論文2000-37SD-9-8

저전력 설계를 위한 절단된 Booth 곱셈기 구조
(A Truncated Booth Multiplier Architecture for Low Power Design)

李 光 鉉*, 林 種 錫*
(Kwang HyunLee and Chong Suck Rim)

요 약

본 논문에서는 DSP등에서 응용될 수 있는 저전력 곱셈기를 제안한다. 많은 DSP 어플리케이션에서 곱셈기의 모든 출력을 사용하는 것이 아니라, 그중 상위 비트만을 취해서 사용한다. Kidambi는 이런 개념에 기본하며 절단된 곱셈기를 제안하였다. 본 논문에서는 이 개념을 실제로 사용이 가능한 Booth 곱셈기에 적용한다. 이전 논문에서는 고려하지 않은 0 입력에 대한 0 출력을 보장하였다. 그리고, 비트수 확장법을 제안하여 더욱더 오차를 감소시켰다. 그리고, 이 필터를 FIR 필터 설계에 적용하여 더욱 효율적으로 회로를 구성할 수 있음을 확인하였다.

Abstract

In this paper, we propose a hardware reduced multiplier for DSP applications. In many DSP applications, all of multiplier products were not used, but only upper bits of product were used. Kidambi proposed truncated unsigned multiplier for this idea. In this paper, we adopt this scheme to Booth multiplier which can be used real DSP systems. Also, zero input guarantees zero output that was not provided in previous paper. In addition, we propose bit extension scheme to reduce truncation error more and more. And, we adopted this multiplier to FIR filters for more efficient design.

Ⅰ. 서 론

Kidambi는 DSP 응용회로에서 사용 가능한 절단된 (truncated) 곱셈기[2](그림 1)를 제안하였다. IIR 필터 같은 피드백이 있는 회로에서는 곱셈기의 모든 출력을 사용하는 것이 아니라, 그 중 일부분만 사용하여 회로를 구성하게 된다[1]. 이 곱셈기는 이런 경우에 사용하지 않는 하위 비트 출력을 계산하는 부분을 잘라 내어 면적을 절약할 수 있는 곱셈기이다. 이 곱셈기는 회로의 손실로 인해서 결과에 오차가 발생하게 되는데, 확률적인 접근을 통해서 오차를 계산한 뒤에, 이를 교정함으로써 오차를 최소화하는 방법을 사용하였다.

그러나, 제안한 곱셈기는 unsigned 곱셈기이며, 이를 signed magnitude 연산에 확대 적용하여 사용하였다. 그리고, 2의 보수 연산이 가능한 Baugh-Wooly 곱셈기[3]에 적용 가능함을 언급하였지만, 실제 회로에서 사용하기 위해서는 Booth 곱셈기[4]와 같은 고성능 곱셈기에 적용이 될 필요가 있다. 그리고, 이 곱셈기는 오차 분석 과정에서 하위 비트에 대한 고려를 하지 않았는데, 실제 회로에서는 이를 고려해서 오차를 더욱 계산할 필요가 있다. 또한, 이 곱셈기는 0 입력에 대해서 0 출력을 보장하지 못한다는 단점이 있다. 오차 교정으로 인해서 0 이 아니라 교정된 오차가 결과로

* 正會員, 서두로직 MyCAD 개발부
(Seodu Logic, Inc, MyCAD development dept.)
* 正會員, 西江大學校 컴퓨터工學部
(Dept. of Computer Science and Engineering, Sogang University.)
接受日字:2000年 6月24日, 수정완료일:2000年 7月29日

나가게 된다.

본 논문에서는 Booth 곱셈기에 대해서 하위 비트 부분을 잘나내어 면적을 절약하는 절단된 Booth 곱셈기를 제안한다. 확률적인 접근 방법을 사용하여 오차를 계산하였고, 실제로 곱셈기 회로를 디자인하여, 면적과 전력 소모의 이득을 비교하였다.

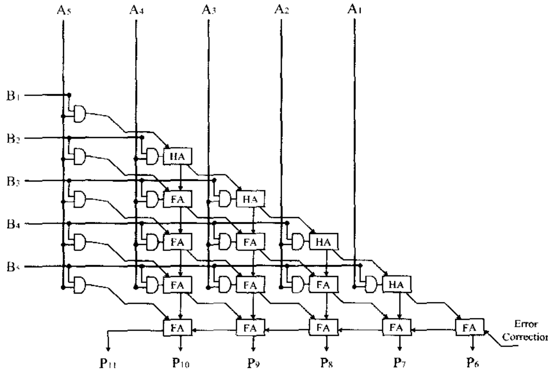

그림 1. 6×6 비트 절단된 unsigned 곱셈기
Fig. 1. A 6×6 truncated unsigned multiplier.

Ⅱ. 절단된 Booth 곱셈기

1. Booth 곱셈 알고리즘

Booth는 2의 보수 곱셈 연산을 효율적으로 수행하기 위한 알고리즘을 제안하였다[4]. 제안된 알고리즘은 승수에서 나타나는 연속된 k개의 '1'을 변환을 통해서 $2k+1-1$로 바꾸어 필요한 연산을 줄인다. 좀더 자세하게 설명하면, 뺄셈 연산을 수행하는 경우 $\bar{1}$로 표현한 고, 0으로 시작되는 연속된 1의 값을 가질 경우, 이는 $011\ldots1 = 0\cdot2^m + 1\cdot2^{m-1} + 1\cdot2^{m-2} + \ldots + 1\cdot2^k$ 으로 표현할 때, $100\ldots\bar{1} = 1\cdot2^m + (-1)\cdot2^k$ 으로 변환하여 계산을 한다. 그리고, 모든 연속된 1의 값을 변환한 뒤에, 중간에 $1\bar{1}$ 또는 $\bar{1}1$이 나오는 경우에는 이를 한번 더 처리하여, 01과 $0\bar{1}$로 한번 더 바꾸어 준다. 예를 들어 승수의 값이 '0111101100100011'이라고 할 때, 연속된 1을 변환하면 '$1000\bar{1}10\bar{1}01\bar{1}0010\bar{1}$'이 된다. 이를 한번 더 처리하면, '$10000\bar{1}0\bar{1}0010010\bar{1}$'이 된다. 이렇게 변환하면, 처음 값의 경우 9번의 덧셈이 필요하지만, 마지막으로 변환된 값은 6번의 덧셈/뺄셈으로 계산이 가능해 진다. 이런 방법을 통해서, 곱셈 연산을 효율적으로 수행할 수 있다.

