論文2000-37SD-12-11

새로운 제산/제곱근기를 내장한 고성능 부동 소수점 유닛의 설계
(Design of a high-performance floating-point unit adopting a new divide/square root implementation)

李 泰 永*, 李 成 淵**, 洪 仁 杓***, 李 溶 錫***
(Tae Young Lee, Sung Youn Lee, In Pyo Hong, and Yong Surk Lee)

요 약

본 논문에서는 고성능 수퍼스칼라 마이크로프로세서에 적합하고, IEEE 754 표준을 준수하는 고성능 부동 소수점 유닛의 구조를 설계한다. 부동 소수점 AU에서는 비정규화 수 처리를 모두 하드웨어적으로 지원하면서 추가적인 지연 시간이 생기지 않도록 점진적 언더플로우 예측 기법을 제안 구현한다. 부동 소수점 제산/제곱근기는 기존의 고정적인 길이의 몫을 구하는 방식과 달리 매 사이클마다 가변적인 길이의 몫을 구하는 구조를 채택하여 성능과 설계 복잡도 면에서 SRT 알고리즘에 의한 구현 보다 우수하도록 설계한다. 또한, 수퍼스칼라 마이크로프로세서에 이식이 용이하도록 익셉션 예측 기법을 세분화하여 적용하며, 제산 연산에서의 익셉션 예측에 필요한 스톨 사이클을 제거하도록 한다. 설계된 부동 소수점 AU와 제산/제곱근기는 부동 소수점 유닛의 구성요소인 명령어 디코더, 레지스터 파일, 메모리 모델, 승산기 등과 통합되어 기능과 성능을 검증하였다.

Abstract

In this paper, a high-performance floating point unit, which is suitable for high-performance superscalar microprocessors and supports IEEE 754 standard, is designed. Floating-point arithmetic unit (AU) supports all denormalized number processing through hardware, while eliminating the additional delay time due to the denormalized number processing by proposing the proposed gradual underflow prediction (GUP) scheme. Contrary to the existing fixed-radix implementations, floating-point divide/square root unit adopts a new architecture which determines variable length quotient bits per cycle. The new architecture is superior to the SRT implementations in terms of performance and design complexity. Moreover, sophisticated exception prediction scheme enables precise exception to be implemented with ease on various superscalar microprocessors, and removes the stall cycles in division. Designed floating-point AU and divide/square root unit are integrated with an instruction decoder, register file, memory model and multiplier to form a floating-point unit, and its function and performance is verified.

* 正會員, 現代電子 시스템 IC 디지털미디어 AV팀
(AV Team, Digital Media, System IC Division, Hyundai Electronics Industries Co., Ltd.)
** 正會員, 亞南半導體 FAB研究所 디자인팀
(Design Team, FAB Laboratory, Anam Semiconductor.)
*** 正會員, 延世大學校 機械電子工學部
(School of Electrical and Mechanical Engineering, Yonsei University.)
※ 이 논문은 (1998)년 한국학술진흥재단의 학술연구비에 의하여 지원되었음.
接受日字 : 2000年 1月12日, 수정완료일 : 2000年11月22日

I. 서 론

기술과 공정의 발전으로 마이크로프로세서의 성능은 날로 증가하고 있으며, 집적도의 증가에 따라서 예전에는 여러 칩으로 구성되던 컴퓨터 시스템이 점차로 하나의 칩 안에 내장되고 있다. 특히, 부동 소수점 유닛은 과학 및 공학 시뮬레이션, 수치 해석, 3차원 그래픽 등에 대한 요구가 급증함에 따라 필수적인 요소가 되고 있다.

최근에는 단순히 하나의 명령어로 하나의 데이터를 처리하던 것을 넘어서 하나의 명령어로 여러 개의 데이터를 처리하는 이른바 SIMD(single-instruction

multiple-data) 구조의 설계가 보편화되면서 8비트 혹은 16비트 정수 데이터에 적용되어 멀티미디어 처리 유닛으로 발전하였는데, 부동 소수점 연산에 대한 요구가 증가함에 따라서 이러한 개념이 부동 소수점 연산에도 적용되기에 이르렀다[1].

또한, IEEE 754 표준[2]은 부동 소수점 연산에 대한 표준으로서 상용화된 거의 모든 마이크로프로세서가 IEEE 754 표준과의 호환 모드를 제공하고 있다. 특히, IEEE 754 표준에서는 정규화 수(normalized number) 보다 작은 결과를 얻었을 때 이를 표현할 수 있는 비정규화 수(denormalized number)를 규정하고 있다. 그러나, 이를 전적으로 하드웨어에 의존하여 구현하는 경우 추가적인 부담이 크고, 전적으로 소프트웨어에 의존하는 경우 호환 모드에서의 수행 속도가 지나치게 느려진다.

부동 소수점 연산에서 제산과 제곱근 연산은 수행 빈도가 5% 이하로 매우 낮지만 처리 지연 시간이 다른 연산보다 매우 길므로 전체적인 부동 소수점 연산의 성능에 미치는 영향은 매우 크다. 따라서, 최신 고성능 마이크로프로세서는 매 사이클 당 2비트 이상 몫을 구할 수 있는 제산/제곱근기를 채택하고 있다[3]. 그러나, SRT 알고리즘은 Radix-4 이상으로 기수가 증가할 경우 설계 복잡도와 면적이 기하급수적으로 증가되어, Radix-4 이상의 효과를 낼 수 있는 다양한 방법이 제안되고 있다[4].

따라서, 본 논문에서는 고성능을 위해 부동 소수점 가감산과 승산에 SIMD 기법을 적용하고, 비정규화 수 처리의 많은 부분을 효율적으로 구현할 수 있는 방식을 제안하며, 성능과 설계 복잡도 면에서 Radix-4 SRT 알고리즘을 능가할 수 있는 새로운 제산/제곱근를 제안 설계한다. 부동 소수점 유닛의 설계에 있어 정수 연산 코어와의 통합도 중요하기 때문에 익셉션 처리를 위해서는 익셉션 예측과 스톨 방식을 적용하고, 제산에서의 익셉션 예측 방식을 개선하여 익셉션 예측에 의한 스톨 사이클을 제거하도록 하였다.

Ⅱ. 부동 소수점 유닛의 전체 구조

본 논문에서 설계된 부동 소수점 유닛의 전체 구조는 그림 1과 같다.

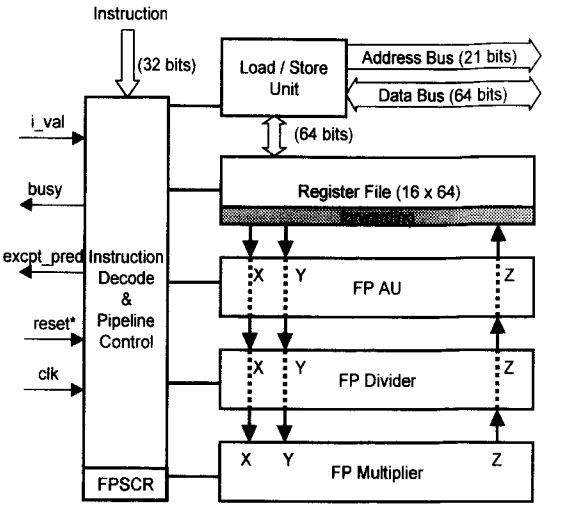


그림 1. 부동 소수점 유닛의 전체 구조
Fig. 1. Overall block diagram of the floating-point unit.

