부동소수점 덧셈 연산기의 저전력화 구조

김윤환, 박인철
한국과학기술원, 대전, 한국
042-866-0845, yhkim@duo.kaist.ac.kr

Low Power Architecture for Floating Point Adder

Y. H. Kim, I.C. Park
KAIST, DAEJON, KOREA
042-866-0845 , yhkim@duo.kaist.ac.kr

Abstract

Conventional floating-point adders have one data-path that is used for all operations. This paper describes a floating-point adder developed for low power consumption, which has three data-paths one of which is selected according to the exponent difference. The first is applied to the case that the absolute exponent difference (AED) of two operands is less than 1, and the second is for 1 < AED < (fraction length + the number of guard bits). Otherwise, the third is used to bypass one operand. By providing the separated data-paths, we can eliminate large amount of spurious transitions caused by unnecessary evaluations. Although the control logic becomes slightly complex and thus consumes a little more power than the conventional adder, the power reduction in the data-paths is so significant that it pays off the overhead in the control logic.

1. Introduction

최근에는 고속화된 프로세서와 더불어 나날이 저전력화된 프로세서 유닛이 요구되고 있다. 부동소수점 덧셈기는 많은 마이크로 프로세서나 DSP 또는 과학적인 연산을 많이 필요로 하는 부동소수점 유닛에 많이 쓰이게 된다. 표-1 에서 살펴볼 수 있듯이 특히 덧셈기는 부동소수점 연산유닛 중에서 많은 프로그램에서 가장 많이 쓰이는 연산으로, 어떤 타겟이 되는 어플리케이션에서 전력소모/성능의 기준으로 스펙이 만족되어야 하는 방향으로 설계를 한다. 이 논문에서 제시되는 구조는 고속의 성능을 가지는 어플리케이션 보다는 저전력을 위한 구조이다. 고속화를 위해서 LZA(Leading Zero Anticipation)[2]등이 많이 쓰이기도 하지만 이 덧셈기는 저전력화를 위한 것이고 그다지 큰 성능을 요구하지 않은 프로세싱을 위한 유닛이므로 고성능을 위한 추가적인 구조는 쓰지 않는다.

저전력화를 위한 방법으로는 공급전압(supply voltage)를 낮추는 방법이 가장 효과적이지만 딜레이(delay)가 커지는 단점이 있다. 그러므로 이를 해결하기 위해서는 병렬화나 파이프라이닝등으로 전체적인 throughput 을 높이는 방법을 사용하기도 한다. 그 다음으로 생각해 볼 수 있는 저전력화 방법으로는 스위칭을 줄이는 방법이 있고, 팬아웃(fan out)을 줄여 커패시턴스 값을 줄임으로써 저전력화할 수 있다.

Function	Benchmark and Program			
	Lattic Filter	SPICE +Trans	SPICE Decomp	knuth Fortran
Add	0.75	1.68	1.47	2.3
Multiply	1.00	1.00	1.00	1.00
Divide	0.083	0.34	0.38	0.38
Square root	-	0.04	-	-

[표-1] 여러 Benchmark program 에서의 명령어 빈도

즉, 다음과 같은 식으로 저전력화에 필요한 요소들을생각해 볼수 있다..

$$P(ower) = \alpha C V_{DD}^2 f \qquad (1)$$

(1)식은 한 노드에서의 전력소모를 나타내는 식이다. α 는 스위칭 확률 , C 는 로드 커패시턴스(load Capacitance), V_{DD} 는 공급전압 , f 는 동작 주파수를 나타낸다.

저전력화를 꾀하기 위해서는 여러 단계에서의 저전력 구현를 생각해 볼 수 있다. 각각의 단계는 Technology, Circuit/Logic, Architecture, Algorithm, System 의 다섯 가지 단계[1]로 나누어 생각할 수 있다. 이 논문에서는 이 다섯 가지의 단계 중에서 쓸데없이 발생하는 스위칭을 줄이기 위한 방법을 Architecture 내지는 Algorithm 단계에서 다루기로 한다.

