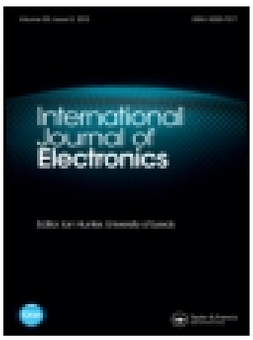

International Journal of Electronics

ISSN: 0020-7217 (Print) 1362-3060 (Online) Journal homepage: http://www.tandfonline.com/loi/tetn20

VLSI Design of a RSA Encryption/Decryption Chip using Systolic Array based Architecture

Chi-Chia Sun, Bor-Shing Lin, Gene Eu Jan & Jheng-Yi Lin

To cite this article: Chi-Chia Sun, Bor-Shing Lin, Gene Eu Jan & Jheng-Yi Lin (2016): VLSI Design of a RSA Encryption/Decryption Chip using Systolic Array based Architecture, International Journal of Electronics, DOI: 10.1080/00207217.2016.1138511

To link to this article: http://dx.doi.org/10.1080/00207217.2016.1138511

Accepted author version posted online: 08 Feb 2016.

To appear in the *International Journal of Electronics*
Vol. 00, No. 00, January 2013, 1–13

Publisher: Taylor & Francis
Journal: International Journal of Electronics DOI:
10.1080/00207217.2016.1138511

VLSI Design of a RSA Encryption/Decryption Chip using Systolic Array based Architecture

Chi-Chia Sun*, Bor-Shing Lin, Gene Eu Jan and Jheng-Yi Lin

(*v4.0 released January 2013*)

This paper presents the VLSI design of a configurable RSA public key cryptosystem supporting the 512-bit, 1024-bit and 2048-bit based on Montgomery algorithm achieving comparable clock cycles of current relevant works but with smaller die size. We use binary method for the modular exponentiation and adopt Montgomery algorithm for the modular multiplication to simplify computational complexity, together with systolic array concept for electric circuit designs effectively lower the die size. The main architecture of the chip consists of four functional blocks, namely input/output modules, registers module, arithmetic module and control module. We applied the concept of systolic array to design the RSA encryption/decryption chip by using VHDL hardware language and verified by the TSMC/CIC 0.35 m 1P4M technology. The die area of the 2048-bit RSA chip without the DFT is 3.9×3.9 mm^2 (4.58×4.58 mm^2 with DFT). Its average baud rate can reach 10.84 Kbps under a 100 MHz clock.

Keywords: VLSI; Cryptology; RSA; 2048-bit; Systolic Array;

1. Introduction

Cryptosystems, as according to the practice of key distribution, can be classified as private key cryptosystems and public key cryptosystems. Rivest, Shamir and Adleman published the RSA public key cryptosystem in 1978 (Rivest, Shamir, & Adleman, 1978). There, each user possesses a pair of keys, the public key and the private key. The public key as the name implied is open to the public while the private key needs to be stored secretly. The encryption process in the RSA public key cryptosystem can be expressed as, $M^E \pmod N$, where M is the plain text, E is the public key and N is the modulus, while the decryption process can be expressed as, $C^D \pmod N$, where C is the cipher text, D is the private key. This type of operation is also called modular exponentiation, which can be decomposed into a series of modular multiplications to save the computation time. In 2010, the 1024-bit RSA has been reported that it is no longer safe to protect the secure data. With a small cluster of 81 Pentium–4 chips and 104 hours of processing time, it is able to hack 1024-bit encryption in OpenSSL on a SPARC-based system (Pellegrini, Bertacco, & Austin, 2010). This gives a strong reason why 2048-bit or even larger RSA cryptosystems are important.

Montgomery algorithm as published by L. P. Montgomery (Montgomery, 1985) has been used constantly in the modular multiplication electric circuit designs.

*Corresponding author. Department of Electrical Engineering, National Formosa University, Taiwan, Email:ccsun@nfu.edu.tw

Using Montgomery algorithm for modular multiplication does not need comparison operations and is suitable for the odd modulus, which just meets the requirements of the RSA public key cryptosystem. In the literature, Montgomery algorithm can save the computation time of modular multiplication (Eldridge & Walter, 1993; Walter, 1993, 1995), but the results fall into the range [0, $2N$), which is far too large than the correct range of [0, N). To obtain the right result, many researches proposed improved methods (Chen, Hwang, & Wu, 1996; Yang, Jen, & Chang, 1996), however, they either increase the space complexity of the hardware or the time complexity of computation. This paper proposes a simple solution based on the Montgomery algorithm with modified architecture (Walter, 1993, 1999), designs and implements a 2048-bit RSA public key cryptosystem (about 617 decimal digits), to achieve comparable clock cycles of relevant works but with smaller die size with near $O(N)$ hardware complexity. Experimental results show that the RSA core is implemented in a small area by using the systolic array and verified by Time Mill postlayout simulation with the TSMC/CIC 0.35 μm 1P4M technology, where the average baud rate can reach 10.84 Kbps under a 100 MHz clock in a 3.9×3.9 mm^2 (2048-bit RSA chip without the DFT). It is worth noting that the presented $O(N)$ hardware complexity 2048-bit RSA core is ideal for being extended to secure the important data in portable devices, such as RSA SecurID Software Token or Bitcoin digital currency. It provides the comparable computing time among the other approaches but requires smaller chip area.