부동 소수점 유닛은 32비트의 명령어를 해석하고 파이프라인을 제어하는 부분, 16개의 배정도 수 또는 32개의 단정도 수를 저장할 수 있는 레지스터 파일, 메모리로부터 64비트의 데이터를 읽고 쓰는 로드/스토어 유닛, 부동 소수점 유닛의 상태 및 제어 정보를 저장하는 부동 소수점 상태/제어 레지스터 그리고 데이터패스로서 부동 소수점 AU, 부동 소수점 제산/제곱근기, 부동 소수점 승산기로 구성된다.

위와 같은 부동 소수점 유닛은 다음과 같은 특징을 갖는다. 즉, 부동 소수점 AU와 승산기는 3단계의 파이프라인(E1, E2, E3)을 가지며, 부동 소수점 제산/제곱근기는 반복적인 단계(E1)와 라운딩 단계(EF)로 구성된다. 제산과 제곱근 연산은 처리 지연 사이클이 크므로 비순차적인 결과 쓰기(out-of-order completion)를 지원하며, 가감산, 승산, 제산 및 제곱근 연산의 마지막 실행 단계에서 데이터 포워딩을 지원한다. 단정도 형식 처리의 속도를 높이기 위해서는 IEEE 754에서 정의하는 단정도 및 배정도 형식 외에 PS (packed single) 형식을 추가로 정의하여 성능을 높인다. 비정규화 수 처리 지원을 위해서는 부동 소수점 AU에서는 점진적 언더플로우 예측 기법을 적용하고, 제산과 승산에서는 일부분을 하드웨어적으로 지원한다. 정확한 익셉션(precise exception)을 위해서는 익셉션 예측과 스톨 기법을 적용한다.

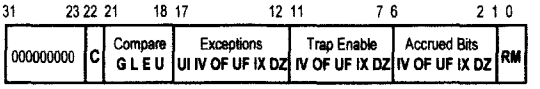


그림 2. 부동 소수점 상태/제어 레지스터
Fig. 2. Floating-Point Status/Control Register (FPSCR).

부동 소수점 유닛의 상태를 확인하고 제어하기 위해서 그림 2와 같이 32 비트의 상태/제어 레지스터가 필요하다. 32 비트 중 비트 31부터 23까지는 0으로 고정되어 사용되지 않으며, 22 비트는 IEEE 호환 모드(1)와 빠른 동작 모드(0) 구분을, 비트 21부터 18까지는 부동 소수점 수의 비교 결과를 G(greater than), L(less than), E(equal to), U(unordered)의 플래그로 나타낸다. 비트 17부터 12는 UI(unimplemented), IV(invalid), OF(overflow), UF (underflow), IX(inexact), DZ (divide by zero)를 나타낸다. 비트 11부터 7까지는 5가지 익셉션에 대한 트랩 활성화 플래그를 저장하고, 비트 6부터 2까지는 트랩이 활성화 되어 있지 않을 경우 발생된 익셉션을 누적하는 비트이다. 마지막으로 비트 1에서 0은 IEEE 754 표준에서 정의하는 네가지 라운딩 방식, 즉, RN (round to nearest; 11), RPI(round to positive infinity; 01), RNI(round to negative infinity; 10), RZ(round to zero; 00)을 나타낸다.

부동 소수점 명령어는 크게 데이터를 읽고 쓰는 로드/스토어 명령어 군과 데이터를 연산하는 연산 명령어 군으로 구분된다. 표 1은 이와 같은 명령어 형식으로 나타낼 수 있는 부동 소수점 유닛의 명령어를 보여준다. 연산 명령어의 경우에는 각 연산에 대해 특정 형식 지정자(fmt)를 가질 수 있으며, 특히 SIMD 기법을 적용하기 위해 정의된 PS 형식도 지정할 수 있다.

표 1. 지원되는 부동 소수점 명령어
Table 1. Supported floating-point instructions.

	니모닉 (mnemonic)	명령어 코드 (op_code)	함수 코드 (function code)	수행 내용
로드/스토어 명령어	LSF	110000	존재하지 않음	단정도 형식을 FPR에 로드
	LDF	110001		배정도 형식을 FPR에 로드
	LCF	110010		32비트 제어정보를 FPSCR에 로드
	SSF	111000		단정도 형식을 FPR로부터 스토어
	SDF	111001		배정도 형식을 FPR로부터 스토어
	SCF	111010		FPSCR로부터 스토어
연산 명령어	FADD	000000	00000	부동 소수점 가산
	FSUB		00001	부동 소수점 감산
	FSWP		00010	PS 형식의 단정도 값을 교환
	FCMP		00011	부동 소수점 비교 (익셉션 무시)
	ROUND		00100	부동 소수점 수를 32비트 정수로 변환
	FCON_D		00101	배정도 형식으로 변환
	FCON_S		00110	단정도 형식으로 변환
	FCMPE		00111	부동 소수점 비교 (익셉션 발생)
	FMOV		01000	부동 소수점 이동
	FMOVA		01001	부동 소수점 절대값 이동
	FMOVN		01010	부동 소수점 부호 역전 이동
	FSUM		01011	PS 형식의 단정도 값 가산
	FMUL		01100	부동 소수점 승산
	FDIV		01101	부동 소수점 제산
	FSQRT		01110	부동 소수점 제곱근

Ⅲ. PS 형식에 의한 연산

PS 형식은 부동 소수점 연산에 SIMD 개념을 적용한 것으로 단정도 형식을 사용하는 과학 및 공학 응용 프로그램, 고도의 3차원 그래픽, 디지털 신호 처리 등에 사용되어 최대 2배의 성능 향상을 도모할 수 있다. PS 형식은 기존에 배정도 형식을 위해 존재하는 데이터패스를 수정하여 사용하기 때문에 지수 및 부호를 계산하기 위해 추가되는 부분을 제외하면 데이터패스의 복잡도 및 면적 증가가 경미하므로 성능과 면적비를 고려하면 매우 효율적인 형식이라고 할 수 있다.

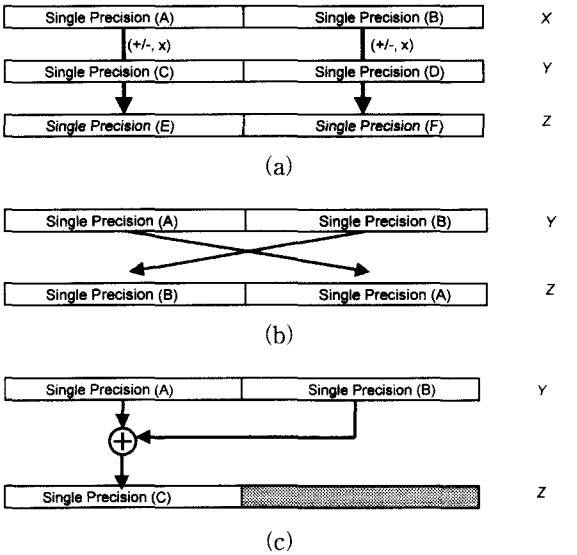


그림 3. PS 형식과 연산 방식 (a) 가감산 및 승산 (b) FSWP 명령어 (c) FSUM 명령어
Fig. 3. PS format and its operations (a) add/sub and multiplication (b) FSWP instruction (c)FSUM instruction.

PS 형식은 그림 4와 같이 배정도 형식의 64비트 필드를 이용해 32비트의 단정도 형식 두 개를 함께 지정하여 연산 시 두 개의 단정도 형식을 동시에 처리할 수 있다. PS 형식이 적용되는 명령어는 가감산(FADD, FSUB) 및 승산(FMUL), 이동(FMOV, FMOVA, FMOVN) 그리고 FSWP, FSUM이 있다. 제산과 제곱근, 형식 변환, 비교 연산에서는 PS 형식의 상위 단정도 만을 처리한다. FSWP은 이와 같은 경우 앞뒤의 단정도 값을 교환하기 위한 명령어이며, FSUM은 PS 형식안의 두 개 단정도 값을 더하여 하나의 단정도 값으로 저장하는 명령어이다. 그림 3은 이와 같은 연산 방식을 도식적으로 나타낸다.

