논문 2011-48SD-10-13

IEEE754-2008을 위한 고속 부동소수점 연산기 설계

(Floating Point Unit Design for the IEEE754-2008)

황 진 하*, 김 현 필*, 박 상 수*, 이 용 석**

(Jinha Hwang, Hyunpil Kim, Sang-su Park, and Yong Surk Lee)

요 약

스마트폰을 비롯한 각종 매체가 발전함에 따라 우수한 성능의 부동소수점 연산기 필요성이 점차 증가하고 있다. 이러한 요구에 따라 이 논문에서는 기본이 되는 덧셈/뺄셈 뿐 아니라 기존보다 향상된 곱셈과 비교, 변환 연산을 수행하는 고속의 단정도/배정도 부동소수점 연산기의 설계를 제안한다. 가장 많이 사용하는 덧셈/뺄셈 연산기는 반올림 연산 시에 병렬화 작업을 수행함으로서 최적화를 구현하였다. 그래픽 연산 등에서 복잡한 수의 행렬연산이 많이 사용되는데, 이를 빠르게 계산하기 위해서 곱셈기 대신에 곱셈 후 덧셈을 수행하는 단일 곱셈-누산기(MAF)를 설계하였다. 분기 명령은 프로그램에서 자주 사용하는 명령으로 비교 연산에 의해 분기 조건이 결정되는데 이 논문에서는 파이프라인이 완료되기 전에 수행된 비교연산의 결과값을 바이패싱함으로서 연산의 수행시간을 감소시켰다. 또한 IEEE754-2008 표준에 추가된 변환연산을 포함하여 설계하였다. RTL 설계를 검증하기 위하여 연산기마다 40만개의 테스트 벡터를 가중치 무작위 방식으로 선별하여 시뮬레이션을 수행하였다. 검증 후에는 삼성 저전력 45nm 공정에서 합성을 수행하여 600MHz의 동작 주파수를 만족하였다. 또한 개선된 FPU와 기존의 FPU와 비교하여 면적의 감소를 확인하였다.

Abstract

Because of the development of Smart phone devices, the demands of high performance FPU(Floating-point Unit) becomes increasing. Therefore, we propose the high-speed single-/double-precision FPU design that includes an elementary add/sub unit and improved multiplier and compare and convert units. The most commonly used add/sub unit is optimized by the parallel rounding unit. The matrix operation is used in complex calculation something like a graphic calculation. We designed the Multiply-Add Fused(MAF) instead of multiplier to calculate the matrix more quickly. The branch instruction that is decided by the compare operation is very frequently used in various programs. We bypassed the result of the compare operation before all the pipeline processes ended to decrease the total execution time. And we included additional convert operations that are added in IEEE754-2008 standard. To verify our RTL designs, we chose four hundred thousand test vectors by weighted random method and simulated each unit. The FPU that was synthesized by Samsung's 45-nm low-power process satisfied the 600-MHz operation frequency. And we confirm a reduction in area by comparing the improved FPU with the existing FPU.

Keywords : Floating-point Unit, Multiply-Add Fused, convertor, ALU, IEEE754-2008

Ⅰ. 서 론

부동소수점 연산기(Floating Point Unit : FPU)는 모바일 단말기나 네트워크 동기 등에서 부동소수점(floating-point) 숫자의 덧셈, 곱셈, 초월함수 연산 등 여러 가지 연산을 수행한다. 최근에 어플리케이션이 발전함에 따라 크고 복잡한 숫자 계산의 필요성이 증가하고 있으며 그에 따라 더욱더 고속의 부동소수점 연산기의 개발이 요구된다. 특히 칩의 설계에서 저전력과 공간적 절약이 강조되면서 나눗셈기, 초월함수 연산기 등을 전부 하드웨어로 구현하기 보다는 소프트웨어가 부동소수점 덧셈과 곱셈을 이용하여 다양하고 복잡한 부동소수점 연산들을 수행하는 형태로 설계되는 방향으로

* 학생회원, ** 정회원-교신저자, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
※ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.2011-0017625).
접수일자: 2011년5월12일, 수정완료일: 2011년10월2일

나아가고 있다.[1] 그러므로 부동소수점 숫자의 덧셈과 곱셈은 부동소수점 연산기 개발에서 매우 중요한 연구 주제이다.

이러한 배경에서 현재 많은 부동소수점 연산기가 개발되고 있는데, 행렬 연산의 중요성으로 인해 행렬연산에 유리한 곱셈 후에 누적 덧셈 연산을 하는 MAC(Multiply ACcumulator) 연산기 개발에 초점이 맞춰져있는 추세이다. 반면에 이에 못지않게 매우 많이 사용되는 컨버터(Convertor)나 비교(Compare) 연산 등을 수행하는 ALU(Arithmetic-Logic Unit)는 연구가 거의 진행되지 않고 있다.

