GP-GPU를 이용한 H.264/AVC 디코더의 IQ/IDCT구현

Implementation of IQ/IDCT in H.264/AVC Decoder Using GP-GPU

정 준 모*, 이 광 엽*★
Jun-Mo Jeong*,Kwang-Yeob Lee*★

Abstract

The need for dedicated hardware continue to decrease as the mobile CPU's performance increases . But, there is a limit to a mobile CPU's performance. GP-GPU(General-Purpose computing on Graphics Processing Units) can improve performance without adding other dedicated hardware. This paper presents the implementation of Inverse Quantization, Inverse DCT and Color Space Conversion module in H.264/AVC decoder using GP-GPU for a mobile environments. The proposed architecture improves approximately 40% of performance when it use all the features.

요 약

모바일 CPU의 성능이 향상됨에 따라 전용 하드웨어의 필요성이 줄어 들고 있다. 그러나 아직까지 모바일 CPU의 성능은 한계가 있다. 이러한 제약 조건을 병렬처리와 실수 연산이 뛰어난 GP-GPU (General-Purpose computing on Graphics Processing Units)를 이용함으로써 다른 전용 하드웨어의 추가 없이 성능을 향상 시킬 수 있다. 본 논문에서는 모바일 환경에 적합하게 설계된 GP-GPU를 이용하여 H.264 디코더의 Inverse Quantization, Inverse DCT, Color Space Conversion 모듈을 구현하였다. G-PGPU를 이용한 전체 시스템 동작 시 40%의 성능 향상이 있었다.

Key words : GP-GPU, IQ/IDCT, H.264 Decoder

Ⅰ. 서론

최근 몇 년간 3D 그래픽스 관련 산업은 규모면에서 가파른 성장을 보였을 뿐만 아니라 그 응용분야도 컴퓨터 게임으로부터 애니메이션 영화, 가상 캐릭터 등으로 폭넓게 확대되었다. 이러한 그래픽스 산업의 발전은 3D 그래픽스 처리를 가속하기 위한 하드웨어, 즉 GPU(Graphics Processing Units) 기술의 발전이 있었기에 가능했다.

*정회원,서경대학교 전자공학과 교수
*★ 정회원,서경대학교 컴퓨터공학과 교수, 교신저자
※ 본 논문은 ETRI [IT-SoC 핵심 설계 인력양성사업]지원으로 작성되었으며 설계에는 IDEC의 지원장비로 이루어졌습니다.
接受日:2010年 5月 30日, 修正完了日: 2010年 6月 26日

최근에는 고사양의 GPU 성능을 요구하는 3D 그래픽스 응용기술이 많이 개발되면서 GPU 성능 향상의 구심점이 되고 있다. PC와 게임콘솔 등에 사용되는 GPU에는 가장 최신의 기술이 동원되어 그래픽 성능을 극대화 시키고 있으며 여기에 사용되었던 기술이 휴대용 모바일 기기에도 점차 적용되고 있다. [1]

모바일 GPU의 발전은 PC나 게임콘솔 GPU와 성능 면에서는 많은 차이가 있지만 모바일용 GPU 역시 프로그램이 가능한 형태를 갖추어가고 있으며 휴대용 게임 콘솔과 스마트폰을 중심으로 PC용 GPU에서 축적된 하드웨어 기술이 적용되고 있다. 모바일용 GPU가 시장에서 성공하기 위해서는 전력 소모의 최소화가 가장 중요하다. PC용 GPU에서는 하드웨어를 병렬로 배치하여 가능한 많은 연산이 동시에 이루어지도록 하여 높은 성능을 이뤄낼 수 있었다. 하지만

모바일용 GPU는 하드웨어 면적이 제한되어 있고 배터리로부터 전원을 공급받기 때문에 하드웨어의 단순 확장보다는 단순하면서도 효율적인 구조를 도입하고 외부메모리 접근을 최소화하는 등의 노력이 필요하다.

