論文 95-5-5-21

# 오디오 신호 처리용 디지탈 시그날 프로세서의 VLSI 설계

正會員 李 鍾 和*, 張 昌 熙**, 鄭 海 龍***, 金 軾*, 黃 善 泳*

# VLSI Design of a Digital Signal Processor for Audio Applications

Jong Hwa Lee*, Chang Hee Jang**, Hae Ryong Jung***,
Sik Kim*, Sun Young Hwang* Regular Members

## 要 約

본 논문은 오디오 신호 처리를 위한 디지탈 시그날 프로세서인 HiDSP의 VLSI 설계에 관하여 기술한다. HiDSP는 오디오 신호 처리를 위한 알고리듬을 효율적으로 수행하기 위하여 52개의 인스트럭션 세트를 지원한다. 4단 인스트럭션 파이프라인으로 구현된 HiDSP은 24 비트 고정 소수점 연산 및 56 비트 승가산 연산을 수행하며 동시에 한 파이프라인 스테이지 내에 두개의 데이타 이동을 완료한다. 하드웨어는 VHDL을 이용한 top-down 방식으로 설계되었으며, 0.8 $\mu$m CMOS 게이트 어레이 테크놀로지를 이용하여 제작되었다. 실험 결과 몇가지의 벤치마크에서 HiDSP가 Motorola 사의 DSP56000보다 빠른 수행을 보였으며 30 MHz의 클록 속도에서 동작함을 확인하였다.

## ABSTRACT

This paper describes the VLSI design of a digital signal processor, HiDSP, tailored to audio applications. HiDSP has 52 instructions for an audio signal processing algorithms. HiDSP, designed to operate in 4-stage pipeline, supports 24-bit fixed point arithmetic and 56-bit multiply-and-accumulation operations in parallel with two data movements within one pipeline stage. The hardware has been designed in top-down approach using VHDL, and implemented by 0.8 $\mu$m CMOS gate array technology. Experimental results show that HiDSP executes several benchmark programs faster than Motorola's DSP56000 chips, and operates at 30MHz clock speed.

* 서강대학교 전자공학과
** 삼정전관(주)
*** 삼성전자(주)
論文番號 : 95016-0114, 95017-0114
接受日字 : 1995年 1月 14日

## Ⅰ. 서 론

최근 VLSI 기술의 발달과 함께 단일 칩에 집적될 수 있는 트랜지스터의 갯수가 증가되고 사용자에게 친숙한 인터페이스의 지원에 대한 요구가 높아짐에 따라 디지탈 신호를 실시간에 처리하는 하드웨어에 대한 연구가 활발히 진행되고 있다 [1][2]. 기존의 컴퓨팅 환경에서는 문자를 중심으로 처리함으로써 사용자에게 친숙한 영상과 음성등 디지탈 신호를 처리하는데 한계를 나타내었다. CISC 혹은 RISC 형태의 범용 마이크로 프로세서는 디지탈 신호 처리를 실시간에 수행하기에는 부적합하며, 신호 처리의 특정 알고리듬만을 지원하는 전용 하드웨어는 수행 속도는 빠르나 응용 분야가 적은 단점이 있다. 이에 따라 다양한 디지탈 신호 처리 알고리듬을 구현할 수 있는 프로그램 가능한 범용 DSP (Digital Signal Processor)가 등장하였다. 현재 상용화된 제품으로는 Texas Instruments 사의 TMS320 시리즈, Motorola 사의 DSP56000 시리즈 등이 있으며, 오디오용 디지탈 신호 처리 등에 많이 사용되고 있는 DSP56000은 데이타의 병렬 이동과 승가산 연산을 전용 하드웨어로 처리함으로써 1024-point FFT를 3.23 ms에 수행할 수 있다 [3].

현재 상용화 되고있는 DSP는 복잡한 인스트럭션 세트를 제공함으로써 디코딩 시간에 많은 시간이 소모된다. DSP는 일반 마이크로 프로세서와는 달리 처리해야 할 응용분야가 한정되어 있으며, 이는 많은 인스트럭션을 요구하지 않는 RISC 형태의 프로세서로 효과적으로 구현 가능하다 [4]. 현재 오디오 신호 처리등에 많이 사용되는 Motorola의 DSP56000의 경우 인스트럭션 필드의 크기를 줄여주기 위해 복잡한 인스트럭션 형태를 채택함으로써 디코딩에 많은 시간이 소모되어 인스트럭션 파이프라인 구현이 3단으로 이루어져 있다 [3]. 오디오 신호 처리를 목적으로 개발된 HiDSP (High-performance DSP)는 RISC 형태의 인스트럭션 세트를 채택함으로써 인스트럭션 디코딩에 소모되는 시간을 줄여주어 4단 파이프라인으로 구성되었다. 또한 HiDSP는 파이프라인의 효율적인 사용을 위하여 일반 산술논리 연산부와 MAC 연산부를 분리하여 MAC 연산을 두개의 파이프라인 스테이지에 할당함으로써 파이프라인의 효율성을 증대시켰다. 한편 어큐뮬레이터 기본의 아키텍춰가 다량의 데이타 이동을 필요로하는 단점의 보완을 위하여 레지스터 파일 기본의 아키텍춰를 채택함으로써 기존의 상용 DSP에 비해 여러 알고리듬의 구현에 있어서 데이타 이동 회수를 줄였으며 코딩을 편리하게 하였다. DSP56000의 경우 인스트럭션 형태에 따라 병렬 이동을 두개의 인스트럭션으로 처리함으로써 결과적으로 하나의 파이프라인 스테이지만큼 처리가 늦어지는데 반해, HiDSP는 이를 하나의 인스트럭션으로 처리하여 신호처리 능력을 증가시켰다. HiDSP에서는 모든 데이타의 병렬 이동을 하나의 파이프라인 스테이지 안에서 처리하고 별도의 추가 파이프라인 스테이지 없이 수행이 가능하도록 함으로써 이로인한 지연을 제거하였으며 데이타 이동시 발생 가능한 데이타 의존관계에 의한 파이프라인 지연 문제를 해결하였다.

디지탈 신호 처리에서 빈번히 발생되는 루프문을 하드웨어에 의해 처리해 줌으로써 발생될 수 있는 파이프라인 hazard를 해결하였다. 루프문을 소프트웨어에 의해 처리할 경우 프로그램의 크기가 증가될 가능성이 있으며 프로그램 수행순서 변경에 의한 파이프라인 hazard가 발생될 가능성이 있다. 이러한 문제점은 하드웨어에 의한 처리방식으로 해결할 수 있으며 이는 DSP의 처리능력을 증가시킨다. 또한 DSP는 외부와의 데이타 통신이 빈번히 일어나며 병렬 구조로 사용되어 보다 많은 양의 데이타 처리 능력을 부여하고 host가 되는 프로세서와 함께 사용되어 디지탈 신호만을 전문적으로 다룰 필요가 있다. 외부와 다양한 인터페이스를 제공을 위하여, HiDSP는 사용자가 프로그래밍할 수 있는 인터럽트 벡터 테이블을 제공하고 부팅시에 이를 칩 내부로 불러들여 인터럽트를 처리함으로써 다양한 외부 인터페이스를 제공한다.

본 논문은 오디오 신호 처리에 적합한 디지탈 시그날 프로세서인 HiDSP의 VLSI 설계에 관하여 기술한다. 2장에서는 HiDSP의 아키텍춰에 대해 설명하고 3장에서는 데이타패스의 특징에 대하여 설명한다. 4장에서는 HiDSP에서 구현된 콘트롤러에 대하여 설명한다. 5장에서는 실험 결과로 시뮬레이션 결과를 통한 DSP56000과의 비교를 benchmark 프로그램을 이용하여 기술하며 제작된 칩의 제원에 대하여 기술한다. 6장에서 결론 및 추후과제를 제시한다.

## Ⅱ. HiDSP 프로세서

### 2.1 HiDSP 아키텍춰

HiDSP의 인스트럭션 처리 능력은 15 MIPS로 4상 클럭을 사용하며 파이프라인 한 stage의 지연시간은 두 클럭주기에 해당된다. 내부 데이타패스는 24 비트이고 MAC 연산의 결과는 56 비트로 표현되어 각각 144 dB와 336 dB의 다이나믹 레인지로 데이타를 표현할 수 있어 오디오에 응용되는 경우 CD와 같이 고음질의 디지탈 신호 처리에 적합한 사양을 만족하고 있다. HiDSP는 오퍼랜드가 되는 두개의 데이타 이동을 위한 데이타패스와 인스트럭션 이동을 위한 데이타패스를 분리시킨 dual Harvard 아키텍춰를 채택하였으며, 칩 내부에 두개의 오퍼랜드와 프로그램을 저장할 수 있는 3개의 24 x 512 크기의 내부 메모리를 탑재하여 응용영역에 맞는 디지탈 신호 처리를 위한 알고리듬을 구현한다. 또한 load/store 아키텍춰를 채택하여 외부 메모리와의 데이타 이동 회수를 최소화 하였으며 전용 버스를 사용하여 각 연산부와 메모리, 주변장치와의 데이타 전송이 동시에 수행됨으로써 인스트럭션 파이프라인에서 발생될 수 있는 자원 공유에 의한 구조적 hazard 문제를 해결하였다. 내부 메모리는 포트와 함께 사용자가 정의하는 프로그램 및 오퍼랜드를 저장할 수 있도록 설계되었다.

