가변 시간 뉴톤-랍손 부동소수점 역수 계산기

김 성 기† · 조 경 연††

요 약

부동소수점 나눗셈에서 많이 사용하는 뉴톤-랍손 부동소수점 역수 알고리즘은 일정한 횟수의 곱셈을 반복하여 역수를 계산한다. 본 논문에서는 오차가 정해진 값보다 작아질 때까지 곱셈을 반복해서 역수를 계산하는 가변 시간 뉴톤-랍손 부동소수점 역수 알고리즘을 제안한다.

'F'의 역수 계산은 초기값 '$X_0=\frac{1}{F}\pm e_0$'에 대하여, '$X_{i+1}= X_i*(2-e_r-F*X_i)$, $i\in\{0, 1, 2,...n-1\}$'을 반복한다. 중간 곱셈 결과는 소수점 이하 p 비트 미만을 절삭하며, 절삭 오차는 '$e_r=2^{-p}$' 보다 작다. p는 단정도실수에서 27, 배정도실수에서 57이다. '$X_i=\frac{1}{F}+e_i$'라 하면 '$X_{i+1}=\frac{1}{F}-e_{i+1}$, $e_{i+1}<Fe_i^2+3e_r$'이 된다. '$|(2-e_r-F*X_i)-1|<2^{\frac{-p+2}{2}}$'이면, '$e_{i+1}< 4e_r$'이 부동소수점으로 표현 가능한 최소값보다 작아지며, '$X_{i+1}\fallingdotseq\frac{1}{F}$'이다.

본 논문에서 제안한 알고리즘은 입력 값에 따라서 곱셈 횟수가 다르므로, 평균 곱셈 횟수를 계산하는 방식을 유도하고, 여러 크기의 근사 역수 테이블($X_0=\frac{1}{F}\pm e_0$)에서 단정도실수 및 배정도실수의 역수 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 역수 계산기의 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.

A Variable Latency Newton-Raphson's Floating Point Number Reciprocal Computation

Sung-Gi Kim† · Gyeong-Yeon Cho††

ABSTRACT

The Newton-Raphson iterative algorithm for finding a floating point reciprocal which is widely used for a floating point division, calculates the reciprocal by performing a fixed number of multiplications. In this paper, a variable latency Newton-Raphson's reciprocal algorithm is proposed, that performs multiplications a variable number of times until the error becomes smaller than a given value.

To find the reciprocal of a floating point number F, the algorithm repeats the following operations: '$X_{i+1}= X_i*(2-e_r-F*X_i)$, $i\in\{0, 1, 2,...n-1\}$' with the initial value is '$X_0=\frac{1}{F}\pm e_0$'. 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 27 for the single precision floating point, and 57 for the double precision floating point. Let '$X_i=\frac{1}{F}+e_i$', there is '$X_{i+1}=\frac{1}{F}-e_{i+1}$', where '$e_{i+1}< Fe_i^2+3e_r$'. If '$|(2-e_r-F*X_i)-1|<2^{\frac{-p+2}{2}}$' is true, '$e_{i+1}< 4e_r$' is less than the smallest number which is representable by floating point number. So, X_{i+1} is approximate to '$\frac{1}{F}$'.

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 ($X_0=\frac{1}{F}\pm e_0$) 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 reciprocal unit. 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(Floating point), 뉴톤-랍손(Newton-Raphson), 역수(Reciprocal), 가변시간(Variable latency)

† 정 회 원 : 부경대학교 대학원 컴퓨터공학과 박사과정
†† 정 회 원 : 부경대학교 전자컴퓨터정보통신공학부 교수
논문접수 : 2004년 10월 28일, 심사완료 : 2005년 3월 16일

1. 서 론

부동소수점 계산은 과학 및 공학 기술 분야에서 많이 사용된다. 최근에는 멀티미디어가 보급되면서 음성 처리 및 3차원 그라픽 분야에도 폭넓게 사용되고 있다. 부동소수점 계산에서 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 이용한 알고리즘으로 뉴톤-랍손(Newton-Raphson) 역수 알고리즘 및 골드스미트(Goldschmidt) 나눗셈 알고리즘이 있다. 골드스미트 나눗셈 알고리즘은 분모와 분자에 반복적으로 동일한 값을 곱해서 분모를 '1.0'에 수렴시키면 분자가 나눗셈 결과가 된다. 뉴톤-랍손 역수 알고리즘은 제수의 역수를 구해서 피제수에 곱하는 방식이다. 역수 계산은 제수의 근사 값을 초기 값으로 하고, 반복해서 오차를 줄여나간다. 반복할 때마다 상대 오차는 자승으로 줄어들며, 한 회의 반복 연산에 2회의 곱셈이 필요하다.

