논문 2008-45CI-3-17

H.264/AVC 표준의 디블록킹 필터를 가속하기 위한 ASIP 설계

(An ASIP Design for Deblocking Filter of H.264/AVC)

이 형 표*, 이 용 석**

(Hyoung Pyo Lee and Yong Surk Lee)

요 약

복호된 영상의 블록 경계에서 발생하는 왜곡을 보정하기 위해 사용된 H.264/AVC 표준의 디블록킹 필터는 개선된 품질의 영상을 제공하지만, 이에 사용되는 복잡한 필터링 연산은 복호기의 처리 시간을 지연시키는 주된 요인이 되고 있다. 본 논문에서는 이러한 필터링 연산을 더 빠르게 수행할 수 있는 명령어를 제안하고 ASIP을 구성하여 디블록킹 필터를 가속하였다. LISA를 이용하여 MIPS 기반의 기준 프로세서를 설계하고 디블록킹 필터 모델을 시뮬레이션하여 제안하는 명령어 적용에 따른 실행 사이클의 성능 향상을 비교하였으며, 설계된 기준 프로세서를 CoWare의 Processor Designer를 통해 HDL을 생성하고 Synopsys의 Design Compiler를 이용하여 TSMC 0.25um 공정으로 합성하고 제안하는 명령어를 추가할 경우에 대해 면적 및 동작 지연시간 등을 비교하였다. 합성 결과, 제안하는 명령어 셋을 적용함에 따라 면적 및 동작 지연시간에서 각각 7.5%와 3.2%의 증가를 보였으며, 이로 인해 실행 사이클 면에서는 평균 18.18%의 성능 향상을 보였다.

Abstract

Though a deblocking filter of H.264/AVC provides enhanced image quality by removing blocking artifact on block boundary, the complex filtering operation on this process is a dominant factor of the whole decoding time. In this paper, we designed an ASIP to accelerate deblocking filter operation with the proposed instruction set. We designed a processor based on a MIPS structure with LISA, simulated a deblocking filter model, and compared the execution time on the proposed instruction set. In addition, we generated HDL model of the processor through CoWare's Processor Designer and synthesized with TSMC 0.25um CMOS cell library by Synopsys Design Compiler. As the result of the synthesis, the area and delay time increased 7.5% and 3.2%, respectively. However, due to the proposed instruction set, total execution performance is improved by 18.18% on average.

Keywords : ASIP, H.264/AVC, deblocking filter, LISA

Ⅰ. 서 론

H.264/AVC는 JVT(Joint Video Team)가 기존의 MPEG-4 및 H.263보다 뛰어난 압축 성능을 위해 제안한 동영상 압축 표준으로 높은 압축률과 개선된 품질로 차세대 동영상 압축 표준으로 자리매김하고 있다[1]. 또한, 높은 압축률로 인해 주파수를 효율적으로 사용할 수 있어, 휴대폰 및 DMB 등에 사용되고 있으며, 디지털 TV 및 화상회의에까지 그 적용이 검토되고 있다.

그러나 이러한 H.264/AVC는 높은 압축률과 개선된 품질의 영상을 제공하기 위해 부호기와 복호기 구현이 복잡해지는 단점이 있으며, 부호기에 있어서는 파라미터 및 부호화 모드 결정이 많아지고, 복호기의 경우에 있어서는 디블록킹 필터나 움직임 보상 등으로 인해 계산량이 증가하는 단점도 갖고 있다. 특히 디블록킹 필터는 복잡한 필터링 연산과 잦은 메모리 참조로 인하여 복호기 전체 프로세스의 약 36%로 가장 큰 비중을 차

* 학생회원, ** 평생회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)

※ 본 연구는 한국과학기술재단 특정기초연구(No. R01-2006-000-10156-0)지원으로 수행되었으며, IDEC (IC Design Education Center)에 의해 지원되는 EDA 툴이 사용되었습니다.
접수일자: 2008년1월11일, 수정완료일: 2008년4월24일

지하고 있어 이를 최적화하기 위한 필터링 순서 및 효율적인 메모리 구조 등이 많이 연구되고 있다. [3]은 메모리로부터 읽어 온 픽셀 값을 다음 필터링에서 재사용할 수 있는 블록 단위의 새로운 필터링 순서를 제안하고 재구성이 가능한 쉬프트 레지스터로 이루어진 데이터 패스와 2-포트 SRAM을 이용하여 메모리 참조에 대한 효율을 높였으며, [4]는 자료의 의존도를 이용, 수직 및 수평 경계면에 대해 교대로 필터링하여 자료 재사용의 효율을 높일 수 있는 필터링 순서를 제안하고 전치 행렬 기법을 이용하여 메모리 참조에 대한 문제를 개선하였다. 또한, [5]는 블록 단위가 아닌 라인 단위의 필터링 순서를 제안하여 픽셀 값을 저장할 레지스터의 개수를 줄였으며, 이로 인해 연산 유닛과 레지스터의 개수가 한정되어 있는 프로세서 기반의 시스템에서 자료 재사용의 효율을 더욱 높일 수 있도록 하였다.

