論文99-36C-4-3

VHDL을 이용한 프로그램 가능한 스택 기반 영상 프로세서 구조 설계

(Design of Architecture of Programmable Stack-based Video Processor with VHDL)

朴 柱 炫 *, 金 榮 民 **

(Ju-Hyun Park and Young-Min Kim)

요 약

본 논문의 주요 목표는 고성능 SVP(Stack-based Video Processor)를 설계하는 것이다. SVP는 과거에 제안된 스택 머신과 영상 프로세서의 최적의 측면만을 선택함으로써 더 좋은 구조를 갖도록 하는 포괄적인 구조이다. 본 구조는 객체 지향형 프로그램이 소규모의 많은 서브루틴을 가지고 있기 때문에 스택 버퍼를 갖는 준범용 S-RISC(Stack-based Reduced Instruction Set Computer)를 이용하여 객체 지향형 영상 데이터를 처리한다. 그리고 MPEG 부호화 속도를 증가시킬 수 있도록 벡터 프로세서를 내장하고 있다. 벡터 프로세서는 MPEG-4의 반화소 단위 처리와 고급 모드 움직임 보상, 움직임 예측, SA-DCT(Shape Adaptive-Discrete Cosine Transform)가 가능하며, 절대값기, 반감기를 가지고 있어서 부호화기로 확장할 수 있도록 하였다. SVP는 0.6㎛ 3-메탈 계층 CMOS 표준 셀 기술을 이용하여 설계되었으며, 110K 로직 게이트와 12Kbit SRAM 내부 버퍼로 이루어지고 50 MHz의 동작 속도를 가진다. MPEG-4의 VLBV(Very Low Bitrate Video) 최대 전송율인 QCIF 15fps(frame per second)로 영상 재생 알고리즘을 수행한다.

Abstract

The main goal of this paper is to design a high performance SVP(Stack based Video Processor) for network applications. The SVP is a comprehensive scheme; 'better' in the sense that it is an optimal selection of previously proposed enhancements of a stack machine and a video processor. This can process effectively object-based video data using a S-RISC(Stack-based Reduced Instruction Set Computer) with a semi-general-purpose architecture having a stack buffer for OOP(Object-Oriented Programming) with many small procedures at running programs. And it includes a vector processor that can improve the MPEG coding speed. The vector processor in the SVP can execute advanced mode motion compensation, motion prediction by half pixel and SA-DCT(Shape Adaptive-Discrete Cosine Transform) of MPEG-4. Absolutors and halfers in the vector processor make this architecture extensive to a encoder. We also designed a VLSI stack-oriented video processor using the proposed architecture of stack-oriented video decoding. It was designed with 0.5㎛ 3LM standard-cell technology, and has 110K logic gates and 12 Kbits SRAM internal buffer. The operating frequency is 50MHz. This executes algorithms of video decoding for QCIF 15fps(frame per second), maximum rate of VLBV(Very Low Bitrate Video) in MPEG-4.

* 正會員, 韓國電子通信研究院
(Electronics and Telecommunications Research Institute)

** 正會員, 全南大學校 電子工學科
(Dept. of Electronics Engineering, Chonnam National University)

接受日字:1998年10月23日, 수정완료일:1999年3月23日

Ⅰ. 서 론

21세기를 앞둔 정보화 사회에서 광대역 통신망을 이용한 멀티미디어 정보의 공유는 필수적인 기술로 인식되고 있다. 정보고속도로가 가정까지 도달하여 고성능 PC에 연결되어 다양한 멀티미디어 정보를 실시간에 제공받을 수 있으며, 일반 사용자 정보를 전송할 수도 있는 양방향 통신 환경으로 발전해 가고 있다. 그 중에서도 영상 정보의 효율적인 처리가 멀티미디어 서비스의 중심에 있음은 부인할 수 없는 사실이다. 특히 인터넷을 이용한 다양한 서비스의 출현은 21세기 최대의 멀티미디어 서비스 환경으로 판단된다. 인터넷 사용이 보편화되면서 웹 브라우저가 모든 컴퓨팅 플랫폼에서 지원하는 범용 GUI로 자리를 잡아가면서 서로 상이한 독립적인 플랫폼간의 정보 접근을 위해 기존의 표준에 덧붙여 새로운 표준의 필요성이 제기되었다. 인터넷의 서비스 범주가 된 영상, 음성 데이터들은 종래의 마이크와 카메라에 의한 자연 영상, 음성의 단순한 녹음, 녹화 뿐만 아니라 컴퓨터 그래픽스 기술을 이용한 인공 데이터의 보편화에 따른 다른 형태의 데이터들 간의 자유로운 상호 편집이 가능한 기술 표준의 필요성을 앞당기게 되었다[1].

이와 같이 인터넷을 이용한 서비스의 성숙과 영상 정보에 대한 패러다임의 변화로 영상 편집 및 생성, 전송, 접근을 포함하는 통합 표준 환경이 필요하게 되었으며, 이는 MPEG-4 표준화를 출범시키는 계기가 되었다. 기존의 화면 기반 부호화와는 달리, 객체(object)를 기반으로 한 부호화를 지향하며, 표준안에서 MPEG-J와 같이 신택스(syntax) 기술 언어로 C++이나 자바(Java)와 같은 객체 지향적 언어를 사용하거나 응용할 수 있도록 권고하고 있듯이 소프트웨어에 의한 구현을 염두에 둔 표준이기 때문에 융통성 있는 프로세서가 필요하게 되었으며, 자바 클래스를 다운로드함으로써 새로운 기능을 추가할 수 있는 자바는 MPEG-4 재생기의 소프트웨어 관리 측면에서 대단히 매력적인 것이라고 할 수 있다. 그러나 자바와 같은 순수한 스택 머신은 일반적인 프로그램 스타일로 작성된 전통적인 프로그램을 실행할 때 레지스터 머신만큼 빠르게 수행할 수 없으며, 특히 영상 데이터와 같은 병렬 처리를 필요로 하는 데이터를 스택 기반 구조로 처리하기엔 실시간 처리가 매우 어렵다. 또한 자바 실행을 위한 기저 하드웨어 구조들을 최적화하는 것만으로 자바가 가지고 있는 문제를 해결할 수는 없다. C 응용과 같은 다른 환경을 갖는 경우에 성능에 영향을 미칠 수 있으며, 서로 다른 응용 환경 및 성능 수준에서 구조의 연속성이 필요할 경우는 자바 전용 프로세서를 사용하는 것이 최선이 아닐 수 있기 때문이다. 따라서 스택 머신에서의 전통적인 언어를 수행하기 위해서는 고성능의 레지스터 머신에 근접할 수 있는 하드웨어를 추가해야 한다.

본 논문에서는 네트워크를 통해 수신된 객체 및 프로그램 데이터를 전용 프로세서로 처리하는 방법을 사용한다. 기존의 방법은 번역기를 통해 범용 CPU에서 처리하기 때문에 코드를 번역하는데 많은 시간이 소요된다. JITC가 없는 시스템에서는 통상적으로 1fps이하의 성능을 갖지만, 기존의 브라우저를 이용할 경우 75MHz이상의 CPU를 내장한 Pentium PC라면 큰 문제가 없다. 그러나 시스템 성능은 다소 향상되지만 영상 데이터와 같이 고속 및 병렬 처리를 필요로 하는 정보는 CPU 점유율이 높기 때문에 성능 향상에 한계가 있다[2]. 이러한 한계를 극복하기 위한 방법은 첫째, 부족한 기능을 채우기 위한 영상 처리 라이브러리를 확장한다. 둘째, 자바 프로그램이 외부 프로그램과 통신할 수 있도록 내부 명령어를 추가하는 방법으로, 대부분의 수행 시간을 클래스 라이브러리에서 충분히 사용하도록 프로그램을 작성하는 방법이다. 그러나 현재 자바 API에는 DCT, 재생 툴과 같은 영상 비트열을 처리하는 클래스가 존재하지 않는다는 한계가 있다.

또한 피코자바와 같은 자바 전용 프로세서가 객체 정보 처리에 있어 범용 CPU에 비해 10~20배 이상 성능이 뛰어나지만 객체 정보의 효율적인 처리를 위한 스택 지향형 구조이기 때문에 고압축으로 수신된 영상 데이터를 재생하기에는 많은 한계가 있으므로 영상 재생 알고리즘을 전용으로 처리할 블록을 내장하는 전용 프로세서를 사용하는 것이 속도 및 효율성 면에서 적당하다고 판단된다. 따라서, 본 논문에서는 영상 데이터의 빠른 재생을 위한 새로운 전용 프로세서 구조를 제안하고, 검증하였다.

본 논문의 Ⅱ장에서는 프로세서 설계에 따른 네트워크를 통한 영상 정보 흐름 및 설계 조건을 살펴본다. Ⅲ장에서는 본 논문의 프로세서 구조에 대하여 설명하고, Ⅳ장에서는 시뮬레이션을 통해 확인된 성능을 평가하고 분석한다. 마지막으로 Ⅴ장에서는 본 연구의 결과를 기술하고 향후 연구방향을 제시한다.

