근사 덧셈을 사용하는 SIMD 포화 덧셈기

[1]윤준기[0], [2]오형철

[1]고려대학교 대학원 전자정보공학과
[2]고려대학교(서창) 공학부
ohyeong@korea.ac.kr

SIMD Saturation Adder using Approximate Addition

[1]Jun-Ki Youn[0], [2]Hyeong-Cheol Oh

[1]Dept. of Elec. & info. Eng., Graduate School, Korea Unversity
[2]School of Engineering, Korea University at Seo-Chang

요 약

0.18μm 표준 셀 라이브러리로 구현할 때 2.69 ns의 임계 경로 지연을 가지는 SIMD구조의 포화 덧셈기를 설계하였다. 기존의 설계에서 임계 경로를 구성하는 CLA를, 8비트 까지만 자리올림(Carry)이 전파될 때 정확한 계산을 보장하는 근사 덧셈기의 형태로 설계한 결과, 임계 경로 시간 지연을 약 22% 감소시킬 수 있었다. 파이프라인 구조 프로세서에서 사용될 포화 덧셈기의 근사계산이 실패하는 경우에는, 추가적인 2개의 클록주기 동안 재계산을 수행하게 된다.

I. 서 론

최근에 미디어 정보의 수요가 급증하면서, 미디어 데이터를 효율적으로 빠르게 처리하는 연산기에 대한 관심이 증가하고 있다. 본 논문에서는 다양한 미디어 데이터를 효율적으로 처리할 수 있도록 근사 산술 기법[1]을 적용시킨 SIMD구조의 포화 덧셈기를 설계하였다.

SIMD구조 연산기[2]는 8비트나 16비트 단위의 영상 또는 음성 데이터를 효율적으로 처리할 수가 있으며, 포화 덧셈기[3]는 연산의 결과가 Wrap Around되어 그 결과 값이 의미했던 바와 반대가 되는 것을 방지하고 포화연산을 수행함으로써 데이터의 손실을 최소화 시킨다.

본 논문에서는 0.18μm 표준 셀 라이브러리로 구현할 때, 임계 경로 지연이 3ns 이하인 SIMD구조의 포화 덧셈기를 설계하는 과정에서 근사 산술 기법을 적용한 결과를 소개한다. 설계된 포화 덧셈기는 SIMD구조의 성질을 이용하여, 자리올림(Carry)이 자연적으로 차단될 수 있는 중간 단계까지만 정확한 계산을 보장하는 근사 산술 기법을 적용함으로써 추가 비용을 최소화하였다. 설계된 포화 덧셈기는 0.18μm 표준 셀 라이브러리로 구현되었으며, 근사 산술 기법을 적용하지 않은 설계와 비교하여 임계경로지연을 약 22% 감소시킬 수 있음을 보였다.

다음 절에서는 설계한 SIMD구조 포화 덧셈기의 구성과 설계 과정에서 내린 설계 선택들을 설명한다. 설계된 포화 덧셈기를 제III절에서 평가하고, 제IV절에서 결론을 맺는다.

II. SIMD 포화 덧셈기

그림 1에 보인 바와 같이, 설계된 SIMD 포화 덧셈기는 *Overflow detection logic*에서 검출된 오버플로우 여부에 따라, CLA의 결과 혹은 포화 결과 값을 선택하여, 4개의 8비트로 구성된 결과값을 출력하도록 구성되어있다. 나머지 블록들은 계속되는 절에서 설명한다.

II-1. SIMD CLA(carry look-ahead adder)

SIMD CLA는 입력되는 2개의 32비트 피연산자에 대한 덧셈과 뺄셈을 수행하거나, 4개의 8비트 연산 또는 2개의 16비트 연산을 수행할 수 있다. 그림 2는 설계된 SIMD CLA의 블록도를 보인 것이다.

그림 2에서 볼 수 있듯이, 설계된 SIMD CLA는 4개의 8비트 CLA를 이용하여 덧셈 및 뺄셈을 수행하는 구조를 갖는다. 8개의 8비트 피연산자($x0$ ~ $x7$)를 입력받

아, 8/16/32비트 연산을 선택하기 위해서 2비트의 제어 신호(*addsub ctrl*)를 사용하고, *Carry Control Logic*에서 연산의 크기에 따라 자리올림의 전파를 허용하거나 차단하도록 하였다.

