JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 19, No. 10 : 2341~2349 Oct. 2015

오차 교정 K차 골드스미트 부동소수점 나눗셈

조경연*

Error Corrected K'th order Goldschmidt's Floating Point Number Division

Gyeong-Yeon Cho*

Department of IT Convergence and Application Engineering, Pukyong University, Pusan 48513. Korea

요약

부동소수점 나눗셈에서 많이 사용하는 골드스미트 부동소수점 나눗셈 알고리즘은 한 회 반복에 두 번의 곱셈을 수행한다. 본 논문에서는 한 회 반복에 K 번 곱셈을 수행하는 가칭 오차 교정 K차 골드스미트 부동소수점 나눗셈 알고리즘을 제안한다. 본 논문에서 제안한 알고리즘은 입력 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. 또한 한 번의 곱셈과 판정으로 나눗셈 결과를 보정하는 알고리즘을 제안한다. 본 논문에서 제안한 알고리즘은 오차가 일정한 값보다 작아질 때까지만 반복 연산을 수행하므로 나눗셈 계산기의 성능을 높일 수 있다. 또한 최적의 근사 테이블을 구성할 수 있다.

ABSTRACT

The commonly used Goldschmidt's floating-point divider algorithm performs two multiplications in one iteration. In this paper, a tentative error corrected K'th Goldschmidt's floating-point number divider algorithm which performs K times multiplications in one iteration is proposed. Since the number of multiplications performed by the proposed algorithm is dependent on the input values, the average number of multiplications per an operation in single precision and double precision divider is derived from many reciprocal tables with varying sizes. In addition, an error correction algorithm, which consists of one multiplication and a decision, to get exact result in divider is proposed. 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 unit. Also, it can be used to construct optimized approximate reciprocal tables.

키워드 : 부동소수점 나눗셈, K차 골드스미트, 오차 교정, 가변시간

Key Words : Floating point divider, K'th order Goldschmidt, Error correction Variable latency

Received 11 March 2015, **Revised** 01 April 2015, **Accepted** 16 April 2015

*** Corresponding Author** Gyeong-Yeon Cho(E-mail:gycho@pknu.ac.kr, Tel:+82-51-629-6252)
Department of IT Convergence and Application Engineering, Pukyong University, Pusan 48153, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2015.19.10.2341 print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright Ⓒ The Korea Institute of Information and Communication Engineering.

Ⅰ. 서 론

부동소수점 계산은 과학 및 공학 기술 분야에서 많이 사용된다. 최근에는 음성 처리, 3차원 그래픽, 자동차 등 실장제어 분야에도 폭넓게 사용되면서 CPU의 기본 기능으로 채택되고 있다[1]. 실장제어 분야에서 고성능의 부동소수점 연산이 요구되면서 실장제어용 마이크로프로세서도 기본적으로 부동소수점 연산기능을 갖추게 되었다[2].

부동소수점 나눗셈은 덧셈, 뺄셈 및 곱셈보다 출현 빈도가 낮지만, Oberman과 Flynn의 연구[3]는 나눗셈의 수행 시간이 덧셈이나 곱셈과 비슷하게 소요됨을 보이고 있다.

부동소수점 나눗셈은 뺄셈을 반복하는 SRT[4] 알고리즘과 곱셈을 이용한 알고리즘으로 뉴톤-랍손(Newton-Raphson) 역수 알고리즘[5] 및 골드스미트(Goldschmidt) 나눗셈 알고리즘이 있다. 곱셈을 이용하는 방식은 SRT와 비교하여 속도가 빠르고 추가적인 하드웨어가 크지 않다는 장점이 있으나 근사 값만을 얻는다.

Pineiro[6]는 작은 곱셈기를 직렬로 연결하여 골드스미트 방식으로 부동소수점의 나눗셈, 역수, 제곱근, 역제곱근을 구하는 방식을 제안하였다. 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[7].

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 수행하여 연산 속도를 저하시키는 단점이 있었다. 김성기는 가변 시간 알고리즘을 제안하여 이러한 문제를 개선했으나 한 회 반복에 항상 두 번의 곱셈을 사용했으므로 가외의 연산이 남아있으며 또한 근사 값만을 얻을 수 있었다[8].

조경연은 한 번 반복에 K번 곱셈을 수행하는 K차 뉴톤-랍손 역수 알고리즘을 제안하였다[9]. 뉴톤-랍손 알고리즘은 오류 예측이 용이하나 병렬 곱셈이 되지 않는다.

