論文97-34C-10-4

부분곱의 재정렬과 4:2 변환 기법을 이용한 VLSI 고속 병렬 곱셈기의 새로운 구현 방법

(A New Scheme for VLSI Implementation of Fast Parallel Multiplier using 2x2 Submultipliers and Ture 4:2 Compressors with no Carry Propagation)

李 相 球 *, 全 英 淑 *

(Sang-Gu Lee and Young-Suk Jun)

요 약

본 논문에서는 VLSI 고속 병렬곱셈기의 부분곱 생성을 위한 새로운 기법을 제시한다. 제시된 기법은 2x2 부분곱셈기와 이들로부터의 원시 부분곱들을 재정렬함으로써 전체 부분곱의 개수를 반으로 줄이는 새로운 부분곱 생성 방법을 채택하고 있다. 새로이 제안된 4진수 정리에 따라서 원시 부분곱들을 적절하게 재정렬함으로써 캐리 전파가 없는 4:2 변환이 가능하다. 제안된 기법을 이용하여 16bit x 16bit 곱셈기를 설계하였으며 SPICE 시뮬레이션을 통하여 제시된 부분곱 생성 방법이 면적 및 속도 면에서 Booth 인코딩 방법을 대체할 수 있음을 보였다. 최적화를 통하여 더 작은 면적과 빠른 속도를 갖는 곱셈기를 실현할 수 있으며, 제시된 기법은 32bit 이상의 곱셈기에도 쉽게 적용이 가능하다.

Abstract

In this paper, we propose a new scheme for the generation of partial products for VLSI fast parallel multiplier. It adopts a new encoding method which halves the number of partial products using 2x2 submultipliers and rearrangement of primitive partial products. The true 4-input CSA can be achieved with appropriate rearrangement of primitive partial products out of 2x2 submultipliers using the newly proposed theorem on binary number system. A 16bit x 16bit multiplier has been designed using the proposed method and simulated to prove that the method has comparable speed and area compared to Booth's encoding method. Much smaller and faster multiplier could be obtained with far optimization. The proposed scheme can be easily extended to multipliers with inputs of higher resolutions.

I. 서 론

곱셈은 덧셈과 함께 디지털 신호처리에 있어서 가장 빈번하게 사용되는 기본적 연산이다. 즉, correlation, convolution, filtering, DFT(discrete Fourier transform) 등 DSP에서 빈번하게 사용되는 연산들은 대부분 반복적인 수렴방식을 통하여 구현되어지므로 매우 많은 횟수의 곱셈이 필요하다. 한편, 최근 멀티미디어에 대한 관심이 높아지면서 MPEG등과 같은 대규모 데이터들의 실시간 처리를 위해서는 DCT등의 비선형 연산이 필요하게 되는데, 이러한 비선형 함수의 계산은 곱셈과 덧셈을 반복적으로 수행함으로써 이루어지기 때문에 덧셈이나 곱셈의 단위 연산 시간이 전체의

* 正會員, 韓南大學校 컴퓨터工學科
(Dept. of Computer Engineering, Hannam University)
接受日字:1997年2月12日, 수정완료일:1997年9月29日

연산 수행 속도를 좌우하게 된다. 곱셈 연산은 기본적으로 덧셈 연산을 포함하고 있기 때문에 덧셈보다 수행속도가 더 느리므로 고속의 곱셈기를 얻는 것이 전체 시스템의 정보처리 성능을 높이는 데에 있어서 매우 중요하다고 할 수 있다.

곱셈기(multiplier)는 크게 두 부분의 functional block으로 나눌 수 있는데, 첫 번째는 입력으로 들어오는 피승수(multiplicand)와 승수(multiplier)로부터 partial product(부분곱)들을 만들어내는 인코더(encoder) 부분과 그것들을 모두 더하여 최종 결과를 만들어내는 adder array 부분이다. 기본적으로는 N-bit word끼리의 곱셈에서는 N에 비례하는 개수의 부분곱들이 만들어지고 이들을 순차적으로 더하기 위하여 역시 N에 비례하는 덧셈 수행시간이 걸린다. 현재까지 고안된 곱셈기 중 가장 빠른 것으로 알려진 곱셈기의 형태는 부분곱들을 생성하는 인코더에는 Booth 인코더[1-3]를, 부분곱들을 더하는 adder array에는 Wallace tree[4]를 이용하는 방식이다. Wallace tree는 인코더로부터 나오는 부분곱들의 덧셈을 최대한 병렬로 수행하여 N-bit word의 곱셈을 위하여 $O(\log_2 N)$에 비례하는 수행시간을 가진다. 즉, 캐리 저장 가산기 (CSA 혹은 1's counter)를 사용하여 N개의 입력을 그들 중 1의 개수를 카운트하여 출력으로 내보내면 출력 bit의 개수는 $\log_2 N$ 개로 줄어드는 원리를 이용하는데, 실제로는 많은 bit수를 입력으로 하는 CSA는 구현하는데 많은 면적이 필요하므로 3:2 혹은 7:3 counter를 다수 사용하여 pipeline stage마다 이러한 counter의 개수가 줄어들도록 하고 있다.