II-2. 포화 덧셈기(Saturation Adder)

포화 덧셈기는 오버플로우 발생시 Wrap Around되는 것을 방지하고 오차값을 최소화하기 위한 포화 연산 기능을 제공한다. 지정된 크기로 나뉜 각 필드마다 포화 연산을 수행하기 위하여 포화 연산 회로는, [2]에서와 같이, SIMD 가감산기에서 각 필드별로 발생시키는 오버플로우 신호와 두 소스 오퍼랜드의 부호를 입력으로 받아 포화된 결과를 생성한다.

그림 3에 보인 바와 같이 설계된 포화 덧셈기는 각각 8bit 구조의 포화 값을 출력하는 구조로 구성되어 있다. 예를 들어 8비트 연산을 살펴보면, 규칙에 따라 오버플로우 여부를 검출하여, 양의 오버플로우가 발생한 경우에는 양의 최대값인 '01111111'을, 음의 오버플로우가 발생한 경우에는 음의 최소값인 '10000000'을 출력시킨다.

II-3. Approximation Adder

근사 산술 덧셈기는 중간 단계에서 자리올림의 전파를 차단하여 연산을 수행하도록 하였다.

그림 4는 16번째 비트에서 자리올림의 전파를 차단하여 동시에 2개의 16비트 가감산을 수행하는 구성을 도시한 것으로서, 32비트 전체 연산에서 자리올림의 전파에 걸리는 지연시간을 단지 16비트의 자리올림의 전파에 필요한 지연시간만으로 감소시킬 수 있음을 보여준다.

만일, 차단된 곳에서 자리올림이 발생했을 경우에는 틀린 결과 값이 산출될 수 있으므로, 이를 검출하기 위한 회로(*Carry Detection Logic*)를 둔다. 틀린 결과 값이 산출된 경우에는, 그 결과 값을 버리고 2단의 파이프라인 계산을 통하여 근사 기법을 사용하지 않고 계산을 하도록 설계하였다.

III. 성능 비교

본 논문에서 설계된 덧셈기들을 Verilog HDL로 모델화하고 Mentor사의 Modelsim을 이용하여 기능을 검증하였으며, Synopsys사의 Design Compiler와 삼성 STD130 라이브러리를 사용하여 합성하였다.

표 1은 일반 SIMD구조의 32비트 포화 덧셈기와, 32비트 수를 2개의 16비트 수, 또는 4개의 8비트 수로 나누어 근사 산술 방식을 적용한 SIMD 포화 덧셈기들의 임계 경로 지연을 비교한 것이다. 근사 산술 기법을 사용하지 않은 포화 덧셈기에 비하여, 16비트 근사 산술 방식을 적용한 포화 덧셈기의 경우 약 8.4%정도, 8비트의 경우에는 약 22%정도의 속도 향상을 보이고 있다.

구 분	SIMD Adder (32bit)	SIMD Approximate Adder(16bit)	SIMD Approximate Adder(8bit)
임계경로지연 [ns]	3.46	3.17	2.69

표 1. 근사 산술 방식의 평가

IV. 결론

본 논문에서 제시한 근사 산술 덧셈기는, 임의의 위치에서 지정된 비트 수에 걸쳐 전파하는 자리올림을 보장하는 일반적인 근사 산술기[1]와는 달리, 정해진 중간 위치에서의 전파가 없는 경우에만 결과 값을 보장한다. 이와 같은 제약으로 추가 비용을 현저하게 절약하는 대신, 결과 값이 틀리는 빈도가 높아지게 되겠다. 그러나, 본 논문의 덧셈기는 음성 신호와 같이 천천히 변하는 데이터는 매우 잘 처리할 것으로 예측된다.