본 논문에서는 모바일 환경에 적합하도록 설계된 GP-GPU를 이용하여 H.264 디코더를 구현하였다. 본 논문에서 사용된 GP-GPU는 멀티스레드 (Multi-Thread)와 듀얼페이즈 (Dual-Phase)의 가변길이 명령어 구조를 채택하고 있다. [2][3]

II. GP-GPU구조 및 H.264 디코더

1. GP-GPU 구조

그림1. 은 본 논문에서 사용된 싱글 코어 GP-GPU의 구조를 나타낸다. 이 프로세서의 구성을 크게 분류하면 코어와 레지스터 모듈 그리고 외부 버스 인터페이스로 나누어 볼 수 있다. 코어는 12개의 스레드로 구성되어 있고 Round-robin 방식으로 순차적으로 실행된다. 명령어 구조는 듀얼페이즈 구조를 적용하여 2개의 페이즈를 통해 각 페이즈에 입력된 명령어들을 동시에 처리한다. 듀얼페이즈 구조는 하나의 ALU를 사용하지만 동시에 두 개의 서로 다른 연산기를 사용함으로써 제한적인 모바일 환경에서 효율적으로 성능을 높일 수 있는 방법이다.

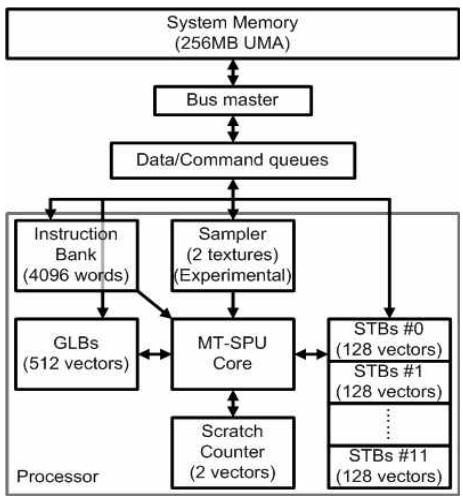

Fig. 1. GP-GPU architecture
그림 1. GP-GPU 구조

2. H.264 디코더

그림2.는 H.264 디코더 구조이다. 디코더는 NAL로부터 압축된 비트스트림을 받아 데이터 요소들에 대해 엔트로피 디코딩을 수행하여 양자화된 계수 X를 생성한다. 생성된 계수들은 역양자화되고 역변환되어 D′n이 생성된다. 디코더는 비트스트림으로부터 디코딩된 헤더 정보를 사용하여 인코더에서 생성된 원래의 예측 블록 PRED와 동일한 예측 블록 PRED를 생성한다. PRED는 D′n에 더해져서 uF′n를 생성하며, F′n는 필터를 거쳐 각각의 디코딩된 블록 F′n을 생성한다.

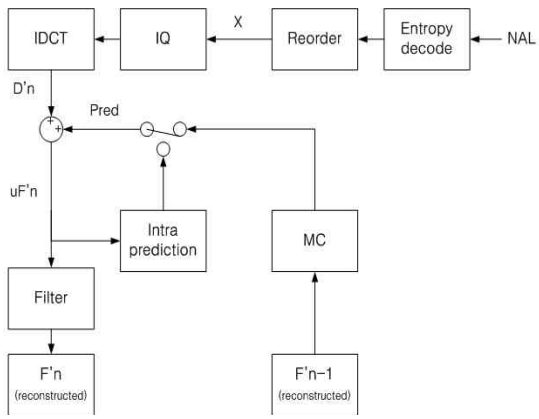

그림 2. H.264 디코더
Fig. 2. H.264 Decoder

III. GP-GPU를 이용한 디코더 구현

1. 병렬처리가 가능한 H.264 디코딩 알고리즘

가. Inverse quantization

양자화는 어떤 범위의 값을 갖는 X를 보다 적은 범위의 값을 갖는 양자화된 신호Y로 대체한다. 양자화된 신호는 원래의 신호보다 적은 범위의 값을 가지므로 보다 적은 비트로 표현이 가능해야 한다. H.264에서는 스칼라 양자화를 사용한다. 순방향 및 역방향 양자화의 방법은 (a) 나눗셈 그리고/또는 부동 소수점 연산을 사용하지 않고 (b) 위에서 설명된 post-스케일링과 pre-스케일링 행렬 E_f와 E_i에 통합시키기 위한 조건에 의해 복잡해졌다.

기본적인 순방향 양자화 연산은 식 (1) 과

같다. [4]

$$Zij = round(Yij/Qstep) \quad (1)$$

여기서 Yij는 위에 설명된 변환의 계수이고, Qstep은 양자화 스텝사이즈이고, Zij는 양자화된 계수이다. 여기서 반올림 연산은 가장 가까운 정수로 반올림할 필요는 없다.

