논문 2004-41SP-5-30

Arithmetic unit를 사용한 저전력 MPEG audio 필터 구현

(Low-power MPEG audio filter implementation using Arithmetic Unit)

장 영 범*, 이 원 상**

(Young-Beom Jang and Won-Sang Lee)

요 약

이 논문에서는 MPEG audio 알고리즘의 필터뱅크를 덧셈을 사용하여 저전력으로 구현할 수 있는 구조를 제안하였다. 제안된 구조는 CSD(Canonic Signed Digit) 형의 계수를 사용하며, 입력신호 샘플을 최대로 공유함으로서 사용되는 덧셈기의 수를 최소화하였다. 제안된 구조는 알고리즘에서 사용된 공통입력 공유, 선형위상 대칭 필터계수를 이용한 공유, 공통입력을 이용한 블록 공유, CSD 형의 계수와 공통패턴 공유를 통하여 사용되는 덧셈의 수를 최소화할 수 있음을 보였다. Verilog-HDL 코딩을 통하여 시뮬레이션을 수행한 결과, 제안된 구조는 기존의 곱셈기 구조의 구현면적과 비교하여 60.3%를 감소시킬 수 있음을 보였다. 또한 제안된 구조의 전력소모는 곱셈기 구조와 비교하여 93.9%를 감소시킬 수 있음을 보였다. 따라서 고속의 곱셈기가 내장된 DSP 프로세서를 사용하지 않고도, Arithmetic Unit나 마이크로 프로세서를 사용하여 효과적으로 MPEG audio 필터뱅크를 구현할 수 있음을 보였다.

Abstract

In this paper, a low-power structure for 512 tap FIR filter in MPEG audio algorithm is proposed. By using CSD(Canonic Signed Digit) form filter coefficients and maximum sharing of input signal sample, it is shown that the number of adders of proposed structure can be minimized. To minimize the number of adders, the proposed structure utilizes the 4 steps of sharing, i.e., common input sharing, linear phase symmetric filter coefficient sharing, block sharing for common input, and common sub-expression sharing. Through Verilog-HDL coding, it is shown that reduction rates in the implementation area and relative power consumption of the proposed structure are 60.3% and 93.9% respectively, comparison to those of the conventional multiplier structure.

Keywords: arithmetic unit, common sub-expression, MPEG audio, CSD form

I. 서 론

최근 인터넷의 발달로 오디오 파일의 인터넷 다운로드가 쉬어지면서 MP3 플레이어의 시장이 매우 빠르게 확대되고 있다. MP3는 MPEG1 audio layer III의 알고리즘을 사용하여 오디오를 부호화 또는 복호화한다. 대부분의 MP3 플레이어는 오디오 복호화만을 수행하며, 주로 general purpose DSP를 사용하여 알고리즘을 코딩한 후에 ASIC으로 IC를 제작하는 것이 일반적인 방법이다. 그러나 최근에는 DSP 프로세서를 사용하지 않고 hard wired 방식으로 구현하는 반도체 IC 제품이 출현하고 있다. 본 논문에서는 MP3 복호기 필터를 AU(Arithmetic Unit)를 사용하여 저전력으로 구현하는 구조를 제안한다. MP3라고 불리는 MPEG audio 알고리즘은 SSB(Single Side Band) 필터뱅크 알고리즘을 사용하며, 구현방식은 그림 1과 같은 WOA(Weighted Overlap-Add) 구조를 사용한다.[1] 즉, IMDCT(Inverse Modified DCT) 블록을 통하여 32개의 MDCT 계수가 64개의 시간영역 신호로 변환되며 다시 8개로 복사하여 512개의 신호가 만들어진다. 이 512개의 신호는 512탭의 필터계수와 windowing되어서 출력용 레지스터에 보관된다. 마지막으로 512개의 출력용 레지스터에서 32개

* 정회원, 상명대학교 공과대학 정보통신공학과
(Dept. of Information and Telecommunication Sangmyung University)
** 정회원, 상명대학교 대학원 컴퓨터정보통신공학과
(Dept. of Computer, Information, and Telecommuni-cation, Sangmyung University)
접수일자: 2004년6월29일, 수정완료일: 2004년7월26일

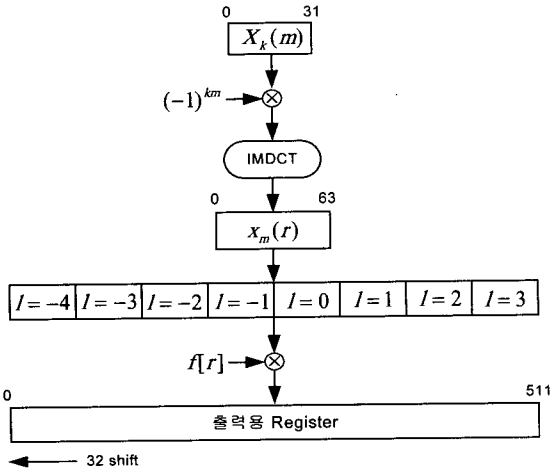

그림 1. MPEG audio 알고리즘의 WOA 구조
Fig. 1. WOA structure for MPEG audio algorithm.

