Taylor 전개식을 이용한 나눗셈 연산기의 VLSI 설계

정회원 권 호 경*, 문 상 국**, 문 병 인**, 이 용 석**

VLSI Design of a Divider Using Taylor-series Expansion

Ho-Kyung Kwon*, Sangook Moon**, Byung-In Moon**, Yong-Surk Lee** *Regular Members*

요 약

본 논문에서는 IEEE-754 부동소수점 표준을 지원하는 나눗셈 연산기를 Verilog HDL(Hardware Description Language)을 사용하여 설계하고 그 기능을 검증하였다. 고속의 나눗셈 연산을 위해 Taylor 전개식을 이용한 역수(reciprocal) 알고리즘으로 구현하였다.

역수 알고리즘에서 사용하는 롬 테이블(ROM)을 두 개의 작은 롬 테이블로 분리, 구현하여 면적을 줄여서 성능을 향상시켰다. 또, 제어 신호를 발생하는 로직을 마이크로제어롬(microcontrolled ROM)이 아닌 간단한 조합 회로로 구현함으로써 임계 경로(critical path)를 단축시켰다.

명령어 해석기와 연결된 신호를 통해서 비록 나눗셈 연산중이더라도 사용중이지 않은 기능 유닛을 사용하는 명령어가 이슈될 수 있도록 하여 나눗셈 연산으로 인한 프로세서의 전체 성능 저하를 막도록 하였고 비순차 처리를 지원하는 수퍼스칼라 구조에서도 사용할 수 있는 아키텍쳐를 제안하였다.

하드웨어 측면에서는 일반적인 부동 소수점 유닛에 존재하는 기능 유닛인 곱셈기와 덧셈기를 약간의 수정을 통해서 사용하도록 하여 하드웨어 공유를 통해 면적에 있어서 잇점을 가지도록 하였다.

이와 같이 구현된 부동소수점 나눗셈 연산기의 성능은 매 iteration당 19비트 이상의 정밀도로 몫과 나머지를 얻을 수 있으며, 결과적으로 IEEE-754 표준안의 단정도 연산시, 몫을 구할 때까지는 12사이클, 나머지를 구할 때까지는 14사이클이 소요되며, IEEE-754 표준안의 배정도 연산시, 몫을 구할 때까지는 17사이클, 나머지를 구할 때까지는 19사이클이 소요되었다.

ABSTRACT

This paper proposes a VLSI design of a divider supporting IEEE-754 floating point standard. It is implemented using a Taylor-series-reciprocal algorithm for high speed division operations. Its functions are modeled and verified with Verilog HDL (Hardware Description Language).

To avoid the weakness of the reciprocal algorithm, the ROM table is divided into two smaller size ROM tables. Although it is implemented with two smaller size ROMs, run-time calculation scheme is adopted to reach the same performance. In order to shorten the critical path delay, a control unit is implemented with combinational logics instead of micro-controlled ROM.

With the help of the control unit, the instruction decoder can determine when an instruction which is expected to use a shared functional unit should be issued. Therefore, this architecture can prevent the processor from degrading the performance and it can be adopted for superscalar processor without any modification.

The proposed divider benefits from reusing the general floating-point multiplier and floating-point adder in common FPUs so that the overhead of additional hardware is minimized. Besides, division process is made simpler by a state machine which generates control signals necessary in each stage.

* 현대전자산업 MCU 설계팀(hkkwon@hei.co.kr), ** 연세대학교 전기전자공학과 프로세서 연구실(lizking@dubiki.yonsei.ac.kr)
논문번호 : 00144-0501, 접수일자 : 2000년 5월 1일

The proposed divider produces a quotient with at least 19bits per iteration and requires 14 cycles for single precision calculation and 19 cycles for double precision.

Ⅰ. 서 론

오늘날의 컴퓨터 환경은 음성 및 영상 처리, 그래픽 및 비디오/오디오를 통합한 멀티미디어, 네트워크 분야로 급속하게 발전하고 있으며, 이러한 기능들은 모두 고도의 정수(integer), 부동소수점(floating-point), 그래픽 처리 능력을 필요로 하고 있다. 특히, 부동소수점 유닛(floating-point unit)은 프로세서에게 수치 연산 능력을 제공해 주는 고성능의 수치 연산 처리부로서, SPICE와 같은 전자회로 시뮬레이션, CAD(Computer Aided Design)와 같은 그래픽 응용, 실시간의 정확성으로 기계를 제어하는 컴퓨터 수치 제어, DSP(Digital Signal Processing) 등 수많은 특정 수치 응용에서 사용될 수 있다. 이러한 추세를 반영하여 오늘날 개발되는 대부분의 프로세서는 부동소수점 유닛을 칩(chip) 내에 내장하여 성능을 대폭 향상시키고 있다.

부동소수점 유닛은 덧셈, 곱셈, 나눗셈, 제곱근 등에 대한 산술 연산을 수행하는데, 특히 나눗셈 및 제곱근 연산은 다른 덧셈이나 곱셈 연산과는 달리 나눗셈 연산을 수행하는 데에는 많은 클럭 사이클을 요구하기 때문에 성능 저하의 원인이 된다[2][3]. 이러한 성능 저하를 막기 위한 나눗셈 연산 유닛을 구현하기 위한 다양한 방법이 존재하고[2][3] 연구되었으며 본 논문에서도 이에 대한 대안을 제시하였다.

고속의 부동소수점 제산 연산기를 구현하는 방안은 (표 1-1)과 같이 크게 두 가지 부류로 나뉠 수 있다. 첫 번째 알고리즘은 덧셈/뺄셈과 쉬프트 동작을 이용하여 나눗셈을 수행하는 알고리즘으로, 여기에는 복원(restoring) 알고리즘, 비복원(non-restoring) 알고리즘, SRT 알고리즘이 이에 속한다. 두 번째 알고리즘은 곱셈을 이용하여 나눗셈을 수행하는 방식으로, 여기에는 수렴 알고리즘과 역수 알고리즘이 이에 속한다.

