내장형 프로세서를 위한 IEEE-754 고성능 부동소수점 나눗셈기의 설계

정 재 원, 홍 인 표, 정 우 경, 이 용 석
연세대학교 전기전자공학과
전화 : 02-2123-2872

IEEE-754 Floating-Point Divider for Embedded Processors

Jaewon Joeng, In Pyo Hong, Woo Kyong Jeong, Yong Surk Lee
Dept. of Electrical and Electronic Engineering, Yonsei University
E-mail : ace@dubiki.yonsei.ac.kr

Abstract

In this paper, a high-performance and small-area floating-point divider, which is suitable for embedded processors and supports all rounding modes defined by IEEE 754 standard, is designed using the series expansion algorithm. This divider shares and fully utilizes the two MAC units for quadratical convergence to the correct quotient. The area increase of two MAC units due to the division is minimized in this design, so that it can be suitable for embedded processors.

The tested HDL codes are synthesized and optimized with 0.35㎛ CMOS standard cell libraries. The results show that the latency of the synthesized divider is 17.43 ns in worst condition. But, the divider calculates the correct rounded quotient through only 6 cycles.

I. 서론

최근 각광받고 있는 멀티미디어 프로세서는 모두 고성능의 정수, 부동소수점 처리능력을 바탕으로 하고 있으며, 하드웨어의 성능 증가와 함께 기존의 설계된 프로세서 코어를 내장한 형태의 프로세서들이 널리 사용되어지면서 소면적 설계에 대한 요구 또한 증대되고 있다. 부동소수점 연산기의 경우에는 사용빈도가 높은 가감산기와 곱셈기의 발전이 상대적으로 사용빈도가 낮은 나눗셈기에 비해 빠르게 이루어져 왔다. 그러나 멀티미디어 분야에서의 나눗셈기 성능은 부동소수점 연산기 전체의 처리능력을 저하시키는 요인으로 작용되고 있으며, 나눗셈기 지연 시간(latency)의 단축은 멀티미디어 연산을 수행하는 부동소수점 연산기의 성능 향상에 필수적이라 할 수 있다.

본 논문은 멀티미디어 내장형 프로세서의 고속 DSP SIMD 연산을 위해 설계된 2개의 MAC(Multiply and ACcumulate) 유닛을 공유하여 소면적의 추가로 부동소수점 나눗셈 연산을 고속으로 처리할 수 있는 나눗셈기를 설계하였다. 설계된 나눗셈 연산기는 급수 전개(series expansion) 알고리즘을 사용하며, ROM 테이블을 이용하여 빠르게 몫에 수렴할 뿐만 아니라, IEEE 754 표준의 모든 rounding 모드를 지원한다.

II. 급수 전개 알고리즘

2.1 급수 전개 알고리즘

급수 전개 알고리즘은 Goldschmidt 알고리즘으로도 알려져 있는데, Newton-Raphson알고리즘이 각 반복 연산(iteration)마다 종속성(dependency)을 갖는 두 번의 곱셈이 필요한 반면, 급수 전개 알고리즘에서는 독립적인 두 번의 곱셈을 수행하므로 두 곱셈의 병렬성

을 이용하여 성능 향상을 꾀할 수 있다.

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

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

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

식(2)의 전개식은 다음의 식(3)의 iteration 함수를 사용하여 구현할 수 있으며, N_i, D_i, q_i는 각각 i 번째 iteration을 거친 후의 분자, 분모, 몫이다. 이 식에서 D_i를 1로 수렴시킨다면 N_i는 Q로 수렴되어 몫을 구할 수 있게된다.

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

또한 각각의 N_i, D_i, R_i는 다음의 *iteration* 함수들을 통해 얻을 수 있다.

$$\begin{aligned} N_{i+1} &= N_i\times R_i \\ D_{i+1} &= D_i\times R_i \end{aligned} \quad (4)$$
$$when,\quad R_{i+1}=2-D_i$$

2.2 Rounding을 위한 알고리즘

급수 전개 알고리즘을 위해서는 원하는 정밀도보다 2 비트의 정밀도가 더 필요하게 된다.

$$-2^{-(pc+2)}<Q-Q'<2^{-(pc+2)} \quad (5)$$

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

$$R=a-b\times Q'' \quad (6)$$

나머지 계산에 쓰여진 Q''는 아래의 범위를 가지며,

$$-2^{-(pc+1)}<Q-Q'<2^{-(pc+1)} \quad (7)$$

Q''를 식(6)에 대입하여 얻어진 나머지의 부호와 절대값 Q''의 마지막 비트인 guard bit를 이용하여 아래의

표 1 각 라운딩 모드의 조건과 동작

Guard Bit	Rem-ainder	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

표 2 각 사이클에 따른 나눗셈기의 동작

사이클	동 작
	Input=(a, b, pk, rm), Output=(q)
1	Table look-up, x0=ROMACCESS(b)
2	d0=IMMMUL(x0, b), n0=IMMMUL(x0, a), r0=1's comp(d0)
3	d1=IMMMUL(r0, b), n1=IMMMUL(r0, a), r1=1's comp(d1)
4	qi=LASTMUL(r1, n1)
5	rem=REMMUL(qi, b, a), (qi+1, qi-1)=ADD(qi, 1, -1)
6	q=ROUND(qi, qi+1, qi-1, rem, rm)

표 1과 같이 rounding을 수행한다. 이때 표 1에서 나머지 열의 부호는 나머지의 부호를 의미하며, 각 라운딩 모드에서의 부호는 계산되어진 몫의 부호를 뜻한다.

