論文2002-39SD-7-8

내장형 프로세서를 위한 IEEE-754 고성능 부동소수점 나눗셈기의 설계

(IEEE-754 Floating-Point Divider for Embedded Processors)

鄭 在 元 *, 洪 仁 杓 *, 鄭 愚 暻 *, 李 溶 錫 *

(Jaewon Jeong, In-Pyo Hong, Woo-Kyong Jeong, and Yong-Surk Lee)

요 약

최근 컴퓨터 그래픽이나 고급 DSP 등 부동소수점 연산의 활용 분야가 늘어나면서 나눗셈 연산의 필요성이 증대되었으나, 기존의 나눗셈 연산기는 큰 하드웨어 면적을 차지할 뿐만 아니라 전체 부동소수점 연산의 병목현상을 초래하는 중요한 요인이 되고 있다. 본 논문에서는 급수 전개 알고리즘을 이용하여 내장형 프로세서에 적합하도록 소면적의 부동소수점 나눗셈기를 설계하였다. 나눗셈기는 SIMD-DSP 유닛의 두 개의 곱셈누적기를 공유하여 연산함으로써, 부동소수점 단정도 형식의 나눗셈 연산을 고속으로 수행함과 동시에 나눗셈 연산을 위한 추가 면적을 최소화하였다. 본 논문에서는 급수 전개 알고리즘 나눗셈 연산기를 설계함에 있어 고려되어야할 오차의 분석을 통해 정확한 라운딩을 위한 몫을 얻어낼 수 있는 구조를 선택하였으며, IEEE-754 표준에서 정의하고 있는 모든 라운딩 모드를 지원하도록 하였다.

Abstract

As floating-point operations become widely used in various applications such as computer graphics and high-definition DSP, the needs for fast division become increased. However, conventional floating-point dividers occupy a large hardware area, and bring bottle-necks to the entire floating-point operations. In this paper, a high-performance and small-area floating-point divider, which is suitable for embedded processors, is designed using the series expansion algorithm. The algorithm is selected to utilize two MAC(Multiply-ACcumulate) units for quadratic convergence to the correct quotient. The two MAC units for SIMD-DSP features are shared and the additional area for the division only is very small. The proposed divider supports all rounding modes defined by IEEE 754 standard, and error estimations are performed for appropriate precision.

Ⅰ. 서 론

최근 각광받고 있는 멀티미디어 통합 기기들은 모두 고성능의 정수, 부동소수점 처리능력을 바탕으로 하고 있으며, 보다 나은 수준의 서비스를 제공하기 위한 프로그램들이 등장함에 따라 하드웨어의 성능 향상을 위한 노력도 계속되어지고 있다. 이러한 멀티미디어 환경이 널리 일반화됨에 따라, 낮은 비용이 중요한 내장형 프로세서에서도 이러한 고성능의 수치처리능력에 대한 요구가 증대되고 있다.

부동소수점의 연산기의 발전은 사용빈도가 높은 가감산기와 곱셈기의 발전을 중심으로 이루어져 왔으며, 상대적으로 사용빈도가 낮은 나눗셈기는 성능이 크게 향상되지 않았다. 그러나 멀티미디어 분야에서의 나눗셈기 성능은 부동소수점 연산기 전체의 처리능력을 저

* 正會員, 延世大學校 電氣電子工學科
(Department of Electrical and Electronic Engineering, Yonsei University)

接受日字:2001年3月5日, 수정완료일:2002年5月11日

하시키는 요인으로 작용되고 있으며, 나눗셈기의 대기 시간의 단축은 멀티미디어 연산을 수행하는 부동소수점 연산기의 성능향상에 필수적이라 할 수 있다.[1]

대부분의 고성능 부동소수점 연산기는 SRT 알고리즘을 이용한 나눗셈기를 채택하고 있으나, 뺄셈을 기본적인 동작으로 하는 SRT 나눗셈기는 몫을 결정하기 위한 시간이 비트 수가 증가함에 따라 선형적으로 증가한다는 단점을 가지고 있다. 고성능을 얻기 위해서는 큰 기수(radix)의 SRT 나눗셈기를 설계되어야하므로 이로 인한 면적의 증가 또한 피할 수 없다.

본 논문에서는 내장형 프로세서에 적합하도록 하드웨어 면적을 최소화하기 위하여, 기존의 내장형 프로세서 코어에 내장되어 있는 두 개의 곱셈기를 이용하여 급수 전개(series expansion) 방법을 사용한 부동소수점 단정도 나눗셈기를 설계하였다. 본 논문의 나눗셈기는 MAC 연산기를 공유하므로 나눗셈기만을 위한 하드웨어가 매우 작으며, 한 사이클 당 결정되는 몫의 비트 열이 두 배로 증가하므로 고속으로 나눗셈을 수행할 수 있다.[2] 또한, IEEE-754 부동소수점 표준안을 만족하는 4가지 라운딩 모드를 지원한다.