Ⅳ. 비정규화 수 처리

비정규화 수 처리는 부동 소수점 AU에서의 처리와 부동 소수점 승산 및 제산 연산에서의 처리로 나뉠 수 있다. 부동 소수점 AU에서는 가감산, 변환, 비교 등 모든 경우를 하드웨어적으로 처리하며, 이러한 처리에 따른 추가적인 지연 시간 증가를 제거하기 위해서 점진적 언더플로우 예측 기법 (gradual underflow prediction ; GUP)을 제안하였다. 부동 소수점 승산 및 제산 연산에서는 전체 비정규화 수 처리 집합 중 약 11.2 %에 해당하는 부분을 하드웨어로 처리하며, 나머지 부분은 트랩의 지원으로 처리된다.

1. 부동 소수점 AU의 비정규화 수 처리

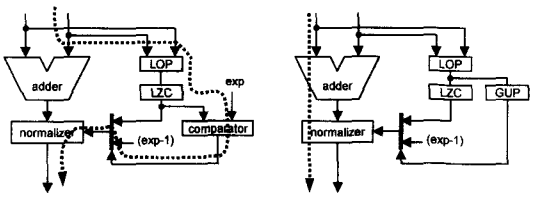
부동 소수점 AU에서 비정규화 수 처리와 관련되는 명령어는 FADD, FSUB, FSUM, FCON_S_D, FCON_D_S이며, 이러한 명령어 수행에서 실제 처리되는 비정규화 수는 5가지 경우로서 표 2와 같다.

표 2. 부동 소수점 AU에서의 비정규화 수 처리

Table 2. Processing of denormalized numbers in floating-point AU.

명령어	연산 내용
FADD, FSUB, FSUM	(Norm ± Norm) → DeNorm
	(Norm ± DeNorm) → Norm or DeNorm
	(DeNorm ± DeNorm) → Norm or DeNorm
FCON_S_D	Double Norm → Single DeNorm
FCON_D_S	Single DeNorm → Double Norm

여기에서 실제 하드웨어 구현 시 문제가 되는 부분은 첫 번째 경우(정규화 수 ± 정규화 수 → 비정규화 수)로서, 나머지 4 경우는 기존의 데이터패스를 적절하게 제어함으로써 구현이 가능하다. 첫 번째 경우는 연산 결과 값의 선행 0의 개수(leading zero count)가 결과 지수 값보다 크거나 같을 경우에 발생된다. 그러므로, 정확한 선행 0의 개수를 알아야 그 발생 여부를 결정할 수 있다. 따라서, 선행 0 예측 기법을 적용한 경우 이와 같은 수행을 처리하려면 그림 4의 (a) 같이 예측된 선행 0의 개수 값과 지수 값을 비교하여 최종적으로 정규화 기에 입력될 쉬프트 값을 결정해야 한다. 그러나, 이 경우 비교 과정과 선택 과정에 의한 지연 시간이 길어지고 항상 부호화 된 선행 0 개수 값을 얻어야 한다. 따라서, 본 논문에서는 부호화 되지 않은 값으로 비교 연산을 미리 수행함으로써 비정규화 수 처리에 의한 지연 시간 증가가 일어나지 않도록 하였다. 즉, 그림 4의 (b)와 같이 예측된 가감산 결과값과 지수값을 바로 비교하여 점진적 언더플로우 예측 신호를 만들어 낸다. 여기에서 확정이 아니고 예측이 되는 이유는 가감산 예측값이 그 특성상 한 비트의 오차를 가질 수 있기 때문이다. 이 경우 선행 0 예측, 점진적 언더플로우 예측 그리고 최종 선택의 지연 시간은 가감산기의 지연 시간 보다 작기 때문에 결국 임계 경로는 종전과 같이 가감산기를 통과하는 경로가 된다.



(a) (b)

그림 4. 비정규화 수 처리에 따른 지연 시간 (a)비교기 사용 방식 (b)GUP 방식

Fig. 4. Latency time due to the denormalized number processing (a)comparator method (b)GUP method.

GUP에서 가감산 예측값과 지수와의 비교 과정은 다음과 같이 설명될 수 있다. 즉, 가감산 예측 결과의 선두 비트로부터 (지수-1) 만큼 논리합(OR) 하여 그 결과가 1이면 (지수-1) 보다 선행 0의 개수가 적기 때문에 언더플로우가 발생하지 않고, 결과가 0이면

(지수-1) 보다 선행 0의 개수가 많기 때문에 언더플로우가 발생할 것으로 예측된다. 발생할 것으로 예측되면 정규화기에서 (지수-1) 만큼 정렬하고 그 결과 최선두 비트가 0이면 언더플로우를 확정하고 최종 지수값에 0을 넣으며, 최선두 비트가 1이면 언더플로우가 발생하지 않은 것이므로 최종 지수값에 (지수-1) 값을 넣는다. GUP의 하드웨어는 스티키 비트를 검출하는 하드웨어와 동일한 방식으로 설계되면 되는데, 본 논문에서는 하드웨어 부담이 적은 2단계의 마스크 방식을 사용하였다.

2. 부동 소수점 제산과 승산에서의 비정규화 수 처리

제산과 승산에서 비정규화 수를 처리하기 위해서는 기본적으로 연산 결과의 가수 부분에 대하여 선행 0 개수를 세고 정규화 할 필요가 있다. 그런데, 부동 소수점 AU에서는 이 부분이 이미 존재하므로 기존의 회로를 활용하여 구현할 수 있었으나, 제산기와 승산기에는 존재하지 않는다. 따라서 모든 비정규화 수 처리를 위해서는 기본적으로 제산기와 승산기에 각각 선행 0 카운터와 정규화기를 첨가해야 하고, 이로 인해 파이프라인 단수와 하드웨어가 늘어날 수 밖에 없다[5]. 따라서, 본 논문에서는 제산과 승산 시 하드웨어 추가를 최소화하고 파이프라인 단에 영향을 주지 않으면서 부분적으로 비정규화 수 처리를 지원하는 방식을 적용하였다. 이러한 방식은 UltraSparc 승산기 설계[6]에서 제안된 것으로, 본 논문에서는 제산기에도 적용할 수 있도록 식을 유도하였다[7]. 그 결과식은 식 (1)과 같으며, 식 (2)는 승산기에 적용되는 식이다.

$$x\_e - y\_e + bias \leq -(F+1) \quad (1)$$

$$x\_e + y\_e - bias \leq -(F+2) \quad (2)$$

식에서 x_e, y_e는 각각 X 피연산자와 Y 피연산자의 바이어스(bias)가 가해진 지수 값이며, bias와 F는 정밀도에 따라 결정되는 바이어스 값과 가수(fraction)의 비트 수이다. 이 식이 의미하는 바는 제산이나 승산의 연산 결과가 지수와 가수의 값과 상관없이 항상 0이 되는 조건이다. 즉, 비정규화 수의 결과를 얻는 경우 중에서 0의 값을 갖는 경우로서, 이 경우에는 하드웨어에서 0의 값을 출력하면 된다. 배정도의 경우 bias = 1023, F = 52이고, 지수의 값이 0에서부터 2047까지 균일한 확률 분포를 갖는다면, 위 식에 의해서 제산과 승산 시 각각 11.2 % 가량을 하드웨어로 처리할 수 있게 된다. 나머지 경우는 익셉션에 의해서 처리되게 된다.

