論文2003-40CI-6-1

Redundant Signed Binary Number에 의한 CORDIC 회로

(The CORDIC Circuit of Redundant Signed Binary Number)

金 承 烈 *, 金 容 大 *, 韓 善 景 *, 劉 泳 甲 *

(Seung-Youl Kim, Yong-Dae Kim, Seon-Kyoung Han, and Younggap You)

요 약

Global carry propagation이 없는 redundant signed number에 의한 CORDIC 회로를 제안하였다. 이 number format은 Booth recording과 유사한 새로운 recoding scheme을 가지고 가감산에서 carry 전파의 문제를 효과적으로 해결하였다. 여기서는 상수 scale factor를 갖고 삼각함수 계산을 하는 pipeline 구조를 채택하였다. 이 CORDIC 회로의 동작시간은 채택한 operand bit에 상관없이 일정하다.

Abstract

A novel CORDIC circuit is presented based on a redundant number system eliminating global carry propagation. The number format employs a new recoding scheme similar to the Booth recoding resolving carry problems in addition. A pipelined architecture is introduced having a constant scale factor in its computation of trigonometric functions. The operational time of the circuit is constant independent of the number of operand digits.

Keywords : Redundant signed binary number, CORDIC, Booth recoding

Ⅰ. 서 론

삼각함수를 계산하는 방법으로서 CORDIC 알고리즘이 널리 사용되고 있다[1]. 이 알고리즘의 속도 개선을 위한 연구가 지속적으로 수행되었다. 최근에 CORDIC 알고리즘의 속도를 향상시키기 위한 방법으로 redundant signed digit number를 사용하고 있다[2]. Redundant number의 채택은 carry free addition을 도입할 수 있도록 하여 속도를 개선하는데 중요한 역할을 한다. 그러나 Redundant number를 이용한 CORDIC의 문제로 scale factor의 값이 상수가 아닌 문제가 있다[2, 3]. 이 문제는 scale factor의 보상을 위하여 별도의 회로를 필요로 한다.

Redundant number 체계를 사용하였을 때 또 하나의 중요한 문제는 그 수의 부호를 쉽게 판단할 수 없다는 것이다. 따라서 결국 모든 자리 수의 변환을 통하여 부호를 결정해야 하는 것이다. 이는 redundant number 체계가 가지는 carry 전파 방지의 효과를 상쇄시키는 것이다. CORDIC의 속도는 반복적인 addition 후에 부호 비트 판정에 따라 속도가 결정된다. 결국 기존 binary adder의 carry propagation delay 문제가 내재하고 있다.

이 논문은 redundant number 변환 방식을 개선하여 가감산 후에 그 부호를 즉각 판별할 수 있도록 하였다. 이로써 redundant number 체계가 가지는 carry 전파

* 正會員, 正會員, 忠北大學校 情報通信工學科
(Dept. of Information Communication Engineering, Chungbuk National University.)

接受日字:2003年5月27日, 수정완료일:2003年10月6日

방지의 효과를 충분히 활용할 수 있도록 한 것이다. 이를 위하여 기존의 Booth recoding 기법을 개선하여 적용하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 CORDIC과 redundant number addition의 배경기술에 대하여 설명하였다. Ⅲ장에서는 제안된 CORDIC의 동작에 대하여 자세히 설명하였다. Ⅳ장에서는 제안된 CORDIC의 VHDL coding을 이용하여 설계한 시뮬레이션 결과를 보이며 그 결과에 대하여 설명하였다. 그리고 제안된 CORDIC과 기존 CORDIC의 지연시간을 분석하였다. 마지막으로 Ⅴ장은 결론을 기술하였다.

Ⅱ. CORDIC과 수치체계

제안된 회로의 설계 문제를 다루기 위하여 두 가지의 배경기술을 소개한다. 첫째, CORDIC 알고리즘에 대하여 소개한다. 둘째, redundant number format에 대하여 소개한다. 삼각함수를 계산하는 CORDIC 알고리즘은 이차원 평면상의 벡터의 회전을 이용한다. CORDIC 알고리즘은 기존의 벡터 회전을 이용하여 pseudo rotation을 시키기 위해 다음과 같은 식으로 전개할 수 있다[1].

$$x' = x - y\tan(\phi) \tag{1.1}$$

$$y' = y + x\tan(\phi) \tag{1.2}$$

위 식의 각도 ϕ는 $\phi = \pm\alpha_0 \pm \alpha_1 \pm \alpha_2 \pm \alpha_3 \pm \alpha_4 \cdots \pm \alpha_i$로 분할하여 나타낼 수 있다. 분할된 각도 ϕ는 $\alpha_i = arctan2^{-i}$의 크기로 연속해서 회전시켜 얻을 수 있다. 회전각도 z의 식은 $z_{(i+1)} = z_{(i)} - \alpha_{(i)}$이며 각도는 0 로 접근시킨다[1]. $\tan(\alpha_i) = d_i 2^{-i}$로 치환하고 d_i는 부호비트일 때 다음과 같은 식으로 전개된다.