이들 곱셈을 이용한 알고리즘은 초기 근사 값을 이용하여 수렴 속도를 향상시키고 있다. 그러므로 초기 근사 값을 결정하는 알고리즘과 수렴 속도를 높이기 위한 방식에 대하여 연구가 진행되었다[5-9]. 종래 연구는 초기 근사 값이 가지는 최대 오차를 계산하고, 오차를 부동소수점에서 표현 가능한 최소 값보다 작게 될 때까지 반복 연산을 수행하였다. 이러한 종래 연구에서는 최대 오차만을 고려했기 때문에, 실제 구하고자 하는 결과 값에 도달했음에도 불구하고 가외의 연산을 수행하여 연산 속도를 저하시키는 단점이 있었다.

본 논문에서는 뉴톤-랍손 부동소수점 역수 알고리즘의 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 수행하는 알고리즘을 제안한다. 뉴톤-랍손 알고리즘은 결과 값의 근사치 '$X_n \fallingdotseq \frac{1}{F}$'로부터 다음 근사치 X_{n+1}을 계산한다. '$|X_{n+1} - X_n| < e$'이 되면 X_{n+1}이 결과 값이다. 그러나 이런 방식에서는 불필요한 'X_{n+1}-X_n' 뺄셈을 해야 하므로 효율적이지 못하다. 본 논문에서는 '$X_{n+1}=X_n(2-FX_n)$'이므로 '$2-FX_n$'으로부터 '$|X_{n+1}-\frac{1}{F}|$'을 예측하는 알고리즘을 도출하고, 예측 오차가 유효자릿수로 표현할 수 있는 최소값보다 작게 되면 반복을 종료한다.

본 논문에서 제안한 가변 시간 뉴톤-랍손 역수 알고리즘에 의한 역수 계산기를 Verilog HDL로 설계하고, 시뮬레이션해서 정상적으로 동작하는 것을 확인하였다.

본 논문에서 제안한 알고리즘은 부동소수점 값에 따라서 반복 곱셈 횟수가 다르므로, 평균 곱셈 횟수를 계산하는 방식을 도출하고, 여러 크기의 근사 역수 테이블에서 단정도실수 및 배정도실수의 역수 계산에 필요한 평균 곱셈 횟수를 계산한다. 이들 평균 곱셈 횟수를 종래 알고리즘과 비교하여 본 논문에서 제안한 알고리즘의 우수성을 증명한다.

본 논문의 구성은 다음과 같다. 2장에서는 뉴톤-랍손 역수 알고리즘을 분석해서, 오차를 예측하는 방법을 제안하고 연산 자릿수 및 반복을 종료할 오차 한계를 계산한다. 3장에서는 제안한 알고리즘을 구현하는 회로와 상태 기계를 설계한다. 4장에서는 근사 테이블을 구성하고, 역수 계산에 소요되는 평균 곱셈 횟수를 계산한다. 그리고 그 결과를 종래 뉴톤-랍손 역수 알고리즘과 비교 분석한다. 5장에서 결론을 맺는다.

2. 가변 시간 역수 알고리즘

2.1 뉴톤-랍손 역수 알고리즘

부동소수점 수 F의 역수를 구하기 위해서 함수 'f(X)=F-$\frac{1}{X}$'를 정의한다. 뉴톤-랍손 역수 알고리즘에서 X_n을 X의 근사 값이라고 하면 X_{n+1}은 식 (1)과 같이 주어진다.

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)} = X_n(2-F*X_n) \qquad (1)$$

IEEE-754[10]로 규정되는 부동소수점은 $1.f_2 * 2^{n+base}$ 이다. 가수부 1.f는 단정도실수에서 24 비트, 배정도실수에서는 53 비트이다. 역수의 지수부 연산은 '-n+base'를 계산하는 것으로 가수부 처리와 별도의 하드웨어에 의해서 병렬적으로 처리하므로 본 논문에서는 생략한다.