Booth 인코딩 (혹은 modified Booth algorithm) 방법을 사용하면 이러한 Wallace tree의 입력이 되는 부분곱들의 개수를 거의 반으로 줄일 수가 있다. 이러한 경우에는 CSA stage가 한 단계 줄어들기 때문에 속도 및 CSA와 interconnection에 소요되는 면적을 많이 줄일 수 있다. 또한 이와 동시에 2의 보수로 표현된 word들의 곱셈도 자동적으로 가능해진다는 장점이 있다. 이러한 이유로 인하여 현재의 거의 모든 곱셈기들에서는 Booth 인코딩 방식을 사용하고 있으며 이보다 더 좋은 성능을 가지는 인코딩 방식을 찾아내기는 쉽지 않을 것으로 보인다.

본 논문에서는 Booth 인코딩을 포함한 기존의 곱셈기들에서 고려하지 않았던 점, 즉 (3진수 이상의) 숫자들의 곱셈 결과는 제한된 가짓수의 숫자만을 가질 수 있다는 사실 등 곱셈 연산 속에 내재되어 있는 규칙들을 이용하여 새로운 원리의 인코딩 방법을 시도해 보고자 한다. 본 논문에서는 4진수 곱셈에서의 내재된 곱셈을 이용하여 앞에서 언급한 Booth 인코딩 방식의 두 가지 장점을 모두 갖는 새로운 인코딩 (혹은 부분곱 생성) 방식을 제안한다.

Ⅱ. 제안된 곱셈기의 원리

1. 4진수 곱셈에서의 정리

모든 곱셈기의 원리는 기본적으로 우리가 일상적으로 사용하는 십진수 곱셈의 hand-calculation과 같은 원리를 등가적으로 이진수에 대해서 적용한 것이다. 즉 곱셈기가 곱셈을 수행하는 중간단계에서 나타나는 각각의 부분곱들은 피승수가 1bit씩 (Booth 인코딩의 경우는 2bit씩) 왼쪽으로 shift된 것이다. 이러한 경우에 각 자릿수마다의 부분곱들의 bit들은 피승수 자체 내에서의 각 bit들 만큼이나 서로 연관성이 없다. 이러한 경우에는 부분곱들의 각 bit들 중 자릿수가 같은 bit들로 이루어진 string에는 모든 경우의 수가 나타날 수 있다. 가장 나쁜 경우로서 string내의 모든 bit가 1이 될 수도 있다. Booth 인코딩의 경우도 이와 같음을 쉽게 증명할 수 있다.

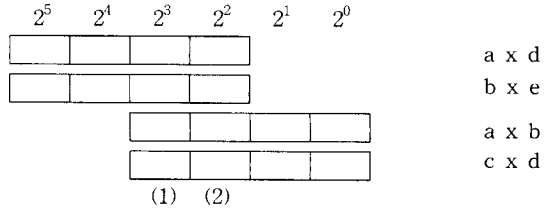
만일 피승수와 승수를 각각 4진수로 표현되어 있다고 가정하면, 각 부분곱들은 4진수(0~3) x 4진수(0~3) 의 조합으로 나타나고 이러한 값들은 {0, 1, 2, 3, 4, 6, 9}의 값만을 가지기 때문에 부분곱의 각 bit 들간에 성립하는 어떠한 규칙을 찾을 수가 있을 것이다. 이에 관련하여 본 연구에서 발견한 정리는 다음과 같다.

[정리]

a, b, c, d, e 를 각각 4진수의 수라고 할 때, 4개의 수 $(a \times b)_2$, $(c \times d)_2$, $(4 \times a \times d)_2$, $(4 \times b \times e)_2$ 를 더하는 경우 모든 자릿수에서의 최대값은 3이다. 즉, 어떠한 비트위치에서도 all 1이 발생되지 않는다.

[증명]

앞의 4개의 항을 자릿수의 위치에 맞게 나타내면



과 같고, 각각의 비트 위치에서 1의 개수가 최대가 되는 위치는 (1)과 (2)이다. 그런데 2개의 4진수를 곱한 결과는 {0, 1, 2, 3, 4, 6, 9}중의 하나이며, 따라서 (1)에서 자릿수의 최대합이 4가 나오기 위해서는 각각의 곱들이 상단의 두 곱은 2, 3 또는 6, 하단의 두 곱은 모두 9가 되어야 한다. 그러나 밑의 두 곱들이 모두 9이기 위해서는 a, b, c, d가 모두 3이어야 하므로 a(d가 9가 되어 최대 합이 4가 될 수 없다. 또한 (2)에서 최대합이 4가 나오기 위해서는 상단의 두 곱이 홀수, 하단의 두 곱은 4나 6이 되어야 하므로 a, b중 하나는 2가 되어 위의 두 곱이 동시에 홀수가 될 수가 없으므로 최대합이 역시 4가 될 수 없다. ■

위의 정리를 이용하면, 4bit을 더할 경우에도 1의 개수가 최대 세 개밖에 나오지 않으므로 출력이 2bit인 4→2 compressor를 쉽게 만들 수가 있을 것이다. 문제는 임의의 N-bit word끼리의 곱으로부터는 $(N/2)^2$ 개의 부분곱들이 나오는데 이것들을 모두 위의 정리에 맞도록 적절하게 배치할 수가 있는가 하는 것이다. 다음절에서 설명하는 바와 같이 8bit과 16bit에 대해서는 이것이 가능함이 증명되어졌고, 임의의 길이의 word에 대해서도 가능함을 쉽게 보일 수 있다.