$$x_{(i+1)} = x_{(i)} - d_{(i)} y_{(i)} 2^{-i} \tag{1.3}$$

$$y_{(i+1)} = y_{(i)} + d_{(i)} x_{(i)} 2^{-i} \tag{1.4}$$

$$z_{(i+1)} = z_{(i)} - d_{(i)} \tan^{-1}(2^{-i}) \tag{1.5}$$

Pseudo rotation에 의해 i=0,1,2,3… 일 때 $x_{(i+1)}$과 $y_{(i+1)}$은 $1/\cos\alpha_{(i)}$만큼 커진다. 이 값을 보정하기 위해 i번의 반복만큼 $1/\cos\alpha_{(i)}$을 곱하여 상수 K factor를 구할 수 있다. 계산된 보정상수 K는 i=7일 때 약 1.6467이 된다. 초기벡터 (1, 0)인 좌표는 K 값을 보정하여 $(1/k = 0.6072,\ 0)$으로 나타낼 수 있다. 그리고 기본적으로 $\cos\phi$, $\sin\phi$는 수식(1.3-1.4)을 이용하여 구할 수 있다. 구하려는 각도 ϕ는 주어진 α의 덧셈 및 뺄셈의 합으로 나타내어진다. 이때 ϕ는 α값의 순서에 상관없이 구하려는 ϕ값에 근사하면 된다. 그리고 x, y의 좌표는 α의 덧셈 및 뺄셈의 회전에 따라 좌표가 변화된다. 이때 x, y좌표가 α값의 순서에 상관없이 주어진 반복횟수 i만큼 회전하여 ϕ만큼 회전하였을 때 x, y의 좌표는 $\cos\phi$, $\sin\phi$를 나타낸다.

이제 수치체계를 보기로 하자. Binary number를 사용한 덧셈은 LSB에서 발생한 carry propagation이 MSB까지 이어진다. 따라서 carry가 digit 전반에 걸쳐 propagation이 발생하는 것을 global carry라 할 것이다. Redundant Signed Binary Adder(RSBA)는 carry propagation을 제거하여 동작속도를 개선한다. Booth recoding 및 RSB recoding을 이용하여 carry free addition을 수행한다. Recoding 특성상 가수 및 피가수의 bit pattern에 연속된 1 및 $\bar{1}$이 존재하지 않는다[5]. 즉 <표 1>의 규칙에 따라 sum을 계산하면 carry propagation이 인접한 상위 비트로 제한되므로 global

표 1. Redundant signed binary number의 덧셈

Table 1. Addition of redundant signed binary number.

a_i	b_i	a_{i-1} b_{i-1}	s_i
0	0	1 1	1
		$\bar{1}$ $\bar{1}$	$\bar{1}$
		X	0
0	1	$\bar{1}$ $\bar{1}$	0
		X	1
0	$\bar{1}$	1 1	0
		X	$\bar{1}$
1	0	$\bar{1}$ $\bar{1}$	0
		X	1
1	1	$\bar{1}$ $\bar{1}$	$\bar{1}$
		X	0
1	$\bar{1}$	X	0
$\bar{1}$	0	1 1	0
		X	$\bar{1}$
$\bar{1}$	1	X	0
$\bar{1}$	$\bar{1}$	1 1	1
		X	0

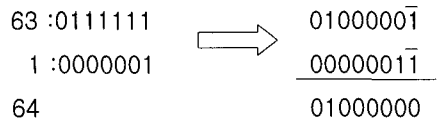


그림 1. Redundant signed binary 덧셈 예
Fig. 1. Example of redundant signed binary addition.

carry propagation이 발생하지 않는다. <표 1>에 명시된 X는 정의된 a_{i-1}, b_{i-1} 패턴 외의 다른 모든 패턴을 나타낸다. <그림 1>은 <표 1>의 규칙에 따른 덧셈 예이다.

Booth recoding을 사용한 adder는 global carry가 발생하지 않으므로, 모든 digit에 대하여 병렬처리가 가능하다. 따라서 입력 digit의 수와 관계없이 addition에 걸리는 전체 지연은 single adder cell의 gate 지연과 같아진다. 그러나 booth recoding을 사용한 adder는 덧셈을 수행한 후 결과가 booth code의 체계를 따르지 않는다. 즉 연속된 1 또는 $\bar{1}$이 발생하게 된다. 그리고 연속된 1 또는 $\bar{1}$이 생성된 곳은 global carry가 발생하게 된다. 따라서 반복되는 덧셈연산을 하게 될 경우 이 알고리즘을 적용할 수 없다.

RSB recoding은 <표 2>의 규칙에 따라 연속된 1 또는 $\bar{1}$을 제거하여 반복되는 덧셈연산을 가능하게 한다. 즉 RSB recoding은 Booth recoding을 사용하여 덧셈을

표 2. Redundant signed binary number recoding
Table 2. Redundant signed binary number recoding.

