論文 83-20-6-13

Residue 수체계에 의한 복소 프로세서의 이론적 고찰

(A Theoretical Consideration of Complex Processor Using RNS)

金 德 顯*, 金 在 功**

(Duck Hyun Kim and Jae Kong Kim)

要 約

본 논문은 조합논리회로를 사용하여 레지듀 수체계에 의한 고속 복소수 곱셈기의 구성에 대하여 검토하였다. 레지듀 덧셈에서 나타나는 부호 결정과 오버플로우 교정 문제의 다른 방법을 제시하였으며 고려된 곱셈기의 연산 추정시간은 약 53.15ns이었다.

Abstract

This paper discussed the high speed complex multiplier based on the Residue Number System (RNS) using combinational logic circuits. In addition, the sign determination and overflow correction problem in residue addition has been studied. The estimated multiplication time of considered processor were about 53.15 ns.

I. 序 論

컴퓨터의 과학기술 응용에 있어서 빠른 복소수 연산처리는 큰 의미를 갖는다. 특히 많은 양의 복소곱셈 및 덧셈을 필요로 하는 신호처리에 있어서 FFT, 콘볼루션, 상관함수 계산은 그 좋은 예이다.[1] 종래의 컴퓨터는 복소 곱셈을 소프트웨어적으로 처리함으로써 속도가 매우 늦다. 물론 소프트웨어적인 진전이 계속되고 있으나 저속의 문제점은 여전하다. 이러한 단점을 하드웨어적으로 보완하기 위해 merged processing의 개념을 도입한 벡터 프로세서는 그 효과가 상당하다.[2,3] 컴퓨터의 연산속도를 향상시킬 수 있는 하드웨어적인 두 방법은 다음과 같다.

첫째, 고속의 장치를 사용하는 경우,

둘째, 빠른 연산이 가능한 수체계에 의한 회로구성,

레지듀 수체계(RNS)는 n개의 원소로 이루어진 모듈러스 집합에 의해 결정되는 수체계이므로 연산은 각 모듈러스끼리 독립적이고, 캐리가 없을 뿐 아니라 full precision의 연산이므로 고속, 고정도(high-precision)의 연산이 가능하다.[4] 이 같은 수체계는 Svoboda-Valach 이래 많은 연구[5,6]에도 불구하고 나눗셈, 부호검출, 오버플로우 탐지 및 교정문제로 인해 실용이 일반화되지 못하고 있다. 그러나 최근 제한된 분야에서 그 효용성이 입증되고 있으며 특히 오차교정 특성을 이용한 데이터 전송 및 fault tolerant system 구성은 새로운 주목을 던져 주고 있다.[10-12] 필자는 이 같은 수체계의 특성에 착안 실수 곱셈에 국한한 RNS 고속 곱셈기의 구성을 검토한 바 있다.[13-14]

본 논문에서는 RNS의 병렬처리 특성을 이용한 고속 복소수 곱셈기를 구성함과 동시 부호비트를 별도처리, 오버플로우 검출기의 출력으로 부터 몇 개의 게이트만으로도 부호를 결정할 수 있는 방법을 제시한다. 모든 구성은 입력 buffer 이후부터이고 각 오퍼랜드는 동시에 access되며, 부호와 크기표현의 16 bit

*準會員, **正會員, 東國大學校 工科大學 電子工學科

(Dept. of Elec. Eng., Dong-Guk Univ.)

接受日字 : 1983年 9月 30日

정수임을 전제로 한다.

Ⅱ. 복소 곱셈기의 구성

임의의 두 복소수 X=A+jB, Y=C+jD의 곱 Z는

$$Z=(AC-BD)+j(BC+AD) \quad (1)$$

이다. (1) 식의 기본적인 실현은 4개의 2진 오퍼랜드 A, B, C, D를 레지듀로 먼저 변환하고 레지듀 상태로 각각 곱한 다음 이를 더한 결과를 2진수로 다시 환원하는 일이다.

1. 입력변환

n bit의 2진 오퍼랜드 A의 레지듀 변환을 위해 A에 mod m_i를 취하면

$$|A|_{m_i}=\left|\sum_{j=0}^{n-1} B_j 2^j\right|_{m_i}$$

$$=\left|\sum_{j=0}^{n-1} B_j |2^j|_{m_i}\right|_{m_i} \quad (2)$$

여기서 B_j는 이진 digit

즉 임의의 2진수의 레지듀 변환은 2진 하중(binary weight) 레지듀의 합에 mod m_i를 취한 것과 같다. 이의 실현은 moduli선택조건[14]에 의거 구한 모듈라이집합 {65, 63, 31, 19, 17, 11}에 의해 각 오퍼랜드의 digit에 대한 레지듀 하중이 얻어지면 이를 디코드하여 mod m_i 가산기로 더한다(표 2 참조). 예로서 mod 17에 대해 입력변환기를 구성하면 그림 1과 같다.

