Proceedings of the 5th WSEAS Int. Conf. on Instrumentation, Measurement, Circuits and Systems, Hangzhou, China, April 16-18, 2006 (pp240-245)

High Speed Systolic Montgomery Modular Multipliers for RSA Cryptosystems

RAVI KUMAR SATZODA, CHIP-HONG CHANG and CHING-CHUEN JONG
Centre for High Performance Embedded Systems
Nanyang Technological University
3rd Storey, Research Techno Plaza
SINGAPORE

Abstract: - Montgomery modular multiplication is one of the most important and frequently used techniques to accelerate the time consuming mathematical operations used in RSA cryptosystems. In this paper, a modified Montgomery modular multiplication algorithm is presented where the carry-save operations are split into two cycles so as to eliminate the generation of the data-dependent control signal from dominating the critical path. Two novel systolic Montgomery multipliers are designed based on the proposed algorithm. A bit-parallel pipelined architecture followed by a one dimensional variant are implemented on FPGA and evaluated against recently reported Montgomery multipliers implemented on the same platform. Improved results have been demonstrated in terms of area and throughput.

Key-Words:- RSA cryptosystems, modular multiplication, Montgomery multiplication, carry-save architectures, FPGA implementation, systolic architectures

1 Introduction

Public key cryptosystems (PKCs) are increasingly replacing the symmetric key cryptosystems owing to its ease of key management [1]. However, the strength of all public key cryptosystems like RSA, Elliptic Curve Cryptosystem (ECC) etc. depends on the mathematical complexity of the encryption and decryption algorithms [1].

RSA cryptosystem is one of the most prevailing and trusted PKC that is currently being used in many security protocols. Encryption and decryption in RSA algorithm are based primarily on two computationally intensive operations – modular exponentiation and modular multiplication [1]. For example if Alice wants to send a message M to Bob, Alice uses the public key C (listed in a public database) of Bob to encrypt her message to give the ciphertext M' using $M' = M^C \bmod N$. The prime modulus N is also listed in the public database which is related mathematically to the public key C and the private key D of Bob. Decryption is an inverse process of encryption. Bob can retrieve the original message M by using $M = M'^D \bmod N$. Thus, the main operation that is being performed in the both encryption and decryption is modular exponentiation which can be further decomposed to repeated modular squaring and modular multiplication [2].

Modular multiplication is the key operation which limits the functioning of the RSA cryptoengine. Modular multiplication is defined by $P = A\,B \bmod N$. Modular multiplication is in itself a complex mathematical operation involving trial divisions. In the context of cryptosystems like RSA, where security and accuracy is a major concern, the inputs have wordlengths ranging from 128 bits to 1024 bits. Therefore, there is a need to curtail the increased hardware complexity associated with unusually long arithmetic operator to control the power, delay and cost of hardware implementation to within a reasonable bound.

Montgomery modular multiplication (MMM) is one of the most frequently used modular multiplication algorithms in RSA cryptosystems that replaces trial divisions with shifts and adds [3]. MMM algorithm has been studied extensively and several hardware and software implementations of MMM have been reported [2]-[9]. Pseudocodes of different Montgomery multiplication schemes have been comprehensively compared in [4]. Systolic hardware implementations of MMM are reported in [5]-[9]. Architectures in [5] and [6] are implemented using carry-save adders but architectures in [7]-[9] employ carry-propagate adders. Thus MMM implementations in [5] and [6] are faster as compared to architectures in [7]-[9]. Moreover, algorithms in [4], [7] and [8] implement MMM in its original form. However, recently reported modified MMM algorithms [5], [6] and [9] involve precomputation of the sum of one of the multiplicands (A or B) and the modulus (N).