This paper is organized as follows. Section 2 briefly introduces the concept of modular exponentiation algorithms in the RSA public key cryptosystem. In Section 3, we will present the proposed RSA encryption/decryption chip architecture and analysis the experimental results in Section 4, while Section 5 concludes this paper.

2. Algorithms

2.1 *Modular Exponentiation Operations*

Modular exponentiation operations are the core operation for the public key cryptosystem. Using binary method (Singh & Datta, 1935), modular exponentiation can be split into a series of modular multiplications (Knuth, 1969). The binary method algorithm is shown below as Algorithm 1 .

Algorithm 1: H(M, E, N) Binary Method Algorithm

Input: Modulus: N (n-bit) Exponent: $E = (1\ e_{k-2}\ e_{k-3}\ \ldots\ e_1\ e_0)_2$
Message: M (n-bit)
Output: $R[k-1]$=$M^E \pmod N$
$R[0] = M$;
for $i = 0; i < k-1; i++$ **do**
  $R[i+1] = R[i] \times R[i] \pmod N$;
  if $(e_{k-i-2} == 1)$ **then**
    $R[i+1] = R[i+1] \times M \pmod N$;
  else
    $R[i+1] = R[i+1]$;
return $R[k-1]$;

2.2 *Modular Multiplication Operations*

Modular multiplication can be expressed as, $A \times B(\text{mod } N)$. Among algorithms commonly used for modular multiplication, Montgomery algorithm is the most comparatively suitable for hardware implementation (Montgomery, 1985). Its multiplication and division operations in each iteration need only add, xor and shift operations and without extra comparison operations. The Algorithm 2 is listed as follows.

Algorithm 2: MONT(A, B, N) Montgomery Algorithm

Input: Modulus: N(n-bit), gcd(N, 2) = 1
Multiplier: $A = (a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0)_2$
Multiplicand: $B = (b_{n-1}\ b_{n-2}\ \ldots\ b_1\ b_0)_2$
Output: $R[n] \equiv A \times B \times 2^{-n}(\text{mod } N)$
$R[0] = 0$;
for $i = 0; i < n; i++$ **do**
  $q_i = R[i] + a_i \times B(\text{mod } 2)$;
  $R[i+1] = (R[i] + a_i \times B + q_i \times N)/2$;
return $R[n]$;

There, q_i is the so-called quotient digit; its purpose is to make the least significant bit (LSB) of the result in calculating $R[i] + a_i \times B + q_i \times N$ to be zero, so that no data will be lost in the division by 2 (right shift one bit operation). q_i is determined by the xor operation of the least significant bit of $R[i]$ and $a_i \times B$.

From the algorithm listed above, it can be deduced that modulus N must be odd numbers. As the modulus in RSA cryptosystem is the product of two prime numbers, it is also an odd number; therefore this algorithm is also suitable for the RSA cryptosystem. However, there are two problems remain to be answered.

Problem 1: Result from Montgomery algorithm is $R[n] \equiv A \times B \times 2^{-n}(\text{mod } N)$ which is 2^{-n} more than intended result. To eliminate this extra factor, we can use approaches published by Eldridge and Walter (Eldridge & Walter, 1993); steps are shown below:

(1) Pre–processing: Prior to exercising modular exponentiation, Montgomery algorithm is used to compute the result for the plain text M, pre–computed constant K_p and modulus N where $K_p = 2^{2n}(\text{mod } N)$. And let $M' \equiv MONT(M, 2^{2n} (\text{mod } N), N) \equiv 2^n \times M\ (\text{mod } N)$.
(2) Normal–processing: Replacing M by M' in the binary method operations, that is: $MONT(2^n \times M(\text{mod } N), 2^n \times M(\text{mod } N), N) \equiv 2^n \times M^2(\text{mod } N)$; $MONT(2^n \times M^2(\text{mod } N), 2^n \times M(\text{mod } N), N) \equiv 2^n \times M^3(\text{mod } N)$; etc.
(3) Post–processing: After completing the modular exponentiation, Montgomery algorithm is again used with its result, modulus N and 1 as shown below, to eliminate this extra factor. $MONT(1, 2^n \times R(\text{mod } N), N) \equiv R(\text{mod } N)$.

Problem 2: It can be deduced that $2^i \times R[i] = \sum_{j=0}^{j=i-1} a_j B + \sum_{j=0}^{j=i-1} q_j N$ where $0 \leq R[i] < 2N$. This means that the final result may need to perform subtraction operation (Walter, 1995). Hence, to use this algorithm, the range for R must be limited to [0, N).

Traditional works on the solution are to modify Montgomery algorithm (Chen et al., 1996; Yang et al., 1996) by separating it into two procedures, one for multiplication procedure and the other for Montgomery modular reduction procedure; partial products obtained from the first procedure are fed into the second procedure

for further processing. Blum and Paar (Blum, 1999) proposed to slightly modify Montgomery algorithm and reconfigure hardware architecture and data stream, the modified algorithm can be used for the modular exponentiation.

Later, Walter (Walter, 1999) applied a redundant number system to limit the influence of carries; modular exponentiation then can be done without modifying Montgomery algorithm.