첫 번째 알고리즘에 속하는 복원 알고리즘과 비복원 알고리즘, SRT 알고리즘에 대해서 간략히 살펴보면 다음과 같다. 복원 알고리즘은 나눗셈 연산 도중에 나머지의 부호에 따라 피젯수의 복원 동작이 필요하며, 나머지가 완전히 결정되기 전에는 다음 동작을 시작할 수 없기 때문에 파이프라인 처리가 어렵고, 몫을 결정하기 위해서는 제수와 피젯수의 모든 비트를 비교해야 하므로 처리 속도가 느리다. 이러한 문제점을 개선한 알고리즘으로는 복원 동작을 배제함으로써 속도를 향상시킨 비복원 알고리즘이 있으며, 이 알고리즘에서는 몫 선택시에 제수와 피젯수의 모든 비트에 대한 완전한 비교 동작이 요구되지 않는다. 하지만 덧셈 및 뺄셈 동작이 모두 필요하며 SRT 알고리즘에서처럼 redundancy가 존재하지 않기 때문에 radix 증가를 통한 성능 향상이 어렵고 나눗셈 연산 후에 추가의 수정 동작이 요구되는 단점이 있다. 덧셈/뺄셈과 쉬프트 알고리즘에 속하면서 성능이 우수하여 현재 가장 널리 구현되어 사용되고 있는 알고리즘은 SRT 알고리즘이다. 이 알고리즘은 크게 보면 비복원 알고리즘에 속한다고 할 수 있으며, 숫자 체계에 redundancy를 두어 radix를 증가하여 계산할 때 발생하는 오차를 수정할 수 있도록 하여 성능을 향상시키는 방식이다. 그러나 PLA를 사용하여 몫을 결정하는 과정이 임계 경로에 포함되기 때문에 PLA의 속도가 동작 주파수를 결정하는 중요한 요소가 된다. 이러한 문제점을 피하기 위해서 일부 상용화된 프로세서의 경우, 발생 가능한 나머지를 모두 구하는 방법을 사용하여 동작 주파수를 높이고 있지만 상당한 면적의 증가를 감수해야 한다. 기본적으로는 덧셈/뺄셈과 쉬프트 알고리즘에 바탕을 두기 때문에 정밀도가 높지만 알고리즘 자체의 iteration의 횟수가 많다. 따라서 속도보다는 정밀도가 중요한 요소인 범용 프로세서에서는 주로 이 알고리즘을 많이 사용하다.

두 번째 알고리즘에 속하는 수렴 알고리즘과 역수 알고리즘에 대해서 간략히 살펴보면 다음과 같다. 수렴 기법을 이용한 알고리즘은 피젯수와 제수에 동일한 수를 계속 곱하되 분모를 1로 수렴시켜서 몫을 얻는 방식으로, 나머지를 얻기 위해서는 다른 별도의 연산이 필요하다. 역수 알고리즘은 제수의 역수를 계산하고 이를 피젯수와 곱하여 몫을 구하는 알고리즘으로 NR(Newton-Raphson) 방식을 사용하여 역수를 구하는 방법이 가장 널리 알려져 있다. NR 방식을 사용하여 구현할 경우, 구현 방법에 따라서 나머지를 구하기 위한 별도의 연산이 필요하기도 하다. 두 경우 모두 이론적으로는 몫의 정밀도가 산술 급수적으로 수렴(quadratic convergence)하기 때문에 iteration의 횟수가 작다. 하지만 실제로 구현할 때, 너무 많이 비트 수의 곱셈기와 덧셈

기를 필요로 하기 때문에 정밀도를 다소 희생하더라도 선형 급수적으로 수렴하도록 설계되고 있다. 또, iteration의 횟수는 적지만 매 iteration마다 여러 번의 곱셈과 덧셈/뺄셈을 수행하야 하기 때문에 나눗셈 연산을 수행하기 위해서는 많은 사이클 수를 필요로 한다. 이러한 문제점을 해결하기 위해서 초기 iteration은 lookup ROM table을 이용하여 구하고 그 이상의 정밀도는 실시간으로 계산하여 사이클 수를 줄이도록 설계하고 있다. 기능 유닛의 크기를 줄이기 위해서 정밀도를 희생하였기 때문에 정밀도가 첫 번째 알고리즘보다 낮아서 IEEE-754 표준안을 만족시키기 어려운 것으로 알려져 있다. 역수 알고리즘을 사용한 프로세서에서는 정밀도를 높이는 것이 중요한 이슈가 되며, 같은 기능 유닛을 사용하더라도 iteration의 횟수를 늘려서 정밀도를 높이기도 한다. 따라서 약간의 오차가 허용되며 실시간 처리와 같이 속도를 요구하는 시그널 프로세서에서는 주로 이 알고리즘을 사용한다.

본 논문에서는 Taylor 전개식을 이용한 역수 알고리즘(reciprocal algorithm)을 사용하여 고속으로 연산하면서도 IEEE-754 부동소수점 표준안을 만족시키는 정확한 몫과 나머지를 구할 수 있어 위에 제시된 두 가지 나눗셈 방식의 서로의 장점을 가지는 알고리즘을 제시하고 이를 하드웨어적으로 구현하여 성능을 평가한다. 본 논문의 2장에서는 역수 알고리즘에 대하여 설명한다. 이러한 알고리즘에 의한 하드웨어 구현을 3장에서 다루며 4장에서는 HDL로 기술된 나눗셈 연산기의 시뮬레이션 및 검증을 수행하고 다른 알고리즘으로 구현된 나눗셈 연산기의 성능과 비교 및 분석한다. 마지막으로 5장에서는 결론으로 본 논문의 내용을 정리한다.

Ⅱ. Taylor 전개식을 이용한 역수 알고리즘

2.1 Taylor 전개식을 이용한 역수 알고리즘과 분석

Taylor 전개식을 사용한 역수 알고리즘은 젯수의 역수를 Taylor 전개식을 이용하여 근사화된 값을 구하고 피젯수와 곱셈하여 나눗셈을 수행하는 알고리즘을 말한다[8].

$$X = Y\,Q + M$$

(단 X: 피젯수, Y: 제수, Q: 몫, M: 나머지)

$$\therefore\ Q = \frac{(X-M)}{Y} < \frac{X}{Y}$$

피젯수와 젯수를 각각 X와 Y라 하고 1/2이상 1미만으로 정규화된 값이라고 가정한다. q 비트로 구성된 X를 상위 p 비트만을 사용하여 표현한다면, 실제 X 값과의 오차는 1/(2p) - 1/(2q) 이내가 된다. X의 최소 값은 X(q-1)...X(q-p)000...0이 되며 최대 값은 X(q-1)...X(q-p)111...1이 된다. 불확실한 범위는 X(q-p)의 값인 1/(2p)보다는 작은 값이다.

같은 방법으로 q 비트로 구성된 Y를 상위 m개의 비트만으로 표현한다면 실제 Y와의 오차는 1/(2m)-1/(2q) 이내에 존재한다. Xh는 아래와 같이 X의 상위 p개 비트와 나머지 비트를 0으로 채운 값으로 정의한다. 같은 방법으로 Yh는 Y의 상위 m개 비트와 나머지 비트를 1로 채운 값으로 정의한다.

$$X_h = X_{(q-1)}\ldots X_{(q-p)}000\ldots0$$
$$Y_h = Y_{(q-1)}\ldots Y_{(q-p)}111\ldots1$$

ΔX, ΔY를 각각 $\Delta X = X - Xh$, $\Delta Y = Y - Yh$으로 정의한다. Xh, Yh의 정의로부터 ΔX는 항상 음이 아닌 수이고, ΔY는 항상 양이 아닌 수이다. 따라서 1/Yh의 값은 실제 역수 1/Y보다는 항상 작거나 같은 값이다. 비슷하게, Xh/Yh의 값도 X/Y의 값보다는 작거나 같은 값이다.