$y_i = 1$				$y_i = \bar{1}$				$y_i = 0$
x_i	x_{i-1}	x_{i-2}	x_{i-3}	x_i	x_{i-1}	x_{i-2}	x_{i-3}	
$\bar{1}$	$\bar{1}$	0	1	1	1	0	$\bar{1}$	
$\bar{1}$	$\bar{1}$	1	d	1	1	$\bar{1}$	d	
$\bar{1}$	0	$\bar{1}$	d	1	0	1	d	
$\bar{1}$	0	0	d	1	0	0	d	for all other combination of $x_i x_{i-1} x_{i-2} x_{i-3}$
0	1	0	0	0	$\bar{1}$	0	0	
0	1	0	1	0	$\bar{1}$	0	$\bar{1}$	
0	1	1	d	0	$\bar{1}$	$\bar{1}$	d	
1	$\bar{1}$	0	1	$\bar{1}$	1	0	$\bar{1}$	
1	$\bar{1}$	1	d	$\bar{1}$	1	$\bar{1}$	d	
1	0	$\bar{1}$	d	$\bar{1}$	0	1	d	

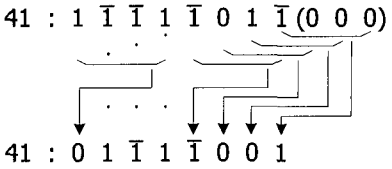


그림 2. RSB recoding의 예
Fig. 2. RSB recoding example.

수행함으로써 발생한 연속된 1 또는 $\bar{1}$을 제거한다. 따라서 결과적으로 얻어지는 bit pattern에서 연속된 1과 $\bar{1}$이 없으므로 global carry가 없는 덧셈연산을 할 수 있다.

<표 2>의 규칙에 따라 redundant signed binary number $\cdots x_{i+1}x_i x_{i-1}\cdots$은 $\cdots y_{i+1}y_i y_{i-1}\cdots$으로 recoding된다. 이때 $x_i(x_{i-1}x_{i-2}x_{i-3})$는 y_i로 recoding이 된다. x_i 이하의 비트열 $(x_{i-1}x_{i-2}x_{i-3})$은 reference bit가 된다. <표 2>의 d는 don't care를 나타낸다. <그림 2>는 <표 2>의 규칙에 따라 연속된 1과 $\bar{1}$이 제거되는 예이다.

Ⅲ. RSBA CORDIC의 동작.

Pipelined RSBA CORDIC은 전체 8단계의 pipeline으로 구성하였다. <그림 3>은 pipelined RSBA CORDIC을 나타낸 전체블록이다. 이 블록은 다시 부호비트 결정블록과 계산블록으로 나뉘어져 있다. 부호 비트 결정블록은 회전에 의하여 결정되는 가감산을 선택하는 것이다. 계산블록은 부호비트를 받아들여서 회전과정에서의 중간 값을 계산한다. 여기에서 사용되는 가감산기는 연산속도에 가장 큰 영향을 미친다. 속도의 개선을 위하여 병렬연산이 가능한 RSB adder가 사용되었다.

Pipelined RSBA CORDIC은 별도의 제어회로를 사용하지 않는다. 별도의 제어 회로 없이 각 단계의 부호비트 결정블록과 계산 블록을 수행한다. 그리고 각 블록의 pipeline 한 단계는 한번의 clock으로 동작한다. 초기 단계에서 각 블록의 RSBA 부호비트는 음수로 결정된다. 그리고 각 단계에서 결정되는 부호비트는 다음 단계의 부호비트 결정블록의 RSBA와 계산 블록의 RSBA의 부호 입력으로 사용된다.

Pipelined RSBA CORDIC의 부호 결정 블록은 <그림 4>와 같다. 부호 결정 블록은 pipeline 각 단계마다

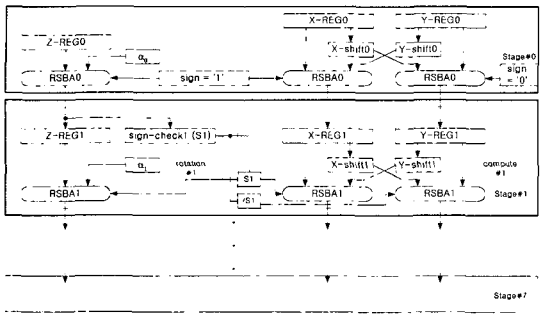


그림 3. Pipelined RSBA CORDIC 구조
Fig. 3. Pipelined RSBA CORDIC architecture.