의 샘플이 출력됨으로서 한 블록의 복호가 끝나게 된다. 이 복호기 알고리즘에서 512탭의 필터를 DSP 프로세서를 사용하여 구현할 경우에 512 cycle이 소요된다. 즉, 100MIPS(Mega Instruction Per Second) 속도의 DSP 프로세서는 5.12μs의 계산시간이 필요하다. 이 논문에서는 그림 1에서의 곱셈연산을 덧셈과 쉬프트 연산을 사용하여 구현면적과 전력소모를 감소시키는 필터구조를 제안한다.

그림 1에서 보듯이, 64개 길이의 $x_m[r]$ 샘플은 8개의 블록으로 복사되어 512개 길이가 된다. 역시 512개 길이의 $f[r]$과 샘플 별로 각각 곱해져서 출력용 레지스터에 저장되므로 일반 필터의 곱셈 컨볼류션 연산에 사용되는 덧셈과 쉬프트 연산 방식은 사용할 수 없다. 따라서 본 논문에서는 그림 1의 구조에서 입력을 최대로 공유하는 구조를 먼저 제안함으로서 덧셈과 쉬프트 연산 기법을 효과적으로 적용할 수 있었다.

II. 제안된 고속 CSD 구조

1. 공통입력 공유를 통한 블록 처리

그림 1의 WOA 필터 구조를 살펴보면 64개의 신호 샘플이 8개가 복사되어 512개의 신호 샘플이 만들어 진다. 이 8개의 블록 중에서 짝수 번째의 블록은 그림 1에서와 같이 -의 부호를 붙인다. 이와 같이 만들어진 512개의 신호 샘플은 512탭의 부밴드 필터계수와 window-ing된다. MPEG audio 알고리즘의 구현을 위하여 제공되는 ISO/IEC 표준안에는 512개의 샘플에 붙는 -부호를 필터계수에 붙였다. 따라서 실제 MPEG audio 연산에서는 64개의 신호 샘플을 복사하여 512개의 신호 샘플을 만들면 된다. 이 512개의 신호 x_n과 512탭의 필터계수 h_n이 각각 곱해져서 다음 식과 같이 512개의 출력신호 y_n을 만든다.

$$\begin{aligned} y_0 &= h_0 x_0 \\ y_1 &= h_1 x_1 \\ &\vdots \\ y_{511} &= h_{511} x_{511} \end{aligned} \tag{1}$$

이와 같은 512개의 곱셈을 덧셈기와 쉬프트 연산을 사용하여 효과적으로 구현하기 위해서는 먼저 같은 입력 샘플끼리 모으는 작업이 선행되어야 한다. 덧셈기와 쉬프트 연산을 사용하여 곱셈을 수행하는 경우, 같은 입력 신호에 서로 다른 계수를 곱하도록 하드웨어를 구성하면 CSS(common sub-expression) 공유와 같은 효율적인 방식을 적용할 수 있기 때문이다. 512개의 입력 샘플들은 64개의 IMDCT 출력이 8개 복사되었으므로 다음과 같이 8개씩의 같은 입력 샘플들이 존재한다.

$$\begin{aligned} x_0 &= x_{64} = x_{128} = \cdots = x_{448} \\ x_1 &= x_{65} = x_{129} = \cdots = x_{449} \\ &\vdots \\ x_{63} &= x_{127} = x_{191} = \cdots = x_{511} \end{aligned} \tag{2}$$

따라서 8개의 입력 신호가 같으므로 이 출력 신호들만 모을 수 있다. 예를 들면 x_1의 입력신호가 사용되는 출력신호 만을 모으면 다음과 같다.

$$\begin{aligned} y_1 &= h_1 x_1 \\ y_{65} &= h_{65} x_1 \\ &\vdots \\ y_{449} &= h_{449} x_1 \end{aligned} \tag{3}$$

식(3)의 x_1의 입력을 사용하는 블록을 블록 1이라고 부르기로 하며 그림 2과 같이 입력신호가 공유된다. 그림 2과 같이 입력신호가 공유된 블록이 블록 0부터 블록 63까지 64개의 블록이 생긴다. 이와 같이 같은 입력신호를 사용하는 곱셈을 모아서 하나의 블록을 구성하는 까닭은 덧셈기와 쉬프트를 사용하여 곱셈을 구현할 때에 공통패턴을 공유할 수 있기 때문이다. 이와 같은 64개의 블록을 hard wired 방식으로 구현하기 위해서 64개의 하드웨어 구조가 필요하다. 그림 2과 같이 입력신호가 공유된 블록이 블록 0부터 블록 63까지 64개의 블록이 생긴다. 이와 같은 64개의 블록을 hard wired