III. 나눗셈기의 설계

3.1 나눗셈기의 전체 동작

본 논문의 나눗셈기에서는 단순 곱셈만을 수행하는 곱셈기 대신 곱셈 후 덧셈을 수행하는 MAC 유닛을 사용하여 곱셈과 덧셈을 동시에 수행하게 하였다. MAC 유닛은 1 사이클에 곱셈과 덧셈을 동시에 수행하도록 설계되었으며, 본 나눗셈기에서는 이러한 MAC 유닛을 2개 내장하고 있고, 이 2개의 MAC 유닛을 최대로 활용하여 나눗셈 연산의 속도를 증가시키며, 나눗셈을 위한 면적 증가를 최소로 하는 것을 설계의 목표로 삼았다.

표 2는 본 논문에서 설계된 나눗셈기의 동작을 사이

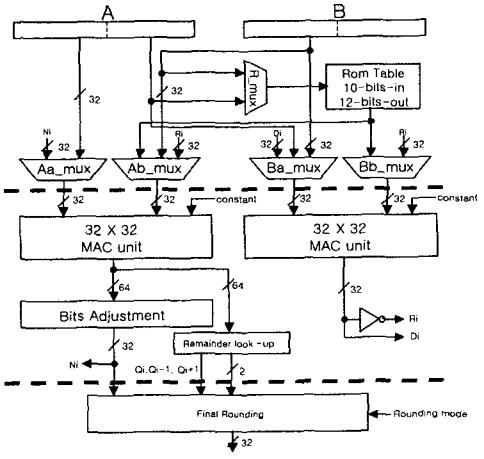

그림 1 나눗셈기의 전체 block diagram

클에 따라 설명하고 있으며, 각각의 동작들의 정의는 3.2 절에서 자세히 다루도록 한다. 따라서 본 논문의 나눗셈기는 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.

3.2 나눗셈기의 동작

(1) ROMACCESS

롬 테이블은 10 비트를 입력으로 받아 12 비트를 출력하며, 12비트 중 첫 번째 비트는 항상 1이므로 롬 테이블에 저장할 필요가 없다. 그러므로 롬 테이블은 총 11 Kbit의 사이즈를 가지며, 이는 나눗셈기에서 필요한 근사화된 역수를 제공하여, 빠르게 몫에 수렴할 수 있도록 설계되었다.

(2) IMMMUL

몫으로 수렴해 가는 과정의 반복되는 곱셈으로서, 32 비트로 반올림된 N_i, D_i, R_i를 입력으로 하여 $N_i \times R_i$, $D_i \times R_i$를 수행한다. 각 연산의 곱은 표 3과 같이 라운딩 상수가 더해지고, 1 비트 왼쪽으로 쉬프트 되어 다음 연산의 입력으로 사용된다. D_i는 1의 보수화 되어 R_i를 구성한다.

(3) LASTMUL

마지막 몫을 구하는 곱셈으로, 전 단계의 N_i와 R_i를 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 인버팅된 값을 라운딩 상수로 입력하고, 덧셈기의 쓰이지 않는 최하위 자리올림(carry) 비트를 1로 세팅함으로써 뺄셈을 수행하게 된다. 이는 하드웨어의 설계를 간단하게 하기 위하여 $B \times Q'' - A$의 연산을 수행하는 것이다.

(5) REMLOOKUP

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

(6) ROUND

표 3 곱셈 누적기에 입력되는 라운딩 상수

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

라운딩되기 전의 몫인 Q_i는 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. 오차 분석

본 나눗셈기의 곱셈기의 입력 오퍼랜드는 32 비트이며, significand는 정수 1을 포함하고 있으므로 곱셈기 입력 ulp(unit in the last position)의 정밀도는 2^{-31}이 된다. 따라서 각 iteration의 곱셈의 입력은 32 비트로 라운딩되어야 하고, 이로 인한 오차를 Em이라 한다면 Em은 아래와 같은 오차 범위를 갖는다.

$$-2^{-32} \le E_m \le 2^{-32} \tag{8}$$

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

$$E_{ones} = -2^{-31} \tag{9}$$

그리고 이와는 별도로 연산이 시작되기 전, 초기값을 얻기 위해 역수값을 근사화하여 사용했으므로 이로 인한 초기 오차가 존재하며, 이 값에 대한 상대 오차의 범위는 다음과 같다.

