가변 시간 골드스미트 부동소수점 나눗셈기

김성기* · 송홍복** · 조경연*

A Variable Latency Goldschmidt's Floating Point Number Divider

Sung-Gi Kim* · Hong-Bok Song** · Gyeong-Yeon Cho*

요 약

부동소수점 나눗셈에서 많이 사용하는 골드스미트 나눗셈 알고리즘은 일정한 횟수의 곱셈을 반복한다. 본 논문에서는 오차가 정해진 값보다 작아질 때까지 곱셈을 반복하여 나눗셈을 수행하는 가변 시간 골드스미트 부동소수점 나눗셈 알고리즘을 제안한다.

부동소수점 나눗셈 '$\frac{N}{F}$'는 'T=$\frac{1}{F}$+e_t'를 분모와 분자에 곱하면 '$\frac{TN}{TF}=\frac{N_0}{F_0}$'가 된다. '$R_i$=(2-$e_r$-$F_i$), N_{i+1}=N_i*R_i, F_{i+1}=F_i*R_i, i∈{0,1,...n-1}'를 반복한다. 중간 곱셈 결과는 소수점이하 p 비트 미만을 절삭하며, 절삭 오차는 'e_r=2^{-p}' 보다 작다. p는 단정도실수에서 29, 배정도실수에서 59이다. 'F_i=1+e_i'이라고 하면 'F_{i+1}=1-e_{i+1}, e_{i+1}< e_i^2+3e_r'이 된다. '$|F_i-1| < 2^{\frac{-p+3}{2}}$'이면, '$e_{i+1}< 16e_r$'이 부동소수점으로 표현 가능한 최소값보다 작아지며, '$N_{i+1} \fallingdotseq \frac{N}{F}$'이다.

본 논문에서 제안한 알고리즘은 입력 값에 따라서 곱셈 횟수가 다르므로, 평균 곱셈 횟수를 계산하는 방식을 도출하고, 여러 크기의 근사 역수 테이블(T=$\frac{1}{F}$+e_t)에서 단정도실수 및 배정도실수의 나눗셈 계산에 필요한 평균 곱셈 횟수를 계산한다. 이들 평균 곱셈 횟수를 종래 알고리즘과 비교하여 본 논문에서 제안한 알고리즘의 우수성을 증명한다.

본 논문에서 제안한 알고리즘은 오차가 일정한 값보다 작아질 때까지만 반복 연산을 수행하므로 나눗셈기의 성능을 높일 수 있다. 또한 최적의 근사 역수 테이블을 구성할 수 있다.

본 논문의 연구 결과는 디지털 신호처리, 컴퓨터 그라픽스, 멀티미디어, 과학 기술 연산 등 부동소수점 계산기가 사용되는 분야에서 폭 넓게 사용될 수 있다.

Abstract

The GoldSchmidt iterative algorithm for a floating point divide calculates it by performing a fixed number of multiplications. In this paper, a variable latency GoldSchmidt's divide algorithm is proposed, that performs multiplications a variable number of times until the error becomes smaller than a given value.

*부경대학교 전자컴퓨터정보통신공학부
**동의대학교 전자정보통신공학부

접수일자 2004. 9. 15

To calculate a floating point divide '$\frac{N}{F}$', multiply 'T=$\frac{1}{F}+e_t$' to the denominator and the nominator, then it becomes '$\frac{TN}{TF}=\frac{N_0}{F_0}$'. And the algorithm repeats the following operations: '$R_i=(2-e_r-F_i)$, $N_{i+1}=N_i*R_i$, $F_{i+1}=F_i*R_i$, $i\in\{0,1,...n-1\}$'. The bits to the right of p fractional bits in intermediate multiplication results are truncated, and this truncation error is less than '$e_r=2^{-p}$'. The value of p is 29 for the single precision floating point, and 59 for the double precision floating point. Let '$F_i=1+e_i$', there is '$F_{i+1}=1-e_{i+1}$', where '$e_{i+1}< e_i^2+3e_r$'. If '$|F_i-1|<2^{\frac{-p+3}{2}}$' is true, '$e_{i+1}< 16e_r$' is less than the smallest number which is representable by floating point number. So, N_{i+1} is approximate to '$\frac{N}{F}$'.

Since the number of multiplications performed by the proposed algorithm is dependent on the input values, the average number of multiplications per an operation is derived from many reciprocal tables (T=$\frac{1}{F}+e_t$) with varying sizes. The superiority of this algorithm is proved by comparing this average number with the fixed number of multiplications of the conventional algorithm.

Since the proposed algorithm only performs the multiplications until the error gets smaller than a given value, it can be used to improve the performance of a divider. Also, it can be used to construct optimized approximate reciprocal tables.

The results of this paper can be applied to many areas that utilize floating point numbers, such as digital signal processing, computer graphics, multimedia, scientific computing, etc.