To exemplify: for an n-bit system, the result from each iteration of Montgomery algorithm fall into the range $0 \leq R[i] < N + B < 2N$. If using n+1-bit system to operate on n-bit data, we need to add one more bit of zero value in front of the most significant bit of the original n-bit data to make it n+1-bit. The parameter then looks like: Modulus: $N' = (0\ N)_2 = (0\ n_{n-1}\ n_{n-2}\ \ldots\ n_1\ n_0)_2$, $gcd(N', 2) = 1$, Multiplier: $A' = (0\ A)_2 = (0\ a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0)_2$, Multiplicand: $B' = (0\ B)_2 = (0\ b_{n-1}\ b_{n-2}\ \ldots\ b_1\ b_0)_2$. Now, for the n_{th} iteration, the result of $R[n]$ is $(R[n-1]+a_{n-1}\times B'+q_{n-1}\times N')/2$ and its value is in $[0, 2N]$ or $0 \leq R[n]/2 < N$.

For the $n + 1_{th}$ iteration, the result is $R[n+1] = (R[n] + a_n \times B' + q_n \times N')/2$; since $a_n = 0$, then $R[n+1] = (R[n] + q_n \times N')/2 = R[n]/2 + (q_n \times N')/2$; also $0 \leq R[n]/2 < N$ and $0 \leq (q_n \times N)/2 < N$ then $0 \leq R[n]/2 + (q_n \times N)/2 < 2N$ therefore $0 \leq R[n+1] < 2N$. From there, we know that, in each iteration of the Montgomery algorithm, the result range can be confined to $2N$ for the next iteration. In this way, modular exponentiation can be done with successive multiplication of Montgomery algorithm. Now, in the post-processing, $R_{final} = MONT(1, R_{post}, N')$, where R_{post} represents the result from normal processing and $0 \leq R_{post} < 2N$; furthermore, $MONT(A', B', N') = (A' \times B' + Q \times N') \times 2^{-(n-1)}$, where the maximum value of Q is $2^{n+1} - 1$, therefore $R_{final} = (R_{post} + Q \times N') \times 2^{-(n+1)} < N$. This means that from post-processing operations, the result from modular exponentiation can be limited to the range of [0, N) and no need for an extra subtraction operation.

Summarizing from above, the Walter algorithm and the modified modular exponentiation algorithm (Eldridge & Walter, 1993; Walter, 1993, 1999) used in this paper are listed below as Algorithms 3 and 4, respectively.

Algorithm 3: M_MONT(A, B, N) Walter Algorithm

Input: Modulus: N(n-bit), gcd(N, 2) = 1
Multiplier: $A = (0\ a_n\ a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0)_2$
Multiplicand: $B = (0\ b_n\ b_{n-1}\ b_{n-2}\ \ldots\ b_1\ b_0)_2$
Output: $R[n] \equiv A \times B \times 2^{-n} (\text{mod } N)$
$R[0] = 0$;
for $i = 0; i < n + 2; i++$ **do**
  $q_i = R[i] + a_i \times B (\text{mod } 2)$;
  $R[i+1] = (R[i] + a_i \times B + q_i \times N)/2$;
return $R[n]$;

Algorithm 4 shown above has three sections, where the normal processing section employing the binary method to achieve modular exponentiation. It can be shown that, from pre–processing, $R[0] \equiv 2^{n+2} \times M (\text{mod } N)$, from normal processing, $R[i]$ in each iteration contains 2^{n+2} and, from post–processing, with $A = 1, 2^{n+2}$ can be eliminated and result is limited to [0, N]. Final result from this algorithm is then $R = M^E (\text{mod } N)$, where $R < N$.

Algorithm 4: M_H(M, E, N, k_p) Modified Modular Exponentiation Algorithm

Input: Modulus: N (n-bit) Message: M (n-bit)
Exponent: $E = (1\ e_{k-2}\ e_{k-3} \ldots\ e_1\ e_0)_2$ Constant: $k_p = 2^{2(n+2)} (\text{mod } N)$
Output: $R[k-1] = M^E (\text{mod } N)$
//Pre–processing;
$R[0] = M'$ = M_MONT(M, k_p, N);
//Normal processing;
for $i = 0; i < k-1; i++$ **do**
  $R[i+1]$ = M_MONT($R[i]$, $R[i]$, N);
  if $(e_{k-i-2} == 1)$ **then**
    $R[i+1]$ = M_MONT($R[i+1]$, M', N);
  else
    $R[i+1] = R[i+1]$;
//Post–processing;
$R[k]$ = M_MONT(1, $R[k-1]$, N);
return $R[k]$;

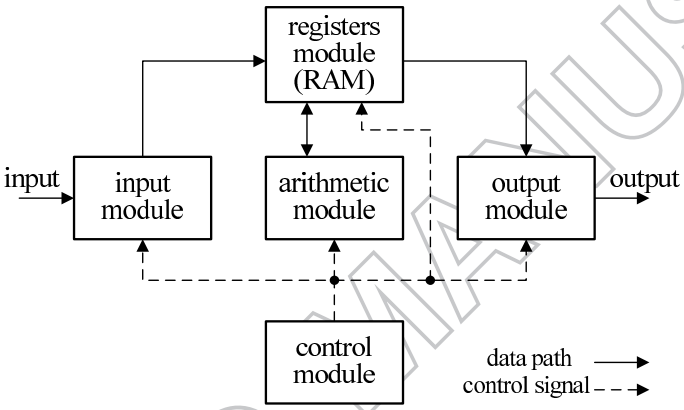

Figure 1. Architecture of the RSA Chip Design

3. RSA VLSI Design

The RSA encryption/decryption chip is configured to compose of four modules: arithmetic module, input/output modules, registers module and control module. We have used the VHDL hardware language to describe each module with the following features: modularity, regularity, local interconnection, and high degree pipelining. Thereby, this design is especially suited for VLSI design. Figure 1 shows its architecture.

