# An Interactive Workbench for Sound Experimentors

Vol. I

Jeff Boone

1991

# An Interactive Workbench for Sound Experimentors

**Jeff Boone**

A Research Paper submitted to Oregon State University in partial fulfillment of the requirements for the degree of Master of Science

Presented March 1, 1991

**Abstract**

*This project is an interactive environment which allows the user to explore various sound generation techniques. The three methods of sound generation used are (1) FM synthesis, (2) Additive synthesis, and (3) Karplus/Strong synthesis of plucked strings. The project also provides the capability to display the generated sound waveforms and analyze the spectrum of the generated sounds. This document describes in detail the theory behind each of the sound generation techniques and details of how they were implemented on the Macintosh™ computer.*

---

Macintosh is a trademark of Apple Computer, Inc.

# Table of Contents

# 1. Introduction

As stated by Scaletti and Johnson[6], many of the early attempts at computer–based interactive sound synthesis suffered from dreadfully slow turnaround times measured in hours or even days. This is due mainly to the bulk of data required to accurately reproduce a sound. Recording or synthesizing frequencies that span the human hearing range (20Hz - 20kHz) requires a sampling rate of 40kHz. Using 16 bit samples, one minute of sound requires 4.8 million bytes; for stereo, the figure doubles. One way around the problem is to implement the sound synthesis algorithms in hardware such as in a synthesizer. This allows real time sound synthesis at the cost of flexibility.

This system is designed around a high speed DSP (Digital Signal Processing) chip and takes advantage of the inherent sound processing power built into it. The result is a flexible system allowing the sound experimentalist real time interaction with the sound.

Another problem with synthesis systems is the cumbersome task of entering and changing parameters to create different timbres. When done by hand, this can be a very tedious and error–prone chore. This interactive tool uses an intuitive graphical representation of the parameters which the user can directly manipulate. This both reduces the manual typing while providing a visual representation of the numerical values.

Synthesis and playback of a sound are only part of the picture. To be able to analyze the synthesized sound the experimenter has the option of viewing the sound in the time domain or the frequency domain. In the time domain the experimenter has the ability to view a graphical representation of the sound samples. In the frequency domain, the system allows the user to "playback" the spectrum of the sound. This is achieved by presenting the user with "tape–deck like" controls which provide the means to navigate through the sound. More information on this can be found in section 6 of this paper.

Sections 2-4 of this paper deal with the synthesis techniques used in the project, section 5 provides insight to the signal analysis tools, section 6 deals with the user interface design issues and section 7 discusses the custom C++ classes that were designed for this project.

# 2. Additive Synthesis

## 2.1 Theory

Additive synthesis, as the name implies, consists of adding together multiple signals to form the output. The "primitive" signals that are added together are normally simple periodic waves such as sine, triangle, square, or saw–tooth. This unit generator is fundamental to almost all computer sound synthesis and is called an oscillator. The parameters that can be sent to an oscillator determine its amplitude, frequency, and waveform. To generate sounds that resemble naturally occurring ones, each oscillator must have separate time-varying functions for frequency and amplitude. The outputs of each of the oscillators are then summed to form the complete signal. This is depicted in figure 1.

Additive synthesis is a very flexible and powerful tool in that, given enough oscillators, any set of spectral components can then be synthesized to form any sound. The drawback of additive synthesis is that it is hard to choose parameters that produce a desired sound. One way around the problem is to sample (digitally record) a sound that is close to what is desired, analyze the sound using spectrum analysis tools, "resynthesize" the sound using additive synthesis, and then experiment by changing the parameters slightly. It may be helpful to have a knowledge of the behavior of functions that describe natural sounds. For example, in many acoustic instruments the higher harmonics attack last

