부동소수점 덧셈과 곱셈에서의 라운딩 병렬화 알고리즘 연구

이원희 강준우
한국외국어대학교 전자공학과
449-791 경기도 용인시 모현면 왕산리 산 89 번지
Tel : 0335-330-4502, Fax : 0335-330-4120, E-mail : leewh@san.hufs.ac.kr

Study on Parallelized Rounding Algorithm in Floating-point Addition and Multiplication

Won-Hee Lee and Jun-Woo Kang
Dept. of Electronics Engineering, Hankuk University of Foreign Studies
89 Wangsan, Mohyun, Youngin, Kyonggi-Do, 449-791, KOREA
Tel : 0335-330-4502, Fax : 0335-330-4120, E-mail : leewh@san.hufs.ac.kr

Abstract

We propose an algorithm which processes the floating-point addition/subtraction and rounding in parallel. It also processes multiplication and rounding in the same way. The hardware model is presented that minimizes the delay time to get results for all the rounding modes defined in the IEEE Standards. An unified method to get the three bits (L, G, S) for the rounding is described. We also propose an unified guide line to determine the 1-bit shift for the post-normalization in the Floating-point addition/subtraction and multiplication.

I. 서론

부동소수점 수는 고정소수점 수 보다 넓은 범위의 수를 표현할 수 있고 더 정밀한 표현도 가능하다. 그러나 무한의 실수를 유한의 수로 표현하는 데는 한계가 있으므로 라운딩이 필수적으로 요구된다. IEEE Standard [1]에서는 4 가지 라운드 모드를 표준으로 삼고 있는데 이 라운드 모드를 모두 지원하는 부동소수점 연산기(Floating Point Unit : FPU)는 라운드 모드를 지원하지 않는 연산기에 비하여 연산 시간이 길고 많은 하드웨어 추가가 필요하다. 일반적인 부동소수점 연산에서는, 덧셈, 정규화, 라운딩의 순서로, 또는 덧셈, 라운딩, 정규화의 순서로 처리한다. 덧셈, 정규화, 라운딩의 처리 순서에서는 라운딩을 위한 최종 결과 값에 증가기가 필요하고 최상위 비트에서 올림수가 생길 경우에는 재정규화(Renormalization)가 필요하다. 또한, 덧셈, 라운딩, 정규화의 처리 순서에서는 라운딩 위치를 파악하는 회로가 추가된다. 따라서, 이러한 지연시간과 추가적인 하드웨어를 줄이기 위해서 연산과 라운딩 병렬화 연구가 수행된 바 있다[2-5].

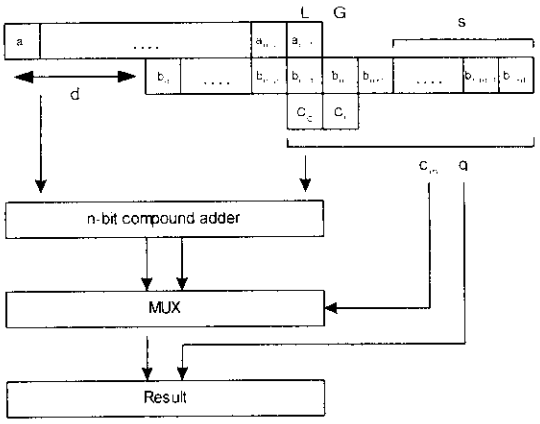
[2,3]에서는 덧셈기 라운딩에서 2 의 보수 시 추가 1 과 라운딩을 동시에 처리하는 알고리즘이 발표되었다. 연산의 종류와 지수차에 따라서 5 가지 경우로 나누어 라운딩 값을 구했다. [4]에서는 곱셈의 라운딩을 곱셈 과정에서 올림수와 합을 더할 때 동시에 처리하는 알고리즘을 발표하였다. [5]에서는 이를 다시 정리하고 구체적인 하드웨어 모델을 제시하였다.

본 논문에서는 덧셈, 뺄셈, 곱셈에서 라운딩 처리에 필요한 3 비트(L, G, S) 값을 구하는 데 있어, 각 라운드 모드에 동일하게 적용할 수 있는 식을 제안하였다. 이 식에 라운딩 할 때 필요한 3 비트를 대입하여 라운드 모드 식에 대입하면 각 라운드 모드에 따르는 라운딩 값을 구할 수 있다. 각 경우에 구한 라운딩 값 중 하나를 선택하는 기준은 연산자와 지수차이고 여기에 결과 값 정규화 과정에서 1 비트 쉬프트를 결정하는 식이 포함된다. 이 식을 모든 연산에서 동일하게 적용할 수 있다.

