Systolic Hardware Implementation for the Montgomery Modular Multiplication

NADIA NEDJAH AND LUIZA DE MACEDO MOURELLE

Department of de Systems Engineering and Computation,
Faculty of Engineering,
State University of Rio de Janeiro
BRAZIL

Abstract:- Modular multiplication is a cornerstone computation in public-key cryptography systems such as RSA cryptosystem. The operation is time consuming for large operands. This paper describes the characteristics of a systolic array-based architecture to implement modular multiplication using the fast Montgomery algorithm. The paper evaluates the prototype using the time×area classic factor.

Key-Words:- Modular multiplication, Systolic architecture, Cryptosystems.

1 Introduction

An RSA cryptosystem consists of a set of three items: a *modulus* M and two integers d and e called *private* and *public* keys that satisfy the property $T^{de} = T$ mod M. Plain text T obeying $0 \leq T < M$. Messages are encrypted using the public key as $C = T^d$ mod M and decrypted as $T = C^e$ mod M. So the same operation, i.e. modular exponentiation is used to perform both processes: encryption and decryption. It consists of a repetition of modular multiplications. Hardware implementation of the RSA cryptosystem is widely studied as in [1], [2], [3], [4].

The performance of public-key cryptosystems is primarily determined by the implementation efficiency of the modular multiplication and exponentiation. As the operands (the plain text of a message or the cipher or possibly a partially ciphered) text are usually large (i.e. 1024 bits or more), and in order to improve time requirements of the encryption/decryption operations, it is essential to attempt to minimise the number of modular multiplications performed and to reduce the time requirement of a single modular multiplication.

There are various algorithms that implement modular multiplication such as Barrett's and Booth's method [5], [6], and Brickell's algorithm [7]. Here, we concentrate on Montgomery algorithm as it is considered the most popular and the more efficient.

In this paper, we present a hardware prototype for implementing Montgomery modular multiplication with a fully systolic parallel architecture. The prototype reduces time response in detriment of area requirement. However it also improves the area/time product.

The rest of this paper is organised as follows: in Section 2, we describes the Montgomery algorithm used to implement the modular operation; in Section 3, we modify Montgomery algorithm to highlight the systolic nature of the computation, in Section 4, we describe the architecture of the systolic prototype; finally in Section 5, we evaluate the obtained prototype in terms of area and time requirements.

2 The Montgomery Algorithm

Algorithms that formalise the operation of modular multiplication generally consist of two steps: one generates the product $P = A \times B$ and the other reduces this product P modulo M.

The straightforward way to implement a multiplication is based on an iterative adder-accumulator for the generated partial products. However, this solution is quite slow as the final result is only available after n clock cycles, n is the size of the operands [8].

A faster version of the iterative multiplier should add several partial products at once. This could be achieved by *unfolding* the iterative multiplier and yielding a combinatorial circuit that consists of several partial product generators together with several adders that operate in parallel [9], [10], [11].

One of the widely used algorithms for efficient modular multiplication is the Montgomery's algorithm [12]. This algorithm computes the product of two integers modulo a third one without performing division by M. It yields the reduced product using a series of additions

Let A, B and M be the multiplicand and multiplier and the modulus respectively and let n be the number of digit in their binary representation, i.e. the radix is 2. So, we denote A, B and M as follows:

$$A=\sum_{i=0}^{n-1} a_i \times 2^i,\quad B=\sum_{i=0}^{n-1} b_i \times 2^i \text{ and } M=\sum_{i=0}^{n-1} m_i \times 2^i$$

The pre-conditions of the Montgomery algorithm are as follows:

- The modulus M needs to be relatively prime to the *radix*, i.e. there exists no common divisor for M and the radix;
- The multiplicand and the multiplicator need to be smaller than M.

As we use the binary representation of the operands, then the modulus M needs to be odd to satisfy the first pre-condition.

The Montgomery algorithm uses the least significant digit of the accumulating *modular partial product* to determine the multiple of M to subtract. The usual multiplication order is reversed by choosing multiplier digits from least to most significant and shifting down. If R is the current modular partial product, then q is chosen so that $R+q\times M$ is a multiple of the radix r, and this is right-shifted by r positions, i.e. divided by r for use in the next iteration. So, after n iterations, the result obtained is $R = A\times B\times r^{-n} \bmod M$. A modified version of Montgomery algorithm is given in Fig. 1.

```
algorithm Montgomery(A, B, M)
  int R = 0;
  for i= 0 to n-1
    R = R + a_i×B;
    if r_0 = 0 then
      R = R div 2
    else
      R = (R + M) div 2;
  return R;
end.
```

Fig. 1: Montgomery modular algorithm.