2. 2x2 부분곱셈기를 이용한 곱셈기의 구현

피승수를 M, 승수를 P라 할 때, 각각을 4진수로 표현하면

$$M = \sum_{i=0}^{\frac{N}{2}-1} m_i 2^{2i}\ ,\ P = \sum_{i=0}^{\frac{N}{2}-1} p_i 2^{2i} \cdot (m_i, p_i = 0,1,2,3)$$

와 같이 되고, 이들의 곱은

$$M*P = \sum_{i=0}^{\frac{N}{2}-1}\sum_{j=0}^{\frac{N}{2}-1} m_i p_j 2^{2(i+j)}$$

$$= \sum_{i=0}^{\frac{N}{2}-1}\left(\sum_{j=0}^{i} m_j p_{i-j}\right)2^{2i} + \sum_{i=\frac{N}{2}}^{N-1}\left(\sum_{j=i-\frac{N}{2}+1}^{\frac{N}{2}-1} m_j p_{i-j}\right)2^{2i}$$

와 같이 된다. 이 때 각 $m_j x p_{i-j}$들은 2x2 부분곱셈기로부터 얻을 수가 있다. 이러한 2x2 부분곱셈기로부터 나온 부분곱들을 원시 부분곱(primitive partial product) 이라고 부르기로 한다.

N bit의 곱셈기에서는 $(N/2)^2$ 개의 원시 부분곱들이 나오는데 서로 overlap되지 않는 원시 부분곱들을 연결하면 총 N-1 개의 부분곱 word들이 나오며 이들 부분곱들을 1개의 3-word group (3개의 부분곱 word)과 N/4-1 개의 4-word group(4개의 부분곱 word)으로 묶을 수 있다. 이 때 4-word group내의 원시 부분곱들을 다른 4-word group 그룹 혹은 3-word group 내의 원시 부분곱들과 적절히 교환하면 4-word group내의 인접하는 모든 원시 부분곱들의 쌍을 위의 정리를 만족하도록 재배열할 수가 있다. 이렇게 재배열한 뒤에는 3-word group의 경우에는 CSA(Carry Save Adder), 4-word group의 경우에는 4-input CSA를 구성하여 총 N/2개의 부분곱으로 줄일 수 있게 된다.

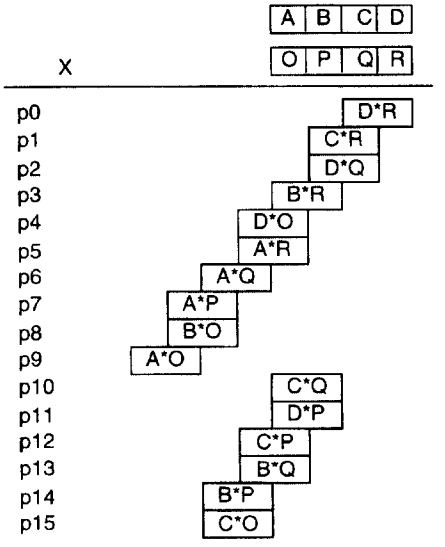


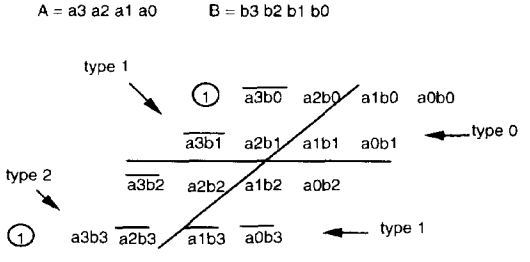
그림 1. 8bit 곱셈기 (A~R은 2비트 2진수를 나타냄)
Fig. 1. 8bit multiplier (A~R are 2 bit binary numbers).