II. 라운딩 알고리즘

1. 덧셈 라운딩 알고리즘

부동소수점 덧셈에서 두 피연산자를 A, B 라고 할 때 덧셈을 위하여 이 중 작은 수를 오른쪽으로 쉬프트해서 지수의 크기를 맞추면 A, B 와 결과 R 의 범위는 $1 \le A < 2$, $0 \le B < 2$, $1 \le R < 4$ 가 된다. n 비트의 덧셈이면 결과 값은 2n 비트이다. 정규화 과정에서 결과 값의 상위 n 비트는 취하나 하위 n 비트는 제외된다. 이 때 취하는 부분을 정수부로 제외된 부분을 분수부로 한다. 분수부 n 비트에서 상위 n 비트로 라운딩 된다. 상위 n 비트의 덧셈과 하위 n 비트의 라운딩을 분리하여 동시에 수행하고 라운딩 값 C_r을 정수부에 더하여 준다.



(그림 1) 부동소수점 덧셈의 하드웨어 모델

이 때, 상위 n 비트의 덧셈에 사용되는 피연산자를 A_g, B_g라고 하면, 결과 값은 A_g+B_g 또는 A_g+B_g+1 이다. 따라서, 연산시간의 절감을 위하여 최하위 비트로의 올림수가 '0'일 때와 '1'일 때를 동시에 계산하여 두 가지 결과를 출력하고 라운딩 값에 따라 결과 값을 선택하는 복식 가산기(Compound Adder)를 사용한다[6]. 하드웨어 모델은 다음 (그림 1)과 같다. 이 때, C_{in}은 분수부에서 정수부로의 올림수이고 d 는 A, B 의 지수차이다.

IEEE 표준 라운딩인 Round-to-Zero, Round-to-Nearest, Round-to-Plus-Infinity, Round-to-Minus-Infinity 중 Round-to-Plus-Infinity 와 Round-to-Minus-Infinity는 부호를 고려하면 Round-to-Infinity로 통합 할 수 있다. 위 3 가지 라운딩을 모두 지원하기 위해서는 L(LSB), G(Guard), S(Sticky) 등 3 비트가 필요한데 L 은 정수부의 최하위 비트이고 G 는 분수부의 최상위 비트이며, S 는 분수부 중 G 를 제외한 모든 비트의 논리합이다. 덧셈 결과 값의 범위는 $1\le R<4$ 이기 때문에 최상위 비트에서 올림수가 발생할 수 있다. 올림수가 발생한 경우 정규화 할 때 1 비트를 오른쪽으로 쉬프트 시켜야 한다. 여기서 정규화 할 때 오른쪽 쉬프트를 하는 결과 값을 ORS(One Right Shift)라고 하고 오른쪽 쉬프트를 하지 않을 땐 NRS(No Right Shift)라고 한다. 이 때 L, G, S 는 다음 식으로 나타낼 수 있다.

NRS 일 때 :

$$\begin{aligned} L &= a_{n-1} \oplus b_{n-1} \\ S &= b_{n-1} \vee s \\ G &= b_n \end{aligned} \tag{1}$$

ORS 일 때 :

$$\begin{aligned} L &= a_{n-2} \oplus b_{n-2} \oplus a_{n-1} b_{n-1} \\ S &= b_n \vee b_{n+1} \vee s \\ G &= a_{n-1} \oplus b_{n-1} \end{aligned} \tag{2}$$

위 식에서 s 는 n+1 비트보다 하위 비트들의 논리합이다. ⊕은 Exclusive-OR 이고 ∨은 논리합이다. L, G, S 에 따르는 3 가지 라운드 모드 식은 다음과 같다.

$$\begin{aligned} &\text{Round-to-Zero(L,G,S)=0} \\ &\text{Round-to-Nearest(L,G,S)=G(L}\vee\text{S)} \\ &\text{Round-to-Infinity(L,G,S)=G}\vee\text{S} \end{aligned} \tag{3}$$

위 라운드 모드에 따라 구해진 C_r 은 정수부로의 올림수 C_{in}이 된다.

$$C_{in} = C_r = \text{RMode}(L,G,S) \tag{4}$$

위 식에서 RMode(L,G,S)는 라운드 모드 식 (3)에 L,G,S 값을 대입한 것이다.

