Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 127147, 10 pages
doi:10.1155/2011/127147

Research Article

Montgomery Modular Multiplication on Reconfigurable Hardware: Systolic versus Multiplexed Implementation

Guilherme Perin,[1] Daniel Gomes Mesquita,[2] and João Baptista Martins[1]

[1] *Grupo de Microeletrônica (GMICRO), Universidade Federal de Santa Maria (UFSM), Rio Grande do Sul, Santa Maria 97105-900, RS, Brazil*
[2] *Arquiteturas, Sistemas Operacionais e Sistemas Distribuídos, Grupo de Redes (GRASS), Universidade Federal de Uberlândia (UFU), Uberlândia 38401-136, MG, Brazil*

Correspondence should be addressed to Guilherme Perin, guilhermeperin@ymail.com

Received 2 June 2010; Accepted 30 November 2010

Academic Editor: Elías Todorovich

This paper describes a comparison of two Montgomery modular multiplication architectures: a systolic and a multiplexed. Both implementations target FPGA devices. The modular multiplication is employed in modular exponentiation processes, which are the most important operations of some public-key cryptographic algorithms, including the most popular of them, the RSA. The proposed systolic architecture presents a high-radix implementation with a one-dimensional array of Processing Elements. The multiplexed implementation is a new alternative and is composed of multiplier blocks in parallel with the new simplified Processing Elements, and it provides a pipelined operation mode. We compare the *time* × *area* efficiency for both architectures as well as an RSA application. The systolic implementation can run the 1024 bits RSA decryption process in just 3.23 ms, and the multiplexed architecture executes the same operation in 4.36 ms, but the second approach saves up to 28% of logical resources. These results are competitive with the state-of-the-art performance.

1. Introduction

Modular multiplication is widely employed in public-key cryptography, especially where modular exponentiation is essential. For instance, the most commonly used asymmetric cryptographic algorithm is the RSA [1]. The RSA security depends on the difficulty of factoring large numbers. Here, large numbers mean prime numbers of up to 4096 bits, used as cryptographic keys.

In this cryptosystem the main operation is the modular exponentiation using the public and private keys, the first to encrypt and the second to decrypt messages. So, the performance of the whole system depends on the efficiency of modular arithmetic implementations.

As modular operations are time consuming, it is common to use hardware devices to perform both the modular multiplication and the exponentiation. Among the hardware approaches, the increased use of reconfigurable devices to implement cryptographic operations, especially the FPGAs, is evident.

One of the most suitable methods for performing modular multiplications in hardware is the Montgomery multiplication [2]. This algorithm is fast and power efficient in hardware implementations. Assuming the modular multiplication as $A \cdot B \bmod N$, the Montgomery multiplication avoids the division by N by replacing the division by right shifts. Also, this method allows the use of multi-precision arithmetic, which is useful for employing high-radix operations. High-radix operations in turn make it easier to develop modular multiplication architectures.

Aiming to implement RSA systems based on hardware, many authors proposed Montgomery multiplications in FPGAs [3–9]. Fully systolic architectures designed to speed up the modular multiplication have been presented. These architectures offer a Processing Elements (PEs) array where each PE performs arithmetic additions and multiplications

Require: $N = \sum_{i=0}^{m-1}(2^k)^i n_i, n_i \in \{0, 1, \ldots, 2^k - 1\}$
$B = \sum_{i=0}^{m-1}(2^k)^i b_i, \quad b_i \in \{0, 1, \ldots, 2^k - 1\}$
$A = \sum_{i=0}^{m-1}(2^k)^i a_i, a_i \in \{0, 1, \ldots, 2^k - 1\}, R = (2^k)^m$
$A, B < 2N; N < R = 2^{km}; N' = -N^{-1} \bmod (2^k)$.
return $S_{i+1} = \mathrm{ABR}^{-1} \bmod N$
$S_0 = 0$
For $i = 0$ to $m - 1$ **do**
$q_i = ((S_0 + a_i \times b_0)N') \bmod (2^k)$
$S_{i+1} = (S_i + q_i \times N + a_i \times B)/2^k$
End for

Algorithm 1: Montgomery modular multiplication.

in a multiprecision context with carry propagation [10]. Depending on the word size (or radix) used, the architecture can employ a high number of Processing Elements, consequently increasing the needs of the logic elements (area) in FPGA implementations.

As a new alternative in terms of implementation, the execution of additions and multiplications can be multiplexed by a block positioned parallel to the Processing Elements. This can be done by inserting multiplexed multipliers in parallel with Processing Elements. Forcing a pipelined operation mode and using a high-radix architecture (16 or 32 bits), the multiplexed multipliers ensure the high speed performance provided by systolic architectures, with reduced arithmetic and logic elements and also minimal carry signals propagation.

This paper presents a trade-off between two proposed modular multiplication architectures: a systolic and very high-radix multiplexed implementation. Our approach uses a radix-16 and radix-32 in both implementations to speed up the processes and to match the resource usage of Virtex-4 and Virtex-5 Xilinx FPGA Series [11]. The proposed architectures show significant improvements compared to our previous work [12]. Systolic architecture provides more simplified Processing Elements in order to reduce the utilization of FPGA resources. The multiplexed implementation is arranged in arithmetic cores, which allow us to handle the quantity of Processing Elements and multiplier blocks. Our goal is to highlight that the small increase in the number of clock cycles needed due to multiplexed multipliers made up for the significant reduction in the use of logical and architectural arithmetic.

