Scientific Research

High-Speed Montgomery Modular Multiplication Using High-Radix Systolic Multiplier

ZHANG Rui, He Debiao, CHEN Jianhua, HU Jin
School of Mathematics & Statistics, Wuhan University, Wuhan, China
e-mail: *zhangrui_ecc@163.com*

Abstract: A new high-radix systolic multiplier architecture is presented for Montgomery modular multiplication. Using a radix 2^w, an n-bit modular multiplication only takes about n/w+6 cycles. This leads to a competitive ASIC implementation for RSA and Elliptic Curve Cryptography (ECC).

Keywords: public key cryptosystems; Montgomery algorithm; modular multiplication; systolic multiplier

1. Introduction

The RSA [1] and Elliptic Curve Cryptography (ECC) [2,3] public-key cryptosystems play an important role in information security. They are based on modular mathematics and modular multiplication is the crucial operation for time and resources.

Montgomery algorithm [4] is widely used for hardware implementation of modular multiplication. There are different implementations of the algorithm for different application environment. The bitwise implementations appeared in [5-8] firstly. Then several high-radix implementations for high-speed design were [9,10] proposed.

Most Field Programmable Gate Arrays (FPGA) implementations based systolic multiplier because FPGA can offer multipliers embedded in the devices [12,14]. But for a very high-speed design, the Application Specific Integrated Circuits (ASICs) offer more advantages. ASICs can operate at higher frequency. This is important for high performance design.

In this work, a new ASIC implementation is presented using high-radix systolic multiplier which takes the least cycles for a modular multiplication. The rest of this paper is organized as follows. In Section 2, the modified simple Montgomery algorithm is introduced. Section 3 is a detailed description of our proposed architecture. Results are presented in Section 4. A concluding summary is given in Section 5.

2. Algorithm Descriptions

For a modulus M and inputs A, B, it performs modular multiplication to output $S \equiv ABR^{-1}(\text{mod } M)$ (R is a constant of form 2^n).

For n bit modulus M with GCD $(M, 2) = 1$, define

$$e = \left\lceil \frac{n}{w} \right\rceil + 2$$

and $R=(2^w)^e$, integers R^{-1} and M' are given by $(R*R^{-1}) \text{mod } M=1$ and $(-M\ M')\text{mod } 2^w=1$, integer $\tilde{M}$ is given by $\tilde{M} = (M' \text{ mod } 2^w)M$ and $4\tilde{M} < R$, a constant

$MC=(\tilde{M}+1)/2^w = \sum_{i=0}^{e-3} m_i (2^w)^i$,.$m_i \in \{0,1, \quad 2^w-1\}$

A high-radix version of Montgomery Algorithm is [11]:

Algorithm 1: Radix 2^w Simplified Version of the Montgomery Algorithm

***Input*:**

Multiplicand

$A = \sum_{i=0}^{e-1} a_i (2^w)^i$, $a_{i<(e-1)} \in \{0,1,...2^w-1\}, a_{e-1} \in \{0,1\}$, $0 \le A \le 2\tilde{M}$;

Multiplier

$B = \sum_{i=0}^{e} b_i (2^w)^i$,

$b_{i<(e-1)} \in \{0,1,...2^w-1\}, b_{e-1} \in \{0,1\}, b_e = 0$, $0 \le B \le 2\tilde{M}$.

Constant

$MC = \sum_{i=0}^{e-1} m_i (2^w)^i$,

$m_{i<(e-2)} \in \{0,1,...2^w-1\}, m_{e-2} = m_{e-1} = 0$, $0 \le A \le 2\tilde{M}$;

***Output*:** An integer S_{e+1} where $S_{e+1} \equiv ABR^{-1}$ (mod M) and $0 \le S_{e+1} < 2\tilde{M}$

```
Step 1: $S_0$:=0;
Step 2: for $i$:=0 to $e$ do
Step 2a: $q_i$:=$S_i$mod $2^w$;
```

Step 2b: S_{i+1}:=$S_i/2^w$+q_i*MC+b_i*A.

end for

Compared with original Montgomery Algorithm, Algorithm 1 eliminates the direct data dependence between the part sum S_i and part quotient q_i. It is required only a simple truncation of S_i. So the main operation of the Algorithm 1 is simplified to Step 2b, i.e., two multiplications and a three-operand addition. Systolic multiplier architecture is proposed to perform this operation in Section 3.

3. Hardware Design

The main design method of systolic multiplier is to split the long integer operation into small segment operation and combine the discrete result. The small function module is often called Process Element (PE). A new PE module is designed to split operation in algorithm Step 2b.

3.1 Processing Element (PE)

To calculate the S_{i+1} in Algorithm 1 Step 2b, it can be observed that only its least significant word is need for next iteration. So it is not necessary to calculate each word of S_{i+1}'s in one cycle and it can be saved as some redundant presentation. The CSA-staged multiplier is usually used for such a form [13]. But the disadvantage for CSA array is greater place & route effort for its irregularity.

