DiSPATCH
Tone Generation Manual

Revision 1.12
Updated 11 Mar. 1997
Part Number: 880-3096-002

Vigra, a division of VisiCom Labs.

This is revision 1.12 of the *DiSPATCH Tone Generation Manual*.

This document was last updated on 11 Mar. 1997.

CONTENTS

1. INTRODUCTION TO TONE GENERATION

Tone generation is a new addition to the DiSPATCH DSP firmware for Vigra's VME-based digital audio signal processing boards. This module allows the DSP to serve as a powerful real-time tone synthesizer.

Using tone generation, the VME host can request that one or more sounds be generated by the DSP and mixed with any other DiSPATCH audio output. The host has complete and immediate control over all tone parameters, even while the tone is playing.

Tone generation is well suited to a variety of common alarm and signal sounds, such as sirens, horns, beeps, bells, buzzers, confirmation tones and alert signals.

An interactive tone editing tool is included for rapid tone development and experimentation. Library functions are also provided for efficient high-level tone control from the host application.

1.1 Features

- Real-time digital mixing of up to 24 independent tones per DSP.
- Complete compatibility with existing DiSPATCH applications.
- Tones can be mixed with other audio tracks, such as speech data.
- All tone parameters can be modified "on-the-fly" (while a tone is playing).
- Programmable amplitude envelope (attack, sustain, and release).
- Frequency and amplitude modulation synthesis techniques supported.
- Selectable sine, triangle, sawtooth, and square waveforms built-in.
- Arbitrary waveform tables can be dynamically supplied by the host.
- Supports variable-rate tone sweeps and two-tone sounds.

1.2 Document Overview

This manual explains the use of the Tone Generation module in DiSPATCH. It is assumed that the reader is familiar with the operation and concepts of DiSPATCH, as documented in the *DiSPATCH User's Manual*. If you are a new user of DiSPATCH, please familiarize yourself with the DiSPATCH firmware and it's operation before attempting to program tone generation.

Digital sound synthesis is a complex and evolving field of computer programming. It is beyond the scope of this document to describe in detail the techniques necessary to produce complex and realistic sounds. This manual assumes that the reader has a sound working knowledge of audio synthesis and a willingness to experiment with the available parameters.

2. Applications of Tone Generation

The tone generation module of DiSPATCH is well-suited to most applications requiring alarm and signal tones. By carefully selecting parameters, DiSPATCH can accurately simulate many electronic and mechanical audio indicators.

However, DiSPATCH it is not a suitable system for complex sound synthesis. Real-world sounds like engines and musical instruments are extremely complex; even simple sounds can have a vast number of interacting components and dynamic characteristics.

The best way to synthesize complex sounds is to simply “sample” them digitally, using the recording capabilities of DiSPATCH. These sounds can then be stored or played back with full realism and only minimal processing requirements.

Unfortunately, the sampling approach cannot be used for sounds that must change during playback. For instance, a tone that changes frequency depending on an operator's selection. Tones that must sound indefinitely or for an unknown length of time can also be difficult to record.

For these unpredictable or lengthy tones, DiSPATCH synthesis offers a convenient solution that can create sounds instantly and without storage constraints. The host is free to change the characteristics of the tone as it is playing.

3. ACCESSING TONE GENERATION

Tone generation is a new playback module available only in DiSPATCH versions 3.20 and newer. It is implemented as a new "format" for audio output, like PCM-16, ADPCM, and VQ. This new format is called **ToneGen**. The format code for ToneGen is `$00FF`.

This ToneGen audio format is very unique in that it is a *playback only* format. It is not possible to record or transform using the ToneGen audio data format. The ToneGen format is also unique because it does not read data buffers from DRAM. Instead, it uses the current set of tone parameters to synthesize audio samples.

Because ToneGen is simply an output format, it can be enabled or disabled separately for each track. Other tracks are still able to operate normally as audio data playback tracks, and the resulting audio streams will be mixed together accordingly.

For example, Track 0 can be set to the PCM-16 data format while Track 1 is set to ToneGen. Audio playback (speech, for example) will proceed normally on Track 0, while Track 1 will contribute a synthesized tone. The resulting mix of the two tracks, using their Play Gain settings, will be passed to the output stage. Track 1 will, of course, be silent when no tones are being synthesized.

The DiSPATCH command, `TONEGEN`, is used to set or change tone parameters. To play a tone, first set the desired audio track to the ToneGen format, and then issue a `TONEGEN` command to set the tone parameters.

The behavior of the `TONEGEN` command is very much like that of `PLAY_BUFFER`. The host issues a `TONEGEN` command specifying the tone to be synthesized, and immediately receives a "Command Acknowledge" message from the DSP. When the tone has completed playing, the DSP will send a "Buffer Completion" message to notify the host.

When the host issues a `TONEGEN` command, the new tone parameters take affect *immediately*. Note that this is different from `PLAY_BUFFER` command which queues up new requests until the present buffer is finished playing. In other words, the parameters to `TONEGEN` always overwrite the current parameters and are never

queued. This allows the host to dynamically change the characteristics of a tone while it is playing.