Ⅱ. Series Expansion Algorithm[4]

많이 사용되는 부동소수점 나눗셈기의 알고리즘은 크게 두 가지로 나뉜다. 하나는 뺄셈을 이용한 방식으로 radix-4 SRT가 대표적이며, 다른 하나는 곱셈을 이용하는 방식으로 Newton-Raphson 방식과 급수 전개 방식이 있다.[1] Radix-4 SRT 방식은 가장 간단하고 나눗셈기를 독립적으로 구현할 수 있어, 가장 널리 사용되었으나, 데이터의 길이에 따라 지연시간이 선형적으로 증가하는 단점이 있다. 곱셈을 이용한 방식은 곱셈기를 공유함으로써 하드웨어 면적을 줄일 수 있으나, 곱셈과 나눗셈을 독립적으로 수행할 수 없어 곱셈기에서 성능의 병목이 발생할 수 있고, 곱셈기의 설계가 복잡해 질 수 있다는 단점을 가지고 있다.

곱셈 방식의 나눗셈을 사용한 기존의 구현은 곱셈기를 하나만 사용하여, 긴 데이터의 나눗셈에서 몫에 빠르게 수렴하는 장점을 크게 살리지 못했기 때문에 면적 대 성능비에서 SRT에 뒤쳐지는 것으로 알려져, 많이 사용되지 않았다.[3] 본 논문에서는 32bit의 곱셈누적기 두 개를 사용하여 단정도 나눗셈의 지연시간을 6 cycle로 크게 단축시켰으며, 곱셈 방식의 장점인 작은 하드웨어 면적으로 고성능 내장형 프로세서에 적합한 부동소수점 나눗셈기를 설계하였다.

두 개의 32bit 곱셈누적기는 적은 양의 하드웨어는 아니지만, 64bit 마이크로프로세서에서는 멀티미디어 성능 향상을 위해 SIMD 가속을 흔히 지원하고 있으므로 충분히 활용할 수 있는 하드웨어이다. 두 개의 곱셈누적기를 활용하여 나눗셈기를 설계하기 위해서는 각 곱셈 연산에 의존성이 존재하는 Newton-Raphson 방식보다 곱셈 연산을 독립적으로 수행할 수 있는 급수 전개 방식이 적합하다.

급수 전개 방식을 이용한 나눗셈기의 단점은 곱셈기의 공유로 인한 성능의 병목현상과 오차의 누적으로 인한 정밀도의 소실이 있다. Oberman과 Flynn의 연구결과에 의하면 곱셈기의 공유로 인한 성능 저하는 6 cycle의 나눗셈 지연시간에서는 0.02 CPI 정도로 극히 경미한 것으로 알려졌다.[4] 오차의 누적으로 인한 문제는 곱셈누적기를 이용한 라운딩을 통해 해결하였으며, 4장에 오차의 분석을 보였다.

1. 급수 전개 알고리즘

급수 전개 알고리즘은 Goldschmidt 알고리즘으로도 알려져 있는데, Newton-Raphson알고리즘이 각 반복 연산마다 종속성을 갖는 두 번의 곱셈이 필요한 반면, 급수 전개 알고리즘에서는 독립적인 두 번의 곱셈을 수행하므로 두 곱셈의 병렬성을 이용하여 성능 향상을 꾀할 수 있다. 급수 전개 알고리즘을 살펴보면 다음과 같다.

$$g(y) = a \times \frac{1}{1+y} = 1 - y + y^2 - y^3 + y^4 - \cdots \qquad (1)$$

위의 식 (1)에서 g(y)=a/b를 얻기 위해 y=b-1을 대입하고, 인수분해하여 아래와 같은 식(2)를 얻는다.

$$Q = a \times [(1-y)(1+y^2)(1+y^4)(1+y^8)\cdots] \qquad (2)$$

식 (2)의 전개식은 다음의 식 (3)의 관계를 갖고 있으며, Ni, Di, qi는 각각 i 번째 iteration을 거친 후의 분자, 분모, 몫이다. 이 식에서 Di를 1로 수렴시킨다면 Ni는 Q로 수렴되어 몫을 구할 수 있게된다.

$$q_i = \frac{N_i}{D_i} \qquad (3)$$

위 식 (3)의 초기값을 $N_0=a$, $D_0=b$로 치환하고, 첫 iteration의 동작을 $N_1=R_0\times N_0$, $D_1=R_0\times D_0$라 하고 R_0가 $R_0=1-y=2-b$라 한다면, iteration 후의 N_1, D_1은 다음과 같이 구할 수 있다.