In order to yield the right result, we need an extra Montgomery modular multiplication by the constant $2^n \bmod M$. However as the main objective of the use of Montgomery modular multiplication algorithm is to compute exponentiations, it is preferable to Montgomery pre-multiply the operands by 2^{2n} and Montgomery post-multiply the result by 1 to get rid of the 2^{-n} factor. Here we concentrate on the implementation of the Montgomery multiplication algorithm of Fig. 1.

3 Systolic Montgomery Algorithm

A modified version of Montgomery algorithm is that of Fig. 2. The least significant bit of $R + a_i\times B$ is the least significant bit of the sum of the least significant bits of R and B if a_i is 1 and the least significant bit of R otherwise. Furthermore, new values of R are either the old ones summed up with $a_i\times B$ or with $a_i\times B + q_i\times M$ depending on whether q_i is 0 or 1.

```
algorithm ModifiedMontgomery(A, B, M)
  int R ← 0;
  1: for i = 0 to n-1 {
  2:     q_i ← (r_0 + a_i×b_0) mod 2;
  3:     R ← (R + a_i×B + q_i×M) div 2;
  return R;
end.
```

Fig. 2: Modified Montgomery algorithm.

Consider the expression $R + a_i\times B + q_i\times M$ of line 2 in the algorithm of Fig. 2. It can be computed as indicated in the last column of the table of Table 1 depending on the value of the bits a_i and q_i.

a_i	q_i	$R + a_i\times B + q_i\times M$
1	1	$R + MB$
1	0	$R + B$
0	1	$R + M$
0	0	R

Table 1: Computation of R + a_i´B + q_i´M.

A bit-wise version of the algorithm of Fig. 2, which is at the basis of our systolic implementation, is described in Fig. 3. All algorithms, i.e. those of Fig. 1, Fig. 2 and Fig. 3 are equivalent. They yield the same result.

```
algorithm SystolicMontgomery(A,B,M,MB)
  int   R ← 0; bit carry ← 0, x;
  for i = 0 to n
      q_i ← r_0^(i) ⊕ a_i.b_0;
      for j = 0 to n
        switch a_i, q_i
          1,1: x ← mb_i;
          1,0: x ← b_i;
          0,1: x ← m_i;
          0,0: x ← 0;
        r_j^(i+1)  ←  r_(j+1)^(i) ⊕ x_i ⊕ carry;

  carry←r_(j+1)^(i).x_i+r_(j+1)^(i).carry+x_i.carry;
  return R;
end.
```

Fig. 3: Systolic Montgomery algorithm.

In the algorithm above *MB* represents the result of $M + B$, which has at most has $n + 1$ bits.

4 Prototype Systolic Architecture

Assuming the algorithm of Fig. 3 as basis, the main processing element (PE) of the systolic architecture of the Montgomery modular multiplier computes a bit r_j of residue R. This represents the computation of line 8.

The left-border PEs of the systolic arrays perform the same computation but beside that, they have to compute bit q_i as well. This is related to the computation of line 1. The duplication of the PEs in a systolic form implements the iteration of line 0. The systolic architecture of the systolic Montgomery multiplier is shown in Fig. 4.