총 52개의 Qstep 값이 표준안에서 지원되며 양자화 파라미터 QP에 의해 지시된다. 표1은 H.264 코덱 표준안의 양자화 스텝 사이즈이다.

표 1. H.264 코덱의 양자화 스텝 사이즈
Table 1. Quantization step sizes in H.264 CODEC

QP	0	1	2	3	4	5	…
Qstep	0.625	0.6875	0.8125	0.875	1	1.125	…
QP	10	…	18	…	24	…	51
Qstep	2	…	5	…	10	…	224

기본적인 역양자화 연산은 식 (2)과 같다.

$$Y'ij = ZijQstep \quad (2)$$

식 (3)는 역변환에 대한 pre-스케일링 계수는 반올림 에러를 방지하기 위한 상수 스케일링 계수 64와 함께 이 연산에 통합된 식이다.

$$W'ij = ZijQstep \cdot PF \cdot 64 \quad (3)$$

$W'ij$는 핵심 역변환 Ci^tWCi에 의해 변환된 스케일된 계수이다. 역변환의 출력 값은 스케일링 계수를 제거하기 위해 64로 나누어진다. H.264 표준안은 Qstep 또는 PF를 직접 정의하고 있지는 않다. 대신 0≤QP≤5인 경우와 각 계수 위치에 대해 파라미터 V=(Qstep·PF·64)를 정의하고 있다. 따라서 스케일링 연산은 식 (4)과 같아진다.

$$W'ij = ZijVij \cdot 2^{floor(QP/6)} \quad (4)$$

식 (4)의 계수 $2^{floor(QP/6)}$는 QP가 6씩 증가할 때마다 스케일된 출력을 2의 계수만큼 증가시키도록 한다.

0≤QP≤5에 대해 표준안에 정의된 V의 값은 표2 와 같다.

표 2. 스케일링 계수 V
Table 2. Scaling factor V

	Positions		
QP	(0,0), (2,0), (2,2), (0,2)	(1,1), (1,3), (3,1), (3,3)	Other
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

나. Inverse DCT

Discrete Cosine Transform(DCT)은 NxN 샘플의 블록 X에 대해서 수행되며 NxN 계수 블록 Y를 생성한다. DCT의 동작은 변환 행렬 A로 설명될 수 있다. H.264에서 사용되는 4x4 블록은 식 (5)와 같이 정의된다.

$$X = AYA^T = \begin{matrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{matrix} X \begin{matrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{matrix} \quad (5)$$

$$a = \frac{1}{2}, b = \sqrt{\frac{2}{5}}, c = \sqrt{\frac{1}{2}}cos(\frac{2\pi}{8})$$

이 행렬의 곱셈은 다음과 같은 등가의 형태 식 (6)으로 인수분해 될 수 있다.

$$X = CXC^T \otimes E = \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{matrix} X \begin{matrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{matrix} \quad (6)$$

$$\otimes \begin{matrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{matrix}$$

CXC^T 2D 변환의 핵심이고, E는 스케일링 계수 (Scaling factors)이다. 기호 ⊗는 (CXC^T) 의 각 요소가 행렬 E의 동일한 위치에 있는 스케일링 계수로 곱해졌다는 것을 나타낸다. 상수 a와 b는 이전과 동일하고 d는 c/b이다. 변환을 간략하게 하기위해 d는 0.5로 근사화되고, 변환이 직교 상태로 남아 있도록

하기 위해서 b도 수정되어야 한다. 행렬 C의 두 번째와 네 번째 행 그리고 C^T의 두 번째와 네 번째 열은 2의 계수에 의해 스케일링 되고, Post-스케일링 행렬 E는 이것을 보상하기 위해 스케일링을 감소시킨다. 이렇게 하여 핵심 변환 CXC^T 의 곱셈을 1/2로 줄일 수 있는데 결과적으로 정수 연산으로 인해 정확도의 손실을 가져오게 된다.

$$X = CXC^T \otimes Ef =$$

$$\begin{matrix}1 & 1 & 1 & 1\\ 2 & 1 & -1 & -2\\ 1 & -1 & -1 & 1\\ d & -2 & 2 & -1\end{matrix} \quad X \quad \begin{matrix}1 & 2 & 1 & 1\\ 1 & 1 & -1 & -2\\ 1 & -1 & -1 & 1\\ 1 & -2 & 1 & -1\end{matrix}$$

$$\otimes \begin{matrix}a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2}\\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4}\\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2}\\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4}\end{matrix} \qquad (7)$$