$$\begin{aligned} N_1 &= N_0\times R_0 = a\times(1-y) \\ D_1 &= D_0\times R_0 = b\times(1-y) \\ &= (1+y)(1-y) = 1-y^2 \end{aligned} \tag{4}$$

다음 iteration을 위해 $R_1=2-D_1$이라 하면, 다음 iteration에서는

$$\begin{aligned} R_1 &= 2-D_1 = 2-(1-y^2) = 1+y^2 \\ N_2 &= N_1\times R_1 = a\times[(1-y)(1+y^2)] \\ D_2 &= D_1\times R_1 = (1-y^2)(1+y^2) = (1-y^4) \end{aligned} \tag{5}$$

가 되고, 위의 과정들을 일반화하면, 다음과 같은 iteration 함수들이 얻어진다.

$$\begin{aligned} N_{i+1} &= N_i\times R_i \\ D_{i+1} &= D_i\times R_i \\ when,\quad R_{i+1} &= 2-D_i \end{aligned} \tag{6}$$

위의 식(4),(5),(6)을 이용하여 i 번째 iteration 후의 Ni와 Di를 구하면

$$\begin{aligned} N_i &= a\times[(1-y)(1+y^2)(1+y^4)\ldots(1+y^{2i})] \\ D_i &= (1-y^{2i}) \end{aligned} \tag{7}$$

가 된다. 이때 식 (7)과 식 (2)를 비교하여 살펴보면 Ni가 q로, Di가 1로 수렴한다는 사실을 알 수 있으며, b가 $0.5\le b<1$의 범위에서는 각 iteration의 보정 항인 $(1+y2i)$이 몫 q_i의 정밀도를 2배로 증가시킴을 확인할 수 있다. 그러므로 원하는 정밀도를 얻을 때까지 계속해서 위의 과정을 반복하여 몫을 계산해낼 수 있다.

2. Rounding 동작

곱수 전개 알고리즘을 위해서는 원하는 정밀도보다 2비트의 정밀도가 더 필요하게 된다.[7]

$$-2^{-(pc+2)} < Q-Q' < 2^{-(pc+2)} \tag{8}$$

이와 더불어 라운딩을 위해서는 guard bit와 함께 sticky bit도 필요한데, 이에 해당하는 값을 얻기 위해 또 한번의 곱셈을 수행해야 한다.

$$R = a - b\times Q'' \tag{9}$$

표 1. 각 라운딩 모드의 조건과 동작
Table 1. The action for each rounding mode.

Guard bit	Remainder	RN	RP	RM	RZ
0	=0	trunc	trunc	trunc	trunc
0	-	trunc	trunc/dec	dec/trunc	dec
0	+	trunc	inc/trunc	trunc/inc	trunc
1	=0	RNE	inc/trunc	trunc/inc	trunc
1	-	trunc	inc/trunc	trunc/inc	trunc
1	+	inc	inc/trunc	trunc/inc	trunc

나머지 계산에 쓰여진 Q''는 아래의 범위를 가지며,

$$-2^{-(pc+1)} < Q-Q'' < 2^{-(pc+1)} \tag{10}$$

Q''를 식 (6)에 대입하여 얻어진 나머지의 부호와 절대값 Q''의 마지막 비트인 guard bit를 이용하여 아래의 표 1과 같이 rounding을 수행한다. 이때 표 1에서 나머지 열의 부호는 나머지의 부호를 의미하며, 각 라운딩 모드에서의 부호는 계산된 몫의 부호를 뜻한다.[7]

Ⅲ. 나눗셈기의 설계

1. 나눗셈기의 전체 동작

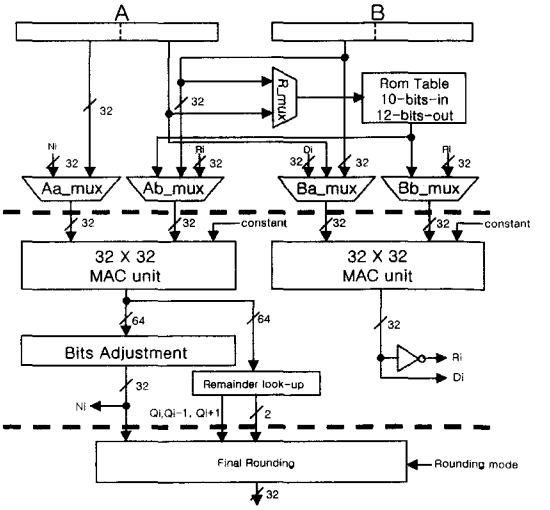


그림 1. 나눗셈기의 전체 block diagram
Fig. 1. Block diagram of the divider.

