논문 2008-45SP-5-6

H.264/AVC Encoder용 저전력 IP 설계 및 FPGA 구현

(Low-power IP Design and FPGA Implementation for H.264/AVC Encoder)

장 영 범*, 최 동 규**, 한 재 웅**, 김 도 한**, 김 비 철**, 박 진 수**,
한 규 훈**, 허 은 성**

(Young Beom Jang, Dong Kyu Choi, Jae Woong Han, Do Han Kim, Bee Chul Kim, Jin Su Park, Kyu Hoon Han, and Eun Sung Hur)

요 약

이 논문에서는 제안한 H.264/AVC 인코더의 서브 블록인 Inter prediction 블록, Intra prediction 블록, 디블로킹 필터블록, Transform & Quantization 블록에 대한 저전력 구조를 FPGA로 구현하였다. Inter/Intra prediction 블록에서는 분산연산방식을 통해 가산기의 수를 줄여 60.2%의 면적감소효과를 나타내었으며, 디블로킹 필터블록에서는 하드웨어 공유를 위한 MUX를 사용하여 덧셈연산의 수를 44.3%감소시켰다. 또한, Transform & Quantization 블록에 사용되는 곱셈연산을 CSD와 CSS방식으로 수행하여 면적을 크게 차지하는 곱셈기를 사용하지 않았다. 제안된 저전력 IP들을 사용하여 FPGA(Field Programmable Gate Array)와 ARM 프로세서 기반의 H.264/AVC 인코더를 구현하였다. Baseline Profile을 사용하였고 FPGA와 ARM프로세서가 연동하는 Platform으로 구현하였다. Platform을 사용한 H.264/AVC 인코더 구현을 통하여 제안된 각각의 저전력 IP들이 효율적으로 H.264/AVC 인코더 SoC에서 사용될 수 있음을 확인하였다.

Abstract

In this paper, we are implemented low-power structure for Inter prediction, Intra prediction, Deblocking filter, Transform and Quantization blocks in H.264/AVC Encoder. The proposed Inter/Intra prediction blocks are shown 60.2% cell area reduction by adder reduction through Distributed Arithmetic, 44.3% add operation reduction using MUX for hardware share in Deblocking filter block. Furthermore we applied CSD and CSS process to reduce the cell area instead of multipliers that take a lot of area. The FPGA(Field Programmable Gate Array) and ARM process based H.264/AVC encoder is implemented using proposed low-power IPs. The proposed structure Platforms are implemented to interlock with FPGA and ARM processors. H.264/AVC Encoder implementation using Platforms shows that proposed low-power IPs can use H.264/AVC Encoder SoC effectively.

Keywords : *H.264/AVC, Inter prediction, Intra prediction, Deblocking filter, FPGA, ARM processor*

I. 서 론

멀티미디어 시대에 접어들면서 비디오와 같은 대용량의 데이터를 압축하는 기술에 대해 많은 관심이 집중됨에 따라 동영상의 압축방법으로 H.264/AVC의 표준방식이 널리 사용되고 있다.[1] H.264/AVC의 기본적인 개념 자체는 MPEG-4와 유사하나, 세부적인 내부 구현에 있어 상당 부분 변경된 방식을 채택하고 있다. H.264/AVC 개발자들은 복잡한 처리 기법을 적용해 높은 효율로 영상 데이터를 코딩할 수 있는 다양한 방법을 적용시켰다. 그 결과 기존의 MPEG-4에 비해 월등한 압축 효율을 보여주는 향상된 압축 성능의 코덱을 만들어냈다. 성능은 향상 되었지만 인코더의 구현 복잡도가 증가하였고 전력소모가 크게 증가 하였다. H.264/AVC는 이동통신 분야에 응용되고 있는데, 이동통신기

* 정회원, 상명대학교 정보통신공학과
(College of Engineering, Sangmyung University)
** 학생회원, 상명대학교 컴퓨터정보통신공학과
(Graduate School, Sangmyung University)
접수일자: 2007년11월23일, 수정완료일: 2008년8월6일

기는 낮은 전력을 이용하여 구현하는 것이 중요하므로 H.264/AVC의 저전력 구현 기술이 필요하다. H.264/AVC 인코더의 전체 구성을 살펴보면 Inter & Intra prediction 블록, Deblocking 필터블록, Transform & Quantization 블록 등이 있으며 각 블록에 대한 효율적인 설계 및 구현 기술이 필요하다. 이 논문에서는 효율적인 구조를 지닌 Inter & Intra prediction 블록[3]과 Debloking 필터블록[5]을 포함한 H.264 인코더를 FPGA와 ARM 프로세서를 사용하여 구현한 결과를 보이고자 한다. Ⅱ장부터 Ⅳ장에서는 H.264/AVC의 구현에 이용한 저전력 설계 및 구현 방식을 기술하고 Ⅴ장에서는 FPGA 및 ARM 프로세서를 사용한 구현에 대하여 기술하기로 한다.

