

# An RSA Encryption Hardware Algorithm using a Single DSP Block and a Single Block RAM on the FPGA

Bo Song, Kensuke Kawakami, Koji Nakano, and Yasuaki Ito

Department of Information Engineering

School of Engineering, Hiroshima University

1-4-1 Kagamiyama, Higashi-Hiroshima, 739-8527, JAPAN

**Abstract**—The main contribution of this paper is to present an efficient hardware algorithm for RSA encryption/decryption based on Montgomery multiplication. Modern FPGAs have a number of embedded DSP blocks (DSP48E1) and embedded memory blocks (BRAM). Our hardware algorithm supporting 2048-bit RSA encryption/decryption is designed to be implemented using one DSP48E1, one BRAM and few logic blocks (slices) in the Xilinx Virtex-6 family FPGA. The implementation results showed that our RSA module for 2048-bit RSA encryption/decryption runs in 277.26ms. Quite surprisingly, the multiplier in DSP48E1 used to compute Montgomery multiplication works in more than 97% clock cycles over all clock cycles. Hence, our implementation is close to optimal in the sense that it has only less than 3% overhead in multiplication and no further improvement is possible as long as Montgomery multiplication based algorithm is used. Also, since our circuit uses only one DSP48E1 block and one Block RAM, we can implement a number of RSA modules in an FPGA that can work in parallel to attain high throughput RSA encryption/decryption.

**Keywords:** Modular exponentiation, Montgomery multiplication, FPGA, RSA, DSP

## I. INTRODUCTION

RSA [1] is one of the most widely used public key cryptography, which can be done by computing modulo exponentiation such as  $P = C^D \bmod M$ . The security of the RSA cryptosystem is based on the problem of factoring large numbers problem. An RSA operation is a modular exponentiation, which requires repeated modular multiplications. For security reasons, greater than 1024-bit length of keys are suggested recently, which leads to a huge time consumption. Therefore, Montgomery Modular Multiplication algorithm [2] is proposed as the most efficient modular multiplication algorithm available. Most of literatures have reported to implement RSA by Montgomery Multiplication such as [3]–[5]. With Montgomery Multiplication algorithm, trial division can be replaced by the modulus with a series of additions and shift operations.

An FPGA is a programmable logic device designed to be configured by the customer or designer by hardware describe language after manufacturing. Since FPGA chip maintains relative lower price and programmable features, it is widely used in those fields which need to update architecture or functions frequently such as communication and education. The most common FPGA architecture consists of an array

of logic blocks, I/O pads, Block RAMs and routing channels. A recent trend has been to take the coarse-grained architectural approach a step further by combining the logic blocks and interconnects of traditional FPGAs with embedded microprocessors which broaden a growing range of other areas. Furthermore, embedded DSP blocks have integrated into an FPGA that makes a higher performance and a broader application.

The main contribution of this paper is to present an efficient hardware algorithm of modular exponentiation, maximized making use of the DSP blocks in our target FPGA, Xilinx Virtex-6 family. Our hardware algorithm requires only one DSP block, as well as one Block RAM with a small quantity of logic blocks. A multiplier in the DSP block works in more than 90% over all the clock cycles. From 64-bit, up to 2048-bit RSA encryption/decryption can be applied in the same architecture without any modification.

Our modular exponentiation algorithm implemented in Xilinx Virtex-6 family FPGA XC6VLX240T-1 uses only one DSP48E1 Block, one Block RAM, and few logic blocks (slices). The implementation results showed that our RSA module for 2048-bit RSA encryption/decryption runs in 447.027MHz using 123940864 clock cycles, that is, in 277.26ms. Quite surprisingly, Montgomery multiplication based RSA encryption/decryption needs 120434688 times 17-bit multiplication, and thus, a multiplier in DSP48E1 is used in more than 97% clock cycles over all clock cycles. Hence our implementation is close to optimal in the sense that it has only less than 3% overhead and no further improvement is possible as long as Montgomery multiplication based algorithm is used. For the comparison purpose, our circuit also implemented in obsolete generation Xilinx Virtex-5 and Virtex-4 FPGA. Also, since our circuit uses only one DSP48E1 block and one Block RAM, we can implement a number of RSA modules in an FPGA that work in parallel to attain high throughput RSA encryption/decryption. Actually, we have implemented 128 RSA encryption/decryption circuits to improve the throughput greatly.

The remaining contents of this paper are organized as follows. Section II introduces modular exponentiation and Montgomery modular multiplication algorithm and its relative researches. Section III describes our proposed hardware algo-

rithm and its architecture. Section IV gives the experimental result, its analysis and comparisons with relative literatures.

## II. MODULAR EXPONENTIATION

In the RSA encryption/decryption, the modular exponentiation  $C = P^E \bmod M$  or  $P = C^D \bmod M$  are computed, where  $P$  and  $C$  are plain and cypher text, and  $(E, M)$  and  $(D, M)$  are encryption and decryption keys. Usually, the bit length in  $P$ ,  $E$ ,  $D$ , and  $M$  is 512 or larger. Also, the modulo exponentiation is repeatedly computed for fixed  $E$ ,  $D$ , and  $M$ , and various  $P$  and  $C$ . Since modulo operation is very costly in terms of the computing time and hardware resources, *Montgomery modular multiplication* [2] is used, which does not directly compute modulo operation.