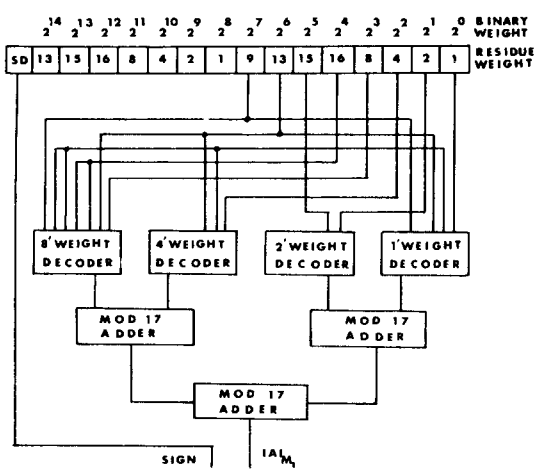


그림 1. 입력변환회로(Mod 17)

Fig. 1. Input translation scheme (Mod 17).

2. Chinese 나머지 정리

Chinese 나머지 정리(CRT)[5][14]는 선형합동 연립방정식

$$x\equiv d_i \bmod m_i \qquad i=1, \cdots, n \quad (3)$$

의 해로서 다음과 같은 표현을 갖는다.

$$|x|_M=\left|\sum_{i=1}^{n} d_i N_i \alpha_i\right|_M \quad (4)$$

여기서 $M=\prod_{i=1}^{n} m_i$: range

$d_i=|x|_{m_i}$

$N_i=M/m_i$

$\alpha_i=\left|\frac{1}{N_i}\right|_{m_i}$: multiplicative inverse

(4) 식에 의해 레지듀 수는 하중된 수(weighted number)로 변환된다. 만일 $0\leq x<M$이면 좌변은 그대로 x가 되나 같은 조건하에서 $\sum_{i=1}^{n} d_i N_i \alpha_i > M$ 이면 $\sum_{i=1}^{n} d_i N_i \alpha_i$에 mod M을 취해 주어야 한다. 그러나 M은 부호비트를 제외한 두 오퍼랜드의 곱 2^{36}보다 크므로 mod M 연산은 쉽지 않다.

x와 M+x는 동일한 표현을 가지므로 $0\leq\sum_{i=1}^{n} d_i N_i \alpha_i - PM < M$이 되도록 P를 결정하면 별도의 mod M 연산은 필요 없다. 따라서 (4)식을 다시 쓰면

$$x=\left|\sum_{i=1}^{n} d_i N_i \alpha_i\right|_M$$

$$=\sum_{i=1}^{n} d_i N_i \alpha_i - PM \quad (5)$$

여기서 P는 magnitude index (M. I.)[8]이다.

3. 레지듀 곱셈 및 M. I. 생성

입력변환기의 출력 $|B|_{m_i}$, $|C|_{m_i}$의 곱셈은

$$|BC|_{m_i}=||B|_{m_i}|C|_{m_i}|_{m_i} \quad (6)$$

(6)식의 실제적 구성은 각 모듈러스에 대해 곱셈표를 만든 다음 곱셈표 기능을 회로화한다.[14]

레지듀 BC 곱셈기의 M. I. P_A를 구하기 위해 (6)식에서

$$a_i=|BC|_{m_i} \qquad i=1, \cdots, n \quad (7)$$

놓고 (5)식을 고쳐쓰면

$$BC=\sum_{i=1}^{n} a_i N_i \alpha_i - P_A M \quad (8)$$

그러므로

$$P_A=\frac{1}{M}\left(\sum_{i=1}^{n} a_i N_i \alpha_i - BC\right) \quad (9)$$

(9)식을 base extention[16] (B. E.) 하면

$$P_A=\left|\sum_{i=1}^{n} a_i \mu_i \alpha_i + Z_s \mu_s\right|_{m_s} \quad (10)$$

여기서 m_s는 B. E. 모듈러스

$\mu_i=\left|\frac{1}{m_i}\right|_{m_s}$: multiplicative inverse

$Z_s=|BC|_{m_s}$: B. E. 레지듀

$\mu_s=\left|\frac{-1}{M}\right|_{m_s}$: 최상 모듈러스

(10)식의 실현 절차는

① $|\mu_i a_i \alpha_i|_{m_s}$의 테이블을 구성한다.