키워드

부동소수점, 나눗셈 계산기, Goldschmidt, Floating Point, Divider

I. 서 론

부동소수점 계산은 디지털 신호처리, 컴퓨터 그라픽스, 멀티미디어, 과학 기술 연산 등에서 폭 넓게 사용되고 있다. 최근에는 휴대용 통신 기기에서 화상 및 음성 처리를 수행하게 되면서SOC(System On Chip)에서도 하드웨어 부동소수점 계산기를 도입하고 있다. 부동소수점에서 나눗셈은 덧셈, 뺄셈 및 곱셈보다 출현 빈도가 낮지만, Oberman과 Flynn의 연구[1]는 나눗셈의 수행 시간이 덧셈이나 곱셈과 비슷하게 소요됨을 보이고 있다. 따라서 나눗셈기의 수행 속도를 높이는 연구가 요구되고 있다. 부동소수점 나눗셈은 뺄셈을 반복하는 SRT[2 4] 알고리즘과 곱셈을 반복하는 뉴톤-랍손(Newton -Raphson) 알고리즘 및 골드스미트(Goldschmidt) 알고리즘이 있다[5-8]. SRT 알고리즘은 나머지를 동시에 구하는 정밀 연산이 가능하지만 수행 시간이 오래 걸리는 문제점이 있다. 반면에 곱셈을 반복하는 방식은 빠른 곱셈기와 근사 역수 테이블을 사용하여 수행 시간을 빨리 할 수 있으나 근사계산이라는 단점을 가진다. 그러나 소수의 정밀 과학 기술 연산 분야를 제외하고 대부분 멀티미디어, 디지털 신호처리, 컴퓨터 그라픽스 등에서는 근사계산만으로 실용상 문제가 없다.

뉴톤-랍손 알고리즘은 제수의 역수를 구해서 피제수에 곱하여 나눗셈을 수행한다. 골드스미트 알고리즘은 제수와 피제수에 반복적인 곱하기로 나눗셈을 수행한다. 이들 알고리즘은 한 번 연산에 2회의 곱셈이 필요하다. 뉴톤-랍손은 2회의 연산이 종속적이지만 골드스미트는 서로 독립적이다. 따라서 골드스미트 알고리즘에서는 두개의 곱셈기를 사용해서 연산 시간을 줄일 수 있으므로 IBM RS/6000, AMD K7 프로세서 등에서 사용되고 있다[9].

본 논문에서는 골드스미트 부동소수점 나눗셈 알고리즘에서 분모가 1.0에 수렴하는 것을 오차 분

석을 통하여 예측하고, 오차가 정해진 값보다 작아지는 시점까지 반복 연산한다. 종래 방식에서는 최대 오차를 계산하고, 항상 일정 회수를 반복 수행하였다. 이러한 종래 방식에서는 구하고자 하는 결과 값에 도달했음에도 불구하고 가외의 연산을 계속하여 연산 속도를 저하시키는 단점이 있었다. 본 논문에서 제안하는 알고리즘은 이러한 종래의 단점을 개선하여 나눗셈기의 성능을 높일 수 있다. 또한 요구하는 나눗셈 성능에 따른 최적의 근사 역수 테이블을 구성할 수 있다.

본 논문에서 제안한 알고리즘은 입력하는 부동소수점 값에 따라서 곱셈 횟수가 다르므로, 평균 곱셈 횟수를 계산하는 방식을 유도하고, 여러 크기의 근사 역수 테이블에서 단정도실수 및 배정도실수의 나눗셈 계산에 필요한 평균 곱셈 횟수를 계산한다. 이들 평균 곱셈 횟수를 종래 알고리즘과 비교, 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 골드스미트 나눗셈 알고리즘을 분석해서 연산 유효자릿수 및 반복 연산을 종료할 오차 한계를 계산한다. 또한 오차 예측 알고리즘을 도출한다. 3장에서는 제안한 알고리즘을 구현하는 회로와 상태 기계를 구성한다. 4장에서는 근사 역수 테이블을 구성하고, 나눗셈에 소요되는 평균 곱셈 횟수를 계산한다. 그리고 그 결과를 종래 골드스미트 나눗셈 알고리즘과 비교 분석한다. 5장에서 결론을 맺는다.

II. 가변 시간 나눗셈 알고리즘

2.1. 골드스미트 나눗셈 알고리즘

$\frac{N_0}{F_0}$를 계산하는 골드스미트 나눗셈 알고리즘은 식 (1)과 같다.

$$\text{For } i \in \{0, 1, 2, \ldots n-1\} \qquad (1)$$

$$R_i = 2 - F_i$$
$$N_{i+1} = N_i * R_i$$
$$F_{i+1} = F_i * R_i$$

반복 연산을 수행하면 F_n은 1.0에 수렴하므로 다음 식이 성립한다.

$$\frac{N_n}{F_n} = \frac{N_0 * R_0 * R_1 \ldots * R_n}{F_0 * R_0 * R_1 \ldots * R_n} = N_n$$

따라서 N_n이 나눗셈 결과이다.

IEEE-754[10]로 규정되는 부동소수점은 $1.f_2 * 2^{n+base}$ 이다. 가수부 1.f는 단정도실수에서 24 비트, 배정도실수에서는 53 비트이다. 나눗셈의 지수부 연산은 피제수의 지수에서 제수의 지수를 빼는 것이므로 본 논문에서 지수부 연산은 생략한다.

부동소수점 수 F의 가수부 1.f는 식 (2)와 같이 두 부분으로 나눌 수 있다.

$$F = 1.g + h \qquad (2)$$

식 (2)에서 g와 h의 길이를 각각 n_g 및 n_h 비트로 정의한다. h는 '$0 \le h < 2^{-n_g}$'이며, h의 최대값은 '$h_{max} = 2^{-n_g} - 2^{-n_g-n_h}$'이다.