본 논문에서는 테일러급수 정리로부터 가칭 K차 골드스미트 부동소수점 역수 알고리즘을 유도한다. K차 골드스미트 부동소수점 역수 알고리즘은 한 번 반복에 K번 곱셈을 수행한다. 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 수행하는 알고리즘을 제안한다. K차 골드스미트 알고리즘은 병렬 곱셈이 가능하다.

한편 골드스미트 알고리즘을 사용한 나눗셈은 근사 계산으로 정확한 결과를 얻기 위해서는 보정이 필요하다. Anderson[10]은 요구되는 정밀도보다 10비트를 더 계산하는 것을 제안했으나 항상 정확한 값을 보장하지는 못한다. Viitanen[11]과 Pasca[12]는 입력 값에 스케일링을 도입하여 오차를 보정하였다. Markstein[13]은 요구되는 정밀도의 2배 길이 계산을 제안했으나 이 또한 항상 정확한 값을 보장하지는 못한다. Schwarz[14]가 제안한 방식은 스티키 비트를 정확히 산출하지 못한다. Brisebarre[15]는 제수가 정해진 경우에 보정하는 알고리즘을 제안했다.

본 논문에서는 나눗셈 $\frac{N}{1.d}$ 에서 제수 $1.d$의 역수로 '$1.d * 0.y \le 1-2^{-w}$'이 되는 $0.y$를 제안한 K차 골드스미트 역수 알고리즘으로 계산하고, 피제수 N을 보정하고 역수 $0.y$를 곱하여 '$(N+s) * 0.y = M.m$, $0.75 \le s < 1$'을 계산하고 그 결과를 보정하여 정확한 나눗셈 값을 산출한다. 이를 오차 교정 K차 골드스미트 나눗셈 알고리즘이라 가칭한다.

본 논문에서 제안한 알고리즘은 C 언어로 프로그래밍하여 정확한 계산이 산출되는 것을 검증하였고, 또한 Verilog HDL로 코딩하고 로직 시뮬레이션하여 동작을 검증하였다.

본 논문의 구성은 다음과 같다. 2장에서는 테일러급수 정리로부터 가칭 K차 골드스미트 역수 알고리즘을 유도하고, 나눗셈 결과 보정 알고리즘을 제안하고, 오차를 예측하는 방법을 제안하고, 연산 자릿수 및 반복을 종료할 오차 한계를 계산한다. 3장에서는 제안한 알고리즘을 구현하는 하드웨어 알고리즘을 제시한다. 4장에서는 근사 테이블을 구성하고, 나눗셈 계산에 소요되는 평균 곱셈 횟수를 계산한다. 그리고 그 결과를 종래 골드스미트 나눗셈 알고리즘과 비교 분석한다. 5장에서 결론을 맺는다.

Ⅱ. K차 골드스미트 나눗셈 알고리즘

2.1. K차 골드스미트 역수 알고리즘

부동소수점 수 D의 역수 X_n은 초기값 X_0를 정의하고, 반복식으로 $X_i(i=1,...,n)$을 구한다. IEEE-754[16]로 규정되는 부동소수점 수 D는 $1.d_2 * 2^{n+base}$이다. 가수부 $1.d_2$는 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 D의 가수부 1.d는 식 (1)과 같이 두 부분으로 나눌 수 있다.

$$1.d = 1.g + h \tag{1}$$

식 (1)에서 g와 h의 길이를 각각 n_g및 n_h 비트로 정의한다. h는 '$0 \le h < 2^{-n_g}$'이고, h의 최대값은 '$h_{\max} = 2^{-n_g} - 2^{-n_g - n_h}$'이다. 반복식의 수렴 속도를 빠르게 하기 위해서 $\frac{1}{1.g}$를 근사계산하여 테이블 T(g)를 미리 작성해놓는다.

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. i번째 반복식에서 X_i는 $\frac{1}{D}$의 근사값으로 오차를 e_i라고 하면 식 (2)가 된다.

$$X_i = \frac{1}{D} - e_i = \frac{1 - e_i D}{D} \tag{2}$$

$a_i = 1 - DX_i$를 정의하면 $\frac{1}{D}$는 식 (3)으로 구해진다.

$$\frac{1}{D} = \frac{X_i}{1 - e_i D} = \frac{X_i}{1 - a_i} = X_i \sum_{j=0}^{\infty} a_i^j = X_i (R_i^{(k)} + e_{ri}) \tag{3}$$

