가변 시간 뉴톤-랍손 부동소수점 역수 제곱근 계산기

김 성 기† · 조 경 연††

요 약

부동소수점 제곱근 계산에 많이 사용하는 뉴톤-랍손 부동소수점 역수 제곱근 알고리즘은 일정한 횟수의 곱셈을 반복하여 역수 제곱근을 계산한다. 본 논문에서는 뉴톤-랍손 역수 제곱근 알고리즘의 반복 과정의 오차를 예측하여 오차가 정해진 값보다 작아지는 시점까지 반복 연산하는 알고리즘을 제안한다.

'F'의 역수 제곱근 계산은 초기값 '$X_0=\frac{1}{\sqrt{F}}\pm e_0$'에 대하여, '$X_{i+1}=\frac{X_i(3-e_r-FX_i^2)}{2}$, $i\in\{0, 1, 2,...n-1\}$'을 반복한다. 중간 곱셈 결과는 소수점 이하 p 비트 미만을 절삭하며, 절삭 오차는 '$e_r=2^{-p}$' 보다 작다. p는 단정도실수에서 28, 배정도실수에서 58이다. '$X_i=\frac{1}{\sqrt{F}}\pm e_i$'라고 하면 '$X_{i+1}=\frac{1}{\sqrt{F}}-e_{i+1}$, $e_{i+1}<\frac{3\sqrt{F}e_i^2}{2}\mp\frac{Fe_i^3}{2}+2e_r$'이 된다. '$|\frac{3-e_r-FX_i^2}{2}-1|<2^{\frac{-p}{2}}$'이면, '$e_{i+1}<8e_r$'이 부동소수점으로 표현 가능한 최소값보다 작아지며, '$X_{i+1}\fallingdotseq\frac{1}{\sqrt{F}}$'이다.

본 논문에서 제안한 알고리즘은 입력 값에 따라서 곱셈 횟수가 다르므로, 평균 곱셈 횟수를 계산하는 방식을 도출하고, 여러 크기의 근사 역수 제곱근 테이블($X_0=\frac{1}{\sqrt{F}}\pm e_0$)에서 단정도실수 및 배정도실수의 역수 제곱근 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 반복하므로 역수 제곱근 계산기의 성능을 높일 수 있다. 또한 최적의 근사 역수 제곱근 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.

키워드 : 부동소수점, 뉴톤-랍손, 역수 제곱근, 가변시간

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

Sung-Gi Kim† · Gyeong-Yeon Cho††

ABSTRACT

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

To find the reciprocal square root of a floating point number F, the algorithm repeats the following operations: '$X_{i+1}=\frac{X_i(3-e_r-FX_i^2)}{2}$, $i\in\{0, 1, 2,...n-1\}$' with the initial value is '$X_0=\frac{1}{\sqrt{F}}\pm e_0$'. 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 28 for the single precision floating point, and 58 for the double precision floating point. Let '$X_i=\frac{1}{\sqrt{F}}\pm e_i$', there is '$X_{i+1}=\frac{1}{\sqrt{F}}-e_{i+1}$', where '$e_{i+1}<\frac{3\sqrt{F}e_i^2}{2}\mp\frac{Fe_i^3}{2}+2e_r$'. If '$|\frac{3-e_r-FX_i^2}{2}-1|<2^{\frac{-p}{2}}$' is true, '$e_{i+1}<8e_r$' is less than the smallest number which is representable by floating point number. So, X_{i+1} is approximate to '$\frac{1}{\sqrt{F}}$'.

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 square root tables ($X_0=\frac{1}{\sqrt{F}}\pm e_0$) 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 square root unit. Also, it can be used to construct optimized approximate reciprocal square root 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.

Key Words : Floating Point, Newton-raphson, Reciprocal Square Root, Variable Latency

† 정 회 원 : 부경대학교 대학원 컴퓨터공학과 박사과정
†† 정 회 원 : 부경대학교 공과대학 전자컴퓨터정보통신공학부 교수
논문접수 : 2004년 10월 28일, 심사완료 : 2005년 7월 18일