V. 익셉션 예측과 스톨

익셉션 예측과 스톨은 부동 소수점 명령어 수행 전에 피연산자를 검사하여 익셉션이 발생할 가능성이 있는 상황에서 익셉션 예측 신호를 내보내 다른 유닛에서 이미 수행 중인 명령어들을 스톨시켜 비순차적인 결과 쓰기를 방지하는 것이다. 만약 문제의 명령어의 마지막 실행 단계에서 실제로 익셉션이 발생하면 그 상태에서 그대로 트랩의 도움을 받으면 되고, 익셉션이 발생하지 않으면 스톨 상태를 거두어 들이고 계속적으로 실행하게 된다. 그림 5는 익셉션 예측과 스톨의 한 예로 후자의 경우를 나타낸 것이다.

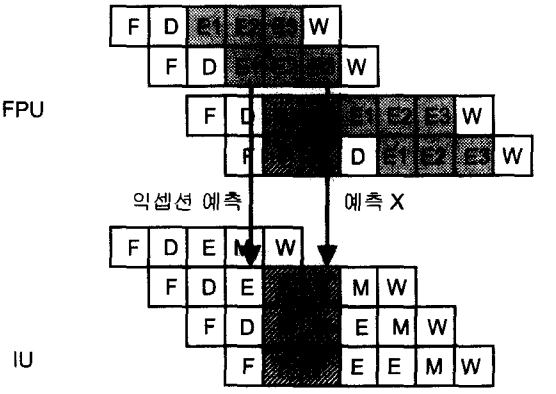


그림 5. 익셉션 예측과 스톨의 예
Fig. 5. Example of exception prediction and stall.

이와 같은 방식을 적용하면 익셉션 발생 명령어 이전의 상태를 저장하야 하는 번거로움이 제거된다[8]. 그러나, 익셉션이 예측된 경우에는 다른 명령어의 수행이 항상 중단되므로, 예측의 정확도가 떨어지는 경우 성능의 저하를 가져온다. 따라서 예측의 정확도를 높이고 미리 결정할 수 있는 경우에는 최대한 빨리 결정하는 것이 중요하다. 특히, 제산의 경우에는 연산 결과가 일반적으로 수십 사이클 이후에 얻어지므로 익셉션 예측 시 스톨 사이클이 많아지므로 익셉션 예측을 제거하는 것이 바람직하다. 익셉션 예측은 기본적으로 피연산자의 지수 간 상호 관계에 의해서 결정된다. 즉, 예측된 결과 지수값이 정규화 수의 경계에 위치하게 되면 최종적으로 가수값에 따라 익셉션의 발생 여부가

결정하게 되므로 이때 익셉션이 발생할 것으로 예측하게 된다. 나머지 경우에는 익셉션이 발생할지 안 할지를 연산 초기에 결정할 수 있다.

제산에서 지수에 의해 익셉션이 예측된 경우에 최종적으로 그 여부를 결정하는 변수는 몫의 최상위 비트(Q[msb])와 라운딩 결과 값이다. 몫의 최상위 비트는 제산 첫 사이클에서 알 수 있으며 라운딩 결과 값은 제산을 종료한 후 라운딩 단계까지 가야 알 수 있다. 그러나, 라운딩에 의해서 지수값에 영향을 줄 수 있는 경우는 최상위 비트를 제외한 몫의 모든 비트가 1일 경우밖에 없다. 따라서, 몫의 모든 비트가 1인 경우만을 미리 검출할 수 있다면 제산 초기에 익셉션 발생 여부를 확정할 수 있을 것이다. 몫의 모든 비트가 1인 경우는 0.111…111 혹은 1.111…111의 두 경우 인데, 후자의 경우는 가드(guard; G), 라운드(round; R), 스티키(sticky; S) 비트가 모두 0이 되므로 결코 라운딩이 발생할 수 없다. 따라서 고려할 필요가 없으며, 전자의 경우는 수학적인 증명에 의해서 제산 첫 사이클의 나머지가 정확이 -0.000…001이 되는 경우에만 발생한다. 이 경우 R 비트는 0, S 비트는 1을 가지므로 라운딩 방식에 따라 그 결과를 미리 알 수 있다[9]. 본 논문에서 설계되는 제산기는 매 사이클 마다 정확한 나머지 값을 구하기 때문에 위와 같은 경우를 제산 첫 사이클에서 검출할 수가 있다. 위의 결과를 오버플로우와 언더플로우에 적용하면 다음과 같다. 제산에서 라운딩에 의해 지수 변경이 일어나는 경우의 몫은 항상 선행 비트(Q[msb])가 0으로 시작한다. 오버플로우 경우에는 정규화를 수행하고 라운딩을 수행하므로, 선행 비트 0을 제거하기 위해서 왼쪽으로 1 비트 쉬프트를 수행하게 된다. 이때 R 비트가 유효수(significand)의 최소 자리로 들어오게 되는데, R 비트는 항상 0으로 결정됨을 이미 설명하였다. 결국 유효수의 최소 자리가 0으로 되기 때문에 라운딩에 의해서 지수의 변경이 발생하지 않는다. 언더플로우 경우에는 정규화를 수행하지 않고 라운딩이 수행되므로 첫 사이클의 나머지 값에 따라 언더플로우 발생 여부를 확정한다. 표 3은 제산에서의 확정 조건과 나머지 명령어 연산에서의 익셉션 예측 조건을 정리한 것이다. 실제 예측 및 확정 신호는 표에 나타난 조건과 라운딩 방식에 의한 라운딩 증가 여부와 논리곱되어 발생된다.

표 3. 익셉션 예측과 확정 조건
Table 3. Conditions for exception prediction and confirmation.

명령어 종류	익셉션	예측 및 확정
가감산	오버플로우 (E1 예측)	{(x_e==254(S)/2046(D)) or (y_e==254(S)/2046(D))} and (effective addition)
	언더플로우 (E1 예측)	(max(x_e, y_e) ≤ 24(S)/53(D)) and |diff(x_e, y_e)| ≤ 1 and (effective subtraction)
ROUND_D	오버플로우 (E1 예측)	{(y_e==1053) and (y_s==positive) and (y_f[53:24]=='h3FFF_FFFF)} or {(y_e==1054) and (y_s==negative) and (y_f[53:23]=='h7FFF_FFFF)}
FCON_S_D	오버플로우 (E1 예측)	(y_e==1150) and (y_f[53:31]=='h7FFF_FF)
	언더플로우 (E1 예측)	(y_e==896) and (y_f[53:32]=='h3FFF_FF)
제산	오버플로우 (E1 확정)	{x_e-y_e+bias > 255(S)/2047(D)} or {(x_e-y_e+bias==255(S)/2047(D)) and Q[msb]}
	언더플로우 (E1 확정)	{x_e-y_e+bias < 1} or {(x_e-y_e+bias==1) and ~Q[msb] and ~(the remainder of the first cycle= -0.000…001 and round_increment=no)}
승산	오버플로우 (E1 예측)	x_e+y_e-bias = 254(S)/2046(D)
	언더플로우 (E1 예측)	x_e+y_e-bias = 0

VI. 부동 소수점 AU의 설계

부동 소수점 AU는 부동 소수점 가감산, 변환, 이동, 비교 등 대부분의 명령을 수행하는 블록으로 기본적인 파이프라인 단계와 구조는 가감산에 적합하도록 설계하며 여기에 나머지 명령어를 수행할 수 있도록 제어한다. 설계된 부동 소수점 AU의 파이프라인 단계는 다음과 같다. 첫 단계는 지수 차이와 가수 비교를 수행하고 이에 따라 가수 정렬을 한다. 두 번째 단계는 가수의 가감산을 수행하고 동시에 선행 0개수 예측과 점진적인 언더플로우 예측을 수행한다. 세 번째 단계는 정규화 혹은 라운딩을 수행한다. 그림 6은 설계된 부동 소수점 AU의 전체 블록 다이어그램으로서 PS 형식을 위해 두 개의 지수 처리부가 존재한다. 첫 번째 파이프라인 단계에서 정렬 과정을 빠르게 수행하기 위해서 두 개의 병렬적인 정렬기를 사용하며 32비트 정수를 단정도 형식으로 바꿀 때 라운딩과 2의 보수 수행을 위해 라운딩 주입(rounding injection; RI) 블록이 있다. 라운딩 주입은 ROUND_S 명령시 사용하지 않는 X 피연산자로 특수한 값을 입력함으로써 라운딩과 2의 보수 과정을 동시에 수행하는 방식으로 ROUND_S 명령어의 파이프라인 단계를 4에서 3으로 줄일 수가 있다[7]. 이와 더불어, 첫 번째 단계에서는

