DiSPATCH
Firmware User's Manual

Revision 3.35
Updated 2 May 1997
Part Number: 880-3095-002

Vigra, a division of VisiCom Labs.

This is revision 3.35 of the *DiSPATCH Firmware User's Manual*.

This document was last updated on 2 May 1997.

CONTENTS

1. INTRODUCTION TO DISPATCH

Vigra's VME-based digital audio signal processing boards are equipped with one or more Digital Signal Processors (DSPs) which manage and process audio data on behalf of the VME host.

DiSPATCH is a powerful firmware package that gives the host a high-level interface to the flow and processing of audio data. DiSPATCH frees the host processor from the demanding real-time performance requirements of audio data control and offers an extensive collection of ready-to-use features which are accessed with a unified, orthogonal interface.

1.1 Features

Feature highlights of the DiSPATCH firmware include:

- Real-time digital mixing of up to 25 asynchronous audio streams per DSP.
- Directly supports many audio data formats, including PCM-8, PCM-16, ADPCM, μ-Law, and A-Law.
- Performs real-time Vector Quantizing audio compression/decompression for very low bit-rate (16 kbps) audio.
- Tight buffering for low-latency.
- Programmable voice signal detection.
- Full-duplex operation in all modes.
- Extensive diagnostics.
- 144 dB (24 bit) digital dynamic range for professional processing.
- High-speed interrupt-driven command interface.
- "RAM-to-RAM" data processing features for audio-crunching applications.

1.2 Document Overview

This document describes the function and operation of DiSPATCH.

The first portion of this document describes the system features implemented in the firmware, explains general control issues, and establishes the protocol used to communicate with DiSPATCH.

The second part of this manual describes each DiSPATCH command in detail, providing a comprehensive reference for the software application engineer.

2. AUDIO DATA PROCESSING

Each incoming or outgoing audio stream is processed by a series of functions in the DSP. DiSPATCH provides a large collection of signal processing and data conversion routines. The host selects which routines are applied to the audio data, and the processing is done transparently in real-time by DiSPATCH.

Data for playback is first read from DRAM prepared by the host. It is then converted to an internal data format, mixed in with other playback tracks, processed by the Play Processor, and finally sent to the Digital to Analog Converter (DAC) for output.

Record data originates from the Analog to Digital Converter (ADC). It is processed by the Record Processor, then converted to the user's requested data format and written to DRAM. Figure 2.1 illustrates the audio data path for playback and recording.

In general, the host controls five independent aspects of data processing:

- The audio data formats for playback.
- The mixing ratios for playback tracks.
- The processing modules to apply to outgoing (playing) data.
- The audio data format for record functions.
- The processing modules to apply to incoming (recorded) data.

Each of these control concepts is explained in this chapter, and the actual commands to implement them are detailed in the *DiSPATCH Command Reference* (Chapter 14).

2.1 Playback Overview

DiSPATCH has a versatile and powerful playback system that can be used in applications ranging in complexity from a simple playback tool to a sophisticated

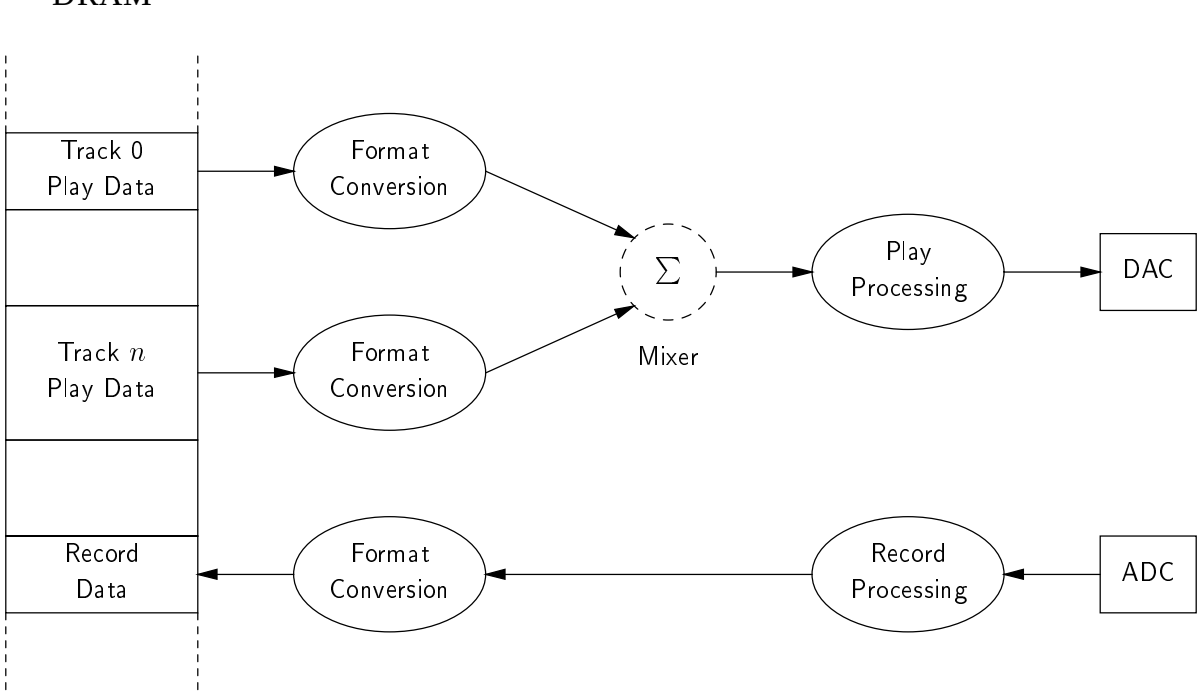

Figure 2.1: Audio Data Flow

event-driven multi-track simulation.

Simple playback example

A simple application of playback is for the host to load the on-board DRAM with a large series of samples and send a single DiSPATCH command to play them. In this command, the host specifies where in DRAM the samples begin and how many are to be played. This command and its arguments could be as follows:

`$0006`	Command code for PLAY_BUFFER
`$000000`	Samples start at the very beginning of DRAM
`$0EA600`	Play 480,000 16-bit samples (one minute at 8 kHz)
`$0001`	Use track number one for this playback

If nothing is currently playing when this command is issued, DiSPATCH immediately begins playing the specified buffer. While the buffer is playing, the host may

still execute other commands and even change playback parameters. DiSPATCH notifies the host when the buffer has finished playing.

Queued playback example

The single-buffer playback method works well for stand-alone blocks of audio that are not part of a larger stream. If a continuous flow of audio is to be played, the host needs a means to tell DiSPATCH what to play next, immediately after the first buffer completes.

By queueing "play requests," the host can have several buffers played in sequence, without any glitch or pause between buffers. This makes the series of buffers appear continuous, even though they may be located anywhere in DRAM.

To queue play requests, the host simply issues a new play command while another is in progress. When the DSP finishes the first buffer, it will immediately move on to the next play request. Up to ten pending requests can be queued for every playback track.

Double-buffered playback

Some applications require long-term continuous audio playback. These programs can use the play queue to implement an efficient double-buffered system. By using two buffers, the host can fill one buffer while DiSPATCH plays the other.

To do this, the host starts by requesting that the two buffers be played. DiSPATCH will immediately start playing the first buffer and queue the second request. When the first buffer finishes, DiSPATCH will notify the host and begin playing the second buffer. At this point, the host can put new data into the first buffer, and queue another play request. After each request is finished, the host puts new data in the buffer and issues a queued play request, always staying one buffer ahead of DiSPATCH.

This model is commonly known as "ping-pong" or double buffering. It enables the host to manage an unlimited-length playback with no interruptions.

Multi-Track Mixing

As shown in Figure 2.1, DiSPATCH can mix and play data from several buffers at once. This is similar to the operation of a multi-track tape-deck and mixer, with several independent audio streams summed together in real-time.

Each DiSPATCH track has its own independent play queue, which is processed in parallel with all the other tracks. DiSPATCH provides a total of 25 playback tracks, and the host may use as few or as many as needed.

The data from all the active tracks is mixed together into one audio channel. In addition to the master gain control, the host sets gain levels for each track to adjust the relative amplitude of each signal.

2.2 Recording Overview

The recording function of DiSPATCH is very similar in operation to the playback function. The host sends a command to start recording to a buffer, and DiSPATCH notifies the host when the buffer is full of incoming data. The host is free to send other commands, queue other record requests, and play buffers while recording is in progress.

Unlike playback, the record system does not have tracks or mixing, since there is only one stream of incoming audio. See Figure 2.1.

3. DSP INITIALIZATION

The DiSPATCH firmware package is uploaded at run-time to each DSP. This allows maximum flexibility, as no on-board components or EPROMs contain the firmware. New firmware revisions can be utilized by simply uploading the new image.

Each DSP on the VME audio board defaults to a reset (idle) state upon power-up. Before any DiSPATCH commands can be executed, the DSP must be brought out of reset and initialized. At that time, the DiSPATCH firmware program is uploaded to the board and DSP private RAM must be initialized.

This firmware upload is performed automatically by the programming library, but the upload procedure is detailed below for those applications that must initialize the DSP without the library.

The operations to be performed are shown, in order, below:

1. Un-reset and configure DSP
2. Upload firmware code
3. Initialize DSP X memory
4. Initialize DSP Y memory

Programmers using the DiSPATCH Programming Library need not understand the boot procedure, since the library provides the necessary routines.

3.1 The Host Port Interface

Until the DiSPATCH firmware is running, all communications with the DSP take place through the **host port**. This port is a memory-mapped communications channel between the DSP and the host. The firmware binary is sent through this channel to begin operation.

Please consult the hardware description in Chapter 13 of this manual for memory maps of specific Vigra audio boards. This text will refer to host port registers and bits by name rather than address, since the physical mapping varies between board models.

The following DSP control registers are used for the boot procedure and during DiSPATCH operation:

BOOT: Boot Source Select

This is one bit that determines where the DSP reads the DiSPATCH firmware program from. *Always* clear this bit to indicate that the VME host will supply the program image via the host port. Setting this bit to one indicates a boot from the EPROM. **The EPROM boot option is not supported.**

RESET: Hardware Reset

Clearing this bit forces the DSP into a hardware reset state. Each DSP has a separate reset bit, and one DSP can be reset without affecting any others. Setting this bit takes the DSP out of reset state and begins the boot process. All data and configuration settings are lost by resetting the DSP.

TXD: Transmit Data Register

This register transfers data to the DSP. It is actually three byte-wide registers which comprise one 24-bit register. The three bytes *must* always be written in high-middle-low order.

RXD: Receive Data Register

This register transfers data from the DSP to the host. It is actually three byte-wide registers which comprise one 24-bit register. The three bytes *must* always be read in high-middle-low order.

ICR: Interrupt Control Register

This read/write register contains the following bits:

HF0: Host Flag 0 Bit 3

This flag is used to signal the end of the firmware program upload. This bit should be clear before and during the upload, and then set by the host after the DSP has received the last program word. This signals the DSP to begin executing the new code.

RREQ: Receive Interrupt Request Bit 0

This bit enables (1) or masks (0) the VME interrupt generated when RXDF is set by the DSP. See Section 5.6.

ISR: Interrupt Status Register

This register holds two **read-only** flags that indicate the current status of the data transfer registers:

RXDF: Receive Data Register Full Bit 0

This flag is set by the DSP when the receive register from the DSP (RXD) is full and a data word is ready for reading. The host should never read a word from the RXD without first making sure that RXDF is set. The host can request to receive a VME interrupt when RXDF is set. The DSP will clear this bit when data is not available for reading.

TXDE: Transmit Data Register Empty Bit 1

This flag is set by the DSP when the transmit register from the host (TXD) is empty and new data can be transmitted. The host should never put a word in the TXD while TXDE is clear. Doing so could overwrite the word waiting in the register. See Section 4.4.1 for details. The DSP will clear the TXDE bit when the transmit register is full.

IVR: Interrupt Vector Register

This read/write register stores the 8-bit interrupt vector value to be sent with a VME interrupt. This interrupt vector is normally used to identify the device that generated the interrupt. See Section 5.6 for details.

3.2 Uploading Firmware

Each DSP has three private banks of 24-bit static RAM which are not directly accessible to the VME host. These banks are the P, X, and Y memory regions for the DSP. Program code must reside in P memory, while X and Y are used by the firmware for variables, internal buffering and state information.

Vigra provides the DiSPATCH firmware in two forms. The first is a set of three raw binary files (P, X, and Y), stored with each 24-bit word in high/middle/low byte order. The host must provide a means of reading these files and uploading the data to the DSP. Alternatively, these raw files can be converted into any form that is more conveniently accessed.

The firmware also comes in the form of a C source file, with each of the P, X, and Y images in a static data array. Any application that is to initialize the DSPs can include these arrays and upload them to the DSP when necessary. This static image is also part of the DiSPATCH library.

3.2.1 Boot Procedure

These steps represent the complete boot procedure from power-up to the operation of DiSPATCH. All the steps must be performed, and they must be done in the order listed.

Reset the DSP

Set: `RESET = 0`

This places the DSP into the reset (idle) state to stop all operations. Upon power-up, all Vigra audio board DSPs default to the reset state, but they should each be forced into reset by the host. This will ensure that the DSP starts from the known default state.

Select VME as the boot source

Set: `BOOT = 0`

The DSP needs to know where to get the boot image from. Setting this bit to zero directs the DSP to get the boot image from the host port on the VME bus.

Wait 100 microseconds

This is necessary to allow the reset and boot bits to settle properly before starting the DSP.

Un-reset the DSP

Set: `RESET = 1`

This starts the DSP running. The first thing the DSP does is check the BOOT flag to determine where to boot from. It then starts downloading code from the host port.

Wait 100 microseconds

This allows the DSP to start up and begin executing the bootstrap sequence.

Send the program to the host port

Check: `TXDE = 1`, then Send: `TXD = program words`

Now the host must send each 24-bit word of the program image to the transmit data register. The TXD register is actually three byte-wide registers, so the host must separate the 24-bit word into its low, middle, and high bytes. The high byte must be written to TXD first, followed by the middle byte, and the low byte last.

Before sending each 24-bit word, the host should wait until TXDE is set, indicating that the transmit register is ready for new data. Usually, the DSP is much faster than the host, so TXDE may appear to be always set.

Wait for the last transfer register to empty

Wait for: `TXDE = 1`

This is to ensure that all words, including the last word, were transferred correctly into the DSP. Again, it is unlikely that the host will have to wait long, if at all.

Set the HF0 flag

Set: `HF0 = 1`

This indicates to the DSP that the host is done sending the program image, and that the DSP should begin executing it.

Wait for Stage-1 boot progress message

Wait for message: Type = Special, Data = $424242

This progress message lets the host know that the DSP has properly received and started the DiSPATCH firmware. If the internal self-test fails, the Stage-1 message will not be sent. See Section 3.2.2 for a description of the self-test.

Upload X memory

Execute: `LOAD_X` command

This initializes the DSP's X region of private memory. See Section 3.2.3 for details on initializing X and Y data memory.

Upload Y memory

Execute: LOAD_Y command

This initializes the DSP's Y region of private data memory.

Wait for Stage-2 boot progress message

Wait for message: Type = Special, Data = $123456

The DSP will send the Stage-2 message when both X and Y memory images are in place. After the Stage-2 message has been received, DiSPATCH is fully operational and commands may be executed. See Section 3.2.4 for details on the Stage-2 boot progress message.

3.2.2 Internal Self-Test

Immediately after receiving the firmware program code, the DSP will conduct a thorough self-test, including an exhaustive test of all static RAM (SRAM) accessible by that DSP. When all tests have completed successfully, the DSP will send the "Stage-1 boot progress" message to the host.

If any of the self-tests fail, the DSP will *not* send the Stage-1 progress message, but will instead display a continuous pattern on the front-panel LED and halt all operations. This unique **Fail** pattern is visible on the LED as a repeating cycle from dim to bright and back to dim again. This fail signal will never appear under normal circumstances.

In the event of a self-test failure, the host will never receive the Stage-1 message. For this reason, the host should implement a timeout while waiting for the Stage-1 message. This timeout should be at least one second. If the DSP has not sent a Stage-1 message within one second after receiving the firmware program image, then there has been an error with the hardware or with the upload procedure.

The host should proceed with the initialization only after receiving this Stage-1 boot progress message. See Section 5.3 for details on the format and content of the Stage-1 boot progress message.

3.2.3 Initializing X and Y Memory

The DiSPATCH firmware also requires that the X and Y private RAM on each DSP contain certain initialized values on startup. These initialization constants are

available in the same formats as the firmware program code. The contents of X and Y memory are each stored in separate files or arrays.

Before initializing the X and Y memory regions on startup, the host must first upload the binary firmware program to the DSP, as described in Section 3.2.1. After sending the Stage-1 boot progress message, the DSP will illuminate the front-panel LED and wait for X and Y memory to be initialized.

At this time, the first command that must be executed is LOAD_X to transfer data to X memory. The entire contents of the supplied X Memory image must be transferred to the DSP at this time. The Load Dest address of the transfer must be $0000. See Section 14.8.1 for details on using the LOAD_X command.

The second command must transfer the entire contents of the Y Memory image to the DSP (at address $0000) using the LOAD_Y command (see Section 14.8.2).

3.2.4 Stage 2 Boot Progress Message

Immediately after completing the LOAD_Y command, the DSP will turn off the front panel LED and send the Stage-2 boot progress message. If the host does not receive the Stage-2 message within one second of completing the LOAD_Y command, then the firmware is not operational, and initialization has failed. See Section 5.3 for details on the format and content of the Stage-2 boot progress message.

When the LOAD_Y command completes and the Stage-2 message has been received, the firmware is fully operational and ready for use. DiSPATCH commands may then be executed.

4. HOST COMMAND PROTOCOL

After initialization, each DSP is controlled by host commands. Every function provided by the DiSPATCH firmware is accessed by one or more host commands. Since the commands provide a high-level control mechanism, the host programmer need not be familiar with the internal operations of the DSP. As more host commands are added to DiSPATCH the functionality and features of the firmware can be enhanced and modified while maintaining compatibility with existing host software.

For convenience, this document refers to specific parts of a command by names in **bold**. A host **Command Block** is an instantiation of a command. It consists of one **Command ID**, and zero or more **Argument** values. The set of available **Command ID**s is defined by the firmware and is called the **Command Set**.

The host executes a DiSPATCH command by placing a **Command Block** anywhere in DRAM, and then telling a DSP where to find and execute it. Both the host and the DSP refer to a **Command Block** by its DRAM offset.

Each DSP on the board is totally independent from the others. When running the DiSPATCH firmware, they do not communicate with one another directly or through DRAM. They may be reset, booted, and used independently at any time.

4.1 Using DRAM for Commands

Each Vigra audio board has a substantial quantity of dual-port DRAM that is accessible by both the VME host and the DSP. This DRAM is used for commands, arguments, and data. The programmer may choose to statically partition the DRAM for separate purposes, or to allocate it dynamically. The firmware imposes no restrictions on DRAM usage, and no DRAM is pre-allocated for DSP use.

Any commands and arguments that are not modified by the DSP can be re-used repeatedly to execute the same command. This enhances performance and reduces the load on the host and the VME bus since only the address of the command sequence needs to be re-transmitted for repeated execution of the same command.