3.1 *Input/Output Modules*

Both the input and output modules have three and one 16-bit shift registers, respectively. Input module receives the external 16-bit data in parallel mode and sends them to the registers module, while the output module receives the data from registers module and sends them to external interface in 16-bit parallel mode. By using the handshaking protocol, the input data clock and system clock can be asynchronous as shown in Figure 2.

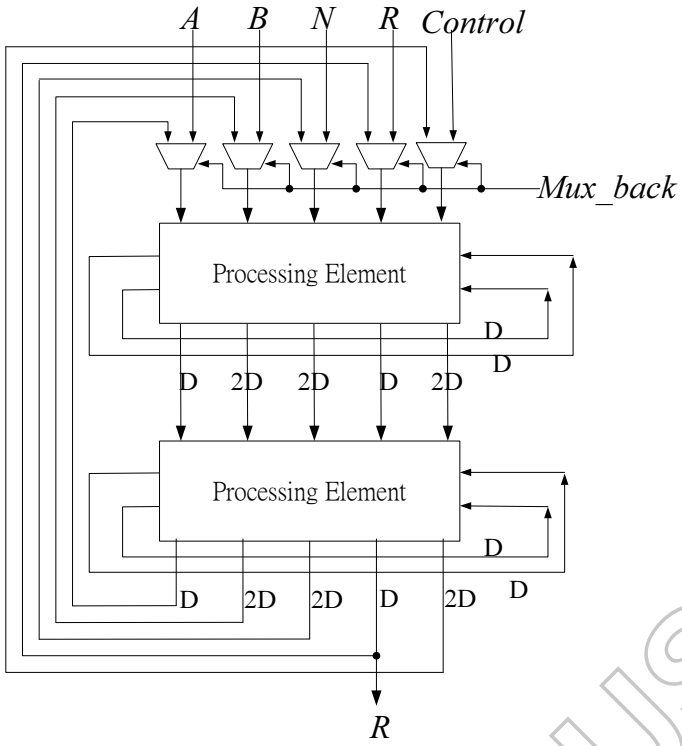

Figure 2. Waveform of the RSA Chip Interface

3.2 *Registers Module*

Registers module satisfied the temporary data storage requirements during the encryption and decryption. It consists four independent Synchronous RAM (SRAM): E register (512∼2048-bit) for the keys, M register (513∼2049-bit) for plant/encrypted text, N register (512∼2048-bit) for modulus and K register (51∼2049-bit) for constant or computation results.

The input data width is 16-bit ordered by MSB to the LSB for each data fetching. In the same way, the output data is also outputted by by MSB to the LSB with 16-bit data width. Since we applied the Walter's algorithm, the input data for modulus systolic array has to be LSB first, hence the data fetching orders of the M, N and K registers are LSB to MSB. Since we also used the H Algorithm, the E register has to be outputted from the MSB to LSB.

3.3 *Arithmetic Module*

Arithmetic module is where modular exponentiation is performed. To facilitate illustration, the following Figure 3 shows the processing scheme of the Montgomery algorithm for n=2. It is obvious that a very regular structure with 3-bit adder array is given in Figure 3 and this regular structure is ideal for systolic array design. Therefore, we applied it to design an n+2 bit by n+2 bit multiplier for Algorithm 3. The Dependency Graph (DG) of Figure 3 is shown in Figure 4. The internal structure of each node in DG is shown in Figure 5.

After completing DG design for supporting the systolic array, Figure 6 shows the corresponding circuit for the Signal Flow Graph (SFG) of modular multiplication, which can be separated into dependency graph projection along the $\vec{d}$ (Projection Direction) and scheduling for modular multiplication DG. Finally, the multiplica-

