가변 시간 골드스미트 부동소수점 제곱근 계산기

김성기* · 송홍복** · 조경연*

A Variable Latency Goldschmidt's Floating Point Number Square Root Computation

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

요 약

부동소수점 제곱근 계산에 많이 사용하는 골드스미트 제곱근 알고리즘은 곱셈을 반복하여 제곱근을 계산한다. 본 논문에서는 골드스미트 제곱근 알고리즘의 반복 과정의 오차를 예측하여 오차가 정해진 값보다 작아지는 시점까지 반복 연산하는 알고리즘을 제안한다.

'F'의 제곱근 계산은 초기값 '$X_0=Y_0=T^2\times F,\ T=\frac{1}{\sqrt{F}}+e_r$'에 대하여, '$R_i=\frac{3-e_r-X_i}{2},\ X_{i+1}=X_i\times R_i^2,\ Y_{i+1}=Y_i\times R_i,\ i\in\{0,1,2,\cdots,n-1\}$'을 반복한다. 곱셈 결과는 소수점 이하 p 비트 미만을 절삭하며, 절삭 오차는 '$e_r\approx 2^{-p}$' 보다 작다. p는 단정도실수에서 28, 배정도실수에서 58이다. '$X_i=1\pm e_i$'이면 '$X_{i+1}=1-e_{i+1},\ e_{i+1}<\frac{3e_i^2}{4}\mp\frac{e_i^3}{4}+4e_r$'이다. '$|X_i-1|<2^{\frac{-p+2}{2}}$'이면, '$e_{i+1}<8e_r$'이 부동소수점으로 표현할 수 있는 최소값보다 작게 되며, '$\sqrt{F}\fallingdotseq\frac{Y_{i+1}}{T}$'이다.

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

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

ABSTRACT

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

* 부경대학교 전자컴퓨터정보통신공학부 **동의대학교 전자 · 정보통신공학부

접수일자 : 2004. 9. 15

To find the square root of a floating point number F, the algorithm repeats the following operations: '$R_i=\frac{3-e_r-X_i}{2}$, $X_{i+1}=X_i\times R_i^2$, $Y_{i+1}=Y_i\times R_i$, $i\in\{0,1,2,\cdots,n-1\}$' with the initial value is '$X_0=Y_0=T^2\times F$, $T=\frac{1}{\sqrt{F}}+e_t$'. 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 '$X_i=1\pm e_i$', there is '$X_{i+1}=1-e_{i+1}$', where '$e_{i+1}<\frac{3e_i^2}{4}\mp\frac{e_i^3}{4}+4e_r$'. If '$|X_i-1|<2^{\frac{-p\pm2}{2}}$' is true, '$e_{i+1}<8e_r$' is less than the smallest number which is representable by floating point number. So, $\sqrt{F}$ is approximate to '$\frac{Y_{i+1}}{T}$'.

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 ($T=\frac{1}{\sqrt{F}}+e_t$) 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 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.

키워드

부동소수점, 제곱근 계산기, Goldschmidt, Floating Point, Square Root Computation

I. 서 론

디지털 신호처리, 컴퓨터 그라픽스, 멀티미디어, 과학 기술 연산 등에서 부동소수점 연산이 많이 사용되고 있다. 특히 최근에 멀티미디어 분야에서 음성과 3차원 그라픽 처리 등에 부동소수점 연산이 사용되면서 SOC(System On Chip)에서도 하드웨어 부동소수점 계산기를 도입하고 있다. 부동소수점 연산에 대한 연구는 출현 빈도가 높은 덧셈(뺄셈)과 곱셈에 집중되고 있다. 이런 결과로 덧셈은 2-4 사이클, 곱셈은 2-5 사이클만에 배정도실수 연산을 할 수 있다. 이에 반하여 출현 빈도가 낮은 나눗셈은 배정도실수 연산에 9 사이클에서 60 사이클 이상, 제곱근은 15 사이클에서 70 사이클 이상 소요되고 있다[1]. 나눗셈과 제곱근은 덧셈(뺄셈) 및 곱셈보다 출현 빈도가 낮지만, Oberman과 Flynn의 연구[2]는 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[5-8]. SRT 알고리즘은 덧셈(뺄셈), 부분 곱셈, 몫 결정의 3 단계 과정을 반복하는 알고리즘으로 매 반복마다 일정 길이 비트의 몫을 얻을 수 있다. 이러한 SRT 알고리즘은 하드웨어가 간단하고, 정밀 계산이 가능하지만, 연산 속도가 느린 단점을 가진다. 반면에 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.

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

저하시키는 단점이 있었다.