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

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

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 근사계산하여 테이블 T(g)를 미리 작성해 놓는다. 근사 테이블은 ROM에 저장하거나 또는 별도의 회로를 사용해서 산출하기도 한다. T(g)는 $\frac{1}{1.g}$의 근사계산이므로 'T(g) = $\frac{1}{1.g}+e_t$'이다. e_t는 근사에 따른 오차이다. T(g)를 X의 초기 근사 값 X_0로 정의한다.

식 (1)에서 중간 곱셈 결과는 소수점 이하 p 비트 미만을 절삭한다. p를 연산 유효자릿수라고 가칭한다. 또한 식 (1)에서 '$2-F*X_n$' 뺄셈은 하드웨어 구현 시에 캐리 전달 지연 시간이 필요하다. 이러한 문제점을 해결하기 위해서 본 논문에서는 근사계산인 '$2-e_r-F*X_n$, $e_r=2^{-p}$'을 계산한다. 본 논문에서 사용하는 뉴톤-랍손 부동소수점 역수 알고리즘을 식 (3)에 보인다.

$$X_0 = T(g) = \frac{1}{1.g} + e_t \quad (3)$$
$$\text{For } i = \{0,1,2,...n-1\}$$
$$X_{i+1} = X_i * (2 - e_r - F*X_i)$$

2.2 오차 분석

뉴톤-랍손 알고리즘은 곱셈을 반복하므로 오차가 누적된다. 그러므로 구하고자 하는 부동소수점의 정밀도보다 긴 자리수의 연산이 요구된다.

'$X_i=\frac{1}{F}+e_i$'라 하면 X_{i+1}은 식 (4)가 된다. $t*e_r$과 $u*e_r$은 곱셈 결과를 절삭하면서 발생한 오차이며, '$0\le t,u<1$' 이다.

$$X_{i+1} = (\frac{1}{F}+e_i)(2-e_r-(F*(\frac{1}{F}+e_i)-t*e_r))-u*e_r \quad (4)$$
$$= \frac{1}{F}-Fe_i^2-(\frac{1}{F}-\frac{t}{F}+e_i-te_i+u)e_r = \frac{1}{F} - e_{i+1}$$

식 (4)에서 e_{i+1}이 최대가 되기 위해서는 t=0, u=1이 되어야 한다. 또한 '$F_{min}=1$'이고, '$e_i<1$'이므로 식 (5)가 성립한다.

$$e_{i+1} < Fe_i^2 + 3e_r \quad (5)$$

'$X_i = \frac{1}{F}-e_i$'이라 하면 식 (4)는 '$e_{i+1}< Fe_i^2+2e_r$'이 된다. 그러므로 식 (3)의 반복 연산 중에 절삭으로 발생하는 최대 오차는 $3e_r$보다 작다. '$e_{i+1}< 4e_r$'이면 오차가 충분히 작으므로 반복식을 종료한다. 식 (5)로부터 식 (6)의 조건을 만족하면 식 (3) 알고리즘의 반복을 종료한다.

$$Fe_i^2 < e_r \quad (6)$$

2.3 오차 예측

식 (3)에서 '$2-e_r-F*X_i$'을 계산하고 그 결과에 X_i을 곱해서 X_{i+1}을 산출한다. '$2-e_r-F*X_i$'은 식 (7)과 같이 된다.

$$2-e_r-F*X_i = 2-e_r-F*(\frac{1}{F}+e_i) \quad (7)$$
$$= 1-Fe_i-e_r = 1-2^{-x}$$

식 (7)에서 '$Fe_i+e_r = 2^{-x}$'이라고 하면 식 (8)이 성립한다.

$$Fe_i = 2^{-x}-2^{-p}$$
$$Fe_i^2 = \frac{2^{-2x}}{F} - \frac{2^{-x-p+1}-2^{-2p}}{F} > \frac{2^{-2x}}{F} - 2^{-p} \quad (8)$$

'$X_i=\frac{1}{F}-e_i$'이라 하면 식 (7)과 식 (8)은 '$2-e_r-F*X_i = 1+2^{-x}$, $Fe_i^2 = \frac{2^{-2x}}{F} + \frac{2^{-x-p+1}+2^{-2p}}{F} > \frac{2^{-2x}}{F} - 2^{-p}$'이 된다. 식 (6)과 식 (8)을 정리하면 식 (9)가 된다.

$$\frac{2^{-2x}}{F} - 2^{-p} < Fe_i^2 < 2^{-p}$$
$$\frac{2^{-2x}}{F} < 2*2^{-p} \quad (9)$$

'$F_{min}=1$'이므로 식 (9)는 식 (10)이 된다.

$$2^{-x} = |(2-e_r-F*X_i) - 1| < 2^{\frac{-p+1}{2}} \quad (10)$$

식 (10)을 만족하면 식 (6) 또한 만족하므로 식 (3) 알고리즘 반복을 종료하고, '$X_{i+1}\fallingdotseq\frac{1}{F}$'이다.

