Journal of the Korea Academia-Industrial cooperation Society
Vol. 15, No. 5 pp. 3093-3099, 2014

http://dx.doi.org/10.5762/KAIS.2014.15.5.3093
ISSN 1975-4701 / eISSN 2288-4688

32 비트 곱셈기를 사용한 골드스미트 배정도실수 역수 계산기

조경연[1*]
[1]부경대학교 IT 융합응용공학과

Goldschmidt's Double Precision Floating Point Reciprocal Computation using 32 bit multiplier

Gyeong-Yeon Cho[1*]
[1]**Department of IT Convergence and Application Engineering, Pukyong National University**

요 약 최근 그래픽 프로세서, 멀티미디어 프로세서, 음성처리 프로세서 등에서 부동소수점이 주로 사용된다. 한편 C, Java 등 고급언어에서는 단정도실수와 배정도실수를 사용하고 있다. 본 논문에서는 32 비트 곱셈기를 사용하여 배정도실수의 역수를 계산하는 알고리즘을 제안한다. 배정도실수 가수를 상위 부분과 하위 부분으로 나누고, 상위 부분의 역수를 골드스미스 알고리즘으로 계산하고, 이를 초기값으로 하여 배정도실수의 역수를 계산하는 알고리즘을 제안한다. 제안한 알고리즘은 입력값에 따라서 곱셈 횟수가 다르므로, 평균 곱셈 횟수를 계산하는 방식을 유도하고, 여러 크기의 근사 역수 테이블에서 평균 곱셈 횟수를 계산한다.

Abstract Modern graphic processors, multimedia processors and audio processors mostly use floating-point number. Meanwhile, high-level language such as C and Java uses both single-precision and double precision floating-point number. In this paper, an algorithm which computes the reciprocal of double precision floating-point number using a 32 bit multiplier is proposed. It divides the mantissa of double precision floating-point number to upper part and lower part, and calculates the reciprocal of the upper part with Goldschmidt's algorithm, and computes the reciprocal of double precision floating-point number with calculated upper part reciprocal as the initial value is proposed. Since the number of multiplications performed by the proposed algorithm is dependent on the mantissa of floating-point number, the average number of multiplications per an operation is derived from some reciprocal tables with varying sizes.

Key Words : Double precision floating point, Goldschmidt algorithm, Reciprocal, Variable latency

1. 서론

부동소수점 계산은 과학 및 공학 기술 분야에서 많이 사용된다. 최근에는 음성 처리 및 3차원 그래픽 분야 등 멀티미디어 분야에도 폭넓게 사용되면서 CPU의 기본 기능으로 채택되고 있다[1,2].

부동소수점 나눗셈은 뺄셈을 반복하는 SRT[3,4] 알고리즘과 곱셈을 이용한 알고리즘으로 뉴톤-랍손(Newton-Raphson) 역수 알고리즘 및 골드스미트(Goldschmidt) 나눗셈 알고리즘이 있다. 곱셈을 반복하는 방식은 SRT와 비교하여 속도가 빠르지만 근사 값만을 얻는다. 32 비트 정수 곱셈기는 대부분의 프로세서가

이 논문은 2012학년도 부경대학교 연구년 교수 지원 사업에 의하여 연구되었음(PS-2012-C-D-2013-0073)
*Corresponding Author : Gyeong-Yeon Cho(Pukyong National Univ.)
Tel: +82-51-629-6252 email: gycho@pknu.ac.kr
Received November 4, 2013 Revised (1st March 11, 2014, 2nd April 1, 2014, 3rd April 29, 2014) Accepted May 8, 2014

기본적으로 가지고 있으므로 곱셈을 반복하는 부동소수점 나눗셈은 추가적인 하드웨어가 크지 않다는 장점을 가진다. 일반적으로 멀티미디어 등의 응용 분야에서는 높은 정밀도를 요구하지 않으므로 단정도실수 연산이 대부분이며 또한 근사 값만으로도 충분하다.

