한국산학기술학회논문지
Vol. 10, No. 12, pp. 3653-3659, 2009

# H.264/AVC 베이스라인 프로파일 디코더의 효율적인 인터예측 하드웨어 구조 설계

김선철[1], 류광기[1*]
[1]한밭대학교 정보통신전문대학원

# An Efficient Inter-Prediction Hardware Architecture Design for the H.264/AVC Baseline Profile Decoder

**Xianzhe Jin[1] and Kwangki Ryoo[1*]**
[1]**Graduate School of Information and Communications, Hanbat National University**

**요 약** 본 논문에서는 H.264/AVC 베이스라인 프로파일 디코더 설계에서 병목현상을 일으키는 주요 부분인 인터 예측 성능 개선을 위한 효율적인 하드웨어 구조를 제안한다. H.264/AVC 디코더는 다양한 블록 모드를 지원하지만 레퍼런스 소프트웨어에서는 중복 픽셀에 대해 제거 하지 않고 항상 4x4 블록에 대하여 최소 4x4, 최대 9x9 참조 블록을 패치한다. 기존의 Nova에서는 이를 해결하기 위하여 8x8 블록 모드와 4x4 블록 모드를 고려하였다. 블록 모드가 8x8 사이즈보다 크거나 같을 경우 여러 8x8 블록으로 나누어서 그에 대한 13x13 레퍼런스 블록을 패치 하고 8x8 블록 보다 작을 경우 여러 개의 4x4 블록으로 나누어 그에 대한 9x9 레퍼런스 블록을 패치하여 중복픽셀을 제거함으로써 사이클 수를 감소시켜 레퍼런스 소프트웨어에 비해 최대 41.5%, 최소 28.2%의 성능을 향상시켰다. 본 논문에서는 성능 향상을 위하여 8x8과 4x4 블록 모드 뿐만 아니라 다양한 레퍼런스 블록 패치를 진행하여 중복픽셀을 제거하고 메모리 패치 사이클 수를 줄여 기존 설계에 비해 최대 18.6%의 참조 블록 패치 사이클 수를 감소시켰다.

**Abstract** Inter-prediction is always the main bottleneck in H.264/AVC baseline profile. This paper describes an efficient inter-prediction hardware architecture design. H.264/AVC decoder supports various block types but reference software considers only the 4x4 block when the reference block is being fetched. This causes duplicated pixels which needs extra fetch cycles. In order to eliminate some of the duplicated pixels, the 8x8 and 4x4 blocks were considered in the previous design. If the block size is larger than or equal to the 8x8 block, it will be decomposed into several 8x8 blocks and if the block size is smaller than the 8x8 block it will be decomposed into several 4x4 blocks. Comparing with the reference software, the maximum and minimum cycle reduction of the previous design are 41.5% and 28.2% respectively. For further reduction of the fetch cycles, the various block types are considered in this paper. As a result, the maximum cycle reduction is 18.6% comparing with the previous design.

**Key Words :** Inter-prediction, Motion compensation, H.264/AVC decoder, Baseline profile

## 1. 서론

H.264/AVC 비디오 압축 표준[1]은 JVT에 의해 개발되어 압축률이 가장 우수한 비디오 압축 표준으로 기존의 압축 표준과 비교하여 MPEG-2의 2배, MPEG-4 ASP의 1.5배의 압축 성능의 향상을 보인다[2]. 그 이유는 가변 블록 크기의 움직임 예측 및 보상, 정수 변환 및 양자화, 1/4 화소 단위의 화면간 예측, 다중 참조 프레임 기반의 움직임 예측 및 보상, 방향성을 띈 화면 내 예측 부호화, 디블로킹 필터, 엔트로피 부호화 등의 다양한 기술이 향상되었기 때문이다. 그러나 인터 예측 및 보상은 H.264/AVC 디코더 가운데 병목현상을 일으키는 가장 주