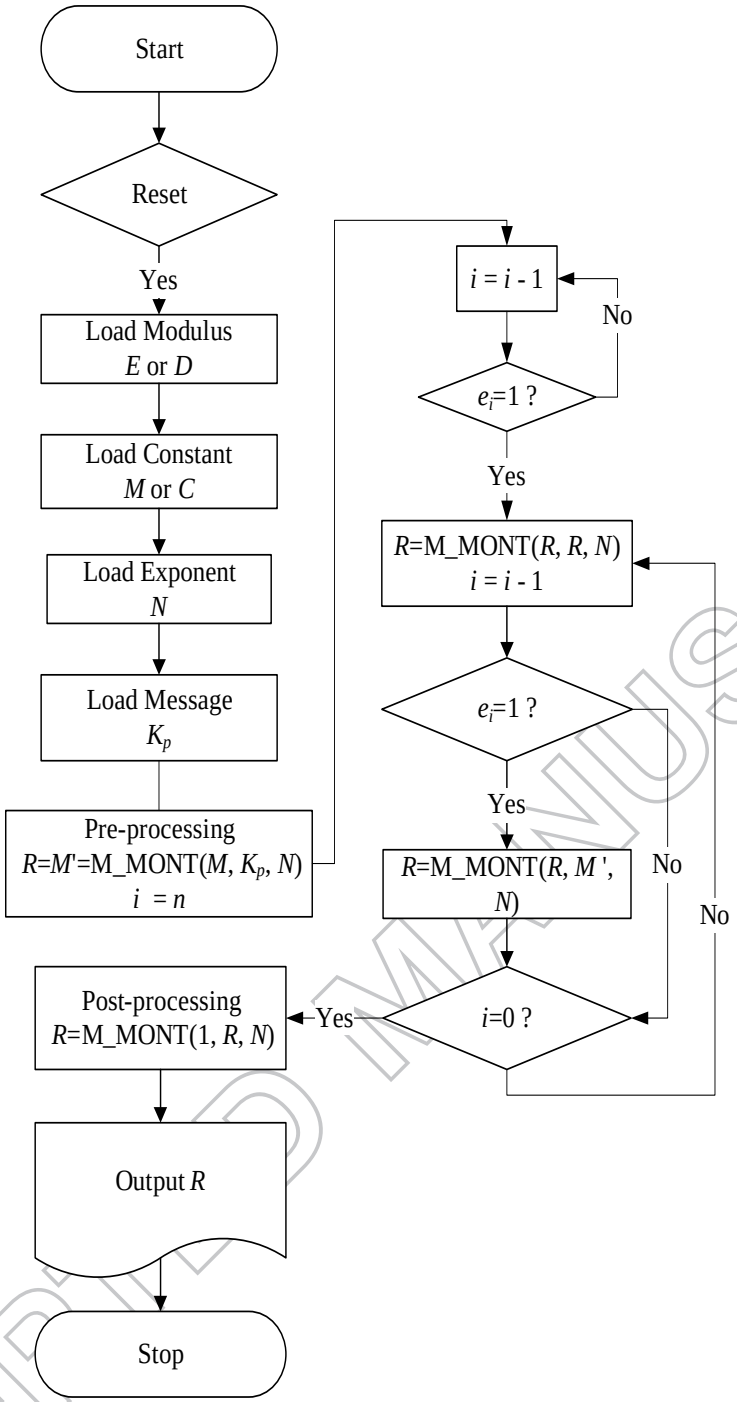

Figure 3. Processing Scheme of the Montgomery Algorithm for $n = 2$

tion result of the SFG will be mapped to the hardware components as shown in the Figure 7, where D indicates the Delay; and Processing Element is the node of Figure 5;

The control signal is $(10\ldots00)_2$. A and B are inputs at top in bit–serial form. The operation $q_i = R[i] + a_i \times B (\mathrm{mod}\ 2)$ can be performed as $R0 \oplus (a_0 \times b_0)$, where $\oplus$ is the xor operation. Since in each basic unit, we must hold the computed quotient digit q_i and the input a_i, we add two latches to the basic unit to lock the correct values of q_i and a_i with control signal.

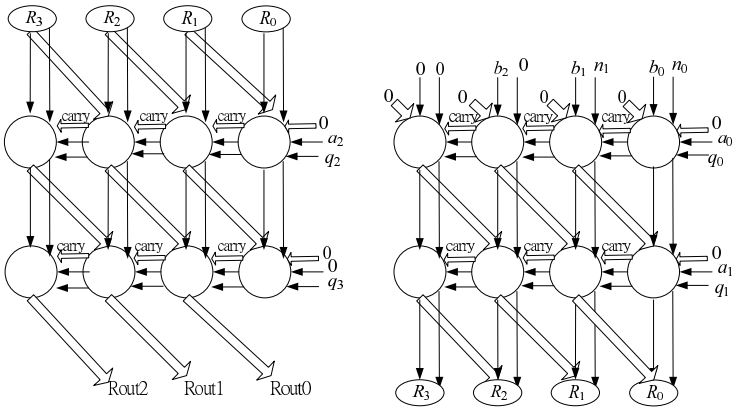

Figure 4. Dependency Graph (DG) for the Montgomery Algorithm for $n = 2$

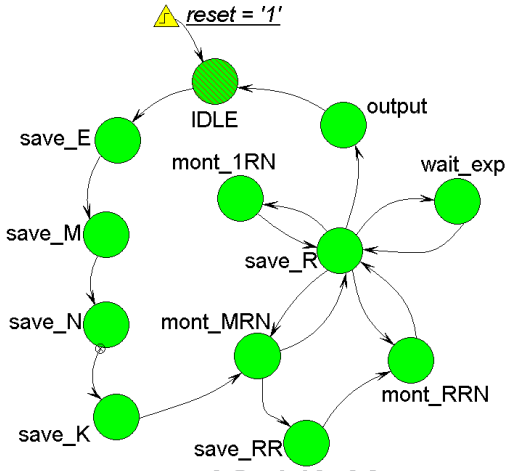

Figure 5. The Internal Structure of each Node in DG

Furthermore, in performing additions, there will encounter the carry problem; we use carry save method to retain and then return the carry to the next basic unit. In addition, the input carries of the least significant unit should be zeroes and its control signal is similar to the latch signal. To effectively utilize idled basic units, the output from the vertical pipeline that can reduce the number of basic units is designed to feedback as input; this is controlled by *Mux_back* signal.