본 논문에서는 골드스미트 제곱근 알고리즘의 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복하는 알고리즘을 제안한다. 골드스미트 제곱근 알고리즘은 $\sqrt{F}$를 계산하기 위하여, 초기값을 '$X_0=Y_0=F$'로 설정한다.

'$R_i=\frac{3-e_r-X_i}{2}$, $X_{i+1}=X_i\times R_i^2$, $Y_{i+1}=Y_i\times R_i$, $i\in\{0,1,2,\cdots,n-1\}$'를 반복하면 X_n은 '1'에 수렴하며, '$Y_n=\sqrt{F}$'이다. '$X_i=1\pm e_i$'이며 오차 e_i로부터 오차 e_{i+1}를 예측하는 알고리즘을 도출하고, 오차 e_{i+1}가 부동소수점으로 표현할 수 있는 최소값보다 작게 되면 반복을 종료한다.

본 논문에서 제안한 가변 시간 골드스미트 제곱근 알고리즘에 의한 제곱근 계산기를 Verilog HDL로 설계하고, 시뮬레이션해서 정상적으로 동작하는 것을 확인하였다.

본 논문에서 제안한 알고리즘은 입력하는 부동소수점 값에 따라서 반복 곱셈 횟수가 다르므로, 평균 곱셈 횟수를 계산하는 방식을 도출하고, 여러 크기의 근사 역수 제곱근 테이블에서 단정도실수 및 배정도실수의 제곱근 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 골드스미트 제곱근 알고리즘을 분석해서 연산 자릿수 및 반복 연산을 종료할 오차 한계를 계산한다. 또한 오차 예측 알고리즘을 유도한다. 3장에서는 제안한 알고리즘을 구현하는 회로와 상태 기계를 구성한다. 4장에서는 근사 테이블을 구성하고, 제곱근 계산에 소요되는 평균 곱셈 횟수를 계산한다. 그리고 그 결과를 종래 골드스미트 제곱근 알고리즘과 비교 분석한다. 5장에서 결론을 맺는다.

II. 가변 시간 제곱근 알고리즘

2.1. 골드스미트 제곱근 알고리즘

$\sqrt{F}$를 계산하는 골드스미트 제곱근 알고리즘의 반복식은 식 (1)과 같다.

$$X_0=Y_0=F \qquad (1)$$

For $i\in\{0,1,2,\cdots,n-1\}$

$$R_i=\frac{3-X_i}{2}$$
$$Y_{i+1}=Y_i\times R_i$$
$$X_{i+1}=X_i\times R_i^2$$

반복 연산을 수행하면 X_n은 1.0에 수렴하므로 다음 식이 성립한다.

$$\frac{Y_n^2}{X_n}=\frac{F^2\times R_0^2\times R_1^2\times\cdots\times R_n^2}{F\times R_0^2\times R_1^2\times\cdots\times R_n^2}=F=Y_n^2$$

따라서 $Y_n=\sqrt{F}$이다.

IEEE-754[10]로 규정되는 부동소수점은 $1.f_2\times 2^{n+base}$ 이다. 가수부 $1.f$는 단정도실수에서 24 비트, 배정도실수에서는 53 비트이다. 지수부 n이 홀수이면 $0.1f_2\times 2^{n+1+base}$로 변환한다. 따라서 본 논문에서 부동소수점 가수부는 '$0.5 \le i.f < 2.0$'이다. 본 논문에서는 제곱근의 지수부 연산은 생략한다.

부동소수점 수의 가수부 '$F=i.f$'는 식 (2)와 같이 두 부분으로 나눌 수 있다.

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

식 (2)에서 g와 h의 길이를 각각 n_g 및 n_h 비트로 정의한다. h는 '$0\le h\le 2^{-n_g}$'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 근사계산한 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. $T(g)$의 자승을 F에 곱해주면 식 (3)과 같이 된다.

$$\begin{aligned}F_0&=T(g)^2\times F\\&=1+\frac{h}{i.g}+2\left(\sqrt{i.g}+\frac{h}{\sqrt{i.g}}\right)e_t+(i.g+h)e_t^2\\&=1+e_0\end{aligned} \qquad (3)$$

식 (3)에서 F는 '1.0'에 근사하므로 식 (1)의 수렴 속도를 빠르게 할 수 있다. 식 (1)에 의하여 '$\sqrt{F_0}=T(g)\sqrt{F}$'를 계산하고 그 결과를 $T(g)$로 나누면 $\sqrt{F}$가 구해진다.