1. 서 론

부동소수점 계산기(floating point unit)는 디지털 신호처리, 컴퓨터 그라픽스, 과학 기술 계산, CAD 등에서 필수적인 기능으로 많은 CPU에서 채용하고 있다. 최근에는 음성처리, 3차원 그라픽, 동영상 등의 멀티미디어 분야에서도 부동소수점 계산이 증가하면서 휴대용 통신 기기를 포함한 SoC(System on Chip)에서도 하드웨어 부동소수점 계산기를 도입하고 있다. 부동소수점에서 나눗셈 및 제곱근 계산은 덧셈(뺄셈) 및 곱셈보다 출현 빈도가 낮기 때문에 상대적으로 연구 성과가 미흡했다. 이런 결과로 덧셈은 2-4 사이클, 곱셈은 2-5 사이클만에 배정도실수 연산을 할 수 있지만, 나눗셈은 9 사이클에서 60 사이클 이상, 제곱근은 15 사이클에서 70 사이클 이상 소요되고 있다[1]. Oberman과 Flynn의 연구[2]는 나눗셈과 제곱근은 덧셈(뺄셈) 및 곱셈보다 출현 빈도가 낮지만, 시스템에서 나눗셈과 제곱근의 수행 시간이 덧셈이나 곱셈과 비슷하게 소요됨을 보이고 있다. 또한 제곱근 계산은 음성이나 3차원 그라픽 같은 멀티미디어 분야에서 출현 빈도가 높다. 따라서 제곱근 계산 속도를 높이는 연구가 요구되고 있다.

부동소수점 제곱근 계산은 뺄셈을 반복하는 SRT[3-4] 알고리즘과 곱셈을 반복하는 뉴톤-랍손(Newton-Raphson) 역수 제곱근 알고리즘 및 골드스미트(Goldschmidt) 제곱근 알고리즘이 있다[5-9]. SRT 알고리즘은 덧셈(뺄셈), 부분 곱셈, 몫 결정의 3 단계 과정을 반복하는 알고리즘으로 매 반복마다 일정한 비트의 몫을 얻을 수 있다. 이러한 SRT 알고리즘은 하드웨어가 간단하고, 정밀 계산이 가능하지만, 연산 속도가 느린 단점을 가진다.

곱셈을 반복하는 방식은 빠른 곱셈기와 근사 테이블을 사용하며, 반복할 때마다 오차가 자승에 비례해서 줄어들므로 연산 속도가 빠른 장점이 있지만, 근사계산만이 가능하다. 그러나 소수의 정밀 과학 기술 연산 분야를 제외하고 대부분 멀티미디어, 디지털 신호처리, 컴퓨터 그라픽스 등에서는 근사계산만으로 실용상 문제가 없다.

뉴톤-랍손 역수 제곱근 알고리즘은 X_0을 $\frac{1}{\sqrt{F}}$의 근사 값이라고 하면, '$X_{i+1}=\frac{X_i(3-FX_i^2)}{2}$, i∈{0, 1, 2,....n-1)'을 반복하면 '$X_n=\frac{1}{\sqrt{F}}$'이 된다. 이 알고리즘은 오차를 스스로 조정하므로 절삭에 따른 오차 누적이 발생하지 않는다.

본 논문에서는 뉴톤-랍손 역수 제곱근 알고리즘의 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복하는 알고리즘을 제안한다. '$X_i=\frac{1}{\sqrt{F}}+e_i$'라고 하면 '$e_{i+1}<\frac{3\sqrt{F}e_i^2}{2}+\frac{Fe_i^3}{2}+2e_r$'이 된다. '$e_r=2^{-p}$'은 절삭 오차이고, p는 연산 유효자릿수이다. e_{i+1}이 부동소수점으로 표현 가능한 최소값보다 작아지는 '$\left|\frac{3-e_r-FX_i^2}{2}-1\right|<2^{\frac{-p}{2}}$'에서 반복 계산을 마친다.

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

본 논문에서 제안한 알고리즘은 부동소수점 값에 따라서 곱셈 횟수가 다르므로, 평균 곱셈 횟수를 계산하는 방식을 유도하고, 여러 크기의 근사 테이블에서 단정도실수 및 배정도실수의 역수 제곱근 계산에 필요한 평균 곱셈 횟수를 계산한다. 이들 평균 곱셈 횟수를 종래 알고리즘과 비교, 분석한 결과 테이블 크기를 최고 반 이하로 줄일 수 있었다.

기존의 뉴톤-랍손 역수 제곱근 알고리즘은 최대 오차를 계산하고, 항상 일정 회수를 반복 계산하였다. 이러한 기존 방식에서는 구하고자 하는 결과 값에 도달했음에도 불구하고 가외의 연산을 수행하여 연산 속도를 저하시키는 단점이 있었다. 본 논문에서 제안한 알고리즘은 이러한 종래의 단점을 개선하여 역수 제곱근 계산기의 성능을 높일 수 있다. 또한 요구하는 역수 제곱근 계산기 성능에 따른 최적의 근사 역수 제곱근 테이블을 구성할 수 있다.