Each playback track set to the ToneGen output format can play one tone. The tone parameters for each Track are completely independent of one another. Pausing playback with the PAUSE_PLAY command will also pause any tones in progress. To abort a tone before it finishes playing, the host can dynamically replace it's length with zero.

3.1 Generating Tones to DRAM

Some applications may require that tones be pre-generated into DRAM rather than synthesized during playback. Once the tone is pre-generated, the host may queue the buffer to be played (using PLAY_BUFFER) just like any other audio data.

This has several advantages:

- This provides a mechanism for tones to be played back-to-back. Since new tone parameters take effect immediately, the host can not queue tone parameters for sequential playback. When the tone audio is in DRAM, it can be queued for playback just like any other audio data.
- Since the tones are played as normal audio data rather than synthesized, it is usually less computationally intensive for the DSP to play pre-generated tones from DRAM than compute the tone. This is useful if the DSP is heavily burdened with other audio tasks, or the same tone is to be played repeatedly.
- When the audio data is present in DRAM, the host may read it for processing or storage. Without RAMTONE, the host does not have direct access to the tone samples.

However, generating tones into DRAM has some unique restrictions:

- The tone can not be modified while it is playing, because tone parameters for a pre-generated tone can not be changed without generating a new tone. Programs that need to immediately respond to events by altering a tone will not be well-suited to pre-generated tones.
- The full length of all tones must fit into the available DRAM. Real-time ToneGen permits any tone length, even infinite tones, but pre-generated tones must fit entirely into DRAM.

- Normal playback and recording are paused while synthesizing any tones to DRAM.

The RAMTONE command is used to generate tones to DRAM. This command takes as arguments:

1. A track number indicating which ToneGen parameters to use.
2. A data format code specifying the data format for the tone samples.
3. A DRAM start address to store the samples.
4. A count of samples to synthesize into DRAM.

When the RAMTONE command is complete, it returns the number of sample words generated into DRAM. For additional details on using the RAMTONE DSP command, see the *DiSPATCH Firmware User's Manual*. The corresponding library function, `mmi_ramtone()`, is documented in the *Host Support Software Manual*.

4. TONE SYNTHESIS MODEL

While no single synthesis model is well-suited to all sounds, a fixed model for tone generation is necessary to accommodate real-time processing constraints. The model chosen for DiSPATCH Tone Generation was carefully selected to provide a wide variety of sounds with minimal complexity and processing demands.

For each tone, the VME host selects 16 adjustable parameters which uniquely specify the resulting sound. Simple tones may not need to utilize all of the available parameters or functions, while complex tones will need to enable more of the features. Section 5 discusses bandwidth constraints and complex tones.

Please take a moment to study the tone generation block diagram in Figure 4.1 (page 18. Understanding the model is essential to efficiently program the synthesis system.

Each element of the synthesizer is explained in the sections below.

4.1 Wave Oscillator

The wave oscillator is the core of the tone generator. It produces a basic waveform that can be processed by the other synthesis modules.

To create the simplest tone possible, the user need only specify three parameters:

1. Length of the tone.
2. Wave Table describing the shape of the waveform.
3. Wave Frequency of the tone.

By using these three parameters, the host can easily create a simple beep or tone.

4.1.1 Tone Length

The Length of the tone specifies when the tone will be terminated. The library computes the tone length as follows:

$$\text{Length} = \text{Attack} + \text{Sustain} + \text{Release}$$

The DSP will play the tone for the specified length, and then stop the tone and return a Buffer Completion notice to the host.

4.1.2 Wave Table

The Wave Table specifies the basic shape of one period of the waveform. The tone will be generated by repeating this wave period over and over at the given frequency. This allows the host to control the shape and harmonics of the tone (i.e. sine-wave, triangle, square, sawtooth).

The wave-shape is specified as a **table number**. This number selects one of six available wave tables to use for synthesis. Each table contains 256 samples that represent one period of the waveform. For example, table number zero holds one period of a sine wave. By selecting table zero, the resulting waveform will be a sine wave.

Upon DiSPATCH initialization, the six tables contain useful default values for tone generation. The six default wave-shapes are shown in Figure 4.2. The FM, AM, and waveform oscillator units on all tracks share the same set of six tables.

If necessary, the host can replace the contents of most wave tables at any time. This is accomplished by executing the LOAD_TABLE command, described in the *DiSPATCH Firmware User's Manual*. Note that the contents of table zero (the sine wave) cannot be changed. All other tables can be replaced with user data. The changed table data will be used in all oscillator functions (FM, AM, and waveform).

4.1.3 Wave Frequency

The Wave Frequency parameter to the wave oscillator specifies how fast the tone generator will cycle through the given Wave Table. Regardless of the contents of the Wave Table, one pass through the 256 table entries is considered one wave period. For example, a frequency of 100 Hz will cause DiSPATCH to output the contents of the wave table 100 times per second.

The Wave Frequency parameter has very high resolution (.00000745 Hz), so precise control is possible, even at low frequencies. The highest useful frequency is always

one half of the sample rate (also called the Nyquist rate). This is equivalent to advancing halfway through the wave table at every sample.