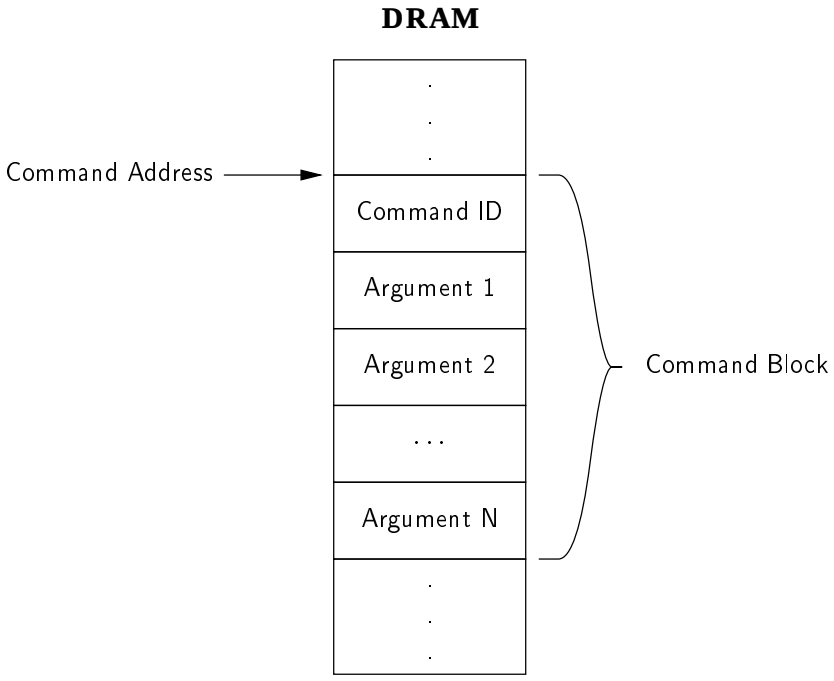

Figure 4.1: Layout of a **Command Block**

4.2 The Command Block

Each command has a unique 16-bit code (**Command ID**) which is the first word in every **Command Block**. A **Command Block** can be anywhere in available DRAM, provided there is enough room after the command for the necessary arguments. The **Command Block** must start on an even 16-bit word boundary. Figure 4.1 shows the organization of an example **Command Block**.

The DRAM address of the **Command Block** is called the **Command Address**. This **Command Address** is used as a unique handle for a particular command invocation. Since all Vigra audio boards have DRAM located at the beginning of the host memory map, the **Command Address** is always the offset from the base address of the VME card.

4.3 Command Arguments

Each command has a fixed number of required argument words, used to communicate command parameters to the DSP. Arguments are stored contiguously in DRAM immediately following the **Command ID**. The order and interpretation of each argument is defined by the given command, and documented in Chapter 14. Each DRAM location stores one 16-bit argument value. Two adjacent locations may be used for extended-range arguments.

Command arguments are in one of three data formats: Word, Extended Word, and Fraction. Each of these formats is described in the sections below.

4.3.1 Word **Format**

A word is the smallest argument type, occupying only a single 16-bit location in DRAM. It has a range from $0000 to $FFFF and is usually used for small integer values or ID codes. The byte order is big-endian, so the most significant byte is at the lowest address. Word values are always positive. In most environments, this is identical to C's "unsigned short int" data type.

4.3.2 Extended Word **Format**

Because the DSP accesses DRAM in 16-bit words, all individual argument words are 16-bit values. Parameters that demand a larger range will use *two* 16-bit argument words to form a 24-bit Extended Word. The first word represents the high 8 bits, and the second represents the 16 least-significant bits.

Since the DSP uses 24-bit words internally, all Extended Word command arguments are restricted to a range of $000000 to $FFFFFF, even though 32 bits are available. The Extended Word format is used for all DRAM address offsets, allowing a full 32 megabyte addressing range (24-bit address, 16-bit words).

The following C macro separates an integer value on the host into the Hi and Low portions of an Extended Word:

```
/* Separate a 24-bit Extended Word into the hi and low 16 bits */
#define HI_WORD(a) (((a) >> 16) & 0xff)
#define LO_WORD(a) ((a) & 0xffff)
```

4.3.3 Fraction **Format**

The 56001 DSP is not a floating point processor, but is instead optimized to operate on 24-bit fractional values from –1.0 to almost +1.0. All non-integer arguments use the 56001's native fractional format.

The most negative value possible (–1.0) is represented by the word `$800000`. The value –0.1 would be represented as `$F33333`. A value of 0.0 is simply `$000000`. The most positive value representable in this notation is 0.999999881, which is represented as `$7FFFFF`. This 24-bit format provides a dynamic range of 144 dB.

The C preprocessor macro shown below converts a floating point value in the host's native format to an integer representation suitable for use as a DSP Fraction. The host programmer need not know the details of the Fraction format, but must be aware of the range limits.

```
/*
 * Turn a floating point value (-1.0 <= flt < 1.0) into Motorola's
 * 24-bit DSP fractional representation.  Returns a C integer.
 * This clips the input value to between -1.0 and 0.999999881.
 */
#define DSP_FRAC(flt) (((flt) >= 1.0) ? 0x7FFFFF \
          : (unsigned)((flt) * (double)((unsigned)1 << 31)) >> 8)
```

Fractions are always 24-bit values stored in two DRAM words, using the same convention as Extended Words: the most significant 8 bits are stored in the first word, and the least significant 16 bits in the second word.

For a detailed explanation of the characteristics of this fractional format, consult Motorola's *DSP56001 User's Manual*.

4.3.4 Argument Modification by the DSP

Occasionally, the host may require small amounts of data from the DSP. Rather than allocate a buffer in DRAM and store the data there, the DSP can alter the argument values in the **Command Block**. Specific commands use this communication mechanism to return values.

Some commands query the DSP for information, and these commands will provide argument slots for the DSP to fill in with valid numbers. An example of this kind of command would be one that asks the DSP for its firmware revision number. When DiSPATCH executes the `VERSION` command, the DSP writes the firmware revision number into the argument words and returns a Command Acknowledge

message. The host can then read that DRAM location for the revision number. For this command, no arguments are passed from the host to the DSP, but some data is sent back from the DSP in the form of modified arguments.

Arguments that are modified by commands are specially marked with a "⇒" in the *DiSPATCH Command Reference* section of this manual (Chapter 14).

4.4 Starting a Command

After placing the appropriate **Command ID** and argument values into DRAM, the host can tell the DSP to begin executing the command. To start the command, the host simply places the **Command Address** into the host port transmit data register on the DSP. Upon receiving the **Command Address**, the DSP will read the **Command ID** and arguments out of DRAM, and begin executing the command. When the command finishes, the DSP will always return a response message, as described in Chapter 5.

4.4.1 When Can Commands Be Executed?

Each DSP can process one command at a time. The host can initiate DiSPATCH commands at any time, except when a command is already in progress. The host must always wait for the present command to finish **before** executing another command.

When these two conditions are met, DiSPATCH is ready for a new command:

1. A response for the last command has been received (Command Error or Command Acknowledge).
2. The TXDE bit is set (Section 3.1).

When a command is in progress, the host must always wait for a command response before issuing the next command. Each command response can generate a VME interrupt, if desired. See Section 5.6 for details on using VME interrupts.

The host must **never** start a command while the TXDE bit is zero. The TXDE bit is clear when there is a command in process or waiting to be processed. Placing data into the TXD register while TXDE is clear may overwrite the pending command and cause unpredictable behavior.

The DRAM containing the **Command Block** may be reused by the host at any time after the command finishes.

5. DSP TO HOST COMMUNICATIONS

All DSP responses to DiSPATCH commands consist of a host interrupt and a two-word message. To send a message to the host, the DSP places two 24-bit words of data into the host port receive data register, where they can be read by the host.

Each DSP message consists of two words, the Message Type, and the Message Data. The first word sent will always be the Message Type. This code tells the host what kind of message this is, and how to interpret the Message Data.

The Message Type can be one of five values. Each type is listed in Table 5.1 and described in the following sections.

Type Code	Message Type
0	Command Acknowledge
1	Buffer Completion
2	Special Data
3	Command Error
4	Signal Detect

Table 5.1: DSP Response Types

5.1 Command Acknowledge

After each command is accepted by the DSP, the DSP sends a Command Acknowledge message to the host. The data word of the message is the **Command Address** of the command responsible for that message. (The **Command Address** is described in Section 4.2.)

The Command Acknowledge response means that the DSP has received the command and accepted the arguments. In the case of immediate commands (commands which do not queue processes), the Command Acknowledge message indi-

cates that the command is done. For a buffered command like PLAY_BUFFER or RECORD_BUFFER, it means that the buffer has been queued and will start as soon as possible.

5.2 Buffer Completion

When the DSP finishes a queued request such as a play or record, the host will receive a Buffer Completion message. The data word of this message is the **Command Address** of the command responsible for the play or record. When the host receives the Buffer Completion message, the DSP is finished with the specified buffer in DRAM, which can then be used by the host.

For example, if the host issues a play request using the PLAY_BUFFER command, the DSP will reply with a Command Acknowledge immediately after parsing and accepting the command arguments. This play request will be queued and processed in turn, playing the data in the specified buffer. As soon as this buffer has finished playing, the DSP will send the Buffer Completion message, and the host is then free to reuse the buffer and **Command Block** DRAM for other things.

5.3 Special Data

Some messages from the DSP are unique to the protocol in that they do not have a **Command Block** associated with them. The only Special Data messages currently supported are the Stage-1 and Stage-2 boot progress messages described in Section 3.2.1.

The data words for Stage-1 and Stage-2 boot progress messages are as follows:

Boot Stage	Value
Stage-1	$424242
Stage-2	$123456

5.4 Command Error

The DSP will send a Command Error response instead of a Command Acknowledge when the **Command ID** or command arguments are considered invalid by the firmware. The data word of the message is the **Command Address** of the command that is in error. See Chapter 11 for a full description of the command error mechanism.

5.5 Signal Detect

DiSPATCH will send a Signal Detect message when the state of an active signal detector has changed. The message data is the **Command Address** of the SIGNAL_DETECT command that configured the detector. Immediately before sending the Signal Detect message, the DSP updates the status arguments of the Signal Detector. See Section 8.2 for information on using the signal detectors.

5.6 Interrupts

DiSPATCH can generate an interrupt on the VME bus when data in the DSP's receive data register is ready for reading. Enabling this interrupt will alert the host whenever a command response is sent by the DSP.

By setting the Interrupt Control Register (ICR), the host can choose if this interrupt is to be masked or enabled. If the RREQ bit in the ICR is set to one, an interrupt will be asserted whenever there is data waiting to be read from the receive data registers (RXD) and RXDF is set. This interrupt is cleared by the DSP when there are no further data words waiting to be read.

Note that the receive interrupt is actually cleared when the *low byte* of the data word read by the host. For this reason, the high and middle bytes of the data registers should always be read first.

The host can mask or unmask the interrupt at any time during DiSPATCH operation by clearing or setting the RREQ bit in the ICR.

The value of the VME interrupt vector is selectable by the host. At any time after starting the DSP, the host can change the interrupt vector by setting the value of the IVR register in the host port. The new vector will take effect immediately.

5.7 Audio Buffers

All audio samples are passed between the host and DiSPATCH via buffers in onboard DRAM. Audio sample buffers may be located anywhere in DRAM space. This allows the host application programmer maximum flexibility in selecting the most appropriate buffering strategy.

When a play or record command is issued by the host, the DSP will queue the request and return from the host command with a Command Acknowledge response. The queued requests are fulfilled in order, and the host is notified after

each one completes. When there are no more buffers in the queue, the DSP will stop playback or recording until more play/record host commands are received.

5.7.1 Buffer Address and Count

All DiSPATCH commands that operate on a buffer in DRAM require the arguments "Address" and "Count". These two values specify the region in DRAM that is the data buffer. Since these arguments are used so frequently, they are described here in detail.

Note that the buffer *must* fit entirely within the available DRAM, or undefined behavior may result. DiSPATCH does not verify that the Count or Address are legal values, so the host must ensure that these values fall within the defined limits.

Address

The DRAM address specifies the start of the data buffer. Buffers can start anywhere in the available DRAM, but must begin on an even 16-bit word boundary. The value of the address argument is simply the offset from the start of on-board DRAM.

The allowable range for all address arguments is from `$000000`, the first word in DRAM, to the last DRAM address, which varies among Vigra audio board models and configurations.

Since the DSP accesses all DRAM as 16-bit words, the address is specified as the *word* address. This is, of course, one half of the byte address. The address is a 24-bit unsigned value, yielding a total addressing range of 32 megabytes (16 megawords).

Count

The length of the buffer is specified by the Count argument. Like the Address, this is a 16-bit word reference, so the buffer length in bytes is two times the Count.

The last word in the buffer is as expected:

$$\text{End address} = \text{Address} + \text{Count} - 1$$

The Count is also a 24-bit unsigned value. A buffer Count of zero is *not* allowed, and should never be used.

Note that the format of the data in the buffer never affects the interpretation of the Count. The Count is always the number of 16-bit words in the buffer. This implies that the number of 8-bit samples (i.e., μ-Law) will be twice the number of words specified by the Count.

5.7.2 Checking Buffer Progress

During playback and recording, DiSPATCH will only alert the host when a buffer has finished. It is sometimes necessary for the host to ask DiSPATCH for the current position within a buffer while an operation is in progress. This function is provided by the PLAY_POS, RECORD_POS, and MONITOR_POS commands.

These commands return the *address* of the current position within the buffer in progress, and a *count* of how many samples are remaining in that buffer.

Since recording and playback are continuous operations, the buffer positions are constantly changing. By the time the host reads the returned position values, they may not be totally accurate; the buffer position is only accurate at the instant it is returned. However, if the record or playback operation is *paused*, then the buffer position is frozen and will not change until the operation is resumed.

For example, one of the most useful applications of the RECORD_POS is for stopped recordings. If the host does not know the length of the recording in advance, it can initiate a maximum-length record request. When the user stops the recording early, the host immediately does the following:

1. Pause the recording operation to freeze the buffer pointer (PAUSE_RECORD).
2. Check the record position (RECORD_POS).
3. Abort the recording (ABORT_RECORD).

Based on the returned address and/or count, the host can compute the number of samples that were actually recorded and save that portion of the audio buffer.

6. PLAYBACK TRACKS AND MIXING

The host may view each DSP on the audio board as a collection of asynchronous audio playback “tracks”, with each track independently controlled. The audio mix of all active tracks is sent to one digital to analog converter (DAC) for output from one channel.

These independent tracks provide the host with a means to play multiple unsynchronized audio streams together, while a single track can be used for one continuous audio stream. The host need not use more than one track, and all tracks are functionally equal.

This gives the application engineer the ability to choose a control model (parallel or sequenced) that best suits the application.

Like audio recording, there are no enforced buffering schemes or allocations, and the host may use any method that the hardware permits.

All play commands pertain to one audio buffer with a specified start address and length (i.e., no play commands auto-repeat). An interrupt and message are provided both upon the command acknowledge and playback completion for each buffer.

6.1 Mixing

The DiSPATCH firmware provides two different methods for mixing audio on the DSP. The first was described above as the concept of parallel asynchronous audio “tracks” that are mixed in real-time. This form of mixing is always available during playback simply by using different tracks for different streams.

The second form of mixing is used to play a collection of equally-sized buffers all at the same time. The set of buffers is played with a single PLAY_SUBBUFS command, and they all finish at the same time, returning a single Buffer Completion Message.

6.1.1 Sub-Buffer Mixing

There are several reasons a host may wish to use PLAY_SUBBUFS instead of separate PLAY_BUFFER commands:

1. Reduced bandwidth. Since one PLAY_SUBBUFS command takes the place of many PLAY_BUFFER commands, there is less interaction between the host and DSP. Only one command and completion response is involved.
2. Synchronization. It may be difficult for the host to start several buffers playing at the same time. PLAY_SUBBUFS guarantees that all buffers will start and finish playing at exactly the same time.
3. Simplicity. The host need not compute the start address for each sub-buffer. This is done by the DSP.

To mix sub-buffers, they must be contiguous in memory and must all be of the same size. For this reason, they are less flexible than playback on separate tracks, which can have buffers of any sizes and anywhere in DRAM. However, sub-buffer mixing may be more useful to some applications which deal exclusively with equal-sized buffers in contiguous memory.

The host specifies two extra parameters for sub-buffer mixing. The Subbuffers argument specifies how many sub-buffers are to be played, and the Spacing argument determines how far apart the sub-buffers are in DRAM. The value of Spacing is the offset between the start of the first and second sub-buffers.

The buffer start address for a PLAY_SUBBUFS command is the start address of the first sub-buffer. The Count argument is the length of just the *first* sub-buffer; all other sub-buffers are assumed to be of the same length.

Note that the Count (length of each sub-buffer) and Spacing of the sub-buffers are independent. This means that buffers can be configured to overlap (Spacing less than Count), or there may be unplayed regions between the sub-buffers (Spacing greater than Count).

When the sub-buffers are mixed, their relative gains are determined by the gains currently in effect for the tracks involved. Before, during, and after playback, the host may use SET_PLAY_GAIN to change the gain for each track.

See Figure 6.1 for an example of a memory layout and the arguments used for playing six synchronous sub-buffers of audio data. In this example, the specified number of sub-buffers to mix is six, while their separation is $0200 words.

Note that the example shows a sub-buffer Spacing equal to the Count. This is probably the most common case, but the firmware does not require that they be

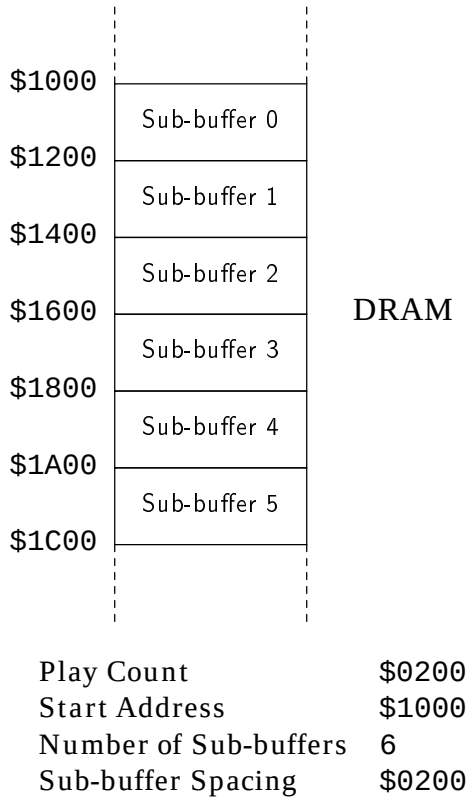

Figure 6.1: Mixed Playback Example

equal. If the Count were $0100 instead, then the command would play samples from the same sub-buffers, but only for half as long.

Once the sub-buffers and tracks have been scaled and summed, they are considered a single audio stream by the firmware. Any processes that act upon buffered data will do so after the sub-buffers have been mixed.

6.1.2 Sub-buffer Mixing and Tracks

Sub-buffer mixing is actually a simplified interface to track mixing. When the host issues a PLAY_SUBBUFS command, DiSPATCH proceeds to queue a play request for each sub-buffer. These play requests are each sent to separate tracks, starting with the one specified by the host.

For example, if the host executes a PLAY_SUBBUFS command on Track 2 specifying three sub-buffers, then DiSPATCH will translate this into three play requests. One will be queued on each of Tracks 2, 3, and 4. When these play requests complete (all at the same time), DiSPATCH will send a single completion message for the PLAY_SUBBUFS command instead of a completion message for each track. In all other respects, sub-buffer mixing is identical to manually playing the sub-buffers (with PLAY_BUFFER) on separate tracks.

