International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 2, Number 1, pages 79–96, January 2012

The Parallel FDFM Processor Core Approach for CRT-based RSA Decryption

Yasuaki Ito, Koji Nakano, and Bo Song
Department of Information Engineering
Hiroshima University
1-4-1 Kagamiyama, Higashi-Hiroshima, Hiroshima, 739-8527, Japan

Received: July 22, 2011
Revised: October 31, 2011
Accepted: December 15, 2011
Communicated by Akihiro Fujiwara

Abstract

One of the key points of success in high performance computation using an FPGA is the efficient usage of DSP slices and block RAMs in it. This paper presents a FDFM (Few DSP slices and Few block RAMs) processor core approach for implementing RSA encryption. In our approach, an efficient hardware algorithm for Chinese Remainder Theorem (CRT) based RSA decryption using Montgomery multiplication algorithm is implemented. Our hardware algorithm supporting up-to 2048-bit RSA decryption is designed to be implemented using one DSP slice, one block RAM and few logic blocks in the Xilinx Virtex-6 FPGA. The implementation results show that our RSA core for 1024-bit RSA decryption runs in 13.74ms. Quite surprisingly, the multiplier in the DSP slice used to compute Montgomery multiplication works in more than 95% clock cycles during the processing. Hence, our implementation is close to optimal in the sense that it has only less than 5% overhead in multiplication and no further improvement is possible as long as CRT-based Montgomery multiplication based algorithm is applied. We have also succeeded in implementing 320 RSA decryption cores in one Xilinx Virtex-6 FPGA XC6VLX240T-1 which work in parallel. The implemented parallel 320 RSA cores achieve 23.03 Mbits/s throughput for 1024-bit RSA decryption.

Keywords: RSA decryption, FPGA, Montgomery modular multiplication, Chinese Remainder Theorem, DSP slice

1 Introduction

An FPGA is a programmable logic device designed to be configured by the customer or designer by hardware description language after manufacturing. Since an 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.

Figure 1 illustrates the diagram of the Virtex-6 FPGA developed by Xilinx. The CLB (Configurable Logic Blocks) in Virtex-6 consists of 2 sub-logic blocks called *slice*. Using LUTs (Look Up

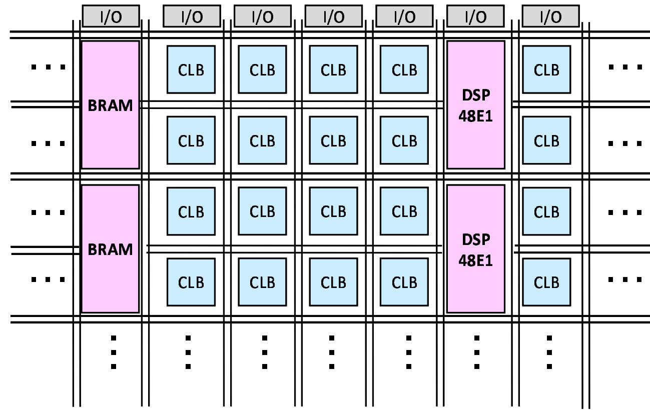

Figure 1: Internal Configuration of Virtex-6 FPGA

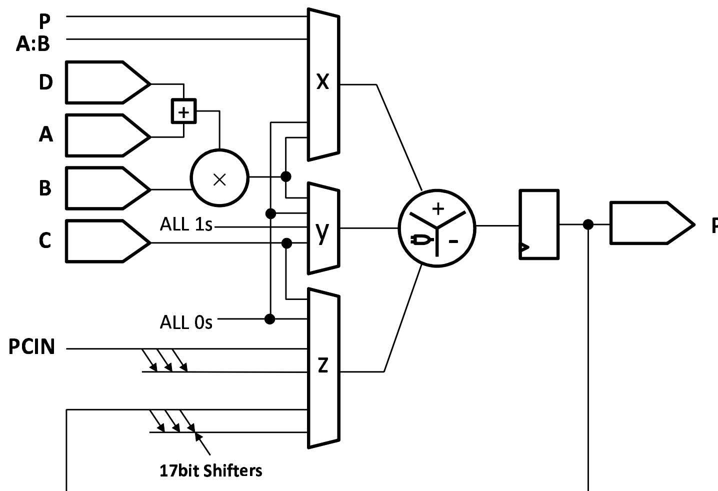

Figure 2: Architecture of DSP48E1

Tables) and Flip-Flops in the slices, various combinatorial circuits and sequential circuits can be implemented. The Virtex-6 FPGAs also have DSP48E1 slices equipped with a multiplier, adders, logic operators, etc. More specifically, as illustrated in Figure 2, the DSP48E1 slice has a two-input multiplier followed by multiplexers and a three-input adder/subtractor/accumulator. The DSP48E1 multiplier can perform multiplication of a 18-bit and a 25-bit two's complement numbers and produces one 48-bit two's complement production. Programmable pipelining of input operands, intermediate products, and accumulator outputs enhances throughput and improves the frequency. The DSP48E1 also has pipeline registers between operators to reduce the delay. The block RAM in the Virtex-6 FPGA is an embedded memory supporting synchronized read and write operations. 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 2k$\times$18-bit dual-port RAM.

RSA [17] is one of the most well known algorithms for public-key cryptography, which is suitable for encryption as well as digital signature. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys such as 1024 bits or more. The RSA algorithm involves three steps: key generation, encryption and decryption. RSA involves a public key and a private key. Messages encrypted with the public key can only be decrypted using the private key. Let p and q be distinct prime numbers chosen uniformly at random with the same bit-length. Their

product $M = p \times q$ is used as the modulus for both the public and private keys. We also select another prime number E, and compute the value $D = E^{-1} \bmod [(p-1)(q-1)]$, where E^{-1} is the modular multiplicative inverse of E. We use a pair (E, M) as a public key to be known everybody and to be use for encryption, and (D, M) as a private key to be secret.

Given a plain text P expressed as a bit sequence corresponding to an integer smaller than M, the RSA encryption can be done by computing the cypher text C using a public key (E, M) as follows:

$$C = P^E \bmod M \tag{1}$$

The original plain text P can be recovered using a private key (D, M) as follows:

$$P = C^D \bmod M \tag{2}$$

Note that E can be usually short in bit-length, say 16 bits for efficient encryption. On the other hand, D becomes as long as the modulus M resulting in huge computing consumption. Thus, the computation of decryption defined by Equation 2 is much larger than that of encryption defined by Equation 1. The main contribution of this paper is to accelerate the decryption using CRT-based decryption algorithm and implement it in the FPGA.