한편 C, Java 등 고급언어는 배정도실수와 단정도실수 모두를 사용한다. 배정도실수 나눗셈은 64 비트 곱셈기를 필요로 하는데, 실장제어분야에서는 배정도실수의 사용빈도가 낮으므로 작은 곱셈기를 사용하여 배정도실수 연산을 수행하는 연구가 요구된다. Wong[5]은 56 X 16 비트 곱셈기를 사용하였으며, Brightman[6]은 17 X 69 곱셈기 어레이를 사용했다. Ozbilen[7]은 SIMD 곱셈기를 사용했다.

김성기[8]는 64비트 곱셈기를 사용하고, 골드스미스 알고리즘의 반복 과정 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 수행하여 가변시간 배정도실수 역수를 계산하였다.

본 논문에서는 IEEE 배정도실수[9] D의 가수부 53 비트의 상위 28 비트의 역수 X_f를 32 비트 곱셈기를 사용하여 골드스미스 부동소수점 역수 알고리즘으로 계산하고, X_f를 초기값으로 하여 골드스미스 부동소수점 역수 알고리즘으로 D의 역수를 계산한다. 또한 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 수행한다.

본 논문에서 제안한 알고리즘은 C 모델링하여 정확한 계산이 산출되는 것을 검증하였고, Verilog HDL로 하드웨어로 구현하여 로직 시뮬레이션하여 동작을 검증하였다.

본 논문의 구성은 다음과 같다. 2장에서는 32 비트 곱셈기를 사용한 배정도실수 역수 알고리즘을 제안하고, 3장에서는 제안한 알고리즘을 구현하는 하드웨어 알고리즘을 제시한다. 4장에서는 근사 테이블을 구성하고, 역수 계산에 소요되는 평균 곱셈 횟수를 계산한다. 그리고 그 결과를 종래 골드스미스 역수 알고리즘과 비교 분석한다. 5장에서는 향후 연구 과제를 제시하고, 6장에서 결론을 맺는다.

2. 배정도 역수 알고리즘

2.1 골드스미트 역수 알고리즘

부동소수점 수 D의 역수 X_n은 초기값 X_0를 정의하고, 반복식으로 $X_i(i=1,...,n)$을 구한다. IEEE-754로 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 세 부분으로 나눌 수 있다.

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

식 (1)에서 g, h와 i의 길이를 각각 n_g, n_h, n_i 비트로 정의한다. X_i의 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.

본 논문에서는 배정도실수 '$D = 1.d$'의 역수를 두 단계로 나누어서 계산한다. 즉, 식 (1)의 '$1.f = F$"의 역수 X_f를 계산하고, X_f를 초기값으로 하여 '$D = 1.d$'의 역수 X_n을 구한다.

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

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

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

$$\frac{1}{F}=\frac{X_i}{1-e_iF}=\frac{X_i}{1-a_i}=X_i(1+a_i+\sum_{j=2}^{\infty}a_i^j) \quad (3)$$

if $|a_i|<1$

식 (3)으로부터 X_{i+1}은 식 (4)와 같이 정의한다.

$$X_{i+1}=X_i(1+a_i)=\frac{1}{F}-e_{i+1} \quad (4)$$

$$e_{i+1}=\frac{1-e_iF}{F}\sum_{j=2}^{\infty}a_i^j \fallingdotseq \frac{a_i^2}{F}=e_i^2F$$

이로부터 a_{i+1}은 식 (5)가 된다.

$$a_{i+1}=1-FX_{i+1}=e_{i+1}F=e_i^2F^2=a_i^2 \quad (5)$$

식 (2)부터 식 (5)까지를 정리하면 식 (6)이 된다.

$$X_0=\frac{1}{F}-e_0 \ ; \quad (6)$$

$a_0=1-FX_0$;

for i=0 to N-1

{ $R_i=1+a_i$;

$X_{i+1}=X_iR_i=\frac{1}{F}-e_{i+1}$;

$(e_{i+1} \fallingdotseq e_i^2F)$

$a_{i+1}=a_i^2$; }

2.2 오차 분석 및 예측

식 (6)에서 '$a_0=1-FX_0$'에서 뺄셈은 하드웨어 구현 시에 캐리 전달 지연이 발생한다. 이러한 문제점을 해결하기 위하여 본 논문에서는 식 (7)로 a_0를 구한다.