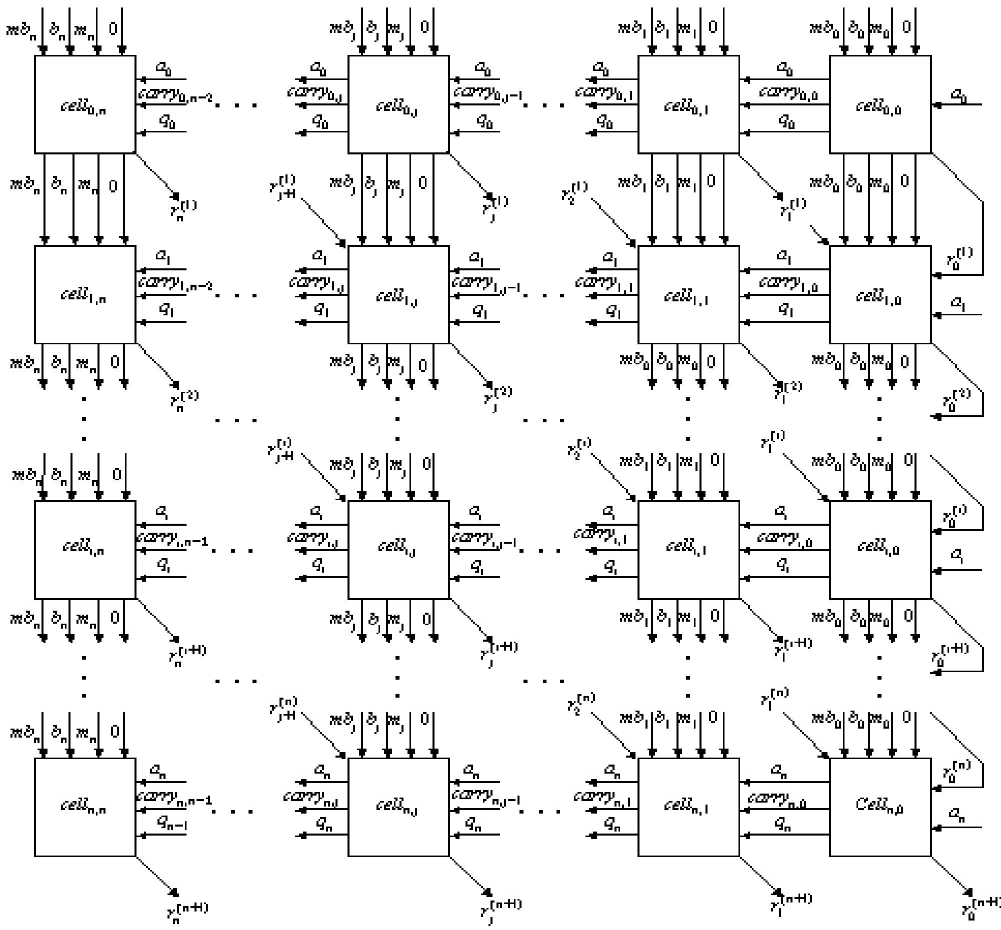

Fig. 4: Systolic architecture of Montgomery multiplier.

The architecture of the basic PE, i.e. $cell_{i,j}$ $1 \le i \le n-1$ and $1 \le i \le n-1$, is shown in Fig. 5. It implements the instructions of lines 2-9 in systolic Montgomery algorithm of Fig. 3.

The architecture of the right-most top-most PE, i.e. $cell_{0,0}$, is given in Fig. 6. Besides the computation of lines 2-9, it implements the computation indicated in line 1. However as $r_0^{(0)}$ is zero, the computation of q_0 is reduced to $a_0 \,.\, b_0$. Besides, the full-adder is not necessary as carry in signal is also 0 so $r_1^{(0)} \oplus x_i \oplus carry$ and $r_1^{(0)} \,.\, x_i + r_1^{(0)} \,.\, carry + x_i \,.\, carry$ are reduced to x_i and 0.

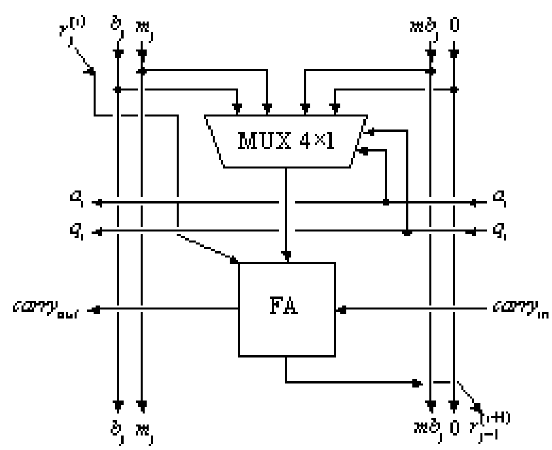

Fig. 5: Basic PE architecture.

The architecture of the rest of the PEs of the first column is shown in Fig. 7. It computes q_0 in the more general case, i.e. when $r_0^{(i)}$ is not null. Moreover, the full-adder is substituted by a half-adder as the carry in signals are zero for these PEs.

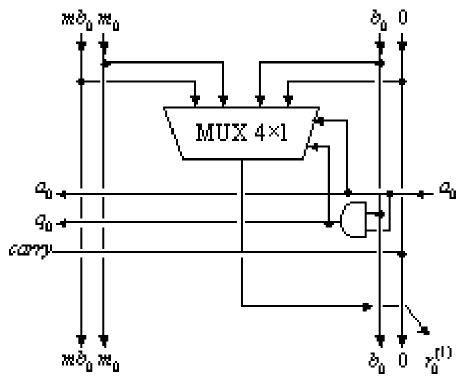

Fig. 6: right-most top-most PE – cell$_{0,0}$.

