Efficient Fixed-Size Systolic Arrays for the Modular Multiplication*

Sung-Woo Lee[1], Hyun-Sung Kim[1], Jung-Joon Kim[2], Tae-Geun Kim[2], and Kee-Young Yoo[1]

[1] Department of Computer Engineering
Kyungpook National University
Taegu, Korea, 702-701
{swlee, hskim}@purple.kyungpook.ac.kr
yook@bh.kyungpook.ac.kr
[2] Wireless Comm. Research Lab., Korea Telecom

Abstract. In this paper, we present an efficient fixed-size systolic array for Montgomery's modular multiplication. The array is designed by the LPGS (Locally Parallel Globally Sequential) partition method [14] and can perform efficiently modular multiplication for the input data with arbitrary bits. Also, we address a computation pipelining technique, which improves the throughput and minimizes the buffer size used. With the analysis of VHDL simulation, we discuss a gap between a theoretical optimal number of partition and an empirical one.

1 Introduction

With the increasing popularity of electronic communications, security is becoming a more and more important issue. In 1978, Rivest, Shamir, and Adleman proposed the RSA public key cryptosystem which is relatively easy to implement and has been satisfactorily secure so far [2]. But it requires the fast modular exponentiation with a large modulus that has 512 or more bits in encryption and decryption.

In the last few years, a number of algorithms for the modular multiplication have been proposed and some of them have been realized. One of the most attractive modular multiplication algorithms was proposed by Peter L. Montgomery[3]. By using this algorithm, software and hardware implementations for the fast modular multiplication have been researched [4-9,12,13]. Walter [9] developed a systolic array with $(n+1)(n+2)$ PEs using the Montgomery's algorithm. Heo[12] proposed a systolic array with $n+1$ PEs by transforming Walter's algorithm. The structure is well suited to VLSI implementation. Iwamura[7] and Chen[13] also designed a systolic array with $n+1$ PEs, respectively. Since n is 512 or more, these full-size systolic arrays require a very large space so that is difficult to implement the array in a single chip.

In this paper, we present a efficient fixed-size systolic arrays for Montgomery's modular multiplication. The array is designed by the LPGS(Locally Parallel Globally

* This work is supported by the Korea Telecom, KOREA under a grant NO. 98-13.

Sequential) partition method[14] and perform efficiently modular multiplication for input data with arbitrary bits. Also, we address a computation pipelining technique, which improve the throughput and minimize the buffer size. With the analysis of VHDL simulation, we discuss a gap between a theoretical optimal number of partitions and a empirical that.

2 Modular multiplication algorithm

The Montgomery's modular multiplication algorithm (MMM) was represented. Let A, B, and N be multi-digit numbers with radix r, but let n_0 be a single digit number which satisfies the condition $n_0 * N[0] \bmod r = -1$, where $N[0]$ is the least significant digit of N. The number of digits of A, B, and N is n, $n+1$, and n, respectively. $A[i]$ means the i th digit of A. The Montgomery's algorithm MMM generates $T=ABR^{-1} \bmod N$. R^{-1} is the multiplicative inverse of $R(=r^{n+1})$ modulo N and satisfies GCD(N,R)=1. The number of digits of T is $n+1$.

```
MMM ($A, B, N, r$)
$n_0$ = -N[0]$^{-1}$ mod r
T=0
for $i=0$ $n$-1 {
  $M = ((T[0] + A[i]B[0])\, n_0) \bmod r$
  $T = (T+A[i]B + MN) \operatorname{div} r$
}
if ($T \geq N$) then return $T-N$
        else return $T$
```

The Montgomery's algorithm is potentially faster and simpler than an ordinary modular computation *AB mod N*. However, the conversion from an ordinary residue to an N residue, computation of n_0, and the conversion back to an ordinary residue are time consuming tasks. Thus, it is not a good idea to use Montgomery's algorithm when only a single modular multiplication will be performed. It is more suitable for cases requiring several modular multiplication with respect to the same modulus, as when one needs to compute modular exponentiations. In the modular exponentiation, the output T of MMM can be directly used repeatedly as the next input B. But T can not be used as the input A. We determine that the number of digits of A is $n+1$. This does not affect the result value T of MMM. So the output of MMM can be directly used as the next input for the squaring and multiplication. In the algorithm MMM, it is necessary to convert $A[i]B$ and MN into digit-level computations. Thus the MMM is converted into the following recurrence algorithm MMM1 [9].