초기 X 값이 1/2(0.1XX...X)이상 1(0.111...1)미만 범위에 있으므로 최상위 비트는 1이다. 다음 iteration시 피젯수 X'으로 가능한 최대 값을 결정하기 위해서 매 iteration마다 제거될 수 있는 최대 값 즉 최대 비트수를 결정해야 한다. 예를 들어서, $X' < 1/2r$를 보장할 수 있다면, 처음으로 0이 아닌 값은 1/2r+1이 되므로 상위 r개의 비트가 0이 되며, 상위 r개의 비트가 매 iteration마다 제거된다. Y=Yh에서의 1/Y에 대한 테일러 전개식은 식 (2-1.1)과 같다.

$$1/Y = B = 1/Y_h - \Delta Y/Y_h^2 + (\Delta Y)^2/Y_h^3 - (\Delta Y)^3/Y_h^4 + \cdots \quad (2\text{-}1.1)$$

$\Delta Y \le 0$이기 때문에 식 (2-1.1)의 모든 항은 음이 아닌 수이다. X의 초기 값은 1/2이상, 1미만의 범위에 있는 값이므로 MSB는 1이다. X'의 최대값을 계산하여 이로부터 iteration마다 제거시킬 수 있는 비트의 최소 개수를 구한다.

X'에 대한 수식은 식 (2-1.2)와 같다.

$$X' = X - X_h \times B \times Y \quad (2\text{-}1.2)$$

여기에서 B는 식 (2-1.2)의 처음 t개의 항이다. 이 경우에 B≈1/Y로 한다면, Rb, Rc, Rd와 같은 오차가 발생하며, 식 (2-1.3)와 같이 기술할 수 있다.

$$B=1/Y-R_b-R_c-R_d \tag{2-1.3}$$

Rb는 테일러 전개식의 처음 t개 이후의 항을 버림으로서 발생하는 오차항이다. 전개식에서 버려지는 항들이 모두 음이 아닌 수이기 때문에 이 오차는 항상 음이 아닌 수이다.

Rc는 계산한 B값을 테이블에 저장할 때 유한한 비트로 제한하기 때문에 발생하는 오차이다. 테이블에 값을 저장할 때 하위 비트를 항상 버림으로 저장하므로 이 오차는 항상 음이 아닌 수이다.

Rd는 B를 계산하기 위해서 사용된 연산시 하위 비트를 버림으로 발생하는 오차이다. 모든 연산시 하위 비트를 버리도록 한다면 이 오차는 항상 음이 아닌 수이다.

X'에 대한 수식인 식 (2-1.2)를 식 (2-1.4)와 같이 정리할 수 있다.

$$\begin{aligned} X' &= X-X_h\times B\times Y \\ &= X_h+\Delta X-X_h\times(1/Y-R_b-R_c-R_d)\times Y \\ &= \Delta X+X_h\times(R_b+R_c+R_d)\times Y \end{aligned} \tag{2-1.4}$$

만일 Rb, Rc, Rd가 취할 수 있는 최대 값이 정해져 있다면 식 (2-1.4)로부터 X'의 최대값을 계산할 수 있다.

첫 번째 오차 항목인 Rb에 대해서 계산한다. 테일러 전개식의 w+1번째 항을 Bw+1이라고 한다면 이 항은 Bw+1= (-ΔY)w/Yh(w+1)이라고 할 수 있다. ΔY의 값이 음이 아닌 수이기 때문에 B의 모든 항들은 음이 아닌 수이다. 따라서 식 (2-1.5)의 부등식이 성립한다.

$$B_{w+1}<(1/2^{(m\times w)})\times(1/(Y_h)^{(w+1)}) \tag{2-1.5}$$

B를 계산하기 위해서 처음 t개의 항을 사용하였다면, 나머지 항들의 총합이 Rb가 되며 이는 식 (2-1.6)과 같다.

$$\begin{aligned} R_b &= \sum_{g=1}^{\infty}B_{g+1}<\sum_{g=1}^{\infty}1/2^{(m\times g)}\times 1/(Y_h)^{(g+1)} \\ &= \frac{(1/2^{(m\times g)})\times(1/Y_h)^{(t+1)}}{(1-1/(2^m\times Y_h))} \end{aligned} \tag{2-1.6}$$

m이 1보다 큰 수라면 식 (2-1.6)의 값은 1/2(m×g)×(1/Yh)(t+1)보다 약간 큰 값이 된다. m이 5이상의 값이라면 Rb는 식 (2-1.7)의 부등식을 만족한다.

$$R_b<\frac{1.1}{2^{(m\times t)}\times(Y_h)^{(t+1)}} \tag{2-1.7}$$

Rc는 테일러 전개식의 처음 t개의 항을 계산하여 이를 테이블로 저장할 때 제한된 비트수로 저장하기 때문에 발생하는 오차였다. Rd는 B를 계산하는 과정에서 하위 비트들이 버려져서 발생하는 오차였다.

테이블에 존재하는 오차를 ε i라고 하고 계산 과정에서 발생하는 오차를 δ i라고 한다면 B는 식 (2-1.8)과 같이 나타낼 수 있다.

$$\begin{aligned} B &= \sum_{i=1}^{t}[\delta_i+(\varepsilon_i 1/Y_h^i)\times(-\Delta Y)^{(i-1)}] \\ &= \sum_{i=1}^{t}\left[\frac{(-\Delta Y)^{(i-1)}}{Y_h^i}\right]+\sum_{i=1}^{t}[(\varepsilon_i)\times(-\Delta Y^{(i-1)}]+\sum_{i=1}^{t}\delta_i \end{aligned} \tag{2-1.8}$$

이로부터 Rc는 ε i를 포함하는 항들의 합과 같다.

$$R_c=\sum_{i=1}^{t}\varepsilon_i\times(-\Delta Y)^{(i-1)} \tag{2-1.9}$$

또, B를 계산하면서 최종합을 구한 뒤 하위 비트를 제거할 때 발생하는 오차를 δ 0라고 한다면 식 (2-1.10)과 같다.

$$R_d=\sum_{i=0}^{t}\delta_i \tag{2-1.10}$$

테이블 Gi의 출력 비트의 개수는 bi라고 가정하고 만일 1/2≤Yh<1이라면 (1/Yh)i의 최대 가능항 값은 2i이지만 계산 과정에서는 하위 비트를 버림하므로 실제로는 2i미만의 값이 된다. 따라서, 테이블 Gi에 있는 값의 LSB는 1/2(bi-i)가 된다. 식 (2-1.11)와 같이 각 ε i는 LSB보다는 작은 값을 가진다.

$$\varepsilon_i<1/2^{(b_i-i)} \tag{2-1.11}$$

ΔY의 worst-case 값은 1/2m-1/2q일 때이다. 이를 사용하여 식 (2-1.9)을 치환하여 정리하면 식 (2-1.12)이 된다.

$$\begin{aligned} R_c &< \sum_{i=1}^{t}1/2^{(b_i-i)}\times(1/2^{(m\times i-m)}) \\ R_c &< \sum_{i=1}^{t}1/2^{(b_i+m\times i-m-i)} \end{aligned} \tag{2-1.12}$$

식 (2-1.12)으로부터 bi의 적당한 값을 결정할 수 있다.

$$b_i=(m\times t-i)+\lceil \log_2 t \rceil-(m\times i-m-i) \quad (2\text{-}1.13)$$

식 (2-1.13)으로부터 테이블 Gi는 다음 테이블 Gi+1보다 m-1비트 더 크다는 사실을 알 수 있다.

