Journal of The Institute of Electronics and Information Engineers Vol.53, NO.4, April 2016 http://dx.doi.org/10.5573/ieie.2016.53.4.106
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2016-53-4-12

VP9 디코더에 대한 행렬 기반의 정수형 역변환 구조

(Integer Inverse Transform Structure Based on Matrix for VP9 Decoder)

이 태 희*, 황 태 호*, 김 병 수*, 김 동 순**

(Tea-Hee Lee©, Tae-Ho Hwang, Byung-Soo Kim, and Dong-Sun Kim)

요 약

본 논문에서는 VP9 디코더에 대한 행렬 기반의 정수형 역변환 구조를 제안한다. 제안하는 구조는 DCT(Discreste Cosine Transform), ADST(Asymmetric Discrete Sine Transform) 그리고 WHT(Walsh-Hadamard Transform)에 대한 알고리즘을 공유하며 버터플라이구조보다 하드웨어 리소스를 줄이고 제어하기 쉬운 하드웨어 구조이다. VP9 구글 모델 내 정수형 역변환은 버터플라이구조 기반의 정수형 역변환 구조를 가진다. 일반적인 버터플라이구조와는 달리 구글모델 내 정수형 역변환은 각 단계마다 라운드 쉬프트 연산기를 가지며, 비대칭 구조의 사인 변환을 포함한다. 따라서 제안하는 구조는 모든 역변환 모드에 대해 행렬계수 값을 근사하고, 이 계수 값을 이용하여 행렬연산 방식을 사용한다. 본 논문의 기술을 사용하면 역변환 알고리즘에 대한 모드별 동작 공유 및 버터플라이구조에 비해 곱셈기 수를 2배가량 감소시킬 수 있다. 그래서 하드웨어 리소스를 효율적으로 관리가 가능해진다.

Abstract

In this paper, we propose an efficient integer inverse transform structure for vp9 decoder. The proposed structure is a hardware structure which is easy to control and requires less hardware resources, and shares algorithms for realizing entire DCT(Discrete Cosine Transform), ADST(Asymmetric Discrete Sine Transform) and WHT(Walsh-Hadamard Transform) in vp9. The integer inverse transform for vp9 google model has a fast structure, named butterfly structure. The integer inverse transform for google C model, unlike universal fast structure, takes a constant rounding shift operator on each stage and includes an asymmetrical sine transform structure. Thus, the proposed structure approximates matrix coefficient values for all transform mode and is used to matrix operation method. With the proposed structure, shared operations for all inverse transform algorithm modes can be possible with reduced number of multipliers compared to the butterfly structure, which in turn manages the hardware resources more efficiently.

Keywords : VP9, IDCT, IADST, Inverse integer transform, Butterfly, Matrix coefficients

Ⅰ. 서 론

변환은 데이터 압축을 위한 일반적인 방식 중 하나이다. 일부 변형구조는 비디오 어플리케이션을 위해 제안되어 왔다[1].

구글은 이전 VP8 기준으로 차세대 오픈-소스 비디오 코덱인 VP9을 개발하기 위해 2011년부터 프로젝트에 착수했다[2]. High Efficiency Video Coding (HEVC)는 UHD 등 고화질 방송에 대응하도록 설계 되었으며, VP8은 WebM에 사용되는 코덱으로 HTML5의 실시간 통신에 필수적인 WebRTC의 의무 지원 비디오 형식이고 VP9은 VP8을 유튜브(YouTube) 등의 서비스에 접목시키면서 발생하는 화질 향상 및 시스템자원의 효율성을 높이고 VP8 대비 동일한 퀄리티에서 50% 가량의 비트레이트(bit-rate) 감소를 목표로 하며 인터넷 비디오 스트리밍에 적합하도록 설계되었다[3].

2014년 유튜브는 VP9 코덱을 활용한 4K 급 콘텐츠를 제공할 것이라고 밝혔으며, 이는 고해상도 동영상 스트리밍 서비스가 갖는 최대 특징인 네트워크 속도 저하 없이

* 정회원, ** 평생회원, 전자부품연구원
(Korea Electronics Technology Institute)
© Corresponding Author (E-mail : bluestarth06@keti.re.kr)
Received ; January 14, 2016 Revised ; March 5, 2016
Accepted ; March 30, 2016

일정한 영상을 제공한다.

2012년에 거대한 블록 크기의 IDCT를 지원하기 위해 J.S. Park은 16x16 그리고 32x32 역변환 구조를 제시했다[4]. Shen은 4/8/16/32-포인트 정수형 IDCT를 지원하는 곱셈기 기반의 구조를 제안했다[5]. 그리고 2013년에 Zhu는 파이프라인 역변환 구조를 제안했다[6]. 게다가 근사 DCT 알고리즘 기반으로 Pramod는 2013년 효율적인 상수 행렬 곱셈 방식을 사용함으로써 병렬 1-D 정수형 DCT 구조를 제안했다[7].

본 논문에서는 VP9 디코더 내의 행렬 기반 정수형 역변환 방식의 하드웨어 구조를 제안한다. VP9은 DCT, DST, 그리고 WHT의 3가지 타입의 변환모드를 지원한다. 일반적으로, DCT 유형은 4x4, 8x8, 16x16, 32x32의 정사각형의 2D-DCT를 사용하며 VP9은 추가적으로 4x4, 8x8, 16x16 크기의 ADST알고리즘을 지원한다[8]. VP9은 DST-VI와 WHT알고리즘을 사용하는 4x4 변환을 제외한 모든 변환 크기에 대하여 DCT-II와 DST-II알고리즘을 적용하며 정수로 근사하여 사용한다. 게다가, VP9의 ADST알고리즘은 특별한 인트라예측모드와 조합할 수 있는 새로운 변환 방식이다. True Motion 모드와 일부 대각선인트라예측모드와 같이 양쪽 가장자리를 예측하는 인트라예측모드는 가로와 세로방향으로 1-D ADST를 사용한다[8]. 일반적으로 IDCT는 대칭구조를 가지며 IADST는 비대칭구조를 가진다.

본 논문의 기술을 사용하면 역변환 알고리즘 모드별 동작 공유 및 버터플라이구조에 비해 곱셈기를 2배로 감소시킬 수 있다. 그래서 하드웨어 리소스를 효율적으로 관리가 가능해지는 장점이 있다.