Note that we have implemented the chip interface with 16-bit parallel input/output as illustrated in Figure 8. It has four 16-bit shift registers for buffering the input/output n-bit data A, B, N and R_{out} in a bit-serial form. As the input and output all are in serial bit form, the electrical circuit for the Montgomery algorithm is the bit–serial systolic array.

3.4 *Control Module*

In RSA encryption/decryption control flow, first step is to read data needed for the operation, in the sequence of exponent E or D, message M or C, modulus N then constant K_p. Once the data are read, the modified modular exponentiation algorithm will be performed, from pre–processing, normal processing to post–processing. Following the steps stated above, Figure 9 shows the finite state machine of the control module; Figure 10 is the status diagram which is used to

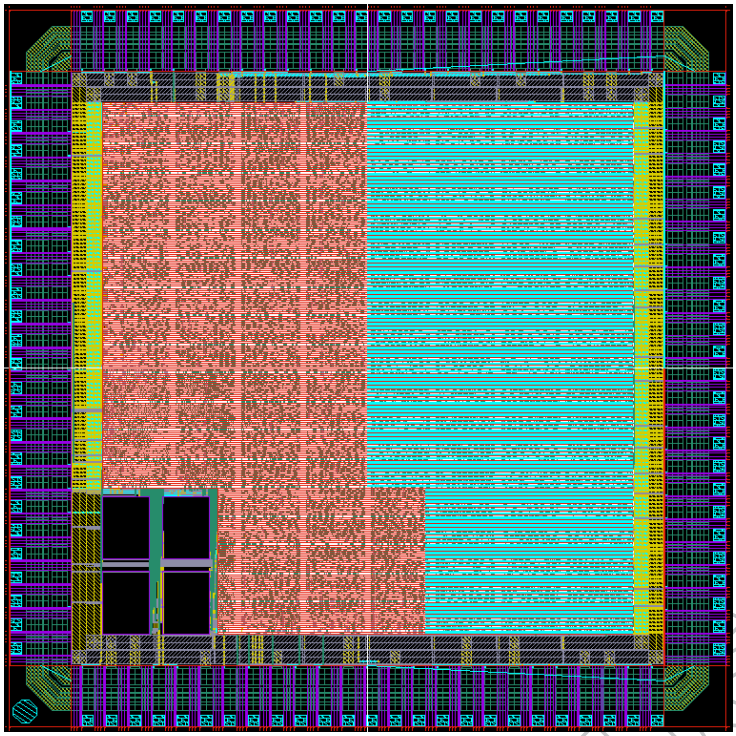

Figure 6. Signal Flow Graph of the Modular Multiplication

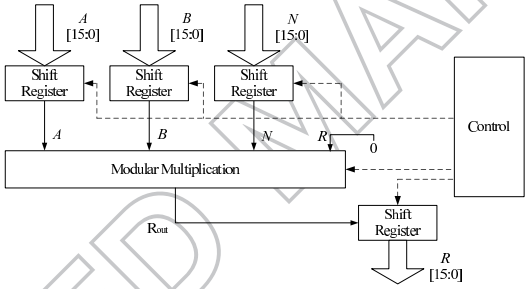

Figure 7. Process Element of Bit–serial Systolic Array Electrical Circuit for Montgomery Algorithm

Table 1. Comparisons on RSA(n-bit) Hardware Resources with Number of Full Adders, Registers, Multiplexes, Memory and the Required Computation Clock Cycles.

Authors	FA	MUX	Reg	RAM	clocks
Juang (Juang, Lu, Lee, & Chen, 1989)	$2n$	$10n$	$14n$	$10 \times n$	$6n^2$
Eldridge (Eldridge & Walter, 1993)	$4n$	$9n$	$16n$	0	$4n^2$
Wang I (Wang, Tsai, & Shung, 1997)	$2n$	$10n$	$13n$	0	$1.5n^2$
Wang II (Wang et al., 1997)	$4n$	$20n$	$26n$	0	n^2
Sheu I (Sheu, Shieh, Wu, & Sheu, 1998)	$3.18n$	$9n$	$10.24n$	0	$2.4n^2$
Sheu II (Sheu et al., 1998)	$3.38n$	$19.1n$	$10.38n$	0	$2.5n^2$
This Design	$n+2$	$n+2$	$n+2$	$4 \times n$	$4.5n^2$

complete the RSA computation.