Ⅱ. 네트워크 정보 흐름 및 설계 조건

1. 프로세서 내로의 네트워크 정보 흐름

인터넷으로 대표되는 네트워크를 기본 환경으로 한 서버-클라이언트 구조는 가장 흔하게 사용될 수 있는 통신 환경이면서 강력한 환경을 제공해 준다[3]. 자바와 같은 객체 지향 성격을 갖는 언어는 인터넷을 통해 다운로드(downloadbility), 디코딩 툴과 알고리즘의 다운로드를 위한 프레임워크가 쉽게 시뮬레이션되고, 구현될 수 장점을 갖는다. 따라서 우리는 이와 같은 다운로드가 가능한 프로세서를 설계한다. 그림 1.은 객체, 프레임 정보, 프로그램 스트림, 영상 데이터 스트림이 영상 재생기에 어떻게 입력되고 있는 지를 보여주고 있다.

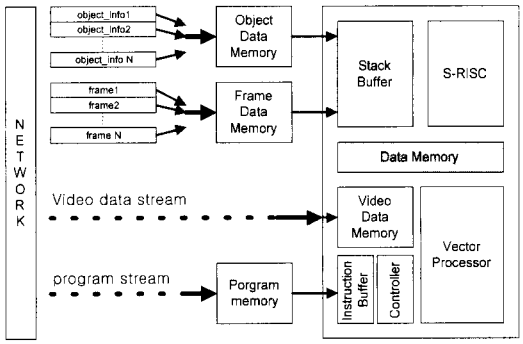


그림 1. 프로세서 내로의 네트워크 정보 흐름
Fig. 1. Network information flow fed into a processor.

자바는 클래스 파일에 객체와 프레임 정보를 가지고 있다[4]. 객체는 메소드(method)와 변수(variable)들로 구성되며, 모든 객체 정보는 포인터로 처리된다. 또한 영상 데이터는 서버에 저장되어 있다가 바이트 코드가 해석되면서 필요에 따라 클라이언트로 전달된다. 물론 지역(local) 데이터는 터미널 메모리에 저장된다. 수행 환경을 제공하는 프레임 정보는 메소드 어레이에 대한 인덱스, 속성(attribute) 어레이에 대한 인덱스, 레퍼런스 지역 변수, 최상위 오퍼랜드 스택, 클래스 파일의 포인터 등을 가지고 있다.

객체 지향형 언어와 같은 멀티태스킹, 서브 루틴 등이 많은 프로그램을 구현하는데는 데이터를 접속하거나 리턴 어드레스를 접속하는데 프로그램 메모리 사이클이 필요치 않는 다중 스택 버퍼가 프로세서 속도를 올릴 수 있는 방법이다. 따라서 본 논문의 스택 버퍼는 데이터 스택, 리턴 스택으로 구성되며, 프로세서 외부 데이터 메모리에 저장되어 있는 객체, 프레임 정보를 입력받는다[5]. 단일 스택의 경우 하드웨어는 단순하지만 리턴 어드레스와 데이터 파라미터를 혼합하는데 많은 비용이 들고, 모듈화된 객체 정보를 처리할 때 오버헤드가 발생하기 때문에 리턴 어드레스를 저장하는데 리턴 스택을 사용하고, 프로그램에서 사용하는 지역 변수 및 파라미터 값은 데이터 스택에 저장한다. 그 이외에 데이터 메모리 일부를 스택 메모리로 사용하고, 이에 필요한 스택 포인터를 둠으로써 이전에 호출되었던 메소드와 지역 변수에 대한 포인터, 오퍼랜드 스택의 시작 부분과 끝부분을 가리키는 포인터 등에 관한 정보를 저장한다. 프레임 정보에 있던 코드 속성(code_attribute)의 PC(Program Counter)값에는 현재 클래스 파일 내에 있는 수행 명령어 시퀀스의 포인터 값이 저장된다. 따라서 이 PC값을 참조해서 프로그램 메모리에 프로그램을 저장한다.

또한 프로세서 내 영상 데이터 스트림을 저장하는 메모리는 3개의 독립적인 메모리로 구성된 영상 데이터 메모리와, 프로세서를 제어하는 S-RISC, 영상 알고리즘을 수행하는 VP(Vector Processor)등이 있다. 스택 기반 명령어와 레지스터 기반 명령어를 동시에 고속으로 패취하기 위한 가변 길이 명령어를 저장하는 명령어 버퍼를 가지고 있으며, 데이터 메모리의 일부는 4개의 스택 포인터로 어드레싱되는 스택 메모리로 사용된다.

2. 설계 조건

S-RISC는 32비트 고정소수점 연산을 한다. 영상 재생에는 통상적으로 16비트 정수 연산을 사용하지만 32비트 연산을 하는 이유는 다음과 같다. 첫째, MPEG-4의 모양 정보 부호화의 CAE(Context Arithmetic Encoding) 연산에는 최소 17비트 이상의 대역폭을 필요로 한다. 둘째, DCT 계수값의 고정소수점 처리에 따른 영상의 찌그러짐을 방지하기 위해 32비트의 누적 레지스터(accumulator)를 사용하므로 이 데이터를 처리하기 위해 32비트 연산을 한다. 셋째, 데이터 스택 버퍼에 저장되는 지역 변수, 상수, 매개변수 포인터 등이 32비트이다. 그러나 객체 지향 프로그램은 32비트 뿐만 아니라 8비트, 16비트, 64비트 변수를 포함하며, 따라서 이를 효과적으로 처리하기 위한 구조가 필요하며, S-RISC내의 레지스터 뱅크는 영상

재생 과정에서 발생하는 패킹(packing)된 연산 결과를 효과적으로 저장할 수 있는 16개의 레지스터로 구성된다. 본 논문의 VP의 연산 결과가 16비트×4, 32비트×4이고, 벡터 명령어의 경우 3-오퍼랜드 명령어 코드를 사용하기 때문에 동시에 최대 3개의 128비트 값을 접속할 수 있어야 한다. 따라서 최소 32비트 길이 12개 이상의 레지스터가 필요하다.

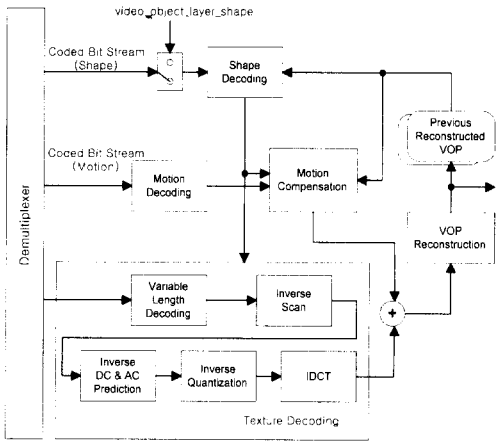


그림 2. MPEG-4 VM의 VOP 기반 부호화기 구조
Fig. 2. VOP based coder structure in the MPEG-4 Video VM.

VP는 영상 알고리즘의 재생 과정에서 발생하는 병렬 및 누적 연산을 주로 한다. IDCT의 입력이 12비트이므로, 본 논문에서는 4개의 16비트 단위의 벡터 연산을 수행하는 구조를 채택한다. 그 이유는 첫째, MPEG 영상 데이터의 입출력은 외부 DRAM과 프로세서 사이에 매크로블록 단위로 이루어지며, 통상적으로 64비트 버스를 사용한다[6]. 따라서 내부 영상 데이터 메모리가 64비트 단위 입출력이 가능하다면 VP에서는 64비트 데이터를 동시에 처리해야 하며, 이는 최소 16비트 단위의 4개 연산 블록이 필요함을 의미한다. 둘째, 본 논문은 실시간 영상 재생을 목표로 하기 때문에 MPEG-4 권고안인 VLBV 15fps(QCIF)를 만족하기 위해 최소 4개의 MAC 유닛이 필요하다. 이는 본 논문의 16비트 곱셈기의 지연 시간이 18.0ns이므로 MAC 유닛은 2단 파이프라인이 되도록 한다. 실시간 재생은 400ms 이내에 영상이 처리되어야 하며, 재생기에서 가장 많은 CPU 점유율을 갖는 것은 IDCT로, 약 35%를 차지하므로 140ms이내에 IDCT를 수행하도록 한다[7]. 따라서 최소 4개의 MAC 연산부가 필요하다.

