고정계수 곱셈을 위한 비트패턴 전용덧셈기 설계

조경주*, 김용은**

Design of Bit-Pattern Specialized Adder for Constant Multiplication

Kyung-Ju Cho*, Yong-Eun Kim**

요 약

FIR 필터, DCT, FFT와 같은 디지털 신호처리 응용에서 다중 고정계수 곱셈의 효율적인 하드웨어 구현문제에 자주 접하게 된다. 고정계수 곱셈기 설계에서 공통 하위식 제거 알고리즘은 면적과 전력소모를 상당히 개선시킬 수 있는 방법을 제공한다. 본 논문에서는 CSD 계수에서 빈번히 나타나는 두 공통 하위식($10\bar{1}, 101$)의 덧셈을 수행하는 전용덧셈기 설계 방법을 제안한다. 제안한 방법을 radix-24 FFT 구조의 고정계수 곱셈블록에 적용한 실험에서 제안한 방법의 면적, 지연시간, 전력소비는 기존 방법보다 각각 21%, 11%, 12% 정도 향상됨을 보인다.

ABSTRACT

The problem of an efficient hardware implementation of multiple constant multiplication is frequently encountered in many digital signal processing applications such as FIR filter and linear transform (e.g., DCT and FFT). It is known that efficient solutions based on common subexpression elimination (CSE) algorithm can yield significant improvements with respect to the area and power consumption. In this paper, we present an efficient specialized adder design method for two common subexpressions ($10\bar{1}, 101$) in canonic signed digit (CSD) coefficients. By Synopsys simulations of a radix-24 FFT example, it is shown that the proposed method leads to about 21%, 11% and 12% reduction in the area, propagation delay time and power consumption compared with the conventional methods, respectively.

키워드

constant multiplier, bit-pattern adder, FFT, FIR filter

I. 서 론

곱셈기의 두 입력 중 계수가 정해진 곱셈기를 고정계수 곱셈기라고 한다. FIR 필터, FFT, DCT와 같은 선형변환과 영상처리 응용에서 곱셈계수가 고정된 경우, 고정계수 곱셈기를 사용하면 효율적으로 하드웨어를 구현할 수 있다.

고정계수 곱셈은 계수에서 0이 아닌 디지트에 따라 입력을 쉬프트하여 부분곱을 생성하고, 부분곱을 더해 곱(product)을 구하므로 Baugh-Wooly 또는 modified Booth 곱셈기와 같은 범용곱셈기를 사용하지 않고 구현할 수 있다. 이진계수에서 0이 아닌 디지트의 수를 줄이기 위해 계수를 CSD(Canonic Signed Digit)로 표현한다. CSD는 $\{1, 0, \bar{1}(=-1)\}$의 디지트 세트를 가지며, 두 개 이상 연속적으로 $1/\bar{1}$을 가질 수 없다[1].

FIR 필터와 같은 다중 고정계수 곱셈(multiple constant

* 전북대학교 BK21-전북 전자정보 고급인력양성 사업단
** 전북대학교 전자정보공학부

접수일자 2008. 08. 25

multiplication)이 사용되는 분야에서 곱셈블록의 면적을 감소키기 위해 다양한 CSE (Common Subexpression Elimination) 알고리즘에 관한 연구가 진행되었다[2-4]. CSE 알고리즘의 목적은 곱셈블록에서 공통 하위식(common subexpression)을 공유하여 연산의 수를 줄이는 것이다. 대부분의 연구에서 공통 하위식의 덧셈에 사용되는 덧셈기의 구현은 고려하지 않고, 최적의 공통 하위식을 찾는데 초점을 두었다[2-4].

본 논문에서는 CSD에서 가장 빈번히 나타나는 비트패턴 $10\bar{1}(\bar{1}01)$과 $101(\bar{1}0\bar{1})$의 덧셈을 효율적으로 계산할 수 있는 전용덧셈기를 설계한다.

Ⅱ 장에서 CSE의 개념과 이를 이용한 다양한 면적감소 알고리즘을 살펴보고, Ⅲ 장에서 비트패턴 $10\bar{1}(\bar{1}01)$과 $101(\bar{1}0\bar{1})$의 덧셈을 효율적으로 수행할 수 있는 전용덧셈기를 설계한다. Ⅳ 장에서 radix-24 FFT 구조에서 고정계수 곱셈블록에 전용덧셈기를 적용하고, 시뮬레이션을 통해 제안한 방법이 기존 방법보다 효율적임을 보인다. 끝으로, Ⅴ장에서 간단히 결론을 맺는다.

