

RSA Cryptosystem Design Based on the Chinese Remainder Theorem

Chung-Hsien Wu, Jin-Hua Hong, and Cheng-Wen Wu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013
ROC

Abstract—In this paper, we present the design and implementation of a systolic RSA cryptosystem based on a modified Montgomery’s algorithm and the Chinese Remainder Theorem (CRT) technique. The CRT technique improves the throughput rate up to 4 times in the best case. The processing unit of the systolic array has 100% utilization because of the proposed *block interleaving* technique for multiplication and square operations in the modular exponentiation algorithm. For 512-bit inputs, the number of clock cycles needed for a modular exponentiation is about 0.13M to 0.24M. The critical path delay is 6.13ns using a 0.6 μ m CMOS technology. With a 150 MHz clock, we can achieve an encryption/decryption rate of about 328 to 578 Kb/s.

I. INTRODUCTION

The electronic communication technology has advanced in a very fast pace during the past few decades, creating new applications and opportunities along the way. Today we can send a multimedia message to or receive one from virtually anyone around the world in seconds through the internet. To protect the transmitted data from eavesdropping by someone else other than the desired receiver, we need to disguise the message before sending it into the insecure communication channel. This is achieved by a cryptosystem. In 1978, Rivest, Shamir, and Adleman invented a method to implement the public-key cryptosystem, which is known as the RSA cryptosystem [1]. It provides high security and is easy to implement, so it quickly became the most widely used public-key cryptosystem. However, developing an inexpensive hardware device for real-time RSA encryption and decryption is still a challenge. Finding an efficient hardware implementation for RSA is one of the important tasks remain to be done.

In the RSA cryptosystem, both encryption and decryption are modular exponentiation, which can be done by a sequence of modular multiplications. Many modular multipliers have been proposed in the past [2–4]. One of the most important algorithms was proposed by Montgomery [5]. Montgomery’s algorithm needs n iterations in each modular multiplication and two additions per iteration, where n is the word length. Cellular arrays based on Montgomery’s algorithm can be found

in [6–8]. In this algorithm, odd modulus is assumed. It calculates the modular multiplication without performing division. Some improved algorithms were later proposed, either by reducing the number of iterations or by removing the final modular reduction step. A modified Montgomery’s algorithm was reported in [9], where the multiplication and modular reduction steps in Montgomery’s algorithm are separated such that only one addition is required in each iteration. However, the number of iterations in the modified algorithm is two times that of Montgomery’s, hence the overall computation time is not reduced. In [10, 11], the algorithm was further modified to reduce the number of iterations, doubling the speed of modular multiplication. Another way to reduce the computation time is to use the Chinese Remainder Theorem (CRT) technique, since CRT is known to reduce the RSA computation by a divide-and-conquer method.

In this paper, we present the design and implementation of a systolic RSA cryptosystem based on a modified Montgomery’s algorithm and the CRT technique. The systolic array has a 100% utilization ratio because of a novel interleaving approach. The throughput of the proposed design is much higher than previous ones. For 512-bit inputs, e.g., the number of clock cycles needed for a modular exponentiation is about 0.13M to 0.24M. The layout and post-layout simulation of the 512-bit RSA chip has been finished, and the critical path delay is 6.13ns using a 0.6 μ m CMOS technology. Under a 150MHz clock, the decryption rate is about 328 to 578 Kb/s.

II. MODULAR EXPONENTIATION ALGORITHM

To encrypt a message using the encryption key (E, N) , we first partition the message into a sequence of blocks and consider each block M as an integer between 0 and $N - 1$. Then, we encrypt the message by raising M to the E th power modulo N , i.e., $C = M^E \pmod{N}$. Similarly, to decrypt the ciphertext C using the decryption key (D, N) , we raise C to the power of D modulo N , i.e., $M = C^D \pmod{N}$. Clearly, modular exponentiation is the main operation of the RSA algorithm. For modular exponentiation, a sequence of modular multiplications can be performed instead. If we transform the expo-

nent E into its binary representation $(e_{k-1}e_{k-2}\dots e_1e_0)_2$, then $M^E = M^{2^{k-1}e_{k-1}+\dots+2^2e_2+2e_1+e_0} = M^{2^{k-1}e_{k-1}} \dots M^{2^2e_2} \cdot M^{2e_1} \cdot M^{e_0}$. Clearly, this can be done by a sequence of squaring and multiplication operations, by scanning the bits of E either from high-order to low-order bits or in the opposite direction. These two methods are known as the *H-Algorithm* and *L-Algorithm*, respectively. Both of them require n iterations for an n -bit exponent. Each iteration contains two modular multiplications.