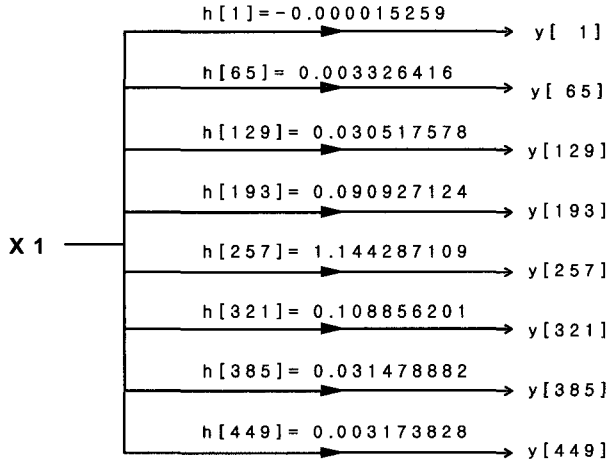

그림 2. 공통입력 공유의 블록 1 구조
Fig. 2. block 1 structure sharing common input.

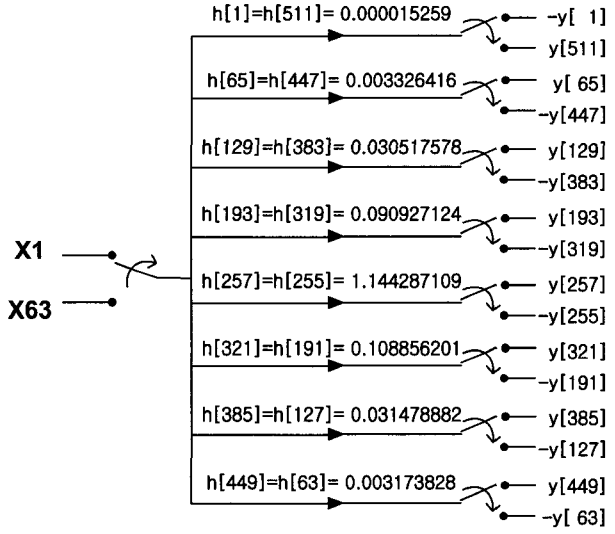

그림 3. 블록 1과 블록 63의 곱셈기를 공유한 블록 1-63 구조
Fig. 3. block 1-63 structure sharing multipliers of block 1 and block 63.

방식으로 구현하기 위해서 64개의 하드웨어 구조가 필요하다.

2. 대칭 필터계수를 이용한 블록 공유 구조

MPEG audio 알고리즘에서는 선형위상의 대칭 필터계수를 사용하므로 64개의 블록들 가운데에 같은 블록들이 2개씩 존재한다. 즉, 블록 1과 블록 63의 필터계수들이 같으므로 구현 하드웨어가 같아지게 된다. 이와 같은 방식으로 블록 2와 블록 62가 같고, 마지막으로 블록 31과 블록 33의 구현 하드웨어가 같음을 알 수 있다. 예외적으로 블록 0과 블록 32는 같은 필터계수가 사용되는 블록이 없으므로 각각 따로 구현하여야 한다. 블록 1과 블록 63을 같은 하드웨어로 구현하는 구조를

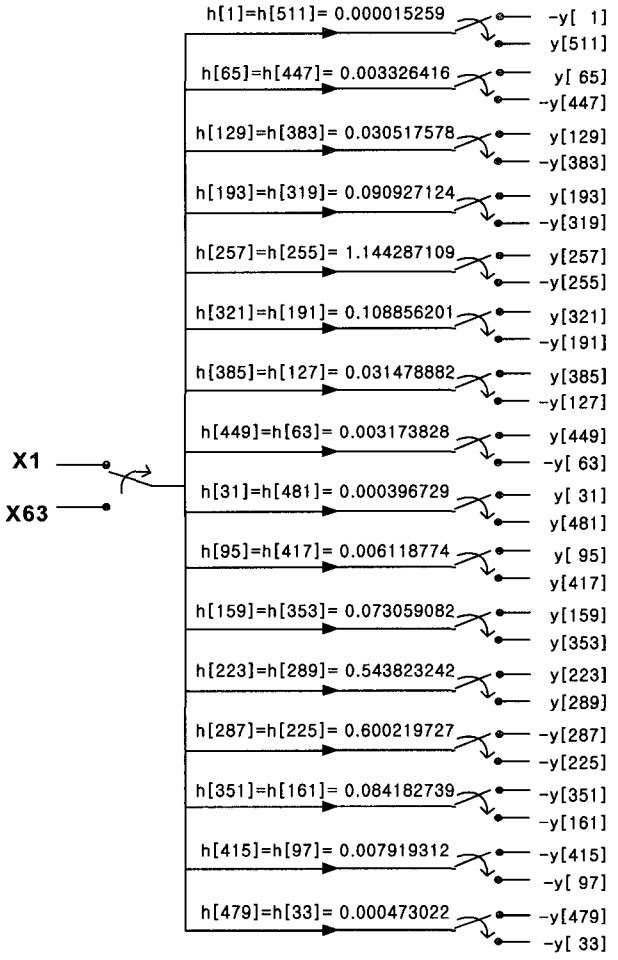

그림 4. 블록 1-63과 블록 31-33의 입력을 공유한 블록 1-63-31-33 구조
Fig. 4. block 1-63-31-33 structure sharing input of block 1-63 and block 31-33.