Ⅱ. CSE 알고리즘

1. CSE의 개념

다음과 같은 고정계수 곱셈에서 이진계수를 CSD 계수로 표현하면 0이 아닌 디지트의 수를 7개에서 4개로 줄여 효율적으로 구현할 수 있다.

$$1001111101\times x \;\Rightarrow\; 101000\bar{1}01\times x \qquad (1)$$

위 식에서 CSD 계수는 0이 아닌 디지트의 수가 4개이므로 일반 곱셈기 대신 쉬프트와 덧셈기/뺄셈기를 사용함으로써 더 효율적인 구현이 가능하다.

그림 1은 고정계수 곱셈의 설계에서 CSE의 개념을 설명하는 예제이다. 계수를 CSD 표현으로 변환하고, 0이 아닌 디지트에 대해 그림 1(b)와 같이 쉬프트와 덧셈을 수행한다. CSE는 CSD 계수 사이 또는 계수 내에서 한번 이상 나오는 비트패턴을 찾아 그 패턴을 공유함으로써 연산량을 줄인다. 그림 1(a)에서 비트패턴 $10\bar{1}$은 2번 나타난다. 여기서 $10\bar{1}$과 $\bar{1}01$은 동일한 패턴으로 간주한다.

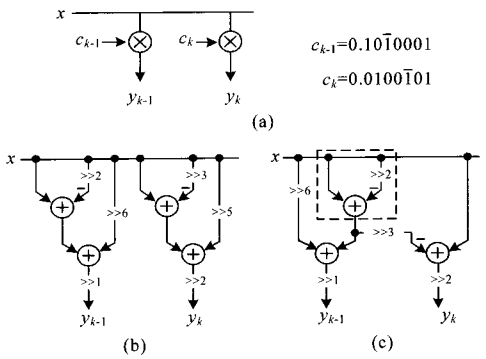

그림 1. CSE의 개념 예제
Fig. 1 Example of CSE concept

그림 1(c)처럼 비트패턴을 공유하면 그림 1(b)보다 쉬프트와 덧셈을 1번씩 감소시킬 수 있다.

2. 기존 CSE 알고리즘

Hartely[2]의 CSE 알고리즘은 FIR 필터의 CSD 계수 사이에서 공통 하위식의 위치에 따라 반복되는 공통 하위식을 회귀적으로 제거하는 방법이다. 또한, NR-SCSE(Nonrecursive Signed CSE) 알고리즘[4]은 비트패턴을 S1($10\cdots01$ 또는 $\bar{1}0\cdots0\bar{1}$)과 S2($10\cdots0\bar{1}$ 또는 $\bar{1}0\cdots01$)의 두 패턴으로 나누고, 각 단계마다 가장 많이 나타나는 공통 패턴을 찾아 공통 하위식을 제거하는 방법이다. 두 패턴 S1과 S2에서 최대 0의 개수는 입력 워드길이보다 2개 적다.

ITM(Iterative Matching) 알고리즘[3]은 두 signed-digit 사이에서 가장 많이 일치하는 비트패턴을 찾아 하위식을 제거하는 방법으로 일치하는 패턴이 생기지 않을 때까지 알고리즘을 반복 수행한다.

Ⅲ. 비트패턴 전용덧셈기 설계

1. 비트패턴 $10\bar{1}$에 대한 전용덧셈기 설계

그림 2는 입력의 워드길이(W)가 8비트이고, 비트패턴이 $10\bar{1}$인 부분곱의 덧셈을 나타낸다. 그림 2(a)처럼 범용덧셈기를 사용하여 더할 경우 10비트 덧셈기가 필요하지만, 그림 2(b)처럼 더하면 8비트 덧셈기면 된다. 또한, 체인입력 즉, $(i-1)$번째 스테이지의 입력 x_{i-1}이 i번째 스테이지 입력으로 다시 나타나는 특징을 이용하

면 범용덧셈기보다 효율적으로 설계할 수 있다.