This paper is organized as follows: Section 2 presents the Montgomery modular multiplication algorithm. Section 3 discusses related state-of-the-art works. The proposed architectures are presented in Section 4. Finally, the results and conclusion are presented in Sections 5 and 6, respectively.

2. Montgomery Modular Multiplication

The Montgomery Multiplication Algorithm is a method of performing modular multiplication $A \cdot B \bmod N$ without needing to divide by N. In cryptography, the Montgomery Algorithm is very suitable for the hardware implementation of modular multiplication, because it allows long integer numbers to be represented in a numeric precision given by a radix (generally a power of two).

The algorithm version used in this work is the original one, with some preconditions. Algorithm 1 shows the modular multiplication with the notation proposed on [13], and used for the remainder of this text.

The N' value is the modular inverse of N regarding the N modulus, computed so that $N \cdot N' = 1 \bmod N$. The final result is placed on S, after m iterations, and is equal to $A \cdot B \cdot R^{-1} \bmod N$, which must be corrected to retrieve the expected result ($A \cdot B \bmod N$). The correction is done by performing an additional Montgomery multiplication with S and $R^2 \bmod N$ as parameters. It is interesting to highlight that this correction is inexpensive during a modular exponentiation, because it only needs to be made one time after the whole exponentiation.

Since its publication in 1985 by Montgomery [2], the Montgomery Algorithm has undergone many modifications and improvements [14, 15]. One of those is particularly interesting, because it avoids the final subtraction simply by choosing the input data correctly. By limiting the operands A and B to integers less than $2N$ and by defining $2N$ as less than 2^{km}, the final S is guaranteed to be less than N [15]. These pre-conditions are shown in Algorithm 1 and applied to our architecture, as explained in Section 4.

3. Related Works

Tenca and Koç are widely referenced for their work on radix-2 Montgomery Algorithm implementations. These authors initially proposed architectures with improvements for the radix-2 Montgomery Algorithm, like in [16]. Even though the input operands are large numbers, radix-2 modular multiplications avoid expensive multiplications, which are visible on high-radix implementations (8 or more). Different from the classic radix-2 Montgomery Algorithm [13], Tenca and Koç's modifications allow the scalable property for modular multiplication architecture, that is, their proposed Montgomery multiplier is able to work with any precision of the input operands. In terms of hardware implementation, there is a systolic array architecture composed of Processing Elements and control blocks for managing the I/O words of the architecture. Each Processing Element contains only a few logic elements, providing a reduced area and high clock frequency, when synthesized for FPGA or ASIC.

Based on the above work, in [4, 17] improvements are presented to the Tenca and Koç proposition. The advantage of these new approaches is concentrated in the Processing Elements optimizations and, consequently, in the reduced latency of the Montgomery modular multiplications by a minimum factor of two, that is, the modular multiplication is twice as fast than [16]. So, the main contributions are in the modular multiplication speed improvement, and in the reduced number of logical elements for the Processing Elements. In [4], a radix-4 scalable Montgomery modular multiplication architecture is proposed to enhance the speed. Despite improvements in speed, these radix-2 and radix-4 architectures are still limited by the large number of clock cycles required.

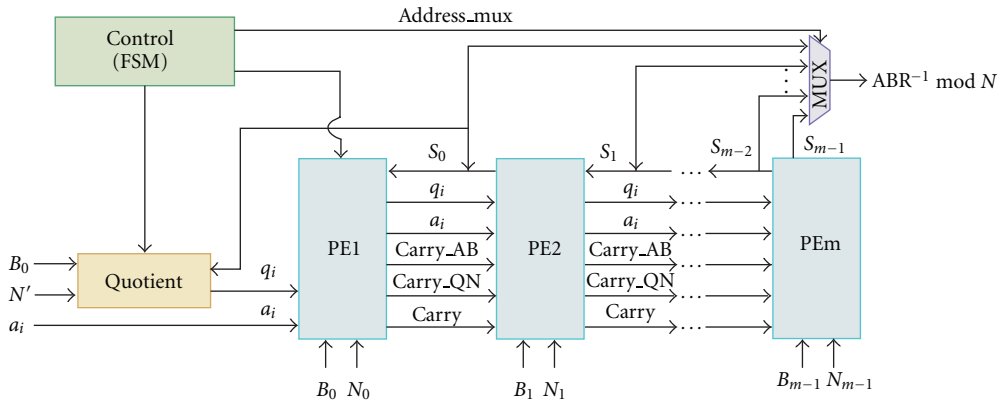

Figure 1: Systolic Architecture.

Furthermore, in the context of high-radix implementations, a systolic architecture is presented in [3] which is composed of Processing Elements able to provide modular multiplication for a radix greater than 4. Despite its time and area efficiency, this architecture requires preprocessing before the modular multiplication execution. The authors make use of the optimized Montgomery algorithm initially proposed in [14], which presented a way to simplify the q_i quotient calculus, making the quotient determine a simple truncation operation $S \bmod 2^k$. However, as a consequence, the input operands must meet the following limitations: $N' = -N^{-1} \bmod 2^k = 1$ and $A, B < 2(N' \bmod 2^k) \cdot N$, and the optimized Montgomery Algorithm will need three additional iterations, because the B input operand is left shifted by 2^k and has to be corrected with these further iterations.

To avoid preprocessing in a high-radix modular multiplication, [5] presents a fully systolic array architecture composed of Processing elements containing internal multipliers and adders. The Montgomery algorithm version used in this implementation is also the optimized version proposed in [14]. As an implementation in radix-16, the modular multiplications take only 103 clock cycles, significantly less than other architectures [3, 16, 17].

