Parametric, Secure and Compact Implementation of RSA on FPGA

Ersin Öksüzoğlu, Erkay Savaş
Sabanci University, Istanbul, TURKEY
ersino@su.sabanciuniv.edu, erkays@sabanciuniv.edu

Abstract

We present a fast, efficient, and parameterized modular multiplier and a secure exponentiation circuit especially intended for FPGAs on the low end of the price range. The design utilizes dedicated block multipliers as the main functional unit and Block-RAM as storage unit for the operands. The adopted design methodology allows adjusting the number of multipliers, the radix used in the multipliers, and number of words to meet the system requirements such as available resources, precision and timing constraints. The architecture, based on the Montgomery modular multiplication algorithm, utilizes a pipelining technique that allows concurrent operation of hardwired multipliers. Our design completes 1020-bit and 2040-bit modular multiplications in 7.62 µs and 27.0 µs, respectively. The multiplier uses a moderate amount of system resources while achieving the best area-time product in literature. 2040-bit modular exponentiation engine can easily fit into Xilinx Spartan-3E 500; moreover the exponentiation circuit withstands known side channel attacks.

1. Introduction

The ASIC's and FPGA's are two commonly used hardware devices for cryptographic implementations where the latter has become more and more popular recently since it is reconfigurable and relatively easy to access from economical and usability point of view. Therefore, some of the previous works utilize resource rich, but relatively expensive FPGA devices to design fast multipliers. There is, however, a paucity of interests in the implementation of multipliers on the smallest and the most economically accessible FPGA devices such as Xilinx Spartan 3 series [6].

As our dependency on public key cryptography is increasing at an impressive rate even on the simplest devices such as car keys and identity cards, there is a great initiative to design fast modular multipliers, which is the most resource-consuming operation in RSA [1], for the most inexpensive devices. Xilinx Spartan 3 FPGAs, one of the most economical reconfigurable devices in the market, make the implementations financially viable and shortens the time-to-market period.

Koc et al. [5] proposed several algorithms to implement the Montgomery multiplication [2] operation in software. These algorithms also prove to be useful for hardware implementations when fast block multipliers are available as in the case of many FPGAs. Moreover, these multipliers can work in a pipelined fashion to take advantage of massive parallelism, despite the fact that these software algorithms are originally designed for a single multiplier available in general-purpose processors.

Previous studies, with ASIC implementations in mind, generally avoid using multipliers which consume a considerable amount of chip space, and have a long combinational delay. Instead, they perform multiplication by repeated addition through carry-save adders. Although the repeated addition approach seems to be a reasonable solution for ASIC realizations, the FPGA's have a different inner structure that allows us to implement alternative circuits. For instance, a recent work by *Suzuki* [3] successfully utilizes powerful DSP macro cells available on an expensive FPGA device to achieve the best execution time for multiplication and exponentiation operations.

The paper is organized as follows. Firstly, we specify the design requirements and give the details about our architecture. In the next sections, we discuss the simulation and synthesis results. In section 5, we compare the performances of previous works with ours. The compatibility problems of our design for practical applications are discussed in section 6. The last section summarizes the features of the implementation and highlights our contributions.

2. Architecture

It is essential to lay out the design criteria to meet the challenges and requirements of the target application. These criteria for RSA implementation on reconfigurable hardware can be enumerated as follows:

1) The design must be flexible to fit in both small and large FPGA's efficiently with adjustable number of processing elements.
2) The bit-length of the words must be parametric so that the full performance of multipliers is utilized.

[1] This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project number 105E089 (TUBITAK Career Award).

3) The design must be scalable to work with operands of virtually any length (e.g. 2048 bit, 4096 bit, etc.)
4) 2048-bit exponentiation engine must easily fit into even a smallest FPGA with a good timing performance.
5) The implementation must resist against side channel attacks with minimal overhead.
6) All hardwired multipliers must work at maximum possible frequency (They are instantiated as registered multipliers).
7) All variables for operands must be kept in Block-RAM to ensure minimum area consumption.
8) The connection network must be simple yet effective.