본 논문의 나눗셈기에서는 단순 곱셈만을 수행하는 곱셈기 대신 곱셈 후 덧셈을 수행하는 MAC 유닛을 사

용하여 곱셈과 덧셈을 동시에 수행하게 하였다. MAC 유닛은 1 사이클에 곱셈과 덧셈을 동시에 수행하도록 설계되었으며, 본 나눗셈기에서는 이러한 MAC 유닛 2개를 공유하여 나눗셈 연산의 속도를 증가시키면서 동시에 면적 증가를 최소로 하는 것을 설계의 목표로 삼았다.

다음의 표 2는 본 논문에서 설계된 나눗셈기의 동작을 사이클에 따라 설명하고 있으며, 롬 테이블 액세스에 1 사이클, iteration 곱셈에 2 사이클, 마지막 몫을 얻어내는 곱셈에 1 사이클, 나머지를 구하는 과정의 1 사이클, 라운딩 처리에 1 사이클 등, 총 6 사이클을 통해 나눗셈 연산을 수행하게 된다.

2. 나눗셈기의 연산 동작

(1) ROMACCESS

롬 테이블은 10 비트 입력에 대한 초기 근사화된 역수 12 비트를 제공하여, 빠르게 몫에 수렴할 수 있도록 사용된다. 12비트 중 첫 번째 비트는 항상 1이므로 롬 테이블에 저장할 필요가 없다. 그러므로 롬 테이블은 총 11 Kbit의 사이즈를 가진다.

(2) IMMMUL

몫으로 수렴해 가는 과정의 반복되는 곱셈으로서, 32 비트로 반올림된 Ni, Di, Ri를 입력으로 하여 Ni×Ri, Di×Ri를 수행한다. 각 연산의 곱은 표 3과 같이 라운딩 상수가 더해지고, 1 비트 왼쪽으로 쉬프트 되어 다음 연산의 입력으로 사용된다. Di는 1의 보수화 되어 Ri를 구성한다.[8]

(3) LASTMUL

마지막 몫을 구하는 곱셈으로, 전 단계의 Ni와 Ri를 입력으로 하여 곱셈을 수행한다. 최종 몫은 (0.5,2)의 범위를 갖게 되므로 왼쪽으로 한 비트 쉬프트될 수 있다. 따라서 라운딩을 위한 조건을 만족시키기 위해서는 2-(pc+3)까지의 정밀도를 갖게 출력되어 라운딩 상수와 더해지게 된다.

(4) REMMUL

나머지를 계산하기 위한 연산으로 LASTMUL에서 구해진 몫의 guard bit까지를 마스킹하여 입력으로 사용한다. 또한 표 3과 같이 피젯수의 반전된 값을 라운딩 상수로 입력하고, 덧셈기의 쓰이지 않는 최하위 자리올림 비트를 1로 세팅함으로써 뺄셈을 수행하게 된다. 이는 하드웨어의 설계를 간단하게 하기 위하여 B×Q''-A의 연산을 수행하는 것이다.

표 2. 각 사이클에 따른 나눗셈기의 동작
Table 2. Cycle for each division action.

사이클	동 작
	Input=(a, b, pk, rm), Output=(q)
1	Table look-up, x_0=ROMACCESS(b)
2	d_0=IMMMUL(x_0, b), n_0=IMMMUL(x_0, a), r_0=1's comp(d_0)
3	d_1=IMMMUL(r_0, b), n_1=IMMMUL(r_0, a), r_1=1's comp(d_1)
4	q_i=LASTMUL(r_1, n_1)
5	rem=REMMUL(q_i, b, a), (q_i+1, q_i-1)=ADD(q_i, 1, -1)
6	q=ROUND(q_i, q_i+1, q_i-1, rem, rm)

(5) REMLOOKUP

Qi+1, Qi-1을 계산하고, 나머지의 최상위 비트와 전 비트를 논리합하여 라운딩에서 필요한 나머지의 부호와 절대값을 계산해낸다. 라운딩 시 선택되어지는 Qi+1과 Qi-1은 나머지를 구하는 곱셈 REMMUL과 병렬적으로 수행되므로 추가 지연이 발생하지 않는다.

(6) ROUND

라운딩되기 전의 몫인 Qi는 원하는 정밀도인 (pc+1) 비트로 ±0.5 ulp의 오차 이내의 값을 갖게 된다. 라운딩 유닛은 REMLOOKUP 유닛의 출력인 나머지의 부호와 절대값을 입력받아 Q_i, Q_i+1, Q_i-1 중의 하나의 값을 선택하게 된다. 이 때, 라운딩의 결과는 표 1의 조건에 따라 결정되어 진다.