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

2. 가변 시간 역수 제곱근 알고리즘

2.1 뉴톤-랍손 역수 제곱근 알고리즘

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

$$X_{i+1} = X_i - \frac{f(X_i)}{f'(X_i)} = \frac{X_i(3-FX_i^2)}{2} \quad (1)$$

IEEE-754[10]로 규정되는 부동소수점은 $1.f_2 * 2^{n+base}$이다. 가수부 1.f는 단정도실수에서 24 비트, 배정도실수에서는 53 비트이다. 지수부 n이 홀수이면 $0.1f_2 * 2^{n+1+base}$로 변환한다. 따라서 본 논문의 부동소수점 가수부는 '0.5≤i.f<2.0'이다. 지수부 연산은 가수부와 독립적인 회로에 의해서 병렬적으로 계산하므로, 본 논문에서는 생략한다.

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

$$F = i.g + h \quad (2)$$

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 $\frac{1}{\sqrt{i.g}}$를 근사 계산하여 테이블 T(g)를 미리 작성해 놓는다. 근사 테이블은 ROM에 저장하거나 또는 별도의 회로를 사용해서 산출하기도 한다. T(g)는 $\frac{1}{\sqrt{i.g}}$의 근사계산이므로 '$T(g)=\frac{1}{\sqrt{i.g}}+e_t$' 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 '$3-FX_n^2$'은 하드웨어 구현 시에 캐리 전달 지연 시간이 필요하다. 이러한 문제점을 해결하기 위해서 본 논문에서는 근사계산인 '$3-e_r-F*X_n^2$, $e_r=2^{-p}$'을 계산한다. 본 논문에서 사용하는 뉴톤-랍손 부동소수점 역수 제곱근 알고리즘을 식 (3)에 보인다.

$$X_0 = T(g) = \sqrt{\frac{1}{i.g}} + e_t \tag{3}$$
$$\text{For } i \in \{0,1,2,\ldots n-1\}$$
$$X_{i+1} = X_i * \frac{X_i(3-e_r-FX_i^2)}{2}$$

2.2 오차 분석

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

'$X_i = \frac{1}{\sqrt{F}}+e_i$'라 하면 X_{i+1}은 식 (4)와 식 (5)가 된다. $t*e_r$, $u*e_r$와 $v*e_r$은 곱셈 결과를 절삭하면서 발생하는 절삭 오차이며, '$0 \le t,u,v < 1$' 이다.

$$\frac{3-e_r-FX_i^2}{2} = \frac{3-e_r-(F*((\frac{1}{\sqrt{F}}+e_i)^2-t*e_r)-u*e_r)}{2}$$
$$= 1-\sqrt{F}e_i-\frac{Fe_i^2}{2}-\frac{1-tF-u}{2}e_r = 1-2^{-x} \tag{4}$$

$$X_i(\frac{3-e_r-FX_i^2}{2})$$
$$=(\frac{1}{\sqrt{F}}+e_i)(1-\sqrt{F}e_i-\frac{Fe_i^2}{2}-\frac{1-tF-u}{2}e_r)-v*e_r$$
$$=\frac{1}{\sqrt{F}}-(\frac{3\sqrt{F}e_i^2}{2}+\frac{Fe_i^3}{2}+(\frac{1-tF-u}{2\sqrt{F}}+\frac{1-tF-u}{2}e_i+v)e_r)$$
$$=\frac{1}{\sqrt{F}} - e_{i+1} \tag{5}$$

식 (5)에서 e_r항은 중간 곱셈 결과를 절삭하면서 발생하는 오차이다. e_r항이 최대가 되기 위해서는 't=u=0, v=1'이 되어야 하며, e_{i+1}은 식 (6)이 된다.

$$e_{i+1} < \frac{3\sqrt{F}e_i^2}{2}+\frac{Fe_i^3}{2}+2e_r \tag{6}$$

'$X_i=\frac{1}{\sqrt{F}}-e_i$'이라 하면 식 (4), 식 (5) 및 식 (6)은 다음과 같이 된다.

$$\frac{3-e_r-FX_i^2}{2} = 1+\sqrt{F}e_i-\frac{Fe_i^2}{2}-\frac{1-tF-u}{2}e_r = 1+2^{-x} \tag{4'}$$

$$X_i(\frac{3-e_r-FX_i^2}{2}) = \frac{1}{\sqrt{F}} - e_{i+1} \tag{5'}$$
$$= \frac{1}{\sqrt{F}} -(\frac{3\sqrt{F}e_i^2}{2}-\frac{Fe_i^3}{2}+(\frac{1-tF-u}{2\sqrt{F}}+\frac{1-tF-u}{2}e_i+v)e_r)$$

$$e_{i+1} < \frac{3\sqrt{F}e_i^2}{2}-\frac{Fe_i^3}{2}+2e_r \tag{6'}$$

식 (6)과 식 (6')로부터 식 (3)의 반복 연산 중에 절삭으로 발생하는 최대 오차는 $2e_r$보다 작다. '$\frac{3\sqrt{F}e_i^2}{2}\pm\frac{Fe_i^3}{2} < 6e_r$'이 되면, '$e_{i+1} < 8e_r$'이고, e_{i+1}이 충분히 작으므로 식 (3) 알고리즘의 반복을 종료한다. 즉, 식 (7)의 조건을 만족하면 식 (3) 알고리즘의 반복을 종료한다.