예를 들면, 8bit의 경우에는 그림 1과 같이 된다. 즉, 앞에서 설명한 바와 같이 16개의 원시 부분곱 (p0~p15) 들 중 p0에서 p9까지의 원시 부분곱들은 각각 overlap되지 않는 것들끼리 묶어서 "p9, p6, p3, p0", "p7, p4, p1", "p8, p5, p2"의 세 개의 word라 할 수 있고 이들이 앞에서 설명한 3-word group이 되며, p10에서 p15까지의 부분곱들은 "p14, p10", "p15, p11", "p12, p13" 으로 한 개의 4-word group이 된다. 3-word group의 경우에는 CSA를 써서 2개의 word로 줄일 수가 있으며, 4-word group의 경우는 그림 1과 같이 p10~p13, p12~p15의 쌍들이 각각 앞의 정리를 만족하므로 4 word를 바로 더해도 항상 2bit 이내로 표현이 되므로 4-input CSA를 사용하여 역시 2개의 word로 줄일 수가 있

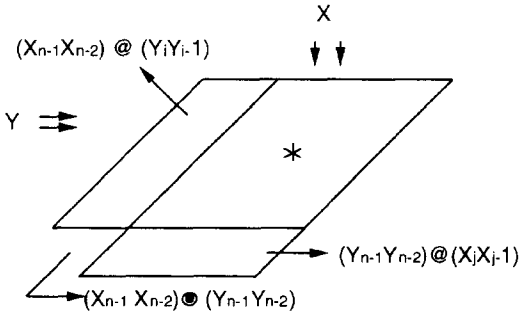
다. 결국, 총 word의 개수가 4개로 줄어들게 된다. 그림 1에서 p10~p13, p12~p15의 쌍들이 각각 앞의 정리를 만족한다는 사실은 p10~p13의 경우, C⇒a, Q⇒b, D⇒c, P⇒d, B⇒e 와 같이, p12~p15의 경우는 P⇒a, C⇒b, Q⇒c, B⇒d, O⇒e 와 같이 대입하여 보면 알 수 있다.

3. 2의 보수에 대한 고려

피승수와 승수가 2의 보수로 표현된 경우에는 Baugh-Wooley의 방법이나 Dadda의 방법을 이용하여 곱셈기를 구현할 수 있다. 본 논문에서는 Dadda의 방법을 이용하여 구현하였다. 그림 2는 4 bit의 경우에 Dadda의 방법을 이용한 곱셈기를 구성한 예를 보여준다. 이러한 경우에는 각 부분곱셈기들 중 일부(좌변과 하변)는 실제로는 곱셈기가 아닌 수정된 operation을 수행하여야 하는데 N-bit 곱셈에 대하여 N-2개의 type-1 곱셈기와 1개의 type-2 곱셈기가 필요하며 각각의 출력은 다음과 같다. (그림 3)



(a)



(b)

그림 2. Dadda의 방법 (a) 4 비트 (b) 일반적인 경우

Fig. 2. Dadda's method (a) 4 bit case (b) general case.

Type-I :

$A \ @ \ B = 4(a_1b_1)' + 2[(a_1b_0)' + a_0b_1] + a_0b_0$

$(A \ ◎ \ B = 4(a_1b_1)' + 2[(a_0b_1)' + a_1b_0] + a_0b_0 = B \ @ \ A)$

Type-II :

$A \ ◉ \ B = 4a_1b_1 + 2[(a_1b_0)' + (a_0b_1)'] + a_0b_0$

type - 1

(a1a0)@(b1b0)

	a1	a0
X	b1	b0
	$\overline{a1b0}$	a0b0
$\overline{a1b1}$	a0b1	

type - 2

(a1a0)◉(b1b0)

	a1	a0
X	b1	b0
	$\overline{a1b0}$	a0b0
a1b1	$\overline{a0b1}$	

그림 3. 2의 보수를 고려한 변형된 형태의 곱셈 오퍼레이션

Fig. 3. Modified operations of the multiplication in case of 2's complement.

N bit의 두 수를 곱하는 경우에 이들 두 숫자가 각각 2의 보수로 표현되어 있다면 피승수와 승수는 모두 다음과 같이 표현된다.

$$A = -a_N 2N + \sum_{i=0}^{N-1} a_i 2^i \cdot {}^{B} = -b_N 2^N + \sum_{i=0}^{N-1} b_i 2^i$$

이들 두수의 곱은 다음과 같이 표현된다.

$$A*B = 2_N b_N 2^{2N} + \sum_{i=0}^{N-1}\sum_{i=0}^{N-1} a_i b_j 2^{i+j} + \sum_{i=0}^{N-1} \overline{a_N b_i} 2^{N+i} + 2^{N-1} - 2^{2N-1}$$

따라서 Dadda의 방법은 일반적으로 그림 2과 같이 나타낼 수 있다. 제안된 기법을 사용하면서 2의 보수를 고려한 8bit 곱셈기를 Dadda의 방법을 이용하도록 구성할 수 있는데 이것을 그림 4에 나타내었다. 16bit 곱셈기의 경우에도 그림 5와 같이 구성할 수가 있다.

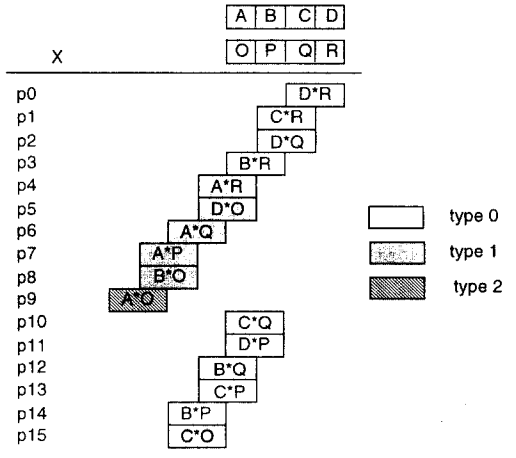


그림 4. 2의 보수를 고려한 8bit 곱셈기
Fig. 4. 8bit multiplier with consideration of 2's complement.