2. 뺄셈 라운딩 알고리즘

2 의 보수를 이용한 뺄셈에서 피연산자에 2 의 보수를 만들 때 1 의 보수 이후 추가되는 '1'이 정수부에 더해지려면 분수부의 값이 '0'이어야 한다. 2 의 보수 후 추가되는 '1'이 정수부에 더해지면 분수부의 값이 '0'이므로 라운딩은 발생하지 않는다. 그래서 2 의 보수를 만들 때 추가되는 '1'과 라운딩을 같이 처리할 수 있다. 2 의 보수를 만들 때 추가되는 '1'이 정수부에 더해지는 값을 C_c라고 한다. C_c는 b_n, b_{n+1}, s 가 모두 '0' 일 때만 '1'이 된다.

$$C_c = \bar{b}_n \bar{b}_{n+1} \bar{s} \tag{5}$$

정규화 쉬프트에 따라서 L, G, S 값이 바뀌므로 지수차가 1 보다 큰 경우와 지수차가 1 인 경우를 나누어 C_{in} 값을 구한다[2]. 지수차가 0 인 경우에는 지수부에서 두 수의 크기 비교도 불가능 하고 라운딩도 필요 없으므로 따로 계산한다. 뺄셈도 (그림 1)의 하드웨어 모델을 사용한다. 뺄셈 라운딩 알고리즘 식들에서 b 의 n-1 보다 상위 비트는 1 의 보수를 취한 값이고 하위 비트는 본래의 값이다.

2.1 지수차가 1 보다 큰 경우의 뺄셈

지수차가 1 보다 큰 경우의 뺄셈에서 피연산자와 결과 값의 범위는 $1\le A<2$, $0\le B<0.5$, $0.5\le R<2$ 이기 때문에 정규화 과정에서 왼쪽 쉬프트가 없는 NLS(No Left Shift) 경우와 1 비트 왼쪽 쉬프트인 OLS(One Left Shift) 경우가 발생한다.

2.1.1 NLS 인 경우

L, S, G 는 다음 식으로 구해질 수 있다.

$$\begin{aligned} L &= \bar{b}_n \bar{b}_{n+1} \bar{s} \oplus a_{n-1} \oplus b_{n-1} \\ S &= \bar{b}_{n-1} \vee s \\ G &= \bar{b}_n \oplus \bar{b}_{n+1} \bar{s} \end{aligned} \tag{6}$$

C_{in}은 다음과 같다.

$$C_{in} = C_r \vee C_c = \text{RMode}(L,G,S) \vee \bar{b}_n \bar{b}_{n+1} \bar{s} \tag{7}$$

2.1.2 OLS 인 경우

라운딩은 n+1 번째 비트에서 n 번째 비트로의 올림수이다. 그래서 L 은 n 번째 비트이고 G 는 그 다음 비트이다. L, G, S 는 다음과 같다.

$$\begin{aligned} L &= \bar{b}_n \oplus \bar{b}_{n+1} \bar{s} \\ S &= s \\ G &= \bar{b}_{n-1} \oplus \bar{s} \end{aligned} \tag{8}$$

C_{in}은 C_e값과 n 번째 비트에서 n-1 비트로의 올림수의 논리합이다.

$$C_{in} = \bar{b}_n \bar{b}_{n-1} \bar{s} \vee LC_r \quad (9)$$

q 는 왼쪽 쉬프트 이후에 마지막 비트로 추가되는 값이다. q 가 최하위 비트이므로 라운딩은 q 로 반올림 된다.

$$q = L \oplus C_r = (\bar{b}_n \oplus \bar{b}_{n-1}(\bar{s}) \oplus RMode(L,G,S) \quad (10)$$

2.2 지수차가 1 인 경우의 뺄셈

지수차가 1 인 경우의 뺄셈에서 피연산자와 결과 값의 범위는 $1 \le A < 2$, $0.5 \le B < 1$, $0 \le R < 1.5$ 이기 때문에 정규화 과정에서 왼쪽 쉬프트가 없는 NLS(No Left Shift)인 경우와 왼쪽 쉬프트가 많은 MLS(Many Left Shift)인 경우가 발생한다. NLS 인 경우에는 라운딩이 필요하지만 MLS 인 경우에는 q 값까지 취하므로 라운딩은 하지 않는다. NLS 인 경우 L, G, S 는 다음과 같다.