An application may use both sub-buffer mixing and track mixing techniques at the same time. However, note that PLAY_SUBBUFS command queues requests on several tracks. While the sub-buffer playback is in progress, those active tracks are not available for standard playback.

6.2 Gain Levels and Mixing

In addition to any hardware output amplifier gain adjustments (programmable analog amplifier) and the Master digital gain, each playback track has a digital gain factor that determines the scaling to be applied when that track is mixed with the others.

These gain values range from -1.0 to almost 1.0, representing the range from total attenuation (0% gain) to the maximum level (200% gain). A gain value of 0.0 does not change the data at all, resulting in unity gain. A gain value of -0.5 attenuates by 50% while a gain value of 0.5 amplifies to 150%. The actual gain factor can therefore be computed as $(1.0 + \text{gainvalue})$.

Gain values for each track are totally independent and may be changed at any time before, during, or after playback. It is the responsibility of the host to ensure that the mixed sum of the audio data does not exceed the dynamic range of the DAC, or clipping and distortion will result. If all buffer gains are set no higher than $\frac{1}{n}$ where n is the total number of audio buffers, then the result is guaranteed to be within range.

7. PLAYBACK PROCESSING FEATURES

The playback processor in DiSPATCH performs the mixing and post-processing of outgoing audio. The post-processing features can be enabled or disabled by the host, and custom processing features can be added to the playback processor.

For the present version of DiSPATCH, playback performs the processing in the following order:

1. Convert all tracks to internal data format (PCM-16).
2. Mix tracks into one audio stream.
3. Perform speed-change processing (if enabled).
4. Apply the equalizer (if enabled).[1]
5. Apply the reverb (if enabled).[1]
6. Apply the Master playback gain (if enabled).
7. Perform Monitor processing (if enabled).
8. Send each outgoing sample to the DAC, incrementing the play counter.

7.1 Playback Audio Data Formats

During playback, DiSPATCH automatically converts the outgoing samples from any of several formats. The host selects an audio data format for each track. In most cases, audio data formats are completely independent, and each track can have the same format or a different data format. See Chapter 10 for a complete description of each of the available audio formats.

The host selects the audio data format for each playback track by use of the SET_PLAY_FORMAT command.

[1]The equalizer and reverb features are *unsupported*. See Section 14.12.

7.2 Speed Change

The playback processor in DiSPATCH has an algorithm for changing the speed of audio during playback. This can be used to slow down or speed up playback without altering the perceived pitch of the audio.

Changing the sample rate also changes the playback speed, but radically alters the pitch of the audio, reducing its intelligibility. By leaving the sample rate at normal and using the speed change algorithm instead, the pitch remains the same, but the audio takes more or less time. This can enhance the intelligibility of audio, especially speech.

The speed change can be used to decrease the speed of playback to 50% of normal speed (takes twice as long to play), or increase the speed to 200% of normal speed (takes half as long to play).

Note that this pitch-corrected speed change algorithm is very efficient, but adds a significant amount of noise to the audio. Also, bear in mind that higher speed playback does use proportionally more CPU time, so the DSP load is affected.

See Section 14.4.10 for details on the `SPEED_CHANGE` command and its arguments.

When the speed change is not needed, it should be disabled by setting the speed to 100% of normal speed (i.e., no change). The speed-change algorithm will then be automatically detached from playback processing.

7.3 Master Gain Adjustment

After all tracks have been mixed at their respective volumes (`SET_PLAY_GAIN`), they form one audio stream for processing. After processing, the host can select a gain adjustment to be applied just prior to output. This is referred to as the “Master Gain” control. It is a digital gain, done by the DSP before playback.

Like any other gain setting, the Master gain may range from zero to two times unity gain. Setting the gain to unity (no change) automatically disables the master gain part of playback processing. This is done to save DSP bandwidth when possible.

To set the Master gain, the host should use the `SET_PLAY_GAIN` command, with a special track number of `$FFFF`. Details can be found in Section 14.4.8.

7.4 Monitor

During playback, the host can request a copy of the finished outgoing samples, after all mixing and processing. By executing a MONITOR_BUFFER command, the host specifies where in DRAM to write the samples and how many samples to monitor.

The monitor feature can be thought of as a "recording" of the outgoing audio. The monitored data can be replayed without processing for the exact same audio output. Pausing and resuming playback also pauses and resumes monitoring.

The samples are placed in the monitor buffer after *all* digital playback processing has taken place, so the DRAM gets an exact copy of what is playing on the DAC. The format of the data placed in the monitor buffer is selected by use of the SET_MONITOR_FORMAT command.

The monitor function has its own request queue, similar to that of the play and record functions. The buffers are filled in the order they were requested. As with play and record, the host can abort the queue (ABORT_MONITOR), request the position of the active buffer (MONITOR_POS), and get a count of the pending monitor buffers (COUNT_BUFFERS).

When no monitor requests are pending, the monitor process is automatically detached from playback processing to conserve bandwidth.

7.5 Play Counter

The real-time nature of some applications requires that the host be able to keep track of how many outgoing samples have been played by DiSPATCH. This count can be provided in the form of a special counter in DRAM that is incremented by the DSP with every outgoing sample. This counter is called the **play counter**.

By default, the DSP will not maintain a play counter. The host must request that a play counter be used by issuing a REGISTER_COUNTER command (see Section 14.4.5). This command has two Word arguments which are set by the DSP instead of the host. These argument slots, combined, are the counter value in DRAM that the DSP will increment after every outgoing sample. Of course, multiple DSPs must not share a single REGISTER_COUNTER command.

The value in the counter will be reset only when the host issues a RESET_COUNTER command (Section 14.4.7). Note that the counter will increment continuously, even when no PLAY_BUFFER commands are in progress. The counter will not increment while playback is paused by the PAUSE_PLAY command.

The host should not manually reset or change the value of the play counter while playing, since a race condition could occur when the DSP tries to increment the current value. Note, however, that the host may safely change the play counter value while playback is paused. When playback is then resumed, the DSP will begin incrementing from the value set by the host. This method can be used to start the play counter from arbitrary values.

7.5.1 The Counter Value

The counter value is **31 bits** wide, and will wrap back to zero after reaching 2,147,483,647 ($2^{31} - 1$).

The value found in the counter reflects the number of **samples** sent to the DAC. The counter always increments at exactly the current sample rate, regardless of the current audio data formats or time-change factor.

Note that the counter does not always equal the number of words or bytes read from DRAM. That is, the DAC may receive samples faster or slower than words are taken from DRAM. This would be the case if a Speed Change (compress/expand) factor were in effect, or if the audio data were in a format other than PCM-16 (one word is one sample).

For example, if samples are stored as 8-bit PCM-8 data in DRAM, the counter will increment twice for every *word* read from DRAM, since one 16-bit DRAM word contains two samples. In addition, if the Slow-Down algorithm were running with a 200% speed ratio (twice as slow), then the counter would increment four times for every word read from DRAM.

Note that since the firmware buffers samples internally, the DSP will be somewhat ahead of the counter reading when it reads samples out of DRAM. The worst-case difference between the counter and read position depends on the internal block size used by the DSP, which is currently 512 samples.

If the application requires only an occasional check of the play position, the host should use the simpler PLAY_POS command to query the current position within a buffer. However, if the host needs to keep a constant and up-to-date count of the samples played, the play counter is more efficient.

7.5.2 Reading the Counter Correctly

Since the counter is stored as two discrete words, and the DSP frequently modifies the value in DRAM, the host must use a simple verification procedure when reading

the counter value, or an erroneous reading can occur.

The potential error (race condition) can occur when the DSP is changing one part of the counter value while the host is reading the other part.

For example, if the current counter value is `$0001FFFF`, the DSP will increment this value to `$00020000` at the next sample. The DSP writes the value in two stages, updating the lower half first.

If the host were to catch the DSP in the middle of the two writes, it would read `$00010000` as a counter value, since the lower bits have wrapped, and the upper bits have not yet been incremented.

The actual chances of this happening are fairly small, but to reduce the probability to zero, it is recommended that the host always read the counter twice and compare the upper words. If they differ, read the counter again until the upper word does not change between readings.

See Section 14.4.5 for more information on using the play counter.

7.5.3 Disabling the Counter

The counter function increases the per-sample processing overhead for the DSP, so it is best not to enable the counter until it is needed. It is also recommended that the counter be disabled when it is no longer used. The host should issue a `DISABLE_COUNTER` command when the counter is no longer needed. After this command, the host can use the play counter location in DRAM for any other purpose and the DSP will cease to increment it.

To re-enable the play counter, the host can issue another `REGISTER_COUNTER` command with the same address or a different one.

8. RECORD PROCESSING FEATURES

The record processor in DiSPATCH manipulates the incoming data from the ADC. Like playback processing, the features can be enabled or disabled by the host, and custom processing features can be added to the record processor.

At the time of this writing, DiSPATCH performs the record processing in the following order:

1. Fetch samples from the ADC.
2. Provide sidetone signal on output.
3. Perform signal detect (if enabled).
4. Apply the digital record gain.
5. Convert to the requested data format.
6. Write to DRAM buffers.

8.1 Audio Sidetone

Sidetone is a copy of the incoming signal that is digitally copied to the output channel. Sidetone audio is unbuffered, resulting in zero delay between input and output. Sidetone is most useful during recording, but is provided even while there is no recording in progress.

A unity-gain sidetone setting would pass all incoming audio immediately out to the output channel. This is well suited for monitoring the exact level of the recorded signal, but may generate feedback in a speaker-microphone setup. Usually, a small amount of sidetone is desirable for voice recording, similar to that provided by a telephone.

Sidetone can be turned off entirely by setting the gain to zero. Settings greater than unity gain will provide an amplified sidetone signal (i.e., louder than the incoming signal). See the `SET_SIDETONE` command for details on setting the sidetone gain (Section 14.5.5).

Since the sidetone operation requires some processing time, it should be turned off

when not needed. This frees up the DSP to be used for other processing.

Note that sidetone is muted when either record or playback is paused.

8.2 Signal Detection

When recording, the DSP can provide an indication to the host of the presence or absence of an audio signal with significant content in the speech range of the spectrum. Using this indicator, the host can decide what processing to perform on the incoming data. Note that the DSP does *not* automatically change the record data flow or behavior, but only provides signal and silence detection notification to the host.

The digital filters involved in signal detection are dependent on the sampling rate used during recording. They are calibrated to a sample rate of 8000 Hz. The signal-detect mechanism is not intended to be run at any other speed, since the filter characteristics would become inappropriate. At a sample rate of 8000 Hz, the filter is most sensitive to signals between 200 and 900 Hz, which encompasses most human speech.

There are four independent and identical signal detectors available, each with configurable parameters. The host may choose to use zero or more detectors at a time. Some applications will configure one for short-term signal detection, and another for long-term detection.

The detectors signal the host when the audio stream makes a valid transition between “energy” and “no energy”, as defined by the control parameters. The State argument to the SIGNAL_DETECT command will be set by the DSP to indicate if the new state is energy (1) or silence (0). The DSP will send the SigDetect Message via the usual protocol, sending the address of the SIGNAL_DETECT command block and causing an interrupt. See Section 5.5 for details on the actual transition message.

The detection parameters can be separated into “Time” controls and “Level” controls. Together, these two parameter groups adjust the sensitivity and time-scale of the signal detect algorithm.

Time controls specify how *long* a signal must be present or absent to constitute a valid transition. These parameters have no relation to the signal strength or content, but only how long it has been present or absent.

Level controls determine the range of input that is considered significant “energy”, based on analysis of small internal buffers. Level controls specify energy thresholds and ratios to qualify the audio signal.

By selecting appropriate level and time controls for each signal detector, it is possible to use one detector for short-term detection (e.g., on the order of spoken phrases), while another detector has a much slower response (e.g., speech activity on a scale of minutes).

8.3 Gain Control

The incoming data for recording can be scaled by a specified gain value before it is written to DRAM. This record gain ranges from zero to two times unity gain and is set with the SET_RECORD_GAIN command (Section 14.5.4).

8.4 Data Format

DiSPATCH automatically converts the recorded data into the audio data format specified by the host. The selected record format can be the same as or different from the format selected for playback. See Chapter 10 for a detailed description of each of the available audio data formats.

The host selects the recording data format by use of the SET_RECORD_FORMAT command (Section 14.5.6).

9. DATA TRANSFORMS

DiSPATCH also provides a system for processing audio data without analog input or output. The DSP reads input data from DRAM, performs a selected operation, and writes the results back to DRAM. This DRAM data manipulation is called a **transform**.

Boards without analog hardware (ADC and DAC) can *only* use the transform operations to process data. Boards with analog converters can also use the transforms to process DRAM data.

To transform a block of data, the host must specify seven things:

1. The format to convert *from*.
2. The format to convert *to*.
3. The address of the source data.
4. How many words of source data to convert.
5. Where to put it.
6. How many sub-buffers to use.
7. The spacing of the sub-buffers.

DiSPATCH reads the input data, converts it from one format to the other, applies a gain scaling, and writes it back to DRAM. This whole operation is performed before the DSP returns the Command Acknowledge message. During transform operations, all playback and record processing is suspended.

Multiple buffers of audio data can be converted in parallel and mixed together by the transform operation. The parameters for specifying mixed input buffers for transform are identical to those of PLAY_SUBBUFS (Section 6.1.1. Transform operations that do not require mixing simply specify one single buffer to convert (i.e., there is only one sub-buffer).

The amplitude of the input data is scaled by a gain value during transform. The gain factors are borrowed from the playback tracks for efficiency. That is, the gain scaling for the first transform sub-buffer is that of Track 0. The second sub-buffer will be scaled by the current gain for Track 1, and so on.

While the gain factors are borrowed from the playback tracks, the data formats are not. The host may specify any of the available data formats for the source and target data. All input sub-buffers for transform use the specified input data format. See Chapter 10 for a description of the supported audio data formats.

See Section 14.11.1 for details on the invocation and arguments of the TRANSFORM command.

10. AUDIO DATA FORMATS

Before an audio signal can be meaningfully represented digitally, an encoding format must be defined. The coding of digital audio is a topic of extensive study and development, with numerous data formats gaining acceptance. Each format has characteristics and constraints that may make it useful for some applications, and inappropriate for others.

DiSPATCH includes direct support for a number of popular audio data formats. These formats are converted internally by the firmware, transparent to the host. This relieves the host of all encoding and decoding that commonly accompanies powerful audio data formats.

Heterogeneous combinations of data formats are supported, and formats can be selected and changed at any time, without resetting or interrupting DiSPATCH operation.

DiSPATCH is designed to be easily extended to support new audio data formats. Contact Vigra if your application requires a data format not currently supported.

Each of the presently supported audio data formats is summarized in Table 10.1 and described in the following sections. Figure 10.2 is an illustration of the sample packing methods used.

Encoding	Format Code	Bits Per Sample	8 KHz Only?	Block Size
PCM-16	$0016	16	No	1
PCM-8	$0008	8	No	2
μ-Law	$0006	8	No	2
A-Law	$000A	8	No	2
ADPCM	$000D	4	Yes	4
VQ	$0002	2	Yes	160

Figure 10.1: Audio Data Formats

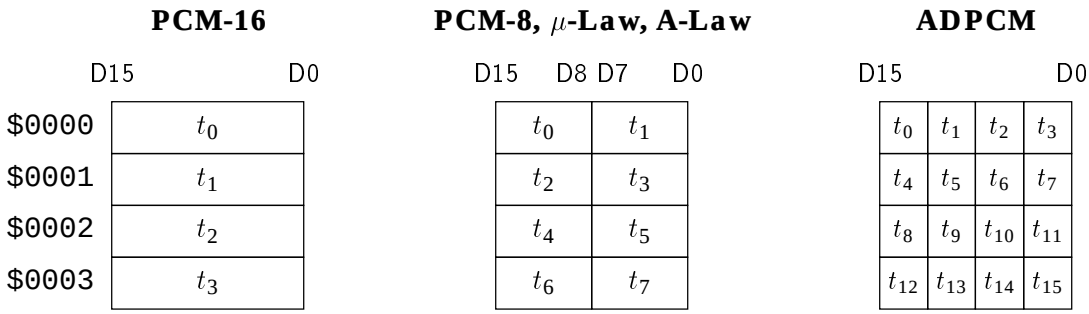

Figure 10.2: Audio Sample Packing

10.1 Linear 16-bit PCM (PCM-16)

This is the audio data format most commonly supported by high-precision audio applications, since it offers a high dynamic range (~96 dB) with the lowest processing demands.

Each sample is stored as a 16-bit value directly representing the waveform at discrete time intervals. The samples are big-endian signed values ranging from –32768 to 32767 (`$8000` to `$7FFF`). This format is usually compatible with the common C data type “signed short int”.

Using PCM-16, each 16-bit word of DRAM represents one sample.

This is the data format used internally by DiSPATCH. As a result, it requires no data conversion and therefore makes the most DSP bandwidth available for other processing. When applications place high processing demands on the DSP, this audio data format is preferred.

The disadvantage to PCM-16 is that it requires the most storage capacity of the data formats (two bytes per sample).

Upon booting, PCM-16 is the default data format for all playback, recording, and monitor commands.

10.2 Linear 8-bit PCM (PCM-8)

This data format is similar to PCM-16, but uses data values ranging from –128 to 127 (`$80` to `$7F`). This format is compatible with the C data type “signed char”.

Each sample is one byte, so two samples are stored in every DRAM word. The first sample is in the most-significant eight bits of the DRAM word, and the following

sample is in the lower eight bits.

Since each DRAM word contains two audio samples, odd numbers of samples are not supported. All PCM-8 buffers contain an even number of samples (word-aligned).

This format has considerably less dynamic range than PCM-16, but is often suitable for voice-grade audio where storage considerations outweigh fidelity. The quality of a PCM-8 signal is best when the input signal is close to full-power, utilizing all of the available resolution. Low-amplitude recordings will sound noisier than strong signals.

10.3 μ-Law Companded (uLaw)

The μ-Law audio format uses signal companding to provide an effective 12 bits of dynamic range using an 8-bit representation. This dynamic range enhancement is accomplished by using logarithmically higher precision for low-amplitude portions of the signal and sacrificing the accuracy of high-intensity regions of the waveform where precision is less important.

For many applications, μ-Law is an attractive alternative to PCM-8, as it provides a significant fidelity enhancement over PCM-8 with identical storage requirements.

Two μ-Law samples are packed into every 16-bit word in the same fashion, with the first sample in the high byte, and the next sample in the low byte. Like PCM-8, buffers always contain an even number of samples.

10.4 A-Law Companded (ALaw)

A-Law companding is very similar to μ-Law, but varies slightly in the actual signal encoding. Both formats have similar audio characteristics, but the data is not compatible. A-Law, as defined by CCITT Recommendation G.712, is commonly used in European voice communication applications, while μ-Law has been widely accepted in the United States.

The encoding, processing demands, and restrictions of A-Law are identical to that of μ-Law.

10.5 Adaptive Differential Pulse Code Modulation (ADPCM)

ADPCM is a sophisticated compression algorithm that provides a low bit rate of 32 kbps (4 bits per sample at 8 kHz). The algorithm is very similar to that described in CCITT Recommendation G.721, but the actual encoded data is not vector-compatible with the CCITT. In DiSPATCH, several optimizations were performed on the CCITT algorithm to make better use of the features of the 56001, while preserving all of the CCITT functionality. Users can expect similar characteristics and quality equal to the CCITT implementation, but it is important to note that the ADPCM data encoding is *not* compatible with the CCITT ADPCM algorithm.