The main contribution of this paper is to present a new approach that we call *the FDFM (Few DSP slices and Few block RAMs) approach*. The key idea of the FDFM approach is to use few DSP slices and few block RAMs to perform routine computation. Let us explain the FDFM approach using a simple example. Figure 3 (1) illustrates a hardware algorithm to compute the output of FIR (Finite Impulse Response) $y_i = a_0 \cdot x_i + a_1 \cdot x_{i-1} + a_2 \cdot x_{i-2} + a_3 \cdot x_{i-3}$. A conventional approach implementing the FIR is to use four DSP slices as illustrated in Figure 3 (2)[20]. In this conventional approach the number of DPS blocks must be the same as that of multipliers in the hardware algorithm. On the other hand, as shown in Figure 3 (3), our FDFM approach uses one or few DSP slices and one or few block RAMs to implement the FIR. The coefficients $a_0, a_1, \ldots$ are stored in the block RAM.

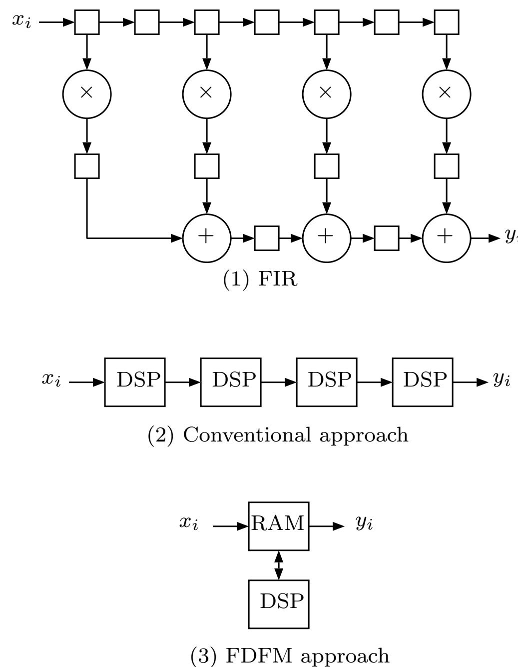

Figure 3: Our FDFM approach

Our FDFM approach has two advantages. First, even if the large main circuit occupies the most of hardware resources in the FPGA, we can implement a necessary hardware algorithm in

the FPGA using remaining few hardware resources as illustrated in Figure 4 (1). Also, if enough hardware resources are available, we can implement multiple FDFM processor cores that work in parallel(Figure 4 (2)). The resulting hardware implementation has maximum throughput by parallel computation. We can use the FPGA effectively by implementing FDFM cores in all the remaining hardware resources in the FPGA to obtain best possible performance. The conventional approach needs DSP slices proportional to the size of hardware algorithm (Figure 3 (1)). Actually, hardware algorithms for RSA encryption/decryption and exhaustive verification of the Collatz conjecture have been implemented in the FPGA using the FDFM approach [4, 10]. Their implementation results are better than the conventional approach [15, 9].

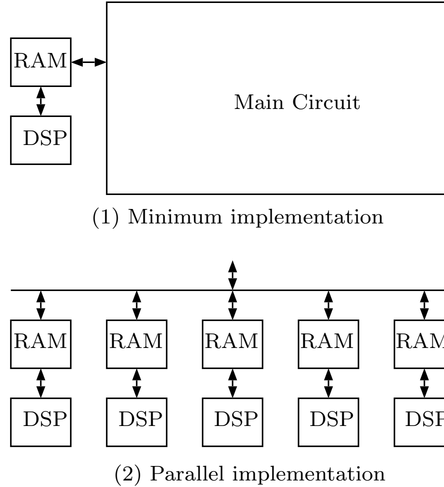

Figure 4: Two advantage of our FDFM approach

In this paper, we propose an efficient hardware algorithm for Chinese Remainder Theorem (CRT) based RSA decryption with Montgomery multiplication algorithm using the FDFM approach to improve our previous work [4]. A single core for RSA decryption uses 1 DSP slice, 1 36k-bit block RAM, and a small quantity of logic blocks in an FPGA. It is noteworthy that this algorithm dose not only need few hardware resources, but also holds the scalability that from 64 bits to 2048 bits RSA decryption can be processed by the same circuit without any modification.

The CRT based RSA decryption algorithm is implemented in Xilinx Virtex-6 FPGA using only one DSP48E1 slice, one block RAM, and few of logic blocks (slices). The implementation results show that our RSA module for 1024-bit RSA decryption runs in 336.700MHz using 4625348 clock cycles, namely 13.737ms. Using our decryption method is based on the CRT, we have achieved approximately 3 times speedup comparing with direct decryption without CRT in our previous work [4].

Our algorithm repeatedly uses a 17-bit multiplier in the DSP48E1 slice. Since the multiplier is used in more than 95% clock cycles over all clock cycles, our implementation is close to optimal in the sense that it has only less than 5% overhead and no further improvement is possible as long as Montgomery modular multiplication based algorithm is applied. We have also succeeded in implementing 320 RSA cores in one Xilinx Virtex-6 FPGA XC6VLX240T-1 which work in parallel. The implemented parallel 320 RSA cores achieve 23.03 Mbits/s throughput for 1024-bit RSA decryption.

This paper is organized as follows: Section 2 shows related works that present RSA encryption/decryption methods. Section 3 presents Modular exponentiation algorithm that is a primary operation in RSA. In Section 4, we show an RSA decryption algorithm using Chinese Remainder Theorem. Section 5 presents a Montgomery modular multiplication circuit and a CRT based RSA decryption circuit with it. Also, we show an implementation of its multicore processor system. In Section 6, we evaluate the performance of our implementation of CRT based RSA decryption.

Finally, Section 7 is a brief conclusion.

2 Related Works

This section shows related works that present RSA encryption/decryption methods.

Recently, many applications have employed GPUs (Graphics Processing Units) as real platforms to achieve an efficient acceleration. To accelerate the RSA encryption/decryption, several research used a GPU support [6, 8]. However, since the iteration of exponentiation in modular exponentiation is not suitable for GPU, the GPU implementations cannot compute RSA encryption/decryption efficiently. Also, Itoh *et al.* implemented RSA decryption on a DSP and achieved a performance of 11.7 ms for 1024-bit RSA decryption [11]. The DSP is an LSI chip for digital signal processing and it consists of 2 multipliers and 6 ALUs. Note that in this paper, we use a *DSP* as an embedded circuit in the FPGA. It is different from the DSP chip.

