AN RNS-BASED ARCHITECTURE TARGETING HARDWARE ACCELERATORS FOR MODULAR ARITHMETIC

Samuel Antão and Leonel Sousa

INESC-ID, Department of Electrical and Computer Engineering
Instituto Superior Técnico - Universidade Técnica de Lisboa - Lisbon, Portugal
Email: {samuel.antao,las}@inesc-id.pt

ABSTRACT

This paper proposes and discusses an architecture with scalability features for the parallel implementation of algorithms relying on modular arithmetic fully supported by the Residue Number System (RNS). The systematic mapping of a generic modular arithmetic algorithm to the architecture is presented. It can be applied as a high level synthesis step for an Application Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA) design flow targeting modular arithmetic algorithms. An implementation with the Xilinx FPGA Virtex 4 technology (xc4vsx55) of modular exponentiation and Elliptic Curve (EC) point multiplication, used in the Rivest-Shamir-Adleman (RSA) and EC cryptographic algorithms, suggests latency results in the same order of magnitude of the fastest hardware implementations of these operations known to date.

***Index Terms*—** Residue Number System (RNS), Modular Arithmetic, Cryptography, Embedded Systems, Electronic Design Automation (EDA).

1. INTRODUCTION

Modular Arithmetic (MA) can be found in a variety of applications including cryptography [1, 2]. Hardware accelerators for such applications may completely rely in MA or in some blocks that apply modular operations. Hence, for the efficiency sake, the designer of such systems should not only be aware of the best system design practices but also of the mathematical details concerning modular arithmetic. While the software solutions based on general purpose processors provide flexibility, these solutions may not be attractive in what concerns latency, cost, and power consumption. An approach to overcome these problems is to develop dedicated hardware accelerators for the most demanding operations. These accelerators should be designed such that the flexibility and potential reuse of the already designed blocks is not constrained.

An approach to obtain fast and efficient accelerators is parallelization. The Residue Number System (RNS) concept has been successfully tested towards low latency computation in highly parallel architectures such as the ones of Graphical Processing Units (GPUs), while targeting hundreds-of-bits wide data for cryptographic purposes [3–5]. The RNS enables an alternative representation of the operands, creating data parallelism even for algorithms with several data dependencies. However, the utilization of RNS also inserts computational overheads that should be balanced with the gains obtained from the parallelization so that performance can be maximized.

This paper proposes and discusses a microprogrammable architecture which: *i*) supports the implementation of efficient parallel RNS-based accelerators; *ii*) eases the mapping of any general modular arithmetic algorithm to the RNS arithmetic; *iii*) can be easily scalable to different algorithms and performance demands; and *iv*) enable tuning the balance between parallelism and performance overhead. The proposed architecture can be described by generic parameters and is regular, which allows one to use an high-level synthesis approach to obtain accelerators for any given modular arithmetic algorithm. A case study for two main applications of modular arithmetic, namely the modular exponentiation and the Elliptic Curve (EC) point multiplication employed in cryptographic protocols, is presented, and evaluated with Field Programmable Gate Array (FPGA) technology. Results suggest competitive performance figures regarding the fastest dedicated implementation proposed in the literature for the same technologies.

The paper organization is the following. Section 2 presents the details of the RNS arithmetic and the implementation of modular arithmetic with the RNS. Section 3 presents how the proposed architecture relate to previous work and Section 4 presents its details. Section 5 evaluates the architecture by using a case study and Section 6 draws the main conclusions.

2. THE RNS AND MODULAR ARITHMETIC

By defining a basis $B_i = \{m_{1,i}, ..., m_{h_i,i}\}$ of pairwise coprime elements and an associated dynamic range $M_i = \prod_{e=1}^{h_i} m_{e,i}$, an integer $X < M$ has a correspondent RNS representation $x_{1,i} = X \bmod m_{1,i}$, ..., $x_{h_i,i} = X \bmod m_{h_i,i}$. The main advantage of the RNS representation is the possibility to perform in parallel the same operations one would do with integers: for three integers X, Y, and $Z = X \odot Y$ smaller than M_i, where $\odot$ is either an addition/subtraction or a multiplication, as $z_{1,i} = x_{1,i} \odot y_{1,i} \bmod m_{1,i}$, ...,

This work was supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, under the project PEst-OE/EEI/LA0021/2011.