② $|\mu_s Z_s|_{m_s}$의 테이블을 구성한다.

③ ①,②의 과정을 동시에 수행한 후 column compresion[17](C. C.)으로 더한다.

④ ③의 결과에 mod m_s를 취한다. 이때 $m_s=2^k$로 택하면 mod m_s 연산에 별도의 로직이 필요없게 된다. (예 $m_s=2^6=64$)

4. 레지듀 덧셈

(1)식의 허수부를 구하기 위해 두 정수곱 BC 및 AD의 RNS 표현은

$$BC=(a_1, a_2, a_3, \cdots, a_n, P_A) \quad (11\text{-a})$$

$$AD=(b_1, b_2, b_3, \cdots, b_n, P_B) \quad (11\text{-b})$$

동부호 및 이부호 덧셈은

$$BC+AD=(|a_1+b_1|_{m_1}, |a_2+b_2|_{m_2}, \cdots, |a_n+b_n|_{m_n}, P_A+P_B) \triangleq (C_1, C_2, \cdots, C_n, P_C) \quad (12)$$

부호와 크기가 분리된 윗식의 모듈러 덧셈은

1) 동부호이면 크기는 서로 더하고 부호는 그들 자신의 부호를 취하며.

2) 이부호이면 addend의 radix 보수를 취해 서로 더한 다음 오버플로우가 있으면 바로 출력시키고 augend 부호를 취하며 오버플로우가 없으면 보수덧셈 결과에 다시 보수를 취해서 출력시키고 addend 부호를 취한다.

이상의 기능을 회로화하면 그림 2와 같다. XOR 게이트는 곱셈결과 부호를 상호비교하여 동부호 및 이부호 덧셈을 결정한다. 이부호 덧셈의 경우 오버플로우 검출기가 오버플로우를 검출하면 덧셈 결과를 그대로 출력시키고 그렇지 않으면 덧셈 경과에 다시 보수를 취한다. 그림 2의 점선내부는 부호결정회로이며 위 과정의 진리표(표 1)의 회로화다. RNS에서 부호결정의 주된 두 방법은 Mixed Radix Conversion(MRC)을 이용한 방법[18]과 M. I. 비교에 의한 방법[15]이다.

표 1. 부호 결정 진리표

Table 1. Truth table for sign determination.

BC	AD	OVF	부호출력
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1

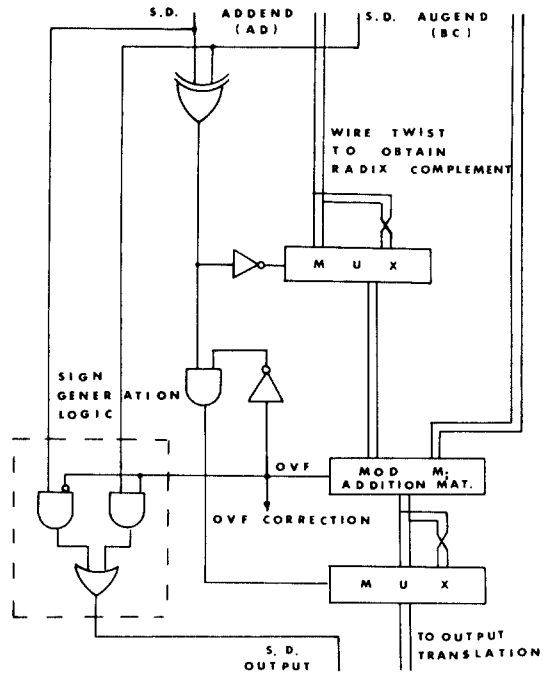


그림 2. 레지듀 덧셈 회로

Fig. 2. Residue addition scheme.

MRC를 이용 레지듀 수를 하중된 수로 바꾸어 두 수를 비교하는 방법은 MRC 구성에 많은 시간이 소비되고 M. I. 비교에 의한 방법 또한 M. I. 생성에 별도의 로직과 시간이 필요하다. 그러나 단지 3개의 게이트만에 의해서 부호비트를 별도처리 덧셈결과의 오버플로우 유무로서 부호를 결정하는 방법은 다른 방법에 비해 구성이 간단함과 동시에 결정시간이 빠르다는 장점을 나타낸다.