스택 버퍼는 총 64개의 스택 요소를 가지며, 데이터 스택 버퍼와 리턴 어드레스 스택 버퍼에 각각 32개씩 할당된다. 단일 스택 버퍼를 사용하면 하드웨어는 단순해지지만 리턴 어드레스와 데이터 매개변수를 혼합하는데 많은 비용이 들고, 모듈화된 객체 정보를 처리할 때 오버헤드가 발생한다. 또한 객체 지향 코드로 작성된 프로그램은 C보다 2배이상의 클래스로 구성되며, 따라서 다량의 근거리 분기 동작을 필요로 한다. 또한 멀티태스킹 환경에서 태스크 교환을 원활히 하기 위해서는 최소 4개의 독립적인 태스크를 저장할 스택 버퍼가 필요하다[8]. 따라서 객체 지향형 언어와 같은 멀티태스킹, 서브루틴 등이 많은 프로그램은 데이터를 접속하거나 리턴 어드레스를 접속하는데 프로그램 메모리 사이클이 필요치 않는 다중 스택 버퍼를 사용하므로써 프로세서 속도를 올릴 수 있다. 본 논문에서는 모듈화 객체 정보를 처리할 때 오버헤드를 줄이기 위해 매개변수, 지역 변수 등의 데이터와 리턴 어드레스를 따로 저장하는 다중 스택 구조를 사용한다. 스택 버퍼는 32개의 스택 요소로 구성하는데, 일반적인 프로그램에서 24개를 기준으로 스택 정보의 1%가 오버플로우되기 때문에 32개의 스택 요소는 오버플로우에 의한 데이터 누출을 방지할 수 있다. 또한 데이터 메모리 일부를 스택 메모리로 할당하여 4개의 태스크 블록으로 나누고, 4개의 스택 포인터를 이용하여 각각을 접속한다. 또한 객체 지향 코드는 스택을 이용한 지역 변수 혹은 상수의 이동이 많기 때문에 스택의 접속이 빈번하여 최상위 요소만을 접근할 수 있을 경우 하드웨어 단순화, 수행 속도 면에서 복잡하게 만드는 결과를 초래하기 때문에 본 논문에서는 2번째 요소까지 접근이 가능하도록 하였다.

명령어 버퍼는 다량의 근거리 분기 동작을 수행하는 객체 지향 프로그램의 특성에 따라 근거리 분기가 일어날 경우 외부 메모리를 참조하지 않고도 프로세서 내에서 분기할 수 있는 장점을 가진다. 일반적인 프로그램에서 분기 동작은 모든 명령어의 약 15%가량을 차지하며, 객체 지향 프로그램의 경우는 훨씬 더 많기 때문에 시스템 성능에 중요한 영향을 미칠 수 있다.

통계적으로 근거리 분기가 빈번히 일어나는 명령어를 수행할 때 분기의 20-30%가 근거리 후방향(backward) 분기(96바이트 이하)이며, 10-20%가 근거리 전방향(forward) 분기(32바이트 이하)가 일어나

는 것으로 알려져 있다. 버퍼 크기는 전형적인 명령어 순서열의 통계적인 분석에 따라 후방향, 전방향 분기를 모두 고려하여 64바이트, 즉 32개의 명령어를 포함할 수 있도록 설계되었다. 본 논문에서는 가변 길이 명령어 코드를 사용하므로 버퍼에는 32~64개 명령어를 포함할 수 있으며, 이에 따라 약 20% 이상의 프로그램 메모리를 절약하는 효과가 발생한다.

Ⅲ. SVP 구조 설계

1. 개요 및 구성[9]

SVP 구조는 그림 3.과 같으며 크게 6개의 블록으로 구성된다.

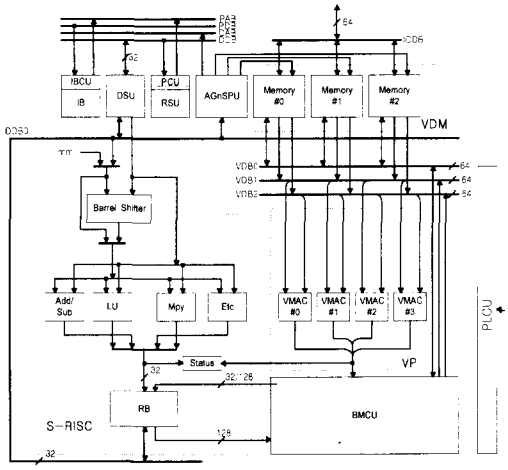


그림 3. SVP 구조
Fig. 3. SVP architecture.

SVP의 각 블록은 코어의 주요 연산을 처리하는 ALU와 레지스터 뱅크로 구성된 S-RISC, 영상 데이터와 같은 벡터 연산을 병렬로 수행하는 VP, 영상 데이터를 저장하는 VDM(Video Data Memory), 객체 지향적 데이터를 효율적으로 처리하고, 서브루틴 동작을 고속으로 처리하기 위한 스택 버퍼(SU, Stack Unit), 명령어의 우선패취를 위해 명령어 코드를 저장하는 명령어 버퍼(IB, Instruction Buffer), 전체 코어의 파이프라인 동작과 명령어 패취, 각종 번지 발생 등을 제어하는 제어기(Controller) 등으로 구성된다.

제어기는 전체 파이프라인을 제어하는 PLCU (PipeLine Control Unit), 프로그램 메모리의 어드레스를 연산하는 PCU(PC Unit), 스택 메모리와 DSU 사이의 데이터 송수신을 제어하는 AGnSPU(Address Generation & Stack Pointer Unit), S-RISC, VP와 내부 영상 메모리 사이의 데이터 이동을 제어하는 BMCU(Bus Mask Control Unit)로 이루어진다.

SVP 버스는 6비트, 32비트, 64비트, 128비트 버스로 나눈다. 6비트 버스는 VDM내 M0, M1, M2에 대한 어드레스를 제공한다. VDB0(Vector Data Bus 0), VDB1, VDB2는 64비트 버스로, VDM과 레지스터 뱅크, VDM과 VP 사이의 데이터를 전송한다. DDB0(Data Data Bus 0)는 모든 레지스터 및 레지스터 뱅크와 연결되어 내부 데이터를 전송하며, 레지스터간의 데이터 이동 및 스택과 레지스터간, SbRISC 입력 및 레지스터 뱅크의 입력으로 사용된다. PAB(Program Address Bus), PDB(Program Data Bus)는 제어기에서 수행되는 외부 프로그램 메모리의 어드레스와 데이터를 입, 출력하는 버스이며, DAB(Data Address Bus), DDB는 외부 데이터 메모리의 어드레스와 데이터를 제어기에 입, 출력시키는 버스이다. 또한 IODB(Input/Output Data Bus)는 VDM 내부에 있는 영상 데이터를 DRAM과 같은 외부 프레임 메모리로부터 프로세서 내부로 입, 출력하기 위한 버스이다.

SVP는 IF(Instruction Fetch), ID(Instruction Decode), RF(Read Forward), EX(EXecute), WB(Write Back) 등 5단의 명령어 파이프라인을 가지고 있다[10]. IF 단계는 우선패취에 의해 외부 프로그램 메모리로부터 입력된 명령어 시퀀스를 저장된 명령어 버퍼에서 패취한다. ID 단계는 IF 단계에서 입력받은 명령어 코드를 디코딩한다. 각 명령어 코드는 32비트이다. 이 단계에서는 각 명령어의 PC 연산 블록 제어, 우선분기 처리[11], 그리고 RF 단계에 필요한 모든 제어 신호를 발생한다. RF 단계는 메모리를 접근하는 명령어의 경우 어드레스의 연산 과정이 필요하며, 이때 단일 사이클에 어드레스를 생성한다. 벡터 명령어중 오퍼랜드 읽기 동작으로 VDM을 읽을 경우 내부 메모리를 읽기 위한 어드레스 발생과 함께 64비트 데이터를 읽어서 파이프라인 레지스터로 읽어온다. EX 단계는 명령어 수행을 통해 데이터를 처리하는 단계이며, WB 단계는 레지스터 뱅크에 데이터를 저장하거나 외부 데이터 메모리, 내부 VDM 등에 데이터를 저장하는 단계이다.

SVP 명령어는 크게 4개 그룹으로 나눈다. 덧셈,

뺄셈, 곱셈 및 분기나 리턴 명령어를 수행하는 G(General)그룹, 데이터 메모리와 레지스터 뱅크간, 내부 메모리와 레지스터 뱅크간의 데이터 전송을 수행하는 M(Memory Reference)그룹, 다양한 스택 동작을 수행하는 S(Stack Manipulation)그룹, 벡터 연산을 수행하는 V(Vector Processing)그룹 등이다. V 그룹을 제외한 모든 명령어 그룹은 DTOS(Top Of Data Stack) 레지스터를 하나의 출발지 오퍼랜드로 사용하기 때문에 또다른 출발지 오퍼랜드 1개와 목적지(Destination) 오퍼랜드 1개 등 2개 오퍼랜드로 모든 연산을 수행하는 반면, V그룹은 3개의 오퍼랜드를 갖는다. 전체 명령어 개수는 63개이며, G, M 그룹 명령어는 명령어 특성에 정수형, 문자형, 바이트형 데이터 처리를 지원하는 57개의 확장 명령어를 갖는다. G 그룹중 덧셈, 뺄셈, 곱셈, 비교 명령어 등은 문자형 데이터와 같이 부호없는 데이터도 처리할 수 있으며, 객체형 데이터는 객체 어드레스로서 정수형 데이터와 같은 형태로 취급한다. M 그룹 명령어 중에는 벡터 단위 영상 데이터 뿐만 아니라 픽셀 단위 데이터 처리가 가능하도록 내부 메모리로부터 8비트, 16비트 데이터를 레지스터에 가져올 수 있는 픽셀 명령어가 존재한다. 또한 S 그룹 명령어는 DS내 데이터의 상호 이동, 복사 등의 동작을 한다. 명령어 마이크로코드는 기본적으로 32비트이고, S그룹만 16비트이며, 23개의 포맷 형태를 갖는다. 명령어 구성 비율은 표 1.과 같다.