A. Modular Multiplication

Our approach is based on the foundation established in [9–11]. Suppose $N = (n_{n-1}, \dots, n_1, n_0)$ is an n -bit odd integer. Let A and B be two other n -bit integers, where $0 \leq A, B < N$. We extend A and B to $(n+1)$ bits by appending a leading zero to the MSB, so $0 < A, B < 2^{n+1}$. Let $X = A \times B$, where $X = (x_{2n+1}, \dots, x_1, x_0)$ is a $(2n+2)$ -bit integer. We can represent it as $X = \sum_{i=0}^{2n+1} x_i 2^i = 2^{n+2} X_M + X_L$, where $X_M = \sum_{i=0}^{n-1} x_{i+n+2} 2^i$ and $X_L = \sum_{i=0}^{n+1} x_i 2^i$. The modular multiplication algorithm is as follows.

```
MM(A, B, N)
{
  A × B = X = X_M · 2^{n+2} + X_L;
  S[0] = 0;
  for (i = 0; i < n + 2; i++) {
    q_i = (S_i + x_i) (mod 2);
    S_{i+1} = (S_i + x_i + q_i N) / 2;
  }
  R = S_{n+2} + X_M;
  return R;
}
```

The advantage of Procedure MM() is that the partial products are always in the range $(0, 2^{n+1})$, which is the same with the input range. Hence, the modular reduction is removed. Based on MM() and the L-algorithm, we obtain the modular exponentiation algorithm MLA() as shown below. Note that by letting $R = 2^{n+2}$ and $C = R^2 \pmod{N}$, the final result P_{k+1} will be equal to $M^E \pmod{N}$, i.e., we let $M_0 = M \times 2^{n+2} \pmod{N}$ and $P_0 = 2^{n+2}$ initially to guarantee that the final result is $M^E \pmod{N}$.

```
MLA(M, E, N, C)
{
  M_0 = MM(C, M, N);           //Initialization
  P_0 = MM(C, 1, N);
  for (i = 0; i < k; i++) {
    M_{i+1} = MM(M_i, M_i, N)   //Square
    if (e_i = 1)
      P_{i+1} = MM(M_i, P_i, N); //Multiplication
    else
      P_{i+1} = P_i;
  }
  P_{k+1} = MM(1, P_k, N);     //Post-process
  return P_{k+1};
}
```

}

B. RSA Computation Using CRT

The Chinese Remainder Theorem (CRT) can be stated as follows.

Theorem 1 (Chinese Remainder Theorem)

Let m_0, m_2, \dots, m_{n-1} be pairwise relatively prime positive integers and let x_0, x_1, \dots, x_{n-1} be any integers which satisfy the linear congruence system in one variable given by

$$\begin{cases} X \equiv x_0 \pmod{m_0} \\ X \equiv x_1 \pmod{m_1} \\ \vdots \\ X \equiv x_{n-1} \pmod{m_{n-1}} \end{cases}$$

has a unique solution modulo $m_0 \cdot m_1 \cdot \dots \cdot m_{n-1}$.

The RSA decryption and signature operation can be speeded up by using the CRT, where the factors of the modulus N (i.e., P and Q) are assumed to be known. By CRT, the computation of $M = C^D \pmod{N}$ can be partitioned into two parts:

$$\begin{aligned} M_P &= C_P^{D_P} \pmod{P}, \\ M_Q &= C_Q^{D_Q} \pmod{Q}, \end{aligned} \quad (1)$$

where

$$\begin{aligned} C_P &= C \pmod{P}, \quad D_P = D \pmod{P-1}, \\ C_Q &= C \pmod{Q}, \quad D_Q = D \pmod{Q-1}. \end{aligned}$$

This reduces computation time since $D_P, D_Q < D$ and $C_P, C_Q < C$. In fact, their sizes are about half the original sizes. In the ideal case, we can have a speedup of about 4 times. Finally, we compute M by CRT as follows:

$$\begin{aligned} M &= [M_P(Q^{-1} \pmod{P})Q \\ &\quad + M_Q(P^{-1} \pmod{Q})P] \pmod{N}. \end{aligned} \quad (2)$$

III. HARDWARE IMPLEMENTATION

Our design consists of a modular multiplier and some logic to control the data interleaving and scheduling mechanism. We follow the standard systolic array design flow [12] to obtain the systolic RSA cryptosystem according to the MM() and MLA() algorithms. After that, we add some partitioning circuit for our systolic RSA architecture to get two individual modular multipliers which can be used for $M_P = C_P^{D_P} \pmod{P}$ and $M_Q = C_Q^{D_Q} \pmod{Q}$ simultaneously. The details follow.