On the other hand, numerous methods with hardware have been developed. Großschädl proposed an algorithm for RSA decryption by Chinese Remainder Theorem which can half the length of operands and be implemented in a hardware core [7]. Our work carries forward with his algorithm a further step by implementing CRT-based RSA decryption on the FPGA. Also, there are several researches reported to implement modular exponentiation by Montgomery multiplication algorithm [14]. In [5], 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 [5] on FPGAs [13]. Blum and Paar proposed a modular exponentiation hardware algorithm with a radix-2 Montgomery multiplication using systolic array [2]. Also, a radix-2^4 modular exponentiation circuit that is an extended method of the radix-2 circuit is proposed [3]. The circuits above are fixed for the length of operands. However, the following methods that are independent of the length of operands were proposed. Tenca *et al.* presented a radix-2 scalable Montgomery multiplication architecture [19]. This architecture uses fixed processing elements to deal with variable bit length of operands. Nakano *et al.* presented a radix-2^{16} Montgomery multiplier and RSA encryption hardware algorithm using embedded block RAMs of an FPGA efficiently [16]. 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 [12]. 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 slices with radix-2^{17} [18]. Several DSP slices are used to achieve a high operation frequency. Alho *et al.* implemented the modular exponentiation using Altera FPGA with a single DSP slices in radix-2^{18} [1]. In our previous work [4], we have presented an RSA encryption hardware using one DSP slice and one block RAM. The implementation result shows that this hardware performs 1024-bit RSA encryption in 36.37ms in XC6VLX240T-1. The performance issues of above works will be discussed in Section 6.

Above literatures introduce methods to implement modular exponentiation in FPGAs using Montgomery multiplication featuring radix, device and scalability. In general, the computing time of RSA decryption is longer than that of RSA encryption because the size of decryption key is usually much larger than that of RSA encryption. To compute RSA encryption and decryption, all of them compute modular exponentiation directly. In this work, introducing CRT to our previous work [4], we further accelerate RSA decryption. We have also implemented 320 cores in the single Xilinx Virtex-6 FPGA for parallel computation.

3 Modular Exponentiation

Modular exponentiation is a type of exponentiation performed over a modulus and is the primary operation in RSA. RSA encryption and decryption is given by Equation 1 and Equation 2 which are the typical modular exponentiation. In RSA, (E, M) and (D, M) are encryption and decryption keys. During the processing, modular exponentiation is repeated by modular multiplication with fixed E, D and M. Usually, the bit-length of P, C, D and M is at least 1024 which leads to a huge

cost in terms of time and hardware resources. In the most of literatures, Montgomery multiplication algorithm [14] is used as the most efficient algorithm for this problem, which replaces trial division by a series of additions and shift operations that modulo operation is not necessary any more.

3.1 Montgomery Multiplication Algorithm

Montgomery multiplication algorithm [14], introduced in 1985 by Peter Montgomery, allows modular arithmetic to be performed efficiently when the modulus is large. Suppose $X \times Y \bmod M$ is required. This formula implies modular reduction which is very expensive computationally equivalent to dividing two numbers. The Montgomery algorithm is used to compute this formula in much more efficient way than the classical method of taking a product over the integers and reducing the result modulus M.

In the Montgomery algorithm, 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 $X \cdot Y + q \cdot M$ can become zero. The value of q can be computed as follows. Let $(-M^{-1})$ denote the minimum non-negative number such that $(-M^{-1}) \cdot M \equiv -1(\text{ or } 2^R - 1) \pmod{2^R}$. Since M is odd, the situation $(-M^{-1}) < 2^R$ always holds. We can select q such that $q = ((X \cdot Y) \cdot (-M^{-1}))[r-1, 0]$. For this q, $(X \cdot Y + q \cdot M)[r-1, 0]$ will become zero. For 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 $X \cdot Y + q \cdot M$ becomes zero. In this case, $(-M^{-1}) = 11001101(205)$, because $(-M^{-1}) \cdot M \equiv 1100100011111111(51455) \equiv -1 \pmod{2^R}$. Thus $q = (X \cdot Y)[R-1, 0] \cdot (-M^{-1}) = 11000100(196)$ 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 $(X \cdot Y + q \cdot M)[R-1, 0] = 00000000$ and $(X \cdot Y + q \cdot M) \cdot 2^{-R} = (X \cdot Y + q \cdot M)[2R-1, R] = 11110101(245)$. 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 no less than M.

- Algorithm 1: radix-2^r Montgomery Multiplication -
radix-2^r, $d = \lceil R/r \rceil$, $X, Y, M \in \{0, 1, ..., 2^R - 1\}$,
$Y = \sum_{i=0}^{d-1} 2^{ir} \cdot Y_i$, $Y_i \in \{0, 1, ..., 2^r - 1\}$
$(-M^{-1}) \cdot M \equiv -1 \bmod 2^r$, $-M^{-1} \in \{0, 1, ..., 2^r - 1\}$
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. $\quad q_i \leftarrow ((S_i + X \cdot Y_i) \cdot (-M^{-1})) \bmod 2^r$
4. $\quad S_{i+1} \leftarrow (X \cdot Y_i + q_i \cdot M + S_i) \ / \ 2^r$
5. **end for**
6. **if** $(M \leq S_d)$ **then** $S_d \leftarrow S_d - M$

Algorithm 1 shows radix-2^r Montgomery multiplication, where $d = \lceil R/r \rceil$ 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 obtained. As far as now, Montgomery multiplication could be computed by multiplication, addition and shift operations without modulo operations. The result of Montgomery multiplication just needs an amendment by inputting it with 1 into the Montgomery multiplier.

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 $C = P^E \bmod M$. Assume that E is a power of two. Since R and M are fixed, we again 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 $P^E \cdot 2^R \bmod M$. At last, we input 1 and the previous result, that is $(P^E \cdot 2^R \bmod M) \cdot 1 \cdot 2^{-R} \bmod M = P^E \bmod M$. Finally, cypher text C is obtained.

- Algorithm 2: Modular Exponentiation -
$0 \leq E \leq 2^{|E|} - 1$, $E = \sum_{i=0}^{|E|-1} 2^i \cdot E_i, E_i \in \{0, 1\}$
Input: $P, E, M, -M^{-1}, 2^{2dr} \bmod M$
Output: $C = P^E \bmod M$
1. $C \leftarrow \underline{(2^{2dr} \bmod M) \cdot 1 \cdot 2^{-dr} \bmod M}$;
2. $P \leftarrow \underline{(2^{2dr} \bmod M) \cdot P \cdot 2^{-dr} \bmod M}$;
3. **for** $i = |E| - 1$ **downto** 0 **do**
4. $\quad C \leftarrow \underline{C \cdot C \cdot 2^{-dr} \bmod M}$;
5. $\quad$ if $E_i = 1$ then $C \leftarrow \underline{C \cdot P \cdot 2^{-dr} \bmod M}$;
6 **end for**
7. $C \leftarrow \underline{C \cdot 1 \cdot 2^{-dr} \bmod M}$;

Algorithm 2 shows the modular exponentiation using Algorithm 1, where $|E|$ represents the bit length of E. Inputs $2^{2dr} \bmod M$ and $-M^{-1}$ are given beforehand. 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 by Montgomery multiplication of Algorithm 1.