Since modular multiplication is an operation used repetitively to accomplish modular exponentiation in RSA, pipelinable modular multiplier is desirable for cryptographic application and it shall be designed with as high throughput rate as possible. For iterative multiplications, the desideratum is the maximum achievable clock speed rather than the number of clock cycles taken to complete a single multiplication. In this paper, we propose an algorithm – MMM_MX that is based on carry-save representation, without any precomputation for pipelined multiplier design. More number of clock cycles is used to complete the multiplication but the throughput rate has increased due to the shorter critical path. Therefore, the proposed algorithm leads to a faster execution of modular exponentiation for RSA at higher clock frequency without compromising the total latency. Moreover, since there is no precomputation, the buffer overhead, that is required to store the precomputed result, is avoided. A novel pipelined systolic Montgomery modular multiplier is implemented based on the proposed algorithm. The proposed architecture is rapid prototyped using the standard FPGA design flow and is compared against implementation results of other architectures recently reported in [5]-[8] for the critical path delay, throughput and utilization of FPGA resources. A variant of the proposed systolic multiplier is also derived with better area efficiency at the cost of lower throughput.

The rest of the paper is organized as follows. In Section 2, we describe existing MMM algorithms. The problem statement is defined in Section 3 and the issues associated with current systolic architectures for MMM are discussed. A modified MMM_MX algorithm is proposed in Section 4. Section 5 introduces the systolic architecture and its variant derived from the proposed algorithm. The FPGA implementation results are reported in Section 6 followed by the conclusion in Section 7.

2 Preliminaries

In this section we introduce existing MMM algorithms. MMM_CS is MMM implemented in carry-save representation and MMM_OP involves precomputation [5], [6], [9]. We present some notations that are used in the rest of the paper. For consistency, a binary variable name is written in lower case letters. A vector of binary variables is represented with variable name that begins with upper case letter. A binary variable, x at the i-th row and the j-th column in a systolic array is denoted by x_j^i.

The input to output mapping of MMM is given by

$$MMM(A,B,N) = ABR^{-1} \bmod N \qquad (1)$$

where R and N are relatively prime. If N is odd, as is the case in RSA, R can be the even number 2^k , k being the length of the cryptographic key. The original algorithm in [3] involves a subtraction at the end which can be eliminated [5], [6]. Let $A = \sum_{i=0}^{k-1} a_i 2^i$, $B = \sum_{i=0}^{k-1} b_i 2^i$ and $N = \sum_{i=0}^{k-1} n_i 2^i$, where $a_i, b_i, n_i \in \{0,1\}$, then, MMM_CS in carry-save representation without the final subtraction is described by Algorithm 1 [6].

```
Algorithm 1: MMM_CS(A,B,N)
1: $Cin2 \leftarrow 0, Cin1 \leftarrow 0, Sin \leftarrow 0$
2: for $i = 0$ to $k-1$ do
3:   $q \leftarrow (Sin_0 + Cin1_0 + Cin2_0 + a_i b_0) \bmod 2$
4:   $Cin2 + Cin1 + Sin \leftarrow Cin2 + Cin1 + Sin + a_i B + qN$
5:   $Cin2 \leftarrow Cin2/2, Cin1 \leftarrow Cin1/2, Sin \leftarrow Sin/2$
6: end for
7: return $Cin2, Cin1, Sin$
```

```
Algorithm 2: MMM_OP(A,B,N)
1:  $Cin \leftarrow 0, Sin \leftarrow 0$
2:  for $i = 0$ to $k-1$ do
3:    $q \leftarrow (Sin_0 + Cin1_0 + a_i b_0) \bmod 2$
4:    switch $(a_i, q)$
5:      case$(0,0)$ :   $I \leftarrow 0$
6:      case$(0,1)$ :   $I \leftarrow N$
7:      case$(1,0)$ :   $I \leftarrow B$
8:      case$(1,1)$ :   $I \leftarrow B+N$
9:    end switch
10:   $Cin + Sin \leftarrow Cin + Sin + I$
11:   $Cin \leftarrow Cin/2, Sin \leftarrow Sin/2$
12: end for
13: Return $Cin, Sin$
```

In Algorithm 1 – MMM_CS, besides the sum signal – *Sin*, two carry signals – *Cin1* and *Cin2*, are generated to compute the vector additions carry-save representation. Also, an additional stage of adders is required to add *Cin2*, *Cin1* and *Sin* in MMM_CS. A further modification to MMM_CS is proposed in [5], [6] and [9], where the sum – $B+N$ (in `Step 4` of MMM_CS) is precomputed. In this step, a_i and q are used to select either B or N or $B+N$ or 0. So, $B+N$ should be precomputed and stored in a buffer. A 4-to-1 multiplexer is employed to select one of these four possible values. Algorithm 2 – MMM_OP shows the pseudo code that involves this precomputation.