4.2 Envelope

The simple tone specified by Wave Frequency, Wave Table and Length will start immediately after the host executes the TONEGEN command. After the given length, the tone will stop abruptly. This instantaneous starting and stopping of the tone will cause the tone to start and end with a harsh pop, because the signal goes from totally silent to full volume.

To make the on/off transition more smooth, the host can enable the amplitude envelope section of tone generation. This allows the tone to *gradually* rise in volume from silence to full volume and then make a gradual fade from full volume back to silence at the end of the tone. These transitions can be very quick to simply dampen the start/stop, or they can be very slow for a long fade-in or fade-out effect.

The host controls three parameters of the envelope:

1. Envelope Enable flag.
2. Attack Time.
3. Sustain Time.
4. Release Time.

These values specify how much time the tone spends in each of the three phases of the envelope. See the Envelope Function block of Figure 4.1 for an illustration.

4.2.1 Envelope Enable

The host selects whether or not the sound will have an amplitude envelope by setting the Envelope Enable flag. When this flag is set to one (true) the Attack Time, Sustain Time and Release Time values will control the amplitude envelope. When this flag is clear (false), the tone will stay at full volume throughout the attack, sustain and release phases. The length of the tone is not effected by the Envelope Enable flag.

4.2.2 Attack Time

During the attack phase of the envelope, the amplitude of the tone will rise from total silence to full volume. The Attack Time specifies how long this transition lasts. A short Attack Time will cause the tone to reach full volume quickly. This can be almost imperceptible, but will dramatically reduce the harsh pop at the onset of the tone. A long Attack Time will cause a slow fade-in effect, as the tone volume slowly rises from silence to full strength.

When using the provided library functions, the Attack Time is specified in seconds. For example, an Attack Time of 2.0 seconds will cause the tone to fade in and reach full strength over a two second period of time.

4.2.3 Sustain Time

When the amplitude of the signal reaches full-strength at the end of the attack phase, the sustain phase begins. During the sustain phase, the level of the signal is kept at maximum. By changing the Sustain Time, the length of the tone is changed without effecting the attack or release shape of the sound.

After the sustain time has elapsed, the envelope moves into the release phase.

4.2.4 Release Time

During the release phase of the envelope, the amplitude of the tone will fall from full strength to total silence. The Release Time specifies how long this transition will take. A short release time will cause the tone to diminish quickly, while a a long release time will cause the tone to fade out slowly.

When using the provided library functions, the Release Time is specified in seconds. For example, a Release Time of 3.0 seconds will cause the tone to fade from full volume to silence over a three second period.

4.3 Amplitude Modulation (AM)

The host can also selectively enable amplitude modulation (AM) processing on each tone. This allows for vibrato effects and ring modulation, as well as unique envelopes.

When AM is enabled, the AM Oscillator will produce an output signal for the

duration of the tone. This signal is *multiplied* by the normal output of the wave oscillator, producing a modulated or pulsed signal. Note that this AM effect is very different from the *summation* of two audio signals, which can be accomplished by mixing tones on different playback tracks.

By using AM with a square wave, DiSPATCH can generate pulsive burst signals, like blips, chirps and periodic beeps. Using AM with a sinewave can produce vibrato or ring modulation effects.

The AM module has the following control parameters:

1. AM Enable flag.
2. AM Table wave selection.
3. AM Start Point, if desired.
4. AM Frequency of modulation.

4.3.1 AM Enable

The host activates the AM synthesis module by setting the AM Enable flag true (one). Setting this flag to zero disables amplitude modulation.

4.3.2 AM Table

Like the main wave oscillator, the AM signal is generated by repeatedly cycling through a table of sample values. These sample values are read from one of the six available tables shown in Figure 4.2. Any table contents that have been replaced by the host with the LOAD_TABLE command will also be used by the AM oscillator.

The values in the table are interpreted on a range of silence ($8000) to maximum amplitude ($7FFFF). The midpoint of the range ($0000) represents 50% of full scale. This range interpretation allows the default tables to represent modulations between zero and full volume.

4.3.3 AM Start Point

It is often useful to specify *where* in the table the AM oscillator is to begin (i.e. the initial phase of the oscillator). This is especially important when the AM frequency is very slow to produce a slow and gradual change in tone volume.

For example, a very slow AM effect using the Sine Table can be used to fade a tone in and out gradually. By setting the AM Start Point to 64 (one fourth of the way into the table), the tone will begin at full volume. An AM Start Point of 192 (three fourths into the table) would start the tone from silence.

Unchanged AM Table Position

Any table position from 0 to 255 can be selected as the initial point for the AM oscillator. In addition, the host can specify the special value $FFFF for AM Start Point. This value will cause the AM Table position to remain unchanged from the current phase (table position) of the oscillator. It will remain at the value it was before the TONEGEN command was issued. This is useful when the host wishes to change other tone parameters but leave the AM oscillator position untouched. Any value from 0 to 255 will immediately set the oscillator to that table position.

4.3.4 AM Frequency

This setting controls the cycle rate of the AM oscillator. High-frequency modulation settings will change the frequency spectrum of the tone, while very low AM frequencies can be used to slowly change the volume.