컨버터는 단정도(Single precision), 배정도(Double precision)와 같은 부동소수점 숫자와 정수(Integer) 사이의 변환 연산을 수행하는 연산기이다. 예를 들어 정수와 단정도 숫자의 덧셈을 수행하는 경우와 같이 서로 다른 형식의 숫자끼리의 연산을 수행해야 하는 경우 컨버터가 필요하다. 2008년에 정수와 부동소수점 숫자 간의 변환 기능이 IEEE754 표준에 추가되었는데 이는 그 필요성과 중요도가 과거에 비해 증가하였다는 것을 의미한다. 2008 표준에서 추가된 동작은 총 세 가지인데, 이번에 설계한 부동소수점 연산기에서는 이 중에서 2진수와 10진수간의 변환을 제외하고 부동소수점 숫자와 정수간의 변환과 부동소수점 숫자를 정수로 반올림하는 기능을 구현하였다.[2~3]

비교 연산에는 SEQ(Set EQual), SNE(Set Not Equal), SLT(Set Less Than), SLE(Set Less or Equal), SGT(Set Greater Than), SGE(Set Greater or Equal)가 있으며 분기 명령(Branch instruction)이나 조건 실행(Conditional execution)의 분기 조건을 계산하는데 사용된다. 평균적으로 전체 프로그램의 약 1/5이 분기 명령으로 이루어져있기 때문에 이러한 비교 연산의 사용 빈도는 매우 높다. 따라서 부동소수점 숫자의 비교 연산 역시 다른 연산들 못지않게 중요한 연산이라고 할 수 있다.[4]

이 논문에서 구현한 부동소수점 연산기는 곱셈기 대신에 MAC이 아니라 Multiply-Add Fused(MAF : 단일 곱셈-누산기)를 사용하였다. 두 연산자를 입력 받아 곱셈 결과를 계속 누적하여 덧셈하는 MAC 연산기보다 3개의 연산자를 받아 두 개를 곱하고 그 결과에 나머지 하나를 더하는 연산을 수행하는 MAF 연산기가 행렬 연산에 더 유리하기 때문이다.[5]

이 논문에서 제시하는 연산기는 MAF와 컨버터(CONV), 그리고 고속의 덧셈, 뺄셈, 비교 연산기(ALU)로 구성되어있다. 6단의 파이프라인(Pipeline) 구조이며, 삼성 Low power(LP) 45nm 공정에서 합성하여 성능을 측정하였다.

Ⅱ장 본론에는 부동소수점 연산기의 전반적인 동작과 이를 구성하는 각 모듈에 대하여 자세히 설명 되어 있다. 또한 하드웨어를 최적화하기 위해 사용한 구조나 기술들에 대하여 설명하였다. Ⅲ장 실험에서는 부동소수점 연산기를 45nm 공정으로 합성한 뒤에 얼마큼의 성능을 보이는지, 전체 모듈의 수행 시간이 어떤지에 대하여 기술되어 있으며 연산의 결과값을 검증하는 방법과 검증된 사항에 대하여 설명하고 있다. 마지막 장에서는 논문의 내용을 정리하고 연산기를 설계하며 도출한 결론에 대하여 서술하였다.

Ⅱ. 본 론

1. 모듈 설명

1.1 부동소수점 연산기(FPU)

부동 소수점 연산기는 연산자들과 연산 코드(Opcode), 그리고 반올림 원칙을 입력 받아 연산 코드에 지정된 연산을 처리하여 출력하는 기능을 수행한다. 6단 파이프라인 구조이며 각 파이프라인의 단은 1 사이클(Cycle) 동안 수행된다. 이때 스톨(stall) 신호에 따라서 파이프라인의 진행을 정지시키고 현재 상태를 유지할 수도 있다.

그림 1은 부동소수점 연산기 모듈 전체의 모습을 보여준다. 파이프라인 첫 번째 단에서 입력값이 들어오면 연산 코드를 디코드 하는 것과 동시에 세 모듈에 입력값이 들어간다. 즉, 디코드 부분을 파이프라인의 첫 번째 단에 넣는 것이다. 그렇게 함으로써 파이프라인의 단 수를 한 단 줄일 수 있다. 하지만 연산 코드를 디코

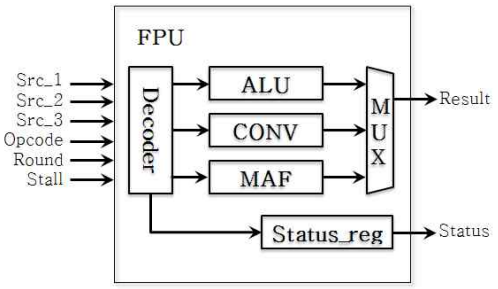

그림 1. 부동소수점 연산기 모듈
Fig. 1. FPU module.

드하지 않은 상태에서 연산에 들어가기 때문에 세 연산 모듈 모두가 입력값을 받아서 연산을 한다. 그리고 마지막에 MUX가 연산 코드를 선택 신호(Select signal)로 받아서 세 모듈에서 나온 결과값들 중에서 적절한 결과값을 선택하게 된다.

부동소수점 연산기의 결과값으로는 세 모듈의 연산 결과 중 선택된 값과 상태 플래그가 있다. 이 중에서 상태 플래그는 결과값이 제로(0), 무한대, 무효 연산, 언더플로우, 오버플로우, 부정확까지 총 6개의 상태 중 하나이거나 복수의 상태일 때 플래그가 올라간다.

파이프라인이 총 6단이기 때문에 6사이클이 지나면 결과값이 나오는데, 비교 연산만은 3사이클만에 완료되며 그 결과가 바이패스(Bypass)되어 나오게 된다. 이 과정에 대한 자세한 사항은 다음 세부 모듈의 ALU부분에 기술되어 있다.

1.2 세부 모듈

1.2.1 Arithmetic-Logic Unit(ALU)

ALU는 부동소수점 숫자의 덧셈/뺄셈 연산과 비교 연산을 수행하는 모듈이다. 그림 2는 ALU의 입력과 출력을 나타내고 있다.