블록 1-63으로 명명한다. 블록 1-63은 그림 3과 같다. 그림 3에서 곱셈을 수행하는데 2 cycle이 필요하다면 첫 cycle에는 x_1을 입력시켜서 $y_1 \cdots y_{449}$의 8개의 출력을 계산하고, 두 번째 cycle에는 x_{63}을 입력시켜서 $y_{511} \cdots y_{63}$의 8개의 출력을 계산하면 된다. 이와 같이 같은 블록의 공유를 통하여 64개의 하드웨어를 33개로 줄일 수 있다. 즉 2 cycle의 31개 블록과 1 cycle의 2개 블록으로 하드웨어가 줄어들도록 설계하였다.

3. 공통입력을 이용한 블록 공유

필터의 입력으로 사용되는 IMDCT 출력 64개는 다음과 같이 같은 값들이 존재한다.

$$x_0 = -x_{32},\ x_1 = -x_{31}, \cdots,\ x_{15} = -x_{17}$$
$$x_{33} = x_{63},\ x_{34} = x_{62},\ \cdots,\ x_{47} = x_{49} \quad (4)$$

이와 같이 입력신호가 같기 때문에 블록 1-63과 블록 31-33의 입력이 같다. 덧셈기와 쉬프트 연산을 사용한 곱셈의 구현에서는 입력의 공유를 통하여 효과적인 구현 하드웨어의 감소를 이룰 수 있다. 블록 1-63과 블록 31-33의 입력을 공유하면 그림 4와 같다. 최종적으로 그림 4의 블록 1-63-31-33과 같은 제안된 구조는 15개가 만들어진다. 즉, 블록 1-63-31-33부터 블록 15-49-17-47까지 15개의 블록이 만들어진다. 예외적으로, 블록 0과 블록 32가 입력이 같으므로 입력 공유를 통한 블록 0-32를 설계할 수 있으며, 블록 16-48은 같은 입력이 없으므로 입력 공유는 발생하지 않는다.

4. CSD & CSS 구조

이전 절에서는 입력 샘플을 최대로 공유할 수 있는 구조를 만들었으며 이 절에서는 이 구조를 덧셈기와 쉬프트 연산으로 구현하는 방식을 제안한다. 먼저 필터계수들을 2진수로 표현하여야 하며, 2진수로 표현한 후에 1의 수가 필요한 덧셈의 수가 된다. 따라서 2의 보수형의 2진수보다는 1의 수가 적은 CSD(Canonic Signed Digit) 형의 2진수가 유리하다.[2][3] CSD형 계수를 사용하기 위하여 블록 1-63-31-33에서 사용되는 16개의 계수들을 18비트의 CSD 형으로 나타내면 표 1과 같다. CSD형은 1, -1, 0을 사용하는데, 표 1에서 보듯이 0은 표기하지 않았으며 -1은 n으로 표기하였다.

표 1의 A열에는 CSD 형의 2진수를 구현하는데 필요한 덧셈의 수를 나타내었으며 총 51개의 덧셈이 필요하다. 덧셈과 쉬프트 연산으로 곱셈을 구현할 때에, CSS

표 1. 블록 1-63-31-33에서 사용되는 곱셈의 CSD 형
Table 1. CSD form for filter coefficients of block 1-63-31-33.

	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	A	B
h1																		1	0	0
h65										1			n		n		1		3	1
h129							1					n		1					2	1
h193					1		n			n		1			1			n	5	2
h257	1				1			1		1				n					4	2
h321					1			n					n				n		3	2
h385							1							1				n	2	1
h449										1		n		1				n	3	1
h31													1		n		1		2	1
h95									1		n			1					2	1
h159						1		1		n		n		n		1			5	2
h223			1			1		n		n		1			n				5	2
h287			1		1		n		1		n		1		1				6	3
h351					1		n		n		n			1		n		1	6	3
h415									1						1			n	2	1
h479													1				n		1	0
																			51	28

방식이 효과적이라고 알려져 있다.[4][5] 이 방식은 2진수로 표현된 계수에서 공통패턴을 공유하는 방식이다. 따라서 표 1에는 2중 실선으로 표현한 것과 같은 공통패턴들을 묶을 수 있으며 이 공통패턴들을 이용하여 덧셈의 수를 다음과 같이 줄인다. 먼저 공통패턴들은 다음과 같은 식으로 나타낼 수 있다.

$$\begin{aligned} 101 &: s_1 = x_0 + x_0 \gg 2 \\ 10n &: s_2 = x_0 - x_0 \gg 2 \\ 1001 &: s_3 = x_0 + x_0 \gg 3 \\ 100n &: s_4 = x_0 - x_0 \gg 3 \\ 1000n &: s_5 = x_0 - x_0 \gg 4 \end{aligned} \tag{5}$$

식(5)에서 정의된 공통패턴을 사용하여 표 1의 16개의 계수들을 다음과 같은 식으로 나타낼 수 있다.