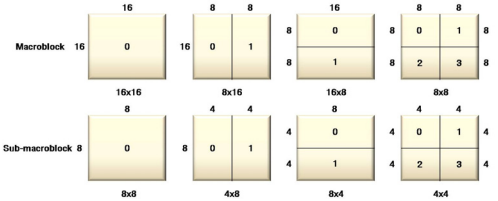
본 연구는 IDEC의 CAD tool 지원, 중소기업청의 산학협력실 지원사업 및 ETRI 시스템반도체산업진흥센터의 IT SoC 핵심설계인력양성사업의 연구결과임.
*교신저자 : 류광기(kkryoo@hanbat.ac.kr)
접수일 09년 09월 20일 수정일 (1차 09년 10월 15일, 2차 09년 11월 12일) 게재확정일 09년 12월 16일

요한 부분이다. 인터 예측의 성능 향상을 위하여 픽셀 보간 블록에 대한 성능 향상[3]에 초점을 맞출 수 있고 메모리 액세스 구조에 대한 성능 향상[4]에 초점을 맞추어 전체 인터 예측 블록에 대한 성능을 향상시킬 수 있다.

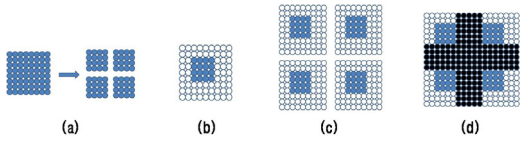
본 논문에서는 인터 예측 기법 중에서 참조 블록을 가져오기 위한 메모리 패치 사이클 수를 감소시키는 새로운 설계 기법을 제시한다.

## 2. 기존의 인터 예측 알고리즘

H.264/AVC 디코더는 그림 1과 같이 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4 등 다양한 블록 타입을 지원한다. 각 매크로 블록의 휘도 성분은 4가지 방법으로 분할되고 8x8 모드가 선택되면 매크로블록의 네 개 8x8 서브 매크로블록은 다시 4가지 방법으로 분할된다. 레퍼런스 소프트웨어[5]에서는 인터 예측 과정에서 블록 타입에 무관하게 중복 픽셀에 대해 제거하지 않고 항상 4x4 블록에 대한 참조 블록을 패치하며 참조 블록 크기는 최소 4x4에서 최대 9x9까지 다양하다. 그림 2는 레퍼런스 소프트웨어에서 블록 타입이 8x8이고 참조 블록 크기가 9x9인 경우를 처리하는 과정을 보인 것이다. 블록 타입이 8x8일 경우 4개의 4x4블록으로 나뉘며 하나의 4x4 블록에 대한 참조 블록 사이즈는 9x9이다. 4개의 4x4 블록에 대한 9x9 참조 블록(그림 2의 c)을 메모리에서 패치하면 그림 2의 d와 같이 크로스로 표시된 중복 픽셀까지도 패치한다. 이때 4개의 4x4 블록에 대한 전체 참조 블록 픽셀 수는 4x9x9=324개이다. 블록 타입이 8x8인 블록을 분해하지 않을 경우 참조 픽셀 크기는 13x13이고 전체 참조 블록 픽셀 수는 169개이므로 8x8 블록을 분해하는 알고리즘에 비해 155개의 중복 픽셀을 줄일 수 있다.




[그림 1] 매크로블록 및 서브 매크로블록 파티션




[그림 2] 8x8블록 타입의 문제점

불필요한 중복 픽셀을 패치하면 추가적인 메모리 패치 사이클이 소요되므로 [6]은 이러한 문제를 해결하기 위하여 8x8 블록 타입과 4x4 블록 타입을 고려하여 설계하였다. 블록 크기가 8x8보다 크거나 같을 경우는 8x8 블록으로 분해, 처리하고 블록 크기가 8x8보다 작을 경우는 4x4 블록으로 분해, 처리하여 일부 중복 픽셀을 줄여 참조 픽셀 패치 사이클 수를 감소시켰다. 또한, 3단계 메모리 구조를 적용하여 첫 단계는 외부 메모리로서 두 개의 SDRAM으로 구성되어 하나가 참조 메모리로 동작할 때 다른 메모리는 디스플레이 메모리로 동작한다. 두 번째 단계는 참조 픽쳐 버퍼로 현재 블록에 대한 참조 블록을 저장하는 임시 저장 공간이다. 기존에는 4x4 블록 타입과 8x8 블록 타입만 고려하기 때문에 9x9 및 13x13 참조 블록을 저장해야 하려면 13x13 크기의 8비트 레지스터가 필요하다. 세 번째 단계의 메모리는 픽셀 보간 과정에서만 동작하며 픽셀 보간 과정에서 계산된 중간 값을 저장한다. 기존 설계에 대한 성능 테스트를 위하여 5개의 테스트 파일을 사용하였다. 표 1은 기존 설계의 참조 픽셀에 대한 패치 사이클 수와 레퍼런스 소스의 참조 픽셀에 대한 패치 사이클 수를 비교한 것이다.