Like the Wave Frequency, the AM Frequency has very high fractional resolution and is specified in Hertz when using the DiSPATCH library.

4.4 Frequency Modulation (FM)

The "center frequency" of a tone is determined by the Wave Frequency, as described in Section 4.1.3. When FM is enabled, the wave frequency is adjusted by a dynamically variable amount. This frequency modulation can be used to create surprisingly complex sounds or to generate a frequency sweeping tone.

The theory of FM synthesis is summarized mathematically by the following formula:

$$x(n) = A \sin\left(\frac{2\pi n f_c}{R} + \frac{f}{f_m}\sin(\frac{2\pi n f_m}{R})\right)$$

R = Sampling rate
f_c = Carrier frequency (Wave Frequency)
f_m = Modulation frequency (FM Frequency)
f = Change in frequency (FM Delta Frequency)

The frequency of the wave oscillator, $f(n)$, is then dependent on the FM oscillator:

$$f(n) = f_c + f \cos\left(\frac{2\pi n f_m}{R}\right)$$

Unfortunately, even a rigorous mathematical description of FM does little to explain how to use it. The best way to understand what FM does for a sound is to experiment with the parameters. The results are often unpredictable and sometimes impressive. In the simplest case, FM can be used to glide the frequency of a tone to effect a continuous sweep over a range of frequencies.

The parameters controlling FM in tone generation are:

1. FM Enable flag.
2. FM Table wave selection.
3. FM Start Point, if desired.
4. FM Frequency of modulation.
5. FM Delta Frequency, the depth of modulation.

4.4.1 FM Enable

Like the AM Enable flag, setting this value to zero disables the FM section, while a value of one enables FM.

4.4.2 FM Delta Frequency

The strength or depth of the frequency modulation is controlled by the FM Delta Frequency parameter. This value determines the size of deviation from the center (carrier) frequency.

When FM is enabled, the wave oscillator frequency changes depending on the FM parameters. The wave frequency can swing from Wave Frequency – FM Delta Frequency to Wave Frequency + FM Delta Frequency. The pattern of this frequency change and its rate is determined by the contents of the FM Table and the FM Frequency, respectively. For example, if the center frequency of the wave oscillator (Wave Frequency) is 1000 Hz, and the FM Delta Frequency is 200 Hz, then the output wave oscillator frequency will swing from 800 to 1200 Hz at the speed specified by FM Frequency.

4.4.3 FM Table

Like the main Wave and AM oscillators, the FM signal is generated by repeatedly cycling through a table of sample values. These sample values are read from one of the six available tables shown in Figure 4.2. Any table contents that have been replaced by the host with the LOAD_TABLE command will also be used by the FM oscillator.

The values in the FM Table represent the range from –Delta Frequency to +Delta Frequency. A table value of –1.0 ($8000) will result in an output signal at Wave Frequency – Delta Frequency. A table value of +1.0 ($7FFF) will result in an output frequency of Wave Frequency + Delta Frequency. A table value of 0.0 ($0000) will keep the center frequency exactly.

4.4.4 FM Start Point

Like the AM Start Point, this parameter specifies where in the FM Table to begin the tone. This is the initial phase of the FM oscillator. A special value of $FFFF indicates to DiSPATCH that the FM table point should keep its current value, and not be changed by the TONEGEN command.

4.4.5 FM Frequency

The FM Frequency specifies how fast the FM oscillator will cycle through the table. An FM Frequency of 1 Hz means that the table will repeat every second. Very high precision allows very small FM Frequencies to be specified. This can be especially useful for slow frequency sweeps which require an FM Frequency much smaller than one Hertz.

4.4.6 Implementing Tone Sweeps with FM

To create a sweeping tone, one that slowly slides from one frequency to another, the application can use FM with a very low FM Frequency. To apply a linear rise or fall, the triangle wave table is most useful, while other tables can be used for nonlinear slides.

The center frequency (Wave Frequency) is the midpoint between the low and high frequencies of the sweep. The FM Delta Frequency is simply the extent of the sweep. The table start point is used to determine whether the wave starts by rising or falling in frequency.

Example

For example, suppose a tone must make a linear slide from 500 Hz to 1400 Hz over a ten-second period. To get a linear slide, select the triangle table for the FM Table. To cause the frequency to rise, the FM Start Point will be at position 192, the bottom of the upward slope.

The center frequency (average) is $500 + 1400/2$, or 950 Hz. The FM Delta Frequency is 450 Hz to reach the desired endpoints. The FM Frequency will determine how fast the rise/fall takes place. A frequency of 0.1 Hz will cause the tone to slide through the *entire* FM wave table over a 10 second period. This is not correct, since the table includes both the up and down slopes. Instead, the tone should spend ten seconds rising from table position 192 (bottom) to position 64, the peak of the wave. This is exactly half of the table, so an FM Frequency of 0.05 Hz and a tone Length of ten seconds gives the correct tone.

4.5 Track Gain

The digital gain control provided for each track is especially important when using tone generation. This gain setting controls the contribution of one track to the sum total of all tracks on one DSP's analog output channel, as explained in Chapter 6 of the *DiSPATCH User's Manual*. The track gain is set with the SET_PLAY_GAIN command.