So a new Processing Element defined as:

Inputs: q, m, b, a, s_in, c_in, h_in. Here q, m, b, a, s_in,h_in ε $[0,2^w-1]$, c_in is one bit value.

Output: {c_o, h_o, s_o}=q*m+b*a+s_in+{c_in,h_in}

Here h_o, s_o ε $[0, 2^w-1]$, c_o is only one bit 0 or 1, {} denotes the combination of different parts. Figure 1 shows the PE's function.

3.2 PE Array and Modular Multiplication Module

To implement the high-radix algorithm, a PE array structure was employed for minimal delay and high place & route utilize ratio.

In this structure, there are process elements: {PE_k,k=0,…,e-1} described above and three row regiters: {regs_in[k]}, {regh_in[k]}, {regc_in[k]}, {regc_in[k]},

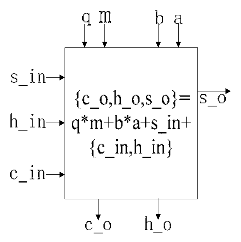

Figure 1. Function of PE

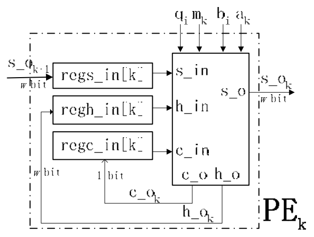

Figure 2. Connections of PE_k

k=0,…,e-1. Figure 2 shows the connections in PE and between PEs (synchronous clock input not included).

Now we can rewrite the Algorithm 1 for PE array implementation:

Algorithm 2:

Input: As Algorithm 1

Output: An integer S where $S \equiv ABR^{-1}$ (mod M) and $0 \leq S < 2\tilde{M}$

```
Step 1: regs_in[k]:=0, regh_in[k]=0, regc_in[k]=0 for k=0,…,$e-1$; regq=0;
Step 2: for $i$:=0 to $e$ do
Step 2a: regs_in[k]=$s\_o_{k+1}$; regh_in[k]=$h\_o_k$; regc_in[k]=$c\_o_k$, for k=0,…,$e-1$; regq=$s\_o_0$;
end for
Step 3a: regs_in[k]=$s\_o_{k+1}$; regh_in[k]=$h\_o_k$; regc_in[k]=$c\_o_k$, for k=0,…,$e-1$; regq=0;
Step 3b: regs_in[k]=$s\_o_{k+1}$; regh_in[k]=$h\_o_k$; regc_in[k]=$c\_o_k$, for k=0,…,$e-1$; regq=$s\_o_0$;
Step 3c: $S$:={regs_in[$e-1$],….,regs_in[0],regq}+ {regh_in[$e-1$],….regh_in[0],0}.
```

Figure 3 shows the PE array structure for implementing Algorithm 2. The registers all initialized as 0. After that, the outputs of each PE are saved in corresponding registers at each cycle. The outputs h_o_k, c_o_k are feed back to regh_in[k], regc_in[k]. The output s_o_k is send to regs_in[k-1] as a input of PE_{k-1}; The output s_o_0 is

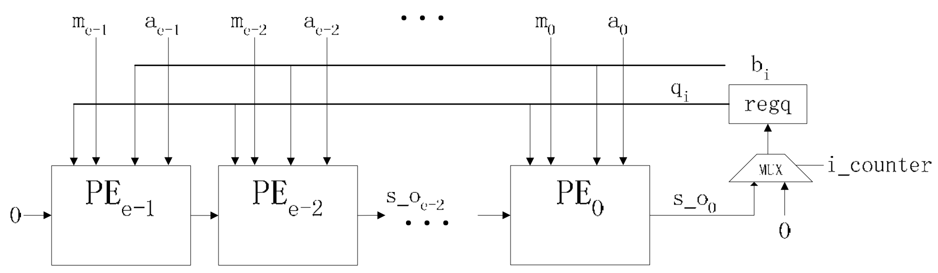

Figure 3. PE array for Modular Multiplication

saved in reg*q*, which corresponds to the q_i in Algorithm 1 Step 2b.

Actually, the S_{i+1} in Algorithm 1 Step 2b are presented as its redundant form in Algorithm 2:

$S_{i+1}==s\_o_0+(\{c\_o_0,h\_o_0\}+s\_o_1)(2^w)+(\{c\_o_1,h\_o_1\}+s\_o_2)(2^w)^2+\ldots+(\{c\_o_{e-1},h\_o_{e-1}\})(2^w)^{e-1}$

In Algorithm 2, the outputs c_o_k (for k=0,…,e-1) of PE array in Step3b are equal to 0 because the inputs b_e, q_e of PE array assigned to 0 when to execute Step3a. At last, Step3c $S=s\_o_0+(\{c\_o_0,h\_o_0\}+s\_o_1)(2^w)+(\{c\_o_1,h\_o_1\}+s\_o_2)(2^w)^2+\ldots+(\{c\_o_{e-1},h\_o_{e-1}\})(2^w)^{e-1}$={regs_in[e-1],….,regs_in[0],regq}+{regh_in[e-1],….regh_in[0],0}. The addition of last step can be completed with adders in 1~2 cycle according to the area and timing requirement.