한편, H.264/AVC의 처리 속도를 개선하기 위해 각 기능별 가속화 엔진들이 많이 연구되고 있는데, 특정 어플리케이션을 위해 사용자가 최적화된 명령어 셋과 데이터 패스를 정의하여 사용하는 프로세서인 ASIP(Application Specific Instruction-set Processor)을 통한 다중 프로세서 환경이 고려되고 있다. 이러한 경향에 맞추어 본 논문에서는 디블록킹 필터를 가속하는 명령어 셋을 제안하고, 여러 가지 아키텍처 기술 언어 중에 컴파일러를 비롯한 소프트웨어 개발 툴 및 HDL 코드 생성이 가능한 LISA(Language for Instruction-Set Architecture)를 통해 ASIP을 설계하여 프로세서 기반의 시스템에 적합한 [5]를 기준으로 제안하는 명령어 셋 적용 전후에 대한 성능을 비교한다.

본 논문의 구성은 다음과 같다. 본론에서는 LISA를 이용하여 기준 프로세서를 설계하며, 디블록킹 필터의 필터링 연산을 분석하고 이를 가속할 수 있는 명령어 셋을 제안하여 ASIP을 구성한다. 실험에서는 [5]의 디블록킹 필터 모델을 기준 프로세서에서 시뮬레이션하고 제안하는 명령어 셋을 사용한 경우와 사용하지 않은 경우에 대해서 각각 그 실행 사이클을 비교한다. 설계된 프로세서는 제안하는 명령어 셋을 적용한 경우와 그렇지 않은 경우에 대해서 각각 HDL을 생성하고 이를 합성하여 면적 및 동작 주파수 등의 성능을 비교한다. 마지막으로 결론에서는 실험 결과를 분석한다.

Ⅱ. 본 론

1. LISA를 이용한 기준 프로세서 설계

LISA는 아키텍처 및 주변장치들에 대해 기술이 가능하며 시스템 검증에 필요한 모든 툴 셋에 대한 지원이 가능한 언어로서 다양한 형태의 프로세서에 대한 명령어 셋을 기술할 수 있도록 높은 유연성을 제공하며, 복잡한 파이프라인을 지닌 프로세서도 쉽게 모델링될 수 있다. 이러한 LISA는 프로세서 자원(resource)과 동작(operation)의 두 가지 요소로 구성되는데 resource는 레지스터, 메모리 및 파이프라인 등과 같은 저장을 위한 객체를 나타내며, operation은 동작의 행위, 명령어 셋의 정보 및 타이밍 등을 특정 짓는다. 또한, C-컴파일러, 어셈블러, 링커, 시뮬레이터 및 디버거 등의 소프트웨어 개발 툴에 대한 생성이 가능하며, 컴파일러를 통해 합성 가능한 HDL 코드를 생성할 수 있다[9~11, 13].

본 논문에서 LISA를 이용하여 설계한 기준 프로세서는 전형적인 5단 파이프라인 시스템으로 이루어진 MIPS 기반의 프로세서로 그림 1에 그 구조를 나타내었다. 각 파이프 단계에서 하는 일을 살펴보면, 패치 단계에서 명령어를 패치하고, 복호 단계에서 명령어를 해석하여 오퍼랜드를 읽고, 실행 단계에서는 연산을 실행하거나 메모리 주소를 계산한다. 그리고 메모리 단계에서는 메모리로부터 데이터를 읽거나 메모리에 데이터를 쓰게 되며, 기록 단계에서 연산의 결과나 메모리에서 읽은 값을 레지스터에 업데이트한다[12]. 또한, 기준 프로세서는 32bit ALU와 shifter, 그리고 32x32bit MAC을

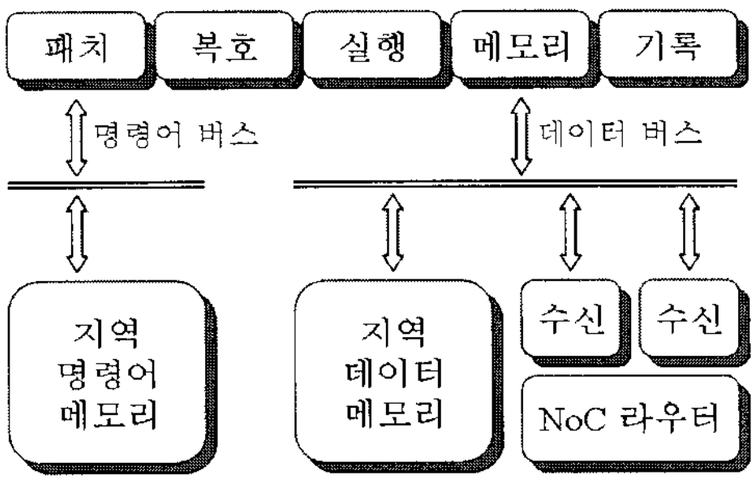

그림 1. 기준 프로세서의 5단 파이프라인 시스템
Fig. 1. 5-stage pipeline of basic processor.

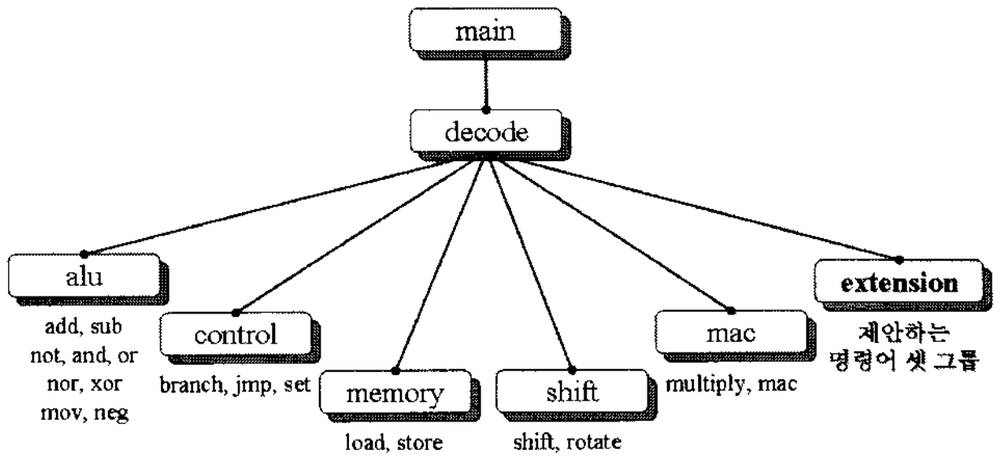

그림 2. 기준 프로세서의 명령어 셋 계층 구조
Fig. 2. Hierarchy of instruction set for basic processor.