A. Multiplier

The dependence graph (DG), the hyperplanes, and the schedule vector [12] of the multiplier are shown in Fig. 1. The

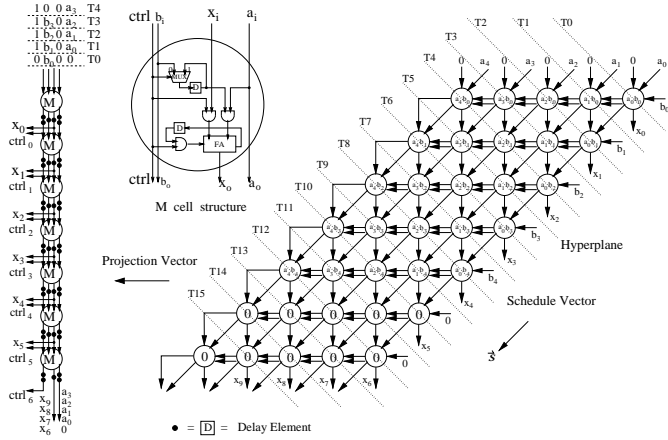


Fig. 1. The DG and SFG of a 4-bit multiplier.

utilization rate of this multiplier is only 50%. However, this can be doubled by a novel block interleaving technique [13], processing two sets of data in the multiplier during each iteration cycle, i.e., $MM(M_i, M_i, N)$ and $MM(M_i, P_i, N)$.

B. Modular Reduction Unit

In the modular reduction operation, the value q_i is generated by XORing the LSB of S_i and x_i . We use an n -bit addition and right shifts to accomplish the modular operation. Fig. 2 shows the DG, hyperplanes, schedule vector, cell structures, and SFG of the modular reduction unit [13]. Note that the Z cell is the combination of X and Y cells, so we need extra control logic to realize the cell function. This circuit can also be interleaved by two sets of data in a similar way as the multiplier discussed above. We combine the systolic array multiplier and modular reduction unit to form a modular multiplier, which is shown in Fig. 3(a).

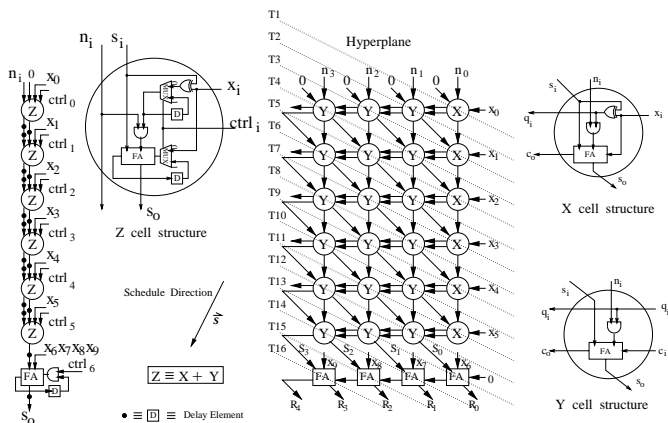


Fig. 2. The DG and SFG of a 4-bit modular reduction unit.

C. Composite Modular Multiplier

By the CRT technique, we partition the modular multiplier into two smaller ones according to the lengths of P and Q . Then, we compute $M_P = C_P^{D_P} \pmod{P}$ and $M_Q = C_Q^{D_Q} \pmod{Q}$ using two independent modular multipliers concurrently. This apparently reduces the computation time. Note that the partition is done on-line according to the lengths of P and Q , so each cell in the array must be able to read the data from the primary inputs directly, and be able to propagate the data to the primary outputs directly to finish the final step of the modular multiplication. Therefore, we use some switches, MUXes, and a control signal to achieve the goals. The multiplier is called the composite modular multiplier (CMM), and is shown in Fig. 3(b).