입력된 피연산자에 따라 특수 경우나 무효 연산을 검출하기 위해서 특수 경우 검출 회로를 가진다. 또한, 가수에 대한 비교기를 두어 첫 번째 파이프라인 단계에서 비교 결과를 얻을 수 있으며 두 번째 파이프라인 단계의 선행 0 예측 회로를 간단화 한다. 두 번째 파이프라인 단계에서는 가수에 대한 가감산을 수행하며, 동시에 선행 0 예측 및 선행 0 카운트 그리고 GUP를 수행한다. 라운딩에 의한 증가 여부도 이때 결정한다. 또한 라운딩과 정규화를 배타적으로 수행할 수 있게 하기 위해 3-to-1 멀티플렉서를 둔다. 마지막 파이프라인 단계에서는 정규화 혹은 라운딩을 수행하고 이에 따라 언더플로우 및 오버플로우 발생을 결정하여 지수와 가수에 반영시킨다.

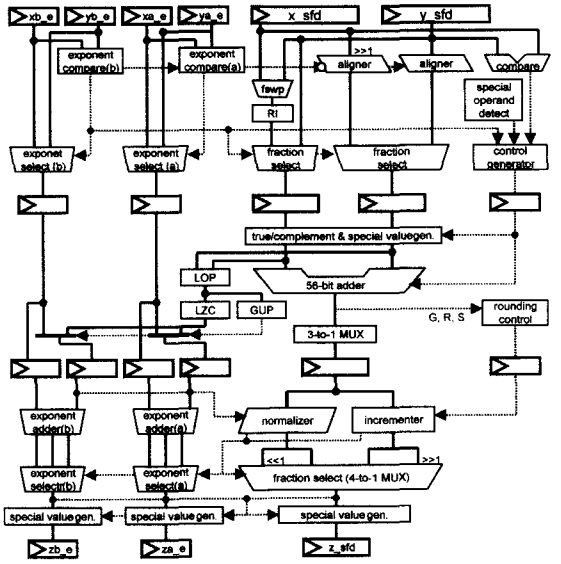


그림 6. 부동 소수점 AU의 전체 블록 다이어그램
Fig. 6. Block diagram of the floating-point AU.

Ⅶ. 부동 소수점 제산/제곱근기 설계

부동 소수점 제산 방식은 일반적으로 감산기를 사용하는 방식(subtractive method)이 많이 사용되고 있는데 SRT 알고리즘이 주로 사용되고 있다. 그러나, SRT 알고리즘은 그 특성상 설계와 검증이 매우 까다롭고, 성능을 향상시키기 위해 기수(radix)를 증가할수록 그로 인해 늘어나는 면적과 설계의 복잡도가 매우 증가한다. 따라서, 본 논문에서는 고정된 기수를 갖는 SRT 알고리즘이 아닌 가변적인 길이의 몫을 얻는 비복원 방식의 알고리즘[10]을 일반화하고 구현하였다.

제산과 제곱근에서 피연산자(operand)들의 범위는 식 (3)과 같다.

$$1 \le |\text{부분 피제수}|, |\text{제수}| < 2 \tag{3}$$

제산이나 제곱근 연산은 매 사이클마다 다음과 같은 동작을 반복한다. 제수 또는 제곱근·감수(새로 얻어진 부분 제곱근에 의해서 감산되는 양; Square Root Subtrahend)가 부분 피제수로부터 가감된다. 부분 몫(이하 q)은 가감산의 결과인 부분 나머지로부터 구해지며, 지금까지 결정된 총 몫(이하 Q)에 덧붙여진다. 부분나머지의 부호로부터 q의 최상위비트가 결정되며, q의 나머지 비트들은 이어지는 비트열로부터 쉽게 결정된다. 그러면, 0의 비트열이 어떻게 처리되는지 살펴보도록 하자. 다음은 비트열의 길이가 3비트인 경우를 예로 들어 처리 과정을 설명한다.

0의 비트열인 경우는 부분 나머지가 양일 경우 발생하며, 이때 양의 부분 나머지는 부호확장 비트, 부호비트 '0', 0의 비트열((4)에서 밑줄 표시)을 순서대로 가진다. 부분 나머지가 양수이고 피연산자들의 범위가 (3)과 같기 때문에, 결과인 부분 나머지는 항상 1비트 이상의 0의 비트열을 가진다. 부분 나머지가 양이라는 것은 제수로부터 피제수가 완전히 감해진 것을 의미하므로 q의 최상위비트는 1이 된다. 0의 비트열은 그 길이만큼 부분나머지가 제수보다 작다는 것을 의미한다. 따라서, 부분 나머지는 0의 비트열의 길이만큼 왼쪽으로 쉬프트되어 다음 순환에서 (3)의 범위를 만족하는 부분 피제수가 되며(6), 이 때, 0의 비트열에서 최상위 0을 제외한 부분이 추가의 몫으로 결정된다(7). 결국, 구해진 q의 길이와 비트열의 길이(=k_j)가 같다. 1의 비트열도 이와 비슷하게 결정된다.

$$000.001100111\cdots \text{ ; 부분나머지}(P_j) \tag{4}$$

$$001.010100101\cdots \text{ ; 제수}(D) \tag{5}$$

$$001.100111010\cdots \text{ ; 쉬프트된 부분 나머지 } (P_{j+1}=2^{k_j}\times P_j) \tag{6}$$

$$\{1(\text{최상위 비트}),00(\text{추가의 몫})\} \text{ ; } q \tag{7}$$

위와 같은 결과를 일반화하여 VQB(variable quotient bit) 알고리즘으로 명명하였으며, 부분 나머지 결정 및 부분 몫 결정의 일반식은 다음과 같다.

$$2^{k_j}P_j \pm f\times D = P_{j+1} \tag{8}$$

$$\text{if } (2^{k_j}\times P_j \ge 0) \ \&\ (P_{j+1} \ge 0)$$

$$\text{then } q_1 q_2 \cdots q_i = \{f,\ c_{i-1} \cdots c_1\} \quad (9a)$$

$$\text{if } (2^{k_j} \times P_j \geq 0)\ \&\ (P_{j+1} < 0)$$

$$\text{then } q_1 q_2 \cdots q_i = \{f\ 1,\ c_{i-1} \cdots c_1\} \quad (9b)$$

$$\text{if } (2^{k_j} \times P_j < 0)\ \&\ (P_{j+1} \geq 0)$$

$$\text{then } q_1 q_2 \cdots q_i = \{f\text{의 2의 보수},\ c_{i-1} \cdots c_1\} \quad (9c)$$

$$\text{if } (2^{k_j} \times P_j < 0)\ \&\ (P_{j+1} < 0)$$

$$\text{then } q_1 q_2 \cdots q_i = \{f\text{의 1의 보수},\ c_{i-1} \cdots c_1\} \quad (9d)$$

여기에서, P_j 및 P_{j+1}은 각각 j 및 j+1 사이클에서의 부분 나머지, f는 계수, $i=k_j$ 그리고 l은 f에 의해서 결정되는 부분몫의 크기이다. 새롭게 결정되는 몫의 길이는 비트열의 길이와 같으므로, 계수는 비트열의 길이를 증가되도록 선택하여 결정되는 몫의 길이를 증가되도록 한다. 결국 SRT에서는 고정된 비트의 정확한 몫을 PLA표를 통해 구하고 이에 따라 나머지를 구해가는 반면, VQB는 근사화 된 몫(계수)을 구하여 나머지를 구한 뒤 나머지의 비트열에 따라 정확한 몫을 구한다고 할 수 있다. 계수는 [1], [1/2, 1, 2], [3/4, 1, 3/2] 등을 사용할 수 있는데, 첫 번째와 두 번째 계수군은 하드웨어 복잡도에 따라 Radix-4 SRT 구현과 비슷하다고 할 수 있다. 따라서, 본 논문에서는 [1], [1/2, 1, 2] 계수를 사용하는 하드웨어(전자는 설계 A, 후자는 설계 B)를 설계하고 이를 Radix-4 SRT 하드웨어와 비교하였다. 위의 알고리즘에 따라 제곱근을 위한 제곱근 감수를 유도하여 구하면 표 4와 같다.