식 (7) 은 4x4 DCT를 근사화시킨 것이지만, 계수 d와 b의 변화로 인해 새로운 변환의 결과는 4x4 DCT의 결과와 동일하지 않다. b와 d에 따라 출력 계수들 사이에 차이가 존재한다. H.264 코덱에서 근사화 변환은 DCT와 거의 동일한 압축 성능을 나타내며, 많은 중요한 장점을 가진다. 변환의 핵심 부분인 CXC^T는 덧셈, 뺄셈, 그리고 쉬프트만을 사용하여 정수 연산을 수행할 수 있다. 입력이 ±255의 범위 내에 존재하므로 16-비트 연산을 변환 과정에서 사용할 수 있다. 스케일링 이후 동작 $\otimes E_f$ 는 모든 계수에 대해 한 번의 곱셈을 필요로 하는데 양자화 과정으로 흡수될 수 있다.

디코더에서 사용되는 역변환은 식 (8) 과 같다. H.264 표준안은 이 변환을 순차적인 산술 연산으로 정의하고 있다. [5]

Y는 각 계수를 행렬 Ei의 적절한 가중치 계수로 곱하여 pre-스케일 된다. Y가 pre-스케일 되었으므로 행렬 C와 C^T내의 계수 ±1/2는 오른쪽 쉬프트에 의해 정확도의 큰 손실 없이 구현될 수 있다.

$$Y = Ci^T(Y \otimes Ei)Ci =$$

$$\begin{matrix}1 & 1 & 1 & \frac{1}{2}\\ 1 & \frac{1}{2} & -1 & -1\\ 1 & -\frac{1}{2} & -1 & 1\\ 1 & -1 & 2 & -\frac{1}{2}\end{matrix} \left(X \otimes \begin{matrix}a^2 & ab & a^2 & ab\\ ab & b^2 & ab & b^2\\ a^2 & ab & a^2 & ab\\ ab & b^2 & ab & b^2\end{matrix} \right) \qquad (8)$$

$$\begin{matrix}1 & 1 & 1 & 1\\ 1 & \frac{1}{2} & -\frac{1}{2} & -1\\ 1 & -1 & -1 & 1\\ \frac{1}{2} & -1 & 1 & -\frac{1}{2}\end{matrix}$$

다. Color Space Conversion

H.264는 전송율을 감소시키기 위해서 YCbCr을 사용하고 있다. 디코딩된 이미지를 디스플레이 하기 위해서는 RGB로 변환하여야 한다. RGB로 변환을 위한 식은 식 (9) 와 같다.

$$\begin{aligned} R &= Y + \frac{1-Kr}{0.5}Cr \\ G &= Y - \frac{2Kb-2Kb^2}{1-Kb-Kr}Cb - \frac{2Kr-2Kr^2}{1-Kb-Kr}Cr \\ B &= Y + \frac{1-Kb}{0.5}Cb \end{aligned} \qquad (9)$$

ITU-R recommendation BT.601은 kb=0.114, kr=0.299로 정의하고 있다. 위의 식에 이를 대입해 보면 식 (11) 같이 일반적으로 사용되는 변환식을 구할 수 있다.

$$\begin{aligned} R &= 0.299R + 0.587G + 0.114B \\ G &= Y - 0.344Cb - 0.714Cr \\ B &= Y + 1.772Cb \end{aligned} \qquad (11)$$

2. GP-GPU를 이용한 디코더 구현

가. 전체적인 구현방법

그림3. 은 GPU를 이용한 H.264 디코더의 전체 구조를 보여주고 있다. CPU에서는 Entropy 디코딩을 수행하고 GPU에 수행할 데이터를 정렬하였다. GPU에서는 IQ, IDCT를 수행하였다. 이후 CPU에서 다시 Intra/Inter Prediction과 De-blocking 필터를 수행하고, GPU에서 CSC 작업을 수행 하였다.