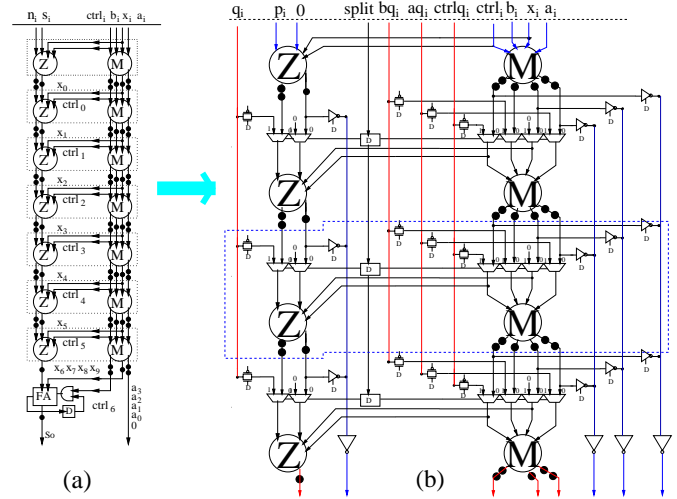


Fig. 3. (a) Systolic array modular multiplier. (b) The composite modular multiplier.

The signal *split*, which is shifted into the register formed by chaining the D-FFs in all blocks, indicates the partition location. The partition circuit is shown in Fig. 4(a). We extend the prime number Q to an n -bit integer with leading zeros and shift it into the n -bit shift register Q_Reg initially, where n is the length of the modulus N , i.e., $n = \lceil \log_2 N \rceil$. The clock signal CLK is disabled until the MSB of Q_Reg is high. The two shift registers, Q_Reg and $Split_Reg$, perform the left-shift function. Finally, the information in the $Split_Reg$ indicates the partition location. Note that there is only a single bit that is high in the $Split_Reg$. Thus, we have two smaller modular multipliers that can operate concurrently to perform the original modular multiplication. It reduces the execution time for the RSA decryption and signature operations. In a typical RSA cryptosystem, once the keys are generated, they are fixed until the new keys are generated, i.e., P and Q are fixed after key generation. Therefore, the partition of N into P and Q needs to be done only once, before the RSA computation starts.

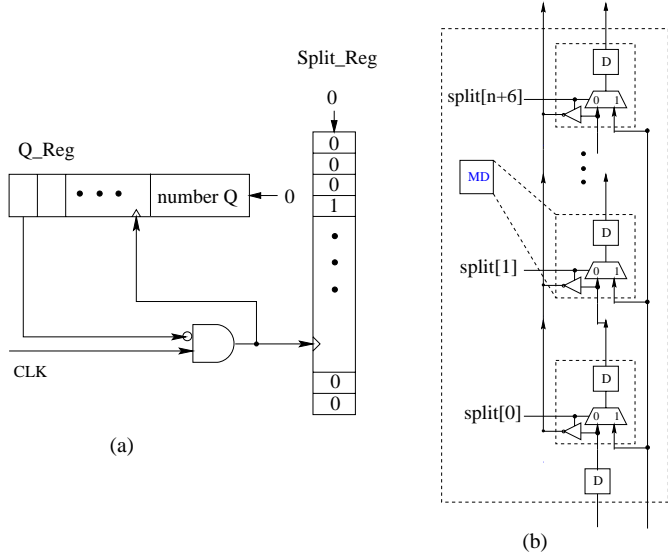


Fig. 4. The partition circuit and the modified register CON_REG.

D. CRT-Based RSA Cryptosystem

We have presented the systolic-array modular multiplier based on the CRT technique. We can add some control logic to accomplish the modular exponentiation operation. The circuit for the entire RSA computation that incorporates the CRT technique is shown in Fig. 5. The details of the CON_REG block is shown in Fig. 4(b), where the partition scheme is the same as CMM. The control signals are supposed to come from the host controller, so they are considered as primary inputs in the core circuit design. The four inputs $in1p$, $in2p$, $in1q$, and $in2q$ are used only for initialization and post-process steps. The control logic is responsible for timing management, data interleaving, iteration handling, etc.

As depicted in Fig. 6, the time complexity of the CRT-based RSA computation is highly dependent on the lengths of P and Q . The best case occurs when $|P| = |Q| = |N|/2$. Suppose N is an n -bit integer, and the lengths of P and Q are p bits and $(n - p + 1)$ bits, respectively. The numbers of clock cycles required to finish a modular exponentiation using the original and the CRT-based approaches are listed, respectively, in Table I.

TABLE I
COMPLEXITY COMPARISON FOR MODULAR EXPONENTIATION.

