CRT-based DSP Decryption using Montgomery Modular Multiplication on the FPGA

Bo Song, Yasuaki Ito, and Koji Nakano
Department of Information Engineering
School of Engineering, Hiroshima University
1-4-1 Kagamiyama, Higashi-Hiroshima, Hiroshima, 739-8527, JAPAN
{*songbo,nakano,kawakami*}*@cs.hiroshima-u.ac.jp*

***Abstract*—The main contribution of this paper is to present an efficient hardware algorithm for Chinese Remainder Theorem (CRT) based RSA decryption using Montgomery multiplication algorithm. Our hardware algorithm supporting up-to 2048-bit RSA decryption is designed to be implemented using one DSP48E1 block, 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 11.263ms. Quite surprisingly, the multiplier in DSP block 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 cores in one Xilinx Virtex-6 FPGA XC6VLX240T-1 which work in parallel. The implemented parallel 320 RSA cores achieve 26.2 Mbit/s throughput for 1024-bit RSA decryption.**

***Keywords*-RSA decryption; FPGA; Montgomery modular multiplication; Chinese Remainder Theorem; DSP block;**

I. Introduction

The main contribution of this paper is to present an efficient hardware algorithm for Chinese Remainder Theorem (CRT) based RSA decryption using Montgomery multiplication algorithm. Our algorithm uses only one DSP block, and one Block RAM and a small quantity of logic blocks in an FPGA. It is noteworthy that this algorithm does 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 block, one Block RAM, and a few of logic blocks (slices). The implementation results show that our RSA module for 1024-bit RSA decryption runs in 410.678MHz using 4625348 clock cycles, namely 11.263ms. Since our decryption method is based on the CRT. We have achieved 4 times speedup comparing with direct decryption by modular exponentiation.

Our algorithm repeatedly uses a 17-bit multiplier in the DSP48E1 block. 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 26.2 Mbit/s throughput for 1024-bit RSA decryption.

A. RSA

RSA [1] 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. The public key can be known to everyone and is used for encrypting messages. 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 to 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

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.208

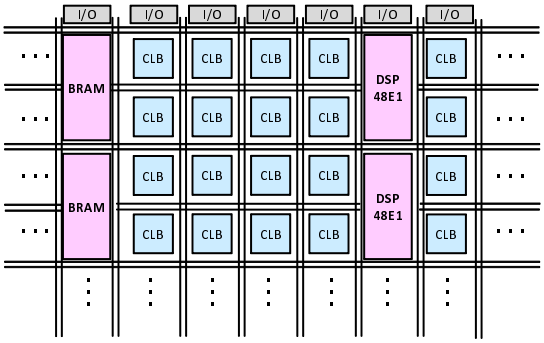

Figure 1. Internal Configuration of Virtex-6 FPGA

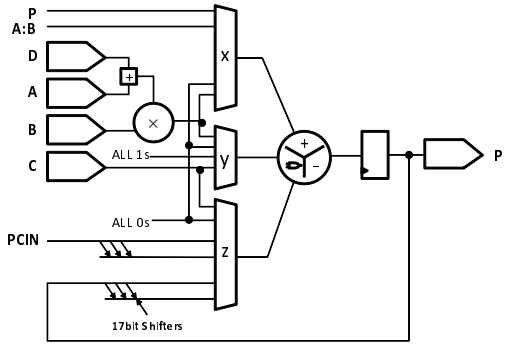

Figure 2. Architecture of DSP48E1

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.

B. FPGA

An FPGA is an integrated circuit designed to be configured by the designer after manufacturing. The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform, but cost less and can easily re-configured if any modification is made. FPGAs show plenty of advantages for many applications thus FPGAs are selected as our target device to evaluate our hardware algorithm of CRT based RSA decryption.

This paper applies the algorithm on Xilinx Virtex-6 FPGA which is the latest high performance FPGA appearing recently [2].