The selective assignment clauses in `Steps 4 to 9` of Algorithm 2 are implemented using a 4-to-1 multiplexer with select signals – a_i and q. The output I from the multiplexer is then added to *sum* and *carry* (from previous iteration).

3 Problem Statement

Let us revisit Algorithms 1 and 2. The main problem in both these algorithms is their dependency on the intermediate signal q. In each iteration, q is computed using the LSB of *sum*, *carry* and input B (`Step 3` in MMM_CS and MMM_OP). In MMM_CS, q is then used to determine *carry* and *sum* signals in `Step 4`. Similarly, in MMM_OP, q is used to select I in `Steps 4` to `10` which is later used for the computation of *carry* and *sum* in `Step 10`. In the systolic array, the basic cell at the least significant position computes q which is propagated to the remaining rows of basic cells. The maximum operating frequency of the architecture is limited by the critical path as a consequence of this dependency on q. MMM_OP also suffers from long critical path due to q. Moreover, it involves buffering of the precomputed result, $B+N$. This step will take up additional clock cycles between successive modular multiplication operations and hence the throughput is affected while performing exponentiation.

In the next section, we propose a modified Montgomery modular multiplication algorithm – MMM_MX, that resolves the dependency on q by introducing an additional clock cycle. Breaking the dependency facilitates hardware reuse. Besides, no precomputation is necessary in the proposed architecture. Therefore, the extra clock cycles needed between successive multiplications in Algorithm 2 have been eliminated. In addition, the number of carry vectors will be reduced from two (in MMM_CS) to one in the proposed algorithm.

4 Proposed Modified MMM

We now derive the modified Montgomery modular multiplication algorithm - MMM_MX. If we consider `Steps 3, 4` and `5` in Algorithm 1, they can be rewritten as:

`Step 3a`: $Ct + St \leftarrow Cin + Sin + a_iB$
`Step 3b`: $q \leftarrow St_0$
`Step 4 `: $C + S \leftarrow Ct + St + qN$
`Step 5 `: $Cin \leftarrow C/2,\ Sin \leftarrow S/2$

In `Step 3a`, the summation of *carry*, *Cin* and *sum*, *Sin* from previous iteration and the input, a_iB, results in the intermediate *carry* and *sum* signals – *Ct* and *St*, respectively. From Algorithm 1, the least significant bit of the sum generated in `Step 3a` is q. With the value of q in hand, the *carry* and *sum* for the current iteration in `Step 4` are calculated and then right shifted in `Step 5`. From the above steps, we notice the following.

(a) If we introduce a register between `Step 3` and `4`, q can be simply obtained from the least significant bit of the intermediate sum, and then used for the calculation of the sum, S and carry, C in the next cycle.

(b) Moreover, `Steps 3a & 4` perform the same operation 'addition', but with different operands. Thus the adder that computes *Ct* and *St* in `Step 3a` can be reused to compute C and S in `Step 4`. Thus a simple 2-to-1 multiplexing of inputs to the adder will enable the same adder to be used in both cycles – `Steps 3a & 4`.

```
Algorithm 3: MMM_MX(A,B,N)
 1: $Cin \leftarrow 0, Sin \leftarrow 0, Ct \leftarrow 0, St \leftarrow 0, ctrl \leftarrow 0, q \leftarrow 0$
 2: for $i = 0$ to $k-1$ do
 3:   for $ctrl = 0$ to $1$ do
 4:     for $j = 0$ to $k-1$ do
 5:       if $ctrl = 0$ then
            $in1 \leftarrow a_k;\ in2 \leftarrow b_j;\ in3 \leftarrow sin_j;\ in4 \leftarrow cin_j$
 6:       else
            $in1 \leftarrow q;\ in2 \leftarrow n_j;\ in3 \leftarrow st_j;\ in4 \leftarrow ct_j$
 7:       end if;
 8:       $carry + sum \leftarrow GFA(in1, in2, in3, in4)$
 9:     end for
10:     $Ct \leftarrow Carry;\ St \leftarrow Sum$
11:     $q \leftarrow St_0$
12:   end for
13:   $Cin \leftarrow Ct/2;$ Sin $\leftarrow St/2$
14: end for
15: Return $Cin, Sin$
```