그림 3에서 c_i는 $(i-1)$번째 스테이지에서 i번째 스테이지로 전파되는 캐리신호이다. 캐리신호를 효율적으로 계산하기 위해 $x_{i-1}=0$일 때 발생되는 캐리신호 c_i^0과 $x_{i-1}=1$일 때 발생되는 캐리신호 c_i^1로 구분하면 다음과 같이 표현할 수 있다.

$$c_i = \overline{x_{i-1}}c_i^0 + x_{i-1}c_i^1 \tag{2}$$

여기서, 연산자 '+'는 OR 연산을 나타내고, c_i^0과 c_i^1은 다음과 같이 표현할 수 있다.

$$c_i^0 = x_{i-2}c_{i-1}, \quad c_i^1 = x_{i-2} + c_{i-1} \tag{3}$$

합신호 s_i는 식 (2)와 (3), 드모르간 법칙을 이용하면 다음과 같이 표현할 수 있다.

$$\begin{aligned} s_i &= x_i \oplus x_{i-1} \oplus c_i = x_i \oplus [\overline{x_{i-1}}c_i + x_{i-1}\overline{c_i}] \\ &= x_i \oplus [\overline{x_{i-1}}(\overline{x_{i-1}}c_i^0 + x_{i-1}c_i^1) + x_{i-1}\overline{(\overline{x_{i-1}}c_i^0 + x_{i-1}c_i^1)}] \\ &= x_i \oplus [\overline{x_{i-1}}c_i^0 + x_{i-1}\overline{c_i^1}] \end{aligned} \tag{4}$$

여기서, 연산자 '$\oplus$'는 XOR 연산을 나타낸다.

$\overline{x_7}$	$\overline{x_7}$	$\overline{x_7}$	$\overline{x_6}$	$\overline{x_5}$	$\overline{x_4}$	$\overline{x_3}$	$\overline{x_2}$	$\overline{x_1}$	$\overline{x_0}$
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0		1
s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

(a) $10\overline{1}$

	x_7	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	
x_7	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	x_0

(b) 011

그림 2. $10\overline{1}$ 패턴에 대한 부분곱 덧셈
Fig. 2 Partial products addition methods for $10\overline{1}$

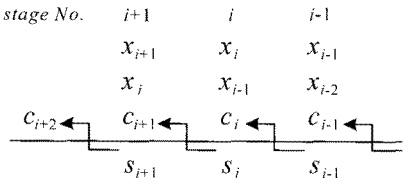

그림 3. 체인입력(011)의 덧셈
Fig. 3 Addition of chain inputs for 011

식 (2)과 (3)를 이용하면 c_{i+1}^0과 c_{i+1}^1을 다음과 같이 표현할 수 있다.

$$\begin{aligned} c_{i+1}^0 &= x_{i-1}c_i = x_{i-1}[\overline{x_{i-1}}c_i^0 + x_{i-1}c_i^1] \\ &= x_{i-1}c \end{aligned} \tag{5}$$

$$\begin{aligned} c_{i+1}^1 &= x_{i-1} + c_i = x_{i-1} + [\overline{x_{i-1}}c_i^0 + x_{i-1}c_i^1] \\ &= x_{i-1} + c_i^0 \end{aligned}$$

따라서, $(i+1)$번째 스테이지로 전달되는 캐리신호 c_{i+1}은 다음과 같이 표현 할 수 있다.

$$\begin{aligned} c_{i+1} &= \overline{x_i}c_{i+1}^0 + x_i c_{i+1}^1 \\ &= \overline{x_i}x_{i-1}c_i^1 + x_i(x_{i-1} + c_i^0) \end{aligned} \tag{6}$$

식 (4)와 (5)를 이용하면 합 신호 s_{i+1}는 다음과 같이 표현할 수 있다.

$$\begin{aligned} s_{i+1} &= x_{i+1} \oplus [\overline{x_i}c_{i+1}^0 + x_i\overline{c_{i+1}^1}] \\ &= x_{i+1} \oplus [\overline{x_i}x_{i-1}c_i^1 + x_i\overline{(x_{i-1} + c_i^0)}] \end{aligned} \tag{7}$$

식 (2)와 (6)으로부터 캐리신호를 계산하는데 소요되는 지연시간은 스테이지가 증가할수록 1개의 AND 또는 OR 게이트의 지연만큼 증가함을 알 수 있다. 또한, 식 (4)과 (7)로부터 합신호 s_i를 계산하는데 캐리신호 c_i^0과 c_i^1을 공유함으로써 면적을 줄일 수 있다.