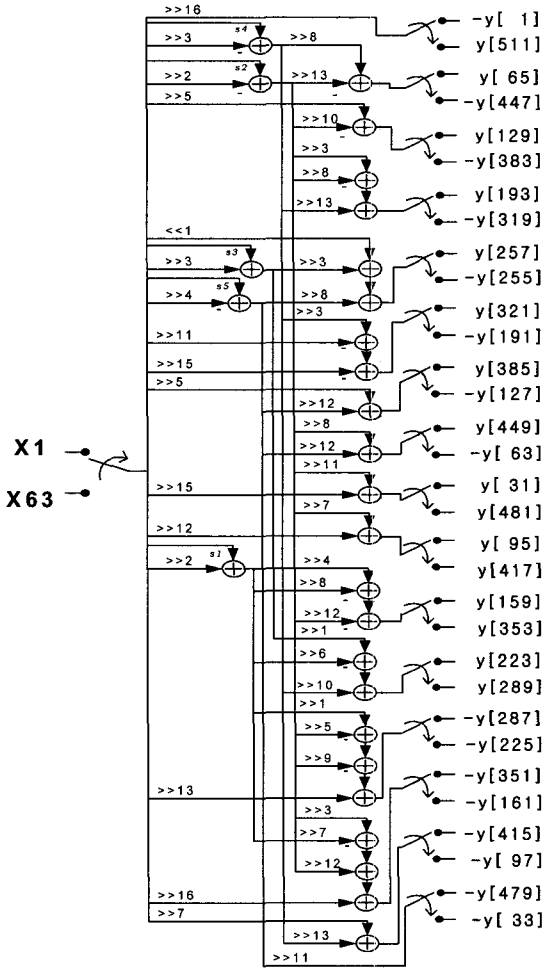

그림 5. 덧셈기와 쉬프트를 사용한 제안된 블록 1-63-31-33 구조
Fig. 5. proposed block 1-63-31-33 structure using adders and shifts.

$$
\begin{aligned}
h_1 &= x_0 \gg 16 \\
h_{65} &= s_4 \gg 8 - s_2 \gg 13 \\
h_{129} &= x_0 \gg 5 - s_2 \gg 10 \\
h_{193} &= s_2 \gg 3 - s_2 \gg 8 + s_4 \gg 13 \\
h_{257} &= s_3 \gg 3 + s_5 \gg 8 \\
&\vdots \\
h_{479} &= s_5 \gg 11
\end{aligned} \tag{6}
$$

이와 같은 공통패턴을 사용하여 구현하면 표 1의 우측 열 B에 표시한 것과 같이 총 28개의 덧셈이 필요하다. 이와 같이 표현된 식(6)을 덧셈기와 쉬프트를 사용하여 구현하면 그림 5와 같다. 이와 같은 네 단계를 MPEG audio 필터를 덧셈기와 쉬프트를 사용하여 효율적으로 설계하는 방법을 제안하였다. 즉, 입력 샘플을 최대로 공유하게 함으로서 공통패턴을 최대한으로 많이 공유할 수 있도록 구성하였다.

Ⅲ. 실 험

1. 사용된 덧셈의 수 비교

MPEG audio 알고리즘은 512탭의 선형위상 FIR 필터를 사용하므로 Ⅱ절에서 제안한 구조가 17개가 사용된다. 따라서 17개의 블록을 설계한 후에 각각의 블록에 사용되는 덧셈기의 수를 비교하면 표 2와 같다.

표 2는 Ⅱ-1, 2, 3을 통하여 만들어진 그림 3의 구조를 3가지 형의 2진수로 구현한 결과이다. 즉 2의 보수형 계수를 사용한 구조와, CSD 형의 계수를 사용한 구조와, CSD 형의 계수와 common sub-expression 공유기법을 사용하는 이 논문이 제안하는 구조의 사용된 덧셈의 수를 비교하였다. 표 2에서 보는 것과 같이 제

표 2. 제안된 구조에서 사용된 총 덧셈의 수
Table 2. Total number of adders in the proposed structure.