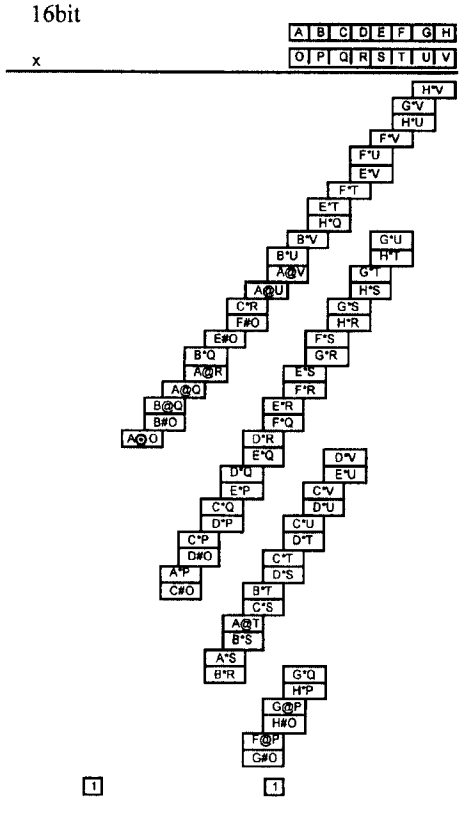


그림 5. 2의 보수를 고려한 16bit 곱셈기
Fig. 5. 16bit multiplier with consideration of 2's complement.

수정된 곱셈 operation을 포함하고 있는 경우에도 8bit의 경우와 16bit의 경우 모두 4-word group 내에서 인접한 4개의 부분 곱들은 항상 앞 절에서 설명한 정리를 만족한다.

Ⅲ. 16bit 곱셈기의 설계

1. Sub-block의 설계

2x2 부분곱셈기를 그림 6과 같이 설계하였다. 2x2 부분곱셈기와 4-input CSA를 Domino logic을 이용하여 설계하였는데 2x2 submultiplier의 출력은 모두 non-inverting이므로 이를 통하여 Domino logic의 특징인 non-inverting 출력을 그대로 이용할 수가 있으며 2x2 부분 곱셈기와 4-input CSA를 하나의 pipeline stage로 묶을 수가 있다.

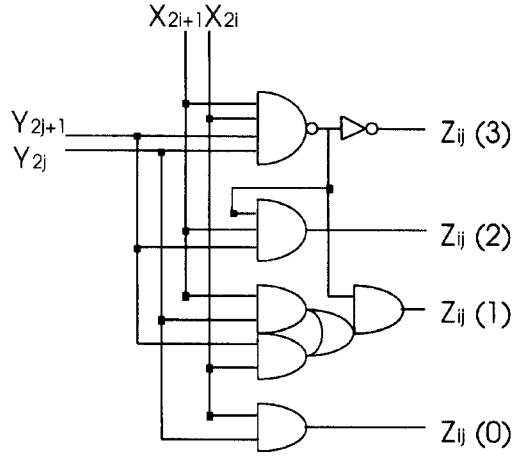


그림 6. 2x2 부분 곱셈기에 대한 블럭 다이어그램
Fig. 6. Schematic diagram of 2x2 submultiplier.

Booth 인코딩을 사용하는 경우에도 부분곱을 생성하기 위하여 두 가지의 function block을 거쳐야 하는데 그것들은 각각 Booth decoder와 Booth selector이다. 각각의 회로도를 그림 7과 그림 8에 나타내었다. Booth decoder의 경우는 그림 7과 같이 5개의 출력 중 항상 하나의 출력은 ground와의 DC path가 형성되기 때문에 원천적으로 Domino logic을 사용하여 구현할 수가 없다.

한편 그림 8의 Booth selector에서는 많은 정도의 delay가 예상된다. 왜냐하면 다수의 개수 $(N^2/2)$가 필요한 Booth selector에서는 transistor의 개수를 줄이기 위하여 pass transistor logic을 사용하는데 thre-

shold 전압에 의한 전압강하를 방지하기 위하여 PMOS를 사용한 positive feedback이 신호와 충돌을 일으켜 입력 신호가 들어왔을 때의 discharge 전류를 감소시키기 때문이다. 또 다른 이유로는 같은 row에 있는 N+1개의 Booth selector의 각 입력을 Booth decoder가 구동해야 하므로 decoder의 면적을 충분히 크게 하여주어야 한다.

