A Radix-4 Modular Multiplication Hardware Algorithm Efficient for Iterative Modular Multiplications

Naofumi Takagi
Department of Information Science
Kyoto University
Kyoto 606, Japan

Abstract

A fast radix-4 modular multiplication hardware algorithm is proposed. It is efficient especially in applications, such as encryption/decryption in RSA cryptosystem, where modular multiplications are carried out iteratively. Each subtraction for the division for residue calculation is embedded in the repeated multiply-addition. Numbers are represented in a redundant representation and addition/subtractions are performed without carry propagation. A serial-parallel modular multiplier based on the algorithm has a regular cellular array structure with a bit slice feature suitable for VLSI implementation.

1 Introduction

In encryption/decryption in RSA cryptosystem [1], modular multiplications with a large modulus (longer than 500-bit) are carried out iteratively. Design of a fast algorithm for modular multiplication with a large modulus is the key to developing a high performance encryption/decryption circuit for such a cryptosystem. In this paper, we propose a fast modular multiplication hardware algorithm which is efficient especially in applications where modular multiplications are carried out iteratively.

Various algorithms for modular multiplication have been proposed and some of them have been realized [2]. Most of them are classified into two methods, i.e., "division-after-multiplication" and "division-during-multiplication". In an n-bit modular multiplication by the former method, an ordinary n-bit multiplication is carried out first and then a 2n-bit by n-bit division for residue calculation is performed. In the latter method, each subtraction step for the division for residue calculation is embedded in the repeated multiply-addition [3]. The latter requires a less amount of hardware than the former does [4]. However, in general, the latter requires more addition/subtractions than the former does [5]. In either method, the key point to increasing the computation speed is to perform addition/subtractions of long numbers fast.

We can perform addition/subtractions of long numbers fast without carry propagation by the use of a redundant representation, such as the carry save form and the signed-digit representations. Vandemeulebroecke et al proposed a division-after-multiplication algorithm with the redundant binary representation, i.e, the radix-2 signed-digit representation where addition/subtractions for the division as well as for the multiplication are performed without carry propagation [5]. Preparata and Vuillemin showed an efficient division-during-multiplication algorithm with redundant representations as an application of their cellular division method [6]. Morita proposed a similar algorithm with the carry save form based on a higher radix, independently [4]. Recently, we developed a radix-2 and a radix-4 division-during-multiplication algorithm with the redundant binary representation [7].

Preparata et al's algorithm, as well as Morita's, overcame the drawback of the division-during-multiplication method. Namely, the number of addition/subtractions required by them is almost the same as that required by the division-after-multiplication method. In particular, the radix-4 version of Morita's algorithm is very efficient, because we can generate the multiples of the modulus required for residue calculation, as well as the multiples of the multiplicand, by negating (complementing) and/or shifting them. However, in these algorithms, the operands have to be in the ordinary binary representation, while the intermediate results are in redundant representations. This fact decreases the computation speed of iterative multiplications where the product of the former multiplication is used as the operands of the next multiplication, because a time-consuming conversion which includes a carry-propagate addition of long numbers is required at each multiplication. Our former algorithms do not have this drawback but require more addition/subtractions.

In this paper, we propose a new radix-4 modular multiplication algorithm with a redundant binary representation. It is a kind of division-during-multiplication method. The number of required addition/subtractions is as the same as that required by Morita's radix-4 algorithm and is about the half of that required by our former radix-4 algorithm. The multiplicand, as well as the multiplier, can be in a redundant representation. The product is also in the same redundant representation as the operands. Hence, the product can be used as either of the operands of the next multiplication in iterative multiplications. Namely, we can keep

the intermediate results in the redundant representation, and perform the conversions only at the beginning and the end of the whole iterative multiplications. We do not need the time-consuming conversion of long numbers at each multiplication. Therefore, our algorithm is more efficient than Preparata and Vuillemin's and Morita's in iterative multiplications for, e.g., RSA encryption/decryption.

A serial-parallel modular multiplier based on the proposed algorithm has a regular cellular array structure with a bit slice feature suitable for VLSI implementation. The depth of its combinational circuit part is a constant independent of n, the length of the modulus, and therefore, it can operate with a fast clock. Its amount of hardware is proportional to n. It seems easy to implement a high performance RSA encryption/decryption circuit based on the multiplier on a VLSI chip using today's technology.

In the next section, we describe redundant representations for a residue class based on the redundant binary representation. We propose a radix-4 modular multiplication hardware algorithm, and show its correctness in Section 3. In Section 4, we consider a serial-parallel modular multiplier based on the algorithm. In Section 5, we apply the multiplier to RSA encryption/decryption. Section 6 is a conclusion. We explain how the proposed algorithm has been derived, in Appendix.

2 Redundant Representations for a Residue Class

We consider multiplication in a residue class $Z_Q = \{0, 1, ..., Q-1\}$ where $2^{n-1} \leq Q < 2^n$. We assume that the multiplicand and the multiplier are represented in a redundant representation, and calculate the product represented in the same redundant representation. We use a redundant representation based on the redundant binary representation, i.e., the radix-2 signed-digit representation. We also represent all partial products (intermediate results) in another redundant representation which is also based on the redundant binary representation. We perform all calculations for modular multiplication in the redundant binary representation. For the partial products, we use a representation which is more redundant than that for the operands and the product. This is one of the key points to getting our very efficient algorithm.