식 (3)에서 $R_i^{(k)} = \sum_{j=0}^{k-1} a_i^j$, $e_{ri} = \sum_{j=k}^{\infty} a_i^j$이다. $R_i^{(k)} \gg e_{ri}$이므로 X_{i+1}은 식 (4)와 같이 정의한다.

$$X_{i+1}^{(k)} = X_i R_i^{(k)} = \frac{1}{D} - e_{i+1} \tag{4}$$

$$e_{i+1} = \frac{1 - e_i D}{D} \sum_{j=k}^{\infty} a_i^j \fallingdotseq \frac{a_i^k}{D} = e_i^k D^{k-1}$$

이로부터 a_{i+1}은 식 (5)가 된다.

$$a_{i+1} = 1 - DX_{i+1} = e_{i+1} D = e_i^k D^k = a_i^k \tag{5}$$

식 (2)부터 식 (5)까지를 정리하면 식 (6)이 된다.

$$X_0 = \frac{1}{D} - e_0 \tag{6}$$

$a_0 = 1 - DX_0$

for i=0 to N-1

{ $R_i^{(k)} = \sum_{j=0}^{k-1} a_i^j$

$X_{i+1}^{(k)} = X_i R_i^{(k)} = \frac{1}{D} - e_{i+1}$ $(e_{i+1} \fallingdotseq e_i^k D^{k-1})$

$a_{i+1}^{(k)} = a_i^k$ }

식 (6)을 가칭 K차 골드스미스 역수 알고리즘이라고 한다. 식 (6)에서 k=2이면 골드스미트 역수 알고리즘이 된다.

2.2. 나눗셈 및 오차 보정

정수 N을 부동소수점 수 1.d로 나누는 것은 $Ceiling(\frac{N}{1.d}) = Q$, $d \neq 0$로 표기할 수 있다.

이를 연산하기 위하여 $\frac{1}{1.d} \fallingdotseq 0.y$, $0.y * 1.d = 1 - e$, $e < 2^{-w-1}$가 되는 $0.y$를 구한다. w는 워드 길이로 '$2^{w-1} \le N < 2^w$'이다. $0.y$은 식 (6)으로 구한다. 작은 수 s를 N에 더하고, 이를 $0.y$을 곱하여 정리하면 식 (7)이 된다.

$$\begin{aligned}(N+s) * 0.y &= \frac{N}{1.d} + \frac{s - eN - es}{1.d} \\ &= Q.q + \frac{s - eN - es}{1.d} = M.m\end{aligned} \tag{7}$$

식 (7)에서 ‘$0.75 \le s < 1$’이라면 ‘$0 \le 0.q + \frac{s-eN-es}{1.d} < 2$’이 된다. 이로부터 ‘$Q \le (N+s) * 0.q < Q+2$’이 되고, 따라서 ‘$Q \le M \le Q+1$’이 성립한다. 이제 ‘$M * 1.d = T.t$’를 계산하면, 다음의 3가지 경우가 생긴다.

[1] ‘$T = N$’이고 ‘$t = 0$’인 경우 -- ‘$Q = M,\ q = 0$’이다. 스티키 비트는 ‘0’이 된다.
[2] ‘$T = N-1$’ 또는 ‘$T = N-2$’인 경우 -- ‘$Q = M,\ q \ne 0$’이다. 스티키 비트는 ‘1’이 된다. ‘$Q * 1.d = (Q.q - 0.q) * 1.d = N - 0.q * 1.d$’인 경우이다.
[3] ‘$T = N+1$’이거나 ‘$T = N, t \ne 0$’인 경우 -- ‘$Q = M-1, q \ne 0$’이다. 스티키 비트는 ‘1’이 된다. $(Q+1) * 1.d = (Q.q + 0.q') * 1.d = N + 0.q' * 1.d$인 경우이다.

‘$Q = M-1, q = 0$’인 경우는 발생하지 않는다.

2.3. 오차 분석 및 예측

식 (6)에서 ‘$a_0 = 1 - DX_0$’에서 뺄셈은 하드웨어 구현 시에 캐리 전달 지연이 발생한다. 이러한 문제점을 해결하기 위하여 본 논문에서는 식 (8)로 a_0를 구한다.

$$a_0 = 1 - 2^{-p} - DX_0 \tag{8}$$

식 (8)에서 DX_0 곱셈은 소수점 이하 p 비트 미만을 절삭하면 식 (9)가 된다.

$$\begin{aligned} a_0 &= 1 - 2^{-p} - (D(\frac{1}{D} - e_0) - u2^{-p}) \\ &= De_0 - (1-u)2^{-p} \ge De_0 - 2^{-p} \end{aligned} \tag{9}$$