식 (1)에서 중간 곱셈 결과는 소수점 이하 p 비트 미만을 절삭한다. p를 연산 유효자릿수라고 가칭한다. 또한 식 (1)에서 '$3-X_i$'는 하드웨어 구현 시에 캐리 전달 지연 시간이 필요하다. 이러한 문제점을 해결하기 위해서 본 논문에서는 근사계산인 '$3-e_r-X_i$, $e_r=2^{-p}$'을 계산한다. 본 논문에서 사용하는 골드스미트 제곱근 알고리즘을 식 (4)에 보인다.

$$X_0=Y_0=T(g)^2\times F \tag{4}$$

For $i\in\{0,1,2,\cdots,n-1\}$

$$R_i=\frac{3-e_r-X_i}{2}$$
$$Y_{i+1}=Y_i\times R_i$$
$$X_{i+1}=X_i\times R_i^2$$
$$\sqrt{F}=\frac{Y_n}{T(g)}$$

2.2. 오차 분석

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

식 (4)에서 Y_i 반복 계산에서 절삭 오차가 누적된다. Y_3까지의 최대 누적 오차를 구하면 식 (5)가 된다. $[X]$는 X의 오차가 없는 값을 나타내며, 절삭 오차는 '$e_r=2^{-p}$'보다 작다. $T(g)$는 근사 값으로 유효자릿수가 $\frac{p}{2}$보다 작다. 그러므로 $T(g)^2$은 절삭 오차가 없다.

$$Y_0=T(g)^2\times F-e_r=[Y_0]-e_r \tag{5}$$

$$R_0=\frac{3-e_r-X_0}{2}=[R_0]-e_r$$

$$Y_1=Y_0\times R_0=([Y_0-e_r])\times([R_0-e_r])-e_r$$
$$=[Y_1]-3e_r$$

$$\because\ Y_0,\ R_0\fallingdotseq 1$$

$$Y_2=Y_1\times R_1=([Y_1-3e_r])\times([R_1-e_r])-e_r$$
$$=[Y_2]-5e_r$$

$$Y_3=[Y_3]-7e_r$$

한편 '$X_i=1+e_i$'이라고 하면 X_{i+1}은 식 (6)과 같이 된다. $u\times e_r$과 $v\times e_r$은 곱셈 결과를 절삭하면서 발생한 오차이며, '$0\le u, v<1$' 이다.

$$X_{i+1}=(1+e_i)\times\left(\left(\frac{3-e_r-1-e_i}{2}\right)^2-u\times e_r\right)-v\times e_r \tag{6}$$
$$=1-\frac{3e_i^2}{4}+\frac{e_i^3}{4}-\left(1+u+v+\frac{e_i}{2}-\frac{e_r}{4}-\frac{e_i^2}{2}-\frac{e_re_i}{4}+ue_i\right)e_r$$
$$=1-e_{i+1}$$

식 (6)가 최대 오차를 가지기 위해서 '$u=v=1$'이 되어야 하며, 식 (7)이 성립한다.

$$e_{i+1}<\frac{3e_i^2}{4}-\frac{e_i^3}{4}+4e_r \tag{7}$$

'$X_i=1-e_i$'이라고 하면 식 (6)과 식 (7)은 다음과 같다.

$$X_{i+1}=1-\frac{3e_i^2}{4}-\frac{e_i^3}{4}-\left(1+u+v-\frac{e_i}{2}-\frac{e_r}{4}-\frac{e_i^2}{2}+\frac{e_re_i}{4}-ue_i\right)e_r \tag{6'}$$
$$=1-e_{i+1}$$

$$e_{i+1}<\frac{3e_i^2}{4}+\frac{e_i^3}{4}+3e_r \tag{7'}$$

식 (7)로부터 식 (4)의 반복 연산 중에 절삭으로 발생하는 X_i의 최대 오차는 $4e_r$보다 작다. 한편 식 (5)에서 Y_i의 최대 누적 오차가 $7e_r$이므로 e_{i+1}의 최소값을 $8e_r$로 상정하면, '$\frac{3e_i^2}{4}-\frac{e_i^3}{4}$'의 최소값은 $4e_r$이 되고, '$\frac{3e_i^2}{4}+\frac{e_i^3}{4}$'의 최소값은 $5e_r$이다. '$e_{i+1}<8e_r$'이면 오차가 충분히 작으므로 식 (4) 알고리즘의 반복을 종료한다.

