2-Stage Pipeline 구조를 이용한 역제곱근 연산기의 설계

김정훈, 김기철
서울시립대학교 전자전기컴퓨터공학부
kjhsharf@naver.com, kkim@uos.ac.kr

Design of Inverse Square Root Unit Using 2-Stage Pipeline Architecture

Junghoon Kim, Kichul Kim
School of Electrical & Computer Engineering, University of Seoul

요 약

본 논문에서는 변형된 Newton-Raphson 알고리즘과 LUT(Look Up Table)를 사용하는 역제곱근 연산기를 제안한다. Newton-Raphson 부동소수점 역수 알고리즘은 일정한 횟수의 곱셈을 반복하여 역수 제곱근을 계산하는 방식이다. 변형된 Newton-Raphson 알고리즘은 하드웨어 구현에 적합하도록 변환되었으며, LUT는 오차를 줄이기 위해 개선되었다. 제안된 연산기는 LUT의 크기를 최소화하고, 순환적인 구조가 아닌 2-stage pipeline 구조를 가진다. 또한 IEEE-754 부동소수점 표준을 기초로 하는 24-bit 데이터 형식을 사용해 면적과 속도 향상에 유리하여 휴대용 기기의 멀티미디어 분야의 응용에 적합하다.

본 역제곱근 연산기는 소수점 이하 8-bit의 정확도를 가지며 VHDL을 이용하여 설계되었다. 그 크기는 0.18um CMOS 공정에서 약 4,000 gate의 크기를 보였으며 150MHz에서 동작이 가능하다.

1. 서 론

부동소수점 연산기(Floating Point Unit)는 디지털 신호처리, 컴퓨터 그래픽스, 과학 기술 계산, CAD 등에서 쓰이는 필수적 요소이다. 이에 따라 부동소수점 연산기를 사용하는 CPU가 점차 늘어나고 있는 추세이다. 또한 최근 음성이나 3차원 영상 같은 멀티미디어 분야에서 부동소수점 연산의 필요성이 대두됨에 따라 휴대용 통신 기기를 포함한 SoC(System On Chip)에서도 부동소수점 연산기가 사용되고 있다. 실제로 Motorola의 Altivec에는 역제곱근 계산의 필요성 때문에 이에 대한 특수 명령어들이 추가되기도 했다. 이처럼 휴대용 기기에서의 제곱근이나 역제곱근 연산기에 대한 연구는 멀티미디어 분야의 시장이 계속적으로 성장함에 따라 그 필요성이 증가하고 있다[1].

따라서 본 논문에서는 휴대용 기기의 3차원 영상처리 및 멀티미디어 분야의 응용에 적합한 역제곱근 연산기를 제안한다. 휴대용 기기에서의 응용을 위해서는 빠른 연산 속도와 작은 면적의 하드웨어, 그리고 적절한 정확도를 만족시켜야 한다. 이를 위해 제안된 역제곱근 연산기는 최적의 근사 값을 가지며 크기를 최소화한 LUT(Look Up Table)를 가지고, 순환회로가 아닌 2-stage pipeline 구조를 취함으로써 면적과 연산 속도, 정확도면에서 장점을 취한다.

본 논문의 구성은 다음과 같다. 2장에서는 역제곱근을 계산할 수 있는 기존의 알고리즘들에 관해 살펴본다. 3장에서는 본 역제곱근 연산기에 쓰이는 변형된 Newton-Raphson 알고리즘에 대해 설명하고, 4장에서는 설계된 역제곱근 연산기의 성능을 보여준다. 5장에서 역제곱근 연산기의 구조에 대해 설명하며, 마지막으로 6장에서 본 논문의 결론을 맺는다.

2. 기존 역제곱근 알고리즘

부동소수점 제곱근이나 역제곱근의 계산을 위한 알고리즘에는 크게 SRT 알고리즘과 Newton-Raphson 알고리즘, Goldschmidt 알고리즘 등이 있다. 이 알고리즘들은 크게 반복 알고리즘과 수렴 알고리즘으로 나뉜다. 반복 알고리즘은 다른 방법들보다 적은 면적으로 구현이 가능하지만, 선형적인 수렴 단계를 거치게 되며 많은 반복 횟수를 요구한다. 반면에 수렴 알고리즘은 짧은 지연시간을 갖고 있지만 많은 양의 메모리와 면적을 요구한다.