$$a_0=1-2^{-p}-FX_0 \quad (7)$$

식 (7)에서 FX_0 곱셈은 소수점 이하 p 비트 미만을 절삭하면 식 (8)이 된다.

$$a_0=1-2^{-p}-(F(\frac{1}{F}-e_0)-u2^{-p}) \quad (8)$$

$$=Fe_0-(1-u)2^{-p} \geq Fe_0-2^{-p}$$

식(8)에서 $u2^{-p}(0 \leq u<1)$는 곱셈 결과를 절삭하면서 발생하는 오차이며, '$u=0$'에서 오차가 최대가 된다. 식 (8)과 식 (6)으로부터 X_1는 식 (9)가 된다.

$$X_1=(\frac{1}{F}-e_0)(1+Fe_0-2^{-p})-u2^{-p} \quad (9)$$

$$\leq \frac{1}{F}-Fe_0^2-2*2^{-p}$$

[Table 1] Maximum accumulated truncating error according to iterated calculation

iterated number	Maximum accumulated truncating error
X_1	$2*2^{-p}$
X_1, X_2	$6*2^{-p}$
X_1, X_2, X_3	$10*2^{-p}$
X_1, X_2, X_3, X_4	$14*2^{-p}$

연산을 반복하면 절삭 오차가 누적된다. 반복 연산에서 누적되는 최대 절삭 오차를 식-9와 같이 계산하여 Table 1에 보인다.

Table 1에서 4번 반복 연산하는 경우에 절삭 오차는 '2^{-p+4}'보다 작다. 본 논문에서는 '$a_{i+1}=a_i^2<2^{-p+4}$'이면 반복 연산을 종료한다. 즉, '$a_i<2^{-x}=2^{-\frac{p}{2}+2}$'이면 반복 연산을 종료한다.

2.3 배정도 역수 확장

IEEE-754 배정도실수의 가수부 길이는 53 비트이다. 숨은 '1' 비트와 반올림 비트를 포함하면 55 비트 정밀도가 필요하다. X_f의 최대 오차가 2^{-f}라고 하면, X_n의 최대 오차는 2^{-2f}이다. '$2f>55$'가 되어야 하므로 식 (1)에서 '$n_f=n_g+n_h \geq 28$'이 되어야 한다. Table 1로부터 최대 누적 오차는 '2^{-p+4}'보다 작으므로 '$2^{-n_f} \leq 2^{-p+4}$'가 되어야 한다. 이로부터 '$p=32$, $x=14$, $n_f=28$'이다.

X_f를 구하고, 이를 초기값으로 하여 식-6을 1회 계산하여 D의 역수를 계산한다.

3. 배정도 역수 계산기

하드웨어 구현을 위한 개선된 골드스미스 배정도실수 역수 알고리즘을 Table 2에 보인다. Table 2에서는 32비트 곱셈기 하나를 사용한다. Table 2에서 P는 $1.d$의 상위 32 비트이고, Q는 $1.d$의 하위 22 비트를 왼쪽으로 정렬한 값이다. 즉, '$1.d = (P \ll 32) + Q$'이다.

Table 2에서 상태-1부터 상태-4에서 $1.f$의 근사 역수 X_f를 구한다. 상태-1에서 $1.g$의 근사 역수 X_0를 테이블로부터 읽어서 레지스터 X에 저장한다. 상태-2에서 '$a_0 = 1 - FX_0$'를 계산하여 레지스터 A에 저장한다. 상태-3에서 $X_{i+1} = X_i(1 + a_i)$을 계산하여 레지스터 X에 저장한다. 또한 A의 소수점 이하부터 연속해서 나타나는 '0' 또는 '1' 비트의 수를 세서 이를 B라고 한다. 하드웨어 설계시에 B는 x보다 작은 경우만이 참조되므로 x 비트 입력 AND 게이트와 OR 게이트로 구현한다. B가 x보다 작으면 상태-5로 전이해서 반복식을 종료한다. 상태-4에서 $a_{i+1} = a_i^2$을 계산한다.