포함하며, 128kbyte의 명령어 메모리 및 데이터 메모리로 구성된다. LISA는 계층 구조적 모델링으로 코드에 대한 관리가 용이하며, 이를 이용해 설계된 기준 프로세서의 명령어 셋 그룹에 대한 계층구조를 그림 2에 나타내었다. 명령어의 성격에 따라 6개의 그룹으로 구성하였으며, 디블록킹 필터를 가속하기 위해 본 논문에서 제안하는 명령어 셋은 확장 명령어(extension) 그룹에 정의되어 ASIP을 구성하게 된다.

2. 디블록킹 필터의 필터링 연산

H.264/AVC의 디블록킹 필터는 복호 과정에서 역 양자화 및 역 변환에 의해 블록의 경계면에 발생하는 왜곡 및 픽셀값의 불연속성을 보정하기 위해 사용되며, 왜곡의 정도를 나타내는 경계면 강도를 결정하는 부분과 이를 바탕으로 필터링하는 부분으로 구성된다. 필터링은 래스터 주사 순서대로 매크로블록 단위로 이루어지며, 매크로블록 안에서는 먼저 수직 경계면의 좌우에 위치한 픽셀들을 필터링하고, 이어서 수평 경계면의 상하에 위치한 픽셀들을 필터링한다[2]. 그림 3은 수직 및 수평 경계면의 양쪽에 위치한 8개의 픽셀을 나타내며, 이 값들을 이용하여 경계면 강도 및 픽셀간의 크기 변화에 따라 최대 6개의 픽셀이 필터링 된다.

q픽셀의 필터링에 사용되는 연산을 식 (1)~(5)에 나타내었으며, 메모리로부터 p3~p0와 q0~q3의 값을 읽어 필터링 연산을 수행하고 필터링된 새로운 값을 다시 메모리에 저장하게 된다[7]. 식에서 볼 수 있듯이, 필터링은 여러 개의 덧셈과 쉬프트 연산으로 이루어지며, 프로세서 기반의 시스템에서 실행할 경우 모두 여러 개의 명령어로 수행되어 디블록킹 필터의 처리시간을 지연시키는 주된 요인이 되고 있으므로 디블록킹 필터를 가속하기 위해서는 이들을 보다 빠르게 연산할 수 있는 명령어가 요구된다.

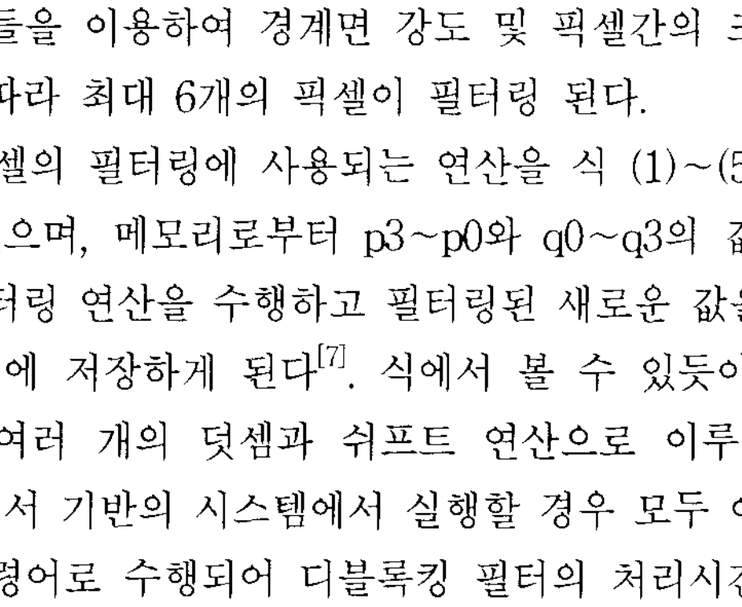

그림 3. 경계면의 양쪽에 위치한 픽셀들
Fig. 3. Pixels on both side of block boundary.

$$(p1+((q1+q0+p0)\ll 1)+q2+4)\gg 3 \quad (1)$$

$$(q2+q0+q1+p0+2)\gg 2 \quad (2)$$

$$(((q3+q2)\ll 1)+q2+q1+q0+p0+4)\gg 3 \quad (3)$$

$$((q1\ll 1)+q0+p1+2)\gg 2 \quad (4)$$

$$(q0+p0+1)\gg 1 \quad (5)$$

3. 제안하는 명령어 셋

식 (1)~(5)의 왼쪽 쉬프트와 오른쪽 쉬프트 연산을 각각 곱셈과 나눗셈 연산으로 바꾸어 정리하고 각각의 픽셀을 a부터 h까지의 알파벳으로 나타내면 식 (5)는 식 (6)으로, 식 (2)와 (4)는 식 (7)로, 그리고 식 (1)과 (3)은 식 (8)로 표현 될 수 있으며 이는 최대 2개, 4개 혹은 8개 픽셀에 대한 반올림 평균을 구하는 식과 같음을 알 수 있다.