2. Booth 곱셈기의 구조

Booth 곱셈기는 이러한 Booth 곱셈 알고리즘을 회로로 구현한 것이다. Booth 곱셈기는 일반적으로 승수 값 2 비트를 한번에 처리하는 방법을 사용한다. 2 비트를 한번에 처리하기 때문에, 전체적인 구조는 승수가 N비트 일 경우 $N/2$개의 열(row)로 구성된다(그림 2). 각 열은 승수 2비트와 하위 1비트 값으로부터 덧셈 또는 뺄셈 연산을 결정하는 Booth 부호기(encoder)와 여기서 결정된 신호로부터 계산에 사용할 값을 만들어 내는 Booth 선택기(selector)가 있으며, Booth 선택기로부터 나온 값을 사용하여 실제로 계산을 하는 Booth 가산기가 있다. 특별히 최상 단의 경우 상위로부터 나오는 중간 결과가 없기 때문에, 이를 위한 Booth first회로가 있다. 그리고, 최하 단에는 각 열로부터 얻어진 중간 결과 값으로 최종 결과를 얻어 내는 CPA가 있다.

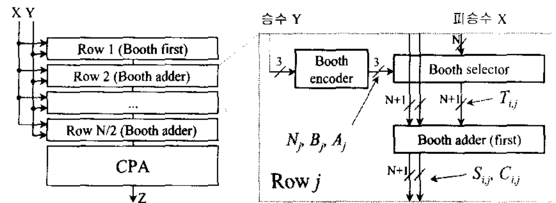

그림 2. Booth 곱셈기의 개략 구조
Fig. 2. Glohal strucrure of the Booth multiplier.

Booth 부호기는 승수 값으로부터 해당 열에서 수행해야 할 연산을 정의하여, 세 출력 N_j, B_j, A_j로 출력을 내보내는 회로이다. 해당 열에서 처리해야할 2비트의 승수 값과 하위 비트 1비트를 사용해서 add X, add 2X, sub X, sub 2X, no operation의 5가지 연산 중 하나를 선택하는 회로이다. Booth 부호기에 대한 진리표는 표 1에 나와 있다.

표 1. Booth 부호기에 대한 진리표
Table 1. Truth table for Booth encoder.

Y_{2j+1}	Y_{2j}	Y_{2j-1}	Operation	N_j	B_j	A_j
0	0	0	no op.	0	0	0
0	0	1	add x	0	0	1
0	1	0	add x	0	0	1
0	1	1	add 2x	0	1	0
1	0	0	sub 2x	1	1	0
1	0	1	sub x	1	0	1
1	1	0	sub x	1	0	1
1	1	1	no op.	1	0	0

세 비트의 값이 000 인 경우는 아무 것도 하지 않는 경우이며, 001 이나 011의 경우는 연속된 1의 시작으로 가정하고 덧셈 연산을 하는 경우로 각각 X, 2X 값을 더해준다. 010 의 경우는 그대로 X를 더해주는 연산이다. 111 의 경우는 연속된 1들의 중간으로 변환되어 000처럼 아무 연산도 하지 않는다. 100과 110의 경우는 연속된 1의 끝으로 가정하여 각각 2X와 X를 빼주는 연산을 한다. 101의 경우는 연속된 1들 내부에 들어 있는 0으로써, $\bar{1}10$로 바뀐 후에, $0\bar{1}0$로 다시 바뀌는 경우이다. 즉, X 값을 빼주는 연산을 한다. 이러한 연산에 따라서 Booth 부호기의 출력 N_j은 덧셈일 경우 0, 뺄셈일 경우 0의 값을 가진다. 그리고, B_j는 2X 값을 더해주거나 빼줄 경우 1이 되며, A_j는 X값을 더해주거나 빼줄 경우에 1이 된다.

Booth 선택기는 Booth 부호기로부터 나온 세 출력 N_j, B_j, A_j로부터 실제로 계산에 사용할 값을 만들어 내는 회로이다. 아무 연산도 하지 않는 $B_j=1$, $A_j=0$ 일때는 0 값을 내보내며, X를 더하거나 빼주는 경우인 $B_j=0$, $A_j=1$일 때는 X_i 값을, 그리고 2X를 더하거나 빼주는 $B_j=1$, $A_j=0$일 때는 한 비트 쉬프트 된 X_{i+1} 값을 사용한다. 그리고, 뺄셈 연산을 수행하는 $N_j=1$일 경우에는 결정된 값을 반전시킨 후, 나중에 1을 더해줌으로써 2의 보수를 취해서 Booth 선택기의 출력 $T_{i,j}$을 구하게 된다. Booth 선택기의 진리표는 표 2와 같다.

표 2. Booth 선택기에 대한 진리표
Table 2. Truth table for Booth Selector.

N_j	B_j	A_j	Operation	$T_{i,j}$
0	0	0	no op.	0
0	0	1	add x	X_i
0	1	0	add 2x	X_{i+1}
1	0	0	no op.	not(0)=1
1	0	1	sub x	not(X_i)
1	1	0	sub 2x.	not(X_{i+1})

Booth 선택기로부터 계산에 사용할 값 $T_{i,j}$가 결정되면, Booth 가산기 에서는 상단에서 나온 Booth 가산기의 중간 결과와 Booth 선택기의 출력을 사용하여 실제 계산을 수행하고, 이로부터 출력 $S_{i,j}$, $C_{i,j}$를 구한다. 최상위 두 비트에 대해서는 2의 보수 연산을 위한 inverter회로가 있으며, 나머지 비트들은 보통의 전가산기를 사용하여 구성한다. 그리고, 최하위 비트에는 뺄셈 연산을 위해서, N_j 가 1인 경우 1을 더해주어 2의 보수 뺄셈 연산을 수행한다. 그리고, 최상 단열의 경우에는 상단에서 내려오는 중간 결과가 없기 때문에, Booth 가산기가 아닌 Booth first 회로를 두어 중간 결과를 만들어 낸다. 그리고, Booth 가산기와 Booth first회로의 출력 중 하위 두 비트 부분은 최종 결과를 위해서 따로 모아지며, 나머지 부분은 하단의 Booth 가산기로 들어가서, 다음 열의 연산에 사용된다. Booth first 회로와 Booth 가산기 회로의 상세한 회로도는 그림 3과 그림 4에 보인다.