본 논문의 구성은 다음과 같다. Ⅱ는 구글에서 제공하는 VP9 오픈 소스의 역변환 구조에 대한 분석 내용을 다루며, Ⅲ에서는 본 논문에서 제안하는 역변환 구조에 대해 설명하고, Ⅳ는 제안하는 구조에 대한 실험 결과를 보여준다. 마지막으로 Ⅴ는 결론을 포함한다.

Ⅱ. VP9에 대한 역변환 구조 검토

그림 1~3은 VP9 구글 소스로부터 분석한 결과를 도식화하여 나타낸다. 그림에서 보듯이, VP9 역변환은 버터플라이구조로 설계되어 있으며 각 단계마다 쉬프트연산을 수행한다. 만약 버터플라이구조로 하드웨어를 설계한다면, 각 단계마다 쉬프트연산 수행을 위하여 충분한 클럭 사이클과 연산기(곱셈기, 덧셈기, 쉬프트기)가 필요하다.

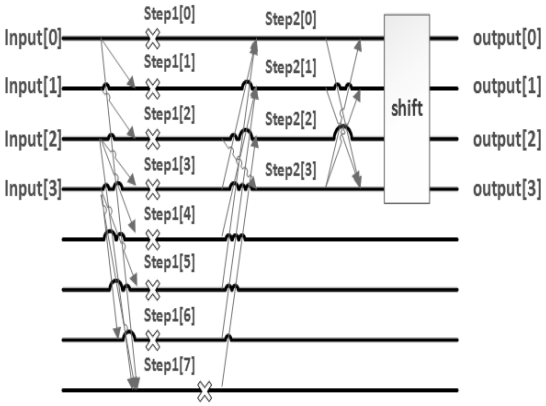

그림 1. VP9에 대한 4x4 IADST 버터플라이 구조
Fig. 1. The 4x4 IADST butterfly structure for vp9.

따라서 본 논문에서는 행렬계수를 사용한 행렬연산 기반의 하드웨어를 설계하여 기존의 버터플라이구조보다 하드웨어 리소스를 줄이고 변환모드의 공유 및 제어하기가 쉬워진다는 이점을 가진다. 행렬구조로 설계하기 위해서는 행렬계수가 필요하며, 이는 구글 VP9 모델의 버터플라이구조부터 계수 값을 근사하여 행렬계수를 구한다.

1. 4x4 IADST 알고리즘

그림 1과 같이 4x4 IADST는 3단계를 가지고 WHT 모드를 제외한 다른 정수형 역변환알고리즘들과는 달리 각 단계마다 쉬프트연산을 하지 않고 오직 마지막 단계에서만 쉬프트연산을 한다. 결국 4x4 IADST는 행렬계수가 근사되지 않고 VP9 구글 모델의 버터플라이구조에 대한 결과 값과 동일한 행렬계수 값을 추출할 수 있다.

식 (1)의 행렬계수는 그림 1로부터 추출한 값이며 그림 1의 입력에서 1단계까지의 4x4 행렬과 1단계에서부터 쉬프트연산 전까지의 4x4 행렬의 곱으로 구해진다.

$$IADST_4 = \begin{bmatrix} 100 & 1 \\ 010 & 1 \\ 001 & 0 \\ 110 & -1 \end{bmatrix} \begin{bmatrix} 5283 & 0 & 15212 & 9929 \\ 9929 & 0 & -5283 & -15212 \\ 13377 & 0 & -13377 & 13377 \\ 0 & 13377 & 0 & 0 \end{bmatrix} \quad (1)$$

$$= \begin{bmatrix} 5283 & 13377 & 15212 & 9929 \\ 9929 & 13377 & -5283 & -15212 \\ 13377 & 0 & -13377 & 13377 \\ 15212 & -13377 & 9929 & -5283 \end{bmatrix}$$

DST-VI 타입의 4x4 IADST 알고리즘은 아래의 공식과 같다. (2)의 공식을 사용하면, 4x4 IADST 계수는 5282, 9929, 13377, 15212의 값을 가진다.

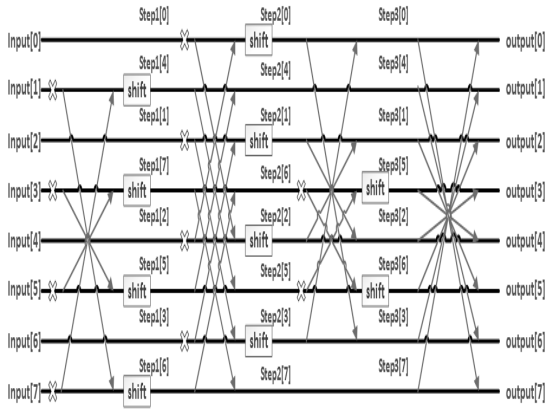

그림 2. VP9에 대한 8x8 IDCT 버터플라이구조
Fig. 2. The 8x8 IDCT butterfly structure for vp9.

$$\left[S_{M-1}\right]_{mn} = x_0\left(\frac{2}{\sqrt[2]{2M-1}} sin\left(\frac{m(2n-1)\pi}{2M-1}\right)\right) \quad (2)$$

$where\ m,n = 1, ..., M-1$

4x4 IADST의 경우엔 4개의 구간을 가지므로 M=5, m=1, 2, 3, 4, n=1, 그리고 x0는 초기 값이며 16384 (2^14)의 값으로 설정한다. 따라서 5283~15212은 Round (16384*sqrt(2)*sin([1~4]pi/9)*2/3)에 의해 계산된다.

2. 8x8 IDCT 알고리즘

8x8 IDCT는 입력에서 출력까지 4단계를 가진다.