식 (7)과 식 (7')로부터 '$e_{i+1}<8e_r$'이면 식 (8)이 성립한다.

$$3e_i^2 - e_i^3 < 16e_r,\ 3e_i^2 + e_i^3 < 20e_r \quad (8)$$

식 (8)을 만족시키면 '$e_i^3 << e_r$'이 된다. 따라서 식 (9)가 성립하면 식 (4) 알고리즘의 반복을 종료하고, '$Y_{i+1} \fallingdotseq T(g) \times \sqrt{F}$'이다.

$$3e_i^2 < 16e_r$$

$$|X_i - 1| = e_i < \sqrt{4e_r} = 2^{\frac{-p+2}{2}} \quad (9)$$

2.3. 연산 유효자릿수

'$X_{i+1} = 1 - e_{i+1}$'이므로 '$e_{i+1} < 8e_r$'은 부동소수점에서 표현 가능한 최소값보다 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 '$8e_r = 8 \times 2^{-p} < 2^{-25}$'이 되어야 한다. 따라서 IEEE-754 단정도실수 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 '$8e_r = 8 \times 2^{-p} < 2^{-54}$'가 되어야 한다. 따라서 IEEE-754 배정도실수 제곱근 계산에 필요한 연산 유효자릿수는 '$p = 57$'이다.

2.4. 가변 시간 골드스미트 제곱근 알고리즘

식 (9)로부터 '$|X_i - 1| = e_i < 2^{\frac{-p+2}{2}}$'까지 연산하면 '$T(g) \times \sqrt{F}$'를 구할 수 있다. e_i이 $2^{\frac{-p+2}{2}}$보다 작은 것을 판별하는 회로는 X_i의 소수점 이하 $\frac{p-2}{2}$ 비트가 모두 '0' 또는 모두 '1'인가를 판단하는 회로이다. $\frac{p-2}{2}$ 비트를 판단하므로 p는 반드시 짝수가 되어야 한다. 그러므로 배정도실수에서 필요한 연산 자릿수는 '$p = 58$'이 된다.

본 논문에서 제안하는 가변 시간 골드스미트 제곱근 알고리즘을 정리하면 표 1과 같다.

표 1. 가변 시간 골드스미트 제곱근 알고리즘
Table 1. Variable latency Goldschmidt's square root algorithm

To compute $\sqrt{F}$, 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) $R = T(g) \times T(g)$;
2) $X = Y = R \times F$;
   $R = \frac{3 - 2^{-p} - X}{2}$;
4) $Y = Y \times R$;
   If $|X - 1| < 2^{\frac{-p+2}{2}}$, *then goto* 7;
5) $R = R \times R$;
6) $X = X \times R$;
   $R = \frac{3 - 2^{-p} - X}{2}$;
   goto 4;
7) $Result = \frac{Y}{T(g)}$;

III. 가변 시간 제곱근 계산기

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

그림 1 블록도와 표 2의 상태 흐름도는 제안한 가변 시간 골드스미스 제곱근 알고리즘을 하드웨어로 구현한 것이다. 상태-1에서 $\frac{1}{\sqrt{F}}$의 근사 값을 테이블로부터 읽어서 $T(g)$ 레지스타에 저장한다. 상태-2에서 $T(g)^2$을 계산해서 R 레지스타에 임시 저장한다. 상태-3에서 '$X_0 = Y_0 = T(g)^2 \times F$'를 계산해서 X 레지스타와 Y 레지스타에 저장한다. 그리고 '$R_i = \frac{3 - e_r - X_i}{2}$'를 계산한다. '$3 - e_r - X_i = 10.111\,1\cdots1_2 - X_i = Z$'이다.

'$X_i \geq 1.0$'이면 'Z = 1.xxxxx$_2$'이고, '$X_i < 1.0$'이면 'Z = 10.xxxxx$_2$'이다. Z의 소수점이하 '0.xxxxx$_2$'는 X_i의 소수점이하의 1의 보수이다. 나누기 2는 오른쪽으로 한 비트 이동하는 것이다. 계산한 R_i를 R 레지스타에 저장한다. 상태-4에서 '$Y_{i+1} = Y_i \times R_i$'를 계산해서 Y 레지스타에 저장한다. 그리고 식 (9)의 '$|X_i - 1| = e_i < 2^{\frac{-p+2}{2}}$' 조건식을 판단한다. 이것은 X_i의 소수점 이하 $\frac{p-2}{2}$ 비트가 모두 '0'이거나 모두 '1'인가를 판단하는 것이다.

