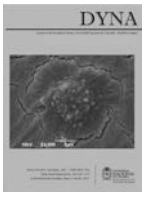
UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN
FACULTAD DE MINAS



DYNA
http://dyna.medellin.unal.edu.co/

Performance evaluation of M-ary algorithm using reprogrammable hardware

Sergio Andrés Arenas-Hoyos & Álvaro Bernal-Noreña

Escuela de Ingeniería Eléctrica y Electrónica, Universidad del Valle, Santiago de Cali, Colombia. sergio.arenas@correounivalle.edu.co, alvaro.bernal@correounivalle.edu.co

Received: June 5th, 2017. Received in revised form: August 23th, 2017. Accepted: September 18th, 2017

Abstract
Several ways to perform data encryption have been found, and one of the functions involved in standard algorithms such as RSA is the modular exponentiation. Basically, the RSA algorithm uses some properties of modular arithmetic to cipher and decipher plain text, with a certain performance dependence on text lengths. The growth in computing capacity has created the need to use robust systems that can perform calculations with significantly large numbers and the formulation of procedures focused on improving the speed to achieve it. One of these is the M-ary algorithm for the execution of the modular exponential function. This paper describes an implementation of this algorithm in reprogrammable hardware (FPGA) to evaluate its performance.
The first section of this work introduces the M-ary algorithm. The second section uses block description for implementation understanding. The third section shows the results in time diagrams, and finally, the last section conclusions.

Keywords: cryptosystems; modular exponentiation; modular arithmetic; RSA algorithm; FPGA; M-ary algorithm.

Evaluación del desempeño del algoritmo M-ary en hardware reprogramable

Resumen
Se han encontrado diversas formas de realizar cifrado de datos, y una de las funciones involucradas en algoritmos estándar como el RSA es la exponencial modular. Básicamente, el algoritmo RSA utiliza algunas propiedades de la aritmética modular para cifrar y descifrar textos planos, con cierta dependencia en la longitud del texto. El crecimiento en la capacidad de cómputo ha creado la necesidad de utilizar sistemas robustos que puedan realizar cálculos con números significativamente grandes, y la formulación de procedimientos enfocados en mejorar la velocidad para lograrlo. Uno de éstos es el algoritmo M-ary para la ejecución de la función exponencial modular. Este artículo describe una implementación de este algoritmo en hardware reprogramable (FPGA) para evaluar su desempeño.
La primera sección introduce el algoritmo M-ary. La segunda, usa descripción en bloques para comprender la implementación. La tercera, muestra los resultados en diagramas de tiempo, y finalmente, la última sección expone conclusiones.

Palabras clave: criptosistemas; exponencial modular; aritmética modular; algoritmo RSA;FPGA; Algoritmo M-ario.

1. Introduction

Modern systems are designed to obtain better performance in terms of merit figures, such as power, area and/or speed. In hardware terms, the natural language of a computer is a set of integers restricted by a number, named modulo, establishing a ring of integers. This introduces the concept of modular arithmetic and its consequent number representation that is used in many areas, including digital signal processing or cryptographic systems [5]. This representation been probed using modular arithmetic, specifically Residue Number System RNS [12], instead of two-complement or classical weighted binary, to increase the speed performance in FIR filters [8,9].

This paper focuses on a specific implementation of a special modular function, modular exponentiation. Taking advantage of the inherent flexible characteristics of FPGA

How to cite: Arenas-Hoyos, S.A. and Bernal-Noreña, A., Performance evaluation of M-ary algorithm using reprogrammable hardware DYNA, 84(203), pp. 75-79, December, 2017.

architecture, it is possible to develop a system that uses the M-ary algorithm [10] to test various characteristics, which are verified with a software equivalent and compared with previous implementations of the method.

Previous systems were made with different schemes, taking advantage of a pure hardware co-design approach (hardware subsystem and software subsystem) or using algorithms such as an addition-chain to reduce multiplication steps [7,11]. These systems were based on combining their best characteristics.

Modular exponentiation consists in finding a solution to the following eq. (1):

$$Z = X^y mod M, 2^{(n-1)} < M < 2^n \quad (1)$$

where *n* is the length of *M* in bits.

The M-ary algorithm can be subdivided into different sub processes as follows [5]: exponent segmentation in *w* windows of *d* bits, preprocessing and storing of every possible power of any base in a range determined by 0 to 2^d, squaring and multiplying to obtain a result. These steps are illustrated in the pseudo code scheme shown in Fig. 1.

