Residue Number System Hardware Emulator and Instructions Generator

Juvenal Araújo*, Pedro Miguens Matutino† and Ricardo Chaves§
*§INESC-ID / IST, Universidade de Lisboa, Lisbon, Portugal - †INESC-ID / ISEL, IPL, Lisbon, Portugal
Email: *juvenal.araujo@tecnico.ulisboa.pt, †pmmm@sips.inesc-id.pt, §ricardo.chaves@inesc-id.pt

***Abstract*—Residue Number System (RNS) is an alternative form of representing integers on which a large value gets represented by a set of smaller and independent integers. Cryptographic and signal filtering algorithms benefit from the use of RNS, due to its capabilities to increase performance and security. Herein, a simulation tool is presented which emulates the hardware implementation of an actual RNS co-processor. An "high-level to assembly" instructions generator is also built into this tool. The programmability and scalable architecture of the considered processor along with the high level description of the algorithm allows researchers and developers to easily evaluate and test their RNS algorithms on an actual architecture, using Java.**

I. Introduction

Residue Number System (RNS) is an alternative form of representing integers which allows to represent very large integer values as a set of smaller integers. With this, the required computations can be performed independently over this set of smaller integers, allowing for faster and more parallel execution flows [1]. A large integer X is represented by the remainder of the division of X by a set of co-prime smaller integers. This co-prime integers forms a base $\mathcal{B}_1 = (m_1, m_2, \ldots, m_h)$, namely as *moduli-set*, which is able to represent any integer no greater than the Dynamic Range (DR):

$$M_1 = \prod_{i=1}^{h} m_i. \tag{1}$$

There is therefore a unique representation of $X < M_1$ in the form

$$x_i = X \bmod m_i \ , \ 1 \leq i \leq h. \tag{2}$$

Given integers X and Y represented in RNS, the result $Z = X \odot Y$ (where $\odot$ denotes the arithmetic operations $+$, $-$ or $\times$), can be independent and simultaneously computed in the RNS execution flows, each one called as RNS channel.

In addition to its parallelisation capabilities, particularly useful in asymmetric cryptography [2] and signal filtering [3], RNS also has the capability of introducing extra security and valuable countermeasures to Side-Channel Attacks (SCA) [4], critical in security applications.

This work presents an RNS emulator, which provides researchers or developers the opportunity to test and simulate their RNS algorithms on an actual hardware architecture [5], emulated in software, using Java to describe operations. Additionally, the assembly instructions to the actual RNS co-processor are also generated by the presented tool.

II. RNS Emulator and Code Generator

Several architectures targeting asymmetrical cryptography in RNS exist in the state of the art [6]–[8]. However, these lack open and easy to use documentation and programmability options. In [5] a programmable and scalable RNS architecture providing efficient channel arithmetic with addition, subtraction and multiplication is proposed. This architecture can be used as a co-processor, speeding up the computation of a general purpose processor, or as an embedded autonomous processing unit. Its programmability and scalable architecture allows it to be used for different algorithms, including signal processing and cryptography. By emulating this hardware architecture, the proposed simulation tool provides the ability to implement, evaluate and test new RNS approaches to their computations, as well as additional RNS security features.

The RNS channel architecture described in [5] is therefore used for every RNS channel. Note that the required number of RNS channels depends on the DR and the number of bits per channel. This set of h channels form an RNS sub-processor.

In order to allow for a broader range of applications, such a
a
C
p
F
i

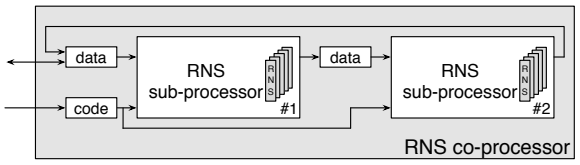

Fig. 1: RNS co-processor architecture.

In order to configure the emulated hardware (e.g. *moduli-set*, DR) the tool allows for a developer to input an XML file describing the intended configuration. Alternatively, the tool is also able to configure it, given the intended DR and the number of bits per RNS channel, writing the hardware configuration to an XML file in the same format.

Running on top of the emulated processor is an interface-like block which translates a high-level Java description into assembly, that is then used to emulate the expected behaviour of the hardware. At the same time, these assembly instructions are recorded into a file, allowing them to be later loaded into the actual processor.

Listing 1: High-level

```
/*Moduli set and hardware configuration*/
Main.loadModuliSet("mult.xml");
/* Operands to be converted to RNS*/
Rns A = new Rns("70000","A");
Rns B = new Rns("51000","B");
/* Reserve register for variable C
    without setting a value */
BigInteger C_bin;
Rns C = new Rns("C");
/* C = A x B */
A.mult(B, C);
/* Convert C back to binary*/
C_bin = C.toIntMRC();
```

Listing 2: Assembly code

```
[01] Bin-to-RNS_p    A
[10] Bin-to-RNS      A
[01] Bin-to-RNS_p    B
[10] Bin-to-RNS      B
[11] mul_a           C,A,B
[01] RNS-to-MRC      aux,C
[01] MRC-to-Bin      aux
```

Fig. 2: Multiplication example

The high-level description language includes instructions for conversion to and from RNS, non-modular addition/subtraction/multiplication operations and Montgomery Modular Multiplication. An example performing a single multiplication is depicted in Figure 2.