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). 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. 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 [3]. Also, Virtex-6 FPGAs have a Block RAM to compensate for memory insufficiency of CLBs. If these blocks are used in design, the consumption of CLBs can be cut down a lot. Since consumption of CLBs is an essential evaluating indicator, a lower consumption usually implies lower cost and power consumption. Designers always try their best to reduce the use of logic blocks in the FPGA on the assumption that a high frequency can be achieved.

As the Figure 2 shows, the DSP48E1 block has a two-input multiplier followed by multiplexers and a three-input adder/subtractor/accumulator. The DSP48E1 multiplier can apply multiplication with a 18-bit and a 25-bit two's complement number 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. Among the operators of the DSP48E1, since the pipeline registers are inserted, 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 $2k \times 18$-bit dual-port RAM.

C. Related Work

To accelerate the RSA encryption/decryption, several research used a GPU (Graphics Processing Units) support [4], [5]. However, the iteration of exponentiation in modular exponentiation is not suitable for GPU. Therefore, the GPU cannot compute effciently.

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 [6]. Our work carries forward with Großschädl's 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 [7]. In [8], 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 [8] on FPGAs [9]. Blum and Paar proposed a modular exponentiation hardware algorithm with a radix-2 Montgomery multiplication using systolic array [10]. Also,

a radix-2^4 modular exponentiation circuit that is an extended method of the radix-2 circuit is proposed [11]. The circuits of the 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 [12]. 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 [13]. 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 [14]. 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^{17} [15]. 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^{18} [16].

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^{17}. The radix-2^{17} is decided by the feature of embedded DSP blocks in our target device.

In our previous work [17], we have presented an RSA encryption hardware using one DSP and one Block RAM. The implementation result shows that this hardware performs 1024-bit RSA encryption in 36.37ms in XC6VLX240T-1.

The main contribution of this paper is to use CRT technique to further accelerate our previous work [17]. We have also implemented 320 cores in XC6VLX240T-1 for parallel computation.

II. 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 [7] 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.

A. Montgomery Multiplication Algorithm

Montgomery multiplication algorithm [7], 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 -

$0 \le E \le 2^{|E|} - 1$, $E = \sum_{i=0}^{|E|-1} 2^i \cdot E_i, E_i \in \{0,1\}$
Input: $P, E, M, -M^{-1}, 2^{2dr} \bmod M$
Output: $C = P^E \bmod M$
1. $C \leftarrow (2^{2dr} \bmod M) \cdot 1 \cdot 2^{-dr} \bmod M$;
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.

III. 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:

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

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 $d_i = {N_i}^{-1} \pmod{n_i}$ $(1 \le i \le k)$. We have,

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

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:

$$P_p = C^D \bmod p = C_p^{D_p} \bmod p \quad (4)$$

$$P_q = C^D \bmod q = C_q^{D_q} \bmod q, \quad (5)$$

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:

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

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 $P \ge M$ 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.

IV. Implementation on the FPGA

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 CRT based RSA decryption circuit with it.

A. 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 II-A. 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: $S_d = X \cdot Y \cdot 2^{-17d} \mod M$

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

1) Architecture of Montgomery Multiplier: Figure 3 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

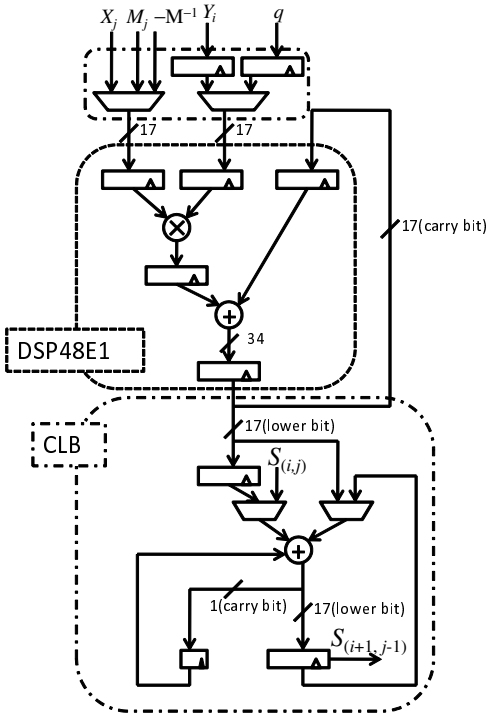

Figure 3. Structure of our Montgomery multiplier

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 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 3.