최종 행렬계수는 1단계에서 4단계까지 각 단계의 행렬 값으로 근사한다. 1단계에서 0.195는 3196에 16384를 나눈 값이다. 각 단계의 유리수들은 곱해지는 값에 16384를 나눈 값으로 계산된다.

$$step1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0.195 & 0 & 0 & 0 & 0 & 0 & -0.981 \\ 0 & 0 & 0 & -0.556 & 0 & 0.831 & 0 & 0 \\ 0 & 0 & 0 & 0.831 & 0 & 0.556 & 0 & 0 \\ 0 & 0.981 & 0 & 0 & 0 & 0 & 0 & 0.195 \end{bmatrix}$$

$$step2 = \begin{bmatrix} 0.707 & 0 & 0.707 & 0 & 0 & 0 & 0 & 0 \\ 0.707 & 0 & -0.707 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.383 & 0 & -0.924 & 0 & 0 & 0 & 0 \\ 0 & 0.924 & 0 & 0.383 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$step3 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.707 & -0.707 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$step4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$IDCT_8 = \begin{bmatrix} 0.707 & 0.981 & 0.924 & 0.831 & 0.707 & 0.556 & 0.383 & 0.195 \\ 0.707 & 0.831 & 0.383 & -0.195 & -0.707 & -0.981 & -0.924 & -0.556 \\ 0.707 & -0.556 & -0.383 & 0.981 & -0.707 & -0.195 & 0.924 & -0.831 \\ 0.707 & 0.195 & -0.924 & -0.556 & 0.707 & 0.831 & -0.383 & -0.981 \\ 0.707 & -0.195 & -0.924 & 0.556 & 0.707 & -0.831 & -0.383 & 0.981 \\ 0.707 & 0.556 & -0.383 & -0.981 & -0.707 & 0.195 & 0.924 & 0.831 \\ 0.707 & -0.831 & 0.383 & 0.195 & -0.707 & 0.981 & -0.924 & 0.556 \\ 0.707 & -0.981 & 0.924 & -0.831 & 0.707 & -0.556 & 0.383 & -0.195 \end{bmatrix}$$

근사된 행렬계수 값들은 유리수를 가지지만 아래의 행렬 값과 같이 유리수 계수 값에 16384를 곱하고 반올림하여 정수 값으로 변환한다. (4)의 $IDCT_{정수,8}$의 11585의 값은 (3)의 $IDCT_8$의 0.707에 16384의 곱에 반올림한 값이다.

$$IDCT_{정수,8} = \begin{bmatrix} 11585 & 16069 & 15137 & 13623 & 11585 & 9102 & 6270 & 3196 \\ 11585 & 13622 & 6270 & -3197 & -11585 & -16069 & -15137 & -9102 \\ 11585 & -9102 & -6270 & 16069 & -11585 & -3197 & 15137 & -13622 \\ 11585 & 3196 & -15137 & -9102 & 11585 & 13623 & -6270 & -16069 \\ 11585 & -3196 & -15137 & 9102 & 11585 & -13623 & -6270 & 16069 \\ 11585 & 9102 & -6270 & -16069 & -11585 & 3197 & 15137 & 13622 \\ 11585 & -13622 & 6270 & 3197 & -11585 & 16069 & -15137 & 9102 \\ 11585 & -16069 & 15137 & -13623 & 11585 & -9102 & 6270 & -3196 \end{bmatrix} \quad (4)$$

DCT-II 유형의 IDCT는 아래의 수식 (5)와 같다[7]. 아래의 수식을 사용하면 8x8 IDCT 계수 값들은 3196, 13623 등의 값을 가진다.

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \quad (5)$$

$where, k = 0, ..., N-1$

여기서 X_1 = x0*cos(pi/64) + x1*cos(3pi/64) + ... ≒ x0*cos(pi/64), X_2 = x0*cos(2*pi/64) + x1*cos(6*pi/64) + ... ≒ x0*cos(2pi/64), X_3 = x0*cos(3*pi/64) + x1*cos(3*pi/64) + ... ≒ x0*cos(3pi/64) 등으로 표현되고 초기 값 x0는 16384(2^14)의 값으로 설정하며, 그 외의 값은 무시하고 근사한다.

따라서 13623은 Round(cos(12*pi/64)*16384)이고, 3196은 Round(cos(28*pi/64)*16384)로 계산된다. 근사된 정수 행렬계수인 13622, 3197의 값을 IDCT알고리즘을 적용한 13623, 3196의 값으로 대체하여 vp9 모델의 IDCT 버터플라이구조에 대한 최종 근사된 행렬계수 값들을 얻는다.

그림 2로부터 근사되고 DCT알고리즘으로 추출된 값이 적용된 최종 1-D DCT행렬는 아래와 같다. (6)의 $IDCT_{final,8}$의 행렬계수 값을 보면 대칭구조임을 확인할 수 있다. 행 0-7, 1-6, 2-5, 3-4는 부호만 다를 뿐 동일한 값을 가진다. 이 특성을 이용하면 하드웨어 설계 시 곱셈기 사용을 줄일 수 있다.

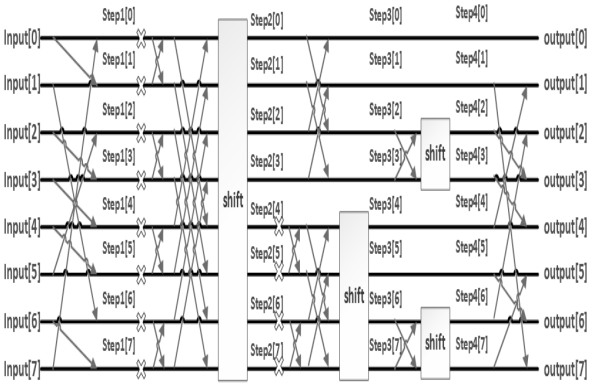

그림 3. vp9에 대한 8x8 IADST 버터플라이 구조
Fig. 3. The 8x8 IADST butterfly structure for vp9.

$$IDCT_{final,8}=\begin{bmatrix}11585 & 16069 & 15137 & 13623 & 11585 & 9102 & 6270 & 3196\\ 11585 & 13623 & 6270 & -3196 & -11585 & -16069 & -15137 & -9102\\ 11585 & -9102 & -6270 & 16069 & -11585 & -3196 & 15137 & -13623\\ 11585 & 3196 & -15137 & -9102 & 11585 & 13623 & -6270 & -16069\\ 11585 & -3196 & -15137 & 9102 & 11585 & -13623 & -6270 & 16069\\ 11585 & 9102 & -6270 & -16069 & -11585 & 3196 & 15137 & 13623\\ 11585 & -13623 & 6270 & 3196 & -11585 & 16069 & -15137 & 9102\\ 11585 & -16069 & 15137 & -13623 & 11585 & -9102 & 6270 & -3196\end{bmatrix} \quad (6)$$

3. 8x8 IADST 알고리즘

IADST 행렬계수 값을 구하는 방식은 IDCT의 방식과 동일한 과정을 거친다.