하나의 register와 하나의 RSB adder로 구성되어 있다. 부호 결정 블록은 구하려는 각도 Z, 회전각도(α) 그리고 부호비트로 되어있다. 이때 회전각도는 이미 결정된 각도로서 pipeline의 각 단계에 따라 다른 회전 각도를 갖는다. 이 회전각도의 값은 이미 결정된 pipeline의 단수에 따라 결정된다. 그리고 0단계에서 RSBA의 부호비트는 음수로 결정된다. 부호비트는 RSBA의 가산 또는 감산을 하는데 사용한다. RSBA는 결정된 두 입력과 부호비트를 통하여 계산된다. 이 때 계산된 RSBA의 출력 값은 다음 단계의 부호비트를 결정하는데 사용된다.

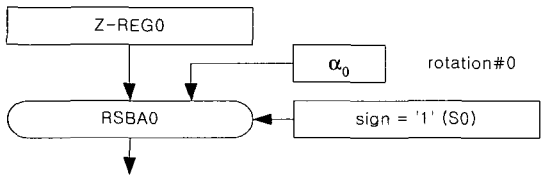


그림 4. 부호비트 결정블록
Fig. 4. Sign bit decision block.

각 단계의 부호 비트들은 다음과 같이 결정된다. 전 단계의 RSBA 출력에서 결정된 부호비트가 다음 pipeline 단계의 RSBA 부호비트의 입력으로 사용된다. 그리고 전 단계의 RSBA의 출력 값을 다음 단계의 입력으로 사용한다. 또 다른 입력으로는 이미 결정된 회전각도의 값을 입력으로 갖는다. 이 입력을 받은 RSBA의 출력으로부터 다음단의 부호비트를 결정한다.

RSBA의 부호비트를 빠르게 결정하는 방법은 매우 중요한 일이다. RSBA에서 부호비트의 결정은 속도에 직접적으로 영향을 미친다. 그래서 부호비트를 빠르게 결정하기 위한 방법으로 맨체스터 캐리체인을 사용한다[6]. 이 방법을 사용함으로써 RSBA의 출력 값에 대한 부호를 빠르게 결정할 수 있다. 이러한 방법으로 결정된 부호비트는 다음 단계의 부호비트 입력으로 사용된다. 결정된 각 단계의 부호비트는 계산블록의 RSBA 부호비트 입력이 된다.

Pipelined RSBA CORDIC의 계산 블록은 <그림 5>와 같다. 계산 블록은 각 pipeline단계마다 2개의 register, 2개의 RSB adder와 2개의 shifter로 구성되어 있다. 이때 shifter는 hardwired 방식으로 구성되어진다. 그리고 크게 계산 블록은 X값을 계산하는 부분과 Y값을 계산하는 부분으로 나누어진다. 이때 X값은 cosine 값이 되고 Y값은 sine 값이 된다.

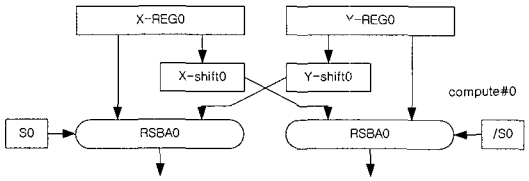


그림 5. 계산블록
Fig. 5. Computation block.

계산 블록의 동작은 다음과 같다. X, Y값을 계산하기 위해 RSBA와 shifter가 사용된다. X값 계산을 위한 동작으로 RSBA의 입력은 Y값의 shift한 입력과 X값을 입력으로 사용한다. 마찬 가지로 Y값 계산을 위한 동작으로 RSBA의 입력은 X값의 shift한 입력과 Y값을 입력으로 사용한다.

계산블록의 RSBA 부호 비트입력으로는 부호 결정 블록에서 결정된 부호비트를 입력으로 받는다. 이때 X값을 계산하는 블록과 Y값을 계산하는 블록의 RSBA 부호비트 입력이 반대이다. X값을 계산하는 블록의 RSBA의 부호비트와 부호 결정 블록의 RSBA의 부호비트가 동일하다. 그리고 Y값을 계산하는 블록의 RSBA의 부호비트와 부호결정 블록의 RSBA의 부호비트는 반대이다. X값을 계산하는 블록과 Y값을 계산하는 블록의 RSBA의 동작은 항상 반대이다.

X값과 Y값을 계산하기 위한 shift 동작은 다음과 같다. Pipelined RSBA CORDIC에서의 shift는 right shift를 의미한다. 이때 shift는 pipeline의 각 단계마다 위치하고 있다. 그리고 각 단계의 shift 수는 각각 다르다. Shift의 출력은 RSBA의 입력이 된다. Pipeline의 0단계의 shift는 입력 값과 출력 값이 같다. 입력과 출력이

같다는 것은 right shift가 없음을 의미한다. 1단계에서는 한번의 right shift를 한다. 2단계에서는 두 번의 right shift를 하게 된다. 이와 같이 각 단계마다 단 수에 따른 hardwired shift 동작을 하게 된다. 이렇게 구성된 shifter는 RSBA와 함께 한 clock으로 동작한다.