2.2. CIOS method for Montgomery multiplication

While all multi-precision Montgomery multiplication algorithms analyzed in [5] require the same number of word-level multiplications, the number of additions and memory requirements slightly differ. The CIOS method (Algorithm 1 in Figure 1), seems to be the best choice for hardware implementation since it has a regular execution pattern and needs only a memory space of s+3 words (the least among the others) where s is the number of words in one operand. Likewise, *McIvor et al.* [4] also conclude that the CIOS method provides the fastest timing results for FPGA implementations.

As the CIOS method is specifically designed for software implementations, we need to modify it for efficient execution in hardware by taking advantage of parallelization through dedicated block multipliers. The execution graph of algorithm modified for pipelined computation is depicted in Figure 4. The circuit consists of processing elements (PE, *cf*. Figure 2) which are responsible for executing a single iteration of the loops in Steps 2 and 8 of Algorithm 1. These steps are performed together within the same PE; therefore each PE is made of two multipliers, two adders and six registers. Once PE_0 generates the first word of the intermediate result (i.e. the least significant word), the next processing unit PE_1 concurrently starts the computation for the second iteration of the loop with the values it obtains from PE_0. When a PE finishes the computation for an iteration it is immediately assigned to the next available iteration. The results of the last PE are kept in dual-port Block-RAM.

2.3. Implementation Details

Before the execution of each iteration of the loop (at each increment of the loop counter "*i*"), the value "*m*" must be calculated as shown in Step 6 in the CIOS method. (The value of "n_0^{-1}" is calculated offline (only one word) and fixed as long as the modulus does not change). However, meanwhile, other PEs are still performing multiplication operation; therefore to maintain a continuous data flow, we need to insert FIFO buffers among the PEs and compensate for the time lost by this pre-calculation step. After "*m*" is ready, there are two important steps remaining for execution: Steps 2.a (multiplication) and 8.a (reduction).

As only one word per cycle can be requested from each Block-RAM, only the first PE directly receives data from Block-RAMs; and similarly only the last PE writes the result words t_i to the Block-RAM. All PEs forward "used input variables" (i.e. a_j and n_j) and the *sum S* to the next PE to exploit data reuse and simplify connection network.

2.4. Parametric Design

We can adjust the Montgomery multiplier to meet the application requirements or to utilize a given FPGA device efficiently by changing the following three parameters at the compile time:

1) Number of PEs (p)**:** Total number of PEs is the main area vs. performance trade-off metric. The proposed design must have at least two processing elements since the first and last processing elements are hardwired to RAM. In other words, total number of block multipliers must be at least four. The upper bound for p is determined with the number of hardwired multipliers of the target FPGA, which is 10 in our case (i.e. 20 block multipliers in total).
2) *Radix*(R)**:** This parameter determines the bit length of the hardwired multipliers and adders shown in Figure 2. As the radix closely relates to the maximum combinational path delay in the adder design, it has a direct effect on the frequency. This parameter must be adjustable to take full advantage of the block multipliers in a given device to achieve the best timing performance.
3) Number of words (s): The radix and the number of words in each operand together determine the bit-length of the operands; for instance, for k = 2048-bit operands and the radix R = 16, the number of words s is 128. The number of words determines also the depth of the Block-RAM.

3. Simulation Results

The clock cycles required for one multiplication heavily depends on the number of PEs. More PEs result in faster designs as expected. However, multiplier utilization decreases when the number of PEs increases. Similarly, using longer words also has a negative effect on the frequency due to longer carry chains in adders used within PEs.

Table 1 shows the exact cycle count for one modular multiplication including the loading time of operands from the Block-RAM. The multiplication circuit has the following timings:

- After start signal is asserted, it takes 9 cycles for the first PE to yield the first word of the result.
- The number of clock cycles spent between the appearances of the first word of the results in consecutive PEs is 9.

The overall cycle count can be approximated (with error margin of less than 5%) using the following formula

$$CC = \frac{s \times max\{(14+s),(12+9p)\}}{p} \approx \frac{s(14+s)}{p}$$

for large s, where CC, p, and s stand for the total clock cycles, the number of processing elements, and the number of words, respectively. As indicated in [5], the CIOS method requires

$2s^2+s$ word multiplications. If there were no data dependencies, the required clock cycles would be $(2s^2+s)/(2p)$, which is not significantly different from what our design achieves. This clearly shows that our mapping of the CIOS algorithm to hardware is near optimal.