### A. Montgomery Modular Multiplication

Montgomery multiplication [2], introduced in 1985 by Peter Montgomery, is an optimal method to calculate modular exponentiation. Three  $R$ -bit numbers  $X$ ,  $Y$ , and  $M$  are given, and  $(X \cdot Y + q \cdot M) \cdot 2^{-R} \bmod M$  is computed, where an integer  $q$  is selected such that the least significant  $R$  bits of  $X \cdot Y + q \cdot M$  are zero. The value of  $q$  can be computed as follows. Let  $(-M^{-1})$  denote the minimum non-negative number such that  $(-M^{-1}) \cdot M \equiv -1 \pmod{2^R}$ . Since  $M$  is odd, then  $(-M^{-1}) < 2^R$  always holds. We can select  $q$  such that  $q = ((X \cdot Y) \cdot (-M^{-1})) \llbracket r-1, 0 \rrbracket$ . For such  $q$ ,  $(X \cdot Y + q \cdot M) \llbracket r-1, 0 \rrbracket$  are zero. For the reader's benefit, we will confirm this fact using an example. Suppose  $X = 10010011(147)$ ,  $Y = 01011100(92)$ ,  $M = 11111011(251)$ , and  $R = 8$ . We have the product  $X \cdot Y = 011010011010100(13524)$ . Next, we need to select an integer  $q$  such that the least significant  $R$  bits of  $X \cdot Y + q \cdot M$  are zero. In this case,  $(-M^{-1}) = 11001101(205)$ , because  $(-M^{-1}) \cdot M \equiv 1100100011111111(51455) \equiv -1 \pmod{2^8}$ . Thus  $q = (X \cdot Y) \llbracket R-1, 0 \rrbracket \cdot (-M^{-1}) = 11000100(196)$  is selected. Then the product  $q \cdot M = 1100000000101100(49196)$  and the sum  $X \cdot Y + q \cdot M = 1111010100000000(62720)$  could be obtained. Now, we have  $(X \cdot Y + q \cdot M) \llbracket R-1, 0 \rrbracket = 00000000$  and  $(X \cdot Y + q \cdot M) \cdot 2^{-R} = (X \cdot Y + q \cdot M) \llbracket 2R-1, R \rrbracket = 11110101(245)$ . Since  $0 \leq X, Y < M < 2^R$  and  $0 \leq q < 2^R$ , it is guaranteed that  $(X \cdot Y + q \cdot M) \cdot 2^{-R} < 2M$ . Therefore, by subtracting  $M$  from  $(X \cdot Y + q \cdot M) \cdot 2^{-R}$ , we can obtain  $(X \cdot Y + q \cdot M) \cdot 2^{-R} \bmod M$  if it is not less than  $M$ .

Radix- $2^r$  Montgomery multiplication is shown in Algorithm 1. In Algorithm 1,  $d = \lceil R/r \rceil$  presents the number of digits in radix- $2^r$  operands. The multiplier  $Y$  is partitioned by each  $r$ -bit and  $Y_i$  represents the  $i$ -th digit of  $Y$ . Therefore,  $Y$  could be given by  $Y = \sum_{i=0}^{d-1} 2^{ir} \cdot Y_i$ . After  $d$  loops,  $R$ -bit Montgomery multiplication can be computed. As far as now, Montgomery multiplication could be computed by multiplication, addition and shift operations without modulo operations. The later is time cost and resource cost.

**- Algorithm 1: radix- $2^r$  Montgomery Multiplication -**  
radix- $2^r$ ,  $d = \lceil R/r \rceil$ ,  $X, Y, M \in \{0, 1, \dots, 2^R - 1\}$ ,  
 $Y = \sum_{i=0}^{d-1} 2^{ir} \cdot Y_i$ ,  $Y_i \in \{0, 1, \dots, 2^r - 1\}$

$(-M^{-1}) \cdot M \equiv -1 \pmod{2^r}$ ,  $-M^{-1} \in \{0, 1, \dots, 2^r - 1\}$

Input:  $X, Y, M, -M^{-1}$

Output:  $S_d = X \cdot Y \cdot 2^{-dr} \bmod M$

1.  $S_0 \leftarrow 0$
2. **for**  $i = 0$  **to**  $d - 1$  **do**
3.      $q_i \leftarrow ((S_i + X \cdot Y_i) \cdot (-M^{-1})) \bmod 2^r$
4.      $S_{i+1} \leftarrow (X \cdot Y_i + q_i \cdot M + S_i) \llcorner 2^r$
5. **end for**
6. **if**  $(M \leq S_d)$  **then**  $S_d \leftarrow S_d - M$

Since  $X \cdot Y + q \cdot M \equiv X \cdot Y \pmod{M}$ , we write  $(X \cdot Y + q \cdot M) \cdot 2^{-R} \bmod M = X \cdot Y \cdot 2^{-R} \bmod M$ . Let us see how Montgomery modular multiplication is used to compute  $C = P^E \bmod M$ . Suppose we need to compute  $C = P^E \bmod M$ . For simplicity, we assume that  $E$  is a power of two. Since  $R$  and  $M$  are fixed, we can assume that  $2^{2R} \bmod M$  is computed beforehand. We first compute  $P \cdot (2^{2R} \bmod M) \cdot 2^R \bmod M = P \cdot 2^R \bmod M$  using the Montgomery modular multiplication. We then compute the square  $(P \cdot 2^R \bmod M) \cdot (P \cdot 2^R \bmod M) \cdot 2^{-R} \bmod M = P^2 \cdot 2^R \bmod M$ . It should be clear that, by repeating the square computation using the Montgomery modular multiplication, we have  $P^E \cdot 2^R \bmod M$ . After that, we multiply 1, that is  $(P^E \cdot 2^R \bmod M) \cdot 1 \cdot 2^{-R} \bmod M = P^E \bmod M$  is computed. In this way, cypher text  $C$  could be obtained.

Algorithm 2 shows the modular exponentiation using Montgomery multiplication of Algorithm 1. In Algorithm 2,  $E_b$  represents the size of  $E$ . Inputs  $2^{2dr} \bmod M$  and  $-M^{-1}$  are given. To use Montgomery modular multiplication,  $C$  and  $P$  are converted from 1 and  $P$  in the 1st line and the 2nd line, respectively. The portion underlined in Algorithm 2 can be computed using Montgomery multiplication of Algorithm 1.