Ⅱ. Inter 및 Intra Prediction 설계

2.1 Inter Prediction 블록의 저전력 설계

H.264/AVC 인코더에서 Inter Prediction과 Intra Prediction은 전체 인코더 구현에서 가장 많은 연산을 요구하므로 하드웨어 구현 시, 구현면적에 가장 큰 영향을 준다.[2] 우리는 이 두 블록의 구현에 같은 저전력 구현 방식을 적용하였다. H.264/AVC에서 Inter Prediction은 움직임 추정과 움직임 보상부분으로 나뉜다. 움직임 추정을 하기위해서는 SAD(Sum of Absolute Differences) 값을 구하고 가장 작은 SAD를 찾아야 한다. H.264/AVC의 움직임 추정은 7가지 종류의 가변블록 크기를 갖고 있다. 움직임 추정용 SAD 프로세서에서 많은 연산을 차지하는 것은 7가지의 가변블록의 SAD를 구하는 것이다. 한 매크로 블록에 16개의 4x4 블록들의 SAD를 구하면 나머지 다른 블록들은 이들을 조합함으로써 총 41개의 SAD를 구할 수 있다. 제안된 저전력 구조는 그림 1과 같다.

한 매크로 블록에 대한 SAD 연산은 크게 3 블록으로 구성된다. 첫 번째 블록은 4x4 SAD 계산기로서 4x4 픽셀의 SAD를 다음의 식 (1)로 계산한다.

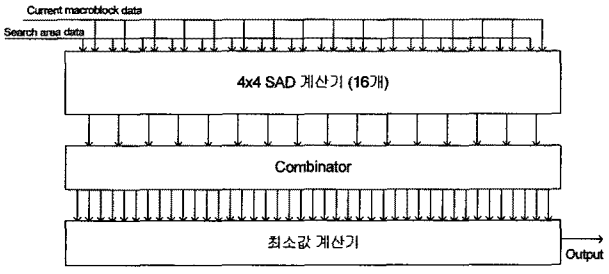


그림 1. 제안된 SAD 프로세서 구조
Fig. 1. Proposed SAD processor structure.

$$SAD(dx,dy)=\sum_{m=x}^{x+N-1}\sum_{n=y}^{y+N-1}|F_k(m,n)-F_{k-1}(m+dx,n+dy)| \quad (1)$$

두 번째 조합기 블록에서는 16개의 4x4 SAD 계산결과를 조합하여 41가지 종류의 SAD를 만들어내며 세 번째 최소값 계산기 블록에서는 다음의 식을 사용하여 최소의 SAD값을 갖는 움직임 벡터를 찾아내는 동작을 수행한다.

$$\overrightarrow{MV}=(MV_x,MV_y)=\min_{(dx,dy)\ R^2}SAD(dx,dy) \quad (2)$$

제안하는 핵심 하드웨어는 첫 번째 블록인 4x4 SAD 계산기의 저전력 구조이다. 즉, 4x4 SAD 계산을 위한 저전력 구조를 제안하며, 이를 16개 사용하여 4x4 SAD 계산기 구조를 설계하였다.[3] 제안된 그림 1의 SAD 프로세서 구조의 첫 번째 블록인 4x4 SAD 계산기에서는 16개의 4x4 SAD를 계산한다. 즉, 16x16 매크로 블록은 16개의 4x4 블록으로 구성되므로 동시에 16개의 4x4 SAD를 계산하는 파트이다. 두 번째 블록인 조합기블록에서는 4x4 SAD들을 조합하여 더 큰 크기의 SAD들을 계산하여 41개의 SAD 값을 만들어낸다. 세 번째 블록인 최소값 계산기에서는 각각의 블록크기에 대한 최소 SAD를 비교하여 가장 작은 값을 골라내는 작업을 수행한다. 4x4 SAD 계산기는 절대 값을 계산하는 AD 연산기와 SAD를 계산하는 하드웨어로 구성된다. 먼저 현재 프레임과 참조 프레임의 탐색영역내의 비교블록의 각 픽셀들의 뺄셈의 절대 값인 AD (absolute deference)를 그림 2와 같이 계산하도록 설계한다.

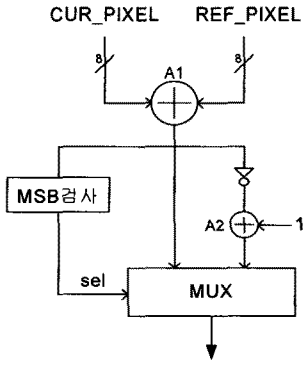


그림 2. 제안된 AD연산기
Fig. 2. AD calculator.

그림 2의 A1 가산기는 전가산기 7개와 반가산기 1개가 사용된다. 그 결과 값의 MSB를 검사하여 0이면, 즉 양수이면 MUX의 sel이 0이 되어 값을 그대로 통과시키게 설계하였다. 그리고 MSB가 1이면, 즉 결과 값이 음수이면 절대값을 계산하기위하여 인버터 회로와 +1 회로를 거치도록 설계하였다. 이 동작이 직렬로 동작하므로 A2의 +1회로는 반가산기 1개로 구성할 수 있다. 기존의 AD 연산기는 A1용으로 전가산기 7개와 반가산기 1개가 사용되며, 절대값 계산용으로 전가산기 8개와 반가산기 1개가 사용된다 .기존의 SAD 방식 계산기 구조와 비교하여 우리는 16개의 연산수를 LSB부터 더하는 방식을 제안한다. 즉, 16개의 AD 값들을 병렬로 받아서 LSB의 1비트부터 계산하는 방식이다. 이와 같은 연산을 분산 연산(Distributed Arithmetic) 방식이라 부르기도 한다. 4x4 SAD 출력 y는 다음과 같이 나타낼 수 있다.

