논문 2008-45SD-9-10

Common Subexpression Elimination 회로의 부호 확장 제거

(Sign-Extension Reduction Method in Common Subexpression Elimination Circuit)

김 용 은*, 정 진 균**, 이 문 호**

(Yong-Eun Kim, Jin-Gyun Chung, and Moon-Ho Lee)

요 약

FIR 필터에서 곱셈기는 대부분의 면적을 차지한다. FIR 필터의 설계시 개별적인 곱셈기 대신 Common Subexpression Elimination(CSE) 알고리즘을 이용하여 덧셈만으로 곱셈기를 구현할 수 있다. CSE방식은 곱셈을 이용하지 않기 때문에 보다 작은 면적으로 필터를 구현할 수 있으나 덧셈에서 발생하는 캐리의 긴 전파 시간으로 인하여 필터 연산시간이 길어지는 단점이 있다. 특히 더해지는 항의 쉬프트가 클수록 부호 확장이 많아지며 부호확장에 의해 덧셈의 면적이 커지고 계산 시간이 길어진다. 본 논문에서는 CSE 알고리즘에서 부호 확장 부분을 제거하는 방법을 제안하며 제안한 알고리즘을 이용하여 주어진 예제를 삼성 0.35μ 공정으로 설계하였을 때 기존 설계 방법 보다 면적, 속도, 파워소모에서 각각 17%, 31%, 12% 의 이득이 있음을 보인다.

Abstract

In FIR filter design, multipliers occupy most of the area. To efficiently reduce the area occupied by multipliers, Common Subexpression Elimination (CSE) algorithm can be used instead of separate multipliers. However, the filter computation time can be increased due to the long carry propagation in CSE circuits. More specifically, when the difference of weights between the two inputs to an adder in CSE circuits is large, long carry propagation time is required due to large sign extension. In this paper, we propose a sign-extension reduction method in common subexpression elimination circuit. By Synopsys simulation using Samsung 0.35um library, it is shown that the proposed method leads to 17%, 31% and 12% reduction in the area, time delay and power consumption, respectively, compared with conventional method.

Keywords: Multiplierless Filter, Common subexpression elimination, Sign extension

I. 서 론

FIR 필터를 설계할 때 파워와 면적을 줄이기 위해 곱셈기를 사용하지 않고 덧셈기만을 이용하여 곱셈기를 설계하는 경우 Common Subexpression Elimination (CSE) 알고리즘을 사용할 수 있다[1]. 덧셈기만을 이용하는 경우 더해질 항들을 공유하여 최소한의 덧셈기를 이용하는 방법들이 연구되었다[1~4]. 그러나 CSE 알고리즘을 이용하는 경우 지연시간이 긴 단점이 있다. 또한 더해질 항들이 쉬프트 될 때 부호확장이 발생하여 곱셈기의 성능을 저하 시킨다.

본 논문에서는 CSE 알고리즘을 이용하여 곱셈기를 설계하는 경우 쉬프트로 인하여 발생한 부호확장을 제거 시키는 방법에 대해서 제안하고 제안한 방법을 이용하였을 때 면적, 속도, 파워소모에서 이득이 있음을 보인다.

Ⅱ장에서는 기존 CSE 알고리즘을 이용한 FIR 필터 설계 방법, Ⅲ장에서는 제안한 방법을 이용하여 부호확장을 제거 시키는 방법, Ⅳ장에서는 기존의 방식과 제

* 학생회원, ** 정회원, 전북대학교 전자정보공학부
(Div. of Electronic & Information Engineering Chonbuk National University)

※ 이 연구에 참여한 연구자는 2단계 BK21사업의 지원비를 받았음, This work was supported by the second stage of Brain Korea 21 Project.
접수일자: 2008년7월14일, 수정완료일: 2008년8월28일

안한 방식으로 FIR 필터를 설계하여 시뮬레이션한 결과에 대해서 설명하고, V장에서 결론을 맺는다.