표 4. 제곱근 감수
Table 4. Square root subtrahend.

	양의 계수일 때	음의 계수일 때
1/2의 계수	{Q, 001}	{Q, 111}
1의 계수	{Q, 01}	{Q, 11}
2의 계수	{Q, 1}	{Q, 1}

하드웨어 구현을 위해서는 비트열의 길이를 제한해야 하는데 하드웨어 부담과 성능을 고려할 때 5비트가 적당하다. 따라서, 설계 A와 설계 B의 사이클 당 결정되는 몫의 평균 길이는 각각 2.54 및 2.72가 된다. 비트열의 길이가 최대 쉬프트량(5비트)을 초과할 경우에는 우선 5비트까지 몫을 결정하고 나머지 부분은 다음 사이클에 수행한다.

VQB 알고리즘 구현을 위한 하드웨어(설계 B)는 그림 7과 같다. 그림에서 점선으로 표시된 계수 결정기를 제거하면 설계 A와 같다. 먼저 계수 결정기는 피제수의 부호 비트와 피제수, 제수(제산), Q(제곱근) 각각의 소수점 이하 2비트를 입력으로 받아 피제수와 제수의 크기를 대략적으로 비교하여 사용할 계수값을 결정한다. 이와 같은 결정표는 표 5와 같다.

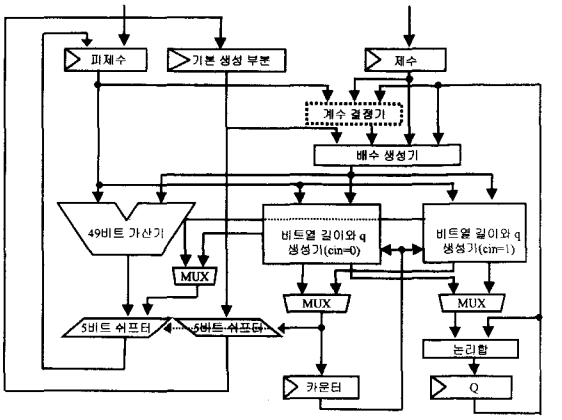


그림 7. VQB 제산/제곱근기의 블록 다이어그램
Fig. 7. Block diagram of the VQB divide/square root implementation.

표 5. 계수 결정기의 진리표
Table 5. Truth table of the factor generator.

피제수[54:51] 또는 Q	제수[53:51]	결정 계수
01.00	1.11	1/2
10.11	1.11	
01.11	1.00	2
10.00	1.00	
기타 경우		1

배수 생성기는 제곱근 감수를 생성하고 제수나 제곱근 감수에 계수를 곱한 배수 값을 만든다. 두 개의 비트열 길이와 q 생성기 그리고 가감산기는 동시에 동작한다. VQB 알고리즘 순서상 가감산이 끝난 후 나머지에 의해 비트열 길이와 q가 결정되는데 이렇게 구현하면 지연 시간이 길어지게 된다. 따라서 두 개의 병렬 블록은 자리올림수(carry)가 각각 0과 1인 경우에 대하여 상위 6비트에서 미리 비트열 길이 검출과 q 생성을 수행한다. 그림의 논리합(OR) 게이트는 선택된 q를 Q에 누적한다. 선택된 비트열 길이는 카운터에 더해지며, 카운터는 정밀도에 따라 언제 순환을 끝낼지 알린다.

부분 나머지는 비트열 길이만큼 왼쪽으로 최대 5비트 쉬프트되어 다음 순환에서 부분 피제수가 되며, 기본 생성 부분은 제곱근 연산시 누적된 비트열 길이만큼 오른쪽으로 쉬프트 되어 다음 순환에서 제곱근 감수 생성에 사용된다.

Ⅷ. 기능 검증 및 성능 비교

기능 검증과 성능 비교를 위해 부동 소수점 AU와 제산/제곱근기는 실제 합성을 전제로 기술되었으며, 부동 소수점 승산기, 명령어 해석기, 파이프라인 제어기, 레지스터 파일, 메모리 모델 등은 상위 레벨에서 기술하여 통합 검증하였다. 기능 검증을 위한 검증 벡터로는 하드웨어 특성에 따라 C 프로그램으로 생성한 벡터와 IEEE 754 표준 호환성 검증을 위한 IEEE 754 Test Suite 벡터[11]를 사용하였다. 성능 검증을 위해서는 응용 프로그램을 어셈블리 프로그램으로 작성하여 사용하였다. 부동 소수점 유닛의 성능 평가시 다음과 같은 조건을 가정하였다. 메모리에 대한 데이터 읽기/쓰기는 모두 캐쉬에 의해서 100 % 처리되며, 이슈되는 명령어는 항상 대기하고 있어 부족함이 없고, 분기 명령어는 100 % 예측된다고 가정하였다. 이와 같은 상황에서 PS 형식에 의한 성능 향상은 표 6과 같다. 표에서 SAXPY (single-precision A×X Plus Y)와 기하 좌표 변환(Geometry Transform)은 3D 그래픽에서 많이 사용되는 형태의 데이터 처리이며, FIR 과 DCT 등은 디지털 신호 처리에 많이 쓰이는 알고리즘이다. 표에서 볼 수 있듯이 PS 형식은 단정도 형식에 비해 경우에 따라 1.81~1.95배의 성능 향상을 얻을 수 있었다. 4-탭 필터의 경우에는 PS 형식의 데이터를 나중에 다시 변환해 주어야 하는 부분이 발생해 1.25배로 다소 못미치는 성능 향상이 이루어졌다. 결과적으로 평균 1.7배 이상의 성능 향상이 이루어졌다.

표 6. PS 형식에 의한 성능 향상
Table 6. Performance improvement by PS format.