2.1 SRT 알고리즘

SRT 알고리즘은 직접법의 일종인데, 직접법은 사람이 종이와 연필을 가지고 계산하는 방식과 유사하게 덧셈, 뺄셈과 쉬프트(shift)를 반복하는 방식이다. SRT 알고리즘은 덧셈(뺄셈), 부분 곱셈, 몫 결정의 3단계 과정을 반복하는 알고리즘으로 매 반복마다 일정한 비트의 몫을 얻을 수 있다.

2.2 Newton-Raphson 알고리즘

Newton-Raphson 알고리즘은 곱셈을 반복하여 역제곱근을 구하는 방식이다. 이 방식은 x와 $1/\sqrt{x}$ 의 근사치가 주어졌을 때 Newton-Raphson 반복식을 수행하여 $1/\sqrt{x}$ 을 구하는 방식으로써, $1/\sqrt{x}$ 의 초기 근사치로부터 정확한 값으로 수렴하는 특성을 갖는다.

2.3 Goldschmidt 알고리즘

Goldschmidt 알고리즘 역시 곱셈을 반복하여 제곱근을 구한다. $\sqrt{F}$ 를 구하기 위해 초기값을 $X_0 = Y_0 = F$ 로 설정한다. 그리고 $X_{i+1} = X_i \times R_i^2$ 식과 $Y_{i+1} = Y_i \times R_i$ 식을 반복하면 $\frac{X_{i+1}}{Y_{i+1}^2} = \frac{X_i}{Y_i^2} = \frac{1}{F}$ 를 얻을 수 있다. $R_i = \frac{3 - e_r - X_i}{2}$ 라고 하면 X_i 는 1로 수렴하고, 이에 따라 Y_i는 $\sqrt{F}$ 의 근사값을 얻을 수 있다[2].

위 알고리즘 중 SRT 알고리즘은 하드웨어가 간단하고 정밀 계산이 가능하지만, 데이터의 길이가 길어질수록 지연시간이 선형적으로 증가하는 단점이 있다. 곱셈을 반복하는 방식의 Newton-Raphson 알고리즘과 Goldschmidt 알고리즘은 곱셈기와 LUT를 사용하여 반복할 때마다 오차가 자승에 비례해서 줄어들므로 연산 속도가 빠른 장점이 있지만, 근사 계산만이 가능하다. 그러나 소수의 정밀 과학 기술 연산 분야를 제외하고 대부분 멀티미디어 같은 분야에서는 근사계산만으로 실용상 문제가 없다. 이 중 Goldschmidt 알고리즘은 매 반복 연산마다 독립적인 두 번의 곱셈을 수행해야 하지만, Newton-Raphson 알고리즘은 두 번의 곱셈이 순차적으로 이루어지므로 pipeline 구조에 더 적합함을 알 수 있다[3].

3. 변형된 Newton-Raphson 알고리즘

설계된 역제곱근 연산기는 Newton-Raphson 알고리즘을 사용하는데, 이 알고리즘은 Look Up Table(LUT)을 참고해 곱셈을 반복하여 계산하는 방식이다. 이는 빠른 연산 속도를 가지지만 정확한 결과값을 얻기 위해서는 순환 회로를 구성해야 하는 단점이 있다. 따라서 본 역제곱근 연산기는 빠른 연산 속도를 취하기 위해 순환 회로를 구성하지 않고, 대신 변형된 Newton-Raphson 알고리즘과 LUT을 사용해 정확도를 높이는 방법을 채택하였다. Newton-Raphson 알고리즘과 같은 수렴 알고리즘을 사용하면 많은 양의 LUT가 필요하므로, 연산 결과의 정확도를 고려해 적절한 크기와 적절한 값을 갖는 LUT를 구성하는 것이 역제곱근 연산기 설계에 있어서 관건이 된다.

역제곱근을 구하기 위한 기본 개념은 X의 역제곱근 값을 구하기 위한 근사 값인 Y를 구하는 것이다. Y를 계산하기 위한 방법은 다음과 같은 단계를 거친다.

가. 최초 근사값 $R \approx \frac{1}{\sqrt{X}}$ 을 계산한다.

나. 더욱 정확한 근사값인 $Y \approx \frac{1}{\sqrt{X}}$ 를 구하기 위해 변형된 Newton-Raphson 알고리즘을 이용한다.