Another important issue is how profitably the allocated resources are used in the implementation; an issue is referred as *utilization*. As can be seen in Table 2, PE utilization is quite high for precision of interest for RSA. For instance, the PE utilization is over 85% for 2040-bit or larger operands as can be seen in Table 2.

4. Synthesis Results

For synthesis, we use XST (Xilinx Synthesis Tool) from Xilinx ISE v9.1 package and the target device is Xilinx 3s500e-4FG320C whose properties are given in [6]. The synthesis values are obtained before PAR stage; so they have an error margin of 5%.

Table 3 shows the resource usage for different number of processing elements from two to ten. As can be observed in the table, the resource usage is modest even for the maximum configuration with the largest number of processing elements.

For 1020-bit or longer operands, a multiplication engine with 4, 5 and 6 PEs offer the lowest time-area product (*cf.* Table 4). The 510-bit key is obsolete; however, we include it for comparison purpose.

With five PEs per multiplication core, we can fit two cores into the same FPGA, to take full advantage of the parallelism in Montgomery Powering Ladder (Algorithm 2 *cf.* Figure 3) [13] which we choose as the exponentiation algorithm, since it offers protection against Simple Power Analysis (SPA). The exponentiation circuit with and without DPA countermeasure (we used exponent blinding method against "the Doubling Attacks" depicted by *Yen el al.* [9]) are synthesized (5 PE × 2) with speed optimization and the results are illustrated in Table 5. The area consumption and the frequency stay approximately the same for larger bit-lengths. The second circuit has a $(1/s\times100)$ percent cycle overhead due to the DPA protection.

5. Performance Analysis

In this section, we provide a comparative analysis of the proposed design with respect to other designs synthesized for various FPGA technologies in literature.

Table 6 summarizes the resource usage and performance of various FPGA designs and the proposed one. Although the proposed design is not the fastest circuit, its execution speed outperforms many others; moreover, it is superior to the others in terms of time-area product.

We do not have the entire performance and area details concerning the multiplication units for designs in [3, 16, 17]; however, their exponentiation timings and areas are available. Our exponentiation engine has DPA and SPA protection, which the other designs lack and our execution time is fixed for a given bit-length.

Our foremost design goal is not achieving the best timing, but the best time-area product on an inexpensive FPGA. The gap in performances can be attributed to the following factors favoring the designs in [3] and [17]: **i)** More advanced (and expensive) FPGA, **ii)** more resource usage, **iii)** higher clock frequency (favoring only [3]), **iv)** powerful DSP cells (favoring only [3]), and finally **v)** special acceleration techniques [3] used for exponentiation that we do not employ.

Considering that the proposed circuit is intended for a low-end device, the achieved exponentiation speed is so far the record for a very low-price FPGA device to best of our knowledge, and is satisfactory for many applications. In Table 6 various designs are mapped onto FPGAs with different speed grades and features; e.g., the multiplier in [3] uses built-in DSP cells, which are available neither in our target device nor in many other FPGA devices. In this work, we try to use the maximum potential available on one of the smallest FPGAs; therefore, the time-area product is the vital criterion for us.

As shown in Table 6, the authors in [7] present two designs; one is based on radix-2 and the other on radix-4, and they both use the distributed RAM as the main storage element and are non-pipelined. Table 7 shows performance of exponentiation engine for approximately 1024-bit RSA.

We cannot directly use the execution times for comparison purposes (due to the technological differences); instead we can use "total clock cycles required for one modular multiplication" as the performance indicator. Table 8 shows that the proposed design achieves the best {time×area} metric, which is an indication of good design and high utilization of the target device.

6. Compatibility Problems

As we use 17-bit×17-bit multipliers in the design to take the full advantage of given features of the FPGA chip, the implemented bit lengths are smaller than the widely employed ones that are the exact powers of 2 (e.g. 512-bit, 1024-bit, 2048-bit, etc). The security level provided by a 1020-bit implementation is approximately the same with 1024-bit implementation; however there can be compatibility problems between 1024-bit and 1020-bit circuits in practice. While the number of words in compatible versions of our circuits is one more than in the previously mentioned designs, there will be no change in the frequency and the area. The average slowdown ratio due to compatibility is 3.6 % in the number of clock cycles.