오차 Rc는 식 (2-1.14)와 같은 부등식을 만족하게 된다.

$$R_c<\sum_{i=1}^{t}1/2^{((m\times t-i)+\lceil \log_2 t \rceil}$$

$$R_c<1/2^{(m\times t-t)} \quad (2\text{-}1.14)$$

두 번째 오차 항목인 Rd에 대해서 계산한다. δ i는 최대 가능한 절단 오차(truncation error)를 나타낸다. 이를 사용하면 B를 계산하는 기능 블록인 곱셈기와 덧셈기의 크기를 줄일 수 있다.

1≤B<2이기 때문에 B의 MSB는 1을 가진다. B가 bb비트로 절단되었다면 δ 0를 제한하면 된다.

$$\delta_0<1/2^{(b_b-1)} \quad (2\text{-}1.15)$$

만일 Rd를 식 (2-1.16)과 같이 제한하고자 한다면 식 (2-1.17)과 같이 δ 0를 제한하면 된다.

$$R_d<1/2^{(m\times t-t+2)} \quad (2\text{-}1.16)$$

$$\delta_0<1/2^{(m\times t-t+3)} \quad (2\text{-}1.17)$$

이제 Rb, Rc, Rd에 대한 수식을 식 (2-1.4)에 대입을 하여서 X'의 최대 값을 계산한다. Xh<1이고 ΔX≤1/2p- 1/2q이기 때문에 X'에 대한 식을 식 (2-1.18)와 같이 정리할 수 있다.

$$X'=\Delta X+X_h\times R_b\times Y+X_h\times R_c\times Y+X_h\times R_d\times Y$$
$$<\Delta X+R_b\times Y+R_c\times Y+R_d\times Y$$

$$X'<1/2^{p}-1/2^{q}+\frac{1.1\times Y}{2^{(m\times t)}\times (Y_h)^{(t+1)}}+\frac{Y}{2^{(m\times t-t)}}+\frac{Y}{2^{(m\times t-t+2)}} \quad (2\text{-}1.18)$$

Yh > Y이기 때문에, Yh= Y로 설정하고 식 (2-1.8)를 정리하면 식 (2-1.19)을 얻을 수 있다.

$$X'<X_2=1/2^{p}-1/2^{q}+\frac{1.1}{2^{(m\times t)}\times (Y_h)^{t}}+\frac{Y}{2^{(m\times t-t)}}+\frac{Y}{2^{(m\times t-t+2)}} \quad (2\text{-}1.19)$$

이제 Y의 가능한 범위내에서 X2의 최대값을 계산할 수 있다. 극대값을 구하기 위해서 미분을 하고 이를 0으로 하는 Y 값을 구하면 식 (2-1.20)과 같다.

$$Y=\left(\frac{2^{(-t+2)}\times 1.1\times t}{5}\right)^{1/(t+1)} \quad (2\text{-}1.20)$$

식 (2-1.20)의 값은 극소값이며 이는 이계도함수를 통해서 알 수 있다. 따라서 X2를 최대가 되게 하는 Y 범위의 경계 값인 1/2이나 1일 때이며 그 결과가 식 (2-1.21)에 나타나 있다.

$$X_2=1/2^{p}-1/2^{q}+\frac{1.725}{2^{(m\times t-t)}} \quad (\text{단, } Y=1/2)$$

$$X_2\le 1/2^{p}-1/2^{q}+\frac{1.525}{2^{(m\times t-t)}} \quad (\text{단, } Y=1,\ t\ge 2) \quad (2\text{-}1.21)$$

식 (2-1.21)로부터 Y= 1/2일 때, X2는 최대값을 가지며 식 (2-1.19)에 대입하면 식 (2-1.22)을 얻을 수 있다.

$$X'<1/2^{p}-1/2^{q}+\frac{1.725}{2^{(m\times t-t)}} \quad (2\text{-}1.22)$$

식 (2-1.22)에 p= m×t-t+2를 대입하면 식 (2-1.23)이 된다.

$$X'<1/2^{(m\times t-t-1)} \quad (2\text{-}1.23)$$

식 (2-1.23)로부터 X'에서 0이 아닌 1이 처음으로 나타나는 자리는 m×t-t번째 비트인 X'(q-m×t+t)가 된다.

2.2. 반올림

기본적으로 역수 알고리즘은 젯수의 역수를 구한 다음에 피젯수와 곱해서 몫을 구하는 방법이다. 이러한 역수 알고리즘을 사용하여 하드웨어를 구현할 때 주의해야 하는 점은 하드웨어로 무한 정밀도를 구현할 수는 없기 때문에 발생하게 되는 반올림 오류에 주의해야 한다. 간단한 예로 13/51을 소수점 이하 두 자리까지 계산할 때 피젯수 51의 역수 1/51은 0.196...이 되어 0.20으로 반올림될 수 있으며 13과 0.20을 곱한 0.26을 계산 결과 값으로 생각할 수 있다. 하지만 실제 값은 0.25이므로 계산 결과 값의 오차가 발생하게 된다. 이러한 반올림 오류는 역수를 취하는 과정에서 발생한 오차가 젯수와 승산되면서 오차가 증폭되었기 때문이다.

이러한 문제점을 해결하고 IEEE-754 부동소수점

표준안을 만족시킬 수 있는 정밀도를 얻기 위해서 수정된 역수 알고리즘[9]들이 사용되고 있다. 수정된 역수 알고리즘은 근사화시킨 역수와 피젯수를 이용하여 근사화시킨 몫을 구하고, 근사화된 몫을 사용하여 오차를 보정하기 때문에 앞에서 설명했던 반올림 오류가 발생하지 않도록 할 수 있다. 본 논문에서도 이와 같은 방법을 사용한다. 본 논문에서 몫의 근사값이 실제 몫을 계산하는 과정중에 정확한 나머지를 동시에 계산하도록 하여 IEEE-754 표준안에서 요구하는 올바른 반올림을 직접 수행할 수 있다.

2.3. 나눗셈 연산기의 연산 처리 흐름도

피젯수(X) ÷ 젯수(Y)를 수행하는 나눗셈 연산기의 연산 과정은 다음과 같은 흐름으로 된다.

[1] 지수(exponent) 결정
지수(결과) = 지수(X) - 지수(Y)

[2] 나눗셈 연산기의 초기화 처리
레지스터를 초기화시키고, 젯수와 피젯수를 레지스터로 읽어서 저장한다.
젯수의 상위 m개 비트를 사용하여 테이블 G1으로부터 1/Yh를, 테이블 G2로부터 1/Y2h를 읽는다.
각 테이블의 출력 비트의 크기는 bi이다. 테이블의 출력을 이용하여 1/Y를 근사화시킨 B를 계산한다.

[3] 첫 번째 iteration 이후의 나머지 반복수행 동작
생성된 B를 사용하여 식 (2-1.2)와 식 (2-3.1)를 계산하여 몫과 나머지를 구한다.

$$Q' = Q + X_h \times B \times 1/2^{(j-h)} \qquad (2\text{-}3.1)$$

p= m×t-t-2로 설정하고 B를 계산할 때 m×t-t-1개의 0을 쉬프트하여 제거하고 다시 레지스터를 설정한다. 원하는 정밀도가 될 때까지 연산을 반복한다. 이러한 반복 수행사이클에 의해 생성된 몫의 비트에는 반올림(rounding)을 위한 G(Guard), R(Round) 비트가 포함되어 있다.