4 CRT-based RSA decryption

The complexity of the RSA decryption defined in Equation 2 directly depends on the size of D and M. The decryption exponent D specifies the numbers of repeated modular multiplications and the modulus M determines the size of the intermediate results. Chinese Remainder Theorem (CRT) provides a method to reduce the size of both D and M so that the complexity of the RSA decryption can be reduced.

Theorem 1 (Chinese Remainder Theorem) *Let n_1, n_2, ..., n_k be k positive integers which are pairwise coprime. For any given set of integers x_1, x_2, ..., x_k, there exists an integer x solving the system of simultaneous congruences:*

$$\begin{gathered} x \equiv x_1 \pmod{n_1} \\ x \equiv x_2 \pmod{n_2} \\ \vdots \\ x \equiv x_k \pmod{n_k} \end{gathered}$$

has a simultaneous solution which is unique modulo $n_1 n_2 \ldots n_k$ and any two solutions are congruent to one another. Furthermore there exists exactly one solution x between 0 and $n - 1$.

Note that the theorem implies that there is a unique solution. However, it does not say how we obtain the value of x. The solution can be obtained by a method known as Gauss's algorithm as follows. Let $N = n_1 n_2 \cdots n_k$, $N_i = N/n_i$ and $d_i = {N_i}^{-1} \pmod{n_i}$ $(1 \leq i \leq k)$. We have,

$$x = x_1 N_1 d_1 + \cdots + x_k N_k d_k \pmod{N}. \tag{3}$$

From the Fermat's Little Theorem, we have $N_i^{n_i - 1} \pmod{n_i} = 1$. Thus, d_i can be easily computed by the following formula: $d_i = {N_i}^{-1} \pmod{n_i} = N_i^{n_i - 2} \pmod{n_i}$.

We use Equation 3 for $k = 2$ to perform RSA decryption defined in Equation 2. Since $M = pq$ and the Chinese Remainder Theorem, the value of P can be computed by the following two equations:

$$P_p = C^D \bmod p = C_p^{D_p} \bmod p \tag{4}$$

$$P_q = C^D \bmod q = C_q^{D_q} \bmod q, \tag{5}$$

where $C_p = C \bmod p$, $C_q = C \bmod q$, $D_p = D \bmod (p-1)$, and $D_q = D \bmod (q-1)$. Let $Z_p = q^{p-1} \bmod M$ and $Z_p = p^{q-1} \bmod M$. Once we have the values of P_q and D_q, we can compute the value of P by Equation 3 by the following formula:

$$\begin{aligned} P &= (P_p q(q^{-1} \bmod p) + P_q p(p^{-1} \bmod q)) \bmod M \\ &= (P_p q(q^{p-2} \bmod p) + P_q p(p^{q-2} \bmod q)) \bmod M \\ &= (P_p(q^{p-1} \bmod M) + P_q(p^{q-1} \bmod M)) \bmod M \\ &= (P_p Z_p + P_q Z_q) \bmod M \qquad (6) \end{aligned}$$

Note that D_p, D_q, Z_p and Z_q can be precomputed, because their values are independent of the value of P. In summary, the following steps can perform the RSA decryption, that is, can compute the plain text P from the cypher text C.

CRT-based RSA decryption

Step 1: Compute $C_p = C \bmod p$ and $C_q = C \bmod q$.

Step 2: Compute $P_p = C_p^{D_p} \bmod p$ and $P_q = C_q^{D_q} \bmod p$.

Step 3: Compute $S_p = P_p Z_p \bmod M$ and $S_q = P_q Z_q \bmod M$.

Step 4: Compute the sum $P = S_p + S_q$. If $P \geq M$ then let $P = P - M$.

Let us briefly compare the computational costs of the direct RSA decryption by Equation 2 and the CRT-based RSA decryption. Suppose that both p and q has $R/2$ bits, and thus, M has R bits. Then, the decryption key and the cypher text C can have R bits. We assume that the computational cost of Equation 2 is R^3, and roughly evaluate the computational cost of the CRT-based RSA decryption. Since in the CRT-based RSA decryption, the cost of Step 2 is dominant, we ignore the other steps. Since all of the p, C_p and D_p has $R/2$ bits, the computational cost of P_p is $R^3/8$. Similarly, that of P_q is also $R^3/8$. Thus, the total cost of Step 2 is $R^3/4$. Consequently, the CRT-based RSA decryption can reduce the computational cost by quarter.

5 Implementation on the FPGA

This section mainly shows a Montgomery modular multiplication circuit and a CRT based RSA decryption circuit with it. In our hardware algorithm, we use an embedded DSP slice and a block RAM in Xilinx FPGA. Also, we introduce an implementation of its multicore processor system.

5.1 Our Montgomery Modular Multiplication Algorithm

Algorithm 3 shows our hardware algorithm of Montgomery multiplication. Let $\{A : B\}$ denote a concatenation of A and B. For example, $\{A : B\} = (FFEC)_{16}$ for $A = (FF)_{16}$ and $B = (EC)_{16}$. Algorithm 3 is an improved algorithm from Algorithm 1 introduced in Section 3.1. Our circuit performs radix-2^{17} based algorithm to match the size of inner multiplier in DSP48E1. Let R denote the size of Montgomery multiplier operands X, Y, and M, then $d = \lceil R/17 \rceil$ is the number of digits of the operands. If $17d \geq R+3$, the subtraction shown in the 6th line of Algorithm 1 can be ignored. If at least 3-bit 0 is padded to the most significant bits of the highest digit as the redundancy, we can guarantee such condition is satisfied. Due to the stringent page limitation, the proof is omitted. Furthermore, $M \geq C$ is always satisfied in the modular exponentiation shown in Algorithm 2. In the practical, the size of operands is radix-2 numbers such as 512-bit, 1024-bit, 2048-bit, and 4096-bit. For the radix-2^{17} system, the condition $17d \geq R+3$ is just satisfied. If the condition is not satisfied, we can append one redundant digit at the highest digit. Thus our hardware Montgomery algorithm does not perform the reduction at last.