7. Conclusion

We designed a fast, efficient and parameterized modular multiplier and a secure exponentiation circuit for simple FPGA devices. The price of intended FPGAs is at least one order of magnitude less than other devices used in some of the previous works, where the primary purpose is to achieve the fastest execution in modular exponentiation. It is true that the speed is always of an important concern; however, the price of the device used for the realization is also an issue in many

applications and there is not much work in this direction. We intended to fill this gap with our design, which achieves the best time-area product to the best of our knowledge in this category.

Our target technology, Xilinx Spartan 3E-500, is a cost-effective solution in many aspects, especially the use of the 90 nm technology significantly reduces the die size, cost and the total power consumption, while increasing the frequency; and therefore it is one of the best choices for practical applications, where the manufacturing cost is the primary concern.

The proposed multiplier is parametric, and therefore can be used for virtually any bit-length, where the upper limit on precision is dictated only by the capacity of Block-RAM available on the device. Since the most popular public key cryptosystem nowadays is RSA [1] (the design can be also used for Diffie-Hellman Key Exchange [12]), we focused on the designs with 1020-bit and 2040-bit key sizes; the latter precision will be favored over the former in the near future due to increased security concerns. Our design completes one 1020-bit and 2040-bit modular multiplication in 7.62 μs and 27.0 μs, respectively with approximately the same device usage. The timing performance achieved for multiplication is either comparable or superior to most of the other designs in the literature despite the low resources available on the target device.

We also achieved to fit 2040-bit exponentiation circuit into the same device. Few designs in literature *can* outperform our design *only* by using more resources, better and expensive devices, and acceleration techniques for the exponentiation. From practical point of view, our exponentiation circuit also resists against all known side-channel attacks explained in [13] and [9] (namely SPA, DPA, fault attacks and (n-1) attacks) with minimal overhead.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signature and Public-key Cryptosystems," *Comm. ACM*, 21(2), 120-126, 1978.

[2] P. L. Montgomery, "Modular Multiplication without Trial Division," *Math. Computation, vol.* 44, *pp.*519-521, 1985.

[3] D. Suzuki, "How to Maximize the Potential of FPGA resources for Modular Exponentiation", *CHES* 2007, *LNCS* 4727, *pp.* 272-288, 2007.

[4] C. McIvor, M. McLoone, J.V. McCanny. "FPGA Montgomery Multiplier Architectures - a Comparison", 12th IEEE Symposium on Field-Programmable Custom Computing Machines (*FCCM* 2004), *pp.* 279 – 282. April 2004.

[5] Ç. K. Koc, T. Acar, B. S. Kaliski: "Analyzing and Comparing Montgomery Multiplication Algorithms". *IEEE Micro, Vol.* 16, *No.* 3, *pp.* 26-33, June 1996.

[6] Xilinx, Inc.: http://www.xilinx.com, "Xilinx Spartan 3E-500 Data Sheets".

[7] E. Oksuzoglu, E. Savas, "A Fast and Efficient Hardware Implementation of 2048-bit Radix-4 Modular Multiplication Circuit for Public Key Cryptosystems", *submitted*, 2007.

[8] N. Mentens, K. Sakiyama, L.Batina, I. Verbauwhede, B. Preneel, "FPGA-Oriented Secure Data Path Design: Implementation of a Public Key Coprocessor", *FPL* 2006, IEEE, pp. 133-138, 2006.

[9] S. M. Yen, W. C. Lien, S. Moon, and J. Ha, "Power Analysis by Exploiting Chosen Message and Internal Collisions – Vulnerability of Checking Mechanism for RSA-Decryption", *Mycrypt* 2005, *LNCS* 3715, pp. 183–195, 2005.

[10] C. McIvor, M. Mcloone, J. N. McCanny, A. Daly, W. Marnane, "Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures", 37*th Annual Asilomar Conference on Signals, Systems and Computers,* California, 2003.

[11] A. Daly and W. Marnane, "Efficient Architectures for implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic", *in proc. of 10th International symposium on FPGA's*, 2002.

[12] W. Diffie and M. E. Hellman, "New Directions in Cryptography," IEEE Trans. Info. Theory, vol. IT-22, Nov. 1976, pp. 644–54.

[13] M. Joye, S.M. Yen, "The Montgomery Powering Ladder", *Cryptographic Hardware and Embedded Systems – CHES* 2002, *LNCS* 2523, pp. 291–302, Springer-Verlag, 2003.

