논문 2006-43SP-3-12

H.264용 디블로킹 필터의 저전력 구조

(Low-power Structure for H.264 Deblocking Filter)

장 영 범*, 오 세 만**, 박 진 수**, 한 규 훈**, 김 수 홍***

(Young-Beom Jang, Se-Man Oh, Jin-Su Park, Kyu-Hoon Han, and Soo-Hong Kim)

요 약

이 논문에서는 H.264 비디오 코딩에 사용되는 디블로킹 필터의 저전력 구조를 제안하였다. 즉, 8 픽셀의 입력에 대한 공통의 필터계수를 공유함으로써 구현 하드웨어를 줄일 수 있는 효율적인 구조를 제안하였다. 제안된 디블로킹 필터 구조는 MUX와 DEMUX 회로를 추가하여 설계하였으며, 기존 구조와 비교하여 44.2%의 덧셈연산 감소효과를 나타내었다. 또한 제안된 구조를 Verilog HDL 코딩과 FPGA로 구현한 결과, 기존의 디블로킹 필터 구조와 비교하여 각각 19.5%와 19.4%의 게이트 카운트 감소 효과를 보였다. 따라서 제안된 디블로킹 필터 구조는 H.264용 encoder와 decoder SoC에 널리 사용될 수 있는 저전력 구조이다.

Abstract

In this paper, a low-power deblocking filter structure for H.264 video coding algorithm is proposed. By sharing addition hardware for common filter coefficients, we have designed an efficient deblocking filter structure. Proposed deblocking filter utilizes MUX and DEMUX circuits for input data sharing and shows 44.2% reduction for add operation. In the HDL coding simulation and FPGA implementation, we achieved 19.5% and 19.4% gate count reduction, respectively, comparison with the conventional deblocking filter structure. Due to its efficient processing scheme, the proposed structure can be widely used in H.264 encoding and decoding SoC.

Keywords: H.264, deblocking filter, blocking artifacts, boundary strength

I. 서 론

최근 동영상의 압축방법으로 H.263과 H.264의 표준 방식이 널리 사용되고 있다.[1] 이와 같은 부호화기들은 블록 기반의 압축방식을 사용하고 있으며, 특히 저비트율로 압축을 수행할 경우에 블록간의 경계가 보이는 블로킹 현상(blocking artifacts)이 발생한다.[2] H.263 부호화기에서는 블로킹 현상을 완화하기 위하여 OBMC (Overlapped Block Motion Compensation) 방법을 채택하였다.[3] 그러나 이 방법으로는 블로킹 현상을 제거하는데 한계가 있으며 후처리(post processing) 기술과 병행하여 사용함으로써 블로킹 현상을 감소시킬 수 있다. H.264 표준 부호화기에서는 블로킹 현상을 제거하기 위하여 디블로킹 필터를 사용하고 있다. 디블로킹 필터에는 두 가지 방식이 있는데, 첫째는 후처리 필터(post filters) 방식이고 두 번째 방식은 루프 필터(loop filters) 방식이다. 후처리 필터 방식은 코딩 루프 밖의 디스플레이 버퍼에서 동작하며 따라서 표준 압축방식에서 반드시 적용해야하는 것은 아니다. 이와 반대로 루프 필터 방식은 코딩 루프 안에서 동작하는 방식이다. 즉, 필터링된 프레임이 움직임 보상을 위한 참조 프레임(reference frame)으로 사용되므로 디코더에서도 같은 필터가 사용된다. 코딩 루프 안에서 동작하는 루프 필터 방식은 후처리 필터 방식과 비교하여 여러 가지 장점을 갖고 있으며 H.264 표준 부호화기에서도 이와 같은 루프 필터 방식의 디블로킹 필터를 사용하고 있다.[4]

* 정회원, ** 학생회원, 상명대학교 정보통신공학과
(College of Engineering, Sangmyung University)
*** 정회원, 상명대학교 컴퓨터소프트웨어공학과
(College of Engineering, Sangmyung University)
※ 본 논문은 산업자원부의 신기술실용화기술개발 사업으로 수행한 연구결과입니다.
접수일자: 2006년3월3일, 수정완료일: 2006년4월24일

이와 같은 디블로킹 필터의 연산량은 전체 복호기 연산량의 10~20%를 차지하므로 저전력 구현이 요구되고 있다.[5] 이 논문에서는 H.264에 사용되는 루프 필터 방식의 디블로킹 필터를 효율적으로 구현하는 저전력 구조를 제안한다.