ADPCM audio is history-dependent, so a given recording cannot be cut and spliced like one encoded with PCM or μ-Law. The algorithm does provide rapid error recovery, but a noticeable audio anomaly may be heard if an ADPCM data stream is damaged or spliced.

Each four bits of ADPCM encoded data represents one sample, so four samples are packed into one 16-bit word of DRAM. Like other formats, the first sample is placed in the upper-most bits of the word (bits D15 through D12). The following samples are placed, in order, in the less significant bits, with the fourth sample in the lowest bits (D03 through D00).

Since each DRAM word stores four samples, only multiples of four samples are allowed for ADPCM data, and data must be word-aligned.

Restrictions

Since ADPCM uses specialized digital filters, the sample rate during ADPCM playback or recording *must* be 8000 Hz. ADPCM is very computation-intensive. A 27 MHz DSP can only perform one ADPCM decode and one ADPCM encode at the same time.

Also, since ADPCM maintains a signal history for the adaptive filter, only two audio streams can be accommodated for encoding and decoding. For this reason, the host must never simultaneously play ADPCM data on two or more playback tracks, or the audio quality will be significantly degraded.

10.6 16-kbps Vector Quantized Compression (VQ)

Vector Quantized compression represents a significant advancement in real-time audio compression technology. Excellent audio quality is achieved with a low bit rate of 16 kbps by use of a sophisticated proprietary speech-compression algorithm.

Unlike the previously described audio data formats, VQ data is not a continuous stream of discrete samples. Instead, VQ-compressed audio is represented by "frames," which describe the speech patterns for the next 20 milliseconds of audio. These frames are the smallest discrete packet of VQ data. Each frame consists of twenty 16-bit DRAM words.

Because VQ is frame-oriented, it is very important that the data stream never lose part of a frame. For maximal compression, no frame sync information is stored, so the the VQ decoder assumes that a valid frame starts every 20 words from the start of data.

If the host recognizes that VQ data was lost, the host should either replicate the last valid VQ frame, or pad the bad data with zeros to make a full 20-word frame. This will ensure that the frames remain synchronized, and VQ will recover quickly ($\sim$40 msec) from the bad data within a frame.

Like ADPCM, VQ decoding is history-dependent, so splicing or jumping to discontinuous parts of a VQ frame stream may introduce a small audio anomaly in the output. This anomaly is less significant that that of ADPCM, and usually unnoticed. VQ will always fully recover after 2 correct frames (40 milliseconds).

The VQ frame size is enforced by DiSPATCH, requiring the host to always Play, Record, and Transform whole VQ frames. This means that word counts for VQ compressed data must always be multiples of 20 words.

VQ decoding is not as computationally expensive as ADPCM decoding, but encoding (compression) consumes more cycles than ADPCM.

Restrictions

Like ADPCM, VQ maintains a small internal signal history, so only one track at a time can play VQ data. Therefore, VQ data can not be digitally mixed during playback, or the audio quality will be degraded.

Also, since VQ is frequency-dependent, it must run at a sampling rate of 8000 Hz. No other sample rates can be selected when playing or recording with the VQ data format.

11. ERROR MANAGEMENT

Because the Motorola 56001 DSP was specifically designed for high-performance signal processing applications, several common processor capabilities were compromised to achieve maximum performance in signal processing. For this reason, operations that are trivial and fast on a conventional processor can result in significant performance degradation when implemented on the DSP.

Such functions include the extensive error checking and safeguards that are normally implemented in quality firmware. The DiSPATCH firmware performs only minimal error-checking and relies heavily on the host to follow the defined protocol and remain entirely within the specified ranges and limits.

For each argument, the range of allowed values is clearly defined in the *DiSPATCH Command Reference* (Chapter 14). While DiSPATCH does check some critical arguments for in-range values, it should be assumed that DiSPATCH will not catch errors for the host. Erroneous or degraded behavior is possible if the host does not obey protocol and parameter restrictions. Most critical argument values will be checked by the DSP, and graceful error detection and recovery will be attempted.

11.1 Error Conditions

A response message of type Command Error indicates that the DSP has rejected a command or its arguments. This usually indicates an invalid or out-of-range argument, or an unrecognized **Command ID**. The host should immediately issue a `FETCH_ERROR` command to get the error code to find the specific reason for the error condition. The error code remains available until the next error condition occurs, at which time it will be overwritten. Reading the error code resets it to a code of `0`, indicating no error.

Each command has a certain set of possible error codes. Codes other than these will not be returned for that command. The *DiSPATCH Command Reference* (Chapter 14) lists the possible error conditions for each command. Chapter 15 contains a summary of all possible error codes and their respective conditions.

12. COMPATIBILITY ISSUES

Future firmware revisions may add functionality by supplying more DSP commands. These new commands will have **Command ID**s that are not currently in use. All **Command ID**s from $0000 to $0FFF are reserved for use by Vigra firmware, while **Command ID**s from $1000 to $FFFF may be defined by customer applications. Vigra will make every effort to ensure that firmware commands of a given **Command ID** behave identically across minor revisions. The host application can use the VERSION command to check the firmware version number to verify compatibility. See Section 14.11.3 for details.

Error messages are each assigned a code value, ranging from $0000 to $FFFF. Vigra reserves all error code values from $0000 to $0FFF. Third-party customizations should use error codes $1000 through $FFFF only.

12.1 MMI Model Differences

Due to hardware differences, some Vigra audio boards support commands unique to that model or restrict the range of valid arguments to within hardware limits. Any such variances are noted in the relevant command descriptions.

If a function is not supported by the available hardware, an “Unsupported Command” error is returned to the host.

13. HARDWARE DEFINITIONS

The DiSPATCH firmware package runs on several Vigra VME audio boards, each with unique hardware features and register mappings. This chapter describes each audio board model and lists the physical address offsets for the registers used by DiSPATCH. A description of the function of each register can be found in Section 3.1.

All addresses are hexadecimal byte offsets from the VME base address of the audio board. All registers are byte-wide (8-bit).

13.1 Common Definitions

The definition of the bits within the ICR and ISR are common to all Vigra audio boards. They are described in Section 3.1 and summarized in Figure 13.1.

The transmit data register (TXD) and receive data register (RXD) are located at the same addresses, listed as Data High, Data Mid, and Data Low for the high, middle, and low byte respectively. Note that the transmit data register can safely be written even while there is data waiting in the receive data register. The waiting data will not be affected.

13.2 MMI-420

The MMI-420 is a four-channel digital signal processor, with one DSP and one front-panel LED for each channel. There are four megabytes of shared on-board DRAM.

The MMI-420 does not have any analog circuitry, and is therefore incapable of audio playback and recording. All DiSPATCH commands are accepted by the MMI-420, but no analog operations will be performed. In effect, playback and recording are permanently paused.

The BOOT source select bit is the lowest bit (D0) in the Boot Register. The RESET

Interrupt Control Register (ICR)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	HF0	0	0	RREQ

Interrupt Status Register (ISR)

D7	D6	D5	D4	D3	D2	D1	D0
x	x	x	x	x	x	TXDE	RXDF

Figure 13.1: ICR and ISR bit definitions.

	DSP A	DSP B	DSP C	DSP D
Boot	0x3FFE03	0x3FFE07	0x3FFE0B	0x3FFE0F
Reset	0x3FFE13	0x3FFE17	0x3FFE1B	0x3FFE1F
Data High	0x3FFE97	0x3FFED7	0x3FFF17	0x3FFF57
Data Mid	0x3FFE9B	0x3FFEDB	0x3FFF1B	0x3FFF5B
Data Low	0x3FFE9F	0x3FFEDF	0x3FFF1F	0x3FFF5F
ICR	0x3FFE83	0x3FFEC3	0x3FFF03	0x3FFF43
ISR	0x3FFE8B	0x3FFECB	0x3FFF0B	0x3FFF4B
IVR	0x3FFE8F	0x3FFECF	0x3FFF0F	0x3FFF4F

Figure 13.2: MMI-420, MMI-4210, and MMI-4211 register map.

bit is the lowest bit in the Reset Register.

The register map for the MMI-420 is shown in Figure 13.2.

The four megabytes of on-board DRAM extends from byte address `0x000000` to `0x3FFDFF`. The General Purpose (GP) front-panel LED is not controlled by DiSPATCH, but may be set directly by the host at the low bit of address `0x3FFE43`.

13.3 MMI-4210

The MMI-4210 is Vigra's model name for an MMI-420 with a monophonic analog daughterboard. The daughterboard is the DB-421, which provides all audio input and output functions for the board.

Since the MMI-4210 has an MMI-420 as its base, all DSP register addresses are identical for the two boards. See Section 13.2 for the description and register map.

13.3.1 Analog Mixers

In addition, the MMI-4210 has four daughterboard registers that control the routing of the analog output signal. These mixers select which DAC output is connected to which front-panel outputs. A given DSP can output to any combination of output ports. The mixer control registers are at the byte offsets shown in Figure 13.3.

Mixer A	0x3FFF91
Mixer B	0x3FFF93
Mixer C	0x3FFF95
Mixer D	0x3FFF97

Figure 13.3: MMI-4210 mixer register map.

The lower four bits of each mixer register select which DSP outputs are routed to one audio output port on the front-panel. The lower four bits of the mixer registers are used to enable (1) or disable (0) the output from each DSP. Mixer A selects the outputs for channel A, mixer B selects the outputs for channel B, etc.

The bits represent the DSP outputs as shown in Figure 13.4. For example, to route only the output of DSPs A and C to audio channel B, the host should write a `0x5` to Mixer B.

Setting all bits to zero disables all output from that channel. The default mixer state for all channels upon power-up is zero (no output).

Note that the mixers route only the audio *output*; the input channels are always passed straight through to their respective DSP.

Mixer Output Select

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	DSP D	DSP C	DSP B	DSP A

Figure 13.4: MMI-4210 mixer bit definitions.

13.4 MMI-4211

The MMI-4211 is Vigra's model name for an MMI-420 with a DB-422 analog daughterboard. The daughterboard provides all audio input and output functions for the board.

Since the MMI-4211 has an MMI-420 as its base, all DSP register addresses are identical for the two boards. See Section 13.2 for the description and register map.

13.4.1 Analog Mixers

The MMI-4211 has four analog signal output routers (like the MMI-4210). On the MMI-4211, these routers are exclusively under DSP control. The host must use the ROUTE_OUTPUT command to select which output port will receive the signal from a given DSP. The default mixer state for all channels upon power-up is **on** (output enabled).

13.4.2 Selectable Input Source

The MMI-4211 also has a software-selectable input source for each audio channel. By executing the SELECT_INPUT command, the host can select between the Microphone and Line-In jacks on the front panel of the MMI-4211. Each channel has an independent selection. All channels default to Line-In upon initialization.

13.4.3 Front-panel LEDs

In addition to the four front-panel red LEDs, there are four small surface-mount LEDs on the DB-422, visible from the side of the board. At the present time, these LEDs are exclusively for Vigra's internal development and production test systems. They should be disregarded by the user.

13.5 MMI-210

The MMI-210 is a two-channel audio signal processor with two independent DSPs. The board is available with either one or four megabytes of on-board DRAM. DiSPATCH supports both boards.

MMI-210 Mode Register

D7	D6	D5	D4	D3	D2	D1	D0
0	0	RESET B	RESET A	BOOT B	BOOT A	0	0

Figure 13.5: MMI-210 mode register bit definitions.

The SRAM chips for DSP private memory must be at least 32K each to run the DiSPATCH firmware. SRAM chips smaller than 32K must be upgraded before running DiSPATCH. Contact Vigra for replacement, if necessary.

The MMI-210 has one write-only 8-bit register for the two RESET bits and two BOOT bits. The bits in this "mode register" are shown in Figure 13.5.

For the one-meg MMI-210 model, the DSP host port registers and the mode register are found at the address offsets shown in Figure 13.6. The four-meg model adds `0x300000` to each offset, as shown in Figure 13.7.

	DSP A	**DSP B**
Data High	0x0FFE57	0x0FFE97
Data Mid	0x0FFE5B	0x0FFE9B
Data Low	0x0FFE5F	0x0FFE9F
ICR	0x0FFE43	0x0FFE83
ISR	0x0FFE4B	0x0FFE8B
IVR	0x0FFE4F	0x0FFE8F
Mode Reg	0x0FFE03	

Figure 13.6: Register map for MMI-210 with **one** megabyte of DRAM.

13.6 MMI-105

The MMI-105 is a one-channel audio signal processor with one DSP. The board is available with one, four or eight megabytes of on-board DRAM. DiSPATCH supports all three boards.

The SRAM chips for DSP private memory must be at least 32K each to run the DiSPATCH firmware. SRAM chips smaller than 32K must be upgraded before running DiSPATCH. Contact Vigra for replacement, if necessary.

	DSP A	DSP B
Data High	0x3FFE57	0x3FFE97
Data Mid	0x3FFE5B	0x3FFE9B
Data Low	0x3FFE5F	0x3FFE9F
ICR	0x3FFE43	0x3FFE83
ISR	0x3FFE4B	0x3FFE8B
IVR	0x3FFE4F	0x3FFE8F
Mode Reg	0x3FFE03	

Figure 13.7: Register map for MMI-210 with **four** megabytes of DRAM.

MMI-105 Mode Register

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	RESET	BOOT	0	0

Figure 13.8: MMI-105 mode register bit definitions.

The MMI-105 has one write-only 8-bit register for the RESET and BOOT control bits. The bits in this “mode register” are shown in Figure 13.8.

For the one-meg MMI-105 model, the DSP host port registers and the mode register are found at the address offsets shown in Figure 13.9. The four-meg model adds `0x300000` to each offset, and the eight-meg model adds `0x700000` to each offset.

Register	Address
Data High	0x0FFC57
Data Mid	0x0FFC5B
Data Low	0x0FFC5F
ICR	0x0FFC43
ISR	0x0FFC4B
IVR	0x0FFC4F
Mode Reg	0x0FFC03

Figure 13.9: Register map for MMI-105 with **one** megabyte of DRAM.

14. DiSPATCH Command Reference

Each of the DiSPATCH firmware commands is documented in this chapter. Users should first read the preceeding text of this document to become familiar with the operation of DiSPATCH, and the protocol for executing commands.

14.1 Notation

All values are shown in decimal unless otherwise noted. Values preceeded by a dollar-sign ("$") are hexadecimal (base 16) values.

14.2 Command Description Format

Each command description has the following parts:

14.2.1 Command Name

The command name is shown in CAPITAL_COURIER at the heading of each description. The name itself is arbitrary, since the firmware identifies commands only by their **Command ID**.

14.2.2 Summary

This is a short summary of the function of the command.

14.2.3 Code

Each command has a unique 16-bit integer code, shown here in hexadecimal. This is the **Command ID** used by the host to specify the command to execute. These are fixed codes, and will not change between firmware versions.

14.2.4 Arguments

This is an ordered list of arguments that follow the **Command ID** in any invocation of the command. Each argument is named and described briefly.

When executing a command, the arguments must be in the same order in DRAM as they are in the listing. The first argument listed must immediately follow the **Command ID** in DRAM, and any other arguments follow in order.

All specified arguments must be present when executing a command; there are no default values.

Extended Word and Fraction arguments are shown as two words. The first word represents the high 8 bits and is tagged with "`(Hi)`". The second word is the low 16 bits, tagged with "`(Lo)`". See Section 4.3 for a complete description of the argument formats.

Unless otherwise stated, all count and address arguments are 16-bit references, *not* byte references, since the DSP only addresses DRAM by 16-bit words. Addresses are specified as the offset from the base of the board's DRAM. For example, the address of the first DRAM word on the board is `$000000`, and the next 16-bit word is at address `$000001`.

Argument values that are set or modified by the DSP are marked with a "⇒" symbol (see Section 4.3.4).

14.2.5 Description

This text clarifies the function of the command, and the role of each argument. Any unique restrictions or special functions are described here.

14.2.6 Data Format

This heading is used for commands that operate on one or more data buffers in DRAM. The format and order of the data is described here.

14.2.7 Ranges

Every argument has a limited range of permitted values. While some commands will check critical arguments for correct range, the host should make every attempt to stay within the specified ranges, as described in Chapter 11. The range and type for each argument is shown in a table with the command description.

Since some ranges depend on board-specific hardware limits, they will vary between different implementations of DiSPATCH. These range limits will be represented by name rather than value, as described below.

Max DRAM

This is the size of the on-board shared DRAM, in 16-bit words. Different MMI models and configurations are equipped with various amounts of on-board DRAM. Maximum values for current Vigra MMI models are listed below.

Model	**RAM Size**	Max DRAM
MMI-105-1	1 Meg	$0007FE00
MMI-105-4	4 Meg	$001FFE00
MMI-105-8	8 Meg	$003FFE00
MMI-210-1	1 Meg	$0007FF00
MMI-210-4	4 Meg	$001FFF00
MMI-420	4 Meg	$001FFF00
MMI-4210	4 Meg	$001FFF00
MMI-4211	4 Meg	$001FFF00

All data buffers must fit entirely within the available DRAM. This means that a buffer's start address added to its size must always be less than Max DRAM.

X RAM Size, Y RAM Size, P RAM Size

These refer to the size of private DSP static RAM for a particular memory space (X, Y or P). This is determined by the size, number, and partitioning of SRAM chips installed on the MMI board. Consult the table below for the limits applicable to your board.

Model	X & Y RAM Size	P RAM Size
MMI-105	$4000	$8000
MMI-210	$4200	$8000
MMI-420	$2200	$4200
MMI-4210	$2200	$4200

14.2.8 Error Codes

Each command has a set of possible error conditions and error codes. The error conditions checked for by each command are listed in the command descriptions. A command will never detect an error condition other than those listed. See Chapter 15 for a complete listing of DiSPATCH error codes.

14.2.9 See Also

Where applicable, the reader is directed to related commands or other commands that affect the operation of this command.