2) 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 [8], 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

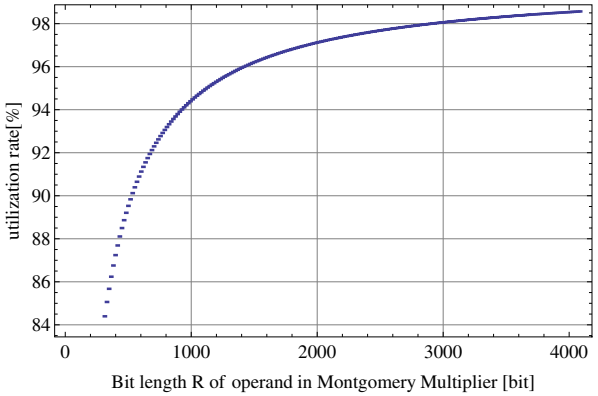

Figure 4. Embedded multiplier utilization rate of our Montgomery multiplier(MM_{mul}/MM_{clk})

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 4 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.

B. Our CRT based RSA decryption circuit

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

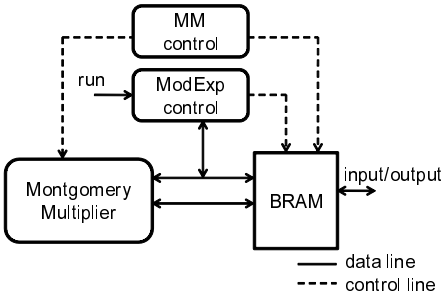

Figure 5. Structure of our CRT based RSA decryption circuit

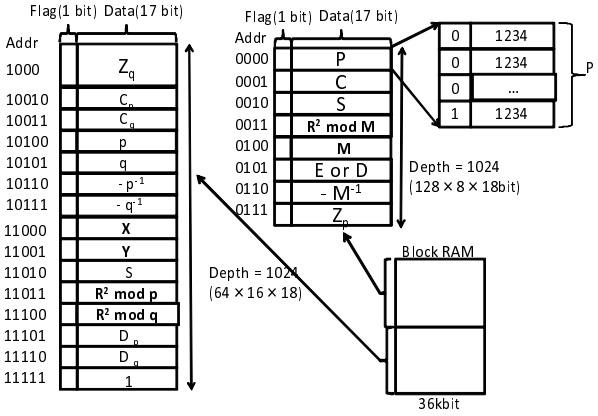

Figure 6. Internal configuration of BRAM in our CRT based RSA decryption circuit

512 bits. We first input C and ${R_p}^2 \bmod p$ into Montgomery multiplier introduced in Section IV-A1 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 3, 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 block 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 block and Block RAM. The block diagram of our CRT based decryption circuit is shown in Figure 5.

The size of a single Block RAM in Virtex-6 is 36 Kbits. In order to reduce the hardware resource cost, only one Block RAM is used in our circuit. The 36 Kbits Block RAM is split to 2 sub-blocks as shown in Figure 6. 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 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.

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,

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

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:

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

3) Our parallel processing system: A parallel system is implemented so that the throughput of our circuit can be improved a lot.