$$\frac{a+b+1}{2} \quad (6)$$

$$\frac{a+b+c+d+2}{4} \quad (7)$$

$$\frac{a+b+c+d+e+f+g+h+4}{8} \quad (8)$$

따라서 이를 가속하기 위해서는 덧셈 연산의 처리 속도를 높일 필요가 있는데, 기준 프로세서에서 지원하는 월리스 트리의 자리 올림 보존 가산기(carry save adder)는 전파지연 없이 여러 개의 덧셈을 한 번에 연산하여 필터링 연산을 가속할 수 있게 해 준다. 그러나 MIPS 기반의 기준 프로세서는 두 개의 소스 레지스터만을 가지므로 위와 같은 연산을 명령어로 제안하고 적용하기 위해서는 먼저 최대 8개의 픽셀 데이터를 두 개의 소스 레지스터에 모을 필요가 있는데, 영상 데이터의 특성상 각각의 픽셀 데이터는 8bit의 크기를 가지므로 32bit의 크기를 가지는 소스 레지스터에는 최대 4개의 픽셀 데이터를 모을 수 있게 된다. 따라서 본 논문에서는 여러 개의 픽셀 데이터를 하나의 레지스터에 모으는 명령어인 gthrh와 gthrw를 제안하며 그 과정을 그림 4에 나타내었다. 그림 4에서 볼 수 있듯이, gthrh와 gthrw는 각각 8bit 혹은 16bit 크기를 갖는 두 개의 소스 레지스터를 모아 목적 레지스터에 16bit 혹은 32bit 크기의 값을 저장하는 명령어이다. 즉, 4개의 인자를 갖

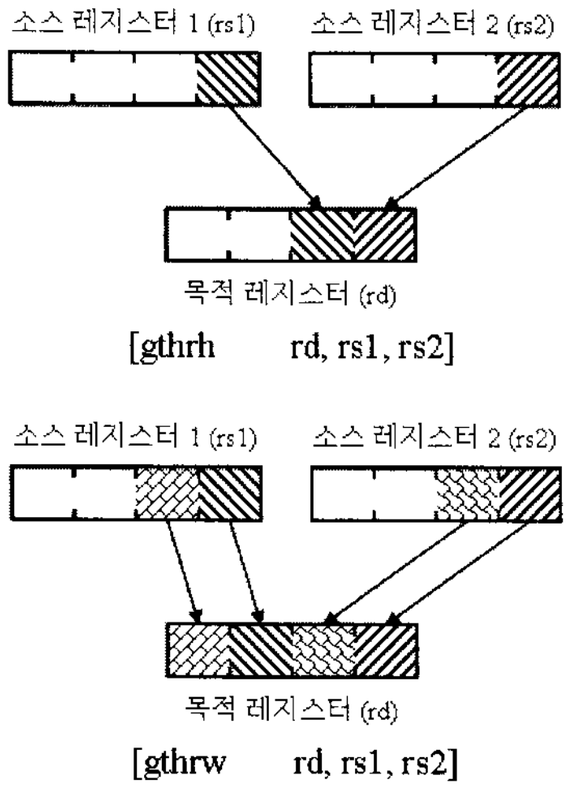

그림 4. gthrh 와 gthrw 명령어
Fig. 4. gthrh and gthrw instructions.

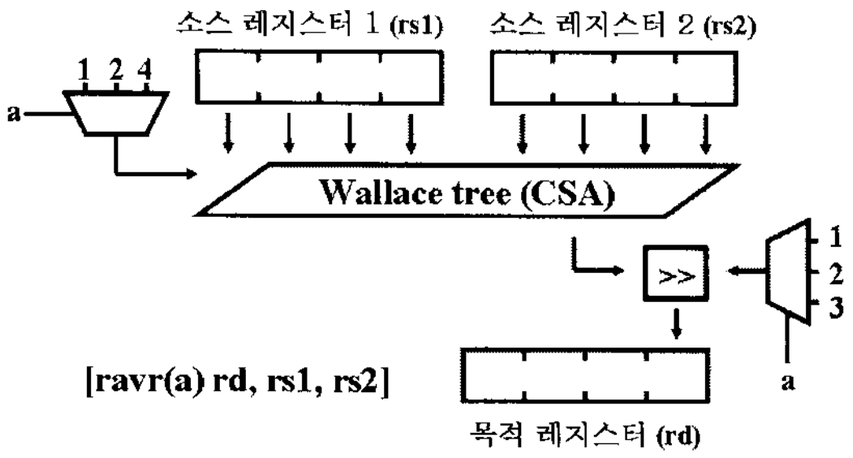

그림 5. ravr2, ravr4 및 ravr8 명령어
Fig. 5. ravr2, ravr4, and ravr8 instructions.

는 반올림 평균을 구하는 데에는 gthrh가 사용되며, 8개의 인자를 갖는 반올림 평균을 구하는 데에는 gthrh와 gthrw가 모두 사용된다. 반올림 평균이 2개의 인자를 가질 때에는 이 과정이 필요 없게 된다.

gthrh와 gthrw를 이용하여 두 개의 소스 레지스터에 최대 8개의 픽셀 데이터를 모은 다음, 필터링 연산인 반올림 평균을 구하기 위해 ravr2, ravr4 그리고 ravr8을 제안하고 그림 5에 나타내었다.