Since the Track Gain is applied digitally, the resulting waveform must not exceed the maximum signal range, or clipping and distortion may result. Since all signals synthesized by tone generation are *full amplitude* signals, they should not be amplified by using a track gain setting greater than 100%.

During playback, normal digital audio signals, such as speech, rarely use the entire available dynamic range and may need to be digitally amplified. Generated tones, however, are always precisely full-scale to make maximum use of the dynamic range.

The Track Gain setting is most useful for *decreasing* the output volume of a generated tone. Tones are always full-scale, so they will tend to dominate or distort the audio output when mixed with other audio tracks. An acceptable signal can be obtained by setting the Track Gain lower than 100% for the ToneGen tracks.

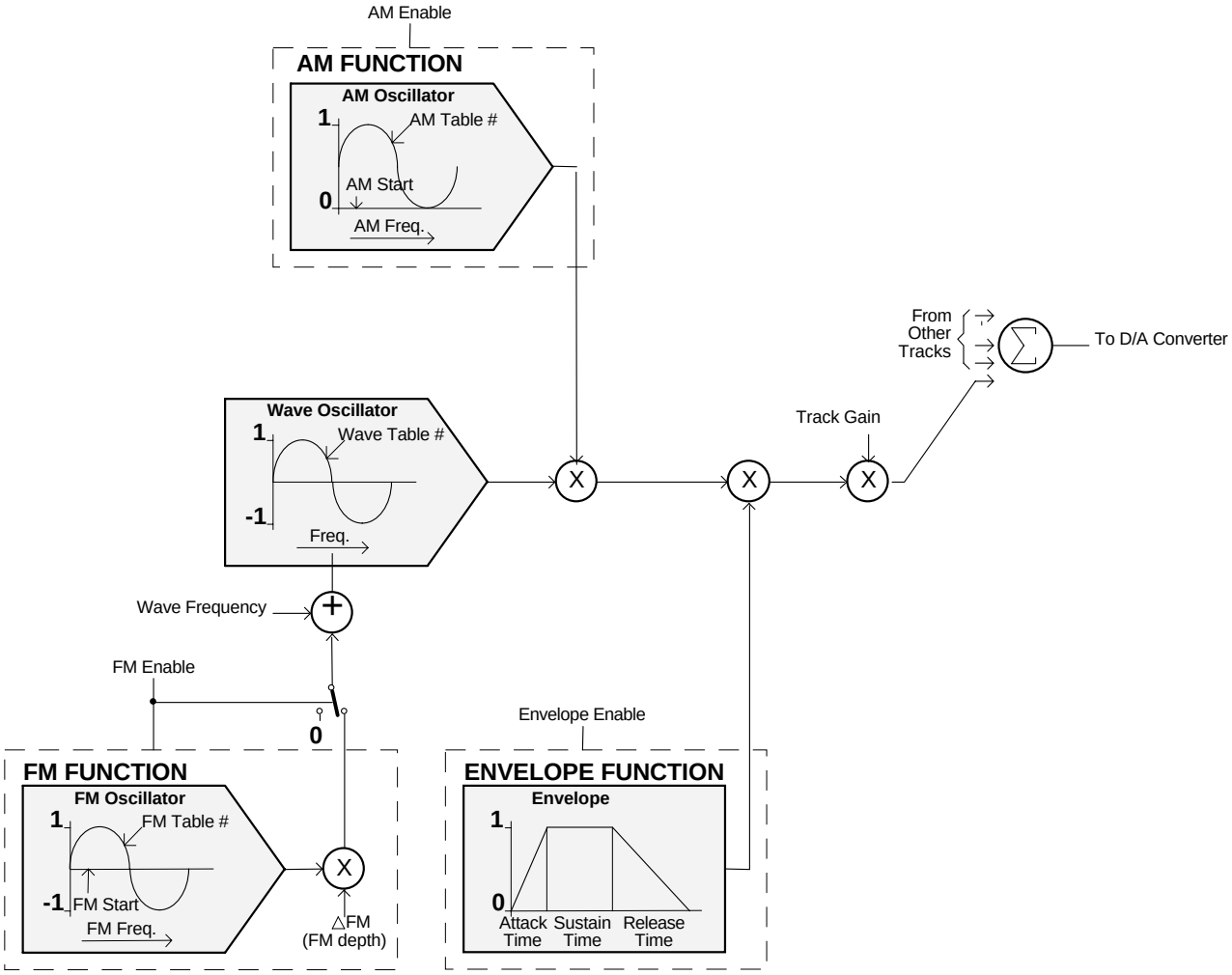
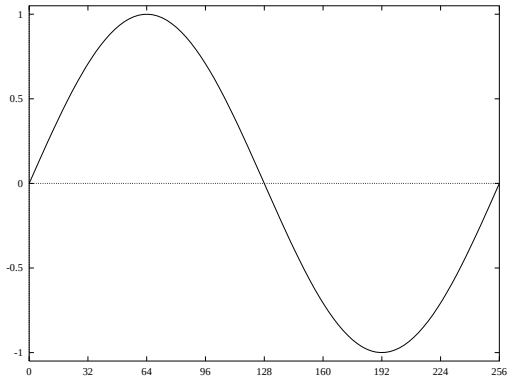
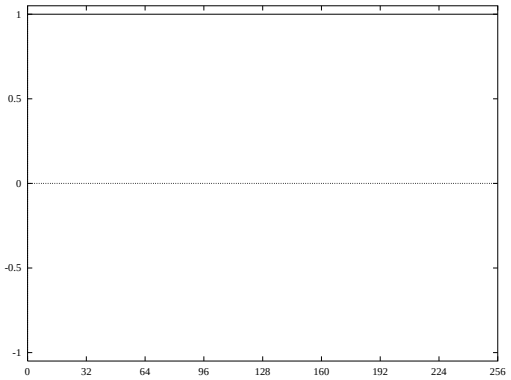