[14] P. Fournaris, O. Koufopavlou, "A New RSA Encryption Architecture and Hardware Implementation based on Optimized Montgomery Multiplication" *in proceedings of* 2005 *IEEE International Symposium on Circuits and Systems (ISCAS* 2005*),* Kobe, May 23 -26, Japan, 2005.

[15] S. B. Ors, L. Batina, B. Preneel and J. Vandawalle, "Hardware Implementation of a Montgomery Modular Multiplier in a Systolic Array", *International Parallel and Distributed processing symposium (IPDPS '*03*),* 2003.

[16] T. Blum, C. Paar, "High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware", *IEEE Transaction on Computers* 50(7), 759-764 (2001).

[17] S. H. Tang, K. S. Tsui, P. H. W. Leong, "Modular Exponentiation using Parallel Multipliers" , *Proc of the 2003 IEEE International Conference on Field Programmable Technology (FTP* 2003*), pp. 52-59* (2003).

Appendix

Table 1. Clock cycles for modular multiplication

	The Number of PEs					
Bit length-words	**2**	**4**	**5**	**6**	**8**	**10**
4080 (240words)	30256	15154	12139	10132	7630	6136
2040 (120 words)	7936	3994	3211	2692	2050	1672
1020 (60 words)	2176	1114	907	772	638	630
510 (30 words)	646	352	330	326	322	318

Table 2. Utilization ratios for multiplication core

	The Number of PEs					
Bit length-words	**2**	**4**	**5**	**6**	**8**	**10**
4080 (240words)	95.4	95.2	95.1	94.9	94.6	94.1
2040 (120 words)	91.1	90.5	90.1	89.5	88.2	86.5
1020 (60 words)	83.4	81.5	80.0	78.4	71.1	57.6
510 (30 words)	70.8	65.0	55.5	46.8	35.5	28.8

Table 3. Synthesis results for 2040-bit multiplier

	The Number of PEs						
	2	**4**	**5**	**6**	**8**	**10**	**Total**
Slices	679	1260	1553	1838	2435	3028	**4656**
FF	809	1505	1854	2199	2901	3602	**9312**
LUT	1180	2232	2760	3292	4353	5426	**9312**
BRAM	4	4	4	4	4	4	**20**
Mult.	4	8	10	12	16	20	**20**

Table 4. Time-area products: normalized to the smallest

	The Number of PEs					
Bit length-words	**2**	**4**	**5**	**6**	**8**	**10**
4080 (240words)	1.106	1.028	1.015	1.002	**1.000**	**1.000**
2040(120 words)	1.089	1.017	1.008	**1.000**	1.009	1.023
1020 (60 words)	1.0536	**1.000**	1.004	1.011	1.107	1.359
510 (30 words)	**1.000**	1.011	1.168	1.366	1.788	2.195

Table 5. Synthesis results for 1020-bit exponentiation circuit

	SPA protected[1]	**SPA+DPA Protected**[2]
Slices	3799 (81 %)	3899 (83 %)
FF	4416 (47 %)	4493 (48 %)
LUT	6750 (72 %)	6931 (74 %)
Block Ram	14 (70 %)	16 (80 %)
Multipliers	20 (100 %)	20 (100 %)
Frequency	119 MHz	119 MHz
Max Clock Cycles	929,519	946,127
Max Ex Time	7.81 ms	7.95 ms

Table 6. Execution Times for 1024-bit Multiplier

Design	**Technology**	**Freq (MHz)**	**Area**	**Ex. Time (µs)**
[17]	xc2v3000-6	90.11	N/A	1.49
[7] radix-4	xc2v6000	132.4	8328 slices	4.23
Proposed (1020 bit)	**xc3s500e-4FG320C**	**119**	**1553 slices + 10 multipliers**	**7.62**
[14][3]	FPGA (?)	129.1	3611 slices	7.93
[7] radix-2	xc2v6000	137.2	6282 slices	8.21
[10]	xc2v3000	75.23	11617 slices	13.45
[11]	xcv1000	~55	5058 slices	18.67
[15]	V812E-BG-560	~96	5706 slices	32.12