식 (1)의 수렴 속도를 빠르게 하기 위해서 $\frac{1}{1.g}$를 근사계산한 테이블 T(g)를 미리 작성해 놓는다. 근사 테이블은 ROM에 저장하거나 또는 별도의 회로를 사용해서 계산하기도 한다. T(g)는 $\frac{1}{1.g}$의 근사계산이므로 '$T(g) = \frac{1}{1.g} + e_t$' 이다. e_t는 근사에 따른 오차이다. 나눗셈 식의 분모와 분자에 T(g)를 곱해주면 식 (3)과 같이 된다.

$$\frac{N}{F} = \frac{T(g) * N}{T(g) * F} = \frac{N_0}{F_0} \qquad (3)$$

식 (3)에서 F_0는 1.0에 근사하므로 식-1의 수렴 속도를 빠르게 할 수 있다.

식 (1)에서 '$N_i * R_i$'와 '$F_i * R_i$' 곱셈 결과는 소수점 이하 p 비트 미만을 절삭한다. p를 연산 유효자릿

수라고 가칭한다. 또한 식 (1)에서 '2-F_i'는 하드웨어 구현 시에 캐리 전달 지연 시간이 필요하다. 이러한 문제점을 해결하기 위해서 본 논문에서는 근사계산인 '2-e_r-F_i, e_r=2^{-p}'을 계산한다. 본 논문에서 사용하는 골드스미트 나눗셈 알고리즘을 식 (4)에 보인다.

$$\frac{N}{F} = \frac{T(g)*N}{T(g)*F} = \frac{N_0}{F_0} \quad (4)$$

For i ∈{0, 1, 2,....n-1}

$R_i = 2 - e_r - F_i$

$N_{i+1} = N_i * R_i$

$F_{i+1} = F_i * R_i$

2.2. 오차 분석

골드스미트 나눗셈 알고리즘은 곱셈을 반복하므로 오차가 누적된다. 그러므로 구하고자 하는 부동소수점 정밀도보다 긴 자리수의 연산이 요구된다.

식 (4)에서 N_i를 반복 계산하면 절삭 오차가 누적된다. N_3까지의 최대 누적 오차를 구하면 식 (5)가 된다. [X]는 X의 오차가 없는 값을 나타내며, 중간 곱셈의 절삭 오차는 '$e_r = 2^{-p}$'보다 작다.

$$N_0 = T(g)*N-e_r = [N_0]-e_r \quad (5)$$

$R_0 = 2-e_r-F_0 = [R_0]-e_r$

$N_1 = N_0*R_0 = ([N_0]-e_r)*([R_0]-e_r)-e_r$

$\fallingdotseq [N_1]-4e_r$

$\because R_0 \fallingdotseq 1,\ N_{i,\max} = 2$

$N_2 = N_1*R_1 = ([N_1]-4e_r)*([R_1]-e_r)-e_r \fallingdotseq [N_2]-7e_r$

$N_3 = N_2*R_2 = ([N_2]-4e_r)*([R_2]-e_r)-e_r \fallingdotseq [N_3]-10e_r$

'F_i= 1+e_i'이라고 하면 F_{i+1}은 식 (6)과 같이 된다.

$$\begin{aligned} F_{i+1} &= (1+e_i)*(2-e_r-1-e_i)-e_r \\ &= 1-e_i^2-(2+e_i)e_r \\ &= 1-e_{i+1} \end{aligned} \quad (6)$$

식 (6)에서 'e_i<<1'이므로 식 (7)이 성립한다.

$$e_{i+1} < e_i^2 + 3e_r \quad (7)$$

'F_i=1-e_i'이라고 하면 식 (7)은 '$e_{i+1} < e_i^2 + 2e_r$'이 된다.

식 (7)로부터 식 (4)의 반복 연산 중에 절삭으로 발생하는 F_i의 최대 오차는 $3e_r$보다 작다. 한편 식 (5)에서 N_i의 최대 누적 오차가 $10e_r$이므로 e_{i+1}의 최소값을 $16e_r$로 정하면, e_i^2의 최대값은 $13e_r$이 된다. 따라서 식 (8)을 만족하면 식 (4) 알고리즘의 반복을 종료한다.

$$e_i^2 < 13e_r \quad (8)$$

하드웨어 구현이 용이하도록 식 (8)의 조건식을 '$e_i^2 < 8e_r$'로 한다. 따라서 식 (9)가 성립하면 반복 연산을 종료하고, '$N_{i+1} \fallingdotseq \frac{N}{F}$'이다.

$$e_i = |F_i-1| < \sqrt{8e_r} = 2^{\frac{-p+3}{2}} \quad (9)$$

2.3. 연산 유효자릿수

'F_{i+1}=1-e_{i+1}'이므로 'e_{i+1}< $16e_r$'은 부동소수점에서 표현할 수 있는 최소값보다 작아야 한다. 한편 IEEE-754 단정도실수에서 가수부의 유효자릿수는 24 비트이다. 유효자릿수에 라운드 한 비트를 더하면 25 비트이므로 '$16e_r$'이 2^{-25}보다 작아야 한다. 그러므로 '$16e_r = 16*2^{-p} < 2^{-25}$'이 되어야 한다. 따라서 IEEE-754 단정도실수 나눗셈 계산에 필요한 연산 유효자릿수는 'p = 29'이다.

