to be directly instantiated. This instantiation, except for the clock signal when provided by a global buffer, is performed recurring to directional slice bus macros. These bus macros are provided with the Xilinx ISE tools that support dynamic reconfiguration. The number and type of the required macros is determined by the number of PU inputs and outputs signals. Each bus macro occupies a Configurable Logic Block (CLB), which correspond do 4 slices, and supports up to 8-bit signals. To determine the number of bus macros, the maximum number of inputs and maximum number of outputs in both the PUs types (AES and EC) have to be considered. For the proposed design a maximum of 89 input signals ($\lceil 89/8 \rceil = 12$ bus macros) and 64 output signals ($\lceil 64/8 \rceil = 8$) is required, corresponding to a total of 20 bus macros.

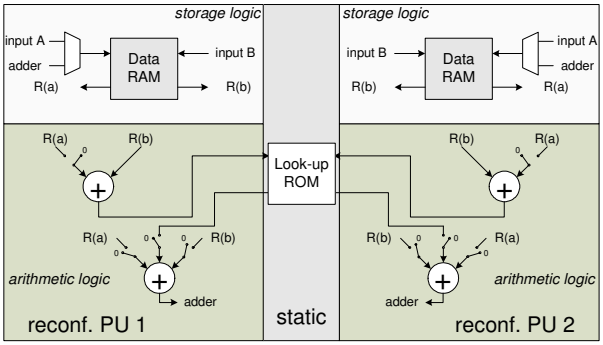


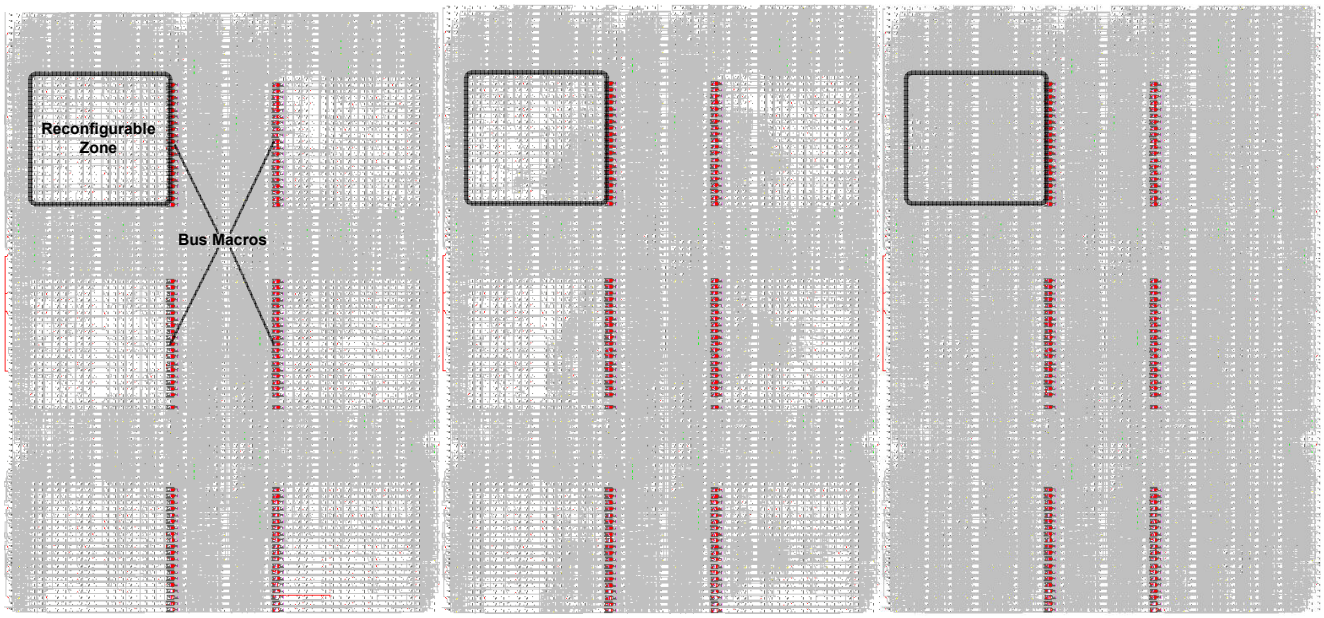
Fig. 4: AES PUs with shared look-up memory.