식(7)에서 $u2^{-p}(0 \le u < 1)$는 곱셈 결과를 절삭하면서 발생하는 오차이며, ‘$u = 0$’에서 오차가 최대가 된다. 식 (9)와 식 (6)로부터 $X_1^{(2)}$는 식 (10)이 된다.

$$\begin{aligned} X_1^{(2)} &= (\frac{1}{D} - e_0)(1 + De_0 - 2^{-p}) - u2^{-p} \\ &\le \frac{1}{D} - De_0^2 - 2 * 2^{-p} \end{aligned} \tag{10}$$

또한 $X_1^{(3)}$은 식(11)이 된다. $u2^{-p}$와 $t2^{-p}$는 곱셈 결과 절삭 오차이다.

$$\begin{aligned} X_1^{(3)} &= X_0(1 + a_0(1 + a_0) - u2^{-p}) - t2^{-p} \\ &\le \frac{1}{D} - D^2 e_0^3 - 5 * 2^{-p} \end{aligned} \tag{11}$$

식 (10) 및 식 (11)과 같이 연산을 반복하면 절삭 오차가 누적된다. 반복 연산에서 누적되는 최대 절삭 오차를 표 1에 보인다.

표 1. 반복 연산에 따른 최대 누적 오차
Table. 1 Maximum accumulated error according to iteration

Iteration	Maximum accumulated truncated error
$X_1^{(2)}$	$2 * 2^{-p}$
$X_1^{(3)}$	$5 * 2^{-p}$
$X_1^{(2)}, X_2^{(2)}$	$6 * 2^{-p}$
$X_1^{(2)}, X_2^{(3)}$	$9 * 2^{-p}$
$X_1^{(2)}, X_2^{(2)}, X_3^{(2)}$	$10 * 2^{-p}$
$X_1^{(2)}, X_2^{(2)}, X_3^{(3)}$	$13 * 2^{-p}$
$X_1^{(2)}, X_2^{(2)}, X_3^{(2)}, X_4^{(2)}$	$14 * 2^{-p}$

표 1에서 $X_i^{(2)}$를 4번 반복 연산하는 경우에 절삭에 따른 오차는 2^{-p+4}보다 작다. 본 논문에서는 a_{i+1}이 2^{-p+4}보다 작으면 반복 연산을 종료한다.

2.4. 연산 유효자릿수

IEEE-754 단정도실수와 배정도실수에서 가수부의 유효자릿수는 각각 24 비트와 53비트이다. 유효자릿수에 라운드 한 비트를 더하면 25 비트와 54비트이다. 2^{-p+4}가 2^{-25} 및 2^{-54}보다 작아야 한다. 또한 연산 중간에 음수가 발생하므로 사인 비트와 나눗셈 오차 보장을 위해 각각 한 비트가 추가로 필요하다. 따라서 단정도실수와 배정도실수에서 p는 각각 31과 60이다. 연산 유효자릿수를 표 2에 보인다.

표 2에서 $x^{(2)}$와 $x^{(3)}$은 반복 종료를 위한 $a_i^{(2)}$와 $a_i^{(3)}$의 소수점 이하 계속되는 ‘0’ 또는 ‘1’ 비트의 수이다. 즉 단정도실수에서 ‘$\left|a_i^{(2)}\right| < 2^{-16}$’이면 반복을 종료한다.

표 2. 유효자릿수. 단위는 비트
Table. 2 Effective digits in bit

	Single Precision	Double Precision
w	24	53
p	31	60
$x^{(2)}$	16	30
$x^{(3)}$	11	20

Ⅲ. 오차 교정 K차 골드스미트 나눗셈 계산기

하나의 곱셈기를 사용하여 하드웨어로 구현한 오차 교정 K차 골드스미트 나눗셈 알고리즘을 표 3에 보인다. 표 3에서 $\frac{N}{D} = Q$, $1 < D = 1.g + h < 2$, $2^{w-1} \le N < 2^w$ 이다.