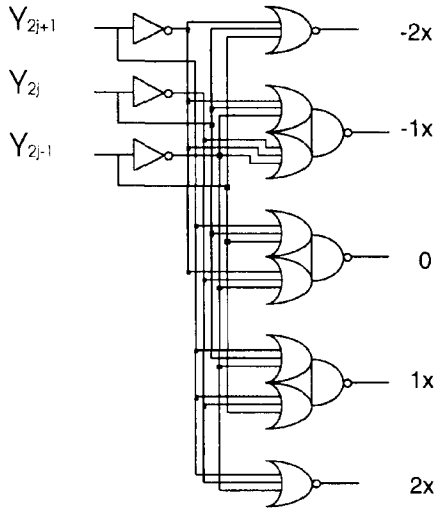


그림 7. 부스 디코더
Fig. 7. Booth decoder.

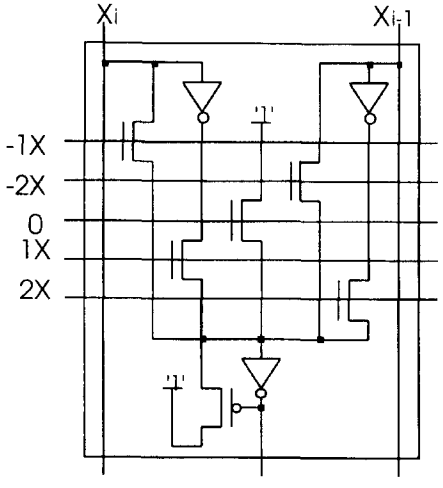


그림 8. 부스 셀렉터
Fig. 8. Booth selector.

그렇지 않으면 예를 들어 16bit 곱셈기의 경우 하나의 decoder가 17개의 selector를 충분히 구동시켜 주지 못하게 되어 많은 delay가 발생하게 된다.

0.8μm CMOS 공정을 사용한 경우의 2x2 부분 곱셈기와 4-input CSA의 layout을 그림 9와 그림 10에 보였다.

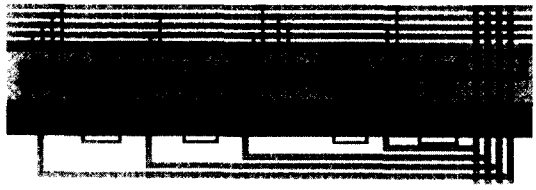


그림 9. 2x2 부분 곱셈기의 레이아웃
Fig. 9. Layout of 2x2 submultiplier.

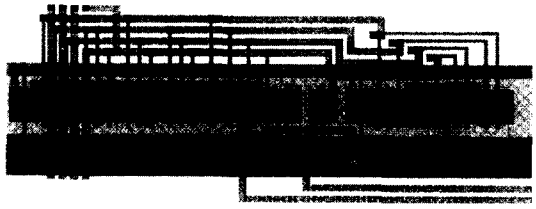


그림 10. 4:2 변환기의 레이아웃
Fig. 10. Layout of 4-input CSA.

2. 전체 곱셈기 설계

전체 곱셈기를 위에서 설계한 sub-block들과 Wallace tree를 이용하여 설계하였다. 전체 곱셈기의 block diagram을 그림 11에 보였다. 2x2 부분곱셈기 내의 숫자들은 각각의 출력 부분곱이 어느 group에 속하는지를 나타낸다. 설계에 사용한 technology는 VTI사의 0.8μm 표준 CMOS 공정이다. (1-poly 2-metal; metal width 1.4μm). Booth 인코더와의 비교를 위하여 Booth's multiplier를 같은 library를 이용하여 설계하였다.

제안된 multiplier의 인코딩 block (2x2 부분 곱셈기)이 외부입력을 직접 받아들이는 반면, Booth's selector는 인코더로부터 입력을 받아들이기 때문에 이론적으로 Booth 인코더의 transistor의 크기는 selector의 transistor크기의 16배가 되어야 하지만 설계에서는 약간의 delay를 감수하도록 하고 적당한 크기로 조정하여 두 multiplier의 전체 크기가 비슷하게 되도록 조정하였다. 두 곱셈기의 전체 면적은 각각 Booth 인코더는 6000μm x 2500μm, 제안된 인코더는 5000μm x 3400μm 이다.

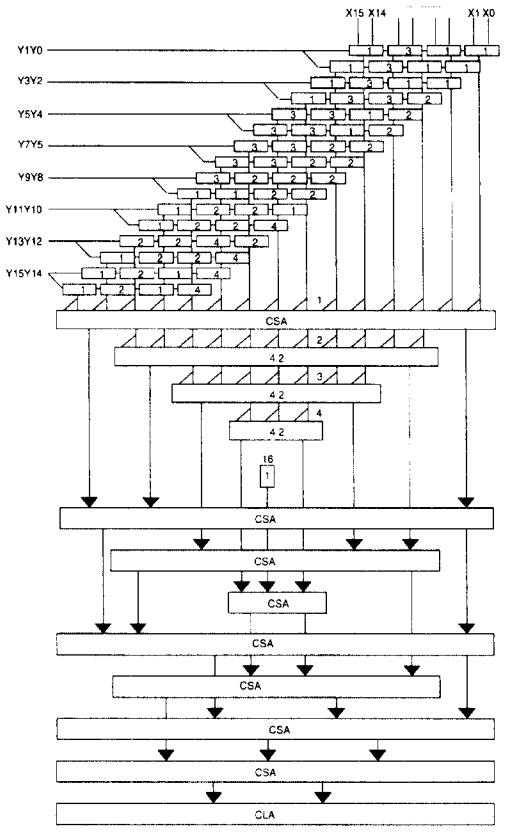


그림 11. 제안된 기법을 사용한 16비트 곱셈기 블럭 다이어그램
Fig. 11. Block diagram of multiplier using the proposed scheme.