블록	2의 보수형 구조	CSD형 구조	제안된 구조
블록 1-63-31-33	73	51	28
블록 2-62-30-34	68	45	27
블록 3-61-29-35	80	49	30
블록 4-60-28-36	71	49	29
블록 5-59-27-37	68	54	30
블록 6-58-26-38	73	49	28
블록 7-57-25-39	80	51	30
블록 8-56-24-40	68	49	29
블록 9-55-23-41	67	45	29
블록 10-54-22-42	69	48	31
블록 11-53-21-43	73	53	30
블록 12-52-20-44	71	51	29
블록 13-51-19-45	77	53	29
블록 14-50-18-46	72	51	30
블록 15-49-17-47	82	51	29
블록 16-48	44	24	17
블록 0-32	43	31	22
Total	1179 (100%)	804 (68.2%)	477 (40.46%)

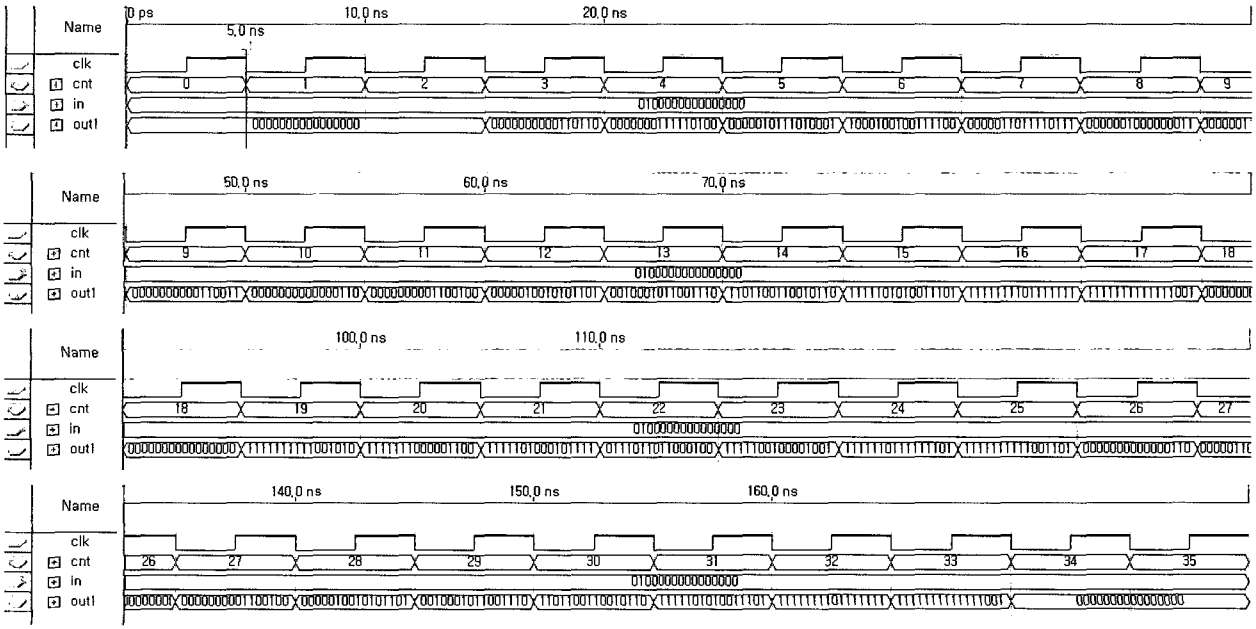

그림 6. 곱셈기를 사용한 블록 1-63-31-33 구조의 Verilog-HDL simulation 결과
Fig. 6. Verilog-HDL simulation result of multiplier based block 1-63-31-33 structure.

안된 구조를 사용한 결과 1179개의 덧셈기를 477개로 줄일 수 있었다.

2. 구현면적과 소모전력 비교

이 절에서는 그림 4의 블록 1-63-31-33 곱셈기 구조와 그림 5의 제안된 구조를 Verilog-HDL로 코딩하여 각각의 구조의 구현 면적과 상대 전력소모를 비교하기로 한다. 사용된 tool은 Altera사의 Quartus II 4.0 Web Edition을 이용하여 Verilog-HDL coding을 수행하였다. 그림 6는 그림 4의 곱셈기 구조에 대한 Verilog-HDL 코딩 결과를 나타냈다. 그림 6에서 cnt2부터 cnt33의 2진수 값들은 그림 4의 출력용 레지스터의 값들을 나타내고 있다. 이 Verilog-HDL 코딩에서 그림 4의 곱셈기 구조는 1개의 곱셈기를 설계하여 32번의 곱셈 연산을 수행하였다. 그림 7은 그림 5의 제안된 구조에 대한 Verilog-HDL 코딩 결과를 나타냈다. 그림 7에서 cnt7부터 cnt55의 2진수 값들이 그림 6의 출력용 레지스터 값들과 같음을 확인할 수 있다. 그림 7의 코딩에서도 역시 1개의 덧셈기를 설계하여 28번의 덧셈 연산을 반복 수행하였다. 표 2에서 보듯이 블록 16-48과 블록 0-32를 제외하면 각 블록은 27개에서 31개의 덧셈을 사용하고 있으므로 연산의 양이 균일하게 분산되어 있다. 따라서 각각의 블록마다 한 개의 덧셈기를 사용하여 구현하면 연산시간은 길어지나, 하드웨어의 크기를 감소시킬 수 있다. 이와 같은 Verilog-HDL 코딩의 결과인 logic cell의 수는 표 3과 같다.