The redundant binary representation has a fixed radix 2 and a digit set $\{\bar{1}, 0, 1\}$, where $\bar{1}$ denotes -1 [8]. An n-digit redundant binary number $A = [a_{n-1}a_{n-2}...a_0]$ ($a_i \in \{\bar{1}, 0, 1\}$) has the value $\sum_{i=0}^{n-1} a_i \cdot 2^i$. Hereafter, we use A to denote both a representation and its value. Note that there may be several redundant binary numbers which have a certain value. Using this redundancy, we can add two redundant binary numbers without carry propagation. For the details of carry-propagation-free addition, see, e.g., [9]. We can get a negation of a redundant binary number by changing the signs of all nonzero digits in it.

For the multiplicand, the multiplier and the product, we represent $\alpha \in Z_Q$ by an n-digit redundant binary number, A, which satisfies $-d_1 \cdot Q < A < d_1 \cdot Q$ and $A \equiv \alpha \pmod{Q}$. For the partial products, we represent $\beta \in Z_Q$ by an $(n+2)$-digit redundant binary number, B, which satisfies $-d_2 \cdot Q < B < d_2 \cdot Q$ and $B \equiv \beta \pmod{Q}$. d_1 and d_2 can be any numbers which satisfy the following conditions.

$$\frac{9}{16} \leq d_1 \leq \frac{55}{96}$$
$$\frac{9}{4} \leq d_2 \leq \frac{55}{24}$$
$$2 \cdot d_1 + 3 \cdot d_2 \leq 8$$
$$4 \cdot d_1 \geq d_2$$

We will show how the above conditions have been derived, in Appendix.

We can convert (the ordinary unsigned binary representation of) α ($\in Z_Q$) to the former redundant representation by comparing α with $Q/2$ down to the $(n-5)$th position and subtracting Q from α if the former is larger. (The i-th position of a number means the i-th position from the least significant position, which has the weight 2^i when the number is in radix-2.) We can perform the subtraction in the redundant binary representation without carry propagation. On the other hand, we can convert A to α by an ordinary binary subtraction and an addition. Namely, we calculate $A^+ - A^-$ or $A^+ - A^- + Q$ and chose the former as α if it is non-negative, where A^+ and A^- are n-bit binary numbers formed from positive and negative digits of A respectively. We need subtraction and addition with carry propagation in this conversion. Note that these conversions are required only at the beginning and the end of the whole iterative multiplications.

3 A Radix-4 Modular Multiplication Algorithm

We consider a modular multiplication in the former redundant representation shown in the previous section. Namely, the multiplicand X and the multiplier Y, as well as the product P are n-digit redundant binary numbers whose absolute values are less than $d_1 \cdot Q$, and $P \equiv X \times Y \pmod{Q}$ holds.

The algorithm is based on the following recursion equation.

$$P_j := 4 \cdot P_{j+1} + \hat{y}_j \cdot X - 4 \cdot c_j \cdot Q$$

Initially, we set $P_{\lfloor n/2 \rfloor + 1}$ to 0. $P_{-1}/4$ is the product.

$\hat{y}_j$ is the j-th digit of the recoded multiplier $\hat{Y}$. We recode the multiplier Y to a $(\lfloor n/2 \rfloor + 1)$-digit radix-4 signed-digit number $\hat{Y} = [\hat{y}_{\lfloor n/2 \rfloor}...\hat{y}_0]$ ($\hat{y}_j \in \{\bar{2}, \bar{1}, 0, 1, 2\}$) which has the same value as Y [7,10]. ($\bar{2}$ denotes -2.) Table 1 shows a recoding rule of the multiplier. Each $\hat{y}_j$ depends on only five digits of Y, i.e., y_{2j+1}, y_{2j}, y_{2j-1}, y_{2j-2}, and y_{2j-3}. We let $\hat{y}_{-1}$ be 0, by regarding y_j as 0 for $j < 0$. We can obtain $\hat{y}_j \cdot X$ by negating and/or shifting X.

In the calculation, we represent each partial product P_j in the latter redundant representation shown in the

Table 1: A recoding rule of the multiplier

(a) Stage 1

$y_{2j+1}\backslash y_{2j}$	yu_{j+1}, yt_j		
	$\bar{1}$	0	1
$\bar{1}$	$\bar{1}$,1	*0,$\bar{2}$/$\bar{1}$,2	0,$\bar{1}$
0	0,$\bar{1}$	0,0	0,1
1	0,1	*1,$\bar{2}$/0,2	1,$\bar{1}$

* : y_{2j-1} is non-negative. / Otherwise.

(b) Stage 2

$yt_j\backslash yu_j$	$\hat{y}_j$		
	$\bar{1}$	0	1
$\bar{2}$	×	$\bar{2}$	$\bar{1}$
$\bar{1}$	$\bar{2}$	$\bar{1}$	0
0	$\bar{1}$	0	1
1	0	1	2
2	1	2	×

×: Never occurs.

previous section. Namely, we represent P_j by an $(n+2)$-digit redundant binary number which satisfies $-d_2 \cdot Q < P_j < d_2 \cdot Q$. We select c_j from $\{\bar{2}, \bar{1}, 0, 1, 2\}$, by comparing $4 \cdot P_{j+1} + \hat{y}_j \cdot X$ with $\pm 2 \cdot Q$ and $\pm 6 \cdot Q$ down to the $(n-4)$th position. Figure 1 shows the Robertson's diagram for the algorithm. We can obtain $-4 \cdot c_j \cdot Q$ by complementing and/or shifting Q. We perform the additions for the recursion equation in the redundant binary representation without carry propagation.

The algorithm is as follows.