4. The Proposed Architectures

The proposed architectures for performing Montgomery modular multiplication are detailed in this section. First, the systolic architecture is described in detail as well as the Processing Elements behaviour. Second, the multiplexed and systolic Montgomery modular multiplication architecture is presented.

4.1. The Systolic Architecture. The concept of systolic architecture combines a highly parallel array of identical Processing Elements or data-paths with local connections, which take external inputs and process them in a predetermined manner and in a pipelined fashion.

The proposed systolic architecture is directly based on the arithmetic operations of the Montgomery Algorithm, which are performed in a numerical base 2^k, in which the large input operands are processed in a multi-precision context containing m words of k bits. As seen in Section 2, the Montgomery Algorithm has additions and multiplications involving large integers that make use of multiple-precision arithmetic.

The architecture is composed of m Processing Elements distributed in a one-dimensional array, where each Processing Element is responsible for the calculus involving k bits words of the input operands with the same index of the Processing Element. For example, for a 1024 bits modular multiplication with radix-32, the operands are split in 32 words of 32 bits which results in a one-dimensional array of 32 Processing Elements.

Between the Processing Elements, there is a propagation of carry signals which are the most significant bits of the arithmetic processes in each PE. The carry signals are processed as input parameters by the Processing Elements that receive them.

In the systolic architecture, the Processing Elements are designed by finite state machines. The control block communicates with the first Processing Element (PE1) and with the block responsible for the quotient calculation $q_i = (S_0 + a_i \cdot b_0)N' \bmod 2^k$, according to line 4 of the Montgomery Algorithm. Figure 1 presents the systolic architecture.

The finite state machine structure of the control block is designed to provide the required words for a modular multiplication to the Processing Elements and to the quotient block. Thus, at each Montgomery Algorithm iteration, these words are read from an external RAM memory and passed to the remaining architecture. At the end of the modular multiplication, the control block provides the Montgomery multiplication result $A \cdot B \cdot R^{-1} \bmod N$ through an output multiplexer.

The one-dimensional array of Processing Elements performs the calculation of $S_{i+1} = (S_i + q_i \times N + a_i \times B)/2^k$, according to the Montgomery Algorithm. In this operation, there are two multiplications between an input operand and a k bits word, and after the addition between the result of these two multiplications. Therefore, the systolic architecture works in a multi-precision context, and each Processing Element is responsible for performing the

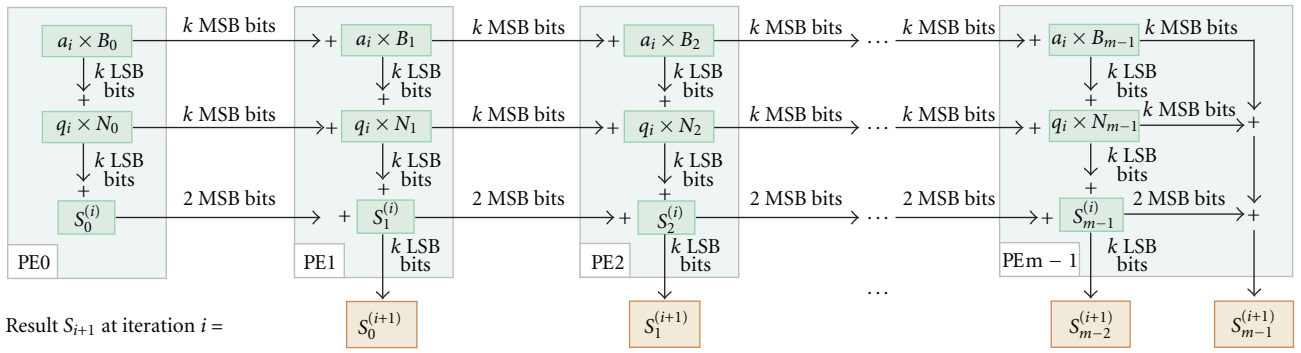

Figure 2: Arithmetic operations of the Processing Elements to obtain $S_i + 1 = (S_i + q_i \times N + a_i \times B)/2^k$.

arithmetic operations involving one word of each input operand. Thus, the number of words of each operand is equal to the number of Processing Elements. Figure 2 shows the arithmetic operations flowchart within each processing element.

According to Figure 2, the multiplication between a_i and B_i words returns a $2k$ bits result, where the least significant bits of this multiplication are added to the least significant bits of the $q_i \times N_i$ multiplication result. Finally, the least significant bits of this add are also added to a k bits word of the S result of the previous iteration. The carry signals propagated to the next Processing Element are the most significant bits of the two multiplications and the most significant bits of the last addition.

4.1.1. First Processing Element. The first Processing Element (PE1) establishes communication with the control block and receives a_i and q_i words at each Montgomery Algorithm iteration. This PE differs from the other Processing Elements because it does not receive any carry signal as input and it discards the first word of the S result, which means the division of $S_{i+1} = (S_i + q_i \times N + a_i \times B)$ by 2^k. The zero index words of B and N (N_0 and B_0) are also provided to this first Processing Element. The internal architecture of PE1 is shown in Figure 3.

4.1.2. General Processing Element. The other Processing Elements are different from PE1 because they have a word from the S result as output and they also transmit and receive carry signals of the multi-precision multiplications and additions. Each Processing Element is activated by the previous Processing Element when the latter finishes its calculation and sends out its carry signals, which means that the architecture works with a pipeline behaviour. Only the last Processing Element provides two words of the S result as a response at each iteration of Algorithm 1 because the S^{m-1} word is obtained with a sum of carry signals. By avoiding a new Processing Element instantiation juts to perform this sum, it is calculated in the last Processing Element. Figure 4 presents the internal architecture of the general Processing Elements.

