Modular Multiplication Hardware Algorithms with a Redundant Representation and Their Application to RSA Cryptosystem

Naofumi Takagi and Shuzo Yajima

Abstract—**A radix-2 and a radix-4 modular multiplication hardware algorithm are proposed. Numbers are represented in a redundant representation and modular additions are performed without carry propagation. Serial–parallel modular multipliers based on them have a regular cellular array structure with a bit slice feature suitable for VLSI implementation. They are efficient especially in applications, such as RSA cryptosystem, where modular multiplications are performed iteratively.**

Index Terms— **Computer arithmetic, cryptosystem, hardware algorithm, modular multiplication, redundant representation, VLSI.**

I. Introduction

In encryption/decryption in RSA public-key cryptosystem [1], modular multiplications with a large modulus (longer than 500-bit) are iteratively performed. Design of a fast algorithm for modular multiplication with a large modulus is the key to developing a high performance RSA encryption/decryption circuit. Various algorithms for modular multiplication have been proposed and some of them have been realized on VLSI [2]. One of the most attractive methods is one in which each subtraction step for division is embedded in the repeated multiply-addition, because of its suitability for VLSI implementation [3], [4]. The key point to increasing the computation speed of such multiplication is to perform modular additions with a large modulus fast.

In this paper, we propose a radix-2 and a radix-4 modular multiplication algorithm suitable for VLSI implementation. In the algorithms, we represent numbers in a redundant representation based on the redundant binary representation, and perform modular additions in the redundant representation without carry propagation [5], [6]. Serial–parallel modular multipliers based on the algorithms have a regular cellular array structure with a bit slice feature suitable for VLSI implementation. The radix-2 multiplier requires about n clock cycles to perform an n-bit multiplication, while the radix-4 one requires about $n/2$ clock cycles. The depth of combinational circuit part of the multipliers is a constant independent of n, and therefore, they can operate with a fast clock. The amount of hardware of these is proportional to n. Through SPICE simulation, we estimated that a 512-bit radix-2 modular multiplier will operate with 33 MHz clock, when it will be fabricated using 2 μm CMOS technology. A 512-bit radix-4 multiplier will operate with 25 MHz clock.

The proposed multipliers are efficient especially in applications where multiplications are performed iteratively. A 512-bit RSA encryption/decryption circuit based on the proposed radix-4 multiplier is expected to have a throughput of more than 50 kb/s at 25 MHz clock.

In the next section, we describe a redundant representation for a residue class and a fast modular addition method. We propose fast modular multiplication algorithms and serial–parallel modular multipliers based on them, in Sections III and IV, respectively. In Section V, we discuss application of the multipliers to RSA encryption/decryption.

Manuscript received November 28, 1989; revised November 11, 1990.

The authors are with the Department of Information Science, Kyoto University, Kyoto 606, Japan.

IEEE Log Number 9103039.

II. A Redundant Representation for a Residue Class and a Modular Addition Method

A. A Redundant Representation for a Residue Class

We consider a redundant representation for a residue class $Z_Q = \{0, 1, \cdots, Q-1\}$ where $2^{n-1} \leq Q < 2^n$. The redundant representation is based on the redundant binary representation, i.e., the radix-2 signed-digit representation [5], [6].

The redundant binary representation has a fixed radix 2 and a digit set $\{\bar{1}, 0, 1\}$, where $\bar{1}$ denotes -1 [7]. An $n+1$-digit redundant binary number $A = [a_n a_{n-1} \cdots a_0]$ ($a_i \in \{\bar{1}, 0, 1\}$) has the value $\sum_{i=0}^{n} a_i \cdot 2^i$. Hereafter, we will use A to denote both a redundant representation and its value.

We represent each element α of the residue class Z_Q except 0 by an $n+1$-digit redundant binary number whose value is α or $\alpha - Q$. For both α and $\alpha - Q$, there are several redundant binary numbers that have these values. For example, when $Q = 6$ and $n = 3$, 4 ($\in Z_6$) has the following representations: redundant binary numbers with the value 4, i.e., $[0100]$ and $[1\bar{1}00]$, and those with the value -2 ($= 4 - 6$), i.e., $[00\bar{1}0]$, $[0\bar{1}10]$, and $[\bar{1}110]$. We represent 0 ($\in Z_Q$) by the $n+1$-digit redundant binary number with the value 0, i.e., $[00\ldots0]$.