응용 프로그램	단정도 사용 시	PS 사용 시	성능 향상
	수행 사이클 수	수행 사이클 수	
SAXPY	90	46	1.95x
4-tap FIR filter	83	66	1.25x
Geometry Transform	80	44	1.81x
1x8 DCT	104	54	1.92x

제산/제곱근기의 성능은 명령어의 지연 및 처리율의 사이클 수로 평가하였다. 표 7은 제안된 VQB 제산/제곱근기와 상용화된 마이크로프로세서의 제산/제곱근기를 배정도 처리 측면에서 비교한 것이다[4]. 비교 대상에서 알파 21164는 VQB 알고리즘과 유사한 방식으로 추정되며 사이클 당 2.4비트의 몫을 얻는다[12]. 따라서, 제산의 경우 평균적으로 24.33 사이클의 지연을 갖고 23.33 사이클의 처리율을 갖는다. 알파 21164는 제곱근 연산을 지원하지 않는다. 나머지 마이크로프로세서의 제산/제곱근기는 SRT 방식으로 설계되었는데, Radix-4 SRT를 사용하는 경우에는 제산 시 30사이클의 지연과 처리율을 갖는다. Radix-8 SRT 급의 방식을 사용하여 경우에는 약 20사이클 전후의 지연과 처리율을 갖는다. 제곱근 연산의 경우 일부 방식은 제산보다 2배 정도 사이클이 소요되는데, 이것은 제곱근 연산이 제산 보다 일반적으로 더 긴 임계 경로 지연을 갖기 때문에 동작 주파수 측면을 고려한 것이다. VQB 제산/제곱근기는 제산과 제곱근을 모두 지원하여 지연 사이클이 거의 동일하다. 제산시 설계 A와 설계 B가 각각 23.04 및 21.43의 지연 사이클을 보여 비슷한 방식의 알파 21164보다 다소 앞서며 제곱근도 처리하므로 더 성능이 우수하다. SRT 알고리즘을 사용한 다른 프로세서와 비교해 보면, 동급의 Radix-4 방식을 사용한 구조 보다 약 10사이클의 우위를 보였으며, Radix-8 방식을 사용한 구조와는 비슷하거나 다소 못 미치는 성능을 보였다. 그러나, 본 논문에서 제안한 VQB 알고리즘에서 Radix-8급에

표 7. 제산/제곱근의 성능 비교 (배정도)
Table 7. Performance comparison of division and square root operation.

비교 칩	사용 알고리즘	지연/처리율	
		제산	제곱근
설계 (B)	VQB (2x)	21.43 / 20.43	21.07 / 20.07
설계 (A)	VQB (1x)	23.04 / 22.04	22.65 / 21.65
알파 21164	VQB와 비슷한 알고리즘으로 추정	24.33 / 23.33	지원 안됨
HP PA8000	Radix-4 SRT	31 / 31	31 / 31
인텔 Pentium	Radix-4 SRT	39 / 39	70 / 70
인텔 Pentium Pro	Radix-4 SRT	30 / 30	53 / 53
MIPS R10000	Radix-8 SRT	18 / 18	32 / 32
PowerPC 604	Radix-4 SRT	31 / 31	지원 안됨
Sun UltraSparc	Radix-8 SRT	22 / 22	22 / 22

해당하는 [3/4, 1, 3/2] 계수군을 사용하는 제산/제곱근기를 설계한다면 Radix-8 SRT 방식보다 더 높은 성능을 보일 것으로 예상된다.

더구나, 설계 복잡도 측면에서 VQB 알고리즘이 SRT 알고리즘보다 우수하다. 즉, 이론적인 측면에서 SRT 알고리즘은 많은 공식과 그래프 분석으로 이루어져 있으며, 불확실성과 여분(redundancy)의 원리 등 어려운 이론을 포함하고 있다. 또한 하드웨어 설계 시 몫 결정 PLA를 작성해야 하며 이를 검증하기 위해서는 P-D 그래프를 분석하여 특수한 경우까지 일일이 검사해야 한다. 반면, VQB는 비트열 패턴에 따라 간단하게 몫이 결정되며 복잡한 표가 필요없고, 검증 또한 알고리즘의 대표적인 경우 검사함으로서 수행될 수 있다. 따라서, VQB 알고리즘은 SRT 보다 설계가 간단하면서도 우수한 성능을 나타낼 수 있으므로 고성능 제산/제곱근 연산에서 SRT의 대안이 될 수 있을 것으로 생각된다.

Ⅸ. 합성 결과

설계된 부동 소수점 AU와 VQB 제산/제곱근기는 RTL 수준에서 HDL로 기술되어 합성되었다. 사용된 표준 셀 라이브러리는 0.35μm 3.3-Volt 삼중 금속 CMOS 라이브러리로서 Synopsys의 디자인 컴파일러를 사용하였다. 합성된 결과는 SDF 파일 형태로 변환하여 게이트 수준에서 검증하였다. 표 8은 일반 조건에서 임계 지연 경로와 동작 주파수, 그리고 게이트 수를 나타낸다. VQB 제산/제곱근기에서 설계 B는 2배수를 사용하기 때문에 설계 A보다 면적과 지연 측면에서 다소 증가된 값을 나타냈다.

표 9와 10은 제안된 구조의 합성 결과와 기존에 발표된 논문을 비교한 것이다. 우선, 부동 소수점 AU의 경우 비교 논문은 S/390 CMOS 마이크로프로세서에 사용된 FPU로서 0.25μm의 유효 채널 길이(effective channel length)를 갖는 표준 셀 방식으로 설계되었다[13]. 일반적으로 특정한 공정을 지칭할 때 유효 채널 길이 보다 큰 값을 갖기 때문에 비교 논문에서 사용한 공정은 본 논문의 0.35μm CMOS 표준 셀 공정과 비슷한 조건이라고 할 수 있다.

표 8. 일반 조건에서의 합성 결과
Table 8. Synthesis results under normal condition.

	FP AU	VQB (설계 A)	VQB (설계 B)
임계 경로 지연	5.44 ns	5.13 ns	6.17 ns
동작 주파수 (게이트 시뮬레이션)	179 MHz	189 MHz	157 MHz
게이트 수	26,438	16,630	17,745

표 9. FP AU 합성 결과 비교
Table 9. Comparison of synthesized floating-point AU.

	S/390 마이크로프로세서의 FPU	설계된 FP AU
설계 방식	표준-셀	표준-셀
디자인 룰	Leff = 0.25	0.35 μm 3M CMOS
게이트 수	250,000 (FPU 전체)	26,438 (FP AU)
구현된 기능	FP AU + ZDC + 2 BS's	FP AU+LOP+PBS+GUP
파이프라인	3 단계	3 단계
동작 주파수	169.5 MHz at 3.3 V	179MHz at 3.3V
총 지연시간	17.7 ns	16.8 ns

표 10. VQB 제산/제곱근기 합성 결과 비교
Table 10. Comparison of synthesized VQB divide/square root unit.

	Radix-4 SRT 제산기		VQB 제산/제곱근기(A)
설계 방식	표준-셀		
디자인 룰	0.6μm 3M CMOS	0.35 μm로 축적된 경우	0.35 μm 3M CMOS
사이클 당 처리 비트 수	2.0		2.54 (평균)
임계 경로 지연	8.3ns	4.98ns	5.13 ns
지연 사이클 (제산)	29		23.04 (평균)
총 지연 시간	240	144	118

비교 논문은 선행 0 예측 기법과 비슷한 ZDC(zero digit count)를 적용하였으며, 본 논문에서 사용한 병렬 배럴 쉬프트(PBS; parallel barrel shift)와 유사한 방식으로 추정되는 방법을 적용하였다. 전체 지연 시간(=사이클 시간×파이프라인 수)은 3.3V 공급 전압에서 17.7ns이다. 본 논문에서 설계된 FP AU는 전체 지연 시간이 16.8ns로 비슷한 공정의 표준 셀 방식을 사용한 비교 논문과 비슷한 속도를 보였다. 특히, 속도 감소 요인이 될 수 있는 비정규화 수 처리 부분에 점진적인 언더플로우 예측(GUP) 기법을 적용하여 임계 경로의 지연 시간 증가를 억제할 수 있었다.