The above observations result in a different Montgomery modular multiplication algorithm – MMM_MX as shown in Algorithm 3. The proposed algorithm has two **for** loops (with i and j as counters) that indicate the bit-level scanning of inputs which are present in MMM_CS and MMM_OP. MMM_MX has one extra loop (with loop counter *ctrl*). This additional loop implements the control signal that multiplexes correct inputs to the adder that is reused in `Steps 3a` and `4`. In Algorithm 3, the inputs to the multiplexer – *in1*, *in2*, *in3* and *in4*, are selected by *ctrl* in `Steps 5` to `7`. These inputs are then added using a gated full adder

(GFA) in `Step 8` that produces the logical *sum* and *carry* of the expression *in1.in2 + in3 + in4* depending the inputs selected by *cntrl*. When *ctrl* = *'0'*, the intermediate carry and sum, *Ct* and *St*, are computed from the inputs, a_k and *B*, and the sum and carry, *Sin* and *Cin*, from the previous iteration.. At the end of the first clock cycle, we obtain $q = St_0$. In the next clock cycle, when *ctrl = '1'* the full adder is now used to compute *Cin* and *Sin* for the current iteration by using *q*, *N* and the intermediate sum and carry, *St* and *Ct*. The control signal then turns back to 0 and the process continues until all operand bits have been exhausted.

5 Architectures

The systolic architecture corresponding to conventional carry-save implementation described by Algorithm 1 is shown in [6] and those based on the precomputation of *B*+*N*, are implemented in [5], [6] and [9]. This section describes the algorithm to architecture translation of the proposed MMM_MX algorithm. The basic cell or computing element that is used in the systolic array is shown in Fig. 1.

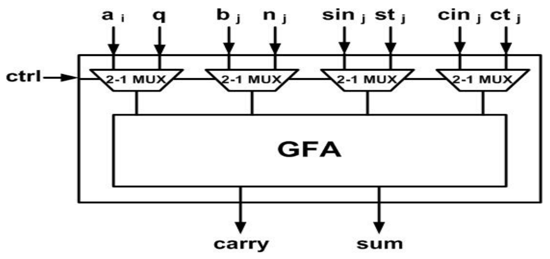


Fig. 1. Basic cell or computation unit

It has four single bit 2-to-1 multiplexers. The control signal selects between the two groups of four input signals (shown in Algorithm 3) to the gated full adder (GFA). The two dimensional systolic architecture 'Design 1' of MMM_MX is shown in Fig. 2. It comprises k^2 replicas of the basic cell in a 2-D array. The outputs of all basic cells are registered using flip-flops denoted by • in Fig. 2. The *sum* output of basic cell from the *i*-th row and the *j*-th column, sum_j^i, is connected to st_j^i and sin_{j-1}^{i+1}. Similarly the carry output from the basic cell, $carry_j^i$ is connected to ct_{j+1}^i and cin_j^{i+1}. Fig. 3(a) illustrates this assignment of *Sum* and *Carry* signals to the GFA at the *i*-th row. The notation used in Fig. 3(a) follows the convention introduced in Section 2. In the *m*-th clock cycle, when *cntrl* = 0, the right shifted *Sum* and *Carry* signals from the (*i*–1)-th row, i.e., sin_{j+1}^{i-1} and cin_j^{i-1} are multiplexed into the GFA (`Step 5` in Algorithm 3). According to Step 6 in Algorithm 3, in the next clock cycle, the inputs of GFA should be assigned to the sum and carry signals that were generated by the *i*-th row of basic cells. Therefore, in the (*m*+1)-th cycle, *Sum* and *Carry* signals, st_j^i and ct_{j+1}^i of basic cells in the same row are selected by the control signal that has just been toggled (`Step 6` in Algorithm 6). Fig. 3(a) illustrates the subscripts of *Sum* and *Carry* in the *m*-th and (*m*+1)-th clock cycle. The *Sum* and *Carry* that are generated by (*i*-1)-th row in (*m* – 1)-th cycle are right shifted to be assigned to the *i*-th row in *m*-th clock cycle. The subscripts of *Sum* and *Carry* resulting from the *i*-th row of basic cells in the (*m*+1)-th cycle retain, indicating that they have not been right shifted.