명령어 뒤에 붙는 숫자는 각각 반올림 평균을 구하는데 입력되는 픽셀 데이터의 개수를 나타내며, 반올림을 위해 픽셀 데이터 개수의 1/2만큼의 정수가 함께 더해진다. 영상의 픽셀 데이터는 그 특성상 0에서 255의 값을 가지며, 이는 8bit의 크기로 표현된다. 본 논문에서 제안하는 gthrx와 ravrx는 이러한 픽셀 데이터의 특성을 이용한 것으로 8bit의 크기를 갖는 독립적인 픽셀 데이터를 소스 레지스터에 모으고, 이를 다시 독립적으로 연산하여 8bit의 크기를 갖는 새로운 픽셀 데이터를 생성하는 과정으로 설명될 수 있다. gthrx와 ravrx를 이용한 필터링 연산의 한 예를 그림 6에 나타내었으며, 이는 gthrw을 이용하여 소스 레지스터인 rs1과 rs2에

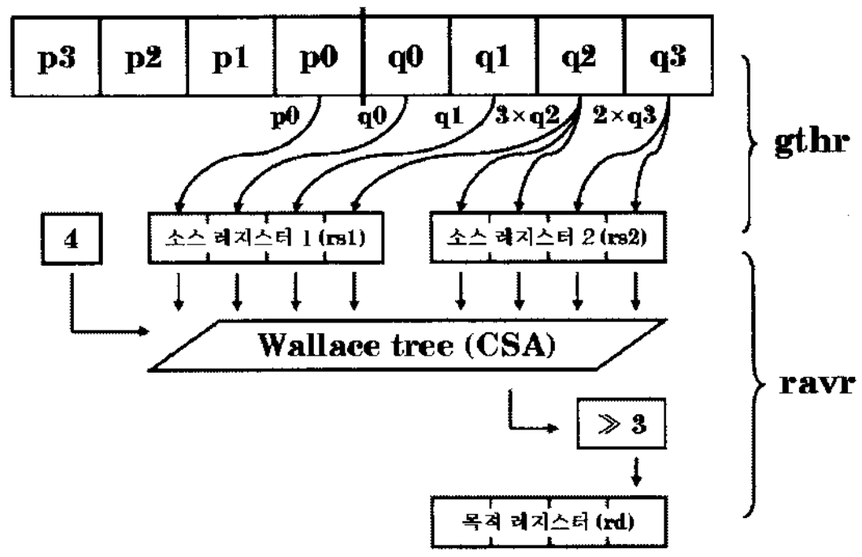

그림 6. gthrx와 ravrx를 이용한 필터링의 예
Fig. 6. a filtering example using gthrx and ravrx.

표 1. 제안하는 명령어 셋
Table 1. proposed instruction set.

명령어	동작 설명
ravr2	두 개의 8bit 수에 대한 반올림 평균 수행
ravr4	네 개의 8bit 수에 대한 반올림 평균 수행
ravr8	여덟 개의 8bit 수에 대한 반올림 평균 수행
gthrh	두 개의 8bit 수를 이용하여 16bit 수를 생성
gthrw	네 개의 8bit 수를 이용하여 32bit 수를 생성
absdiffb	두 개의 8bit 수의 차이에 대한 절대값 수행
absdiffh	두 개의 16bit 수의 차이에 대한 절대값 수행
min	두 개의 수에서 보다 작은 값을 저장
max	두 개의 수에서 보다 큰 값을 저장

모으고, ravr8을 이용하여 목적 레지스터인 rd에 저장하는 과정을 나타낸다. gthrx와 ravrx외에 본 논문에서는 추가로 min, max, absdiff 등의 명령어를 제안한다. 이는 각각 두 수의 최소값, 최대값, 그리고 차에 대한 절대값을 구하기 위한 명령어로서 DSP 프로세서에서 볼 수 있는 명령어지만 MIPS 기반의 기준 프로세서에서는 지원하지 않는 명령어로서 본 논문에서 구현하는 ASIP에 추가하여 디블록킹 필터를 가속한다. 지금까지 제안된 명령어를 표 1에 요약하여 나타내었다.

III. 실 험

LISA를 이용해 제안하는 명령어 셋을 모델링하여 기준 프로세서에 적용하였으며, ravr의 실행단계에 대한 동작을 그림 7에 나타내었다. 이는 두 개의 소스 레지스터에서 픽셀 값을 읽어 반올림 평균 연산을 수행한 다음 결과를 기록단계의 레지스터에 저장하는 과정을 나타낸다.

제안하는 명령어 셋에 대한 성능을 평가하기 위해 프로세서 환경에 적합한 [5]의 디블록킹 필터 모델을 LISA를 이용해 설계한 기준 프로세서에서 컴파일하고

```
GROUP func = { ravr2 || ravr4 || ravr8 || gthrh || ghtrw || max || min || absdiffb || absdiffh };
CODING { 0b110000 rd rs1 rs2 0b00000000 func }
SYNTAX { func ~" " rd "," rs1 "," rs2 }

OPERATION ext_ex POLL IN pipe.EX {
  BEHAVIOR {
    uint8 ext_func;
    uint32 ext_tmp;
    switch (ext_func) {
      case EXT_RAVR2 :
        ext_tmp = ( (ext_in1 & 0x000000ff) + (ext_in2 & 0x000000ff) + 1 ) >> 1;
        ext_out = ext_tmp & 0x000000ff;
        break;
      case EXT_RAVR4 :
        ext_tmp = ( (ext_in1 & 0x000000ff) + ((ext_in1 & 0x0000ff00) >> 8)
                + (ext_in2 & 0x000000ff) + ((ext_in2 & 0x0000ff00) >> 8) + 2 ) >> 2;
        ext_out = ext_tmp & 0x000000ff;
        break;
      case EXT_RAVR8 :
        ext_tmp = ( (ext_in1 & 0x000000ff) + ((ext_in1 & 0x0000ff00) >> 8)
                + ((ext_in1 & 0x00ff0000) >> 16) + ((ext_in1 & 0xff000000) >> 24)
                + (ext_in2 & 0x000000ff) + ((ext_in2 & 0x0000ff00) >> 8)
                + ((ext_in2 & 0x00ff0000) >> 16) + ((ext_in2 & 0xff000000) >> 24) + 4 ) >> 3;
        ext_out = ext_tmp & 0x000000ff;
        break;
      OUT.wbv = ext_out;
    }
  }
}
```