[Table 2] Improved Goldschmidt's double precision floating point reciprocal algorithm

(state-1)
  Reciprocal table T(1.g) => X;
(state-2)
  $1 - 2^{-p} - FX => A$;
(state-3)
  $X(1 + A) => X$;
  No. of Leading bits
    after period of A= B ;
  If $B \geq x$, then goto state-5 ;
(state-4)
  $A^2 => A$;
  goto state-3 ;
(state-5)
  $QX => \{V : -\}$;
(state-6)
  $PX + V => \{U : V\}$;
  $2 - 2^{-63} - \{U : V\} => \{U : V\}$;
(state-7)
  $VX = \{V : -\}$;
(state-8)
  $UX + V => \{U : V\}$;

상태-5와 상태-6에서는 레지스터 X에 F의 근사 역수 X_f가 저장되어 있으므로, '$2 - 2^{-63} - DX_i$'를 계산하여 하위 32 비트 워드는 레지스터 V에, 상위 32 비트 워드는 레지스터 U에 각각 저장한다. 이를 위해서 상태-5에서는 D의 하위 32 비트 워드와 레지스터 X를 곱해서 상위 32 비트 워드를 레지스터 V에 저장하고, 하위 32 비트 워드는 버린다. 상태-6에서는 D의 상위 32 비트 워드와 레지스터 X를 곱하고, 그 결과에 레지스터 V를 더해서 상위 32 비트 워드를 레지스터 U에, 하위 32 비트 워드는 레지스터 V에 저장한다. 레지스터 V는 32 비트 곱셈기의 부분곱의 한 행을 늘리는 것에 해당하므로 추가적인 지연이 크지 않으므로 한 상태에서 처리할 수 있다. 이렇게 곱하고 더한 결과의 1의 보수를 취하면 '$2 - 2^{-63} - DX_i$'을 계산한 것이다. 상태-7과 상태-8은 $X_n = X_f * \{U : V\} => \{U : V\}$를 계산하는 것으로 상태-5 및 상태-6와 유사하다.

제시한 알고리즘은 C 언어로 모델링하였다. SHA 해쉬 함수[10]를 사용하여 D 10^7 개를 생성하고, 제시한 알고리즘으로 역수를 계산하고, 그 결과를 SRT로 계산한 결과와 비교하여 일치하는 것을 확인하였다.

IBM-PC의 Window-7에서 Icarus Verilog를 사용하여 Verilog HDL로 코딩하고 시뮬레이션하여 동작을 확인하였다.

4. 연구 결과 및 분석

DasSarma[11]의 연구 결과 최적의 근사 역수는 식(10)으로 주어진다.

$$T(g) = \frac{1}{1.g} \fallingdotseq RN\left(\frac{1}{1.g + 2^{-2n_g - 1}}\right) \quad (10)$$

RN is round to nearest

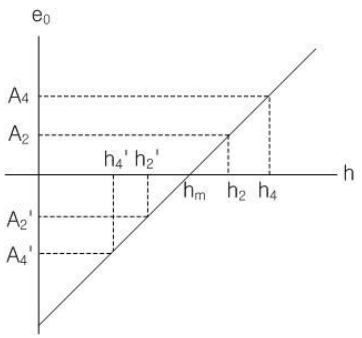
T(g)의 소수점 이하 길이를 t 비트라고 하면 '$T(g) = (b_0.b_1,...,b_t)_2$, $0.5 < T(g) \leq 1.0$. '$b_0b_1 = 10$'인 경우는 '$g = 0$'일 때이다. 이외의 경우는 항상 '$b_0b_1 = 01$'이다. 그러므로 근사 역수 테이블에 '$b_2,...b_t$'만을 저장하면 된다. 따라서 근사 역수 테이블의 크기는

'$2^{n_y} * (t-1)$' 비트가 되어서, 테이블의 길이는 2^{n_y}이며, 폭은 't-1' 비트이다.