Initail value :

$n[0,j] = N[j] \quad 0 \le j \le n-1, \quad n[0,n] = n[0,n+1] = 0$

$b[0,j] = B[i] \quad 0 \le j \le n, \quad b[0,n+1] = 0$

$a[i,0] = A[i] \quad 0 \le i \le n, \quad a[n+1,0] = 0$

$T[0,j] = 0 \quad 0 \le j \le n+1, \quad T[i,n+1] = 0 \quad 0 \le i \le n+1$

```
MMM1 $(A, B, N, r, n0)$
for $i=0$ to $n+1$ do
  for $j=0$ to $n+1$ do {
    if $(j=0)$ {
        $m[i, j+1] = ((T[i, j] + a[i, j]b[i,0])\ n0[i,0]) \bmod r$
        $c_1[i, j+1] = ((T[i, j] + a[i, j]b[i, j]) + m[i, j+1]n[i, j])$ div $r$ div $r$
        $c_0[i, j+1] = ((T[i, j] + a[i, j]b[i, j]) + m[i, j+1]n[i, j])$ div $r$ mod $r$
        $n0[i+1,0] = n0[i,0]$
    } else {
        $m[i, j+1] = m[i, j]$
        $c_0[i, j+1] = ((T[i, j] + a[i, j]b[i, j]) + m[i, j]n[i, j] + c_0[i, j] + c_1[i, j]r)$ div $r$ mod $r$
        $c_1[i, j+1] = ((T[i, j] + a[i, j]b[i, j]) + m[i, j]n[i, j] + c_0[i, j] + c_1[i, j]r)$ div $r$ div $r$
        $T[i+1, j-1] = (T[i, j] + a[i, j]b[i, j] + m[i, j]n[i, j] + c_0[i, j])$ mod $r$
    }
    $a[i, j+1] = a[i, j]$    $b[i+1, j] = b[i, j]$    $n[i+1, j] = n[i, j]$
}
```

In MMM1, each computation occurs at each index point $[i, j]^T$, $(0 \le i, j \le n+1)$ the so-called a computation point. The final result value $T[n+2, 0]$ … $T[n+2, n]$ is ABr^{-n-2} *mod N*, and carry is two digits, i.e. $c_0[i, j+1]$, $c_1[i, j+1]$. The input n_0 is precomputed from $-N[0]^{-1}$ *mod r* and always is not 0. If r is binary, n_0 is always 1.

The recurrence algorithm can be represented by the data dependence graph (DG) as shown in Fig.1. In the DG, each node means computation point and each edge means data flow.

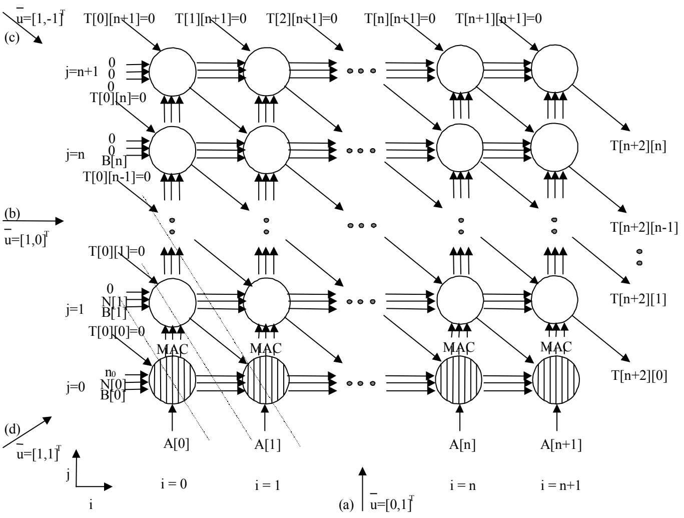

Fig. 1. Dependence Graph of MMM1. The vector $\overline{u}$ is valid projection vectors