The ordinary binary representation of α ($\in Z_Q$) is one of its redundant representations. Therefore, we need no computation for the conversion from the ordinary binary representation to the redundant one. On the other hand, we need a binary subtraction and a binary addition with carry propagation for the reverse conversion. First, we convert A to the $n+1$-bit two's complement binary number A^* which has the same value as A by performing an ordinary binary subtraction $A^+ - A^-$ where A^+ and A^- are $n+1$-bit unsigned binary numbers formed from the positive digits and the negative digits in A, respectively. Then, we add Q to A^*, if A^* is negative. (Since α is represented by an n-bit ordinary binary number, in the conversions, a digit "0" is attached to or deleted from the most significant position.)

When $A = [a_n a_{n-1} \cdots a_0]$ is a redundant representation of α, a redundant representation of $-\alpha \bmod Q$ is directly derived by changing the signs of all nonzero a_i's. Namely, $\bar{A} = [\bar{a}_n \bar{a}_{n-1} \ldots \bar{a}_0]$, where $\bar{a}_i$ is 1 or 0 or $\bar{1}$ accordingly as a_i is $\bar{1}$ or 0 or 1, is a representation of $-\alpha \bmod Q$.

B. A Modular Addition Method

We consider modular addition in the redundant representation. The augend A and the addend B are $n+1$-digit redundant binary numbers and satisfy $-Q < A < Q$ and $-Q < B < Q$, respectively. We calculate the sum S which is also an $n+1$-digit redundant binary number and satisfies $-Q < S < Q$ and $S \equiv A + B \pmod{Q}$.

We perform modular addition through two stages [5], [6].

The first stage is a redundant binary addition. We add A and B in the redundant binary representation without carry propagation, and obtain an $n+2$-digit redundant binary number $T = [t_{n+1} t_n t_{n-1} \cdots t_0]$. For the details of carry-propagation-free addition, see, e.g., [8]. T satisfies $T \equiv A + B \pmod{Q}$ and $-2Q < T < 2Q$.

The second stage is a correction stage to obtain an $n+1$-digit redundant binary number S which satisfies $S \equiv T \pmod{Q}$ and $-Q < S < Q$. First, we evaluate the value tv of the three most significant digits of T, i.e., $4t_{n+1} + 2t_n + t_{n-1}$. Then, we add Q or 0 or $-Q$ to T, accordingly as tv is negative or zero or positive. (Note that $-2Q < T < 0$ when $tv < 0$, that $-2^{n-1} < T < 2^{n-1}$ when $tv = 0$, and that $0 < T < 2Q$ when $tv > 0$.) We perform

TABLE I
A COMPUTATION RULE FOR THE SECOND STAGE OF MODULAR ADDITION (a) AT THE MOST SIGNIFICANT TWO POSITIONS (b) AT THE OTHER POSITIONS

		s_n		
t_{n+1} t_n	$\backslash$ t_{n-1}	$\bar{1}$	0	1
$\bar{1}$	0	–	$\bar{1}$	$\bar{1}$
$\bar{1}$	1	$\bar{1}$	0	0
0	$\bar{1}$	$\bar{1}$	0	0
0	0	0	0	0
0	1	0	0	1
1	$\bar{1}$	0	0	1
1	0	1	1	–

(a)

	u_i	
$tv \backslash q_i$	0	1
neg.	0	1
zero	0	0
pos.	1	0

	v_i, w_i	
$t_i \backslash u_i$	0	1
$\bar{1}$	$0, \bar{1}$	$0, 0$
0	$0, 0$	$1, \bar{1}$
1	$1, \bar{1}$	$1, 0$

	s_i	
$w_i \backslash v_{i-1}$	0	1
$\bar{1}$	$\bar{1}$	0
0	0	1

$v_{-1} = 1$ when t_v is positive. Otherwise, $v_{-1} = 0$.
(b)