인스트럭션의 종류에는 33개의 산술, 논리 연산 인스트럭션과 11개의 제어 인스트럭션이 있으며 6개의 데이타 이동 인스트럭션 및 외부 메모리와의 데이타 전송을 위한 2개의 인스트럭션이 제공되어 전체적으로 52 개의 인스트럭션이 지원된다. 그림 1에 HiDSP의 어셈블리 언어를 이용하여 표현한 산술 연산과 병렬 데이타 이동을 위한 인스트럭션의 구현 예를 보였다.

HiDSP의 데이타패스는 2의 보수 방식의 24 비트 고정 소수점 산술 논리 연산을 지원하며 56 비트 승가산 연산을 지원한다. 두 24 비트 데이타에 대한 곱셈 결과는 48 비트로 나타나며 MAC 연산 시 발생하는 overflow를 방지하고 다이나믹 레인지를 높여주기 위한 8 비트의 SE (Sign Extension)를 첨가하여 56 비트로 표현된다.

### 2.2 시스템 개관

HiDSP의 전체 시스템 블록도를 그림 2에 나타내었다. HiDSP는 전체 칩의 동작을 제어하는 콘트롤러와 데이타의 병렬 이동을 위한 버스 및 프로그램과 두 오퍼랜드를 담고있는 메모리 블록과 데이타의 연산 기능을 담당하는 데이타 처리 유닛, 프로그램 상에서 지정된 메모리의 오퍼랜드의 주소를 계산하는 어드레스 생성 유닛 그리고 외부와의 입출력을 담당하는 포트로 이루어진다. HiDSP는 병렬 어드레스 생성과 데이타 이동을 지원하기 위한 3개의 어드레스 버스와 4개의 데이타 버스를 가진다. 어드레스 버스는 16 비트의 PAB (Program Address Bus), XAB (X Address Bus), YAB (Y Address Bus)로 구성되며, 데이타 버스는 24 비트의 PDB (Program Data Bus), XDB (X Data Bus), YDB (Y Data Bus), GDB (Global Data Bus)로 구성된다. PAB는 부트스트랩 롬과 인터럽트 벡터 테이블, 칩 내부 혹은 외부 프로그램 메모리의 어드레스 지정에 사용되며 XAB, YAB는 X, Y 메모리 혹은 외부 데이타 메모리의 어드레스 지정에 사용된다. 내부 메모리는 9 비트의 어드레스를 가지며 16 비트 어드레스 버스를 이용하여 내부 및 외부 메모리 지정이 가능하다.

PDB는 외부 프로그램 메모리로부터의 프로그램 데이타 load, 인스트럭션 패치 시의 프로그램 이동을, XDB와 YDB는 X, Y 메모리와 각 블럭 간의 오퍼랜드 데이타 이동을 담당하며, GDB는 메모리 이외의 각 블럭 간의 데이타 이동에 사용된다. 콘트롤러는 인스트럭션 패치를 위한 프로그램 어드레스의 생성과 인스트럭션 디코딩, 인터럽트 처리를 담당한다. 사용자에 의해 정의되는 X, Y, P의 주소는 AGU (Address Generation Unit)에서 생성되며, 데이타 버스를 통하

```
MACR  X1, Y1, X2   &   X:(B0+I0), X0   Y:(B1+I1), Y0
```

(opcode operands seperator 1st parallel move 2nd parallel move)

그림 1. 산술 연산과 병렬 데이타 이동의 예
Fig. 1. An example of arithmetic operation and parallel data movement

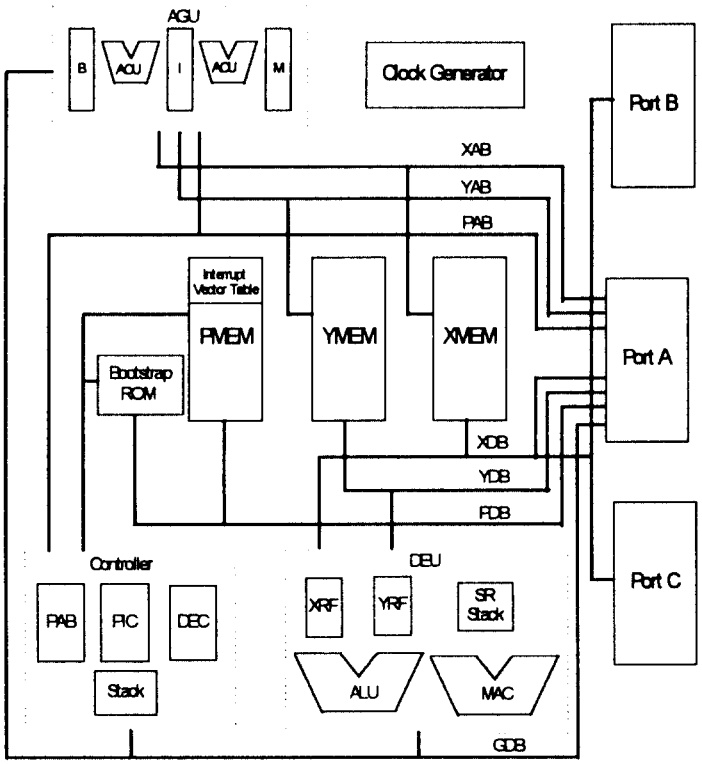



그림 2. HiDSP의 전체 블록도
Fig. 2. Block diagram of top-block of HiDSP

여 전송된다. AGU는 병렬 데이타 이동을 위해 동시에 두 어드레스를 생성할 수 있으며, 연산 중에도 XDB, YDB 버스를 통한 두 오퍼랜드의 메모리와 레지스터 간의 병렬 데이타 이동을 지원한다. DEU (Data Execution Unit)는 XDB, YDB, GDB를 통하여 데이타를 받아들이며 24 비트 산술 논리 연산과 MAC 연산을 지원한다. HiDSP는 3 개의 포트를 통하여 내부와 외부 메모리 혹은 주변기기와의 데이타 이동, 호스트 프로세서와의 인터페이스, 다양한 전송 속도에 의한 직렬 데이타 송수신을 지원한다.

### 2.3 파이프라인 구조

HiDSP는 RISC 형태의 아키텍춰를 채택하였으며 인스트럭션은 그림 3에 보인 바와 같이 4 단 파이프라인 구조에 의하여 수행된다.

IF (Instruction Fetch) 단계에서는 PAB (Program Address Block)에서 생성된 주소를 이용하여 프로그램 메모리의 인스트럭션을 PDB를 통해서 읽어들여 IF 레지스터에 래춰시키는 동작을 수행한다. ID (Instruction Decode) 단계에서는 IF 레지스터의 인스트럭션을 디코딩하여 인스트럭션 수행에 필요한 콘트롤 신호를 생성하며, 메모리를 참조하는 인스트럭션인 경우 AGU를 통해 메모리의 주소를 생성한다. 데이타 연산을 위한 파이프라인 구조는 24 비트 일반 산술 논리 연산과 56 비트 결과를 가지는 연산을 분리하여 각각 2 단의 파이프라인 스테이지에 할당하였다. EXE1,

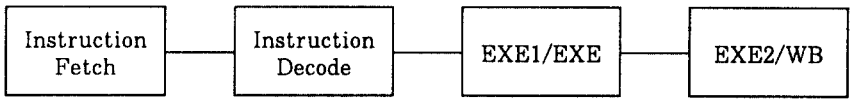



그림 3. HiDSP 파이프라인 구조
Fig. 3. Pipeline structure of HiDSP

EXE2 단계에서는 56 비트 MAC (Multiply-and-ACcumulation) 연산의 수행과 저장을 2개의 단계로 나누어서 수행을 하고 EXE 단계에서는 24 비트 산술 논리 연산을 수행하며 WB 단계에서는 그 결과를 레지스터 파일에 저장한다.

## Ⅲ. HiDSP 데이타패스

### 3.1 데이타패스의 병렬화

HiDSP는 데이타의 throughput 증가를 위해 4 단 파이프라인 구조로 설계되었으며, 일반적인 디지탈 신호 처리에서 자주 등장하는 MAC 연산과 병렬 데이타 이동을 고속 지원하는 파이프라인 구조로 이루어져있다. HiDSP의 연산부는 레지스터 파일을 기본으로 하는 아키텍춰로 구성되어 일반 연산의 source와 destination은 기본적으로 레지스터 파일 내의 각 레지스터들이 되며, 어큐뮬레이터는 주로 56 비트의 결과를 가지는 연산에 이용된다. 하드웨어의 관점에서 볼 때 어큐뮬레이터 기본의 데이타 연산 구조에서는 MAC 연산과 일반 연산이 동일한 데이타패스에서 수행되므로 MAC 연산에 비해 절반 정도의 수행 시간이 소모되는 일반 연산의 경우에도 MAC 연산과 동일한 지연 시간을 가지게 되며, 데이타 연산부를 곱셈과 ALU 연산으로 파이프라인화하는 경우 곱셈이 필요없는 일반 연산에서 곱셈을 위한 스테이지가 낭비되므로 파이프라인화가 용이하지 않다. 따라서 데이타 연산부가 칩의 임계 경로가 되어 전체적으로 프로세서의 성능을 저하시키는 요인이 된다. 이러한 문제점의 개선을 위하여 설계된 DEU에서는 MAC 유닛과 ALU를 분리하여 MAC 연산과 일반 연산의 병렬 수행이 가능하게 하였으며 MAC 유닛을 2단 파이프라인 구조로 설계하여 ALU는 MAC 연산의 영향을 받지 않고 한 파이프라인 스테이지에 연산을 수행하여 전체적으로 한 파이프라인 스테이지 소요시간마다 출력을 생성하도록 하였다.