The reservation table in Fig. 3(b) shows the filling of pipeline. Thus, although the total time taken to generate the first *Sin* and *Cin* outputs is 2*k* clock cycles, the throughput for subsequent results after the first product has been output is merely two clock cycles.

A variant of the 2-D systolic architecture – 'Design 2' is shown in Fig. 4. This implementation involves only one row of basic cells. Input a_i is sent serially in alternate clock cycles into this row of basic cells. Inputs *B* and *N* are applied in a parallel fashion. Since each a_i takes 2 clock cycles, Design 2 would require 2*k* clock cycles to compute one Montgomery multiplication result. The chip area however, is reduced substantially at the expense of lower throughput.

6 Results

MMM_MX 'Design 1' and 'Design 2' are evaluated against recently reported Montgomery modular multipliers in [5-8] for maximum clocking frequency, throughput and area (in terms of logic slices of targeted FPGA). For applications that would benefit from a pipelined architecture, throughput is a more important measurement for performance than the latency required for a single multiplication. A recent paper [5] by Fournaris et al. compares some the latest reported Montgomery modular multiplier architectures with their proposed one-dimensional precomputation based architecture. We compare the implementation of our architectures against the results reported by Fournaris et al. in [5]. The comparisons are based on FPGA implementation. The proposed designs are implemented on Xilinx Virtex 2 chip and synthesized using Xilinx Synthesis Tool (XST). Design 1 and Design 2 are compared against different kinds of architectures, with and without precomputation of *B*+*N*, carry-save/normal representations and FPGA specific implementations, reported in [5]-[8]. The MMM architectures in [5], [7] and [8] are one-dimensional arrays whereas those in [6] are two-dimensional systolic architectures. [5] and [6] employ carry-save representation but [7] and [8] are based on carry-propagate adders. Unlike the 'Conventional' architecture in [6], the 'Optimized' architecture in [6] is a two-dimensional systolic array that is based on the precomputation of *B*+*N*. The more recent architecture in [5] is a one-dimensional variant of 'Optimized' architecture in [6].

In Table I, the FPGA implementation results of the proposed 1-D architecture – Design 2, are compared against the 1-D architectures in [5], [7] and [8]. We see from Table I that the proposed Design 2 is faster than the recently reported 'Optimized' [5] by 3 times. The throughput of the proposed architecture is 1.5 times that of 'Optimized' in [5]. Thus, even though the proposed architecture takes one extra clock cycle in every iteration to compute the Montgomery product, due to its shorter critical path, it is still faster than all the