augend A			0	0	1	0	$\bar{1}$	1	0	1	0	0	$\bar{1}$	0	(462)	
addend B	+		1	0	$\bar{1}$	$\bar{1}$	$\bar{1}$	0	1	$\bar{1}$	1	0	0	0	(1176)	stage 1
T		1	$\bar{1}$	0	$\bar{1}$	0	1	$\bar{1}$	1	1	$\bar{1}$	0	$\bar{1}$	0		
$-Q$	+		$\bar{1}$	0	1	1	0	1	0	0	1	1	1	1(1)		stage 2
sum S			0	0	1	0	$\bar{1}$	1	0	$\bar{1}$	1	$\bar{1}$	1	0	(438)	
augend A			0	0	1	0	$\bar{1}$	1	0	1	0	0	$\bar{1}$	0	(462)	
addend B	+		$\bar{1}$	0	1	1	1	0	$\bar{1}$	1	$\bar{1}$	0	0	0	(24)	stage 1
T		0	$\bar{1}$	1	0	1	0	1	0	0	$\bar{1}$	0	$\bar{1}$	0		
$+Q$	+		0	1	0	0	1	0	1	1	0	0	0	0(0)		stage 2
sum S			0	0	1	0	0	0	0	$\bar{1}$	$\bar{1}$	0	$\bar{1}$	0	(486)	

Fig. 1. Examples of modular addition ($Q = 1200$).

the addition in the redundant binary representation without carry propagation. When Q is $[1q_{n-2} \cdots q_0]$, the redundant binary number $[\bar{1}0q'_{n-2} \cdots q'_0]$ (q'_i is 1 or 0 accordingly as q_i is 0 or 1) has the value $-Q - 1$. Using this fact, we can make all addend digits except the most significant one nonnegative, i.e., 0 or 1. Hence, the addition in this stage is easier than that in the first stage [6]. Table I shows a computation rule for this addition. The addend digit, u_i ($\in \{0, 1\}$), is q_i or 0 or q'_i, accordingly as tv is negative or zero or positive. The intermediate carry, v_i ($\in \{0, 1\}$), and the intermediate sum digit, w_i ($\in \{\bar{1}, 0\}$), satisfy $2v_i + w_i = t_i + u_i$. We obtain the final sum digit, s_i, by adding w_i and v_{i-1} without generating a carry. We let v_{-1} be 1 when tv is positive, and be 0 otherwise. At the most significant two positions, when $tv > 0$, $s_n = 2t_{n+1} + t_n - 1 + v_{n-1}$, and otherwise, $s_n = 2t_{n+1} + t_n + v_{n-1}$.

Fig. 1 shows examples of modular addition with $Q = 1200$.

III. MODULAR MULTIPLICATION ALGORITHMS

We consider modular multiplication in the redundant representation. The multiplicand A and the multiplier B are $n + 1$-digit redundant binary numbers and satisfy $-Q < A < Q$ and $-Q < B < Q$, respectively. We calculate the product P which is also an $n + 1$-digit redundant binary number and satisfies $-Q < P < Q$ and $S \equiv A \times B \pmod{Q}$.

A. A Radix-2 Modular Multiplication Algorithm

First, we show a radix-2 modular multiplication algorithm. The algorithm is based on the following recursion equation.

$$P_j := (P_{j+1} \cdot 2 + A \cdot b_j) \bmod Q.$$

Initially, we set P_{n+1} to 0. b_j is the jth digit of the multiplier B. P_0 is the product.

We represent all intermediate results in the redundant representation, and perform all calculations in the redundant representation. The algorithm is as follows.

Algorithm [MODMUL2]

```
Step 1: $P_{n+1} := 0$
Step 2: for $j := n$ downto 0 do
        begin
            1) $X_j := P_{j+1} \cdot 2 \bmod Q$
            2) $Z_j := A \cdot b_j \bmod Q$
            3) $P_j := (X_j + Z_j) \bmod Q$
        end
Step 3: $P := P_0$                                    □
```

We represent all intermediate results P_j's, X_j's and Z_j's in the redundant representation. In Step 2 1), we perform a modular doubling. We first shift P_{j+1} with one position to the left and obtain an $n + 2$-digit redundant binary number T_j which satisfies $-2Q < T_j < 2Q$. Then, we apply the correction stage (the second stage) of the modular addition mentioned in Section II-B to T_j and obtain X_j. In 2), we let Z be $\bar{A}$ (a negation of A) or 0 or A, accordingly as b_j is $\bar{1}$ or 0 or 1. We can obtain $\bar{A}$ by the method mentioned at the end of Section II-A. 1) and 2) can be performed in parallel. In 3), we perform a modular addition. $n + 1$ iterations are required to perform an n-digit modular multiplication.