Algorithm [MODMUL]

(Inputs)

Modulus Q

(an n-bit binary number, $2^{n-1} \leq Q < 2^n$)

Multiplicand X

(an n-digit redundant binary number, $-d_1 \cdot Q < X < d_1 \cdot Q$)

Multiplier Y

(an n-digit redundant binary number, $-d_1 \cdot Q < Y < d_1 \cdot Q$)

(Output)

Product P

(an n-digit redundant binary number, $-d_1 \cdot Q < P < d_1 \cdot Q$, $P \equiv X \times Y \pmod{Q}$)

(Algorithm)

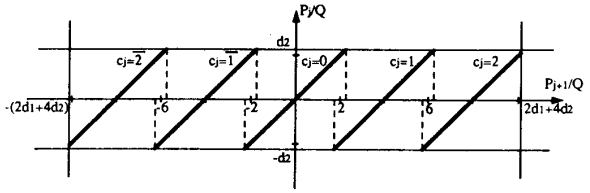

Figure 1: Robertson's diagram for [MODMUL]

Step 1: $P_{\lfloor n/2 \rfloor + 1} := 0$

Step 2: for $j := \lfloor n/2 \rfloor$ down to -1 do

begin

Calculate $\hat{y}_j$

$R_j := 4 \cdot P_{j+1} + \hat{y}_j \cdot X$

(redundant binary addition)

$c_j :=$

$$\begin{cases} \bar{2} & if \quad T(R_j) \leq -T(6 \cdot Q) \\ \bar{1} & if \quad -T(6 \cdot Q) < T(R_j) \leq -T(2 \cdot Q) \\ 0 & if \quad -T(2 \cdot Q) < T(R_j) < T(2 \cdot Q) \\ 1 & if \quad T(2 \cdot Q) \leq T(R_j) < T(6 \cdot Q) \\ 2 & if \quad T(R_j) \geq T(6 \cdot Q) \end{cases}$$

$P_j := R_j - 4 \cdot c_j \cdot Q$

(redundant binary addition)

end

Step 3: $P := P_{-1}/4$ □

R_j is an $(n+4)$-digit redundant binary number, because of its value. Table 2 shows a computation rule for this addition. (sx_i is the i-th digit of $\hat{y}_j \cdot X$ and is $\bar{x}_{i-1}$ or $\bar{x}_i$ or 0 or x_i or x_{i-1} accordingly as $\hat{y}_j$ is $\bar{2}$ or $\bar{1}$ or 0 or 1 or 2. ($\bar{x}_i$ is 1 or 0 or $\bar{1}$ accordingly as x_i is $\bar{1}$ or 0 or 1.) We need a special computation rule at the most significant two positions, in order to let R_j be an $(n+4)$-digit number.)

$T(R_j)$ is the most significant 8 digits of R_j. $T(2 \cdot Q)$ is the most significant 5 bits of Q. We can calculate $T(6 \cdot Q)$ from the most significant 6 digits of Q so that $|6 \cdot Q - T(6 \cdot Q)| < 2^{n-4}$. (We can either calculate it beforehand and store it or calculate it at each iteration step.) In the determination of c_j, each boundary can be included in either of the corresponding regions.

Since Q is a binary number, the second redundant binary addition for obtaining P_j is easier than the first one. Table 3 shows a computation rule for this addition. (sq_i is the i-th digit of $-4 \cdot c_j \cdot Q$ and is q_{i-3} or q_{i-2} or 0 or q'_{i-2} or q'_{i-3} accordingly as c_j is $\bar{2}$ or $\bar{1}$ or 0 or 1 or 2. (q'_i is 1 or 0 accordingly as q_i is 0 or 1.) $sq_{n+3}sq_{n+2}$ is 01 or 00 or 00 or $\bar{1}1$ or $\bar{1}0$ according to c_j. We need a bit complicated rule at the most significant four positions, in order to make P_j an $(n+2)$-digit number.)

Table 2: A computation rule for the first addition

(a) Stage 1

$p_{n+1} \backslash p_n$	r_{n+3}, rt_{n+2}		
	$\bar{1}$	0	1
$\bar{1}$	$*\bar{1},\bar{1}/\times$	$\bar{1}$,0	$*0,\bar{1}/\bar{1},1$
0	$*0,\bar{1}/\bar{1},1$	0,0	$*1,\bar{1}/0,1$
1	$*1,\bar{1}/0,1$	1,0	$*\times/1,1$

* : p_{n-1} is non-negative. / Otherwise.
× : Never occurs.

∘ for $0 \leq i \leq n+1$

$p_{i-2} \backslash sx_i$	ru_{i+1}, rt_i		
	$\bar{1}$	0	1
$\bar{1}$	$\bar{1}$,0	$*0,\bar{1}/\bar{1},1$	0,0
0	$*0,\bar{1}/\bar{1},1$	0,0	$*1,\bar{1}/0,1$
1	0,0	$*1,\bar{1}/0,1$	1,0

* : Both p_{i-3} and sx_{i-1} are non-negative. / Otherwise.

(b) Stage 2

∘ for $0 \leq i \leq n+2$

$rt_i \backslash ru_i$	r_i		
	$\bar{1}$	0	1
$\bar{1}$	×	$\bar{1}$	0
0	$\bar{1}$	0	1
1	0	1	×

$(ru_0 = 0)$
× : Never occurs.

The most significant n digits of P_{-1} is the product. (Note that P_{-1} is an $(n+2)$-digit redundant binary number and its least significant two digits are 0.)

Figure 2 shows an example of a modular multiplication according to the algorithm [MODMUL]. $\lfloor n/2 \rfloor + 2$ clock cycles are required to carry out a modular multiplication excluding the I/O.

In the rest of this section, we show the correctness of the algorithm [MODMUL].