초기 근사값 R_0를 계산한 후에 변형된 Newton-Raphson 알고리즘으로 더욱 정확도를 높인다. 입력 X에 대한 Newton-Raphson 알고리즘은 식 (1)과 같이 나타낼 수 있다.

$$R_{i+1} = \frac{R_i(3 - XR_i^2)}{2},\ i \in \{0,\ 1,\ 2,\ .....n-1\} \qquad (1)$$

위의 식에서 R_i는 i번째 근사값을 나타내고 R_{i+1}은 i+1번째의 조금 더 정확한 근사값이 된다. 또한 R_0는 임의의 초기 근사값이다. 이 식을 그대로 구현하기 위해서는 3개의 가산기, 1개의 감산기 그리고 1-bit의 오른쪽 쉬프트 유닛이 필요하다. 따라서 이 식을 하드웨어에서 좀 더 효율적으로 구현하기 위해 아래와 같이 변형된 Newton-Raphson 알고리즘을 사용한다[4]. 식 (2)는 3개의 곱셈기와 1개의 가산기 그리고 1-bit의 쉬프트 유닛이 필요하다.

$$W = R^2,\ D = 1 - WX,\ Y = R + \frac{RD}{2} \qquad (2)$$

위의 식들을 구현함에 있어서 하드웨어의 복잡도를 고려해 $D = 1 - WX$ 식을 감산기를 사용하여 계산하지 않고 WX 를 1의 보수로 표현해 뺄셈을 계산한다. 이로 인해 정확도에서 약간의 손실을 가져올 수 있지만, 감산기를 사용하지 않고 1의 보수화 유닛을 사용함으로써 하드웨어가 간단해지는 효과를 갖는다.

설계된 역제곱근 연산기는 2-stage pipeline 구조를 가지며, 2개의 LUT와 2개의 곱셈기, 1개의 덧셈기 등을 사용하고 있다. 초기 근사값을 위한 2개의 LUT들은 입력 $1 \le X < 2$에 대해 아래의 식 (3)을 기초로 R_0와 R_0^2에 해당하는 데이터를 저장하고 있다. 연산 결과의 정확도를 위해 식 (3)에 따라 계산된 값을 기초로

추가로 시뮬레이션을 거쳐 좀 더 높은 정확도를 가지는 LUT를 구성하였다.

$$R_0 = \frac{1}{\frac{2X}{3} + 0.354167} \quad (3)$$

위의 R_0^2 LUT를 이용해 첫 번째 곱셈기에서 변형된 Newton-Raphson 알고리즘의 'WX'를 계산한다. 1의 보수화 유닛(Unit)은 이를 이용해 'D'를 계산한다. 두 번째 곱셈기는 1-bit 오른쪽 쉬프트 기능을 갖고 있으며 R_0 LUT를 이용해 '$\frac{RD}{2}$'를 계산한다. 마지막으로 덧셈기에서 R_0 LUT를 이용해 가수 부분을 나타내는 'Y'를 계산한다. 지수 부분은 역제곱근 연산기 내부의 지수 계산부에서 'Y'값을 참고해 계산한다.

4. 역제곱근 연산기의 구조

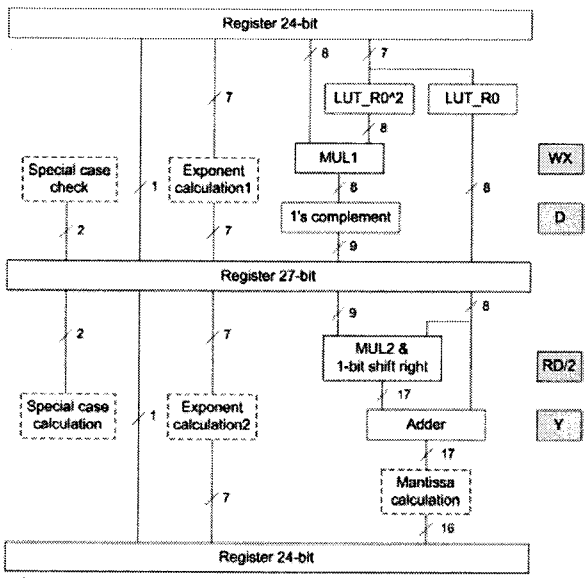
본 역제곱근 연산기는 IEEE-754 32-bit 부동소수점 형식을 토대로 한 24-bit 부동 소수점 데이터 형식을 사용하는데, 그 연산체계는 (그림 1)과 같다. 24-bit 부동 소수점 데이터 형식은 부호부는 1-bit로 표현을 하고, 지수부는 7-bit로 '-62~63'의 크기를 가지며, 가수부는 16-bit의 크기를 갖는다. 이 같은 24-bit 부동 소수점 데이터 형식은 기존 32-bit 형식에 비해 작은 데이터 단위를 사용하므로 연산 처리에 있어서 속도를 향상시킬 수 있다.