IEEE-754 배정도실수에서 가수부의 유효자릿수는 53 비트이다. 유효자릿수에 라운드 한 비트를 더하면 54 비트이므로 $16e_r$이 2^{-54}보다 작아야 한다.

그러므로 '$16e_r = 16*2^{-p} < 2^{-54}$'가 되어서 'p = 58'이다. 그런데 식 (9) 조건식의 하드웨어 구현을 용이하게 하기 위해서는 p가 홀수가 되어야 한다. 따라서 IEEE-754 배정도실수 나눗셈 계산에 필요

한 연산 유효자릿수는 'p = 59'이다.

2.4. 가변 시간 골드스미트 나눗셈 알고리즘

식 (9) '$e_i = |F_i - 1| < 2^{\frac{-p+3}{2}}$'를 판단하는 회로는 F_i의 소수점 이하 '$\frac{p-3}{2}$'개의 비트가 모두 '0' 또는 모두 '1'인가를 판단하는 회로이다. 본 논문에서 제안하는 가변 시간 골드스미트 나눗셈 알고리즘을 정리하면 표 1과 같다.

표 1. 가변 시간 골드스미트 나눗셈 알고리즘
Table 1. Variable latency Goldschmidt's divide

To compute $\frac{N}{F}$, where F is (1.g + h). T(g) is pre-calculated approximate $\frac{1}{1.g}$ p = 29 if IEEE-754 single precision p = 59 if IEEE-754 double precision
1) N = N * T(g) ; 2) F = F * T(g) ; R = 1's complement of F ; 3) N = N * R ; If $\|F-1\| < 2^{\frac{-p+3}{2}}$, then exit ; 4) F = F * R ; /* R_i */ R = 1's complement of F ; /* R_{i+1} */ Goto 3 ;

algorithm

III. 가변 시간 나눗셈기

그림 1에 하나의 곱셈기를 사용한 가변 시간 나눗셈기의 블록도를 보인다. 또한 표 2에 상태 기계의 흐름을 보인다.

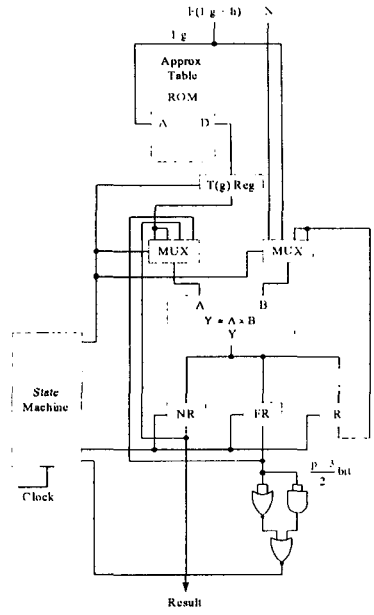


그림 1. 하나의 곱셈기를 사용한 가변 시간 골드스미트 나눗셈기
Fig 1. Variable latencyGoldschmidt's divider with single multiplier

표 2. 하나의 곱셈기를 사용한 가변 시간 골드스미트 나눗셈기 상태 흐름도
Table 2. State flow of variable latency Goldschmidt's divider with single multiplier

/* Calculate $\frac{N}{F}$ */ (State-1) Get approximate reciprocal table T(g) ; (State-2) Multiplier in port A = T(g) ; Multiplier in port B = N ; Multiplier out port Y = NR ; (state-3) Multiplier in port A = T(g) ; Multiplier in port B = F ; Multiplier out port Y = FR ;

```
   R = 1's complement of FR ;
(state-4)
   Multiplier in port A = NR ;
   Multiplier in port B = R ;
   Multiplier out port Y = NR ;
   If msb (p-3)/2 bit of FR is all '0' or
all '1',
        then Result is NR ;
(state-5)
   Multiplier in port A = FR ;
   Multiplier in port B = R ;   /* R_i */
   Multiplier out port Y = FR ;
   R = 1's complement of FR ;   /* R
i+1  */
   Goto state-4 ;
```

그림 1 블록도와 표 2의 상태 흐름도는 제안한 가변 시간 골드스미스 나눗셈 알고리즘을 하드웨어로 구현한 것이다. 상태-1에서 $\frac{1}{F}$의 근사 값을 테이블로부터 읽어서 T(g) 레지스타에 저장한다. 상태-2와 상태-3에서는 식 (4)와 같이 T(g) 레지스타를 나눗셈의 분모 F와 분자 N에 각각 곱하여 F_0와 N_0를 계산해서 FR 레지스타와 NR 레지스타에 각각 저장한다. 또한 상태-3에서는 'R_i=2-e_r-F_i =1.11$\cdots$$1_2$-$F_i$'를 계산해서 R 레지스타에 저장한다. 상태-4에서는 식 (4)의 'N_{i+1}=N_i*R_i'를 계산해서 NR 레지스타에 저장한다. 또한 '$|F_i-1| < 2^{\frac{-p+3}{2}}$'를 판단한다. 이 판단은 F_i의 소수점 이하 $\frac{p-3}{2}$ 비트가 모두 '0' 또는 모두 '1'인지를 판별하는 것으로 $\frac{p-3}{2}$ 개의 입력을 가지는 NOR 게이트와 $\frac{p-3}{2}$ 개의 입력을 가지는 AND 게이트로 구현할 수 있다. 판단이 맞으면 나눗셈을 종료한다. 맞지 않으면 상태-5에서 'F_{i+1} = F_i*R_i'를 계산해서 FR 레지스타에 저장하고, 'R_{i+1} =2-e_r-F_{i+1}'를 계산해서 R 레지스타에 저장하고 상태-4로 가서 반복 계산한다.