ALU는 두 개의 64비트 연산자와 연산 코드, 반올림 원칙, 스톨 신호를 입력으로 받는다. 또한 덧셈/뺄셈 연산 결과값 64비트와 상태 플래그를 출력으로 내보낸다. 단정도와 배정도 숫자를 연산자로 받을 수 있으며 단정도를 입력으로 할 때는 지수부 11비트 중 하위 8비트를, 가수부 52비트 중 하위 23비트를 사용하고 나머지는 0을 채워 표현한다.

그림 3에서 알 수 있듯이 ALU는 6단의 파이프라인 구조이다. 첫 번째 단에서는 두 개의 64비트 연산자를 입력으로 받아서 각각 부호부(Sign part), 지수부, 가수부로 나눈다. 그리고 연산자가 정규화 수인지, 비정규화 수인지, 제로인지, 무한대인지 체크한다. 체크된 결과는 연산에도 사용되지만 예외 처리나 상태 플래그를 업데

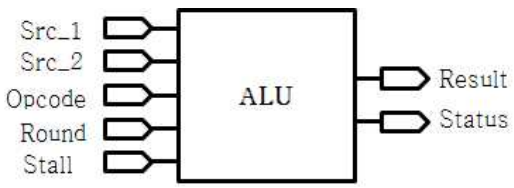

그림 2. ALU의 입출력
Fig. 2. ALU input/output.

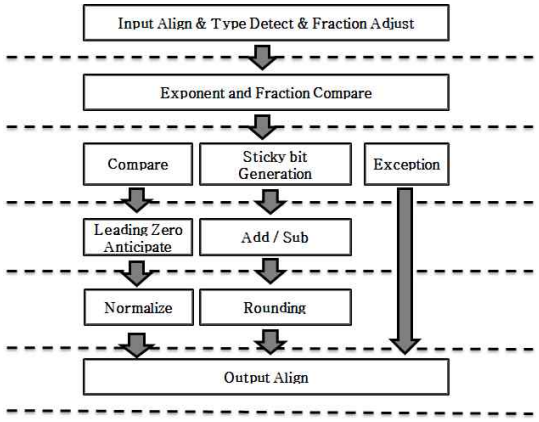

그림 3. ALU의 블록선도
Fig. 3. ALU Block Diagram.

이트하는데도 사용된다. 이어서 체크 결과에 따라 정규화 숫자이면 지수부를 1.XXX...로, 비정규화 숫자이면 0.XXX...로 만들어 주는 작업을 한다. 두 번째 단에서는 두 연산자의 지수부와 가수부의 크기를 서로 비교하고 그 차이를 구하는 연산을 수행한다. 세 번째 단에서는 앞단에서 구한 결과를 가지고 연산 코드에 따라서 다른 연산을 한다. 비교 연산일 경우, 연산을 수행한 뒤 그 결과를 바이패스하여 내보낸다. 비교 연산은 입력된 두 숫자 간의 비교를 수행하여 지정된 연산과 비교 연산이 일치하였을 경우에는 출력값의 최하위 비트를 HIGH(=1)로, 일치하지 않을 경우에는 LOW(=0)으로 출력한다. 덧셈/뺄셈 연산일 경우 두 연산자를 큰 값(Large value), 작은 값(Small value)로 나누고 작은 수를 이용하여 가드, 라운드, 스티키 비트를 생성한다. 또한 예외 처리도 이 부분에서 수행한다. 네 번째 단에서는 덧셈/뺄셈 연산과 Leading Zero Anticipator(LZA)을 병렬적으로 수행하여 덧셈/뺄셈의 결과가 나오자마자 정규화를 바로 할 수 있도록 한다. 이 때 덧셈/뺄셈 연산을 수행하는 덧셈기는 코게-스톤(Kogge-stone) 덧셈기를 사용했다. 코게-스톤 덧셈기는 올림수 예측(Carry look-ahead) 덧셈기에서 올림수를 병렬적으로 처리(Parallel prefix)하여 속도를 극대화한 덧셈기이다.[6] 그리고 LZA라는 것은 정규화를 할 때 수행할 시프트의 양을 미리 예측하는 작업이다.[7] LZA에 대해서는 하드웨어의 최적화 부분에서 좀 더 자세히 설명되어 있다. 다섯 번째 단에서는 정규화와 반올림, 그리고 반올림 후 정규화를 병렬적으로 수행한다. 이 병렬화 기법 역시 하드웨어 최적화를 설명하는 부분에 좀 더 자세히 설명되어 있다. 여섯 번째 단에서는 앞 단의 정규화, 반

올림, 반올림 후 정규화 모듈의 결과값 중에서 적절한 결과값을 선택하여 출력한다. 이 때 상태 플래그도 함께 출력된다.

1.2.2 Convertor(CONV)

CONV은 부동소수점 단정도/배정도 숫자, 그리고 정수 사이의 변환 연산을 수행하는 모듈이다. 그림 4는 CONV의 입력과 출력을 나타내고 있다.

CONV은 64비트 연산자로 부동소수점 숫자나 정수를 입력으로 받는다. 그리고 7비트 연산자로 소수점의 위치를 입력으로 받는다. 그 외의 입출력 사항은 ALU와 동일하다.

CONV은 그림 5에서 알 수 있듯이 6단의 파이프라인 구조로 다음과 같은 세 가지 유형의 변환 작업을 수행하게 된다.

① 정수를 부동소수점 숫자로 변환하는 경우.
② 부동소수점 숫자를 정수로 변환하는 경우.
③ 부동소수점끼리 변환하는 경우.