$$y = \sum_{k=0}^{15} x_k, \qquad x_k = \sum_{n=1}^{8} b_{kn} 2^{-n} \tag{3}$$

식 (3)의 두 식을 결합하면 4x4 SAD 출력 y는 다음과 같이 나타낼 수 있다.

$$y = \sum_{k=0}^{15} [\sum_{n=1}^{8} b_{kn} 2^{-n}] = \sum_{n=1}^{8} [\sum_{k=0}^{15} b_{kn}] 2^{-n} \tag{4}$$

식 (4)를 계산하기 위한 제안된 구조는 그림 3과 같다. 그림 3에서 보면 1단계에서는 반가산기가 8개 필요하다. 2단계에서는 반가산기 4개와 전가산기 4개가 필요하다. 3단계에서는 3비트의 데이터를 더해야하므로 반가산기 2개와 전가산기 4개가 필요하다. 마지막으로 4 단계에서는 4비트의 데이터를 더하므로 반가산기 1개와 전가산기 3개가 요구된다. 이렇게 4단계를 거치면 5비트의 LSB의 합이 출력된다. 이 결과 값은 1 비트 우로 쉬프트 되며 다음 클락에 (LSB-1)비트의 합과 더해진다. 그림 3의 전체 가산기를 게이트수준으로 보면 전가산기 26개 반가산기 16개를 사용하여 구현하였다. 기존의 구조와 제안된 그림 3의 구조에 대한 가산기 수를 비교하면 다음 표 1과 같다.

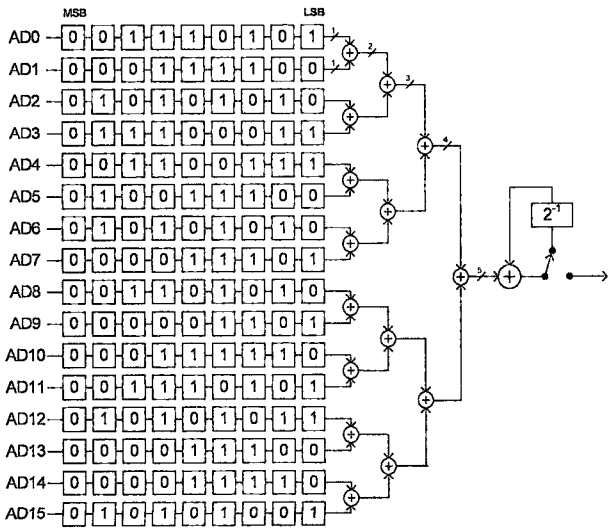


그림 3. 제안된 4x4 SAD 계산기
Fig. 3. Proposed 4x4 SAD calculator.

표 1. 제안 구조의 전가산기와 반가산기의 수 비교
Table 1. Number of full adders and half adders in the proposed structure.

	기존 구조		제안 구조	
	전가산기	반가산기	전가산기	반가산기
AD 연산	240	32	112	32
SAD 연산	131	15	26	16
계	371	47	138	48
상대면적	1113	47	414	48
총상대면적	1160		462	
%	100		39.8	

표 1에서 보듯이 기존 구조와 비교하여 전가산기의 수가 371개에서 138개로 감소되었으며, 반가산기의 수는 47개에서 48개로 1개가 증가하였다. 위의 표에서 상대면적의 계산은 반가산기 1개의 면적을 1로 정의하고 전가산기의 상대면적은 반가산기의 3배로 정의하였다. 그 결과 총상대면적은 기존구조가 1160이고 제안구조가 462로 예상된다. 따라서 제안 구조의 4x4 SAD 연산기와 기존 구조를 비교하였을 때 60.2%의 면적 감소가 예상된다.

2.2 Intra Prediction 블록의 저전력 설계

H.264/AVC의 Intra Prediction은 화면 내에서 유사도를 찾아서 픽셀의 일부 정보만을 가지고 추정된 영상을 만들어낸다. Intra Prediction은 4x4 Intra Prediction과 16x16 Intra Prediction로 나뉜다. Intra Prediction의 하드웨어 블록도는 다음 그림 4와 같다.

그림 4의 16x16 SAE 계산블록과 4x4 SAE 계산블록의 구현에 지난 절에서 제안한 저전력 SAD 방식을 적용하였다. 즉, SAE(Sum of Absolute Error)는 SAD와 같은 개념이므로 SAD 블록의 저전력 구현에 사용한 방식을 여기에도 사용할 수 있다.

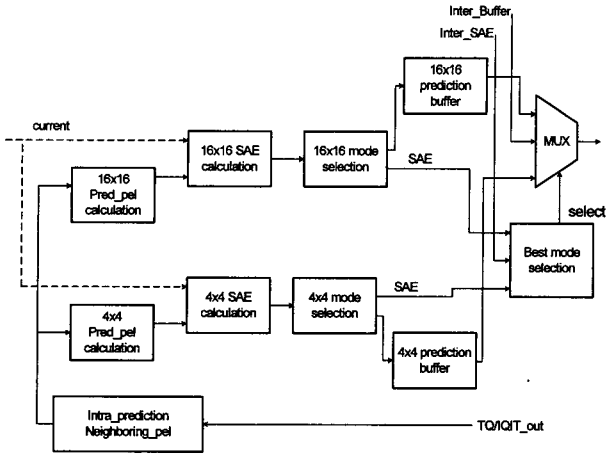


그림 4. Intra Prediction 블록도
Fig. 4. Block diagram of Intra Prediction.