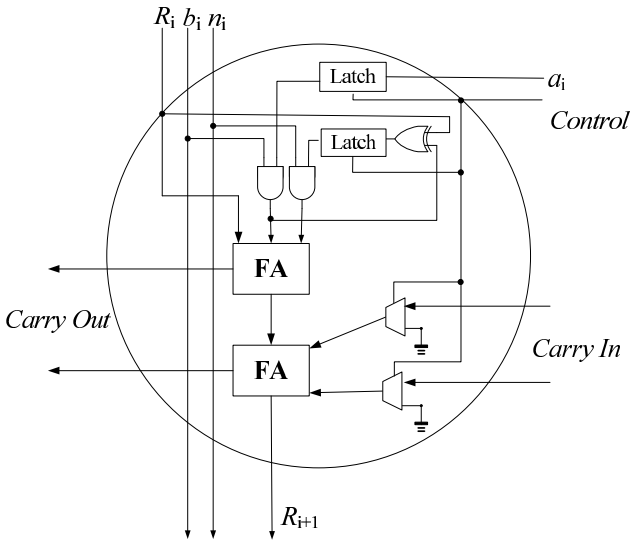

Figure 8. Architecture of the Chip IO Interface

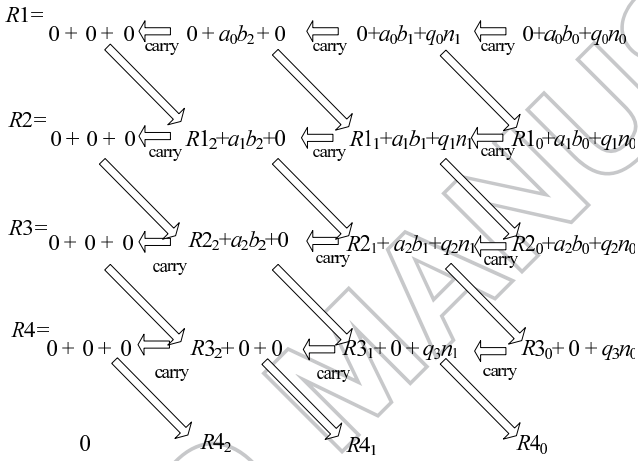

Figure 9. Finite State Machine of the RSA Chip

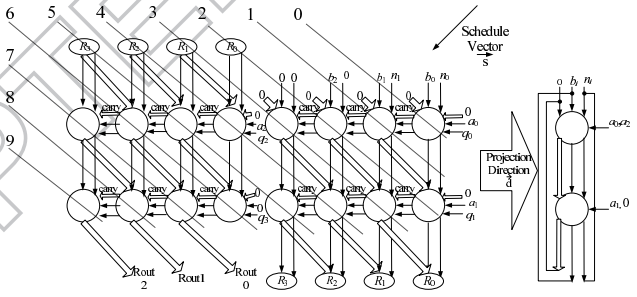

Figure 10. Controller Follow Diagram of the RSA Chip

4. Experimental Result

We have modeled the presented systolic based RSA in VHDL and synthesized it by Synopsys Design Compiler with TSMC/CIC 0.35 μm 1P4M standard cell library. At the end, we used the Cadence Silicon Ensemble to perform the Auto Place Route where Figure 11 shows the chip layout. The post-sim power consumption is

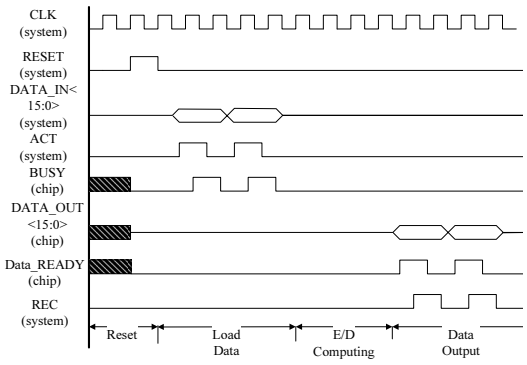

Figure 11. IC layout of the proposed 2048-bit RSA chip design

Table 2. Characteristics of this RSA Encryption/Decryption Chip (TSMC 0.35 μm 1P4M Silicide CMOS).

	Proposed	Hisakado 2006	Chen 2007	Zheng 2008	Miyamoto 2011
Tech	TSMC 0.35	TSMC 0.18	UMC 0.18	TSMC 0.18	STM 0.09
RSA bit	2048bit	2048bit	1024bit	2048bit	2048bit
Voltage	3.3V	1.8V	1.8V	1.8V	1.2V
Frequency	100 MHz	40 MHz	370 MHz	200 MHz	433 MHz
Baud rate	10.84kbps	unknown	83kps	107.5kps	8.64kbps
Power	307.8 mW	61.5 mW	unknown	32.5mW	unknown
Frequency (normalized)	100 MHz	19.6 MHz	181.3 MHz	100 MHz	104 MHz
Baud rate (normalized)	10.84 kbps	unknown	40.67kps	52.675kps	2.07kbps
Power (normalized)	307.8 mW	232.5 mW	unknown	122.9 mW	unknown
Gate count	37.5K	98.5K	175.8K	61.0K	49.8K

estimated by the fast spice PowerMill with random patterns. The die area without the DFT is 3.9 $\times$ 3.9 mm^2 and 4.58 $\times$ 4.58 mm^2 with DFT. Table 2 lists characteristics of this chip while Table 1 lists the hardware resources and the complexity of clock cycles of our design as compared with those of other designs. In Table 2, compared to other 2048-bit RSA Chip Designs, the gate counts of the proposed design is minimal. Although the throughput is not fast, the area is smallest due to the architecture of systolic array. Compared to (Zheng, Liu, & Peng, 2008), the proposed design's normalized power dissipation is higher but it required much less gate counts. Compared to (Miyamoto, Homma, Aoki, & Satoh, 2011), we can obtain a higher baud rate with less gate counts. It provides the comparable computing time among the other approaches but requires smaller chip area.