표 3에서 상태-1부터 상태-5에서 D의 근사 역수 X를 구한다. 상태-1에서 $1.g$의 근사 역수 X_0를 테이블로부터 읽어서 레지스터 X에 저장한다. 상태-2에서 식 (6)의 a_0를 계산하여 레지스터 A에 저장한다. 또한 A의 소수점 이하부터 연속해서 나타나는 ‘0’ 또는 ‘1’ 비트의 수를 세서 레지스터 B에 저장한다. 하드웨어 설계시에 B는 $x^{(2)}$보다 큰 경우와 $x^{(3)}$보다 작은 경우만이 참조되므로 $x^{(2)}$ 비트 입력 AND 게이트와 OR 게이트, $x^{(3)}$ 비트 입력 AND 게이트와 OR 게이트로 구현한다. B가 $x^{(2)}$보다 크면 $X_{i+1}^{(2)}$ 이 구하려는 근사 역수이므로 상태-4로 전이한다. 또한 B가 $x^{(3)}$보다 작으면 $X_{i+1}^{(2)}$ 을 구하고 반복 연산을 해야 하므로 상태-4로 전이한다. 상태-3은 B가 $x^{(2)}$보다 작으면 $x^{(3)}$보다 큰 경우로 $X_{i+1}^{(3)} = X_i(1+a_i(1+a_i))$가 구하려는 근사 역수이므로 $a_i(1+a_i)$를 연산하여 레지스터 A에 저장하고, 상태-4로 전이한다. 상태-4에서 $X_{i+1}^{(2)}$ 또는 $X_{i+1}^{(3)}$ 을 계산하여 레지스터 X에 저장한다.

표 3. 오차 교정 K차 골드스미트 나눗셈 알고리즘. 하나의 곱셈기를 사용한 경우.
Table. 3 Error corrected K'th order Goldschmidt's Floating Point Number divider algorithm using single multiplier

(State-1)
Reciprocal table T(1.g) => X;
(State-2)
$1-2^{-p}-DX => A$;
No. of Leading bits
after period of A $=> B$;
If $B \ge x^{(2)}$ OR $B < x^{(3)}$,
then goto state-4 ;
(state-3)
$A(1+A) => A$;
(state-4)
$X(1+A) => X$;
If $B \ge x^{(3)}$, then goto state-6 ;
(state-5)
$A^2 => A$;
No. of Leading bits
after period of A $=> B$;
If $B \ge x^{(2)}$ OR $B < x^{(3)}$,
then goto state-4 ;
else goto state-3 ;
(state-6)
$X((N \ll 2)+3) => X$;
(state-7)
$XD => T.t$;
$T \wedge 3 => r$;
(state-8)
If r==0 AND t==0,
then $\{(X \gg 2) => Q$, 0=>ST } ;
If r==1 OR (r==0 AND t!=0),
then $\{(X \gg 2) - 1 => Q$, 1=>ST} ;
If r>1,
then $\{(X \gg 2) => Q$, 1=>ST } ;

레지스터 B가 $x^{(3)}$보다 작으면 반복 연산이 필요하므로 상태-5로 전이하고, 크면 반복 연산을 종료하고 상태-6으로 전이해서 나눗셈을 수행한다. 상태-5에서는 ‘$a_{i+1}^{(2)} = a_i^2$’을 연산해서 레지스타 A에 저장하고, A의 소수점 이하부터 연속해서 나타나는 ‘0’ 또는 ‘1’ 비트의 수를 세서 레지스터 B에 저장하고, B의 값에 따라서 $X_{i+1}^{(2)}$ 또는 $X_{i+1}^{(3)}$ 을 구하기 위하여 해당하는 상태로 전

이한다.

상태-6에서는 N을 왼쪽으로 2 비트 이동시키고 3을 더한 후에 $0.y$를 곱해서 식 (7)의 M을 계산하여 레지스터 X에 저장한다. 2 비트 왼쪽으로 이동시킨 이유는 2.2절의 조건 $T=N-1$, $T=N-2$, $T=N$, $T=N+1$을 T의 비트 0와 비트 1로 판별할 수 있기 때문이다. 또한 식 (7)의 s 값을 3으로 설정하여 N에 더해주는 것이다. 상태-7에서 '$M*1.d=T.t$'를 계산하여 레지스터 T와 t에 저장한다.

표 4. 오차 교정 K차 골드스미트 나눗셈 알고리즘. 두 개의 곱셈기를 사용한 경우.
Table. 4 Error corrected K'th order Goldschmidt's Floating Point Number divider algorithm using dual multiplier

```
(State-1)
  Reciprocal table T(1.g) => X;
(State-2)
  X((N ≪ 2)+3) => X ;
  1-2^(-p) - DX => A ;
  No. of Leading bits
     after period of A => B ;
  If B ≥ x^(2) OR B < x^(3),
     then goto state-4 ;
(state-3)
  A(1+A) => A ;
(state-4)
  X(1+A) => X ;
  A^2 => A ;
  If A < 2^(-p), then goto state-5 ;
   else
   { No. of Leading bits
       after period of A => B ;
     If B ≥ x^(2) OR B < x^(3),
       then goto state-4 ;
       else goto state-3 ; }
(state-5)
  XD => T.t ;
  T∧3 => r ;
(state-6)
  If r==0 AND t==0,
   then {(X ≫ 2) => Q, 0=>ST } ;
  If r==1 OR (r==0 AND t!=0),
   then {(X ≫ 2)-1 => Q, 1=>ST} ;
  If r>1,
   then {(X ≫ 2) => Q, 1=>ST } ;
```