[표 1] 기존설계[6]과 레퍼런스소스[5]의 성능비교

| 테스트 파일 | highway | coastguard | foreman | mother-daughter | trevor |
|---|---|---|---|---|---|
| 프레임 수 | 300 | 300 | 300 | 300 | 150 |
| 인터 MB 수 | 29299 | 29442 | 28920 | 29540 | 14077 |
| 휘도블록 패치사이클 [Cycle][5] | 4257384 | 6183925 | 7529101 | 3811502 | 2298536 |
| 휘도블록 패치사이클 [Cycle][6] | 2954324 | 4019073 | 4407679 | 2734982 | 1547479 |
| MB당 평균 패치 사이클 [Cycle/MB][5] | 145.3 | 210.0 | 260.3 | 129.0 | 163.3 |
| MB당 평균 패치 사이클 [Cycle/MB][6] | 100.8 | 136.5 | 152.4 | 92.6 | 109.9 |
| 패치사이클 감소율[%] | 30.6 | 35.0 | 41.5 | 28.2 | 32.7 |

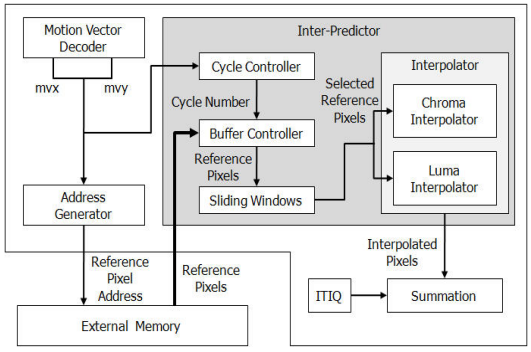
전체 5개의 테스트 파일 중 trevor는 150프레임을 사용하고 나머지는 300프레임을 사용하여 테스트를 진행하였다. 휘도 블록에 대한 패치 사이클 수와 인터 예측에 사용된 매크로블록 수를 측정하여 하나의 매크로블록을 패치하는데 필요한 평균 사이클 수를 계산하였다. 레퍼런스 소스에서 하나의 MB에 대한 평균 패치 사이클(Reffetch)과 [6]에서 하나의 MB에 대한 평균 패치 사이클(Novafetch)을 이용하여 패치 사이클 감소율을 식(1)과 같이 계산하였다.

$$\text{패치사이클 감소율} = \frac{Reffetch - Novafetch}{Reffetch} \times 100\% \quad (1)$$

측정결과 기존 설계는 레퍼런스 소스에 비해 패치 사이클 감소율이 최소 28.2%, 최대 41.5%인 것을 확인하였다. 비록 패치 사이클을 줄여 성능은 향상되었지만 여전히 중복픽셀이 존재하여 추가적으로 중복 픽셀을 제거하여 패치 사이클 수를 줄일 필요가 있다.

## 3. 제안하는 구조

제안한 구조에서는 더 많은 중복 픽셀을 줄여 참조 픽셀 패치 사이클 수를 줄이기 위하여 8x8 블록 타입과 4x4 블록 타입 뿐만 아니라 다양한 블록 타입의 참조 픽셀 패치 사이클을 고려한 구조를 설계하였다. 제안된 구조는 H.264/AVC 베이스라인 프로파일에 속하며 프레임 크기는 176x144이다. 그림 3은 제안한 디자인의 블록도를 나타낸다. 제안된 디자인은 motion vector decoder, address generator, cycle controller, sliding windows, interpolator, summation 블럭과 외부 메모리로 구성된다.