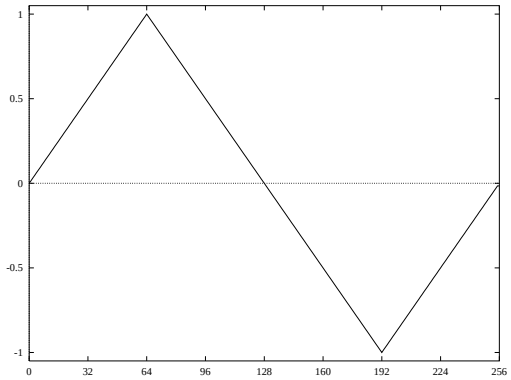
Figure 4.1: Tone Generation Block Diagram



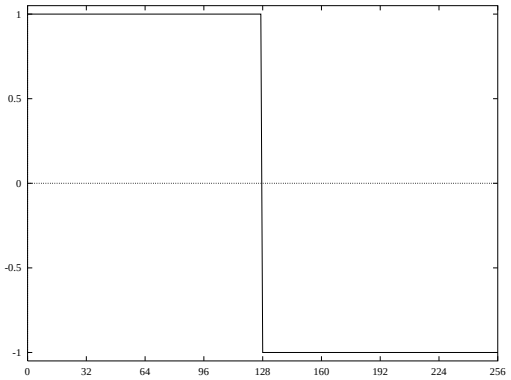
Wave 0: Sine



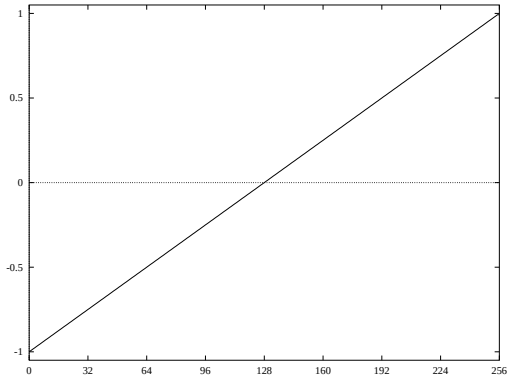
Wave 1: Flat



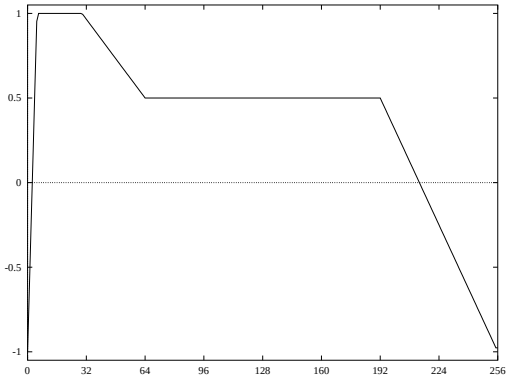
Wave 2: Triangle



Wave 3: Square



Wave 4: Sawtooth



Wave 5: User Defined

Figure 4.2: Default Table Waveforms

5. BANDWIDTH CONSTRAINTS

Tone generation places heavy computation demands on the on-board DSP. The complexity and number of tones is limited by the available DSP bandwidth. By enabling only those parts of tone generation that the application requires, the host can effectively minimize the load on the DSP.

For example, at a sample rate of 32000 Hz, the DSP can generate three simultaneous tones, all with AM, FM, and Envelope enabled. At the same sample rate, the DSP can generate 12 simultaneous tones if AM, FM, and Envelope are all disabled. With only the Envelope enabled, up to 9 tones can be generated by one DSP.

To get the maximum available bandwidth for tone generation, the host may halt recording by issuing the PAUSE_REC command. This will reduce the CPU overhead for record processing to zero, leaving the maximum cycles available for tone generation.

In general, the CPU bandwidth required is directly proportional to the sampling rate of the generated tone. Reducing the sample rate increases the available bandwidth for tone generation, allowing more tones. The actual frequency of the tone oscillators have no impact on the DSP load.

Figure 5.1 shows some approximations of the required CPU bandwidth for each of the tone generation modules at a sample rate of 32000 Hz. These are only guidelines and may vary considerably. The most accurate way to check the DSP load is to use the QUERY_LOAD command while generating tones. This will report the actual load incurred for specific tones.