5. 오차교정과 출력변환

(5)식에 의해 (11)식을 다시 쓰면

$$BC=\sum_{i=1}^{n} a_i N_i \alpha_i - P_A M \quad (13\text{-a})$$

$$AD=\sum_{i=1}^{n} b_i N_i \alpha_i - P_B M \quad (13\text{-b})$$

두 식을 더하면

$$BC+AD=\sum_{i=1}^{n} (a_i N_i \alpha_i + b_i N_i \alpha_i) - (P_A+P_B)M \quad (14\text{-a})$$

$$=\sum_{i=1}^{n} (a_i+b_i) N_i \alpha_i - (P_A+P_B)M \quad (14\text{-b})$$

$$\triangleq\sum_{i=1}^{n} c_i N_i \alpha_i - P_C M \quad (14\text{-c})$$

a_i+b_i는 모듈러 연산이므로 덧셈에서 오버플로우가 생기면 출력 c_i는 a_i+b_i와 달리 $|a_i+b_i|_{m_i}$

즉 $a_i+b_i-m_i$로 나타난다. 이 경우 (14-b)식을 다시 쓰면

$$\sum_{i=1}^{n}(a_i+b_i)N_i\alpha_i-(P_A+P_B)M$$

$$=\sum_{i=1}^{n}(c_i+m_i)N_i\alpha_i-(P_A+P_B)M$$

$$=\sum_{i=1}^{n}c_iN_i\alpha_i-P_CM+\sum_{i=1}^{n}m_iN_i\alpha_i \qquad (15)$$

(15)식에 $N_im_i=M$을 대입하면

$$BC+AD=\sum_{i=1}^{n}C_iN_i\alpha_i-P_C'M \qquad (16)$$

여기서 $P_C'\triangleq P_C-\sum_{i=1}^{n}\alpha_i$

따라서 오버플로우가 나타나면 P_C에서 오버플로우가 나타난 모듈러스의 multiplicative inverse 합만큼 빼주어야 한다. 표 2는 a_i와 b_i의 모듈러 덧셈표로서 △ 부분은 오버플로우를 나타낸다. 임의의 모듈러스에 대한 오버플로우의 검출은 덧셈 매트릭스내에서 오버플로우가 나타날 때 AND 게이트 출력을 모두 ORing 하면 된다. 검출된 오버플로우는 $\sum_{i=1}^{n}\alpha_i$를 미리 계산하여 같은 것끼리 ORing하여 P_C에서 빼준다. 그림 3은 이상의 오차교정 과정을 나타낸다.

표 2. 덧셈에서의 오버플로우(Mod 17)
Table 2. Overflows in addition (Mod 17).

a_i \ b_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	0
2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	0	1
3	3	4	5	6	7	8	9	10	11	12	13	14	15	16	0	1	2
4	4	5	6	7	8	9	10	11	12	13	14	15	16	0	1	2	3
5	5	6	7	8	9	10	11	12	13	14	15	16	0	1	2	3	4
6	6	7	8	9	10	11	12	13	14	15	16	0	1	2	3	4	5
7	7	8	9	10	11	12	13	14	15	16	0	1	2	3	4	5	6
8	8	9	10	11	12	13	14	15	16	0	1	2	3	4	5	6	7
9	9	10	11	12	13	14	15	16	0	1	2	3	4	5	6	7	8
10	10	11	12	13	14	15	16	0	1	2	3	4	5	6	7	8	9
11	11	12	13	14	15	16	0	1	2	3	4	5	6	7	8	9	10
12	12	13	14	15	16	0	1	2	3	4	5	6	7	8	9	10	11
13	13	14	15	16	0	1	2	3	4	5	6	7	8	9	10	11	12
14	14	15	16	0	1	2	3	4	5	6	7	8	9	10	11	12	13
15	15	16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
16	16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

그림 3에서 교정된 P_C'와 덧셈기의 출력으로 2진 형태의 $C_iN_i\alpha_i$와 PM의 보수값을 선택하면 C.C.에 의해 동시에 합해져 2진수로 최종 출력된다. 문헌 [14]는 그 구체적 출력방법을 제시한다.

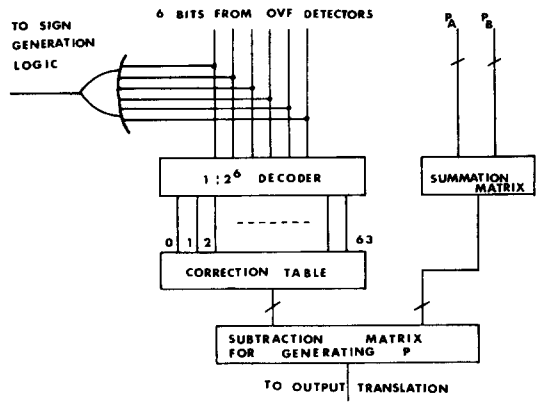


그림 3. 오버플로우 교정회로
Fig. 3. Overflow correction scheme.