	Original Modular Exponentiation	CRT-Based Modular Exponentiation	
Operation	$M^E \pmod{N}$	$M_P^{E_P} \pmod{P}$	$M_Q^{E_Q} \pmod{Q}$
Length	N : n -bit	P : p -bit	Q : $(n - p + 1)$ -bit
Clock cycles	$(2n + 8)(n + 2)$	$(2p + 8)(p + 2)$	$(2n - 2p + 10)(n - p + 3)$
Initial setup	Unnecessary	$n + p$	

From the table, if P and Q have equal length, i.e., $n/2$, the number of clock cycles required will be the smallest. Although

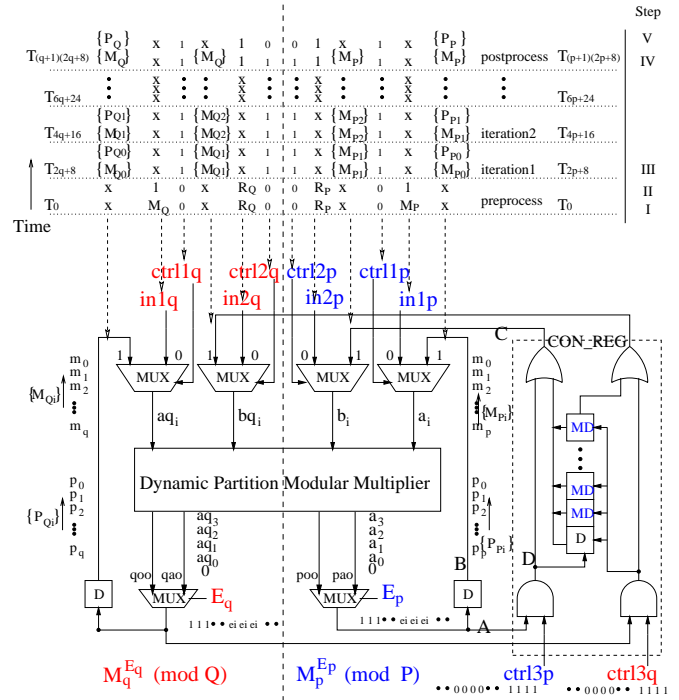


Fig. 5. The control circuit for modular exponentiation.

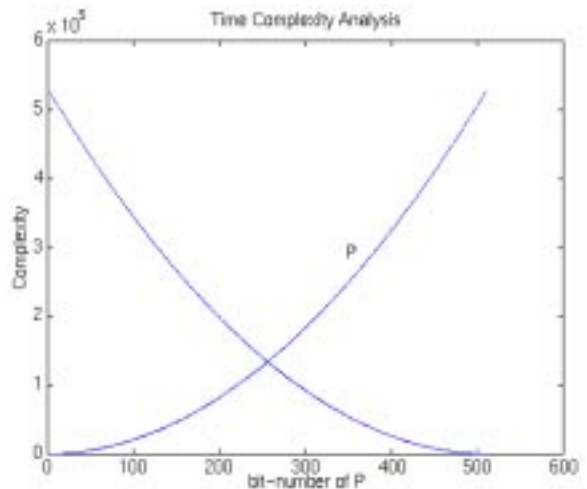


Fig. 6. Time complexity of the RSA circuit.

this is not likely to occur in practice (since P and Q are prime and $P \neq Q$), their lengths should be as close as possible to ensure security. In the worst case, it is assumed that the maximum length of P or Q is $2n/3$ (and the minimum is $n/3$).

The circuit has been implemented with a $0.6\mu\text{m}$ CMOS standard-cell library. The post-layout simulation result shows that the critical path delay is about 6.13 ns, i.e., we expect the core to operate at a 150MHz clock rate. For a 512-bit RSA cryptosystem, e.g., the best case needs 0.13M clock cycles, while the worst case requires 0.24M cycles to finish an RSA operation. Under a 150MHz clock, the baud rate is 578K in the best case and 328K in the worst case. Fig. 7 shows the layout of the 512-bit CRT-based RSA cryptosystem. The core area is about $7653\mu\text{m} \times 7486\mu\text{m}$ with about 109K gate count.

