CORDIC연산기 구현을 위한 Bit-level 하드웨어 시뮬레이션

이성수(李聖洙),이정아(李禎娥)
조선대학교 산업대학 전자계산학과

Bit-level Simulator for CORDIC Arithmetic based on carry-save adder

Seong-Soo Lee, ,Jeong-A Lee
Dept. of Computer Science,
College of Industry, Cho-Sun University
Dong-Gu,Seo-suk-Dong, #375
Kwang-Ju city, 501-759
E-mail: jalee@chonnam.chonnam.ac.kr
Voice:062-230-7711
Fax:062-227-8345

CORDIC연산기 구현을 위한 Bit-level 하드웨어 시뮬레이션

Bit-level Simulator for CORDIC Arithmetic based on carry-save adder

요 약

본 논문에서 다루는 내용은 멀티미디어 정보처리시 이용되는 여러 신호 처리용 하드웨어에서 필요로 하는 벡터 트랜스퍼메이션(Vector Transformation)및 오소그날 트랜스퍼메이션(Orthogonal Transformation)에 유용할 뿐만 아니라 여러 형태의 다양한 연산(elementary function including trigonometric functions)을 하나의 단일화된 알고리즘으로 구현할 수 있게 한 CORDIC(COordinate Rotation DIgital Computer)연산[1][2]에 관한 연구이다. CORDIC연산기를 실현함에 있어서 고속 연산을 위해 고속 가산기(fast adder)로서 CSA(Carry Save Adder)를 선택하는데, 본 논문의 연구 초점은 CORDIC연산기를 하드웨어로 실현하기 전에 Bit-Level의 시뮬레이터를 통하여, CSA의 특정상 발생할 수 있는 문제점에 대해 설명하고, 해결 방법[3]을 이용하여 원하는 값에 접근하는가를 확인하여 다양한 Bit의 조작으로 오차의 정도에 따라 유효한 CORDIC연산기를 실현하는데 도움이 되고자 한다.

1. 소개

CORDIC 알고리즘은 볼더(Volder)에 의하여 삼각함수(trigonometric functions), 곱셈 나눗셈(multiplications division)과 좌표 변환(coordinate transfermation)의 계산을 위해서 처음으로 소개되었고 나중에 Walther에 의하여 다양한 연산을 할 수 있게 되었다. 전통적인 계산을 이용한 CORDIC 프로세서의 계산시간[O(n)]과 처리율은 덧셈과 뺄셈시 발생되는 캐리 전달(Carry propagation chain)에 의해[O(n)] 결정된다. Redundant number system을 이용하면 캐리 전달의 제거로 매우 높은 처리율[O(1)]을 얻게 된다.

알고리즘에 의하여 제작되는 하드웨어는 덧셈(addition)에 있어서 합벡터(sum vector)와 캐리 벡터(carry vector)를 추출하는 CSA로 연산이 이루어지기 때문에 전가산기(FA)에서 발생되어지는 캐리 전달에 영향을 받지 않는다. 이러한 높은 처리율은 실시간 신호 처리 응용에 매우 유용하다. 본 논문은 하드웨어에 유용하게 적용할 수 있는 시뮬레이터의 구성으로 16-bit, 32-bit, 48-bit의 다양한 Input data를 수행시켜 결과를 비교하고자 한다.

2. CORDIC의 구조

CORDIC 연산기는 다음과 같은 반복 구조로 표현되어, Z-equation를 통하여 $\tan^{-1}(y/x)$를 구할 수 있다.

for i = 0,n {

$$X[i+1] = X[i] + \sigma i * 2^{-i} * Y[i]$$

$$Y[i+1] = Y[i] - \sigma i * 2^{-i} * X[i]$$

$$Z[i+1] = Z[i] + \sigma i * \arctan(2^{-i})$$

}

where,

$$\sigma i = +1 \text{ if } Y[i] \geq 0,$$

$$-1 \text{ if } Y[i] < 0$$

CORDIC을 구현함에 있어서 기본적으로 필요한 요소는 (그림1)에서 보여진 것처럼 Adder와Shifter이다.