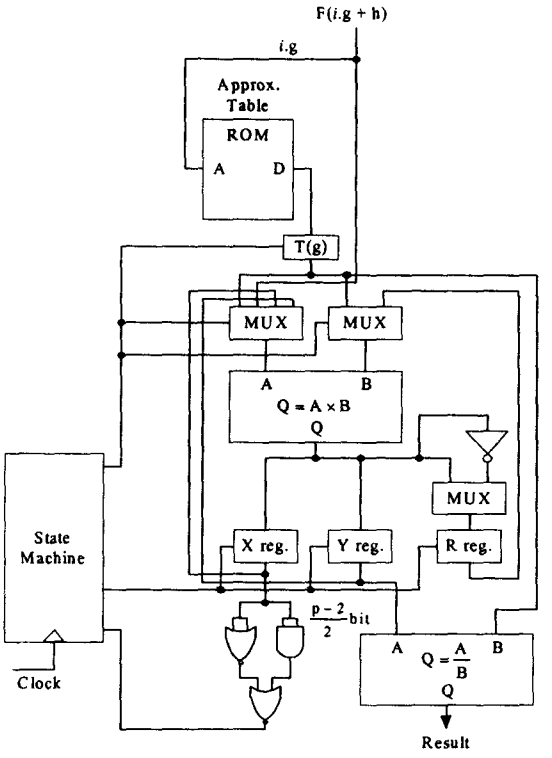


그림 4. 하나의 곱셈기를 사용한 가변 시간 골드스미트 제곱근 계산기
Fig 1. Variable latency Goldschmidt's square root unit with single multiplier

표 2. 하나의 곱셈기를 사용한 가변 시간 골드스미트 제곱근 계산기 상태 흐름도
Table 2. State flow of variable latency Goldschmidt's square root unit with single multiplier

```
/*  Compute √F  */
(State-1)
  Get approximate reciprocal table T(g) ;
(State-2)
  Multiplier in port A = T(g) ;
  Multiplier in port B = T(g) ;
  Multiplier out port Q = R ;
(state-3)
  Multiplier in port A = F ;
  Multiplier in port B = R ;
  Multiplier out port Q = X, Y ;
  If X ≥ 1.0 then Z = 1 ;
               else Z = 2 ;
  Z = Z + (1's complement of X's fraction part) ;
  R = Z >> 1 ;
(state-4)
  Multiplier in port A = Y ;
  Multiplier in port B = R ;
  Multiplier out port Q = Y ;
  If msb (p-2)/2 bit of X is all '0' or all '1',
       goto state-7 ;
(state-5)
  Multiplier in port A = X ;
  Multiplier in port B = R ;
  Multiplier out port Q = X ;
(state-6)
  Multiplier in port A = X ;
  Multiplier in port B = R ;
  Multiplier out port Q = X ;
  If X ≥ 1.0 then Z = 1 ;
               else Z = 2 ;
  Z = Z + (1's complement of X's fraction part) ;
  R = Z >> 1 ;
  Goto state-4 ;
(state-7)
  Divider in port N = Y ;
  Divider in port D = T(g) ;
  Divider out port Q = Result ;
```

판단 회로는 $\frac{p-2}{2}$개의 입력을 가지는 NOR 게이트와 $\frac{p-2}{2}$개의 입력을 가지는 AND 게이트로 간단하게 구현할 수 있다. 조건식을 만족하면 상태-7로 간다. 조건이 맞지 않으면 상태-6과 상태-7에서 '$X_{i+1}=X_i\times R_i^2$'을 계산해서 X 레지스타에 저장하고, '$R_{i+1}=\frac{3-e_r-X_{i+1}}{2}$'를 계산해서 R 레지스타에 저장한다. 그리고 상태-4로 가서 반복한다. 상태-7에서는 '$\frac{Y_n}{T(g)}=\sqrt{F}$'를 계산해서 제곱근 계산을 완료한다.

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

2개의 곱셈기를 병렬로 사용하는 가변 시간 골드스미트 제곱근기의 상태 흐름도는 표 2에서 상태-4와 상태-5를 하나의 상태로 묶어서 구현할 수 있다. 표 3에 두개의 곱셈기를 사용한 가변 시간 골드스미트 곱근기의 상태 흐름도를 보인다.

설계한 가변 시간 골드스미트 제곱근 계산기는 Verilog HDL로 작성하고, 시뮬레이션해서 동작을 검증하였다.

표 3. 두 개 곱셈기를 사용한 가변 시간 골드스미트 제곱근 계산기 상태 흐름도

Table 3. State flow of variable latency Goldschmidt's square root unit with dual multiplier

```
/*  Compute √F  */
(State-1)
  Get approximate reciprocal table T(g) ;
(State-2)
  Multiplier 1 in port A = T(g) ;
  Multiplier 1 in port B = T(g) ;
  Multiplier 1 out port Q = R ;
(state-3)
  Multiplier 1 in port A = F ;
  Multiplier 1 in port B = R ;
  Multiplier 1 out port Q = X, Y ;
  If X ≥ 1.0 then Z = 1 ;
             else Z = 2 ;
  Z = Z + (1's complement of X's fraction
part) ;
  R = Z >> 1 ;
(state-4)
  Multiplier 1 in port A = Y ;
  Multiplier 1 in port B = R ;
  Multiplier 1 out port Q = Y ;
  If msb (p-2)/2 bit of X is all '0' or all '1',
       goto state-6 ;

  Multiplier 2 in port A = X ;
  Multiplier 2 in port B = R ;
  Multiplier 2 out port Q = X ;
(state-5)
  Multiplier 1 in port A = X ;
  Multiplier 1 in port B = R ;
  Multiplier 1 out port Q = X ;
  If X ≥ 1.0 then Z = 1 ;
             else Z = 2 ;
  Z = Z + (1's complement of X's fraction
part) ;
  R = Z >> 1 ;
  Goto state-4 ;
(state-6)
  Divider in port N = Y ;
  Divider in port D = T(g) ;
  Divider out port Q = Result ;
```