$$\begin{aligned} L &= a_{n-1} \oplus b_{n-1} \oplus \bar{b}_n \\ S &= 0 \\ G &= b_n \end{aligned} \quad (11)$$

C_e, C_{in} 은 다음과 같다.

$$C_e = \bar{b}_n \quad (12)$$

$$C_{in} = C_e \vee C_r = \bar{b}_n \vee RMode\ (L,S,G) \quad (13)$$

MLS 인 경우에는 C_{in}은 위 식의 C_e와 같고 q 값은 b_n 이다. 결과 값은 leading zero 를 검사하여 정규화 할 때 왼쪽 쉬프트 수를 결정한다.

2.3 지수차가 0 인 경우의 뺄셈

C_{in}은 일반적인 뺄셈과 마찬가지로 항상 1 이 되고 leading zero 를 검사한다.

3. 곱셈 라운딩 알고리즘

n 비트의 곱셈 결과는 2n 비트이다. 2n 비트의 결과 값 R_{2n}은 다음과 같다.

$$R_{2n} = r_{2n-1} r_{2n-2} \cdots r_{n+1} r_{n-2} \cdots r_1 r_0 \quad (14)$$

결과 값은 2n-2 에서 n-1 까지를 취하므로 최종 결과 값을 정규화 할 때 r_{2n-1}의 값이 '0'이면 오른쪽 쉬프트를 하지 않고(No Right Shift : NRS) '1'이면 오른쪽 쉬프트를 한다(Right Shift : RS). 이 쉬프트에 따라서 라운딩이 달라지므로 2 가지 경우를 분리하여 C_{in}을 구한다. n 비트의 곱셈 중간 결과 값으로는 합 S(Sum)와 올림수 C(Carry)가 있는데 이 두 값을 더하여 최종 결과 값 R 이 다음 식 (15)와 같이 구해진다.

$$R = C[2n-1:n-1] + S[2n-1:n-1] + (C_{n-2} + S_{n-2}) + C_{in}^{m}{}_{n-2}$$

$$+Cr_{n-1} = R_n + (C_{n-1} + S_{n-1}) + (C_{n-2} + S_{n-2}) + C_{in}^{m}{}_{n-2} + Cr_{n-1} \quad (15)$$

여기서, $C_{in}^{m}{}_{n-1}$은 n-1 번째 비트로의 올림수 값이고 Cr_{n-1}은 n-1 번째 비트로의 라운딩 값이다. C[2n-1:n-1]+S[2n-1:n-1]은 합과 올림수의 2n-1 비트에서 n-1 비트까지의 합이고 R_n은 C[2n-1:n]+S[2n-1:n]이다.

결과 값 R 은 식 (15)에서는 R_n+(0, 1, 2, 3, 4) 이고 n 번째 비트 다음을 소수점으로 보면 R 은 R_n+ (0, 1, 2) 이다. n 번째 비트로의 입력 올림수를 C_{in}으로 하면 R_n+2 는 C_{in}이 1 일 때 R_n+C_{in}+1 로 나타낼 수 있다. 여기서 추가 1 을 P(Plus one)로 하자. P 는 C_{n-1}과 S_{n-1}이 모두 '1' 이 되는 경우에 1 이 된다.

$$P = C_{n-2} \wedge S_{n-2} = overflow(C_{n-2} + S_{n-2}) \quad (16)$$

식 (15)를 P 를 포함해서 다시 쓰면 다음과 같다.

$$R = R_n + (C_{n-1} + S_{n-1}) + P + (C_{n-2} \oplus S_{n-2}) + C_{in}^{m}{}_{n-2} + Cr_{n-1} \quad (17)$$

위 식에서 R_n 과 P 를 제외하면 다음과 같다.

$$(C_{n-1} + S_{n-1}) + (C_{n-2} \oplus S_{n-2}) + C_{in}^{m}{}_{n-2} + Cr_{n-1} \quad (18)$$

3.1 NRS 인 경우

NRS 인 경우에는 R 의 2n-2 에서 n-1 번째 비트까지의 값을 결과 값으로 취하므로 라운딩은 n-2 번째 비트에서 n-1 번째 비트로 반올림된다. 식 (18)에서 n 번째 비트로의 올림수가 C_{in}이 되고 n-1 번째 비트에 남아있는 값이 q 가 된다.