### - Algorithm 2: Modular Exponentiation -

$0 \leq E < 2^{E_b} - 1$ ,  $E = \sum_{i=0}^{E_b-1} 2^i \cdot E_i$ ,  $E_i \in \{0, 1\}$

Input:  $P, E, M, -M^{-1}, 2^{2dr} \bmod M$

Output:  $C = P^E \bmod M$

1.  $C \leftarrow (2^{2dr} \bmod M) \cdot 1 \cdot 2^{-dr} \bmod M$ ;
2.  $P \leftarrow \frac{(2^{2dr} \bmod M) \cdot P \cdot 2^{-dr} \bmod M}{2^R}$ ;
3. **for**  $i = E_b - 1$  **downto**  $0$  **do**
4.      $C \leftarrow \underline{C \cdot C \cdot 2^{-dr} \bmod M}$ ;
5.     **if**  $E_i = 1$  **then**  $C \leftarrow \underline{C \cdot P \cdot 2^{-dr} \bmod M}$ ;
6. **end for**
7.  $C \leftarrow \underline{C \cdot 1 \cdot 2^{-dr} \bmod M}$ ;

### B. Related Researches

There are several researches reported to implement modular exponentiation by Montgomery multiplication algorithm. In [6], the number of multiplications and additions, the times of memory access, and the size of memory necessary to compute Montgomery modular multiplication are evaluated by software implementation. McIvor *et al.* implemented and evaluated three algorithms shown in [6] on FPGAs [7]. Blum and Paar proposed a modular exponentiation hardware algorithm with a radix-2 Montgomery multiplication using systolic array [3]. Also, a radix- $2^4$  modular exponentiation circuit that is an extended method of the radix-2 circuit is proposed [4]. The

circuits of the above are fixed for the size of operands. However, the following methods that are independent of the size of operands were proposed. Tenca *et al.* presented a radix-2 scalable Montgomery multiplication architecture [5]. This architecture uses fixed processing elements to deal with variable bit length of operands. Nakano *et al.* presented a radix-2<sup>16</sup> Montgomery multiplier and RSA encryption hardware algorithm using embedded Block RAMs of an FPGA efficiently [8]. In the algorithm, they use a method to prevent a long carry delay in huge integer addition with redundant number system. Mazzeo *et al.* proposed a small RSA encryption circuit [9]. They compute Montgomery multiplication in Digit-Serial way using Radix-2. Suzuki proposed a high speed modular exponentiation circuit featuring a Xilinx FPGA which contains DSP blocks with radix-2<sup>17</sup> [10]. Several DSP blocks are used to achieve a high operation frequency. Alho *et al.* implemented the modular exponentiation using Altera FPGA with a single DSP block in radix-2<sup>18</sup> [11].

Above literatures introduce methods to implement modular exponentiation in FPGA using Montgomery multiplication featuring radix, device and scalability. In this work, we propose an efficient method to implement modular exponentiation using Xilinx FPGA in radix-2<sup>17</sup>. The radix-2<sup>17</sup> is decided by the feature of embedded DSP blocks in our target device.

### III. OUR MODULAR EXPONENTIATION ALGORITHM

In our hardware algorithm, we use an embedded DSP block and a Block RAM in Xilinx FPGA. This section mainly shows a Montgomery modular multiplication circuit and a modular exponentiation circuit with it.

#### A. FPGA architecture

Our proposed algorithm is implemented in a Xilinx Virtex-6 family FPGA which is a low-power-cost and high speed device [12]. In this section, features of Virtex-6 are briefly described necessary to explain our hardware algorithm. However, our algorithm can be implemented to other families of Xilinx FPGA; Xilinx Virtex-5 [13] and Virtex-4 [14]. The implementation results will be discussed in Section IV.

The schematic diagram of Virtex-6 FPGA is shown as Figure 1. An FPGA chip is composed by CLBs (Configurable Logic Blocks), which are the basic programmable logic blocks, configurable inner connections and input/output blocks (I/O Blocks). To compensate for processing speed insufficiency of CLBs, Virtex-6 FPGAs have a DSP48E1 block that is a DSP block with a multiplier and an adder, which can perform multiply-accumulate operation in high clock frequency. Also, Virtex-6 FPGAs have a Block RAM to compensate for memory insufficiency of CLBs. In our proposed algorithm, these blocks are used efficiently.

The CLB in Virtex-6 consists of 2 sub-logic blocks called Slice. With the components LUT (Look Up Table) and Flip-Flop in the slice, various combinatorial circuits and sequential circuits can be implemented.

The DSP48E1 block has a two-input multiplier followed by multiplexers and a three-input adder/subtractor/accumulator.

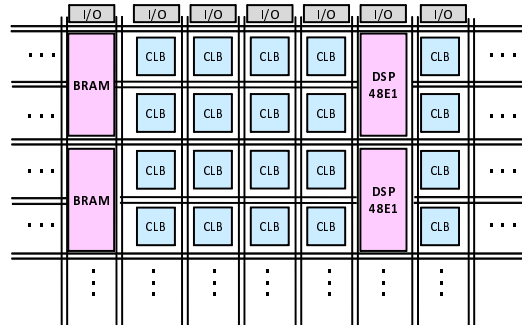


Fig. 1. Internal Configuration of Virtex-6 FPGA

The DSP48E1 multiplier has an 18-bit and a 25-bit two's complement operands and produces one 48-bit two's complement operand. Programmable pipelining of input operands, intermediate products, and accumulator outputs enhances throughput and improves the frequency. Our algorithm utilizes a DSP48E1 block using multiply accumulate (MACC) of 17-bit operands. Among the operators of the DSP48E1, since the pipeline registers are used, its latency has been increased. This latency is absorbed by always performing the multiplier in our algorithm.

The Block RAM is a synchronized write and read embedded memory. In Virtex-6 FPGA, it can be configured as a 36k-bit dual-port Block RAMs, FIFOs, or two 18k-bit dual-port RAMs. In our architecture, it is used as a 1k×18-bit dual-port RAM.