4.1.3. Quotient Block. At each iteration of Algorithm 1, line 3 presents the q_i quotient computation so that $S + a_i * B + q_i * N$ becomes a multiple of 2^k. The internal architecture of the quotient block is shown in Figure 5. This structure has a combinational behaviour where the q_i result is obtained in one clock cycle. S_0, a_i, B_0, and N' are k bits words which are provided for this block at each iteration of Algorithm 1.

The zero index of B and S means that these words contain the k least significant bits (LSBs) of B and S operands, respectively. As we can see in the right side of Figure 5, a multiplication between a_i and b_0 will provide a $2k$ bits result. Just the LSB part of this result is used in the next operation. Another input of the quotient block, S_0, is then added to the LSB part obtained from the first multiplication. Again, we only need the LSB part of this addition, which is finally multiplied by N', which corresponds to the modular inverse of N modulo 2^k. The LSB part of this last multiplication is the q_i desired result. As seen in Algorithm 1, the numerical basis is power of 2, so for hardware architecture, the mod 2^k operation is simply performed by a right shift operation (LSB selection).

So, the complexity of the quotient block relies on two single precision multiplications and one single precision addition. To evaluate the number of clock cycles for a modular multiplication, we have to consider the first m cycles to read the A and B operands from RAM memories for a square or modular multiplication, respectively. The first iteration of Algorithm 1 also needs m clock cycles. The remaining iterations of Algorithm 1 are performed in $4 * m$ clock cycles.

4.2. The Multiplexed Systolic Architecture. As seen in the previous section, the systolic architecture presents a one-dimensional array of Processing Elements, and each PE is responsible for operations of addition and multiplication. When the numerical basis (2^k) is high (2^{16}, 2^{32}), the internal multiplications become more complex, mainly if the design is applied to an FPGA or an ASIC. So, as the number of multipliers increases, the physical limitations will increase proportionally, for example, in the maximum clock frequency, area, (etc.).

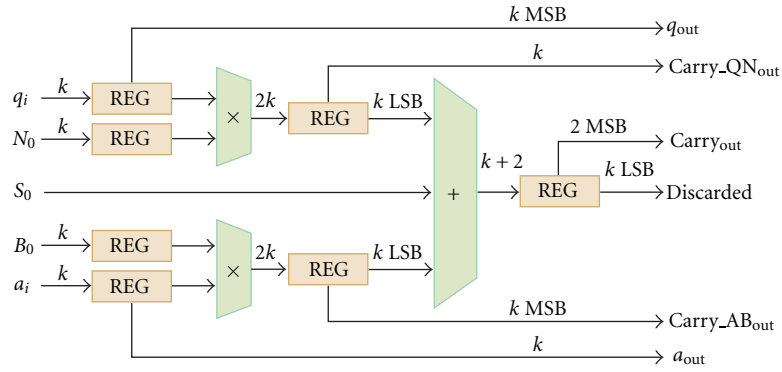

Figure 3: First Processing Element internal architecture.

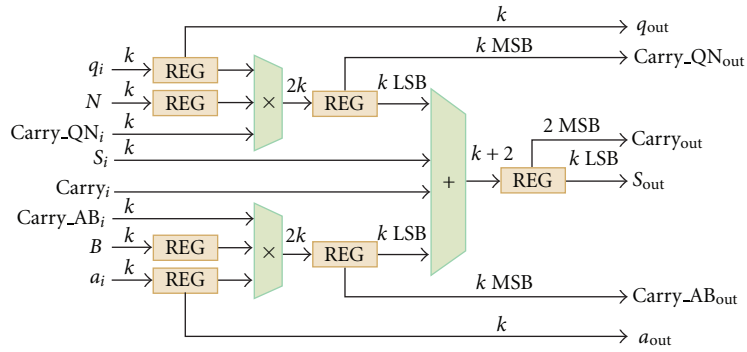

Figure 4: General processing element internal architecture.

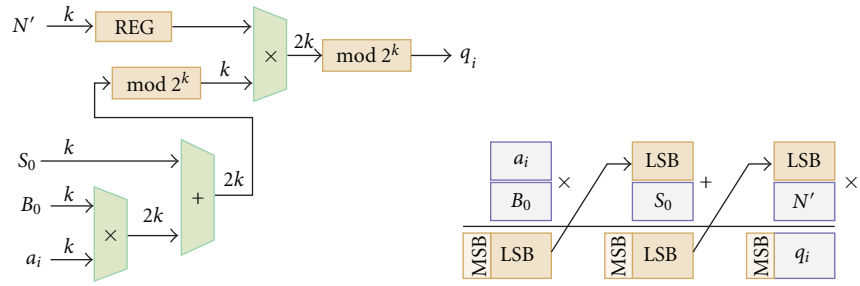

Figure 5: Internal architecture of the quotient block.