Ⅱ.제안된 디블로킹 필터 구조

1. Boundary strength의 선택

H.264에서는 16×16 luma 블록과 8×8 chroma 블록에 대하여 디블로킹 필터를 사용한다. 이와 같은 블록에서 4개 픽셀마다의 수직경계와 수평경계에서 디블로킹 필터를 수행하게 된다. 따라서 luma 블록에는 각각 4개의 수직경계와 수평경계가 있으며 chroma 블록에는 각각 2개의 수직경계와 수평경계가 있다. 수평 디블로킹 필터의 경우에 수직경계면의 각각 4개의 픽셀들이 입력신호로 사용되며 수직경계면 양쪽의 각각 3개의 픽셀들이 새로 필터링되어 출력된다. 이 경우에 수직경계면의 좌측 4개의 신호들을 경계면에서 먼 신호부터 p_3, p_2, p_1, p_0로 명명하며, 수직경계면의 우측 4개의 신호들은 경계면에서 가까운 신호부터 q_0, q_1, q_2, q_3로 명명한다. 이와 같은 명명은 수평경계에서도 똑같이 적용된다. 경계면에서 항상 같은 필터링이 적용되는 것이 아니라 조건에 따라 5가지의 필터링이 사용되며 사용되는 조건 및 필터 종류는 그림 1과 같다.

그림 1에서 bS=0은 필터링이 적용되지 않는다. 따라서 실제로 사용되는 필터는 bS=1, 2, 3, 4등의 4개가 필요하다. 또한 경계면 강도가 가장 센 bS=4가 가장 강력한 필터를 요구한다.

2. 제안된 bS=4 필터의 저전력 구조

전 절에서 기술된 것과 같이 4가지 종류의 필터가 사용되며, 먼저 bS=4의 필터의 사양을 알아보기로 한다. 연산된 필터의 출력은 대문자로 표기하기로 한다. 즉, p_3, p_2, p_1, p_0의 연산 후 출력은 P_3, P_2, P_1, P_0로 표기한다. 먼저 bS=4 모드에서 다음의 조건을 만족하는지 검사한다.

$$|p_2 - p_0| < \beta,\ |q_2 - q_0| < \beta,\\ |p_0 - q_0| < ((\alpha \gg 2) + 2) \tag{1}$$

위의 식에서 α, β는 H.264의 스펙에 제시된 상수값

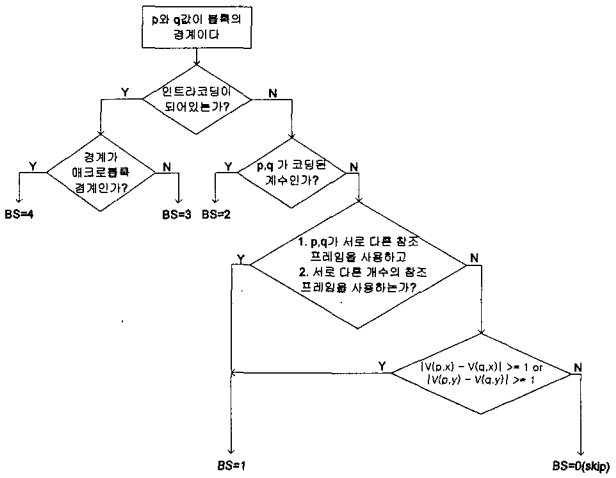


그림 1. 경계면 강도의 조건에 따른 필터의 종류
Fig. 1. Filter modes from boundary strength condition.

이다.

식 (1)의 조건을 만족하면 다음의 필터링을 수행하여 출력을 구한다.

$$\begin{aligned} P_0 &= (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) \gg 3\\ P_1 &= (p_2 + p_1 + p_0 + q_0 + 2) \gg 2\\ P_2 &= (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) \gg 3\\ Q_0 &= (p_1 + 2p_0 + 2q_0 + 2q_1 + q_2 + 4) \gg 3\\ Q_1 &= (p_0 + q_0 + q_1 + q_2 + 2) \gg 2\\ Q_2 &= (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4) \gg 3 \end{aligned} \tag{2}$$

식 (1)의 조건을 만족하지 않으면 다음의 필터링을 수행하여 출력을 구한다.