Ⅳ. 오차분석

본 나눗셈기의 정밀도는 significand의 비트 수에 의해 결정되며 입력과 출력의 significand는 모두 IEEE-754 부동소수점 연산의 표준에 의해 24비트로 고정되어 있다. 그러나 몫으로 수렴하기 위해 사용되는 곱셈기의 입력 오퍼랜드는 32 비트이며, significand는 정수 1을 포함하고 있으므로 곱셈기 입력 ulp의 정밀도는 2^{-31}이 된다. 따라서 각 iteration의 곱셈의 입력은 32 비트로 라운딩되어야 하고, 이로 인한 오차를 E_m이라 한다면 E_m은 아래와 같은 오차 범위를 갖는다.

$$-2^{-32} \le E_m \le 2^{-32} \quad (11)$$

표 3. 곱셈 누적기에 입력되는 라운딩 상수
Table 3. Rounding constant.

동 작	라 운 딩 상 수
IMMMUL	{25 b0,1 b1,6 b0}
LASTMUL	{32 b0,1 b1,31 b0}
REMMUL	{!Dividend,8 b1}, carry LSB=1

또한 알고리즘에서 D_i의 2의 보수를 계산해 R_i를 얻어내는 과정을 하드웨어의 성능을 증대시키기 위하여 1의 보수를 구하는 논리 연산으로 대치하였으므로 이에 따라 발생되는 오차가 존재하게 된다. 2의 보수 연산을 1의 보수 연산으로 대체함으로써 발생되는 오차를 E_{ones}라고 하면, E_{ones}는 항상 일정한 오차, -ulp를 갖게되며 그 값은 아래와 같다.

$$E_{ones} = -2^{-31} \tag{12}$$

이와는 별도로 연산이 시작되기 전, 초기값을 얻기 위해 역수값을 근사화하여 사용했으므로 이로 인한 초기 오차가 존재하며, 이는 $E_{x0}<2^{-10}$의 범위를 갖는다.

위에서 언급한 3가지의 오차항을 포함한 각 사이클별 연산은 다음과 같이 진행된다. 첫 번째 사이클에서는 롬 테이블을 액세스하며, 근사화된 역수를 얻어낸다. 따라서 E_{x0} 항이 생성됨을 알 수 있다.

$$x0 = \frac{1}{b} + E_{x0} \tag{13}$$

두 번째 사이클에서는 D_0, N_0, R_0가 각각 계산되며, N_0와 D_0를 얻기 위해서는 각각 곱셈이 필요하므로 E_m이 첨가되며, R_0를 구하기 위해서는 1의 보수화 과정이 필요하므로 E_{ones}가 첨가된다.

$$\begin{aligned} D_0 &= (\frac{1}{b} - E_{x0}) \times b + E_m \\ R_0 &= 2 - (\frac{1}{b - E_{x0}}) \times b - E_m + E_{ones} \\ N_0 &= (\frac{1}{b} - E_{x0}) \times a + E_m \end{aligned} \tag{14}$$

세 번째 사이클에서도 두 번째 사이클과 같은 연산이 수행되므로 아래와 같은 연산결과를 얻게 된다.

$$\begin{aligned} D_1 = & -E_m^2 + E_m E_{ones} - 2bE_{x0}E_m + E_m \\ & - bE_{x0}E_{ones} + E_{ones} + 1 - b^2E_{x0}^2 \\ N_1 = & -E_m^2 + E_m E_{ones} + 2E_m - bE_{x0}E_m \\ & - aE_{x0}E_m - \frac{aE_m}{b} + \frac{aE_{ones}}{b} \\ & + aE_{x0}E_{ones} - aE_{x0}^2 b + \frac{a}{b} \\ R_1 = & \ 2 - \left\{(\frac{1}{b} - E_{x0})b + E_m\right\} \\ & \times \left\{2 - (\frac{1}{b} - E_{x0})b - E_m + E_{ones}\right\} \\ & - E_m + E_{ones} \end{aligned} \tag{17}$$

그러나 곱셈기의 입력 오퍼랜드가 2^{-31}까지를 허용하므로 2^{-62} 이하의 오차는 연산의 결과 식에서 고려하지 않아도 된다. 따라서 식 (15)의 E_m^2이나 $E_m \times E_{ones}$ 등은 제외될 수 있으며, 식 (15)는 다음과 같이 간략화될 수 있다.

$$\begin{aligned} D_1 = & -2bE_{x0}E_m + E_m + E_{ones} + 1 - b^2E_{x0}^2 \\ N_1 = & \ 2E_m - bE_{x0}E_m - aE_{x0}E_m + \frac{aE_{ones}}{b} \\ & - \frac{aE_m}{b} + aE_{x0}E_{ones} - aE_{x0}^2 b + \frac{a}{b} \\ R_1 = & \ 1 + 2bE_{x0}E_m - E_m + b^2E_{x0}^2 \end{aligned} \tag{18}$$

네 번째 사이클은 위의 식 (16)에서 얻어진 N_1과 R_1을 곱하여 라운딩에 사용할 몫인 Q_i를 계산한다. 위와 같은 방법으로 곱셈기의 정밀도 이하의 오차항들을 제거한 후의 Q_i는 다음의 식 (17)와 같다.