#### B. Our Montgomery Modular Multiplication Algorithm

1) *Montgomery Modular Multiplication Algorithm:* Algorithm 3 shows our proposed algorithm of Montgomery multiplication. Let  $\{A : B\}$  denote a concatenation of  $A$  and  $B$ . For example, if  $A = (FF)_{16}$  and  $B = (EC)_{16}$ ,  $\{A : B\} = (FFEC)_{16}$ . Algorithm 3 is an improved algorithm from Algorithm 1 introduced in Section II-A. Considering the features of our target Virtex 6 FPGA, radix-2<sup>17</sup> is selected. Let  $R$  denote the size of Montgomery multiplier operands  $X$ ,  $Y$ , and  $M$ . Also,  $d = \lceil R/17 \rceil$  is the number of digits of the operands on radix-2<sup>17</sup>. In the algorithm, we introduce the condition  $17d \geq R + 3$  to ignore the subtraction shown in the 6th line of Algorithm 1. If the condition is satisfied, we can guarantee that at least 3-bit 0 is padded to the most significant bits of the most significant digit as the redundancy. Due to the stringent page limitation, the proof is omitted. However, we can say that  $M \geq C$  is always satisfied in the modular exponentiation shown in Algorithm 2. Further, in practical RSA encryption, the size of operands is radix-2 numbers such as 512-bit, 1024-bit, 2048-bit, and 4096-bit. For radix-2<sup>17</sup> system, the condition  $17d \geq R + 3$  is satisfied. If the condition is not satisfied, we just need to append one redundant digit at the most significant digit.

Algorithm 3 is a radix-2<sup>17</sup> digit serial Montgomery algorithm from Algorithm 1. In other words, each 17-bit, as 1 digit, is processed every clock cycle. For this reason, the operands

$X$ ,  $Y$ ,  $M$ , and  $S_i$  are split into 17-bit digits  $X_j$ ,  $Y_j$ ,  $M_j$ , and  $S_{(i,j)}$ , respectively. The loop from the 2nd to 11th lines of Algorithm 3 corresponds to the 2nd to 5th lines of Algorithm 1. Comparing the two algorithms,  $S_{i+1} \leftarrow (X \cdot Y_i + q_i \cdot M + S_i) / 2^r$  of the 4th line of Algorithm 1 corresponds to the digit serial processing by 4th to 10th lines of Algorithm 3. In Algorithm 3,  $C_\alpha$ ,  $C_\beta$ ,  $C_\gamma$ , and  $C_S$  are carries and they are added at the next loop. In the algorithm,  $C_\alpha, C_\beta$  are 17-bit carries for 17-bit MACC, and  $C_\gamma, C_S$  are 1-bit carries for 17-bit addition. For example, at the 6th line a product of  $X_j$  and  $Y_i$ , and an addition of the product and  $C_\alpha$  are computed. The resulting upper 17-bit denotes a carry  $C_\alpha$  which can be added at next loop. The lower 17-bit of result is  $\alpha$  which is used at the 8th and 9th lines. These carries in our algorithm appear in both the 17-bit MACC and the 17-bit adder to prevent a long carry chain that causes circuit delay.

**- Algorithm 3: Our Montgomery Algorithm -**

radix- $2^{17}$ ,  $d = \lceil R/17 \rceil$ ,  $17d \geq R + 3$ ,  
 $X, Y, M, S_i \in \{0, 1, \dots, 2^R - 1\}$ ,  
 $-M^{-1}, \alpha, \beta, \gamma, C_\alpha, C_\beta \in \{0, 1, \dots, 2^{17} - 1\}$ ,  $C_\gamma, C_S \in \{0, 1\}$ ,  
 $X = \sum_{i=0}^{d-1} 2^{17i} \cdot X_i, X_i \in \{0, 1, \dots, 2^{17} - 1\}, X_d = 0$   
 $Y = \sum_{i=0}^{d-1} 2^{17i} \cdot Y_i, Y_i \in \{0, 1, \dots, 2^{17} - 1\}$   
 $M = \sum_{i=0}^{d-1} 2^{17i} \cdot M_i, M_i \in \{0, 1, \dots, 2^{17} - 1\}, M_d = 0$   
 $S_i = \sum_{j=0}^{d-1} 2^{17j} \cdot S_{(i,j)}, S_{(i,j)} \in \{0, 1, \dots, 2^{17} - 1\}, S_d = 0$   
Input:  $X, Y, M, -M^{-1}$   
Output:  $S_d = X \cdot Y \cdot 2^{-17d} \bmod M$

1.  $S_0 \leftarrow 0$
2. **for**  $i = 0$  **to**  $d - 1$  **do**
3.  $q \leftarrow ((X_0 \cdot Y_i + S_{(i,0)}) \cdot (-M^{-1})) \bmod 2^{17}$
4.  $C_\alpha, C_\beta, C_\gamma, C_S \leftarrow 0$
5. **for**  $j = 0$  **to**  $d$  **do**
6.  $\{C_\alpha : \alpha\} \leftarrow X_j \cdot Y_i + C_\alpha$
7.  $\{C_\beta : \beta\} \leftarrow q \cdot M_j + C_\beta$
8.  $\{C_\gamma : \gamma\} \leftarrow \alpha + \beta + C_\gamma$
9.  $\{C_S : S_{(i+1,j-1)}\} \leftarrow \gamma + S_{(i,j)} + C_S$
10. **end for**
11. **end for**

2) *Architecture of Montgomery Multiplier*: Figure 2 shows the architecture of Montgomery multiplier using Algorithm 3. The inputs of Montgomery multiplier are supplied from a Block RAM and registers of modular exponentiation circuit. Given the inputs, the operations of Algorithm 3 are executed by the MACC composed with one DSP48E1 and one adder composed with CLBs. The data flow of these operations is shown in Table I.

The computations of the 3rd, 6th and 7th lines are executed with the DSP48E1. In order to obtain  $q$  in the 3rd line,  $X_0 \cdot Y_0 + S_{(i,0)}$  is obtained first. After that,  $(X_0 \cdot Y_i + S_{(i,0)}) \cdot (-M^{-1})$  is computed. According to Table I, 6 clock cycles are necessary to compute  $q$ . In the 6th line, 17-bit multiplication  $X_j \cdot Y_i$  is computed and the carry  $C_\alpha$  for the digit is added at the same time. The production and the addition are computed using the DSP48E1. After that, the lower 17-bit of the result will be