(그림 1) 24-bit 부동소수점 데이터 형식

설계된 역제곱근 연산기는 모바일 기기의 특성상 아주 높은 정확도 보다는 저전력과 저비용의 효과를 취하기 위해 2-stage pipeline 구조로 되어있으며, 각 유닛들의 크기는 다음과 같다. 먼저 두 개의 LUT는 각각 128개의 8-bit 데이터를 갖고 있어 각 128-byte, 총 256-byte의 크기를 갖는다. 각 8-bit 데이터는 1≤X<2 입력에 대한 역제곱근의 근사값으로 이진수 '1.xxxxxxx'의 형식을 갖는다. 첫 번째 곱셈기는 두 개의 8-bit 입력을 받고, 두 번째 곱셈기는 9-bit와 8-bit의 입력을 받는다. 1의 보수화 유닛은 8-bit의 입력을 받고, 덧셈기는 17-bit와 8-bit의 입력을 받는다. 첫 번째 stage의 레지스터는 28-bit이다. (그림 2)는 역제곱근 연산기의 구조를 나타내고 있다.

(그림 2)에서 오른쪽의 회색 박스는 변형된 Newton-Raphson 알고리즘의 각 단계가 설계된 역제곱근 연산기의 어느 유닛에서 연산이 되는지를 보여주고 있다. 첫 번째 stage의 곱셈기에서 WX가 연산되고, 이 출력이 1의 보수화 유닛의 입력으로 연결되어 $D = 1 - WX$가 연산된다. 두 번째 stage의 곱셈기에서는 $\frac{RD}{2}$가 연산되고, 이 값이 덧셈기를 거쳐 $Y = R + \frac{RD}{2}$가 연산된다.

설계된 역제곱근 연산기의 출력은 입력 1≤X<2의 범위에 대한 역제곱근 근사값으로 0.7<Y≤1 범위의 값이다.



(그림 2) 제안된 2-stage pipeline 역제곱근기의 구조

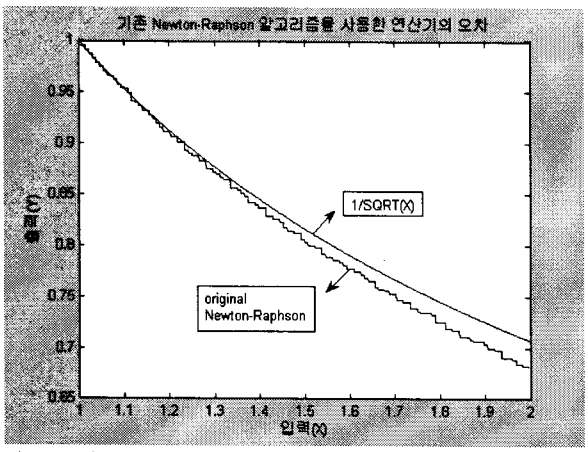
5. 성 능

제안된 역제곱근 연산기의 성능을 평가하기 위하여 동일한 하드웨어 구조에서 기존 Newton-Raphson 알고리즘을 이용한 연산 결과와 변형된 Newton-Raphson 알고리즘과 LUT를 이용한 연산 결과를 비교하였다.

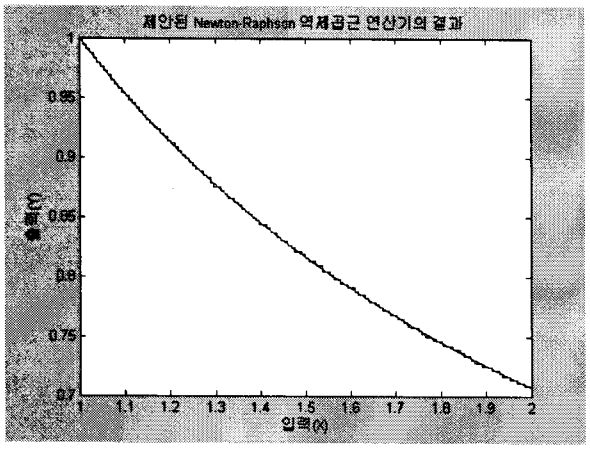
정확도 측면에서 비교하기 위하여 Matlab을 사용하여 1≤X<2 범위의 1,024개 입력에 대한 연산 결과를 이상적인 역제곱근 값과 비교하여 오차 범위를 확인하였다. 그래프에서 X축은 1≤X<2 범위의 입력을 나타내고, Y축은 입력에 대한 역제곱근 값을 나타낸다. (그림 3)은 기존 Newton-Raphson 알고리즘을 이용한 역제곱근 연산기의 연산 결과와 이상적인 역제곱근