표 3. C로 표현한 CORDIC pseudo code
Table 3. Pseudo code of CORDIC expressed in C.

```
int sign_detection(i)
{
      if(i>=0) return 1; //rotation direction
      if(i<0) return -1;
}
double computation(digit,stage,theta)
{
      for(i=0; i<stage ;i++)
      begin
       alpha[i]=alpha[i]*sign_detection(theta);
       theta=theta-alpha[i];
       x(i)=x-sign_detection(alpha[i])*y(i-1)*pow( 2, -1*i );
       y(i)=y+sign_detection(alpha[i])*x(i-1)*pow( 2, -1*i );
      end
}
```

<표 3>은 CORDIC을 pseudo code로 나타낸 것이다. 이 코드의 sign_detection은 부호를 결정하는 블록으로서 맨체스터 캐리체인방법을 이용한다. 이 코드는 부호결정과 계산의 흐름에 대하여 나타낸다.

Ⅳ. 구현 및 성능분석

앞에서 제안한 pipelined RSBA CORDIC의 동작을 확인하기 위하여 8-bit의 CORDIC을 설계하였다. 제안된 회로는 VHDL coding을 이용하여 설계하였다. 그리고 이 회로는 Alter사의 Quartus II 2.0을 이용하여 시뮬레이션 하였다. Device는 APEX20KE EP20K200EQC 240-1을 사용하였으며 전체 logic elements의 8320개중 2147개를 사용하였다. 그리고 8단의 pilpeline 으로 구성된 CORDIC의 최대 동작주파수는 88.32MHz로써 8 클럭 후 11.322ns의 속도로 결과 값이 출력된다. <그림 7>은 이 회로의 시뮬레이션 값을 보여준다. 이 값은 계산하고자 하는 각도를 입력 후 클럭 신호의 진행에 따라 각 단계별 계산을 하게 되어 8 클럭 후에 의도했던 sine 또는 cosine 값이 출력된다. 이 과정에서 RSBA는 global carry의 전파를 방지하여 빠른 계산이 가능하도록 한다. Pipelined RSBA CORDIC은 pipeline 각 단계마다 한번의 clock을 필요로 한다. 이때 부호결정 블록의 각 단계와 계산블록의 각 단계가 모두 한번의 clock으로 동작을 한다.

초기 값 X, Y의 기본벡터는 (1, 0)을 사용하였으며 입력각도(θ)는 radian으로 계산하였다. Pipeline 한단의 동작은 1clock을 사용하며 결과는 8clock이후 출력된다. 입력 값 및 출력 값은 digit으로 이루어져 있으며 1-digit은 2-bit의 상위비트와 하위비트로 나뉘어 진다. X, Y의 초기값은 pseudo rotation에 의해 발생한 K factor를 보상하여 (0.6072 , 0)를 입력으로 사용하였다. 출력 X는 cosine값 이며 Y는 sine값 이다.

다음 예로서 <그림 7>의 시뮬레이션 입력각도가 $\pi/6$일 경우이다. X, Y값의 출력은 10-bit로 이루어져 있다. 모든 파이프라인 단계를 거쳐서 얻은 각도 $\pi/6$의 값은 <그림 6>과 같이 계산된다. cosine 값은 X값의 출력으로 상위비트열 "0000101001"과 하위비트열 "0100000100"이다. 이것을 이진수로 나타내면 0.110110111이 되며 십진수로 0.8554가 된다. 또한 sine 값은 Y값의 출력으로 상위비트열 "0000000010"과 하위비트열 "0010000100"로서 이진수로 나타내면 0.10000010이 되며 십진수로 0.5078이 된다. 이 값은 근사 값으로서 약간의 오차가 포함되어 있다. 데이터의 비트 수와 파이프라인의 단수를 증가시키면 이 오차는 감소한다.

RSBA는 RSBA cell(<그림 8>)과 RSB recoding cell (<그림 9>)로 구성되어 있다. RSBA cell은 두 개의 입

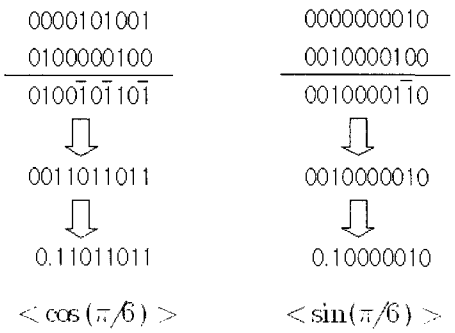


그림 6. 2진수 recoding 에 의한 삼각함수 계산 예
Fig 6. Example of trigonometric calculation based on binary recoding.

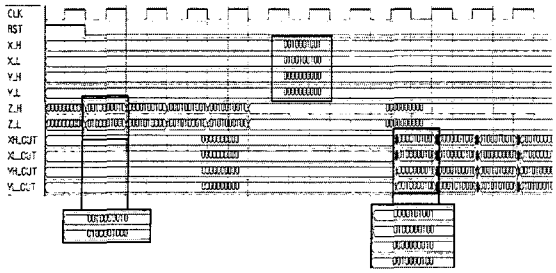


그림 7. Pipelined RSBA CORDIC 시뮬레이션 결과.
Fig. 7. Pipelined RSBA CORDIC simulation result.