V. Experimental Results and Related Work

The proposed design was successfully implemented and experimentally tested on a prototyping board powered by a Xilinx XC4VSX35-10 FPGA [9]. We implemented and evaluated different combinations in the number of AES and EC cores. These implementations refer to EC arithmetic over $GF(2^{163})$ and AES arithmetic with 128 bit key size. The FPGA programming files were obtained from a VHDL description of the hardware, synthesized with the Synplify Premier C-2009.06 tools and Placed&Routed with the ISE 9.2.04i_PR14 tools. The Virtex 4 technology supports the handling of dynamic reconfiguration using the Internal Configuration Access Port (ICAP). The advantage of using this port is the possibility to direct instantiate and conjugate it with the remaining design, including the communication logic, that can write the reconfiguration bitstream directly to this port.

The Virtex 4 FPGA contain block RAMs that provide true dual port capabilities. This allows for all the memories employed in the design (instruction, data, and look-up) to be dual port, saving resources. As discussed in Section IV, the maximum number of AES PUs competing for an instruction memory port can be up to 3. Thus, we will use 6 reconfigurable zones that can be reconfigured with an EC or AES PU. We also implement another 6 (3 per instruction BRAM port) static EC PUs. Thus the design can have up to 12 PUs working simultaneously, and up to 6 PUs can be AES PUs. The reason for implementing only 6 static PUs is related with the Slice resources constraint and with the increasing number of conflicts while accessing the instruction BRAM. We considered an instruction memory with 1024 36-bit instructions to contain all the routines for EC and AES arithmetic.

The static design contains the required logic to implement the communication with the host, the random numbers generation, the AES look-up memories, and the 6 static EC PUs. The required resources to implement the static design are 8,446 slices and 11 BRAMs (2 for the instruction memory, 3 for the AES look-up ROMs, and 6 for the data storage in the 6 static EC PUs).



(a) No reconfigurable PUs (b) AES PUs only (c) ECC PUs only

Fig. 5: Processor layout with different reconfigurations.

There are 6 reconfigurable zones in the design with rectangular shape of 13 CLB width and 21 CLB height ($13 \times 21 \times 4 = 1092$ total slices). Considering the size of the Virtex 4 configuration frames which have 1 CLB width and 16 CLB height, the reconfiguration of a reconfigurable area requires the communication of 26 frames. The different reconfigurable zones do not intercept reconfiguration frames of the others. For this, the bottom boundaries of the reconfigurable zones are at the CLB coordinates 0, 32, and 64 (slices 0, 64, and 128). The layout of the system, as well as the bus macros location, is depicted in Figure 5 for different contents of the reconfigurable zones after place and routing. Each PU employs 1 BRAM for its data memory. The reconfigurable AES PUs requires 157 ± 2 slices and the EC PU requires 943 ± 7 slices. The variation in the slices resources employed by each PU is due to the slightly different placing of the resources by the tools for the different reconfiguration zones. The occupation of the reconfigurable zones by the PUs is 14% and 86% for the AES and EC PUs, respectively. Although the occupation of the reconfigurable zones is not complete, the margin of free resources allow to enhance the routing delays. Regarding the complete design, the required resources are 14,092 slices (92% of the total resources) with all the reconfigurable zones implementing EC PUs and 9,387 (61% of the total resources) with all the reconfigurable zones implementing AES PUs. Considering the reconfigurable zones completely occupied, the required resources for the complete design are 14,998 slices (98% of the complete resources). The obtained system can run at the maximum frequency of 100.3 MHz.