3 Fixed size systolic array design

3.1 Full size systolic array

A linear systolic array can be designed according to the systematic procedure [10,11]. From the DG systolic arrays can be designed by a space-time linear transformation. Heo [12] had proved that the systolic array derived by a space vector S=[1, 0] and a schedule vector Π=[2,1] is optimal. Let $c=[i, j]^T$ be a computation point, i.e. a node in the DG. The product Sc gives the coordinate of the PE that the node c is mapped into. And the product Πc gives the time step at which it's computation is performed.

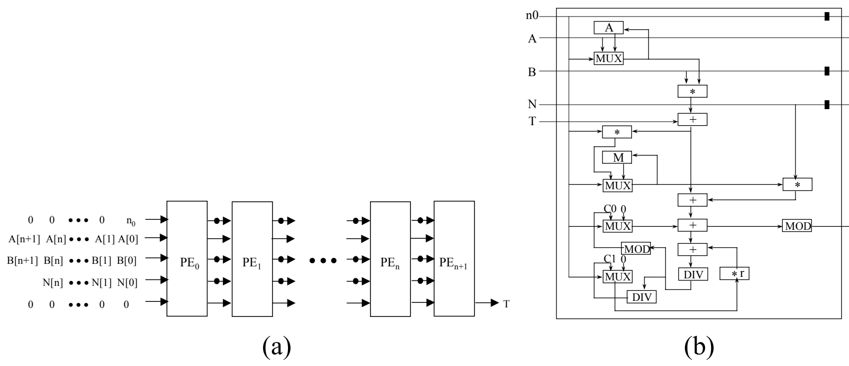

Fig. 2. Full-size Systolic Array (a) Array structure with input data. (b) PE structure

The full-size systolic array derived by the space vector S and the schedule vector Π is shown in Fig. 2. Note the input A is not preloaded, but supplied to the first PE. If the input A is preloaded on each PEs, the number of I/O pins depends on the problem size, causing a implementation to be hard. The systolic array has n+2 PEs and the computation time step is 3n+4.

3.2 A fixed-size systolic array

As the RSA cryptosystem uses 512 or more bit numbers, it is difficult to implement such a large full-size systolic array in a single RSA hardware chip. Thus, it is important to design systolic array who's the number of PEs is much smaller than that of full-size array. For this, we use the well-known *locally parallel, globally sequential* (LPGS) partitioning scheme [14]. The approach is as follows: First of all, assign an index point $c=[i, j]^T$ to a ($\lfloor Sc/p \rfloor$)th band, and a (Sc mod p)th PE where p is the number of PEs. Next, arrange the input data layout of the fixed-size systolic array by partitioning total input data and arranging them for each band computation. As shown in Fig. 3, the DG can be partitioned into the fixed-size systolic array.

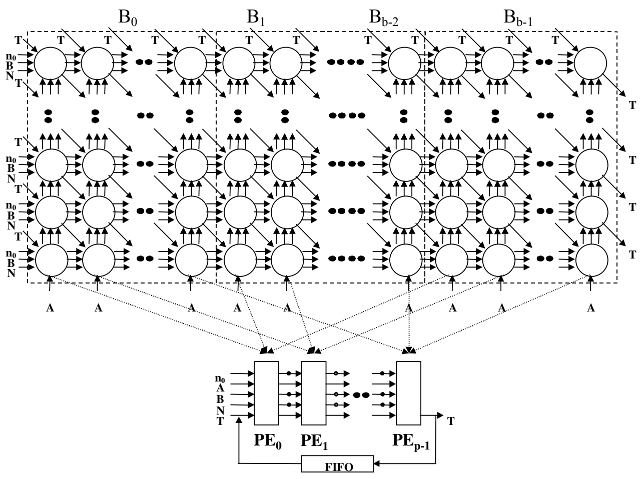

Fig. 3. LPGS partition method. The rectangles imply a band which is referred as B_i.