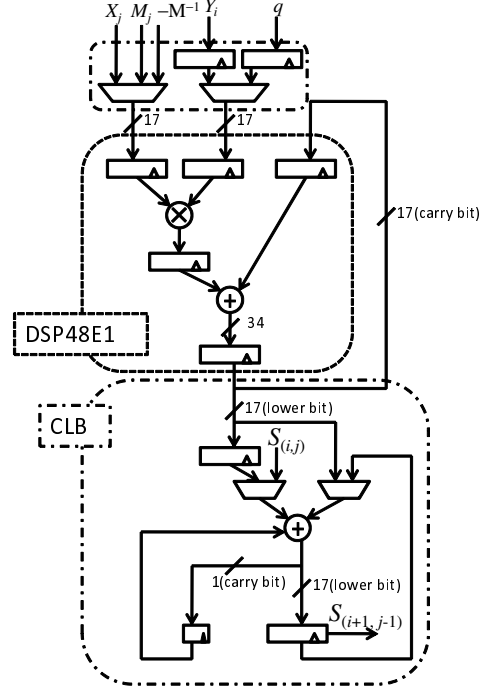


Fig. 2. Structure of our Montgomery multiplier

added in the following adder composed by CLB. On the other hand, the upper 17-bit of the result is stored as a carry into the pipeline register and added at the next clock. The 7th line  $q \cdot M_j + C_\beta$  is computed as the same as the 6th line using DSP48E1. As shown in Table I, the sums of products of the 6th and 7th lines in Algorithm 3 are computed by alternate input of  $X_j, Y_i$  and  $M_j, q$ . Since the carries are stored to the pipeline register in the DSP48E1, our circuit is able to be performed efficiently.

The adder, that is composed by CLBs, following the DSP48E1 computes  $\alpha + \beta + C_\gamma$ ,  $\gamma + S_{(i,j)} + C_S$  of the 8th and 9th lines in the Algorithm 3. Since  $C_\gamma$  and  $C_S$  are 1-bit carry, they can be computed by a two-input 17-bit adder. The operands  $S_{(i,j)}$  comes from the Block RAM,  $\alpha, \beta$  come from DSP48E1 and  $\gamma$  is feedback of  $\alpha + \beta + C_\gamma$ . The most significant bit of the output is feedback to the adder as carries  $C_S$  and  $C_\gamma$ . Also, the lower 17-bit of the output is feedback to the adder, while at the same time  $S_{(i+1,j-1)}$  is stored into Block RAM. These can be computed using registers and multiplexers as shown in Figure 2.

3) *Necessary Clock Cycles of Our Algorithms*: In our algorithm, based on the radix- $2^{17}$  number system,  $R$ -bit operands are split into  $d = \lceil R/17 \rceil$  blocks. Let  $MM_{mul}$  denote the number of clock cycles to compute the Montgomery multiplication. In [6], the number is computed by the following equation;

$$MM_{mul} = 2d^2 + d \quad (1)$$

TABLE I  
DATA FLOW OF OUR MONTGOMERY MULTIPLIER

T(clock)	Multiplier(DSP48E1)	Adder(DSP48E1)	Adder(CLB)
...	...	...	...
$k$	$X_0 \cdot Y_i$	$\{C_\beta : \beta\} \leftarrow q \cdot M_d + C_\beta$	$\{C_S : S_{(i,d-2)}\} \leftarrow \gamma + S_{(i-1,d-1)} + C_S$
$k+1$		$X_0 \cdot Y_i + S_{(i,0)}$	$\{C_\gamma : \gamma\} \leftarrow \alpha + \beta + C_\gamma$
$k+2$			$\{C_S : S_{(i,d-1)}\} \leftarrow \gamma + S_{(i-1,d)} + C_S$
$k+3$	$q \leftarrow (X_0 \cdot Y_i + S_{(i,0)}) \cdot (-M^{-1})$		
$k+4$			
$k+5$			
$k+6$	$X_0 \cdot Y_i$		
$k+7$	$q \cdot M_0$	$\{C_\alpha : \alpha\} \leftarrow X_0 \cdot Y_i + C_\alpha$	
$k+8$	$X_1 \cdot Y_i$	$\{C_\beta : \beta\} \leftarrow q \cdot M_0 + C_\beta$	
$k+9$	$q \cdot M_1$	$\{C_\alpha : \alpha\} \leftarrow X_1 \cdot Y_i + C_\alpha$	$\{C_\gamma : \gamma\} \leftarrow \alpha + \beta + C_\gamma$
$k+10$	$X_2 \cdot Y_i$	$\{C_\beta : \beta\} \leftarrow q \cdot M_1 + C_\beta$	$\{C_S : S_{(i+1,-1)}\} \leftarrow \gamma + S_{(i,0)} + C_S$
...	...	...	...
$k+2d+6$	$X_d \cdot Y_i$	$\{C_\beta : \beta\} \leftarrow q \cdot M_{(d-1)} + C_\beta$	$\{C_S : S_{(i+1,d-3)}\} \leftarrow \gamma + S_{(i,d-2)} + C_S$
$k+2d+7$	$q \cdot M_d$	$\{C_\alpha : \alpha\} \leftarrow X_d \cdot Y_i + C_\alpha$	$\{C_\gamma : \gamma\} \leftarrow \alpha + \beta + C_\gamma$
$k+2d+8$	$X_0 \cdot Y_{i+1}$	$\{C_\beta : \beta\} \leftarrow q \cdot M_d + C_\beta$	$\{C_S : S_{(i+1,d-2)}\} \leftarrow \gamma + S_{(i,d-1)} + C_S$
$k+2d+9$		$X_0 \cdot Y_{i+1} + S_{(i+1,0)}$	$\{C_\gamma : \gamma\} \leftarrow \alpha + \beta + C_\gamma$
$k+2d+10$			$\{C_S : S_{(i+1,d-1)}\} \leftarrow \gamma + S_{(i,d)} + C_S$
$k+2d+11$	$q \leftarrow (X_0 \cdot Y_{i+1} + S_{(i+1,0)}) \cdot (-M^{-1})$		
$k+2d+12$			
$k+2d+13$			
$k+2d+14$	$X_0 \cdot Y_{i+1}$		
...	...	...	...