Modular exponentiation implies another operation, modular multiplication. This function is implemented with an algorithm called the Montgomery Method [6,12]. This process takes two cycles, one for pre-calculating a result in a space called the M-residues Space with an undesirable factor R^-n called the multiplicative inverse of R = 2^n mod M. After the first cycle, a second cycle is used to recover the desired multiplication without the R^-n factor. This method avoids computing the integer division, which is an expensive operation in hardware.

The Montgomery Method is described in the pseudo code shown in Fig. 2.

```
Set V0 = 1 and V1 = X
        From j = 2 to (2d-1) do {
                Vj= Vj-1XmodM;
        From i=k-1 to 0 do
                        d-1
                F(i) = ∑yid+t2t
                        t=0
        Set Z = VF(k-1)
        From i=k-2 to 0 do
                From j=0 to (d-1) do
                        Z = Z*ZmodM
                If F(i) ≠ 0 then Z = Z*VF(i) modM
Halt
```

Figure 1. Pseudo code for M-ary algorithm
Source: [4]

```
Z = 0
for i = 0 to n-1
    Z = Z + x_i × Y
    if Z is odd then Z = Z + M
    Z = Z/2
if Z = M then Z = Z - M
```

Figure 2. Pseudo code for Modular Multiplication using the Montgomery algorithm.
Source: [2]

Modular multiplication
Inputs: *A, B, M, Inv*
Outputs: *R_final*

R_MM, R_final = 0
1: R_MM= Montgomery (A, B, M)
2:R_final= Montgomery (R_MM, Inv, M)

Return R_final

Figure 3. Pseudo code for Modular Multiplication
Source: The authors.

Modular multiplication can be described as shown in Fig. 3.

2. Implementation

The complete system was modeled with VHDL language using basic libraries like the IEEE Standard, allowing the system to be more generic and portable among different architectures. Reprogrammable hardware is used because it allows the design of digital architectures with certain flexibility.

In a block description, modular exponentiation has four subsystems that work together including the Montgomery Multiplier, the Storage Unit, the Exponent Segmentation Unit, and the Control Unit. These blocks are shown in Fig. 4.

Sequential logic uses a Finite States Machine under the Moore scheme, where the outputs depend only on the actual state because asynchronous behaviors should be avoided due to a complex and significant quantity of states (approximately 40 for the principal control module).

Internally, the Montgomery Multiplier has another sub control unit, which synchronizes a data path and advertises to the central unit when multiplication is finished. This controller also switches sources in the multiplier when a Montgomery cycle is completed, as illustrated in Fig. 5.

Operands have a 12-bits length, implying the use of 12-bits and 1-bit carry units.

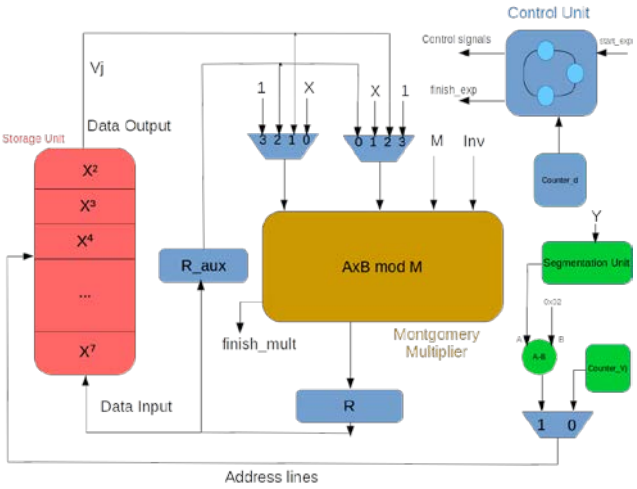


Figure 4. Modular exponentiation general system
Source: The authors.

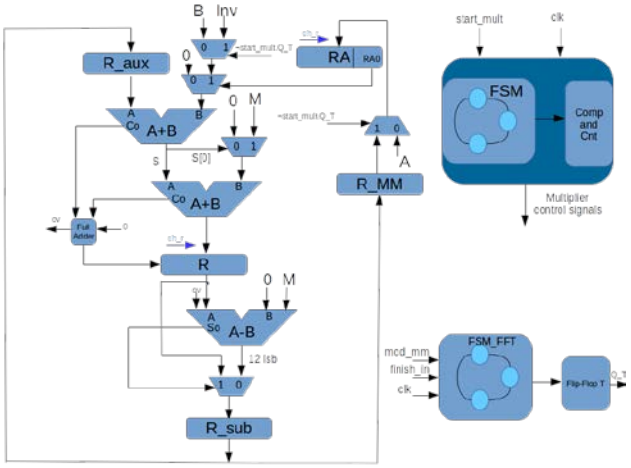


Figure 5. Modular multiplication subsystem
Source: The authors.