Algorithm 3 is a radix-2^{17} digit serial Montgomery algorithm from Algorithm 1. In other words, each 17 bits, 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. While C_α, C_β, C_γ, and C_S are carries and they are added at the next loop. In the algorithm, C_α and C_β are 17-bit carries for 17-bit MACC, and C_γ and 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_α are computed. The resulting upper 17-bit denotes a carry C_α which can be added at next loop. While the lower 17-bit of result is α 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}, $d = \lceil R/17 \rceil, 17d \geq R+3$,
$X, Y, M, S_i \in \{0, 1, ..., 2^R - 1\}$,
$-M^{-1}, \alpha, \beta, \gamma, C_\alpha, C_\beta \in \{0, 1, ..., 2^{17} - 1\}$, $C_\gamma, C_S \in \{0, 1\}$,
$X = \sum_{i=0}^{d-1} 2^{17i} \cdot X_i, X_i \in \{0, 1, ..., 2^{17} - 1\}, X_d = 0$
$Y = \sum_{i=0}^{d-1} 2^{17i} \cdot Y_i, Y_i \in \{0, 1, ..., 2^{17} - 1\}$
$M = \sum_{i=0}^{d-1} 2^{17i} \cdot M_i, M_i \in \{0, 1, ..., 2^{17} - 1\}, M_d = 0$
$S_i = \sum_{j=0}^{d-1} 2^{17j} \cdot S_{(i,j)}, S_{(i,j)} \in \{0, 1, ..., 2^{17} - 1\}, S_d = 0$
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**

5.1.1 Architecture of Montgomery Multiplier

Figure 5 shows the architecture of Montgomery multiplier using Algorithm 3. The inputs of Montgomery multiplier are supplied from a block RAM and registers of CRT based RSA decryption 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 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. The number of clock cycles necessary to compute q is 6. In the 6th line, 17-bit multiplication $X_j \cdot Y_i$ is computed and the carry C_α 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 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. 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 registers in the DSP48E1, our circuit is able to be performed efficiently.

The adder, that is composed by CLBs, following the DSP48E1 computes $\alpha + \beta + C_\gamma$ and $\gamma + S_{(i,j)} + C_S$ of the 8th and 9th lines in the Algorithm 3. Since C_γ and C_S are 1-bit carries, they can be computed by a two-input 17-bit adder. The operands $S_{(i,j)}$ come from the block RAM, α and β come from DSP48E1, and γ is a feedback of $\alpha + \beta + C_\gamma$. The most significant bit of the output is

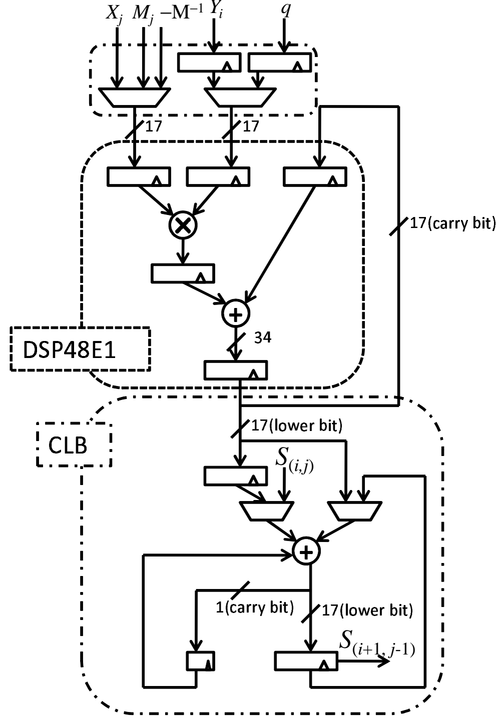

Figure 5: Structure of our Montgomery multiplier

a feedback to the adder as carries C_S and C_γ. Also, the lower 17-bit of the output is a feedback to the adder, while at the same time $S_{(i+1,j-1)}$ is stored into the block RAM. These can be computed using registers and multiplexers as shown in Figure 5.

5.1.2 Necessary Clock Cycles of Our Montgomery Algorithm

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 [5], the number is computed by the following equation:

$$MM_{mul} = 2d^2 + d \tag{7}$$

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 8.

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

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

Figure 6 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 1024-bit and 2048-bit, the utilization rate is more than 95% and 97%, respectively. Since the size of operands should be large in practice, our proposed algorithm is optimal for a single DSP48E1 slice.

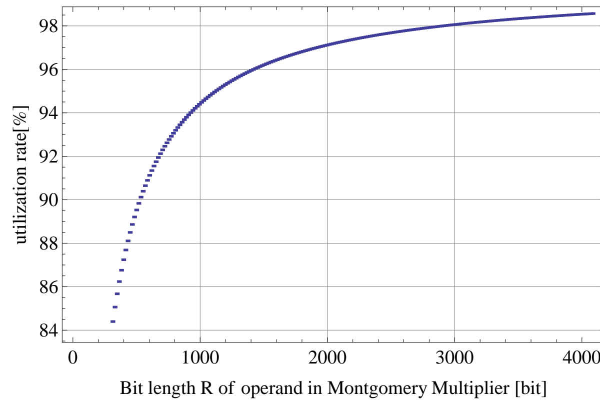

Figure 6: Embedded multiplier utilization rate of our Montgomery multiplier(MM_{mul}/MM_{clk})

5.2 Our CRT based RSA decryption circuit

5.2.1 Architecture of CRT based RSA decryption circuit

Recall that there are 4 steps to compute CRT based RSA decryption. In Step 1, Step 2, and Step 3, two independent but same computations can be performed, respectively. More specifically, in Step 1, $C_p = C \bmod p$ and $C_q = C \bmod q$, in Step 2, $P_p = C_p^{D_p} \bmod p$ and $P_q = C_q^{D_q} \bmod p$, and in Step 3, $S_p = P_p Z_p \bmod M$ and $S_p = P_p Z_p \bmod M$ can be computed in parallel, respectively. However, to reduce the cost of the hardware resource, we process them in serial. In other words, we serially compute C_p, C_q, P_p, P_q, S_p, and S_q in our CRT based decryption circuit. Also, in Step 4, it is easy to obtain the final result P only by adding S_p and S_q together.

We just discuss one of the procedures in the following paragraphs and assume the modulus M is 1024 bits. The other is totally the same. In Step 1, $C_p = C \bmod p$ is computed. In the case of 1024-bit RSA, C is 1024 bits while C_p is 512 bits on the assumption that the size of p is 512 bits. We first input C and ${R_p}^2 \bmod p$ into Montgomery multiplier introduced in Section 5.1.1 to get an intermediate result, where R_p represents the integer $2^{bitlength(p)}$ which is computed beforehand. Next the intermediate result and 1 (padding to 512 bits with all 0 in the significant bits) are input to correct the result.