VTI사의 Compass design tool의 0.8mm CMOS library를 사용하여 두 종류의 곱셈기 회로를 layout 한 뒤 parasitic element들을 고려하여 회로를 extract한 뒤 HSPICE로 simulation한 결과를 그림 12에 정리하였으며 제안된 인코딩 방법을 사용한 multiplier의 전체 layout을 그림 13에 보였다.

먼저 Booth 인코더 및 제안된 인코더를 precharge 한 후 입력(Y-input)을 인가했을 때부터 부분곱이 생성되기까지의 delay를 비교하였다. Booth 인코더의 경우는 부분곱을 생성하기까지의 delay time은 $T_{Booth}=T_{Booth_decoder}+T_{Booth_selector}$이고, 제안된 인코더의 경우의 delay time은 $T_{Prop}=T_{2x2\text{-}submultiplier}+T_{4\text{-}input_CSA}$이 된다. 그림 12를 보면 Booth decoder의 출력(중간단 그래프, 실선)은 sign bit을 포함하여 17개의 Booth selector 입력을 drive해야 하므로 rising time이 긴 반면에(time delay는 약 0.8ns) 제안된 2x2 부분 곱셈기는(하단 그래프, 실선) 외부 입력을 받아서 단 하나의 4-input CSA 입력을 drive하므로 rising time이 훨씬 짧고 time delay도 약 0.5ns정도이다. 한편, 두번째 stage, 즉, Booth selector의 출력(중간단 그래프, 점선)과 4-input CSA의 출력(하단 그래프, 점선)은 각각 0.46ns, 0.54ns로 큰 차이를 보이지 않음을 알 수 있다. 입력이 들어와서 최종 출력(Wallace tree로의 입력)이 나갈 때까지 Booth encoder/selector는 0.8ns+0.46ns=1.26ns, 제안된 인코더는 0.5ns+0.54ns=1.04ns로 제안된 인코더가 약 0.22ns 빠름을 알 수 있다.

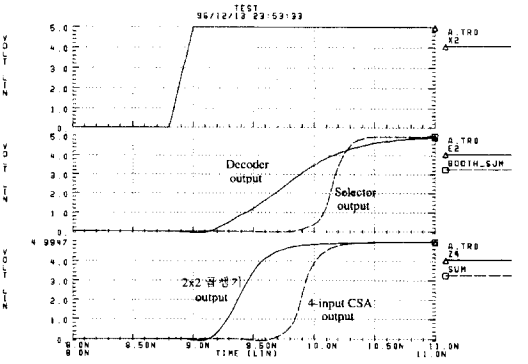


그림 12. 제안된 방식과 Booth의 인코딩 방식의 시뮬레이션 출력 파형
Fig. 12. Simulated output waveforms of the proposed and Booth's scheme.

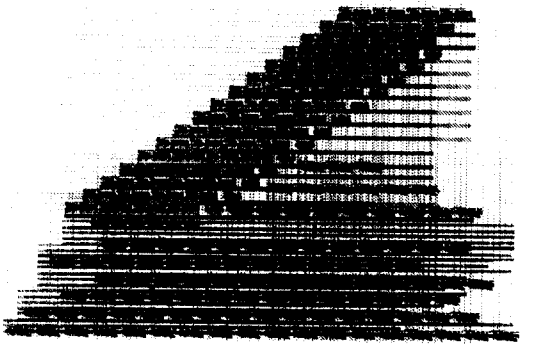


그림 13. 제안된 16비트 곱셈기의 레이아웃
Fig. 13. Layout of the proposed 16bit multiplier.

32bit 이상의 곱셈기에서는 제안된 방식을 이용하면 기존의 방식보다 더 빠른 속도를 가질 수 있을 것으로 예상된다. 곱셈기의 전체 delay중 인코딩, 즉 부분곱 생성까지의 delay가 차지하는 비중은 대략 15%정도

이나, IEEE double precision multiplication에 쓰이는 54bit 곱셈기와 같이 입력 bit 수가 많은 곱셈기에서는 Booth selector들의 load capacitance가 매우 커서 인코딩에서 소비되는 delay가 더 많은 비중을 차지하게 된다. 실제로 이러한 경우에 Booth decoder의 출력을 tapered buffering하여 내보내주는 등의 기법을 사용하고 있다[9]. N-bit multiplier의 경우, 제안된 2x2 부분 곱셈기는 단 하나의 4-input CSA를 drive하는 반면, Booth 인코더의 경우는 N+1개의 Booth selector들을 drive하게 되므로 load capacitance가 N에 비례하여 커지므로 otpimal buffering하여 주는 경우에도 Booth 인코더의 시간 지연은 제안된 방식에 비해 $\log_2 N$ 에 비례하여 증가하게 된다[11]. 결론적으로 제안된 인코딩 방법은 X입력(multiplicand)과 Y입력(multiplier)이 서로 대칭적이며 또한 이들 외부입력이 인코더내의 block들을 직접 구동하므로 Booth 인코딩에서와 같이 한쪽 입력에 대한 delay가 critical해지는 일이 없게 되며 따라서 전체적으로 빠른 속도를 가지게 된다.