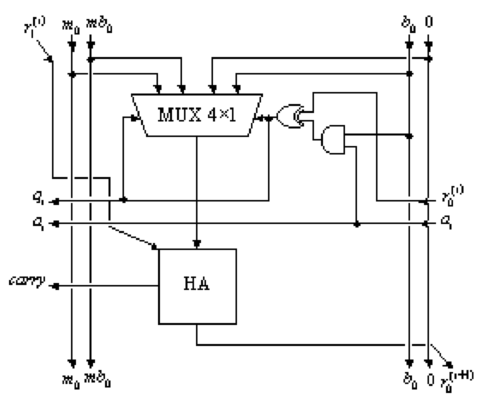

Fig. 7: Right border PEs – cell$_{i,0}$.

The architecture of the architecture of the left border PEs, i.e. $cell_{0,j}$, is given in Fig. 8. As $r_n^{(i)} = 0$, the full-adder is unnecessary and so it is substituted by a half-adder.

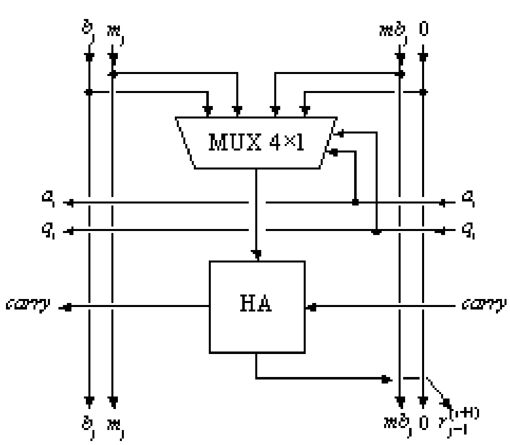

Fig. 8: Left border PEs – cell$_{0,j}$.

The sum M+B is computed only once at the beginning of the multiplication process. This is done by a row of full adder.

5 Time and Area Requirements

The entire design was done using the Xilinx Project Manager (version Build 6.00.09) [7] through the steps of the Xilinx design cycle shown in Fig. 9. The design was elaborated using VHDL [13]. The *synthesis* step generates an optimised netlist that is the mapping of the gate-level design into the Xilinx format: *XNF*. Then, the *simulation* step consists of verifying the functionality of the elaborated design. The *implementation* step consists of partitioning the design into logic blocks, then finding a near optimal placement of each block and finally selecting the interconnect routing for a specific device family. This step generates a logic PE array file from which a bit stream can be obtained. The implementation step provides also the number of configurable logic blocks (CLBs). The *verification* step allows us to verify once again the functionality of the design and determine the response time of the design including all the delays of the physical net and padding. The *programming* step consists of loading the generated bit stream into the physical device.

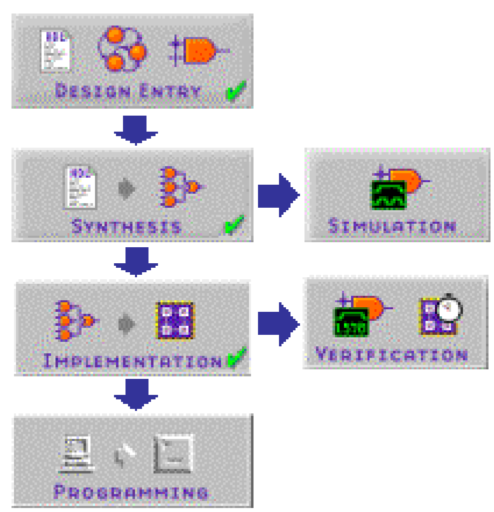

Fig. 9: *Design cycle*.

The output bit $r_j^{(n+1)}$ of the modular multiplication is yield after $2n + 2 + j$ after bits b_j, m_j and mb_j are fed into the systolic array plus an extra clock cycle, which is needed to obtain the bit mb_j. So the first output bit appears after $2n + 3$ clock cycles. Table 1 shows the performance figures obtained by the Xilinx project synthesiser for the iterative multiplier the systolic modular multiplier. The synthesis was done for VIRTEX-E family.

Table 2 includes the clock cycle time required, the area, i.e. the number of CLBs necessary as well as the time/area product delivered by the synthesis and the verification tools of the Xilinx project manager for the systolic hardware of Montgomery modular multiplier.

operand size	***Area*** (CLBs)	***clock cycle time*** (ns)	***area´time*** (CLBs×ns)
128	259	23	5957
256	304	42	12767
512	492	76	37392
768	578	82	47396
1024	639	134	85626

Table 2: Performance figures: iterative vs. systolic Montgomery modular multiplier.