14.3 Command Summary

Command	Command ID	Page	Section
ABORT_ALL_PLAY	$0028	84	14.4.14
ABORT_MONITOR	$002B	135	14.10.3
ABORT_RECORD	$001B	95	14.5.9
ABORT_TRACK	$0027	83	14.4.13
ANALOG_TEST	$0030	129	14.9.5
CLIP_LED	$004B	101	14.5.14
CONFIGURATION	$0011	147	14.12.4
COUNT_BUFFERS	$002E	143	14.11.5
DISABLE_MAIL	$001A	152	14.12.8
DISABLE_MEASURE	$0046	97	14.5.11
ENABLE_MAIL	$0010	151	14.12.7
ENABLE_MEASURE	$0045	96	14.5.10
END_COUNTER	$0013	75	14.4.6
END_LOOPBACK	$0096	145	14.12.2
EPROM_CHECKSUM	$0039	131	14.9.6
EQUALIZER	$0031	148	14.12.5
FETCH_ERROR	$000A	125	14.9.1
HAMMER	$0038	132	14.9.7
INPUT_GAIN	$0015	114	14.7.3
INPUT_BIAS	$0044	99	14.5.13
INPUT_PEAK	$0043	98	14.5.12
LED_CTRL	$000D	139	14.11.2
LOAD_P	$0003	124	14.8.3
LOAD_TABLE	$003A	108	14.6.3
LOAD_X	$0001	121	14.8.1
LOAD_Y	$0002	123	14.8.2
LOOPBACK	$0069	144	14.12.1
LR_INPUT_GAIN	$003E	115	14.7.4
LR_OUTPUT_GAIN	$003D	117	14.7.6
LSHIFT	$000B	128	14.9.4
MONITOR_BUFFER	$002A	133	14.10.1
MONITOR_POS	$002C	134	14.10.2
NOP	$0026	142	14.11.4
OUTPUT_GAIN	$0016	116	14.7.5
PAUSE_PLAY	$000E	71	14.4.3
PAUSE_RECORD	$0020	86	14.5.2

Command	Command ID	Page	Section
PLAY_BUFFER	$0006	67	14.4.1
PLAY_POS	$001D	82	14.4.12
PLAY_SUBBUFS	$001E	69	14.4.2
QUERY_LOAD	$0029	127	14.9.3
QUERY_STATE	$0032	146	14.12.3
RAMTONE	$0047	106	14.6.2
RECORD_BUFFER	$0019	85	14.5.1
RECORD_POS	$001C	94	14.5.8
REGISTER_COUNTER	$0012	73	14.4.5
RESAMPLE	$004A	80	14.4.11
RESET_COUNTER	$0034	76	14.4.7
RESUME_PLAY	$000F	72	14.4.4
RESUME_RECORD	$0021	87	14.5.3
REVERB	$0033	150	14.12.6
ROUTE_OUTPUT	$003C	118	14.7.7
SELECT_INPUT	$003F	119	14.7.8
SET_MONITOR_FORMAT	$002D	136	14.10.4
SET_PLAY_FORMAT	$0018	78	14.4.9
SET_PLAY_GAIN	$0008	77	14.4.8
SET_PNM	$0014	112	14.7.2
SET_RECORD_FORMAT	$0022	90	14.5.6
SET_RECORD_GAIN	$0023	88	14.5.4
SET_SIDETONE	$0024	89	14.5.5
SET_SRATE	$0009	110	14.7.1
SIGNAL_DETECT	$002F	91	14.5.7
SPEED_CHANGE	$0025	79	14.4.10
STEREO_MODE	$0040	120	14.7.9
TEST_DRAM	$0037	126	14.9.2
TONEGEN	$0036	103	14.6.1
TRANSFORM	$001F	137	14.11.1
VERSION	$000C	141	14.11.3

14.4 Playback

14.4.1 PLAY_BUFFER

Summary: Initiate audio playback from DRAM, or queue a play request if one is already in progress.

Code: `$0006`

Arguments:

Count (Hi) Count (Lo)	Number of words to play from DRAM.
Address (Hi) Address (Lo)	DRAM source address for data words.
Track	Audio playback track to use.

Description: This command is used to queue audio playback requests on the MMI. Audio samples are read from DRAM, processed by the DSP, and sent to the DAC on the audio board. The playback function of DiSPATCH is described in Section 2.1.

The DRAM source address is the word offset from the base of the board, with `$000000` being the first playable word. The Address and Count are always in terms of 16-bit words, *not bytes*.

Data Format: The format of the samples expected by the DSP depends entirely on the data format currently selected for the specified Track. For each track, the host can select an independent data format for playback by using the SET_PLAY_FORMAT command. Regardless of the format, the host must always specify the Count in terms of 16-bit words.

Ranges:

Argument	Type	Min	Max
Count	Extended	0	Max DRAM
Address	Extended	0	Max DRAM
Track	Word	0	24

Error Codes: Invalid Track Number

See Also: PLAY_SUBBUFS, § 14.4.2.
SET_PLAY_FORMAT, § 14.4.9.
SET_PLAY_GAIN, § 14.4.8.

14.4.2 PLAY_SUBBUFS

Summary: Mix and play several sub-buffers in one region of DRAM.

Code: `$001E`

Arguments:

Count (Hi) Count (Lo)	Number of words to play from DRAM.
Address (Hi) Address (Lo)	DRAM starting address for first sub-buffer.
Track	First audio playback track to use.
Subbuffers	Number of sub-buffers to mix.
Spacing (Hi) Spacing (Lo)	Offset between first sample of each sub-buffer.

Description: This command is similar to PLAY_BUFFER, but treats the given DRAM buffer as a set of sub-buffers to be mixed and played together. Each sub-buffer contains Count samples, and begins Spacing words from the previous sub-buffer.

Each sub-buffer is played on a separate track, beginning with the track named in the Track argument. Executing this command is very similar to executing a PLAY_BUFFER command on each track numbered Track through (Track + Subbuffers). The start address for each sub-buffer is computed as:

$$\text{Address} + (\text{Subbuffer\#} * \text{Spacing})$$

When all sub-buffers are done playing, the host will receive a single Buffer Completion message from the DSP, as explained in Section 5.2.

Data Format: All the tracks to be used for sub-buffer mixing *must* use the same audio data format. The host must select the audio format for each track prior to issuing the PLAY_SUBBUFS command.

Ranges:

Argument	Type	Min	Max
Count	Extended	0	Max DRAM
Address	Extended	0	Max DRAM
Track	Word	0	24
Subbuffers	Word	1	24
Spacing	Extended	1	Max DRAM

Error Codes: Invalid Track Number

See Also: PLAY_BUFFER, § 14.4.1.
SET_PLAY_FORMAT, § 14.4.9.
SET_PLAY_GAIN, § 14.4.8.

14.4.3 PAUSE_PLAY

Summary: Pause all audio playback.

Code: `$000E`

Arguments: None.

Description: This command immediately pauses all audio playback. If a Play Counter is running, it will stop incrementing while playback is paused. This command is safely ignored if playback is already paused.

Error Codes: None.

See Also: RESUME_PLAY, § 14.4.4.

14.4.4 RESUME_PLAY

Summary: Resumes paused audio playback.

Code: $000F

Arguments: None.

Description: This resumes audio playback, if it was paused with PAUSE_PLAY. The Play Counter will also resume counting. This command is safely ignored if playback is not paused.

Error Codes: None.

See Also: PAUSE_PLAY, § 14.4.3.

14.4.5 REGISTER_COUNTER

Summary: Register a 31-bit Playback sample counter

Code: $0012

Arguments:

⇒ Counter (Hi 16)

⇒ Counter (Lo 15) Locations in which to store the counter.

Description: This command provides a way for the host to track the progress of playback in real-time. After executing the REGISTER_COUNTER command, the DSP will continuously update the Counter until an END_COUNTER command is executed.

The 31-bit Counter is incremented once per outgoing sample (at exactly the current sample rate). This Counter may be used to approximate the DRAM position of the next outgoing sample, or count how many samples have been played.

This counter is updated at the last stage of processing, and is exactly in sync with the outgoing audio samples. The host should take into account any processing performed by the DSP when using the Counter to estimate the present DRAM source address.

See Section 7.5 for important details on reading and verifying the counter value.

Data Format: The Counter argument is split across two DRAM words in a format different than most Extended Word (24-bit) arguments. This is done to increase the range and significantly improve DSP performance when updating the counter.

The first argument represents the upper 16 bits of the counter value, and the second argument represents the lower 15 bits. Together, the counter represents a 31-bit value, which will wrap back to zero after 2,147,483,647 samples (12.42 hours at 48 KHz).

Notes:

1. DSP processing is internally buffered at many levels, so samples will be read from DRAM somewhat sooner than they are played.

2. Since the play counter puts an additional burden on the DSP, it is recommended that it be disabled by use of the END_COUNTER command when not in use. By default, there is no Play Counter.
3. The Counter value is only reset by use of the RESET_COUNTER command.
4. The play counter does not stop incrementing when all playback requests have completed, but will continue to increment indefinitely.
5. Any RAM testing procedures will certainly fail if a Counter is running, because the counter continuously modifies the two Counter locations in DRAM.
6. The play counter is not incremented while playback is paused by the PAUSE_PLAY command.

Ranges:

Argument	Type	Min	Max
Count-Hi	Word	0	$7FFF
Count-Low	Word	0	$7FFF

Error Codes: None.

See Also: END_COUNTER, § 14.4.6.
RESET_COUNTER, § 14.4.7.

14.4.6 END_COUNTER

Summary: Disable the active Play Counter, if any.

Code: $0013

Arguments: None.

Description: This is used to terminate the Play Counter. It should be used if the Counter is not needed, since it relieves the DSP of the need to update it.

If this command is executed without a play counter running, it will be safely ignored.

Error Codes: None.

See Also: REGISTER_COUNTER, § 14.4.5.
RESET_COUNTER, § 14.4.7.

14.4.7 RESET_COUNTER

Summary: Reset the value of the Play Counter to zero.

Code: $0034

Arguments: None.

Description: The play counter increments every sample, and wraps around to zero after 31-bits. By using this command, the host can reset the play counter value to zero immediately. This command is safe to use even while playback is in progress and the play counter is running.

If this command is executed while the play counter is not enabled, it will be ignored by DiSPATCH.

Error Codes: None.

See Also: REGISTER_COUNTER, § 14.4.5.
END_COUNTER, § 14.4.6.

14.4.8 SET_PLAY_GAIN

Summary: Set the digital playback gain factor for one track.

Code: $0008

Arguments:

Track	The track to change, or $FFFF for Master.
Left Gain (Hi) Left Gain (Lo)	Left playback gain level for this track.
Right Gain (Hi) Right Gain (Lo)	Right playback gain level for this track.

Description: This command immediately changes the digital playback gain (volume). Each track has an independent gain setting, and all tracks are scaled by the Master Gain.

To set the gain level for one track, the track number is used as the Track argument. To set the Master Play Gain for all tracks, use the special Track value $FFFF.

The values of the Left and Right Gain arguments are described in Section 6.2. They range from silence (Gain = -1.0) to times-two amplification (Gain = 1.0). The default Gain setting for all tracks is 0.0, for unity gain.

This command may be issued at any time (before, during, or after playback), and will take effect immediately upon execution.

Notes: Monophonic MMI boards that do not support stereo playback must always set both the Left Gain and Right Gain arguments to the same value.

Ranges:

Argument	Type	Min	Max
Track	Word	0	24, $FFFF
Left Gain	Fraction	-1.0	0.999999881
Right Gain	Fraction	-1.0	0.999999881

Error Codes: Invalid Track Number

See Also: OUTPUT_GAIN, § 14.7.5.
SET_RECORD_GAIN, § 14.5.4.

14.4.9 SET_PLAY_FORMAT

Summary: Set audio data format for playback.

Code: $0018

Arguments:

Track	Track to change.
Format	Data format to use on this track.

Description: Changes the playback data format for the specified Track to be Format. The Format argument must be one of the audio data formats listed below. The default data format for all tracks is PCM-16. See Chapter 10 for a description of the available audio data formats.

Encoding	Format Code
PCM-16	$0016
PCM-8	$0008
μ-Law	$0006
A-Law	$000A
ADPCM	$000D
VQ	$0002

Figure 14.1: Playback Data Formats

Ranges:

Argument	Type	Min	Max
Track	Word	0	24
Format	Word	see Table 14.1	

Error Codes: Invalid Format Code

See Also: PLAY_BUFFER, § 14.4.1.
PLAY_SUBBUFS, § 14.4.2.
SET_MONITOR_FORMAT, § 14.10.4.
SET_RECORD_FORMAT, § 14.5.6.

14.4.10 SPEED_CHANGE

Summary: Change the pitch-corrected speed of playback.

Code: `$0025`

Arguments:

Speed	New speed factor to use.

Description: DiSPATCH supports a pitch-corrected speech rate-change algorithm that can be engaged during playback. This routine attempts to change the speed of the audio without significantly altering the pitch. However, note that it does add noise and other artifacts to the output signal. It is best suited to medium-amplitude voice recordings.

The time change is controlled by the setting of the Speed argument. Allowed values range from 128 to 384. The special Speed value 256 sets the playback speed to normal, and bypasses speed-change processing.

The approximate speed (in percent) is related to the Speed argument as follows:

$$\text{Play Speed\%} = \frac{\text{Speed} * 100}{256}$$

$$\text{Speed} = \frac{\text{Play Speed\%} * 256}{100}$$

The lowest speed possible is one half (50%) of normal speed (i.e. buffers take twice as long to play). This corresponds with the Speed value 128. The highest speed possible is twice normal speed (200%), represented by the Speed value 384.

Ranges:

Argument	Type	Min	Max
Speed	Word	128	384

Error Codes: Time Change Out Of Range

14.4.11 RESAMPLE

Summary: Scale the effective sample rate by resampling the audio data.

Code: $004A

Arguments:

Multiplier	Resampling factor (1 or 4).

Description: DiSPATCH provides a software multi-rate filter to lower the effective audio sample rate for playback and recording. By using this filter, it is possible to have sample rates that are exactly one quarter of the hardware settings.

This feature is only supported on the MMI-4210 and MMI-4211 audio board models. Since these boards have a minimum hardware sample rate of 6000 Hz, it is possible to achieve effective sample rates as low as 1500 Hz through the application of this resampling feature. Of course, signal bandwidth is degraded proportionally at lower sample rates.

To divide the effective sample rate by a factor of four, execute the RESAMPLE command using a Multiplier value of four (4). To disable the multirate filter and return to normal (hardware) sample rates, issue the command with a Multiplier value of one (1).

Use of the resampling feature is *not permitted* with hardware sample rates higher than 24000 Hz. For higher rates, the hardware should be set directly to the required sample rate and the resampling filter should be disengaged.

The hardware sample rate may be adjusted with the SET_SRATE command even while the resampling filter is in use. When the filter is in effect, both the playback and recorded audio are resampled to simulate the lower sample rate. This will cause the playback and record counters to run at one quarter normal speed, while buffers take four times as long to play or record.

The use of sidetone (see Section 14.5.5) is not permitted while the resampling filter is in use. Disable sidetone before engaging the multi-rate filter, or distorted audio may result.

Ranges:

Argument	Type	Min	Max
Multiplier	Word	1	4

Error Codes: Parameter Out of Range Unsupported Command (MMI-210, MMI-105)

See Also: SET_SRATE, § 14.7.1.

14.4.12 PLAY_POS

Summary: Check current playback position in DRAM

Code: $001D

Arguments:

⇒ Track	Which track to report on.
⇒ Count (Hi) ⇒ Count (Lo)	Words remaining in the present play request.
⇒ Address (Hi) ⇒ Address (Lo)	DRAM address of next sample to play.

Description: While playback is in progress, the host can use this command to check the current position within the buffer that is playing. Section 5.7.2 gives additional information about reading the position of an active buffer.

Since each track can play a different buffer, the host must use the Track argument to specify which track is of interest. The position within the active buffer on this track will be returned.

Ranges:

Argument	Type	Min	Max
Track	Word	0	24
Count	Extended	0	Max DRAM
Address	Extended	0	Max DRAM

Error Codes: Invalid Track Number

See Also: PLAY_BUFFER, § 14.4.1.
PAUSE_PLAY, § 14.4.3.
RECORD_POS, § 14.5.8.

14.4.13 ABORT_TRACK

Summary: Abort all active and pending playback requests on one track.

Code: $0027

Arguments:

Track	Track number to clear requests from.

Description: While playback is in progress, the host may request that all pending requests for one track be discarded, and playback aborted. This results in an immediate end of audio output from this track. Playback on tracks other than the one specified are unaffected.

When playback is aborted, the host will not receive Buffer Completion messages from those requests that were discarded or not finished.

The host is allowed to abort playback at any time, even when there is no playback in progress or playback is paused.

Ranges:

Argument	Type	Min	Max
Track	Word	0	24

Error Codes: Invalid Track Number

See Also: ABORT_ALL_PLAY, § 14.4.14.
PLAY_BUFFER, § 14.4.1.
PAUSE_PLAY, § 14.4.3.

14.4.14 ABORT_ALL_PLAY

Summary: Abort all active and pending playback requests on all tracks.

Code: `$0028`

Arguments: None.

Description: While playback is in progress, the host may request that all pending requests be discarded, and playback aborted. This results in an immediate end of all audio output (except sidetone).

This command is equivalent to executing an ABORT_TRACK command on all available tracks.

When playback is aborted, the host will not receive Buffer Completion messages from those requests that were discarded or not finished. The host is allowed to abort playback at any time, even when there is no playback in progress or playback is paused.

Error Codes: None.

See Also: ABORT_TRACK, § 14.4.13.
PLAY_BUFFER, § 14.4.1.
PAUSE_PLAY, § 14.4.3.

14.5 Recording

14.5.1 RECORD_BUFFER

Summary: Initiate audio recording to DRAM, or queue a record request if one is in progress.

Code: $0019

Arguments:

Count (Hi) Count (Lo)	Number of words to record to DRAM.
Address (Hi) Address (Lo)	DRAM destination address for data words.

Description: This command is used to queue audio record requests. Audio samples are taken from the ADC, processed by the DSP, and the results written to DRAM.

Ranges:

Argument	Type	Min	Max
Count	Extended	0	Max DRAM
Address	Extended	0	Max DRAM

Error Codes: None.

See Also: PAUSE_RECORD, § 14.5.2.
SET_RECORD_GAIN, § 14.5.4.

14.5.2 PAUSE_RECORD

Summary: Pause all audio recording.

Code: $0020

Arguments: None.

Description: This immediately pauses all audio recording. The command is safely ignored if recording is already paused.

Error Codes: None.

See Also: RESUME_RECORD, § 14.5.3.

14.5.3 RESUME_RECORD

Summary: Resumes paused audio recording.

Code: $0021

Arguments: None.

Description: Resumes any audio recording if paused. This command is safely ignored if recording is not paused.

Error Codes: None.

See Also: PAUSE_RECORD, § 14.5.2.

14.5.4 SET_RECORD_GAIN

Summary: Set the digital recording gain factor.

Code: $0023

Arguments:

Left Gain (Hi) Left Gain (Lo)	Left digital recording gain level.
Right Gain (Hi) Right Gain (Lo)	Right digital recording gain level.

Description: This command immediately changes the digital recording gain for incoming audio. Using a maximum value (1.0) for the Gain arguments results in a digital gain of two. A gain value of -1.0 will attenuate entirely, recording only silence. The default recording gain is 0.0, representing unity gain.

This command may be issued at any time (before, during, or after recording), and will take effect immediately upon execution.

Notes: Monophonic MMI boards that do not support stereo recording from one DSP should always set both the Left and Right Gain arguments to the same value.

Ranges:

Argument	Type	Min	Max
Left Gain	Fraction	-1.0	0.999999881
Right Gain	Fraction	-1.0	0.999999881

Error Codes: None.

See Also: INPUT_GAIN, § 14.7.3.
SET_PLAY_GAIN, § 14.4.8.

14.5.5 SET_SIDETONE

Summary: Set recording audio sidetone level.

Code: $0024

Arguments:

Gain (Hi) Gain (Lo)	Digital gain level for audio side-tone.

Description: This sets the gain level of the sidetone signal. Sidetone is a copy of the incoming signal that is digitally copied to the output channel, as described in Section 8.1. Sidetone audio is unbuffered, resulting in zero delay between input and output.

A unity-gain sidetone setting would pass all incoming audio immediately to the output channel. This is well suited for monitoring the exact level of the recorded signal, but may generate feedback in a speaker-microphone setup. Usually, a small amount of sidetone is desirable for voice recording, similar to that provided by a telephone.

Sidetone can be turned off entirely by setting the Gain argument to -1.0. Settings greater than 0.0 will provide an amplified sidetone signal (i.e., louder than the incoming signal). See the discussion of gain values in Section 6.2.

Since the sidetone operation requires some processing time, it should be turned off when not needed. This frees up the DSP to be used for other processing.

Ranges:

Argument	Type	Min	Max
Gain	Fraction	-1.0	0.999999881

Error Codes: None.