그림 7. LISA를 이용한 제안하는 명령어 셋의 모델링
Fig. 7. LISA model for proposed instruction set.

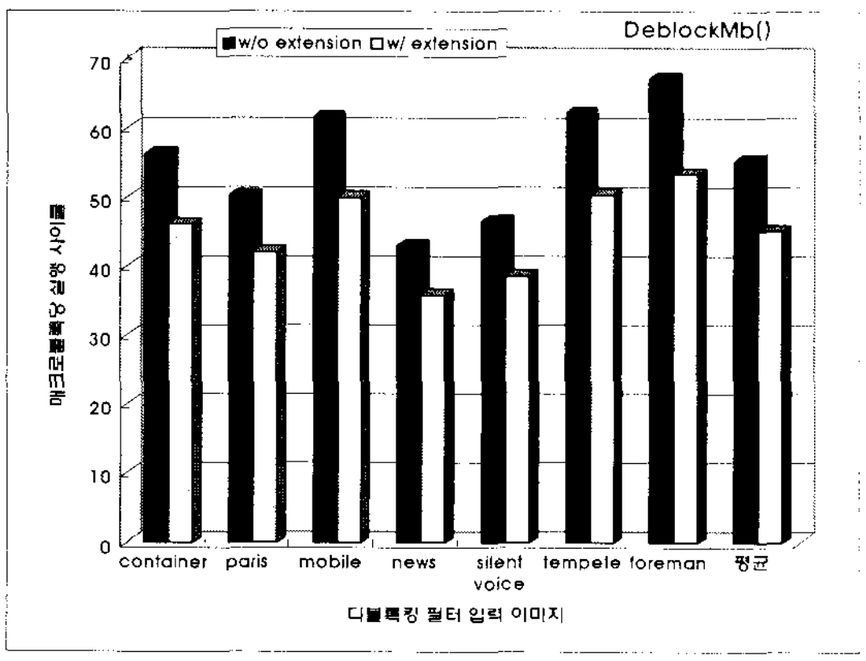

그림 8. DeblockMb 함수의 실행 사이클 비교
Fig. 8. comparison of execution cycles for DeblockMb.

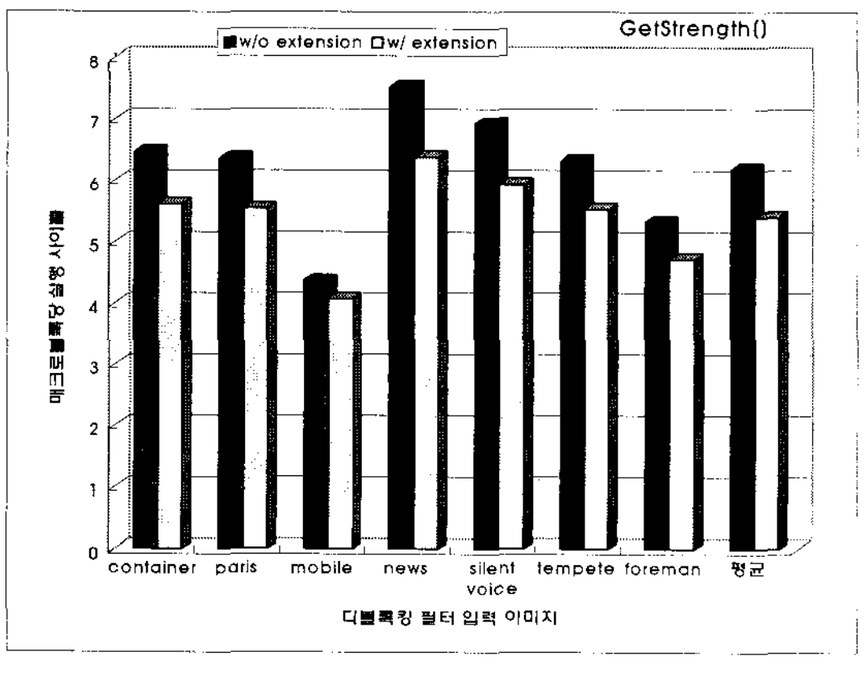

그림 9. GetStrength 함수의 실행 사이클 비교
Fig. 9. comparison of execution cycles for GetStrength.