그림 3으로부터 근사된 행렬계수는 (7)의 $IADST_8$과 같다.

$$IADST_8=\begin{bmatrix}0.098 & 0.290 & 0.471 & 0.634 & 0.773 & 0.882 & 0.957 & 0.995\\ 0.290 & 0.773 & 0.995 & 0.882 & 0.471 & -0.098 & -0.634 & -0.957\\ 0.471 & 0.995 & 0.634 & -0.290 & -0.957 & -0.773 & 0.098 & 0.882\\ 0.634 & 0.882 & -0.290 & -0.995 & -0.098 & 0.957 & 0.471 & -0.773\\ 0.773 & 0.471 & -0.957 & -0.098 & 0.995 & -0.290 & -0.882 & 0.634\\ 0.882 & -0.098 & -0.773 & 0.957 & -0.290 & -0.634 & 0.995 & -0.471\\ 0.957 & -0.634 & 0.098 & 0.471 & -0.882 & 0.995 & -0.773 & 0.290\\ 0.995 & -0.957 & 0.882 & -0.773 & 0.634 & -0.471 & 0.290 & -0.098\end{bmatrix} \quad (7)$$

(7)의 근사된 행렬계수 또한 초기 값을 곱하고 반올림하여 아래의 (8)과 같이 정수 값으로 변환한다.

$$IADST_{정수,8}=\begin{bmatrix}1606 & 4756 & 7723 & 10394 & 12665 & 14449 & 15679 & 16305\\ 4756 & 12666 & 16305 & 14450 & 7723 & -1606 & -10394 & -15679\\ 7723 & 16305 & 10394 & -4756 & -15678 & -12664 & 1606 & 14449\\ 10394 & 14449 & -4756 & -16305 & -1606 & 15678 & 7724 & -12665\\ 12665 & 7724 & -15678 & -1606 & 16305 & -4756 & -14449 & 10394\\ 14449 & -1606 & -12664 & 15678 & -4756 & -10394 & 16305 & -7723\\ 15679 & -10394 & 1606 & 7723 & -14450 & 16305 & -12666 & 4756\\ 16305 & -15679 & 14449 & -12665 & 10394 & -7723 & 4756 & -1606\end{bmatrix} \quad (8)$$

DST-II 유형의 IADST는 아래의 수식 (9)를 따른다.

$$X_k=\sum_{n=0}^{N-1}x_n\sin\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)(k+1)\right] \quad (9)$$

$where, k=0,\ldots,N-1$

여기서 X_1 = x0*sin(pi/64) + x1*sin(3pi/64) + ... ≒ x0*sin(pi/64), X_2 = x0*sin(2*pi/64) + x1*sin(6*pi/64) + ... ≒ x0*sin(2pi/64), X_3 = x0*sin(3*pi/64) + x1*sin(3*pi/64) + ... ≒ x0*sin(3pi/64) 등으로 근사되고 초기 값 x0는 16384(2^14)의 값으로 설정되며 그 외엔 무시한다.

따라서 12665는 Round(sin(18*pi/64)*16384)로 구해지며 $IADST_{정수,8}$의 12666, 14450, 7724등의 값은 ADST 알고리즘 수식을 적용한 결과 값으로 대체한다.

그림 3으로부터 근사된 최종 ADST 행렬계수는 아래의 (10)과 같다.

$$IADST_{final,8}=\begin{bmatrix}1606 & 4756 & 7723 & 10394 & 12665 & 14449 & 15679 & 16305\\ 4756 & 12665 & 16305 & 14449 & 7723 & -1606 & -10394 & -15679\\ 7723 & 16305 & 10394 & -4756 & -15679 & -12665 & 1606 & 14449\\ 10394 & 14449 & -4756 & -16305 & -1606 & 15679 & 7723 & -12665\\ 12665 & 7723 & -15679 & -1606 & 16305 & -4756 & -14449 & 10394\\ 14449 & -1606 & -12665 & 15679 & -4756 & -10394 & 16305 & -7723\\ 15679 & -10394 & 1606 & 7723 & -14449 & 16305 & -12665 & 4756\\ 16305 & -15679 & 14449 & -12665 & 10394 & -7723 & 4756 & -1606\end{bmatrix} \quad (10)$$

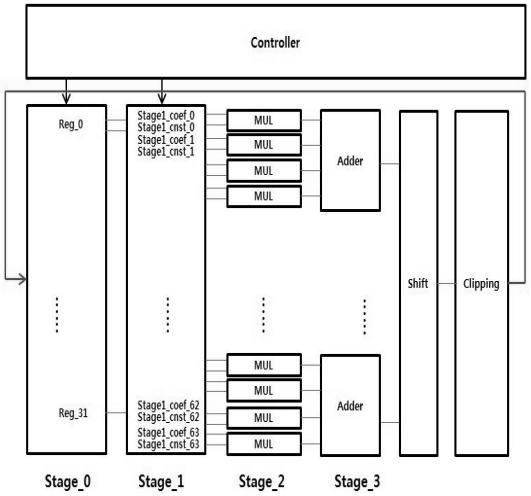

그림 4. 제안하는 행렬 기반의 정수형 역변환 구조
Fig. 4. The proposed matrix based inverse integer transform structuretransform architecture.

Ⅲ. 제안하는 행렬 기반의 역변환 아키텍처

그림 4에서 제안하는 행렬 기반의 정수형 역변환 구조는 전체 4단계를 가지며 파이프라인으로 동작한다.

Stage_1은 제어기로부터 변환모드와 크기정보를 받아서 각 변환모드에 대한 행렬계수 값들을 설정한다. Stage_2는 14bit 행렬계수 값을 가지는 64개의 곱셈기로 구성되며 Stage3은 4입력/1출력의 16개의 덧셈트리로 형성된다.

4x4 크기를 가지는 역변환의 경우엔 제안하는 정수형 역변환 구조는 1차원 트랜스포즈 버퍼 없이 내부에서 1차원(가로 방향) 역변환 결과 데이터를 피드백하여 2차원(세로 방향) 역변환을 수행한다.