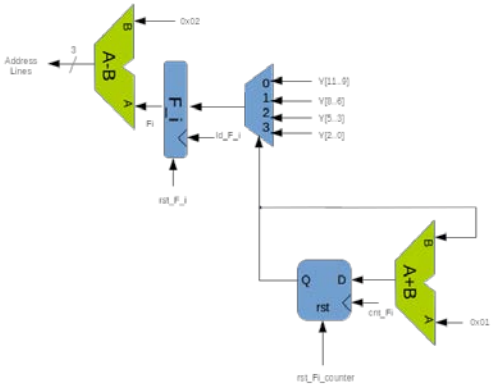


Figure 6. Segmentation Unit
Source: The authors.

The segmentation unit uses a 4-to-3 multiplexer in order to fragment the exponent in fields that are used as a kind of pointer to preprocessing powers. A counter helps select which of those segments is going to be used. There is an offset of 2 addresses because X^0 and X^1 are not stored in the memory register bank. The block diagram (Fig. 6) shows the internal constitution of this unit.

Address lines are used to control a storage unit, which is composed by a set of 16 registers with 12 bits width each, and an input decoder takes the address and enables writing to one register. When reading is required, a 16-to-12 multiplexer connected to the address lines takes the register data and puts it into the output register; these lines send a Vj signal to the Montgomery Multiplier. This system allows writing and reading one register at a time, and the internal structure of the system is shown in Fig. 7.

3. Experimental results

For testing purposes, a test block is added. The test block contains the inputs pattern to the M-ary block. Each block contains a set of 48 bits, which are subdivided in 4 numbers

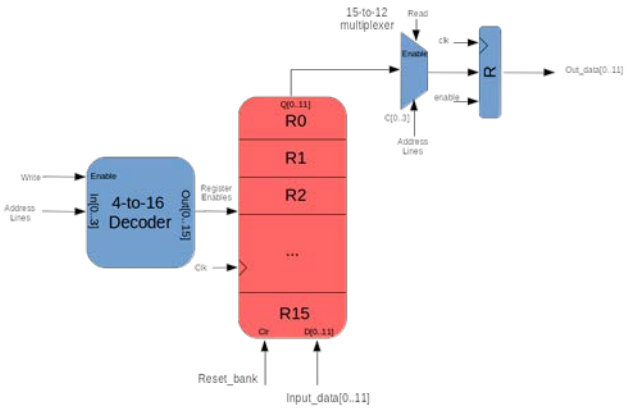


Figure 7. Storage Unit
Source: The authors.

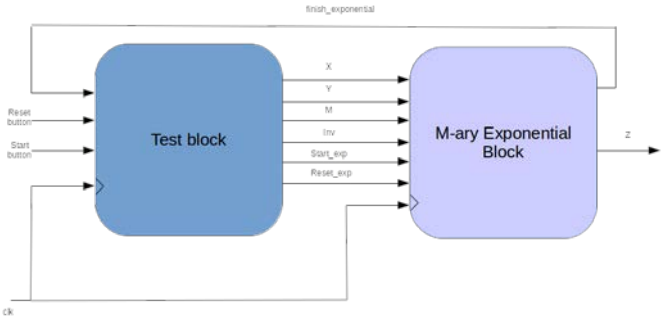


Figure 8. Testing system
Source: The authors.

of 12-bit length and represent the needed inputs as follows: a base (X), an exponent (Y), a modulus (M) and a correction factor (R). There are a total of 4 samples, as reflected in the use of a storage system with an attribute of 4 words - 48 bits ROM. This system starts each exponential process and senses a finish flag bit provided by the exponential block. When an exponential process finishes, a counter changes the address pointer that is controlling the ROM behavior and loads the respective data subsets in the output registers. Fig. 8 illustrates the testing structure.

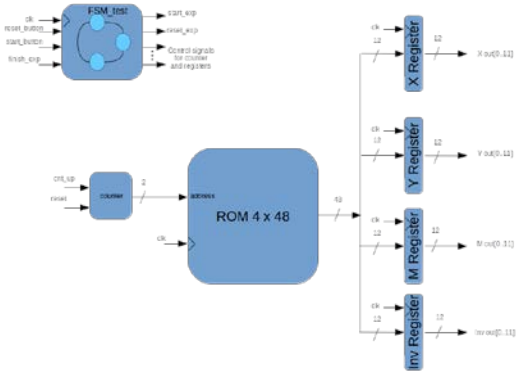


Figure 9. Test block, detailed overview
Source: The authors.