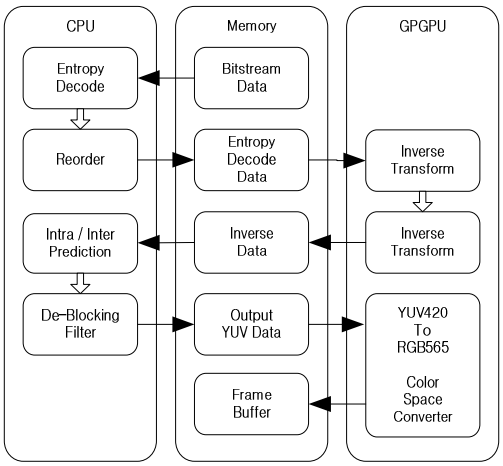

그림 3. GPGPU를 이용한 H.264 디코더 구조
Fig. 3. H.264 decoder architecture using GPGPU

나. Inverse Quntizition

Entropy 디코더에서 출력된 QP값을 이용해서 역양자화 계수를 구하고, 역양자화 계수와 양자화된 계수를 GPGPU의 글로벌 메모리에 저장한다. GPGPU는 글로벌 메모리에서 역양자화 계수와 양자화된 계수를 로컬 메모리로 읽은 후 SIMD 구조를 이용한 벡터연산을 수행하여 이를 시스템 메모리에 저장한다.

다. Inverse DCT

Entropy 디코더에서 출력된 CBP값을 GP-GPU의 글로벌 메모리에 저장한다. GP-GPU는 역양자화된 데이터를 시스템 메모리에서 로컬 메모리로 읽은 후 CBP 값에 따라 역 이산 코사인 변환의 수행하고, 출력 데이터를 시스템 메모리에 저장한다.

라. Color Space Conversion

CPU에서 YCbCr의 주소 값과 프레임버퍼의 주소 값을 G-PGPU의 글로벌 메모리에 저장한다. GP-GPU는 글로벌 메모리의 YCbCr 주소 값을 이용하여 시스템 메모리에서 데이터를 읽어 식 (11) 에 맞게 색 공간 변환을 수행하고, 출력 데이터를 시스템 메모리의 프레임 버퍼에 저장한다. GP-GPU에서는 멀티스레드와 SIMD 구조를 이용하여 스레드당 32개의 픽셀 단위로 처리하도록 하였다.

Ⅳ. 실험 결과

모든 모듈들은 우선 CPU 코드로 구현되었고 GP-GPU의 병렬처리에 적합하게 변환하였다. 표 3 . 은 CPU에서 구현된 코드와 GP-GPU의 병렬처리에 적합하게 변환된 코드의 예이다.

표 3. C 코드와 GPU 어셈블리 코드로 구현된 4 x4 IDCT 연산 예
Table 3. Example, C code and GPU assembly code, 4x4 IDCT operation.

C Code

```
for (j=0;j<4;j++)
 {
    for(i=0;i<4;i++)
        M5[i]=coeffs[blk_idx][i+(j<<2)];
    M6[0]=(M5[0]+M5[2]);
    M6[1]=(M5[0]-M5[2]);
    M6[2]=(M5[1]>>1)-M5[3];
    M6[3]=M5[1]+(M5[3]>>1);

    M7[0][j] =M6[0]+M6[3];
    M7[1][j] =M6[1]+M6[2];
    M7[2][j] =M6[1]-M6[2];
    M7[3][j] =M6[0]-M6[3];
}
```

GPGPU Assembly Code

```
r[T0]=r[Blk]*r[M0];
r[T1]=r[Blk]*r[M1] / r[T0].xy=r[T0].xy+r[T0].zw;
r[T2]=r[Blk]*r[M2] / r[T1].xy=r[T1].xy+r[T1].zw;
r[T3]=r[Blk]*r[M3] / r[T2].xy=r[T2].xy+r[T2].zw;
r[T3].xy=r[T3].xy+r[T3].zw;
r[T_blk0].x=r[T0].x+r[T0].y;
r[T_blk1].x=r[T1].x+r[T1].y
r[T_blk2].x=r[T2].x+r[T2].y
r[T_blk3].x=r[T3].x+r[T3].y
```

실험에 사용된 환경은 Platform으로 Virtex5가 탑재된 Xilinx사의 ML-506 보드를 사용하였고, CPU는 소프트 코어인 MicroBlaze를 사용하였다. 메모리는 DDR2 SODIMM 256 MB를 사용하였다. GP-GPU는 모바일 환경에 적합하게 설계된 MTSP를 사용하였다. 소프트 IP인 CPU와 GP-GPU를 합성하여 FPGA에 배열한 후 동작클럭 100㎒로 실험하였다.