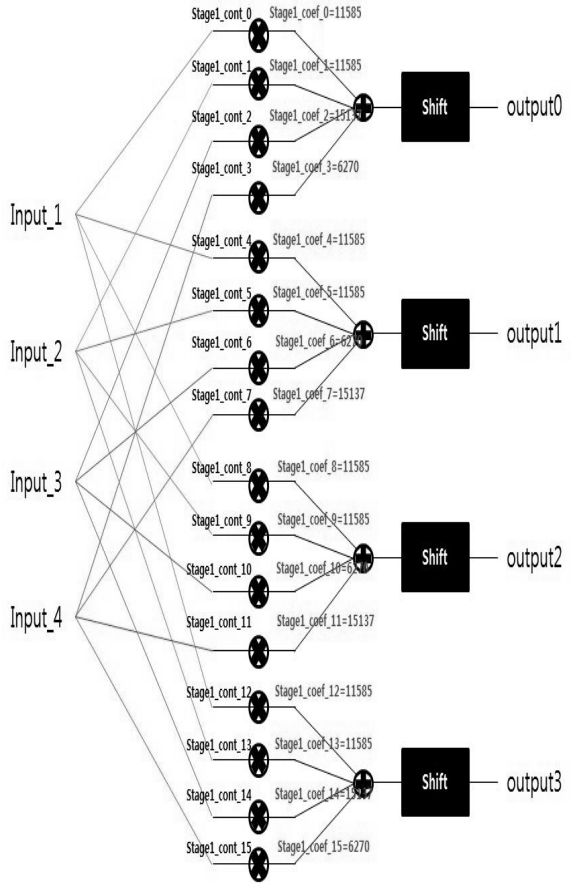

그림 5. 4x4 IDCT 변환
Fig. 5. The 4x4 IDCT transform.

예를 들면, 그림 5는 4입력/4출력에 대한 정수형 IDCT 행렬연산을 보여준다. 일반적인 행렬연산과 같이, 출력0은 입력 값과 4x4 근사된 IDCT 행렬계수 값의 곱의 합으로 계산된다. 즉, 출력0은 ((input_1*11585) + (input_2*11585) + (input_3*15137) + (input_4*6270))>>14(=/2^14)으로 구해진다.

그림 5는 4x4 행렬와 4x1 행렬의 곱으로 4x1 행렬 결과를 출력한다. 4x4 행렬의 값들은 버터플라이구조로부터 근사된 행렬계수이며 stage1_coeff는 변환모드와 크기에 의해 결정된다. 만약 변환모드가 4x4 크기의 ADST면 stage1_coeff0 = 5283, stage1_coeff1 = 13377, ..., stage1_coeff14 = 9929, stage1_coeff15 = -5283으로 IDCT4의 계수 값들이 설정된다.

그리고 쉬프트 연산 통해 16384로 나누는 효과를 가지는데, 이 16384의 값은 초기 값 x0의 값과 같다. 따라서 (6)의 $IADST_{final,8}$과 (10)의 $IDCT_{final,8}$ 의 계수 값으로 설정하였지만, 전체적인 동작을 보았을 때 쉬프트 연산의 효과로 입력되는 데이터에 (3)의 $IADST_8$이나 (7)의 $IDCT_8$을 곱하는 결과를 가져온다.

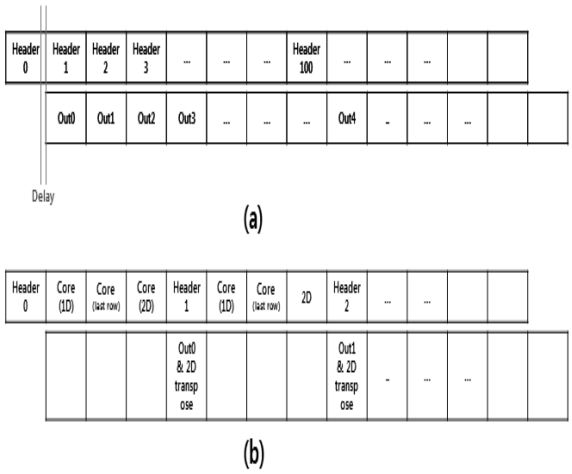

그림 6. 변환에 대한 파이프라인 (a) 4x4 크기 (b) 8x8, 16x16, 32x32 크기
Fig. 6. Pipeline for transformation (a) 4x4 size (b) 8x8, 16x16, 32x32 size.

제안하는 구조는 그림 6과 같이 파이프라인을 형성하여 처리속도를 향상시킨다. 그림 6의 경우 CBF (Coded Block Flag)가 '1'이고 동일한 크기가 입력되었을 경우를 나타낸다. 4x4 크기와 그 외의 변환크기에 따른 파이프라인 형성방식이 다름을 확인할 수 있다.

4x4 크기의 그림 6(a) 경우엔 트랜스포즈 버퍼를 사용하지 않고 내부에서 피드백 받아서 2D 변환을 수행하므로 2~3 사이클 정도의 지연를 주고 다음 해더(Header)를 읽어서 연속적으로 동작을 수행한다. 반면에 그림 6(b)와 같이 2D Core 변환 수행이 완료되고 최종 변환 결과를 출력할 시점에서 해더를 읽어서 다음 변환 Core 동작을 수행하여 연속적으로 동작을 진행한다.

그러나 실제로는 동일한 크기가 연속적으로 들어오는 경우가 드물며, 현재의 변환동작이 32x32 크기이고 다음의 변환동작이 4x4, 8x8, 16x16 크기라면 지연이 필요하다. 만약 지연시간을 주지 않는다면 현재의 변환동작이 수행이 완료되지 않은 상황에서 다음 해더에 대한 변환동작이 수행될 수도 있으며 Core 동작이 겹친다든지 2D 트랜스포즈 버퍼에서 겹쳐 쓰게 되는 경우가 발생하는 등의 오동작이 발생된다. 그러므로 상태머신을 통해 제어함으로써 오동작 문제를 해결한다.

그림 7은 제안하는 구조에서의 상태머신을 나타낸다. 파이프라인을 형성하기 위하여 제어에 필요한 지연신호는 Compare Delay와 Delay Output 상태이다. 여기서 Compare Delay 상태는 현재의 변환크기와 다음에 수행될 변환크기를 비교함으로 지연시간을 준다. 예를 들어, 현재의 크기가 다음의 크기보다 큰 상태에서 지연시간

표 1. 4x4 IDCT에 대한 구글 모델과 RTL의 결과 비교
Table 1. The compare Google model and RTL for 4x4 IDCT.