### 3.2 고속 데이타 이동

HiDSP는 고속 데이타 이동의 지원을 위하여 병렬 데이타 이동을 포함한 모든 move 명령을 EXE1 스테이지에 수행한다. DSP56000의 경우는 2 워드 인스트럭션의 수행에 두 인스트럭션 사이클이 소요되므로, 2 워드 병렬 데이타 이동이 발생하면 주 인스트럭션의 수행 종료 후 move 스테이지를 추가하여 병렬 데이타 이동을 수행하여 파이프라인 latency가 증가한다. HiDSP에서는 2 워드 인스트럭션을 한 인스트럭션 사이클에 처리하여 인스트럭션 길이와 어드레싱 모드에 관계없이 모든 종류의 move 명령을 파이프라인 한 스테이지 내에 처리하므로 발생 가능한 파이프라인 지연의 증가를 방지하였다. 한편 move 인스트럭션의 데이타를 다음 인스트럭션에서 사용하는 경우, 데이타 의존관계가 형성됨에 따라 hazard에 의한 지연 시간의 발생을 방지하기 위하여 4상 클럭을 사용하여 타이밍을 맞추어 줌으로써 다음 인스트럭션의 오퍼랜드 패치 이전에 데이타 저장을 완료한다. 데이타 이동을 위하여 XDB, YDB, GDB를 사용하며, DEU 내부에 세 개의 별도의 버스를 두어 데이타 이동을 지원한다. 병렬 데이타 이동이 발생하면 레지스터 파일에 한 인스트럭션 사이클에 두 번의 read, 혹은 write가 발생하므로 표 1과 같이 4상 클럭을 충분히 활용하여 데이타 이동을 구현하였다.

### 3.3 DEU의 구조

그림 4에 설계된 DEU의 블록도를 보였다. DEU는 내부적으로 XIB (X Internal Bus), YIB (Y

표 1. 데이타 처리의 read/write 동작 시간
Table 1. Read/write timing for data execution

(RF : 레지스터 파일, Acc : 어큐뮬레이터)

| 스테이지 | | 클럭 | read/write 동작 시간 |
|---|---|---|---|
| MAC | ALU | | |
| EXE1 | EXE | Φ0 | RF/Acc read(ALU, MAC 연산) |
| | | Φ1 | RF/Acc read (데이타 move) |
| | | Φ3 | FR/Acc write (데이타 move) |
| EXE1 | WR | Φ1 | RF write (ALU 연산) |
| | | Φ3 | Acc write (MAC 연산) |

Internal Bus), ZIB (Z Internal Bus)의 세 버스를 통하여, 레지스터 파일과 어큐뮬레이터, MAC과 ALU 연산 블럭 간의 데이타 이동을 수행한다. X 및 Y 레지스터 파일의 출력 데이타는 XIB와 YIB를 통하여 다른 블럭으로 전달되며 ALU 연산의 결과와 어큐뮬레이터의 데이타는 ZIB를 통하여 레지스터 파일로 전달된다.

그림에서 좌측 패스는 24 비트 ALU를 나타내며 우측 패스는 MAC 유닛을 나타낸다. DEU는 내부에 2개의 1-read/1-write 포트를 가지는 4 x 24 비트로 구성된 X 및 Y 레지스터 파일을 가지며 이들은 24 비트 혹은 48 비트 데이타 연산의 source와 destination으로 사용된다. ALU는 24 비트 산술 연산과 논리 연산을 수행하며, 쉬프터는 1 - 8 비트 산술 논리 쉬프팅과 로테이션을 수행한다. 쉬프터는 면적과 속도를 고려하여 1, 3, 4 비트의 3 단 쉬프터를 직렬 연결하여 1, 3, 4의 조합에 의해 쉬프팅을 수행하는 3 단 배럴 쉬프터를 이용하였으며, 가감산기로는 24 비트 CSA (Carry Select Adder)를 사용하였다 [5][6]. MAC 유닛의 승산기는 부호 생성 방식의 Booth 인코더와 Wallace 트리로 구성된 24 x 24 signed multiplier를 사용하였으며 56 비트 가산기로는 CSA를 사용하였다. Rounding 로직은 56 비트 가산기의 출력을 받아서 사용자의 결정에 따라 로직이 enable되면 반올림된 결과를 출력한다. 블럭 부동 소수점 연산의 경우 발생되는 mantissa의 scaling을 위하여 사용자가 정의하는 SM (Scaling Mode) 비트에 따라 rounding 위치를 바꾸어 준다. Rounding은 SM에 따라 이에 해당하는 rounding 상수를 처리할 56 비트 데이타에 더해줌으로써 수행된다. 반올림 여부를 결정하는 알고리듬은 IEEE 표준인 convergent rounding 알고리듬을 사용하였다 [7]. Convergent rounding 알고리듬은 round-off 에러를 최소화 하기 위하여 하위 24 비트의 값이 1/2 인 경우의 round up/down 횟수를 통계적으로 비슷하게 배정하는 것이 특징이다. MAC 유닛의 source와 destination으로 사용되는 56 비트의 어큐뮬레이터 레지스터는 8 비트의 부호 확장부와 24 비트의 상위부 및 하위부로 구분되며, 상 하위부로부터 레지스터 파일 간의 데이타 송수신이 가능하다. Saturation은 56 비트 데이타의 24 비트 출력에 overflow가 발생할 때 수행되며, 이때 출력 값은 양수 혹은 음수의 최대값이 된다. Saturation 회로는 SM 비트에 따른 스케일링을 위하여 좌우 산술 논리 쉬프팅을 수행하는 다이나믹 스케일링 쉬프터를 포함한다. 정규화 쉬프터는 56 비트의 데이타에 대한 정규화를 위하여 1 비트 좌우 산술 논리 쉬프팅을 수행한다. 인스트럭션에서 정규화가 요구되면 어큐뮬레이터의 데이타는 정

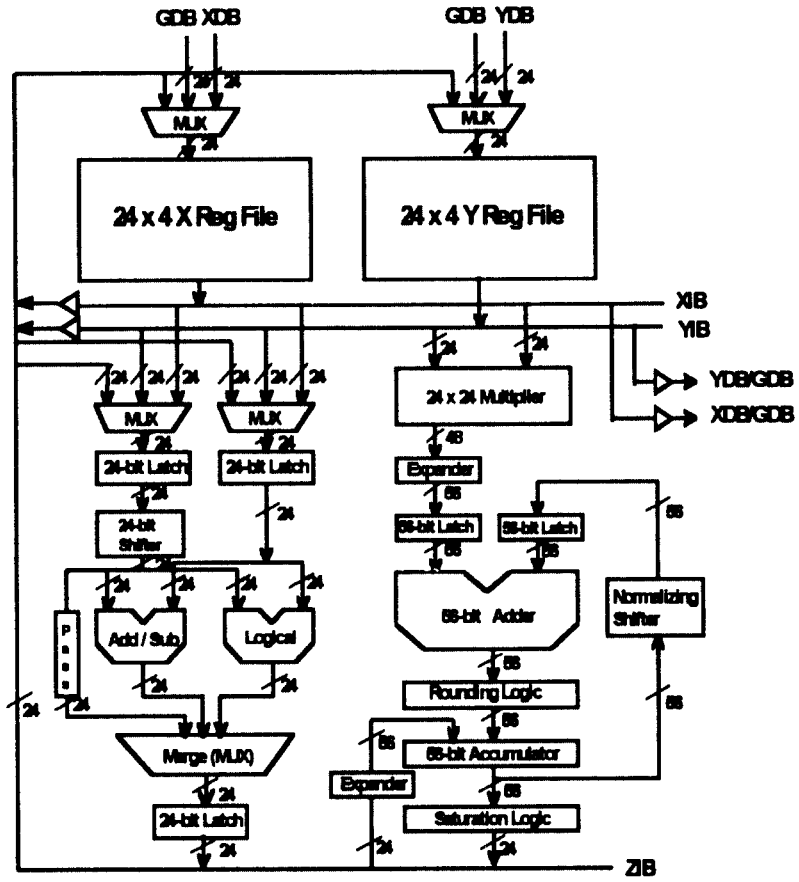
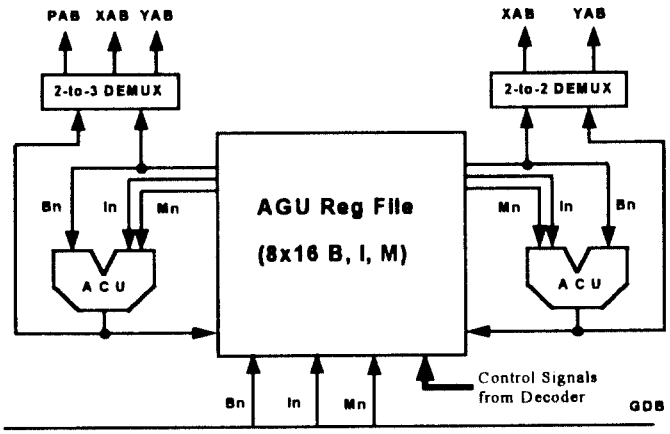