First, we show that $P_j \equiv X \times \hat{Y}_j \pmod{Q}$ holds for all j's ($\lfloor n/2 \rfloor + 1 \geq j \geq -1$), where $\hat{Y}_j$ is the most significant $\lfloor n/2 \rfloor - j + 1$ digits of $\hat{Y}$, i.e., $\sum_{i=j}^{\lfloor n/2 \rfloor} \hat{y}_i \cdot 4^{i-j}$. Note that $\hat{y}_{-1}$ is 0 and that $\hat{Y}_{-1} = 4 \cdot \hat{Y}$. We can prove this fact by induction on j.

When $j = \lfloor n/2 \rfloor + 1$, $P_{\lfloor n/2 \rfloor + 1} = 0$ and $\hat{Y}_{\lfloor n/2 \rfloor + 1} = 0$. Hence, the equation holds.

Assume that $P_{j+1} \equiv X \times \hat{Y}_{j+1} \pmod{Q}$ holds. Since

Table 3: A computation rule for the second addition

(a) Stage 1

∘ for $0 \leq i \leq n+1$

$r_i \backslash sq_i$	pu_{i+1}, pt_i	
	0	1
$\bar{1}$	0,$\bar{1}$	0,0
0	0,0	1,$\bar{1}$
1	1,$\bar{1}$	1,0

(b) Stage 2

∘$pv := 2(r_{n+3} + sq_{n+3}) + (r_{n+2} + sq_{n+2}) + pu_{n+2}$
(pv must be $\bar{1}$ or 0 or 1.)

∘ for $0 \leq i \leq n+1$

$pt_i \backslash pu_i$	$\dot{p}_i$	
	0	1
$\bar{1}$	$\bar{1}$	0
0	0	1

($pu_0 = 0$, if c_j is non-negative. $pu_0 = 1$, otherwise.)

(c) Stage 3

$pv \backslash \dot{p}_{n+1}\dot{p}_n$	p_{n+1}, p_n				
	$\bar{1}$-	0$\bar{1}$	00	01	1-
$\bar{1}$	×	×	×	$\bar{1},\bar{1}$	$\bar{1},\dot{p}_n$
0	$\dot{p}_{n+1},\dot{p}_n$				
1	$1,\dot{p}_n$	1,1	×	×	×

× : Never occurs.

∘ for $0 \leq i \leq n-1$ $\quad p_i := \dot{p}_i$

$P_j = 4 \cdot P_{j+1} + \hat{y}_j \cdot X - 4 \cdot c_j \cdot Q$, $P_j \equiv 4 \cdot X \times \hat{Y}_{j+1} + \hat{y}_j \cdot X \pmod{Q}$. Hence, $P_j \equiv X \times \hat{Y}_j \pmod{Q}$ holds, and the fact has been proved.

Next, we show that $-d_2 \cdot Q < P_j < d_2 \cdot Q$ holds for all j's. Again, we can prove this fact by induction on j. Recall that $\frac{9}{4} \leq d_2 \leq \frac{55}{24}$, that $2 \cdot d_1 + 3 \cdot d_2 \leq 8$, and that $Q \geq 2^{n-1}$.

When $j = \lfloor n/2 \rfloor + 1$, $P_{\lfloor n/2 \rfloor + 1} = 0$. Hence, the inequality holds.

Assume that $-d_2 \cdot Q < P_{j+1} < d_2 \cdot Q$ holds. Since $R_j = 4 \cdot P_{j+1} + \hat{y}_j \cdot X$, $-(4 \cdot d_2 \cdot +2 \cdot d_1) \cdot Q < R_j < (4 \cdot d_2 \cdot +2 \cdot d_1) \cdot Q$ holds. (Note that $-d_1 \cdot Q < X < d_1 \cdot Q$ and that $-2 \leq \hat{y}_j \leq 2$.) Now, we have to consider the following five cases.

(1) $T(R_j) \leq -T(6 \cdot Q)$

$-(4 \cdot d_2 + 2 \cdot d_1) \cdot Q < R_j < -6 \cdot Q + 2 \cdot 2^{n-4}$,

Q = 100101011 (299)
X = 10$\bar{1}\bar{1}$00101 (165)
Y = $\bar{1}$01100001 ($-159 \equiv 140 \pmod{299}$) $\Rightarrow \hat{Y}$ = $\bar{1}2\bar{2}01$