$$|r.e|_{\max} \le \frac{2^{k-1}+2^{k+g}}{2^n} \le \frac{2^{k-1}+2^{k+g}}{2^{2k+g+1}} \le 2^{-(k+1)}\left(1+\frac{1}{2^{g+1}}\right) \tag{10}$$

식(8)(9)(10)을 이용해 얻은 최종 라운딩 전의 오차항은 다음과 같다.

$$E_q = 4E_m + 2E_{ones} \pm aE_{x0}^4 b^3 \tag{11}$$

위의 식(11)에서 마지막 항인 $\pm aE_{x0}^4 b^3$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 가정하더라

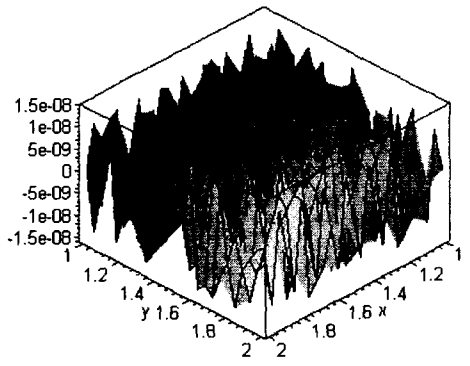

그림 2 입력에 따른 오차의 분포

도 E_q는 다음의 범위에 존재하게 된다.

$$-2^{-28} < E_q < 2^{-28} \qquad (12)$$

그러므로 식(12)의 오차 범위는 올바른 IEEE-754의 라운딩을 수행할 수 있는 범위에 포함된다. 또한 기호 계산 프로그램인 Maple™V를 이용하여 입력의 범위 안에서 계산된 오차의 범위는 그림 2과 같다.

V. 시뮬레이션 및 합성

본 논문에서는 부동소수점 나눗셈 연산기의 동작을 확인하기 위해 기능 수준과 게이트 수준에서 시뮬레이션을 수행하여 검증하였다. 먼저 C 언어로 기술된 급수 전개 알고리즘의 나눗셈기에 임의의 벡터 40만개를 입력하여 각 라운딩 모드로 알고리즘을 검증하고, 테이블 값의 최대의 오차를 발생시키는 벡터를 추가로 입력하여 검증하였으며, 이때 사용된 입력 벡터는 HDL의 기능 수준의 시뮬레이션과 게이트 레벨의 시뮬레이션에 사용하였다. 또한 테스트된 HDL 코드는 0.35㎛의 표준 셀 라이브러리(standard cell library)를 시용하여 Synopsys® 툴로 합성을 수행하였으며, 최악 조건 3.0V, 85℃에서의 합성 결과는 표 4와 같다.

표 4 각 유닛의 게이트 수와 최대 지연 시간

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

합성 결과, 나눗셈기 전체의 게이트 수는 57211개였으며, MAC 유닛이 나눗셈기에서 차지하는 게이트 수는 47183.2 게이트로, 전체의 82% 이상이었다. 따라서 나눗셈 연산을 위해 추가된 면적은 18% 이하이며, 최악 조건에서 나눗셈으로 인한 사이클 시간의 증가는 1.63 ns로, 마지막 곱셈을 위해 guard bit이하의 비트열을 마스킹하는 과정과 MAC 유닛의 입력을 선택하는 Mux의 지연 시간이 임계 경로에 포함되었다.

VI. 결론

본 나눗셈기는 SRT 알고리즘을 탈피하여, 곱셈기를 공유하는 방식의 나눗셈기를 설계하였고, 기존의 프로세서 설계에 포함되어 있는 2개의 MAC 유닛을 최대한 활용, 병렬적인 곱셈을 통해 성능 향상을 꾀하였다. 나눗셈기는 MAC 유닛의 사이클 시간에 대한 오버헤드를 최소화하였으며, 나눗셈 연산을 위해 MAC 유닛에 추가되는 면적을 줄여 전체 나눗셈기의 면적의 18% 미만으로 설계하였기 때문에 소면적의 추가로 곱셈 누적 연산과 나눗셈 연산을 동시에 지원할 수 있다. 또한 IEEE-754 부동소수점 표준의 모든 rounding 방식을 지원하여 정확히 라운딩된 값들을 출력할 수 있고, 부동소수점 단정도 연산을 6 사이클만에 수행하므로 성능이 우수하다.

이와 같이 본 논문에서 최소한의 면적 증가로 고성능의 나눗셈 연산을 수행하는 나눗셈기를 제공함에 따라 면적 증가에 민감한 내장형 프로세서에 멀티미디어 연산에 필수적인 고속의 나눗셈 연산을 지원하는 하드웨어를 적은 비용으로 구현할 수 있다.

참고문헌

[1] ANSI/IEEE std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, 1985.

[2] 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.

[3] Israel Koren, *Computer Arithmetic Algorithms*, Prentice Hall, pp.153-161, 1993.

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