종래 골드스미트 나눗셈에 '$\frac{p-3}{2}$' 개의 입력을 가지는 NOR 게이트와 '$\frac{p-3}{2}$' 개의 입력을 가지는 AND 게이트를 추가하고, 상태 흐름도를 일부 변경하는 것으로 가변 시간 골드스미트 나눗셈기를 구현할 수 있다.

2개의 곱셈기를 병렬로 사용하는 가변 시간 골드스미스 나눗셈기의 상태 흐름도는 표 2에서 상태-2와 상태-3을 하나의 상태로 묶고, 상태-4와 상태-5를 하나의 상태로 묶어서 구현할 수 있다. 표 3에 2개의 곱셈기를 병렬로 사용하는 가변 시간 골드스미스 나눗셈기의 상태 기계 흐름을 보인다.

설계한 가변 시간 골드스미트 나눗셈기는 Verilog HDL로 기술하고 시뮬레이션해서 동작을 확인하였다.

표 3. 두개의 곱셈기를 사용한 가변 시간 골드스미트 나눗셈기 상태 흐름도

Table 3. State flow of variable latency Goldschmidt's divider with dual multiplier

```
/*  Calculate N/F  */
(State-1)
   Get approximate reciprocal table T(g) ;

(State-2)
   Multiplier 1 in port A = T(g) ;
   Multiplier 1 in port B = N ;
   Multiplier 1 out port Y = NR ;

   Multiplier 2 in port A = T(g) ;
   Multiplier 2 in port B = F ;
   Multiplier 2 out port Y = FR ;
   R = 1's complement of FR ;

(state-3)
   Multiplier 1 in port A = NR ;  /* N_i
*/
   Multiplier 1 in port B = R ;   /* R_i
*/
   Multiplier 1 out port Y = NR ; /*  N
i+1 */
```

```
If msb (p-3)/2 bit of FR is all '0' or all '1',
    then Result is NR ;        /* Check F_i */

Multiplier 2 in port A = FR ;  /* F_i */
Multiplier 2 in port B = R ;   /* R_i */
Multiplier 2 out port Y = FR ; /* F_{i+1} */
R = 1's complement of FR ;     /* R_{i+1} */
Goto state-3 ;b
```

IV. 연구 결과 및 분석

식 (4)로부터 F_0의 오차 e_0는 식 (10)이 된다.

$$e_0 = T(g)*(1.g+h) - 1 \quad (10)$$

식 (10)으로부터 e_0는 h=0에서 'T(g)*1.g - 1'이 되며, $0 \le h \le h_{max}$에서 단조 증가함수이다. 식 (10)으로부터 h는 식 (11)이 된다.

$$h = \frac{1+e_0}{T(g)} - 1.g \quad (11)$$

h에 따른 e_0의 변화를 그림 2에 보인다.

식 (9)와 표 2의 상태 흐름도로부터 '$e_0 < A_2 = 2^{\frac{-p+3}{2}}$'이면 3회의 곱셈으로 나눗셈이 완료된다. 그림 2에서 h_2는 A_2에서의 h 값이다. 그러므로 '$0 \le h \le h_2$' 구간에서는 3회의 곱셈으로 나눗셈이 계산된다.

식 (6)으로부터 e_1은 식 (12)가 된다.

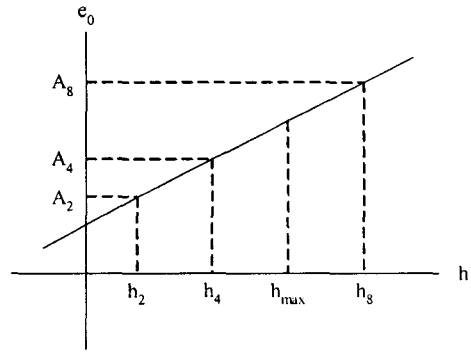


그림 2. h와 e_0의 그라프

Fig 2. e_0 versus h

$$e_1 = e_0^2 + e_r e_0 + 2e_r \quad (12)$$

식 (12)와 표 2의 상태 흐름도로부터 '$e_1 < A_2$'이면 5회의 곱셈으로 나눗셈을 계산할 수 있다. 식 -12에서 '$e_1 < A_2$'이 되는 e_0는 식 (13)이 된다.

$$e_0 < A_4 = \frac{-e_r + \sqrt{e_r^2 - 8e_r + 4A_2}}{2} \quad (13)$$

그림 2에서 h_4는 A_4에서의 h 값이다. 그러므로 '$h_2 < h \le h_4$' 구간에서는 5회의 곱셈으로 나눗셈을 계산한다. 식 (12)와 식 (13)과 동일한 방법으로 '$e_2 < A_2$'가 되는 e_0는 식 (14)가 된다

$$e_0 < A_8 = \frac{-e_r + \sqrt{e_r^2 - 8e_r + 4A_4}}{2} \quad (14)$$

그림 2에서 h_8은 A_8에서의 h 값이다. '$h_4 < h \le h_{max}$' 구간에서는 7회의 곱셈으로 나눗셈이 계산된다. '$h > h_8$'인 구간에서는 9회의 곱셈으로 나눗셈을 계산한다. e_0가 음수인 경우에도 유사한 방법으로 h에 따른 곱셈 회수를 산출할 수 있다.