표 3. 제안된 구조의 구현면적 비교
Table 3. Implementation area for proposed structure.

	제안 구조	곱셈기 구조
사용된 logic cell의 수	421	1065
블록의 수	17	16
예상 총 logic cell의 수	7,157	18,040
상대구현면적(%)	39.7	100

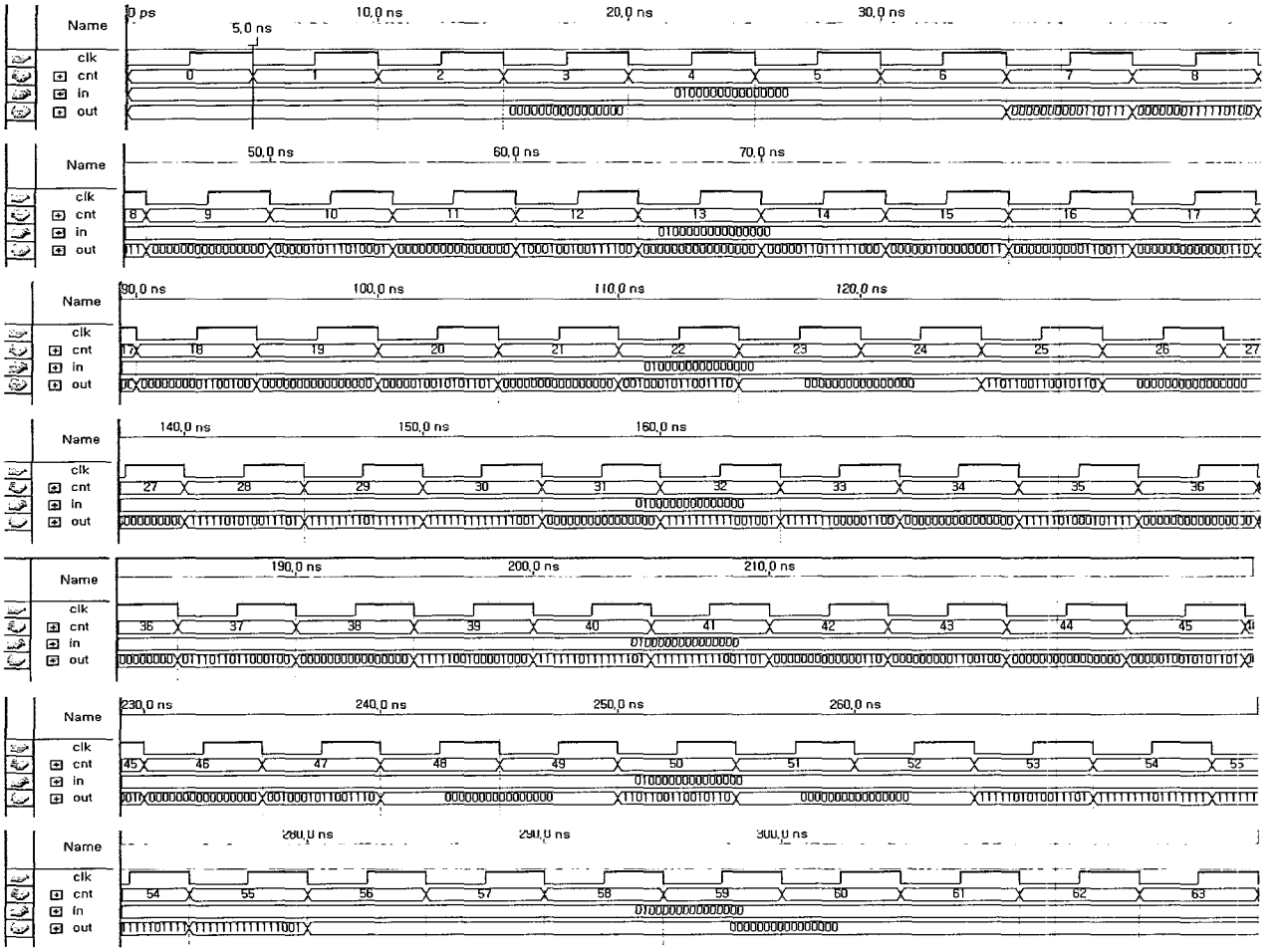

그림 7. 제안된 블록 1-63-31-33 구조의 Verilog-HDL simulation 결과
Fig. 7. Verilog-HDL simulation result of proposed block 1-63-31-33 structure.

표 4. 제안된 구조의 상대 전력소모 비교
Table 4. power consumption for proposed structure.