14.5.6 SET_RECORD_FORMAT

Summary: Set audio data format for recording.

Code: $0022

Arguments:

Format	Data format to use during recording.

Description: Changes the audio data format for recorded data to be Format. The Format argument must be one of the audio data formats listed below. The default data format for recording is PCM-16. See Chapter 10 for a description of the available audio data formats.

Encoding	Format Code
PCM-16	$0016
PCM-8	$0008
μ-Law	$0006
A-Law	$000A
ADPCM	$000D
VQ	$0002

Figure 14.2: Recording Data Formats

Ranges:

Argument	Type	Min	Max
Format	Word	see Table 14.2	

Error Codes: Invalid Format Code

See Also: RECORD_BUFFER, § 14.5.1.
SET_PLAY_FORMAT, § 14.4.9.
SET_MONITOR_FORMAT, § 14.10.4.

14.5.7 SIGNAL_DETECT

Summary: Enable, disable, or adjust a signal energy detector.

Code: $002F

Arguments:

Detector	Which detector to configure.
Voice (Hi) Voice (Lo)	Voice-band energy threshold.
Energy (Hi) Energy (Lo)	Raw signal energy threshold.
Up Notify	Number of energy periods to trigger an UP transition.
Down Notify	Number of energy periods to trigger a DOWN transition.
⇒ State	The present state for this detector.
⇒ Address (Hi) ⇒ Address (Lo)	The DRAM buffer address of the last transition.

Description: This command configures one of the available DiSPATCH signal detectors. By adjusting the parameters, the host can select and tune the criteria for signal detection and reporting. The operation of the signal detectors is described in detail in Section 8.2.

The Detector argument specifies which of the available signal detectors is to be configured. Setting the same detector twice will overwrite the previous settings, and can be done at any time.

The Voice parameter determines how much of the incoming signal must be within the speech spectrum. A value of 1.0 means that all of the signal must be completely within the speech spectrum. This is rarely the case, so the detector will probably never trigger an UP transition. Setting the Voice parameter to 0.0 specifies none of the signal needs to lie in the speech band of the audio spectrum. This effectively makes the Energy setting the only criteria for signal detecting, since the Voice threshold will always be met.

The Energy parameter determines the threshold for the raw signal energy. If this value is set to 0.5, the detector will only trigger when the average RMS energy of the signal is 50% of full-scale, which

is a rather high setting. A reasonable setting for Energy is about 5–15% of full scale.

Negative values for Voice and Energy are not allowed and will trigger a "Parameter Out of Range" error.

After the specified number of periods has passed with the signal "present," DiSPATCH notifies the host that an "UP" transition has occurred. Then, after the specified number of periods passes with no signal present, a "DOWN" transition message is sent. When each transition message is sent to the host, the State argument is set to 1 or 0 by the DSP to indicate an UP or DOWN state, respectively.

If recording is in progress, the Address argument is changed to reflect the DRAM address of the transition. This tells the host approximately where in the buffer the UP or DOWN transition took place. The Address is undefined if no record is in progress.

To filter out spurious transitions, a certain amount of time must go by with the detector sensing a new state. These lengths of time are controlled by the Up Notify and Down Notify arguments. These are expressed in units of "blocks," where each block is 20 milliseconds (160 samples). A value of 0 blocks will set the detector to trigger immediately, while a value of 100 will require about two seconds of the new state before the host is notified.

To disable a signal detector currently in use, set both the Energy and Voice parameters to zero. Signal detectors disabled in this way will not consume DSP processing bandwidth.

Notes: At a sample rate of 8000 Hz, the signal detectors are most sensitive to signals between 200 and 900 Hz, which encompasses most human speech. As explained in Section 8.2, signal detect is *not supported* at sample rates other than 8000 Hz.

Ranges:

Argument	Type	Min	Max
Detector	Word	0	3
Voice	Fraction	0.0	0.999999881
Energy	Fraction	0.0	0.999999881
Up Notify	Word	0	65535
Down Notify	Word	0	65535
State	Word	0	1
Address	Extended	0	Max DRAM

Error Codes: Invalid Detector Number
Parameter Out of Range

14.5.8 RECORD_POS

Summary: Check current record position in DRAM

Code: $001C

Arguments:

⇒ Count (Hi)
⇒ Count (Lo) Words remaining in the present record request.

⇒ Address (Hi)
⇒ Address (Lo) Address where next recorded sample will go.

Description: This command reveals how much is remaining of the current record request in progress, and where in DRAM the next incoming word will be placed. These values are valid *only* when recording is in progress. If there is no record request being processed, then the values are meaningless.

Since recording is a continuous operation, the record position is constantly changing. The returned values are therefore only accurate at the instant they are returned. See Section 5.7.2 for information on reading the buffer position.

Ranges:

Argument	Type	Min	Max
Count	Extended	0	Max DRAM
Address	Extended	0	Max DRAM

Error Codes: None.

See Also: RECORD_BUFFER, § 14.5.1.
PAUSE_RECORD, § 14.5.2.
PLAY_POS, § 14.4.12.
MONITOR_POS, § 14.10.2.

14.5.9 ABORT_RECORD

Summary: Abort all active and pending record requests.

Code: $001B

Arguments: None.

Description: While recording is in progress, the host may request that all pending requests be discarded, and recording aborted. This will result in an immediate termination of all recording.

After this command is acknowledged, the host will not receive Buffer Completion messages from those requests that were discarded or not finished. The host is allowed to abort recording at any time, even when there is no recording in progress or recording is paused.

To determine the position of the active buffer at the time it was aborted, the host may follow a procedure similar to that outlined in Section 5.7.2.

Error Codes: None.

See Also: RECORD_BUFFER, § 14.5.1.
PAUSE_RECORD, § 14.5.2.
ABORT_ALL_PLAY, § 14.4.14.

14.5.10 ENABLE_MEASURE

Summary: Begin taking input signal measurements.

Code: $0045

Arguments: None.

Description: The DSP can optionally make measurements of the input signal to determine the peak input level and the amount of DC bias. Because these measurement computations consume a small but constant amount of bandwidth, they are **disabled** by default and must be turned on with this command.

Before the INPUT_PEAK or INPUT_BIAS commands can be used, the host must enable measurements by issuing ENABLE_MEASURE. When input measurements are no longer used, they should be disabled (with DISABLE_MEASURE) for efficiency.

Note that DiSPATCH requires approximately 150 audio samples after the ENABLE_MEASURE command before valid measurement values are available for reading. If measurement values are requested immediately after enabling measurements, then the returned values may be invalid.

All measurements are frozen while recording is paused.

Error Codes: None.

See Also: DISABLE_MEASURE, § 14.5.11.
INPUT_PEAK, § 14.5.12.
INPUT_BIAS, § 14.5.13.

14.5.11 DISABLE_MEASURE

Summary: Stop taking input signal measurements.

Code: $0046

Arguments: None.

Description: When the host has enabled input signal measurements by executing ENABLE_MEASURE, the DSP will spend a small amount of time computing peak and bias values for each incoming sample. To maximize the amount of DSP computing bandwidth for other tasks, these measurements should be disabled when they are not needed.

Error Codes: None.

See Also: ENABLE_MEASURE, § 14.5.10.
INPUT_PEAK, § 14.5.12.
INPUT_BIAS, § 14.5.13.

14.5.12 INPUT_PEAK

Summary: Request the peak input level since last reported.

Code: $0043

Arguments:

⇒ Left	Peak value for the left channel.
⇒ Right	Peak value for the right channel.

Description: This command can be used to request the peak input signal level, but only after input signal measurements have been enabled by executing the ENABLE_MEASURE command. The returned Left and Right values are the magnitude of the highest amplitude sample since the last execution of INPUT_PEAK.

This measurement is useful for displaying a real-time VU meter or warning the user of potential clipping. When stereo mode is not enabled, the left and right values will both reflect the peak value of the single monophonic channel.

The peak value is updated both when DiSPATCH is recording and when it on standby. However, the peak value is **not** updated while recording is paused via the PAUSE_RECORD command.

The reported peak values range from $0000 (absolute silence) to $FFFF (fully saturated). The reported peak value is always the highest signal value since INPUT_PEAK was last called. This allows the host to make either slow or frequent peak readings without missing high-amplitude transients.

Ranges:

Argument	Type	Min	Max
Left	Word	0	$FFFF
Right	Word	0	$FFFF

Error Codes: Feature Not Enabled

See Also: ENABLE_MEASURE, § 14.5.10.
DISABLE_MEASURE, § 14.5.11.
INPUT_BIAS, § 14.5.13.
INPUT_GAIN, § 14.7.3.
CLIP_LED, § 14.5.14.

14.5.13 INPUT_BIAS

Summary: Report the DC input bias level.

Code: $0044

Arguments:

⇒ Left	Bias value for the left channel.
⇒ Right	Bias value for the right channel.
⇒ Samples	Number of samples used in bias computation.

Description: After input signal measurements have been enabled by calling ENABLE_MEASURE, this command can be used to request an approximation of the DC input bias, as seen by the analog to digital converter. The returned value is the arithmetic mean value of a large number of sequential input samples.

For audio applications, the DC input bias should be close to zero. Hardware design factors often impart a small DC offset to the signal. If this offset becomes significant, then the maximum dynamic range of the input digitizer is compromised.

For some applications, Vigra has specially modified certain audio boards to defeat the normal AC input signal coupling and instead digitize the actual input voltage level. In these applications, the INPUT_BIAS command can be used to determine the current signal voltage. Special care must be taken to stay within the hardware limits; excessive input voltage can damage the sensitive input circuitry.

The reported bias value is simply the sum of a number of recently digitized samples. The number of input samples used in the bias computation is reported via Samples.

When stereo mode is not enabled, the left and right values will both reflect the bias level of the single monophonic channel.

Ranges:

Argument	Type	Min	Max
Left	Word	0	$FFFF
Right	Word	0	$FFFF
Samples	Word	0	$FFFF

Error Codes: Feature Not Enabled

See Also: ENABLE_MEASURE, § 14.5.10.
DISABLE_MEASURE, § 14.5.11.
INPUT_PEAK, § 14.5.12.

14.5.14 CLIP_LED

Summary: Use front-panel LED to indicate peak input level.

Code: $004B

Arguments:

Ceiling	Peak sample value.

Description: For maximum resolution during recording, the analog input signal should make maximum use of the available dynamic range. To help adjust the input level, the application may enable the Clipping LED feature.

The Ceiling is a 16-bit sample value selected by the application to represent a "full-strength" input signal. When the amplitude of the recorded signal exceeds this sample value, the front-panel LED will light briefly to indicate potential clipping. When the signal strength returns to a value less than the Ceiling, the LED will turn off.

By selecting appropriate Ceiling values, the LED may be used to indicate a clipped and distorted signal (full-scale), or simply to show an appropriate target signal level (less than full-scale). A very small Ceiling value may be used to indicate that any recording signal is present and active.

While the CLIP_LED feature is in use, the LED must not be manipulated by the normal LED_CTRL command. To disable the Clipping LED feature and return the LED to normal use, execute the CLIP_LED command with a Ceiling value of zero.

The Clipping LED is active even when audio is not being recorded, but it is temporarily disabled any time recording is paused with PAUSE_RECORD. On the MMI-210 model, only one DSP may use the Clipping LED feature at any given time.

Ranges:

Argument	Type	Min	Max
Ceiling	Word	0	32767

Error Codes: Parameter Out of Range

See Also: INPUT_PEAK, § 14.5.12.
INPUT_GAIN, § 14.7.3.
LED_CTRL, § 14.11.2.

14.6 Tone Generation

The commands described in this section are used to synthesize audio tones and indicators. The function and operation of tone generation are documented in the *DiSPATCH Tone Generation Manual*.

14.6.1 TONEGEN

Summary: Set synthesis parameters for tone generation.

Code: $0036

Arguments:

Track	Track number to change.
Length (Hi) Length (Lo)	Length of the tone, in samples.
Step Int	Integer part of Wave Step.
Step Frac (Hi) Step Frac (Lo)	Fractional part of Wave Step.
Wave Table	Wave table for main oscillator.
AM Enable	Enable/Disable Amplitude Modulation.
AM Step Integer	Integer part of AM Step.
AM Step Frac (Hi) AM Step Frac (Lo)	Fractional part of AM Step.
AM Table	Wave table for AM oscillator.
AM Start Position	Table start position for AM.
Envelope Enable	Enable/Disable amplitude envelope.
Attack Incr (Hi) Attack Incr (Lo)	Attack rate control.
Sustain Len (Hi) Sustain Len (Lo)	Length of sustain, in samples.
Release Dec (Hi) Release Dec (Lo)	Decay rate control.
FM Enable	Enable/Disable Frequency Modulation.
FM Step Integer	Integer part of FM Step.
FM Step Frac (Hi)	

FM Step Frac (Lo)	Fractional part of FM Step.
FM Table	Wave table for FM oscillator.
FM Start Position	Table start position for FM.
Delta Integer	Integer part of FM Delta.
Delta Frac (Hi) Delta Frac (Lo)	Fractional part of FM Delta.

Description: This command immediately changes the tone generation parameters for the specified track. Any tone that was previously playing is replaced by the new parameters.

The arguments and their application are described in detail in the *DiSPATCH Tone Generation Manual*. The TONEGEN command is usually accessed via the friendlier library functions that compute the argument values from more common units.

The wave frequency, AM frequency, FM frequency and FM delta are given to the DSP as "step rates". These rates specify how much to add to the table pointer after each sample. The table pointer wraps the the beginning of the wave table after reaching 256.

Each step value has two 24-bit parts: The integer portion and the fractional portion. This permits a full 24-bit fractional resolution for all frequencies. The step values can be computed as follows:

$$\text{Step} = \frac{256.0 * \text{Frequency}}{\text{Sample Rate}}$$

Ranges:

Argument	Type	Min	Max
Track	Word	0	24
Length*	Extended	0	$7FFFFF
Step Int	Word	0	127
Step Frac	Fraction	0.0	0.999999881
Wave Table	Word	0	5
AM Enable	Word	0	1
AM Step Integer	Word	0	127
AM Step Frac	Fraction	0.0	0.999999881
AM Table	Word	0	5
AM Start Position†	Word	0	255
Envelope Enable	Word	0	1
Attack Incr	Fraction	0.0	0.999999881
Sustain Len	Extended	0	$7FFFFF
Release Dec	Fraction	0.0	0.999999881
FM Enable	Word	0	1
FM Step Integer	Word	0	127
FM Step Frac	Fraction	0.0	0.999999881
FM Table	Word	0	5
FM Start Position†	Word	0	255
Delta Integer	Word	0	127
Delta Frac	Fraction	0.0	0.999999881

Error Codes: Invalid Track Number
Invalid Wavetable Number

See Also: LOAD_TABLE, § 14.6.3.
SET_PLAY_FORMAT, § 14.4.9.
SET_PLAY_GAIN, § 14.4.8.

*The special value $FFFFFF can be used to specify a tone of infinite length.

†The special value $FFFF will not change the table position, but keep the current pointer instead.

14.6.2 RAMTONE

Summary: Synthesize a tone and place the sample data in DRAM.

Code: $0047

Arguments:

Track	Track number for ToneGen parameters.
Format	Data format for output samples.
Address (Hi) Address (Lo)	DRAM destination address for sample data.
Count (Hi) Count (Lo)	Number of samples to synthesize into DRAM.
⇒ Wrote (Hi) ⇒ Wrote (Lo)	Number of sample words generated.

Description: The RAMTONE command is used to generate tones into DRAM rather than play them immediately with TONEGEN. This allows the host to retrieve a digital copy of a synthesized tone, or pre-generate several tones into DRAM and play them back-to-back using PLAY_BUFFER.

For a details concerning the advantages and restrictions of the RAMTONE command, please read the *DiSPATCH Tone Generation Manual*, specifically the "Generating Tones to DRAM" section.

The Track argument specifies the playback track which holds the desired tone generation parameters. The host must have previously configured this track by using the TONEGEN command. The host must ensure that playback is paused, or that this track is not set to the FORMAT_TONEGEN data format. This will prevent the tone from playing immediately after executing the TONEGEN command, and the parameters will remain unchanged until RAMTONE is executed.

The Format parameter selects one of the audio data formats for the resulting tone data. Any format (other than ToneGen) is permitted, but complex encoding formats require longer to generate.

The Address and Count specify the region of DRAM where the resulting audio samples will be written. Note that Count is expressed in **samples**, not words, therefore the Count does not change for different audio data formats.

The RAMTONE command will temporarily halt all DSP activity while processing the tone. When complete, the number of words generated will be placed into the Wrote argument.

The number of words generated is controlled by three factors:

1. The maximum Count of samples specified in the RAMTONE command
2. The remaining length of the tone, as specified by the Length argument to the TONEGEN command.
3. The requested audio data Format.

Error Codes: Invalid Track Number
Invalid Format Code

See Also: TONEGEN, § 14.6.1.

14.6.3 LOAD_TABLE

Summary: Load the contents of a wavetable from DRAM.

Code: $003A

Arguments:

Table	Table number of the wavetable to load into.
Address (Hi) Address (Lo)	Address of the input data in DRAM.

Description: This command replaces the contents of one wavetable with user data from DRAM. This command can be used at any time, even while the specified wavetable is in use.

Upon receiving the LOAD_TABLE command, the DSP will fetch 256 16-bit words from DRAM, starting at address offset Address and replace the contents of the specified Table with those samples.

The table size is fixed at 256 samples per table. Each 16-bit sample word is in the same format as PCM-16 data. The actual interpretation of the table contents depends on the Tone Generation application.

There is a total of six available wavetables. By default, the first five tables are initialized to common waveform periods, listed in Figure 14.3 and illustrated in the *DiSPATCH Tone Generation Manual*.

Table Number	Default Contents
0	Sine wave
1	Flat maximum (Unity)
2	Triangle wave
3	Square wave
4	Sawtooth wave
5	User defined

Figure 14.3: Tone Generation Tables

The contents of table number zero (Sine wave) are permanent and cannot be loaded with LOAD_TABLE. The contents of all other tables

can be replaced by the host at any time. The six data tables are shared by the AM, FM, and wave oscillators.

Ranges:

Argument	Type	Min	Max
Table	Word	0	5
Address	Extended	0	Max DRAM

Error Codes: Invalid Wavetable Number

See Also: TONEGEN, § 14.6.1.

14.7 Hardware Controls

14.7.1 SET_SRATE

Summary: Set the audio sample rate for recording and playback

Code: `$0009`

Arguments:

Play Rate (`Hi`) Play Rate (`Lo`)	Playback sampling rate in Hertz.
Rec Rate (`Hi`) Rec Rate (`Lo`)	Record sampling rate in Hertz.

Description: This command immediately changes the sampling frequencies used by the MMI audio converters (DAC and ADC). The frequency pair must be one that is supported by the MMI board in use, or an error condition will result.

MMI-210

The MMI-210 supports the following sample rate pairs for playback and recording:

Play Rate	Rec Rate
8000	8000
32000	8000
32000	16000
32000	32000
44100	44100
48000	12000
48000	24000
48000	48000
48000	96000

MMI-105

The MMI-105 supports the following sample rate pairs for playback and recording:

Play Rate	Rec Rate
5246	5246
8000	8000
13642	13642
19058	19058

MMI-4210

The MMI-4210 must always use the same sample rate for recording and playback (i.e., Play Rate and Rec Rate must be equal).

Starting with version 3.18 of the DiSPATCH firmware, the SET_SRATE command will accept any sample rate between **6,000** and **50,000** Hz (inclusive). This allows the continuous frequency range to be used directly (without use of the SET_PNM command).