그림 2(b)의 $s_1 \sim s_8$을 계산하면 다음과 같다.

$$\begin{aligned} s_1 &= x_1 \oplus x_0 \\ s_2 &= x_2 \oplus (\overline{x_1}c_2^0 + x_1\overline{c_2^1}) \quad (c_2^0 = 0,\ c_2^1 = x_0) \\ s_3 &= x_3 \oplus (\overline{x_2}c_3^0 + x_2\overline{c_3^1}) \quad (c_3^0 = x_1c_2^1,\ c_3^1 = x_1 + c_2^0) \\ s_4 &= x_4 \oplus (\overline{x_3}c_4^0 + x_3\overline{c_4^1}) \quad (c_4^0 = x_2c_3^1,\ c_4^1 = x_2 + c_3^0) \\ s_5 &= x_5 \oplus (\overline{x_4}c_5^0 + x_4\overline{c_5^1}) \quad (c_5^0 = x_3c_4^1,\ c_5^1 = x_3 + c_4^0) \\ s_6 &= x_6 \oplus (\overline{x_5}c_6^0 + x_5\overline{c_6^1}) \quad (c_6^0 = x_4c_5^1,\ c_6^1 = x_4 + c_5^0) \\ s_7 &= x_7 \oplus (\overline{x_6}c_7^0 + x_6\overline{c_7^1}) \quad (c_7^0 = x_5c_6^1,\ c_7^1 = x_5 + c_6^0) \\ s_8 &= c_8 = \overline{x_7}c_8^0 + x_7c_8^1 \quad (c_8^0 = x_6c_7^1,\ c_8^1 = x_6 + c_7^0) \end{aligned} \tag{8}$$

그림 4는 식 (8)에 대한 회로도이다. $s_2 \sim s_8$의MUX 연산은 그림 4(b)와 같이 설계할 수 있다. 그림 4(c)에서 알 수 있듯이 캐리신호 c_i^0과 c_i^1을 계산하는데 소요되는 시간은 입력 워드길이의 증가에 따라 2입력 AND 또는

OR 소자가 하나씩 증가한다. Critical path는 s_8이 아닌 s_7을 계산하는 경로이고, 그 계산시간은 4AND+3OR +XOR이다. 그림 4(b)의 cell 1과 그림 4(c)의 캐리생성 회로에 AND/OR-to-NAND/NAND 변환과정을 적용하면 그림 5와 같이 NAND, NOT, XOR 소자로만 구성된 회로를 얻을 수 있고, critical path의 계산시간을 7NAND+ XOR로 줄일 수 있다.

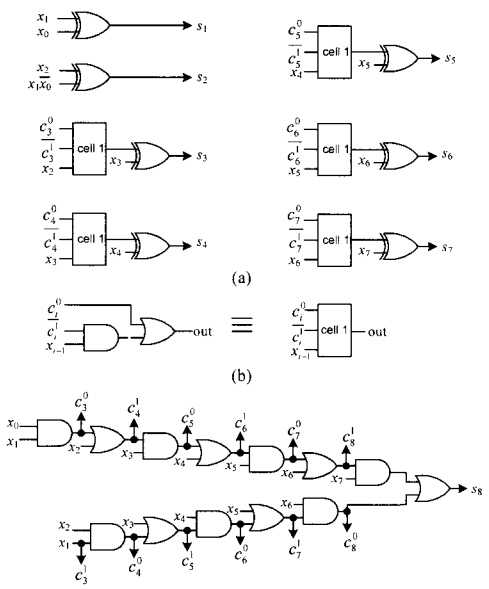

그림 4. 비트패턴 $10\bar{1}$을 위한 전용덧셈기(W=8)
Fig. 4 Specialized adder for $10\bar{1}$(W=8)

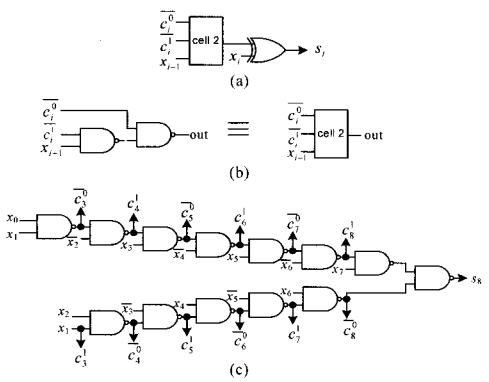

그림 5. 그림 4의 AND/OR-to-NAND/NAND로 변환
Fig. 5 Converted circuits of Fig. 4 using AND/OR-to-NAND/NAND conversion