III. Deblocking 필터의 저전력 설계

3.1 제안된 bS=4 필터의 저전력 구조

H.264/AVC 인코더의 Deblocking 필터에서는 16×16 luma 블록과 8×8 chroma 블록에 대하여 Deblocking 필터를 사용한다.[4] 이와 같은 블록에서 4개 픽셀마다의 수직경계와 수평경계에서 Deblocking 필터를 수행하게 된다. 수평 Deblocking 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 사용된다. bS=0에서는 필터링이 적용되지 않는다. 그리고 실제로 사용되는 필터는 bS=1, 2, 3, 4가 있다. 경계면 강도가 가장 센 bS=4가 가장 강력한 필터링을 요구하며 입력 p_3, p_2, p_1, p_0는 다음의 조건을 만족하는지 검사한다.

$$\begin{aligned} &|p_2 - p_0| < \beta,\ |q_2 - q_0| < \beta, \\ &|p_0 - q_0| < ((\alpha \gg 2) + 2) \end{aligned} \tag{5}$$

위의 식에서 α, β는 H.264/AVC의 스펙에 제시된 상수 값이다. 식 (5)의 조건을 만족하면 다음의 필터링을 수행하여 출력을 구한다.

$$\begin{aligned} P_0 &= (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) \gg 3 \\ P_1 &= (p_2 + p_1 + p_0 + q_0 + 2) \gg 2 \\ P_2 &= (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) \gg 3 \\ Q_0 &= (p_1 + 2p_0 + 2q_0 + 2q_1 + q_2 + 4) \gg 3 \\ Q_1 &= (p_0 + q_0 + q_1 + q_2 + 2) \gg 2 \\ Q_2 &= (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4) \gg 3 \end{aligned} \tag{6}$$

식 (5)의 조건을 만족하지 않으면 다음의 필터링을 수행하여 출력을 구한다.

$$\begin{aligned} P_0 &= (2p_1 + p_0 + q_1 + 2) \gg 2 \\ Q_0 &= (2q_1 + q_0 + p_1 + 2) \gg 2 \end{aligned} \tag{7}$$

bS=4 필터를 수행하기 위한 식 (6), (7)을 살펴보면 모두 필터계수가 1, 2, 또는 3이므로 총 36개의 덧셈연산으로 필터링을 수행할 수 있다. 이와 같은 덧셈연산의 수를 감소시키기 위하여 다음과 같이 MUX를 사용하여 구현 하드웨어를 공유하는 방식을 제안한다.[5] 즉, 덧셈연산의 수를 줄이기 위해 이제 P_2, Q_2를 다음과 같이 나타낼 수 있다.

$$[P_2\ \ Q_2] = \left\{ [2\ 3\ 1\ 1\ 1] \begin{bmatrix} p_3 & q_3 \\ p_2 & q_2 \\ p_1 & q_1 \\ p_0 & q_0 \\ q_0 & p_0 \end{bmatrix} + [4\ 4] \right\} \begin{bmatrix} 0.125 & 0 \\ 0 & 0.125 \end{bmatrix} \tag{8}$$

식 (8)에서 보듯이 $[p_3 p_2 p_1 p_0 q_0]$의 연산과 $[q_3 q_2 q_1 q_0 p_0]$의 연산이 같은 계수 $[2\,3\,1\,1\,1]$을 사용하므로, 5개의 덧셈연산으로 구현이 가능하다. 역시 [4 4]의 연산도 1개의 덧셈연산이면 된다. 또한 식 (8)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 다음 식과 같이 나타낼 수 있다.

$$[P_1\ Q_1] = \left\{ [1\ 1\ 1\ 1] \begin{bmatrix} p_2 & p_0 \\ p_1 & q_0 \\ p_0 & q_1 \\ q_0 & q_2 \end{bmatrix} + [2\ 2] \right\} \begin{bmatrix} 0.25 & 0 \\ 0 & 0.25 \end{bmatrix} \tag{9}$$

식 (9)를 사용하면 P_1, Q_1를 구현하는데 4개의 덧셈연산이 필요하게 되어 8개에서 4개로 줄일 수 있다. 같은

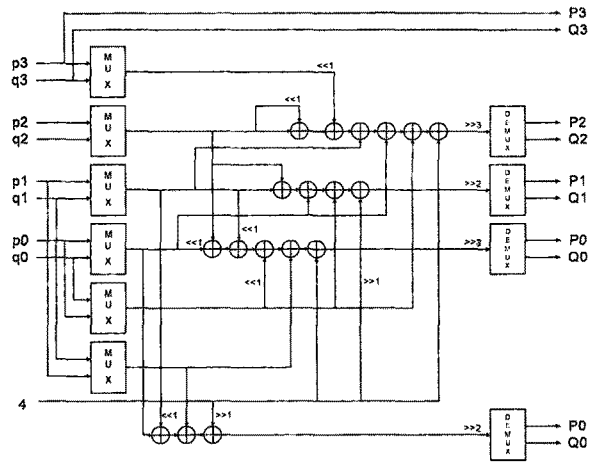


그림 5. 제안된 bS=4 필터 구조
Fig. 5. Proposed bS=4 filter structure.