T(g)에서 초기 오차 e_0는 식 (11)이 된다.

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

식 (11)로부터 e_0는 $h_m = 100...0$에서 가장 작으며, $h_z = 000...0$과 $h_{\max} = 111...1$에서 가장 커서 Fig. 1과 같이 된다.



[Fig 1] h versus e_0

'$a_0^2 < 2^{-p+3}$'이면 2회의 곱셈으로 근사 역수를 계산할 수 있다. 즉, 식 (12)가 성립하면 2회의 곱셈으로 근사 역수를 계산할 수 있다.

$$e_0 < A_2 = \frac{2^{-p} + 2^{-\frac{p-3}{2}}}{1.g+h_{\max}} \tag{12}$$

식 (12)에서 A_2가 최소가 되는 값을 선택했다. 초기 오차 e_0는 양수와 음수의 두 가지 값을 가지며, Fig. 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 X_f를 계산할 수 있다.

$a_1 = a_0^2$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. $h_4' < h < h_2'$와 $h_2 < h < h_4$에서 4회의 곱셈으로 X_f를 계산할 수 있다. 이와 같은 계산을 계속하여 수행하면 초기 오차 e_0에 따라서 D의 근사역수 X_n을 계산하기 위한 곱셈의 횟수를 산출할 수 있다.

본 논문에서 제안한 알고리즘에 의한 테이블 크기에 따른 IEEE 배정도실수의 역수 계산에 필요한 곱셈 횟수를 Table 3에 보인다.

[Table 3] Average number of multiplication to calculate double precision floating point reciprocal

Table size	Average No. of Multiply
16 X 3	9.66
32 X 4	9.32
64 X 5	8.71
128 X 6	8.00
256 X 7	7.96

종래 골드스미스 알고리즘에서는 최대 오차를 고려해서 반복 횟수를 정했다. 64 비트 곱셈기를 사용한 종래의 골드스미스 알고리즘에서 배정도실수의 역수를 구하려면 '64x5' 테이블에서 8회, '128x7' 테이블에서 6회의 곱셈이 필요하다. 또한 64 비트 곱셈을 32 비트 곱셈기 3회로 계산하는 경우에는 종래의 골드스미스 알고리즘에서 배정도실수의 역수를 구하려면 '64x5' 테이블에서 28회, '128x7' 테이블에서 22회의 곱셈이 필요하다. 본 논문에서 제안한 알고리즘은 Table 3으로부터 32 비트 곱셈기와 '128x6' 테이블을 사용하면 평균 8회의 곱셈으로 배정도실수의 역수를 계산할 수 있다.

5. 향후 연구 과제

골드스미스 알고리즘은 식-6에서 X_{i+1}과 a_{i+1}을 두 개의 곱셈기를 사용하여 병렬로 계산할 수 있다. Ozbilen[7]은 64 비트 곱셈기를 SIMD 형식으로 2 개의 32 비트 곱셈기로 나눈다. 배정도실수 계산에서는 64 비트 곱셈기로, 단정도실수 계산에서는 두 개의 32 비트 곱셈기로 사용하여 골드스미스 알고리즘을 적용했다.

Dimitri 등[12-13]은 '76 bit X 27 bit' 곱셈기를 사용하여 배정도실수 계산을 했고, 이를 2개로 나누어서 단정도실수 계산을 수행했다.

본 논문에서 제안한 알고리즘은 X_f를 구하는 반복연산은 단정도실수 계산에 해당하므로 2 개의 짧은 곱셈기로 병렬 계산하고, X_f로부터 X_n을 구하는 것은 긴 계산기로 1회 연산을 수행하도록 확장하는 연구를 향후 과제로 제시한다.