본 논문에서는 틀린 결과 값이 산출된 경우에는, 그 결과 값을 버리고 2단의 파이프라인 계산을 통하여 근사 기법을 사용하지 않고 재계산을 하도록 근사 덧셈기를 설계하였다. 이와 같이, 틀린 결과 값이 산출되었을 때 발생하는 추가비용을 줄이기 위하여, 단지 1개 클록의 추가 비용으로 올바른 값을 산출하도록 하는 산술기를 현재 개발 중이다. 즉, 16비트 근사 산술 방식을 적용한 포화 덧셈기에 관하여 그림 5에 도시한 바와 같이, 하위 부분의 16비트는 다시 연산을 수행하지 않고 상위 16비트 부분에만 자리올림을 더함으로써 결과값을 산출하는 SIMD구조의 포화 덧셈기를 현재 설계하고 있다.

감사의 글

본 연구는 반도체설계교육센터(IDEC)의 지원을 받아 수행되었습니다.

References

[1] S.-L. Lu. "Speeding up processing with approximation circuits," *IEEE Trans. Comp*, pp. 67-73, 2004.

[2] R. B. Lee. "Multimedia extensions for general-purpose processors," *Proc. IEEE Workshop on Signal Processing Systems*, pp. 9-23, 1997.

[3] M. Schulte, P. Balzola, J. Ruan, and J. Glossner. "Parallel Saturating Multioperand Adders," *Proc. CASES 2000*, p.172-179, 2000.

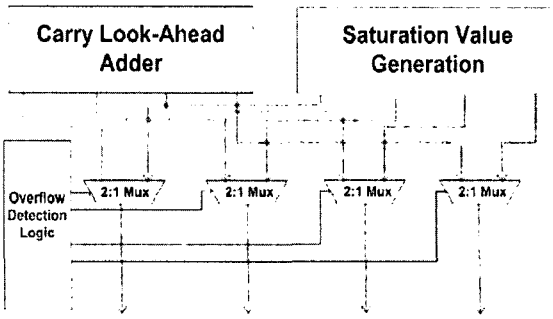

그림 1. SIMD Saturation Adder

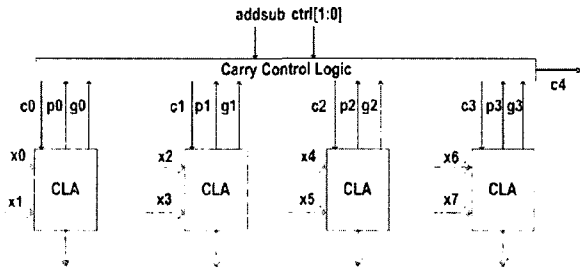

그림 2. SIMD CLA

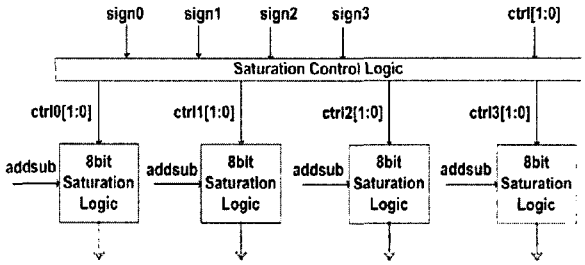

그림 3. Saturation Value Generation Logic

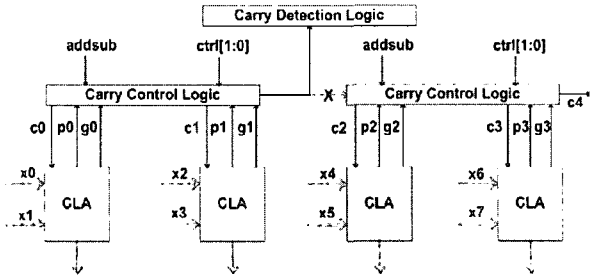

그림 4. Approximation Adder

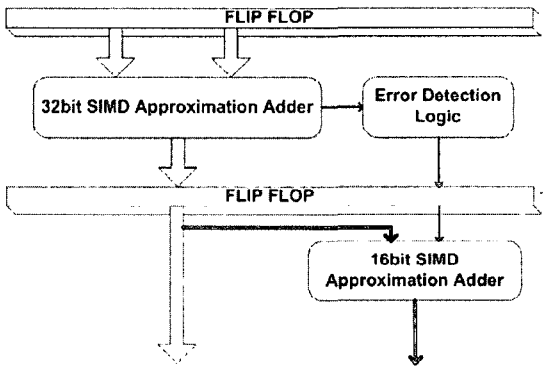

그림 5. Pipelined Approximation Adder