$$\begin{aligned} P_0 &= (2p_1 + p_0 + q_1 + 2) \gg 2\\ Q_0 &= (2q_1 + q_0 + p_1 + 2) \gg 2 \end{aligned} \tag{3}$$

bS=4 필터를 수행하기 위한 식 (2)와 (3)을 살펴보면 모두 필터계수가 1, 2, 또는 3이므로 곱셈연산이 필요하지 않으며, 오직 덧셈연산으로 필터링을 수행할 수 있다. 예를 들면 식 (2)의 P_0를 계산하기 위해서 5개의 덧셈연산이 필요하다. 또한 3의 곱셈연산도 1개의 쉬프트 연산과 1개의 덧셈연산으로 구현될 수 있다. 따라서 식 (2)와 (3)을 구현하기 위하여 총 36개의 덧셈연산이 필요하게 된다. 이와 같은 덧셈연산의 수를 감소시키기 위하여 우리는 다음과 같이 MUX를 사용하여 구현 하드웨어를 공유하는 방식을 제안한다. 먼저 P_2, Q_2를 식 (2)에서 직접 구현하면 각각 6개씩 12개의 덧셈연산이 필요하게 된다. 그러나 덧셈연산의 수를 줄이기 위해 이제 P_2, Q_2를 다음과 같이 나타내 보자.

$$[P_2\ Q_2]=\left\{[2\ 3\ 1\ 1\ 1]\begin{bmatrix}p_3 & q_3\\ p_2 & q_2\\ p_1 & q_1\\ p_0 & q_0\\ q_0 & p_0\end{bmatrix}+[4\ 4]\right\}\begin{bmatrix}0.125 & 0\\ 0 & 0.125\end{bmatrix} \quad (4)$$

식 (4)에서 보듯이 $[p_3p_2p_1p_0q_0]$의 연산과 $[q_3q_2q_1q_0p_0]$의 연산이 같은 계수 [23111]을 사용하므로, MUX를 사용하면 5개의 덧셈연산으로 구현이 가능하다. 역시 [4 4]의 연산도 1개의 덧셈연산이면 된다. 또한 식 (4)의 2×2 행렬은 쉬프트연산으로 구현할 수 있다. 따라서 P_2, Q_2를 구현하는데 6개의 덧셈연산이 필요하게 되어 12개에서 6개로 줄일 수 있다. 이제 P_1, Q_1도 덧셈연산 수를 감소시키기 위하여 다음 식과 같이 나타낼 수 있다.

$$[P_1\ Q_1]=\left\{[1\ 1\ 1\ 1]\begin{bmatrix}p_2 & p_0\\ p_1 & q_0\\ p_0 & q_1\\ q_0 & q_2\end{bmatrix}+[2\ 2]\right\}\begin{bmatrix}0.25 & 0\\ 0 & 0.25\end{bmatrix} \quad (5)$$

식 (5)를 사용하면 P_1, Q_1를 구현하는데 4개의 덧셈연산이 필요하게 되어 8개에서 4개로 줄일 수 있다. 같은 방법으로 P_0, Q_0는 다음과 같이 나타낼 수 있다.

$$[P_0\ Q_0]=\left\{[1\ 2\ 2\ 2\ 1]\begin{bmatrix}p_2 & p_1\\ p_1 & p_0\\ p_0 & q_0\\ q_0 & q_1\\ q_1 & q_2\end{bmatrix}+[4\ 4]\right\}\begin{bmatrix}0.125 & 0\\ 0 & 0.125\end{bmatrix} \quad (6)$$

역시 식 (6)을 사용하면 P_0, Q_0를 구현하는데 5개의 덧셈연산이 필요하게 되어 10개에서 5개로 줄일 수 있다. 마지막으로 예외 조건에서의 필터링인 식 (3)의 P_0, Q_0도 다음과 같이 나타낼 수 있다.

$$[P_0\ Q_0]=\left\{[2\ 1\ 1]\begin{bmatrix}p_1 & q_1\\ p_0 & q_0\\ q_1 & p_1\end{bmatrix}+[2\ 2]\right\}\begin{bmatrix}0.25 & 0\\ 0 & 0.25\end{bmatrix} \quad (7)$$

식 (7)을 사용하면 P_0, Q_0를 구현하는데 3개의 덧셈연산이 필요하게 되어 6개에서 3개로 줄일 수 있다. 지금까지 제안한 식 (4)~(7)을 덧셈기, 쉬프트, MUX와 DEMUX를 사용하여 구현한 구조는 그림 2와 같다. 그림 2에서 보듯이 제안된 bS=4의 필터구조를 사용한 결과, 36개의 덧셈연산을 18개로 줄일 수 있었다. 부가적으로 6개의 MUX 회로와 4개의 DEMUX 회로가 사용

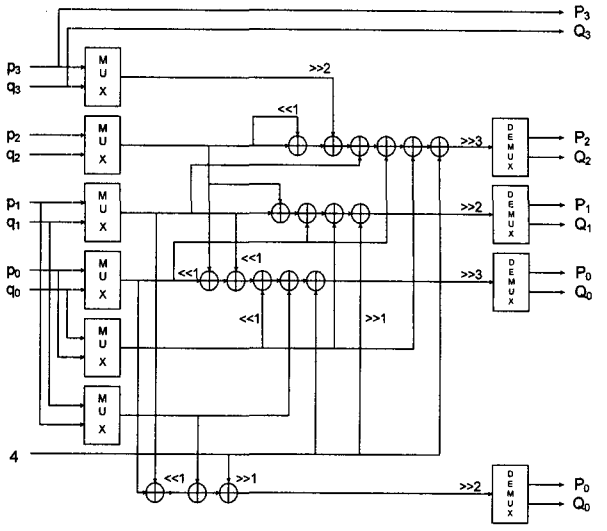


그림 2. 제안된 bS=4 필터 구조
Fig. 2. Proposed bS=4 filter structure.