방법으로 P_0, Q_0는 다음과 같이 나타낼 수 있다.

$$[P_0\ Q_0]=\left\{[1\ 2\ 2\ 2\ 1]\begin{bmatrix}p_2 & p_1\\ p_1 & p_0\\ p_0 & q_0\\ q_0 & q_1\\ q_1 & q_2\end{bmatrix}+[4\ 4]\right\}\begin{bmatrix}0.125 & 0\\ 0 & 0.125\end{bmatrix} \tag{10}$$

역시 식 (10)을 사용하면 P_0, Q_0를 구현하는데 5개의 덧셈연산이 필요하게 되어 10개에서 5개로 줄일 수 있다. 마지막으로 예외 조건에서의 필터링인 식 (7)의 P_0, Q_0도 다음과 같이 나타낼 수 있다.

$$[P_0\ Q_0]=\left\{[2\ 1\ 1]\begin{bmatrix}p_1 & q_1\\ p_0 & q_0\\ q_1 & p_1\end{bmatrix}+[2\ 2]\right\}\begin{bmatrix}0.25 & 0\\ 0 & 0.25\end{bmatrix} \tag{11}$$

식 (11)을 사용하면 P_0, Q_0를 구현하는데 3개의 덧셈연산이 필요하게 되어 6개에서 3개로 줄일 수 있다. 지금까지 제안한 식 (8)~(11)을 덧셈기, 쉬프트, MUX와 DEMUX를 사용하여 구현한 구조는 그림 5와 같다.

그림 5에서 보듯이 제안된 bS=4의 필터구조를 사용한 결과, 36개의 덧셈연산을 18개로 줄일 수 있었다.

3.2 제안된 bS=1/2/3 필터의 저전력 구조

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 모드 또한 3.1 절과 같은 원리로 구조를 구현 할 수 있다. 즉, 3.1 절에서 기술한 방식을 사용하면 P_1, Q_1은 다음과 같다.

$$[P_1\ Q_1]=C\left(\left\{[1\ -2\ 0.5\ 0.5]\begin{bmatrix}p_2 & q_2\\ p_1 & q_1\\ p_0 & q_0\\ q_0 & p_1\end{bmatrix}+[0.5\ 0.5]\right\}\begin{bmatrix}0.5 & 0\\ 0 & 0.5\end{bmatrix}\right)+[p_1\ q_1] \tag{12}$$

기존의 방식으로 구현하면 10개의 덧셈연산이 필요하지만 식 (12)를 사용하여 5개로 줄일 수 있게 된다. 즉, 식 (12)는 계수용으로 3개, [0.5 0.5]용으로 1개, $[p_1, q_1]$용으로 1개 등 5개의 덧셈연산으로 구현할 수 있다. 식 (12)에서 C는 클리핑 연산을 나타낸다. 마지막으로 P_0, Q_0는 다음과 같이 계산된다.

$$\begin{aligned}&P_0=p_0+\Delta_0,\quad Q_0=q_0-\Delta_0\\ &\Delta_{0i}=(4(q_0-p_0)+(p_1-q_1)+4)\gg 3\end{aligned} \tag{13}$$

식 (13)에서 구한 값은 블러링을 막기 위하여 클리핑 한다. 지금까지 제안한 bS=1/2/3의 식을 구현한 구조는 그림 6과 같다.

그림 6에서 보듯이 제안된 구조를 사용함으로써 16개의 덧셈연산을 11개로 줄일 수 있었다. 지금까지 제안된 bS=4와 bS=1/2/3 필터 구조를 통하여 총 52개의 덧셈연산을 29개로 감소시킬 수 있었다.

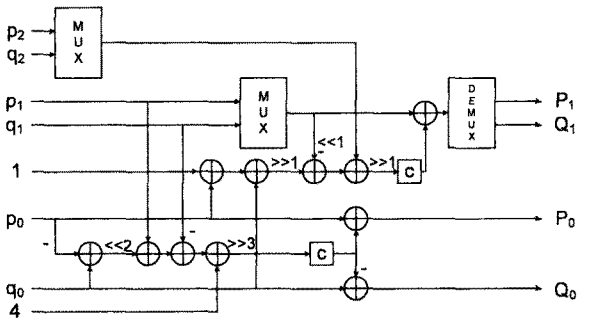


그림 6. 제안된 bS=1/2/3의 필터 구조
Fig. 6. Proposed bS=1/2/3 filter structures.

Ⅳ. Transform 및 Quantization 블록 설계

H.264/AVC 인코더는 원영상과 추정된 영상의 차이 값을 다시 압축하는 방법을 사용한다.[6] 이번 장에서는 Residual 값에 대한 변환과 양자화의 저전력 구조에 대해서 기술한다. H.264/AVC 인코더에서는 Integer Transform을 사용하는데 이는 4x4 화소단위의 직교변

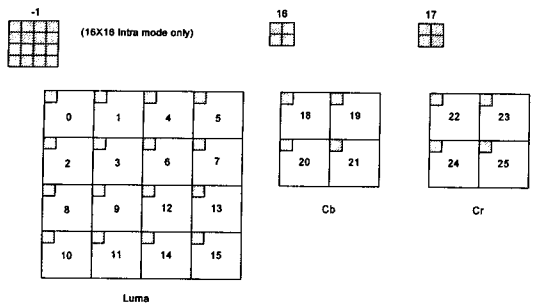


그림 7. 매크로블록 내 오차 블록의 스캔순서
Fig. 7. Scan order of residual block in macroblock.

환이다.[7] Integer Transform은 정수변환으로써 3가지의 변환이 있고 매크로블록 안에서 다음 그림 7과 같은 순서로 변환이 이루어진다.