$$3\sqrt{F}e_i^2 \pm Fe_i^3 < 12e_r \tag{7}$$

2.3 오차 예측

식 (7)을 만족시키는 e_i은 단정도실수에서도 2^{-12}를 넘지 않으므로 '$\frac{Fe_i^3}{3\sqrt{F}e_i^2} << \frac{1}{12}$'이다. 이로부터 식 (8)이 성립한다.

$$3\sqrt{F}e_i^2 < 11e_r$$
$$e_i < \sqrt{\frac{11e_r}{3\sqrt{F}}} \tag{8}$$

식 (4)와 식 (4')에 식 (8)을 대입하면 식 (9)가 된다.

$$|\frac{3-e_r-FX_i^2}{2} - 1| = 2^{-x}$$
$$< \sqrt{\frac{11\sqrt{F}}{3}}\sqrt{e_r} \pm (\frac{11\sqrt{F}}{6}+\frac{1-tF-u}{2})e_r$$
$$< \sqrt{e_r} = 2^{\frac{-p}{2}} \tag{9}$$

식 (9)의 조건을 만족하면 식 (7)도 만족되므로 식 (3) 알고리즘의 반복을 종료하고, '$X_{i+1} \fallingdotseq \frac{1}{\sqrt{F}}$'이다.

2.4 연산 유효자릿수

'$X_{i+1}=\frac{1}{\sqrt{F}}-e_{i+1}$'이므로 '$e_{i+1}<8e_r$'은 부동소수점 유효자릿수보다 작아야 한다. IEEE-754 단정도실수에서 가수부의 유효자릿수는 24 비트이다. 유효자릿수에 라운드 한 비트를 더하면 25 비트이므로 $8e_r$이 2^{-25}보다 작아야 한다. 그러므로 '$8e_r=8*2^{-p}<2^{-25}$'이 되어야 한다. 따라서 IEEE-754 단정도실수 역수 제곱근 계산에 필요한 연산 유효자릿수는 'p = 28'이다.

IEEE-754 배정도실수에서 가수부의 유효자릿수는 53 비트이다. 유효자릿수에 라운드 한 비트를 더하면 54 비트이므로 $8e_r$이 2^{-54}보다 작아야 한다. 그러므로 '$8e_r=8*2^{-p}<2^{-54}$'가 되어야 한다. 따라서 IEEE-754 배정도실수 역수 제곱근 계산에 필요한 연산 유효자릿수는 'p = 57'이다.

2.5 가변 시간 뉴톤-랍손 역수 제곱근 알고리즘

식 (9)의 조건을 판단하는 회로는 '$\frac{3-e_r-FX_i^2}{2}$'의 계산 결과 소수점 이하 $\frac{p}{2}$ 비트가 모두 '1' 또는 모두 '0'인가를 판단하는 회로이다. 이로부터 p는 짝수가 되어야 한다. 그러므로 배정도실수에서는 'p = 58'이 되어야 한다.

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

〈표 1〉 가변 시간 뉴톤-랍손 역수 알고리즘

To compute, $X = \frac{1}{\sqrt{F}}$ where F is (i.g + h).

T(g) is pre-calculated approximate $\frac{1}{\sqrt{i.g}}$

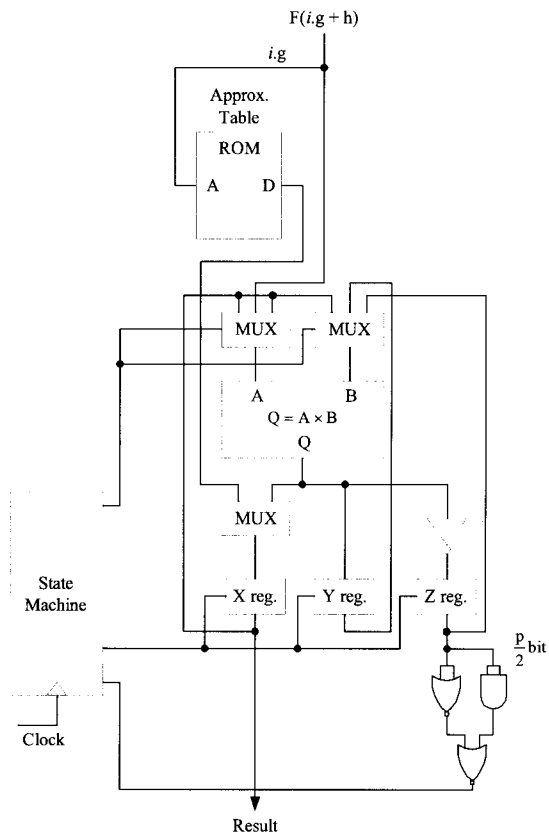
p = 28 if IEEE-754 single precision
p = 58 if IEEE-754 double precision