2.4 연산 유효자릿수

'$X_{i+1}=\frac{1}{F}-e_{i+1}$'이므로 '$e_{i+1}<4e_r$'은 부동소수점에서 표현할 수 있는 최소값보다 작아야 한다. IEEE-754 단정도실수에서 가수부의 유효자릿수는 24 비트이다. 유효자릿수에 라운드 한 비트를 더하면 25 비트이므로 $4e_r$이 2^{-25}보다 작아야 한다. 즉, '$4e_r = 4*2^{-p} < 2^{-25}$'이 되어야 한다. 따라서 IEEE-754 단정도실수 역수 계산에 필요한 연산 유효자릿수는 '$p = 27$'이다.

IEEE-754 배정도실수에서 가수부의 유효자릿수는 53 비트이다. 유효자릿수에 라운드 한 비트를 더하면 54 비트이므로 $4e_r$이 2^{-54}보다 작아야 한다. 즉, '$4e_r = 4*2^{-p} < 2^{-54}$'가 되어야 한다. 따라서 IEEE-754 배정도실수 역수 계산에 필요한 연산 유효자릿수는 '$p = 56$'이다.

2.5 가변 시간 뉴톤-랍손 역수 알고리즘

식 (10)의 조건을 판단하는 회로는 '$2-e_r-F*X_i$'의 계산

〈표 1〉 가변 시간 뉴톤-랍손 역수 알고리즘

To compute $X = \frac{1}{F}$, where F is (1.g + h). T(g) is pre-calculated approximate $\frac{1}{1.g}$ p = 27 if IEEE-754 single precision p = 57 if IEEE-754 double precision
1) X = T(g) ; 2) Y = 2 - 2^{-p} - F * X ; 3) X = X * Y ; If | Y-1 | > $2^{\frac{-p+1}{2}}$ then goto 2

결과 소수점 이하 $\frac{p-1}{2}$ 비트가 모두 '1' 또는 모두 '0'인가를 판단하는 회로이다. 이로부터 p는 홀수가 되어야 한다. 그러므로 배정도실수에서는 'p = 57'이 되어야 한다.

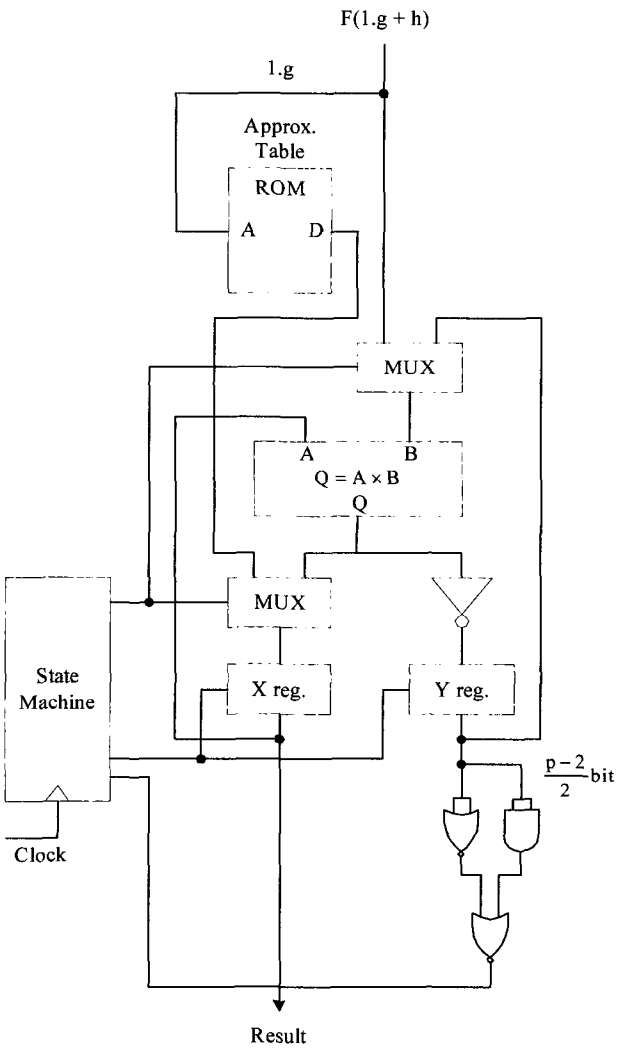
본 논문에서 제안하는 가변 시간 뉴톤-랍손 역수 알고리즘을 정리하면 <표 1>과 같다.