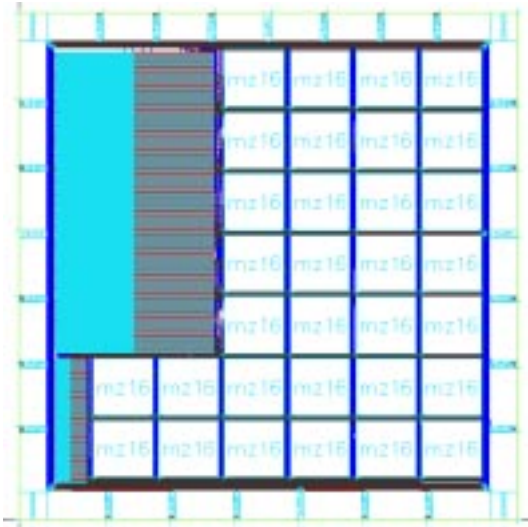


Fig. 7. Layout of the 512-bit CRT-based RSA cryptosystem.

TABLE II
PERFORMANCE AND AREA COMPARISON.

	Year	Gate	# of Clk	Tech.	Clk (Hz)	Baud Rate
[14]	1994	75K	0.125M	1	25M	100K
[11]	1996	77K	1.05M	0.8	50M	24.3K
[10]	1998	74K	0.39M 0.54M	0.6	125M	164k 118k
[13]	1999	77K	0.53M	0.6	250M	241K
Ours	2000	109K	0.13M 0.24M	0.6	150M	578k 328K

IV. CONCLUSIONS

In this paper, we have presented a systolic RSA cryptosystem design, based on an improved Montgomery's modular multiplication algorithm [13] and the CRT technique. The

design is suitable for decryption and digital signature. The implementation of a 512-bit RSA cryptosystem with a $0.6\mu\text{m}$ CMOS standard-cell library has been done. The comparison of our implementation with other designs is summarized in Table II, which shows clear advantage of our approach in terms of speed. The number of clock cycles is 0.13M in the best case (when $|P| = |Q| = n/2$) and 0.24M in the worst case (when $||P| - |Q|| = n/3$). Therefore, the throughput of our design is the highest among all the designs, but with about 50% higher hardware cost. The circuit is easily extendible for large moduli due to the systolic-array design.

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, Feb. 1978.
- [2] Ç. K. Koç and C. Y. Hung, "Bit-level systolic arrays for modular multiplication", *J. VLSI Signal Processing*, vol. 3, pp. 215–223, 1991.
- [3] Ç. K. Koç, "RSA hardware implementation", Technical report, RSA Laboratories, RSA Data Security, Inc., Redwood City, CA, 1995.
- [4] N. Takagi and S. Yajima, "Modular multiplication hardware algorithms with a redundant representation and their application to rsa cryptosystem", *IEEE Transactions on Computers*, vol. 40, pp. 887–891, July 1992.
- [5] P. L. Montgomery, "Modular multiplication without trial division", *Math. Computation*, vol. 44, pp. 519–521, 1985.
- [6] P. Kornerup, "A systolic, linear-array multiplier for a class of right-shift algorithms", *IEEE Transactions on Computers*, vol. 43, no. 8, pp. 892–898, Aug. 1994.
- [7] C. D. Walter, "Systolic modular multiplication", *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 376–378, Mar. 1993.
- [8] M. Shand and J. Vuillemin, "Fast implementation of RSA cryptography", in *Proc. 11th IEEE Symp. Computer Arithmetic*, Windsor, Ontario, June 1993, pp. 252–259.
- [9] P.-S. Chen, S.-A. Hwang, and C.-W. Wu, "A systolic RSA public key cryptosystem", in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, Atlanta, May 1996, pp. 408–411.
- [10] C.-C. Yang, T.-S. Chang, and C.-W. Jen, "A new RSA cryptosystem hardware design based on Montgomery's algorithm", *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 7, pp. 908–913, July 1998.
- [11] C.-Y. Su, S.-A. Hwang, P.-S. Chen, and C.-W. Wu, "An improved Montgomery algorithm for high-speed RSA public-key cryptosystem", *IEEE Trans. VLSI Systems*, vol. 7, no. 2, pp. 280–284, June 1999.
- [12] S.-Y. Kung, *VLSI Array Processors*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1988.
- [13] J.-H. Hong, P.-Y. Tsai, and C.-W. Wu, "Interleaving schemes for a systolic RSA public-key cryptosystem based on an improved Montgomery's algorithm", in *Proc. 11th VLSI Design/CAD Symp.*, Pingtung, Aug. 2000, pp. 163–166.
- [14] H. Orup, "A 100Kbits/s single chip modular exponentiation", in *Proc. HOT Chips VI*, 1994, pp. 53–59.