표 1. 그림 2(b) 덧셈의 critical path와 정규화 지연 비교.
Table. 1 Comparison of critical path and normalized delay in Fig. 2(b)

	Critical path	정규화된 지연
RCA	7(AND+OR)+2XOR	11.8
CSA	3(AND+OR)+2 XOR+MUX	7.2
CLA	3(AND+OR)+2XOR	6.2
전용덧셈기	7NAND+XOR	4.5

입력 워드길이가 W일 경우, 이 전용덧셈기의 최대 지연시간은 ((W-1) NAND+XOR)이다.

표 1에 W=8인 경우에 대한 덧셈기의 critical path와 정규화된 지연을 비교하였다. 2입력 게이트의 정규화된 지연은 XOR(1.0), MUX(1.0), AND(0.7), OR(0.7), NAND (0.5), NOR(0.5)이다[6]. CSA(Carry Selection Adder)와 CLA(Carry Look-ahead Adder)는 4비트씩 2개의 그룹으로 나눈다고 가정하였다. 전용덧셈기의 면적(23 NAND+11NOT+7XOR)은 RCA(Ripple Carry Adder)의 면적(16XOR+16AND+8OR)보다 작으면서 고속 덧셈기보다 빠르다.

입력 워드길이가 큰 경우, 앞서 설명한 방법을 CSA에 적용할 수 있다. $c_1=0$이면 식 (8)에서 $s_1 \sim s_4$를 사용하고, $c_1=1$이면 $s_1 \sim s_4$와 c_5는 다음과 같이 구할 수 있다.

$$
\begin{aligned}
&s_1^1 = \overline{x_1 \oplus x_0} \\
&s_2^1 = x_2 \oplus (\overline{x_1} c_2^0 + x_1 \overline{c_2^1}) \quad (c_2^0 = x_0,\ c_2^1 = 1) \\
&s_3^1 = x_3 \oplus (\overline{x_2} c_3^0 + x_2 \overline{c_3^1}) \quad (c_3^0 = x_1 c_2^1,\ c_3^1 = x_1 + c_2^0) \\
&s_4^1 = x_4 \oplus (\overline{x_3} c_4^0 + x_3 \overline{c_4^1}) \quad (c_4^0 = x_2 c_3^1,\ c_4^1 = x_2 + c_3^0) \\
&c_5^1 = \overline{x_4} c_5^0 + x_4 c_5^1 \quad (c_5^0 = x_3 c_4^1,\ c_5^1 = x_3 + c_4^0)
\end{aligned}
\tag{9}
$$

2. 비트패턴 101에 대한 전용덧셈기 설계

비트패턴 101은 입력을 5배 해주므로 입력이 W비트라면 (W+3)비트 곱을 출력한다. 그림 6에서 $(i-1)$번째 스테이지의 입력 x_{i-1}은 $(i+1)$번째 스테이지의 입력으로 다시 나타난다.

캐리신호 c_i는 다음 식에 의해 계산할 수 있다.

$$c_i = g_{i-1} + c_{i-1} p_{i-1} \tag{10}$$

여기서 $g_{i-1}(=x_{i-1}x_{i-3})$와 $p_{i-1}(=x_{i-1}+x_{i-3})$는 두 입력($x_{i-1}$, x_{i-3})으로부터 생성된 generate signal과 propagate signal 이다.

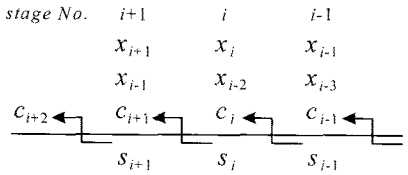

그림 6. 101의 체인입력의 덧셈
Fig. 6. Addition of chain inputs for 101

예를 들면, c_6은 식 (10)에 의해 다음과 같이 계산할 수 있다.

$$c_6 = g_5 + g_4p_5 + g_3p_4p_5 + g_2p_3p_4p_5 + c_2p_2p_3p_4p_5 \quad (11)$$

그림 6에서 $g_{i-1}=1(i \geq 3)$이면 $p_{i+1}=1$이므로 식 (11)에서 p_{i+1}을 생략할 수 있으며, $p_{i-1}p_{i+1}$은 $x_{i-1}+x_{i+1}x_{i-3}$로 간단히 할 수 있다.