The average number of clock cycles needed to complete 2048-bit encrypting/decrypting operation of this chip is: $T_{total} \equiv T_{in} + T_m + T_m \times (n + n/2) + T_m + T_{out} \cong 4.5n^2 + 8n = 18.9$ M, for n=2048, where T_{in} is the number of clock cycles needed for data input, T_m is the number of clock cycles needed for one modular multiplication, T_{out} is the number of clock cycles needed for data output, and n is the number of bits for the key. Since the time complexity is $T_m \times (n + n/2)$, when T_m is $3n$, it requires $4.5n^2$. According to the post–simulation of the TimeMill, the average baud rate under a 100 MHz clock is $(n \times f)/T_{total} = 2048 \times 100$ M / 18.9 M $\cong$ 10.84 Kbps.

In comparison with the other designs in Table 1, the proposed systolic based RSA chip needs only $n + 2$ adders, multiplexers, and registers but $4 \times n$ SRAM. Although the proposed architecture required $4 \times n$ bit for storage, the SRAM cell is much smaller than the full adder as shown in the left-bottom layout. In the meantime, high destiny 6–T SRAM cell is already available for the advanced technology node (Sinangil, Mair, & Chandrakasan, 2011). **Therefore, the proposed systolic**

based RSA design complexity is $O(n)$. The scalability can be achieved by reconfiguring the parameter of hardware description language for direct expansion 4096-bit RSA in the future.

Finally, there are two applications that benefit from the proposed $O(N)$ hardware complexity 2048-bit RSA core. Since the proposed design has integrated with a simple 16-bit handshake I/O module as shown in Figure 8, which is very suitable for the low-cost 8051 MCU controller as a RSA SecurID reader. Besides, it can be integrated as a sub-system IP core into a SoC embedded processor design for being extended to secure the important data in portable devices, such as Virtual credit card on smart-phone or NFC reader as well.

5. Conclusion

In this paper, we employ the binary method to split modular exponentiation into a series of modular multiplications, which is then achieved by using the Walter algorithm. We applied the concept of systolic array to design this configurable RSA encryption / decryption chip based on the Montgomery algorithm with modified architecture by using VHDL hardware language. The design was implemented and verified by the TimeMill with TSMC/CIC 0.35 μm 1P4M technology that its area can be reduced to 3.93.9 mm^2 without the DFT and its average baud rate can reach 10.84 Kbps under a 100MHz clock.

References

Blum, T. (1999, Apr). Montgomery modular exponentiation on reconfigurable hardware. In *Ieee symposium on computer arithmetic* (p. 70-77).

Chen, P.-S., Hwang, S.-A., & Wu, C.-W. (1996, May). A systolic rsa public key cryptosystem. In *Ieee international symposium on circuits and systems* (Vol. 4, p. 408-411).

Eldridge, S. E., & Walter, C. D. (1993). Hardware implementation of montgomerys modular multiplication algorithm. *IEEE Transactions on Computers*, *42*(6), 693-699.

Juang, Y.-J., Lu, E.-H., Lee, J.-Y., & Chen, C.-H. (1989). A new architecture for fast modular multiplication. In *Ieee international symposium on vlsi technology, system and application* (p. 357-360).

Knuth, D. E. (1969). *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley.

Miyamoto, A., Homma, N., Aoki, T., & Satoh, A. (2011, July). Systematic design of rsa processors based on high-radix montgomery multipliers. *IEEE Transactions on Very Large Scale Integration Systems,*, *19*(7), 1136-1146.

Montgomery, P. L. (1985). Modular multiplication without trial division. *Mathematics of Computation*, *44*(170), 519-521.

Pellegrini, A., Bertacco, V., & Austin, T. (2010). Fault-based attack of rsa authentication. In *Design, automation & test in europe (date)* (p. 855-860).

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signature and public-key cryptosystems. *Communications of The ACM*, *21*(2), 120-126.

Sheu, J.-L., Shieh, M.-D., Wu, C.-H., & Sheu, M.-H. (1998). A pipelined architecture of fast modular multiplication for rsa cryptography. In *Ieee international symposium on circuits and systems* (Vol. 2, p. 121-124).

Sinangil, M. E., Mair, H., & Chandrakasan, A. P. (2011, February). A 28nm high-density 6t sram with optimized peripheral-assist circuits for operation down to 0.6v. In *Ieee international solid-state circuits conference* (p. 260-262).

Singh, A., & Datta, B. (1935). *History of hindu mathematics 1.*

Walter, C. D. (1993). Systolic modular multiplication. *IEEE Transactions on Computers*, *42*(3), 376-378.

Walter, C. D. (1995). Still faster modular multiplication. *Electronics Letters*, *31*(4), 263-264.

Walter, C. D. (1999, October). Montgomery exponentiation need no final subtractions. *Electronics Letters*, *35*(21), 1831-1832.

Wang, P., Tsai, W., & Shung, C. (1997). New vlsi architecture of rsa public-key cryptosystem. In *Ieee international symposium on circuits and systems* (Vol. 2, p. 2040-2043).

Yang, C.-C., Jen, C.-W., & Chang, T.-S. (1996, November). The ic design of a high speed rsa processor. In *Ieee asia pacific conference on circuits and system* (p. 33-36).

Zheng, X., Liu, Z., & Peng, B. (2008, October). Design and implementation of an ultra low power rsa coprocessor. In *International conference on wireless communications, networking and mobile computing* (p. 1-5).