B. A Radix-4 Modular Multiplication Algorithm

We can reduce the number of iteration steps for a multiplication by the use of the radix-4 method. The radix-4 algorithm is based on the following recursion equation.

$$P_j := (P_{j+1} \cdot 4 + A \cdot \hat{b}_j) \bmod Q.$$

Initially, we set $P_{\lceil n/2 \rceil + 1}$ to 0. $\hat{b}_j$ is the jth digit of the recoded multiplier $\hat{B}$. P_0 is the product.

TABLE II
A RECODING RULE OF A MULTIPLIER

	c_j, d_j		
$b_{2j+1} \backslash b_{2j}$	$\bar{1}$	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	$0, 1$	$^*1, \bar{2}/0, 2$	$1, \bar{1}$

*: b_{2j-1} is nonnegative. / Otherwise.

	$\hat{b}_j$		
$d_j \backslash c_{j-1}$	$\bar{1}$	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	—

Here, we show an efficient multiplier recoding method [6]. We recode the multiplier B to a $\lceil n/2 \rceil + 1$-digit radix-4 signed-digit number $\hat{B} = [\hat{b}_{\lceil n/2 \rceil} \hat{b}_{\lceil n/2 \rceil - 1} \cdots \hat{b}_0]$ ($\hat{b}_j \in \{\bar{2}, \bar{1}, 0, 1, 2\}$) which has the same value as B. ($\bar{2}$ denotes -2.) The conversion is performed through two steps. In the first step, we determine c_j ($\in \{\bar{1}, 0, 1\}$) and d_j ($\in \{\bar{2}, \bar{1}, 0, 1, 2\}$) at each 2-digit group in B with satisfying $2b_{2j+1} + b_{2j} = 4c_j + d_j$. In the second step, we obtain $\hat{b}_j$ ($\in \{\bar{2}, \bar{1}, 0, 1, 2\}$) at each group by adding d_j and c_{j-1}. We determine c_j and d_j in the first step so that no carry generates in the second step, by looking into the upper digit of the next-lower group, b_{2j-1}. Table II shows a computation rule for the recoding. Each $\hat{b}_j$ depends on only five digits of B.

The radix-4 modular multiplication algorithm is as follows:

Algorithm [MODMUL4]

```
Step 1: $P_{\lceil n/2 \rceil + 1} := 0$
Step 2: for $j := \lceil n/2 \rceil$ downto 0 do
            begin
            1.a)  $X_j := P_{j+1} \cdot 2 \bmod Q$
            1.b)  $Y_j := X_j \cdot 2 \bmod Q$
            2.a)  Calculate $\hat{b}_j$
            2.b)  $Z_j := A \cdot \hat{b}_j \bmod Q$
            3)    $P_j := (Y_j + Z_j) \bmod Q$
            end
Step 3: $P := P_0$                                    □
```

In Step 2 1), we perform a modular doubling twice. In 2.a), we calculate $\hat{b}_j$ from the corresponding five digits of B as mentioned above. In 2.b), a modular doubling may be required as well as a modular negation. 1) and 2) can be performed in parallel. 3) is the same as in the radix-2 algorithm. $\lceil n/2 \rceil + 1$ iterations are required to perform an n-digit modular multiplication.

We can reduce the amount of the required hardware without increasing the computation time, by postponing the residue calculations in Steps 2 1.b) and 2.b) and carrying out a bit more complicated residue calculation in 3). In 1.b) and 2.b), we do not perform the correction stage. Then, Y_j and Z_j become $n + 2$-digit redundant binary numbers and satisfy $-2Q < Y_j < 2Q$ and $-2Q < Z_j < 2Q$, respectively. In 3), since $Y_j + Z_j$ is an $n + 3$-digit redundant binary number and satisfies $-4Q < Y_j + Z_j < 4Q$, we can obtain P_j by applying the correction stage twice. Note that the first application is carried out with Q being shifted with one position to the left.