표 1. 명령어 구성 비율
Table 1. Ratio of instruction composition.

명령어	차지하는 비율(%)
연산 명령어	42
분기 명령어	5
메모리 명령어	6
스택 명령어	18
벡터 명령어	29

연산 명령어는 다른 명령어 비해 차지하는 비율이 상대적으로 높다. 그러나 오퍼랜드에 스택과 레지스터가 올 수 있기 때문에 스택내의 데이터를 처리할 때는 스택 명령어로 분류할 수 있으므로 스택형 명령어가 차지하는 비율은 최고 70%라고 할 수 있다. V그룹 명령어는 레지스터 뱅크를 패킹할 수 있는 명령어를 포함한다. G 그룹 명령어는 오퍼랜드 데이터 형태에 따라 확장 명령어를 갖는다. C 프로그램 코드에서는 다양한 데이터 형태를 지원함으로써 다른 구조에서도 호환이 가능하도록 하는 장점을 지니고 있다. 따라서 본 연구에서 확장 명령어는 객체 지향 프로그램 및 C 언어 응용이 가능하도록 다양한 형태의 데이터를 지원한다. 지원 가능한 데이터 형태는 byte, short, int, char 형등을 지원하며, 각 형태에 따른 니모닉은 B, S, I, C이다[12]. long형은 int형을 이용해 계산을 할 수 있다. 곱셈의 경우 int형 데이터를 이용해 long형 데이터를 계산할 때 부호있는 데이터와 부호없는 데이터의 곱셈이 불가피하므로 이를 해결하기 위해 곱셈 명령어는 혼합 모드를 지원한다.

2. S-RISC

S-RISC는 SVP의 벡터 연산을 제외한 모든 연산 및 논리 동작을 수행하며, 구조는 그림 4.와 같다. S-RISC는 크게 덧셈기(Adder), 논리연산기(LU, Logic Unit), 기타 연산(Etc) 블록으로 구성되는 ALU, 곱셈기(MPY), 배럴 쉬프터(Barrel shifter), 레지스터 뱅크(RB, Register Bank)로 구성된다.

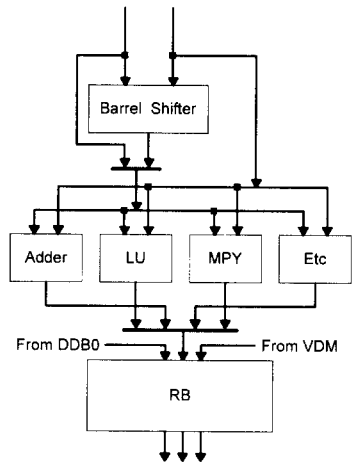


그림 4. S-RISC 구조
Fig. 4. S-RISC architecture.

LU는 AND, OR, XOR, NAND 기능이 가능한 32비트 로직으로 구성된다. 곱셈기는 16비트 부호있는 곱셈 연산과 부호없는 곱셈 연산을 수행하며, 32비트 출력을 갖는다. 배럴 쉬프터는 32비트까지 논리, 산술 연산 쉬프트를 할 수 있으며, 일반 쉬프트 동작과 함께 덧셈기, 곱셈기 블록의 상단에서 연산에 사용되는 오퍼랜드를 변환하는 역할을 한다. 레지스터 뱅크는

S-RISC 연산과 VP 연산에 사용할 16개의 32비트 레지스터로 구성된다. 레지스터 뱅크는 4개의 레지스터 모듈로 구성되며, 한 모듈은 각각 4개의 32비트 데이터나 2개의 64비트 데이터를 저장한다.

S-RISC의 모든 블록은 기본적으로 32비트 데이터 오퍼랜드를 취급하며, 확장형 명령어 필드에 따라 8비트형, 16비트형, 혹은 부호없는 32비트 데이터로 변환되어 입력된다. ALU의 출력은 배럴 쉬프터, 덧셈기, 논리연산기, 곱셈기 출력 중 하나이다. 또한 ALU는 N(Negative), Z(Zero), V(oVerflow), C(Carry) 등 4개의 상태값을 출력한다.

곱셈은 부호있는 곱셈과 부호없는 곱셈 방법이 서로 다르고, 계산 결과 값 또한 다르기 때문에 각각 16비트 부호없는 곱셈기와 이를 이용한 16비트 부호있는 곱셈을 한다. 또한 필요에 따라 부호없는 곱셈과 부호있는 곱셈을 따로 계산할 수 있기 때문에 연산 속도를 높일 수 있는 장점이 있다. 또한 혼합 모드 곱셈을 지원하기 위해 각 입력에 대해 부호있는(signed) 경우와 부호없는(unsigned) 경우에 대해 데이터 변환을 한다. 변환된 16비트는 보수화(Complement) 연산을 하며, 16비트 곱셈을 한 후 부호 여부를 가리키는 신호에 따라 32비트 결과를 나타내며, Signed×Signed, Unsigned×Unsigned, Signed×Unsigned, Unsigned×Signed 곱셈을 지원한다.

레지스터 뱅크는 4개의 모듈로 구성되며, 각 모듈은 4개의 32비트 레지스터 요소(RE, Register Element)로 구성된다. 각 모듈은 2개의 64비트 레지스터로도 사용할 수 있다. 또한 각 모듈은 VP로부터 입력된 128비트 데이터를 64비트로 저장하기 위한 패킹 기능을 수행한다[13].

레지스터 뱅크의 포트 수는 연산의 소스 오퍼랜드를 제공하기 위한 4개의 읽기 포트와 연산의 결과값을 저장하기 위한 3개의 쓰기 포트를 갖는다. 레지스터 뱅크는 32비트 16개에 ALU 결과를 저장하고, DDB0에 32비트 데이터를 출력하며, VP 연산 결과 및 VDM 데이터를 저장할 때는 8개의 64비트 레지스터로 사용된다. 패킹 모드는 영상 데이터 처리에서 레지스터 뱅크 및 메모리를 효율적으로 활용하기 위함이다. 일반적으로 영상 데이터 값은 16비트이며, 32비트 워드를 갖는 레지스터 뱅크에서는 레지스터의 낭비를 가져온다. 따라서 패킹 동작을 통해 레지스터 뱅크를 효율적으로 사용한다.

3. VP

VP는 DCT, MC, 필터링, 샘플링 등 영상 데이터를 처리하며, VMAC(Vector Multiply-ACcumulator)을 이용하여 4개의 16비트 데이터를 동시에 처리하는 벡터 연산 구조이다. VP는 VDM이나 레지스터 뱅크에서 출력된 두 개의 64비트를 입력으로 받으며, VMAC마다 명령어에 따라 레지스터 뱅크나 어큐뮬레이터, VDM으로 16, 32비트를 출력한다. VP 구조는 그림 5.와 같다.

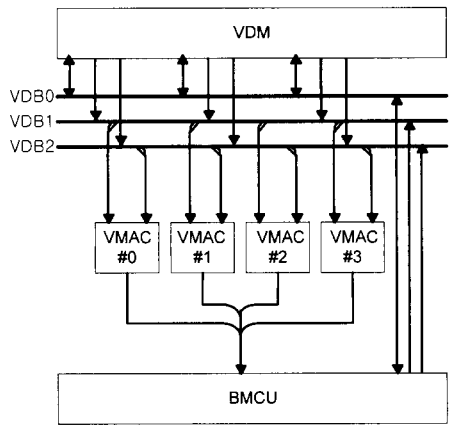


그림 5. VP 구조
Fig. 5. VP architecture.

VP는 64비트 데이터를 4개의 VMAC에서 연산한 후 버스나 어큐뮬레이터에 출력한다. 출력은 각 VMAC마다 16, 32비트를 출력하며, 누적 연산이 필요할 때는 VMAC 내에 있는 32비트 어큐뮬레이터 레지스터에 각각 저장된다. 어큐뮬레이션이 일어나지 않는 경우 상태 값은 EX 사이클에서 저장되며, 어큐뮬레이션이 일어나는 경우는 WB 사이클에서 일어난다.

VMAC(Vector Multiply-ACcumulator)은 덧셈, 뺄셈을 수행하는 VAdder(Vector Adder), 절대값, 반감값을 연산하는 VABS/VHAF(Vector ABSsolute/Vecot HAlF), 곱셈을 수행하는 VMPY(Vector Multiplier), 32비트 어큐뮬레이터인 VACC(Vector ACCumulator), 어큐뮬레이터의 입력을 선택하는 MuxSFT(Multiplex-Shift)로 구성된다. VMAC의 구조는 그림 6.과 같다.