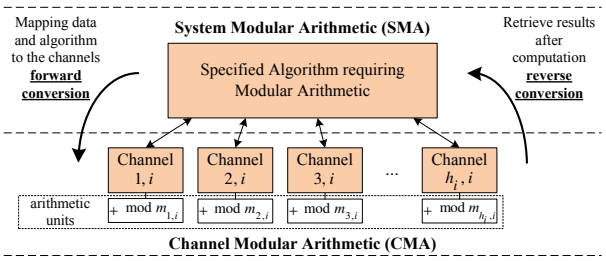


Fig. 1: Hierarchy for computing the target designer's algorithm based on the RNS representation. The operands used in the description of a MA algorithm are converted to the RNS representation so that the SMA is computed in parallel by the several channels, each one computing CMA. Thereafter, the results are converted back to the original representation used in the SMA. The challenge in the implementations with the RNS is to find efficient ways for computing operations such as addition, multiplication and modular reduction described in SMA with the CMA [see [6] for details].

Algorithm 1 Optimized RNS reduction. The values λ are precomputed constants [6] and the values $\hat{\alpha}_i$ result from the accumulation of part of the bits of $q_{e,1}$ values.

Require: ${}_M r_{e,1}$, ${}_M r_{e,2}$.
Ensure: ${}_M u_{e,1} = (RM_1 \bmod N)_{e,1}$.
Ensure: ${}_M u_{e,2} = (RM_1 \bmod N)_{e,2}$.
1: In ch. $e, 1$: $q_{e,1} = |{}_M r_{e,1} \lambda_{e,1}|_{m_{e,1}}$
2: In ch. $e, 2$: $q_{e,2} = \left| \left(\sum_{j=1}^{h_1} q_{j,1} \lambda_{2ej} + \hat{\alpha}_1 \right) \lambda_{3e} \right|_{m_{e,2}}$
3: In ch. $e, 2$: ${}_M u_{e,2} = |({}_M r_{e,2} + q_{e,2}) \lambda_{4e}|_{m_{e,2}}$
4: In ch. $e, 2$: $q_{e,1} = |{}_M u_{e,2} \lambda_{5e}|_{m_{e,2}}$
5: In ch. $e, 1$: ${}_M u_{e,1} = \left| \left(\sum_{j=1}^{h_2} q_{j,1} \lambda_{6ej} + \hat{\alpha}_2 \right) \lambda_{7e} \right|_{m_{e,1}}$
6: **return** ${}_M u_{e,1}$, ${}_M u_{e,2}$.

$z_{h_i,i} = x_{h_i,i} \odot y_{h_i,i} \bmod m_{h_i,i}$. An operation performed over the RNS representation $\bmod m_{e,i}$ is said to be performed on the RNS channel defined by $m_{e,i}$. Figure 1 depicts the correspondence between an integer X, processed with System Modular Arithmetic (SMA), and the residues $x_{e,i}$ which are operated with Channel Modular Arithmetic (CMA).

Concerning the modular arithmetic, besides addition and multiplication which are straightforward, a main operation to be implemented is the modular reduction. Algorithm 1 presents a method to accomplish the modulo N reduction of an integer R. In order to optimize the performance of Algorithm 1 the input R and outputs U are represented in a different domain (Montgomery Domain) so that ${}_M R = R(M_1^2 \bmod N)$ and ${}_M U = U(M_1 \bmod N)$. The presented modular reduction method is often referred to as Basis Extension (BE) given that it extends the operands representation for two RNS bases, B_1 and B_2 [see [6] for details].