The following facts are observed from the computation behavior of the derived fixed-size systolic array. (1)Each PE computes all nodes consisting of a corresponding column of each band B_i in the DG. (2)The p digits of A must be inputted at each band B_i. (3) In each band computation, all digits of B and N should be inputted because all digits of B and N are needed for each band computation. (4) Control signals consisting of n_0 and some 0s depending on the time step at which each band B_i is computed, are needed.

From the above facts whenever each band B_i is computed, values are inputted as follows: if p is the number of PEs of fixed-size array, the p digits from pith to $(p(i+1)-1)$th digit of A are inputted at every time step $(2p+n)i+1$. Because all digits of B and N are needed for all band computations, all digits of B and N are inputted. The $n+p+1$ digits of T are need for every band computation. $n+1$ digits among them are inputted from the FIFO buffer but other p digits of T are inputted from the outside at time step $(2p+n)i+n+2$. The $2p+n$ values of control signals are inputted. The first value of control signals is n_0 and the others are all 0s.

Let $I = (P_{ctl}, P_A, P_B, P_N, P_T)$ denote the layout of the input values for the full-size systolic array. P_{ctl}, P_A, P_B, P_N, and P_T are data sequences of control signals, A, B, N, and T, respectively. Each sequence is as follows: note $|P_A|$ denotes the length of sequence P_A. $P_{ctl} = \{n_0, 0, 0, \ldots, 0\}$, $P_A = \{A[0], A[1], \ldots, A[n], 0\}$, $P_B = \{B[0], B[1], \ldots, B[n], 0\}$, $P_N = \{N[0], N[1], \ldots, N[n-1], 0, 0\}$, $P_T = \{T[0][0], \ldots, T[0][n], T[1][n], \ldots, T[n][n]\} = \{0, 0, \ldots, 0, 0\}$. $|P_A| = |P_B| = |P_N| = n+2$. $|P_{ctl}| = |P_T| = 2n+3$.
Let $I^k = (P^k_{ctl}, P^k_A, P^k_B, P^k_N, P^k_T)$ denote the layout of input values for the fixed-size systolic array with k-th band. P^k_{ctl}, P^k_A, P^k_B, P^k_N, and P^k_T are data sequences of control signals, A, B, N, and T for k th bands, respectively. The sequences P^k_B and P^k_N are the same as P_B, P_N for all bands, respectively. But the data sequences P^k_{ctl}, P^k_A, and P^k_T are as follows: $P^k_{ctl} = \{n_0, 0, \ldots, 0\}$, $|P^k_{ctl}| = n+1+p$. Sequence $P^k_A = \{A[kp+0]$,

$A[kp+1]$, …, $A[kp+(p-1)]\}$ and $|P^k_A| = p$. Sequence $P^k_T=\{0, 0, …., 0, 0\}$ and $|P^k_T| = n+1+p$. The sequences P_T and P^k_T are generated by the first PE instead of being inputted. The Fig. 4 shows the fixed-systolic array structure and the input layout for the first band computation. The structures of PE are the same as PE in the Fig. 2(b) except for the last PE.

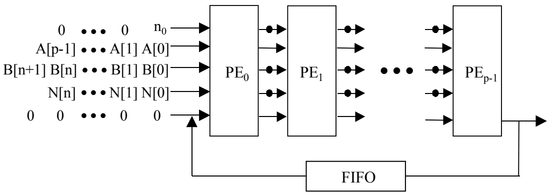

Fig. 4. Derived fixed-size systolic array with input layout for the first band computation

3.2.1 Non-pipelined fixed-size systolic array

After mapping algorithm into a fixed-size systolic array, we should arrange input data. According to a formal method of input data arrangement [11], after each band B_i is completely computed, the computation of the next band B_{i+1} begins in the fixed-size systolic arrays. The intermediate results of band B_i are stored into the FIFO buffer and will be used for the computation of the next band. We analyze the size of buffer and the computation time step.