비교 논문의 SRT 제산기에 사용된 공정은 0.6μm CMOS 표준 셀 방식을 사용하였다[14]. 따라서, 동등한

조건에서의 비교를 위해 0.6μm 공정을 0.35μm 공정으로 변환 시켰을 경우를 가정하여 비교하였다. 128 비트 가산기의 합성 실험 결과 0.6μm 공정에서 6.92 ns의 지연 시간이 0.35μm 공정에서 4.87ns로 감소하였으므로, 30% 지연 시간의 감소 효과가 있었다. 따라서, 최대 40% 지연 시간 감소가 있다고 가정할 때, Radix-4 SRT 제산기의 지연 시간은 8.3ns에서 4.98(= 8.3 × 0.6)ns로 감소될 것으로 예측된다. 본 논문에서 설계된 VQB 제산/제곱근기는 5.13ns의 지연 시간을 가지므로 다소 느리지만, 제산/제곱근 연산에서 동작 속도를 좌우하는 것은 제곱근 연산이므로 Radix-4 SRT 제산기에 제곱근 연산을 추가한다면 동작 속도는 결코 뒤진다고 할 수 없다. 평균적인 총 지연 시간(= 제산의 지연 사이클 수×사이클 시간)은 Radix-4 SRT가 144ns이고 VQB가 118ns로서, 약 18% 실행 시간 감소를 얻을 수 있었다. 따라서, 본 논문에서 제안한 VQB 제산/제곱근기는 기존 SRT 제산/제곱근기와 동작 속도는 동등하면서 사이클 당 처리율 및 총 지연 시간 면에서 더 우수하다고 할 수 있다.

X. 결 론

본 논문에서는 익셉션 예측과 스톨 방식을 채택하고 비정규화 수 처리의 많은 부분을 하드웨어적으로 지원하며, 새로운 제산/제곱근기를 내장한 고성능 부동 소수점 유닛을 설계하였다. 부동 소수점 연산 처리의 성능 향상을 위해서 PS 형식을 정의하여 단정도 데이터 값 처리 시 1.7배 이상의 성능 향상이 가능하도록 하였으며, 부동 소수점 AU에서는 IEEE 754에서 정의한 비정규화 수 연산 시 점진적 언더플로우 예측 기법을 적용함으로써 추가적인 지연 시간의 증가를 억제하면서 처리가 가능하도록 하였다. 승산과 제산에서는 최소한의 하드웨어로 약 11.2%의 비정규화 수 처리를 하드웨어로 지원하도록 하였다. 정확한 익셉션(precise exception)을 위해서는 익셉션 예측과 스톨 방식을 적용하여 다양한 마이크로프로세서에 이식이 용이하도록 하였으며, 제산에서의 익셉션 예측을 제거하여 그 효율성을 높였다. 제산/제곱근기는 기존의 SRT 알고리즘에서 벗어나, 매 사이클마다 가변 길이의 몫을 얻을 수 있는 VQB 알고리즘을 제안하고 구현하였다. 제안된 VQB 방식은 SRT에서 사용되는 복잡한 테이블이 필요 없으므로 설계와 검증이 모두 간단하다. 동급의 Radix-4 SRT 구현과 비교하여 동작 속도에서는 비슷하나, 사이클 당 평균 2.54~2.74 비트의 몫을 구할 수 있어 총 지연 시간 측면에서 약 18% 우수하다. 합성 결과 일반 조건에서 부동 소수점 AU는 179 MHz, VQB 설계 A는 189 MHz, VQB 설계 B는 157 MHz의 동작 주파수를 얻었다. 이와 같이 본 논문에서 설계된 부동 소수점 AU와 VQB 제산/제곱근기는 높은 동작 주파수와 비정규화 수 처리, 그리고 익셉션 예측과 스톨에 의한 정확한 익셉션 지원하므로 고성능 마이크로프로세서 내장에 적합하다.

참 고 문 헌

[1] Keith Diefendorff, "Pentium III = Pentium II + SSE", Microprocessor Report, pp. 6-11, March 8, 1999.

[2] An American National Standard, "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Std 754, 1985.

[3] 이원, 권호경, 이용환, 이용석, "Radix-4 SRT 알고리즘을 사용한 나눗셈/제곱근 연산기에 관한 연구", 전자공학회논문지, 제 33 권, A 편, 제 9 호, 1996년 9월

[4] Peter Soderquist and Miriam Leeser, "Division and Square Root: choosing the right implementation", IEEE Micro, Vol. 17, No. 4, pp. 833-854, August 1997.

[5] 송인호, 문중석, 정덕균, "비정규화수의 하드웨어 처리 가능한 부동 소수점 곱셈기와 나눗셈기", 대한전자공학회 추계종합학술대회 논문집(B), 제 19 권, 제 2 호, pp. 1570-1573, 1996년 11월

[6] Robert K. Yu and Gregory B. Zyner, "167MHz Radix-4 Floating Point Multiplier", Proceedings of 12th Symposium on Computer Arithmetic, pp. 149-154, 1995.

[7] 이태영, 새로운 제산/제곱근기를 내장한 고성능 부동 소수점 유닛의 VLSI 설계, 연세대학교 전기컴퓨터 공학과, 공학박사학위논문, 1999년 12월

[8] Nobuhiro Ide, et al, "A 320 MFLOPS CMOS Floating-Point Processing Unit for

Superscalar Processors", IEEE Journal of Solid-State Circuits, Vol. 28, No. 3, pp. 352-361, March 1993.

[9] 이성연, 부분 나머지의 비트열 분석을 이용한 새로운 Variable Quotient Bit 제산/제곱근 연산기의 설계, 연세대학교 전기컴퓨터공학과, 공학석사학위논문, 1999년 12월

[10] O. L. MacSorley, "High-Speed Arithmetic in Binary Computers", Proceedings of the Institute of Radio Engineers, 49:67-91, 1961.

[11] IEEE 754 Test Suite, Computer Science Division, University of California, Berkeley, 1983.

[12] John H. Edmondson, et al, "Internal Organization of the Alpha 21164", Digital Technical Journal, Vol. 7, No. 1, pp. 119-135, January 1995.

[13] Guenter Gerwig and Michael Kroener, "Floating-Point Unit in standard cell design with 116 bit wide dataflow", Proceedings of 14th Symposium on Computer Arithmetic, April 1999.

[14] Alberto Nannarelli and Tomas Lang, "Low-Power Divider", IEEE Trans. Computers, Vol. 48, No. 1, January 1999.

저 자 소개

李 泰 永(正會員)

1972년 2월 27일생, 1994년 2월 연세대학교 전자공학과 공학사, 1996년 2월 연세대학교 전자공학과 공학석사, 2000년 2월 연세대학교 전기전자공학과 공학박사, 2000년 3월~현재 현대전자 시스템 IC 디지털미디어 AV팀 선임 연구원. <주관심분야 : 마이크로프로세서 설계, VLSI 설계, 부동 소수점 유닛 아키텍쳐, 영상 압축 알고리즘 구현>

洪 仁 杓(學生會員)

1976년 5월 28일생, 1999년 2월 연세대학교 전자공학과 공학사, 1999년 2월~현재 연세대학교 전기전자공학과 석사과정 <주관심 분야 : 마이크로 프로세서 설계, VLSI 설계>

李 成 淵(正會員)

1976년 2월 9일생, 1998년 2월 연세대학교 전자공학과 공학사, 2000년 2월 연세대학교 전기전자공학과 공학석사, 2000년 3월~현재 아남반도체 FAB 연구소 디자인팀. <주관심 분야: 마이크로프로세서 설계, VLSI 설계, DFT>

李 溶 錫(正會員)

1950년 10월 23일생, 1973년 2월 연세대학교 전자공학과 공학사, 1977년 2월 University of Michigan Electrical Engineering 공학석사, 1981년 2월 University of Michigan Electrical Engineering 공학박사, 1993년~현재 : 연세대학교 기계전자공학부 교수. <주관심 분야: 마이크로 프로세서 설계, VLSI 설계, 캐쉬 아키텍쳐, 부동 소수점 유닛 아키텍쳐, DSP 프로세서 설계>