또한 레지스터 T의 하위 2비트를 레지스터 r에 저장한다. 상태-8에서 2.2절의 조건을 판별하여 나눗셈 결과 Q와 스티키 비트 ST를 구한다. 제시한 알고리즘은 C 언어로 프로그램하였다. 단정도실수에서 상태-1부터 상태-5까지의 근사 역수 $0.y$를 전수 계산하여 SRT로 계산한 결과와 비교하여 일치하는 것을 확인하였다. 단정도실수와 배정도실수 각각에서 SHA 해쉬 함수를 사용하여 제수 정수 N과 피제수 부동소수점수 D 각각 10^7 개를 생성하고, 제시한 알고리즘으로 나눗셈을 수행하고, 그 결과를 SRT로 계산한 결과와 비교하여 일치하는 것을 확인하였다.

IBM-PC의 window-7에서 Icarus Verilog version 0.9를 사용하여 HDL로 코딩하고 시뮬레이션하여 동작을 확인하였다. 두 개의 곱셈기를 사용하여 하드웨어로 구현한 오차 교정 K차 골드스미트 나눗셈 알고리즘을 표 4에 보인다.

Ⅳ. 연구 결과 및 분석

DasSarma[17]의 연구 결과 최적의 근사 역수는 식 (12)로 주어진다.

$$T(g)=\frac{1}{1.g} \fallingdotseq RN\left(\frac{1}{1.g+2^{-2n_g-1}}\right) \qquad (12)$$

RN is round to nearest

T(g)의 소수점 이하 길이를 t 비트라고 하면 '$T(g)=(b_0,b_1,...,b_t)_2$, $0.5<T(g)\le 1.0$'. '$b_0b_1=10$'인 경우는 '$g=0$'일 때이다. 이외의 경우는 항상 '$b_0b_1=01$'이다. 그러므로 근사 역수 테이블에 '$b_2,...b_t$'만을 저장하면 된다. 따라서 근사 역수 테이블의 크기는 '$2^{n_g}*(t-1)$' 비트가 되어서, 테이블의 길이는 2^{n_g}이며, 폭은 't-1' 비트이다.T(g)에서 초기 오차 e_0는 식 (13)이 된다.

$$e_0=\frac{1}{1.g+h}-T(g) \qquad (13)$$

식 (12)로부터 e_0는 $h_m=100...0$에서 가장 작으며,

$h_z = 000...0$과 $h_{\max} = 111...1$에서 가장 커서 그림-1과 같이 된다.

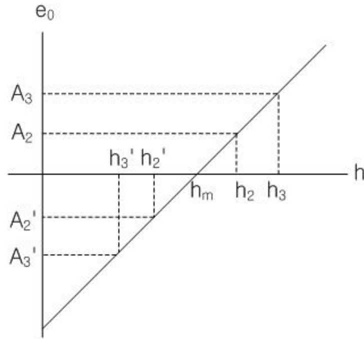

그림 1. 테이블 오차(h)와 초기 오차(e_0)의 그라프
Fig. 1 Error in table(h) versus initial error(e_0) graph

식 (10)에서 최대오차 '$De_0^2 + 2^{-p+1}$'이 2^{-p+4}보자 작으면 2회의 곱셈으로 근사 역수를 계산할 수 있다. 즉, 식 (14)가 성립하면 2회의 곱셈으로 근사 역수를 계산할 수 있다.

$$e_0 < A_2 = \frac{\sqrt{14} * 2^{-p/2}}{\sqrt{1.g + h_{\max}}} \tag{14}$$

식 (14)에서 A_2가 최소가 되는 값을 선택했다. 초기 오차 e_0는 양수와 음수의 두 가지 값을 가지며, 그림-1에 각각 A_2와 A_2'로 나타나고 있으며, A_2와 A_2'에서의 h 값이 각각 h_2와 h_2'이다. $h_2' < h < h_2$에서 2회의 곱셈으로 근사 역수를 계산할 수 있다.