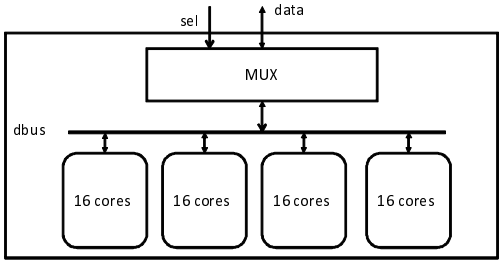

Figure 7. The top module of our parallel system

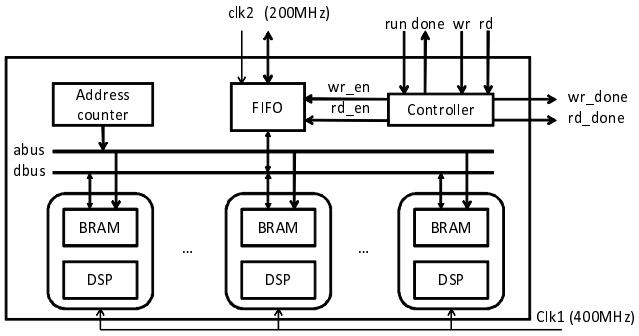

Figure 8. The parallel sub-processor

Our parallel system is built up by several sub-processors each containing 16 CRT based RSA cores. Figure 7 shows the top module of our parallel system. There is one data input/output port with a selection port to select the datapath. Input data follows the value of selection port going to the target sub-processor. A multiplexer controls the selection of the datapath according to the value of selection port. A data bus is shared by all the sub-processors that only one sub-processors can write or read data at one time.

Figure 8 shows the structure of our sub-processors. Every sub-processor is embedded 16 CRT based RSA decryption cores with an data exchanging interface. The interface is composed by complicated multiplexer network, thus it works much slower than RSA cores. Since the data exchanging is occurred only twice before processing and after processing, and it will not cause undesirable effect on data processing, we separate the clock resources. The slower one is for the interface and the higher one is for the CRT core. An FIFO is used between interface and RSA core in order to packing the timing gap between two clocks. Data is written or read into each core in serial that means all the cores have to wait until all the core receiving the data. If there is no data, 0 is padded into the Block RAM.

In our implementation, we made maximum use of the FPGA which around 300 cores is embedded in such parallel system.

V. 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 11.5 mapping tool.

Table I shows the synthesized results in Virtex-6 and lists the resource costs for single core and multiple cores, where the SRs and SLUTs stand for how many configurable logic blocks including slice registers and slice LUTs are occupied. BRAMs represents the number of used Block RAMs. DSP48E1s shows how many DSP blocks are necessary. The maximum frequency means how fast our circuit could be executed. For a single core, there is just a global clock resource. While for the multiple cores, two clocks are necessary which the slower one is connected with data interface and the other faster one is for the data processing. The numbers below each item shows the maximum resource of our target device. From Table I, it shows that the scale of our circuit for a single core is so small that only 201 slices registers, 374 slices LUTs, 1 BRAM and 1 DSP48E1 are necessary. Thus, we can implement a multiple cores system easily. Also, since the maximum clock frequency of DSP48E1 is 600MHz, an extremely high frequency can be obtained by our algorithm.

Table II shows the necessary clock cycles, execution time and maximum throughput in the worst case from 64-bit to 2048-bit CRT based RSA decryption. The execution time is computed by the production from necessary clock cycles with circuit delay. The data is taken from Equation 10 and Table II. Any size of operands less than 2176-bit can be executed in the same circuit without any modification. Our CRT based RSA decryption circuit processes 1024 bits RSA decryption in 11.263 ms. It achieves nearly 3.5 times speedup comparing with our previous work. Since our previous RSA circuit does not apply CRT, the architecture is less complicated, the maximum frequency is higher and up to 447MHz. Our circuit works on 410MHz, thus the speedup is less than 4 times. If our circuit works on the same frequency, in theorem, we can achieve at most 4 times speedup by CRT based algorithm. Also, since the architecture of our single core so compact that we can

Table I
SYNTHESIS RESULT OF OUR CRT BASED RSA

Cores Subs/Cores	SRs -/301440	SLUTs -/150720	BRAMs -/416	DSP48E1s -/768	Maximum frequency[MHz] clk1/clk2
-/1	201	374	1	1	410.678(2.435ns)
20/320	68038	133164	340	320	237.530(4.210ns)/379.203(2.673ns)
16/272	54614	106434	288	272	242.424(4.125ns)/425.512(2.350ns)