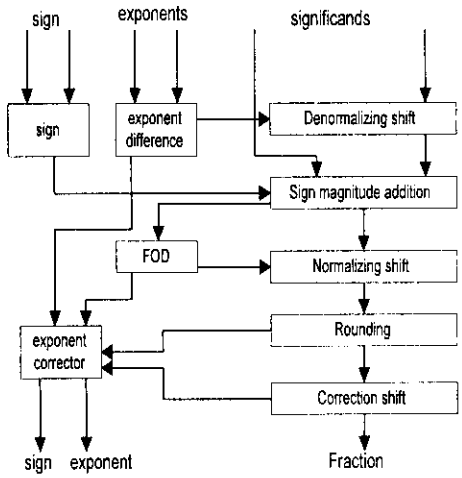
부동소수점 덧셈 연산을 하기 위해서는 일반적으로 pre-alignment, addition, normalization, rounding, and correction shift 의 단계를 거쳐 연산을 완료하게 된다. 저전력화를 위해선 모든 오퍼런드마다 균일하게 위와같은 패쓰를 가지도록 하는 것보단 오퍼런드의 특성에 따라 패쓰를 나누고, 연산을 끝낸 단계에서는 게이팅이나 래치를 사용해 더 이상의 transition 을 막아 저전력화를 도모한다.

2. Floating Point Addition

부동소수점 덧셈연산은 Denormalizing shift, Sign magnitude addition, Normalizing shift, Rounding, Correction shift 의 다섯 단계로 나뉘고, IEEE Standard 754 규약에 따라 설계된다. Denormalizing shift 단계에서는 지수(exponent)가 큰 오퍼런드가 큰 쪽에 맞추어서 지수가 작은 부분을 큰 쪽에 맞추어 연산을 하게 된다. 그러기 위해 우선 작은 지수를 갖는 오퍼런드를 오른쪽으로 shift 하여 지수를 맞춘 후에 연산을 하게 된다. 다음으로 덧셈이나 뺄셈연산을 Addition 단계에서 하게 되며, 그 결과를 다시 Normalizing shift 를 한 후에 Rounding 하여 Standard 에 맞는 연산결과를 도출할 수 있다. Correction shift 는 Rounding 후에 오버플로우(overflow)가 생기면 오른쪽으로 1 비트를 shift 하여 노멀라이징을 하기 위해서이다.

라운딩을 할 때에는 LSB 와 guard, round, sticky 비트를 이용해 라운딩을 하게 되며, Round to nearest even, Round upward, Round downward, 그리고 Round to zero 의 네 가지 모드가 있다.

그림. 1 은 부동소수점 덧셈기의 구조도를 나타낸다. 보통 고속의 성능을 내기 위해서는 addition stage 의 입력을 LZA 와 LZC 를 사용해 Leading Zero 를 예측해 내서 더 빠른 성능을 낼 수 있다. Rounding 단계는 LSB 에 1 을 더하거나 더하지 않는 경우가 생기는데, 만약 파이프라이닝을 위해 위 단계의 가산기를 다시 사용하지 않은 경우에는 또 다른 간단한 가산기가 필요하게 된다. 그러므로 고성능을 요구하지



[그림. 1] 부동소수점 덧셈기의 구조도

않는 경우는 Denormalizing shift 와 Normalizing shift 를 위한 시프터는 하나만을 사용하고 라운딩을 위한 가산기 역시 위 단계의 가산기를 그대로 사용하여 구현할 수 있다.

3.Triple data path in Floating Point Adder

모든 입력 오퍼런드를 같은 경로를 사용하는 것보단 특성에 따라 나눠서 경로를 결정하여 구현할 수 있다. 두 오퍼런드의 지수차이가 소수(Fraction) 부분의 비트 길이보다 길 경우는 굳이 연산을 할 필요가 없게 된다. 이런 경우는 연산을 생략할 수가 있다. 더블 포맷일 경우의 소수 부분의 비트길이는 52 비트,

라운딩을 위한 비트와 히든비트를 포함하면 56 비트의 길이를 갖게 된다. 그리고 MSB 위치에 한 비트를 더 두어 오버플로우를 나타내는 비트를 둔다. 즉, 57 비트를 연산하게 된다. 따라서 지수차이가 6 비트 이상이 되면 연산을 생략할 수 있다. 이것을 하나의 경로로 한다.