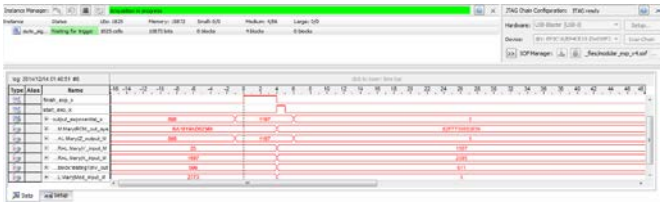


Figure 10(a). Data set X = 1697, Y = 25, M = 2773, Inv = 566, Z = 1197

Figure 10(b). Data set X = 2000, Y = 140, M = 2901, Inv = 733, Z = 982
Source: The authors.

A detailed scheme of the internal structure in the testing block can be seen in Fig. 9.

This system is for testing purposes, so the hardware requirements were not great, allowing for a low-cost system, such as the DE0 evaluation board.

Physical implementation of the previous block was carried out using Quartus II software, Signal-Tap, and a DE0 development board that has an EP3C16F484C6N FPGA from Altera Corp. Figs. 10(a) and 10(b) show the timing diagrams for some data sets.

An estimation of the merit figures was obtained using Timer Quest and Quartus II tools as follows:

Max. working frequency = 77,57 MHz
Total Thermal Power = 74,29 mW, Area percentage: 19 %

With all this examples in mind, it is convenient to show another. The following example addresses the cryptographic application of the modular exponentiation, and its aim is to cipher and decipher a character that is written in an ASCII-like style.

The next steps are used to obtain the public and private keys of the RSA algorithm [3]:

1. Takes two prime numbers p and q as well as their product n:

$$p = 11, q = 227, n = 2497$$

2. Calculates Euler's Phi function of n using:

$$\varphi(n) = \varphi(p).\varphi(q) = (p-1).(q-1) \quad (2)$$

$$\varphi(n) = (11-1).(227-1) = 10.226 = 2260$$

3. Chooses a prime number e, which is less than φ(n)

$$e = 137$$

4. Finds the multiplicative inverse in modulus φ(n) of e, defined by:

$$d = e^{-1} mod(\varphi(n)) \quad (3)$$

$$d = 137^{-1} mod(2260)$$

5. Uses an iterative algorithm in Matlab to find the multiplicative inverses, based on the following fact:

$$d.e = i.\varphi(n) + 1 \quad (4)$$

$$d = \frac{(i.\varphi(n)+1)}{e}, d \text{ is an integer}$$

Iterating over i, where d is an integer, to find that:

$$d = e^{-1} mod(\varphi(n)) = 33$$

Multiplies both d and e:

$$d.e\ mod\ (\varphi(n)) = 33.137\ mod(2260)$$
$$d.e\ mod(\varphi(n)) = 4521\ mod\ (2260) = 1$$

This proves that *e* is the multiplicative inverse of *d* in modulo φ(n).

6. Creates a private key with the set (p, q, d) = (11, 227, 33) and a public key (n, e) = (2497, 137). Now, using ASCII code, "s" is represented with 83 as the message to transport to use the cipher and decipher steps.
7. Cipher step consists of using the modular exponentiation to solve the next equation:

$$C = M^e mod(n) \quad (5)$$

$$C = 83^{137} mod\ 2497 = 283$$

Fig. 11 shows cipher results.

8. Finally, the decipher step consists of using modular exponentiation to solve the next equation:

$$M = C^d mod(n) \quad (6)$$

$$M = 283^{33} mod\ 2497 = 83$$

Figure 11. Ciphering "s" character (plain text)
Source: The authors

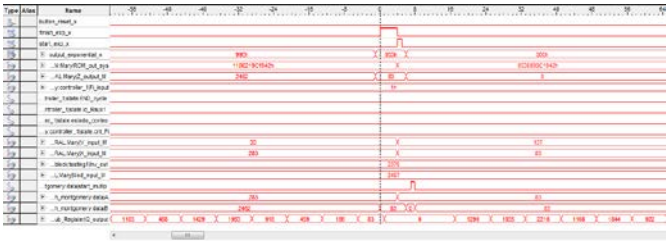


Figure 12. Deciphering the cipher equivalent of “s” character
Source: The authors.

Fig. 12 shows the decipher results.