그림 7에서 보듯이 변환의 순서는 -1부터 25까지 처리된다. 그림 7의 0부터 15와 18부터 25의 4x4 Residual 변환은 식 (14)와 같이 나타낼 수 있다.

$$Y = C_f X C_f^T * E_f$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) * \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (14)$$

E_f는 스케일링 계수이며 양자화 블록으로 포함시켜서 양자화 블록에서 연산한다. 양자화 블록에서는 변환에서 온 post-scaling 행렬 E_f를 사용하여 MF를 만들어낸다. 실제 양자화 블록에서는 MF와 qbits를 이용하여 식 (15)와 같이 곱셈과 쉬프트 연산만으로 구현이 가능하다.

$$|Z_{ij}| = (|W_{ij}| . MF + f) \gg qbits \quad (15)$$

W_{ij}는 변환되고 스케일링은 되지 않은 계수이고 qbits는 QP값으로 결정된다. 이 식에서 곱셈 계수 MF는 표 2와 같이 각 위치에 따라 계수 값이 결정된다.

표 2. Multiplication factor(MF)
Table 2. Multiplication factor(MF).

QP	Position (0,0),(2,0),(2,2),(0,2)	Position (1,1),(1,3),(3,1),(3,3)	other
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

이 계수들은 양자화 과정에서 곱셈의 계수로 사용되는데 곱셈기를 사용하면 하드웨어의 면적이 크고 전력 소모가 커진다. 따라서 곱셈기 대신에 쉬프트연산과 덧셈기를 사용하여 구현할 수 있다. 2의 보수형 계수보다는 CSD(Canonic Signed Digit)형의 계수를 사용하여 구현하면 덧셈의 수를 더욱 감소시킬 수 있다.[8] 또한 Common sub-expression sharing(CSS) 방식을 사용하여 덧셈의 수를 더욱 감소시킬 수 있다. 우리는 이 논문에서 CSD와 CSS 방식을 MF 계수를 곱하는데 사용하여 저전력 구조를 제안하였다.

V. 구조 검증 및 구현

5.1 ARM 프로세서

이 절에서는 ARM과 FPGA의 연동과 입출력 화면 및 최종 구현 방법을 기술한다. H.264/AVC Baseline Profile 인코더를 SoC MasterIII의 FPGA 와 ARM 프로세서를 사용하여 구현하였다. 각 블록들에 대한 구현은 다음 그림 8과 같이 FPGA와 ARM 프로세서로 역할을 구분하여 구현 하였다. 구현에 사용된 ARM 프로세서는 그림 9의 ARM926-EJS을 사용하였다.

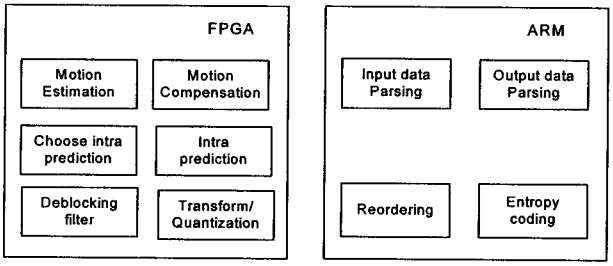


그림 8. FPGA와 ARM 프로세서의 역할
Fig. 8. A part of FPGA and ARM processor.

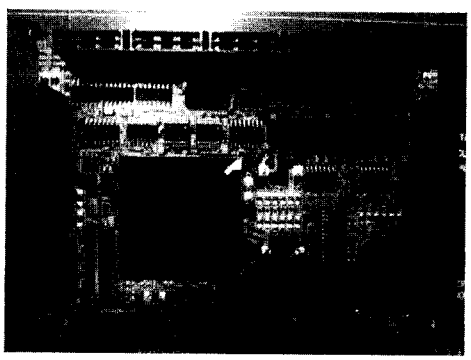


그림 9. ARM926-EJS CoreTile
Fig. 9. ARM926-EJS CoreTile.

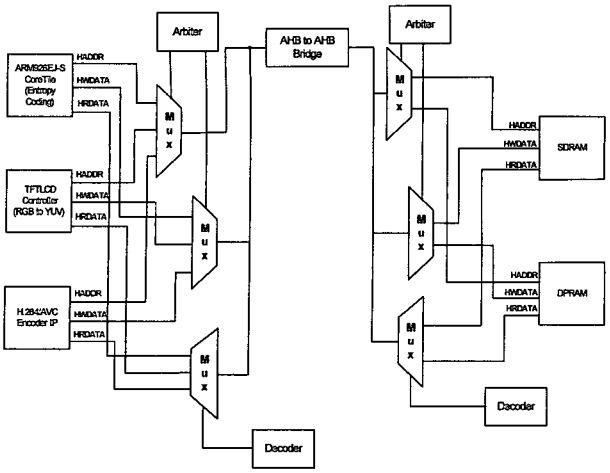


그림 10. 구현에 적용한 AMBA 블록도
Fig. 10. Block diagram of AMBA applied to implementation.

AMBA는 Master, Slave, Arbiter, Decoder로 구성되고 AHB Bus와 APB Bus가 있다. 이를 본 구현 형식에 맞추어 구성하였는데 이는 그림 10과 같다.

그림 10에서 보듯이 H.264/AVC 인코더 IP, TFT-LCD Controller, ARM926-EJS를 MASTER로 설정하였고 SDRAM과 DPRAM은 SLAVE로 설정하였다. Debug 할 때에는 Multi-ICE 포트를 사용하였고 Debug Tool로는 AXD를 사용하였다.