그림 4. DEU의 블록도
Fig. 4. Block diagram of DEU

규화 쉬프터에서 1 비트 쉬프트되며 ALU에서는 쉬프팅 양을 계수하여 DEU 레지스터에 저장한다. 이러한 과정은 정규화가 완료될 때 까지 반복된다. 정규화 순환 과정이 완료되면 어큐뮬레이터에는 결과 mantissa가 저장되며 카운트한 레지스터의 내용에 따라 exponent를 갱신한다. DEU는 프로세서의 현재 상태를 지시하는 16 비트의 상태 레지스터 (SR : Status Register)와 인터럽트 혹은 서브루틴 점프 발생시 SR을 저장하는 8개의 SR 스택을 포함한다. DEU에서 지원되는 상태 레지스터는 컨디션 코드를 나타내는 carry, overflow, zero, negative, unnormalized, sign extension, limitation과 모드를 지정하는 interrupt mask, scaling mode, bus wait/strobe, bus wait/grant 등으로 구성된다.

### 3.4 AGU

AGU는 오디오 신호 처리를 위한 다양한 어드레싱 모드를 지원하며, 데이타 처리와 동시에 병렬로 두 어드레스를 생성하는 하드웨어 블록으로 그림 5에 그 구조를 나타내었다. 어드레스와 옵셋, modifier 등의 저장을




# 24 비트 GDB를 제외한 모든 데이타패스는 16 비트임

그림 5. AGU의 블록도.
Fig. 5. Block diagram of AGU

위해 AGU는 8 개의 16 비트 레지스터들로 구성된 세 레지스터 파일, 즉 베이스 어드레스 레지스터 파일 (B), 인덱스 어드레스 파일 (I), modifier 레지스터 파일 (M)을 가지며, 파일 내의 모든 레지스터들은 초기에 FFFFh 로 preset 된다. Modifier 레지스터는 연산의 타입을 지정하며 지원되는 연산의 타입은 선형, bit-reverse, 모듈로 연산 등이다. 이를 지원하기 위하여 2 개의 어드레스 계산 유닛 (ACU : Address Calculation Unit)을 둔다. ACU는 선형, 역방향 캐리, 모듈로 가감 연산을 병렬로 수행하며 modifier에 의해 지정된 연산의 타입에 따라 결과를 선택한다. 즉 Mn의 값이 FFFFh이면 선형 연산을, 0000h이면 bit-reverse 연산을, 그 외의 값을 가지면 모듈로 연산을 수행하며, 이 때 Mn의 값은 연산의 모듈러스를 나타낸다. AGU는 GDB로부터 B, I, M 레지스터 파일의 내용을 받아들이며 ACU에서 레지스터 파일로부터 어드레스 (Bn), 인덱스 (Nn), 연산의 타입 (Mn)을 받아 어드레스를 생성하여 다시 Bn에 저장한다.AGU로의 어드레스 데이타의 이동은 사용자가 정의하는 상수와 DEU의 레지스터로부터 가능하다. 따라서 프로그램의 진행에 의해 어드레스가 갱신되어야 하는 경우 DEU 레지스터 값에 의해 값을 변경할 수 있다. 생성된 어드레스는 디멀티플렉서를 통하여 해당 어드레스 버스로 출력된다.

### 3.5 분산 제어 방식의 디코더 설계

분산 제어 방식의 디코더는 다수의 블럭들을 제어하는 고집적 디코더의 설계에 있더서 중앙 콘트롤러가 전담하는 것에 비해 디코딩 시간과 실리콘 면적을 감소시켜 주며 설계를 용이하게 한다 [8]. HiDSP의 데이타패스의 각 블럭들은 중앙 콘트롤러로부터 제어 신호들을 받아들여 해당 블럭의 동작을 위한 제어 신호를 생성한다. DEU가 사용하는 파이프라인 스테이지는 ID/OF, EXE1/EXE, EXE2/WB이며 이에 따라 OF 제어기, EXE1 제어기, WB 제어기, EXE2 제어기와 데이타 이동을 담당하는 MOVE 제어기로 구분된다. EXE1의 제어 신호와 MOVE 제어 신호는 동일한 파이프라인 스테이지 내에 수행되므로 종류에 따라 제어 신호를 mux를 통하여 선택한다. AGU는 ID/OF 스테이지에 중앙 콘트롤러의 제어 신호를 받아들여 주소 생성 방식에 적

합한 자체 제어 신호를 만들어 EXE1 스테이지에 주소를 출력하며 결과 주소를 레지스터에 저장한다. DEU와 AGU는 소프트웨어, 하드웨어 인터럽트와 점프, load, store 인스트럭션이 발생할 때 파이프라인을 지연시키기 위한 stall 신호와 외부에서 발생하는 wait 신호등에 의해 칩의 동작을 정지시키는 경우 halt 신호를 받아들여 파이프라인 상의 제어 신호를 disable 시키거나 상태 레지스터를 저장하는 등의 기능을 위한 제어 신호를 생성한다.

## Ⅳ. HiDSP 콘트롤러

### 4.1 콘트롤러의 구조

HiDSP의 콘트롤러는 4단 파이프라인상의 디코딩 단계에서 소모되는 시간을 줄이기위해 hardwired 로직으로 구현하였다 [9]. 콘트롤러의 구조는 그림 6에 보인 바와 같이 프로그램 어드레스 생성을 위한 PAB와 인스트럭션 디코딩을 위한 Decoder 및 인터럽트 처리를 위한 PIC (Program Interrupt Controller)의 3개 블록으로 구성되며, 프로그램 콘트롤러의 기능 수행을 위해 6개의 레지스터와 시스템 스택의 저장 장치가 존재한다.

이들 레지스터는 PC (Program Counter), SR (Status Register), LA (Loop Address), LC (Loop Counter), RASP (Return Address Stack Pointer), RCSP (Return Count Stack Pointer)로 구성되며 시스템 스택과 함께 루프 문을 하드웨어로 처리함으로써 칩의 성능을 향상시켰다.

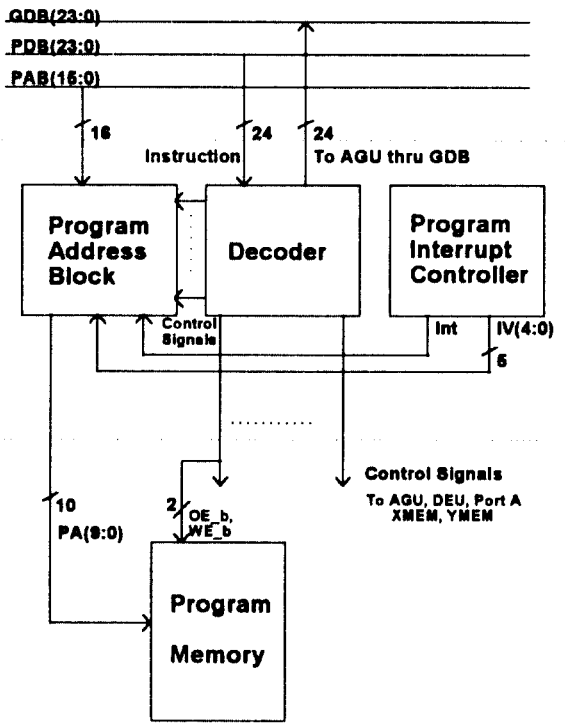



그림 6. 콘트롤러의 블록도
Fig. 6. Block diagram of Controller

4.1.1 PAB

PAB는 프로그램 주소를 생성하는 블록으로 PC (Program Counter), SS (System Stack), PAB_CTRL (PAB_Controller) 세 부분으로 구성되며, 이들의 전체적인 구성과 상호 연결은 그림 7에 보였다.

PC 블록은 프로그램 메모리의 주소를 생성하는 블록으로 인스트럭션의 길이에 따라 short type(24 bit)이면 1, long type(48 bit)이면 2가 증가한다. HiDSP는 이를 위해 decoder 블록에서 short/long을 지정하는 postEN 신호가 출력된다. PC 블록에서는 postEN 신호와 함께 JumpEN 신호를 입력으로 사용한다. JumpEN 신호는 PAB_CTRL에서 생성되어지는 신호로서 branch가 발생하였을때 '1'이 되어 PC 블록이 PAB를 통해 AGU에서 생성되는 target 주소를 읽어들여 프로그램 메모리의 주소를 출력한다. 주소의 출력은 4상 클럭의 2에 첫번째 주소가 출력되고 3에 두번째 주소가 출력된다.