Step 2 is a typical modular exponentiation. We compute it by Algorithm 2. Note that in this step, all the operands are 512 bits if the size of modulus M is 1024 bits.

Step 3 is a single Montgomery multiplication as the same as Step 1. We also first input P_p with $R^2 \bmod M$ which is compute beforehand. Next we compute $S_p = P_p Z_p \bmod M$ by input intermediate result with Z_p. At last, again the intermediate result and 1(padding to 512 bit with all 0 in the significant bits) are input to get the final result. In Step 3, we compute a reduction that the production of a 512-bit operand and a 1024-bit operand with a 1024-bit modulus.

Note that the sizes of operands in each step are different. In our Montgomery multiplier shown in Figure 5, if the size of input Y equals to the size of modulus, we can guarantee that the computation is correct.

In our CRT based RSA decryption circuit, we use only one DSP slice and one block RAM. The circuit contains two controllers. One is used for Montgomery algorithm and the other is used for the state machine and deciding the address between DSP slice and block RAM. The block diagram of our CRT based decryption circuit is shown in Figure 7.

The size of a single block RAM in Virtex-6 is 36k bits. In order to reduce the hardware resource cost, only one block RAM is used in our circuit. The 36k-bit block RAM is split to 2 sub-blocks as shown in Figure 8. The upper one is used to store the plain text, cypher text, encryption parameters as well as intermediate results, and thus furthermore split to smaller blocks with each size in 128×18 bits. Our circuit is a decryption circuit, while we also use the circuit as encryption. Thus we have reserved a space for encryption. The lower space of the block RAM is also split to several parts

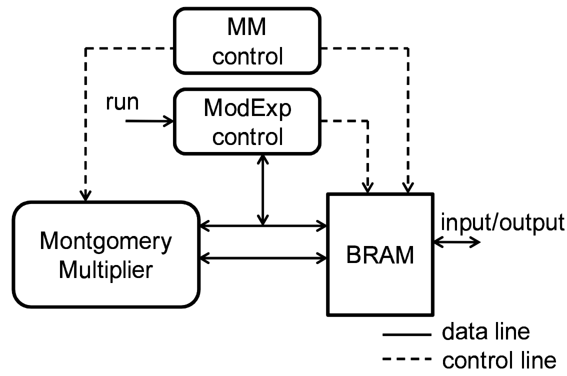

Figure 7: Structure of our CRT based RSA decryption circuit

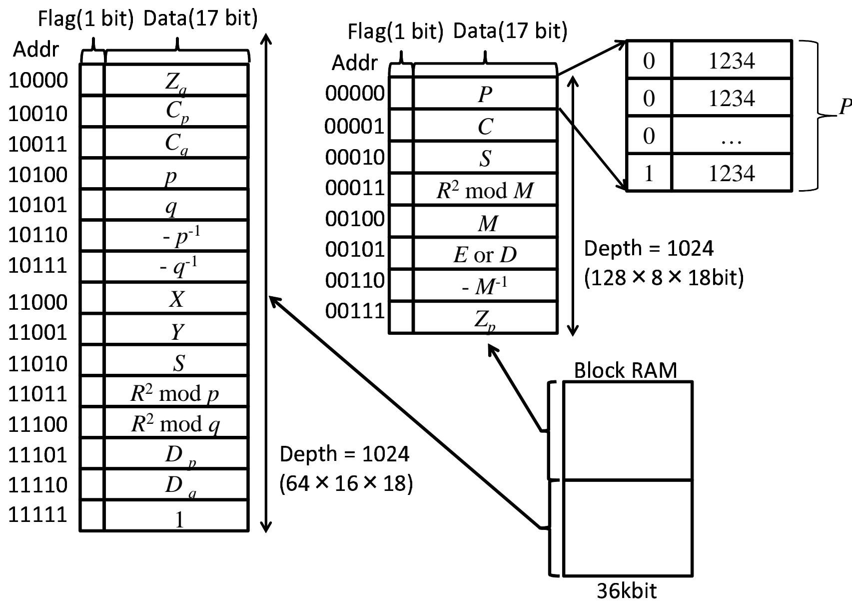

Figure 8: Internal configuration of block RAM in our CRT based RSA decryption circuit

to store CRT based parameters, intermediate result and decrypted text. Since in our CRT based circuit, the size of operands is half, the size of the lower sub-block is half compared with upper sub-blocks, that is 64×18 bits.

Note that our algorithm is radix-17 based data, which means that we split operands into every 17 bits. However, the width of our block RAM is 18 bits. The most significant bit of each data is used as a flag bit to indicate the end 17 bits of the input sequence. It means if a flag bit is equal to 1, all 17-bit blocks of the operand has been input to the Montgomery multiplier. Since every sub-block is 64×18 bits, our circuit supports RSA decryption from 17 bits to 2176 ($64 \times 17 \times 2$) bits without any modification. Therefore, our circuit can be said scalable.

5.2.2 Necessary clock cycles of CRT based RSA decryption

The number of necessary clock cycles of a single Montgomery multiplication can be computed by Equation 8. Note that this equation is for the ordinary Montgomery multiplication whose size of 2 inputs X, Y and modulus M is the same. However, in our CRT based RSA decryption, the size of operands is different since the number of digits d for X and Y is different. We modify Equation 8 to be fit for our algorithm as following,

$$MM_{clk} = ((d_1 + 1) \cdot 2 + 6) \cdot d_2 + 4 = 2d_1d_2 + 8d_2 + 4 \tag{9}$$

where d_1 and d_2 denote the numbers of digits for input X and Y, respectively. Specifically, suppose the modulus M is 1024 bits, then the sizes of these two inputs are 1024 bits and 512 bits, respectively. That is $d_1 = \lceil 1024/17 \rceil = 61$ and $d_2 = \lceil 512/17 \rceil = 31$. With Equation 9, we can compute the necessary clock cycles for our CRT based algorithm.