	INPU (hex/dec)	Google Model (hex/dec)	RTL (hex/dec)
0	FC4A(-950)	FEFF(-257)	FF00(-256)
1	00E4(228)	FCAC(-851)	FCAD(-851)
2	00E4(228)	FCD1(-815)	FCD1(-815)
3	0072(114)	FD03(-765)	FD03(-765)

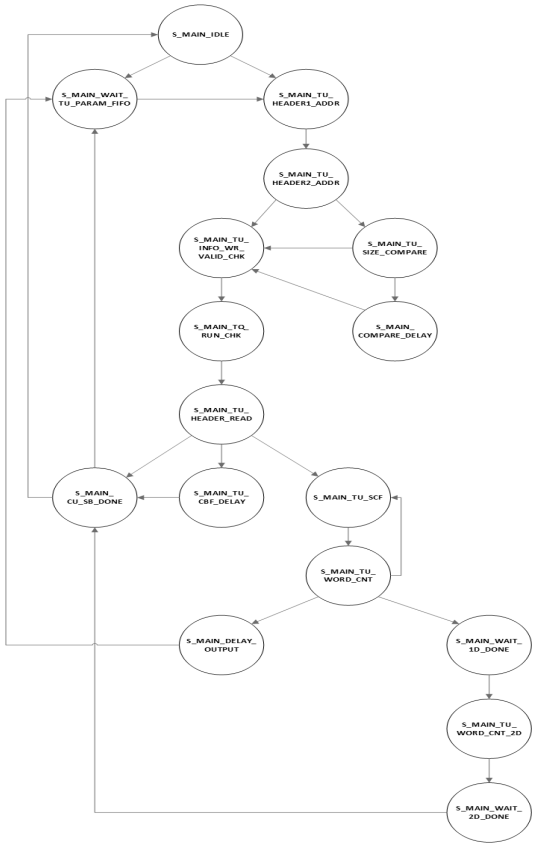

그림 7. 제안하는 구조의 상태머신도
Fig. 7. State Machine of the proposed structure.

으로 제어하지 않으면 현재의 변환수행이 완료 도지 않았음에도 다음 변환동작이 수행된다. 왜냐하면 변환 크기가 클수록 Core 동작 및 처리속도가 오래 걸리기 때문이다. Delay Output 상태는 그림 6의 (a)에서의 지연시간을 나타낸다. 따라서 제안하는 구조가 상태머신(FSM : Finite State Machine)의 지연시간를 통해 파이프라인 동작을 하도록 제어했다.

Ⅳ. 실 험

IDCT 경우에 표 1을 보면 VP9 구글 모델과 RTL의

표 2. 4x4 IADST에 대한 구글 모델과 RTL 결과 값 비교
Table 2. The compare Google model and RTL for 4x4 IADST.

	INPU (hex/dec)	Google Model (hex/dec)	RTL (hex/dec)
0	FEFF(-257)	FDFC(-516)	FDFC(-516)
1	FEA5(-347)	FE7D(-387)	FE7D(-384)
2	FF5F(-161)	FFB2(-78)	FFB2(-78)
3	0000(0)	FFCB(-53)	FFCB(-53)

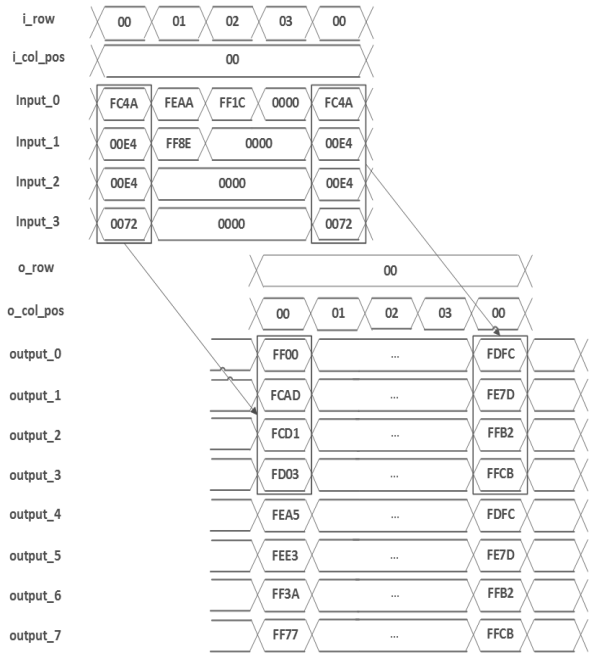

그림 8. 4x4 IDCT 결과 파형
Fig. 8. Waveform of result for 4x4 IDCT.

결과 값을 비교하면 오차가 발생함을 확인할 수 있다. VP9 구글 모델로부터 계수 값들이 근사된 값이기 때문에 VP9 구글 모델과 RTL의 결과는 100 %동일한 결과를 가지기 힘들다. 그림 8은 4x4 IDCT에 대한 RTL 결과 파형을 나타내며 FEFF의 값이 아닌 FF00의 값이 출력됨을 볼 수 있다.

반면에 표 2를 보면 4x4 IADST에 대한 VP9 구글 모델과 RTL의 결과는 100% 완전히 동일한 값을 가진다. 다른 변환모드들과는 달리 각 단계마다 쉬프트 연산이 수행되지 않아 VP9 구글 모델 결과 값과 일치하는 행렬계수 추출이 가능하다.

그림 9은 Dimension 제어 신호와 1-Dimension 트랜스포즈 버퍼에 의해 가로방향 변환결과가 세로 방향으로 데이터가 입력되는 시뮬레이션 결과를 보여준다. 제안하는 구조는 하나의 변환코어에서 Row/Column 변환

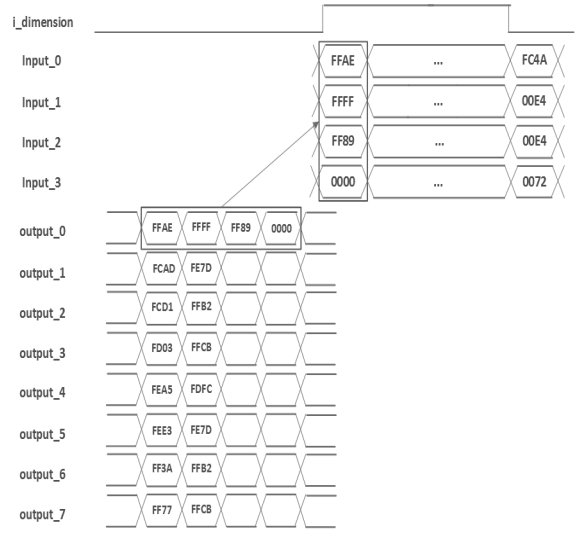

그림 9. 트랜스포즈 버퍼 동작
Fig. 9. Transpose buffer operation.