됨을 알 수 있다. 그러나 이는 덧셈연산의 감소량과 비교하여 구현면적이 상대적으로 매우 작다.

3. 제안된 bS=1/2/3 필터의 저전력 구조

H.264 디블로킹 필터의 bS=1/2/3 모드는 기본 필터와 클리핑 회로로 구성된다. 클리핑 회로를 사용하는 까닭은 이 회로를 통하여 과도한 저역통과 필터링이 되는 것을 방지할 수 있기 때문이다. bS=1/2/3 모드에서는 P_1, Q_1은 다음과 같이 계산된다.

$$\begin{aligned}P_1 &= p_1 + \Delta_{p1}\\ Q_1 &= q_1 + \Delta_{q1}\end{aligned} \quad (8)$$

식 (8)의 Δ_{p1}, Δ_{q1}의 초기값인 $\Delta_{p1i}, \Delta_{q1i}$를 다음과 같이 먼저 계산한다.

$$\begin{aligned}\Delta_{p1i} &= (p_2 + ((p_0 + q_0 + 1) \gg 1) - 2p_1) \gg 1\\ \Delta_{q1i} &= (q_2 + ((p_0 + q_0 + 1) \gg 1) - 2q_1) \gg 1\end{aligned} \quad (9)$$

식 (9)에서 구한 값들은 블러링을 막기 위하여 다음과 같이 클리핑된다.

$$\begin{aligned}\Delta_{p1} &= (\ \mathrm{Min}(\ \mathrm{Max}\ (-c_1, \Delta_{p1i}),\ \ c_1)\\ \Delta_{q1} &= (\ \mathrm{Min}(\ \mathrm{Max}\ (-c_1, \Delta_{q1i}),\ \ c_1)\end{aligned} \quad (10)$$

식 (8)과 (9)를 직접 구현하면 10개의 덧셈연산이 필요하다. 덧셈 연산의 수를 줄이기 위해 식 (8)과 (9)를 다음과 같이 나타냄으로써 10개의 덧셈연산을 5개로 줄일 수 있게 된다.

$$[P_1\ Q_1]=C\left(\left\{[1\ -2\ 0.5\ 0.5]\begin{bmatrix}p_2 & q_2\\ p_1 & q_1\\ p_0 & q_0\\ q_0 & p_1\end{bmatrix}+[0.5\ 0.5]\right\}\begin{bmatrix}0.5 & 0\\ 0 & 0.5\end{bmatrix}\right)+[p_1\ q_1] \quad (11)$$

제안된 식 (11)은 계수용으로 3개, [0.5 0.5]용으로 1개, $[p_1, q_1]$용으로 1개 등 5개의 덧셈연산으로 구현할 수 있다. 식 (11)에서 C는 클리핑 연산을 나타낸다. 마지막으로 P_0, Q_0는 다음과 같이 계산된다.

$$\begin{aligned} P_0 &= p_0 + \Delta_0 \\ Q_0 &= q_0 - \Delta_0 \end{aligned} \quad (12)$$

식 (12)의 Δ_0의 초기 값인 Δ_{0i}를 다음과 같이 먼저 계산한다.

$$\Delta_{0i} = (4(q_0 - p_0) + (p_1 - q_1) + 4) \gg 3 \quad (13)$$

식 (13)에서 구한 값은 블러링을 막기 위하여 다음과 같이 클리핑 된다.

$$\Delta_0 = (\ \mathrm{Min}(\ \mathrm{Max}(-c_0, \Delta_{0i}),\quad c_0) \quad (14)$$

식 (12)와 (13)을 구현하는 데에는 6개의 덧셈연산이 필요하다. 이 덧셈연산은 식(13)이 동시에 식 (12)에 더해지므로 줄일 수 없다. 지금까지 제안한 bS=1/2/3의 식을 덧셈기, 쉬프트, MUX와 DEMUX를 사용하여 구현한 구조는 그림 3과 같다. 그림 3에서 보듯이 제안된 bS=1/2/3의 필터구조를 사용함으로써 16개의 덧셈연산을 11개로 줄일 수 있었다. 제안 구조는 부가적으로 2개의 MUX 회로와 1개의 DEMUX 회로를 사용한다.