Table 7. Exponentiation Engine Performance for 1024 bit

Design	**Technology**	**Freq.**	**Area**	**Ex. Time (ms)**
[3]	xc4vfx-10sf363	200/400[4]	3937 slices + 17 DSP48	1.71 (max)
[17]	xc2v3000-6	90.11	14334 slices + 62 multipliers	2.33 (avg.)
Prop.(1020 bit)	**xc3s500e**	**119**	**3899 slices + 20 multipliers**	**7.95 (max)**
[7] radix-4	xc2v6000	132.4	9837 slices	8.66 (max)
[7]	xc3s4000	66	18247 slices + 66 multipliers	11.1 (?)
[16]	xc40250xv	45.66	6633 slices	11.95 (max)
[7] radix-2	xc2v6000	137.2	7286 slices	16.8 (max)

Table 8. Time-area products normalized to the proposed design (for ≈1024-bit modular multiplication).

Design	**Area (slices)**	**Clock Cycles**	**Time × Area**
[15]	5706	3072	12.607
[10]	11617	1025	8.564
[7] radix-2	6282	1058	4.780
[11]	5058	1027	3.736
[7] radix-4	8330	560	3.354
[14]	3611	1024	2.659
Proposed	**1553[5](3453)**	**907**	**1.000 (2.223)**

Algorithm 1 – CIOS Montgomery multiplication

Inputs: a_j, b_j, n_j: Operand and modulus words (w bits each), where $a = (a_{s-1}, ...a_1, a_0)$; n_0^{-1}:= multiplicative inverse[6] of n_0

Output: $t = a \times b \times 2^{-k} \bmod n$, where $k = \lceil \log_2 n \rceil$

for $i = 0$ to s-1

1. $C \leftarrow 0$
2. for $j = 0$ to s-1
   - **a.** $\{C, S\} \leftarrow t_j + a_j \times b_i + C$
   - **b.** $t_j \leftarrow S$
3. $\{C, S\} \leftarrow t_s + C$
4. $t_s \leftarrow S$; $t_{s+1} \leftarrow C$
5. $C \leftarrow 0$
6. $m \leftarrow t_0 \times (-n_0^{-1}) \bmod 2^w$
7. $\{C, S\} \leftarrow t_0 + n_0 \times m$
8. for $j = 1$ to s-1
   - **a.** $\{C, S\} \leftarrow t_j + n_j \times m + C$
   - **b.** $t_{j-1} \leftarrow S$
9. $\{C, S\} \leftarrow t_s + C$
10. $t_{s-1} \leftarrow S$
11. $t_s \leftarrow t_s + 1 + C$

Figure 1. CIOS Montgomery multiplication

[1] Montgomery Powering Ladder [13] is used as SPA protection.
[2] Exponent blinding is used for DPA protection.
[3] The authors in [14] use pre-computed values, but the pre-computation unit is not included in the multiplier (it is a part of the exponentiation circuitry).
[4] The control unit is running at 200 MHz, while DSP48 cells at 400 MHz
[5] The area of the hardwired multipliers is included in the total area which is shown in parenthesis.
[6] "Least significant word of inverse of n" in mod 2^k, where $2^{k-1} < n < 2^k$