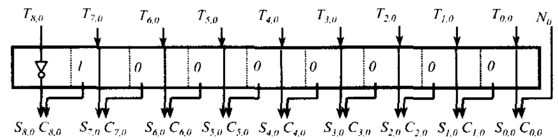

그림 3. 8×8 비트 곱셈기의 Booth first 상세 회로
Fig. 3. Booth first circuit in 8×8 multiplier.

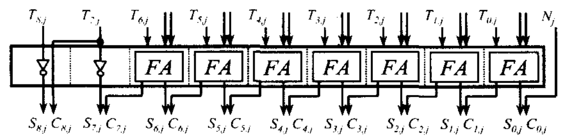

그림 4. 8×8 비트 곱셈기의 Booth 가산기 상세 회로
Fig. 4. Booth adder blocks in 8×8 multiplier.

마지막으로, 각 열에서 나온 하위 두 비트 부분과 최하 단의 Booth 가산기에서 나온 출력들을 모두 더해서 최종 결과를 구하는 부분은 CPA이다. 일반적으로 고속 연산을 위해서 속도가 빠른 Manchester adder와 같은 덧셈기 구조를 사용하여 구성한다. Booth 곱셈기의 전체 회로는 그림 5에 보인다. 8비트의 피승수와 8비트의 승수가 곱해져서 15비트의 출력을 내보내는 회로이다.

3. 절단된 Booth 곱셈기

본 논문에서는 Booth 곱셈기의 출력 중 상위 비트 부분의 일부만을 내보내도록 하고, 하위 비트를 계산하는 부분을 잘라 낸 절단된 Booth 곱셈기를 제안한다. 예를 들어 그림 5의 곱셈기 회로중 상위 비트인 $Z[7]...Z[14]$를 계산하는 부분만 남겨 두고, 하위 비트인 $Z[0]...Z[6]$을 계산하는 부분을 잘라 내어 곱셈기를 재구성한다. 다시 말해서, cut line을 기준으로 왼쪽 부분만이 남아 곱셈기의 회로를 재구성하게 된다.

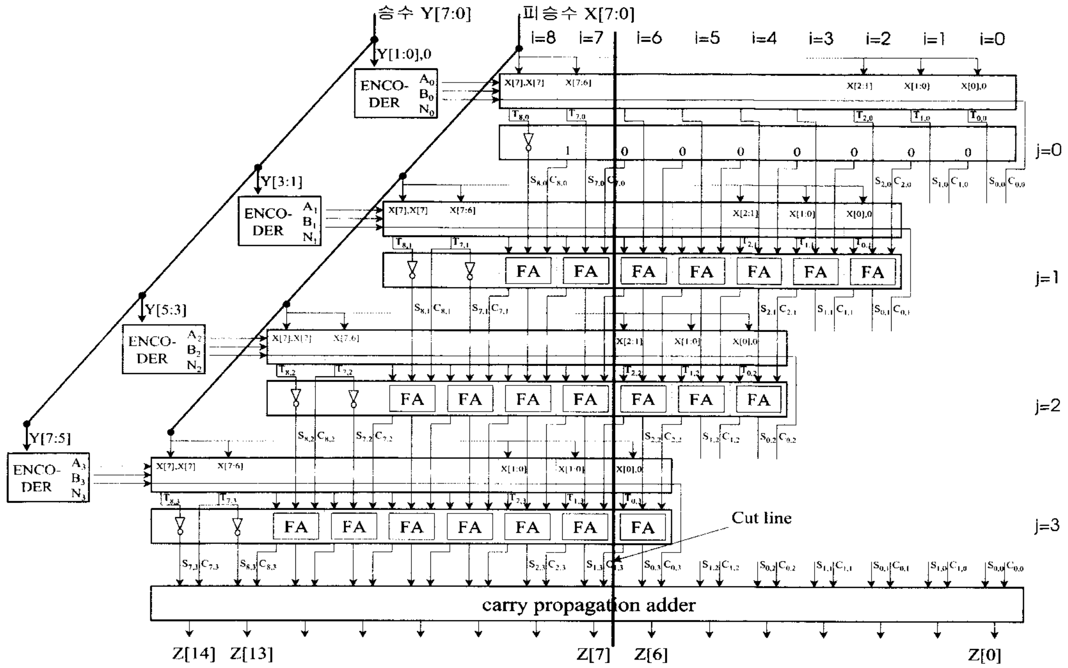

그림 5. 8×8 비트 Booth 곱셈기의 전체 회로
Fig.5.Enrire 8×8 Booth multiplier block diagram.

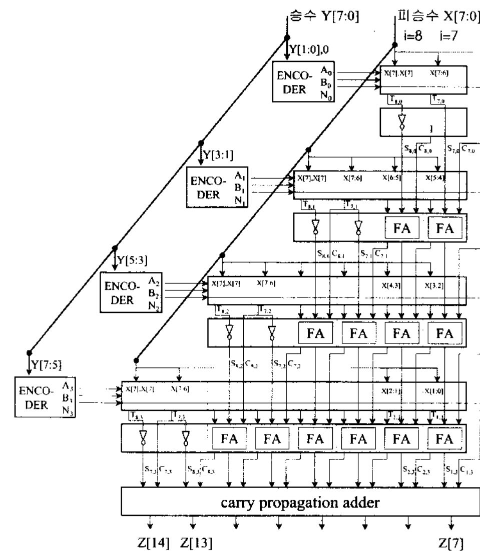

그림 6. 8×8 비트 절단된 Booth 곱셈기
Fig. 6. Truncated 8×8 Booth multiplier.

몇 번째 열인지에 따라서 적절한 비트 수만큼 Booth 선택기와 Booth 가산기 그리고 Booth first회로를 잘라 낸다. 그리고, CPA도 필요 없는 비트 수 만큼만을 잘라 내어서 면적의 이득을 얻도록 구성한다.

이런 식으로 하위 비트 계산 부분을 잘라 내어 곱셈기 면적에 대한 이득을 얻을 수 있으며, 하위 비트 계산하는 부분을 잘라낸 절단된 Booth 곱셈기의 구조는 그림 6에 보인다.