그림 3에서 네모상자 C는 클리핑회로를 나타낸다. 이 절과 지난 절에서 H.264에서 사용되는 디블로킹 필터의 저전력 구조를 제안하였다. 지금까지 제안된 bS=4와 bS=1/2/3 필터 구조를 통하여 총 52개의 덧셈연산을 29개로 감소시킬 수 있었다. 이는 44.2%의 덧셈연산 감소를 의미한다. 다음 절에서는 모드 결정의 하드웨어를 포함한 전체 디블로킹 필터를 HDL 코드로 구현하여 제안된 구조의 실용성을 알아보기로 한다.

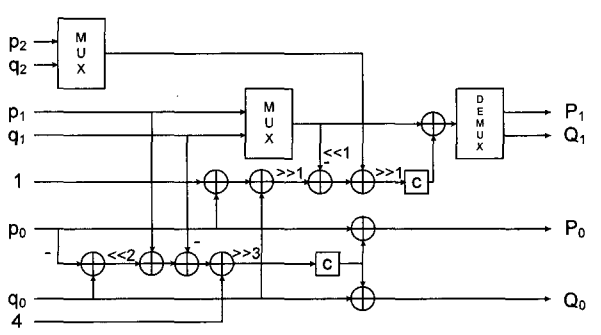


그림 3. 제안된 bS=1/2/3의 필터 구조
Fig. 3. Proposed bS=1/2/3 filter structures.

Ⅲ. 실험 및 고찰

1. Verilog-HDL 시뮬레이션

이 절에서는 제안된 구조를 HDL로 시뮬레이션하여 구조의 효과를 검증한다. H.264의 디블로킹 필터에 들어오는 입력데이터는 integer transform과 quantization 블록을 거쳐서 들어오게 된다. bS=4 모드에서의 필터링 실험을 위하여 식 (1) 조건식의 파라미터 α와 β를 알아야한다. α와 β는 표준안에서 정의된 임계값으로서, 다음과 같이 결정된다.

$$\alpha(x) = 0.8(2^{\frac{x}{6}} - 1) \quad (15)$$

$$\beta(x) = 0.5x - 7 \quad (16)$$

식 (15)와 (16)의 x값은 각각 IndexA와 IndexB로 나타내며, 다음과 같이 구한다.

$$\mathrm{IndexA} = \mathrm{Min}(\ \mathrm{Max}(\ 0,\ \mathrm{QP}+\mathrm{OffsetA}),\ 51) \quad (17)$$

$$\mathrm{IndexB} = \mathrm{Min}(\ \mathrm{Max}(\ 0,\ \mathrm{QP}+\mathrm{OffsetB}),\ 51) \quad (18)$$

식 (17)과 (18)에서 사용되는 QP(quantization parameter) 값은 quantization 블록에서 이미 구한 값이다. IndexA와 IndexB에 의해 결정되는 α와 β를 Index 별로 유도하여 표로 나타내면 표 1과 같다.

또한 bS=1/2/3 모드에서의 필터링 실험을 위하여 식 (10)과 (14) 조건식의 파라미터 c_1과 c_0를 알아야한다. 먼저 c_1은 IndexA와 bS 값으로 결정되며 표 2와 같다. c_0 파라미터는 c_1의 값을 근거로 하여 결정되는데 MB이 luma Block이면 다음 조건에 따라 달라진다.

표 1. indexA와 indexB에 따른 α, β의 값
Table 1. Derivation of α and β from indexA and indexB.

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	131	14	15	16	17
α	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4
β	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2

idx	18	19	20	21	22	23	24	25	26	27	28	29	30	311	32	33	34	35
α	5	6	7	8	9	10	12	13	15	17	20	22	25	28	32	36	40	45
β	2	3	3	3	3	4	4	4	6	6	7	7	8	8	9	9	10	10

idx	36	37	38	39	40	41	42	43	44	45	46	47	48	491	50	51
α	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

표 2. IndexA와 bS에 따른 클리핑 변수 c_1의 결정
Table 2. Clipping variable c_1 determination from indexA and bS.

idxA	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
bS=1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bS=2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bS=3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

idxA	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
bS=1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2
bS=2	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	3
bS=3	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4

idxA	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
bS=1	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13
bS=2	3	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17
bS=3	4	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25

표 3. 제안 구조의 덧셈연산 수 비교
Table 3. Comparison of addition for proposed structure.