In our implementation, the first 3 steps are processed. In Step 1, Montgomery multiplication is performed twice with different input size. Step 2 is a modular exponentiation. Therefore Equation 8 is available. Note that the size of operands in Step 2 is half of the size of Step 1. Thus the size of operands is $d = d_2$. In Step 3, three times of Montgomery multiplication are necessary with different input size. Finally, we can obtain the necessary clock cycles as follows:

$$\begin{aligned} CRT_{clk} &= 2 \times \{(2{d_2}^2 + 8d_2 + 4)(2d_2 \times 17 + 4) \\ &\quad + (2d_1d_2 + 8d_2 + 4) + (2d_2d_1 + 8d_1 + 4) \\ &\quad + (2d_1d_2 + 8d_1 + 4)\} \\ &= 4(34{d_2}^3 + 140{d_2}^2 + 3d_1d_2 + 8d_1 + 88d_2 + 14) \end{aligned} \tag{10}$$

5.3 Multicore System

We have implemented a multicore system that contains many processor cores of the FDFM approach that works in parallel. Figure 9 shows the structure of our multicore system that has 320 cores. Since the number of cores is large, if a common data bus is used to connect to each core, a large and complicated multiplexer is necessary. It causes the decrease of the frequency of the circuit. Therefore, in our system, shift registers are used instead of a data bus. Each shift register consists of a 1-bit send/receive flag, a 9-bit core ID, an 11-bit address, and an 18-bit send/receive data. The value stored in each shift register is moved from left to right in every clock cycle. To send an 18-bit data d to address a of the block RAM in core p, 0(send), p, a, and d are input to the leftmost shift register. After p clock cycles, these values are moved to core p, and d is stored to address a. On the other hand, to receive an 18-bit data d from address a of the block RAM in core p, 1(receive), p, a, and d are input to the leftmost shift register. After p clock cycles, these values are moved to core p, and d stored in address a is output to the shift register. The data is moved to output port through the shift registers. Since shift registers are used, 18-bit data can be supplied in every clock cycle. Therefore, it takes $320 \times d_1$ clock cycles necessary to store all input data. From Equation 10, compared with the number of clock cycles to compute RSA decryption, the number of the clock cycles is small enough.

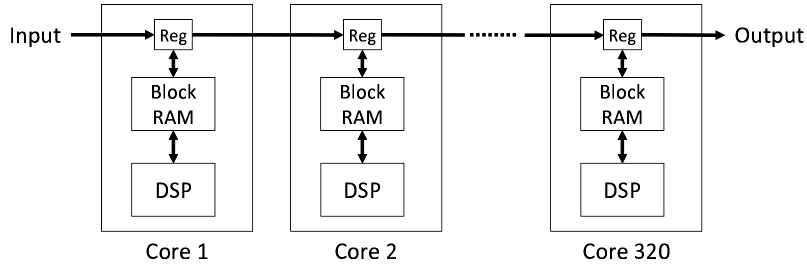

Figure 9: Structure of the multicore system

Table 1: Synthesis result of our CRT based RSA decryption system

	1 core	320 cores
Slices (max. 37,680)	140	34,635
36k-bit block RAMs (max. 416)	1	320
DSP48E1s (max. 768)	1	320
Maximum Frequency[MHz]	336.700	325.098

6 Experimental Result and Discussion

The proposed CRT based RSA decryption circuit is implemented and evaluated on the Xilinx Virtex-6 FPGA XC6VLX240T-1, programmed by hardware description language Verilog HDL and synthesized by Xilinx ISE Foundation 13.1.

Table 1 shows the synthesized result in the Virtex-6 and lists the resource costs for single core and multicore systems. According the table, the size of the single core system is quite small. Also, the frequency of the multicore system is almost same as that of the single core system. In general, if the size of circuit is large, the frequency is decreased. However, our processing core is very compact and each core is connected by shift registers as shown in Figure 9. Therefore, although the size of circuit is increased, the frequency of the multicore system is not decreased.

Table 2 shows the necessary clock cycles and execution time in the worst case from 64-bit to 2048-bit CRT based RSA decryption. The execution time is computed by Equation 10 and Table 2. Any size of operands less than 2176-bit can be executed in the same circuit without any modification. Our CRT based RSA decryption circuit can process 1024-bit RSA decryption in 11.263 ms. Using CRT, it achieves nearly 2.6 times speedup comparing with our previous work. Since our previous RSA circuit [4] does not use CRT, the architecture is less complicated, the maximum frequency is higher and up to 447.027 MHz. If our circuit works on the same frequency, in theorem, we can achieve at most 4 times speedup by CRT based algorithm. However, our circuit works in 336.700 MHz. Thus the speedup is less than 4 times.

There are a number of literatures reported to implement RSA using FPGA as described in

Table 2: Execution time of our CRT based RSA decryption circuit for the worst-case

Bit length R	64	128	256	512	1,024	2,048
Blocks d_1	4	8	16	31	61	121
Blocks d_2	2	4	8	16	31	61
Clock cycles	4,312	19,768	110,392	713,048	4,625,348	33,067,148
Time with CRT[ms]	0.013	0.059	0.328	2.118	13.737	98.210
Time without CRT[ms] in [4]	0.020	0.113	0.742	4.995	36.402	277.413

Section 2. Performances such as device, circuit size, frequency, execution time, throughput and scalability for 1024-bit RSA decryption are compared in Table 3. Recall that in the RSA decryption, the modular exponentiation $P = C^E \bmod M$ is computed, where P and C are plain text and cypher text, respectively, and (E, M) is a decryption key. In typical 1024-bit RSA decryption, the bit length of E and M is approximately 1024. Also, its execution time depends on the size of E and the number of 1's in E. In the execution time in Table 3, the worst case means that all the 1024-bit of E are 1 and average case means that half of 1024-bit of E is 1. Blum *et al.* [3] implemented a high speed modular exponentiation circuit based on radix-2^4 using Montgomery multiplication. Comparing with proposed algorithm, it is not scalable and too many logic blocks are used without memory blocks or DSP blocks. Nakano *et al.* [16] 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 used a pipeline structure whose registers are composed by logic blocks. Mazzeo *et al.* have shown that radix-2 based Montgomery multiplier can run in Digit-Serial way without memory blocks or DSP blocks [12]. Their circuit has a small scale, however nevertheless about 6 times larger than our circuit. They evaluated the performance using $E = 2^{17} + 1$, which means that the size of E is only 18 bits long, and that there are only two bits in E set to 1, that is only the most significant bit and the least significant bit are 1. Such a small bit length is reasonable for encryption, but safe decryption typically requires lengths around 1024 bits. Under the same $E = 2^{17} + 1$, our method outperforms Mazzeo *et al.*'s by a factor of 33 times. Similar to the proposed architecture, Alho et al. implemented modular exponentiation using one DSP block in Digit Serial way [1]. However, their DSP block requires two multipliers, while only one is necessary in our solution. If we had used 2 multipliers, the two multiplications listed in the Algorithm 3 (lines 6 and 7) could have being computed in parallel, thus reducing the computation time. Our method is also faster in the average case, although a direct comparison is difficult due to the fact that Alho et al. used a different FPGA. Itoh *et al.* have used a DSP chip that is an LSI chip for digital signal processing, not an FPGA [11]. It is difficult to compare the performance to that of FPGA implementations directly. However, the DSP chip consists of two 16-bit multipliers and six ALUs. Since our implementation uses 1 DSP and 1 block RAM in each core, the performance is better than that of our implementation for one core. On the other hand, our multicore implementation is much better performance.