1) X = T(g) ;
2) Y = X * X ;
3) Q = F * Y ;
$Z = \frac{3-2^{-p}-Q}{2}$;
4) X = X * Z ;
If $|Z-1| > 2^{\frac{-p}{2}}$ then goto 2

3. 가변 시간 역수 제곱근 계산기

2장에서 제안한 알고리즘을 하드웨어로 설계한 블록도를 (그림 1)에 보인다. (그림 1) 블록도내의 상태 기계는 식 (9)의 조건을 판별하는 기능을 수행하며 상태 흐름도를 <표 2>에 보인다.

(그림 1) 블록도와 <표 2>의 상태 흐름도는 제안한 가변 시간 뉴톤-랍손 역수 제곱근 알고리즘을 하드웨어로 구현한 것이다. 상태-1에서 $\frac{1}{\sqrt{i.g}}$의 근사 값을 테이블로부터 읽어서 X 레지스타에 저장한다. 상태-2에서 '$Y = X_i^2$'을 계산하여 임시 저장 장소인 Y 레지스타에 저장한다. 상태-3에서 '$Q=FX_i^2$'을 계산하고, '$Z = \frac{3-e_r-FX_i^2}{2} = \frac{3-e_r-Q}{2}$'을 계



(그림 1) 가변 시간 뉴톤-랍손 역수 제곱근 계산기

〈표 2〉 가변 시간 뉴톤-랍손 역수 제곱근 계산기 상태 흐름도

/* Compute $\frac{1}{\sqrt{F}}$ */
(State-1)
X = Approximate reciprocal square root table T(g) ;
(State-2)
Multiplier in port A = X ;
Multiplier in port B = X ;
Multiplier out port Q = Y ;
(state-3)
Multiplier input port 1 = F ;
Multiplier input port 2 = Y ;
Multiplier output port = Q ;
If Q ≥ 1.0 then T = 1 ;
else T = 2 ;
T = T + (1's complement of Q's fraction part) ;
Z = T >> 1 ;
(state-4)
Multiplier input port 1 = X ;
Multiplier input port 2 = Z ;
Multiplier output port = X ;
If msb $\frac{p}{2}$ bits of Z are not all '0' or not all '1', then goto state-2 ;

산한다. '3-e_r-Q = 10.11$\cdots$$1_2$-Q = T'이다. 'Q ≥ 1.0'이면 'T = 1.xxxxx$_2$'이고, 'Q < 1.0'이면 'T = 10.xxxxx$_2$'이다. T의 소수점이하 '0.xxxxx$_2$'는 Q의 소수점이하의 1의 보수이다. 나누기 2는 오른쪽으로 한 비트 이동하는 것이다. 계산된 Z를 Z 레지스타에 저장한다. 상태-3에서 '$X_{i+1} = X_i * Z$'를 연산한다. 동시에 '$|Z-1| < 2^{\frac{-p}{2}}$'를 판단한다. 이 판단은 Z의 소수점 이하 $\frac{p}{2}$ 비트가 모두 '0' 또는 모두 '1'인지를 판별하는 것으로 $\frac{p}{2}$개의 입력을 가지는 NOR 게이트와 $\frac{p}{2}$개의 입력을 가지는 AND 게이트로 구현할 수 있다. 판단 결과가 맞으면 계산을 종료하며, 맞지 않으면 상태-2로 가서 계속 반복한다.

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

설계한 역수 제곱근 계산기는 Verilog HDL로 코딩하고, 시뮬레이션해서 정상적으로 동작하는 것을 확인하였다.