구분	기존구조	제안구조
bS=4	36	18 (50%)
bS=1/2/3	16	11 (68.75%)
Total	52	29 (55.77)

$$|p_2 - p_0| < \beta$$
$$|q_2 - q_0| < \beta \qquad (19)$$

조건이 하나라도 참이면 c_0 는 c_1에 1을 더하고 둘 다 참이면 2를 더한다. 둘 다 거짓이면 $c_0 = c_1$이 된다. MB이 chroma 블록이면 $c_0 = c_1 + 1$이 된다.

HDL 시뮬레이션에 앞서 제안된 구조와 기본구조의 덧셈연산의 수를 비교해 보면 다음과 같다. 비교된 기존 구조는 bS4 필터는 식 (2)와 (3)을 직접 구현하는 경우이며, bS 1/2/3 필터는 식 (8)과 (9)를 직접 구현한 경우이다. 제안 구조는 기존 구조와 비교하여 표 3과 같은 덧셈 연산의 감소를 달성하였다.

표 3은 이론적인 제안구조의 덧셈연산 감소이다. 따라서 그림 2와 3의 제안구조를 Verilog-HDL 코딩하여 기존 구조와 비교한 결과를 표 4에 나타냈다.

표 4는 제어 및 모드결정을 제외한 순수 필터부분의 HDL 코딩 결과이다. Xilinx ISE 7.1i를 사용하여 합성하였고 Mentor Graphics사의 modelSim을 이용하여 시뮬레이션 하였다.

표 4에서 보듯이 디블로킹 필터에 대한 제안 구조의 코딩 결과 24.9%의 게이트 카운트 감소효과를 얻을 수 있었다.

디블로킹 필터는 필터회로와 제어 및 모드결정 (Control & mode select) 회로로 구성되었으므로 전체 디블로킹 필터를 HDL로 코딩하여 제안구조의 게이트 카운트를 비교해보기로 한다. 그림 4는 디블로킹 필터의 HDL 코딩을 위한 블록도이다. 제어 및 모드결정 블록은 조건에 따라 bS의 값을 결정하는 블록이다. 또한 MB 단위로 입력된 데이터들의 헤더를 분석하고 필터링에 필요한 p배열과 q배열을 재배열하여 다음 필터링에 사용 할 수 있도록 한다. 즉 수직 경계를 필터링 한 후 결과 값을 Memory 블록에 저장하고 있다가 수평 경계 필터링 할 때 사용되도록 제어한다.

표 4. 제안된 필터구조의 HDL 게이트 카운트 비교
Table 4. Comparison of HDL gate count for proposed filter structure.

구분	기존구조	제안구조
bS=4	1,412	1,079 (76.4%)
bS=1/2/3	2,127	1,579 (74.2%)
Total	3,539	2,658 (75.1%)

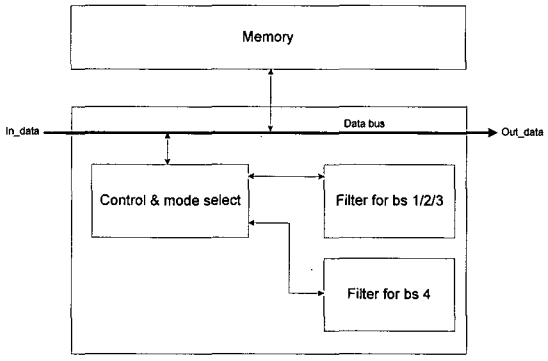


그림 4. 디블로킹 필터의 HDL 코딩을 위한 블록도
Fig. 4. Block diagram for deblocking filter HDL coding.

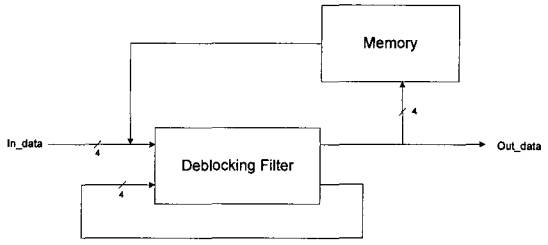


그림 5. 데이터 흐름도
Fig. 5. Data flow diagram.

실험을 위한 데이터의 흐름은 그림 5와 같이 구성하였다. 즉, 그림 5에서 보듯이 8개의 입력이 한 번에 필터부에 들어가서 필터링 되고 필터의 출력 중 다음연산에 필요한 4개의 출력은 피드백 되어 다시 필터로 들어가고 다른 4개의 출력은 나중에 수평 경계 필터에 사용되기 위해 메모리에 저장된다. 메모리에 256개의 데이터가 저장되면 수평방향 필터링이 시작되고 수평방향 필터링된 값은 최종 출력으로 나간다.