SS는 진행되고 있는 프로그램의 주소가 순차적인 증가를 벗어나 JSR (Jump Subroutine), return 인

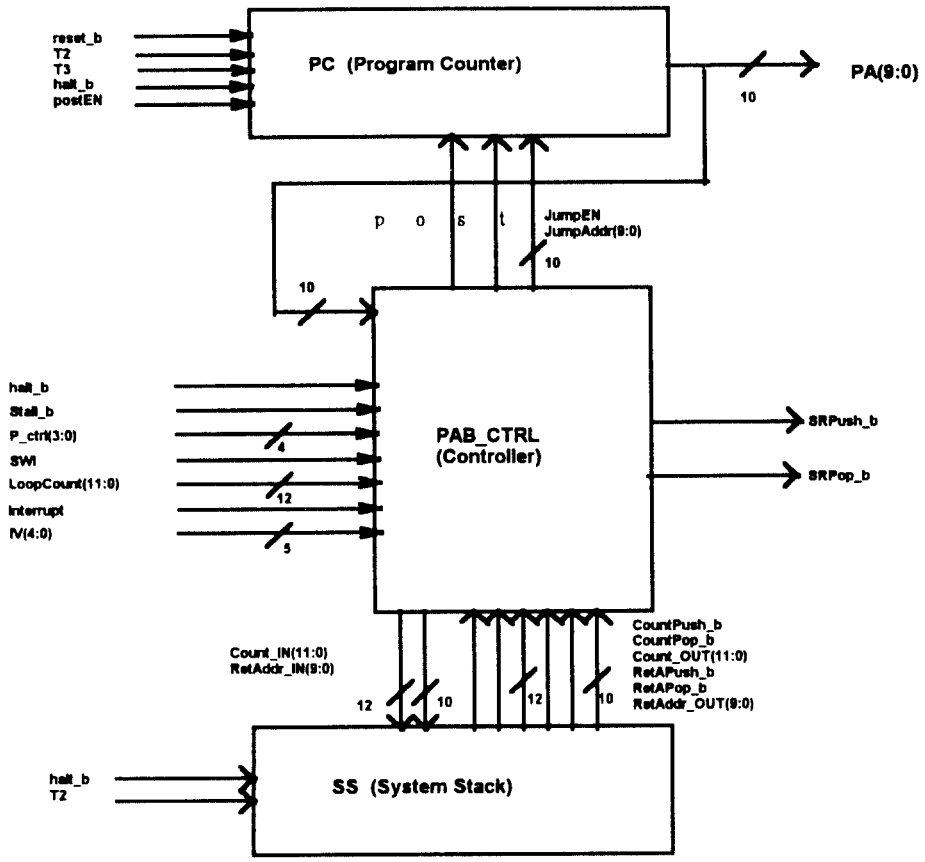



그림 7. PAB의 블록도
Fig. 7. Block diagram of PAB

스트럭션의 수행에서 현재의 상태를 임시적으로 저장하는 기억장치로 사용된다. 현재의 상태를 구성하는 레지스터로는 loop의 주소 (LA), loop의 현재 반복횟수 (LC), 상태 레지스터 (SR)등이 SS에 저장된다. 인터럽트, JSR (Jump Subroutine) 인스트럭션인 경우에는 LA와, SR이 저장되며, DO (loop의 시작) 인스트럭션, REPEAT 인스트럭션의 경우에는 현재의 LA, LC가 저장된다. SR은 DEU 블록에 위치하므로 이를 위한 SS는 DEU 블록에 위치한다. LA를 저장하기 위한 SS, RAS (Return Address Stack)은 16 개가, LC의 저장을 위한 SS, RCS (Return Count Stack)은 8 개가 존재한다. RASP, RCSP는 RAS, RCS 지정을 위해 사용되며, RASP, RCSP는 2에 출력되어 SS을 지정하게 된다.

HiDSP는 loop 문을 하드웨어로 처리하기 위한 인스트럭션 (DO, REPEAT)을 가지고 있으며, PAB_CTRL은 이들 인스트럭션을 효과적으로 처리한다. 반복되는 주소를 저장하기 위해 LC와 LA 레지스터가 존재한다. LC는 DO, REPEAT 인스트럭션일때 Decoder로부터 입력되는 12 비트의 반복횟수로 초기화하거나 ENDDO 인스트럭션의 경우 1 만큼씩 감소하며 인터럽트 발생시나 JSR 시에는 루프 처리를 중단하고 '0'으로 reset된다. 인터럽트 발생 혹은 JSR 인스트럭션 수행시는 루프문 안에서 수행중이던 인스트럭션의 주소가 저장되며, RETURN 혹은 IRETURN (Return from Interrupt) 인스트럭션시에 SS로 부터 pop된 주소가 다시 PC로 저장되어 루프문을 계속 수행한다.

#### 4.1.2 Decoder

Decoder 블록은 DSP core의 인스트럭션 수행을 위하여 콘트롤러에서 가장 기본이되는 블록으로, 인스트럭션 레지스터의 short 인스트럭션 또는 long 인스트럭션을 디코딩하여 파이프라인 제어에 필요한 제반 신호들을 PAB, DEU, AGU, Port A, 메모리 블록등에 출력한다. 24 비트 인스트럭션 또는 48 비트 인스트럭션을 래치하기 위해 24 비트의 IF 레지스터 두개가 존재한다.

출력되는 콘트롤 신호중 Stall_b 신호는 파이프라인을 지연시켜야 하는 경우에 activation 된다. 즉 jump시에 target address의 생성을 AGU 블록에서 수행 해야하므로 파이프라인 단계가 한 단계씩 지연되어야 하고 이를 위해서 출력된다. 파이프라인 stall시의 타이밍도를 그림 8에 보였다.

디코딩되어 출력되는 신호로 PAB에 인스트럭션 종류를 지정하는 4 비트와 LC 횟수를 지정하기 위한 12 비트가 있으며, DEU를 위해서 산술 연산 인스트럭션과 레지스터 지정을 위한 15 비트가 출력되며, 병렬 이동 지원을 위한 10 비트가 디코딩되어 출력된다.

AGU 블록을 위한 디코딩 신호로 AGU 블록의 레지스터 파일의 초기화를 위한 값과 immediate 값 16 비트가 GDB의 상위 16 비트를 통해 출력되며, 병렬 이동을 위한 immediate 값 9 비트는 GDB 하위 8 비트와 ToAGU_imm을 통해서 AGU 블록으로 출력된다. 데이타 값과 함께 AGU 블록의 레지스터 지정과 동작을 위한 신호 22 비트가 출력된다. Port A를 위한 디코딩 신호는 X, Y, P 지정을 위한 2 비트와 Read/Write, Enable를 위한 신호 한 비트씩이 출력된다. 또한 메모리 블록을 위해 X, Y, P에 각각 OE_b, WE_b 신호 2 비트씩 6 비트가 출력된다. 이외에도 모든 포트의 제어를 위한 신호가 생성된다.

#### 4.1.3 PIC (Program Interrupt Controller)

PIC 블록은 인터럽트 서비스를 요구하는 블록들로부터 발생되는 인터럽트를 관리하고 PAB 블록에 처리할 인터럽트의 주소를 생성시키는 역할을 한다. PIC 블록과 인터럽트를 발생시키는 블록은 handshaking 방식으로 역할을 수행한다. PIC 블록의 자세한 내용은 4.3 절에서 다룬다.

### 4.2 하드웨어 루프 처리

디지탈 신호 처리에서는 데이타에 대한 반복 연산이 빈번히 사용되므로 인스트럭션 처리의 관점에서 볼 때 효과적인 루프 문의 처리가 중요하다. HiDSP에서는 DO와 REPEAT 인스트럭션을 지원하며 이를 콘트롤러에서 하드웨어로 처리함으로써 overhead 없이 파이프라인 한 주기 당 루프 문내의 하나의 인스트럭션을 수행할 수 있다. 특히 디지탈 신호 처리에서는 한 루프 문이 다른 루프 문을 포함하는 nested 루프 형태를 이루는 알고리듬이 자주 쓰이므로 이를 소프트웨어로 처리하는 경우 loop unrolling에 의하여 프로그램의 크기가 증가하여 많은 메모리를 사용하므로 비효율적이다 [10]. HiDSP는 nested 루프의 처리를 콘트롤러에서 하드웨