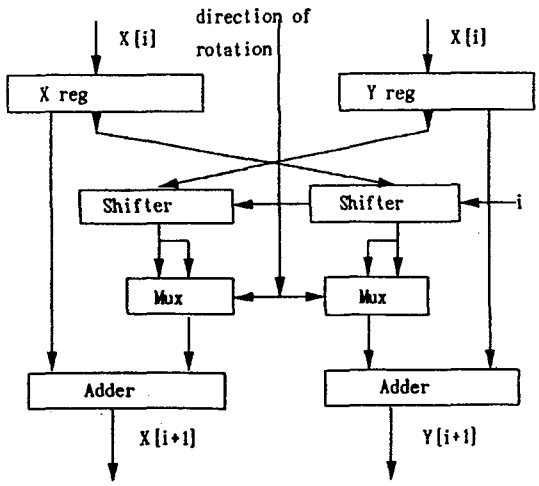


그림1. fig1. Gerneric of CORDIC

(그림1)에서 CSA를 상세히 그리면 다음 그림과 같다.

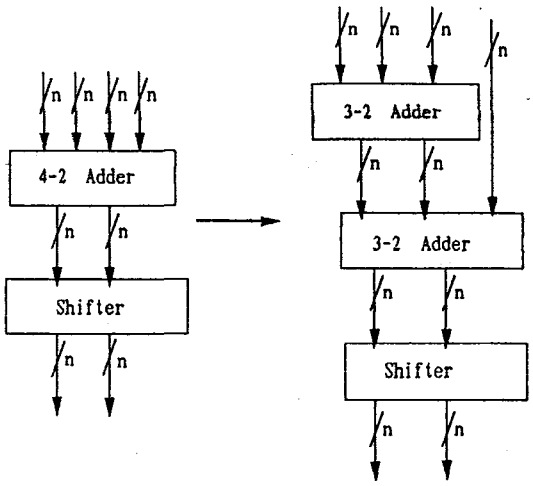


그림2. fig2. CSA of 4-2 Adder &3-2 Adder

CSA에서의 수치표현 방식은 두개의 벡터, 합벡터와 캐리벡터로 이루어진다. 10진수 15를 8-bit output으로 나타내보면 캐리벡터가 00001010이고 그와 쌍을 이루는 합벡터는 00000101이 될 수 있고, 합벡터, 캐리벡터가 00001000, 00000111로 표현될 수도 있다.

3. 문제점과 해결방법

CSA로 CORDIC를 구현할 시 발생하는 문제점은 ASR(Arithmetic shift right)과 관련된 MSBs(Most Significant Bit)와 LSBs(Least Significant Bit)에 관련된다. MSB에서 ASR을 위한 간단한 부호확장은 다음과 같은 문제를 발생 시킨다. (그림2.)에서 CSA의 8-bit output 캐리벡터와 합벡터가 01010011, 00101110일 때, 즉 10000001인 음수 값으로 표현된다 이 두개의 vector를 1 만큼 ASR 한 후의 결과로 양수값 01000000인 합벡터와 캐리벡터 00101001,00010111로 표현된다. 10000001를 ASR 후의 정확한 값은 음수 값인 11000000이다. 여기서 우리는 하위 위치에서 발생되는 캐리 전달(carry propagation chain)로부터 최후의 부호가 결정되므로 MSB의 부호 확장과 관련된 output이 정확한 부호로 지적된다는 것을 보장할 수는 없다. 여기에 유효한 해결방법을 제시한 것이 있다[3]. 해결방법에 대해서 간단히 설명하면 다음과 같다. X_0, X_1, Y_0, Y_1에서 Index 0은 MSB를 나타낸다. X_0, X_1이 01로 시작하고 Y_0, Y_1이 01로 시작한다면 경우1., 경우2.에서 보여진 것처럼 캐리전달과 상관없이 ASR후에는 부호(sign)가 절대적으로 바뀌게 된다.