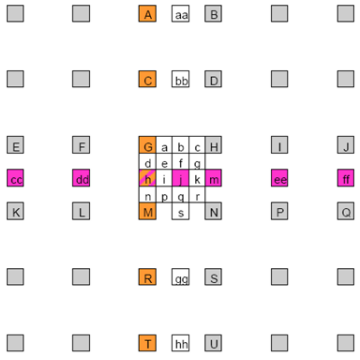



[그림 3] 인터 예측 구조

Motion vector decoder 블록에서는 움직임 벡터 차이(MVD: motion vector difference)를 이용하여 움직임벡터(MV: motion vector)를 계산하고 계산된 움직임 벡터는 cycle controller 블록과 address generator 블록에 전송한다. Cycle controller는 계산된 움직임 벡터와 블록 타입에 근거하여 메모리 패치 사이클을 계산하고 address generator 블록은 수직, 수평 움직임 벡터를 이용하여 참조 픽셀 주소를 계산한다. Buffer controller 블록은 계산된 사이클에 근거하여 외부 메모리에서 참조 픽셀을 패치 하고 sliding windows는 픽셀 보간에 필요한 수직, 수평 픽셀을 선택하고 interpolator 블록에서는 휘도 블록과 색차 블록에 대한 픽셀 보간을 진행한다.

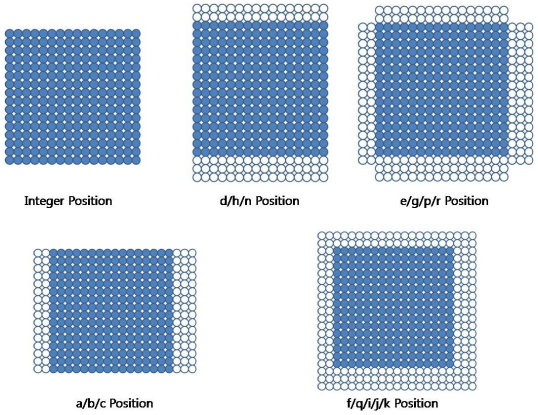
### 3.1 Cycle controller

MPEG-4의 움직임 벡터는 휘도 성분에 대해 1/2의 해상도를 가지는 반면 H.264/AVC의 움직임 벡터는 휘도 성분에 대해 1/4 샘플의 해상도를 가지며 cycle controller는 motion vector decoder에서 디코딩된 움직임 벡터와 블록 타입에 의하여 참조 픽셀 패치 사이클을 계산한다. 그림 4는 휘도 블록에서의 정수 픽셀, 1/2 픽셀, 1/4 픽셀 위치를 나타낸다. 정수 픽셀 사이에 1/2 픽셀, 1/4 픽셀 위치인 a~r이 존재하며 위치는 각각 움직임 벡터의 하위 두 비트에 의해 결정되고 위치에 따른 픽셀 보간 순서와 필요한 참조 픽셀 위치가 다르므로 필요한 참조 픽셀 크기와 모양도 다르게 패치하도록 설계하였다. 따라서 서로 다른 참조 픽셀에 대한 패치 사이클 수도 다르게 계산된다.




[그림 4] 휘도 블록 픽셀 위치

그림 5는 16x16 블록의 경우 각 픽셀 위치에서 패치할 참조 블록의 크기와 모양을 보인다.