$$C_{in} = overflow((C_{n-1} + S_{n-1}) + overflow((C_{n-2} \oplus S_{n-2}) + C_{in}^{m}{}_{n-2}) + Cr_{n-1}) \quad (19)$$

$$q = C_{n-1} \oplus S_{n-1} \oplus overflow((C_{n-2} \oplus S_{n-2}) + C_{in}^{m}{}_{n-2}) + Cr_{n-1}) \quad (20)$$

식 (19), (20)에서 라운딩 값은 각 라운드 모드의 L, G, S 값에 각각 r_{n-1}, r_{n-2}, Sy 를 대입하면 된다. Sy 는 n-3 번째 비트 이후의 값들을 논리합 한다.

3.2 RS 인 경우

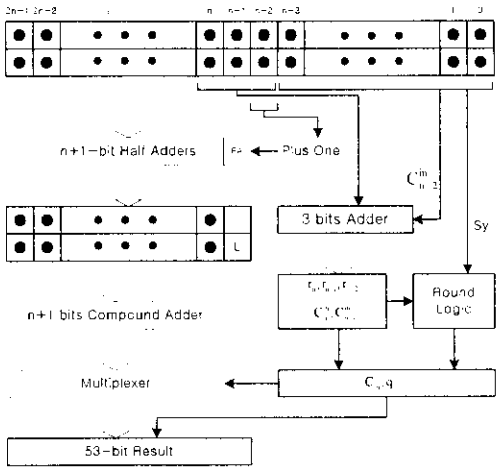
RS 인 경우에는 2n-1 에서 n 번째 비트까지의 값을 취하므로 라운딩은 n-1 번째 비트에서 n 번째 비트로 반올림된다. 식 (18)의 Cr_{n-1}을 Cr_n으로 바꾸면 n 번째 비트로의 올림수 C_{in}이 된다.

$$C_{in} = overflow((C_{n-1} + S_{n-1}) + overflow((C_{n-2} \oplus S_{n-2}) + C_{in}^{m}{}_{n-2})) + Cr_n \quad (21)$$

위 식에서 라운딩 값은 각 라운드 모드의 L, G, S 값에 r_n, r_{n-1}, $r_{n-2} \vee Sy$ 를 대입하면 된다. 이상을 바탕으로 설계한 하드웨어 모델은 다음 (그림 2)와 같다. 여기서, 3 비트 덧셈기를 이용하면 r_n, r_{n-1}, r_{n-2}, C_n^m, C_{n-1}^{in}을 간단히 구할 수 있다.

4. 1 비트 정규화 쉬프트 결정

위 과정에서, C_{in}을 구하는 식은 연산자와 지수차에 의하여 한 가지 식이 선택된다. 그러나, 덧셈에서는 ORS 와 NRS, 뺄셈에서는 NLS 와 OLS, 곱셈에서는 RS 와 NRS 중 하나의 식을 선택하는 기준은 명확하지 않다. 예를 들어, 덧셈의 경우에는 NRS 와 ORS 중 한 가지 경우의 C_{in}이 선택되어야 하는데 이 선택 기준을



(그림 2) 부동소수점 곱셈의 하드웨어 모델

결정하는 값은 최상위 비트가 '0' 또는 '1'인 경우로 간단하게 생각할 수도 있지만, 최하위 비트로의 올림수가 '1'이 되어 최상위 비트가 '0'에서 '1'로 바뀌는 경우를 생각해 볼 수 있다.

그러므로, 선택 기준을 결정하는 값들은 NRS, ORS 의 C_{in}과 최하위 비트로의 올림수가 '0' 일 때와 '1' 인 때의 최상위 비트 값이다. 이 값들에 의해서 정규화 할 때 오른쪽 쉬프트를 할 것인가 하지 않을 것인가도 결정할 수 있다. 다음 <표 1>은 각 경우에 C_{in} 과 1 비트 정규화 쉬프트를 구하는 표이다.