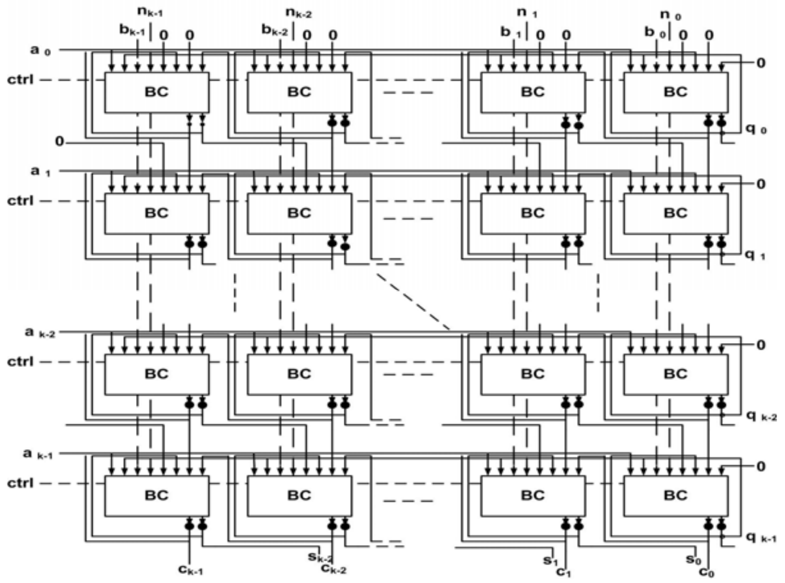
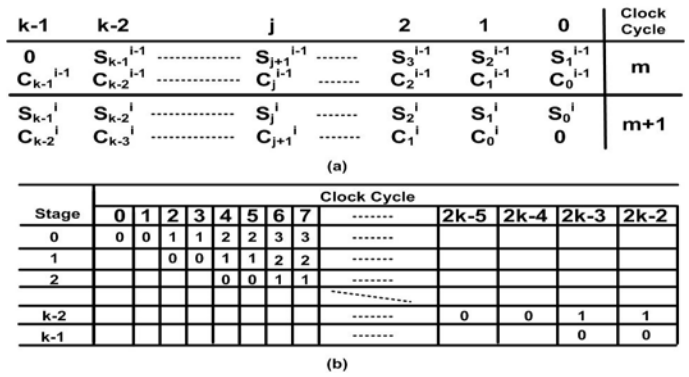


Fig. 2. Two-dimensional systolic architecture of Design 1



k-1	k-2		j		2	1	0	Clock Cycle
0 C_{k-1}^{i-1}	S_{k-1}^{i-1} C_{k-2}^{i-1}	------------- -------------	S_{j+1}^{i-1} C_j^{i-1}	------- -------	S_3^{i-1} C_2^{i-1}	S_2^{i-1} C_1^{i-1}	S_1^{i-1} C_0^{i-1}	m
S_{k-1}^i C_{k-2}^i	S_{k-2}^i C_{k-3}^i	------------- -------------	S_j^i C_{j+1}^i	------- -------	S_2^i C_1^i	S_1^i C_0^i	S_0^i 0	m+1

(a)

Stage	Clock Cycle												
	0	1	2	3	4	5	6	7	-------	2k-5	2k-4	2k-3	2k-2
0	0	0	1	1	2	2	3	3	-------				
1			0	0	1	1	2	2	-------				
2					0	0	1	1	-------				
									-------				
k-2									-------	0	0	1	1
k-1									-------			0	0

(b)

Fig. 3. (a) Indices of Sum *S* and Carry *C* of *i*-th row of basic cells in the *m*-th and (*m*+1)-th clock cycle (b) Reservation table of the pipeline of the 2-D architecture

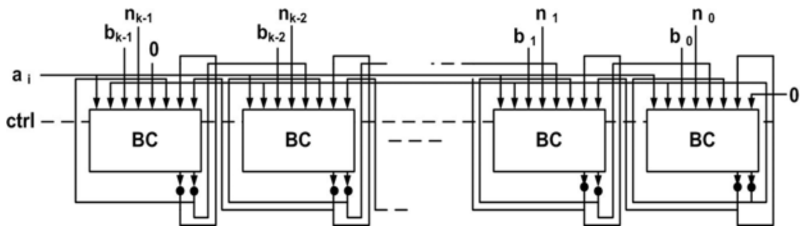


Fig. 4. One-dimensional systolic architecture of Design 2

one-dimensional architectures shown in Table I. As a result of this, the total latency is reduced and the reduction in the total latency is prominent when it is used for modular exponentiation, where the operands are multiplied repetitively. Moreover, the number of slices is also the least in comparison with all the listed architectures.