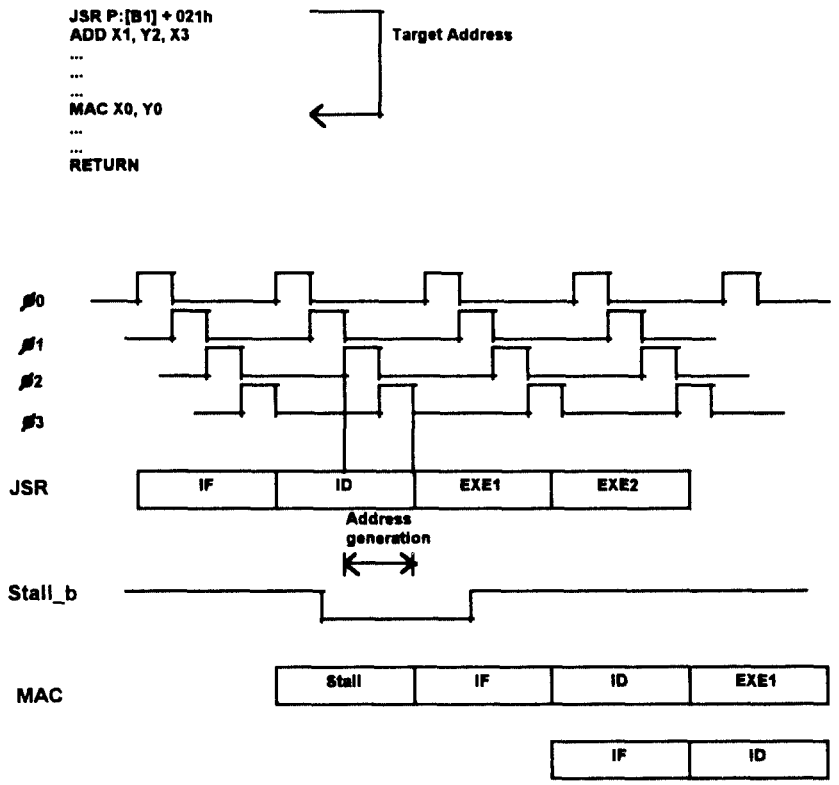
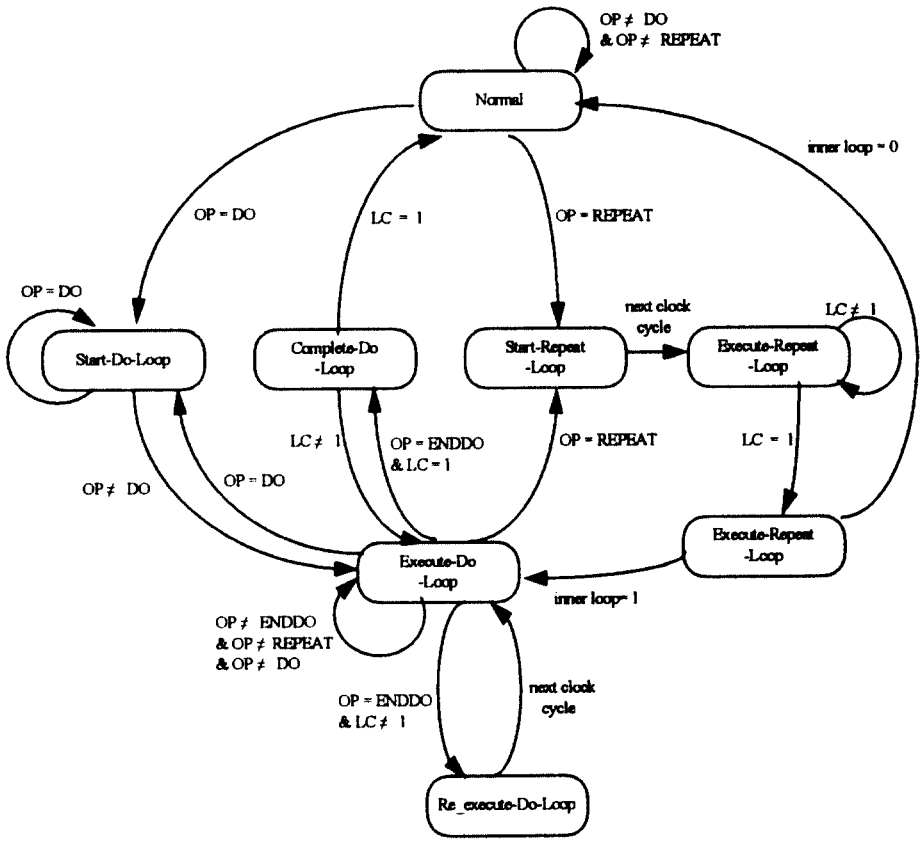



그림 8. 파이프라인 stall 타이밍도
Fig. 8. Timing diagram for pipeline stall

어로 처리함으로써 프로그램 메모리의 사용을 줄이며 고속으로 프로그램을 수행한다. DO 루프는 사용자가 반복 수행하고자 하는 인스트럭션의 갯수가 하나 이상일 경우에 사용되며 REPEAT 루프는 반복 수행하는 인스트럭션이 하나인 경우에 사용된다. 전술한 바와 같이 HiDSP의 콘트롤러는 하드웨어에 의하여 nested 루프를 처리하며 8단의 depth까지 지원한다. 루프의 반복 횟수는 12 비트 이하로 0h-FFFh 까지의 횟수를 지정할 수 있다. 루프 문의 파이프라인 지연없는 제어를 위한 콘트롤러의 상태도는 그림 9에 제시하였다.

루프 문 처리를 위한 콘트롤러의 상태는 PC와 LC값의 변화에 따라 Normal state, SDL state, EDL state, CDL state, RDL state, SRL state, ERL state 등으로 나뉘어지며, 각각의 상태에서 다음 상태로의 천이는 인스트럭션의 opcode 및 루프의 반복 횟수에 따라 이루어진다. 각 상태의 정의 및 PC와 LC 값의 변화는 표 2와 같다. Normal 상태는 루프 문 이외의 인스트럭션 수행 상태를 나타내며 SDL 상태와 SRL 상태는 각각 DO 루프 및 REPEAT 루프의 시작을 나타낸다. EDL 상태와 ERL 상태는 각각 DO 루프와 REPEAT 루프 내의 수행해야 할 인스트럭션을 수행하는 상태이며, CDL 상태는 DO 루프 문의 수행을 모두 끝내고 루프 문의 다음 인스트럭션으로 분기하는 상태이고, RDL 상태는 DO 루프의 수행을 한번 끝내고 다시 루프의 수행을 위해 jump 하는 상태를 나타낸다.




*Flag 'innerloop' is set when REPEAT is issued in a DO loop, and is cleared in "Normal" state

그림 9. 루프 문 처리를 위한 상태도
Fig. 9. State diagram for execution of loop statement

### 4.3 인터럽트 처리

HiDSP는 사용자가 프로그래밍 가능한 인터럽트를 다양하게 지원하며, 인터럽트 처리를 단순화하여 파이프라인 hazard 없는 콘트롤을 가능하게 한다. HiDSP는 단일 프로세서로 사용될 뿐만 아니라 병렬 구조로 혹은 host 프로세서의 slave 프로세서로도 동작 가능하도록 설계되었으며, 이들과의 연결 과정에서 데이타 및 콘트롤은 외부 인터럽트에 의해 전달 받음으로써 HiDSP의 동작과 동시에 외부와 인터페이스를 수행할 수 있다.

인터럽트 요청에 대한 서비스를 위한 PIC 블록은 발생된 모든 인터럽트를 받아들여 우선순위에 따라 가장 우선순위가 높은 인터럽트의 해당 인터럽트 벡터 어드레스를 생성하여 인터럽트 서비스를 수행한다. HiDSP에는 3 종류의 non-maskable 인터럽트와 13 종류의 maskable 인터럽트가 지원되며, 각 인터럽트들은 우선순위인 IPL (Interrupt Priority Level)이 배정되어 있어 인터럽트의 중요성을 나타낸다. IPL은 레벨 0 부터 3까지 있으며, 레벨 0부터 2까지는 매스킹이 가능하

표 2. 루프 문 처리를 위한 상태 및 동작
Table 2. States and operations for loop statement

| 상 태 | 동 작 |
|---|---|
| Normal State | PC ⟨ - OP(PC)<br>LC ⟨- 1 |
| Start_DO_Loop (EDL) State | PC ⟨- PUSH(LC)<br>LAstack ⟨- PUSH(LA)<br>PC ⟨- PC + 1<br>LA ⟨- PC<br>LC ⟨- LoopCount |
| Execute_Do_Loop (EDL) State | PC ⟨-OP(PC)<br>LC ⟨- LC |
| Re_execute_DO_Loop (CDL) State | PC ⟨- LA<br>LC ⟨- LC -1 |
| Complete_DO_Loop (CDL) State | PC ⟨- PC + 1<br>LC ⟨- POP(LCstack)<br>LC ⟨- POP(LCstack) |
| Start_Repeat_Loop (SRL) State | LCstack ⟨-PUSH(LC)<br>PC ⟨- LC + 1<br>LC ⟨- LoopCount |
| Execute_Repeat_Loop (ERL) State | PC ⟨- PC<br>LC ⟨-PC - 1 |
| Return_Repeat_Loop (RRL) State | PC ⟨- PC + 1<br>LC ⟨- POP(LCstack) |

고 레벨 3는 매스킹이 불가능하다. IPL은 DEU블록에 위치한 MR (Mode Register)의 하위 2 비트에 저장된다.

Maskable 인터럽트는 하위 블록에서 발생하여 PIC 블록으로 인터럽트를 요구하고 PIC 블록은 응답하는 방식인 handshaking 방법에 의해 처리된다. 인터럽트 요구가 있을 때 이의 처리 과정은 다음과 같다.

단계 1 : MR의 IPL에 따라 발생된 인터럽트들 중에서 우선순위가 높은 것을 선택하며, 선택된 인터럽트 요구에 대해 인지 (acknowledge)신호를 assert하고 우선 순위가 낮은 인터럽트 요구에 대해서는 인지 신호를 deassert시킨다.