경우1.

$$\begin{bmatrix} X_0 & X_1 \\ Y_0 & Y_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \text{ shift 후 양수값 } \begin{bmatrix} 00..1 \\ 00..1 \end{bmatrix}$$

그러므로 다음과 같이한다 $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

경우1.과 유사한 아래와 같은 경우도 마찬가지다.

경우2.

$$\begin{bmatrix} X_0 & X_1 \\ Y_0 & Y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \text{ shift 후 음수값 } \begin{bmatrix} 11..0 \\ 11..0 \end{bmatrix}$$

그러므로 다음과 같이한다 $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

주어진 해결방법을 구현할 때는, 최상위 bit에 그 다음 bit를 복사해서 해결한다. 예를 들어, 8-bit의 output에서 합벡터와 캐리벡터가 10111000, 10001110 이면 표현되는 값 01000110 은 10진수로 70이다. 해결방법에 따라 합벡터와 캐리벡터가 00111000, 00001110이 되고 표현되는 값 01000110, 즉 10진수 70에는 변함이 없다.

또 다른 문제는 LSB(Least significant bit)에 관련된 것으로서, 각 벡터에서 잘려나간 합벡터와 캐리벡터가 01110, 10010 이었다면, 잘려나간 부분에서 발생하는 캐리 전달은 반복에 적용되는 실제 값에 영향을 줄 수도 있다. 본 논문의 시뮬레이터는 shifter를 통해 잘려나간 부분에서의 캐리전달이 있는가를 확인하여, 캐리전달이 있을 시에는 LSB에 1을 더하여 반복하게 하였다. 그러나 접근해 가는 부호 즉, σ i에 영향을 미치지 못하므로 LSBs로 인한 오차는 $\tan^{-1}(x/y)$의 결과값에 영향을 미치지 않음을 시뮬레이터를 통하여 확인 하였다.

4. 시뮬레이터의 구성과 데이터의 변환 과정

하드웨어 측면으로, 시뮬레이터는 주 레지스터(Register)로 사용될 n-bit열 저장공간 4개와 Shifter를 위한 보조레지스터로 사용될 n-bit열 저장공간 4개로 시뮬레이터가 구성된다. (그림3.)에서 보는 것처럼 주어진 입력 데이터 X, Y는 각각 캐리레지스터(CX, CY)와 합레지스터(SX, SY)로 저장되고, 4개의 보조 레지스터에 데이터가 복사된다.다음으로 본문3.에서 설명하고 있는 문제에 대한 해결방법을 적용시켜 보조 레지스터에 있는 데이터를 loop index 만큼 ASR(arithmetic shift right)하고, i번째는 Y[i-1]에의해 결정된 부호로 레지스터를 선택하여 2진 보수화한다. 마지막으로 output 데이터가 다시 input 데이

터로 사용되어지는 과정를 (그림4.)에서 보인다.

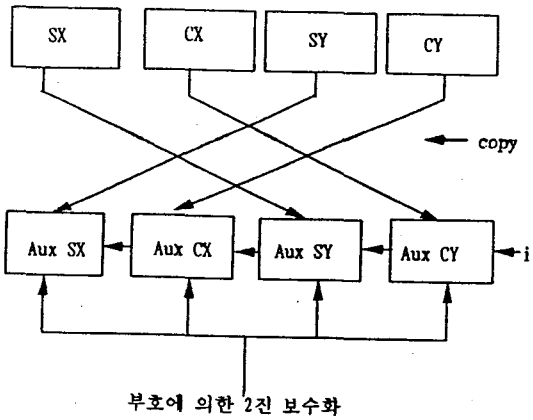
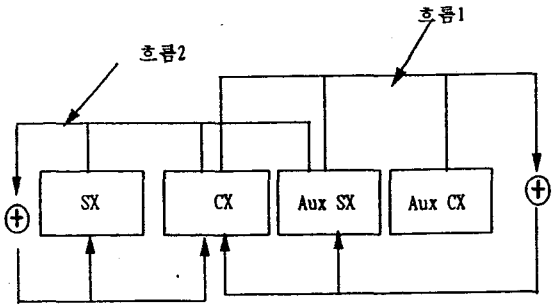