Function	Load Factor
Playback CPU overhead	5.5%
Each active tone	7.4%
Each enabled envelope	3.1%
Each enabled AM	7.8%
Each enabled FM	8.6%

Figure 5.1: Tone Generation DSP Load

6. TONESHOP INTERACTIVE EDITOR

While the tone generation model and parameters could be explained though exhaustive mathematics and formulas, the quickest and most effective way to learn how tone generation behaves it to experiment with the parameters and listen to the resulting tones.

To facilitate this exploration, an interactive tone editor called ToneShop is included with the DiSPATCH software distribution. This program allows the user to edit tone parameters and hear the resulting tone immediately. When an acceptable tone has been developed, the tone parameter values can be saved to a file for use in an application.

6.1 Running the ToneShop program

An executable binary for ToneShop is included with the SunOS and SGI Irix distributions of DiSPATCH. The DiSPATCH device driver must be installed before ToneShop will run.

ToneShop uses the `curses(3)` terminal interface library to manipulate the screen display and cursor, so it is important that the proper terminal settings be in effect. In general, if the editor `vi` can run correctly, so can ToneShop.

The general command line for `toneshop` is as follows:
`toneshop <sample rate> <MMI Model> <device name> <channel>`

The default values for any unspecified command arguments are shown in Figure 6.1. These defaults can be changed by editing and recompiling `toneshop.c`.

6.2 Editing with ToneShop

Upon running `toneshop`, the program will attempt to initialize the MMI board and start the DiSPATCH firmware. If any errors are encountered, the program will exit with an error message. If the initialization is successful, the screen shown

Sample Rate	32000
MMI Model	MMI-4210
Device Name	/dev/mmidsp0
Channel	0

Figure 6.1: ToneShop default arguments.

in Figure 6.2 will appear.

This is the tone editing and display screen. To hear the current tone, press the P key.

6.2.1 Cursor Motion

At any time, there is one active field on the ToneShop screen. This field shows the value that is currently being edited. The active field is shown in bold or inverse video to highlight the value. This active field can be moved from field to field like a typical screen cursor.

To move the cursor left, right, up or down, any of the available cursor motion keys can be used. If your terminal supports cursor arrow keys, ToneShop will attempt to recognize them. The common Emacs cursor motion control keys are also accepted. For veteran Unix users, the standard `vi` cursor keys (H, L, K, and J) will work as well.

To clear and redraw the screen, the user can press Control-L at any time. This will correct any line noise or formatting errors.

6.2.2 Entering Values

To enter a new parameter value, simply type in the new number and press Return. Only positive values are allowed in all fields.

To add or subtract 1.0 from a value, use the + and - keys. These will also cycle forward and backward through table values and flags. The spacebar is equivalent to the + key.

Due to screen space constraints, the numeric values are only displayed to the

```
==============================<([ ToneShop ])>==============================

                   Frequency:  1000.00      Table: SINE       Count:   1
 AM Enable:  OFF   AM Freq:       5.00   AM Table: TRIAN   AM Start: ---
 FM Enable:  OFF   FM Freq:       2.00   FM Table: SAW     FM Start: ---
                   FM Depth:    200.00

 Env Enable: ON    Attack:        0.10    Sustain:   3.00   Release:   1.90

============================================================================

 Keys: [+]     Increment by 1          [-]  Decrement by 1
       [Q]     Quit ToneShop           [S]  Save tone parameters
       [SPACE] Toggle ON/OFF           [P]  Play current tone
       [^P]    Cursor up               [^N] Cursor down
       [^B]    Cursor left             [^F] Cursor right
       [0-9.]  Enter value             [^L] Redraw screen

       ARROW keys move cursor

                                                        $Revision: 1.12 $
```

Figure 6.2: Example ToneShop display.

hundredth place. Additional precision is maintained, but not shown.

6.3 Saving Tone Parameters

At any time, the current set of tone parameters can be saved to a file by pressing the S key. An example tone output file is shown in Figure 6.3.

The output format is intended to be similar to what may be required in a C program. Simply insert or `#include` the tone block and replace the reference to `YOUR_TONE` with the name of your tone structure. Of course, the output format can be modified by editing the `toneshop.c` source file.

```
/* ToneShop DiSPATCH tone parameter save output */
{
  struct mmi_tone *tone = &YOUR_TONE;

  tone->wave_freq = 1000.000000;
  tone->wave_table = 0;

  tone->am_enable = 0;
  tone->am_table = 2;
  tone->am_start_point = 65535;
  tone->am_freq = 5.000000;

  tone->count = 1;
  tone->envelope_enable = 1;
  tone->attack_time = 0.100000;
  tone->sustain_time = 3.000000;
  tone->release_time = 1.900000;

  tone->fm_enable = 0;
  tone->fm_table = 4;
  tone->fm_start_point = 65535;
  tone->fm_freq = 2.000000;
  tone->fm_delta_freq = 200.000000;
}
```

Figure 6.3: Example tone save file.

6.4 Exiting ToneShop

To exit ToneShop, press the Q key. This will first halt the DSP, silencing any tone in progress. The ToneShop program will then exit.