The conversions of the data to and from the RNS representation are also important components in an RNS implementation. The required input data X can be converted to RNS with the direct computation of $x_{e,i} = X \bmod m_{e,i}$ for each channel e of the basis B_i. The reverse conversion can be accomplished by using a scalable method based on the Chinese Remainder Theorem (CRT) that states for the basis B_i that $X = \sum_{e=1}^{h_i} (x_{e,i} \Psi_e) \bmod M_i$, where Ψ_e are precomputed constants and h_i is the number of RNS channels [3]. The aforementioned methods for conversion rely in operations whose operands have the same size of X, whereas the channel arithmetic only requires arithmetic for the channel width k. Therefore, in order to reuse the k-bit wide computing resources, these methods are mapped to multiprecision versions that split the large operands by k-bit limbs [6].

3. RELATION TO PRIOR WORK

Two different approaches have been proposed to design accelerators for modular arithmetic based on RNS [7, 8]. In [7] an accelerator for EC cryptography over a finite field $GF(p)$ with p prime was proposed. This accelerator consists of three main components: *i*) an RNS forward converter, *ii*) a register file connected through buses to the channel arithmetic units, and *iii*) an RNS reverse converter followed by a projective to affine converter (EC coordinates conversion). In this architecture the multiplication is performed by a Horner scheme and no optimizations are introduced regarding the reductions. An accelerator dedicated to the modular exponentiation applied to the Rivest-Shamir-Adleman (RSA) cryptographic protocol was proposed in [8]. It is based on an architecture composed by as many multiply-accumulate units as the number of RNS channels connected to each other in a ring shape. Each one of these units is fed by a Random Access Memory (RAM) that stores the input dependent data and a Read-Only Memory (ROM) that stores the required constants. This architecture implements the RNS version of the modular multiplication with reduction by using a BE approach such as the one herein presented. In order to handle the required conversions between bases the conversion offset values of $\hat{\alpha}_j$, similar to the one in Algorithm 1, are computed by a dedicated unit in each channel. This architecture was updated to support the EC point multiplication used in EC cryptography [1]. Although all these architectures allow one to explore the RNS properties toward a more efficient design of the accelerators, they consist of dedicated implementations that only suit that particular application.

The architecture herein proposed is of the type proposed in [8]. However the herein proposed architecture is programmable and scalable, enabling the support of modular arithmetic employing a general-purpose approach.

4. A SCALABLE ARCHITECTURE

The architecture proposed in this work is presented in Figure 2. The architecture consists of several Processing Elements

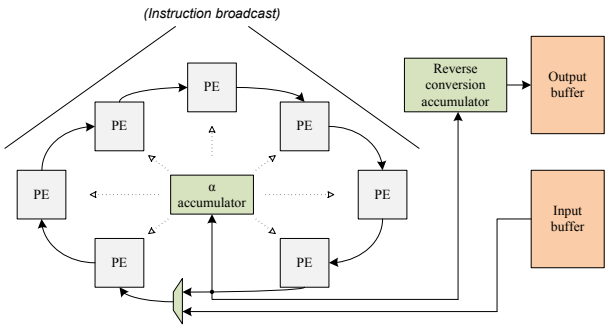


Fig. 2: Proposed architecture.

(PEs) connected as a ring, each one responsible for computing all the arithmetic for one of the RNS channels in the two bases B_1 and B_2. The architecture contains one input buffer and one output buffer, implemented as First-In First-Out (FIFO) queues, and are the only component that a host system has to interface with. There is a dedicated unit ('α accumulator') that computes the RNS conversion offset ($\hat{\alpha}_j$) required for the BE method in Algorithm 1. An accumulator to aid in the required final conversion to binary is also included, feeding the output buffer with the already converted data (there is no other overhead for computing the RNS forward and reverse conversions). The control is assured by microcoded instructions streamed into the hardware structure that operates with a Single-Instruction-Multiple-Data (SIMD) approach. Given that the microcode is vertical there is no need of an instruction decoding stage.