The reconfiguration bitstreams were generated in compressed format, using the appropriate Xilinx tools options. The maximum and minimum size in 32-bit words of the runtime reconfiguration bitstreams are 30662 and 31067 for EC PUs, and 27898 and 29500 for the AES PUs. Although the reconfiguration area is the same for the AES and EC PUs, the AES PUs result in approximately 5% smaller bitsreams due to the lower utilization of resources, allowing for a slightly higher compression. The reconfiguration time is directly correlated with the bitstreams and the clock frequency. The ICAP in Virtex 4 technologies allow to write a 32-bit reconfiguration word in each clock cycle. The maximum ICAP working frequency is 100 MHz [13], thus we expect that the maximum reconfiguration time can be of $31067/100\ MHz \approx 310\mu s$. Although, in the developed prototype, the reconfiguration bitstream is communicated from outside the device and written directly in the ICAP, being the reconfiguration time limited by the communication process. The communication is performed through a PCI bus, working at 33 MHz. Hence, we use the same bus frequency driving the ICAP, being the incoming data immediately transferred to the ICAP. With this, we obtain a maximum reconfiguration time of $31067/33\ MHz \approx 941\mu s$.

In the next subsection we present the results specific for the RNG and PUs operation.

A. Random Number Generator

In order to validate the implementation of the RNGs, random bitstreams were collected from the processor and their randomness tested using a battery of tests. Two main batteries of tests are used for this purpose: the National Institute of Standards and Technology (NIST) test [14] and the Diehard one [15]. For the implementation proposed in [12], in order to pass both batteries of tests successfully, the authors obtained a RNG with 25 oscillators of 3 inverters each, sampled at

100 MHz. The option of using 3 inverters is justified by the enhanced compactness of the implementation.

The randomness of the bitstream is enhanced if the number of oscillators increase or/and the sampling frequency decrease. For the processor herein presented, using 3 inverters per oscillator, the number of required oscillators to pass both NIST and Diehard tests at 100 MHz, which is the operating frequency for the prototype, was shown to be 20. Each oscillator is implemented within a CLB, resulting in a very compact RNG.

B. Processing Units

Using the proposed architecture and microcode format, we were able to program the EC scalar multiplication and point addition in 401 instructions, and the AES key expansion, ciphering and deciphering in 253 instructions. The total latency for the EC PUs is 201,661 clock cycles for the EC scalar multiplication and 4,796 clock cycles for the point addition. The latency for the key expansion and ciphering/deciphering in our AES PU is 610 clock cycles and 2,290 clock cycles, respectively.

Performance metrics for different combinations of simultaneously working PUs in the cryptographic processor are presented in Table I. This metrics are measured at the prototype operating frequency, 100 MHz.

The evaluation in Table I use 1 EC scalar multiplication and 88 consecutive AES ciphering operations, because the time consumption of one individual EC point multiplication is approximately the time of 88 AES operations, allowing a fair analysis. Although the instruction memory has two ports, we focus our analysis on a single arbiter individually, thus one of the instruction memory ports. This analysis hold for both arbiters, even if the configuration of the PUs attached to them is different.

An EC point multiplication produces a result in 2.02 ms if no conflicts occur, thus the proposed design provides a throughput of 496 Op/s for only one PU. For 6 EC PUs running simultaneously, the throughput is of 1,536 Op/s, which is lower than 6 times the throughput for one PU, due to the conflicts accessing the instructions. Performing the same analysis for the AES arithmetic, considering the ciphering of 128 bit blocks, the proposed processor provides a throughput from 5.6 Mbit/s for 1 PU to 16.8 Mbit/s for the 3 PUs. In this case, the throughput of the system scales directly with the number of PUs, since all the instructions for the 3 the AES PUs competing for the instruction memory take the same 3 clock cycles, thus no conflicts will occur. Intermediary configurations can be useful for the dynamic requirements of the host.

We also introduce an efficiency metric in Table I. This efficiency measures the impact of the request conflicts solved by the instruction memory arbiter. This efficiency measures the ratio of time used for useful computing by all the operating PUs within a specific time interval. To perform this efficiency measurement we programmed all the PUs to run consecutively the same program, and after a specific time interval T measured in clock cycles the number of complete EC (n_{EC}) and AES (n_{AES}) operations were counted. The efficiency (E) is given by:

$$E = \frac{n_{EC}T_{EC} + n_{AES}T_{AES}}{n_{PU}T}; \quad (2)$$

where T_{EC} and T_{AES} is the time of a single EC and AES operation without conflicts in the memory accessed measured in clock cycles, respectively, and n_{PU} is the number of the active PUs. From Table I, it can be observed that the efficiency is very close to 100% for configurations with less than 4 PUs. This result arose from the fact that an instruction takes at least 3 clock cycles to complete, thus the number of conflicts in the arbiter will be meaningless. Moreover, for the other configurations the efficiency is always greater than 61%.