II. CSE 곱셈기에서 부분항 덧셈 방법

FIR 필터를 설계할 때 대표적으로 사용되는 CSE 알고리즘은 Hartley 알고리즘이다[2, 4]. 또한 가장 적은 수의 덧셈을 사용하여 필터를 설계할 때 Bull-Horrocks modified(BHM) 알고리즘을 이용한다[5~6]. 하지만 BHM은 지연시간이 가장 긴 CSE 알고리즘이다. 어떠한 CSE 알고리즘을 이용한다 할지라도 쉬프트로 인한 부호확장은 반드시 존재한다. 예를 들어 필터의 계수가 16bit로 고정되어 있고 $h_0 = 155, h_1 = 109, h_2 = 93, h_2 = 98$일 때 식 (1)과 같은 필터가 존재한다고 가정하면 이를 BHM알고리즘을 이용하여 공통된 항을 묶으면 그림 1과 같은 구조로 나타낼 수 있다[7].

$$y(n) = h_0x(n) + h_1x(n-1) + h_2x(n-2) + h_3x(n-3) \quad (1)$$

그림 1의 내부 구조가 모두 12비트로 고정되어 있다고 가정할 때 쉬프트로 인한 부호확장을 고려한 그림 1의 포인트 P1, P2, P3의 계산 과정은 표 1과 같다. 표 1에서 P2, P3는 쉬프트 후 덧셈을 수행 할 때 부호 확장이 각각 5, 7개 씩 발생한다. 덧셈이 리플캐리 덧셈기로 수행된다면 덧셈이 수행되는 변수의 비트 수 만큼 전가산기가 사용된다. 그림 2는 전가산기 구조와 반가산기 구조를 나타내고 있다. 그림 2-(a)의 전가산기는 캐리아웃이 발생할 때 게이트 3개를 통과하여야 하며 총 6개의 게이트로 구성된다. 반면 그림 2-(b)의 반가산기는 캐리아웃이 발생할 때 게이트 1개만 거치면 되고 2

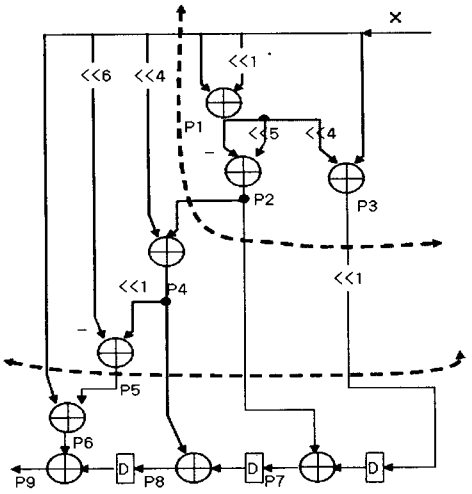


그림 1. BHM 알고리즘을 이용한 FIR 필터 구조.
Fig. 1. FIR filter structure using BHM algorithm.

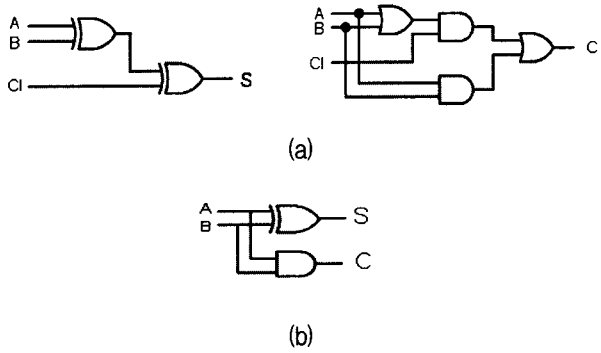


그림 2. 전가산기와 반가산기 구조:
(a) 전가산기, (b) 반가산기
Fig. 2. Full adder and half adder structures:
(a) FA, (b) HA.

개의 게이트만 필요하다. 만약 표 1에서 부호 확장이 사라지게 되면 부호 확장이 사라진 부분이 전가산기에서 반가산기로 대체되어 면적이 작아지고 속도가 빨라지게 된다.

표 1. 그림 1의 P1, P2, P3의 연산과정.
Table 1. Operation process of P1, P2 and P3 in fig. 1.