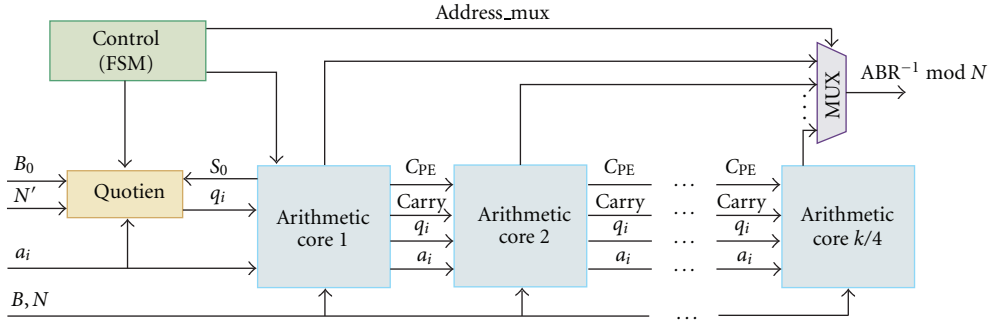

Figure 6: Proposed multiplexed modular multiplication architecture.

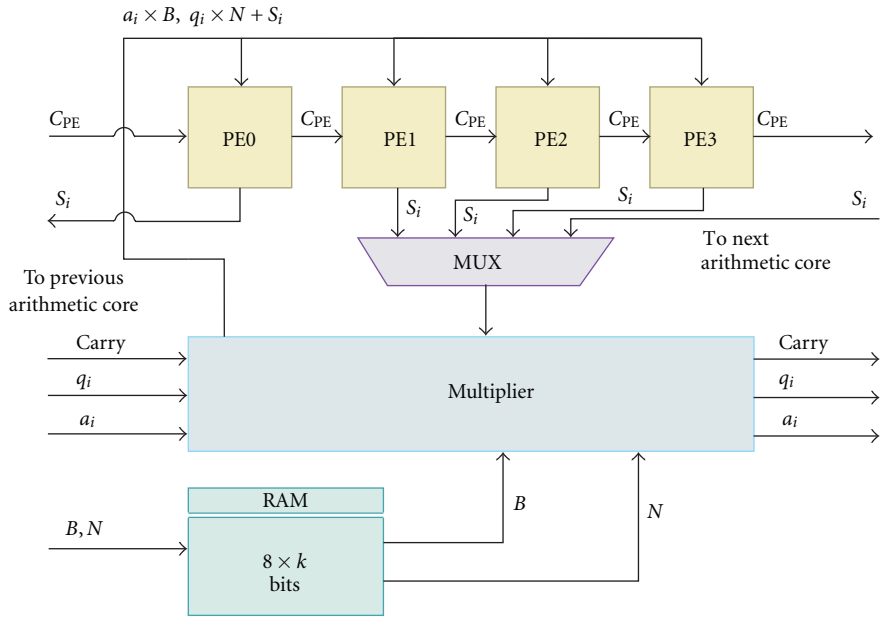

Figure 7: The Arithmetic core architecture.

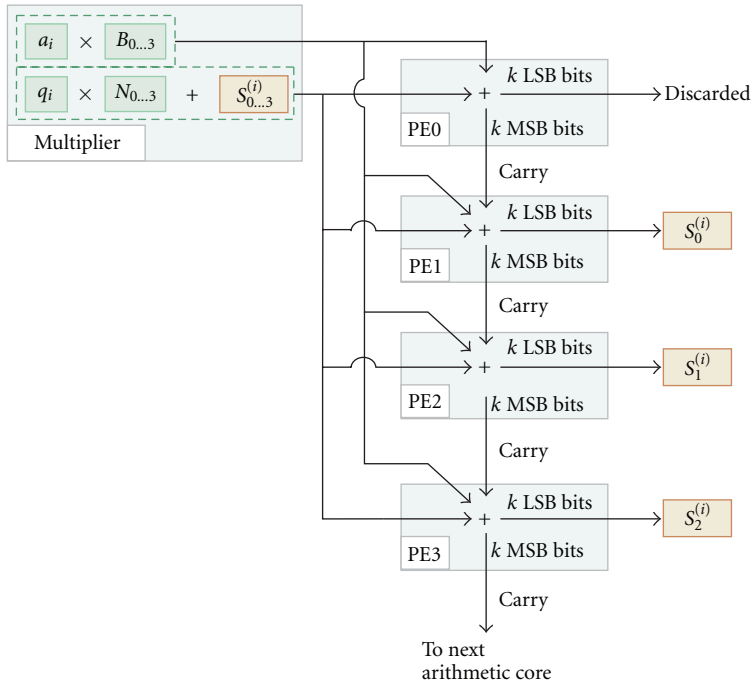

Figure 8: Arithmetic operations performed in arithmetic core 1.

Based on these constraints, a multiplexed and systolic architecture with multiplier blocks working parallel to the Processing Elements is presented in this section. It provides a migration of $k \times k$ bits multipliers from the Processing Elements to the multipliers blocks. Each multiplier block, together with the four Processing Elements, forms an arithmetic core. The one-dimensional arrangement of these arithmetic cores forms the structure of the modular multiplication architecture. Figures 6 and 7 show the multiplexed and systolic architecture and the arithmetic core structure, respectively.