Comparing the presented results with the related work is not straightforward, since different technologies and different metrics are used by different authors. Nonetheless, we introduce some related art results to comparatively evaluate our design.

In [4] a compact AES/EC design is proposed, supported on a Xilinx Virtex XCV800 platform running at 41 MHz. Several Logical Units (LUs) that support the basic field operations over $GF(2^8)$ are organized by two reconfigurable modes: a Single-Instruction-Multiple-Data (SIMD) mode that support the AES arithmetic, and a Single-Instruction-Single-Data (SISD) mode that supports the EC arithmetic. This design does not support simultaneous EC and AES arithmetic, since the LUs must be reconfigured to reuse resources. This design offers a throughput of 3.8 Mbit/s for the AES ciphering (128 bit key), and a Point multiplication (in $GF(2^{163})$) latency of 5.36 ms. Our AES throughput when using one PU is 5.5 Mbit/s (1.4 times higher) and the latency for the EC point multiplication is 2.02 ms (2.65 times lower). The design in [4] occupies 220K gates (approx. 2329 Slices), which is 2.1 times more than one reconfigurable zone in our design. In [4], the sharing of the datapath between the AES and EC results in the splitting of an operation in smaller ones, when these operations could be more efficiently computed in dedicated hardware or using look-up tables. This could justify the lower performance metric of this design.

In [3] a 0.18μm ASIC solution operating at 100 MHz is proposed. In this solution the AES and EC arithmetic share most the multipliers and registers. With 56K gates, the authors in [3] state that a throughput of 64 Mbit/s for the AES, and a latency of 1.8 μs for a field multiplication can be achieved. Considering that 983 field multiplications and 650 squaring operations are required for implementing the EC multiplication algorithm, we estimate that the EC point multiplication latency would be >2,9 ms. The herein proposed design is able to perform the EC point multiplication 1.4 times faster. Although our AES throughput is lower, our design can operate AES and EC simultaneously and offer a flexibility and programmability that an ASIC solution can not.

In [16], a compact solution for AES supported by a Xilinx XC2S15 FPGA running at 67 MHz is proposed. This design is supported by two main arithmetic units: a multiply accumulate, and a byte substitution unit, to support the non-linear function

TABLE I: Performance metrics for different combinations of simultaneously working PUs.

# ECC PUs	# AES PUs	Latency (K clk cycles)	Latency ms	ECC throughput (Op/s)	AES throughput (Mbit/s)	Efficiency (%)
0	0	-	-	-	-	-
1	0	201.7	2.02	496	-	100.00
2	0	201.7	2.02	992	-	100.00
3	0	201.7	2.02	1488	-	100.00
4	0	342.3	3.42	1169	-	82.50
5	0	344.9	3.45	1450	-	71.60
6	0	390.5	3.91	1536	-	61.67
0	1	201.5	2.02	-	5.59	99.98
1	1	206.9	2.07	483	5.44	99.08
2	1	223.5	2.24	895	5.04	96.61
3	1	348.8	3.49	860	3.23	81.80
4	1	354.5	3.55	1128	3.18	71.24
5	1	391.4	3.91	1278	2.88	61.59
0	2	201.5	2.02	-	11.18	99.98
1	2	208.1	2.08	481	10.83	98.57
2	2	348.7	3.49	574	6.46	79.27
3	2	350.3	3.50	856	6.43	70.72
4	2	385.9	3.86	1037	5.84	61.20
0	3	201.5	2.02	-	16.77	99.98

required in the AES. These units are controlled by microinstructions and a microprogram counter controls the program flow and branches. This design achieves a throughput of 2.2 Mbit/s occupying 124 slices and 2 BRAMs. Our AES PU offers a throughput 2.5 times higher with 1092 slices allocated for its reconfigurable zone and 4 BRAMs. These 4 BRAMs are the minimum required for an AES PU to operate in the herein proposed design.