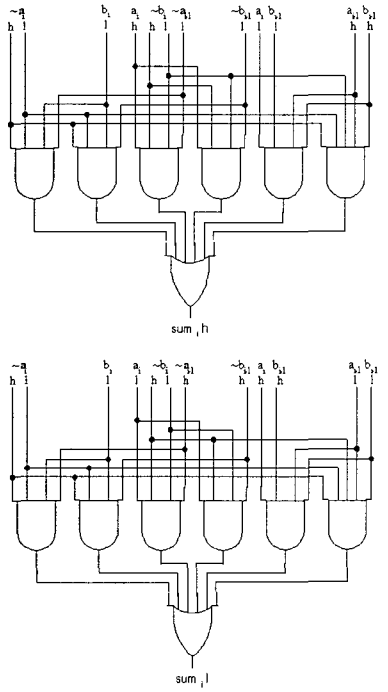


그림 8. RSBA cell
Fig. 8. RSBA cell.

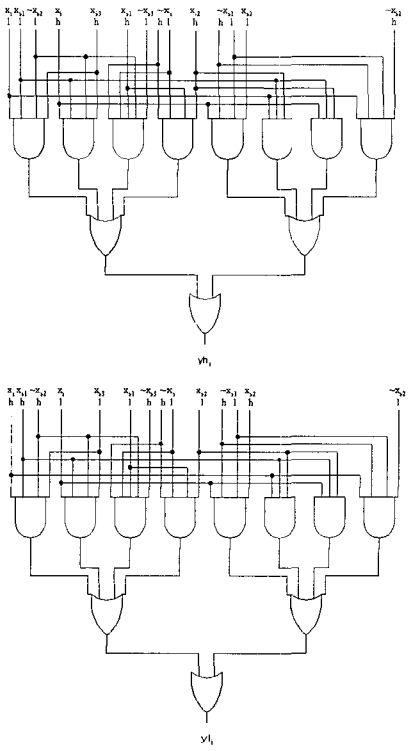


그림 9. RSB recoding cell
Fig. 9. RSB recoding cell.

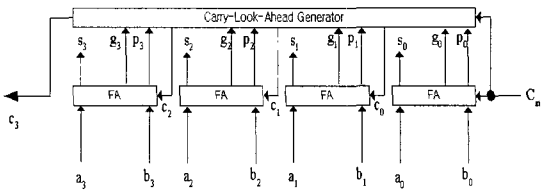


그림 10. 4-bit CLA
Fig. 10. 4-bit CLA.

력 a_ia_{i-1}, b_ib_{i-1}와 하나의 출력 s_i를 갖는다. RSB recoding cell은 RSB adder의 sum vector를 입력으로 사용한다. 입력 $x_ix_{i-1}x_{i-2}x_{i-3}$과 출력 y_i를 갖는다. 한 개의 gate 지연시간을 ΔT라고 하자. 이때 하나의 RSBA cell을 통과하는데 걸리는 시간은 3ΔT가 걸린다. 그리고 하나의 RSB recoding cell을 통과하는데 걸리는 시간은 3ΔT이다. RSBA cell과 RSB recoding cell을 통과하는데 걸리는 시간은 총 6ΔT이다. RSBA는 병렬연산을 하므로 입력 bit의 수에 관계없이 6ΔT의 지연시간을 갖는다.

Carry lookahead adder의 경우 4-bit CLA(<그림 10>)는 carry propagation(p)과 generation(g)의 계산에 1ΔT가 걸린다. 그리고 carry를 계산하는데 2ΔT가 걸리며 sum을 계산하는데 2ΔT가 걸린다. 지연시간은 carry propagation과 generation, carry 계산 그리고 sum 계산에 걸리는 총 합으로 5ΔT가 된다. CLA의 N-bit에 따른 지연시간은 4-bit lookahead 블록일 때 $(4\lceil \log_4 N \rceil + 1)\Delta T$가 된다[7].

RSBA 와 CLA 입력이 8-bit일 때 지연시간을 비교해 보자. RSBA는 입력비트 수에 관계없이 지연시간이 동일한 6ΔT가 된다. CLA는 지연시간이 입력비트 N에 따라 달라진다. RSBA와 동일한 입력비트인 8-bit에서 지연시간은 $(4\lceil \log_4 8 \rceil + 1)\Delta T$로서 9ΔT가 된다.