부분항													
P1		x[11]	x[11]	x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]
	+	x[11]	x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]
		p1[11]	p1[10]	p1[9]	p1[8]	p1[7]	p1[6]	p1[5]	p1[4]	p1[3]	p1[2]	p1[1]	p1[0]
P2		~p1[11]	~p1[11]	~p1[11]	~p1[11]	~p1[11]	~p1[11]	~p1[10]	~p1[9]	~p1[8]	~p1[7]	~p1[6]	~p1[5]
	+	p1[11]	p1[10]	p1[9]	p1[8]	p1[7]	p1[6]	p1[5]	p1[4]	p1[3]	p1[2]	p1[1]	p1[0]
		p2[11]	p2[10]	p2[9]	p2[8]	p2[7]	p2[6]	p2[5]	p2[4]	p2[3]	p2[2]	p2[1]	p2[0]
P3		p1[11]	p1[11]	p1[10]	p1[9]	p1[8]	p1[7]	p1[6]	p1[5]	p1[4]	p1[3]	p1[2]	p1[1]
	+	x[11]	x[11]	x[11]	x[11]	x[11]	x[11]	x[11]	x[11]	x[10]	x[9]	x[8]	x[7]
		p3[11]	p3[10]	p3[9]	p3[8]	p3[7]	p3[6]	p3[5]	p3[4]	p3[3]	p3[2]	p3[1]	p3[0]

표 2. 보상벡터를 이용한 표 1의 표현
Table 2. The expression of table 1 using compensation vectors.

부분항													
P1	+	1	1	~x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]
		1	~x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]
			1	1									
		p1[11]	p1[10]	p1[9]	p1[8]	p1[7]	p1[6]	p1[5]	p1[4]	p1[3]	p1[2]	p1[1]	p1[0]
P2	+	1	1	1	1	1	p1[11]	~p1[10]	~p1[9]	~p1[8]	~p1[7]	~p1[6]	~p1[5]
		~p1[11]	p1[10]	p1[9]	p1[8]	p1[7]	p1[6]	p1[5]	p1[4]	p1[3]	p1[2]	p1[1]	p1[0]
		1					1						
		p2[11]	p2[10]	p2[9]	p2[8]	p2[7]	p2[6]	p2[5]	p2[4]	p2[3]	p2[2]	p2[1]	p2[0]
P3	+	1	~p1[11]	p1[10]	p1[9]	p1[8]	p1[7]	p1[6]	p1[5]	p1[4]	p1[3]	p1[2]	p1[1]
		1	1	1	1	1	1	1	~x[11]	x[10]	x[9]	x[8]	x[7]
			1						1				
		p3[11]	p3[10]	p3[9]	p3[8]	p3[7]	p3[6]	p3[5]	p3[4]	p3[3]	p3[2]	p3[1]	p3[0]

표 3. 표 2의 보상 벡터를 모두 제거 시키고 계산한 P'
Table 3. Computed P' after removing all compensation vectors.

부분항													
P1′	+		~x[11]	~x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]
				x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]
		p1[11]′	p1[10]′	p1[9]′	p1[8]′	p1[7]′	p1[6]′	p1[5]′	p1[4]′	p1[3]′	p1[2]′	p1[1]′	p1[0]′
P2′	+						p1[11]′	~p1[10]′	~p1[9]′	~p1[8]′	~p1[7]′	~p1[6]′	~p1[5]′
		~p1[11]′	p1[10]′	p1[9]′	p1[8]′	p1[7]′	p1[6]′	p1[5]′	p1[4]′	p1[3]′	p1[2]′	p1[1]′	p1[0]′
		p2[11]′	p2[10]′	p2[9]′	p2[8]′	p2[7]′	p2[6]′	p2[5]′	p2[4]′	p2[3]′	p2[2]′	p2[1]′	p2[0]′
P3′	+		~p1[11]′	p1[10]′	p1[9]′	p1[8]′	p1[7]′	p1[6]′	p1[5]′	p1[4]′	p1[3]′	p1[2]′	p1[1]′
									~x[11]	x[10]	x[9]	x[8]	x[7]
		p3[11]′	p3[10]′	p3[9]′	p3[8]′	p3[7]′	p3[6]′	p3[5]′	p3[4]′	p3[3]′	p3[2]′	p3[1]′	p3[0]′