이상의 절차에 의한 복소 곱셈기의 구성은 그림4와 같다.

2진 입력 오퍼랜드 A, B, C, D는 선택한 모듈라이에 의해 레지듀로 입력변환된 다음 같은 모듈러스끼리 곱해져 AC, BD, BC, AD의 결과를 생성한다. 이 레지듀 곱과 병렬로 출력변환과 오차교정을 위한 M.I. 즉 P_A, P_B, P_C, P_D가 생성된다. AC+(−BD)와 BC+AD의 덧셈에서 오버플로우의 유무에 따라 P_A+P_B의 결과를 교정해 주는 오차교정 과정이 수행된다. 교정된 P_C'가 생성되면 CRT에 의하여 실수부와 허수부의 레지듀 수를 출력변환하여 복소수곱셈 결과를 얻는다. 연산 결과의 부호는 입력 오퍼랜드의 부호비트를 X-OR하여 AC, BD, BC, AD의 부호를 만든 다음 그림3의 부호발생기에 의해 결정한다.

Ⅲ. 연산시간 추정

모든 오퍼랜드에 대해 중복되지 않는 가장 긴 경로(critical path)를 따라 E.C.L.의 최대 게이트 지연[18]을 기준으로 각 부분의 연산시간을 추정한다. 단 곁선에 의한 지연은 무시한다.

입력변환에 decoder 3.6 nS, ORing 0.95 nS, mod 덧셈기는 2 level이므로 4.1nS, 小計 8.65nS, M.I. 생성에 있어서 ①, ②의 table에 4.1nS, ③과정의 C.C.에(7, 3) 카운터 5.5nS, (3, 2) 카운터 5.5nS, (2, 1) 카운터는 CLA(carry lookahead adder)로서 6 bit 크기므로 7.3nS, 小計 18.3nS가 산출된다.

M.I. 생성 과정과 동시에 레지듀 곱셈, 덧셈 및 오버플로우 검출 과정이 수행되나 이 과정은 M.I.가 생성되기 전에 충분히 이루어질 수 있다. M.I.가 생성된 다음 P_A+P_B의 덧셈에 2.05nS, 교정에 2.05nS, PM

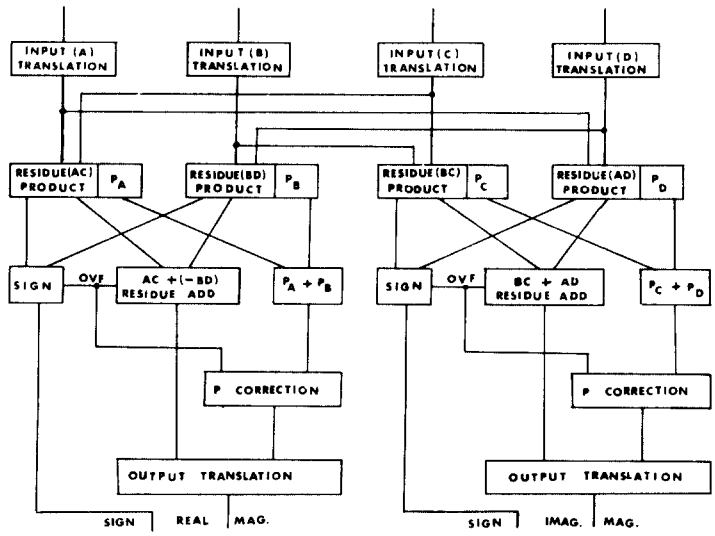
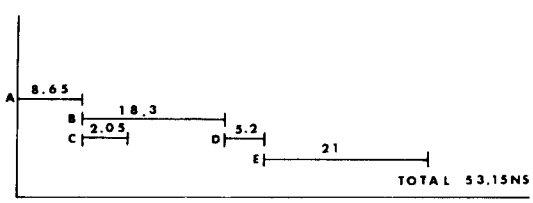


그림 4. 복소 곱셈기의 블럭선도

Fig. 4. Block diagram of complex multiplier.