이렇게 하위 비트를 계산하는 회로가 잘려 나갈 경우, 뺄셈 연산시 2의 보수 연산을 위해 반전시킨 결과에 $1(=N_j)$을 더해 주는데, 이 부분도 같이 잘려 나간다. 그래서, 만약 피승수가 0이어서 모든 비트들이 '0'을 가지고 있을 때, 반전되어 모두 '1'이 된 후에 이 값에 1이 더해져서 모두 '0'으로 다시 한번 반전되어야 하지만, 그렇게 되지 않는다. 이로 인해서 얻어지는 결과는 0에 가까운 값이 되지만, 0은 아니기 때문에 0 입력에 대해서 0출력이 보장이 되지 않을 수 있다. 본 논문에서는 이를 위해서 뺄셈일 경우 $1(=N_j)$을 더해 주는 부분을 필요한 만큼 쉬프트 시켜서 더해줌으로써 문제를 해결할 수 있다.

4. 출력 비트 수 확장

하위 비트 부분을 계산하는 부분을 잘라 내었기 때문에, 이 곱셈기는 결과 값에 항상 오차가 포함되어 있다. 곱셈 결과에 이 오차만큼을 교정하여, 오차를 줄이는 방법이 가능하다. 그러나, 이 방법으로는 오차의

분포까지는 바뀌지 않으며 1 이하의 값에 대해서는 교정이 어렵다는 문제점이 있다. 이런 방법 외에 곱셈기의 오차를 줄이기 위해서 출력 비트 수를 확장하는 방법이 가능하다. 비트 수가 증가함에 따라서 정밀도는 그만큼 높아지며, 오차의 분포는 좁아지게 된다. 필요한 만큼 비트 수를 증가시켜, 원하는 만큼의 정확도를 만족하도록 할 수 있다. 당연히 비트 수가 증가하는 만큼 필요한 회로는 증가하고, 곱셈기 면적의 이득은 감소하게 된다. 그림 7은 1비트를 확장하여 9비트 결과를 얻어내는 8×8 비트 Booth 곱셈기의 구조이다.

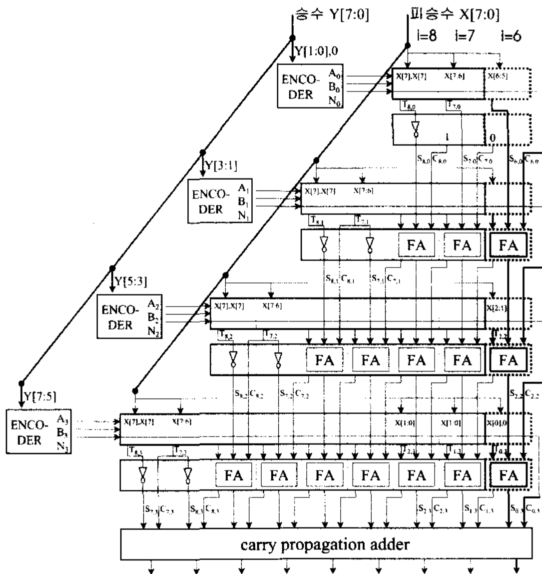

그림 7. 1비트 확장한 8×8 비트 절단된 곱셈기
Fig. 7. Truncated Booth multiplier with 1 bit extenscan.

5. 절단된 곱셈기의 오차 분석

절단으로 인해 하위 비트 부분이 잘려 나갔기 때문에, 당연히 계산 오차가 발생하게 된다. 본 논문에서는 이런 오차를 확률적인 접근을 통해서 분석, 계산하였다. 회로 내부의 모든 출력들에 대해서 확률을 구한 뒤, 이를 이용하여 발생하는 오차를 예상하는 것이 가능하다. 이렇게 계산된 오차는 후에 교정을 위해서 사용이 가능하며, 그 곱셈기의 대한 한가지 특성으로 나타나게 된다.

확률을 구하기 위해서 먼저 각 신호들에 대한 정의를 한다. 첨자 i, j는 행과 열을 나타낸다. 그림 5에서처럼 행은 제일 오른쪽이 0행이며, 왼쪽으로 가면서 1씩 증가한다. 열을 제일 위쪽이 0열이며, 아래쪽으로 가면서 1씩 증가한다. P_{X_i}, P_{Y_i}는 각각 승수(multiplicand) X, 피승수(multiplier) Y의 i번째 비트가 1일 확률이다. P_{N_j}, P_{B_j}, P_{A_j}는 j열의 Booth 부호기의 세 출력 N_j, B_j, A_j가 각각 1일 확률이다. $P_{T_{i,j}}$는 j열 i행의 Booth 선택기의 출력 $T_{i,j}$가 1일 확률이다. $P_{S_{i,j}}, P_{C_{i,j}}$는 Adder row의 각 블록에서의 출력 $S_{i,j}$, $C_{i,j}$가 1일 확률이다.

곱셈기의 두 입력 피승수와 승수는 각각 k 비트이며, 피승수와 승수의 각 비트가 1일 확률이 0.5라고 가정한다.

$$P_{X_i} = P_{Y_i} = 0.5 \quad (\text{for all } 0 \le i < k, \text{ 가정}) \qquad (1)$$

먼저 각 열에 대한 Booth 부호기의 출력들이 1일 확률(수식 2, 3, 4)을 계산한다. 이는 Booth 부호기의 진리표로부터 계산할 수 있다. Y_i가 1일 확률이 0.5라고 가정하였기 때문에, 이로부터 쉽게 구해질 수 있다.

$$P_{N_j} = P_{Y_{2j+1}} = 0.5 \qquad (2)$$

$$P_{A_j} = 0.5 \qquad (3)$$

$$P_{A_j} = 0.5 \; P_{N_j} = P_{Y_{2j-1}} = 0.5 \qquad (3)$$

$$P_{B_j} = 0.25 \qquad (\text{for all } 0 \le j < k/2) \qquad (4)$$

구해진 Booth 부호기의 출력으로부터 Booth 선택기의 출력인 $T_{i,j}$의 확률(수식 5)을 구한다. 이 역시 Booth 선택기의 진리표로부터 얻어 진다.

$$\begin{aligned} P_{T_{i,j}} &= P_{N_j} \cdot (P_{A_j} \cdot P_{X_i} + P_{B_j} \cdot P_{X_{i+1}}) \\ &\quad + Q_{N_j} \cdot [1 - (P_{A_j} \cdot P_{X_i} + P_{B_j} \cdot P_{X_{i+1}})] \\ &= 0.5 \cdot 0.375 + 0.5 \cdot 0.625 = 0.5 \end{aligned} \qquad (5)$$