The shape of the architecture is mainly motivated by the need to improve the performance of the reduction, which is the most demanding operation in the RNS based arithmetic. This operation has $O(h_i^2)$ complexity, with h_i the number of channels ($O(h_i)$ in a single channel), due to the summation in the steps 2 and 5 of Algorithm 1; in these steps each one of the channels e needs to gather data from the other channels j. It is of practical interest to maintain a common control for all the PEs: while computing $\sum_{j=1}^{h_1} q_{j,1}\lambda_{ej}$ the value of $q_{j,1}$ is forwarded to the next channel after its contribution is stored in the summation result. Once a value $q_{j,1}$ completes a round in the the ring, all the PEs have all the information they need to proceed with the computation. Another, yet not less important, characteristic of the architecture is the scalability, which introduces extra flexibility in the design and the possibility of tunning it. Scalability can identified in the variety of algorithms and operand sizes the architecture supports. It is straightforward to trade-off between the number of PEs and the datapath's size inside PEs, allowing the prospection of the best configuration restricted to the timing and resource constraints as well as balance parallelism gains with overheads due to the RNS implementation (RNS forward and reverse conversions, synchronization, data communication between PEs). Figure 3 illustrates the effect of these overheads in the performance. The performance due to parallelism needs to be

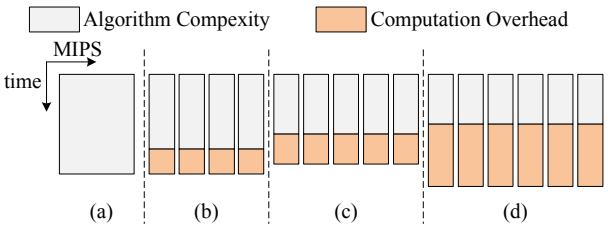


Fig. 3: Illustration of the complexity and computation time variation with the number of parallel flows: *a)* the algorithm is computed by a single serial flow; *b)* the algorithm is split by 4 parallel flows but the performance is similar to the approach with a single flow due to the overheads; *c)* the algorithm is split by 5 parallel flows and the acceleration due to the parallelization compensates the increase in the computation overhead; and *d)* the algorithm is split by 6 parallel flows and the computation overhead supersede the performance gain of the parallelization.

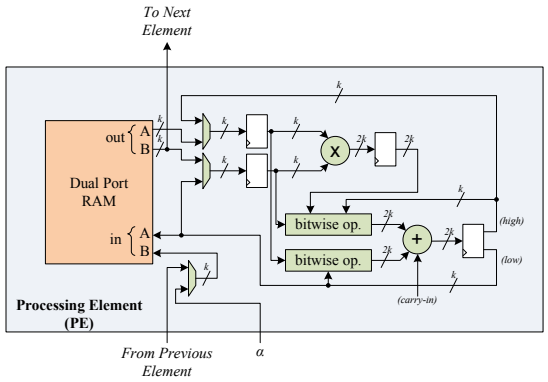
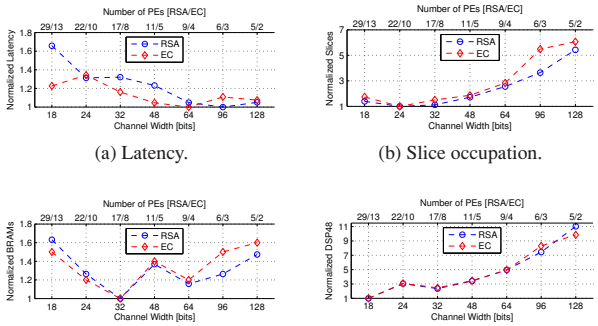


Fig. 4: The PE: the basic processing element.

balanced with the increasing complexity of operations such as the modular reduction.