	제안 구조	곱셈기 구조
processing time (f_clk)	0.29 us	0.17 us
블록 1-63-31-33의 logic cell의 수(C_L)	421	1065
transition(P_t)	5298	62643
블록 1-63-31-33의 상대전력소모	646,954	11,341,515
총 상대전력소모	10,998 K	181,464 K
총 상대전력소모(%)	6.1	100

표 3에서 보듯이 블록 1-63-31-33을 구현하는데 곱셈기 구조는 1065개의 logic cell이 사용되었으며, 제안된 구조는 421개의 logic cell을 사용하였다. 사용된 블록의 수가 각각 17개와 16개 이므로 이를 곱하면 총 logic cell의 수를 예상할 수 있다. 예상되는 총 logic cell의 수는 제안된 구조의 경우에 7157개 이며, 곱셈기 구조는 18040개 이다. 따라서 제안된 구조는 기존의 곱셈기를 사용하는 구조와 비교하여 구현면적을 60.3% 감소시킬 수 있었다.

두 구조의 전력소모를 비교하기 위하여 다음과 같은 식을 사용하였다

$$P_{dyna} = P_t \cdot C_L \cdot V_{dd}^2 \cdot f_{clk} \quad (7)$$

식 (7)의 dynamic 전력소모 P_{dyna} 은 CMOS 디지털 회로의 총 전력소모 가운데 가장 큰 비중을 차지한다. 이 식에서 P_t는 전력이 소모되는 transition의 확률이고, C_L은 부하 커패시턴스, V_{dd}는 사용전압, f_{clk}는 클록의 주파수이다. P_t는 Verilog-HDL 코딩에서 얻을 있으며, C_L은 표 3에서 얻은 logic cell의 수를 사용하였다. 또한 f_{clk}도 Verilog-HDL 코딩에서 얻은 처리시간을 사용하였다. 이와 같이 계산된 블록 1-63-31-33에 대한 곱셈기 구조와 제안된 구조의 전력소모의 비교는 표 4와 같다. 표 4에서 볼 수 있듯이 제안된 구조의 전력 소모는 곱셈기 구조를 사용하였을 때보다 93.9% 감소함을 확인하였다.

Ⅳ. 결 론

MPEG audio 알고리즘의 512탭 FIR 필터를 덧셈기를 사용하여 저전력으로 구현할 수 있는 필터구조를 제안하였다. 즉, CSD 형의 계수와 공통패턴 공유를 최대로 이용하기 위하여 입력신호 샘플을 최대로 공유할 수 있는 구조를 제안하였다. 제안된 구조를 만들기 위하여 (1) 알고리즘에서 사용된 공통입력 공유, (2) 선형위상 대칭 필터계수를 이용한 공유, (3) 공통입력을 이용한 블록 공유, (4) CSD 형의 계수와 공통패턴 공유를 통하여 저전력 구조를 제안하였다. 이와 같이 제안된 구조를 통하여 기존의 곱셈기 구조와 비교하여 구현 면적을 60.3% 감소시킬 수 있었으며, 전력소모는 94%를 감소시킬 수 있었다. 따라서 제안된 MPEG audio 필터 구조는 고속의 DSP 프로세서 대신에 AU나 마이크로 프로세서를 사용하여 효과적으로 구현될 수 있음을 보였다.

참 고 문 헌

[1] "ISO/IEC 11172 MPEG-1 Committee Draft," Part 3 AUDIO.
[2] W. Reitwiesner, "Binary arithmetic," in Advances in Computers, New York: Academic, vol. 1, pp. 231-308, 1966.
[3] K. Hwang, Computer Arithmetic: Principles, Architecture, and Design, New York: Wiley, 1979.
[4] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing, vol. 43, No. 10, pp. 677-688, Oct. 1996.
[5] M. Yagyu, A. Nishihara, and N. Fujii, "Fast FIR digital filter structures using minimal number of adders and its application to filter design," IEICE Trans. Fundamentals of Electronics Communica -tions & Computer Sciences, vol. E79-A No. 8, pp. 1120-1129, Aug. 1996.

저 자 소 개

장 영 범(정회원)

1981년 연세대학교 전기공학과 학사 졸업

1990년 Polytechnic University 대학원 공학석사 졸업

1994년 Polytechnic University 대학원 공학박사 졸업

1981년~1999년 삼성전자 System LSI 사업부 수석연구원

2002년~현재 상명대학교 정보통신공학과 교수

<주관심분야: 통신신호처리, 오디오/음성 신호처리, 신호처리용 SOC 설계>

이 원 상(정회원)

2004년 2월 상명대학교 컴퓨터 시스템 공학전공 학사졸업

2004년 2월~ 현재 상명대학교 대학원 컴퓨터 정보 통신 공학과 석사과정

<주관심분야: 통신신호처리, 통신 신호처리용 SOC 설계>