The multiplexed architecture is composed of exactly $k/4$ arithmetic cores, and the first one is managed by a control block designed by a finite state machine. According to Figure 7, each arithmetic core contains four Processing Elements, a multiplier, and an $8 \times k$ bits RAM memory. Being a multi-precision arithmetic architecture, the number of Processing Elements is equivalent to the number of words

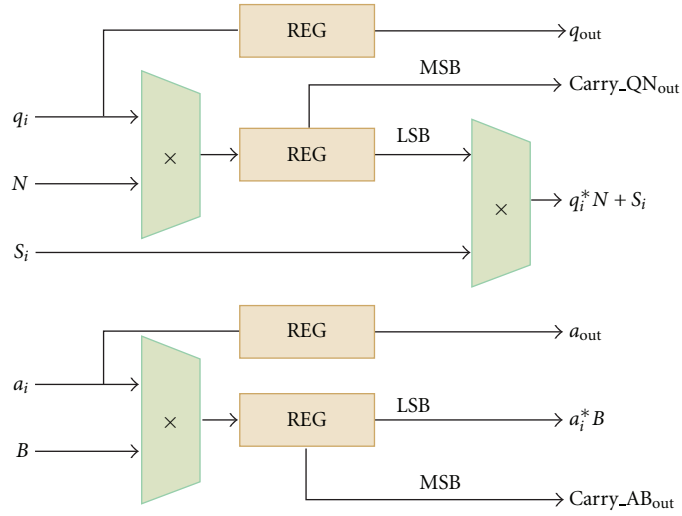

Figure 9: Multiplier block architecture.

in each input operand. So, the RAM memory placed in each arithmetic core stores four words of B and N operands.

The multiplier block performs the $q_i \times N_i$ and $a_i \times B_i$ multiplications. The least significant bits of $q_i \times N_i$ multiplication are added to a k bits word of previous S_i result. The least significant bits of this add operation and the least significant bits of the $a_i \times B_i$ multiplication are sent to the Processing Elements to be added. The Processing Elements provide the S words of the current iteration result. Figure 8 illustrates the executions performed by Arithmetic Core 1. By analysing this illustration, we can realize that instead of having two single precision multiplications in each Processing Element, there is a multiplier block that performs all single precision multiplications for a total of four Processing Elements. In other words, the quantity of single precision multiplications is reduced four times. With these improvements, each Processing Element needs to perform just one addition.

The calculation of the quotient q_i is performed by a block with architecture that is identical to that of the quotient block presented in the systolic architecture.

The Montgomery Algorithm's multiplications are made by a multiplier block that utilizes the multipliers available in the FPGA. The internal architecture of the multiplier blocks is shown in Figure 9.

The carry signals propagated inside the multiplexed architecture are the k most significant bits of the $q_i \cdot N$ and $S_i + a_i \cdot B$ operations presented in Algorithm 1 and are propagated between the multiplier blocks. The last multiplier block sends its carry signals to the fourth and final Processing Element placed in the last arithmetic core. The other carry signal, C_{PE}, is the most significant bits of the result of the addition between the $q_i \cdot N$ and $S_i + a_i \cdot B$ terms. This last addition is performed by the Processing Elements.

At the end of the $m - 1$ iteration, the $S_{i+1} = A \cdot B \cdot R^{-1} \bmod N$ is sent out by an $m : 1$ k bits multiplexer. This result is sent to the memory that is part of the modular exponentiation architecture (described in the next section).

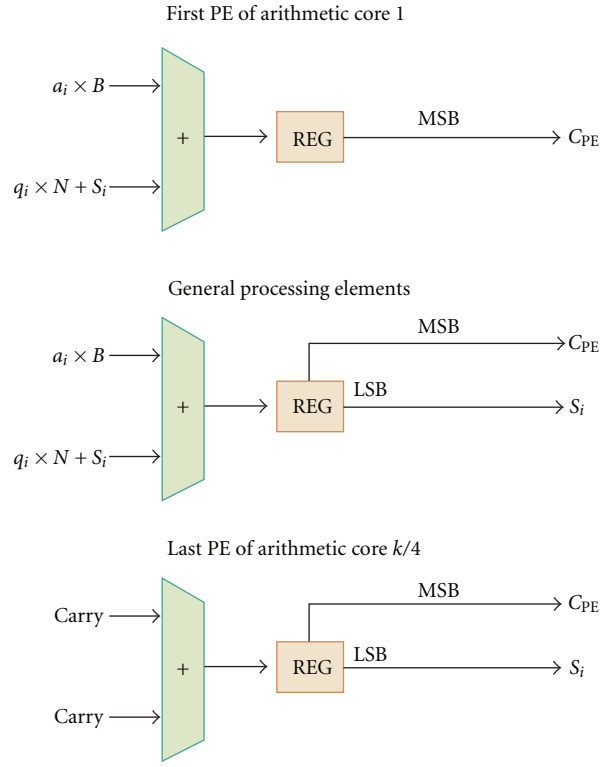

Figure 10: General PE with the carry propagation.

In terms of clock cycles for the Montgomery modular multiplication, we can define the following: initially, m clock cycles are reserved for B operand internal storage. This operand is read from RAM memories. Considering that the N modulus is already available on internal RAM memories placed in arithmetic cores, the first iteration also takes m clock cycles and, it takes the architecture $6 \times m$ clock cycles to perform the remaining iterations of Montgomery Algorithm.