In our previous work [4], we have presented an RSA encryption hardware using one DSP slice and one block RAM. In this work, introducing CRT to our previous work, we further accelerate RSA decryption. Also, optimizing the design of the circuit, the size of circuit in this work is less than that of previous one. However, since the circuit becomes complicated introducing CRT, the frequency is reduced from 447.027MHz to 336.700MHz.

In our circuit, since DSP48E1 slices and block RAMs are efficiently used, the size of our modular exponentiation circuit is very small. Also, the DSP48E1 works almost all the clock cycles shown in Section 5.1.2. 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. Also, since the architecture of our single core is so compact, we can implement multiple cores in parallel system easily. Thus, an extremely high throughput can be obtained by our system. For 1024-bit RSA decryption, the maximum throughput is up to 23.031Mbit/s.

7 Conclusion

This paper introduced the FDFM (Few DSP slices and Few block RAMs) approach for the FPGA design. In the FDFM approach multiple processor cores with few DSP slices and few block RAMs are used. We have proposed a hardware algorithm for CRT based RSA decryption using minimum logic units with maximized use of a DSP slice. 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 slice 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 and

Table 3: Comparison with related works for 1024-bit RSA decryption

	Blum [3]	Nakano [16]	Suzuki [18]
Device	Xilinx XC40250XV	Xilinx XC2VP30-6	Xilinx XC4VFX12-10
Logic block	6,633 CLBs	11,589 Slices	3,937 Slices
Memory block	—	29 BRAMs	7 BRAMs
DSP slice	—	64 18× 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)
Throughput[kbit/s]	85.690	406.349	598.830
Scalable	No	No	Yes
CRT	No	No	No

	Mazzeo [12]	Alho [1]	Itoh [11]
Device	Xilinx Virtex-E2000-8	Altera Stratix EP1S40	TI TMS320C6201
Logic block	1,188 Slices	341 LEs	—
Memory block	—	13,604-bit	—
DSP slice	—	1 DSP	2 multipliers, 6 ALUs
Frequency[MHz]	86.2	198	200
Execution time[ms]	3.86($E = 2^{17} + 1$)	28(average case)	11.7(worst case)
Throughput[kbit/s]	265.285	36.571	87.521
Scalable	No	Yes	Yes
CRT	No	No	Yes

	Our previous work [4]	This work (1 core)	This work (320 cores)
Device	Xilinx XC6VLX240T-1	XC6VLX240T-1	XC6VLX240T-1
Logic block	180 Slices	140 Slices	34,635 Slices
Memory block	1 BRAM	1 BRAM	320 BRAMs
DSP slice	1 DSP48E1	1 DSP48E1	320 DSP48E1
Frequency[MHz]	447.027	336.700	325.098
Execution time[ms]	36.401(worst case)	13.737(worst case)	14.228(worst case)
Throughput[kbit/s]	28.130	74.542	23,031.370
Scalable	Yes	Yes	Yes
CRT	No	Yes	Yes

throughput.

References

[1] Timo Alho, Panu Hämäläinen, Marko Hännikäinen, and Timo D. Hämäläinen. Compact modular exponentiation accelerator for modern FPGA devices. *Computers and Electrical Engineering*, 33(5-6):383–391, 2007.

[2] Thomas Blum and Christof Paar. Montgomery modular exponentiation on reconfigurable hardware. In *Proc. of the 14th IEEE Symposium on Computer Arithmetic*, pages 70–77, 1999.

[3] Thomas Blum and Christof Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Trans. on Computers*, 50(7):759–764, 2001.

[4] Song Bo, Kensuke Kawakami, Koji Nakano, and Yasuaki Ito. An RSA encryption hardware algorithm using a single DSPblock and a single block RAM on the FPGA. *International Journal of Networking and Computing, International Journal of Networking and Computing*, 1(2):277–289, 2011.

[5] Çetin Kaya Koç, Tolga Acar, and Burton S. Kaliski, Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[6] Wenjun Fan, Xudong Chen, and Xuefeng Li. Parallelization of RSA algorithm based on compute unified device architecture. In *Proc. of The Ninth International Conference on Grid and Cloud Computing*, pages 174–178, 2010.

[7] Johann Großschädl. The Chinese remainder theorem and its application in a high-speed RSA crypto chip. In *Computer Security Applications, 2000. ACSAC '00. 16th Annual Conference*, pages 384–393, 2000.

[8] Owen Harrison and John Waldron. Public key cryptography on modern graphics hardware. In *Booklet of posters, Eurocrypt 2009*, April 2009.

[9] Yasuaki Ito and Koji Nakano. A hardware-software cooperative approach for the exhaustive verification of the collatz conjecture. In *Proc. of International Symposium on Parallel and Distributed Processing with Applications*, pages 63–70, 2009.

[10] Yasuaki Ito and Koji Nakano. Efficient exhaustive verification of the Collatz conjecture using DSP blocks of Xilinx FPGAs. *International Journal of Networking and Computing*, 1(1):49–62, 2011.

[11] Kouich Itoh, Masahiko Takenaka, Naoya Torii, Syouji Temma, and Yasushi Kurihara. Fast implementation of public-key cryptographic on a dsp tms320c6201. In *Proc. of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 61–72, 1999.

[12] A. Mazzeo, L. Romano, G. P. Saggese, and N. Mazzocca. FPGA-based implementation of a serial RSA processor. In *Proc. of Design, Automation and Test in Europe Conference and Exhibition*, 2003.

[13] Ciaran McIvor, Máire McLoone, and John V. McCanny. FPGA Montgomery multiplier architectures - a comparison. In *Proc. of Field-Programmable Custom Computing Machines*, pages 279–282, 2004.

[14] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.

[15] Koji Nakano, Kensuke Kawakami, and Koji Shigemoto. RSA encryption and decryption using the redundant number system on the FPGA. In *Proc. IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, May 2009.

[16] Koji Nakano, Kensuke Kawakami, and Koji Shigemoto. RSA encryption and decryption using the redundant number system on the FPGA. In *Proc. of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8, May 2009.

[17] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[18] Daisuke 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*, pages 272–288, 2007.

[19] Alexandre F. Tenca and Cetin Kaya Koç. A scalable architecture for Montgomery multiplication. In *Proc. of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 94–108, 1999.

[20] Xilinx Inc. *XtremeDSP for Virtex-4 FPGA UG073 (v2.7)*, 2008.