그리고, Booth first회로(j =0)의 출력에 대한 확률(수식 6,7)을 계산한다. $S_{i,0}$의 경우 최상위 비트는 Booth 선택기의 출력 $T_{i,j}$ 가 inverter를 거쳐 나오고, 나머지 비트들은 $T_{i,j}$가 그대로 나온다. 그리고, $C_{i,0}$의 경우는 최상위 비트의 경우 1이며, 중간은 0, 최하위 비트는 N_0가 된다

$$P_{S_{i,0}} = \begin{cases} 1 - P_{T_{i,0}} & (i=k) \\ P_{T_{i,0}} & (0 \le i \le k-1) \end{cases} \qquad (6)$$

$$P_{C_{i,0}} = \begin{cases} 1 & (i=k) \\ 0 & (0 \le i \le k-1) \\ P_{N_0} = 0.5 & (i=0) \end{cases} \qquad (7)$$

이제 각 Booth 가산기($j \geq 1$)에서의 출력이 1일 확률(수식 8.1, 8.2, 9.1, 9.2, 9.3)을 구한다. 상단에서 구해진 Booth first 또는 Booth 가산기의 출력과 Booth 선택기의 출력으로부터 구해진다. $S_{i,j}$의 경우 최상위의 두 비트는 Booth 선택기의 출력 $T_{i,j}$가 inverter를 거쳐 나오고, 나머지의 경우는 전가산기의 sum 출력이 1일 확률을 계산하여 얻어진다. 그리고, $C_{i,j}$의 경우는 최상위 비트의 경우 $T_{i-1,j}$이며, 최하위 비트는 N_j가 되고, 중간의 경우는 전가산기의 carry 출력이 1일 확률로부터 계산되어 진다.

$$P_{S_{i,j}} = 1 - P_{T_{i,j}} \qquad (k-1 \leq i \leq k) \qquad (8.1)$$

$$\begin{aligned} P_{S_{i,j}} = & P_{T_{i,j}} \cdot P_{S_{i-1,j-1}} \cdot P_{C_{i,j-1}} \\ & + P_{T_{i,j}} \cdot Q_{S_{i-1,j-1}} \cdot Q_{C_{i,j-1}} \\ & + Q_{T_{i,j}} \cdot P_{S_{i-1,j-1}} \cdot Q_{C_{i,j-1}} \\ & + Q_{T_{i,j}} \cdot Q_{S_{i-1,j-1}} \cdot P_{C_{i,j-1}} \end{aligned} \qquad (0 \leq i \leq k-2) \qquad (8.2)$$

$$P_{C_{i,j}} = P_{T_{i-1,j}} \qquad (i=k) \qquad (9.1)$$

$$\begin{aligned} P_{C_{i,j}} = & P_{T_{i,j}} \cdot P_{S_{i-1,j-1}} \cdot P_{C_{i,j-1}} \\ & + P_{T_{i,j}} \cdot P_{S_{i-1,j-1}} \cdot Q_{C_{i,j-1}} \\ & + P_{T_{i,j}} \cdot Q_{S_{i-1,j-1}} \cdot P_{C_{i,j-1}} \\ & + Q_{T_{i,j}} \cdot P_{S_{i-1,j-1}} \cdot P_{C_{i,j-1}} \end{aligned} \qquad (1 \leq i \leq k-1) \qquad (9.2)$$

$$P_{C_{i,j}} = P_{N_j} = 0.5 \qquad (i=0) \qquad (9.3)$$

이런 방법을 통해서 곱셈기 내부의 모든 라인들에 대해서 1 값을 가지게 될 확률(수식 10)을 구할 수 있고, 이로부터 절단된 Booth 곱셈기의 오차를 계산할 수가 있다. 차례대로 모든 열의 $P_{S_{i,j}}$와 $P_{C_{i,j}}$가 구해졌으면, 이로부터 cut line에 걸쳐져 있는 carry들의 확률을 모두 더해 준다. 이 값이 carry로 인해서 생기는 오차의 예상 값이 된다.

$$E(C) = \sum_{\text{for each } C_{i,j} \text{ of Cut line}} P_{C_{i,j}} \qquad (10)$$

Carry에 의한 오차뿐만 아니라 하위 비트 출력이 곱셈이 결과에 미치는 영향도 고려해야 한다. CPA로 들어가게 되는 $S_{i,j}$과 $C_{i,j}$들의 확률을 각각 구해서 자릿수에 맞게 더해 주어서 구한다(수식 11).

$$E(LSB) = (P_{C_{0,k/2}} + P_{S_{0,k/2}}) \cdot 2^{-1} + \sum_{j=0}^{k/2-1} [(P_{C_{0,j}} + P_{S_{0,j}}) \cdot 2^{2j-1-k} + (P_{C_{1,j}} + P_{S_{1,j}}) \cdot 2^{2j-2-k}] \qquad (11)$$

그리고, 2의 보수 뺄셈 연산을 위해서 N_j를 이동시켜 주었는데, 이 역시 오차에 영향을 준다. 그러나, 이 N_j를 쉬프트 시켜서 더해줌에 따라 생기는 오차는 음의 값을 가지며, 이는 결과 값을 $0.5 = P_{N_j}$ 만큼 교정하는 효과(수식 12)를 가진다. N_j에 의한 오차는 각 열마다 이루어지게 되며, N_j로 인한 오차 교정 기대값은 0.5이므로

$$E(N) = -\sum P_{N_j} = -0.5 \cdot (\textit{number of rows}) \qquad (12)$$

이 된다. 그러므로 전체 오차의 기대값 $E(Tr)$ (수식 13)은 이 세 가지 오차를 더해 줌으로 구할 수 있고, 이는 다음과 같이 표현될 수 있다.

$$E(Tr) = E(C) + E(LSB) + E(N) \qquad (13)$$

위에서 설명한 방법을 확률적인 접근으로 통해서 계산한 예상 오차와, C언어를 사용하여 4, 8, 16, 24, 32 비트 곱셈기에 대해서, 무작위 입력을 넣어 실험한 결과를 정리하여 표 3에 보인다.

표 3. 절단된 Booth 곱셈기의 예상 오차 및 실제 오차

Table 3. Errors in a Booth multiplier.