7. LIBRARY INTERFACE

The DiSPATCH `TONEGEN` command requires that many of the parameter arguments be given in a unique format for efficient processing by the DSP. This format is rarely convenient for the application programmer, so a higher-level tone generation interface was added to the DiSPATCH library.

7.1 Using the `mmi_play_tone()` Function

The library interface uses a single C structure to store all tone parameters. The user application fills in the fields of this structure, and passes it into the library by calling `mmi_play_tone()`. The library then does the necessary translations and executes the DiSPATCH `TONEGEN` command.

The parameter structure (defined in `dispatch.h`) is shown in Figure 7.1. The fields are defined as follows:

`double wave_freq`
: The base frequency of the waveform oscillator, in Hertz.

`int wave_table`
: The wave table number to use for the waveform oscillator (0–5).

`int am_enable`
: Value 1 enables the Amplitude Modulator, while 0 disables all AM.

`int am_table`
: The wave table number to use for the AM oscillator (0–5).

`int am_start_point`
: The starting position in the AM wave table (0–255), or 65535 to keep the current position

`double am_freq`
: The frequency of the AM oscillator, in Hertz.

`int count`
: The number of times to repeat the tone, or –1 to repeat forever.

`int envelope_enable`
: Set to 1 to enable the linear amplitude envelope, or 0 to disable it.

`double attack_time`
: The length of time (in seconds) to spend in the attack phase of the envelope.

`double sustain_time`
: The length of time (in seconds) to spend in the sustain phase of the envelope.

`double release_time`
: The length of time (in seconds) to spend in the release phase of the envelope

`int fm_enable`
: Set to 1 to enable Frequency Modulation, or 0 to disable FM.

`int fm_table`
: The wave table number to use for the FM oscillator (0–5).

`int fm_start_point`
: The starting position in the FM wave table (0–255), or 65535 to keep the current position

`double fm_freq`
: The frequency of the FM oscillator, in Hertz.

`double fm_delta_freq`
: The depth of FM modulation, in Hertz. The wave oscillator frequency will swing from (`wave_freq` + `fm_delta_freq`) to (`wave_freq` – `fm_delta_freq`).

The library interface to tone generation is via the `mmi_play_tone()` function, defined as follows:

```
int
mmi_play_tone (dsp, track, srate, tone)
     dsp_t dsp;
     int track, srate;
     mmi_tone_t tone;
{
  /* .... */
}
```

The track argument specifies which track will receive the new tone parameters. This track must be previously set to the FORMAT_TONEGEN audio format by the use of the mmi_set_play_format() function.

The srate argument must be the current sample rate for the specified DSP. This is necessary because many of the tone generation parameters are dependent on the current sample rate. The mmi_play_tone() function uses this sample rate to adjust those parameters.

The tone argument must be a valid pointer to an mmi_tone structure, described above.

The value returned by the mmi_play_tone() function is the DRAM offset of the TONEGEN command. This command address will be sent by the DSP in a Buffer Completion message when the tone finishes. A value of -1 will be returned if any invalid tone parameters are detected. This mechanism is similar to that of the mmi_play_buf() function.

The mmi_play_tone() function returns immediately after starting the tone. If necessary, the application can then call mmi_complete() to wait until the tone finishes.

7.2 Using the mmi_play_tone() Function

To generate a tone to DRAM instead of playing it, the application must call mmi_play_tone() to set the tone parameters, and then mmi_ramtone() to generate the tone data.

In order to generated a complete tone to DRAM, the selected track for both function calls must be one that is *not* currently selected for the FORMAT_TONEGEN audio data format. This is to prevent the tone from being played immediately after setting the parameters. When the application sets tone parameters for a track that does not have FORMAT_TONEGEN selected, the tone parameters get stored but not played. The host can then switch to FORMAT_TONEGEN to play the tone, or call mmi_ramtone() instead to generate the tone data into DRAM.

Note that all active playback and recording are *paused* while the tone generation is in progress. The call to mmi_ramtone() will not return until the requested tone buffer is complete. This is similar to the behavior of the mmi_transform() function.

For details on the arguments and calling convention of mmi_ramtone, consult the reference portion of the *Host Support Software Manual*.

7.3 Example Program (sampletones.c)

A small example program for generating tones with the DiSPATCH library is included as `sampletones.c`. This code shows how to create many common tones and can provide a useful example to the application programmer.

```
/*
 * This structure fully describes a synthesized Tone, as
 * generated by the CMD_TONEGEN command of DiSPATCH.
 * These members get translated by mmi_play_tone() into low-level
 * argument values for CMD_TONEGEN.
 */
struct mmi_tone {
  double wave_freq;
  int wave_table;

  int am_enable;
  int am_table;
  int am_start_point;
  double am_freq;

  int count;
  int envelope_enable;
  double attack_time;
  double sustain_time;
  double release_time;

  int fm_enable;
  int fm_table;
  int fm_start_point;
  double fm_freq;
  double fm_delta_freq;
};

typedef struct mmi_tone *mmi_tone_t;
```

Figure 7.1: Definition of `struct mmi_tone` from `dispatch.h`