[4] 반올림과 지수(exponent) 조정
1 비트를 왼쪽으로 쉬프트할 때, 지수를 1 만큼 감소시키며 반올림 모드(rounding mode)에 따라 반올림을 수행한다.

Ⅲ. 나눗셈 연산기의 설계

제 2장에서 제안한 나눗셈 알고리즘을 하드웨어로 구현하기 위해서는 곱셈을 수행할 수 있는 유닛과 덧셈/뺄셈을 수행할 수 있는 유닛이 존재하여야 한다. 오늘날의 부동소수점 유닛에는 이와 같은 연산을 수행하는 mantissa 곱셈기와 mantissa 덧셈기를 기본적으로 가지고 있으며 이들의 하드웨어 크기가 부동소수점 유닛의 상당 부분을 차지한다. 따라서 본 논문에서 제안한 나눗셈 알고리즘만을 수행하는 전용 mantissa 곱셈기와 mantissa 덧셈기를 설계하여 하나의 부동소수점 유닛에 집적화시킨다면, 부동소수점 유닛의 크기가 지나치게 증가하여 실현 가능하지도 않고, 다른 부동소수점 명령어에 비해 나눗셈 명령어의 사용 빈도수가 낮다는 점을 고려한다면 이러한 방법은 바람직한 방법이라고도 할 수 없다.

이러한 제한점을 고려하여 본 논문에서는 나눗셈을 위한 전용 하드웨어를 설계하여 추가시키는 방법 대신에 (그림 3-1)과 같이 부동소수점 유닛에 기본적으로 존재하는 mantissa 곱셈기와 mantissa 덧셈기를 변형시키고, 이 기능 유닛들을 이용하여 나눗셈을 수행할 수 있도록 하였다. 따라서 기존의 부동소수점 유닛에 나눗셈 기능을 추가시킬 때 발생하는 추가적인 하드웨어를 최소화시킬 수 있었다.

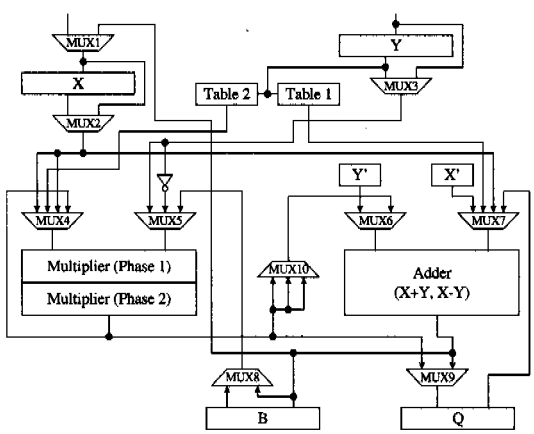


그림 3-1 블록 다이어그램

IEEE-754 부동소수점 표준안의 배정도 승산 연산을 위한 수행 사이클은 2사이클이 소요되므로 (그림 3-1)에서는 phase1과 phase2로 표기하였다.

3.1. 테이블

역수 알고리즘은 원하는 정밀도에 이르는 iteration의 횟수는 적지만 하나의 iteration을 수행하기 위해서는 여러 번의 곱셈과 덧셈이 필요하기 때문에 수행 시간을 많이 소모하게 된다 따라서 초기 Q값은 실시간으로 계산하지 않고 테이블로부터 미리 계산한 값을 로드하도록 설계하여 iteration의 횟수도 줄이고 수행 시간을 줄일 수 있었다.

본 논문에서는 3번의 iteration만에 배정도 연산이 완료될 수 있도록 테일러 전개식의 두 번째 항까지를 근사식에 활용하였으며 각 비트수들을 정하였다. 만일 테일러 전개식의 첫 번째 항만으로 활용하여 같은 iteration 횟수만에 같은 정밀도 연산이 완료되도록 구현한다면 테이블이 입력이 피젯수의 20비트(MSB의 leading 1은 제외)가 되어야 하며 테이블의 출력은 22비트이어야 한다. 따라서, 이를 구현한다면 1M 엔트리의 ROM으로 각 엔트리는 22비트가 되므로 전체 ROM 크기는 4000기가비트가 될 것이다. 그러나, 이러한 테이블을 실제로 구현한다는 것은 실현 불가능한 일이므로 테일러 전개식의 두 번째 항까지를 근사식으로 이용하여 구현하도록 하였다.

식 (2-1.13)은 각 테이블 크기를 정할 수 있는 수식으로 m을 11이라고 한다면 각 테이블의 엔트리는 1K 엔트리가 되며 각 테이블의 엔트리 크기는 1/Yh에 대한 테이블의 경우는 22비트, 1/Y2h에 대한 테이블의 경우는 12비트이다. 따라서 1/Yh에 대한 테이블은 22Kbit ROM으로, 1/Y2h에 대한 테이블은 12Kbit ROM으로 구현할 수 있었으며 전체 ROM 크기는 34Kbit ROM이며 이는 상당히 작은 오버헤드만으로도 충분히 구현할 수 있는 수치이므로 이 값을 선택하였다.

각 테이블 값은 C 프로그램을 사용하여 발생시켰다. 테이블 발생 C 프로그램은 곱셈 알고리즘과 나눗셈 알고리즘으로 pencil & paper 알고리즘을 채택하여 설계하여 C 프로그램의 알고리즘 오류로 인한 오류가 존재하지 않도록 하였다. C 프로그램은 내부 연산시에는 무한 정밀도로 연산을 수행하도록 하여 반올림으로 인한 오차의 누적이 없도록 하였으며 최종 결과만 반올림 모드를 절단 모드로 하여 본 논문에서 제안한 알고리즘과 호환되도록 하여 오류가 발생하지 않도록 하였다.

3.2. 곱셈기

곱셈기는 입력 오퍼랜드를 받아서 2 클럭 사이클만에 계산하여 그 결과를 출력하는 2 latency, 1 throughput 곱셈기를 사용하였다. 각 입력 오퍼랜드는 54비트이고 출력은 108 비트이다. IEEE-754 표준안의 배정도 데이터 포맷이 53 비트임에도 본 논문에서 설계한 곱셈기의 오퍼랜드가 54 비트인 이유는 나눗셈 알고리즘에서 사용되는 정규화된 값을 0.1xxx...x으로 사용하였기 때문이며, 곱셈기의 오퍼랜드는 정수 부분 1 비트와 fraction 부분 53 비트가 된다.

3.3. 덧셈기

덧셈기는 101 비트의 덧셈/뺄셈을 수행할 수 있어야 하며 IEEE-754 표준안에서 요구하는 덧셈기보다 큰 덧셈기를 사용해야 하는 이유는 나눗셈을 수행할 때 Q를 계산하기 위해 85 비트의 가산이, X'을 계산하기 위해서 101 비트의 감산이 필요하기 때문이다.

3.4. 제어

본 논문에서 제안한 알고리즘을 사용하여 나눗셈 연산기를 구현한다면 기존의 부동소수점 유닛에 존재하는 기능 유닛을 최대로 활용하여 큰 추가 면적 없이 구현할 수 있었다.