The SET_SRATE command in DiSPATCH firmware versions prior to 3.18 accepts only a fixed set of common sample rates. The SET_PNM command can be used to produce other sample rates. For these firmware versions, the sample rates (in Hertz) accepted by the SET_SRATE command on the MMI-4210 are:

6000	16000	22050	40000
8000	18000	24000	44100
10000	20000	32000	48000
12000	22000	38000	50000

Ranges:

Argument	Type	Min	Max
Play Rate	Extended	see text	
Rec Rate	Extended	see text	

Error Codes: Invalid Sampling Frequency

See Also: SET_PNM, § 14.7.2.

14.7.2 SET_PNM

Summary: Set the Programmable Frequency Synthesizer (PFS) parameters.

Code: $0014

Arguments:

N Value	Frequency base ratio numerator.
M Value	Frequency base ratio denominator.
P Value	Post-divider value.

Description: This command directly sets the P, N, and M parameters on MMI boards equipped with a hardware PFS. This sets the sample rate by directly manipulating the frequency clock.

Starting with version 3.18 of the DiSPATCH firmware, this command is **obsolete** and should not be used. The simpler SET_SRATE command can now access the full range of available frequencies. Both the SET_PNM and SET_SRATE commands set the current sample rate, but SET_PNM requires that the host compute the frequency synthesizer parameters.

The PFS implements a phase-locked loop to generate the sampling rate based on the three parameters, P, N, and M. The frequency is computed as follows:

$$\text{Sample Rate} = \frac{\text{Clock} * N'}{2^{P'} * M' * \text{Divider}}$$

Sample Rate: The desired sampling frequency in Hertz.

Clock: The PFS input clock, in Hertz. This is factory set to 24,576,000 Hz.

Divider: The DAC bit clock divider. This is 384 for the MMI-4210 and 256 for the MMI-110.

P'**:** The post-divider. Permitted values are 3, 2, 1, and 0 only.

N'**:** The numerator. Permitted values are 1 through 16384.

M'**:** The denominator. Permitted values are 1 through 16384.

After selecting P', N', and M', the P, N, and M values for arguments are computed as follows:

$$\begin{aligned} \text{P} &= P' * 4 \\ \text{N} &= N' * 4 + 1 \\ \text{M} &= M' * 4 + 2 \end{aligned}$$

These P, N, and M values can then be used as the arguments to the SET_PNM command.

Ranges:

N: (Word) $n * 4 + 1$, where n is 1 through 16384.

M: (Word) $n * 4 + 2$, where n is 1 through 16384.

P: (Word) 0, 4, 8, or 12 only.

Error Codes: Unsupported Command (MMI-210, MMI-105)

See Also: SET_SRATE, § 14.7.1.

14.7.3 INPUT_GAIN

Summary: Set hardware analog input gain amplifier level.

Code: $0015

Arguments:

Gain	New input gain setting.

Description: This command sets the value of the analog input gain amplifier on MMI boards equipped with such hardware (such as the MMI-4210). This amplifier boosts or attenuates the input signal before it is digitized by the ADC. Proper setting of the input gain will allow for maximum resolution and minimum noise.

The actual value of the Gain argument depends on the analog hardware used. Each board is described below.

MMI-4210

The input amplifier on the MMI-4210 is capable of a gain from 1 to 1000. This is intended to accommodate input sources from magnetic microphones to line-level equipment.

Setting the Gain argument to 0 gives the lowest possible gain, suitable for line-level input. A Gain value of 255 is the highest value allowed, and yields an input gain of about 1000.

MMI-210 and MMI-105

This command is rejected by the MMI-210 and MMI-105, because they have no analog input gain control.

Error Codes: Unsupported Command (MMI-210, MMI-105)

See Also: SET_RECORD_GAIN, § 14.5.4.
OUTPUT_GAIN, § 14.7.5.

14.7.4 LR_INPUT_GAIN

Summary: Set stereo analog input gain amplifier level.

Code: $0015

Arguments:

Left Gain	New input gain setting for the left channel.
Right Gain	New input gain setting for the right channel.

Description: This command is similar to INPUT_GAIN, but is used to set gain levels for stereo inputs. This command is is not valid on hardware that does not support stereo input. Note that stereo boards may also use INPUT_GAIN to set both channels identically.

MMI-4211

Setting the Gain argument to 0 gives the lowest possible gain, suitable for line-level input. A Gain value of 255 is the highest value allowed.

MMI-4210

This is not a stereo-capable board, but this command will be accepted and processed as an INPUT_GAIN command. Only the Left value will be used.

MMI-210 and MMI-105

This command is rejected by the MMI-210 and MMI-105, because they have no analog input gain control.

Error Codes: Unsupported Command (MMI-210, MMI-105)

See Also: SET_RECORD_GAIN, § 14.5.4.
INPUT_GAIN, § 14.7.3.
LR_OUTPUT_GAIN, § 14.7.6.

14.7.5 OUTPUT_GAIN

Summary: Set hardware analog output gain amplifier level.

Code: $0016

Arguments:

Gain	New output gain setting.

Description: This command sets the value of the analog output gain amplifier on MMI boards equipped with such hardware (such as the MMI-4210). This amplifier boosts or attenuates the input signal before it is digitized by the ADC. Proper setting of that output gain will allow for maximum playback resolution and minimum noise.

The actual value of the Gain argument depends on the analog hardware used. Each board is described below.

MMI-4210

The output amplifier on the MMI-4210 is capable of a gain from 0.5 to 2.5.

Setting the Gain argument to 0 gives the lowest possible gain (0.5), while a Gain value of 255 yields the maximum available gain (2.5).

MMI-210 and MMI-105

This command is rejected by the MMI-210 and MMI-105, because they have no analog output gain control. However, there is a front-panel adjustment for the speaker output volume.

Error Codes: Unsupported Command (MMI-210, MMI-105)

See Also: SET_PLAY_GAIN, § 14.4.8.
INPUT_GAIN, § 14.7.3.

14.7.6 LR_OUTPUT_GAIN

Summary: Set stereo analog output gain amplifier level.

Code: $0015

Arguments:

Left Gain	New output gain setting for the left channel.
Right Gain	New output gain setting for the right channel.

Description: This command is similar to OUTPUT_GAIN, but is used to set gain levels for stereo outputs. This command is is not valid on hardware that does not support stereo output. Note that stereo boards may also use OUTPUT_GAIN to set both channels identically.

MMI-4211

Setting the Gain argument to 0 gives the lowest possible gain, suitable for line-level output. A Gain value of 255 is the highest value allowed.

MMI-4210

This is not a stereo-capable board, but this command will be accepted and processed as an OUTPUT_GAIN command. Only the Left value will be used.

MMI-210 and MMI-105

This command is rejected by the MMI-210 and MMI-105, because they have no analog output gain control.

Error Codes: Unsupported Command (MMI-210, MMI-105)

See Also: SET_PLAY_GAIN, § 14.4.8.
OUTPUT_GAIN, § 14.7.5.
LR_INPUT_GAIN, § 14.7.4.

14.7.7 ROUTE_OUTPUT

Summary: Set the analog output signal mixers. (MMI-4211 only)

Code: $003C

Arguments:

Mixer Bits	New mixer setting.

Description: This command selects which audio output ports will receive input from a specific DSP. The DSP that executes this command will send its output to the audio ports specified by Mixer Bits, where the signal will be mixed with the output from any other DSPs sending to that output port.

Each bit in the lower four bits of the Mixer Bits argument controls one output port. Setting the bit to one (1) will cause output from the DSP to be sent to the port. Clearing the bit (0) will cause the output port not to receive audio from the DSP. The bits are assigned as shown in Figure 14.4

Output Port	Bit Number	Byte Mask
A	0	$01
B	1	$02
C	2	$04
D	3	$08

Figure 14.4: MMI-4210 mixer register map.

Note that this command only applies to the MMI-4211. The MMI-210 and MMI-105 do not have analog mixers, and the mixers on the MMI-4210 are directly under host control. See Section 13.3.1 for information on the MMI-4210 mixers. The MMI-4210 will silently ignore the ROUTE_OUTPUT command.

Error Codes: Unsupported Command (MMI-210, MMI-105)

See Also: OUTPUT_GAIN, § 14.7.5.
SELECT_INPUT, § 14.7.8.

14.7.8 SELECT_INPUT

Summary: Select the analog input source. (MMI-4211 only)

Code: $003F

Arguments:

Input Source	0 selects Microphone; 1 selects Line-In.

Description: This command selects which audio input source will be used by the MMI-4211. This command is silently ignored on the MMI-4210 and will return an "Unsupported Command" error on other boards.

Each channel (DSP) of the MMI-4211 has two separate audio input jacks on the front panel. One is for microphone level input (source 0), and the other is for standard line-level input (source 1). The gain of either input source can be adjusted with the INPUT_GAIN command. The microphone input provides a fixed-gain preamplifier and filter suitable for direct connection to a dynamic or magnetic microphone.

Each DSP has an independent input selection, and all inputs default to Line-In upon initialization. The input source may be changed at any time, even during recording.

Error Codes: Unsupported Command (MMI-210, MMI-105)

See Also: INPUT_GAIN, § 14.7.3.

14.7.9 STEREO_MODE

Summary: Select stereo or monophonic mode. (MMI-4211 only)

Code: $0040

Arguments:

Enable	1 selects stereo mode, 0 selects mono.

Description: This command enables or disables stereophonic operation. When operating in stereo, there are **two** samples per time division: one for the left channel and one for the right. These samples are interleaved in the audio data stream beginning with the left channel.

Stereo operation is supported only by the MMI-4211, which has stereo inputs and outputs for each DSP. DiSPATCH firmware prior to version 3.41 sends mono audio to only the left output channel. Versions 3.41 and newer will send identical mono audio to both output channels.

Since stereo uses two samples per time division, stereo consumes twice the bandwidth of mono operation, as it will produce or require twice as much data per second. Note that the sample rate (in Hertz) always represents time-frames per second and does not change for stereo or mono mode.

Notes: Stereo interleaved samples conflict with certain DiSPATCH operating modes. Specifically, both algorithmic audio encoding formats (VQ and ADPCM) will not work correctly if stereo is enabled. The remaining PCM encoding formats are compatible with stereo operation (PCM-8, PCM-16, μ-Law, and A-Law).

Most special playback and recording signal processing features (FIR filters, effects, time change) are not compatible with stereo operation, and will produce distorted audio. They should not be enabled when using stereo mode.

Error Codes: Unsupported Command (MMI-210, MMI-105)

14.8 Initialization and Configuration

14.8.1 LOAD_X

Summary: Transfer data from DRAM to the DSP's private X memory space.

Code: `$0001`

Arguments:

Word Count	Number of 24-bit words to copy into X memory.
Load Dest	Destination address in DSP X memory.
Address (Hi) Address (Lo)	DRAM source address for data bytes.

Description: Each DSP has three private regions of high-speed local static RAM, not directly accessible by the VME host. This command allows the host to replace blocks of memory in the DSP's "X" memory space by copying words from shared DRAM. During the boot sequence, this command must be used to initialize X memory with constants used by the firmware.

The DSP LOAD_X, LOAD_Y, and LOAD_P commands may be used at any time during DiSPATCH operation, provided that constants, variables, and program code used by the firmware are not overwritten. This allows a DSP programmer to load program code overlays, filter coefficient tables, or waveform data during operation. Of course, a thorough understanding of the firmware's internal operation is necessary to avoid overwriting active data or code.

Data Format: The DSP X memory consists of 24-bit data words. Since not all MMI boards have 24-bit shared memory, this command reads *three 8-bit values* from DRAM to build one 24-bit word for X memory. Note that the Word Count argument must be the number of DSP words, not the number of bytes provided by the host.

The host must place each 24-bit word into three words, using only the lowest eight bits of shared DRAM for data. Any DRAM bits above the lowest eight should be zero. The bytes are to be in *high-middle-low* order, with the highest eight bits in the lowest DRAM memory location.

Ranges:

Argument	Type	Min	Max
Word Count	Word	0	X RAM Size
Load Dest	Word	0	X RAM Size
Address	Extended	0	Max DRAM

Error Codes: None.

See Also: LOAD_Y, § 14.8.2.
LOAD_P, § 14.8.3.

14.8.2 LOAD_Y

Summary: Transfer data from DRAM to the DSP's private Y memory space.

Code: $0002

Arguments:

Word Count	Number of 24-bit words to copy into Y memory.
Load Dest	Destination address in DSP Y memory.
Address (Hi) Address (Lo)	DRAM source address for data bytes.

Description: This command is identical to the LOAD_X command, but it copies data to the Y region of DSP private memory instead of X.

Ranges:

Argument	Type	Min	Max
Word Count	Word	0	Y RAM Size
Load Dest	Word	0	Y RAM Size
Address	Extended	0	Max DRAM

Error Codes: None.

See Also: LOAD_X, § 14.8.1.
LOAD_P, § 14.8.3.

14.8.3 LOAD_P

Summary: Transfer data from DRAM to the DSP's private P memory space.

Code: $0003

Arguments:

Word Count	Number of 24-bit words to copy into P memory.
Load Dest	Destination address in DSP P memory.
Address (Hi) Address (Lo)	DRAM source address for data bytes.

Description: This command is identical to the LOAD_X command, but it copies data to the P region of DSP private memory instead of X.

Ranges:

Argument	Type	Min	Max
Word Count	Word	0	P RAM Size
Load Dest	Word	0	P RAM Size
Address	Extended	0	Max DRAM

Error Codes: None.

See Also: LOAD_X, § 14.8.1.
LOAD_Y, § 14.8.2.

14.9 Diagnostics

14.9.1 FETCH_ERROR

Summary: Retrieve and clear the current error code.

Code: $000A

Arguments:

⇒ Error Code The current error status code

Description: When the host has been notified that a command resulted in an error condition, the host should immediately request the error code from the DSP. The Error Code argument is set by the DSP to reflect the reason for the latest error. Error codes prior to the last error are not retrievable, so it is important to read the error code soon after the error condition, before issuing another command. See Chapter 11 for more information on the error reporting mechanism.

After an error code has been retrieved, any further calls to FETCH_ERROR will return $0000 indicating that there is no error pending. See Chapter 15 for a comprehensive list of the firmware error codes and their definitions.

Error Codes: None.

14.9.2 TEST_DRAM

Summary: Initiates a comprehensive and destructive test of shared DRAM.

Code: $0037

Arguments:

Size (Hi) Size (Lo)	Number of DRAM words to test.

Description: This command instructs the DSP to start a thorough test of every word in the on-board shared DRAM. The DSP notifies the host when it has completed the test or found an error. The test takes approximately 10 seconds per megabyte of on-board DRAM.

After the test has completed, the DRAM will contain a special tag sequence that the host should read and verify. This will ensure that the DSP and host access DRAM identically. The tag value for each 16-bit word can be computed as follows:

```
Tag = Address & 0xFFFF ^ (Address >> 16)
```

Since the test is completely destructive, any commands or data stored in DRAM must be replaced after the test.

If any errors are detected during the DRAM test, the DSP will abort the test and return a "Hardware Failure" error. This indicates a defective board.

Notes: This command ($0037) replaces the previous TEST_DRAM command (code $0017). The old TEST_DRAM command has a fixed size (4 megabytes) and took no arguments.

The old command is still supported by the firmware, but will always test four megabytes of DRAM. This will cause it to fail on all boards with less than four megabytes, such as the MMI-210-1. Please consider command code $0017 obsolete and use only the new sized DRAM test command: $0037.

Ranges:

Argument	Type	Min	Max
Size	Extended	1	Max DRAM

Error Codes: Hardware Failure

14.9.3 QUERY_LOAD

Summary: Request a report of the approximate DSP bandwidth utilization.

Code: $0029

Arguments:

⇒ Play Load	Worst play load since last report
⇒ Play Max	Maximum allowable load for real-time playback
⇒ Record Load	Worst record load since last report
⇒ Record Max	Maximum allowable load for real-time recording

Description: During the course of processing, the DSP keeps track of how long it takes to process samples for playback and recording. The DSP has a limited amount of real time, and a certain number of samples that must be produced in that time. The QUERY_LOAD command can be used to ask how much processing time is being consumed by the DSP.

The reported Play Load is a representation of how much time it took to process one batch of samples. At most, the DSP can take Play Max units of time for each block. If, at any time, the Play Load exceeds Play Max, then the DSP did not finish preparing samples in time, and there was a glitch or pop in the audio stream.

A rough approximation of bandwidth utilization during playback can be derived as follows:

$$\text{Utilization} = \frac{\text{Play Load}}{\text{Play Max}}$$

Since the load varies rapidly during processing, the DSP only keeps track of the *maximum* load since the last report. Each invocation of the QUERY_LOAD command resets the maximum load to zero.

The Record Load and Record Max are analogous to the Play counterparts, but record the loading during record processing (incoming samples).

Error Codes: None.

14.9.4 LSHIFT

Summary: Left-shift the argument value by one bit and write it back to DRAM.

Code: $000B

Arguments:

⇒ Value The value to shift left

Description: This command is primarily used to quickly verify that the command and argument mechanisms to the DSP are in working order. The host can store a value, ask the DSP to shift it left by one bit, read it back, and compare it to the expected value.

This is not intended as a substitute for the comprehensive diagnostic tests, but may be used as a quick probe to determine if a DSP is initialized and running the firmware.

Ranges:

Argument	Type	Min	Max
Value	Word	$0000	$7FFF

Error Codes: None.

14.9.5 ANALOG_TEST

Summary: Test the analog subsystem of the MMI board.

Code: `$0030`

Arguments:

	CPU Speed	DSP speed, in megahertz.
⇒	Sample Rate	Measured approximate sample rate.
⇒	Energy	Measured loopback signal energy.
⇒	Impurity	Measured loopback signal impurity.

Description: This command performs a comprehensive test of the analog subsystem on MMI boards equipped with analog conversion circuitry. This test does not make a PASS or FAIL decision, but reports the signal measurements to the host for analysis.

Before performing the analog test, the operator must connect the output channel of the audio board directly to the input of the same DSP. This allows the DSP to send signals out and monitor their quality.

The one argument the host must provide is the DSP speed in Megahertz. Round the actual DSP clock rate to the nearest whole number for the CPU Speed argument. This value is used for calibrating the internal timers, and must be valid to make a correct Sample Rate measurement.

The returned Sample Rate is an approximate measurement of the sample clock of the ADC and DAC. The host should verify that it is a reasonable value, and very close to the rate last selected by the host (see SET_SRATE and SET_PNM).

The next two measurements are the results of the signal quality analysis. The Energy rating is a measure of the average RMS received by the input circuit during the test. This should be at least 10% of the available range for the test to produce meaningful results.

The Impurity reading is a measurement of how much distortion was present in the received signal. A reading of greater than about 3% may indicate a noise problem with the board.

Note that the test is significantly affected by the current analog input and output gain settings. Some adjustment may be necessary to properly match the outgoing and incoming signal strengths.

Ranges:

Argument	Type	Min	Max
CPU Speed	Word	see text	
Sample Rate	Word	0	65535
Energy	Word	0	65536
Impurity	Word	0	65536

Error Codes: None.

14.9.6 EPROM_CHECKSUM

Summary: Compute a checksum for the on-board EPROM if one is present.

Code: $0039

Arguments:

Checksum	16-bit EPROM checksum value.

Description: This command computes a simple checksum of the DSP's EPROM, if one is installed. If an EPROM is not installed on the board, then the Checksum will be random and meaningless.