Let q be the number of bands in the DG. For convenience' sake, assume that $(n+2)$ is a multiple of q.

Lemma 1. Let p be the number of PEs of the non- pipelined fixed-size systolic array. The size of the FIFO buffer, $FIFO_{\text{non}}$ of the derived non-pipelined fixed-size systolic array is as follows:

$$FIFO_{\text{non}} = n. \tag{1}$$

Proof: Suppose that the first value of T is computed on PE_{p-1} for band B_i on time step t. If the output value T is input into PE_0 for the computation of its next band B_{i+1} at time step $t+1$, the buffer does not needed. However, the computation of band B_{i+1} can be started after the computation of band B_i is finished. So, the buffer size depends on only the difference between the time step of the first output from band B_i and the time step which the first input for the next band B_{i+1} is inputted into. The first output from band B_i can be computed at time step $(2p+n)i+2p$ and the first input for band B_{i+1} can be processed at time step $(2p+n)(i+1)+1$ (note: i is the index of all bands, $0\le i<q$). The difference between two time steps is $n+1$. Therefore, the size of the FIFO buffer becomes n. ■

The computation time steps of the non-pipelined fixed-size systolic array is summarized in the following theorem.

Theorem 1. The total computation time step, t_{non} of the derived non-pipelined fixed-size systolic array is as follows:

$$t_{\text{non}} = 3n+4+n\times(\,(n+2)/p-1). \tag{2}$$

Proof: The total computation time of the derived non-pipelined fixed-size systolic array increases more than the full-size systolic array, t_{full} due to delay of the FIFO buffer. So the computation of the band B_{i+1} has to start after the previous band B_i is computed. This delayed time is repeated as the number of bands–1, i.e. q-1. Note that $q=(n+2)/p$. The overhead for partitions, t_{delay} becomes $n\times(q-1)$, i.e. $n\times((n+2)/p-1)$. Therefore, the total computation time step of the derived non-pipelined fixed-size systolic array results in $t_{\text{full}} + t_{\text{delay}}$. So, it is equal to $3n+4+n\times((n+2)/p-1)$. ■

3.2.2 Pipelined fixed-size systolic array

The performance of the non-pipelined array is not efficient, because many idle PEs exist between each band computation. From observation about the computation behavior of the non-pipelined array, the following fact can be obtained: all data flows (links) are the same direction [1,0]. Thus, the first PE (PE_0) can start the computation of the next band immediately after computing of the last node in the last column of the previous band on the DG.

Though the non-pipelined array supplies the input data of a band to the first PE after the last PE performs computation of the last node of the previous band, the pipelined array supplies them after the first PE performs computation of the last node of the previous band. Consequently, we can optimize a total computation time and a PE utilization and minimize a FIFO size.

Lemma 2. The computation time step of value T, which is needed for the computation of the first node in each band is equal to the computation time step of the last node in the previous band.

Proof: Suppose that the time step of the last computation on the PE_0 is t_{last} and the time step of the second computation on the PE_{p-1} is t_{first}. For example, as shown in Fig. 5(a), t_{last} is the time step of node A and t_{first} is the time step of node B. If the first time step on the PE_0 is t_0, t_{first} is $t_0 + n + 1$ and t_{last} is $t_0 + 2(p-1)+1$. In the pipelined fixed-size systolic array, t_{last} is equal to t_{first} because $p = (n+2)/2$. ■

Lemma 3. The FIFO buffer size, $FIFO_{\text{pipe}}$ for the pipelined fixed-size systolic array is as follows:

$$FIFO_{\text{pipe}} = (n+2)-2p \tag{3}$$

Proof: Suppose that the first value of T is computed on $\text{PE}_{\text{p-1}}$ within B_i at time step t, the buffer does not used when value T is inputted at PE_0 for the computation of the next band $\text{B}_{\text{i+1}}$ at time step t+1. On the other band, every value T which will be used for the next time step should be stored in the buffer if PE_0 is working on a node in B_i at time step t+1. So the needed size of the buffer is expected to be the difference between the computation time step of the last node on PE_0 in band B_i, t_{last} and the computation time step of second node on PE_{p-1}, t_{first}. If the computation time step of a band is t_0, t_{first} becomes t_0+n+1 and t_{last} becomes $t_0+2(p-1)+1$. Therefore, the size of the buffer is $(n+2)-2p$. ■