추가되는 기능 유닛이 적은 대신에 제어는 비교적 복잡하게 된다. 따라서 별도의 제어 유닛이 필요하며, 이 제어 유닛은 mux 제어 신호, 레지스터 제어 신호, 기능 유닛 제어 신호와 상태들을 출력하여 (그림 3-2)와 같은 흐름으로 수행하도록 제어하게 된다. 나눗셈 연산 시작 신호를 받으면 각 상태(state)가 순차 진행항 각 상태에 따라 어떠한 제어를 할 것인지를 결정한다. 전체 상태는 15개의 상태로 구성되어 있으며 단정도 나눗셈 연산시 15개의 상태를 순차적으로 수행하며 배정도 연산시 14번째 상태까지 순차 진행을 한 뒤 10번째 상태부터 다시

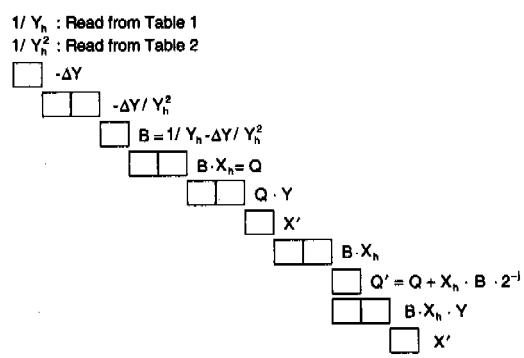


그림 3-2 단정도 나눗셈에 대한 파이프라인 흐름과 제어

진행하도록 하였다. 제어 신호 중에는 나눗셈 연산을 제어하는 신호는 아니지만 이들은 각각 다음사이클에 해당 유닛을 사용한다는 의미이므로 명령어 해석기는 이 신호를 이용하여 해당 유닛을 사용해야 하는 명령어의 이슈 여부를 제어할 수 있게 된다. 이러한 제어 유닛의 도움으로 단정도 나눗셈을 수행하면서도 의존성이 없는 곱셈 명령의 경우 9개의 명령을, 덧셈 명령의 경우 10개를 수행할 수 있다. 배정도 나눗셈의 경우 곱셈 명령은 12개, 덧셈 명령은 13개를 수행할 수 있다.

Ⅳ. 설계 검증, 레이아웃 및 결과 고찰

본 논문에서는 부동소수점 나눗셈 연산기의 동작을 확인하기 위해 Verilog HDL을 사용하여 기능 수준에서 시뮬레이션을 수행하여 검증하였으며 특히 시뮬레이션의 검증은 pencil & paper 알고리즘을 C 언어로 구현한 프로그램을 사용하여 수행되었으며, 테스트 벡터를 이용하여 시뮬레이션을 수행한 결과와 C 언어로 기술한 프로그램의 결과를 비교하여 본 연산기를 검증하였다.

특히 펜티엄 프로세서의 나눗셈 유닛의 오류가 크게 이슈화되었던 것처럼 SRT 알고리즘을 사용할 경우, 테스트 벡터를 사용하여 PLA의 각 엔트리에 직접 접근하기 어려워서 검증이 까다로운 반면 본 유닛에서 사용한 2개의 룩 테이블은 테스트 벡터를 사용하여 각 엔트리를 직접 접근할 수 있기 때문에 쉽게 검증될 수 있다.

4.1. 다른 나눗셈 알고리즘과의 성능 비교

오늘날 개발되어 상용화된 프로세서의 부동소수점 나눗셈 연산기에서 사용한 알고리즘과 아키텍처와 본 논문에서 제안한 알고리즘과 아키텍처를 비교한다. 비교 대상 프로세서는 IEEE-754 부동소수점 표준안을 지원하는 i486, Pentium, UltraSPARC, IBM RISC System/6000을 대상으로 하여 동일한 정밀도에서 비교하였다. 비교는 연산을 완료할 때까지의 수행 사이클의 수, 나눗셈 연산을 하드웨어로 수행하기 위해서 필요한 기능 유닛의 면적과 각 프로세서에서 제산 연산시 임계경로에 해당하는 유닛들을 비교하였으며 수퍼스칼라 구조와 같은 다중 명령어 이슈 구조의 적합성도 비교하였다.

우선 (표 4-1)에서는 각각의 정밀도(precision)에 따른 수행 사이클을 다른 마이크로프로세서에 탑재된 부동소수점 유닛에서의 나눗셈 연산기와 비교해 보았다. (표 4-1)에서 보듯이 본 연구의 나눗셈 연산기는 MIPS 계열과 SPARC 계열의 나눗셈 연산기보다 단정도 연산의 경우 25% ~ 50%, 배정도 연산의 경우 70% 정도 더 빠르다. 또한 i486보다는 단정도 연산의 경우 2배 이상, 배정도 연산의 경우 약 3배정도 더 빠른 성능을 나타냈는데, 이는 i486은 한 클럭 사이클 동안 단지 1 비트의 몫만을 생성하는 전통적인 비복원 방식의 "쉬프트 & 빼기" 알고리즘을 사용하기 때문이다. 또한 radix-4 SRT 알고리즘을 사용한 Pentium보다는 단정도 연산의 경우 약 20%, 배정도 연산의 경우 약 50%의 성능 향상을 나타내었다. 본 논문에서 사용한 알고리즘과 비슷한 역수 알고리즘인 Newton-Raphson 알고리즘을 사용한 IBM RISC System/6000보다는 2~3사이클의 향상이 있었다. 다만 UltraSPARC의 단정도 연산과 비교했을 때 4사이클의 성능 저하가 존재하였다. 따라서, 본 논문에서 제안한 나눗셈 연산기의 성능은 상용화된 프로세서와 비교했을 때 상위급에 해당한다는 사실을 알 수 있다.

표 4-1 각 정밀도에 따른 다른 나눗셈 연산기와의 연산 수행사이클의 비교

정밀도 (precision)	단정도 (single)	배정도 (double)
i486	35	62
MIPS R4000	23	36
SPARC compatible	20~23	35~37
Pentium (Radix-4 SRT)	19	33
UltraSPARC (Radix-8 SRT)	12	22
IBM RISC/6000 (Newton-Raphson)	19	23
본 논문 (제안한 series)	16	23

이제 (표 4-2)에서는 최근에 개발된 프로세서에서 사용되었던 각 알고리즘과 아키텍처에 따라 나눗셈 연산을 수행하기 위해서 추가된 면적을 비교하였다. 다른 공정, 다른 아키텍처를 사용하여 구현된 프로세서들을 비교하기 위해서 면적을 많이 차지하게 되는 기능 유닛을 비교하였다.