$4P_5$		0	0	0	0	0	0	0	0	0	0	0	0	0
$(\hat{y}_4 = \bar{1})$	+				0	$\bar{1}$	0	1	1	0	0	$\bar{1}$	0	$\bar{1}$
R_4		0	0	0	0	$\bar{1}$	1	0	$\bar{1}$	0	0	$\bar{1}$	0	$\bar{1}$
$(c_4 = 0)$	+	0	0	0	0	0	0	0	0	0	0	0	0	0(0)
P_4					0	0	0	$\bar{1}$	0	$\bar{1}$	0	0	$\bar{1}$	0	$\bar{1}$
$4P_4$		0	0	0	$\bar{1}$	0	$\bar{1}$	0	0	$\bar{1}$	0	$\bar{1}$	0	0
$(\hat{y}_3 = 2)$	+				1	0	$\bar{1}$	$\bar{1}$	0	0	1	0	1	0
R_3		0	0	0	0	$\bar{1}$	0	$\bar{1}$	0	$\bar{1}$	1	0	$\bar{1}$	0
$(c_3 = 0)$	+	0	0	0	0	0	0	0	0	0	0	0	0	0(0)
P_3					0	0	$\bar{1}$	0	$\bar{1}$	0	0	$\bar{1}$	0	$\bar{1}$	0
$4P_3$		0	0	$\bar{1}$	0	$\bar{1}$	0	0	$\bar{1}$	0	$\bar{1}$	0	0	0
$(\hat{y}_2 = \bar{2})$	+				$\bar{1}$	0	1	1	0	0	$\bar{1}$	0	$\bar{1}$	0
R_2		0	$\bar{1}$	0	1	0	$\bar{1}$	1	$\bar{1}$	$\bar{1}$	0	0	$\bar{1}$	0
$(c_2 = \bar{1})$	+	0	0	1	0	0	1	0	1	0	1	1	0	0(0)
P_2					0	$\bar{1}$	0	1	$\bar{1}$	0	0	0	$\bar{1}$	$\bar{1}$	0
$4P_2$		0	$\bar{1}$	0	1	$\bar{1}$	0	0	0	$\bar{1}$	$\bar{1}$	0	0	0
$(\hat{y}_1 = 0)$	+				0	0	0	0	0	0	0	0	0	0
R_1		0	$\bar{1}$	0	1	$\bar{1}$	0	0	$\bar{1}$	1	$\bar{1}$	0	0	0
$(c_1 = \bar{2})$	+	0	1	0	0	1	0	1	0	1	1	0	0	0(0)
P_1					1	$\bar{1}$	0	1	$\bar{1}$	0	0	0	0	0	0
$4P_1$		1	$\bar{1}$	0	1	$\bar{1}$	0	0	0	0	0	0	0	0
$(\hat{y}_0 = 1)$	+				0	1	0	$\bar{1}$	$\bar{1}$	0	0	1	0	1
R_0		1	$\bar{1}$	0	1	0	$\bar{1}$	1	$\bar{1}$	0	1	$\bar{1}$	1	$\bar{1}$
$(c_0 = 2)$	+	$\bar{1}$	0	1	1	0	1	0	1	0	0	1	1	1(1)
P_0					0	0	0	1	$\bar{1}$	0	1	$\bar{1}$	1	0	1
$4P_0$		0	0	0	1	$\bar{1}$	0	1	$\bar{1}$	1	0	1	0	0
$(\hat{y}_{-1} = 0)$	+				0	0	0	0	0	0	0	0	0	0
R_{-1}		0	0	0	1	$\bar{1}$	0	1	0	$\bar{1}$	1	$\bar{1}$	0	0
$(c_{-1} = 0)$	+	0	0	0	0	0	0	0	0	0	0	0	0	0(0)
P_{-1}					1	$\bar{1}$	$\bar{1}$	1	$\bar{1}$	0	0	$\bar{1}$	$\bar{1}$	0	0

P = 1$\bar{1}\bar{1}$1$\bar{1}$00$\bar{1}\bar{1}$ (77)

Figure 2: An example of modular multiplication

and hence, $-(8+d_2)\cdot Q < R_j < -(8-d_2)\cdot Q$.

Since $c_j = \bar{2}$, $P_j = R_j + 8\cdot Q$.

Therefore, $-d_2\cdot Q < P_j < d_2\cdot Q$ holds.

(2) $-T(6\cdot Q) < T(R_j) \le -T(2\cdot Q)$

$-6\cdot Q - 2\cdot 2^{n-4} < R_j < -2\cdot Q + 2\cdot 2^{n-4}$,

and hence, $-(d_2+4)\cdot Q < R_j < -(4-d_2)\cdot Q$.

Since $c_j = \bar{1}$, $P_j = R_j + 4\cdot Q$.

Therefore, $-d_2\cdot Q < P_j < d_2\cdot Q$ holds.

(3) $-T(2\cdot Q) < T(R_j) < T(2\cdot Q)$

$-2\cdot Q - 2^{n-4} < R_j < 2\cdot Q + 2^{n-4}$,

and hence, $-d_2\cdot Q < R_j < d_2\cdot Q$.

Since $c_j = 0$, $P_j = R_j$.

Therefore, $-d_2\cdot Q < P_j < d_2\cdot Q$ holds.

(4) $T(2\cdot Q) \le T(R_j) < T(6\cdot Q)$

$-d_2\cdot Q < P_j < d_2\cdot Q$ holds,

from a similar discussion to (2).

(5) $T(R_j) \ge T(6\cdot Q)$

$-d_2\cdot Q < P_j < d_2\cdot Q$ holds,

from a similar discussion to (1).

Thus, $-d_2\cdot Q < P_j < d_2\cdot Q$ holds in any case, and the fact has been proved.

From the above facts, we get $P_{-1} \equiv 4\cdot(X\times Y) \pmod{Q}$ and $-d_2\cdot Q < P_{-1} < d_2\cdot Q$. Since P_{-1} is obtained by the calculation of $4\cdot P_0 + 0 - 4\cdot c_{-1}\cdot Q$, its least significant two digits are 0. Hence, we can calculate $P := P_{-1}/4$ by just taking the most significant n digits of P_{-1}. Then, P satisfies $P \equiv X\times Y \pmod{Q}$ and $-\frac{1}{4}\cdot d_2\cdot Q < P < \frac{1}{4}\cdot d_2\cdot Q$. Since $4\cdot d_1 \ge d_2$, $-d_1\cdot Q < P < d_1\cdot Q$ holds.

Thus, the obtained P satisfies $P \equiv X\times Y \pmod{Q}$ and $-d_1\cdot Q < P < d_1\cdot Q$. Therefore, the algorithm [MODMUL] is correct.