Figure 4 sketches the structure of a typical PE for the architecture in Figure 2. The PE includes a dual port RAM that is responsible for storing data, input/output and temporary, as well as the RNS constants, which is directly addressed by the broadcasted instructions. It is possible, instead of the RAM, to adopt a RAM+ROM construct where the ROM would be used only for storing the required constants. In the RAM+ROM configuration both memory types reside in the same address space and can be interpreted as a single memory. The PEs include the required resources for computing the arithmetic. These resources include a binary multiplier and adder, as well as two bitwise operation blocks which are responsible for multiplexing, logical complement, and padding. Pipeline can be used to increase throughput. There are some data forwarding paths that reduce the number of clock cycles per operation. All the components can be easily configured to any datapath bit-width, i.e. a different value k.



(a) Latency. (b) Slice occupation.

(c) Block RAM occupation. (d) DSP48 slices occupation.

Fig. 5: Normalized experimental results for a modular exponentiation (RSA) and EC point multiplication on a Xilinx Virtex 4 FPGA. The value 1 stands for the most competitive configuration for each one of the metrics.

5. EXPERIMENTAL RESULTS

Two typical cryptographic operations that make extensive use of MA, a modular exponentiation used in the RSA [9] and an EC point multiplications [10], are implemented with the proposed architecture so as to support its experimental evaluation. We herein address 512-bit and 224-bit wide modular computations for each target operation, respectively. For both operations, highly efficient dedicated processors have been proposed in the literature [10, 11]. Details about these algorithms can be found in [6].

In order to obtain the implementation, the architecture and its generic components were described with technology independent VHDL (Behavioral) along with the RAMs' contents. The Synplify synthesis tool (version E-2010.09-SP2) was used to infer the main architecture building blocks from the generic description. The Xilinx ISE place and route tools (version 12.4) was used to obtain the programming bitstream targeting a Xilinx Virtex 4 (part xc4vsx55ff1148-12) FPGA technology. Note that other target technologies can also be considered giving the behavioral specification employed in the design.

Several configurations of the architecture with a different number of PEs were obtained for each operation. Figure 5 highlights the trade-off between number of PEs and their complexity for the proposed architecture. Concerning latency, low-width channels correspond to more channels, which results in more clock cycles to implement the BE (ring size increases while the channel width decreases). For larger width channels the number of clock cycles decreases, but since the complexity of the PEs increase thus the operating frequency decreases. Hence, as Figure 5a shows, the minimum latency is achieved at 96 and 64-bit channel width mark for the RSA and EC, respectively. Concerning the RAM resources, the FPGA has fixed sized Block RAMs that are inferred by the synthesis tool and can be configured with up to 32-bit per position. Hence, some widths, multiples of 32, will better suit the Block RAMs configuration and the behavior in Figure 5c is not monotonic. The DSP48 slices in the target FPGA are only used to multiply since they possess a dedicated multiplier and considering a high-level synthesis as technology agnostic, no extra effort in fully utilizing the FPGA DSP48 slices capabilities was taken. The number of required DSP48 slices increases when the channel width is increased, as Figure 5d shows. The number of slices is a trade-off between the number of PEs and the cost of a single PE, since they are used for the additions and bitwise operations, as well as in DSP48 slices interconnections. The slices are expected to increase for large values of the channel width as an effort of the synthesis tool to enhance timing. The number of resources of the RSA application (the operations' modulus is 512-bit wide) is larger than the EC's (the operations' modulus is 224-bit wide), but since the resources are strongly related with the channel implementation, the normalized figures for the resources are similar for both applications.

Given that no technology dependent optimizations were introduced in the implementation, it competes with highly optimized implementations in the literature in terms of flexibility but not in terms of performance. Notwithstanding, some performance values are stated: a modular exponentiation can be obtained in 1.6 ms with 12,181 slices, 24 Block RAMs and 216 DSP48 slices, and an EC point multiplication is accomplished in 5.8 ms with 3,435 slices, 12 Block RAMs and 64 DSP48 slices. As a reference, to the best of the authors' knowledge, [11] and [10] provide the fastest FPGA implementations for the modular exponentiation and EC point multiplication, respectively: in [11] a 512-bit modular exponentiation takes 261 μs using 3,983 slices, 7 Block RAMs, and 17 DSP48 slices whereas in [10] 365 μs, 1,580 slices, 11 Block RAMs, and 26 DSP48 slices are required for an EC point multiplication. Both implementations target also the Xilinx Virtex 4 technology.