을 수행한다.

그림 10의 시뮬레이션 결과는 제안하는 정수형 역변환 구조가 4x4, 8x8, 16x16, 32x32 블록 사이즈에 대한 행렬 기반 하에 동작함을 보여준다.

주요 제어신호를 보면 dimension, dst, size에 의해 결정된다. dimension = 0, dst = 0, size = 3이면 32x32 크기의 DCT 가로방향변환을 의미한다.

32x32 크기의 경우 입력 row 제어 신호는 1~31까지 32개의 변수 값을 가지면, 입력제어신호의 값은 1~7까지 8개의 변수 값을 가진다. 입력데이터는 4개씩 받으므로 전체 1024(4x8x32)개의 데이터를 입력받고, 출력데이터는 8개씩 내보내므로 동일하게 1024(8x4x32) 결과 값을 출력한다.

제안하는 구조는 Verilog HDL에서 구현했으며 1GHz@TSMC 28nm 셀 라이브러리로 합성했다.

아래의 표 3은 변환구조에 따른 하드웨어구조에 따른 특성을 나타내며, 현재 VP9에 대하여 하드웨어로 구현되어 있지 않아 기존의 HEVC 기반의 하드웨어 구조와 비교한다.

[12]은 DCT 알고리즘만 지원하는 반면 본 논문에서 제안하는 구조는 ADST, WHT알고리즘도 포함한다. 동일한 행렬 기반의 구조임에도 불구하고 게이트 수가 대략 1.78배 정도의 차이가 나는데 그 이유는 곱셈기의 크기와 개수에 영향을 받아서이다.

곱셈기 크기에서 HEVC와 VP9의 행렬계수의 차이가 발생한다. HEVC의 행렬계수는 최대 7-bits까지 표현되지만, VP9은 14-bits으로 HEVC 행렬계수보다 2배 정도 크다. 또한 제안하는 구조는 [12]보다 곱셈기를 더 많

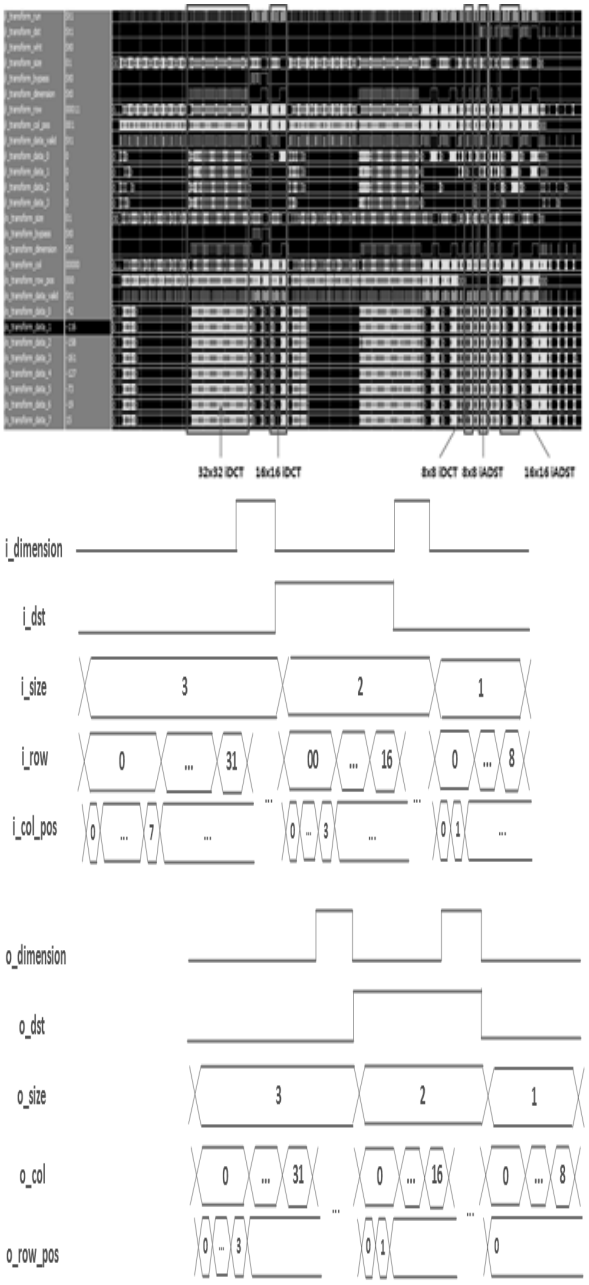

그림 10. 블록 크기에 대한 변환 조합
Fig. 10. Transform combinations for block sizes.

이 사용한다. [12]의 경우엔 DCT알고리즘만 제공하는데, DCT 행렬계수는 대칭성을 가지므로 곱셈기를 절반으로 줄여 하드웨어를 구현할 수 있으나, 제안하는 구조는 비대칭 구조인 ADST를 포함하므로 64개의 곱셈기를 사용하므로 게이트 수에 차이가 난다.

제안하는 구조가 버터플라이구조의 [4]에 비해 게이트 수가 작다. 비록 HEVC 버터플라이구조와 비교하지만 VP9의 구조를 버터플라이구조로 구현한다면 [4]의 게이트 수보다 크게 발생하리라 예상할 수 있다. 왜냐하면 HEVC와 VP9의 계수 값 차이 뿐만 아니라 ADST와 WHT

표 3. 변환구조에 따른 하드웨어 비교
Table 3. Hardware Comparisions with transform structure.