4 A Serial-Parallel Modular Multiplier

A serial-parallel modular multiplier based on the algorithm [MODMUL] has a regular cellular array structure with a bit slice feature suitable for VLSI implementation. Figure 3 shows a block diagram of the multiplier. Here, we assume that the multiplier performs one iteration step in each clock cycle. The multiplier consists of four registers and a combinational circuit part. The registers are for storing redundant binary numbers X, Y, and P_j, and a binary number Q. The register for Y is a shift register, and Y is shifted with two positions to the left in each clock cycle. When n is odd, initially, we have to attach 0 to Y from the left (to the most significant position). The combinational circuit part consists of a $\hat{y}_j$ calculating circuit, a c_j selecting circuit, and circuits for the slices. The circuit for the slices are composed of a negate-shift-and-select circuit for generating $\hat{y}_j\cdot X$, a redundant binary adder for calculating R_j, a complement-shift-and-select circuit for generating $-4\cdot c_j\cdot Q$, and a simpler redundant binary adder for calculating P_j. The depth of the combinational circuit part is a constant independent of n. The amount of hardware of the whole multiplier is proportional to n.

Figure 4 illustrates a block diagram of the combinational circuit part of the slice for a middle position (the region enclosed with the dashed line in Figure 3). It consists of four basic cells, i.e., an XSEL for the generation of $\hat{y}_j\cdot X$, an RBA1 for the redundant binary addition for R_j, a QSEL for the generation of $-4\cdot c_j\cdot Q$, and an RBA2 for the simpler redundant binary addition for P_j. According to our CMOS logic designs, the depth and the gate count of these cells are 3 and 5 (28 transistors), 4 and 7 (40 transistors), 2 and 2 (20 transistors), and 3 and 5 (24 transistors), respectively. We can shorten the clock period by performing the calculation of $\hat{y}_j$ in the previous cycle concurrently with the addition for

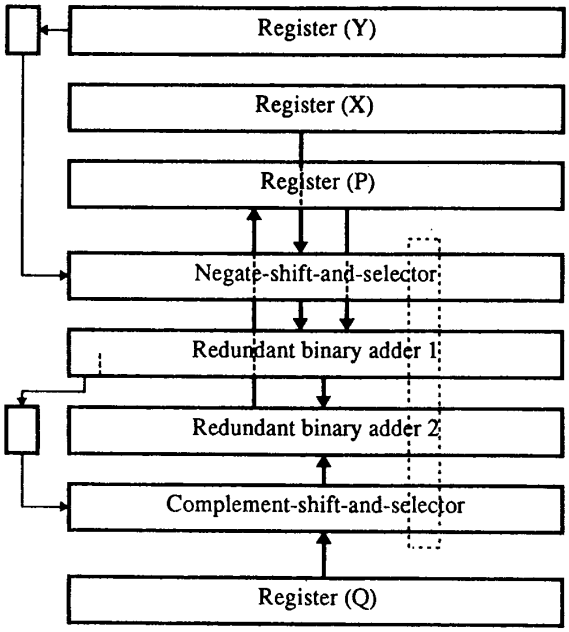

Figure 3: A block diagram of a modular multiplier

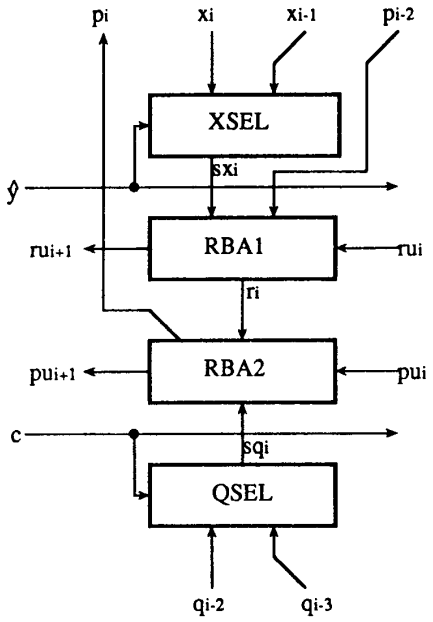

Figure 4: A block diagram of a slice of the multiplier

calculating P_{j+1}. When we prepare $T(6 \cdot Q)$ beforehand, the depth of the combinational circuit part becomes 19. (However, in practice, we need buffers for driving long lines for $\hat{y}_j$ and c_j.) The gate count of a slice is 19 (112 transistors). The gate count of the combinational circuit part of the whole multiplier is about 19n (112n transistors) for a large n. The total number of bits for the registers is about 7n. A 512-digit modular multiplier will consist of about 100,000 transistors including the buffers. It is expected to operate with about 33MHz clock and to carry out 512-digit multiplication in about $7.8\mu sec$ (excluding I/O), when it is fabricated with $2\mu m$ CMOS technology. It may operate with faster clock, if it is fabricated with today's advanced technologies.

Once a multiplier is fabricated, the length of its registers, adders and etc. is fixed. Assume that the length of the register for the modulus is n'-bit. The multiplier can perform any modular multiplication with a modulus which is shorter than or equal to n'-bit. When the modulus is shorter than n'-bit, we put the operands in the registers from the left side and fill the rest with 0's. The number of required clock cycles for a multiplication depends on the length of the operands but does not depend on the length of the registers.

As the advance of VLSI technologies, we may realize a faster multiplier which performs more than one iteration steps in each clock cycle.

5 Application to RSA Encryption/ Decryption

The proposed multiplier is efficient especially in applications where modular multiplications are performed iteratively. In such applications, we keep intermediate results in the redundant representation and convert only the final result to the ordinary representation.

As an example of such an application, we consider RSA encryption/decryption. In RSA encryption, we calculate $cipher := (message)^e \bmod Q$, where e is an encryption key. Decryption is carried out in the same way. A simple way to perform modular exponentiation is to repeat squaring and multiplication as shown below [1]. (We assume e is represented by a k-bit binary number $[1e_{k-2}...e_0]$.)