As shown in fig. 5(a), if the first output value T of band B_0 can be used in the computation of band B_1 on PE_0 as soon as the computation of band B_0 on PE_{p-1} is finished, the buffer size is to be 0. Note that if a pipelined array can process the next band without the buffer, the same performance as the full-size systolic array can be obtained just using the only half number of PEs.

If the DG is partitioned into more than 2 bands, the buffer is needed as described in the equation (3). The fig.5(b) shows the case of partition with 3 bands. In fig5(b), the computation nodes on diagonal line can be computed at the same time and the values T computed at the black colored nodes have to be stored in the buffer. Therefore the buffer size is 4.

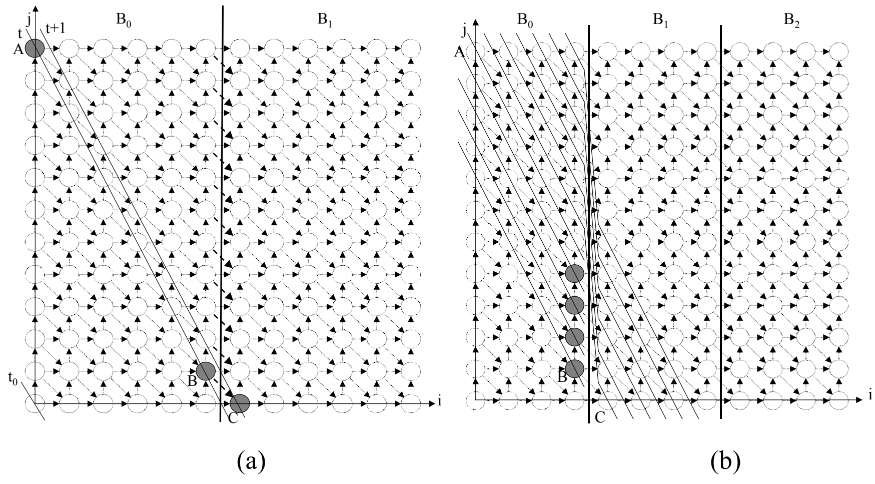

Fig. 5. Partioning of DG.(a) Partitioned DG with 2band, (b) Partitioned DG with 3band

Theorem 2. Let p be the number of PEs of the pipelined fixed-size systolic array. The total computation time step, t_{pipe} for the array is as follows:

$$t_{pipe} = 3n+4+(n+2-2p)((n+2)/p-1) \tag{4}$$

Proof: The proof of this theorem is similar to that of **Theorem** 1. In the pipelined fixed-size systolic array, the buffer size is $(n+2)-2p$. The delayed time is repeated as the number of bands–1. So the delay t_{delay} is $(n+2-2p)((n+2)/p-1)$. Therefore, the time step of the derived pipelined fixed-size systolic array results in $t_{full} + t_{delay}$. So the total computation time step is equal to $3n+4+(n+2-2p)\ ((n+2)/p-1)$. ■

The difference of computation time steps between the non-pipelined and pipelined fixed-size systolic array is $(2p-2)((n+2)/p-1)$ and the difference of the buffer size is $2p-2$. It shows that the pipelined fixed-size systolic array has more efficient performance than the non-pipelined fixed-size systolic array. Moreover when DG is partitioned into 2 bands, the buffer is no longer needed.

3.3 Examples of a pipelined fixed-size systolic array

Let n be 10. The $(n+2)\times(n+2)$ DG of the Montgomery's algorithm is partitioned into 2 bands, which are mapped into a fixed-size systolic array with 6 PEs. In the nonpipelined array, the size of the FIFO buffer is 10, each band computation time step is 22 and the total computation time step is 44. In the pipelined fixed-size systolic array, the total computation time step is 34 which is equal to the full-size systolic array's computation time step and the buffer is no longer needed.