3. 가변 시간 역수 계산기

(그림 1)에 가변 시간 역수 계산기의 블록도를 보인다. 또한 <표 2>에 상태 기계의 흐름을 보인다.

〈표 2〉 가변 시간 뉴톤-랍손 역수 계산기 상태 흐름도

/* Compute $\frac{1}{F}$ */ (State-1) X = Approximate reciprocal table T(g) ; (State-2) Multiplier in port A = X ; Multiplier in port B = F ; /* 1.f */ Multiplier out port Q = Q ; Y = One's complement of Q ; (state-3) Multiplier in port A = X Multiplier in port B = Y ; Multiplier out port Q = X ; If msb $\frac{p-2}{2}$ bits of Y are not all '0' or not all '1', then goto state-2 ;



(그림 1) 가변 시간 뉴톤-랍손 역수 계산기

(그림 1) 블록도와 <표 2>의 상태 흐름도는 제안한 가변 시간 뉴톤-랍손 역수 알고리즘을 하드웨어로 구현한 것이다. 상태-1에서 $\frac{1}{F}$의 근사 값 T(g)를 테이블에서 읽어서 X 레지스타에 저장한다. 상태-2에서 'Y = 2-e_r-F*X_i'를 계산한다. X 레지스타와 F를 곱하고 그 결과를 '2-e_r=1.11$\cdots 1_2$'에서 뺀다. 뺄셈은 곱셈 결과 값의 1의 보수를 취하는 것에 해당한다. 연산 결과를 Y 레지스타에 저장한다. 상태-3에서 'X_{i+1} = X_i*Y'를 연산한다. 동시에 '$|Y-1| < 2^{\frac{-p+2}{2}}$'를 판단한다. 이 판단은 Y의 소수점 이하 $\frac{p-1}{2}$ 비트가 모두 0 또는 모두 1인지를 판별하는 것으로 $\frac{p-1}{2}$ 비트의 입력을 가지는 NOR 게이트와 $\frac{p-1}{2}$ 비트의 입력을 가지는 AND 게이트로 구현할 수 있다. 판단 결과가 맞으면 역수 계산을 종료하며, 맞지 않으면 상태-2로 가서 계속 반복한다.

종래의 뉴톤-랍손 역수 계산기에 $\frac{p-1}{2}$ 개의 입력을 가지는 NOR 게이트와 $\frac{p-1}{2}$ 개의 입력을 가지는 AND 게이트를 추가하고, 상태 흐름도를 일부 변경하는 것으로 가변 시간 뉴톤-랍손 역수 계산기를 구현할 수 있다.

설계한 역수 계산기는 Verilog HDL로 코딩했고 시뮬레이션을 통해서 동작을 확인했다.

4. 연구 결과 및 분석

식 (3)으로부터 초기 오차 e_0는 식 (11)이 된다.

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

식 (11)로부터 e_0는 h=0에서 'T(g)-$\frac{1}{1.g}$'가 되며, $0 \le h \le h_{max}$에서 단조 증가함수이다. 식 (11)로부터 h는 식 (12)가 된다.

$$h = \frac{1}{T(g) - e_0} - 1.g \tag{12}$$

h에 따른 e_0의 변화를 (그림 2)에 보인다.

식 (7)과 식 (10)으로부터 식 (13)이 성립하면 <표 1>의 알고리즘으로부터 2회의 곱셈으로 역수를 계산할 수 있다.