IV. SERIAL–PARALLEL MODULAR MULTIPLIERS

In this section, we consider serial–parallel modular multipliers based on the algorithms proposed in the previous section. We perform one iteration step in one clock cycle. The multipliers have a regular cellular array structure with a bit slice feature suitable for VLSI implementation.

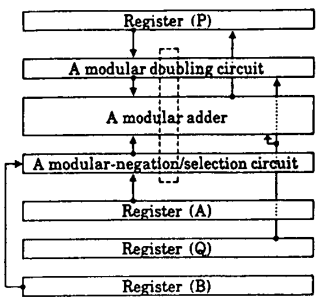

Fig. 2. A block diagram of a modular multiplier.

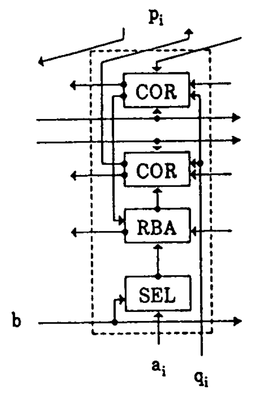

Fig. 3 A block diagram of a slice of the multiplier.

Fig. 2 shows a block diagram of a radix-2 serial–parallel modular multiplier. It consists of four registers and a combinational circuit part. The registers are for storing redundant numbers A, B, and P_j, and a binary number Q. The register for B is a shift register, and B is shifted with one position to the left in each clock cycle. The combinational circuit part consists of a modular doubling circuit for Step 2 1), a modular-negation/selection circuit for 2), and a modular adder for 3). The area of the multiplier is proportional to n.

Fig. 3 shows a block diagram of the combinational circuit part of a slice for a middle position (the region enclosed with the dashed line in Fig. 2). It consists of four cells of three types, i.e., two COR's, an RBA and an SEL. A COR is a cell for the correction stage (the second stage) of modular addition. An RBA is for the redundant binary addition (the first stage of modular addition). An SEL is for modular-negation/selection.

Fig. 4 shows CMOS logic designs of the basic cells. In the designs, we encode each redundant binary digit a_i with two bits a_{is} and a_{id}. $a_{is}a_{id}$ is 11 or 00 or 01, accordingly as a_i is $\bar{1}$ or 0 or 1. In the figure, tv_+ is 1 if tv is positive and q' means the inverse of q. The depth and the gate count of an RBA, a COR, and an SEL are 4 and 8 (42 transistors), 4 and 7 (34 transistors), and 2 and 4 (20 transistors), respectively. Since the depth of a value (tv) evaluation circuit for the correction stage is 2, the depth of the combinational circuit part of the multiplier is 16. (However, in practice, we need buffers for driving long lines for, e.g., a signal for tv.) The gate count of the combinational circuit part of a slice is 25 (128 transistors). The gate count of the combinational circuit part of the whole multiplier