첫 번째 단에서는 ①, ②, ③ 모두 ALU와 마찬가지로 입력된 연산자값의 타입을 확인한 후 내부 형식(Internal format)으로 변환하여 정렬을 수행한다. 내부

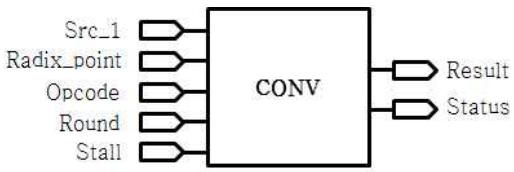

그림 4. CONV의 입출력
Fig. 4. CONV input/output.

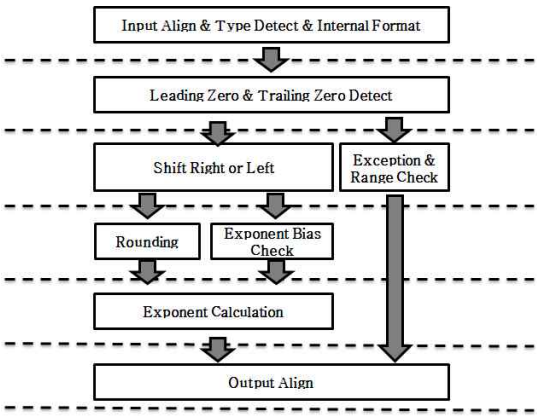

그림 5. CONV의 블록선도
Fig. 5. CONV Block Diagram.

형식이란 IEEE 754-2008 표준에 새로 생긴 확장 형식(Extend format)을 변형시킨 자체적인 형식이다. ②의 경우에는 지수부에서 편향값을 빼서 진짜 지수값을 추출한다. ③의 경우에는 단정도에서 배정도로 변환할 때는 지수부에서 편향값을 빼주고, 배정도에서 단정도로 변환할 때 더해준다. 두 번째 단에서는 ①의 경우에는 리딩 제로(Leading zero)를, ②, ③의 경우에는 트레일링 제로(Trailing zero)를 카운트 한다. 리딩 제로는 최상위 비트로부터, 트레일링 제로는 최하위 비트로부터 최초로 1이 나올 때까지의 0의 개수를 세는 것이다. 세 번째 단에서는 정규화를 수행한다. ①의 경우에는 리딩 제로 카운트(Leading zero count)를 받아 왼쪽으로 시프트(left shift) 연산을 수행하고, ②의 경우에는 트레일링 제로 카운트(Trailing zero count)를 받아 오른쪽으로 시프트 연산을 수행한다. ③의 경우에는 단정도 비정규화 숫자를 배정도 정규화 숫자로 변환하는 경우에는 왼쪽으로 시프트를, 반대의 경우 오른쪽으로 시프트를 수행한다. 동시에 예외 처리를 수행한다. 네 번째 단에서는 반올림 원칙에 따라 반올림을 실시한다. 그리고 ①의 경우 이와 병렬적으로 출력 형태가 단정도인지 배정도인지를 구분하여 단정도일 때는 127, 배정도는 1023을 편향(Bias)값으로 더한다. ②, ③의 경우에는 오버플로우를 체크한다. 다섯 번째 단에서는 ①, ③의 경우에 앞 단의 반올림 과정에서 가수부에 오버플로우가 발생했는지의 여부에 따라 지수부를 보상해주는 동작을 수행하게 된다. 그리고 지수부를 보상함으로서 오버플로우가 발생했는지를 확인한다. ②의 경우에는 결과값을 그냥 넘겨준다. 여섯 번째 단에서는 출력 형식을 확인하여 결과값과 상태 플래그를 출력한다.

1.2.3 Multiply-Adder Fused(MAF) unit

MAF는 세 개의 입력값을 받아서 A×B+C의 연산을 수행하는 모듈이다. B값이 1이라면, A+C가 되어 덧셈을 수행할 수도 있고, C값이 0이라면 A×B가 되어 곱셈을 수행할 수도 있다. 일반적으로 A×B+C를 수행할 때 가장 문제가 되는 부분은 곱셈 후 반올림과 정규화를 한 번 수행하고, 덧셈 후 반올림과 정규화를 한 번 더 수행한다는 것인데, 이를 해결하기 위해서 161비트 정렬 시프터(Alignment Shifter)를 사용하여 문제를 해결하였다.[5] 이에 대한 자세한 설명은 하드웨어의 최적화 부분에 자세히 기술되어있다. 그림 6은 MAF의 입력과 출력을 나타낸다.

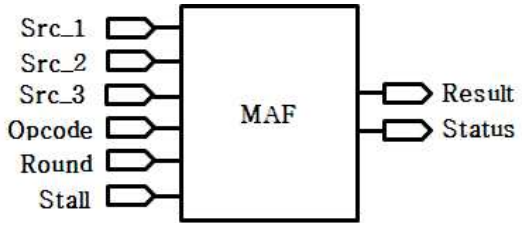

그림 6. MAF의 입출력
Fig. 6. MAF input/output.

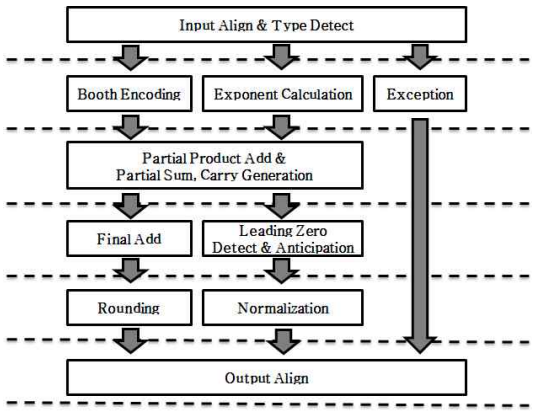

그림 7. MAF의 블록선도
Fig. 7. MAF Block Diagram.