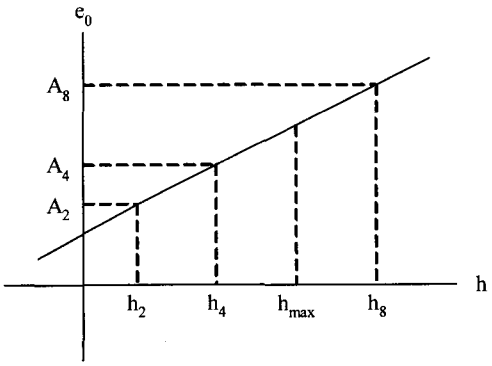
$$2^{-x} = Fe_0 - 2^{-p} < 2^{\frac{-p+2}{2}}$$

$$e_0 < A_2 = \frac{2^{\frac{-p+2}{2}} + 2^{-p}}{1.g + h_{max}} \tag{13}$$

식 (13)에서 분모는 'F =1.g+h'이고, A_2를 최소로 하는 값을 선택했다. (그림 2)에서 h_2는 A_2에서의 h 값이다. 그러므로 '$0 \le h \le h_2$' 구간에서는 2회의 곱셈으로 역수가 계산된다.

식 (5)로부터 e_1은 식 (14)가 된다.

$$e_1 = Fe_0^2 + (2+e_0)*2^{-p} = (1.g+h_{max})e_0^2 + (2+e_0)*2^{-p} \tag{14}$$



(그림 2) h와 e_0의 그라프

식 (14)로부터 '$e_1 < A_2$'이면 4회의 곱셈으로 역수를 계산할 수 있다. 식 (14)에서 '$e_1 = A_2$'이 되는 수치해 '$e_0 = A_4$'를 뉴톤-랍손 알고리즘으로 구할 수 있다. (그림 2)에서 h_4는 A_4에서의 h 값이다. 그러므로 '$h_2 < h \le h_4$' 구간에서는 4회의 곱셈으로 역수가 계산된다.

식 (5)로부터 e_2은 식 (15)가 된다.

$$e_2 = (1.g+h_{max})e_1^2 + (2+e_1)*2^{-p} \tag{15}$$

식 (15)로부터 '$e_2 < A_2$'이면 6회의 곱셈으로 역수를 계산할 수 있다. 식 (15)에서 '$e_2 = A_2$'이 되는 수치해 '$e_0 = A_8$'를 뉴톤-랍손 알고리즘으로 구할 수 있다. (그림 2)에서 '$h_4 < h \le h_{max}$' 구간에서는 6회의 곱셈으로 역수가 계산된다.

'$h > h_8$'인 구간에서는 8회의 곱셈으로 역수를 계산한다.

e_0가 음수인 경우에도 유사한 방법으로 h에 따른 곱셈 회수를 산출할 수 있다.

DasSarma의 연구 결과 최적의 근사 역수는 식 (16)으로 주어진다[11].

$$T(g) = \frac{1}{1.g} \fallingdotseq RN\left(\frac{1}{1.g + 2^{-n_g-1}}\right) \quad (16)$$

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 '$g=g_1g_2\cdots g_{n_g}$=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_g} X (t-1)' 비트가 되어서, 테이블의 길이는 2^{n_g}이며, 폭은 't-1' 비트이다.

본 논문에서 제안한 알고리즘에 의한 IEEE 단정도실수 역수 계산에 필요한 곱셈 횟수를 <표 3>에, 배정도실수 역수 계산에 필요한 곱셈 회수를 <표 4>에 각각 보인다.

〈표 3〉 IEEE 단정도실수 역수 계산에 필요한 곱셈 횟수

Table size	Average No. of multiply	No. of Multiply			
		2 mul	4 mul	6 mul	8 mul
16 X 3	5.53	0.26%	22.9%	76.8%	0.0%
32 X 4	5.06	0.53%	45.7%	53.7%	0.0%
64 X 5	4.30	1.08%	83.0%	15.9%	0.0%
128 x 6	3.96	2.15%	97.8%	0.0%	0.0%
256 x 7	3.91	4.31%	95.7%	0.0%	0.0%
64 x 6	3.96	1.87%	98.1%	0.05%	0.0%
32 x 5	4.48	0.92%	74.0%	25.1%	0.0%
32 x 6	4.27	1.16%	84.2%	14.6%	0.0%
32 x 7	4.14	1.16%	90.6%	8.3%	0.0%
32 x 8	4.11	1.16%	91.9%	6.9%	0.0%