$$\begin{aligned} Q_i = & -bE_{x0}E_m + 2E_m + aE_{x0}E_m - 2\frac{aE_m}{b} \\ & - b^3E_{x0}^3E_m - 3aE_{x0}^3b^2E_m + 2b^2E_{x0}^2E_m \\ & + aE_{x0}^3b^2E_{ones} + aE_{x0}E_{ones} + \frac{aE_{ones}}{b} \\ & + abE_{x0}^2E_{ones} - aE_{x0}^4b^3 + \frac{a}{b} \end{aligned} \tag{19}$$

식 (17)에서 얻어진 Q_i는 2절의 식 (10)을 만족해야 정확히 라운딩된 몫을 얻을 수 있다. $E_{x0}<2^{-10}$, $|E_m|<2^{-32}$, $E_{ones} = -2^{-31}$, $1 \le a,b < 2$의 조건을 염두에 두고 식 (17)을 살펴보면, E_{x0}는 식 (18)에서 고려될 항을 제외하면 모두 E_m과 E_{ones}와 곱해지므로 2^{-40} 이하의 오차를 발생시키며, E_m과 E_{ones}만을 포함하는 항이 최대의 오차를 구성한다. 근사화된 오차 E_q는 $Q_i-(a/b)$로 나타낼 수 있으며, 다음의 식 (18)과 같다.

$$E_q = 4E_m + 2E_{ones} - aE_{x0}^4b^3 \tag{20}$$

위의 식 (18)에서 마지막 항인 $-aE_{x0}4b3$은 초기 근사 역수값에서 비롯된 것이며, E_{x0}가 $E_{x0}<2^{-10}$이므로 최종

몫의 연산에 영향을 주지 못한다. 그러므로, 앞의 두 항에서 발생될 수 있는 최대 오차가 라운딩에서 고려되어야할 값들이며, E_m을 최대 2^{-31}이라 가정하더라도 E_q는 다음의 범위에 존재하게 된다.

$$-2^{-28} < E_q < 2^{-28} \tag{21}$$

본 논문의 2.2절에서 IEEE-754 표준안의 4가지 라운딩 모드를 수행하기 위한 몫의 범위를 식 (10)과 같이 규정하였으며, 부동소수점 단정도 연산에서의 LSB는 2^{-23}이므로 라운딩으로 인한 1 비트의 쉬프트를 고려한다면 E_q는 다음의 범위에 존재하여야 한다.

$$\begin{aligned} -2^{-(23+2+1)} &< E_q < 2^{-(23+2+1)} \\ -2^{-26} &< E_q < 2^{-26} \end{aligned} \tag{22}$$

그러므로 식 (20)의 오차 범위는 올바른 IEEE-754의 라운딩을 수행할 수 있는 범위에 포함된다. 또한 기호계산 프로그램인 Maple^{TM}V를 이용하여 입력의 범위 안에서 계산된 오차의 범위는 그림 2와 같다.

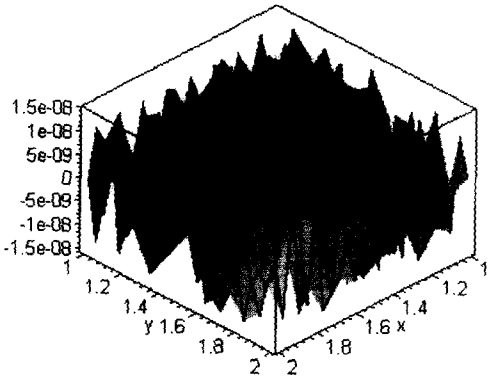


그림 2. 입력에 따른 오차의 분포
Fig. 2. Worst case error in last iteration.

V. 검증과 합성 및 평가

1. 기능 검증

본 논문에서는 부동소수점 나눗셈 연산기의 동작을 확인하기 위해 Verilog HDL(Hardware Description Language)을 사용하여 기능 수준과 게이트 수준에서 시뮬레이션을 수행하여 검증하였다. 먼저 series expansion의 알고리즘의 나눗셈기를 C 언어로 기술하여 기존의 나눗셈의 결과와 같은 결과를 계산해내는지를 확인하는 작업을 통해 series expansion 알고리즘을 검증하고, 각 라운딩 모드로 임의의 벡터 10만개를 입력하여 각 라운딩 알고리즘의 올바른 동작을 검증하였다. 또한 테이블 값의 최대의 오차를 발생시키는 벡터를 추가로 입력함으로써, 오차 분석을 통해 결정된 허용 오차의 범위 내에서 나눗셈기가 동작하고 있는지를 확인하였으며, 사용된 모든 입력 벡터는 HDL의 기능 수준의 시뮬레이션과 게이트 레벨의 시뮬레이션에 동일하게 사용함으로써 HDL 기술단계에서 발생할 수 있는 오류를 검증하는데 활용하였다.