IV. 연구 결과 및 분석

식 (3)으로부터 X_0의 오차 e_0는 식 (10)이 된다.

$$e_0=T(g)^2\times(i.g+h)-1 \tag{10}$$

식 (10)으로부터 e_0는 h=0에서 '$T(g)^2\times i.g-1$'이 되며, $0 \le h \le h_{max}$에서 단조 증가함수이다. 식 (10)으로부터 h는 식 (11)이 된다.

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

h에 따른 e_0의 변화를 그림 2에 보인다.

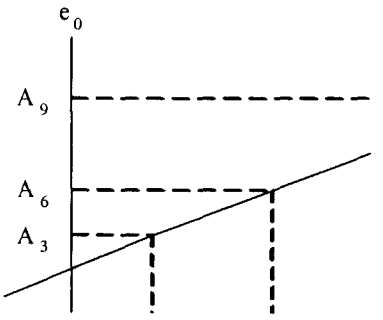


그림 5. h와 e_0의 그래프

Fig 2. e_0 versus h

식 (9)와 표 1의 알고리즘으로부터 '$e_0 < A_3 = 2^{\frac{-p+2}{2}}$'이면 3회의 곱셈으로 제곱근 연산이 완료된다. 그림 2에서 h_3는 A_3에서의 h 값이다. 그러므로 '$0 \le h \le h_3$' 구간에서는 3회의 곱셈으로 제곱근이 계산된다.

식 (7)로부터 e_1은 식 (12)가 된다.

$$e_1 \doteqdot \frac{3e_0^2}{4} - \frac{e_0^3}{4} + 4e_r \qquad (12)$$

식 (12)로부터 '$e_1 < A_3$'이면 6회의 곱셈으로 제곱근을 계산할 수 있다. 식 (12)에서 '$e_1 = A_3$'이 되는 수치해 '$e_0 = A_6$'를 뉴톤-랍손 알고리즘으로 구할 수 있다. 그림 2에서 h_6는 A_6에서의 h 값이다. 그러므로 '$h_3 < h \le h_6$' 구간에서는 6회의 곱셈으로 제곱근을 계산한다.

식 (7)로부터 e_2는 식 (13)이 되며, 식 (13)에서 '$e_2 = A_6$'가 되는 수치해 '$e_0 = A_9$'를 뉴톤-랍손 알고리즘으로 구할 수 있다.

$$e_2 \doteqdot \frac{3e_1^2}{4} - \frac{e_1^3}{4} + 4e_r \qquad (13)$$

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

$$T(g) = \frac{1}{\sqrt{i.g}} \doteqdot RN\left(\frac{1}{\sqrt{i.g+2^{-n_t-1}}}\right) \qquad (15)$$

RN is round to nearest

$T(g)$의 소수점 이하 길이를 t 비트라고 하면 '$T(g) = (b_0.b_1b_2\cdots b_t)_2$, $\frac{1}{\sqrt{2}} < T(g) < \sqrt{2}$'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. 따라서 근사 역수 테이블의 크기는 '$2^{n_t} \times t$ 비트가 되어서, 테이블의 길이는 2^{n_t}이며, 폭은 t 비트이다.

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

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

Table 4. Number of multiply of IEEE single precision square root
(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	6 mul	9 mul	12 mul
24×4	8.44	6.63	0.18%	18.4%	81.5%	0.0%
48×5	7.85	6.23	0.37%	37.8%	61.9%	0.0%
96×6	6.91	5.61	0.71%	68.3%	31.0%	0.0%
192×7	5.96	4.97	1.44%	98.6%	0.01%	0.0%
384×8	5.91	4.94	2.87%	97.1%	0.0%	0.0%
96×7	6.09	5.06	1.25%	94.4%	4.33%	0.0%
96×8	5.97	4.98	1.80%	97.2%	0.97%	0.0%
96×9	5.96	4.98	1.95%	97.3%	0.74%	0.0%