$C := M$
for $i := k-2$ down to 0 do
  begin
    $C := C^2 \bmod Q$
    if $e_i = 1$ then $C := C \cdot M \bmod Q$
  end

Here, M is the message and the calculated C is the cipher.

$2k-2$ multiplications are required in the worst case. (About $1.5k$ multiplications are required in the average.) We keep C in the redundant representation (the

former one shown in Section 2) during the calculation, and convert only the final C to the ordinary binary representation. We can convert the input into the redundant representation in the same way as the calculation of P from P_0 in the algorithm [MODMUL]. Therefore, we do not need additional hardware for this conversion. On the other hand, we need a carry propagate adder for the conversion of the result from the redundant representation to the ordinary binary.

In encryption of a message block sequence, we can perform the exponentiation concurrently with the input of the next message block, and the conversion and the output of the calculated cipher block. Namely, pipeline processing for continuous blocks is possible. The processing speed is dominated by the exponentiation speed. When the size of the message block (the length of the modulus) is 512-bit and the length of the encryption key is also 512-bit, the throughput for encryption with 33MHz clock will be at least 65kbps, which is more than 6 times as large as that of the fastest actual RSA chip listed in [2]. Note that we have used a very simple exponentiation algorithm and have assumed the worst case. (Recently, Shand et al reported a very high performance RSA encryption/decryption system based on an efficient modular multiplication algorithm cooperated with the Chinese remaindering [11]. Our multiplier may have comparable performance when it is cooperated with the Chinese remaindering.)

6 Concluding Remarks

We have proposed a radix-4 modular multiplication hardware algorithm, which is efficient especially in applications where modular multiplications are performed iteratively. In the algorithm, we represent numbers in redundant representations, and perform modular additions without carry propagation. We use a more redundant representation for representing intermediate results than for the operands. This enables us to reduce the number of addition/subtractions required in the "division-during-multiplication" method. This technique might be useful for design of new efficient arithmetic algorithms with a redundant representation.

A serial-parallel modular multiplier based on the proposed algorithm has a regular cellular array structure with a bit slice feature suitable for VLSI implementation. It seems easy to fabricate an RSA encryption/decryption circuit based on the proposed multiplier on a VLSI chip using today's technology, which is expected to have a throughput of several times as large as that of the fastest actual RSA chip.

Acknowledgements

This work was done while the author was staying at Computer Systems Laboratory, Stanford University. The author would like to thank Professor Shuzo Yajima of Kyoto University who gave the author the chance to stay at Stanford. The author also would like to appreciate the hospitality of Professor Michael J. Flynn of Stanford University.

References

[1] R.L.Rivest, A.Shamir and L.Adleman: "A method for obtaining digital signatures and public-key cryptosystems", Commun. ACM, vol.21, no.2, pp.120-126, Feb. 1978.

[2] E.F.Brickell: "A survey of hardware implementations of RSA", Lecture Notes in Computer Science, vol.435, G.Brassard Ed., 'Advances in Cryptology - CRYPTO'89 Proceedings', pp.368-370, Springer-Verlag, 1990.

[3] E.F.Brickell: "A fast modular multiplication algorithm with application to two key cryptography", D.Chaum et al Eds., 'Advances in Cryptology, Proceedings of CRYPTO 82', pp.51-60, Plenum Press, New York, 1983.

[4] H.Morita: "A fast modular-multiplication algorithm based on a higher radix", Lecture Notes in Computer Science, vol.435, G.Brassard Ed., 'Advances in Cryptology - CRYPTO'89 Proceedings', pp.387-399, Springer-Verlag, 1990.

[5] A.Vandemeulebroecke, E.Vanzieleghem,T.Denayer and P.G.A.Jespers: "A new carry-free division algorithm and its application to a single-chip 1024-b RSA processor", IEEE J. Solid-State Circuits, vol.25, no.3, June 1990.

[6] F.P.Preparata and J.E.Vuillemin: "Practical Cellular Dividers", IEEE Trans. Comput., vol.C-39, no.5, pp.605-614, May 1990.

[7] N.Takagi and S.Yajima: "A modular multiplication hardware algorithm with a redundant representation", Report of Technical Group on Computation, Institute of the Electronics, Information and Communication Engineers of Japan, COMP89-103, Jan. 1990. (also to appear in IEEE Trans. Comput.)

[8] A.Avizienis: "Signed-digit number representations for fast parallel arithmetic", IRE Trans. Electron. Comput., vol.EC-10, no.3, pp.389-400, Sep. 1961.

[9] N.Takagi, H.Yasuura and S.Yajima: "High-speed VLSI multiplication algorithm with a redundant binary addition tree", IEEE Trans. Comput., vol.C-34, no.9, pp.789-796, Sep. 1985.

[10] N.Takagi: 'Studies on hardware algorithms for arithmetic operations with a redundant binary representation', Doctoral dissertation, Dept. Info. Sci., Kyoto Univ., Aug. 1987.

[11] M.Shand, P.Bertin and J.E.Vuillemin: "Hardware speedups in long integer multiplication", Proc. 2nd Annual ACM Symp. on Parallel Algorithms and Architectures - SPAA '90, pp.138-145, July 1990.

Appendix: Derivation of the Algorithm

Assume that the multiplicand, the multiplier and the product are represented in n_1-digit redundant binary numbers with the value larger than $-d_1 \cdot Q$ and smaller than $d_1 \cdot Q$. $2 \cdot d_1$ must be larger than or equal to 1. Hence, $d_1 \geq \frac{1}{2}$. Since $2^{n-1} \leq Q < 2^n$, $n_1 = n + \lceil \log_2 d_1 \rceil$.