곱셈기를 설계할 때 부호 확장을 보상벡터로 변환하고 보상벡터들을 더하여 최종보상벡터를 생성하고 이를 마지막 출력에 더함으로서 부호 확장을 제거 시키는 방법[8]이 존재하나 이러한 방법을 CSE에 적용시키지 못한다. 그 이유는 각 P의 연산이 완전히 이루어져야 다음 P 위치에서 일반적인 보상벡터 공식이 적용되기 때문이다. 예를 들어 표 2는 P1, P2, P3의 부호확장을 보상벡터로 변환한 표이고 표 3과 같이 표 2의 보상벡터들을 제거하고 더한 값을 P'이라하면 출력 P'에 임의의 상수(최종 보상벡터)를 더해주면 기존의 출력 P를 다시 복원할 수 있어야 된다. 하지만 표 3의 P2'를 계산할 때 보상벡터를 더하지 않은 값을 P1'을 더하여 P2'를 생성하고 최종 보상 벡터를 더하여도 원래 표 1의 P2 값을 복원할 수 없다.

III. 제안한 부호확장 제거 방법

II 장에서 일반적인 보상벡터 변환 방법으로 부호확장을 제거할 수 없음을 확인하였다. 이러한 문제를 해결하기 위해 상수 곱셈기에서 사용되는 보상벡터 변환 방식을 고려해 볼 때 상수곱셈의 경우 II 장에서와 같이 부호확장을 제거하고 최종 보상 벡터를 이용하여 더하여도 출력 값이 정확하다. 그 이유는 부분 곱의 부호확장이 변수 하나에 대해서 전개되기 때문이다. 예를 들어 식(2)과 같은 상수곱셈 부분곱이 존재할 때 식(2)의 부호확장을 보상벡터로 표시하게 되면 식(3)와 같이 변화된다.

$$\begin{array}{l} x[11]x[11]x[11]x[11]x[10]x[9]\cdots x[0] \\ x[11]x[11]x[11]x[10]x[9]\cdots x[0] \\ x[11]x[10]x[9]\cdots x[0] \end{array} \quad (2)$$

표 4. 표 1의 P2의 연산과정을 변수 x로 표시
Table 4. Expression with variable x of operation process of P2 in table 1.

부분항													
P2	+	~x[11]	~x[11]	~x[11]	~x[11]	~x[11]	~x[11]	~x[11]	~x[10]	~x[9]	~x[8]	~x[7]	~x[5]
		~x[11]	~x[11]	~x[11]	~x[11]	~x[11]	~x[11]	~x[10]	~x[9]	~x[8]	~x[7]	~x[6]	~x[5]
		x[11]	x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]
		x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]	x[0]
		p2[11]	p2[10]	p2[9]	p2[8]	p2[7]	p2[6]	p2[5]	p2[4]	p2[3]	p2[2]	p2[1]	p2[0]
P2	+	1	1	1	1	1	1	x[11]	~x[10]	~x[9]	~x[8]	~x[7]	~x[5]
		1	1	1	1	1	x[11]	~x[10]	~x[9]	~x[8]	~x[7]	~x[6]	~x[5]
		1	~x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]
		~x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]	x[0]
		1	1				1	1					
		p2[11]	p2[10]	p2[9]	p2[8]	p2[7]	p2[6]	p2[5]	p2[4]	p2[3]	p2[2]	p2[1]	p2[0]

표 5. 표 2의 부호 확장을 제거하고 정확히 계산한 P'
Table 5. precisely computed P' after removing of sign extension of table 2.