MAF의 세 개의 64비트 연산자를 입력으로 받는다. 그 외의 입출력 사항은 ALU와 동일하다.

그림 7는 MAF의 블록선도이다. MAR는 앞의 ALU와 CONV과 마찬가지로 6단 파이프라인 구조이다. 첫 번째 단에서는 입력된 연산 코드로 연산자들을 부호부, 지수부, 가수부로 나누고 각각의 타입을 구분한다. 두 번째 단에서는 디코딩(Decoding)한 명령어를 바탕으로 지수부는 지수 간의 덧셈을 수행하고 가수부에는 Radix-4가 아닌 Modified radix-8 부스 인코딩(Booth encoding)을 수행한다. Radix-4를 사용하면 Radix-8에서 발생하는 3X의 문제는 없지만 부분 합(Partial product)이 54/2 = 27개나 생성된다. 하지만 Radix-8을 사용하면 54/3 = 18개의 부분 합만 생성되기 때문에 Radix-8을 사용하였다.[8] 부스 인코딩과 동시에 다음 단에서 최종적으로 누적 덧셈 되는 부분 합의 소수점 자리를 고정하기 위해 시프트를 얼마나 할 것인지를 결정한다. 세 번째 단에서는 부스 인코딩 결과 생성된 부분 합들을 월레스 트리(Wallace tree)를 이용하여 덧셈을 수행하고, 최종적인 누적 덧셈을 위해 161비트 3:2 CSA(Carry Save Adder)로 덧셈 연산을 수행하여 A×B를 마무리한다. 이 때, 마지막 부분 합에 앞 단에서 결정한 시프트의 크기만큼 시프트를 수행하여 소수점을 고정한다. 네 번째 단에서는 C를 더해주는 최종 덧셈을 수행한다. 이를 빠르게 수행하기 위해 ALU에서도 사용했던 코게-스톤 덧셈기를 사용하였다. 그리고 ALU와 마찬가지로 덧셈 연산 후 정규화를 바로 수행할 수 있도록 위해 LZA를 덧셈과 병렬로 수행한다. 다섯 번째 단에서는 반올림과 정규화를 병렬로 수행하며 마지막 여섯 번째 단에서는 출력 형식을 확인하여 결과값과 상태 플래그를 출력한다.

2. 최적화

2.1 병렬 구조

2.1.1 Leading Zero Anticipator(LZA)

정규화를 수행하기 위해서는 가수부를 코게-스톤 덧셈기로 계산한 뒤에 최상위 비트로부터 리딩 제로를 세야하는 작업이 필요하다. 기존의 연산기 구조에서는 이와 같은 작업을 순차적으로 수행하는 구조였다.

하지만 이는 시간적으로나 공간적으로 낭비이기 때문에 이 논문에서는 ALU와 MAF에 LZA를 사용하여 병렬화하였다.

LZA는 최상위 비트부터 최초의 1까지 연속된 0의 개수를 예상할 수 있는 모듈이다. 기본적인 원리는 캐리(Carry)를 발생시키는 함수와 전달하는 함수, 그리고 끊는 함수를 이용하여 결과값이 양수일 때와 음수일 때를 나눠서 예측하는 것이다.

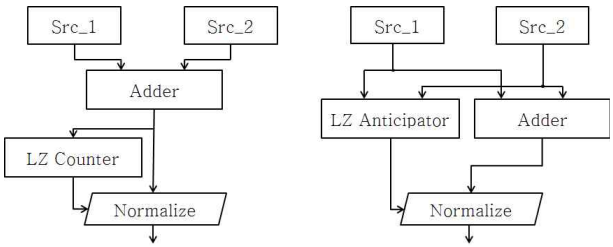

그림 8. 기존 LZ Counter 구조와 LZA 구조
Fig. 8. Existing LZ Counter architecture versus LZA architecture.

2.1.2 정규화과 반올림, 그리고 반올림 후 정규화의 배타성

반올림이 수행될 경우에는 정규화를 수행할 필요가 없거나, 좌우 1비트 시프트의 간단한 정규화를 거친 후 반올림이 수행된다. 또한 정규화가 수행될 경우에는 반올림을 수행할 필요가 없다. 따라서 반올림과 정규화는 동시에 병렬적으로 처리 가능하다.[9]

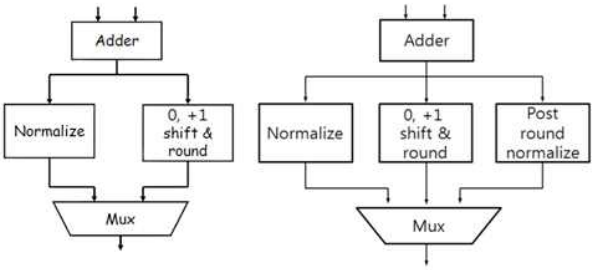

그림 9. 반올림, 정규화의 배타성과 반올림 후 정규화의 배타성을 추가한 구조
Fig. 9. The parallel rounding, normalization architecture vs. the parallel rounding, normalization and post-round normalization architecture.