값과의 비교 결과를 나타낸다. 기존 Newton-Raphson 알고리즘을 이용한 연산 결과는 이상적인 역제곱근 값에 대해 이진수 상에서 소수점 이하 5-bit까지의 정확도를 만족하였다.

(그림 4)는 본 논문에서 제안된 변형된 Newton-Raphson 알고리즘과 LUT를 이용한 역제곱근 연산기의 연산 결과와 이상적인 역제곱근 값과의 비교 결과를 나타낸다. 변형된 Newton-Raphson 알고리즘과 LUT를 이용한 연산 결과는 이상적인 역제곱근 값에 대해 이진수 상에서 소수점 이하 8-bit까지의 정확도를 만족한다.



(그림 3) 기존 Newton-Raphson 역제곱근기의 결과



(그림 4) 제안된 Newton-Raphson 역제곱근기의 결과

위 검증 결과에서 보여지듯이 본 논문에서 제안된 변형된 Newton-Raphson 알고리즘과 LUT를 이용한 역제곱근 연산기의 연산 결과가 기존의 Newton-Raphson 알고리즘을 이용한 연산 결과보다 높은 정확도를 갖는 것을 알 수 있다.

제안된 역제곱근 연산기는 Mentor사의 Modelsim을 사용하여 검증하였고 Synopsys사의 Design Compiler를 사용하여 합성하였다. 합성 결과는 0.18um CMOS 공정에서 합성하였을 때 약 4,000 gate의 크기를 보였고, 150MHz에서의 동작이 가능하다.

6. 결 론

본 논문에서는 변형된 Newton-Raphson 알고리즘과 LUT를 이용하여 휴대용 기기의 3차원 영상 처리를 포함한 멀티미디어 분야에 적합한 역제곱근 연산기를 제안하였다. 본 역제곱근 연산기는 변형된 Newton-Raphson 알고리즘과 LUT를 사용하여 기존 Newton-Raphson 알고리즘을 사용했을 때 보다 높은 정확도를 갖게 되어 이진수 상에서 소수점 이하 8-bit까지의 정확도를 만족한다. 또한 IEEE-754 부동 소수점 표준을 기초로 하는 24-bit 데이터 형식을 기반으로 설계되어 기존 32-bit 데이터 형식에 비해 연산 속도를 향상시킬 수 있다. 또한 2개의 LUT, 2개의 곱셈기, 1개의 덧셈기 등의 간단한 구조로 이루어진 2-stage의 pipeline 방식을 취하여 저비용과 저전력의 특징을 가진다.

연산기의 크기는 0.18um CMOS 공정에서 합성하였을 때 약 4,000 gate의 크기를 가졌으며, 150MHz에서 동작이 가능하다.

향후 연구에서는 본 논문의 연산기가 보다 높은 정확도를 필요로 하는 응용분야에서 사용 가능하도록 연산의 반복횟수 혹은 LUT의 크기를 늘리거나, Newton-Raphson 알고리즘 자체를 개선시키는 연구가 진행되어야 할 것이다.

참고문헌

[1] 김성기, 조경연, *"가변 시간 뉴톤-랍손 부동소수점 역수 제곱근 계산기"*, 정보처리학회논문지, 제12-A권, 제5호, pp.413~420, 2005.

[2] Peter Soderquist, Miriam Leeser, *"An Area/Performance Comparison of Subtractive and Multiplicative Divide/Square Root Implementations"*, IEEE Proc. of 12th Symp. on Computer Arithmetic, pp.132~139, 1995.

[3] 정우경, *"고성능 내장형 마이크로프로세서를 위한 SIMD-DSP/FPU 설계"*, 박사학위논문, 연세대학교, 2002.

[4] Michael J. Schulte, Kent E. Wires, *"High-Speed Inverse Square Roots"*, IEEE Proc. of 14th Symp. on Computer Arithmetic, pp.124~131, 1999.