부분항													
P1′	+			~x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]
			~x[11]	x[10]	x[9]	x[8]	x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]
		p1[11]′	p1[10]′	p1[9]′	p1[8]′	p1[7]′	p1[6]′	p1[5]′	p1[4]′	p1[3]′	p1[2]′	p1[1]′	p1[0]′
P2′	+						~p1[11]′	~p1[10]′	~p1[9]′	~p1[8]′	~p1[7]′	~p1[6]′	~p1[5]′
		p1[11]′	p1[10]′	p1[9]′	p1[8]′	p1[7]′	p1[6]′	p1[5]′	p1[4]′	p1[3]′	p1[2]′	p1[1]′	p1[0]′
		p2[11]′	p2[10]′	p2[9]′	p2[8]′	p2[7]′	p2[6]′	p2[5]′	p2[4]′	p2[3]′	p2[2]′	p2[1]′	p2[0]′
P3′	+		p1[11]′	p1[10]′	p1[9]′	p1[8]′	p1[7]′	p1[6]′	p1[5]′	p1[4]′	p1[3]′	p1[2]′	p1[1]′
									~x[11]	x[10]	x[9]	x[8]	x[7]
		p3[11]′	p3[10]′	p3[9]′	p3[8]′	p3[7]′	p3[6]′	p3[5]′	p3[4]′	p3[3]′	p3[2]′	p3[1]′	p3[0]′

$$\begin{array}{l} 1\ \ 1\ \ 1\ {\sim}x[11]x[10]x[9]\cdots x[0] \\ 1\ \ 1\ {\sim}x[11]x[10]x[9]\cdots x[0] \\ {\sim}x[11]x[10]x[9]\cdots x[0] \\ 1\quad 1\quad 1 \end{array} \tag{3}$$

식(3)의 보상벡터를 최종 보상 벡터로 정리하면 식(4)와 같이 나타 낼 수 있다.

$$\begin{array}{l} \quad {\sim}x[11]x[10]x[9]\cdots x[0] \\ {\sim}x[11]x[10]x[9]\cdots x[0] \\ {\sim}x[11]x[10]x[9]\cdots x[0] \\ 1\qquad 1 \end{array} \tag{4}$$

따라서 Ⅱ 장과 같이 어떠한 값이 연산된 후 출력된 P위치의 부호확장을 보상벡터를 변환하지 않고 변수 x 한개에 대해서 보상벡터를 변환한 후 최종보상벡터를 마지막 값에 더해주면 계산결과 값을 정확히 연산할 수 있음을 알 수 있다.

표 1에서 P2의 연산과정을 P1의 덧셈으로 나타내지 않고 초기에 주어진 변수 즉 x로 표현하고 이를 보상벡터로 변환 시키면 표 4와 같다.

표 2에서 x와 보상 벡터로 나타낸 부분항들에서 x의 MSB 즉 $x[11]$의 값만 반전되어 있음을 알 수 있다. 이렇게 부호확장을 위해서 초기 입력 x의 MSB만 반전되는 구조가 부분항이 전개되는 과정에서 유지 되어야 함을 알 수 있다. 따라서 표 4에서 부호확장을 제거하여 P1′를 이용하여 P2′를 계산할 때 P1′의 MSB 즉 P1[11]′를 반전 시켰기 때문에 표 4와 같이 $x[11]$의 값만 반전되는 구조가 변형되어 값이 틀리다. 따라서 $x[11]$만 반전되는 구조를 유지하며 부호확장을 제거시키면 부호 확장을 제거 시킬 수 있다. 표 5는 표 4를 수정한 덧셈 과정이다. 표 5에서 P1′을 계산하기 위해 보상벡터를 제거시킨 x의 MSB $x[11]$을 반전시키고 P2′를 구하기 위해 보상벡터를 제거시킨 P1′의 MSB $P1^{[11]}$′은 반전하지 시키지 않아야 된다. 또한 P3′과 같이 x와 P1′과 덧셈이 이루어 질 때 x의 MSB만 반전되는 구조를 유지하기 위해서 $x[11]$은 반전시키고 P1[11]′반전시키지 않아야 된다. 이렇게 보상벡터를

제거시킬 수 있는 구조를 유지하며 부호 확장을 제거하기 위한 방법을 정리하면 다음과 같다.

곱셈계수 그룹화 알고리즘

1. 입력 x와 입력 x의 연산 수행 시 x의 MSB를 반전시킨다.
2. 입력 x가 연산된 항 P와 P 연산 수행 시 P의 MSB를 반전시키지 않는다.
3. 입력 x와 입력 x가 연산된 P의 연산 수행 시 x의 MSB는 반전시키고 P의 MSB는 반전시키지 않는다.