이를 좀 더 확장하여 반올림 후 정규화와 정규화, 그리고 반올림 사이의 배타성을 찾을 수 있다. 반올림 후 정규화가 필요한 경우는 반올림 과정 중 라운드업이 일어났을 때, 입력 가수부가 1.1111...11일 때만 발생한다. 이 때 실제 반올림을 수행하여 +1 덧셈을 수행하는 것이 아니라 1.0000..00을 출력 가수부로 발생시켜주고, 지수부에 +1을 수행하는 것으로 반올림 후 정규화가 수행될 수 있다. 정규화에서는 라운드업 상황 자체가 발생하지 않으므로 이 과정은 정규화와 배타성이 있다. 또한 반올림 원칙이나 라운드 비트, 스티키 비트, 또는 가수부의 최하위 비트에 영향을 받는 것은 반올림 과정과 동일하지만, 실제로 반올림이 수행되는 것은 아니기 때문에 반올림 과정과도 배타성이 있다. 따라서 정규화와 반올림, 그리고 반올림 후 정규화는 동시에 병렬적으로 처리 가능하다.

2.2 161비트 정렬 시프터(161 bits Alignment Shifter)

MAF에서 부동 소수점 숫자의 A×B+C 연산을 수행할 때 가장 크게 문제가 되는 것은 반올림과 정규화를 두 번씩 수행하게 되어 수행시간이 증가하고 정확도가 떨어진다는 점이다. 이러한 이유 때문에 A×B의 부분합을 최종적으로 더해줄 때 C까지 함께 더해주게 되는데, 이때 C의 소수점 위치는 지수의 편향값을 반영하여 배정도 기준으로 -1023에서 +1023만큼 시프트되는 경우가 발생한다. 하지만 이를 위해서는 ±1023만큼 시프트할 수 있는 배럴 시프터(Barrel shifter)가 필요한데 이는 많은 하드웨어 자원을 필요로 하는데다 느리다는 단점이 있다.

이런 문제를 해결하기 위해 A×B의 결과와 C의 데이터 중 유효한 결과를 산출하는 161비트 정렬 시프터를 지수부에 사용하여 MAF를 설계하였다. 가수부에서 보면 기본 52비트에 정규화를 위해 숨겨진 1비트와 부호를 위한 1비트를 더해 54비트가 주어진다. A×B로 인해 결과값은 108비트가 된다. 이 때 C를 시프트 해주는 범위가 108비트 앞뒤로 53비트씩 가능하므로 108+53=161비트만 유효하다. 따라서 161비트 배럴 시프터로 모든 시프트 연산이 가능하다.

Ⅲ. 실 험

1. 합성 결과

그림 10의 왼쪽에 명시된 수치는 45nm 삼성 LP 공정에서 합성했을 때 세 모듈의 각 단계별 소모된 지연시간을 나타낸다. 수치를 보면 알 수 있듯이, ALU의 세 번째 단이 0.72ns, CONV의 네 번째 단이 0.73ns, MAF의 세 번째 단이 0.89ns로 최대 지연 시간을 가지는데, 모두 600MHz 동작속도에서 제로 와이어 부하 모델(Zero Wire Load Model)로 45%의 마진(Margin)을 고려한 0.9185ns 이내의 패스 딜레이(Path delay)를 가지기 때문에 600MHz의 동작 속도를 만족함을 알 수 있다. 따라서 설계한 부동소수점 연산기는 본 논문에서 목표로 하는 하드웨어 스펙을 충분히 만족한다.

그림 11, 12, 13, 14는 기존의 FPU와 본 논문에서 설계한 FPU의 면적 gate수를 측정하여 비교한 표이다. 기존의 FPU의 gate수를 구하기 위하여 부동소수점 덧셈/뺄셈기, 비교기, 변환기, MAC의 DesignWare IP 블럭들을 합성하여 비교하였다. DesignWare IP란 Synopsys사의 합성툴인 Design Compiler에서 주어지

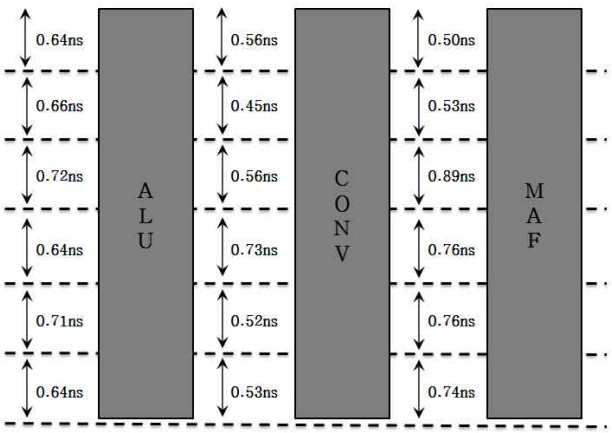

그림 10. 파이프라인 단계의 시간 지연
Fig. 10. Delay time of pipeline stages.

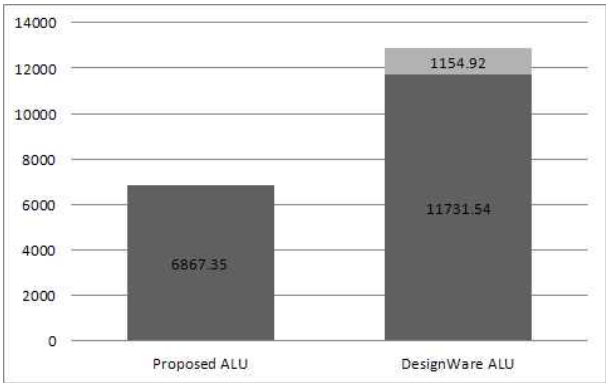

그림 11. 제안된 ALU vs. 기존의 ALU
Fig. 11. Proposed ALU vs. Existing ALU.