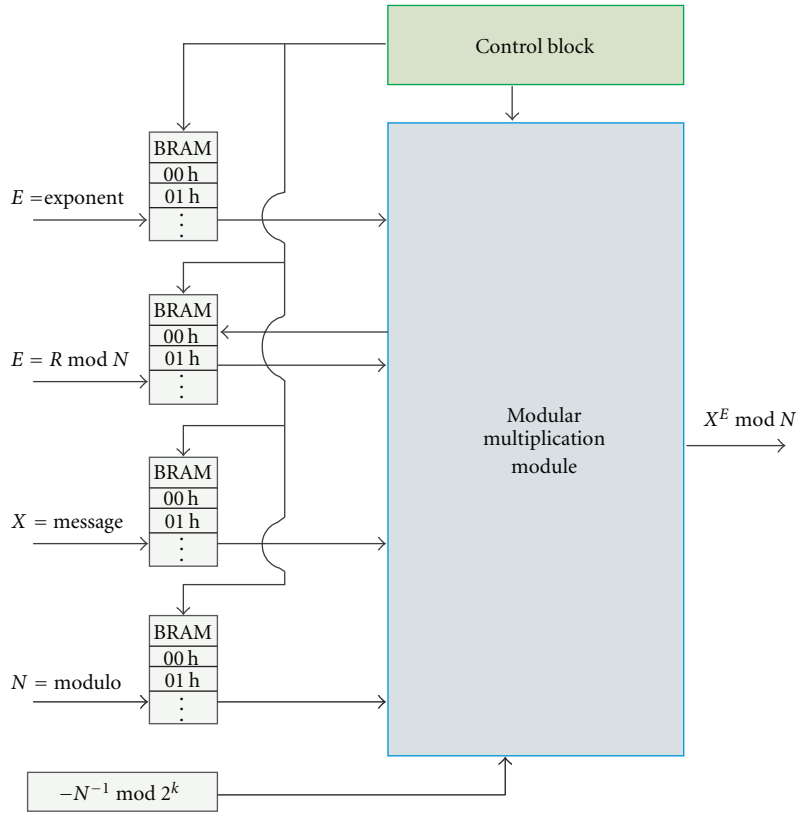

Figure 11: Modular exponentiation architecture.

Thus, the total number of clock cycles, for a modular (or squared) multiplication is $n_{\mathrm{MM}} = m + m + 6 \times m = 8m$.

4.2.1. The Processing Elements PEs. The proposed modular multiplication architecture is composed of m Processing Elements (where m is the number of words of the operands and also the number of iteration on Algorithm 1). Due to the placement of a multiplier block in each arithmetic core, each Processing Element needs to perform just one addition between two $2k$ bits words and sends out a word of S_{i+1} result at each iteration of the Algorithm 1. The first Processing Element must discard the least significant bits of its first addition in order to perform the right shift operation, which corresponds to the division of $(S_i + q_i \times N + a_i \times B)$ by 2^k.

The remaining Processing Elements perform the addition between $(S_i + a_i \times B)$ and $q_i \times N$ terms and the resultant k least significant bits word of this addition are sent out as a word of the S result. The $k + 1$ most significant bits are sent to the next Processing Element as a carry signal. The last Processing Element (PE_m) is responsible for providing two words of S result (S^{m-2} and S^{m-1}), considering that the input words for S^{m-1} calculus are the carry signals from the last multiplier block. Figure 10 shows the first, general case and the last Processing Elements.

4.3. Modular Exponentiation. For a real cryptographic application concerning the RSA algorithm, a modular exponentiation structure that incorporates the modular multiplication architecture is proposed in this section. The modular exponentiation algorithm used in this work is left-to-right square and multiply [13], and thus in average $1.5 * n$ modular multiplications (including squares and multiplies executions) are performed to achieve the final exponentiation result, which n is the operand's precision. Algorithm 2 shows the Montgomery modular exponentiation algorithm.

Require: $E = \sum_{i=0}^{n-1} e_i 2^i$, $e_i \in \{0, 1\}$.
retun: $A = X^E \bmod N$
$A = R \bmod (N)$
$\overline{X} = \mathrm{mont}(X, R^2 \bmod (N))$
for $i = n - 1$ to 0 **do**
$A = \mathrm{mont}(A, A)$
if $e_i = 1$ **then**
$A = \mathrm{mont}(A, \overline{X})$
end if
end for
$A = \mathrm{mont}(A, 1)$

Algorithm 2: Montgomery modular exponentiation—square and multiply.

Four Block RAM memories generated through *Xilinx Coregen* tool were placed to store the input operands of size n. These input operands are the N modulus, the E exponent, the message X in the Montgomery domain

Table 1: Proposed architectures synthesis.

Virtex-4						
n	k	Slices	Clock cycles	DSP48	Freq. (MHz)	BRAM (Bytes)
Systolic architecture						
512	16	3322	192	68	110	128
512	32	4199	96	36	78	128
1024	16	7012	384	130	110	256
Multiplexed architecture						
512	16	2199	256	32	120	256
512	32	2499	128	32	80	256
1024	16	4876	512	64	120	512
1024	32	5118	256	64	80	512
Virtex-5						
Systolic architecture						
512	16	3205	192	68	130	128
512	32	3876	96	36	95	128
1024	16	6642	384	130	130	256
Multiplexed architecture						
512	16	2078	256	32	120	256
512	32	2370	128	32	90	256
1024	16	4876	512	64	120	512
1024	32	5005	256	64	90	512

Table 2: RSA application (Virtex-5).

n	Freq. (MHz)	RSA decryption	Clock cycles
Systolic Architecture			
1024	130	3.23 ms	491520
Multiplexed Architecture			
1024	90	4.36 ms	393216

Table 3: State-of-art implementations of modular multiplication architectures.

Design	FPGA	Clock	Area	Mod exp
Systolic	XC5VLX110T	130 MHz	6642 Slices	3.23 ms
Multiplexed	XC5VLX110T	90 MHz	5005 Slices	4.36 ms
[5]	XV2VP70	101.86 MHz	5709 Slices	3.01 ms
[12]	XC5VLX110T	95 MHz	3044 Slices	6 ms
[4]	XC2V2000	248 MHz	4051 Slices	9.4 ms
[1]	Virtex-4	150.5 MHz	2613 Slices	13.94 ms

($X = X \cdot R \mod N$), and an auxiliary term $A = R \bmod N \cdot A$ control block with a finite state machine manages the read and write operations from the memories (see Figure 11).