DasSarma의 연구 결과 최적의 근사 역수는 식 (15)로 주어진다[11].

$$T(g) = \frac{1}{1.g} \fallingdotseq RN\left(\frac{1}{1.g + 2^{-n_g-1}}\right) \quad (15)'$$

RN is round to nearest

T(g)의 소수점 이하 길이를 t 비트라고 하면 'T(g)=$(b_0.b_1b_2\cdots b_t)_2$, $0.5<T(G)\leq 1.0$'이다. 'b_0b_1=10'인 경우는 '$g=g_1g_2\cdots g_{n_q}$=00…0'일 때이다. 이외의 경우는 항상 'b_0b_1=01'이다. 그러므로 근사 역수 테이블에 '$b_2\cdots b_t$'만을 저장하면 된다. 따라서 근사 역수 테이블의 크기는 '2^{n_q} X (t-1)' 비트가 되어서, 테이블의 길이는 2^{n_q}이며, 폭은 't-1' 비트이다.

본 논문에서 제안한 알고리즘에 의한 IEEE 단정도실수 나눗셈 계산에 필요한 곱셈 횟수를 표 4에, 배정도실수 나눗셈 계산에 필요한 곱셈 회수를 표 5에 각각 보인다.

표 4. IEEE 단정도실수 나눗셈에 필요한 곱셈 횟수
(1) -- 한 개의 곱셈기를 사용한 경우
(2) -- 두 개의 곱셈기를 사용한 경우

Table 4. Number of multiply of IEEE single precision divide
(1) -- in case of one multiplier
(2) -- in case of two multiplier

Table size	Average No. of multiply 1	Average No. of multiply 2	No. of Multiply 1			
			3 mul	5 mul	7 mul	9 mul
16×3	6.52	3.76	0.27%	23.5%	76.2%	0.0%
32×4	6.05	3.52	0.54%	46.3%	53.2%	0.0%
64×5	5.29	3.15	1.08%	83.3%	15.6%	0.0%
128×6	4.96	2.98	2.16%	97.8%	0.0%	0.0%
256×7	4.91	2.96	4.32%	95.7%	0.0%	0.0%
64×6	4.96	2.98	1.88%	98.1%	0.05%	0.0%
16×4	6.13	3.57	0.48%	42.5%	57.1%	0.0%
16×5	6.01	3.51	0.58%	48.2%	51.2%	0.0%
16×6	5.93	3.46	0.59%	52.5%	47.0%	0.0%
16×7	5.93	3.46	0.59%	52.4%	47.0%	0.0%

종래 골드스미트 나눗셈 알고리즘에서는 최대 오차를 고려해서 반복 횟수를 정했다. 즉, 종래 알고리즘에 의한 단정도실수 나눗셈은 '64x5' 테이블을 사용하면 7회의 곱셈을 수행하였고, '128x6' 테이블을 사용하면 5회의 곱셈을 수행하였음을 표 4로부터 알 수 있다. 또한 '64x6' 테이블을 사용해도 7회의 곱셈을 수행하였었다.

그러나 본 논문에서 제안한 가변 시간 골드스미트 나눗셈 알고리즘에서는 '64x5' 테이블에서 평균 5.29회의 곱셈, '128x6' 테이블과 '64x6' 테이블에서 모두 평균 4.96 회의 곱셈으로 나눗셈을 할 수 있다. 즉, 본 논문에서 제안한 알고리즘에서 평균 5회의 곱셈으로 나눗셈을 수행하려면 '64x6' 테이블을 사용하면 되므로, 종래 알고리즘에서 사용하던 '128x6' 테이블과 비교하여 테이블 크기를 반으로 줄일 수 있다.

표 5. IEEE 배정도실수 나눗셈에 필요한 곱셈 횟수
(1) -- 한 개의 곱셈기를 사용한 경우
(2) -- 두 개의 곱셈기를 사용한 경우

Table 5. Number of multiply of IEEE double precision divide
(1) -- in case of one multiplier
(2) -- in case of two multiplier

Table size	Average No. of multiply 1	Average No. of multiply 2	No. of Multiply 1			
			3 mul	5 mul	7 mul	9 mul
128×6	7.00	4.00	0.0%	1.08%	97.9%	1.03%
256×7	6.96	3.98	0.0%	2.16%	97.8%	0.0%
512×8	6.91	3.96	0.0%	4.32%	95.7%	0.0%
1024×11	6.83	3.91	0.0%	8.63%	91.4%	0.0%
64×5	7.71	4.35	0.0%	0.54%	63.5%	36.0%
64×6	7.19	4.10	0.0%	0.94%	88.6%	10.5%
64×7	6.99	4.00	0.0%	1.17%	98.2%	0.64%
64×8	6.98	3.99	0.0%	1.17%	98.6%	0.18%