제어 및 모드결정을 포함한 디블로킹 필터 전체의

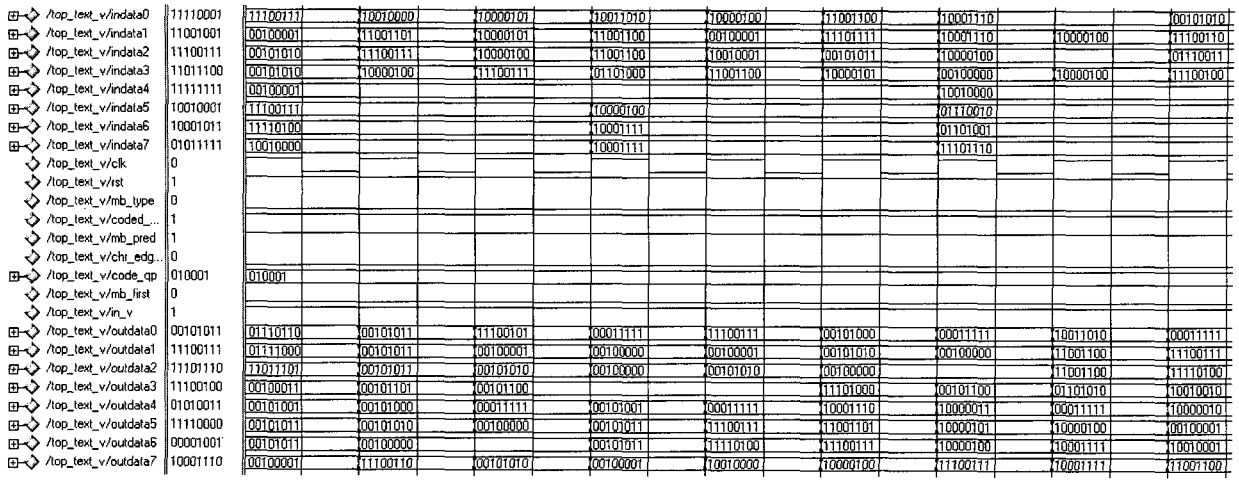


그림 6. 제안구조를 적용한 디블로킹 필터의 Verilog-HDL 시뮬레이션 결과
Fig. 6. Verilog-HDL simulation result for proposed deblocking filter.

표 5. 제안된 디블로킹 필터의 게이트 카운트 비교
Table 5. Comparison of gate count for deblocking filter using proposed structure.

구분	기존구조	제안구조
필터 회로	3,539	2,658 (75.1%)
제어 및 모드결정 회로	920	932
Total	4,459	3,590 (80.5%)

HDL 코딩 결과는 다음 표 5와 같다.

표 5에서 보듯이 제어 및 모드결정을 포함한 전체 디블로킹 필터에 대한 제안 구조의 코딩 결과 19.5%의 게이트 카운트 감소효과를 달성할 수 있었다

검증을 위한 입력신호로서 Foreman(CIF)의 영상을 사용하였고 입력값을 8비트 2진수로 변화하여 입력하였다. 4개의 입력이 들어갈 때 마다 헤더 정보가 함께 들어가게 되고 출력은 첫 입력이 들어가고 나서 256 클럭 후에 출력이 나온다.

그림 6은 modelSim을 사용한 출력 타이밍도이다. 제안구조는 MUX와 DEMEX를 사용하기 때문에 메인 클럭을 분주하여 2개의 클럭를 사용하며 클럭 generator를 이용하여 만들어 사용하였다.

2. FPGA 구현

이 절에서는 실제 FPGA Device에 설계된 디블로킹 필터를 합성하여 구현하여 제안 구조의 효율성을 비교한다. 구현에 사용된 Device는 Xilinx사의 spartan2 xc2s200을 사용하였다. Xilinx ISE 7.1i은 기본적으로 XST(Xilinx synthesis technology)라는 synthesis program을 사용한다. 시뮬레이션용 RTL 코드를 FPGA 구현용 코드로 수정하여 게이트 카운트를 비교한 결과

표 6. 제안 필터의 FPGA 구현 게이트 카운트
Table 6. FPGA Gate count for proposed filter.

구분	기존구조	제안구조
bS=4	5,349	4,257 (79.58%)
bS=1/2/3	4,780	3,746 (78.36%)
Total	10,129	8,003 (79.01%)

표 7. 제안 디블로킹 필터의 FPGA 구현 게이트 카운트
Table 7. FPGA Gate count for proposed deblocking filter.

구분	기존구조	제안구조
필터 회로	10,129	8,003 (79.3%)
제어 및 모드결정 회로	869	872
Total	10,998	8,875 (80.6%)

표 8. 제안된 디블로킹 필터의 FPGA Device 소자 사용수
Table 8. Number of FPGA devices for proposed deblocking filter.