비트 수	이론적 계산치				실제 결과치
	$E(C)$	$E(LSB)$	$E(N)$	$E(Tr)$	$E'(Tr)$
4	0.188	0.718	-1.0	-0.094	-0.075
8	0.954	0.924	-2.0	-0.122	-0.124
16	2.890	0.995	-4.0	-0.125	-0.129
24	4.876	0.999	-6.0	-0.125	-0.127
32	6.875	1.000	-8.0	-0.125	-0.129

오차는 unsigned 절단된 곱셈기와는 달리, N_j로 인한 영향에 의해서, 비트 수가 커질수록 오차 값이 증가하는 것이 아니라, 일정 값에 수렴함을 확인할 수 있다. 다만, 비트 수가 커질수록 오차의 분산은 증가한다. 이는 열이 증가할수록 P_{C_i}가 0.5에 수렴하기 때문에, N_j에 의한 -0.5와 상쇄되기 때문이다.

그리고, 출력의 비트 수를 확장한 경우에도 이와 똑같은 방법을 사용하여 계산할 수 있다. 출력 비트 수를 1, 2, 3 비트씩 증가시킨 곱셈기들에 대해서도 오차의 평균과 오차의 분산을 정리하여 표 4와 표 5에 보였다. 그리고, 출력 비트 수를 확장시킨 후에, 다시 반올림을 통해서 자릿수를 맞추었을 경우의 오

차의 평균과 분산은 표 6과 표 7에 정리되어 있다. 1비트나 2비트 확장 후에 반올림하면 오히려 오차가 커지게 되는 현상이 발생하는 데, 이는 반올림이 하위 비트의 비트 수가 너무 작기 때문이다. 이런 경우 반올림에 의한 오차는 0이 아니며, 이로 인해 곱셈기의 오차에 영향을 받는다. 3비트 이상 확장한 후에 반올림을 하면, 오차가 줄어드는 것을 확인 할 수 있다.

표 4. *Var(Tr)* 계산표 (반올림 없음)
Table 4. Truncated values of *Var(Tr)*.

비트 수	no extension	1 bit ext.	2 bits ext.	3 bits ext.
4	0.232	0.033	0.004	0.000
8	0.550	0.114	0.025	0.005
16	1.153	0.263	0.064	0.014
24	1.745	0.416	0.101	0.024
32	2.254	0.522	0.129	0.030

표 5. *E(Tr)* 계산표 (반올림 없음)
Table 5. Truncated values of *E(Tr)*.

비트 수	no extension	1 bit ext.	2 bits ext.	3 bits ext.
4	-0.075	-0.026	0.001	0.000
8	-0.124	-0.061	-0.028	-0.014
16	-0.123	-0.069	-0.031	-0.015
24	-0.120	-0.070	-0.032	-0.015
32	-0.127	-0.063	-0.031	-0.016

표 6. *E(Tr)* 계산표 (반올림한 결과)
Table 6. Rounded values of *E(Tr)*.

비트 수	no extension	1 bit ext.	2 bits ext.	3 bits ext.
4	-0.075	-0.224	-0.107	-0.077
8	-0.124	-0.307	-0.156	-0.076
16	-0.123	-0.309	-0.150	-0.084
24	-0.120	-0.308	-0.150	-0.082
32	-0.127	-0.311	-0.157	-0.078

표 7. *Var(Tr)* 계산표 (반올림한 결과)
Table 7. Rounded values of *Var(Tr)*.

비트 수	no extension	1 bit ext.	2 bits ext.	3 bits ext.
4	0.232	0.091	0.068	0.068
8	0.550	0.176	0.101	0.085
16	1.153	0.328	0.139	0.094
24	1.745	0.492	0.177	0.104
32	2.254	0.586	0.210	0.113

그리고, MAC(Multiply-ACcumulate) 회로(그림 8)와 같이 곱셈 결과를 계속 누적하여 사용하는 경우, 적은 양의 오차라도 누적되어 큰 오차가 될 수 있다. 이러한 오차의 누적은 연산에 지장을 줄 정도로 크게 영향을 줄 수도 있는데, 이런 오차 누적을 피하기 위해서, 별도의 오차 교정 회로를 사용할 수 있다. 비트를 확장하지 않은 경우 곱셈기의 오차는 -0.125로 8번 누적이 되면 -1이 된다. 즉, 8번의 곱셈 연산마다 1을 더해 줌으로써 평균 오차를 0으로 교정하는 것이 가능하다.

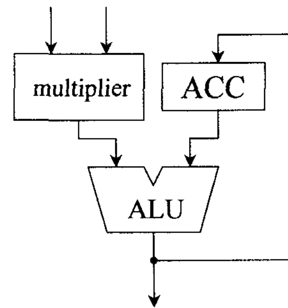

그림 8. MAC 회로
Fig. 8. MAC block diagram.

6. 곱셈기의 레이아웃

절단된 Booth 곱셈기는 전체적인 회로 형태가 삼각형이다. 이런 모습으로는 회로의 면적에 이득이 있다고 말할 수 없다. 면적 이득을 보다 확실하게 하기 위해서는 곱셈기의 전체 모습을 삼각형에서 사각형으로 바꾸어 줄 필요가 있다. 이를 위해 두 가지 방법 folding과 concatenation을 제안한다.

Folding이란 회로를 접는 것을 의미한다. 삼각형 모양의 회로의 일부를 잘라 접어 올림으로써, 전체적인 모양을 사각형 형태로 구성하는 것이 가능하다(그림 9). 하단 회로의 일부분을 잘라서 공간의 여유가

있는 상단으로 접어 올린다. 그리고, 이를 연결하기 위해서 부가적인 배선을 해 준다. 이로 인해서, 회로의 정형성이 많이 무너진다는 점과, 이를 위해서 배선을 위한 공간이 많이 필요하다는 점이 단점이 될 수 있지만, 면적 이득을 확실히 할 수 있다는 장점이 크다.

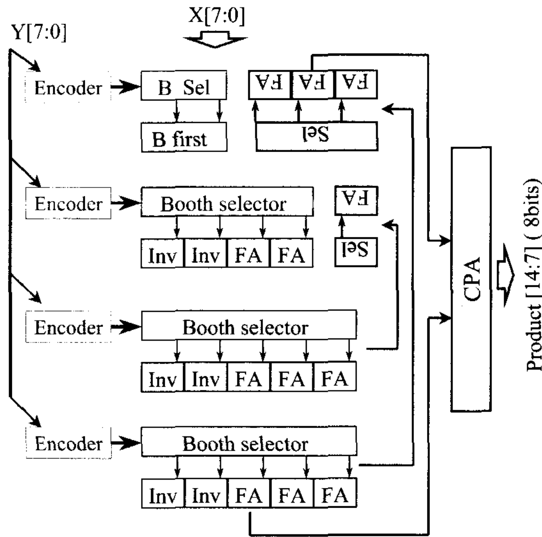

그림 9. Folding 후의 절단된 Booth 곱셈기
Fig. 9. Truncated Booth multiplier after folding.