입력은 VDB1, VDB2를 통하여 레지스터 뱅크나 VDM으로부터 입력된다. VACC는 32비트로 많은 수

의 MAC 연산에도 오버플로우가 발생하지 않으며, 따라서 스케일링 작업이 필요없다. VAdder, VMPY, VABS/VHAF 등은 부호 있는 8, 16비트 연산을 하며, MuxSFT는 32비트 VACC와 16비트 레지스터 뱅크의 입력으로 사용할 수 있도록 출력값을 선택한다. VMAC은 16비트인 두 개의 입력을 받아 연산결과인 32비트 데이터와 상태 값인 Z, V를 출력한다. 어큐뮬레이션이 일어나는 경우는 중간에 파이프라인 되어 연산 결과를 출력하는데 두 사이클이 필요하며, 출력은 어큐뮬레이션 레지스터에 저장한다.

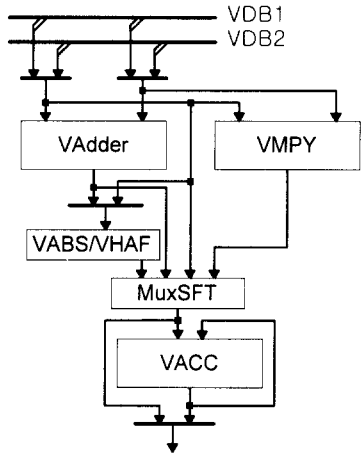


그림 6. VMAC 구조
Fig. 6. VMAC architecture.

VAdder 출력은 VABS/VHLF의 입력으로도 사용될 수 있어 또 다른 연산을 만들어 낸다. 덧셈과 반감값 연산을 이용하여 두 입력의 평균 및 반화소 연산을 하며, 뺄셈과 절대값 연산을 이용하여 SAD (Summation of Absolute Difference)를 구할 수 있다. VMPY는 부호있는 연산만 하기 때문에 부호에 대한 판단을 하지 않는다. VMPY의 2의 보수화 선택은 입력되는 16비트 데이터의 부호 비트인 최상위 비트에 의해 이루어진다. VABS/VHAF는 8, 16비트 절대값과 반감값을 연산한다. VP에서는 동시에 4개의 16비트 절대값/반값 또는 8개의 절대값/반값 연산을 수행한다. MuxSFT는 VMAC의 VAdder, VMPY, VABS/VHAF 연산 결과를 VACC나 레지스터 뱅크의 입력으로 사용할 수 있도록 32비트 데이터 형태로 패킹해제(unpacking)를 한다. VACC는 VAdder와 어큐뮬레이션 레지스터 VAC로 구성된다. 대부분의 경우 MAC 연산에서는 하위나 상위 16비트를 선택해서 사용하나 본 VP내의 VMPY는 32비트 출력을 모두 사용한다. 이는 DCT 연산과 같은 높은 해상도를 필요로 하는 연산에서 유효 자리수를 확보하기 위해서이다. DCT, IDCT의 계수 행렬 값은 1보다 크지 않는 실수 값들로 이루어져 있기 때문에 부동소수점 연산을 하면 IDCT이후 원 영상과의 오차를 줄일 수 있지만 하드웨어 설계시 연산 시간이 많이 소요되므로 실시간 처리에 방해 요인이 될 수 있다. 따라서 본 연구에서는 고정소수점 연산에 따른 영상의 찌그러짐을 방지하기 위해 32비트 어큐뮬레이터를 사용하여 유효 자리수를 32비트로 하였다.

4. 메모리

VDM은 M0, M1, M2 등 3개의 독립적인 정적 메모리로 구성되며, 64비트 입출력을 갖는다. M0와 M1은 32 × 64비트 크기이고, M2은 4배인 8Kb 크기이다. M0와 M1은 매크로 블록을 1개씩 저장할 수 있는 크기이고, M2은 4개의 매크로블록을 저장할 수 있는 크기이다. 연산의 입력이나 출력으로 사용할 경우 최장 경로가 될 수 있으므로 파이프라인 레지스터를 사용하여 읽기, 쓰기 동작이 일어나도록 한다. 따라서 메모리를 입출력으로 사용할 경우 2사이클의 수행시간이 걸린다. 메모리를 읽는 사이클인 RP와 쓰는 사이클인 WB와는 2 사이클 차이가 있다. 만약에 한 메모리에 쓰기를 한 후 2 사이클 후에 읽기를 하는 경우 동일한 사이클에 읽기, 쓰기를 다 해야하므로 충돌이 일어나게 된다. 따라서 프로그래머는 이러한 충돌을 피할 수 있도록 주의해야 한다.

외부 데이터 메모리는 32비트 어드레스와 32비트 데이터를 사용한다. 메모리 영역은 LD(LoaD), ST(STore) 명령어에 의해 접근되는 영역과, 4개의 스택 포인터를 어드레스로 하는 스택 영역으로 나눌 수 있다. 데이터 메모리는 사용 영역에 따라 AGU, SPU(Stack Pointer Unit), VPU(Variable Pointer Unit), OPU(Operand Pointer Unit), FPU(Frame Pointer Unit) 등 5개의 어드레스 발생기가 필요하다. SP는 외부 데이터 메모리의 스택 영역과 레지스터 사이의 데이터 교환을 위해 스택의 어드레스를 저장하며, VP는 현재 객체의 지역 변수가 저장된 곳의 어드레스를 가지고 있다. 이 레지스터가 필요한 이유는 VP가 가리키는 외부 데이터를 데이터 스택으로 가져온 후 실행을 하는 동안 데이터 스택은 내부 연산 과정에서

계속적인 동작을 통해 스택을 갱신하기 때문에 현재 지역 변수 위치를 저장해 두지 않으면 외부 데이터의 위치를 알 수 없기 때문이다. 또한 현재 수행되고 있는 프로그램의 변수 값과 객체 정보를 참조하는 역할을 하는 FP가 있으며, 오퍼랜드 스택의 최상위를 저장하고 있는 OP 등이 있다[14].

어드레싱 모드는 A(Absolute)모드, I(Immediate) 모드, IP(Immediate Postincrement)모드, X(Index) 모드, XP(Index Postincrement)모드, PR(PC-Relative)모드 등 6가지이다. BsR은 베이스 레지스터이고, IxR은 인덱스 레지스터, IMM, NUM은 각각 16비트, 6비트 상수이다. A 모드는 CALLcond, JUMPcond 명령어에서 사용되는 모드이다. PR 모드에서 포괄하는 $-2^5 \sim +2^5-1$ 범위를 제외한 $-2^{21} \sim -2^5-1$, $+2^5 \sim 2^{21}-1$에서 어드레스를 갖는다. I 모드는 BsR에 IMM 값을 더한 값을 어드레스로 사용하는 모드이다. LD, ST 명령어만 지원한다. IP 모드는 BsR 값에 IMM 값을 더한 값을 어드레스로 사용하며, 다음 사이클에 연산 결과를 BsR에 저장하는 모드이다. X 모드는 BsR 값에 IxR 값을 더한 값을 어드레스로 사용한다. G, V 명령어 그룹에서 지원한다. XP 모드는 BsR 값에 IxR 값을 더한 값을 어드레스로 사용하며, 다음 사이클에 그 연산 결과를 다시 BsR에 저장하는 모드이다. PR 모드는 CALLcond, JUMPcond 명령어에서 사용되는 모드이다. NUM은 오프셋 값으로 과거 PC 값에 이 오프셋의 반 값을 더한 값이 새로운 PC 값이다. 왜냐하면 NUM값은 명령어 버퍼내의 오프셋으로 실제 어드레스 증가폭의 두 배이기 때문이다. 따라서 이 모드로 발생시킬 수 있는 어드레스 범위는 $-2^4 \sim +2^4-1$ 이다.

5. 스택 버퍼

스택 버퍼는 총 64개의 스택 요소를 내장하고 있으며, DSU(Data Stack Unit), RSU(return Stack Unit) 각 32개이다. 지역 변수 및 파라미터 데이터는 데이터 스택에 저장하고, 서브루틴 프로그램에 효율적으로 사용할 수 있는 리턴 어드레스 값은 리턴 스택에 저장한다.

DSU는 32개의 스택 요소와 DTOS, 스택의 포인터를 저장하고 있는 6비트 스택 포인터인 DSP(Data Stack Pointer), 스택 포인터를 카운팅하는 카운터(Counter), 포인터에 대한 디코딩과 스택 요소에 대한 로드 신호를 발생시키는 DLU(Decode & Load Unit) 등으로 구성된다. DSU 구조는 그림 7.과 같다.

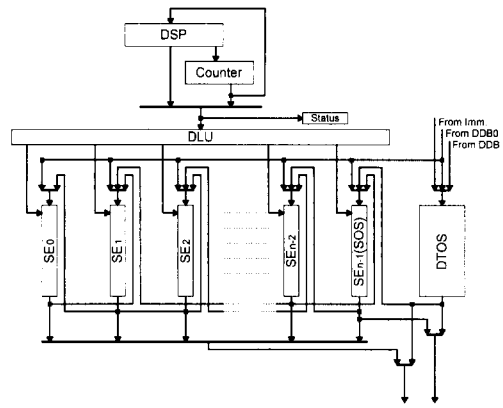


그림 7. DSU 구조
Fig. 7. DSU architecture.