이 패턴의 덧셈기를 CSA에 적용하여 설계하면 입력이 $x_5x_4x_3x_2$와 $x_3x_2x_1x_0$이고, $c_2=0$일 경우, s_i^0과 c_i^0은 다음과 같이 계산할 수 있다.

$$\begin{aligned} s_i^0 &= x_i \oplus x_{i-2} \oplus c_i^0 \\ c_3^0 &= g_2 \\ c_4^0 &= g_3 + g_2p_3 \\ c_5^0 &= g_4 + g_3p_4 + g_2p_3 \\ c_6^0 &= g_5 + g_4p_5 + g_3p_4 + g_2p_3p_5 \end{aligned} \quad (12)$$

또한, $c_{in}=1$이면 s_i^1과 c_i^1은 다음과 같다.

$$\begin{aligned} s_i^1 &= x_i \oplus x_{i-2} \oplus c_i^1 \\ c_3^1 &= g_2 + p_2 = p_2 \\ c_4^1 &= g_3 + p_2p_3 \\ c_5^1 &= g_4 + g_3p_4 + p_2p_3p_4 \\ c_6^1 &= g_5 + g_4p_5 + g_3p_4 + p_2p_3p_4p_5 \end{aligned} \quad (13)$$

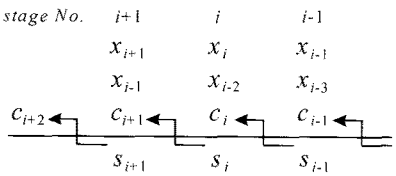

그림 6. 101의 체인입력의 덧셈
Fig. 6. Addition of chain inputs for 101

이 패턴의 경우도 범용덧셈기보다 적은 면적이 요구되며 고속으로 동작한다.

표 2. 식 (14)에서 고정계수의 CSD표현
Table 2. CSD representation of coefficients in (14)

계수	CSD 계수	$10\bar{1}$ 빈도	101 빈도
$\cos(\pi/8)$	$1.000\bar{1}01001$	0	1
$\sin(\pi/8)$	$0.10\bar{1}000\bar{1}01$	2	0
$\cos(\pi/4)$	$1.0\bar{1}0\bar{1}01010$	2	2

Ⅳ. 전용 덧셈기 응용 및 실험 결과

Radix-2^4 FFT 알고리즘[6]에 의해 DFT 식을 5차원 인덱스 맵으로 분해한 후 네 번째 스테이지까지 정리하면 두 번째와 세 번째 스테이지 사이에서 복소 고정계수 곱셈연산이 존재한다. 이것은 회전인자 $W_N^n(n=0\sim3)$와의 곱셈으로 삼각함수 공식을 이용하면 다음 식과 같이 표현할 수 있다.

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} W_N^0 \\ W_N^1 \\ W_N^2 \\ W_N^3 \end{bmatrix} X = \begin{bmatrix} 1 \\ \cos(\pi/8) - j\sin(\pi/8) \\ \cos(\pi/4) - j\cos(\pi/4) \\ \sin(\pi/8) - j\cos(\pi/8) \end{bmatrix} (x+jy) \quad (14)$$

위 식은 다중화기로 제어하면 세 종류의 고정계수 곱셈기로 구현할 수 있다. 표 2에 고정계수 곱셈 $\cos(\pi/8)$, $\sin(\pi/8)$,$\cos(\pi/4)$에 대한 CSD 표현과 비트패턴의 빈도를 나타내었다. 우선, 비트패턴 $10\bar{1}$와 101의 빈도를 계산한다. 비트패턴 $10\bar{1}101$일 경우, $10\bar{1}$이 1번 나타나는 것으로 간주한다. 표 2에서 비트패턴 $10\bar{1}$과 101은 각각 4번과 3번씩 나타난다. $10\bar{1}$을 공유할 경우 101 패턴은 $\cos(\pi/8)$에서 1번만 나타나므로 101 패턴은 공유되지 못한다. 따라서, 세 개의 고정계수 곱셈은 다음과 같이 설계할 수 있다.