The two-dimensional systolic architectures were implemented for a bit-length of 128. FPGA implementation results of the proposed 2-D architecture – Design 1, are compared against the two-dimensional 'Optimized' and 'Conventional' architectures (from [6]) in Table II. In 2-D architectures, the proposed multiplier also outperforms its counterparts in both aspects.

TABLE I. FPGA IMPLEMENTATION RESULTS OF 1024-BIT ONE-DIMENSIONAL MONTGOMERY MODULAR MULTIPLIERS

Architecture	*Chip Area (slices)*	*Clock Frequency (MHz)*	*Throughput (bit/sec)*
Design 2	2947	386.84	193.42 M
Optimized [5]	3611	129.10	129.00 M
Daly [7]	5458	54.61	54.40 M
Ors [8]	5706	95.62	31.83 M

TABLE II. FPGA IMPLEMENTATION RESULTS OF 128-BIT ONE-DIMENSIONAL MONTGOMERY MODULAR MULTIPLIERS

Architecture	*Chip Area (slices)*	*Clock Frequency (MHz)*
Design 1	44330	358.17
Optimized [6]	48767	168.70
Conventional [6]	65473	156.30

7 Conclusion

The most important operation in RSA encryption and decryption is the hardware-intensive modular multiplication. This operation can be accomplished by Montgomery modular multiplication. An improved Montgomery multiplication – MMM_MX has been proposed in this paper which reduces the critical path by removing dependency on an intermediate signal. Since MMM_MX is free from any precomputations, modular exponentiation can be efficiently carried out with this operator by feeding back the result in line with the flow of the input data continuously. Two systolic carry-save architectures – one-dimensional bit-serial and two-dimensional bit-parallel architectures, are discussed and implemented. The bit-serial architecture trades throughput for area. The two architectures were evaluated against different kinds of recently reported Montgomery modular implementations. The proposed algorithm reduces the minimum clock period by three times when compared with one of the fastest algorithms.

References:

[1] B. Schneier, *Applied Cryptography*. 2nd Edition, Wiley 1996.

[2] Y. Ching-Chao, C. Tian-Sheuan and J. Chein-Wei, "A new RSA cryptosystem hardware design based on Montgomery's Algorithm," *IEEE Trans. on Circuits and Systems - II: Analog and Digital Singal Processing*, vol. 45, no. 7, pp. 908 -913, July 1998.

[3] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Math. Comput.*, vol. 44, pp. 519-521, Apr. 1985.

[4] C. K. Koc, T. Acar and B. S. Kaliski, Jr., "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, pp. 26-33, June 1996.

[5] A. P. Fournaris and O. Koufopavlou, "A new RSA encryption architecture and harware implementation based on optimized Montgomery multiplication," in *Proc. of IEEE Symp. on Circuits and Systems (ISCAS 2005)*, pp. 4645-4648, May 2005.

[6] A. P. Fournaris and O. Koufopavlou, "Montgomery modular multiplier architecture and hardware implementations for an RSA cryptosystem," in *Proc. of 46th IEEE Intl. Midwest Symp. on Circuits and Systems (MWSCAS 2003)*, pp. 27-30, Dec. 2003.

[7] A. Daly and W. Marnane, "Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," in *Proc. of the 2002 ACM/SIGDA 10th Intl. Symp. on Field-programmable Gate Arrays*, pp. 40-49, 2002.

[8] S. B. Ors, L. Batina, B. Preneel and J. Vandewalle, "Hardware implementation of a Montgomery modular multiplier in a systolic array," in *Proc. of Intl. Parallel and Distributed Processing Symp. (IPDPS'03)*, 8 pp., Apr. 2003.

[9] N. Nedjah and L. de Macedo Mourelle, "Two hardware implementations for the Montgomery modular multiplication: sequential versus parallel," in *Proc. of the 15th Symp. on Integrated Circuits and Systems Design (SBC CI'02)*, pp. 3-8, Sept. 2002.