표 5. EEE 배정도실수 제곱근 계산에 필요한 곱셈 횟수
(1) -- 한 개의 곱셈기를 사용한 경우
(2) -- 두 개의 곱셈기를 사용한 경우
Table 5. Number of multiply of IEEE double precision square root
(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	6 mul	9 mul	12 mul
96×6	10.38	7.92	0.0%	0.41%	53.3%	46.3%
192×7	9.15	7.10	0.0%	0.83%	93.2%	5.97%
384×8	8.95	6.97	0.0%	1.66%	98.3%	0.0%
768×9	8.90	6.93	0.0%	3.32%	96.7%	0.0%
192×8	8.96	6.97	0.0%	1.44%	98.5%	0.02%
192×9	8.94	6.96	0.0%	2.10%	87.9%	0.0%
96×7	9.47	7.32	0.0%	0.72%	82.8%	16.5%
96×8	9.13	7.08	0.0%	1.04%	93.7%	5.21%
96×9	9.09	7.06	0.0%	1.13%	94.8%	4.06%
96×10	9.08	7.06	0.0%	1.13%	94.9%	3.94%

종래 골드스미트 제곱근 알고리즘에서는 최대 오차를 고려해서 반복 횟수를 정했다. 즉, 종래 알고리즘에 의한 단정도실수 제곱근은 '192×7' 테이블을 사용하면 9회의 곱셈을 수행하였고, '384×8' 테이블을 사용하면 6회의 곱셈을 수행하였음을 표 4로부터 알 수 있다. 또한 '96×8' 테이블을 사용해도 9회의 곱셈을 수행하였었다.

그러나 본 논문에서 제안한 가변 시간 골드스미트 제곱근 알고리즘에서는 '384×8' 테이블에서 평균 5.91회의 곱셈, '192×7' 테이블과 '96×8' 테이블에서 모두 평균 5.96 회의 곱셈으로 제곱근을 계산할 수 있다. 즉, 본 논문에서 제안한 알고리즘에서 평균 6회의 곱셈으로 제곱근을 계산하려면 '96×8' 테이블을 사용하면 되므로, 종래 알고리즘에서 사용하던 '384×8' 테이블과 비교하여 테이블 크기를 사분의 일로 줄일 수 있다.

이러한 결과는 배정도실수 연산에서도 나타난다. 배정도실수 제곱근은 표 5로부터 종래 알고리즘에서는 '192×7' 테이블을 사용하면 12회의 곱셈, '384×8' 테이블을 사용하면 9회의 곱셈을 수행하였다. 그러나 본 논문에서 제안한 알고리즘에서는 '192×7' 테이블을 사용하면 평균 9.15회의 곱셈, '384×8' 테이블을 사용하면 평균 8.95회의 곱셈을 수행한다. 또한 '192×8' 테이블을 사용해도 평균 8.96회 곱셈으로 제곱근을 계산할 수 있다. 따라서 평균 9회 곱셈으로 배정도실수 제곱근 계산을 하려면 종래 알고리즘에서는 '384×8' 테이블을 사용했지만, 본 논문에서 제안하는 알고리즘을 사용하면 '192×8' 테이블을 사용하면 가능하다. '192×8' 테이블의 크기는 '384×8' 테이블의 반이다. 따라서 본 논문에서 제안한 알고리즘은 테이블 크기를 반으로 줄이면서 동일한 성능을 보인다.

근사 역수 제곱근 테이블의 길이와 폭의 관계를 표 4 및 표 5로부터 알 수 있다.

표 4의 단정도실수 제곱근에서 '96×6', '96×7', '96×8' 및 '96×9' 테이블은 길이는 96으로 같으며, 폭은 6 비트에서 9 비트까지로 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 9회의 곱셈으로 제곱근을 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 6.91회, 6.09회, 5.97회와 5.96회의 곱셈으로 제곱근을 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 5의 배정도실수 제곱근에서 '96×6', '96×7', '96×8', '96×9' 및 '96×10' 테이블은 길이는 96으로 같으며, 폭은 6 비트에서 10 비트까지로 다르다. 종래 알고리즘에서는 이들 모든 테이블에서 12회의 곱셈으로 제곱근을 계산하였지만, 본 논문에서 제안하는 알고리즘에서는 각각 10.38회, 9.47회, 9.13회, 9.09회와 9.08회의 곱셈으로 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 골드스미트 제곱근 알고리즘에서는 테이블 크기가 제한적이었다. 단정도실수에서 12회 곱셈으로 제곱근을 계산하기 위해서는 '24×4' 테이블을, 9회 곱셈을 위해서는 '384×8' 테이블을 사용하였다. 중간 크기의 테이블은 의미가 없었다. 즉, '48×5', '96×6', '192×7' 테이블을 사용하면

제곱근 계산에 12회의 곱셈이 필요하였다.