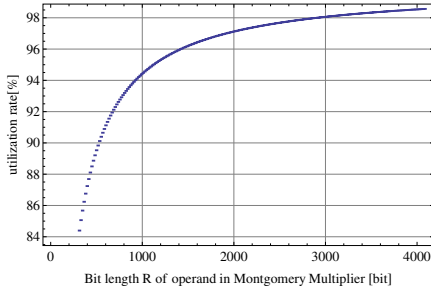


Fig. 3. Embedded multiplier utilization rate of our Montgomery multiplier ( $MM_{mul}/MM_{clk}$ )

The equation means that  $d^2$  multiplications are necessary to compute  $X \cdot Y$  and  $q \cdot M$ , and  $d$  multiplications are needed to obtain  $q$ .

On the other hand, the number of clock cycles  $MM_{clk}$  of our Montgomery algorithm is computed by Equation 2.

$$MM_{clk} = ((d+1) \cdot 2 + 6) \cdot d + 4 = 2d^2 + 8d + 4 \quad (2)$$

It shows that from the 5th to 10th lines of Algorithm 3,  $(d+1) \cdot 2+6$  cycles are necessary for the loop, and  $d$  cycles are needed for loop from 2nd to 11th lines. Also, in order to complete the computation of operands of the modular exponentiation Montgomery circuit shown in Section III-C, another 4 cycles are necessary.

Figure 3 shows the utilization rate of the multiplier in our proposed algorithm. From this figure, when the size of operands  $R$  is larger than 500-bit, the utilization rate is more than 90%. Also, if the size of operands is 2048-bit, the utilization rate is more than 97%. Since the size of operands should be large in practice, our proposed algorithm is optimal for a single DSP48E1 slice.

### C. Our Modular Exponentiation Circuit

In our modular exponentiation circuit, the modular multiplication shown in Algorithm 2 is applied. In the algorithm, the modular exponentiation  $C = P^E \pmod{M}$  can be computed by iterations of the Montgomery multiplication. The block diagram of our modular exponentiation circuit is shown in Figure 4. The internal configuration of Block RAM is shown in Figure 5. The modular exponentiation circuit consists of MM control circuit and ModExp control circuit. The data flow shown in Table I is controlled by MM control circuit, and it supplies the inputs of the multiplier inside of the Montgomery processor. Also, the number of shift for  $E$  to decide the inputs of the Montgomery block by ModExp.

The inputs of modular exponentiation are  $R$ -bit integers  $P, E, M, 2^{2dr} \pmod{M}$  and 17-bit  $-M^{-1}$ . The output is  $R$ -bit integer  $C$ . Also,  $R$ -bit  $S$  is used to store the interim results of Montgomery multiplier. The storage architecture of a  $1k \times 18$ -bit Block RAM is shown as Figure 5. According to the figure, six  $R$ -bit memory spaces and one 17-bit memory space are necessary. In our work, in order to simplify the control circuit, 1k address space is used and split into 8 portions as shown in Figure 5. Furthermore, a flag bit is appended as MSB to every 17-bit block to find the end of each data. Considering the condition  $d = \lceil R/17 \rceil$  of Algorithm 3, when only one Block RAM is used, the maximum size of operands is  $R = 128 \cdot 17 - 3 = 2173$ -bit.

The number of clock cycles necessary to perform modular exponentiation using Algorithm 2 and Montgomery multiplier shown in Section IV can be calculated by Equation 3 and Equation 4. Equation 3 represents the maximum number of cycles when all the bit of  $E$  are 1. Actually it could not happen in practice since  $E$  is a prime number, then, the average number of cycles are computed as Equation 4 which represents the condition that  $E_b/2$ -bit of  $E$  is 1.

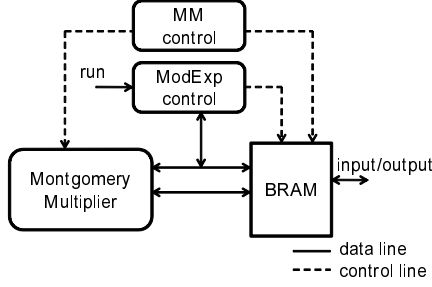


Fig. 4. Structure of our modular exponentiator

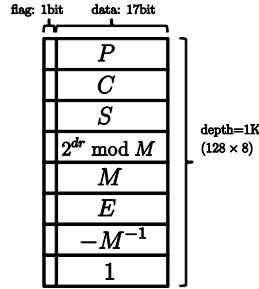


Fig. 5. Internal configuration of BRAM in our modular exponentiator

$$ME_{clk,max} = (2d^2 + 8d + 4) \cdot (2E_b + 3) \quad (3)$$

$$ME_{clk,avr} = (2d^2 + 8d + 4) \cdot (1.5E_b + 3) \quad (4)$$

#### D. Our Parallel Processing Circuit

Our proposed modular exponentiation is implemented on Xilinx Virtex-6 FPGA. The experimental results discussed in Section IV shows that the resource consumption of our algorithm is quite small. Comparing with [8], [10], it seems that our circuit is executed slower. However, the architecture from Nakano's paper [8] has no scalability and architecture from Suzuki's paper [8] uses 17 DSP blocks with substantial slice consuming. Thus, one of the advantages of our proposed architecture is that our modular exponentiation circuit could be executed in parallel conveniently.

Figure 6 illustrates our parallel execution modular exponentiation circuit. In this circuit, 128 processors work in parallel. Each processor is independent and controlled by individual *run* signal. Signal *done* is used when a processor has accomplished its processing. The output *done* signal is the result of *AND* operation collecting all *done* signals of every processor. Although our circuit fetches out only one *done* signal, according to application, each signal *done* flag could be fetched out only doing a little modification. Instead, a parameter *n* can be used to enable necessary number of processors. For example, the maximum number of processors in this circuit is 128. If we set the parameter *n* = 128, all the processors could be used by user. However, if parameter *n* is set to 64, only 64 processors could be used while other