The chart of Fig. 10 compares the area/time product of iterative multiplier implementation we developed in [11] vs. the systolic implementation. It shows that the latter improves the product as well as time requirement while the former improves area at the expense of both time requirement and the product.

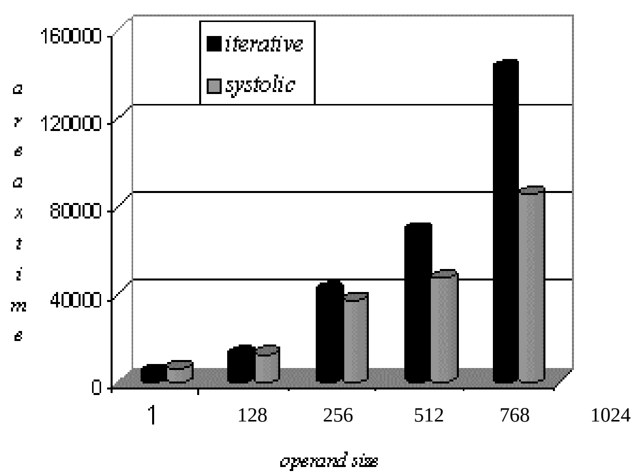

Figure 10: The areaxtime factor for iterative vs. systolic multiplier

So the iterative modular multiplier reduces the required hardware area at the expense of response-time as they have to include a synchronised in-control. The systolic implementation of the modular multiplier attempts to minimise time requirements at the expense of hardware area as we think that one can afford hardware area if one can gain in encryption/decryption time.

6 Conclusion

In this paper, we described a novel systolic architecture to implement Montgomery modular multiplication algorithm.

The modular multiplier was synthesized and implemented using the Xilinx Project Manager [14]. The implementation device used is an FPGA: family SPARTAN and model S05PC84-4.

We compared the space and time requirements of an iterative Montgomery modular multiplier we developed in [11] vs the new systolic modular multiplier and this for different operand sizes. The results were produced by the Xilinx project manager. The results show clearly that despite of requiring much more hardware area, the systolic implementation improves substantially the time requirement and the area/time product when the operand size is bigger than 512 bits, which is almost always the case in RSA encryption/decryption systems.

References:

[1] R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signature and public-key cryptosystems*, Communications of the ACM, **21**:120-126, 1978.

[2] E. F. Brickell, *A survey of hardware implementation of RSA*, Proceedings of CRYPTO'98, Lecture Notes in Computer Science **435**:368-370, Springer-Verlag, 1989.

[3] C. D. Walter, *Systolic modular multiplication*, IEEE Transactions on Computers, **42**(3):376-378, 1993.

[4] S. E. Eldridge and C. D. Walter, *Hardware implementation of Montgomery's Modular Multiplication Algorithm*, IEEE Transactions on Computers, **42**(6):619-624, 1993.

[5] A. Booth, *A signed binary multiplication technique*, Quarterly Journal of Mechanics and Applied Mathematics, pp. 236-240, 1951.

[6] G. W. Bewick, *Fast multiplication algorithms and implementation*, Ph. D. Thesis, Department of Electrical Engineering, Stanford University, United States of America, 1994.

[7] C. D. Walter, *A verification of Brickell's fast modular multiplication algorithm*, International Journal of Computer Mathematics, **33**:153:169, 1990.

[8] J. Rabaey, *Digital integrated circuits: A design perspective*, Prentice-Hall, 1995.

[9] N. Nedjah, L. M. Mourelle, *Yet another implementation of modular multiplication*, Proceedings of 13^{th}. Symposium of Computer Architecture and High Performance Computing, IFIP, Brasilia, Brazil, September 2001.

[10] N. Nedjah, L. M. Mourelle, *Simulation Model for Hardware implementation of modular multiplication*, Proceedings of WSES/IEEE International. Conference on Simulation, Knights Island, Malta, September 2001.

[11] N. Nedjah, L. M. Mourelle, *Hardware Architecture for the Montgomery Modular Multiplication*, WSEAS Transactions on Systems, vol. 1, no. 1, pp. 63-67, January 2002.

[12] P.L. Montgomery, *Modular Multiplication without trial division*, Mathematics of Computation 44, pp. 519-521, 1985.

[13] Z. Navabi, *VHDL - Analysis and Modeling of Digital Systems*, McGraw Hill, Second Edition, 1998.

[14] Xilinx, Inc. *Foundation Series Software*, Http://www.xilinx.com.