그러나 본 논문에서 제안한 알고리즘은 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.

V. 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.

본 논문에서는 골드스미트 제곱근 알고리즘의 반복 연산 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 연산을 수행하는 알고리즘을 제안했다.

'F'의 제곱근 계산은 초기값 '$X_0=Y_0=T^2\times F$, $T=\frac{1}{\sqrt{F}}+e_t$'에 대하여, '$R_i=\frac{3-e_r-X_i}{2}$, $X_{i+1}=X_i\times R_i^2$, $Y_{i+1}=Y_i\times R_i$, $i\in\{0,1,2,\cdots,n-1\}$'을 반복한다. e_r는 절삭 오차이다. '$X_i=1\pm e_i$'이면 '$X_{i+1}=1-e_{i+1}$, $e_{i+1}<\frac{3e_i^2}{4}\mp\frac{e_i^3}{4}+4e_r$'이다. '$e_{i+1}=\square X_i-1\square 2^{\frac{-p+2}{2}}$'이면, '$e_{i+1}<8e_r$'이 부동소수점으로 표현할 수 있는 최소값보다 작게 되며, '$\sqrt{F}\fallingdotseq\frac{Y_{i+1}}{T}$'이다.

본 논문에서 제안한 가변 시간 골드스미트 제곱근 알고리즘에 의한 제곱근 계산기를 Verilog HDL로 설계하고, 시뮬레이션해서 정상적으로 동작하는 것을 확인하였다.

종래의 골드스미트 제곱근 알고리즘에서 단정도 실수 제곱근 계산을 6회의 곱셈으로 수행하려면 '384×8' 근사 역수 제곱근 테이블을 사용했지만, 본 논문에서 제안한 가변 시간 골드스미트 제곱근 알고리즘에서는 '96×8' 테이블을 사용해서 평균 5.06회에 제곱근 계산을 할 수 있었다. 따라서 종래 알고리즘에서 사용하던 테이블 크기를 사분의 일로 줄일 수 있다. 배정도실수 제곱근에서도 유사한 결과를 얻었다.

또한 본 논문의 연구 결과 근사 역수 제곱근 테이블의 폭을 늘리는 것이 길이를 크게 하는 것보다 가격대비 성능에서 유리함을 보였다. 단정도실수 제곱근 계산에서 '192×7' 테이블은 평균 5.96회 곱셈, '96×8' 테이블은 평균 5.97회 곱셈을 수행하였다. '96×8' 테이블이 차지하는 면적은 '192×7' 테이블의 약 반이다. 배정도실수 제곱근에서도 유사한 결과를 얻었다.

본 논문에서 제안한 가변 시간 골드스미트 제곱근 알고리즘은 평균 곱셈 횟수가 중요한 디지털 신호처리, 컴퓨터 그라픽스, 멀티미디어, 과학 기술 연산 등에서 폭 넓게 사용될 수 있다. 또한 제곱근 성능에 따른 최적의 근사 역수 제곱근 ($T=\frac{1}{\sqrt{F}}+e_t$) 테이블을 구성할 수 있으므로 하드웨어가 제한적인 SOC(System On Chip)에 유용하게 적용될 수 있다.

참고문헌

[1] Peter Soderquist and Miriam Leaser, "Division And Square Root, Choosing the Right Implementation," IEEE Micro, pp. 56-66, Jul. 1997

[2] 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

[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

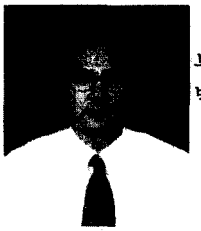
[8] D. L. Fowler and J. E. Smith, "An Accurate, High Speed Implementation of Division by Reciprocal Approximation," Proc. 9th IEEE symp. Computer Arithmetic, IEEE, pp. 60-67, Sep. 1989

[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

저자 소개



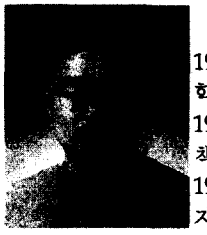
김성기(Sung-Gi Kim)

부경대학교 대학원 컴퓨터공학과 박사과정

※관심분야 : 컴퓨터 구조, 암호 알고리즘, 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 정보공학전공 (공학박사)
1983-1991 삼보컴퓨터 기술연구소 책임연구원
1991-현재 부경대학교 공과대학 전자컴퓨터정보통신공학부 교수
1991-2001 삼보컴퓨터 기술연구소 비상임기술고문
1998-현재 에이디칩스 사외이사 겸 비상임기술고문
※관심분야 : 전산기구조, 반도체회로설계, 암호 알고리즘