4. 연구 결과 및 분석

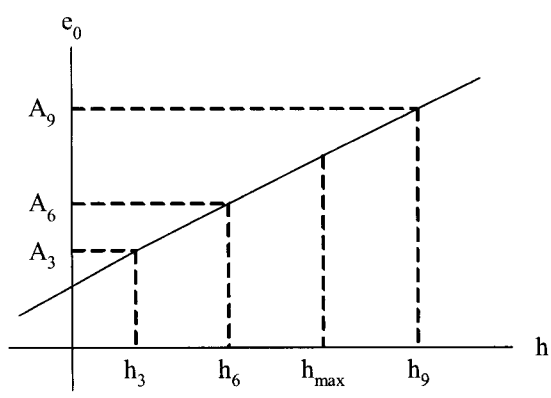
식 (3)으로부터 초기 오차 e_0는 식 (10)이 된다.

$$e_0 = T(g) - \frac{1}{\sqrt{i.g+h}} \tag{10}$$

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

$$h = \frac{1}{(T(g)-e_0)^2} - i.g \tag{11}$$

h에 따른 e_0의 변화를 (그림 2)에 보인다.



(그림 2) h와 e_0의 그라프

식 (4)와 식 (9)로부터 식 (12)가 성립하면 3회의 곱셈으로 역수 제곱근을 계산할 수 있다.

$$2^{-x} = \sqrt{F}\,e_0 + \frac{Fe_0^2}{2} + \frac{1-tF-u}{2}e_r \fallingdotseq \sqrt{F}\,e_0 + \frac{Fe_0^2}{2} < \sqrt{e_r}$$

$$e_0 < A_3 = \frac{-\frac{2}{\sqrt{i.g+h_{max}}} + \sqrt{\frac{4}{i.g+h_{max}} + \frac{8}{i.g+h_{max}}\sqrt{e_r}}}{2} \tag{12}$$

식 (12)에서 분모는 'F = i.g + h'이므로 A_3을 최소로 하는 값을 선택했다. (그림 2)에서 h_3는 A_3에서의 h 값이다. 그러므로 '$0 \le h \le h_3$' 구간에서는 3회의 곱셈으로 역수 제곱근이 계산된다.

식 (5)로부터 e_1은 식 (13)이 된다.

$$e_1 \fallingdotseq \frac{3\sqrt{i.g+h_{max}}e_0^2}{2} + \frac{(i.g+h_{max})e_0^3}{2} + 2e_r \tag{13}$$

식 (13)으로부터 '$e_1 < A_3$'이면 6회의 곱셈으로 역수 제곱근을 계산할 수 있다. 식 (13)에서 '$e_1 = A_3$'이 되는 수치해 '$e_0 = A_6$'를 뉴톤-랍손 알고리즘으로 구할 수 있다. (그림 2)에서 h_6는 A_6에서의 h 값이다. 그러므로 '$h_3 < h \le h_6$' 구간에서는 6회의 곱셈으로 역수 제곱근을 계산한다.

식 (5)로부터 e_2는 식 (14)가 되며, 식 (14)에서 '$e_2 = A_6$'가 되는 수치해 '$e_0 = A_9$'를 뉴톤-랍손 알고리즘으로 구할 수 있다.

$$e_2 \fallingdotseq \frac{3\sqrt{i.g+h_{max}}e_1^2}{2} + \frac{(i.g+h_{max})e_1^3}{2} + 2e_r \tag{14}$$

(그림 2)에서 h_9는 A_9에서의 h 값이다. '$h_6 < h \le h_{max}$' 구간에서는 9회의 곱셈으로 역수 제곱근이 계산된다. '$h > h_9$'인 구간에서는 12회의 곱셈으로 역수 제곱근을 계산한다.

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

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

$$T(g) = \frac{1}{\sqrt{i.g}} \fallingdotseq RN\left(\frac{1}{\sqrt{i.g+2^{-n_g-1}}}\right) \tag{15}$$

RN is round to nearest

T(g)의 소수점 이하 길이를 t 비트라고 하면 '$T(g)=(b_0.b_1 b_2\cdots b_t)_2$, $\frac{1}{\sqrt{2}} < T(G) \le \sqrt{2}$'이다. '$b_0b_1$=10'과 '$b_0b_1$=01'인 두 가지 경우만 존재하므로, 근사 역수 테이블에는 '$b_1b_2\cdots b_t$'만을 저장하면 된다. 근사 역수 테이블의 크기는 '2^{n_g} X t' 비트가 되어서, 테이블의 길이는 2^{n_g}이며, 폭은 t 비트이다.