Pipelined CORDIC의 지연시간은 다음과 같이 나타낼

수 있다. Pipeline의 단수를 PN이라하고, pipeline 한단의 지연시간을 T라 하면 전체지연시간= $PN \times T$가 된다. 전체 8단계의 pipeline을 갖는 세 가지 방법의 8-bit CORDIC의 지연시간을 비교하였다. 첫째, 제안된 RSBA CORDIC의 경우이다. 계산블록은 RSBA, shifter 그리고 register로 이루어져 있다. RSBA는 입력비트 수에 관계없이 6ΔT가 된다. 계산블록의 pipeline 한단의 지연시간은 6ΔT가 된다. 부호결정블록은 RSBA, register 그리고 맨체스터 캐리체인으로 이루어져 있다. 맨체스터 캐리체인의 지연시간을 1ΔT하면 부호결정블록의 pipeline 한단의 지연시간은 7ΔT가 된다. Pipelined RSBA CORDIC의 총 지연시간은 pipeline 단수와 각단의 지연시간의 곱으로 나타낼 수 있다. 이때 부호결정블록과 계산블록은 동시에 수행 되므로 pipeline 한단의 지연시간은 7ΔT라 할 수 있다. 전체 지연시간은 8단의 pipeline과 한단의 pipeline 지연시간 7ΔT의 곱으로 56ΔT가 된다.

둘째, RSBA와 double rotation을 이용한 CORDIC의 경우이다. Double rotation방법은 scale factor보상을 위한 추가의 addition이 필요하다[2]. 그러므로 pipeline 한단의 지연시간에 2배의 addition시간이 필요하다. 그러나 보상회로는 전체 pipeline 단수의 1/2에 한하여 적용된다. 전체 8단의 pipeline중 4단만이 2번의 addition을 한다. 그러므로 전체 pipeline 단에 걸리는 addition 시간은 1.5배가된다. Pipeline 한단의 지연시간은 부호판별에 걸리는 지연시간과 계산과정에 걸리는 지연시간의 합이다. 부호판별은 상위 3digit을 이용하여 판별하며 2ΔT의 시간이 걸린다. 계산과정은 보상을 하는 pipeline 단과 그렇지 않은 pipeline 단으로 나누어진다. 그러므로 8단의 pipeline에 걸리는 전체지연시간은 $(2+1.5\cdot 6)\cdot 8 = 88\Delta T$가 된다.

셋째, CLA를 이용한 CORDIC의 경우이다. 계산블록은 CLA, shifter 그리고 register로 이루어져 있다. CLA는 8-bit의 입력을 가질 때 9ΔT를 갖는다. 그러므로 계산블록의 pipeline 한단의 지연시간은 9ΔT가 된다. 부호결정블록은 CLA, register로 이루어져 있다. 부호결정블록의 pipeline 한단의 지연시간은 9ΔT가 된다. 그러나 부호결정 블록과 계산블록의 pipeline의 각단은 동시에 수행되므로 pipeline 한단의 지연시간은 9ΔT가 된다. Pipelined CLA CORDIC의 총 지연시간은 8단의 pipeline과 9ΔT의 지연시간의 곱으로 72ΔT가 된다.

위의 세 가지 CORDIC에 대하여 입력비트 N에 대하여 전체 지연시간을 나타내면 다음과 같다. RSBA의 지연시간은 6ΔT라 하고 pipeline 단수는 입력비트 N과 같다고 한다. 그리고 N-bit CLA의 지연시간은 $(4\lceil \log_4 N \rceil + 1)\Delta T$라 한다. 그리고 제안된 RSBA CORDIC과 double rotation CORDIC의 결과값이 redundant number에서 binary number로의 변환이 필요하다면 최종 결과값에 대하여 한번의 binary addition을 취함으로써 계산할 수 있다. 따라서 제안된 RSBA CORDIC의 지연시간은 $(7N+(4\lceil \log_4 N \rceil + 1))\Delta T$가 된다. 그리고 double rotation CORDIC의 지연시간은 $(11N+(4\lceil \log_4 N \rceil + 1))\Delta T$ 이고 CLA CORDIC의 지연시간은 $((4\lceil \log_4 N \rceil + 1)\cdot N)\Delta T$가 된다.

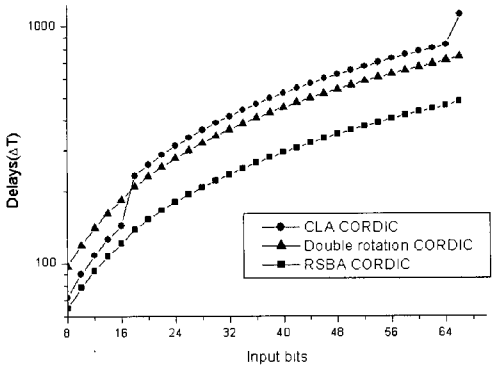


그림 11. CORDIC의 지연시간 비교
Fig. 11. Comparison of delays for CORDIC circuits.