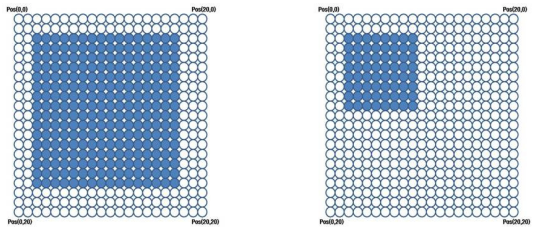



[그림 5] 16x16블록 타입의 참조 블록

정수 위치이고 수평 움직임 벡터의 2번 비트와 3번 비트의 값이 모두 0일 경우 하나의 수평라인을 패치하는데 필요한 사이클 수는 32비트 데이터 폭을 갖는 메모리를 기준으로 4 사이클이 필요하고 그렇지 않을 경우 5 사이클이 필요하다. 따라서 16x16 크기의 참조 픽셀을 패치하는데 64 혹은 80 사이클이 필요하다. 마찬가지로 d/h/n 위치에서는 84 혹은 106 사이클, e/g/p/r 위치에서는 122 사이클, a/b/c 위치에서는 96 사이클, f/q/i/j/k 위치에서는 126 사이클이 필요하다. 마찬가지 방법으로 16x16 블록 타입이 아닌 다른 블록 타입에 대해서도 동일한 방법으로 계산될 수 있다. 계산된 사이클 수는 buffer controller에 전송된다.

### 3.2 Buffer controller

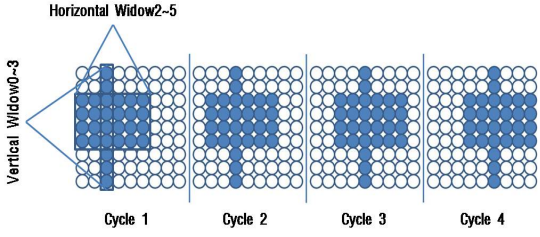
Buffer controller는 외부 메모리로부터 cycle controller에서 계산된 사이클 수와 움직임 벡터에 의하여 참조 픽셀을 패치하여 버퍼에 저장한다. 동일한 블록 타입일 경우 정수 위치 사이에 존재하는 1/2픽셀, 1/4픽셀 위치에 따라서 패치할 참조 픽셀 모양이 다르므로 저장될 버퍼의 위치가 다르게 설계되었다. 그림 6은 블록 타입이 16x16, 8x8일때 참조 픽셀이 버퍼에 저장되는 위치를 보인다. 그림 6은 수평 움직임 벡터의 하위 두 비트가 0일 경우인데, 즉 픽셀 위치가 정수 픽셀일 때 두 블록에 대한 픽셀 보간을 위하여 16x16, 8x8 참조 블록이 필요하며 16x16일 경우 버퍼의 위치 (2,2)로부터 (17,17)까지 위치에 저장되고 8x8일 경우 버퍼의 위치 (2,2)로부터 (9,9)까지 위치에 저장된다. 버퍼는 다양한 블록 타입에 대한 참조 블록을 저장해야 하므로 최소 4x4, 최대 16x16 블록 타입에 대한 참조 블록 픽셀을 저장하도록 21x21개의 8비트 레지스터로 구성된다.




[그림 6] 16x16, 8x8 블록 타입의 참조 픽셀이 버퍼에 저장되는 위치

### 3.3 Sliding windows

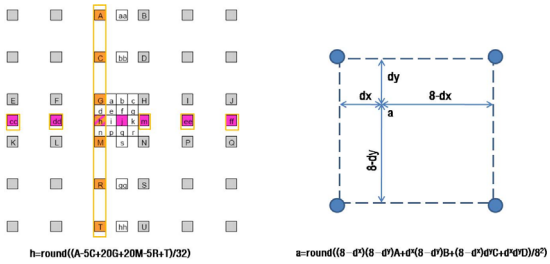
Buffer controller에 저장된 참조 픽셀은 픽셀 보간에 사용되며 픽셀 보간은 한 번에 4x4 블록에 대해서 진행한다. 각 블록 타입에 존재하는 4x4 블록에 필요한 참조 블록을 임시 저장 공간에 저장하며 그 공간이 sliding windows이다. Sliding windows는 9개의 수평 윈도우와 4개의 수직 윈도우로 구성된다. 하나의 블록 타입에 한 개 또는 여러 개의 4x4 블록이 존재하며 sliding windows의 참조 픽셀의 처리 순서는 지그재그 형태를 가진다. 16x16 블록 타입일 경우, 4x4 블록에 대한 처리 순서는 0, 1, 4, 5, 2, 3, 6, 7, 8, 9, 12, 13, 10, 11, 14, 15이다. 그림 7은 픽셀 보간이 4 사이클이 필요할 때 sliding windows의 처리 과정을 보여 준다. 픽셀 보간은 수직, 수평 위치에서 동시 진행되므로 픽셀 보간이 4 사이클일 경우 4개의 수평 슬라이딩 윈도우와 4개의 수직 슬라이딩 윈도우에 동시에 참조 블록을 저장한다. 첫 번째 사이클에서는 4개의 수직 윈도우와 4개의 수평 윈도우가 첫 번째 열의 4개 픽셀을 위한 픽셀 보간을 위하여 선택되고 픽셀 보간 결과 값은 임시 저장 공간에 저장된다. 두 번째 사이클에서는 수평 윈도우와 수직 윈도우가 동시에 오른 쪽으로 한 열을 시프트하는데 시프트하여 선택된 참조 픽셀은 두 번째 열의 4개 픽셀을 보간하는데 사용된다. 이러한 순서로 4개 열의 픽셀 위치에 대한 픽셀 보간이 끝나면 슬라이딩 윈도우는 지그재그 처리 순서에 따라 다음 4x4 블록에 대한 참조 픽셀 위치로 이동한다. 픽셀 위치에 따라 픽셀 보간에 필요한 사이클 수가 다르며 4x4 블록에 대하여 계산을 하며 j/f/q 위치일 때 5 사이클, i/k 위치일 때 8 사이클, 나머지 경우는 4 사이클이 필요하다.