4. Conclusions

- With practical work experience, it was clearly shown that the synchronization between blocks is affected by the dependence of the data. Its solution facilitates the treatment of the signals involved in the design of digital systems.
- The design of equivalent procedures in software facilitated error debugging in the hardware implementation.
- The modular multiplication control system was optimized using a Flip-Flop T to reduce the number of steps in FSM. Additionally, if a zero value segment was detected in the exponent, the central control unit avoided a multiplication.
- Using an EP3C16F484C6N Cyclone III FPGA from Altera Corp., the merit figures reported were as follows:
  - Maximum working frequency = 77.57 MHz.
  - Total thermal power = 74,29 mW.
  - Area percentage = 19 % of 15408 logic elements.
- In future work, the optimization of system controllers is a possible aim to reduce the number of clock cycles per operation. For the data path, the systolic design should be evaluated and considered to reduce complexity in the system controllers and to avoid synchronization between blocks. On the other hand, it would be interesting to create a completely parameterizable block where the amount of data and window bits are adjustable to requirements within cryptographic designs. Another important point to be explored is the use of modular arithmetic to implement more efficient digital signal processing filters in terms of time [5].

Bibliographic

[1] De Macedo-Mourelle, L. and Nedjah, N., Fast reconfigurable hardware for the M-ary modular exponentiation. In Digital System Design, DSD 2004. Euromicro Symposium on System Design, pp. 516-523, IEEE. August, 2004.

[2] Harris, D., Krishnamurthy, R., Anders, M., Mathew, S. and Hsu, S., An improved unified scalable radix-2 Montgomery multiplier. In Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on IEEE. June, 2005, pp. 172-178.

[3] Rivest, R.L., Shamir, A. and Adleman, L., A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), pp. 120-126, 1978.

[4] Bernal, A., Conception et étude d'une architecture numérique de haute performance pour le calcul de la fonction exponentielle modulaire Dr. dissertation, Institut National Polytechnique de Grenoble-INPG, France, 1999.

[5] Chang, C.H., Molahosseini, A.S., Zarandi, A.A.E. and Tay, T.F., Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. IEEE circuits and systems magazine, 15(4), pp. 26-44, 2015.

[6] Montgomery, P.L., Modular multiplication without trial division. Mathematics of Computation, 44(170), pp. 519-521, 1985.

[7] Nedjah, N. and de Macedo-Mourelle, L., Four hardware implementations for the m-ary modular exponentiation. In Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on 2006 IEEE. April, 2006, pp. 210-215.

[8] Di Claudio, E.D., Orlandi, G. and Piazza, F., Fast RNS DSP algorithms implemented with binary arithmetic. In Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on IEEE. April, 1990, pp. 1531-1534.

[9] Ramírez, J. and Meyer-Baese, U., High performance, reduced complexity programmable RNS-FPL merged FIR filters. Electronics Letters, 38(4), pp. 199-200, 2002.

[10] Egecioglu, O. and Koç, C.K., Fast modular exponentiation. Communication, Control and Signal Processing, 1, pp. 188-194, 1990.

[11] Nedjah, N., Mourelle, L.M., Santana, M. and Raposo, S., Massively parallel modular exponentiation method and its implementation in software and hardware for high-performance cryptographic systems. IET Computers and Digital Techniques, 6(5), pp. 290-301, 2012.

[12] Menezes, A.J., Van Oorschot, P.C. and Vanstone, S.A., Handbook of applied cryptography, Chap 14 Efficient Implementation. CRC press. Retrieved from: http://cacr.uwaterloo.ca/hac/(1996).

S. Arenas-Hoyos, received the BSc. in Electronic Eng. from Universidad del Valle, Colombia in 2015. In September 2011, he joined the Digital Architectures and Microelectronic Group of the Universidad del Valle as researcher. His areas of interests are the programmable architectures, digital systems design, hardware description languages, embedded systems and digital signal processing.
ORCID: 0000-0002-5396-241X

A. Bernal-Noreña, received the BSc. in Electrical Eng. in 1987 from Universidad del Valle, Cali, Colombia, the MSc. degree in Science in Electrical Engineering majoring in VLSI circuit design from Escola Politécnica da Universidade de São Paulo, São Paulo, Brazil in 1997, and the PhD degree in Microelectronics from Institute National Polytechnique de Grenoble, Grenoble, France in 1999. Currently, is full professor at the Engineering Faculty of Universidad del Valle, Cali, Colombia and leader of the Group of Digital Architectures and Microelectronic.
ORCID: 0000-0003-4766-8086