SRT 알고리즘을 채택한 대표적인 프로세서는 Intel의 Pentium과 SUN의 UltraSPARC이 있다. Pentium의 경우, radix-4 SRT 알고리즘을 사용하였기 때문에 PLA를 사용하여 몫을 정하게 된다. 만일

표 4-2 각 프로세서간의 나눗셈 연산을 위한 추가 유닛의 비교

	추가 유닛			
	몫 결정 유닛	MAC	CSA	Final Adder
Pentium (Radix-4 SRT)	PLA	없음	56비트 CSA	56비트 고속 덧셈기
UltraSPARC (Radix-8 SRT)	QS (거의 없음)	없음	9개의 56비트 CSA	56비트 고속 덧셈기
IBM RISC/6000 (Newton-Raphson)	ROM	MAF	없음	없음
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

PLA를 ROM 테이블로 구현한다면 8K 비트에 해당하는 ROM이 된다. 또, 56 비트의 CSA와 56 비트의 고속 덧셈기가 필요하다. 반면, UtraSPARC의 경우, radix-8 SRT 알고리즘과 같은 성능이면서도 임계 경로와 면적에 영향을 많이 주는 유닛인 PLA를 사용하지 않기 위해서 radix-2 overlapped stages 방식을 사용하여 구현되었다. 이 방식은 발생 가능한 7개의 나머지를 모두 발생시키면서 몫을 결정하여 나머지를 mux로 선택하는 방식이며 몫을 결정하는 로직은 PLA가 아닌 간단한 조합회로로 구현 가능하다. 따라서, 나눗셈 연산을 위해서 추가된 유닛은 크게 보면 9개의 56 비트 CSA, 56 비트의 고속 덧셈기라 할 수 있다.

Newton-Raphson 방식을 사용한 IBM RISC System/6000 프로세서의 경우, MAF(Multiply-Adder-Fused) 유닛을 사용하여 연산을 수행하기 때문에 나눗셈 연산을 위해 추가된 면적은 없었다. 하지만 수퍼스칼라 구조에서 사용될 수 있도록 하기 위해서 MAF 2개 사용하기 때문에 결국 나눗셈 연산을 위해서 추가된 유닛은 1개의 MAF이다.

이와 같은 분석을 통해서 기존의 어떠한 프로세서 아키텍쳐보다도 본 논문에서 제안한 알고리즘과 아키텍쳐를 사용하는 것이 면적을 더 적게 차지한다는 사실을 알 수 있다.

각 구현 알고리즘에 따른 임계 경로를 살펴보기로 한다. Pentium 프로세서에서 사용되었던 radix-4 SRT와 UltraSPARC에서 채용되었던 radix-2 overlapped stages SRT, 그리고 IBM RS/6000에서 사용되었던 Newton-Raphson 방식과 비교하였다. 각 프로세서들에 적용된 공정이 다르기 때문에 직접 비교는 어려우며 사용된 유닛을 비교하였다. 우선 각 나눗셈 연산기 내에서 임계경로를 비교해보면 다음 표 (4-3)과 같다. (단, multiplier는 54 * 54 승산기의 처리 시간, add8은 8비트 adder의 처리 시간, pla는 pla access time, csa는 3:2 csa의 처리 시간, qslc는 radix-2 overlapped stage SRT에서 몫을 결정하는 로직의 처리 시간, 4mux3는 3:1 mux 4개의 지연 시간, MAF는 IBM RS/6000의 MAF 연산 시간이다)

표 4-3 제산기의 임계경로 비교 분석

	본 제산기	radix-4 SRT	radix-2 overlapped stage SRT	Newton-Raphson
제산기의 임계경로	multiplier	add8+pla+csa	qslc+3csa+4mux3	MAF

이와 같이 나눗셈 연산기의 임계경로는 나눗셈 알고리즘에 따라서 다양하게 나타낼 수 있는데 역수 알고리즘을 구현한 제산기의 경우 승산기의 임계경로와 일치하게 된다. 그런데 SRT를 적용한 부동소수점 연산기 역시 승산기를 포함하고 있으며 이를 고려한 임계 경로는 다음 표 (4-4)와 같다.

표 4-4 부동소수점 연산기의 임계경로 비교 분석

	본 제산기	radix-4 SRT	radix-2 overlapped stage SRT	Newton-Raphson
부동소수점 연산기의 임계경로	multiplier	max (add8+pla+csa, multiplier)	max (qslc+3csa+4mux3, multiplier)	MAF

이와 같은 비교로 SRT 알고리즘을 사용하는 부동소수점 연산기들의 임계경로는 역수 알고리즘을 사용한 부동소수점 연산기들의 임계경로보다 더 길거나 같다는 결론을 얻을 수 있다. 따라서 나눗셈 연산기의 성능은 임계경로가 같다고 생각했을 때 나눗셈 연산을 위한 iteration의 횟수와 관계가 있음을 알 수 있으며 이는 앞에서 제시했던 연산 사이클 비교를 통한 본 제산기의 우수성을 확인할 수 있었다.

제안된 나눗셈기가 최근의 마이크로프로세서 설계 추세인 수퍼스칼라 구조에 적합하도록 설계되었기 때문에 수퍼스칼라 구조에서 나눗셈 명령과 곱셈/덧셈/뺄셈 명령등이 기능 유닛의 충돌없이 수행될 수 있다는 장점을 가진다. 실제로 SRT 알고리즘을 채택한 가장 빠른 프로세서인 Ultra SPARC의 경우, 컴파일러가 데이터 의존성 없는 곱셈 명령과

FALU 명령을 각각 18개(배정도 연산의 경우)나 찾아야 기능 유닛이 쉬지 않고 동작하게 된다.

마지막으로 Newton-Raphson 역수 알고리즘을 사용한 프로세서는 곱셈기와 덧셈기를 공유하기 때문에 나눗셈 명령어의 수행중에는 데이터 의존성이 없는 명령어라도 수행시킬 수 없는 경우가 발생한다. 특히, Newton-Raphson 역수 알고리즘을 사용한 가장 빠른 프로세서인 IBM RISC System/6000의 경우, MAF를 사용하며 나눗셈 연산 사이클 동안 MAF가 쉬지 않고 연산에 사용되므로 이러한 문제를 가지게 된다. 이러한 문제를 해결하기 위해서 IBM RISC System/6000에서는 MAF를 2개 사용하는 구조를 채택하고 있다. 반면 본 유닛에서는 곱셈기와 덧셈기가 독립적으로 존재하는 구조이므로 데이터 의존성이 없는 곱셈 명령과 덧셈/뺄셈 명령 등이 이슈될 수 있다. 즉, 컴파일러가 의존성이 없는 12개의 곱셈 명령과 13개의 FALU 명령(배정도 연산의 경우)를 찾으면 기능 유닛이 쉬지 않고 수행되게 된다. 따라서 나눗셈 명령을 수행중이더라도 데이터의존성이 없는 명령어들이 수행될 수 있기 때문에 수퍼스칼라 프로세서에서도 본 논문에서 제안한 부동소수점 유닛을 수정없이 사용할 수 있다. 또한, 각 나눗셈 유닛에 대한 나눗셈 명령어의 지연시간(배정도 연산시 SRT의 경우 28사이클, 본 알고리즘의 경우 19사이클)과 수퍼스칼라 프로세서의 평균 이슈율이 2가 되지 못한다는 점을 고려한다면, 본 논문에서 제안한 별도의 나눗셈 전용 유닛을 가지지 않는 구조로 인해서 프로세서의 성능이 저하된다고 할 수 없으므로 본 논문에서 제안한 알고리즘과 아키텍쳐가 우수함을 알 수 있다.