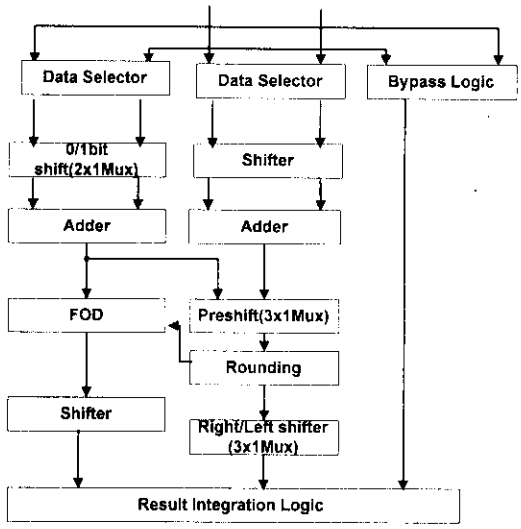
또 다른 경로는 지수의 차이가 0 또는 1 인 경우이다. 이 경로는 처음에 디노멀라이징 시프트단계에서 시프터를 사용하지 않고 먹스(mux)를 사용하여 구현한다. 시프트를 할 때 많은 트랜지션(transition)이 없이 구현된다. 이 경로를 통한 오퍼런드는 가산기를 통한 결과가 덧셈일 경우는 오버플로우나 히든비트 자리에 리딩(leading) 1 이 위치하게 되고 뺄셈의 경우는 리딩 1 이 히든비트 자리에 위치하는 경우도 있지만 대부분 리딩 0 가 3 개이상이 된다. 그러므로 2 자리 이상의 시프트를 하게 된다. 이러한 경우는 라운딩이 필요하지 않으며 단지 왼쪽으로 노멀라이징 시프트만 하면된다. [그림. 2] 에서 왼쪽의 경로가 그를 나타낸다.

마지막으로 지수차이가 2 이상인 경우에는 처음 디노멀라이징 시프트 단계에서 많은 자리의 시프트가 필요하게 되며 이러한 경우 덧셈이나 뺄셈을 하게 될 경우 항상 리딩 1 이 앞의 3 비트 위치에 놓이게 된다. 그러므로 라운딩하기 전에 가산기 결과를 노멀라이징하기 위해 mux 를 통한 preshift 단계를 거침과 동시에 가산기를 사용해 라운딩 로직결과에 따라 소수부분의 마지막 비트 위치에 1 을 더한다. 이를 가산기를 다시 사용하여 구현한다.

왼쪽의 경로의 경우 리딩 1 이 앞의 3 비트자리수 밑쪽에 있게 되는 경우가 낮은 확률로서 생겨날 수 있다. 가산기의 결과를 보고 그러한 경우가 나타날 경우에는 가운데의 preshift 와 라운딩 단계를 거치도록 경로를 수정해야 한다.

라운딩은 LSB 와 가드비트들을 입력으로 받아 구현하게 되고, Round to nearest even, Round upward, Round downward, Round to zero 의 네가지 모드가 있다. Round upward, Round downward, Round to zero 는 결과가 음수인지 양수인지에 따라 증가되거나 증가되지 않는다.

실제적으로 전력소모를 많이 줄일 수 있는 것은 바



[그림. 2] 세 경로로 구성된 구조.

이패스되는 경로를 많이 사용하게 되는 경우이다. 이러한 경우는 무작위로 오퍼런드가 들어올 경우에는 많은 비중을 차지하지만 실제 어플리케이션에서는 그러한 빈도가 크지 않다. uniformly distributed exponents 의 경우 더블 프리시전(double precision)일 때 지수의 차이가 소수 부분의 비트 길이 보다 클 확률는 0.95 정도로 상당히 큰 값을 가지게 된다. 지수를 단지 비트 길이 6 의 차이로 한다고 해도 상당히 큰 값을 가지게 된다.