이러한 결과는 배정도실수 연산에서도 나타난다. 배정도실수 나눗셈은 표 5로부터 종래 알고리즘에서는 '128X6' 테이블을 사용하면 9회의 곱셈, '256X7' 테이블을 사용하면 7회의 곱셈을 수행하였다. 그러나 본 논문에서 제안한 알고리즘에서는 '128X6' 테이블을 사용하면 평균 7.00회의 곱셈, '256X7' 테이블을 사용하면 6.96회의 곱셈을 수행한다. 또한 '64X7' 테이블을 사용해도 평균 6.99회 곱셈으로 나눗셈을 구할 수 있다. 따라서 평균 7회 곱셈으로 배정도실수 나눗셈을 하려면 종래 알고리즘에서는 '256X7' 테이블을 사용했지만, 본 논문

에서 제안하는 알고리즘을 사용하면 '128X6' 테이블이나 '64X7' 테이블을 사용하면 충분하다. '64X7' 테이블의 크기는 '256X7' 테이블의 사분의 일 크기이다. 따라서 본 논문에서 제안한 알고리즘은 테이블 크기를 사분 일로 줄이면서 동일한 성능을 보인다.

역수 근사 테이블의 길이와 폭의 관계를 표 4 및 표 5로부터 알 수 있다. 표 4의 단정도실수 나눗셈에서 '16X3', '16x4', '16x5', '16x6' 및 '16x7' 테이블은 길이는 16으로 같으며, 폭은 3 비트에서 7 비트까지로 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 7회의 곱셈으로 나눗셈을 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 6.52회, 6.13회, 6.01회, 5.93회와 5.93회의 곱셈으로 나눗셈을 계산한다. 이들 결과로부터 테이블의 폭을 1-2 비트 증가시키면 성능이 크게 개선되지만 3 비트 이상을 증가시켜도 성능이 개선되지 않음을 알 수 있다.

테이블의 길이와 폭의 관계는 배정도실수에서도 동일하게 나타난다. 표 5에서 '64x5', '64x6', '64x7'과 '64x8' 테이블은 길이가 64로 같지만 폭은 5 비트에서 8 비트까지 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 9회의 곱셈으로 나눗셈을 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 7.71회, 7.19회, 6.99회와 6.98회의 곱셈으로 나눗셈을 계산한다.

이들 결과로부터 본 논문의 알고리즘에서는 역수 근사 테이블의 폭 t는 길이 L의 '$\log_2 L$ - 1'보다 1-2 비트 큰 것이 가격대비 성능 면에서 좋다.

종래의 골드스미트 나눗셈 알고리즘에서는 테이블 크기가 제한적이었다. 단정도실수에서 7회 곱셈으로 나눗셈을 하기 위해서는 '16X3' 테이블을, 5회 곱셈을 위해서는 '128X6' 테이블을 사용하였다. 중간 크기의 테이블은 의미가 없었다. 즉, '32X4' 테이블이나 '64X5' 테이블을 사용하면 나눗셈에 7회의 곱셈이 필요하였다.

그러나 본 논문에서 제안한 알고리즘은 나눗셈 성능에 따라 다양한 테이블을 선택할 수 있는 장점을 가진다. 이것은 SOC처럼 제한된 크기의 실리콘에 CPU, 메모리, 입출력장치 등을 모두 집적하는 응용 분야에서 최적의 성능을 실현할 수 있는 방법을 제공해 준다.

V. 결 론

부동소수점 나눗셈은 뺄셈을 반복하는 SRT 알고리즘과 곱셈을 반복하는 뉴톤-랍손 (Newton-Raphson) 알고리즘 및 골드스미트(Goldschmidt) 알고리즘이 있다. 골드스미트 알고리즘은 제수와 피제수에 반복적으로 곱셈하여 나눗셈을 수행한다.

본 논문에서는 골드스미트 부동소수점 나눗셈 알고리즘에서 제수가 1.0에 수렴하는 것을 오차 분석을 통해서 예측하고, 오차가 정해진 값보다 작아지는 시점까지만 반복 연산을 수행하였다. 이를 위해서 골드스미트 나눗셈 알고리즘의 오차를 분석했고, 연산 자릿수 및 반복 연산을 종료할 오차 한계를 계산하였다. 제안한 알고리즘을 구현하는 회로와 상태 기계를 설계하였다. 설계한 나눗셈기는 Verlig HDL 언어로 기술하고 시뮬레이션을 통하여 동작을 확인하였다. 또한 근사 역수 테이블을 구성하고, 나눗셈에 소요되는 평균 곱셈 횟수를 계산해서 그 결과를 종래 골드스미트 나눗셈 알고리즘과 비교 분석하였다.

배정도실수 나눗셈은 종래 알고리즘에서는 '128X6' 근사 역수 테이블을 사용하면 9회의 곱셈, '256X7' 테이블을 사용하면 7회의 곱셈을 수행하였다. 그러나 본 논문에서 제안한 가변 시간 골드스미트 나눗셈 알고리즘에서는 '64X7' 테이블을 사용하면 6.99회의 곱셈을 수행하였다. 따라서 평균 7회 곱셈으로 나눗셈을 하려면 종래 알고리즘에서는 '256X7' 테이블을 사용했지만, 본 논문에서 제안한 알고리즘을 사용하면 '64X7' 테이블을 사용해도 충분하였다. 따라서 종래 알고리즘에 비하여 역수 근사 테이블의 크기를 사분의 일로 줄일 수 있었다.