Folding이 하나의 곱셈기의 면적을 사각형으로 만드는 것이라면, concatenation은 두 개의 곱셈기를 덧붙여서 사각형 형태로 만드는 것이다. FFT, FIR 필터 같은 많은 양의 곱셈기를 사용하는 회로에서 사용할 수 있는 방법이다. 두 개씩 짝을 지어 하나의 블록처럼 구성할 수 있다. 이럴 경우, 부가적인 배선이 필요하지 않다는 장점이 있지만, 데이터의 흐름이 한쪽 방향이 아니라는 점이 문제가 될 소지는 있다. 서로 반대 방향으로 데이터가 흐르기 때문에, 회로 설계자가 이러한 사실을 고려하고 회로를 설계한다면 큰 문제점은 되지 않을 것이다.

7. 면적 및 전력 소모 비교

곱셈기의 면적 이득을 실제로 확인하기 위해서, LG 0.6 um 공정의 디자인 룰에 맞게 magic을 사용하여 회로를 제작하였다. 8비트, 12비트, 16비트에 대해서 각각 절단이 없는 경우, 절단한 경우, 1,2,3 비트 확장한 경우를 구성하여 비교하였다. 절단한 곱셈기의 경우 모두 folding을 사용하여 면적을 최소화시킨 구조이다. 각 회로에 대한 면적 비교는 표 8에 보인다. 표에 보인 회로 면적의 단위는 um2이다. 표에서도 알 수 있듯이 절단된 곱셈기는 약 37~42%, 1 비트 확장한 경우 약 27~36%의 면적 이득을 얻었다.

회로의 일부분이 잘려 나간만큼 전력 소모에서도 이득을 얻을 수 있었다. Magic으로 구성한 회로에서 필요한 데이터를 추출하여 irsim을 사용하여 시뮬레이션 하였다. C언어를 사용하여 무작위로 생성된 값을 입력으로 회로를 동작하였고, 얻어진 전력 소모량의 평균을 계산하였다. 표 9에 전력 소모에 대한 결과를 정리하였으며 단위는 mW/operation이다. 절단된 곱셈기는 대략 44 %, 1 비트 확장한 경우 27~38 %의 전력 이득을 얻었다.

표 8. 면적 계산표 (절단하지 않은 경우를 1로 정규화함)
Table 8. Normalized area of multipliers.

비트수	normal	truncation	1 bit ext.	2 bits ext.	3 bits ext.
8	1051×219 (1.00)	627×233 (0.63)	727×237 (0.73)	816×245 (0.85)	905×254 (0.98)
12	1479×327 (1.00)	837×342 (0.59)	937×346 (0.66)	1025×355 (0.75)	1114×363 (0.83)
16	1834×435 (1.00)	1030× (0.58)	1130×454 (0.64)	1219×463 (0.70)	1308×471 (0.77)

표 9. 전력 소모 계산표 (절단하지 않은 경우를 1로 정규화 함)
Table 9. Normalized power consumptions.

비트수	normal	truncation	1 bit ext.	2 bits ext.	3 bits ext.
8	3.404 (1.00)	1.899 (0.56)	2.498 (0.73)	2.797 (0.82)	3.099 (0.91)
12	8.923 (1.00)	4.976 (0.56)	5.783 (0.65)	6.424 (0.72)	7.103 (0.80)
16	18.036 (1.00)	10.031 (0.56)	11.101 (0.62)	12.181 (0.68)	13.250 (0.73)

절단하지 않은 8×8 비트 곱셈기의 실제 레이아웃(그림 11)과 절단한 곱셈기의 레이아웃(그림 10)을 보인다.

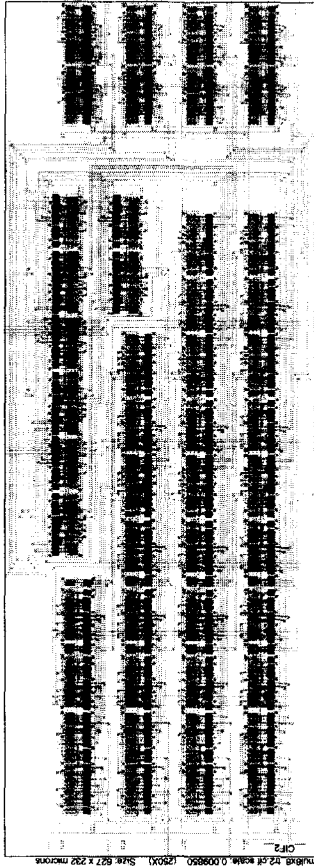

그림 10. 8×8 비트 절단된 곱셈기의 레이아웃(Folding 적용)
Fig. 10. Folded 8×8 truncated Booth multiplier layout.

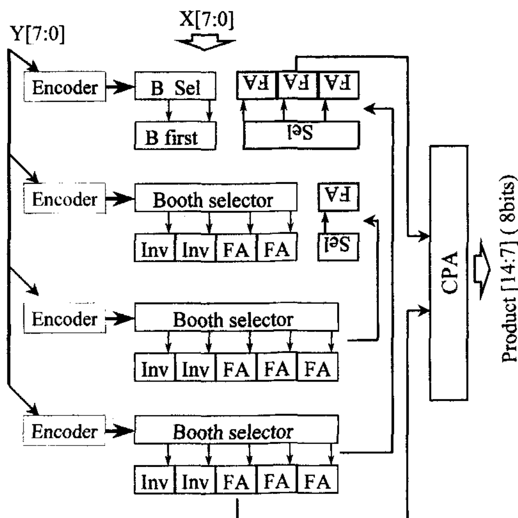

그림 12. 8×8 비트 곱셈기의 레이아웃
Fig. 12. Folded 8×8 truncated Booth multiplier block diagram.