〈표 4〉 IEEE 배정도실수 역수 계산에 필요한 곱셈 횟수

Table size	Average No. of multiply	No. of Multiply			
		2 mul	4 mul	6 mul	8 mul
64 X 5	6.71	0.0%	0.54%	63.2%	36.3%
128 X 6	6.00	0.0%	1.08%	97.8%	1.07%
256 X 7	5.96	0.0%	2.16%	97.8%	0.0%
512 X 8	5.91	0.0%	4.32%	95.7%	0.0%
128 x 7	5.96	0.0%	1.87%	98.1%	0.0%
64 X 6	6.20	0.0%	0.93%	88.3%	10.7%
64 x 7	5.99	0.0%	1.17%	98.1%	0.73%
64 x 8	5.98	0.0%	1.17%	98.6%	0.24%

종래 뉴톤-랍손 역수 알고리즘에서는 최대 오차를 고려해서 반복 횟수를 정했다. 즉, 종래 알고리즘에 의한 단정도 실수 역수 계산기는 '64x5' 테이블을 사용하면 6회의 곱셈을 수행하였고, '128x6' 테이블을 사용하면 4회의 곱셈을 수행하였음을 <표 3>으로부터 알 수 있다. 그러나 본 논문에서 제안한 알고리즘에서는 '64X5' 테이블에서 평균 4.30회의 곱셈, '128x6' 테이블과 '64x6' 테이블에서 평균 3.96 회의 곱셈으로 역수 계산을 할 수 있다. 즉, 본 논문에서 제안한 알고리즘에서 평균 4회의 곱셈으로 역수 계산을 하려면 '64x6' 테이블을 사용하면 되므로, 종래 알고리즘에서 사용하던 '128x6' 테이블과 비교하여 테이블 크기를 반으로 줄일 수 있다.

이러한 결과는 배정도실수 연산에서도 동일하게 나타난다. 배정도실수 역수 계산은 <표 4>로부터 종래 알고리즘에서는 '128X6' 테이블을 사용하면 8회의 곱셈, '256X7' 테이블을 사용하면 6회의 곱셈으로 역수를 계산하였다. 그러나 본 논문에서 제안한 알고리즘에서는 '128X6' 테이블을 사용하면 평균 6.00회의 곱셈, '256X7' 테이블을 사용하면 5.96회의 곱셈으로 역수를 계산한다. 또한 '128X7' 테이블을 사용하면 평균 5.96회, '64X7' 테이블을 사용하면 평균 5.99회의 곱셈으로 역수를 계산한다. 따라서 평균 6회 곱셈으로 역수를 계산하려면 종래 알고리즘에서는 '256X7' 테이블을 사용했지만, 본 논문에서 제안하는 알고리즘에서는 '64X7' 테이블을 사용해서 평균 6회 곱셈으로 역수를 계산할 수 있다. '64X7' 테이블의 크기는 '256X7' 테이블의 사분의 일이다. 따라서 본 논문에서 제안한 알고리즘은 테이블 크기를 사분의 일로 줄이면서 동일한 성능을 보인다.

역수 근사 테이블의 길이와 폭의 관계를 <표 3> 및 <표 4>로부터 알 수 있다. <표 3>의 단정도실수 역수 계산에서 '32X4', '32x5', '32x6', '32x7' 및 '32x8' 테이블은 길이는 32로 같으며, 폭은 4 비트에서 8 비트까지로 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 6회의 곱셈으로 역수를 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 5.06회, 4.48회, 4.27회, 4.14회와 4.11회의 곱셈으로 역수를 계산한다. 이들 결과로부터 테이블의 폭을 1-2 비트 증가시키면 성능이 크게 개선되지만 3 비트 이상을 증가시켜도 성능이 개선되지 않음을 알 수 있다.

또한 '64x5' 테이블을 사용하면 평균 4.30회 곱셈으로 역수를 계산한다. 이것은 '32x6' 테이블의 4.27회와 비슷하다. 반면에 '32x6' 테이블의 크기는 '64x5' 테이블의 약 반이다. 따라서 폭을 늘리는 것이 길이를 크게 하는 것보다 유리하다.

테이블의 길이와 폭의 관계는 배정도실수에서도 동일하게 나타난다. <표 4>에서 '64x5', '64x6', '64x7'과 '64x8' 테이블은 길이가 64로 같지만 폭은 5 비트에서 8 비트까지 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 8회의 곱셈

으로 역수를 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 6.71회, 6.20회, 5.99회와 5.98회의 곱셈으로 역수를 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 우수함을 알 수 있다. 또한 근사 테이블의 길이를 크게 하는 것보다 폭을 늘리는 것이 가격대비 성능면에서 유리함을 알 수 있다.