일반적인 구조로 할 경우는 모든 경우에 대해 $2P_{shifter} + 2P_{adder} + P_{FOD}$ 의 전력소모를 가지게 되지만, 세 경로를 가지게 만들면 덧셈의 경우는 거의 가운데의 경로를 사용하게 되고 그 때의 전력소모는, 대략

$$(P_{shifter}+2P_{adder})P(|e1-e2|>1)+P_{adder}P(|e1-e2|\le 1)$$ 만큼의 전력소모를 가진다.

뺄셈의 경우는 지수의 차이에 따라 오른쪽이나 왼쪽의 경로를 사용하게 되는데,

$$(P_{shifter}+2P_{adder})P(|e1-e2|>1)+(P_{FOD}+P_{adder})P(|e1-e2|\le 1)$$

의 전력을 소모한다고 할 수 있다.

mux 에서 소모되는 전력은 고려하지 않았을 때이다. $P(|e1-e2|>1)$ 은 가운데 경로를 사용하게 되는 경우를 나타낸다.

지수의 크기만을 비교하고 소수부분의 크기는 비교하지 않기 때문에 지수의 크기가 같을 경우, 가산기에서의 뺄셈의 결과가 음수가 될 수 있다. 이 경우는 가운데의 경로를 이용하여 양수로 바꿔주고 부호 비트를 바꿔주게 되는데 이 때는 리딩 0 가 많이 생겨서 왼쪽경로의 FOD 와 시프터를 다시 이용해야 할 경우가 생긴다. 이럴 때가 가장 레이턴시가 길어진다.

실제적인 구조는 하나의 가산기와 하나의 시프터로만 구성된다. 즉 라운딩은 가산기를 통해 이루어지고 왼쪽 경로에서의 마지막 노멀라이징 시프트는 같은 시프터를 사용해 면적을 줄일 수 있다.

4.Discussion

전력소모를 많이 줄이기 위해서는 바이패스되는 경로를 갖는 오퍼런드들이 많아야 한다. 실제 어플리케이션에서 이러한 경우가 많이 생길 수 있는지를 알아보아야 한다. 또한 나머지 경로에서 시프터와 가산기에 대한 저전력화가 필요하며, 사용되지 않는 데이터패스 부분은 신호를 게이팅하거나 래치를 시켜 놓음으로써, 쓸데없이 발생할 수 있는 트랜지션을 막아야 한다. 또한 이렇게 경로를 나누어 구성하게 될 경우 이를 콘트롤할 로직이 좀 더 복잡해지게 되며 이에 따른 콘트롤 로직의 전력소모 증가를 파악해야 한다. 그러나, 이는 거의 무시할 만하다고 생각할 수 있다. 실제 위와 같은 구조로 가져갈 경우 하나의 시프터와 가산기를 가지고 사용하게 되므로 파이프라이닝을 하기 위해선 좀 더 복잡한 콘트롤을 요구하게 된다.

5.References

[1] Anantha P. Chandrakasan, Robert W. Brodersen "Low Power Digital CMOS Design" , Kluwer Academic Publishers

[2] "ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetics" ,Los Alamitos.CA, 1985

[3] P.V.K. Pillai, D. Al-Khalili and A.J.Al-Khalili "A Low Power Approach to Floating Point Adder Design", ICCD, 178-185, 1997

[4] Israel Koren "Computer Arithmetic Algorithms", University of Massachusetts, Amherst

[5] Shin Je Tak "A Design of Pipelined Floating-Point Arithmetic Unit", Master's Thesis, Dept. of EE, KAIST, Korea, 1994

[6] Hiroaki Suzuki, Hiroyuki Morinaka, Hiroshi Makino, Yasunobu Nakase, Koichiro Mashiko, Member, IEEE, and Tadashi Sumi, "Leading -Zero Anticipatory Logic for High-Speed Floating Point Addition", IEEE JSSC ,VOL.31, NO.8 1157-1164 ,1996