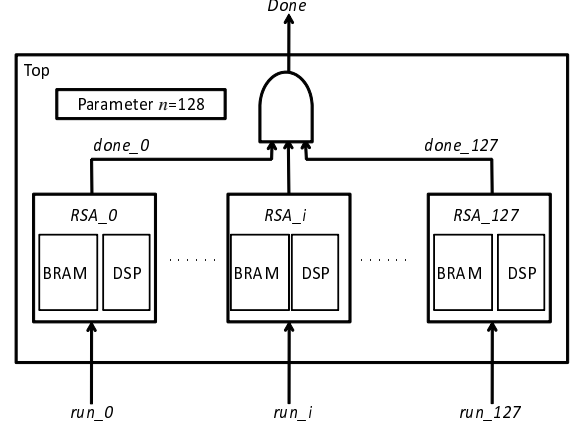


Fig. 6. Our modular exponentiator executed in parallel way

64 will be limited. Moreover, based on different devices, the maximum number of available processors can be calculated by Equation 5,

$$P\_Num = \text{Min}\left(\frac{S\_Slice}{O\_Slice}, DSP, BRAM\right) \quad (5)$$

where  $P\_Num$  represents the maximum processor numbers,  $S\_Slice$  denotes the usable slices of device,  $O\_Slice$  denotes the occupied slices,  $DSP$  means maximum usable DSP blocks of device and  $BRAM$  means maximum usable Block RAMs. Any minimum number of the three elements decide the maximum reproducible number of processors. Since, in practice, our modular exponentiation circuit could be implemented in a small scale, the main impact factor is the number of DSP blocks and Block RAMs. Even if the device contains no DSP blocks, parallel processing is also possible using our architecture only replacing DSP with embedded multiplier.

#### IV. EXPERIMENTAL RESULT AND DISCUSSION

The proposed modular exponentiation circuit is implemented and evaluate on Xilinx Virtex-6 FPGA XC6VLX240T-1, programmed by hardware description language Verilog HDL and synthesized by Xilinx ISE Foundation 11.4 mapping tool.

Table II shows the synthesized result of Virtex-6. As shown in Section III-A, Table II lists the resource cost, where the number of slices means how many configurable logic block we have used. The number of RAMB16s represents how many embedded memory we use. Also the number of DSP48E1s shows how many DSP blocks we used. The maximum frequency means how fast our circuit could be executed. From Table II, it shows that the scale of our circuit is so small that only 180 slices, one BRAM and one DSP48E1 are used. Also, since the maximum clock frequency of DSP48E1 is 600MHz, an extremely high frequency can be obtained by our algorithm. Table III shows the worst execution time of modular exponentiation based on Equation 3 and Table II. Any bit length of operands of Modular exponentiation less

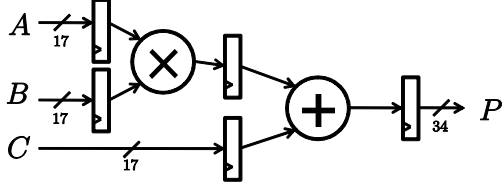


Fig. 7. Internal Configuration of DSP48E1 in Our Architecture

than 2173-bit can be executed in the same circuit without any modification.

Table III shows the number of clock cycles and execution time of our modular exponentiation circuit using Virtex-6. In this table, the number of clock cycles is computed by Equation 3.

For comparison, our proposed algorithm is also implemented on Xilinx Virtex-5 FPGA XC5VVSX50T-1 and Xilinx Virtex-4 FPGA XC4VVSX35-10. Virtex-5 FPGA and Virtex-4 FPGA are the previous generations FPGA produced by Xilinx. Comparing with Virtex-6 FPGA, although there are some differences on programmable logic and DSP block, proposed algorithm can be implemented in these device using almost same Verilog code. However, description of DSP should be modified. For Virtex-5 and Virtex-4 FPGA, their pipeline register and MACC are also contained in DSP block, as shown in Figure 7. Thus proposed algorithm can be applied with the same configuration. Table IV and V show the synthesized results. Although it is difficult to compare the performance because of the different structures of these devices, it is obviously shown that our proposed architecture is compatible to all kind of FPGAs.

TABLE II  
EXPERIMENTAL RESULT OF OUR MODULAR EXPONENTIATOR USING  
VIRTEX-6 FPGA

	Virtex-6
Number of occupied Slices	180/301440
Number of 18k-bit BRAMs	1/416
Number of DSP48E1s	1/768
Maximum Frequency[MHz]	447.027

TABLE IV  
EXPERIMENTAL RESULT OF OUR MODULAR EXPONENTIATOR USING  
VIRTEX-5 FPGA

	Virtex-5
Number of occupied Slices	128/8160
Number of 18k-bit BRAMs	1/132
Number of DSP48Es	1/288
Maximum Frequency[MHz]	362.5

Table VI shows experimental results of our proposed parallel modular exponentiation circuit with the Virtex 6 FPGA. From the table, 128 processors occupy 23040 slices. Since a single processor occupies 180 slices and  $23040 = 180 \times 128$ , no extra slice is necessary in our circuit. Also, 64 Block RAMs are used in our 128 core circuit. Since one Block RAM

TABLE V  
EXPERIMENTAL RESULT OF OUR MODULAR EXPONENTIATOR USING  
VIRTEX-4 FPGA

	Virtex-4
Number of occupied Slices	251/15360
Number of RAMB16s	1/192
Number of DSP48s	1/192
Maximum Frequency[MHz]	291.4

in Virtex 6 is  $2K \times 18k$ -bit in size, it is obvious that two cores have shared one Block RAM. According to Equation 5, slices, BRAM and DSP supporting core number are decided by  $1674 = 301440/180$ ,  $832 = 416 \times 2$  and 768 respectively. Therefore the maximum number of cores supported in our target FPGA is 768. Table VII shows the maximum throughput for above condition.