4.2. 회로의 합성 및 레이아웃

레이아웃 결과 크기는 3.35mm × 3.18mm 이며 면적은 10.65mm2 이다. 곱셈기만의 레이아웃 결과 크기는 2.57mm × 2.72mm 로 약 6.99mm2이고 덧셈기만의 레이아웃 결과 크기는 1.32mm × 0.74mm로 약 0.98mm2이었다. 곱셈기의 오버헤드는 나눗셈 연산을 지원하기 위해서 승수와 피승수 모두 1비트씩 확장된 것이며 이에 따른 오버헤드는 0.01mm2에 불과하였다. 덧셈기의 오버헤드는 53비트에서 101비트로 확장되었으므로 면적 대비 약 61%에 해당하는 0.6mm2이었다. 22K ROM과 12K ROM의 크기는 0.287mm2이었다. 따라서 나눗셈 연산을 지원하기 위해서 기존의 부동소수점 연산기에 추가된 하드웨어는 겨우 0.897mm2에 불과하였다. (표 4-5)에서는 이러한 면적 분석을 표로 정리하였다.

이상의 분석을 통해서 본 논문에서 제안한 나눗셈 연산기가 적은 면적의 오버헤드만으로도 우수한 성능을 나타낸다는 것을 증명할 수 있다.

표 4-5 설계된 나눗셈 연산기의 면적 분석
(단, 수치는 0.6㎛ CMOS triple-metal 공정에 기반한 값임)

유닛	면적 (mm^2)	나눗셈 연산을 위한 추가 면적(mm^2)	%
곱셈기	6.99	0.01 (53 * 53 -> 54 * 54)	0.14
덧셈기	0.98	0.60 (53 + 53 -> 101 + 101)	61.2
ROM 테이블	0.287	0.287	100
전체	10.65	0.897	8.4

V. 결 론

본 논문에서는 IEEE-754 부동소수점 표준을 지원하는 나눗셈 연산기를 설계하고, Verilog HDL을 사용하여 기능 수준에서 이의 동작을 검증하였다.

고속의 나눗셈 연산을 위해 본 연산기를 다음과 같은 방식으로 설계하였다.

첫째, 테일러 전개식을 사용한 역수 알고리즘을 사용하여 역수 알고리즘이 가지는 고속의 나눗셈 연산을 실현하면서도 IEEE-754표준안을 만족시킬 수 있도록 수정하였다. 또, 역수 알고리즘의 분석을 통해서 테이블 ROM의 크기를 예측하였고 이를 바탕으로 테이블 ROM의 크기를 줄일 수 있도록 알고리즘을 수정하였고 이를 채용하여 비록 첫 번째 iteration 때에 근사화된 젯수의 역수를 구하는 과정이 필요하지만 기존에 상용화된 어떠한 프로세서의 나눗셈 연산기보다도 높은 성능을 나타낼 수 있도록 설계하였다.

둘째, 기존의 부동 소수점 유닛에 존재하는 곱셈기와 덧셈기를 최대로 활용할 수 있도록 하여 하드웨어적인 측면에서 면적의 이점을 얻을 수 있었다.

셋째, 현재 널리 사용되는 대표적인 나눗셈 알고리즘들을 분석을 통해서 장점과 단점을 파악하여 동일한 성능을 가지는 다른 나눗셈 연산기보다는 적은 면적을, 비슷한 면적을 가지는 다른 나눗셈 연산기보다는 높은 성능을 나타내도록 설계하였다.

넷째, 제어 시점을 정확히 분석하고, MAC 구조

가 아닌 곱셈기와 덧셈기를 각각 설계하여 나눗셈 연산 중에라도 기능 유닛이 필요하지 않을 때에는 이를 외부 신호로 알려주어서 다른 명령어가 수행될 수 있도록 하였다. 따라서, 본 논문에서 제안한 구조를 비순차 처리를 지원하는 수퍼스칼라 구조에서도 수정없이 사용될 수 있도록 하였다.

다섯째, 본 논문에서는 C 프로그램에 의한 테스트를 통해 각 테이블의 검증은 물론 전체 구조를 시뮬레이션을 통해 검증을 완료하였다.

참 고 문 헌

[1] IEEE Standard Committee, *IEEE Standard for Binary Floating Point Arithmetic*, ANSI/IEEE std. 754-1985

[2] Kai Hwang, *Computer Arithmetic : Principle, Architecture, & Design* , John Wiley & Sons, Inc. 1979

[3] Israel Koren, *Computer Arithmetic Algorithms*, John Wiley & Sons, Inc. 1993

[4] Joe Heinrich, *MIPS R4000 Microprocessor User's Manual*, 2nd Edition, MIPS Technologies, Inc.,1994

[5] Sun Inc., *STP1012PGA microSPARC-II User's Manual*, 2nd SPARC Technology Business, Inc.,1994

[6] Sun Inc., *UltraSPARC-I User's Manual*, SPARC Technology Business, 1995

[7] J.A. Prabhu and G. B. Zyner, "167MHz radix-8 divide and square-root using overlapped radix-2 stages," *in Proc. 12th Symp. Comput. Arithmetic*. July, 1995, pp.155-162.

[8] D. Wong and M. Flynn, "Fast division using accurate quotient approximation to reduce the number of iterations," *in Proc. 10th IEEE Symp. Comput. Arithmetic*, June 1991, pp. 191-201.

[9] P. W. Markstein, "Computation of Elementary Functions on the IBM RISC System/6000 Processor," *IBM J. Res. Develop*, 34, pp. 111-119, 1990.

[10] R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC System/6000 Floating-Point Execution Unit," *IBM J.Res. Develop*. 34, pp. 59-70, 1990

권 호 경(Kwon Ho-Kyung) 정회원

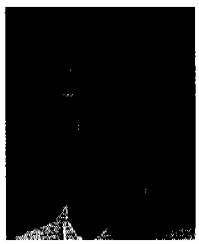
1996년 2월 : 연세대학교 전자공학과 졸업

1998년 2월 : 연세대학교 전자공학과 석사

1998년 3월~현재 : 현대전자산업 MCU 설계팀 연구원 근무중

<주관심 분야> MCU 설계, MCU 아키텍처

문 상 국(Moon Sangook) 정회원



1995년 2월 : 연세대학교 전자공학과 졸업

1997년 2월 : 연세대학교 전자공학과 석사

1997년 3월~현재 : 연세대학교 전자공학과 박사과정

<주관심 분야> VLSI 설계, 암호용 프로세서 설계

문 병 인(Moon Byung-In) 정회원

1995년 2월 : 연세대학교 전자공학과 졸업

1997년 2월 : 연세대학교 전자공학과 석사

1997년 3월~현재 : 연세대학교 전자공학과 박사과정

<주관심 분야> 마이크로프로세서 설계, SMT

이 용 석(Lee Yong-Surk) 정회원

1973년 5월 : 연세대학교 전기공학과 졸업

1977년 2월 : University of Michigan Ann Arbor 석사

1981년 3월 : University of Michigan Ann Arbor 박사

1993년~현재 : 연세대학교전기전자공학과 교수 재임중

<주관심 분야> 마이크로프로세서 설계