4 VHDL simulation

We simulate the pipelined fixed-size systolic array using ALTERA MAX+PLUS II which is a programmable logic device simulation tools. The FLEX10KA families of ALTERA's FPGA device is used. Fig.6 shows the result of simulation when base is 16 and the bit number of input data A, B, and N is 126. In the VHDL simulation of the pipelined fixed-size systolic array, the array is partitioned into 2, 4, and 8 bands. As shown fig.6, there are some differences between the theoretical analysis and experimental results. The reason of it is that the larger a chip area is, the longer a minimum clock cycle of the tool. If a execution time is only considered, a 2 bands partitioning is best. But if the chip area and execution time are all considered, a 4 bands partitioning is the best.

Table 1. Simulation result

Pes	FIFO size	# of logic cell	Clock cycle (ns)	Execution time(us)
128 (full size)	0	31101	214	81.748
64 (2 bands)	0	15309	151	57.833
32 (4 bands)	64	7809	104.4	59.925
16 (8 bands)	96	3928	90.2	95.071

5 Conclusion

We presented a pipelined fixed-size systolic arrays for Montgomery's modular multiplication which has input data with arbitrary bits. Also we show that our pipelining technique improves the throughput and minimize the buffer size. It's total computation time step is $3n+4+(n+2-2p)((n+2)/p-1)$, and a buffer size is $(n+2)-2p$. Moreover, when the DG is partitioned into 2 bands, the pipelined fixed-size systolic array has the same time steps as the full-size systolic array and the number of PEs are reduced by half. As the analysis of VHDL simulation, for 504 bits input data, when partitioning the algorithm into 4bands, the proposed pipelined fixed-size systolic array requires only 1/4 chip area of full-size array's area and its computation time is approximately best.

References

1. W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Trans. on Info. Theory*, vol. IT-22(6) (1976) 644-654
2. R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Commu. ACM.* vol. 21 (1978) 120-126
3. P.L. Montgomery, "Modular Multiplication Without Trial Division," *Mathemat. of Computat.* vol. 44 (1985) 519-521
4. Antoon Bosselaers, Rene Govaerts, and Joos Vandewalle, "Comparison of three modular reduction functions," *Advances in Cryptology-CRYPTO '93* (1993) 175-186
5. S.R. Dusse and B.S. Kaliski Jr., "A Cryptographic Library for the Motorola DSP56000," in *Advances in Cryptology EUROCRYPT '90*, Springer-Verlag (1991) 230-244
6. S.E. Eldridge and C.D. Walter, "Hardware Implementation of Montgomery's Modular Multiplication Algorithm," *IEEE Trans. Comput.* vol. 42 (1993) 693-699
7. K. Iwamura, T. Matsumoto, and H. Imai, "Systolic Arrays for Modular Exponentiation using Montgomery Method," *Proc. Euro CRYPT '92* (1992) 477-481
8. J. Sauerbrey, "A Modular Exponentiation unit based on Systolic Arrays," *Abst. AUSCRYPT '92* (1992) 1219-1224
9. C.D. Walter, "Systolic Modular Multiplication," *IEEE Trans. on Computers,* vol. 42 (1993) 376-378
10. S.Y. Kung, *VLSI Array Processors*, Prentice-Hall (1987)
11. K.Y. Yoo, "*A Systolic Array Design Methodology for Sequential Loop Algorithms*," Ph.D. thesis, Rensselaer Polytechnic Institute (1992)
12. Y.J. Heo, K.J. Lee and K.Y. Yoo, "An optimal linear systolic array for computing montgomery modular multiplication", *PDCS '98* (1998)
13. Po-Song Chen, Shih-Arn Hwang and Cheng-Wen Wu, "A systolic RSA public key cryptosystem", *IEEE International Symposium on Circuits and Systems,* vol. 4 (1996)
14. D.I. Moldovan and J.A.B. Fortes, " Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays", IEEE transaction on Computers, vol. C-35, no.1, January (1986) 1-12