식 (11)의 최대오차 '$D^2e_0^3 + 5 * 2^{-p}$'이 2^{-p+4}보다 작으면 3회의 곱셈으로 근사 역수를 계산할 수 있으므로 식 (15)가 성립하면 3회의 곱셈으로 근사 역수를 계산할 수 있다.

$$e_0 < A_3 = \frac{\sqrt[3]{11 * 2^{-p}}}{(\sqrt[3]{1.g + h_{\max}})^2} \tag{15}$$

그림-1에 A_3와 A_3'에서의 h 값이 각각 h_3와 h_3'이다. $h_3' < h < h_2'$와 $h_2 < h < h_3$에서 3회의 곱셈으로 근사 역수를 계산할 수 있다.

$a_1^{(2)} = a_0^2$이므로 '$a_0^2 < A_2$'가 되는 $e_1 = A_4$를 구할 수 있다. 이로부터 A_4와 A_4'에서의 h 값 h_4와 h_4'을 구할 수 있다. $h_4' < h < h_3'$와 $h_3 < h < h_4$에서 4회의 곱셈으로 근사 역수를 계산할 수 있다. 이와 같은 계산을 계속하여 수행하면 초기 오차 e_0에 따라서 근사 역수를 계산하기 위한 곱셈의 횟수를 산출할 수 있다.

본 논문에서 제안한 알고리즘에 의한 IEEE 단정도실수 및 배정도실수의 테이블 크기에 따른 나눗셈 계산에 필요한 곱셈 횟수를 표 5와 표 6에 보인다.

종래 골드스미스 알고리즘에서는 최대 오차를 고려해서 반복 횟수를 정했다. 표 5와 표 6에서 'GS No. of Multiply'는 종래 골드스미트 알고리즘에서의 곱셈 횟수이다. '1 mult'와 '2 mult'는 각각 곱셈기를 하나 사용한 경우와 두 개 사용한 경우이다.

하나의 곱셈기를 사용하는 경우에 종래 알고리즘에 의한 단정도실수 나눗셈은 '128x6' 테이블을 사용하면 7회의 곱셈을 수행하였고, '256x8' 테이블을 사용하면 5회의 곱셈을 수행하였음을 표 5로부터 알 수 있다. 그러나 본 논문에서 제안한 알고리즘에서는 '128x6' 테이블에서 평균 4.70회의 곱셈, '256x7' 테이블에서 평균 4.66 회의 곱셈으로 나눗셈을 할 수 있다.

평균 5회의 곱셈으로 나눗셈을 계산하려면 종래 알고리즘에서는 '256x7' 테이블을 사용했지만, 본 논문에서 제안하는 알고리즘을 사용하면 '64x6' 테이블을 사용해도 5회 곱셈으로 나눗셈을 계산할 수 있다. '64x6' 테이블의 면적은 '256x7' 테이블의 사분의 일에 불과하다.

표 5. IEEE 단정도실수 나눗셈 계산에 필요한 곱셈 횟수
Table. 5 Number of multiplier of IEEE single precision floating pointer number division

Table size	Average No. of Multiply		GS No. of Multiply	
	1 mult	2 mult	1 mult	2 mult
16x3	6.36	3.76	7	4
32x4	6.03	3.52	7	4
64x5	5.41	3.15	7	4
64x6	4.90	2.98	7	4
128x6	4.70	2.98	7	4
256x7	4.66	2.96	5	3

표 6. IEEE 배정도실수 나눗셈 계산에 필요한 곱셈 횟수
Table. 6 Number of multiplier of IEEE double precision floating pointer number division

Table size	Average No. of Multiply		GS No. of ultiply	
	1 mult	2 mult	1 mult	2 mult
64x5	7.60	4.35	9	5
64x6	7.07	4.10	9	5
128x6	6.81	4.00	9	5
256x7	6.67	3.98	7	4
512x8	6.53	3.96	7	4

이러한 결과는 배정도실수 연산에서도 동일하게 나타난다. 배정도실수 나눗셈 계산은 표 6으로부터 종래 알고리즘에서는 '128x6' 테이블을 사용하면 9회의 곱셈, '256x7' 테이블을 사용하면 7회의 곱셈을 수행하였다. 그러나 본 논문에서 제안한 알고리즘에서는 '128x6' 테이블을 사용하면 평균 6.81회의 곱셈, '256x7' 테이블을 사용하면 평균 6.67회의 곱셈으로 나눗셈을 수행한다.