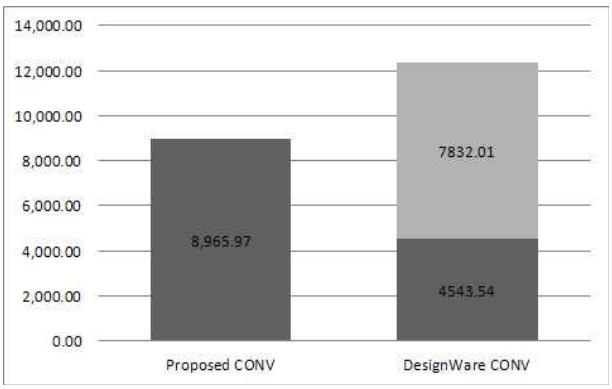

그림 12. 제안된 CONV vs. 기존의 CONV
Fig. 12. Proposed CONV vs. Existing CONV.

는 라이브러리로, 덧셈, 뺄셈, 곱셈 등의 기능 블럭을 합성할 때 사용되는 최적화된 IP이다.[10] 기존의 ALU는 DesignWare의 덧셈/뺄셈기와 비교기를 선택하였다, 기존의 CONV는 DesignWare의 정수를 단정도/배정도 부동소수점 숫자로 변환하는 유닛과 단정도/배정도 부동소수점 숫자를 정수로 변환하는 유닛을 선택하였다. 그리고 기존의 MAF는 DesignWare에 없으므로 MAC을 선택하여 비교를 수행하였다.

그림 11은 제안된 ALU와 DesignWare IP인 DW_fp_addsub와 DW_fp_cmp의 면적의 합을 비교한 그래프이며 47%의 면적 감소를 확인할 수 있다.

그림 12는 제안된 CONV와 DesignWare IP인 DW_fp_flt2i와 DW_fp_i2flt의 면적의 합을 비교한 그래프이며 28%의 면적 감소를 확인할 수 있다.

그림 13은 제안된 MAF와 DesignWare IP인 DW_fpmac의 면적을 비교한 그래프이며 28%의 면적 감소를 확인할 수 있다.

종합해보자면 동일한 길이의 단일 클럭 환경에서 실험한 결과, 개선된 FPU가 기존의 FPU보다 총 20.24%

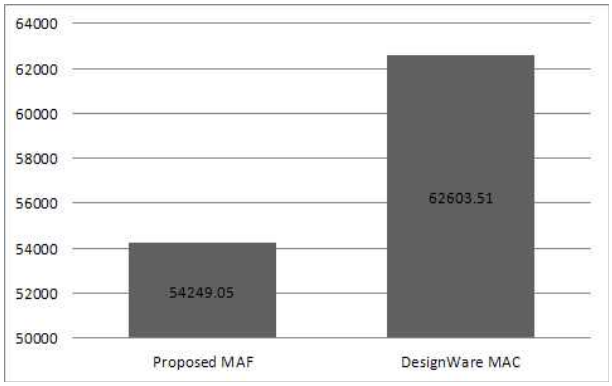

그림 13. 제안된 MAF vs. 기존의 MAC
Fig. 13. Proposed MAF vs. Existing MAC.

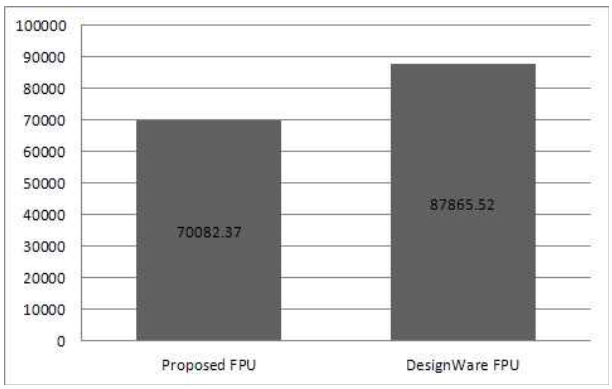

그림 14. 제안된 FPU vs. 기존의 FPU
Fig. 14. Proposed FPU vs. Existing FPU.

만큼 면적이 감소하였다는 결과를 그림 14에서 확인할 수 있다.

2. 검증 방법

논문에서 설계한 부동소수점 연산기가 제대로 동작하는지 검증하기 위해 그림 11과 같은 검증 방법을 사용하였다. 이때 입력 테스트 벡터로 들어가는 데이터들은 그림 12와 같은 방법으로 샘플링하여 각 모듈의 검

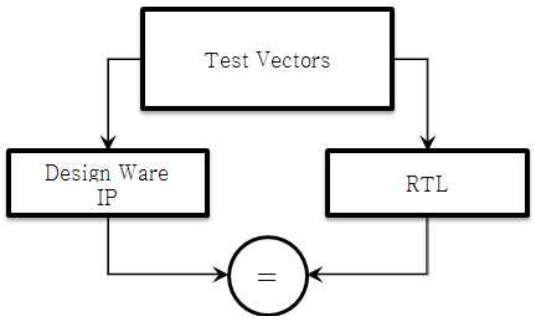

그림 15. 검증 방법
Fig. 15. Verification method.