표 4. 각 유닛의 게이트 수와 최대 지연 시간
Table 4. Maximum delay of the units in divider.

unit	gate	delay	면적 비(%)
Divider	57211	17.43 ns	100
MAC	47183.2	15.80 ns	82.47
ROM table	4071.4	-	7.12
REMLOOKUP	2217.8	10.70 ns	3.88
ROUND	297.8	0.54 ns	0.52
Control unit	206.2	0.86 ns	0.36
BITMASK	49.8	0.36 ns	0.09

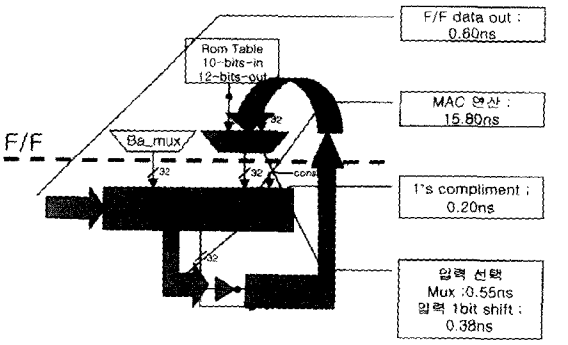


그림 3. 나눗셈기의 임계 경로
Fig. 3. Critical path.

2. 회로의 합성

테스트된 HDL 코드는 0.35㎛의 표준 셀 라이브러리(standard cell library)를 사용하여 Synopsys® 툴로 합성을 수행하였으며, 최악 조건 3.0V, 85℃에서의 합성 결과는 표 4와 같다.

그림 3에서와 같이 본 논문의 나눗셈기에서 곱셈기의 최대 경로 지연에 추가되는 시간은 mux의 지연 시간과 1의 보수를 구성하는 인버터의 지연 시간, 1비트

쉬프터의 지연시간만을 더한 것이다. 최악 조건에서 mux의 지연 시간은 0.55ns, 1비트 쉬프터의 지연시간은 0.38ns이며, 합성 과정에서 덧붙여진 버퍼 등을 포함해서 곱셈누적기의 최대 경로 지연에 1.72ns가 추가로 지연된다. 또한 곱셈 누적기 2개의 면적은 47183.2 게이트로 이것은 롬 테이블을 포함한 전체 나눗셈기의 면적의 82.5%를 차지하였으며, 그 나머지 17.5%인 약 1 만 게이트 정도만이 나눗셈 연산을 위해 추가된 하드웨어이다. 기존 radix-4 SRT나 8-bit seed Goldschmidt 나눗셈기는 대략 4070rbe 정도인데,[4] 1rbe가 약 2.5 게이트라고 봤을 때 이와 비슷한 정도의 하드웨어라 볼 수 있다. 나눗셈을 위한 유닛들은 임계경로에 포함된 Bit adjust 블록을 제외하면 최대 경로 지연에 영향을 미치지 않으며, 나눗셈기의 전체 최대 경로는 함께 구현된 마이크로프로세서의 최대 지연인 20ns 보다 작으므로, 동작 주파수에 영향을 주지 않는다.

3. 다른 나눗셈기와의 비교 및 평가

기존의 FPU의 나눗셈기들은 주로 SRT, Newton-Raphson 등의 알고리즘을 사용하여 구현되었다. 표 5는 상용화 된 여러 프로세서에 적용된 나눗셈기들과 본 논문의 나눗셈기의 수행 사이클을 나타내고 있다. 성능비 항목은 본 논문에서 설계한 나눗셈기의 성능 대비 각 나눗셈기의 성능을 수행 사이클 수에 근거해 비율로 나타낸 것이다. 여기서 보는 바와 같이 본 논문에서 설계한 나눗셈기는 지연 사이클 면에서 월등한 성능을 가지고 있으며, 사이클 시간을 고려하더라도 적은 면적 증가로 고성능의 나눗셈 연산을 지원할 수 있음을 보여주고 있다.

VI. 결 론

본 나눗셈기는 SRT 알고리즘을 탈피하여, 곱셈기를 공유하는 방식의 나눗셈기를 설계하였으며, 기존의 프로세서 설계에 포함되어 있는 2개의 MAC 유닛을 공유하여 병렬적인 곱셈을 통해 성능 향상을 얻을 수 있는 구조를 제시하였다. 아울러 나눗셈 연산을 위해 MAC 유닛에 추가되는 면적을 줄임으로써 소면적의 추가로 곱셈, 곱셈 누적 연산, 나눗셈 연산을 하나의 유닛으로 수행할 수 있다. 또한 IEEE-754 부동소수점 표준의 모든 라운딩 방식을 지원하며 부동소수점 단정도 연산을 6 사이클만에 수행한다.