$$\begin{aligned} x_1 &= x - x \gg 2 \\ x\cos(\pi/8) &= x - x \gg 4 - x \gg 6 + x \gg 9 \\ x\sin(\pi/8) &= x_1 \gg 1 + x_1 \gg 7 \\ x\cos(\pi/4) &= x_1 - x_1 \gg 4 + x \gg 8 \end{aligned} \quad (15)$$

ITM 알고리즘을 적용하면 $\cos(\pi/8)$와 $\cos(\pi/4)$에서 $1000\bar{1}$ 패턴이 공통으로 나타나고, 계수 $\sin(\pi/8)$에서 $10\bar{1}$ 패턴을 공유할 수 있다. 다음 식은 ITM 알고리즘에 의한 설계이다.

$$x_1 = x - x \gg 2, \quad x_2 = x - x \gg 4 \qquad (16)$$
$$x\cos(\pi/8) = x_2 - x \gg 6 + x \gg 9$$
$$x\sin(\pi/8) = x_1 \gg 1 + x_1 \gg 7$$
$$x\cos(\pi/4) = x_2 - x \gg 2 + x \gg 6 + x \gg 8$$

NR-SCSE 알고리즘을 적용하면 식 (15)와 동일하지만, 제안한 전용덧셈기를 사용하면 저면적이면서 고속으로 동작하도록 설계할 수 있다.

제안한 방법을 검증하기 위해 radix-24 FFT에 사용되는 고정계수 곱셈기를 Verilog를 이용하여 구현하고, 삼성 0.35μm 공정의 standard cell library를 사용하여 Synopsys 툴로 합성하였다. 표 3에 면적, 지연시간, 전력소모에 대한 결과를 비교하였다. 제안한 방법이 ITM 방법보다 면적, 지연시간, 전력소모가 각각 21%, 11%, 12% 정도 적음을 알 수 있다.

V. 결 론

본 논문에서는 CSD 계수에서 빈번히 나타나는 비트 패턴 $10\bar{1}(\bar{1}01)$, $101(\bar{1}\bar{1}01)$을 위한 전용덧셈기를 설계하였다. 제안한 방법을 radix-24 FFT 구조의 고정계수 곱셈블록에 적용하여 기존 방법보다 면적, 지연시간, 전력소비 면에서 우수함을 확인하였다.

표 3. 시뮬레이션 결과
Table. 3 Simulation results

	면적(cell)	지연시간(ns)	전력소모(uW)
ITM[3]	564(1)	6.02(1)	570.1(1)
NR_SCSE[4]	517(0.92)	6.39(1.06)	520.0(0.91)
Proposed	448(0.79)	5.38(0.89)	500.4(0.88)

참고문헌

[1] Keshab K. Parhi, *VLSI Digital Signal Processing Systems*. Wiley-Interscience, 1999.

[2] R. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, Oct. 1996.

[3] M. Potkonjak et al., "Multiple constant multiplication: efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 151 - 165, Feb. 1996.

[4] M. Martínez-Periró, et al., "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm", *IEEE Trans. Circuits Syst. II*, vol. 49, Mar. 2002.

[5] W. Yeh and C. Jen, "High-speed Booth encoded parallel multiplier design," *IEEE Trans. Computers*, vol. 49, pp. 692-701, July 2000.

[6] J. Y. Oh and M. S. Lim, "New radix-2 to the 4th power pipeline FFT processor", *IEICE Trans. Electron.*, vol. E88-C, no. 8, pp. 1740-1746, Aug. 2005.

저자소개

조경주(Kyung-Ju Cho)

2000년 2월: 원광대학교 전자공학과 (공학사)
2000년 2월: 전북대학교 정보통신학과(공학석사)
2006년 8월: 전북대학교 정보통신공학과(공학박사)
2006년 8월 ~ 현재: BK21-전북 전자정보 고급인력양성사업단 Post-Doc
※관심분야 : VLSI 신호처리, OFDM, UWB 통신

김용은(Yong-Eun Kim)

2005년 2월: 전북대학교 전자공학과 (공학사)
2002년 2월: 전북대학교 정보통신공학과(공학석사)
2007 ~ 현재: 전북대학교 정보통신공학과 박사과정
※관심분야 : VLSI설계, 신호처리, 반도체