VI. Conclusions

In this paper, a microcoded and customizable cryptographic processor prototype is presented, capable of efficiently computing the AES and EC algorithms, as well as the generation of secrets through a RNG. The adopted approach relies on efficient and compact EC and AES processing units that share the same control from a central microinstruction memory, allowing simultaneous computing of AES and EC routines. With this processor, customization can be performed by adding processing units according to the processing needs. Additional configuration can be achieved in runtime through the dynamic reconfiguration capabilities of the FPGA. These characteristics make this processor highly adaptable and flexible. The reconfiguration time for a single PU is smaller than an EC multiplication, resulting in negligible impact in the system performance if several reconfigurations need to be performed. The proposed processing units, that provide the computing power of the processor, have shown to be very compact and suitable for embedded systems, supporting AES and EC with configurations fitting reconfiguration zones of 1092 slices each, and throughputs up to 1536 Op/s for EC and 16.8 Mbit/s for AES. Another advantage of the proposed processor is the inclusion of a compact true RNG in the architecture. This true RNG allows for the internal generation of secrets (such as private keys), thus enhancing the system security.

References

[1] N. I. of Standards and Technology, "Federal Information Processing Standards Publication 186-3: Digital Signature Standard," June 2009.

[2] ——, "Federal Information Processing Standards Publication 197: Advanced Encryption Standard," November 2001.

[3] J. Wang, X. Zeng, and J. Chen, "A VLSI implementation of ECC combined with AES)," *Proc. 8th International Conference on Solid-State and Integrated Circuit Technology*, pp. 1899–1904, March 2006.

[4] W. Lim and M. Benaissa, "Subword parallel GF(2^m) ALU: an implementation for a cryptographic processor," *Proc. IEEE Workshop on Signal Processing Systems*, pp. 63–68, Aug. 2003.

[5] R. Szerwinski and T. Guneysu, "Exploiting the Power of GPUs for Asymmetric Cryptography," *Proc. Workshop on Cryptographic Hardware and Embedded Systems CHES*, pp. 79–99, Aug. 2008.

[6] S. Manavski, "CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography," *Proc. IEEE International Conference on Signal Processing and Communications*, pp. 65–68, Nov. 2007.

[7] O. Kocabas, E. Savas, and J. Grossschadl, "Enhancing an Embedded Processor Core with a Cryptographic Unit for Speed and Security," *Proc. International Conference on Reconfigurable Computing and FPGAs*, pp. 409–414, Dec. 2008.

[8] S. Antão, R. Chaves, and L. Sousa, "Compact and Flexible Microcoded Elliptic Curve Processor for Reconfigurable Devices," *Proc. 7th IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM*, March 2009.

[9] Annapolis Micro Systems, Inc., *Wildcard 4 Summary Description*, 2007, http://www.annapmicro.com/wc4.html.

[10] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Proc. 19th Annual International Cryptology Conference, Advances in Cryptology, CRYPTO*, vol. 1666, pp. 388–397, 1999.

[11] B. Sunar, W. Martin, and D. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on computers*, vol. 56, no. 1, p. 109, 2007.

[12] K. Wold and C. Tan, "Analysis and Enhancement of Random Number Generator in FPGA Based on Oscillator Rings," *Proc. International Conference on Reconfigurable Computing and FPGAs, REConFig*, pp. 385–390, 2008.

[13] Xilinx, Inc., *Virtex-4 FPGA Data Sheet: DC and Switching Characteristics, version 3.7*, 2009, http://www.xilinx.com/support/documentation/data_sheets/ds302.pdf.

[14] N. I. of Standards and Technology, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, Special Publication 800-22, Revision 1," August 2008, http://csrc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf.

[15] G. Marsaglia, "Diehard Battery of Tests of Randomness," 1995, http://stat.fsu.edu/pub/diehard/.

[16] T. Good and M. Benaissa, "AES on FPGA from the Fastest to the Smallest," *Proc. Workshop on Cryptographic Hardware and Embedded Systems CHES*, pp. 427–440, September 2005.