5.2 FPGA 구현

프로세서와 FPGA가 연동되려면 FPGA 내에 프로세서가 접근 할 수 있는 내용을 코딩하고 데이터를 주고받을 수 있는 입출력 포트들에 대한 내용도 코딩되어 있어야한다. 프로세서 또한 FPGA에 접근 할 수 있는 준비와 신호를 주고받을 수 있는 준비가 되어 있어야 한다. 연동을 하기위해서 앞 절에서 설명한 AMBA 중 AMBA AHB 2.0 Protocol을 사용하였다. H.264/AVC 인코더를 구현하기 위해서는 많은 Memory가 사용되는데 FPGA Device내에 DPRAM 형태의 Memory를 설계하였고 또한 외부에 SDRAM을 사용하였다. 이 두 종류의 메모리를 공유함으로써 FPGA와 ARM을 연동시켰다. 그림 11은 사용한 FPGA Device와 Memory이다. FPGA Device는 Xilinx사의 Virtex-xc4vlx100을 사용하였다. Virtex-xc4vlx100는 622 Mb/s~10 Gb/s의 처리 능력을 가지고 있으며 본 구현에서는 FPGA 내부 클락으로 25Mhz를 사용하였다. H.264/AVC 인코더 IP와 DPRAM은 직접 설계하여 사용하였고 AMBA AHB 2.0은 SoCbase1.0을 사용하였다.

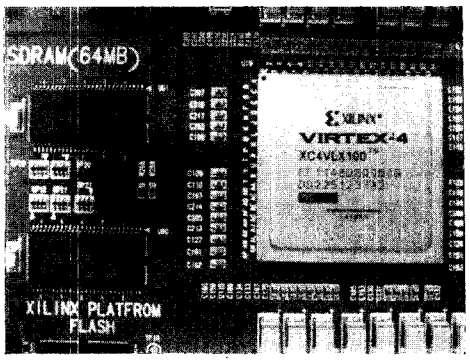


그림 11. FPGA Device와 Memory
Fig. 11. FPGA Device and Memory.

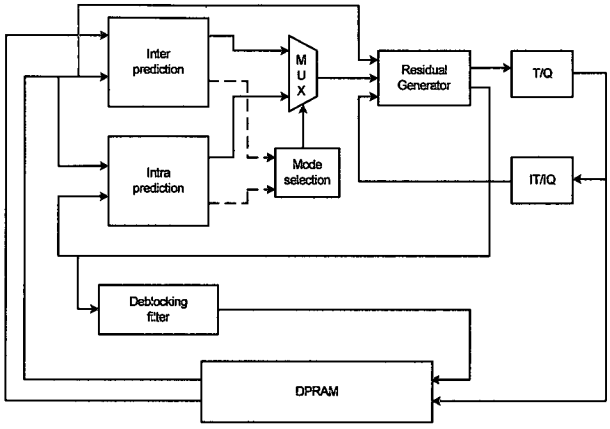


그림 12. FPGA 내의 데이터 흐름
Fig. 12. Data of flow in FPGA.

구현 검증하기 위한 영상은 5 프레임의 Carphone 영상을 사용하였다. 해상도는 QCIF(176x144)로 하였고 luminance 데이터만을 사용하였다. 화면 간 예측 블록의 구현에서는 16x16 블록만을 사용하여 추정하고 보상하도록 구현하였으며 Interpolation 필터는 사용하지 않았다. FPGA내의 만들어진 DPRAM은 입력영상과 출력영상 및 Inter Prediction, Intra Prediction등의 블록에서 발생하는 연산 내용의 저장 공간으로도 활용하였다. FPGA 내의 데이터 흐름은 그림 12와 같이 설계하였다. 그림 12의 실선은 8bit vector이고 점선은 SAD 및 SAE 값으로 13 bit vector이다. Residual Generator 블록은 IP 통합과정에서 전체블록상의 뺄셈연산을 통하여 Residual 데이터를 만들고 이는 T/Q 블록으로 전달되고 덧셈을 수행하여 나온 결과물은 Deblocking 필터블록과 Intra Prediction블록으로 전달한다. 양자화 된 데이터들은 다시 역양자화와 역변환 과정을 거쳐 화면 내 예측에 참조블록으로 사용되기 위해 Residual Generator 블록을 거치고 이 데이터는 Deblocking 필터를 거쳐 화면 간 예측의 참조 영상으로 사용된다. ARM 프로세서

그림 13. TFT- LCD 출력영상
Fig. 13. Output image of TFT- LCD.

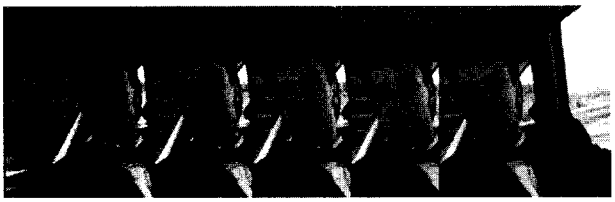


그림 14. 5 프레임의 출력영상
Fig. 14. Output image of 5 frame.