Ⅳ. 결 론

2x2 부분곱셈기와 이들로부터의 원시 부분곱들을 재정렬함으로써 전체 부분곱의 개수를 반으로 줄이는 새로운 부분곱 생성 방법을 제안하였다. 제안된 방식의 근본적 원리인 곱셈 특유의 내재된 규칙을 이용하여 원시 부분곱들을 적절하게 재정렬함으로써 캐리 전파가 없는 4:2 변환을 단일 stage로서 가능하도록 하였다.

본 연구에서 새로이 발견한 4진수 곱셈에 대한 정리를 이용하여 Booth 인코딩 방식에 대하여 몇가지 장점을 갖는 새로운 인코딩 방식을 제안하였으며 이를 이용한 16bit 곱셈기를 설계하였다. 제안한 인코딩 방식은 승수와 피승수 입력에 대하여 구조 및 지연시간에 있어서 서로 대칭적이며 이론적으로 bit수가 많아질수록 Booth 인코딩보다 더 작은 delay를 가진다. 0.8μm technology를 이용하여 곱셈기를 설계하였으며 시뮬레이션을 통해 기존의 Booth 인코딩 방식에 비해 같은 면적에 대해서 더 빠른 특성을 가짐을 확인하였다. 설계를 최적화하여 면적과 속도를 더 향상시킬 수 있을 것으로 기대된다. 같은 원리를 적용하여 24bit 혹은 54bit 곱셈기를 제작하는 것이 가능하며 이러한 고속 곱셈기는 멀티미디어를 위한 MPEG encoder/decoder나 실시간 신호처리 등에 광범위하게 사용될 수 있을 것이다.

참 고 문 헌

[1] K. Yano, Y. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi and A. Shimizu, "A 3.8-ns CMOS 16 x 16-b multiplier using complementary pass-transistor logic," *IEEE JSSC*, vol. 25, no. 2, Apr. 1990, pp. 388-395.

[2] M. Uya, K. Kaneko, and J. Yasui, "CMOS floating point multiplier," *IEEE International Solid-State Circuits Circuits Conference, Digest of Technical Papers*, Feb. 1984, pp. 90-91.

[3] R. F. Lyon, "Two's complement pipeline multiplier," *IEEE Transaction on Communication*, vol. COM-24, Apr. 1976, pp. 418-425.

[4] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, Mcgraw-Hill, 1984.

[5] A. Booth, "A signed Binary Multiplication Technique," *Quarterly Journal of Mechanical and Applied Mathematics*, vol. 4, no. 2, 1951.

[6] J. Hennessy, "VLSI Processor Architecture," *IEEE Transaction on Computer*, December, 1984.

[7] P. R. Cappello and K. Steiglitz, "A VLSI Layout for a Pipelined Dadda Multiplier," *ACM Transcations on Computer Systems*, 1983, pp. 157-174.

[8] J. Vuillemin, "A Very Fast Multiplication Algorithm for VLSI Implementation," *the VLSI Journal*, 1983, pp. 39-52.

[9] M. Hanawa, K. Kaneko, T. Kawashimo, and H. Marayama, "A 4.3ns 0.3μm CMOS 54 x 54b Multiplier using Precharged Pass-Transistor Logic," *IEEE International Solid State Circuits Conference, Digest of Technical Papers*, 1996, pp. 364-365.

[10] S. Hilker, N. Phan, and D. Rainey, "A 3.4ns 0.8μm BICMOS 53b x 53b Multiplier Tree," *IEEE International Solid State Circuits Conference, Digest of Technical Papers*, 1994, pp. 292-293.

[11] R. C. Jaeger, "Comments on 'An optimized output stage for MOS integrated circuits'," *IEEE J. Solid-State Circuits*, vol. 27, pp. 473-483, Apr. 1992.

[12] J. Kernhof, M. Selzer, M. A. Beunder and *et al*, "Mixed Static and Domino Logic on the CMOS Gate Forest," *IEEE JSSC*, vol. 25, no. 2, Apr. 1990, pp. 396-402.

저 자 소 개

李 相 球(正會員)

현재 한남대학교 컴퓨터공학과 교수

全 英 淑(正會員)

1996년 한남대학교 컴퓨터공학과 졸업. 현재 한남대학교 대학원 컴퓨터공학과 석사과정중. 관심분야는 병렬처리, 컴퓨터구조 등임