Pipeline 한단의 지연시간은 주로 adder의 지연시간으로 평가하였다. 이 값들은 실제 구현 후에는 물리적 배치배선 등에 의하여 달라질 수 있다. 여기서는 상대적인 비교를 위하여 부수적인 지연요인들은 무시하였다. <그림 11>은 입력비트 수의 증가에 따른 RSBA CORDIC, double rotation CORDIC 그리고 CLA CORDIC의 지연시간을 비교하였다. 입력비트 수가 증가할수록 RSBA를 이용한 CORDIC과 CLA CORDIC의 지연시간의 차이는 점점 커질 것이다.

V. 결 론

삼각함수계산을 위한 pipelined RSBA CORDIC을 구현하였다. 제안된 pipelined RSBA CORDIC은 상수 scale factor를 갖고 병렬연산이 가능한 RSB adder를 사용함으로써 속도를 개선하였다. Redundnat number format의 도입과정에서 Booth recording 기법을 개선하

여 가감산 직후에 그 결과 값이 연속된 1 또는 $\bar{1}$이 올 수 없도록 하였다. 이로서 global carry 없이 신속한 연산이 가능하도록 하였다. 제안된 pipelined RSBA CORDIC은 operand의 bit수에 관계없이 지연시간이 일정하다. 기존의 carry lookahead adder를 채택한 고속 회로들에 비하여 상당한 속도 개선 효과를 가져 왔다.

참 고 문 헌

[1] J. E. Volder, "The CORDIC trigonometric computing technique," IRE Trans. Electron. Computers, vol. EC-8, pp. 330-334, Sept. 1959.

[2] N. Takagi, T. Asada and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation," IEEE Trans. Computers, vol. 40, pp. 989-995, Sept. 1991.

[3] M. D. Ercegovac and T. Lang, "Redundant and on-line CORDIC: application to matrix triangularization and SVD," IEEE Trans. Computers, vol. 39, pp. 725-740, June 1990.

[4] A. D. Booth, "A signed binary multiplication technique," Quart. J. Mech. Appl. Math., vol. 4, pt. 2, pp. 236-240, June 1951.

[5] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," IRE Trans. Electronic Computers, vol. 10, pp. 389-400, Sept. 1961.

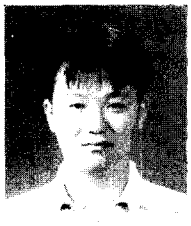
[6] T. Kilburn, D. B. G. Edwards and D. Aspinal, "A parallel arithmetic unit using a saturated transistor fast-carry circuit," Proc. IEE, vol. 107 Pt. B, pp. 573-584, Nov. 1960,

[7] B. Parhami, Computer Arithmetic Algorithms and Hardware Designs, Oxford University Press, 2000.

[8] 남현숙, 유영갑, "파이프라인형 CORDIC를 이용한 직접 디지털 주파수 합성기 설계," 대한전자공학회논문지, 36권, D편, 5호, 36-43쪽, 1999년 5월

[9] 안영호, 남승현, 성원용, "CORDIC 구조를 이용한 디지털 위상 오차 보상기의 VLSI 구현," 대한전자공학회논문지, 36권, C편, 3호, 35-46쪽, 1999년 3월

저 자 소 개

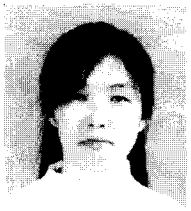


金 承 烈(正會員)

2002년 2월 : 충북대학교 정보통신공학과 공학사. 2002년~현재 : 충북대학교 정보통신공학과 석사과정. <주관심분야 : 고속 인쇄회로 설계, 연산회로 설계>

金 容 大(正會員)

1990년 2월 : 충북대학교 정보통신공학과 공학사. 1993년 2월 : 충북대학교 컴퓨터공학과 공학석사. 1989년 10월~1998년 6월 : 신흥기술연구소 팀장. 2000년 3월~현재 : 충북대학교 정보통신공학과 박사과정. <주관심분야 : Computer arithmetic, Cryptographic system, ASIC 설계>



韓 善 景(正會員)

1991년 2월 : 충북대학교 정보통신공학과 공학사. 1993년 2월 : 충북대학교 정보통신공학과 공학석사. 1995년 3월~현재 : 충북대학교 정보통신공학과 박사과정. <주관심분야 : Computer arithmetic, Cryptographic system, ASIC 설계>

劉 泳 甲(正會員)

1975년 8월 : 서강대학교 전자공학과 공학사. 1975년~1979년 : 국방과학연구소 연구원. 1981년 8월 : Univ.of Michigan, Ann Arbor 전기전산학과 공학석사. 1986년 4월 : Univ.of Michigan, Ann Arbor 전기전산학과 공학박사. 1986년~1988년 : 금성반도체 (주) 책임 연구원. 1993년~1994년 : 아리조나 대학교 객원교수. 1998년~2000년 : 오레곤 주립대학교 교환교수. 1988년~현재 : 충북대학교 정보통신공학과 교수. <주관심분야 : VLSI 설계 및 Test, 고속 인쇄회로 설계, Cryptography>