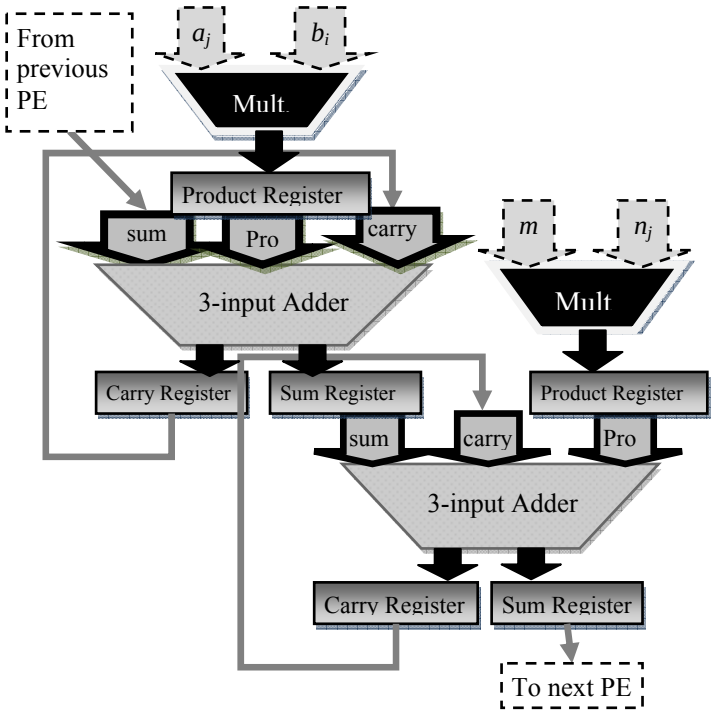

Figure 2. Structure of a processing element

Algorithm 2 – Montgomery powering ladder

Inputs: m: input message,
$d = (d_{k-1},...,d_0)$ exponent.
Output: $C = m^d$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow m$
2. for $i = k$ -1 downto 0
   a. if ($d_i == 1$)
      $R_0 \leftarrow R_0R_1$; $R_1 \leftarrow (R_1)^2$ /* in parallel */
   b. else
      $R_1 \leftarrow R_1R_0$; $R_0 \leftarrow (R_0)^2$ /* in parallel */
3. return R_0

Figure 3. Montgomery powering ladder [13]

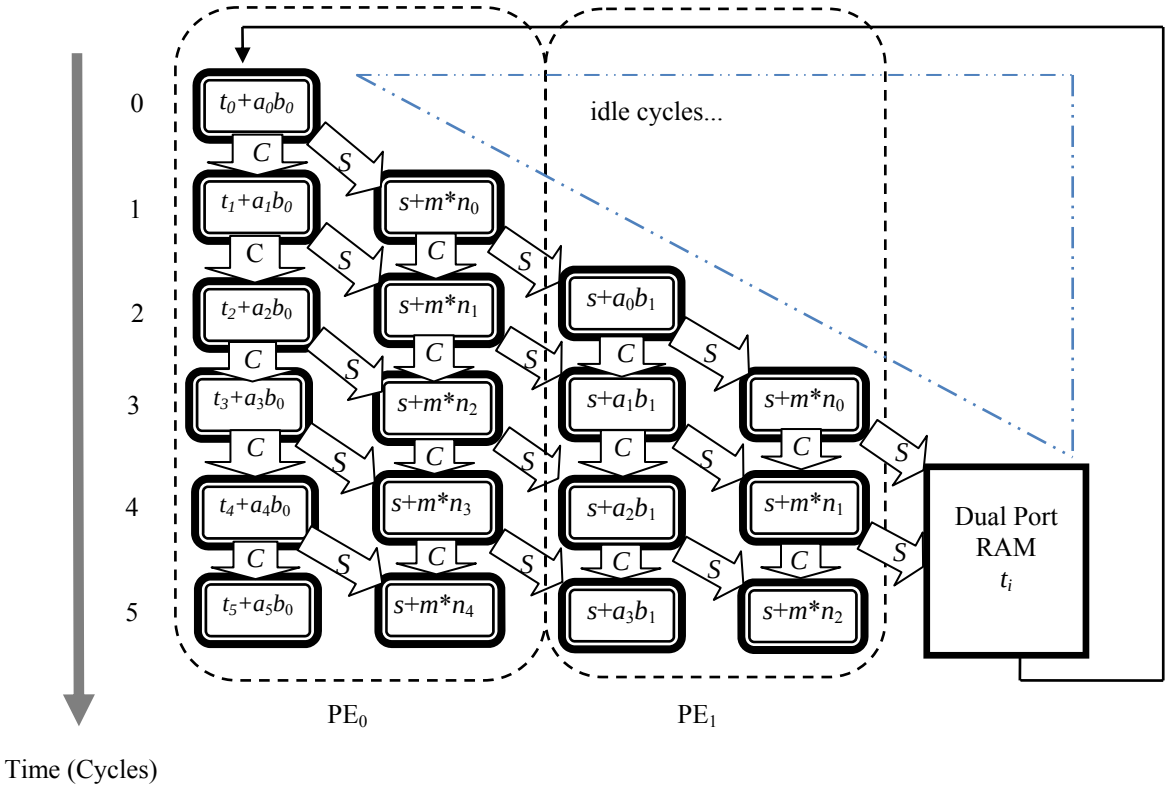

Figure 4. Execution graph of CIOS method