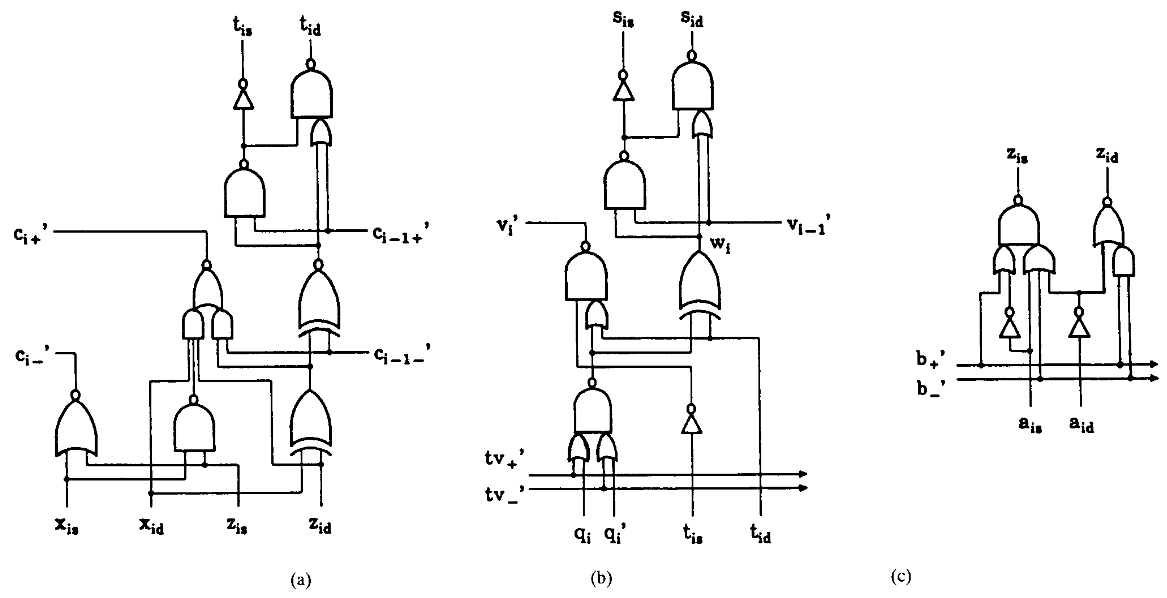

Fig. 4. CMOS logic designs of the basic cells of the multiplier. (a) RBA. (b) COR. (c) SEL.

is about $25n$ ($128n$ 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. According to our rough layout, it will occupy the area of about 50 mm^2 on a VLSI chip, when it will be fabricated using double metal 2 μm CMOS technology. We estimated through SPICE simulation that it will operate with 33 MHz clock.

A radix-4 serial–parallel modular multiplier has a similar structure to the radix-2 one. It also consists of four registers and a combinational circuit part. B is shifted with two positions to the left in each clock cycle. When n is even, initially, a 0 has to be attached to B from the left (to the most significant position). Each slice of the combinational circuit part consists of 6 basic cells of three types, i.e., four COR's, an RBA, and a more complicated SEL. The depth and the amount of hardware of the combinational circuit part are about 40% and 60% larger than those of the radix-2 multiplier, respectively. (Recall that the number of iterations is reduced to about the half.) The number of transistors of a 512-digit radix-4 modular multiplier will be about 140 000. We estimated that, with 2 μm CMOS technology, it will occupy the area of about 65 mm^2 and will operate with 25 MHz clock. Using the method shown in the end of Section III-B, we can omit one COR cell at each slice, and can reduce the amount of hardware of the combinational circuit part with about 15%.