〈표 3〉 IEEE 단정도실수 역수 제곱근 계산에 필요한 곱셈 횟수

Table size	Average No. of multiply	No. of Multiply			
		3 mul	6 mul	9 mul	12 mul
24 X 4	8.45	0.17%	18.0%	81.8%	0.0%
48 X 5	7.86	0.36%	37.4%	62.2%	0.0%
96 X 6	6.92	0.71%	67.9%	31.4%	0.0%
192 x 7	5.96	1.43%	98.5%	0.02%	0.0%
384 x 8	5.91	2.87%	97.1%	0.0%	0.0%
96 x 7	6.10	1.24%	94.2%	4.56%	0.0%
96 x 8	5.98	1.80%	97.2%	1.04%	0.0%
96 x 9	5.97	1.95%	97.2%	0.81%	0.0%

〈표 4〉 IEEE 배정도실수 역수 제곱근 계산에 필요한 곱셈 횟수

Table size	Average No. of multiply	No. of Multiply			
		3 mul	6 mul	9 mul	12 mul
96 X 6	10.39	0.0%	0.41%	53.0%	46.6%
192 X 7	9.16	0.0%	0.83%	93.0%	6.16%
384 X 8	8.95	0.0%	1.66%	98.3%	0.0%
768 x 9	8.90	0.0%	3.33%	96.7%	0.0%
192 x 8	8.96	0.0%	1.43%	98.5%	0.02%
96 x 7	9.48	0.0%	0.72%	82.5%	16.8%
96 x 8	9.13	0.0%	1.04%	93.6%	5.40%
96 X 9	9.10	0.0%	1.12%	94.7%	4.20%

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

종래 뉴톤-랍손 역수 제곱근 알고리즘에서는 최대 오차를 고려해서 반복 횟수를 정했다. 즉, 종래 알고리즘에 의한 단정도실수 역수 제곱근은 '192x7' 테이블을 사용하면 9회의 곱셈을 수행하였고, '384x8' 테이블을 사용하면 6회의 곱셈을 수행하였음을 <표 3>으로부터 알 수 있다. 또한 '96x8' 테이블을 사용해도 9회의 곱셈을 수행하였었다.

그러나 본 논문에서 제안한 가변 시간 뉴톤-랍손 역수 제곱근 알고리즘에서는 '384x8' 테이블에서 평균 5.91회의 곱셈, '192x7' 테이블과 '96x8' 테이블에서 각각 5.96회와 5.98회의 곱셈으로 역수 제곱근을 계산할 수 있다. 따라서 본 논문에서 제안한 알고리즘에서 평균 6회의 곱셈으로 역수 제곱근을 계산하려면 '96x8' 테이블을 사용하면 되므로, 종래 알고리즘에서 사용하던 '384x8' 테이블과 비교하여 테이블 크기를 사분의 일로 줄일 수 있다.

이러한 결과는 배정도실수 연산에서도 나타난다. 배정도실수 역수 제곱근은 <표 4>로부터 종래 알고리즘에서는 '192X7' 테이블을 사용하면 12회의 곱셈, '384X8' 테이블을 사용하면 9회의 곱셈을 수행하였다. 그러나 본 논문에서 제안한 알고리즘에서는 '192X7' 테이블을 사용하면 평균 9.16회의 곱셈, '384X8' 테이블을 사용하면 평균 8.95회의 곱셈을 수행한다. 또한 '192X8' 테이블을 사용해도 평균 8.96회 곱셈으로 역수 제곱근을 계산할 수 있다. 따라서 평균 9회 곱셈으로 배정도실수 역수 제곱근 계산을 하려면 종래 알고리즘에서는 '384X8' 테이블을 사용했지만, 본 논문에서 제안하는 알고리즘을 사용하면 '192X8' 테이블을 사용하면 가능하다. '192X8' 테이블의 크기는 '384X8' 테이블의 반이다. 따라서 본 논문에서 제안한 알고리즘은 테이블 크기를 반으로 줄이면서 동일한 성능을 보인다.

근사 역수 제곱근 테이블의 길이와 폭의 관계를 <표 3> 및 <표 4>로부터 알 수 있다.

<표 3>의 단정도실수 역수 제곱근 계산에서 '96X6', '96x7', '96x8' 및 '96x9' 테이블은 길이는 96으로 같으며, 폭은 6 비트에서 9 비트까지로 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 9회의 곱셈으로 역수 제곱근을 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 6.92회, 6.10회, 5.98회와 5.97회의 곱셈으로 역수 제곱근을 계산한다. 이들 결과로부터 테이블의 폭을 1-2 비트 증가시키면 성능이 크게 개선되지만 3 비트 이상을 증가시켜도 더 이상 성능이 개선되지 않음을 알 수 있다.

테이블의 길이와 폭의 관계는 배정도실수에서도 동일하게 나타난다. <표 4>의 배정도실수 역수 제곱근에서 '96X6', '96x7', '96x8' 및 '96x9' 테이블은 길이는 96으로 같으며, 폭은 6 비트에서 9 비트까지로 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 12회의 곱셈으로 역수 제곱근을 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 10.39회, 9.48회, 9.13회와 9.09회의 곱셈으로 역수 제곱근을 계산한다.