DSU내의 포인터는 6비트 크기를 가지고 있으며, 5비트 LSB는 스택 요소를 선택하는데 사용되며, 1비트 MSB는 스택의 오버플로우, 언더플로우를 찾기 위해 사용된다. DSU는 최상위 스택 요소를 저장해 두는 DTOS를 가지고 있다. 예를 들어 스택을 이용한 덧셈의 경우 DTOS가 없다면 두 개의 오퍼랜드를 패취하고, 그 결과를 저장하기 위해 버퍼 접속이 3 사이클 추가된다. 그러나 DTOS에 오퍼랜드 값을 저장해 두면 연산 결과 값을 저장하는데 필요한 1사이클이 추가될 뿐이다. 모든 스택은 체인 구조이기 때문에 32개 스택을 동시에 이동시킬 수 있다. DDB0 버스로부터 입력되는 모든 데이터는 32개 스택중 임의의 스택에 저장이 가능하고, 스택 메모리로부터 DDB 버스를 통해 입력되는 모든 데이터는 DTOS에 우선 저장되며, 스택 명령어에 따라 푸쉬, 팝 동작을 한다. DSU의 최상위 스택인 SE_{n-1}는 실제는 DTOS 다음의 두 번째 스택인 SOS(Second Of Stack)이다. 이와 같은 구조는 최상위 두 개의 스택 값을 연산할 때 단일 포트만을 이용해서 연산이 가능하도록 해 준다. 또한 DTOS와 SOS을 오퍼랜드로 취함과 동시에 목적지 오퍼랜드를 DTOS로 취하면 어큐뮬레이션 동작이 가능하다.

RSU는 EX 파이프라인 단계에서 동작하는 6비트 EPC(Encoding Pointer & Counter)와 ID 단계에서 동작하는 DPC(Decoding Pointer & Counter)를 갖는다. EP는 EX 단계에서 로드가 되며, DP는 우선-

리턴 동작을 하기 위해 ID 단계에서 동작한다. RTOS(Return Top OF Stack)는 각 스택에 리턴 어드레스를 저장하기 전에 일시 저장하는 역할을 하며, 조건 분기할 때 조건이 맞지 않으면 프로그램 메모리에서 다음 명령어를 패취하기 위한 어드레스로 사용한다. RSU의 출력은 프로그램 메모리의 어드레스로 사용하기 위해 PAB 버스에 실린다.

6. 명령어 버퍼

명령어 버퍼는 프로그램 메모리에서 하나의 가변 길이 명령어를 입력받아 명령어 길이 정보와 함께 디코더 블록으로 보내는 역할을 한다. 명령어 버퍼는 프로그램 메모리로부터 입력된 가변 길이 명령어의 명령어 길이를 디코딩하는 MAU(Mark Appending Unit), 가변 길이 명령어를 저장하는 IBU(Instruction Buffer Unit), MAU로부터 출력된 명령어 길이 정보를 저장하는 MB(Mark Buffer), MB 정보가 의미가 있는 지 여부를 판단해 주는 FB(Flag Buffer)으로 구성된다. 또한 버퍼의 포인터를 발생하는 PGU (Pointer Generation Unit), 디코딩 유닛으로 바로 전달되는 바이패싱(Bypassing) 등이 있다. 명령어 버퍼 구조는 그림 8.와 같다.

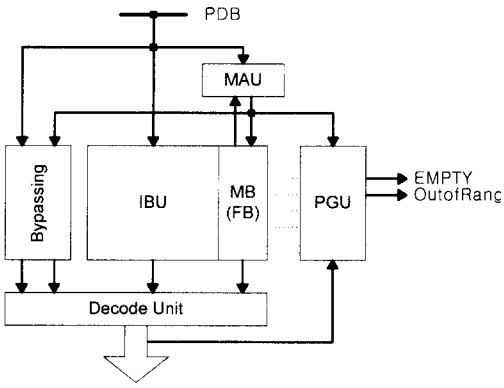


그림 8. 명령어 버퍼 구조.
Fig. 8. Instruction buffer architecture.

IBU내에 다음 패취될 명령어가 존재하면 상응하는 가변 길이 명령어와 그 길이 정보는 각각 IBU, MB로부터 CPU에 전달된다. 반면에 다음 패취될 명령어가 버퍼 안에 존재하지 않으면, 상응하는 가변 길이 명령어는 외부 프로그램 메모리로부터 IBU로 다시 가져오며, 명령어 길이는 MAU에서 다시 디코딩된 후 바이패싱 블록을 거쳐 명령어와 길이 정보를 CPU에 전달한다. MAU는 명령어가 CPU에서 디코드 되기 전에 먼저 가변 길이 명령어를 디코드한 후 길이 정보를 출력하는 우선-디코드(predecode) 특성을 갖는다. MB(Mark Buffer)는 MAU에서 만든 명령어 길이 정보를 저장하는 버퍼로 64×2b이며, 64×1b 크기의 FB(Flag Buffer)를 포함한다. 현재 명령어 길이 정보를 구할 때 전 상태의 MB 유효 여부가 필요하므로 MB값을 구한 후 유효 여부를 판단하여 FB에 저장한다. PGU는 IBU, MB에 명령어와 길이 정보를 읽거나 쓸 때 정확한 포인터를 발생시킨다. 또한 버퍼 내에 명령어가 있는지 없는지를 판단하며, 분기 명령어 패취에 따른 분기 동작이 일어날 때 타켓 명령어가 버퍼 범위 안에 있는지 여부를 판단한다.

Ⅳ. 결과 및 고찰

결정된 설계 사양에 따라서 VHDL로 행위적 수준(behavioral level) 및 구조적 수준(structural level) 모델을 작성하였다. 각 블록별로 행위적 수준 모델을 작성하고 작성된 VHDL 모델은 Model Technology Inc.의 V-System PC 버전 4.4를 이용하여 시뮬레이션하였다. 사용한 라이브러리는 V-System에서 기본으로 제공하는 IEEE 라이브러리만을 사용하였다. 행위적 수준의 모델을 작성할 때부터 파이프라인과 하드웨어를 고려하여 작성함에 따라 구조적 수준의 모델링을 용이하게 하였다.

구조적 수준의 모델링을 수행한 후 COMPASS AsicSyn 툴을 이용하여 합성하였다. 사용한 라이브러리는 0.6㎛ 5-Volt CMOS TLM COMPASS 라이브러리이다. 총 게이트 수는 약 160,000개이며, 게이트 수준의 시뮬레이션을 통해 얻은 최장 경로는 S-RISC 블록 내의 곱셈기로서 약 18.2ns의 지연 시간을 갖는다. 따라서 전체 동작 주파수는 약 50MHz이다.

그림 9.는 본 논문의 SVP를 피코자바 및 다른 구조의 RISC와 구조적인 특징을 비교하는 그림이다[15-20].

그림 9.는 SVP를 구조적인 면에서 피코자바 및 RS/6000, MCF5307와 비교한 것이다. SVP의 구조적인 특징은 원 모양으로 연결되어 있다. 그림 9의 모든 프로세서는 RISC와 같은 구조를 가지고 있으며, 특히 MCF5307은 가변 길이 명령어 구조를 가지고 있으며, M68020[21]을 기본 모델로 한 프로세서이다.

피코자바와 같은 스택을 기반으로 한 프로세서는 레지스터 수가 적고, 스택 레지스터를 많이 가지고 있다. 반면, RISC는 레지스터를 많이 가지고 있으며, 스택 레지스터를 가지고 있지 않다.

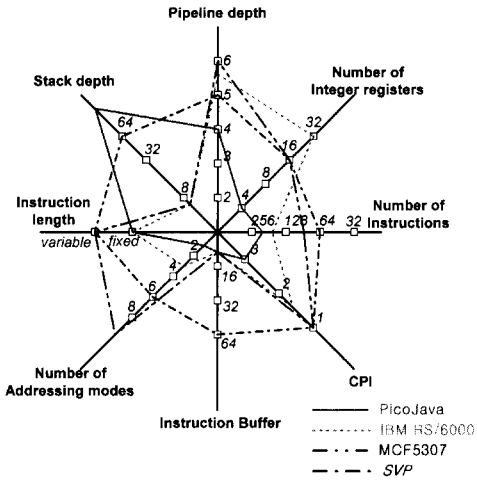


그림 9. 여러 프로세서와 SVP의 구조적 파라미터 비교

Fig. 9. Comparison of architectural parameters between several processors and SVP.

SVP는 두 종류의 프로세서의 중간 정도의 구조적인 특징을 가지고 있음을 확인할 수 있다. 또한 캐쉬를 가지고 있는 프로세서는 명령어 버퍼를 적게 내장하지만 SVP는 따로 캐쉬를 내장하지 않으며, 따라서 상대적으로 큰 명령어 버퍼를 허용한다. 또한 피코자바에 비해 더 작은 명령어 수를 가지고 있지만 스택을 이용한 대부분의 연산을 수행하며, 한 명령어당 한 사이클 수행이 가능하다. 스택 명령어중에서 스택 간의 데이터 상호 교환 등을 하는 명령어 또한 피코자바의 3사이클보다 2 사이클이 적은 1 사이클로 동작이 가능하다. SVP는 영상 데이터를 처리하기 위해 벡터 프로세서를 내장하고 있고, 처리된 연산 정보를 임시로 저장하기 위한 레지스터 뱅크를 갖는다. 따라서 범용 RISC와 같은 동작이 가능하여 단일 사이클 처리가 가능하다. 벡터 연산이 이루어질 경우 피코자바 및 RISC는 레지스터에 저장해야 할 값을 스택에서 가져오거나, 반복적인 레지스터 접속을 하기 때문에 동작 사이클 수가 늘어나야 하지만 SVP는 단일 사이클 처리가 여전히 유효하다. 표 2.는 본 프로세서의 MPEG-4 코어 알고리즘에 따른 처리 성능을 보여 주고 있다.