Design		[4]	[12]	Ours
Standard		HEVC	HEVC	VP9
Architecture		Butterfly	Matrix	Matrix
Algorithm	DCT	O	O	O
	ADST	X	X	4x4,8x8, 16x16
	WHT	X	X	4x4
Technology(nm)		0.18um	90nm	28nm
Speed(MHz)		292	270	1000
Gate Count(K)		287	133.8	238.4

표 4. 변환크기에 따른 버터플라이와 행렬의 곱셈기 수 비교
Table 4. The number of multipliers between a butterfly and a matrix according to transform size.

Transform Size	4x4	8x8	16x16	32x32
Butterfly(IDCT)	8	20	52	134
Butterfly(IADST)	8	24	80	-
Matrix	64			

알고리즘을 포함해야 하기 때문이다.

또한 표 4에서 보듯이 버터플라이(IDCT)의 경우 곱셈기가 134개가 필요한데 반해 행렬 기반으로 하드웨어로 설계 시 64개의 곱셈기만을 필요로 한다. 따라서 VP9의 버터플라이구조가 행렬구조보다 2배가량의 곱셈기가 더 필요할 것으로 예상할 수 있다.

V. 결 론

본 논문에서는 게이트 수에 큰 영향을 미치는 곱셈기의 수를 줄이기 위한 방안으로 변환방식을 버터플라이 구조에서 행렬연산구조로 변경함으로써 곱셈기의 사용을 최소화하였다. 버터플라이구조에서 대략 134개의 곱셈기가 필요한 반면 행렬구조에서는 64개의 곱셈기만을 필요로 한다. 즉, 연산구조의 변경으로 곱셈기의 수를 2배가량 줄일 수 있다. 행렬연산방식으로 설계하기 위해 변환행렬계수를 버터플라이구조로부터 추출하고 근사한다.

따라서 제안하는 구조는 하드웨어 리소스를 줄일 수 있으며, 또한 행렬계수 설정만 변경하면 되므로 변환알고리즘을 효율적으로 공유 및 제어하기가 쉬운 이점이 있다.

ACKNOWLEDGMENT

본 논문은 전자부품연구원(KETI)의 “스마트 디바이스에서 인터넷 동영상 재생을 위한 고화질(4K급) 비디오디코더(HEVC/VP9) SoC 개발” 과제를 통해 지원되었음.

REFERENCES

[1] Fatma Belghith, Hassen Loukil, Nouri Masmoudi, “Efficient Hardware Architecture of a Modified 2-D Transform for the HEVC Standard”, International Journal of Computer Science and Application(IJCSA), Volumn 2 IIssue 4, November 2013.
[2] J. Bankoski, J. Koleszar, L. Quillio, j. Salonen, P.Wilkins, Y.Xu, “VP8 Data Format and Decoding Guide”,RFC6386,http://datatracker.ietf.org/doc/ref6386.
[3] Il-Koo Kim, “Coding Efficiency Comparision between Next Generation Video Codecs:HEVC vs VP9”, The Korean Society of Broadcase Engineers, pp.176-179, June 2013.
[4] J.S.Park, W.J.Nam, S.M.Han, and S.S.Lee, “2-D large inverse transform (16x16, 32x32) for HEVC”, Journal of Semiconductor Technology and Science, Vol.12, No.2, pp.203-211, June, 2012.
[5] S. Shen, W. Shen, Y. Fan, and X. Zeng, “A unified 4/8/16/32-point integer IDCT architecture for multiple video coding standard”, Multimedia and Expo(ICME), 2012 IEEE International Conference, Melbourne, VIC, pp.788-793, July 2012.
[6] J.Zhu, Z. Liu, and D. Wang, “Fully Pipelined DCT/IDCT/Hadmard Unified Transform Architecture for HEVC Codec”, IEEE International Symposium on Circuits and Systems(ISCAS), Beijing, pp.677-680, May, 2013.
[7] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim, and C. Yeo, “Efficient integer DCT architectures for HEVC”, IEEE Transactions on Circuits and Systems for Video Technology, Vol.24, pp.168-178, 2013.
[8] J. Han. A. Saxena, and K. Rose, “Towards jointly optimal spatial prediction and adaptive transform in video/image coding”, IEEE International Conference on Acoustics Speech and Signal Processing(ICASSP), Dallas, TX, pp.726-729, March 2010.
[9] V. Britanak, P.C. Yip and K.R. Rao, “Discrete Cosine and Sine Transform, General Properties, Fast Algorithm and Integer Approximation”, Academic Press, London, 2006.

[10] A. Grange, H. Alvestrand, “A VP9 Bitstream Overview(Internet-Draft)”, Google, Auguest 2013.
[11] W. Chen, C. H. Smith, and S.C. Fralick, “A fast computational algorithm for the discrete cosine transform” IEEE Trans. Commun, Vol. COM-25, no.9, pp. 1004-1009, Sep. 1977.
[12] P. T. Chiang, T. S. Chang, “A reconfigurable inverse transform architecture design for HEVC decoder”, IEEE International. Symposium., Circuits and Systems(ISCAS), beijing, pp.1006-1009, May 2013.

저 자 소 개

이 태 희(정회원)
2014년 한양대학교 전자통신공학과 석사 졸업.
2014년～현재 전자부품연구원 임베디드SW융합 연구센터 연구원

<주관심분야 : 임베디드 하드웨어, 영상처리>

황 태 호(정회원)
1997년 한국외국어대학교 전자공학과 석사 졸업.
2013년 한국외국어대학교 컴퓨터공학과 박사 졸업.
2000년～현재 전자부품연구원 임베디드SW융합 연구센터 책임연구원.

<주관심분야 : RTOS, WPAN/WBAN, 임베디드 시스템 S/W>

김 병 수(정회원)
2008년 인하대학교 정보통신공학과 석사 졸업.
2013년 인하대학교 정보통신공학과 박사 졸업.
2013년～현재 전자부품연구원 임베디드SW융합 연구센터 선임연구원.

<주관심분야: 임베디드 하드웨어, 영상처리 하드웨어 VLSI 설계>

김 동 순(평생회원)
1997년 인하대학교 전자재료공학과 석사 졸업.
2005년 인하대학교 전자재료공학과 미디어시스템 박사 졸업.
1999년～현재 전자부품연구원 임베디드SW융합 연구센터 책임연구원.

<주관심분야: 임베디드 하드웨어, 멀티미디어 SoC Design>