Furthermore, the inputs m_{e-1}, m_{e-2} of PE_{e-1}, PE_{e-2} are equal to 0 and a_{e-1}, b_{e-1} is 0 or 1, so PE_{e-1}, PE_{e-2} have a simpler structure.

4. Results

If the last addition is completed in *r* cycle, an n-bit modular multiplication only takes

$$\left\lceil \frac{n}{w} \right\rceil + 5 + r$$

the proposed architecture. This is the fastest result compared with the similar architecture. Let

$$t=\left\lceil \frac{n}{w} \right\rceil$$

Table 1 shows the clock cycles needed for an n-bit modular multiplication and different structure.

The crucial module for timing and area is PE in the PE array architecture. Using SMIC 0.18μm standard cell library and worst operation condition, the Synopsys Design Compiler synthesized results are showed in Table 2.

Table 1. Comparisons with previous works

Proposed implementation	clock cycles
Ours	about t+6
[12]	about (3t+7)/2
[14]	about 2(t+5)

Table 2. Synthesized result of PE using 0.18μm process library

w	Area(mm^2)	Max_Delay(ns)
8	0.011	3.3
16	0.054	4.2
32	0.189	5.0

The architecture also has simper connections between each base unit. Using the place & route tool Encounter, the place & route utilize ratio of modular multiplication module ranges in 90%~95%. The DC's ability of arithmetic optimization contributes to the high utilize radio without much handwork.

5. Conclusions and Future Work

To achieve a high speed modular multiplication design, we have described an efficient implementation of Montgomery algorithm. The architecture enables a high radix up to 2^{32}, reducing the number of cycles for a modular multiplication. Only about

$$\left\lceil \frac{n}{w} \right\rceil + 6$$

clocks are taken for an n-bit modular multiplication using the proposed architecture.

One direction in which this work should go is to simplify the PE's structure reducing the area and delay. The other is to design an extendable structure for different modular length in a PE array.

References

[1] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, Vol. 21, pp. 120-126, 1978.

[2] V. S. Miller. Use of Elliptic Curves in Cryptography. Proceedings of Advances in Cryptography, pp. 417-426, 1986.

[3] N. Koblitz. Elliptic Curve Cryptosystems. Math Computation, Vol.48, pp. 203-209, 1987.

[4] P. L. Montgomery. Modular Multiplication without Trial Division. Math Computation, Vol. 44, pp. 519-521, 1985.

[5] C.D. Walter. Systolic Modular Multiplication. IEEE Transactions on Computers, Vol. 42, pp. 376–378, 1993.

[6] K. Iwamura, T. Matsumoto, H. Imai. Montgomery Modular Multiplication Method and Systolic Arrays Suitable for Modular Exponentiation. Electronics and Communications in Japan, Part 3, Vol. 77, pp.40–51, 1994.

[7] P. Wang. New VLSI Architectures of RSA Public-key Cryptosystems. In Proceedings IEEE International Symposium on Circuits and Systems, Vol. 3, pp. 2040–2043, 1997.

[8] A. Tiountchik. Systolic Modular Exponentiation via Montgomery Algorithm. Electronics Letters, Vol. 34, pp. 874–875, 1998.

[9] T. Blum, C. Paar. Montgomery Modular Exponentiation on Reconfigurable Hardware. In Proceedings 14th IEEE Symposium on Computer Arithmetic, pp. 70–77, 1999.

[10] T. Blum, C. Paar. High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware. IEEE Transactions on Computers, Vol. 50, pp. 759-764, 2001.

[11] H. Orup. Simplifying quotient determination in high-radix modular multiplication. Proc. of the 12th Symposium on Computer Arithmetic, 1995.

[12] C. McIvor, M. McLoone, J.V. McCanny. High-Radix Systolic Modular Multiplication on Reconfigurable Hardware. Proceedings of the 2005 IEEE International Conference on Field- Programmable Technology, pp.13 – 18, 2005.

[13] A. Cilardo, A. Mazzeo, L. Romano, G. P. Saggese. Carry-Save Montgomery Modular exponentiation on Reconfigurable Hardware. Proceedings of the Design and Test Europe (DATE) Conference 2004, Vol. 3, pp. 206-211, 2004.

[14] S.H. Tang, K.S. Tsui, P.H.W. Leong. Modular exponentiation using parallel multipliers. Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology, pp. 52-59, 2003.