In general, a radix-4 modular multiplication algorithm is based on the following recursion equation.

$$P_j := 4 \cdot P_{j+1} + \hat{y}_j \cdot X - c1_j \cdot Q$$

Initially, we set $P_{\lfloor n_1/2 \rfloor + 1}$ to 0. P_0 is the product. $\hat{y}_j$ ($\in \{\bar{2}, \bar{1}, 0, 1, 2\}$) is the j-th digit of the recoded multiplier $\hat{Y}$. $-c1_j \cdot Q$ is for the residue calculation.

We let $c1_j$ be $2^{r1} \cdot c_j$ ($r1 \geq 0$) and select c_j from $\{\bar{2}, \bar{1}, 0, 1, 2\}$, so that we can obtain $c1_j \cdot Q$ by complementing and/or shifting Q. Namely, we rewrite the recursion equation as follows.

$$P_j := 4 \cdot P_{j+1} + \hat{y}_j \cdot X - 2^{r1} \cdot c_j \cdot Q$$

In the calculation, we represent each partial product P_j by an n_2-digit redundant binary number which satisfies $-d_2 \cdot Q < P_j < d_2 \cdot Q$. $n_2 = n + \lceil \log_2 d_2 \rceil$. In order that P_j stays in this range, the following inequality must hold.

$$4 \cdot d_2 + 2 \cdot d_1 - 2^{r1} \cdot 2 \leq d_2 \quad (1)$$

Furthermore, in order that we can determine c_j by evaluating only several digits of $4 \cdot P_{j+1} + \hat{y}_j \cdot X$ and Q, the following inequality must hold. (There must exist an overlap between each contiguous regions in the Robertson's diagram shown in Figure 1.)

$$2^{r1} - d_2 < d_2 \quad (2)$$

According to the above calculation, we obtain P_0 which satisfies $-d_2 \cdot Q < P_0 < d_2 \cdot Q$. We have to convert it to P which satisfies $-d_1 \cdot Q < P < d_1 \cdot Q$ and $P \equiv P_0 \pmod{Q}$. We can perform this conversion by the following calculation.

$$P := P_0 - c2 \cdot Q$$

As in the case of $c1_j$, we let $c2$ be $2^{r2} \cdot c$ ($r2 \geq 0$) and select c from $\{\bar{2}, \bar{1}, 0, 1, 2\}$, so that we can obtain $c2 \cdot Q$ by complementing and/or shifting Q. The following two inequalities must hold.

$$d_2 - 2^{r2} \cdot 2 \leq d_1 \quad (3)$$

$$2^{r2} - d_1 < d_1 \quad (4)$$

Inequality (4) is the condition that we can determine c by evaluating only several digits of P_0 and Q.

From (2), $d_2 > 2^{r1-1}$. From (4), $d_1 > \frac{1}{2}$. Substituting these to (1), we get $r1 > 1$. Since the increment of $r1$ causes the increment of n_2, we should select $r1$ as small as possible. Here, we select 2 as $r1$. Then, from (1) and (2), we get $3 \cdot d_2 + 2 \cdot d_1 \leq 8$ and $d_2 > 2$, respectively. From these, $d_1 < 1$. From this and (4), we get $r2 = 0$.

Substituting $r1 = 2$ and $r2 = 0$ to the recursion equation and the equation for the conversion, we get the following equations.

$$P_j := 4 \cdot P_{j+1} + \hat{y}_j \cdot X - 4 \cdot c_j \cdot Q$$

$$P := P_0 - c \cdot Q$$

Here, c_j and c are selected from $\{\bar{2}, \bar{1}, 0, 1, 2\}$.

We can rewrite the second equation as follows.

$$4 \cdot P := 4 \cdot P_0 - 4 \cdot c \cdot Q$$

When $d_2 \leq 4 \cdot d_1$, we can perform the final conversion using the circuit for the iteration step. Then, we can combine the above equations and get the following equations for our algorithm.

$$P_j := 4 \cdot P_{j+1} + \hat{y}_j \cdot X - 4 \cdot c_j \cdot Q$$

$$P := P_{-1}/4$$

Substituting $r1 = 2$ and $r2 = 0$ to the inequalities, we get $\frac{1}{2} < d_1 < \frac{7}{12}$, $2 < d_2 < \frac{7}{3}$, $2 \cdot d_1 + 3 \cdot d_2 \leq 8$, and $4 \cdot d_1 \geq d_2$. We can select d_1 and d_2 so that they satisfy these conditions. In any case, n_1 and n_2 become n and $n+2$, respectively. The larger d_2 is, the fewer the number of digits to be looked into in the determination of c_j is.

Now, we consider how many digits we should look into for determining c_j. Assume that we compare $4 \cdot P_{j+1} + \hat{y}_j \cdot X$ with $\pm 2 \cdot Q$ and $\pm 6 \cdot Q$ down to the k-th position. We should select k as large as possible to reduce the hardware. k must satisfy the following condition.

$$2 \cdot 2^k \leq (d_2 - 2) \cdot Q$$

Since Q can be 2^{n-1}, $k \leq n - 2 + \log_2(d_2 - 2)$ must hold. Since $d_2 < \frac{7}{3}$, k is at most $n - 4$. In reverse, k is $n - 4$ when $d_2 \geq \frac{9}{4}$. Hence, we get the condition $d_2 \geq \frac{9}{4}$.

Putting it all together, we get $\frac{9}{16} \leq d_1 \leq \frac{55}{96}$, $\frac{9}{4} \leq d_2 \leq \frac{55}{24}$, $2 \cdot d_1 + 3 \cdot d_2 \leq 8$, and $4 \cdot d_1 \geq d_2$.