표 2. MPEG-4 코어 알고리즘에 대한 SVP 성능

Table 2. SVP performance for MPEG-4 core algorithm(15fps, QCIF)

Algorithm			Performance (ms)
IDCT			95.0
SA-IDCT			95.0
VOP reconstruction			0.002
Motion prediction	P-VOP	General mode	12.0
		Advanced mode	67.0
	B-VOP	General mode	26.0
		Advanced mode	29.0
CAE context arithmetic			14.0
Sampling	up-sampling	CR=1/2	85.0
		CR=1/4	107.0

SVP에서 IDCT와 SA-IDCT 처리 속도는 같다. 그러나 IDCT 연산은 모양 정보를 이용한 패딩 과정이 필요하기 때문에 처리 속도가 다소 많이 걸린다. QCIF 포맷의 영상 프레임을 IDCT할 경우 23.6fps 전송 속도를 가지며[22], VLBV의 15fps 조건을 만족한다. 표 3.에서는 움직임 예측이 IDCT에 비해 다소 많은 시간을 차지하는 것으로 나타나고 있는데 이는 고급 모드(advanced mode)에서 반화소 단위 예측 이외에 OBMC를 하기 때문이다. B-VOP에서는 고급 모드에서 8×8 예측만 하고, OBMC를 하지 않기 때문에 일반 모드와 별 차이가 나지 않는다. CAE는 BAB가 저장되어 있는 메모리의 픽셀 데이터를 다소 불규칙하게 읽는 과정이 포함되므로 많은 시간을 필요로 한다. 모든 BAB에 대해서 행해지는 샘플링 과정은 부호화 과정에서 사용되는 다운샘플링보다 재생 과정에서 사용되는 업샘플링 시간이 더 많이 걸린다. 이는 다운샘플링이 단순히 평균 값을 구하는 것에 비해 업샘플링은 12개 픽셀 값에 대한 연산과 기준 값과의 비교 과정 등이 포함되기 때문이다.

표 3.의 결과를 사용하여 제안한 구조의 성능과 기존의 예를 영상 해상도에 따라 프레임 전송 속도의 관점에서 비교할 수 있다[22]. 표 3.에서 사용한 영상은 'GARDEN' 시퀀스로 프레임 크기는 QCIF 포맷인 176 x 144 이며, 총 프레임 수는 50개이다. Pentium-200과 Sun Ultra Sparc은 번역기(inter-

preter)를 내장한 웹 브라우저 환경을 갖는 PC 및 워크스테이션에서 네트워크를 통해 전달된 영상을 재생한다[23].

표 3. SVP, Pentium-200, Sun Ultra Sparc의 네트워크 기반 MPEG-4 재생 성능
Table 3. Performance of MPEG-4 decoding over network for SVP, Pentium-200, and Sun Ultra Sparc.

video format (pixels)	SVP	Pentium-200	Sun Ultra Sparc
176×144	15.0fps	3.0fps	3.0fps

표 3.의 Pentium-200과 Sun Ultra Sparc은 현재 사용되고 있는 웹 브라우저 환경이 내장된 PC나 워크스테이션의 CPU로 자바를 구현한 브라우저 국제 표준인 2fps[24]를 만족하지만 MPEG-4 VLBV의 실시간 전송율인 15fps에는 미치지 못한다. 피코자바와 같은 자바 전용 프로세서를 사용한 방법 또한 자바 지향적인 특정 알고리즘에 대해 기존의 범용 CPU 보다 높은 성능을 보이나, 영상 처리와 같은 응용에서는 레지스터 없이 스택에 기반한 구조 특성으로 인해 실시간 처리를 기대할 수 없다. 표 3.에 나타낸 본 논문의 영상 처리 성능은 MPEG-4 VLBV의 최대 프레임 전송율인 15fps를 만족하며, 구현된 알고리즘은 영상 재생 알고리즘의 총 연산량의 50%를 차지하는 IDCT, 움직임 보상을 기준으로 추정한 것이다. 선택 사항으로 포함될 수 있는 모양 정보 부호화 알고리즘은 제외되며, 이후 하드와이어드 회로나 전용 프로세서로 구현될 것이다.

'GARDEN' 시퀀스는 누락(skip)시킬 수 있는 매크로블록은 각 프레임마다 평균 6.4% 정도이다. 따라서 표 2.의 IDCT 성능은 실제 영상에서는 다소 상승할 수 있으며, 표 4.는 SVP를 이용하여 8×8 IDCT 연산을 YCbCr 색깔 성분을 갖는 여러 연속 영상 프레임에 대해 수행했을 때 프레임 전송율을 비교한 것이다. IDCT가 영상 재생 과정에서 차지하는 비율은 35%로 계산하였다.

MPEG-4의 최대 해상도는 QCIF 기준 VLBV는 15fps이하이며, 따라서 IDCT를 기준으로 VLBV를 만족한다고 할 수 있다. 표 4.의 값은 IDCT 변환 계수 행렬중 DCT_8을 기준으로 한 것이다.

표 4. 여러 영상 포맷에 대한 IDCT의 최대 프레임 전송 속도
Table 4. Maximum frame per second of IDCT for various video formats.

video format(pixels)	Maximum runtime(fps)
128×96	47.5
176×144	23.6
352×288	5.8

즉, 8×8 세그멘트내의 64개 픽셀이 모두 '0'이 아닌 값을 가지고 있음을 의미한다. 만약 각 행과 열에 픽셀들의 DCT 벡터가 모두 '1'일 때, 즉 DCT1을 변환 계수로 사용하였을 때 전송 프레임 수는 2배로 증가한다. 이는 DCT1는 8×8 변환 계수 값중 $\frac{1}{2}$이 '0' 값을 가지고 있기 때문에 전체 계산 시간이 50% 감소하며, 따라서 4개의 MAC을 가지고 있는 본 논문의 SVP는 1차원만으로 연산을 수행할 수 있기 때문이다. IDCT는 SVP 내의 모든 VMAC 연산부와 영상 데이터 메모리 및 어드레스 연산을 동시에 수행하기 때문에 영상 재생 알고리즘중 가장 높은 MOPS(Million Operations Per Second)를 가진다. 따라서 IDCT를 기준으로 계산된 fps 값은 각 프로세서의 연산력(Compute power)를 보여준다. 그림 10.은 각 프레임마다 약 6.4%의 매크로블록을 누락시킬 수 있는 영상 시퀀스 'GARDEN'을 객체 지향 프로그램된 IDCT 알고리즘에 대해 본 논문의 SVP를 피코자바, Pentium II와 각각 비교하였다.

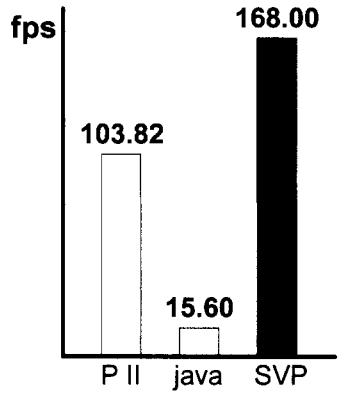


그림 10. IDCT 성능
Fig. 10. IDCT performance.

Pentium II에서 실행되는 알고리즘은 Berkeley의 1-D IDCT 알고리즘을 수행한 결과이며, 피코자바에서 수행되는 알고리즘은 8×8 블록에 대해 벡터 연산 없이 화소 단위로 연산을 한다. Pentium II의 동작 속도는 233MHz이며, 피코자바는 50MHz로 가정한다. 본 논문의 SVP는 50MHz이다.