V. Application to RSA Cryptosystem

The proposed multipliers are 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 as encryption. A simple way to perform modular exponentiation is to repeat squaring and multiplication [1]. The procedure is as follows: (We assume e is a k-bit binary number, $[1e_{k-2}\cdots e_0]$.)

```
$C := M$
for $i := k-2$ downto 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. We keep C in the redundant representation during the calculation, and convert only the final C to the ordinary binary representation.

In encryption of a message block sequence, we can perform the exponentiation concurrently with the input of the next message block, as well as the conversion and the output of the previously calculated cipher block. Namely, 3-level 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 is at least 33 kb/s by the radix-2 multiplier at 33 MHz clock and at least 50 kb/s by the radix-4 one at 25 MHz clock. These throughputs are 3 to 5 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.

VI. Concluding Remarks

We have proposed a radix-2 and a radix-4 modular multiplication hardware algorithm. In the algorithms, we represent numbers in a redundant representation, and perform modular additions without carry propagation. Serial–parallel modular multipliers based on the algorithms have a regular cellular array structure with a bit slice feature suitable for VLSI implementation.

The proposed multipliers are efficient especially in applications where modular multiplications are performed iteratively. It seems easy to fabricate an RSA encryption/decryption circuit based on the proposed multiplier on a VLSI chip using today's technology. The circuit is expected to carry out encryption/decryption with the throughput several times as large as that of the fastest actual RSA chip.

The redundant representation for a residue class and the modular addition method shown in this paper are also interesting and useful by themselves.

The use of a still higher radix is interesting [4]. The higher the radix is, the less the required clock cycles are. However, the amount of hardware becomes larger and the clock period becomes longer. The whole multiplication speed may increase. We may adopt a higher radix according to the advance of VLSI technologies. The use of a higher radix signed-digit representation in a redundant representation for the residue class is also interesting. It may decrease the amount of hardware for registers, because we can encode a number with fewer bits. However, it may increase the depth and the amount of hardware of the combinational circuit part. We discussed a redundant representation for a residue class based on a higher radix signed-digit representation in [9].

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 Proc.* New York: Springer-Verlag, 1990, pp. 368–370.

[3] ——, "A fast modular multiplication algorithm with application to two key cryptography," in *Advances in Cryptology, Proc. CRYPTO 82*, D. Chaum *et al.*, Eds. New York: Plenum, 1983, pp. 51–60.

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

[5] N. Takagi, Y. Okabe, H. Yasuura, and S. Yajima, "Modulo m addition using redundant representation and its application to residue-number/binary conversion," Rep. Tech. Group on Computation, IECEJ, COMP86-14, June 1986 (in Japanese).

[6] N. Takagi, "Studies on hardware algorithms for arithmetic operations with a redundant binary representation," Doctoral dissertation, Dep. Inform. Sci., Kyoto Univ., Aug. 1987.

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

[8] 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, Sept. 1985.

[9] H. Yasuura, N. Takagi, and S. Yajima, "Redundant coding for local computability," Perspectives in computing, vol. 15, D. S. Johnson *et al.*, Eds., *Discrete algorithms and complexity: Proceedings of Japan-US Joint Seminar 1986.* New York: Academic, 1987, pp. 145–159.

Design of Self-Checking Circuits Using DCVS Logic: A Case Study

N. Kanopoulos, Dimitris Pantzartzis, and Frederick R. Bartram

***Abstract*—This paper presents a technique for designing self-checking circuits using differential cascode voltage switch (DCVS) logic. This technique is used in a design case study and the results obtained through actual implementation and testing of a self-checking circuit are discussed. It is demonstrated that the self-checking capability of the circuit can be used to implement fault tolerance at low hardware-overhead costs.**

***Index Terms*—Differential cascode voltage switch logic, error detection and correction, fault secure circuits, self-checking circuits, triple modular redundancy.**

I. Introduction

DCVS circuits represent an important direction in complementary metal oxide semiconductor (CMOS) integrated circuit design. Potential advantages over conventional static CMOS circuits include reduced circuit delay, higher layout density, lower power dissipation, extended logic synthesis flexibility, and improved testability when realizing certain types of logic functions [1]–[4]. When comparing static CMOS to DCVS, preliminary results showed that DCVS compares reasonably well in terms of the device count when implementing complex Boolean functions [2]–[4]. It was also noted that the random pattern testability of DCVS circuits is much better than that of their static CMOS counterparts [1]. All of these observations indicate that DCVS offers the potential for faster, easily testable, and possibly smaller-area circuits with less power dissipation, when compared to static CMOS in certain applications. Moreover, the output and its complement are available in DCVS, but not in conventional dynamic CMOS such as domino-CMOS, thereby significantly increasing the logic synthesis efficiency and flexibility.

A study conducted by Barzilai *et al.* indicates that single stuck-at, stuck-closed, and stuck-open faults in most transistors of any DCVS circuit result in either correct values or in loss of complementarity at the outputs. (See Section II for a brief review of the operation of DCVS circuits.) [1] It was also shown that such a property holds in multi-level DCVS circuits in addition to single-level DCVS circuits [1], [5]. The latter result is quite significant since designers often have to use multilevel DCVS circuits in order to minimize charge sharing effects within each DCVS net [2]. This inherent fault-secureness property of DCVS circuits permits us to devise efficient, self-checking schemes for DCVS circuits. On-line self-checking has several useful applications in fault-tolerant systems. It can reduce the average fault detection and fault isolation times significantly when compared to off-line testing and, thus, help in achieving faster reconfiguration and recovery of the fault-tolerant system. Reduction in fault detection and fault isolation times also results in increased availability of systems due to the resulting reduction in mean-time-to-repair. Self-checking features can also be used to implement traditional fault-tolerant configurations at lower hardware costs.

Manuscript received January 4, 1990; revised October 22, 1990. This work was supported in part by the U.S. Army.

N. Kanopoulos is with the Center for Digital Systems, Research Research Triangle Institute, Research Triangle Park, NC 27709.

D. Pantzartzis and F. R. Bartram are with the Department of Electrical Engineering, Duke University, Durham, NC 27706.

IEEE Log Number 9103102.