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

종래의 뉴톤-랍손 역수 제곱근 알고리즘에서는 테이블 크기가 제한적이었다. 단정도실수에서 12회 곱셈으로 역수 제곱근을 계산하기 위해서는 '24X4' 테이블을, 9회 곱셈을 위해서는 '384X8' 테이블을 사용하였다. 중간 크기의 테이블은 의미가 없었다. 즉, '48X5', '96X6', '192X7' 테이블을 사용하면 역수 제곱근 계산에 12회의 곱셈이 필요하였다.

그러나 본 논문에서 제안한 알고리즘은 역수 제곱근 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.

5. 결 론

부동소수점 제곱근 계산은 뺄셈을 반복하는 SRT 알고리즘과 곱셈을 반복하는 뉴톤-랍손(Newton-Raphson) 역수 제곱근 알고리즘 및 골드스미트(Goldschmidt) 제곱근 알고리즘이 있다. 곱셈을 반복하는 방식은 빠른 곱셈기와 근사 테이블을 사용하며, 반복할 때마다 오차가 자승에 비례해서 줄어들므로 연산 속도가 빠른 장점이 있지만, 근사계산만이 가능하다. 그러나 소수의 정밀한 과학 기술 연산 분야를 제외하고 대부분 멀티미디어, 디지털 신호처리, 컴퓨터 그라픽스 등에서는 근사계산만으로도 실용상 문제가 없다.

본 논문에서는 뉴톤-랍손 역수 제곱근 알고리즘의 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 연산을 수행하는 알고리즘을 제안했다.

'F'의 역수 제곱근 계산은 초기값 '$X_0=\frac{1}{\sqrt{F}}\pm e_0$'에 대하여, '$X_{i+1} = \frac{X_i(3-e_r-FX_i^2)}{2}$, $i\in\{0, 1, 2,...n-1\}$'을 반복한다. e_r는 절삭 오차이다. '$X_i=\frac{1}{\sqrt{F}}\pm e_i$'라고 하면 '$X_{i+1}=\frac{1}{\sqrt{F}}-e_{i+1}$, $e_{i+1} < \frac{3\sqrt{F}e_i^2}{2} \mp \frac{Fe_i^3}{2} + 2e_r$'이 된다. '$\left| \frac{3-e_r-FX_i^2}{2} - 1 \right| < 2^{\frac{-p}{2}}$, 이면, '$e_{i+1} < 8e_r$'이 부동소수점으로 표현 가능한 최소값보다 작아지며, '$X_{i+1} \fallingdotseq \frac{1}{\sqrt{F}}$'이다.

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

종래의 뉴톤-랍손 역수 제곱근 알고리즘에서 배정도실수 역수 제곱근 계산을 9회의 곱셈으로 수행하려면 '384X8' 근사 역수 제곱근 테이블을 사용했지만, 본 논문에서 제안한 알고리즘에서는 '192X8' 테이블을 사용해서 평균 8.96회 곱셈으로 역수 제곱근을 계산할 수 있었다. '192X8' 테이블의 크기는 '384X8' 테이블의 반이다. 따라서 본 논문에서 제안한 알고리즘은 테이블 크기를 반으로 줄이면서 동일한 성능을 보인다. 단정도실수 계산에서도 유사한 결과를 보였다.

또한 본 논문의 연구 결과 근사 역수 제곱근 테이블의 폭을 늘리는 것이 길이를 크게 하는 것보다 가격대비 성능에서 유리함을 보였다. 단정도실수 역수 제곱근 계산에서 '192x7' 테이블은 평균 5.96회 곱셈, '96x8' 테이블은 평균 5.98회 곱셈을 수행하였다. '96x8' 테이블이 차지하는 면적은 '192x7' 테이블의 약 반이다. 배정도실수 제곱근에서도 유사한 결과를 얻었다.

본 논문에서 제안한 가변 시간 뉴톤-랍손 역수 제곱근 알고리즘은 평균 곱셈 횟수가 중요한 디지털 신호처리, 컴퓨터 그라픽스, 멀티미디어, 과학 기술 연산 등에서 폭 넓게 사용될 수 있다. 또한 제곱근 계산기 성능에 따라 최적의 근사 역수 제곱근 ($X_0 \fallingdotseq \frac{1}{\sqrt{F}}$) 테이블을 구성할 수 있으므로 하드웨어가 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.

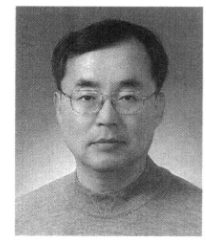
[10] IEEE, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard, Std. 754-1985.

[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.

김 성 기

e-mail : heewookim@empal.com
1977년 울산대학교 응용물리학과(학사)
1994년 부경대학교 산업대학원 전자계산학과(학사)
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