[그림 7] 픽셀 보간이 4 사이클인 슬라이딩윈도우

### 3.4 Interpolator

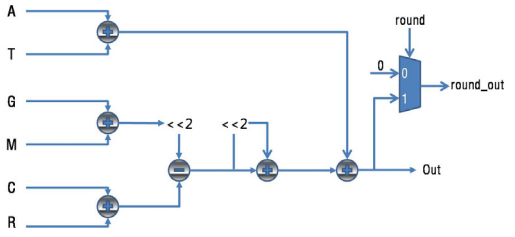
레퍼런스 소스에서 제안한 픽셀 보간 구조는 2-D 구조인데 본 논문에서는 2-D 구조의 FIR interpolator를 1-D 구조인 수직 FIR interpolator와 수평 FIR interpolator로 분해하였다. 픽셀 보간에 필요한 참조 픽셀은 sliding windows에 의해 선택된 픽셀을 사용한다. 픽셀 보간은 휘도 블록 보간과 색차 블록 보간으로 나뉘며 그림 8은 표준에서 정의 된 픽셀 보간 방법을 보인다.




(a) (b)

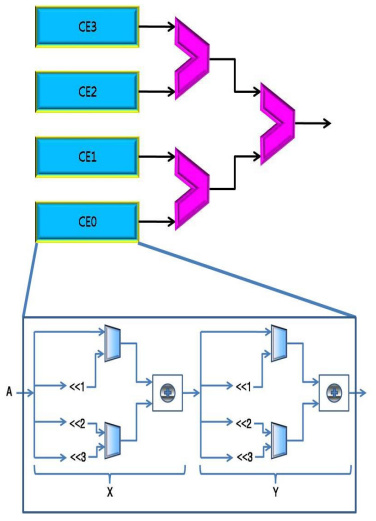
[그림 8] 휘도 블록 및 색차 블록의 계산 공식

그림 (a)는 휘도 블록에 대한 1/2 위치 픽셀 보간 공식이고 그림 (b)는 색차 블록에 대한 픽셀 보간 공식이다. 또한 H.264/AVC에서 지원하는 휘도 블록에 대한 1/4 픽셀 위치의 샘플을 생성하기 위하여 선형 보간 필터가 필요하다. 그림 9는 휘도 블록에 대한 interpolator 하드웨어 구조이다. 휘도 블록의 한 픽셀에 대한 픽셀 보간을 진행하는데 필요한 수평 휘도 interpolator는 9개, 수직 휘도 interpolator는 4개이며 수직, 수평 위치에서 동시에 픽셀 보간을 진행한다. 동일 사이클 동안 수직위치에서 4개의 픽셀 위치에 대해 계산한다. 정수 위치일 경우 필터링을 수행하지 않고 직접 참조 픽셀을 복사한다. 픽셀 위치에 따라 선형 필터가 필요한 경우도 존재하며 이러한 선형 필터는 두 번째 사이클부터 처리하고 첫 번째 사이클에서 생성된 픽셀 샘플을 사용한다.