종래의 뉴톤-랍손 역수 알고리즘에서는 테이블 크기가 제한적이었다. 단정도실수에서 6회 곱셈으로 역수를 계산하기 위해서는 '16X3' 테이블을, 4회 곱셈을 위해서는 '128X6' 테이블을 사용하였다. 중간 크기의 테이블은 의미가 없었다. 즉, '32X4' 테이블과 '64X5' 테이블을 사용하면 역수 계산에 6회의 곱셈이 필요하였다.

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

5. 결 론

부동소수점 나눗셈은 뺄셈을 반복하는 SRT 알고리즘과 곱셈을 반복하는 뉴톤-랍손(Newton-Raphson) 역수 알고리즘 및 골드스미트(Goldschmidt) 나눗셈 알고리즘이 있다. 뉴톤-랍손 역수 알고리즘은 제수의 역수를 피제수에 곱해서 나눗셈을 계산한다. 역수 계산은 제수의 역수의 근사 값을 초기 값으로 해서 반복 연산으로 오차를 줄여나간다. 반복 연산을 수행할 때마다 상대 오차는 자승으로 줄어들며, 한 회의 반복 연산에 2회의 곱셈이 필요하다.

본 논문에서는 뉴톤-랍손 부동소수점 역수 알고리즘의 반복 연산 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 연산을 수행하는 가변 시간 뉴턴-랍손 부동소수점 역수 알고리즘을 제안하였다. 제안한 알고리즘을 구현하는 회로와 상태 기계를 구성하였다, 또한 근사 테이블을 구성하고, 역수 계산에 소요되는 평균 곱셈 횟수를 계산해서 그 결과를 종래의 뉴톤-랍손 알고리즘과 비교 분석하였다.

종래의 뉴톤-랍손 역수 알고리즘에 의한 배정도실수 역수 계산은 '128X6' 근사 역수 테이블을 사용하면 8회의 곱셈, '256X7' 테이블을 사용하면 6회의 곱셈을 수행하였다. 그러나 본 논문에서 제안한 가변 시간 뉴톤-랍손 역수 알고리즘에서는 '64X7' 테이블을 사용해서 평균 5.99회의 곱셈으로 역수를 계산할 수 있었다, '64X7' 테이블의 크기는 '256X7' 테이블의 사분의 일에 불과하다. 따라서 본 논문에서 제안한 알고리즘은 근사 테이블 크기를 크게 줄일 수 있다

본 논문의 연구 결과 근사 역수 테이블의 폭을 크게 하는 것이 길이를 크게 하는 것보다 가격대비 성능에서 유리함을 보였다. 단정도실수 역수에서 '32x5' 테이블은 평균 4.48회 곱셈으로 역수를 계산하였다. 그리고 '64x5' 테이블을 사용하면 평균 4.30회 곱셈으로 역수를 계산하였다. '32x6' 테이블의 면적은 '64x5' 테이블의 약 반이지만 비슷한 성능을 보였다. 이러한 테이블의 길이와 폭의 관계는 배정도실수에서도 동일하게 나타났다.

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

참 고 문 헌

[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, 1997.

[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 Microprocessors," Proc. 14th IEEE Symp. Computer Arithmetic, pp. 106-115, Apr. 1999.

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

김 성 기

e-mail : heewookim@empal.com

1984년 울산대학교 응용물리학과 졸업

1994년 부경대학교 산업대학원 전자계산학과 졸업

2000년~현재 제이미(주) 대표이사

2004년~현재 부경대학교 대학원 컴퓨터공학과 박사과정

관심분야 : 전산기구조, 반도체 회로 설계

조 경 연

e-mail : gycho@pknu.ac.kr

1990년 인하대학교 공과대학 전자공학과 공학박사

1983년~1991년 삼보컴퓨터 기술연구소 책임연구원

1991년~2000년 삼보컴퓨터 기술연구소 기술고문

1998년~2003년 에이디칩스 기술고문

1991년~현재 부경대학교 공과대학 전자컴퓨터정보통신공학부 교수

관심분야 : 전산기구조, 반도체 회로 설계, 암호 알고리즘