그림3. fig3. 시뮬레이터의 구성과 데이터의 변환과정

CSA는 (그림2.)같이 두개의 3-2Adder로 4-2Adder를 형성하는데, 시뮬레이터는 (그림4.)에서 보여진 것처럼 적절한 연결-보조 레지스터 SX, CX와 레지스터 CX의 데이터를 보조 레지스터 SX와 레지스터 CX에 3-2Adder하여 저장하고[(그림4.)의 흐름1],다시 보조 레지스터 SX와 주 레지스터 SX, CX의 내용을 주 레지스터 SX, CX저장[(그림4.)의 흐름2]-로 output 데이터를 바로 input 데이터로 사용할 수 있고, 중요한 것은 하드웨어 입장에서는 4-2Adder의 pipeline 구현시 추가 사용 레지스터를 필요로 하지 않게 할 수 있다.



(그림4. fig4.) 시뮬레이터의 4-2 Adder

5. 시뮬레이터의 결과

예를 들어 초기조건으로 Z[0] = 0, X[0] = 0.5, Y[0] = 0.5로 하면 각 반복에서 Y[i]의 부호에 따라 σi가 결정이 되고 $\tan^{-1}*(Y[0]/X[0])$값, 이 경우는 45^0값이 Z-반복에 축척이 되어 최종 Z 값에서 $\Pi/4$, 즉45^0값을 얻을 수 있다. 시뮬레이터를 통한 Z-값을< 표1.>에서 보인다.

<표1.> 각 반복에서 얻은 Z값

N bit / Loop Index	16-bit operation	32-bit operation	48-bit operation
N-2	0.78544209478991	0.785398163028305	0.785398163397450
N-1	0.785383174322797	0.785398163969628	0.785398163397460
N	0.785423691900913	0.785398163493966	0.785398163397453
원래의 Z값($\Pi/4$) = 0.7853981633974482			

CORDIC연산기를 구현함에 있어서 <표1>에서 알 수 있듯이 16-bit CORDIC연산기는 소수점 셋째자리까지 유효하며, 32-bit CORDIC연산기는 소수점 아홉째 자리까지, 48-bit CORDIC연산기는 열셋째자리까지 유효하다.

6. 결론.

본 논문에서는 많은 고속의 신호처리 응용에서 나타날 수 있는 CSA의 output를 직접사용하는 것과 관련된 두 가지 MSB와 LSB의 문제에 대해서 논하였다. 그리고 CSA 모듈의 고속 실행 시간에 저해됨이 없는 해결방법을 이용한 시뮬레이터를 구현하였다. 그리고 세 종류의 입력 데이터(16-bit, 32-bit, 48-bit)를 이용하여 시뮬레이터의 output를 조사한 결과 예상했던 값들과 동일함을 보였다.

[1]J.E.volder. "The CORDIC Trigometric computing technique," *IRE Transactions on Electronic computers*, Vol. EC-8, pp.330-334, 1959.

[2]J.S.Walther. "A Unified Algorithms for Elemantary Functions," *Spring Joint Computer Conference Proceedings*, Vol.38, pp.379-385, 1971.

[3] J.A.Lee,T.Lang. "Floating-point Implementation of Redundant CORDIC for QR Decomposition," *Technical Report,Computer Science Department, University of California, Los Angles*, NO.CSD-890044, 1989.