단계 2 : PIC는 인터럽트가 발생하였음을 각 하드웨어 블록에 알리고 IV (Interrupt Vector) 신호를 PAB 블록에 출력한다.

단계 3 : PAB 블록은 인터럽트를 인식하여 현재의 주소와 LC를 저장하고, DEU 블록에 위치한 SR을 저장하기 위한 제어 신호와 PIC로부터 IV를 입력받아 프로그램 주소로 출력한다.

단계 4 : 디코딩 블록에서 IReturn이 디코딩되면 저장되었던 LC 및 SR을 pop시키고 저장되었던 주소로 되돌아 간다.

인터럽트 벡터 테이블은 프로그램 메모리의 $000H - $01FH 번지에 위치하며 칩이 동작을 시작할 때 bootstrap ROM에 의해 칩 내부로 로딩된다. 인터럽트 벡터 테이블은 2 개의 인스트럭션으로 구성되어 있으며 이들은 사용자가 프로그램할 수 있으며 인터럽트 처리가 끝났음을 IReturn 인스트럭션을 이용하여 알린다.

## Ⅴ. 실험결과

HiDSP의 하드웨어는 VHDL을 이용한 top-down 방식에 의해 설계되었다. Top-down 설계 방식은 bottom-up 방식의 설계에 비해 설계의 효율성과 시스템의 성능을 높여준다 [11]. 그림 10에 HiDSP의 설계과정을 나타내었다. 그림의 좌측 패스는 설계과정을 나타내며 우측은 시뮬레이션 단계를 나타낸다. 작성된 VHDL 코드에 대한 시뮬레이션을 거쳐 논리 합성과 최적화에 의해 생성된 스케메틱 데이타에 대한 유닛 딜레이 시뮬레이션과 라이브러리 매핑에 의해 추출된 게이트 딜레이를 포함하는 pre-layout 시뮬레이션을 수행하였으며, 검증이 완료된 스케메틱 데이타에 대한 배치 배선을 통하여 얻어진 레이아웃 데이타의 각 콤포넌트의 딜레이와 wire capacitance 값에 따르는 딜레이를 포함하는 post-layout 시뮬레이션을 통하여 공정 이전 단계의 동작 검증을 완료 하였다. 제작된 하드웨어는 웨이퍼 테스트와 본딩을 완료한 패키지에 대한 테스트를 통하여 동작 검증을 수행하였다. 레이아웃 이후의 시뮬레이션에서 나타난 시스템의 안정적인 동작을 위한 클럭 주파수는 30 MHz이며 best condition에서 40MHz 까지 동작함을 검증하였다. 테크놀로지 라이브러리로는 삼성전자의 0.8 $\mu$m 테크놀로지 게이트 어레이인 60000 kg을 사용하였으며,전체 칩은 170000 게이트 (약 70만 트랜지스터)로 나타났다. 설계된 HiDSP의 pre-layout 시뮬레이션과 post-layout 시뮬레이션에서의 결과 나타난 칩의 동작 주파수를 표 3에 보였으며, HiDSP의 칩 제원은 표 4와 같다.

제작된 HiDSP의 wafer 사진을 그림 11에 보였다.HiDSP에 대한 성능 평가를 위하여 벤치마크 프로그램으로 HiDSP와 동일한 24 비트 데이타패스와 내부 메모리 크기를 갖고있는 모토롤라사의 DSP56000에서 벤치마크로 사용된 20 탭 저역 FIR 필터, direct form IIR 필터와 통신 시스템에서 많이 사용되는 convolutional 인코더 프로그램을 HiDSP 인스트럭션으로 변환하여 실험하였다 [12][13]. DSP56000의 벤

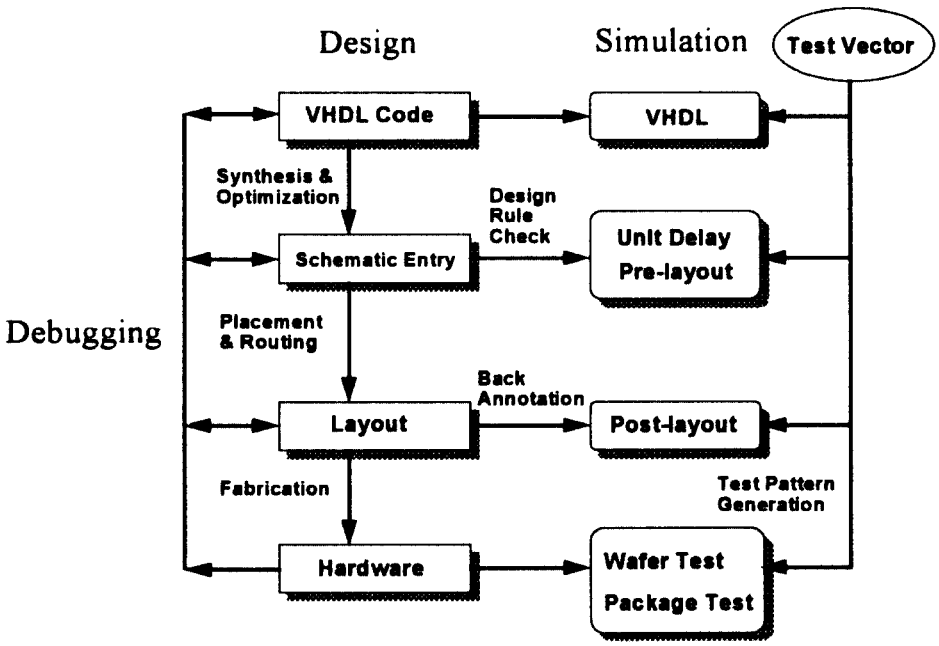



그림 10. HiDSP의 설계 과정
Fig. 10. Design process of HiDSP

표 3. 동작 속도 시뮬레이션 결과
Table 3. Results of simulation for speed estimation

| | Pre-layout Simulation | post-layout Simulation |
|---|---|---|
| best condition (5.5 volt, 0 ℃) | 55 MHz | 40 MHz |
| typical condition (5.0 volt, 25 ℃) | 40 MHz | 30 MHz |
| worst condition (4.5 volt, 75 ℃) | 22 MHz | 18 MHz |

표 4. HiDSP의 칩 제원
Table 4. HiDSP chip specification

| 항 목 | 값 |
|---|---|
| Power Consumption | 110mW |
| Die Size | 13.557 X 13.557 $mM^2$ |
| Package | 160 QFP |
| Clock Frequency | 30 MHz |
| Throughput | 15 MIPS |
| Number of Gates | 175600 |
| Technology | 0.8 $\mu$m Gate Array |
| Number of Power Pads | VDD : 16 / GND : 32 |

치마크 프로그램은 X 메모리상의 데이타 영역이 모두 초기화되고 Y 메모리 상의 계수가 모두 칩 내부로 로딩된 조건하에서 필요한 P 메모리의 크기와 수행 시간을 결과로 제시하였으므로 동작에 필수적인 초기화 과정이 선행되었음을 가정하였다. Benchmark 프로그램을 HiDSP의 어셈블러를 이용하여 HiDSP의 실행 가능한 인스트럭션으로 변환하여 실행하였을 때의 프로그램의 크기와 수행 시간을 같은 조건에서 DSP56000의 인스트럭션으로 실행하였을 때와 비교하면 표 5와 같다.

실험결과 프로그램 메모리의 사용면에 있어서는 컴팩트 코드 방식을 이용하는 DSP56000이 효과적임을 알 수 있다. 그러나 컴팩트 코드 방식의 인스트럭션 포맷은 디코딩이 복잡하여 디코더의 면적과 속도에 있어서는 비효율적이며 오퍼랜드 specifier를 사용하는 방식의 인스트럭션 포맷을 채택한 HiDSP는 프로그램 크기는 크나 디코더의 면적과 속도면에서는 효율적이다. 또한 빠른 디코딩과 MAC 유닛의 파이프라인 화에 의해 클럭 속도를 향상시켰다. 프로그램 클럭 사이클에 있어서 DSP56000에 비해 적은 인스트럭션과 기능을 제공하는 HiDSP가 IIR 필터의 경우 25 % 크게 나타났으며 FIR 필터의 경우 동일함을 알 수 있다. V.32 인코더의 경우는 HiDSP에 의한 클럭 사이클 수가 25 % 적게 나타났다. V.32의 경우 레지스터 파일 아키텍춰로 인한 데이타 이동 회수 감소로 코드 길이가 감소 하여