6. CONCLUSIONS

This paper proposes an architecture to accelerate modular arithmetic algorithms with the RNS. The properties of this architecture potentiate its scalability and adaptability to any modular arithmetic algorithm. Given the generic building blocks that construct the architecture and its programmable features, the implementation and the tunning are easily accomplished. Furthermore, these characteristics validate the architecture as a suitable entity to support a high-level synthesis approach for generic modular arithmetic algorithms.

A case study of the architecture targeting an FPGA implementation (Xilinx Virtex 4 technology) suggests that it combines competitive performance figures regarding the related state of the art and a systematic implementation method which accelerates the design. Future work comprises the development of a library of optimized components for several FGPA and ASIC technologies that can be used in the architecture to improve its performance.

7. REFERENCES

[1] Nicolas Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over Fp," in *Lecture Notes in Computer Science: Advances in Cryptology - Cryptographic Hardware and Embedded Systems - CHES 2010*, Stephan Mangard and François-Xavier Standaert, Eds., pp. 48–64. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[2] Samuel Antão, Ricardo Chaves, and Leonel Sousa, "Compact and Flexible Microcoded Elliptic Curve Processor for Reconfigurable Devices," in *IEEE Symp. on Field Programmable Custom Computing Machines - FCCM*, Napa - CA, Mar. 2009, pp. 193–200, IEEE.

[3] Samuel Antão, Jean-Claude Bajard, and Leonel Sousa, "RNS based elliptic curve point multiplication for massive parallel architectures," *The Computer Journal 2011 - Oxford Journals*, vol. 55, no. 5, pp. 629–647, 2011.

[4] Samuel Antão, Jean-Claude Bajard, and Leonel Sousa, "Elliptic Curve point multiplication on GPUs," in *IEEE International Conference on Application-specific Systems Architectures and Processors - ASAP*, Rennes, Apr. 2010, pp. 192–199, IEEE.

[5] Robert Szerwinski and Tim Güneysu, "Exploiting the Power of GPUs for Asymmetric Cryptography," in *Lecture Notes in Computer Science: Advances in Cryptology - Cryptographic Hardware and Embedded Systems - CHES 2008*, Elisabeth Oswald and Pankaj Rohatgi, Eds., pp. 79–99. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2008.

[6] Samuel Antão and Leonel Sousa, "Modular arithmetic implementation with the Residue Number System (RNS)," Tech. Rep., INESC-ID, Lisbon, 2012.

[7] Dimitrios M. Schinianakis, Apostolos P. Fournaris, Harris E. Michail, Athanasios P. Kakarountas, and Thanos Stouraitis, "An RNS Implementation of an F_p Elliptic Curve Point Multiplier," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 6, pp. 1202–1213, 2009.

[8] Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shinichi Kawamura, "Implementation of RSA Algorithm Based on RNS Montgomery Multiplication," in *Lecture Notes in Computer Science: Advances in Cryptology - Cryptographic Hardware and Embedded Systems - CHES 2001*, Çetin Kaya Koç, David Naccache, and Christof Paar, Eds., pp. 364–376. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2001.

[9] Ronald L. Rivest, Adi Shamir, and Leonard Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[10] Tim Güneysu and Christof Paar, "Ultra High Performance ECC over NIST Primes on Commercial FPGAs," in *Lecture Notes in Computer Science: Advances in Cryptology - Cryptographic Hardware and Embedded Systems - CHES 2008*, Elisabeth Oswald and Pankaj Rohatgi, Eds., pp. 62–78. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2008.

[11] Daisuke Suzuki, "How to Maximize the Potential of FPGA Resources for Modular Exponentiation," in *Lecture Notes in Computer Science: Advances in Cryptology - Cryptographic Hardware and Embedded Systems - CHES 2007*, Pascal Paillier and Ingrid Verbauwhede, Eds., pp. 272–288. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007.