부호 확장이 제거된 부분은 그림 2의 반가산기를 이용하여 계산할 수 있으며 면적이 작아지고 속도가 빨라진다. 하지만 그림 3과 같이 P9′에서 최종 보상벡터를 더해 주어야 된다. P9′에서 P6, P8, 최종 보상 벡터 3줄이 더해져야 된다. 이때 그림 4-(a)와 같이 Wallce_tree[9] 방식을 이용하여 더하면 효율적으로 3줄 덧셈을 수행할 수 있다. 또한 P9′에서 더해질 때 보상 벡터는 1또는 0이므로 그림 4-(a)에서 부분항을 3줄에서 2줄로 만들 때 전가산기를 사용하지 않고 그림 4와 (b),(c)와 같이 더 작은 유닛으로 계산 될 수 있다[10]. 그림 3에서 최종 보상 벡터를 용이하게 계산하기 위해서 임의의 x 값을 입력하여 계산 결과와 그림 1의 x에 동일한 입력을 인가하여 계산된 결과를 서로 빼면 그림 3의 P9′에 더해지는 최종 보상 벡터 값을 얻을 수 있다.

최종 보상벡터를 덧셈을 위한 그림 4-(b), (c)회로의 오버헤드가 존재함에도 불구하고 부호 확장을 제거하여 전가산기에서 반가산기로 대치하게 되면 캐리전파시간

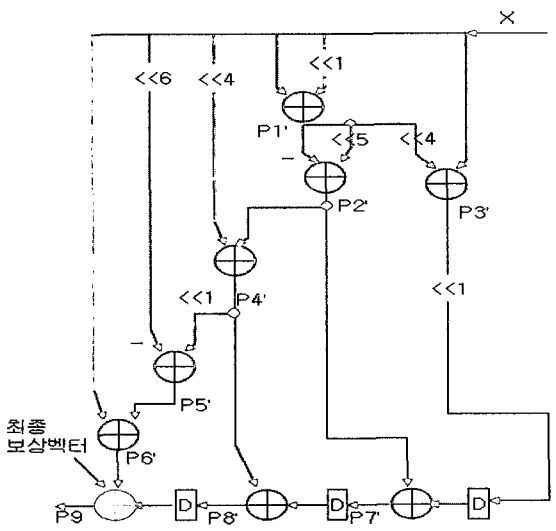


그림 3 그림 1에서 부호 확장을 제거한 구조

Fig. 3. Structure that sign extension is removed in fig. 1.

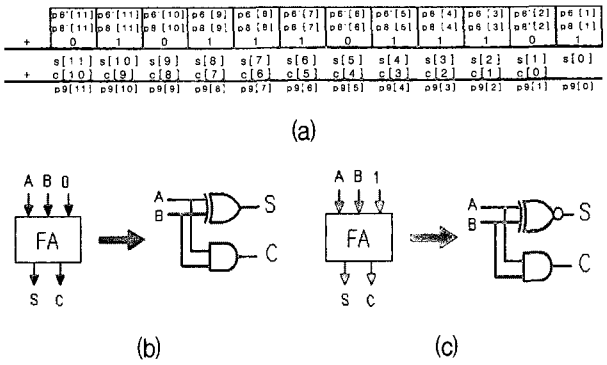


그림 4. 그림 3에서 P9′와 보상벡터가 더해지는 과정과 덧셈기 구조: (a)Wallce Tree방식을 이용한 덧셈과정, (b)보상벡터의 임의의 비트가 0일 때 덧셈기 구조, (c) 보상벡터의 임의의 비트가 1일 때 덧셈기 구조.

Fig. 4. added process of P9′ and compensation vector in fig. 3 and adder structure: (a) addition process using Wallace Tree method, (b) adder structure when some bit of compensation vector is 0, (c) adder structure when some bit of compensation vector is 1

을 눈에 띄게 감소시킬 수 있고 하드웨어면적도 작아 효율적인 회로 설계가 가능하다.