The EPROM is not used by DiSPATCH, and is not necessary for correct operation. Boards may be shipped with or without EPROMs.

Ranges:

Argument	Type	Min	Max
Checksum	Extended	$0000	$FFFF

Error Codes: None.

14.9.7 HAMMER

Summary: Execute a fixed number of instructions to measure the DSP clock speed.

Code: $0038

Arguments: None.

Description: This command causes the DSP to execute a long loop with all interrupts masked. The host can measure the time it takes to execute this loop and then compute the DSP clock speed. This diagnostic procedure is used to ensure that the DSP CPU master clock is operating at the desired frequency.

The HAMMER processing loop takes exactly 162,024,000 DSP clock cycles. A 27 MHz DSP executes the loop in approximately six seconds. Small time variations are expected when the command latency and host interrupt response times are added, but gross variations in the execution time may indicate a failed DSP clock.

DiSPATCH does not measure the execution time or make any pass/fail judgements during the HAMMER delay loop. These functions must be performed by the host.

Since all normal processing is suspended during the HAMMER command, all playback and recording will be paused during the test.

Error Codes: None.

14.10 Playback Monitor

14.10.1 MONITOR_BUFFER

Summary: Begin copying the outgoing audio data into DRAM, or queue a request if monitor is already in progress.

Code: $002A

Arguments:

Count (Hi) Count (Lo)	Number of words to copy to DRAM.
Address (Hi) Address (Lo)	DRAM start address for data words.

Description: This command requests that all outgoing audio be copied into the specified DRAM buffer. This is similar to the RECORD_BUFFER command, but it "records" the outgoing audio. While monitoring is in progress, the MONITOR_BUFFER command queues requests exactly like RECORD_BUFFER.

The format for the monitor data is selected by use of the SET_MONITOR_FORMAT command. The default format is PCM-16.

Ranges:

Argument	Type	Min	Max
Count	Extended	0	Max DRAM
Address	Extended	0	Max DRAM

Error Codes: None.

See Also: MONITOR_POS, § 14.10.2.
ABORT_MONITOR, § 14.10.3.
SET_MONITOR_FORMAT, § 14.10.4.

14.10.2 MONITOR_POS

Summary: Check current monitor position in DRAM.

Code: $002C

Arguments:

⇒ Count (Hi) ⇒ Count (Lo)	Words remaining in the current monitor request.
⇒ Address (Hi) ⇒ Address (Lo)	Address where next monitored sample will go.

Description: While a monitor command is in progress and filling a buffer, the host can check the current position within the buffer by using this command. The returned values are only valid when a monitor command is in progress, and undefined otherwise.

See Section 5.7.2 for information on reading the buffer position.

Ranges:

Argument	Type	Min	Max
Count	Extended	0	Max DRAM
Address	Extended	0	Max DRAM

Error Codes: None.

See Also: MONITOR_BUFFER, § 14.10.1.
ABORT_MONITOR, § 14.10.3.
RECORD_POS, § 14.5.8.
PLAY_POS, § 14.4.12.

14.10.3 ABORT_MONITOR

Summary: Abort all active and pending monitor requests.

Code: $002B

Arguments: None.

Description: While monitoring is in progress, the host may request that all pending requests be discarded, and monitoring aborted. This will result in an immediate termination of all monitoring.

After this command is acknowledged, the host will not receive Buffer Completion messages from those requests that were discarded or not finished. The host is allowed to abort monitoring at any time, even when there is no monitor in progress.

To determine the position of the active buffer at the time it was aborted, the host may follow a procedure similar to that outlined in Section 5.7.2.

Error Codes: None.

See Also: MONITOR_BUFFER, § 14.10.1.
MONITOR_POS, § 14.10.2.

14.10.4 SET_MONITOR_FORMAT

Summary: Set audio data format for monitoring playback.

Code: $002D

Arguments:

Format	Data format to use for monitoring.

Description:

This changes the format for monitored data to be Format. The Format argument must be one of the audio data formats listed in the table below.

The default data format for monitoring is PCM-16.

Encoding	Format Code
PCM-16	$0016
PCM-8	$0008
μ-Law	$0006
A-Law	$000A
ADPCM	$000D
VQ	$0002

Figure 14.5: Monitor Data Formats

Ranges:

Argument	Type	Min	Max
Format	Word	see Table 14.5	

Error Codes: Invalid Format Code

See Also: MONITOR_BUFFER, § 14.10.1.
SET_PLAY_FORMAT, § 14.4.9.
SET_RECORD_FORMAT, § 14.5.6.

14.11 Miscellaneous

14.11.1 TRANSFORM

Summary: Process data in DRAM, returning the results to DRAM.

Code: $001F

Arguments:

Source Format	Input data format.
Source Addr (Hi) Source Addr (Lo)	DRAM address of source data.
Source Count (Hi) Source Count (Lo)	Number of DRAM input words to process.
Dest Format	Resulting data format.
Dest Addr (Hi) Dest Addr (Lo)	DRAM start address for finished data.
⇒ Dest Count (Hi) ⇒ Dest Count (Lo)	Number of resulting DRAM words.
Subbuffers	Number of sub-buffers.
Spacing (Hi) Spacing (Lo)	Offset between sub-buffers.
Process	Special processing program code.

Description: This performs in-RAM data format conversion and/or signal processing. The host specifies a data format, address, and count for a source buffer in DRAM. DiSPATCH converts the buffer to an internal format and then to the destination format. The finished data is written to the destination buffer in DRAM starting at Dest Addr. When done, the DSP puts the count of finished samples into the Dest Count argument.

Like PLAY_SUBBUFS, the TRANSFORM command can operate on sub-buffers. The source data from all specified sub-buffers will be converted from the source data format, mixed together, and then converted to the destination format. For a normal (no sub-buffers) transform, the Subbuffers argument must be set to one. All sub-buffers must be in the same data format.

The gains used during mixing are the same as those used during playback. Since there is no Track argument, the first sub-buffer uses the gain from track zero, the second sub-buffer uses the gain from track one, etc.

This command does not return a Command Acknowledge until the transform operation is entirely complete. During command execution, the DSP is entirely occupied and suspends playback and recording.

The Process argument is reserved for future use and must be set to $FFFF. In later DiSPATCH revisions, this argument may select special processing to be applied to the data.

Ranges:

Argument	Type	Min	Max
Source Addr	Extended	0	Max DRAM
Source Count	Extended	0	Max DRAM
Dest Addr	Extended	0	Max DRAM
Dest Count	Extended	0	Max DRAM
Subbuffers	Word	1	24
Spacing	Extended	1	Max DRAM
Process	Word	$FFFF	$FFFF

Error Codes: Sub-buffer Count is Zero
Invalid Format Code

14.11.2 LED_CTRL

Summary: Turn front-panel LEDs on or off.

Code: $000D

Arguments:

LED ID	Which LED to set.
State	On or off.

Description:

Each MMI board has one or more front-panel LEDs that are under DSP control. The LED_CTRL command turns a front-panel LED on or off.

The first argument specifies which LED is to be set. The LEDs are coded as follows:

MMI-4210

Each DSP controls one red front-panel LED. The LED ID argument must always be set to $0000.

MMI-105

The green LED is LED ID $0000, and the red LED is LED ID $0001.

MMI-210

There are two front-panel LEDs on the MMI-210, but both must be controlled by DSP A. Like the MMI-105, the green LED is LED ID $0000, and the red LED is LED ID $0001.

The State is set to $0001 to turn the LED on, or $0000 to turn the LED off.

Ranges:

Argument	Type	Min	Max
LED ID	Word	0	see text
State	Word	0	1

Error Codes: Invalid LED Number

See Also: CLIP_LED, § 14.5.14.

14.11.3 VERSION

Summary: Retrieve the revision number of the DiSPATCH firmware.

Code: $000C

Arguments:

⇒ Major Version

⇒ Minor Version Version code for the DiSPATCH firmware.

Description: Each release of DiSPATCH carries a unique version number for identification. The version is in two parts: the Major and Minor number.

Firmware versions of the same Major number are intended to be fully command-compatible. Most application software should at least check the Major version number to verify that the firmware supports the same command set as expected. If significant functionality is added or commands are rendered incompatible with the previous versions, the Major number will be incremented.

The Minor number is used for any firmware updates that do not affect application compatibility. This includes bug fixes, minor added features that can be safely disregarded, optimizations, etc. *Any* and all changes to DiSPATCH will change the Minor number.

Ranges:

Argument	Type	Min	Max
Major Version	Word	0	65535
Minor Version	Word	0	99

Error Codes: None.

See Also: CONFIGURATION, § 14.12.4.

14.11.4 NOP

Summary: Acknowledge command, but do nothing.

Code: $0026

Arguments: None.

Description: This command behaves like any normal command, but does nothing at all. It can be used to verify the command protocol during debugging, but the LSHIFT command provides a more complete protocol test.

Error Codes: None.

See Also: LSHIFT, § 14.9.4.

14.11.5 COUNT_BUFFERS

Summary: Request a count of the active and pending play, record, and monitor buffers.

Code: $002E

Arguments:

⇒	Play Count	Total number of pending play requests
⇒	Record Count	Number of pending record requests
⇒	Monitor Count	Number of pending monitor requests

Description: This command reports the number of requests that are currently queued for processing. The counts include the buffer currently in progress.

The Play Count is the total number of play requests on all available tracks, including active buffers. The Record Count and Monitor Count are counts of the total requests for recording and monitoring, respectively.

Ranges:

Argument	Type	Min	Max
Play Count	Word	0	250
Record Count	Word	0	10
Monitor Count	Word	0	10

Error Codes: None.

See Also: PLAY_POS, § 14.4.12.
RECORD_POS, § 14.5.8.
MONITOR_POS, § 14.10.2.
ABORT_TRACK, § 14.4.13.
ABORT_ALL_PLAY, § 14.4.14.
ABORT_RECORD, § 14.5.9.
ABORT_MONITOR, § 14.10.3.

14.12 Unsupported

The commands described in this section are completely unsupported by Vigra. They are loosely documented, generally untested, and provided for amusement only. They were developed for proof-of-concept, demonstration, or debugging purposes. None are necessary for full DiSPATCH operation.

These commands may change without notice between DiSPATCH revisions, especially if the resources used are needed elsewhere. These commands may totally disappear without notice from future versions of this document or the firmware itself.

14.12.1 LOOPBACK

Summary: Begin digital audio loopback test. (obsolete)

Code: `$0069`

Arguments: None.

Description: This command initiates a digital loopback test. During the test, the DSP "passes through" all audio by digitizing and immediately sending the data out with no intermediate digital processing.

This command is now obsolete, because SET_SIDETONE can perform the same function. By setting the sidetone volume to unity gain (0.0), all incoming audio is digitally passed on to the output.

The LOOPBACK command may not be present in future versions of DiSPATCH.

Unlike sidetone, a loopback test suspends all normal play and record activity for the duration of the test. The test is terminated by the END_LOOPBACK command.

Error Codes: None.

See Also: END_LOOPBACK, § 14.12.2.
SET_SIDETONE, § 14.5.5.

14.12.2 END_LOOPBACK

Summary: End digital audio loopback test.

Code: $0096

Arguments: None.

Description:

This command ends a digital loopback test initiated by the LOOPBACK command.

Error Codes: None.

See Also: LOOPBACK, § 14.12.1.

14.12.3 QUERY_STATE

Summary: Request internal operating status.

Code: $0032

Arguments:

⇒ Speed Change	Speed change enabled
⇒ Monitor	Monitoring in progress
⇒ Sig Detect	One or more signal detectors active
⇒ Equalizer	Equalizer enabled
⇒ Loopback	Digital loopback test running
⇒ Mailbox	Debugging message mailbox enabled
⇒ Reverb	Reverb enabled
⇒ Is DB422	A DB-422 daughterboard was detected
⇒ In Stereo	Stereo mode active
⇒ Interpolate	Interpolating filters active
⇒ Decimate	Decimating filters active
⇒ Can't Sidetone	Current settings exclude sidetone
⇒ Play FIR	Playback FIR filter enabled
⇒ Record FIR	Recording FIR filter enabled
⇒ Measurements	Input signal measurements enabled

Description:

This command may be used to check the operating state of DiSPATCH. The returned arguments reveal whether the specified modules are enabled (1) or disabled (0). This command is an internal development tool and the arguments are subject to change. Specifically, flags are frequently added to reflect new operating modes.

Error Codes: None.

See Also: ENABLE_MAIL, § 14.12.7.
DISABLE_MAIL, § 14.12.8.

14.12.4 CONFIGURATION

Summary: Query DSP firmware configuration parameters

Code: $0011

Arguments:

⇒ Major Version	
⇒ Minor Version	DiSPATCH version code
⇒ Model ID	MMI model number intended for firmware
⇒ Resp Queue	Size (in entries) of the host response queue
⇒ Num Tracks	Number of available tracks
⇒ Play Slots	Size of the play request queue
⇒ Record Slots	Size of the record request queue
⇒ Play Bufsize	Internal playback buffer size
⇒ Record Bufsize	Internal record buffer size
⇒ P RAM Size	Expected size of P memory
⇒ X RAM Size	Expected size of X memory
⇒ Y RAM Size	Expected size of Y memory
⇒ Num Commands	Number of recognized commands
⇒ P Tested RAM	Size of P memory detected by RAM test
⇒ X Tested RAM	Size of X memory detected by RAM test
⇒ Y Tested RAM	Size of Y memory detected by RAM test

Description: This command may be used to peek at the configuration parameters used to build DiSPATCH.

This command is *not supported* and provided only for debugging and curiosity. The interpretation, order, and contents of the returned values is always subject to change. Consider this command undocumented and volatile.

See Also: VERSION, § 14.11.3.

14.12.5 EQUALIZER

Summary: Set the 10-band spectrum equalizer

Code: $0031

Arguments:

Enable	Enable or disable the equalizer.
Gain 31 Hz	Equalizer gain setting for band 1.
Gain 62 Hz	Equalizer gain setting for band 2.
Gain 125 Hz	Equalizer gain setting for band 3.
Gain 250 Hz	Equalizer gain setting for band 4.
Gain 500 Hz	Equalizer gain setting for band 5.
Gain 1 kHz	Equalizer gain setting for band 6.
Gain 2 kHz	Equalizer gain setting for band 7.
Gain 4 kHz	Equalizer gain setting for band 8.
Gain 8 kHz	Equalizer gain setting for band 9.
Gain 16 kHz	Equalizer gain setting for band 10.

Description: The 10-band spectrum equalizer is an unsupported feature that will likely disappear in future revisions of DiSPATCH. It is calibrated to work only at a 44100 Hz sample rate, since the filters have fixed coefficients. It's quite a cycle-hog, so only limited playback processing can be accommodated while the EQ is enabled.

Each gain has a range of –0.2 to 1.0, with 0.0 giving unity gain at that frequency. If all gains are set to unity, the spectrum is perfectly flat. With all gains set to –0.2, the overall gain is about –14 dB. All gains set to 1.0 produces a gain of about five (+14 dB). The gains can be changed at any time during playback.

The gain argument values are 16-bit signed words representing 32767 ($7FFF) times the fractional gain (–0.2 to 1.0).

Ranges:

Argument	Type	Min	Max
Enable	Word	0	1
Gains (10)	Word	-6553	32767

Error Codes: None.

14.12.6 REVERB

Summary: Configure digital reverb effect.

Code: $0033

Arguments:

Sustain (Hi) Sustain (Lo)	How long to retain the signal (feedback gain).
Delay	Time delay between reverberations.

Description: This is a playback filter that is well suited to special audio effects and Cylon voice synthesis. If the reverb is set to be reasonably long, and not too prominent, it may even pass as a decent reverb as well!

The Sustain argument specifies the strength of the reverb, setting the feedback level. Small values give a quick decay, while larger values last a long time or amplify to distortion.

The Delay specifies how long the reverb delay line is, and can be set to small values ($\sim$ 30) for interesting comb filter effects. It cannot be set any higher than 2046, but slowing the sample rate can make the delay longer.

Note that this command will most likely be removed, since it hogs a large portion of the on-board SRAM for the delay lines. As soon as a more useful feature needs the RAM, reverb will have to go. Until then, have fun!

Ranges:

Argument	Type	Min	Max
Sustain	Fraction	-1.0	0.999999881
Delay	Word	0	2046

Error Codes: Parameter Out of Range

14.12.7 ENABLE_MAIL

Summary: Request debugging messages from DiSPATCH.

Code: $0010

Arguments: None.

Description: This turns on special messages from the DSP that aid in debugging and tracing. This feature provides the rough equivalent of `printf()` when customizing the DiSPATCH program itself. Mail messages are disabled by default.

Note that the Stage-1 and Stage-2 boot progress messages (Section 3.2.2) are not considered mail messages and are always sent.

Error Codes: None.

See Also: DISABLE_MAIL, § 14.12.8.

14.12.8 DISABLE_MAIL

Summary: Suspend debugging messages from DiSPATCH.

Code: $001A

Arguments: None.

Description: This turns off the special mail messages from the DSP. The default state is off.

Error Codes: None.

See Also: ENABLE_MAIL, § 14.12.7.

15. Error Code Definitions

`$0000`: **OK (No error)**
This indicates that the host has requested an Error Code when there was no new error condition pending.

`$0001`: **Unrecognized Command Code**
The host has attempted to use a **Command Code** that is not recognized by the firmware.

`$0002`: **Invalid Sampling Frequency**
The host has requested a sampling rate not available for the MMI model in use.

`$0003`: **Invalid Track Number**
An invalid or out-of-range Track number was used.

`$0006`: **Invalid Format Code**
An invalid audio data format code was selected. See Table 10.1 on page 43 for a listing of available format codes.

`$0007`: **Invalid LED Number**
The requested LED code does not exist on the MMI model in use.

`$0008`: **Unsupported Command**
The requested command is not supported on the MMI model in use, or has not yet been implemented. Most hardware-specific commands will return **Unsupported Command** on MMI boards without the required hardware.

`$0009`: **Hardware Failure**
This indicates that a hardware malfunction was detected by the DSP. The board should be returned to Vigra for replacement or repair.

`$000A`: **Speed Change Out of Range**
An invalid speed-change setting was selected. Only speeds from 50% to 200% are supported (see Section 7.2).

`$000B`: **Sub-Buffer Count Is Zero**
There must be at least one sub-buffer for any command that processes sub-buffers. Zero sub-buffers in a buffer is not valid.

`$000C`: **Invalid Signal Detector Number**
There are four available signal detectors, numbered 0 through 3. Any other signal detector number is invalid.

`$000E`: **Invalid CPU Speed**
The analog self-test requires that a valid CPU speed be specified. This value is in megahertz.

`$000F`: **Parameter Out of Range**
A given parameter was outside the allowable range of values.

`$0010`: **Invalid Wavetable Number**
The table number given is not valid. Allowable table codes are 0 through 5. The contents of table 0 can not be overwritten.

`$0011`: **Sidetone Not Available**
Since sidetone is a direct digital pass-through, the playback and record sample rates must be equal. Unequal rates will inhibit sidetone and cause this error.

`$0012`: **Invalid Number of FIR Taps**
The playback and recording FIR filters allow a maximum of 256 filter taps. Specifying more than 256 taps or zero taps is not allowed and will result in this error.

`$0013`: **Feature Not Enabled**
The host has requested a feature that is presently disabled or inactive. For example, this error is returned if the host requests an input bias measurement while input measurements are not enabled.

INDEX