구분	Logic Utilization	Used	Available	Utilization
기존 구조	Slice Flip Flops	529	4,704	11%
	4 input LUTs	818	4,704	17%
제안 구조	Slice Flip Flops	469	4,704	9%
	4 input LUTs	731	4,704	15%

를 표 6과 7에 나타냈다.

FPGA 게이트 카운트의 결과도 Verilog-HDL의 게이트 카운트와 유사한 결과를 얻었다. 즉, 기존구조와 비교하여 19.4%의 감소 효과를 얻었다.

마지막 실험으로써 FPGA에서의 소자사용 수를 비교해보면 다음과 같다. logic cell의 세부 항목인 Slice Flip Flops과 4 input LUTs의 사용 수를 기존 구조와 비교한 결과를 표 8에 나타냈다.

IV. 결 론

이 논문에서는 H. 264의 encoder와 decoder에서 사용되는 디블로킹 필터의 저전력 구조를 제안하였다. H.264용 디블로킹 필터는 필터회로와 제어 및 모드 결정회로로 구성되는데, 이 논문은 필터회로에 대하여 덧셈연산의 수를 감소시킨 저전력 구조를 제안하였다. 제안된 필터회로는 기존의 구조와 비교하여 44.2%의 덧셈연산 감소효과를 나타내었다. 또한 제안된 필터회로와 전체 디블로킹필터를 Verilog-HDL 코딩하여 비교한 결과, 기존의 구조와 비교하여 각각 24.9%와 19.5%의 게이트 카운트 감소효과를 나타내었다.

마지막으로 제안된 필터회로와 전체 디블로킹필터를 FPGA로 구현하여 비교한 결과, 기존의 구조와 비교하여 각각 20.99%와 19.4%의 게이트 카운트 감소효과를 얻었으며, 이는 Verilog-HDL 실험과 매우 유사한 결과이다. 시뮬레이션과 FPGA 구현을 통하여, 제안된 디블로킹 필터 구조는 H.264용 SoC에 사용될 수 있는 효과적인 구조임을 보였다.

참 고 문 헌

[1] Draft ITU-T recommendation and Final Draft International Standard Of Joint Video Specification(ITU-T Rec. H.264/ISO/IEC 14496-10 AVC), Mar. 2003.

[2] B. Jeon, et al, "Blocking artifacts reduction in image coding based on minimum block discontinuity criterion," IEEE Trans. Circuits and Systems for Video Technology, Vol. 8, no. 3, pp. 345-357, Jun. 1998.

[3] M. Orchard and G. Sullivan, "Overlapped block motion compensation: An estimation-theoretic approach," IEEE Trans. Image Proc., Vol. 3, pp. 693-699, Sep. 1994.

[4] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, "Adaptive deblocking filter," IEEE Trans. Circuits and Systems for Video Technology, Vol. 13, no. 7, pp. 614-619, Jul. 2003.

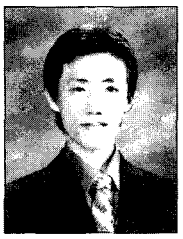
[5] Y. Huang, T. Chen, B. Hsieh, T.Wnag, T. Chang, and L. Chen, "Architecture design for deblocking filter in H.264/JVT/AVC," Proceedings of International Conference on Multimedia and Expo, pp. 693-696, 2003.

저 자 소 개

장 영 범(정회원)
1981년 연세대학교 전기공학과 졸업(공학사)
1990년 Polytechnic University 대학원 졸업(공학석사)
1994년 Polytechnic University 대학원 졸업(공학박사)
1981년~1999년 삼성전자 System LSI 사업부 수석연구원
2000년~2002년 이화여자대학교 정보통신학과 연구교수
2002년~현재 상명대학교 정보통신공학과 교수
<주관심분야 : 통신신호처리, 비디오신호처리, SoC 설계>

박 진 수(학생회원)
2006년 상명대학교 정보통신 공학과 졸업(공학사)
2006년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>

김 수 홍(정회원)
1974년 서울대학교 공과대학 응용수학과 졸업
1990년 서울대학교 대학원 졸업 (이학석사)
1992년 서울대학교대학원 졸업 (이학박사)
1992년 3월~현재 상명대학교 컴퓨터소프트웨어 공학과 교수
<주관심분야 : 병렬처리시스템, 컴퓨터구조, 멀티미디어시스템, E-Business Solution, 공장자동화 시스템>



오 세 만(학생회원)
2005년 상명대학교 정보통신 공학과 졸업(공학사)
2005년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>

한 규 훈(학생회원)
2006년 상명대학교 정보통신 공학과 졸업(공학사)
2006년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>