<표 1> Cin 과 1 비트 정규화 쉬프트 결정표

Temp C_{in}		MSB		1 Bit	Determined
NRS	ORS	$C_{in}0$	$C_{in}1$	Shift	C_{in}
0	0	0	0	NRS	0
0	0	0	1	NRS	0
0	0	1	0	X	X
0	0	1	1	ORS	0
0	1	0	0	NRS	0
0	1	0	1	NRS	0
0	1	1	0	X	X
0	1	1	1	ORS	1
1	0	0	0	NRS	1
1	0	0	1	ORS	1
1	0	1	0	X	X
1	0	1	1	ORS	0
1	1	0	0	NRS	1
1	1	0	1	ORS	1
1	1	1	0	X	X
1	1	1	1	ORS	1

<표 1>에서, Temp C_{in}의 NRS 와 ORS 는 각각 C_{in}과 1 비트 정규화 쉬프트를 결정하기 위한 NRS 와 ORS 인 경우에 C_{in}값을 가정한 것이다. MSB 의 $C_{in}0$ 와 $C_{in}1$ 은 각각 최하위 비트로의 올림수가 '0'과 '1' 일 때 최상위 비트에서 올림수이다. 올림수 입력이 없는 경우의 수가 올림수 입력이 있는 경우의 수 보다 크다는 것은 옳지 않으므로 X(don't care)로 표시했다. <표 1>에서 1 비트 쉬프트와 결정된 C_{in}을 식으로 표현하면 다음과 같다.

$$\text{Shift} = C_{in}0_{MSB} \vee C_{in}^{NRS}C_{in}1_{MSB} \quad (22)$$

$$C_{in} = C_{in}^{NRS}\overline{C}_{in}0_{MSB} \vee C_{in}^{ORS}C_{in}0_{MSB} \quad (23)$$

<표 1>에서 뺄셈인 경우에는 ORS, NRS 를 각각 NLS, OLS 로 바꾸고 곱셈인 경우에는 RS, NRS 로 바꾸면 C_{in}과 1 비트 정규화 쉬프트를 구할 수 있다. 식 (22)와 (23)도 같은 방법으로 바꾸면 된다.

III. 결론

본 논문에서는 덧셈과 곱셈에서의 병렬화 라운딩 알고리즘을 구하는 식을 제안하였으며, 이때, 각 라운드 모드에 따르는 C_{in} 값 결정식이 같으므로 이 부분의 하드웨어를 통합할 수 있다. 곱셈기에서는 [5]와는 달리 P 를 모든 라운딩에서 동일하게 함으로써 반가산기 부분에 이 값을 지연시간 없이 공급해 줄 수 있고, 3 비트 덧셈기를 이용하면 간단히 r_n, r_{n-1}, r_{n-2}, C_{in}^{in},C_{in-1}^{in}을 구할 수 있다. 결과 값 정규화를 위한 1 비트 쉬프트를 할 때는 C_{in}결정과 1 비트 쉬프트 결정을 덧셈, 뺄셈, 곱셈에서 동일하게 함으로써 하드웨어를 줄일 수 있다.

제안된 알고리즘은 Verilog-HDL 로 모델링하여 검증하였다. 이 때 덧셈기의 복잡가산기는 올림수 선택 덧셈기(Carry Select Adder)를 기반으로 하여 입력 올림수가 '0' 일 때와 '1' 일 때 두 가지 결과 값을 출력하는 53 비트 덧셈기로 모델링 하였고, 곱셈기는 하드웨어의 절감과 고속 연산 수행을 고려하여 Booth 곱셈기와 배열 곱셈기(Array multiplier)를 이용하여 53 비트 곱셈기를 모델링 하였다[6].

참고문헌

[1] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985. New York, The Institute of Electrical and Electronics Engineers, Inc., Aug., 1985.

[2] Nhon Quach and Michael Flynn, "An improved algorithm for high-speed floating-point addition," Technical Report CSL-TR-91-501, Stanford University, Aug., 1990.

[3] Nhon Quach and Michael Flynn, "Design and implementation of the SNAP floating-point adder," Technical Report CSL-TR-91-501, Stanford University, Dec., 1991

[4] Nhon Quach, "On fast IEEE rounding," Technical Report CSL-TR-91-459, Stanford University, Jan., 1991.

[5] 박우찬 외 3 명, "IEEE 반올림과 덧셈을 동시에 수행하는 부동 소수점 곱셈 연산기 설계," 전자공학회논문지, C 편, 제 11 호, pp. 47-55, 1997.

[6] 이원희, 강준우, "1998 년도 추계종합학술대회 논문집," 대한전자공학회.