표 4 과 그림4는 QCIF(176x144) 크기의 영상에 대한 각 모듈의 성능 측정 결과이다. 모듈에

따라 CPU와 GP-GPU의 속도 차이를 비교하였다. 비교 결과 각 모듈에 대해서 CPU를 이용하는 것보다 GP-GPU를 이용하여 처리할 경우 약 1.1배~7배 정도의 성능 향상이 있다는 것을 알 수 있다.

표 4. 모듈별 CPU와 GPGPU의 성능측정 결과
Table 4. Result, performance of CPU and GP-GPU

모듈	CPU	GPGPU	Speed-up
IQ	0.0075	0.0068	1.1
IDCT	0.009	0.0075	1.2
CSC	0.016	0.0023	6.96

<시간단위:sec>

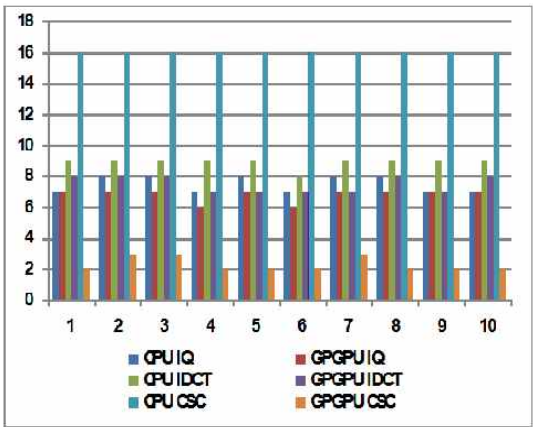

그림 4. 모듈별 샘플 프레임 결과
Fig. 4. Result, sample frame

V 결론

본 논문에서는 모바일 GP-GPU를 이용하여 H.264 디코더 일부를 구현하였다. Inverse Quantization, Inverse DCT, Color Space Conversion 모듈을 멀티스레드, SIMD 구조를 이용하여 병렬처리 하였다. Inverse Quantization과 Inverse DCT의 경우 CPU를 이용한 것보다 각각 1.1배와 1.2배의 성능이 향상되었고, Color space conversion의 경우 약 7배 정도의 성능 향상이 있었다. 전체 시스템 동작시 1.4배 정도의 성능 향상이 있었다.

최근 모바일 CPU의 성능이 향상됨에 따라 전용 하드웨어의 필요성이 줄어 들고 있다. 그러나, 아직까지 모바일 CPU의 성능은 한계가 있다. 이를 병렬처리와 실수 연산이 뛰어난 GP-GPU를 이용함으로써 다른 전용 하드웨어의 추가 없이 성능을 향상 시킬 수 있을 것으로 예상된다.

참고문헌

[1] General Purpose GPU Programming(GPGPU) Website, http://www.gpgpu.org
[2] Hyung-Ki Jeong, "A Fullly Programmable Shader Processor for Low Power Mobile Devices" Journal of IKEEE, Vol.13, No.2, February, 2009.
[3] H.K. Jeong, "A Multi-thread Processor Architecture With Dual Phase Variable-Length Instructions" Proceeding of ITC-CSCC 2008, July 6-9, 2008, Japan.
[4] A. Hallapuro, M. Karczewicz and H. Malvar, Low Complexity Transform and Quantization - Part I : Basic Implementation, JVT document JVT-B038, Geneva, February 2002.
[5] ISO/IEC 14496-10 and ITU-T Rec. H,264, Advanced Video Coding, 2003.

저 자 소 개

정 준 모 (정회원)

1985년 한양대학교 전자공학과 학사
1987년 한양대학교 전자공학과 석사
1992년 한양대학교 전자공학과 박사
1991년~1995년 부천대학 전자계산기과 조교수
1995년~현재 서경대학교 전자공학과 부교수
<주관심분야> 반도체회로 설계 및 테스팅, 마이크로 프로세서

이 광 엽 (정회원)

1985년 8월 서강대학교 전자공학과 학사
1987년 8월 연세대학교 전자공학과 석사
1994년 2월 연세대학교 전자공학과 박사
1989~1995년 현대전자 선임연구원
1995년~현재 서경대학교 컴퓨터공학과 부교수
<주관심분야> 마이크로 프로세서, Embedded System, 3D Graphics System