본 논문에서는 근사 역수 테이블의 길이와 폭의 관계를 밝혔다. 배정도실수에서 '64x5', '64x6', '64x7'과 '64x8' 테이블은 길이가 64로 같지만 폭은 5 비트에서 8 비트까지 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 9회의 곱셈으로 나눗셈을 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 7.71회, 7.19회, 6.99회와 6.98회의 곱셈으로 나눗셈을 계산하였다. 이들 결과로부터 테이블의 폭을 1-2 비트 증가시키면 성능이 크게 개선되지만 3 비트 이상을 증가시켜도 성능이 개선되지 않음을 알 수 있었다.

본 논문에서 제안한 가변 시간 골드스미트 나눗셈 알고리즘은 평균 곱셈 횟수가 중요한 디지털 신호처리, 컴퓨터 그라픽스, 멀티미디어, 과학 기술 연산 등에서 폭 넓게 사용될 수 있다. 또한 나눗셈 성능에 따른 최적의 근사 역수 테이블을 구성할 수 있으므로 제한적인 하드웨어 사양을 가지는 SOC(System On Chip)에 유용하게 적용될 수 있다.

참고문헌

[1]S. F. Oberman and M. J. Flynn, "Design Issues in Division and Other Floating Point Operations," IEEE Transactions on Computer, Vol. C-46, pp. 154-161, 199

[2] C. V. Freiman, "Statistical Analysis of Certain Binary Division Algorithm," IRE Proc., Vol. 49, pp. 91-103, 1961

[3] S. F. McQuillan, J. V. McCanny, and R. Hamill, "New Algorithms and VLSI Architectures for SRT Division and Square Root," Proc. 11th IEEE Symp. Computer Arithmetic, IEEE, pp. 80-86, 1993

[4] D. L. Harris, S. F. Oberman, and M. A. Horowitz, "SRT Division Architectures and Implementations," Proc. 13th IEEE Symp. Computer Arithmetic, Jul. 1997

[5] M. Flynn, "On Division by Functional Iteration," IEEE Transactions on Computers Vol. C-19, no. 8, pp. 702-706, Aug. 1970

[6] R. Goldschmidt, Application of division by convergence, master's thesis, MIT, Jun. 1964.

[7] M. D. Ercegovac, et al, "Improving Goldschmidt Division, Square Root, and Square Root Reciprocal," IEEE Transactions on Computer, Vol. 49, No. 7, pp.759-763, Jul. 2000

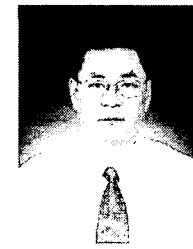
[8] D. L. Fowler and J. E. Smith, "An Accurate, High Speed Implementation of Division by Reciprocal Approximation," Proc. 9th IEEE symp. Computer Arithmetic, IEEE, pp. 60-67, Sep. 1989

[9] S. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microproce ssors, " Proc. 14th IEEE Symp. Computer Arithmetic, pp. 106-115, Apr. 1999

[10] IEEE, IEEE Standard for Binary Floating-Po int Arithmetic, ANSI/IEEE Standard, Std. 754-1985

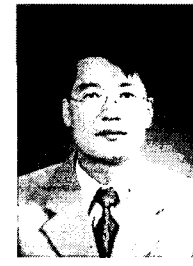
[11] D. DasSarma and D. Matula, "Measuring and Accuracy of ROM Reciprocal Tables," IEEE Transactions on Computer, Vol.43, No. 8, pp. 932-930, Aug. 1994

저자 소개



김성기(Sung-Gi Kim)

1984 울산대학교 응용물리학과 졸업
1994 부경대학교 산업대학원 전자계산학과 졸업
2000-현재 제이미(주)대표이사
2004-현재 부경대학교 대학원 컴퓨터공학과 박사과정
※관심분야 : 전산기 구조, 반도체 회로 설계



송홍복(Hong-Bok Song)

1983년 광운대학교 전자통신공학과 졸업
1985년 인하대학교 대학원 전자공학과 졸업(공학석사)
1985 - 1990년 : 동의공업대 전자통신과 조교수
1989 - 1990년 : 일본 구주공대 정보공학부 객원연구원
1990년 동아대학교 대학원 전자공학과 졸업(공학박사)
1994-1995년 일본 미야쟈키 대학교 전기.전자공학부(POST-DOC)
1991년-현재 동의대학교 전자. 정보통신공학부 교수
※관심분야 : 다치논리 이론 및 시스템 설계, VLSI설계, 마이크로프로세서 응용

조경연(Gyeong-Yeon Cho)

1990 인하대학교 공과대학 전자공학과 공학박사
1983-1991 삼보컴퓨터 기술연구소 책임연구원
1991-2000 삼보컴퓨터 기술연구
1998-2003 에이디칩스 기술고문
1991-현재 부경대학교 공과대학 전자컴퓨터정보통신공학부 교수
※관심분야 : 전산기구조, 반도체회로설계, 암호알고리즘소 기술고문