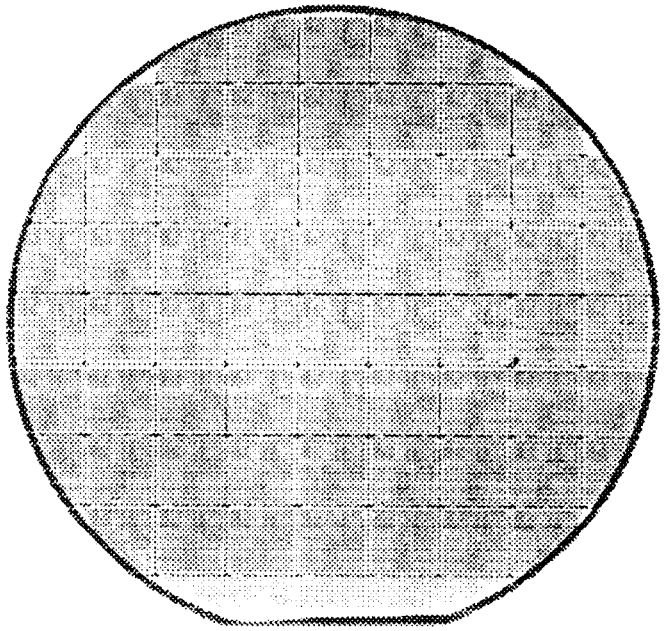


그림 11. HiDSP의 wafer 사진.
Fig. 11. HiDSP wafer photograph

표 5. HiDSP와 DSP56000의 benchmark 프로그램 크기와 수행 시간 비교
Table 5. Comparison of HiDSP with DSP56000 in terms of program size and execution time
(클럭 주파수 DSP56000 : 20 MHz / HiDSP : 30 MHz)

DSP56000/HiDSP

| | Benchmark Programs | | |
|---|---|---|---|
| 프로그램 | FIR 필터 | IIR 필터 | V. 32 인코더 |
| P 메모리 크기 | 10/18 | 6/12 | 29/38 |
| Y 메모리 크기 | 20/20 | n/a | n/a |
| X 메모리 크기 | 20/20 | n/a | n/a |
| 클럭 사이클 수 | 66/66 | 16/20 | 62/50 |
| 수행 시간($\mu$ sec) | 3.20/2.06 | 0.88/0.66 | 3.1/1.65 |

DSP56000보다 좋은 결과를 나타냄을 알 수 있다. V.32 인코더 프로그램에서 병렬 데이타 이동을 제외한 move 인스트럭션의 등장 회수는 DSP56000의 경우 18 번이며 HiDSP의 경우는 11 번으로 나타났다. 따라서 일반적인 통신 시스템과 같이 데이타 이동이 빈번한 예제의 경우 HiDSP의 성능이 우수함을 입증하였다.

## Ⅵ. 결 론

본 논문에서는 오디오 신호 처리용 디지탈 시그날 프로세서인 HiDSP의 VLSI 설계에 대하여 설명하였다. HiDSP의 데이타패스는 24 비트 고정 소수점 연산을 지원하며, 산술 논리 연산과 데이타 이동을 고속으로 지원하며 AGU를 통하여 각종 어드레싱 모드를 제공하며 주 연산과 병렬로 어드레스 생성한다. HiDSP는 4단 파이프라인 구조로 인스트럭션 처리를 병렬화 하였으며 hardwired 로직으로 콘트롤러를 구성하였다. HiDSP는 루프문를 하드웨어에 의해 파이프라인 hazard 없이 처리한다. 또한 인스트럭션 형태를 단순화 시킴으로써 인스트럭션 디코딩에 소모되는 시간을 줄여주어 4단 인스트럭션 파이프라인을 구성하였으며 critical section이 될 수 있는 디코딩 시간을 줄여줌으로써 30 MHz의 동작 주파수에서 안정적으로 동작함을 확인하였다. 결과적으로 칩의 동작 속도가 증가하고 파이프라인 깊이를 크게 해 줌으로써 전체적으로 파이프라인 throughput을 증가시킴으로써 성능의 향상을 가져왔다.

HiDSP는 VHDL을 이용한 top-down 방식으로 설계되었으며, 삼성전자의 게이트 어레이 kg60000 테크놀로지로 구현되었다. 사용자는 제공되는 HiDSP의 인스트럭션 세트를 이용하여 다양한 프로그램이 가능하며 프로그램된 알고리듬은 어셈블러에 의하여 HiDSP에 의하여 수행된다.

Benchmark를 통한 DSP56000과의 성능 비교에 있어서는 HiDSP가 적은 디코딩 시간과 MAC 연산부의 파이프라인 화에 의해 DSP56000에 비해 높은 클럭 주파수를 나타내며, 레지스터 파일 기본의 아키텍춰를 채택하여 다량의 변수를 사용하는 알고리듬에 있어서 우수한 성능을 나타내었다.추후 과제로는 게이트 어레이로 구현된 HiDSP을 스탠다드셀로 구현하고, 높은 클럭 주파수에서 동작할 수 있도록 새롭게 critical section이 된 AGU 블록의 재설계로 높은 주파수에서 동작가능하도록 해야한다. 또한 현재 개발되어진 어셈블러를 사용자가 사용하기 간편하고 최적화된 기계어 코드를 생성할 수 있게 보강하여야 하며 칩의 동작을 미리 확인할 수 있게 시뮬레이터의 개발이 필요하다.

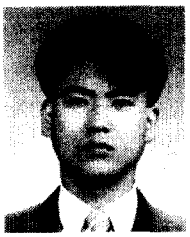
## 참고문헌


1. H. Ahmed and R. Kline, "Recent Advances in DSP Systems," IEEE Commun. Magazine, Vol. 29, No. 5, pp.32-45, May 1991.
2. P. Ruetz, "The Architectures and Design of a 20-MHz Real-Time DSP Chip Set," IEEE J. Solid-State Circuits, Vol. 24, No. 2, pp.338-348, April. 1989.
3. DSP56000/DSP56001 Digital Signal Processor User's Manual, Motorola Inc., 1990.
4. M. Smith, "How RISCy Is DSP," IEEE Micro Magazine, Vol. 12, No. 6, pp.10-22, Dec. 1992.
5. S. Kang, "Domino-CMOS Barrel Switch for 32-bit VLSI Processor," IEEE Circuits and Devices, Vol. 27, No. 2, pp.3-8, May, 1987.
6. K. Hwang, *Computer Arithmetic*, John Willy and Son's Inc. : NY, pp.1-96, 1979.
7. IEEE Standard for Binary Floating Point Arithmetic, IEEE Society, IEEE std. 754, 1985.
8. H. Nakagawa, "A 24-b 50-ns Digital Image Signal Processor," IEEE J. Solid-State Circuits, Vol. 25, No. 6, pp.1484-1493, Dec. 1990.
9. W. Stallings, "Reduced Instruction Set Computer Architecture", Proceedings of IEEE, Vol. 76, No 1, pp.38-55, Jan. 1988.
10. J. Spillane and Z. Navabi, "Describing Controlling Hardware in VHDL", in Proceedings of 4th Annual IEEE International ASIC Conference and Exhibit, pp.P8-1.1~P8-1.4, Sept. 1991.
11. J. Huber, *Successful Asic Design* : The First Time Through, Van Nostrand Reinhold : NY, pp.60-61, 1991.
12. Implementing IIR/FIR Filters with Motorola's

DSP56000/DSP56001, Motorola Inc., pp.2.10, 1990.
13. Convolutional Encoding and Viterbi Decoding Using the DSP56001 with a V.32 Modem Trellis Example, Motorola Inc., pp.2.3~2.8, 1989.

**李鍾和**(Jong Hwa Lee) 정회원

1967년 11월 서울생
1991년 2월 : 서강대학교 전자공학과 졸업
1993년 2월 : 서강대학교 전자공학과 공학석사 취득
1993년 3월~현재 : 서강대학교 대학원 전자공학과 박사과정 재학중
※주관심 분야 : VLSI 설계, Computer Architecture 및 Systems Design, CAD 등임



**鄭海龍**(Hae Ryong Jung) 정회원

1969년 서울생
1993년 2월 : 서강대학교 전자공학과 졸업
1995년 2월 : 서강대학교 전자공학과 공학석사 취득
1995년 3월~현재 : 삼성전자(주) 근무중
※주관심 분야 : Computer Architecture 및 Systems Design, CAD 등임

**黃善泳**(Sun Young Hwang) 정회원

1976년 2월 : 서울대학교 전자공학과 졸업
1976년~1981년 : 삼성반도체 주식회사 연구원
1978년 2월 : 한국과학원 전기 및 전자공학과 공학석사 취득
1986년 10월 : 미국 Stanford대학 공학박사학위 취득
1986년~1989년 : Stanford대학 Center for Intergrated Systems 연구소 책임연구원 Fairchild Semiconductor Palo Alto Research Center 기술자문
1989년 3월~현재 : 서강대학교 전자공학과 부교수
※주관심 분야 : CAD 시스템, Computer Architerturea 및 Systems Design, VLSI 설계 등임.

**張昌熙**(Chang Hee Jang) 정회원

1969년 7월 서울생
1993년 2월 : 서강대학교 전자공학과 졸업
1995년 2월 : 서강대학교 전자공학과 공학석사 취득
1995년 3월~현재 : 삼성전관(주) 근무중
※주관심 분야 : VLSI 설계, Computer Architecture 및 Systems Design, CAD 등임

**金 斌**(Sik Kim) 정회원

1971년 9월 대구생
1994년 2월 : 서강대학교 전자공학과 졸업
1994년 3월~현재 : 서강대학교 전자공학과 공학석사과정 재학중
※주관심 분야 : Computer Architectuer, VLSI 설계 및 Systens Design 등임