Ⅳ. 시뮬레이션 및 비교

논문에서 예제로 사용한 그림 1을 일반적인 CSE 설계 방법과 제안한 방법을 이용하여 설계하고 삼성 0.35um 라이브러리를 이용하여 Synopsys로 합성한 결과를 표 6에 보였다. 기존의 방법에 비해 면적, 속도, 파워소모에서 각각 17%, 31%, 12% 이득이 있음을 보인다.

표 6. 논문에서 사용된 FIR 필터 예제를 제안한 방법과 기존의 방법으로 설계하였을 때 결과.

Table 6. Design result of FIR filter exercise used in this paper using proposed and conventional method

	기존	제안	이득
면적(cell)	1084	897	17%
지연시간(ns)	12.6	8.7	31%
파워소모(mW)	86.2	76	12%

Ⅴ. 결 론

본 논문에서는 CSE 알고리즘을 이용하여 회로를 설계하는 경우 제안한 방법으로 부호 확장을 제거시킬 수

있음을 보였다. CSE방식의 FIR 필터를 설계 할 때 특히 쉬프트가 많은 경우 제안된 알고리즘을 이용하여 부호 확장을 제거하여 회로를 설계하면 지연시간, 면적, 파워소모를 크게 줄일 수 있을 것이다.

참 고 문 헌

[1] M. Potkonjak et al., "Multiple constant multiplication: efficient and versatile freamework and algorithms for exploring common subexpression elimination," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 2, pp. 151-165, Feb. 1996.

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

[3] M. Mehendale, S. D. Sherlekar, and G. Vekantesh, "Systhesis of multiplierless FIR filters with minimum number of additions, " *in Proc. 1995, IEEE/ACM Int. Conf. Computer-Aided Design*, Los Alamitos, CA, pp. 668-671. 1995.

[4] Marcos Martinez-Peiro, Eduardo I. Boemo, and Lars Wanhammar, "Design of high-speed multiplierless filter using a nonrecursive signed common subexpression algorithm, " *IEEE Transactions on Circuits and Systems II*, vol. 49, pp. 196-203. 2002.

[5] A. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders, " *Proc. Inst. Elec. Eng. Circuits and Systems*, vol. 141, no. 5, pp.407-413, Oct. 1994.

[6] A. Dempster and M. D. Macleod, "Use of minimum adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II*, vol. 42, pp569-557, Sept. 1995.

[7] Marcos Martínez-Peiró, Eduardo I. Boemo, and Lars Wanhammar, "Design of High-Speed Multiplierless Filters Using a Nonrecursive Signed Common Subexpression Algorithm" *IEEE Transactions on Circuits and Systems II*, vol. 49, pp. 198. 2002.

[8] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, NJ: Prentice-Hall International, Inc. 1993.

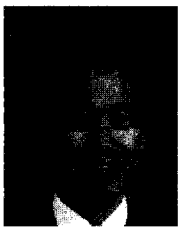
[9] C. S. Wallace, "Suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, pp. 14-17, 1964.

[10] Sang-Min Kim, Jin-Gyun Chung, and Keshab K. Parhi, "Low error fixed-width CSD multiplier with efficient sign extension", *IEEE Transactions on Circuit and System-II*, vol.50, no.12, pp.984-993 dec. 2003.

저 자 소 개

김 용 은(학생회원)
2005년 전북대학교 전자공학과 학사 졸업
2007년 전북대학교 정보통신 공학과 석사 졸업
2007년~현재 전북대학교 전자 정보공학부 박사
<주관심분야 : VLSI설계, 신호처리, 반도체>

정 진 균(정회원)
1985년 전북대학교 전자공학 학사 졸업
1989년 미국 미네소타 주립대학 전기공학 석사 졸업
1991년 미국 미네소타 주립대학 전기공학 박사 졸업
1995년~현재 전북대학교 전자정보공학부 교수
<주관심분야 : 통신, 컴퓨터, 신호처리, 반도체>



이 문 호(정회원)
1990년 동경대학 박사 졸업
1980년 3월~현재 전북대학교 전자정보공학부 교수
1985년 미네소타 주립대 전기과 포스트닥터
1983년 통신 기술사
<주관심분야 : 채널코딩, 이동영상통신>