이와 같이 본 논문에서는 최소한의 면적 증가로 고성능의 나눗셈 연산을 수행하는 유닛을 설계하였으며, 급수 전개 방식의 나눗셈기를 설계함에 있어 고려되어야 할 곱셈 유닛의 정밀도에 따른 몫의 오차를 분석하여 IEEE-754 표준의 라운딩 동작을 보장함으로써 면적 증가에 민감한 내장형 프로세서에 멀티미디어 연산에 필수적인 고속의 나눗셈 연산을 지원하는 하드웨어를 적은 비용으로 구현하였다. 앞에서 제시한 곱셈의 병렬성을 이용한 나눗셈 연산기의 올바른 동작은 게이트 수준의 시뮬레이션을 통해 확인, 검증하였다.

표 5. 다른 나눗셈기와의 연산 수행 사이클 비교

Table 5. Latency comparison.

Processor	Latency (cycles)	perf. ratio (%)
Intel i486	73*(35)	17.14
MIPS R4000	23	26.09
SPARC compatible	20~23	30
Intel Pentium(radix-4 SRT)	19	31.58
UltraSPARC(radix-8 SRT)	12	50
IBM RISC/6000 (Newton-Raphson)	19	31.58
Elbrus E2k	10~13	60
Intel Pentium III	18~36	33.33
본 논문의 나눗셈기 (series expansion)	6	100

참 고 문 헌

[1] Stuart F. Oberman and Michael J. Flynn, "Division Algorithms and Implementations," *IEEE Transactions on Computers*, Vol. 46, No. 8, August 1997.

[2] Peter Soderquist and Miriam Leeser, "Division and Square Root Choosing the Right Implementation," *IEEE Micro*, July 1997, pp. 56 ~66.

[3] Peter Soderquist and Miriam Leeser, "An Area/Performance Comparison of Subtractive and Multiplicative Divide/Square Root Implementations," *Proc. 12th IEEE Symp. Computer Arithmetic*, IEEE, 1995, pp. 132~139.

[4] Stuart F. Oberman and Michael J. Flynn, "Design Issues in Division and Other Floating-Point Operations," *IEEE Transactions on Computers*, Vol. 46, No. 2, Feb. 1997, pp. 154~161.

[5] D. DasSarma and D. Matula, "Measuring the Accuracy of ROM Reciprocal Tables," *IEEE Transactions on Computers*, Vol. 43, No. 8, pp. 932~940, August 1994.

[6] Robert E. Goldschmidt, "Applications of Division by Convergence," *MS thesis, Dept. of Electrical Eng.*, Massachusetts Inst. of Technology, Cambridge, Mass., June 1964.

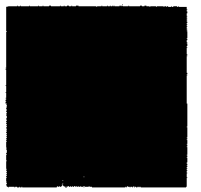
[7] Stuart F. Oberman, "Design Issues in High Performance Floating Point Arithmetic Units," *Ph.D. thesis, Stanford University*, Nov. 1996.

[8] Stuart F. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7TM Microprocessor," *Proc. 14th IEEE Symp. on Computer Arithmetic*, pp. 106~115, 1999.

저 자 소 개

鄭 在 元(正會員)

1972년 11월 10일생, 1999년 2월 연세대학교 전자공학과 공학사, 2001년 2월 연세대학교 전기컴퓨터공학과 공학석사, 현재 LG전자 액세스 망연구소에서 DSLAM 장치 개발. <주관심분야 : Computer Architecture, VLSI 설계, SOC, 고성능 연산기 설계>



鄭 愚 暻(正會員)

1974년 2월 8일생, 1996년 2월 연세대학교 전자공학과 공학사, 1998년 2월 연세대학교 전자공학과 공학석사, 1998년 3월~현재 연세대학교 전기전자공학과 박사과정. <주관심분야 : 마이크로프로세서 설계, VLSI 설계, 부동소수점 연산기 설계, 고성능 연산기 설계>

洪 仁 杓(正會員)

1976년 5월 28일생, 1999년 2월 연세대학교 전자공학과 공학사, 2001년 2월 연세대학교 전기컴퓨터공학과 공학석사, 2001년 3월~현재 연세대학교 전기전자공학과 박사과정. <주관심분야 : 마이크로프로세서 설계, VLSI 설계, 고성능 연산기 설계>

李 溶 錫(正會員)

1950년 10월 23일생, 1973년 연세대학교 전자공학과 공학사, 1977년 2월 University of Michigan Electrical Engineering 공학석사, 1981년 2월 University of Michigan Electrical Engineering 공학박사, 1993년~현재 : 연세대학교 전기전자공학과 교수. <주관심분야 : 마이크로프로세서 설계, VLSI 설계, DSP 프로세서 설계, 고성능 연산기 설계>