The assembly instructions can be sent to each one of the RNS sub-processors, or both, defined by the first 2 bits of the instruction. The first instruction is sent to RNS sub-processor #1 with an added "_p" indicating that it should propagate the value being read from the exterior data-bus to sub-processor #2, since this sub-processor is not directly connected to the exterior [5].

With this approach the user does not need to know the defined ISA for this architecture and minimum RNS knowledge is needed to write the high-level description of a given algorithm. This way, developers and designers do not require any particular knowledge on RNS to implement their algorithms and can immediately benefit from an architecture exploring the RNS parallelisation properties.

III. Testing and Validation

In order to test and validate the proposed tool, RSA [10] and Elliptic Curve Cryptography (ECC) [11] algorithms were implemented along with other arithmetic operations. RSA is based on a sequence of modular multiplications [12], whereas ECC is based on a mix of non-modular operations and modular multiplications [13]. Three other simpler examples are also provided for educational purposes, performing a single multiplication, a single modular multiplication, and an Elliptic Curve Point Addition [13].

The RNS emulator tool is publicly available in [14], together with the aforementioned examples.

IV. Conclusions

In conclusion, a development tool was created offering the possibility to emulate an existing RNS processing architecture and to generate its assembly code. Java based high-level instructions are used to describe the desired algorithms, which do not require any RNS knowledge from the developers. Furthermore, the correctness of the tool was verified and tested by implementing complex examples, such as the two asymmetrical cryptographic algorithms RSA and ECC, and running the produced assembly instructions in the actual hardware RNS co-processor.

Acknowledgments

"This work was partially supported by the ARTEMIS Joint Undertaking under grant agreement n^o 621429 and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013."

References

[1] H. L. Garner, "The Residue Number System," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 2, pp. 140–147, June 1959.

[2] J. Bajard and L. Imbert, "A full RNS implementation of RSA," *Computers, IEEE Transactions on*, vol. 53, no. 6, pp. 769–774, June 2004.

[3] R. Conway and J. Nelson, "Improved rns fir filter architectures," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 51, no. 1, pp. 26–28, Jan 2004.

[4] D. Schinianakis and T. Stouraitis, "Hardware-fault attack handling in rns-based montgomery multipliers," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, May 2013, pp. 3042–3045.

[5] P. Matutino, R. Chaves, and L. Sousa, "An Efficient Scalable RNS Architecture for Large Dynamic Ranges," *Journal of Signal Processing Systems*, pp. 1–15, 2014.

[6] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-Rower Architecture for Fast Parallel Montgomery Multiplication," in *Advances in Cryptology – EUROCRYPT 2000*, ser. Lecture Notes in Computer Science, B. Preneel, Ed. Springer Berlin Heidelberg, 2000, vol. 1807, pp. 523–538.

[7] J. Wei, W. Guo, H. Liu, and Y. Tan, "A Unified Cryptographic Processor for RSA and ECC in RNS," in *Computer Engineering and Technology*, ser. Communications in Computer and Information Science, W. Xu, L. Xiao, C. Zhang, J. Li, and L. Yu, Eds. Springer Berlin Heidelberg, 2013, vol. 396, pp. 19–32.

[8] D. Schinianakis, A. Fournaris, H. Michail, A. Kakarountas, and T. Stouraitis, "An RNS Implementation of an F_p Elliptic Curve Point Multiplier," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, no. 6, pp. 1202–1213, June 2009.

[9] D. Schinianakis and T. Stouraitis, "A RNS Montgomery multiplication architecture," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, May 2011, pp. 1167–1170.

[10] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[11] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast Key Exchange with Elliptic Curve Systems," in *Advances in Cryptology – CRYPT0'95*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1995, vol. 963, pp. 43–56.

[12] M. Joye and S.-M. Yen, "The Montgomery Powering Ladder," *Cryptographic Hardware and Embedded Systems*, vol. 2523, 2003.

[13] J.-C. Bajard, S. Duquesne, and M. D. Ercegovac, "Combining leak-resistant arithmetic for elliptic curves defined over Fp and RNS representation." *IACR Cryptology ePrint Archive*, vol. 2010, p. 311, 2010.

[14] J. Araujo, P. Matutino, and R. Chaves, http://sips.inesc-id.pt/~rjfc/cores/RNS/, September 2016.