가 메모리에 입력 데이터를 분석하여 DPRAM 일부에 저장하고 H.264/AVC 인코더 IP를 내장하고 있는 FPGA에 신호를 보내면 FPGA가 연산을 시작하고 연산 과정의 데이터는 DPRAM의 다른 영역에 저장되고 ARM 프로세서에 인터럽트 신호를 보내면 AMBA BUS를 통해 내부 DPRAM에 저장된 연산 결과는 SDRAM으로 다시 저장되는데 이 과정을 ARM 프로세서가 수행하고 엔트로피코딩 후 저장한다. 최종 출력으로 저장된 데이터들은 Memory Dump 과정을 통하여 binary로 PC에 저장되고 이 데이터들을 PC에서 확인 할 수 있다. 또한 Platform 보드의 TFT- LCD를 통하여 출력을 가시적으로 확인 할 수 있는데 TFT- LCD에 영상을 보내는 과정도 ARM 프로세서가 수행하게 된다. TFT- LCD화면은 그림 13과 같다.

그림 14는 엔트로피부호화를 거치지 않은 5프레임 출력 영상이며 약간의 열화를 확인 할 수 있다. 최종적으로 구현된 FPGA의 면적은 AMBA AHB 2.0이 175,646gate를 차지하였고 H.264/AVC 인코더 IP가 435,540gate를 차지하였다.

VI. 결 론

H.264코덱이 사용되는 휴대용 멀티미디어의 수요가 증가하면서 제한된 전력으로 복잡한 복호화를 효율적으로 수행하기 위한 저전력 설계연구의 필요성이 커지고 있다. 이 논문에서는 H.264/AVC 인코더의 각 서브 블록에 대한 저전력 구조를 제안하였으며 제안된 구조의 구현 및 검증을 위하여 Verilog-HDL을 사용하였고, SoCMASTER3의 FPGA(Virtex4-lx100)디바이스를 타겟으로 합성을 수행하였으며 ARM926-EJS를 사용하였다. 하드웨어의 효율성을 높이기 위해 각 서브IP의 저전력구조를 고안하는데 중점을 두었으며, 그 결과 각각의 IP블록에서 이용되던 기존구조와 비교하였을 때, Inter/Intra prediction 블록에서는 60.2%, Deblocking 필터블록에서는 44.3%의 면적감소효과를 확인하였다. 그리고 해상도가 QCIF(176x144)인 5 프레임의 Carphone 영상을 입력하여 제안된 구조를 수행한 결과를 TFT-LCD화면으로 확인을 하였다. 이러한 제안구조의 검증을 통하여 제안된 저전력 서브 IP 구조들은 제한된 전력을 필요로 하는 단일 칩 H.264/AVC 인코더에 충분히 사용될 수 있음을 보였다.

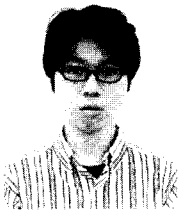
참 고 문 헌

[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] S. Y. Yap and J. V. McCanny, "A VLSI Architecture for Variable Block Size Video Motion Estimation", IEEE Trans. on Circuits and Systems II: Express Briefs, Vol. 51, issue 7, pp. 384-389, July 2004.
[3] 장영범, 오세만, 김비철, 유현중, "H.264 움직임 추정을 위한 효율적인 SAD 프로세서" 대한전자공학회논문지, 제44권 SP편, 제2호, 74-81쪽, 2007년 3월
[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, July 2003.
[5] 장영범, 오세만, 박진수, 한규훈, 김수홍, "H.264용 Deblocking 필터의 저전력 구조", 대한전자공학회 논문지, 제43권 SP편, 제3호, 92-99쪽, 2006년 5월
[6] lain E,G Richardson "H.264 and MPEG-4 Video Compression, Video Codig for Next-Generation Multimedia", WILEY, 2003
[7] 角野 眞也 ; 菊池 義活 ; 鈴木 輝彦 ; 정제창 (역), "H.264 TEXTBOOK", 2005, 서울, 홍릉과학출판사
[8] 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, pp677-688, Oct. 1996.

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>



최 동 규(학생회원)
2007년 상명대학교 정보통신 공학과 졸업(공학사)
2007년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>

한 재 웅(학생회원)
2007년 상명대학교 정보통신 공학과 졸업(공학사)
2007년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>

김 도 한(학생회원)
2006년 상명대학교 정보통신 공학과 졸업(공학사)
2006년~2008년 상명대학교 대학원 컴퓨터정보통신 공학과 졸업(공학석사)
<주관심분야 : 통신신호처리, SoC 설계>

김 비 철(학생회원)
2006년 상명대학교 정보통신 공학과 졸업(공학사)
2006년~2008년 상명대학교 대학원 컴퓨터정보통신 공학과 졸업(공학석사)
<주관심분야 : 통신신호처리, SoC 설계>

박 진 수(학생회원)
2006년 상명대학교 정보통신 공학과 졸업(공학사)
2006년~2008년 상명대학교 대학원 컴퓨터정보통신 공학과 졸업(공학석사)
<주관심분야 : 통신신호처리, SoC 설계>

한 규 훈(학생회원)
2006년 상명대학교 정보통신 공학과 졸업(공학사)
2006년~2008년 상명대학교 대학원 컴퓨터정보통신 공학과 졸업(공학석사)
<주관심분야 : 통신신호처리, SoC 설계>

허 은 성(학생회원)
2006년 상명대학교 정보통신 공학과 졸업(공학사)
2006년~2008년 상명대학교 대학원 컴퓨터정보통신 공학과 졸업(공학석사)
<주관심분야 : 통신신호처리, SoC 설계>