Table II
WORST-CASE EXECUTION TIME AND THROUPUT OF OUR CRT BASED RSA DECRYPTION CIRCUIT

bit length R	64	128	256	512	1024	2048
blocks d_1/d_2	4/2	8/4	16/8	31/16	61/31	121/61
frequency[MHz]	410.678	410.678	410.678	410.678	410.678	410.678
clock cycles	4312	19768	110392	713048	4625348	33067148
execution time(CRT based decryption)[ms]	0.011	0.048	0.269	1.736	11.263	80.519
execution time(direct decryption in 447MHz)[ms]	0.02	0.12	0.74	4.99	36.37	277.26
throughput(single core)[kbit/s]	5952.54	2596.86	930.044	287.973	88.786	24.936
throughput(320 cores)[kbit/s]	1.7589×10^6	767339	274816	85093	26236	7368
throughput(272 cores)[kbit/s]	1.4951×10^6	652238	233594	72329	22301	6263

Table III
COMPARISON WITH PREVIOUS 1024-BIT MODULAR EXPONENTIATOR ALGORITHMS

	Blum [11]	Nakano [13]	Suzuki [15]
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× 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]	83.682	396.825	584.795
scalable	no	no	yes

	Mazzeo [14]	Alho [16]	this work
device	Xilinx Virtex-E2000-8	Altera Stratix EP1S40	Xilinx XC6VLX240T-1
logic block	1188 Slices	341 LEs	201 Slices Registers and 374 Slice LUTs
memory block	-	13604-bit	1 BRAM
DSP block	-	1 DSP	1 DSP48E1
frequency[MHz]	86.2	198	410.678
execution time[ms]	3.86($E = 2^{17} + 1$)	28(average case)	11.263(worst case)
throughput[kbit/s]	295.067	35.714	88.786
scalable	no	yes	yes

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 26236kbit/s.

There are a number of literatures reported to implement RSA using FPGA as described in Section I-C. Performances such as device, circuit size, frequency, execution time, throughput and scalability of 1024-bit modular exponentiation circuit are compared in Table III. Note that direct RSA decryption sharing the same modular exponentiation circuit with encryption. It makes sense that we do such comparison. 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 half of 1024-bit of E is 1. Blum *et al.* [11] 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. Nakano *et al.* [13] 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 [15] 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 with scalability and a high throughput that nearly 585kbit/s is achieved. In paper [14], 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 III 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.* [16] 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 numbers of clock cycles are just half comparing with our algorithm.

Since DSP48E1s and Block RAMs are efficiently used in our circuit, the size of our modular exponentiation circuit is very small. Also, the DSP48E1 works almost all the clock cycles shown in Section IV-A2. 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.

VI. Conclusion

In this paper, we have proposed a hardware algorithm for CRT based RSA decryption 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 and throughput.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] Xilinx Inc., "Virtex 6 ML605 Hardware User Guide(v1.2.1)," 2010.

[3] ——, "Virtex-6 FPGA DSP48E1 Slice User Guide(v1.2)," 2009.

[4] W. Fan, X. Chen, and X. Li, "Parallelization of RSA algorithm based on compute unified device architecture," in *Proc. of The Ninth International Conference on Grid and Cloud Computing*, 2010, pp. 174–178.

[5] O. Harrison and J. Waldron, "Public key cryptography on modern graphics hardware," in *Booklet of posters, Eurocrypt 2009*, April 2009.

[6] J. 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*, 2000, pp. 384–393.

[7] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[8] Çetin Kaya Koç, T. Acar, and B. S. Kaliski, Jr., "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, 1996.

[9] C. McIvor, M. McLoone, and J. V. McCanny, "FPGA Montgomery multiplier architectures - a comparison," in *Proc. of Field-Programmable Custom Computing Machines*, 2004, pp. 279–282.

[10] 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.

[11] ——, "High-radix Montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. on Computers*, vol. 50, no. 7, pp. 759–764, 2001.

[12] 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.

[13] K. Nakano, K. Kawakami, and K. 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*, May 2009, pp. 1–8.

[14] 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.

[15] 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.

[16] 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.

[17] B. Song, K. Kawakami, K. Nakano, and Y. Ito, "An RSA encryption hardware algorithm using a single DSP block and a single block RAM on the fpga," in *Proc. of International Conference on Networking and Computing*, Nov 2010, pp. 140–147.