표 5와 표 6은 근사 나눗셈에 소요되는 곱셈 회수이며 정확한 결과를 구하기 위한 오차 보정에 한 번의 곱셈과 판정이 추가된다.

Ⅴ. 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 테일러 급수로부터 가칭 K차 골드스미트 역수 알고리즘을 유도하였다. K차 알고리즘에서는 한 회 반복에 K번의 곱셈을 수행한다. 또한 반복 연산 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 연산을 수행한다.

나눗셈 $\frac{N}{1.d}$ 에서 $1.d$의 역수로 '$1.d * 0.y \le 1-2^{-w}$'이 되는 $0.y$를 제안한 알고리즘으로 계산하고, '$(N+s) * 0.g = M.m,\ 0.75 \le s < 1$'을 계산하고 그 결과를 보정하여 정확한 나눗셈 값을 산출한다. 이를 오차 교정 K차 골드스미트 나눗셈 알고리즘이라 가칭한다.

제안한 알고리즘을 verilog로 구현하고 시뮬레이션하여 동작을 검증하였으며, 다양한 근사테이블에서 기존의 골드스미트 알고리즘과 비교하여 계산속도가 개선되었음을 증명하였다.

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

ACKNOWLEDGMENTS

This work was supported by a Research Grant of Pukyong National University(2015 Year).

REFERENCES

[1] V. Lappalainen, et al, "Overview of Research Efforts on Media ISA Extension and their Usage in Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 12, pp. 660-670, Aug. 2002.

[2] Taek-Jun Kwon, Jeff Sondeen and Jeff Draper, "Floating-Point and Square Root Implementation using a Taylor-Series Expansion Algorithm," *Circuits and Systems Signal Processing Systems, IEEE 50th Midwest Symposium on*, pp. 702-705, 2008.

[3] 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, Feb. 1997.

[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] Nicolas Louvet, Jean-Michel Muller, and Adrien Panhaleuax, "Newton-Raphson Algorithms for Floating-Point Divsion Using and FMA," *ASAP, 2010 21st IEEE International Conference on*, pp. 200-207, 2010.

[6] J. A. Pineiro, et al, "High-speed double-precision computation of reciprocal, division, square root and inverse square root ," *IEEE transaction on Computers*, Vol. 51, No. 12, pp. 1377-1388, Dec. 2002.

[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] Sung-Ki Kim, Hong-Bok Song and Gyeong-Yeon Cho, "A Variable Latency Goldschmidt's Floating Point Number Divider," *Journal of the Korea Institute of Maritime Information and Communication Sciences*, Vol. 9, No. 2, pp. 380-389, April, 2005.

[9] Gyeong-Yeon Cho, "A Variable Latency K'th Order Newton-Raphson's Floating Point Number Divider," *Journal of IEMEK*, Vol. 9, No. 4, pp. 285-292, Oct. 2014.

[10] S. F. Anderson, et al, "The IBM System/360 model 91 Floating Point Execution Unit," *IBM Journal of Research and Development*, 11(1), pp. 34-53, Jan. 1967.

[11] Timo Viitanen, Pekka Jaakelainen, and Jarmo Takala, "Inexpensive Correctly Rounded Floating-Point Division and Square Root with Input Scaling," *Proceedings of the 2013 IEEE Workshop on Signal Processing Systems, SiPS* 2013, Oct. 2013.

[12] Bogdan Pasca, "Correctly Rounded Floating-Point Division for DSP-Enabled FPGAs," *Field Programmable Logic and Applications 22nd International Conference on*, pp. 240-254, Aug. 2012.

[13] P. Markstein, "Computation of elementary functions on the IBM RISC system/6000 processor," *IBM Journal of Research and Development*, 34(1), pp. 111-119, Jan. 1990

[14] Etic M. Schwarz, "Rounding for Quadratically Converging Algorithm for Division and Square Root," *In Proc. 29th Asilomar Conference on Signals and Computers, IEEE*, pp. 600-603, 1996

[15] Nicolas Brisebarre, Jean-Michel Muller, and Saurabh Kumar, "Accelerating Correctly Rounded Floating-Point Division when the Divider Is Known in Advance," *IEEE Transactions on Computers*, Vol. 53, No. 8, pp. 1069-1072, Aug. 2004.

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

[17] 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.

조경연(Gyeong-Yeon Cho)

1990 인하대학교 전자공학과 졸업(박사)
1991~현재 부경대학교 IT융합응용공학과 교수
1998~현재 에이디칩스(주) 기술고문
※관심분야 : 컴퓨터구조, 반도체설계, 정보보안