6. 결론

최근 그래픽 프로세서, 멀티미디어 프로세서, 음성처리 프로세서 등 실장제어분야에서 부동소수점 계산은 빠른 계산과 저전력이 요구되면서 부동소수점 연산을 하드웨어로 구현하고 있다. 이들 분야에서는 높은 정밀도를 요구하지 않으므로 단정도실수 연산이 대부분이며 또한 근사 값만으로도 충분하다. 한편 C, Java 등 고급언어는 배정도실수와 단정도실수 모두를 사용한다. 배정도실수 나눗셈은 실장제어분야에서는 배정도실수의 사용빈도가 낮으므로 단정도실수 연산에 추가적인 부담이 작으면서도 성능이 좋은 배정도실수 연산이 요구된다.

본 논문에서는 IEEE 배정도실수 D의 가수부 53 비트의 상위 28 비트의 역수 X_f를 32 비트 곱셈기를 사용하여 골드스미스 역수 알고리즘으로 계산하고, X_f를 초기값으로 하여 골드스미스 역수 알고리즘으로 D의 역수를 계산한다. 그리고 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 수행하는 알고리즘을 제안하였다. 제안한 알고리즘은 C 모델링하여 정확한 계산이 산출되는 것을 검증하였고, Verilog HDL로 하드웨어로 구현하여 로직 시뮬레이션하여 동작을 검증하였다.

본 논문에서 제안한 알고리즘은 32 비트 곱셈기와 '128x6' 근사 테이블을 사용하면 평균 8회의 곱셈으로 배정도실수의 역수를 계산할 수 있다. 한편 종래 골드스미스 알고리즘에서는 32 비트 곱셈기와 '128x7' 근사 테이블에서 22회의 곱셈이 필요하다.

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, 2002. DOI: http://dx.doi.org/10.1109/TCSVT.2002.800865

[2] R. B., Lee, "Multimedia extensions for general purpose processor," Signal Processing Systems, SIPS 97 - Design and Implementation., IEEE Workshop, pp. 9-23, 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. DOI: http://dx.doi.org/10.1109/ARITH.1993.378106

[4] D. L. Harris, S. F. Oberman, and M. A. Horowitz, "SRT Division Architectures and Implementations," Proc. 13th IEEE Symp. Computer Arithmetic, Jul. 1997. DOI: http://dx.doi.org/10.1109/ARITH.1997.614875

[5] W. F. Wong, et al, " Fast Hardware-Based Algorithms for Elementary FUnction Computations Using Rectangular Multiplier," IEEE Transactions on Computers, Vol. 43, No. 3, pp. 278-294, Mar. 1994.

[6] T. Brightman, "Advancing the standard in floating point performance," High Perform., Syst., pp 59-64. Nov. 1989.

[7] Metin Mete Ozbilen, Mustafa Gok, "A Single/Double Precision Floating-Point Reciprocal Unit Design for Multimedia Applications," International Conference on Electrical and Electronics Engineering, 2009, Vol. 2, pp 352-356, Nov. 2009.

[8] Sung-Gi Kim, Hing-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, Feb. 2005.

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

[10] Secure Hash Standards, Federal Information Processing Standards Publication 180-3, http://www.itl.nist.gov/fipspubs, Oct. 2008.

[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.
DOI: http://dx.doi.org/10.1109/12.295855

[12] Dimitri Tan, et al, "Low-Power Multiple-Precision Iterative Floating-Point Multiplier with SIMD Support," IEEE Transactions on Computers, Vol. 58, No. 2, pp. 175-187, Feb. 2009.
DOI: http://dx.doi.org/10.1109/TC.2008.203

[13] Michael J. Schulte, et al, "Floating-Point Division Algorithms for an x86 Microprocessor with a Rectangular Multiplier," Proceeding IEEE International Conference Computer Design(ICCD'07), pp. 304-310, Oct. 2007.

조 경 연(Gyeong-Yeon Cho) **[정회원]**

- 1990년 2월 : 인하대학교 전자공학과 박사
- 1983년 3월 ~ 1991년 2월 : 삼보컴퓨터 기술연구소 책임연구원
- 1998년 1월 ~ 현재 : 에이디칩스(주) 기술고문
- 1991년 3월 ~ 현재 : 부경대학교 공과대학 IT융합응용공학과 교수

<관심분야>
컴퓨터구조, 반도체회로 설계, 암호 알고리즘