[그림 9] 휘도 블록 interpolator

색차 픽셀에 대한 보간은 1/8 샘플 단위까지 정확하게 처리되는 선형 필터로 구성된다. 휘도 블록과 비교할 때 색차 블록에 대한 참조 픽셀은 더 적은 저장 공간이 필요하므로 휘도 블록과 동일한 버퍼를 사용하게 된다. 그림 10은 색차 블록 픽셀 보간 구조를 보인다. 색차 블록 interpolator는 그림과 같은 구조를 가진다.

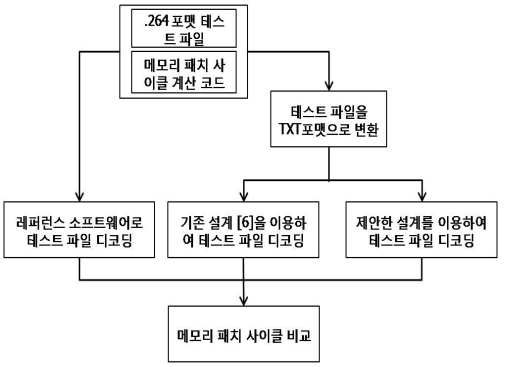



[그림 10] 색차 블록 interpolator

곱하기 연산은 두 단계의 시프트와 더하기 연산으로 구성되며 한 사이클 동안에 처리되는 양을 증가하기 위하여 4개의 interpolator가 동시에 실행되는 병렬구조이다. 한 사이클 내에 2x2 블록에 대한 픽셀 보간을 진행하므로 4x4 블록을 보간 하는데 4 사이클이 소요되고 하나의 매크로 블록에 존재하는 Cb, Cr 색차 블록에 대한 픽셀 보간은 32 사이클이 소요된다.

## 4. 실험 환경 및 절차

설계에 대한 성능을 비교하기 위하여 그림 11과 같은 실험 절차를 거쳤다. 실험에서 사용된 입력 파일은 .264 포맷으로 H.264/AVC 압축표준에 의해 이미 압축된 바이너리 형태의 파일이다. 사용된 레퍼런스 소프트웨어는 9.2버전이고 .264포맷인 파일을 입력 파일로 사용하였다. Verilog HDL로 설계된 기존 설계[6]와 제안한 구조는 .264포맷이 아닌 텍스트 파일을 테스트벡터로 사용해야 하므로 .264포맷을 TXT포맷으로 변환하여 입력 파일로 사용하였다. 레퍼런스 소프트웨어와 각 디자인의 메모리 패치 사이클 수를 계산하는 코드를 추가하였고 레퍼런스 소프트웨어는 비주얼스튜디오, 기존 설계[6]과 제안한 설계는 ModelSim을 이용하여 동작 상태를 검증하고 메모리 패치 사이클 수를 측정하였다.




**[그림 11]** 실험 절차

## 5. 성능 비교

기존 설계[6]는 8x8 블록과 4x4 블록에 대한 참조 블록을 고려하였고 제안된 설계에서는 H.264/AVC 압축 표준에서 제안한 다양한 블록 타입에 대한 참조 블록을 고려하여 설계하였다.

표 2는 기존 설계와 제안된 설계의 참조 블록 패치 사이클에 대한 성능을 비교한 표이다. trevor는 150 프레임을, 나머지 4개 테스트 파일은 300프레임을 사용하는 전체 5개의 입력 파일을 사용하여 실험을 진행하여 참조 블록 패치 사이클 수의 감소 결과를 확인하였다.