시뮬레이션 하여 제안하는 명령어 셋을 적용하였을 경우와 그렇지 않은 경우에 대하여 실행 사이클을 비교하였다. 디블록킹 필터의 입력으로는 VCEG(Video Coding Expert Group)에서 제안하는 QCIF 및 CIF 급 이미지가 사용되었으며[8] H.264/AVC 디블록킹 필터의

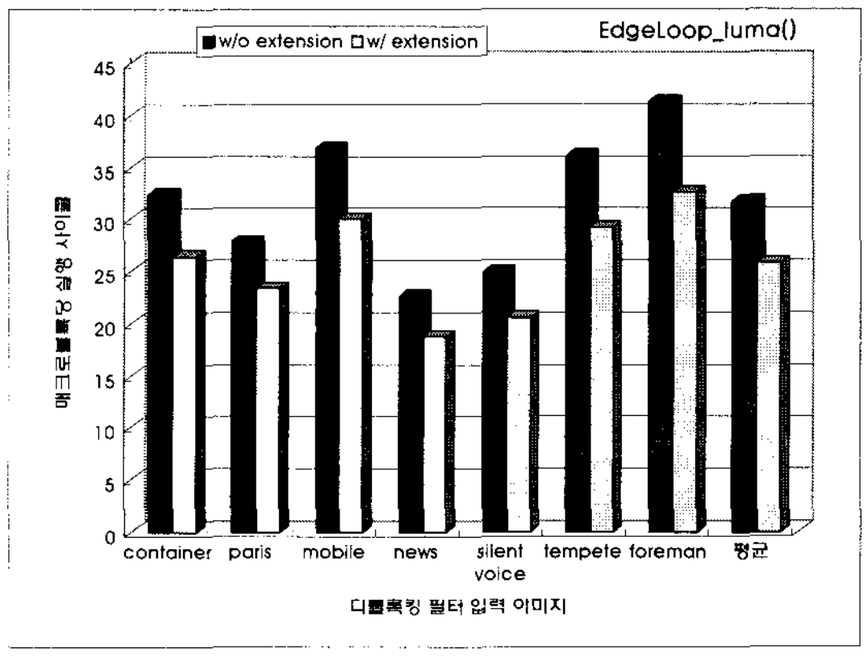

그림 10. EdgeLoop_luma 함수의 실행 사이클 비교
Fig. 10. comparison of execution cycles for EdgeLoop_luma.

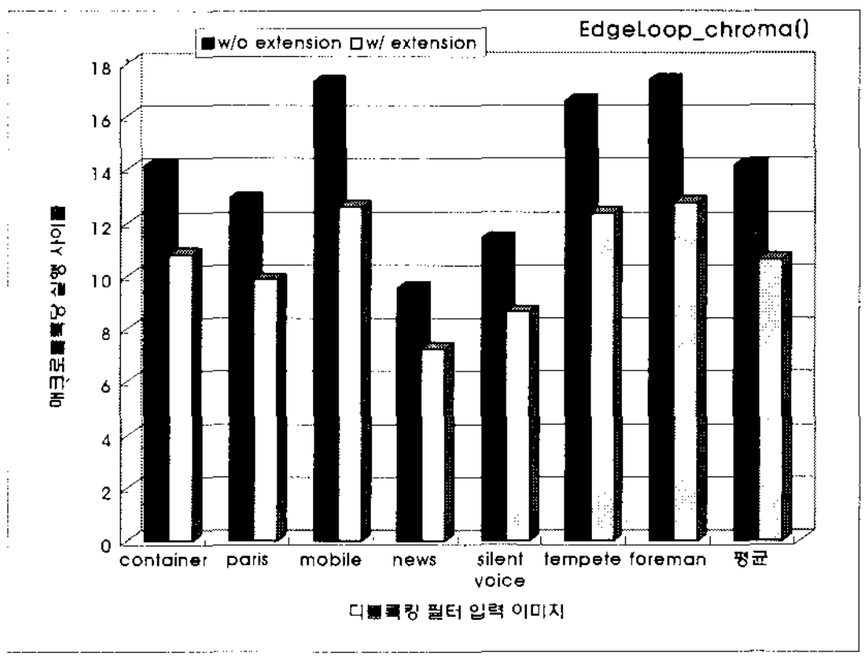

그림 11. EdgeLoop_chroma 함수의 실행 사이클 비교
Fig. 11. comparison of execution cycles for EdgeLoop_chroma.

주요 함수인 DeblockMb, GetStrength, EdgeLoop_luma, EdgeLoop_chroma에 대한 그래프를 각각 그림 8, 그림 9, 그림 10 및 그림 11에 나타내었다.

시뮬레이션 결과는 입력되는 이미지의 특성에 따라 다른 경향을 보였는데, 디블록킹 필터 전체적으로 볼 때 제안하는 명령어 셋을 사용한 경우, 그렇지 않은 경우에 비해 평균 18.18%의 성능향상을 보였다. 함수별로 살펴보면, 먼저 EdgeLoop_luma와 EdgeLoop_chroma함수의 경우 각각 휘도 및 색차 신호의 변화가 심한 foreman과 mobile에서 제안하는 명령어가 많이 사용되어 21.11% 및 27.16%의 높은 성능 향상을 보였고, GetStrength함수의 경우 paris, news 및 silent voice등 정지된 배경을 포함하는 이미지에서 움직임벡터를 비교하는 과정에 제안하는 명령어가 많이 사용되어 높은 성능 향상을 보였으며, 특히 news에서 15.15%로 높은 성능 향상을 보였다. 반면, paris는 정지된 배경으로 인해 GetStrength함수에서는 비교적 높은 성능 향상을 보였

표 2. 함수별 성능 향상 비교
Table 2. Comparison of performance improvements.