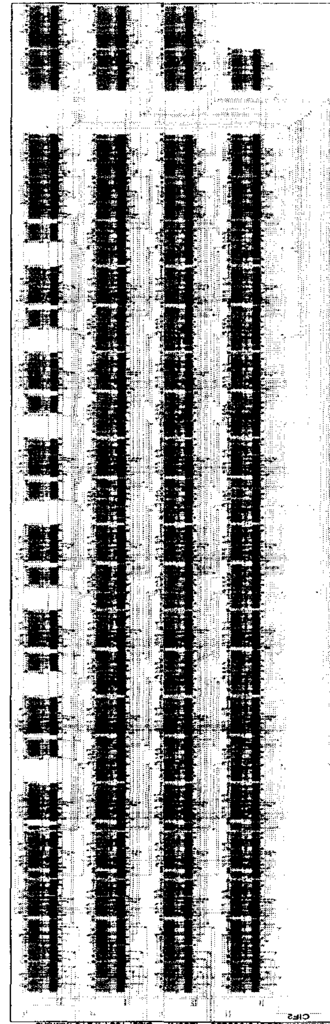

그림 11. 8×8 비트 곱셈기의 레이아웃
Fig. 11. Conrentional 8×8 truncated Booth multiplier layout.

Ⅲ. 응 용

본 논문에서는 이 곱셈기를 FIR(Finite Impulse Response) 필터[5][6]에 적용시켜 절단된 Booth 곱셈기를 사용할 경우의 회로의 크기를 비교하였다. 필터란 계속적으로 들어오는 입력에 대해서 특정한 주파수 대역만 취해서 내보내는 회로로, 이러한 필터 가운데 내부에 피드백이 없는 경우를 FIR 필터라고 한다.

원하는 필터의 특성을 만족하면서 최소한의 하드웨어로 필터를 구성하는 기존의 방법은 필터의 탭수를 조절하는 것이다. 본 논문에서는 절단된 Booth 곱셈기를 사용하여 원하는 필터 특성을 만족하도록 하는

방법을 사용하며, 이 두가지 방법에 대해 비교한다. 곱셈기의 기본 구조는 transposed 구조[7]을 사용하여 구성하였다. 주어진 조건은 입력 8 비트, 계수 12비트, 레지스터라인 14비트이며, passband ripple 은 3 dB이다, stopband attenuation 은 40 dB이상이다. 그리고, impulse response에 대한 mean square error가 0.03 이하이며, phase response error 가 0.001 rad 이하이다. 첫 번째 필터는 곱셈기 출력을 반올림하여 사용하며, 위 조건을 만족하는 최소의 탭수로 필터를 구성한 것이다. 두 번째 필터는 48 탭으로 구성되며, 절단된 Booth 곱셈기를 이용하여 최적화 시킨 경우이다.

두 필터에 대한 실험 결과는 표 10에 정리되어 있다. 첫 번째 필터는 조건을 만족하기 위해서 44탭이 필요하였으며, 이때의 stopband attenuation은 41.63 dB 이었다. 두 번째 필터는 최대한도로 절단 한 경우이며, 40.59 dB의 특성을 가졌다. VHDL 언어를 사용하여 회로를 합성하고, 회로의 크기를 비교한 결과, 절단된 곱셈기를 사용한 두 번째 필터가 15% 정도 더 적은 양의 하드웨어를 사용하여 구성하였다.

표 10. 탭 수 조절과 절단된 곱셈기 사용의 트레이드 오프 비교

Table 10. Comparison of using tap number controlled multiplier and truncated multiplier.

	필터 1	필터 2
입력	8 비트	
계수	12 비트	
레지스터	14 비트	
곱셈기	곱셈기 출력을 반올림하여 사용	절단된 곱셈기 사용
탭 수	44	48
회로 크기(게이트수)	30885	26254
Passband ripple (dB)	2.24	2.26
Stopband attenuation (dB)	41.63	40.59
Mean square error	2.695×10^{-5}	2.576×10^{-5}
Phase response error	0.0	0.0

Ⅳ. 결 론

본 논문에서 상위의 k 비트의 출력만을 내보내는 절단된 Booth 곱셈기의 구조를 제안하였다. 하위 비트에 대한 출력을 계산하는 부분을 잘라 내 버린 후에, 이로 인해 발생할 수 있는 오차를 확률적인 접근을 통해서 계산하였다. 또한 0 입력에 대한 0 출력을 보장하기 위해서 회로를 바꾸어 주었다. 그 결과, 절단으로 인한 오차는 무시할 수 있을 정도로 적다는 것을 확인할 수 있었으며, 또한, MAC 유닛과 같은 오차에 크게 영향을 받는 회로등에 사용하기 위해서, 보다 더 정확한 결과 값을 얻기 위한 비트수 확장법을 제안하였다. 실제 곱셈기를 설계하여서 비교한 결과, 절단개념을 사용하지 않은 곱셈기에 비해서 본 곱셈기는 면적이 대략 37~42% 감소 하였으며, 전력 소모는 대략 44% 감소함을 확인 할 수 있었다. 그리고, 이 곱셈기를 FIR 필터 설계에 사용하여, 같은 수준의 오차 범위 한계를 가지는 회로를 설계할 경우에 회로의 크기가 감소하는 것을 확인하였다.

참 고 문 헌

[1] Neil H. E. Weste, Kamran Eshraghian, *Principles of CMOS VLSI Design: A systems Perspective 2nd Ed.,* Addison-Wesley, 1993.

[2] Sunder S. Kidambi, Fayez El-Guibaly, Andreas Antoniou, "Area-Efficient Multipliers for Digital Signal Processing Applications," *IEEE Trans. on C&S-II*, vol. 43, pp. 90-95, NO. 2, FEB, 1996.

[3] Baugh, C. R. and Wooley, B. A., "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Trans. on Computers*, Vol. C-22, No. 1-2, Dec. 1973, pp.1045-1047.

[4] Booth, A.D., "A signed Binary Multiplication Algorithm," Quart. J. Mech. Appl. Math., Vol. 4, Pt. 2, 1951, pp. 236-240.

[5] Andreas Antoniou, *Digital Filters: Analysis, Design, and Applications*, 2nd Ed. McGraw-Hill, 1993.

[6] Alan V. Oppenheim, Ronald W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall, 1989.

[7] Eero Pajarre, Tapio Saramaki, "Efficient VLSI Implementation Techniques for FIR Filters," *DSPx 1994*, pp. 560-566, 1994.

저 자 소 개

李 光 鉉(正會員)
1974년 7월 22일생. 1997년 2월 서강대학교 전자계산학과 졸업(학사). 1999년 2월 서강대학교 컴퓨터학과 대학원 졸업(석사). 1999년 2월~현재 (주) 서두로직 전자기술연구소 연구원 재직중

林 種 錫(正會員) 第37卷 SD編 第2號 參照