**[표 2]** 기존 설계[6]과 제안된 설계의 성능비교

| 테스트 파일 | highway | coastguard | foreman | mother-daughter | trevor |
|---|---|---|---|---|---|
| 프레임 수 | 300 | 300 | 300 | 300 | 150 |
| 인터 MB 수 | 29299 | 29442 | 28920 | 29540 | 14077 |
| 휘도블록 패치사이클 [Cycle][6] | 2954324 | 4019073 | 4407679 | 2734982 | 1547479 |
| 제안된 설계의 휘도 블록 패치사이클 [Cycle] | 2555928 | 3327067 | 3587541 | 2467449 | 1395812 |
| MB당 평균패치 사이클 [Cycle/MB][6] | 100.8 | 136.5 | 152.4 | 92.6 | 109.9 |
| 제안된 설계의 MB당 평균 패치사이클 [Cycle/MB] | 87.2 | 113.0 | 124.1 | 83.5 | 99.1 |
| 패치사이클 감소율[%] | 13.5 | 17.3 | 18.6 | 9.8 | 9.8 |

기존 설계[6]에서 하나의 MB에 대한 평균 패치 사이클(Novafetch)과 제안된 설계에서 하나의 MB에 대한 평균 패치 사이클(Profetch)을 이용하여 패치 사이클 감소율을 식(2)로 계산하였다.

$$\text{패치사이클감소율} = \frac{Novafetch - Proetch}{Novafetch} \times 100\% \quad (2)$$

계산 결과 제안하는 설계는 기존 설계에 비해 최소 9.8%, 최대 18.6%의 참조 블록 패치 사이클 수의 감소를 가져 왔다.

## 6. 결론

본 논문에서는 H.264/AVC 베이스라인 프로파일을 이용한 디코더의 인터 예측 블록에 대한 성능을 향상시키기 위한 하드웨어 구조를 제안하였다. 기존 설계는 8x8 블록과 4x4 블록에 대한 참조 픽셀을 패치 하는 것과 달리 제안한 설계에서는 다양한 블록 타입에 대해 고려하고 그에 대한 참조 픽셀 메모리 패치 사이클 수를 줄이는 하드웨어 구조를 제안하였다. 실험 결과 제안된 구조는 기존 설계와 비교하여 평균 13.8%의 성능을 향상시켰다.

## 참고문헌

[1] Joint Video Team (JVT) of ISO/IEC & ITU-T VCEG, H.264 : Advanced video coding for generic audiovisual services, March 2005.

[2] 김대연, 임성창, 이영렬, “H.264/AVC의 화면 내 예측을 위한 새로운 고속 모드 결정 방법”, 한국방송공학회학술대회, pp. 117-120, 2006년 11월.

[3] Sheng-Zen Wang, Ting-An Lin, Tsu-Ming Liu, and Chen-Yi Lee, “A new motion compensation design for H.264/AVC decoder”, International Symposium on Circuits and Systems, May 2005.

[4] Ronggang Wang, Mo Li, Jintao Li, and Yongdong Zhang, “High throughput and low memory access sub-pixel interpolation architecture for H.264/AVC HDTV decoder”, IEEE Transactions on Consumer Electronics, Vol. 51, pp. 1006-1013, August 2005.

[5] JVT reference software JM9.4.

[6] Ke Xu, Nova : a H.264/AVC Baseline Decoder Specification Rev.0.1, April 2008.

**김 선 철(Xianzhe Jin)** **[비회원]**

- 2007년 2월 : 배재대학교 정보통신공학과 공학사
- 2009년 2월 : 한밭대학교 정보통신공학과 공학석사
- 2009년 2월 ~ 현재 : 한밭대학교 정보통신전문대학원 정보통신공학과 박사과정

<관심분야>
SoC 플랫폼 설계, 멀티미디어 코덱 설계

**류 광 기(Kwangki Ryoo)** **[정회원]**

- 1986년 2월 : 한양대학교 전자공학과 공학사
- 1988년 2월 : 한양대학교 전자공학과 공학석사
- 2000년 2월 : 한양대학교 전자공학과 공학박사
- 1991년 4월 ~ 1994년 7월 : 육군사관학교 교수부 전자공학과 전임강사
- 2000년 2월 ~ 2002년 12월 : 한국전자통신연구원 집적회로설계연구부 시스템IC설계팀
- 2003년 1월 ~ 현재 : 한밭대학교 정보통신공학과 부교수

<관심분야>
SoC 플랫폼 설계 및 검증, 하드웨어/소프트웨어 통합설계 및 통합검증, 멀티미디어 코덱 설계