증을 위해 사용하였다. RTL 시뮬레이션 결과와 비교하기 위한 기준이 되는 값으로 DesignWare IP 블럭의 결과값을 대조군으로 사용하였다.

2.1. Test Vector Sampling

그림 16은 검증을 위해 샘플링한 테스트 벡터의 분포를 나타낸다. 그림에서 회색으로 표시된 부분이 샘플링한 구간인데, 이와 같이 단정도, 배정도의 전 범위로부터 가중치 무작위 방식으로 40만 개의 테스트 벡터를 추출하여 연산기를 검증하였다.

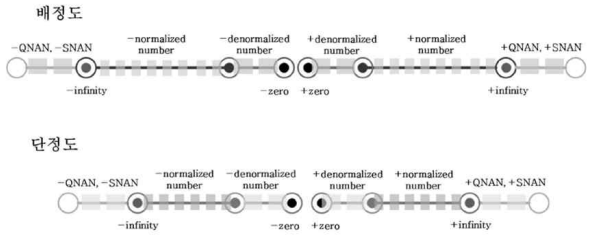

그림 16. 테스트 벡터 추출
Fig. 16. Test vectors sampling.

Ⅳ. 결 론

부동소수점 연산기는 고성능의 컴퓨팅을 요하는 프로세서나 그래픽카드 등에서 다양하게 사용되고 있다. 이러한 이유로 많은 연구가 진행되고 있지만 복잡한 구조로 인해 쉽게 설계가 가능한 모듈이 아니며 따라서 성능 개선의 여지가 있는 연산기이다. 이 논문에서는 단일 곱셈-누산기와 최적화 작업을 거친 ALU와 비교기, 그리고 최신 표준에 추가된 변환을 지원하는 컨버터를 포함하는 부동소수점 연산기를 설계하였다. 연산기마다 40만개의 테스트 벡터를 사용하여 검증하였으며 합성하여 목표했던 600MHz의 동작 주파수를 만족하면서 면적은 기존의 연산기에 비하여 20.24%가 감소하는 것을 확인하였다.

참 고 문 헌

[1] 이영상, 강준우, “다중 칩 수퍼스칼라 마이크로프로세서용 부동소수점 연산기의 설계 (Design of Floating-point Processing Unit for Multi-chip Superscalar Microprocessor).”, 대한전자공학회 학술대회 논문집, 제21권, 2호, p1089-1092, 1998.11

[2] “IEEE Standard for Binary Floating-Point Arithmetic,” ANSI/IEEE Std 754-1985 , vol., no., pp.0_1, 1985.

[3] “IEEE Standard for Floating-Point Arithmetic,” IEEE Std 754-2008 , vol., no., pp.1-58, Aug. 29 2008.

[4] Israel Koren, “Computer Arithmetic Algorithms”, 2nd edition, Prentice-Hall, New Jersey, 2001.

[5] Erdem Hokenek, Montoye, R.K., Cook, P.W., “Second-Generation RISC Floating Point with Multiply-Add Fused”, IEEE Journal of Solid-State Circuits, vol.25, no.5, pp.1207-1213, Oct 1990.

[6] Kogge, Peter M., Stone, Harold S., “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations”. Computers, IEEE Transactions on, vol.C-22, no.8, pp.786-793, Aug. 1973.

[7] Suzuki, H., Morinaka, H., Makino, H., Nakase, Y., Mashiko, K., Sumi, T., “Leading-Zero Anticipatory Logic for High-Speed Floating Point Addition”, Solid-State Circuits, IEEE Journal of , vol.31, no.8, pp.1157-1164, Aug 1996.

[8] Goto, G., Inoue, A., Ohe, R., Kashiwakura, S., Mitarai, S., Tsuru, T., Izawa, T., “A 4.1-ns compact 54×54-b multiplier utilizing sign-select Booth encoders,” Solid-State Circuits, IEEE Journal of, vol.32, no.11, pp.1676-1682, Nov 1997.

[9] Yong Surk Lee, “A 4 Clock Cycle 64X64 Multiplier with 60 MHz Clock Frequency.”, 대한전자공학회 영문논문지 KITE JOURNAL, 61-68, 1991년 12월

[10] Synopsys homepage , http://www.synopsys.com/dw/buildingblock.php, (September 24, 2011), Data path and Building Block IP

저 자 소 개

황 진 하(학생회원)
2010년 연세대학교 전기전자공학과 학사.
2010년~현재 연세대학교 전기전자공학과 석사 과정.

<주관심분야 : 마이크로 프로세서, 네트워크 프로세서, 고성능 연산기 설계, SOC>

박 상 수(학생회원)
2008년 충남대학교 전기정보통신공학부 학사
2010년~현재 연세대학교 전기전자공학과 석사 과정.

<주관심분야 : 마이크로 프로세서, 네트워크 프로세서, 고성능 연산기 설계, SOC>

김 현 필(학생회원)
2005년 연세대학교 전자공학과 학사.
2005년~현재 연세대학교 전자공학과 석박사 통합 과정.

<주관심분야 : 네트워크 프로세서, SVC, 컴퓨터 아키텍쳐, 멀티미디어 프로세서>

이 용 석(정회원)
1973년 연세대학교 전자공학과 학사.
1977년 University of Michigan, Adnn Arbor 전기공학과 석사.
1981년 University of Michigan, Ann Arbor 전기공학과 박사.
1993년~현재 연세대학교 전기전자공학과 교수
<주관심분야 : 마이크로 프로세서, 네트워크 프로세서, 고성능 연산기 설계, SOC>