함수	성능향상	함수	성능향상
DeblockMb	18.18 %	EdgeLoop_luma	18.53 %
GetStrength	12.60 %	EdgeLoop_chroma	25.21 %

표 3. 설계된 ASIP에 대한 합성 결과 비교
Table 3. Synthesis results for proposed ASIP.

	기준 프로세서	ASIP
총 면적	63990.28	68780.87
동작 지연시간	7.865 ns	8.113 ns
동작 주파수	127.15 MHz	123.26 MHz

지만, 휘도 신호 및 색차 신호의 변화가 거의 없어 필터링이 상대적으로 적게 이루어져 EdgeLoop_luma 및 EdgeLoop_chroma함수에서 가장 낮은 성능 향상을 보였다. 지금까지 살펴본 디블록킹 필터의 주요 함수별 성능 향상을 표 2에 나타내었다.

기준 프로세서와 여기에 제안하는 명령어 셋을 적용하여 ASIP을 구성한 경우에 대하여 CoWare의 Processor Designer를 이용해 HDL을 생성하고 Synopsys의 Design Compiler를 통해 TSMC 0.25㎛ 공정으로 합성하여 이에 대한 면적, 동작 지연시간 및 동작 주파수를 비교하여 표 3에 나타내었다.

IV. 결 론

본 논문에서는 H.264/AVC 표준의 디블록킹 필터를 가속하기 위한 ASIP을 구성하였다. 이 과정에서 필터링 연산을 가속할 수 있는 명령어 셋을 제안하였으며, 프로세서 기반의 시스템에 가장 적합한 [5]의 모델을 바탕으로 제안하는 명령어 셋을 적용하여 VCEG 권장 이미지를 입력으로 한 시뮬레이션에서 평균 18.18%의 성능 향상을 확인하였다. 기준 프로세서를 CoWare의 Processor Designer로 HDL을 생성하고 Synopsys의 Design Compiler를 이용하여 TSMC 0.25um 공정으로 합성한 결과, 제안하는 명령어를 추가할 경우, 약 7.5%의 면적이 증가하였으며 동작 지연시간은 약 3.2% 증가하였다. 동작 지연시간의 증가에 따라 디블록킹 필터의 처리 시간은 실행 사이클에 비해 다소 감소된 11.40%의 성능 향상을 보였다. 면적과 동작 지연시간에서의 이와 같은 증가는 제안하는 명령어 셋으로 인해 성능이 향상되는 정도로 볼 때, 본 논문에서 설계한 ASIP은 충분히 디블록킹 필터를 가속한다고 볼 수 있으며, 기존의 고성능 프로세서에 적용할 경우 더 높은 성능 향상이 기대된다.

참 고 문 헌

[1] JVT, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)," May 2003.

[2] L. E. G. Richardson, "H.264 and MPEG-4 Video Compression," John Willey & Sons, pp. 184-187, Dec. 2003.

[3] Y. W. Huang, T. W. Chen, B.Y Hsieh, T. C. Wang, T.H. Chang, and L.G. Chen, "Architecture Design For Deblocking Filter In H.264/JVT/AVC," International Conference on Multimedia and Expo, pp. 693-696, Baltimore, Maryland, USA, July 2003.

[4] B. Sheng, W. Gao, and D. Wu, "An Implemented Architecture of Deblocking Filter for H.264/AVC," International Conference on Image Processing, pp. 665-668, Singapore, Oct. 2004.

[5] 이형표, 이용석, "H.264/AVC 비디오 코덱을 위한 효율적인 자료 재사용 디블록킹 필터 알고리즘," 전자공학회논문지, 제44권 CI편, 제6호, 30-35쪽, 2007년 11월

[7] K. Suhring, H.264/AVC software JM11.0, http://iphome.hhi.de/suehring/tml/, Nov. 2006.

[8] G. Sullivan and G. Bjontegaard, "Recommended Simulation Common Conditions for H.26L Coding Efficiency Experiments on Low-Resolution Progressive- Scan Source Material," ITU-T VCEG, Doc. VCEG-N81, Sep. 2001.

[9] A. Hoffmann, "A Novel Methodology for the Design of Application Specific Integrated Processors (ASIP) Using a Machine Description Language," IEEE Trans. on Computer Aided Design, vol.XX, no.Y, Month 2001

[10] A. Hoffmann, "A Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using the Machine Description Language LISA," International Conference on Computer-Aided Design, pp.625-630, San Jose, CA, USA, Nov. 2001

[11] CoWare Inc., "LISA 2.0 Language Tutorial," http://www.coware.com

[12] J.L. Hennessy and D.A. Patterson, "Computer Architecture, a Quantitative Approach," Morgan Kaufmann Publishers, pp.A1-A37

[13] T. Glokler and H. Meyr, "Design of Energy-Efficient Application Specific Instruction-Set

Processors (ASIPs)," Kluwer Academic Publishers, pp.117-143

저 자 소 개

이 형 표(학생회원)

2001년 건국대학교 전자공학과 학사 졸업.
2008년 연세대학교 전기전자 공학과 석사 졸업
2001년~현재 삼성전자 DM총괄 선임연구원

<주관심분야 : 영상신호처리, SoC 설계>

이 용 석(평생회원)

1973년 연세대학교 전자공학과 학사 졸업.
1977년 University of Michigan Electrical Engineering 석사 졸업.
1981년 University of Michigan Electrical Engineering 박사 졸업.
1993년~현재 연세대학교 전기전자공학과 교수.

<주관심분야 : 마이크로프로세서 설계, VLSI 설계, DSP 프로세서 설계, 고성능 연산기 설계>