피코자바에서 IDCT 연산은 8×8 블록에 대해 각 화소마다 8번의 곱셈과 7번의 덧셈을 한다. 따라서 한 개의 프레임을 처리하는데 소요되는 총 사이클은 QCIF 포맷의 99개의 매크로블록이 실제 'GARDEN' 시퀀스에서 6.4%의 매크로블록을 누락시킬 수 있으므로 이를 고려하여 총 3,202,468 사이클이다. 피코자바는 명령어당 평균 소요되는 사이클은 3사이클이며, 동작 속도를 50MHz로 가정하였으므로 사이클마다 지연시간은 20ns이기 때문에 피코자바 fps는 (1)과 같이 계산할 수 있다.

$$\text{피코자바 } fps = \frac{1}{3,202,467 cycles \times 20 ns/cycle} \quad (1)$$

SVP의 한 개의 프레임을 처리하는데 소요되는 총 사이클은 (2)와 같다.

$$(128 \text{ VMACs} \times 2 \text{ pipeline/vmac} + 12 \text{ overhead cycles}) \times 2D \times 6 \text{blocks/MB} \times (99 \text{MB} \times 0.936) \fallingdotseq 298,007 \text{ cycles} \quad (2)$$

SVP fps는 (3)과 같이 계산할 수 있다.

$$\text{SVP } fps = \frac{1}{298,007 cycles \times 20 ns/cycle} \quad (3)$$

본 논문의 SVP를 이용할 경우 객체 지향 코드로 구현된 IDCT 알고리즘을 고려하였을 때 기존의 Pentium II CPU와 피코자바를 이용한 구현보다 각각 1.6배, 10배 이상의 처리 속도를 향상시킬 수 있음을 확인하였다. 또한 선택사항인 모양 정보 부호화를 제외한 MPEG 재생 알고리즘도 15fps를 만족함을 확인하였다.

V. 결 론

본 논문에서는 네트워크 환경에 적응이 용이한 객체 지향형 영상 프로그램을 효율적으로 처리하는 프로세서 구조를 제안하고, 이를 이용하는 영상 신호 처리의 효율성을 분석하였다. 또한 제안한 구조를 몇 가지 알고리즘에 적용한 결과를 통해서 제안한 구조가 영상 신호를 효율적으로 처리할 수 있다는 것을 검증하였다.

SVP는 과거에 제안되었던 스택 머신과 영상 프로세서의 최적의 측면 만을 선택함으로써 더 좋은 구조를 갖도록 하는 포괄적인 구조이다. 따라서 스택에 기반한 프로세서이면서도 영상 응용에 대해 고성능의 레지스터 기반 프로세서 성능에 근접할 수 있는 구조를 갖는다. 결국 스택 머신의 고유한 특징을 희생시키지 않고, 영상 프로세서의 성능과 조화되도록 한 것이다.

SVP는 0.6㎛ 0.5V TLM CMOS 기술로 합성하였으며, 50MHz까지 동작이 가능하다. 또한 160,000 게이트 수를 갖는다. MPEG-4의 VLBV 최대 전송율인 QCIF 15fps로 알고리즘을 수행한다.

본 논문은 기존의 연구 결과를 다양하게 반영하여 많은 부분에서 개선하였으나, 결과에 덧붙여 보완 연구가 필요하며, 이러한 본 논문의 결과에 덧붙여 향후 뒤따라야 할 연구 분야는 다음과 같다.

제안된 구조는 MPEG-4 표준의 선택사항인 모양 정보 부호화를 수용할 경우 VLBV 요건을 만족할 수 없기 때문에 이를 수용할 수 있는 연구가 필요하며, 또한 제안된 구조가 객체 지향형 언어로 프로그램된 MPEG-4 구현을 위한 전체 영역에서 어느 정도의 범위를 차지하는 가를 알 수 있는 척도를 제시하는 연구가 보완되어야 한다.

그리고 제안한 SVP 구조는 코어 연산 블록에 대해서만 설계가 되었기 때문에 프로세서 외부와의 인터페이스 및 내,외부 메모리를 효율적으로 제어할 수 있는 DMA 등이 포함되어 있지 않다. 따라서 이와 관련된 명령어 연구가 덧붙여져야 한다. 그리고, 내부의 메모리를 논리 합성을 통해 설계하였기 때문에 최적의 메모리 크기라고 하기 어렵고, 더 높은 성능을 위해서 이에 관련된 보완 연구가 필요하다.

참 고 문 헌

[1] 전병우, 이광기, "MPEG-4 응용", 한국통신학회지 제 14권 제 9호, pp.120-127, 1996년 9월

[2] P. J. Koopman, Jr., "Stack Computers: the new wave", http://www.cs.cmu.edu/~koopman/stack_computers/, 1989.

[3] S. H. Huseby, "Video on the World Wide Web", http://www.ifi.uio.no/~ftp/ publications/cand-scient-theses/SHuseby/,

Feb. 1997.

[4] J. B. Lee, et. al., "Java Downloadable MPEG-4 Decoder", http://www.ctr.columbia.edu/~jbl/FLEXVIEW.htm, 1997.

[5] P. J. Koopman, Jr., "Stack Computers: the new wave", http://www.cs.cmu.edu/~koopman/stack_computers/, 1989.

[6] L. W. Ho, et.al., "An Efficient Controller Scheme for MPEG-2 Video Decoder," IEEE Trans. on Consumer Electronics, Vol. 44, No. 2, pp. 451-458, May 1998.

[7] "Feig's scaled 2-D DCT now tested with the Berkeley MPEG-Player," http://rnvs.informatik.tu-chemnitz.de/~ja/MPEG/-HTML/idct_discussion/Index.html, 1998.

[8] A. J. Jeroen, et. al., "Prophid: A Heterogeneous Multi-Processor Architecture for Multimedia," Proc. of ICCD, pp. 164-169, 1997.

[9] 박주현, 김영민, "네트워크 기반 객체 지향형 영상 처리를 위한 MPEG 디코더 코어 설계", 한국통신학회논문지 제 23권 제 8호, pp. 2120-2128, 1998

[10] 박주현, 김영민, "스택 기반 객체 지향형 영상 제어기 설계", 전자공학회 논문지 제 35권 C편 제 7호, pp. 69-77, 1998년

[11] M. R. Cosgrove, et. al., "Instruction address stack in the data memory of an instruction-pipelined processor", United States Patent, Appl.no.: 280,417, 1983.

[12] M. Hjersing and A. Ive, "JAVAX : An implementation of the Java Virtual Machine", Master Thesis, Dept. of Computer Science Lund Institute of Tech. Sweden, pp. 10-19, 1996.

[13] 정보통신부, 고성능 DSP Core 개발에 관한 연구, 선도기술개발사업 3차년도 보고서, pp. 124-128, 1998년 6월

[14] T. Lindholm and F. Yellin, The JavaTM Virtual Machine Specification, Addison Wesley, 1996

[15] Sun Microsystems, picojava I Microprocessor Core Architecture. Mountain View: Sun Microsystems, 1996.

[16] Motorola, MCF5307(ColdFire) User's Manual, 1995.

[17] J. Circello, "ColdFire: A Hot Architecture", http://www.byte.com/art/9505/sec13/art1.htm#coldfire, May 1995.

[18] G.F. Grohoski, "Machine organisation of the IBM RISC System/6000 processor", IBM Journal of Research and Development, Vol. 34, No. 1, pp. 37-58, Jan. 1990.

[19] Brian Hall, Kevin O'Brien, "Performance Characteristics of Architectural Features of the IBM RISC System/6000", Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, Santa Clara, California, pp. 303-309 Apr. 1991.

[20] R.R. Oehler and R.D. Groves, "IBM RISC System/6000 processor architecture", IBM Journal of Research and Development, Vol. 34, No. 1, pp. 23-3. Jan. 1990.

[21] Robin W. Edenfield et al, "The 68040 Processor: Part 1, Design and Implementation", IEEE Micro, pp. 66-78, Feb. 1990.

[22] 박주현, 김영민, "데이터패스를 이용한 SA-DCT 구현", 전자공학회논문지 제 35권 C편 제 5호, pp. 25-32, 1998년

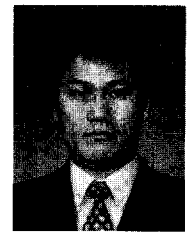
[23] D. Y. Liu, C. W. Tsai and J. L. Wu, "A Java-based MPEG-4 like Video Codec,", IEEE Trans. on Consumer Electronics, Vol. 44, No. 1, pp. 200-204, Feb. 1998.

[24] Java H263 VOD Client Framework, http://www.npac.syr.edu/users/marek/info.html.

저 자 소 개

朴 柱 炫(正會員)

1969년 7월 13일생. 1993년 2월 전남대학교 전자공학과 졸업. 1995년 2월 전남대학교 대학원 전자공학과 졸업(공학석사). 1999년 2월 전남대학교 대학원 전자공학과 졸업(공학박사). 1999년 2월 ~ 현재 한국전자통신연구원(ETRI) 집적회로설계연구부 선임연구원. 주관심분야는 MPEG4 코덱 설계, 객체 지향 프로세서, DSP, RISC 프로세서 설계 등임



金 榮 民(正會員)

1954년 4월 18일생. 1976년 2월 서울대학교 전자공학과 졸업(공학사). 1978년 2월 한국과학원 전기 및 전자공학과 졸업(공학석사). 1978년 3월 1979년 7월 한국선박해양 연구소(주임연구원). 1986년 오하이오 주립대학교 전기공학과(공학박사). 1988년 6월 ~ 1991년 8월 한국전자통신연구소(실장). 1991년 9월 ~ 현재 전남대학교 전자공학과 교수. 1998년 2월 ~ 현재 전남대학교 공과대학 전자통신기술연구소(ETTRC) 소장. 1997년 12월 ~ 현재 반도체설계교육센터(IDEC) 전남대학교 지역센터장. 주관심분야은 ADSL 모뎀 설계, MPEG2, MPEG4 시스템 설계 등임