TABLE VI  
EXPERIMENTAL RESULTS OF OUR PROPOSED PARALLEL MODULAR  
EXPONENTIATION CIRCUIT

	Virtex-6
Number of occupied Slices	23040/301440
Number of 18k-bit BRAMs	64/416
Number of DSP48E1s	128/768
Maximum Frequency[MHz]	447.027

There are a number of literatures reported to implement modular exponentiation using FPGA as described in Section II-B. Performances such as device, circuit size, frequency, execution time and scalability of 1024-bit modular exponentiation circuit are compared in Table VIII. Execution time denotes the worst case when all the 1024-bit of  $E$  are 1. Average case evaluates the execution time corresponding to the average case that a half (512-bit) of 1024-bit of  $E$  is 1. Blum *et al.* [4] implemented a high speed modular exponentiation circuit based on radix-2<sup>4</sup> using Montgomery multiplication. Comparing with proposed algorithm, it is not scalable and too many logic blocks are used without memory blocks or DSP. Nakano *et al.* [8] implemented modular exponentiation by redundant number system and LUT. The scale of circuit is huge and scalability is not supported. However, the authors have used the embedded Block RAMs and embedded Multipliers to achieve a high speed circuit. Suzuki [10] implemented the circuit on Xilinx Virtex-4 FPGA with DSP blocks. In his work, 17 DSP blocks DSP48 are used and little resource costs. His circuit works in extremely high speed and it is scalable. In paper [9], Mazzeo *et al.* have presented that radix-2 based Montgomery multiplier works in Digit-Serial way without memory blocks or DSP blocks. The scale is small. Although Table VIII shows the execution time when  $E = 2^{17} + 1$ , comparing with our work, it is too slow in worst case. As the same as proposed architecture, Alho *et al.* [11] implemented modular exponentiation using DSP blocks in Digit Serial way. Since the target device is different that there are 2 multipliers in DPS block of Alhos' circuit, the number of clock cycles are just half comparing with our algorithm.

Since DSP48E1s and Block RAMs are efficiently used in

TABLE III  
WORST-CASE EXECUTION TIME OF OUR MODULAR EXPONENTIATOR USING VIRTEX-6 FPGA

bit length $R$	64	128	256	512	1024	2048
blocks $d$	4	8	16	31	61	121
frequency[MHz]	447.027	447.027	447.027	447.027	447.027	447.027
clock cycles	9344	51456	332288	2231296	16259072	123940864
execution time[ms]	0.02	0.12	0.74	4.99	36.37	277.26

TABLE VII  
THROUGHPUT OF OUR PROPOSED PARALLEL MODULAR EXPONENTIATION CIRCUIT APPLYING MAXIMUM CORES

bit length $R$	64	128	256	512	1024	2048
maximum processor $P\_Num$	768	768	768	768	768	768
maximum throughput[Mbits/s]	2343.75	781.25	253.378	75.15	20.62	5.41

TABLE VIII  
COMPARISON WITH PREVIOUS 1024-BIT MODULAR EXPONENTIATOR ALGORITHMS

	Blum [4]	Nakano [8]	Suzuki [10]
device	Xilinx XC40250XV	Xilinx XC2VP30-6	Xilinx XC4VFX12-10
logic block	6633 CLBs	11589 Slices	3937 Slices
memory block	-	29 BRAMs	7 BRAMs
DSP block	-	64 $18 \times 18$ -bit multipliers	17 DSP48s
frequency[MHz]	45.6	52.9	400, 200
execution time[ms]	11.95(worst case)	2.52(worst case)	1.71(worst case)
scalable	no	no	yes

	Mazzeo [9]	Alho [11]	this work
device	Xilinx Virtex-E2000-8	Altera Stratix EP1S40	Xilinx XC6VLX240T-1
logic block	1188 Slices	341 LEs	180 Slices
memory block	-	13604-bit	1 BRAM
DSP block	-	1 DSP	1 DSP48E1
frequency[MHz]	86.2	198	447.027
execution time[ms]	$3.86(E = 2^{17} + 1)$	28(average case)	36.37(worst case)
scalable	no	yes	yes

our circuit, the size of our modular exponentiation circuit is very small. Also, the DSP48E1 works almost all the clock cycles shown in Section III-B3. Therefore we have achieved a quality performance with high execution frequency and our architecture could be said most optimal when only 1 multiplier is used.

## V. CONCLUSION

In this paper, we have proposed a hardware algorithm for modular exponentiation using minimum logic units with maximized use of a DSP block. Our hardware algorithm is close to optimal in the sense that running clock cycles is close to the lower bound of the number of multiplications involved in Montgomery multiplication. In other words, a multiplier in a DSP block works during almost all the processing clocks. Our algorithm is evaluated in the latest Xilinx Virtex-6 family FPGA. Experimental result shows that our implementation performs in extremely high speed. Also, our algorithm can be executed in parallel to attain high throughput RSA encryption/decryption.

## REFERENCES

- [1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [2] P. L. Montgomery, "Modular multiplication without trial division," *Math. of Comput.*, vol. 44, pp. 519–521, 1985.
- [3] T. Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," in *Proc. of the 14th IEEE Symposium on Computer Arithmetic*, 1999, pp. 70–77.
- [4] —, "High-radix Montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. on Computers*, vol. 50, no. 7, pp. 759–764, 2001.
- [5] A. F. Tenca and C. K. Koç, "A scalable architecture for Montgomery multiplication," in *Proc. of the First International Workshop on Cryptographic Hardware and Embedded Systems*, 1999, pp. 94–108.
- [6] C. K. Koc, T. Acar, and J. Burton S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, 1996.
- [7] C. McIvor, M. McLoone, and J. V. McCanny, "FPGA Montgomery multiplier architectures - a comparison," *Proc. of Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, vol. 0, pp. 279–282, 2004.
- [8] K. Nakano, K. Kawakami, and K. Shigemoto, "RSA encryption and decryption using the redundant number system on the FPGA," *Proc. of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, vol. 0, pp. 1–8, 2009.
- [9] A. Mazzeo, L. Romano, G. P. Saggese, and N. Mazzocca, "FPGA-based implementation of a serial RSA processor," *Proc. of Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, p. 10582, 2003.
- [10] D. Suzuki, "How to maximize the potential of FPGA resources for modular exponentiation," in *Proc. of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, 2007, pp. 272–288.
- [11] T. Alho, P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Compact modular exponentiation accelerator for modern FPGA devices," *Computers and Electrical Engineering*, vol. 33, no. 5-6, pp. 383–391, 2007.
- [12] Xilinx Inc., "Virtex 6 ML605 Hardware User Guide(v1.2.1)," 2010.
- [13] —, "Virtex-5 FPGA User Guide(v5.2)," 2009.
- [14] —, "Virtex-4 FPGA User Guide(v2.6)," 2008.