The results of the successive modular multiplications are stored in the RAM memory that previously has stored the $A = R \bmod N$ operand, because this operand is necessary just in the first square execution.

5. Results

Table 1 summarizes the FPGA synthesis results of two proposed modular multiplication architectures. The designs were described in hardware description languages (VHDL and Verilog) and synthesized for Virtex-4 and Virtex-5 Xilinx FPGAs. All results are postimplementation, and no area or speed optimizations were set for the synthesis. The results presented in this paper are improvements when compared with our previous work [12]. The multiplexed architecture is implemented with a reduced number of slices registers and DSP48s. However, synthesis for the systolic architecture presented high clock frequencies.

Table 2 presents an RSA encryption and decryption applications of the proposed architectures. Since the modular exponentiation is performed by successive modular multiplication executions, the left-to-right (MSB) binary square and multiply algorithm was employed in the modular exponentiation. The results show that, considering the amount of clock cycles for a modular multiplication execution, the multiplexed architecture is faster than the systolic implementation. On the other hand, the systolic architecture has a clock frequency higher than the clock frequency presented by the multiplexed architecture.

Table 3 shows a state-of-art comparison with our results. Every work referred in this table used the Montgomery Algorithm for their hardware modular multiplication architectures, and for a direct comparison with our approaches just 1024 bits applications are exposed. The time of modular multiplications, when not explained in the references, are estimated considering a modular exponentiation of

$n = 1024$ bits through the Square and Multiply algorithm, running $1.5n$ modular multiplications.

6. Conclusion

This paper presented two Montgomery modular multiplication architectures and the results of their synthesis for Xilinx Virtex-4 and Virtex-5 FPGAs. A systolic implementation and a multiplexed implementation, suitable for RSA public-key cryptosystem, were developed, and the designs were carefully matched with features of the FPGAs, utilizing embedded DSP48Es Slices and Block RAM. The designs are improvements of a previous work. The multiplexed implementation presented a good performance considering *time* $\times$ *area* efficiency. The systolic architecture can run the 1024 bits RSA decryption process in 3.23 ms, and the multiplexed implementation executes the same operation in 4.36 ms. Because of the multiplexed approach, the architecture is scalable. If the key size increases, the architecture can be easily modified by adding arithmetic cores, keeping the performance. Another speed improvement can be achieved by using a parallel modular exponentiation algorithm, for example, the Montgomery Powering Ladder [18] where a full modular exponentiation would be performed in exactly $n \times n_{\mathrm{MM}}$ clock cycles, that is, 33% faster than square and multiply algorithm.

Acknowledgment

This paper is result of project "INOVALAB-Laboratories Technological Innovation in Electronic and Microelectronic". We acknowledge the financial support received from FINEP.

References

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

[2] P. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[3] T. Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," in *Proceedings of the Symposium on Computer Arithmetic (ARITH '99)*, pp. 70–77, IEEE Computer Society, Adelaide, Australia, 1999.

[4] N. Pinckney and D. M. Harris, "Parallelized radix-4 scalable montgomery multipliers," in *Proceedings of the 20th Symposium on Integrated Circuits and System Design (SBCCI '07)*, pp. 306–311, 2007.

[5] C. McIvor and J. V. McCanny, "High-radix systolic modular multiplication on reconfigurable hardware," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, vol. 2005, pp. 13–18, 2005.

[6] C. D. Walter, "Systolic modular multiplication," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 376–378, 1993.

[7] F. Bernard, "Scalable hardware implementing high-radix Montgomery multiplication algorithm," *Journal of Systems Architecture*, vol. 53, no. 2-3, pp. 117–126, 2007.

[8] Y. Wang, D. L. Maskell, and J. Leiwo, "A unified architecture for a public key cryptographic coprocessor," *Journal of Systems Architecture*, vol. 54, no. 10, pp. 1004–1016, 2008.

[9] A. Daly and W. Marnane, "Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," in *Proceedings of the ACM/SIGDA 10th International Symposium on Field Programmable Gate Arrays (FPGA '02)*, pp. 40–49, 2002.

[10] D. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, Mass, USA, 1951.

[11] Xilinx, "Virtex-5 user guide," March 2006, http://www.xilinx.com.

[12] G. Perin, D. G. Mesquita, F. L. Herrmann, and J. B. Martins, "Montgomery modular multiplication on reconfigurable hardware: fully systolic array vs parallel implementation," in *Proceedings of the 6th Southern Programmable Logic Conference (SPL '10)*, pp. 61–66, 2010.

[13] A. Menezes, *Handbook of Applied Cryptography*, CRC Press, New York, NY, USA, 5th edition, 2001.

[14] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," in *Proceedings of the 12th Symposium on Computer Arithmetic (ARITH '95)*, pp. 193–199, July 1995.

[15] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, 1999.

[16] A. F. Tenca and C. K. Koç, "A scalable architecture for modular multiplication based on montgomery's algorithm," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1215–1220, 2003.

[17] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An improved unified scalable radix-2 montgomery multiplier," in *Proceedings of the Symposium on Computer Arithmetic (ARITH '05)*, pp. 172–178, 2005.

[18] M. Joye and S. M. Yen, "The montgomery powering ladder," in *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES '02)*, vol. 2523 of *Lecture Notes in Computer Science*, pp. 291–302, 2003.