의 보수와 C_i N_i α_i 의 선택에 1.1nS, 小計 5.2nS가 산출되며, 출력변환의(7, 3) (3, 2) 카운터는 위와 같고(2, 1) 카운터의 CLA는 30 bit 길이므로 10 nS 小計 21nS가 산출된다. 이상으로 부터 총 53.15nS가 산출된다. 각 부분의 동작시간을 도표화하면 그림 5와 같다.



A ; Input Translation
B ; M.I. Generation
C ; Residue Multiplication
D ; M.I. Correction
E ; Output Translation

그림 5. 연산 추정시간

Fig. 5. Estimated operation timing.

Ⅳ. 結 論

레지듀 수체계의 고속 연산과 병렬처리 특성을 이용 연산 추정시간 53.15nS의 고속 복소곱셈기의 이론적 구성을 시도하였다. 종래의 부호검출 방법대신 3개의 게이트만으로도 부호를 바로 결정할 수 있는 방법을 제시하였으며 덧셈에서 오버플로우로 인한 오차교정 방법을 제안하였다. 최근 RNS의 신호처리 응용연구와 merged processing의 추세로 보아 상기 구성의 실제 회로화 방법이 고안된다면 다량의 복소곱셈을 필요로 하는 FFT, 콘볼루션, 상관함수, 디지탈 필터링 등에 유용하게 이용될 수 있으리라 사료된다.

參 考 文 獻

[1] C.H. Huang, D.G. Peterson, H.E. Rauch, J.W. Teague, D.F. Fraser, "Implementation of a fast digital processor using the residue number system," *IEEE Trans. Circuit. Syst.*, vol CAS-28, no. 1, Jan. 1981.

[2] D.J. Kuck, D.H. Laware, A.H. Sameh, *High Speed Computer and Algorithm Organization*. Academic Press, New York, 1977.

[3] E.E. Swartzlander Jr., "Merged Arithmethic for signal processing," *IEEE SOUTHCON Conf.*, pp. 239-244, 1978.

[4] R.D. Merrill, "Improving digital computer performance using residue number theory," *IEEE Trans. Comput.*, vol. EC-13, pp. 93-101. 1964.

[5] N.S. Szabo, R.I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. McGraw-Hill, New York, 1967.

[6] A. Sasaki, "Addition and subtraction in residue number system," *IEEE Trans. Comput.*, vol. EC-16, no.2, Apr. 1967.

[7] D.K. Banerji, J.A. Brzozowski, "On the translation algorithms in residue number system," *IEEE Trans.. Comput.*, vol. C-21, no. 12, pp. 1281-1285, Dec. 1972.

[8] F. Barsi, P. Maestrini, "Error correcting properties of redundant residue number systems, " *IEEE Trans. Comput.*, vol. C-22, no. 3, March 1973.

[9] G.A. Jullien, "Residue number scaling and other operations using ROM arrays," *IEEE Trans. Comput.*, vol. C-27, no.4, Apr. 1978.

[10] M.H. Etzel, W.K. Jenkins, "Redundant residue number systems for error detection and correction in digital filters," *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. ASSP-28, no. 5, Oct. 1980.

[11] V. Ramachandran, "Single residue error correction in residue number systems," *IEEE Trans. Comput.*, vol. C-32, no.5, May 1983.

[12] W.K. Jenkins, "The design of error checkers for self-checking residue number arithmetic," *IEEE Trans. Comput.*, vol. C-32, no. 4, Apr. 1983.

[13] S.Y. Kim, D.H. Kim, J.K. Kim, "An architecture for high-speed multiplier using NRNS," *KIEE Summer Conf.*, vcl. 6, no. 1, July 1983.

[14] S.Y. Kim, J.K. Kim, "A study on the high-speed multiplier using RNS," *Journal of KIEE*, vol. 20, no. 5, Sept. 1983.

[15] T.R.N. Rao, A.K. Trehan, "Binary logic for residue arithmetic using magnitude index," *IEEE Trans. Comput.*, vol. C-19, no. 8, Aug, 1970.

[16] K.H. O'Keefe, J.L. Wright, "Remarks on base extention for modular arithmetic," *IEEE Trans. Comput.*, vol. C-22, no.9, pp. 833-835, Sept. 1973.

[17] I.T. Ho, T.C. Chen, "Multiple addition by residue threshold functions and their representation by array logic," *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 762-767, Aug. 1973.

[18] D.K. Banerji, J.A. Brzozowski, "Sign detection in residue number systems," *IEEE Trans. Comput.*, vol. C-8, no. 4, Apr. 1969.

[19] Fairchild, *F100k Data book*. 1982.