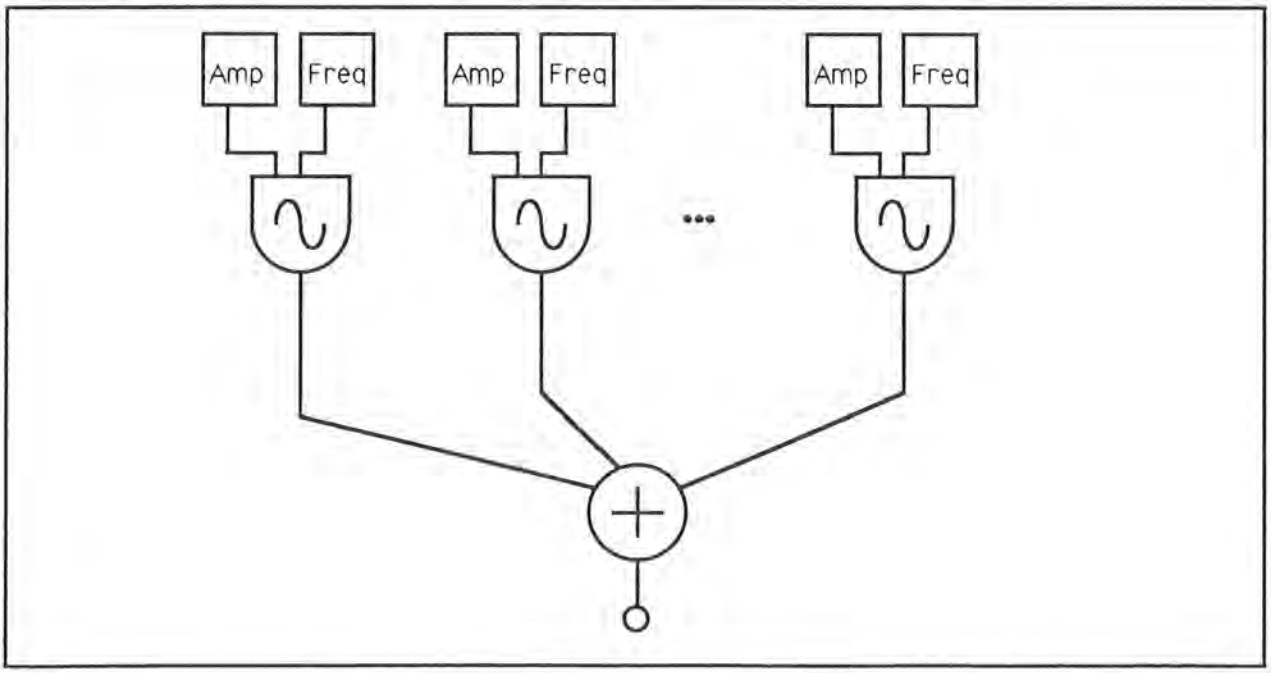



**Figure 1.** Additive Synthesis Diagram.

and decay first. Further information on characteristics of musical instruments can be found in Grey [4].

One further drawback of additive synthesis is the tremendous amount of computation to describe the sound, since each of the oscillators require two separate time–variant functions (frequency and amplitude). Another complication is that once the desired timbre has been synthesized, it is normally useful only for a small range of pitch and volume. For example, if an experimenter sampled the sound of a piano and resynthesized it using additive synthesis, the synthesized sound would sound authentic only within a small frequency deviation from the sampled sound. Playing the sound at a louder volume would not change the dynamics of the sound, but would give the listener the illusion that the piano was closer.

## 2.2 Implementation

This system provides the user with four sine wave oscillators to generate a sound. The user is able to draw an "envelope" (a time–variant function) for both the amplitude and the frequency of the oscillator. When the program is asked to generate a sound using additive synthesis, it loads the appropriate program into the DSP, gives it the parameters that the user has specified, and lets the DSP output the sound samples to the digital-to-analog converter.

On the DSP side, the output of the oscillators is derived from a table lookup algorithm with linear interpolation. The wave table is a 256 entry sine table. The signal/noise ratio (S/N) of the sound is related to the table size. Let k be related to the table size (N) by $k=\log_2 N$. If the entries in the table are stored with sufficient precision to prevent significant quantization noise, the worst S/N ratio that can occur is given by the approximate expressions 12(k-1)dB[3]. Using this formula, the worst case S/N ratio is approximately

12*(log2256 - 1) = 84 dB.

The noise level resulting from a fractional sampling increment varies directly with the amplitude of the signal. Thus, S/N ratio due to this effect is the same on loud sounds as it is on soft sounds.

# 3. FM Synthesis

## 3.1 Theory

In Frequency Modulation (FM) synthesis, the instantaneous frequency of a carrier wave is varied according to a modulating wave, such that the rate at which the carrier varies is the frequency of the modulating wave, or modulating frequency[2]. The amount the carrier varies around its average, or the peak frequency deviation, is proportional to the amplitude of the modulating wave. The parameters of an FM signal are c (carrier frequency or average frequency), m (modulation frequency), and d (peak deviation). The equation for a frequency-modulated wave of peak amplitude A where both the carrier and modulating waves are sinusoids is

$$e = A \sin(\partial t + I \sin ßt) \text{ [2]}$$

where e = the instantaneous amplitude of the modulated carrier, $\partial$ = the carrier frequency in radians/second, ß = the modulating frequency in radians/second, and I = d/m = the modulation index (the ratio of the peak deviation to the modulating frequency). When I = 0 the frequency deviation is also 0 so there is no modulation. When I is greater than 0, however, frequencies occur above and below the carrier frequency at intervals of the modulating frequency. The number of side frequencies which occur is related to the modulation index in such a way that as I increases from 0, energy is "stolen" from the carrier and distributed among an increasing number of side frequencies. This increasing bandwidth as I increases (with a constant modulating frequency) is shown in figure 2 on the next page.

The amplitudes of the carrier and sideband components are determined by Bessel functions of the first kind and the nth order, Jn(I), the argument to which is the modulation index. The zeroth-order Bessel function and index I, J0(I), yields an amplitude scaling coefficient for the carrier frequency; the first-order, J1(I), yields a scaling coefficient for the first upper and lower side frequencies; the second-order, J2(I), for the second upper and lower side frequencies; and so forth. The higher the order of the side frequency the larger the index must be for that side frequency to have a significant amplitude. The first eight Bessel functions are plotted in figure 3. The total bandwidth is approximately equal to twice the sum of the frequency deviation and the modulating frequency, or

Bandwidth $\approx 2(d + m)$.

All of the above relationships are expressed in the following trigonometric expansion:

**e = A{J0(I)sin∂t**
**+ J1(I)[sin(∂ + ß)t - sin(∂ - ß)t]**
**+ J2(I)[sin(∂ + 2ß)t + sin(∂ - 2ß)t]**
**+ J3(I)[sin(∂ + 3ß)t - sin(∂ - 3ß)t]**
**+ ...} [2]**

As can be seen, the complexity of the spectrum is related to the modulation index in such a way that, as the index increases, the bandwidth of the spectrum also increases. If the modulation index were made to be a function of time, the evolution of the bandwidth of the spectrum could be generally described by the shape of the function.

This implementation uses an envelope function on the modulation index to provide an index that

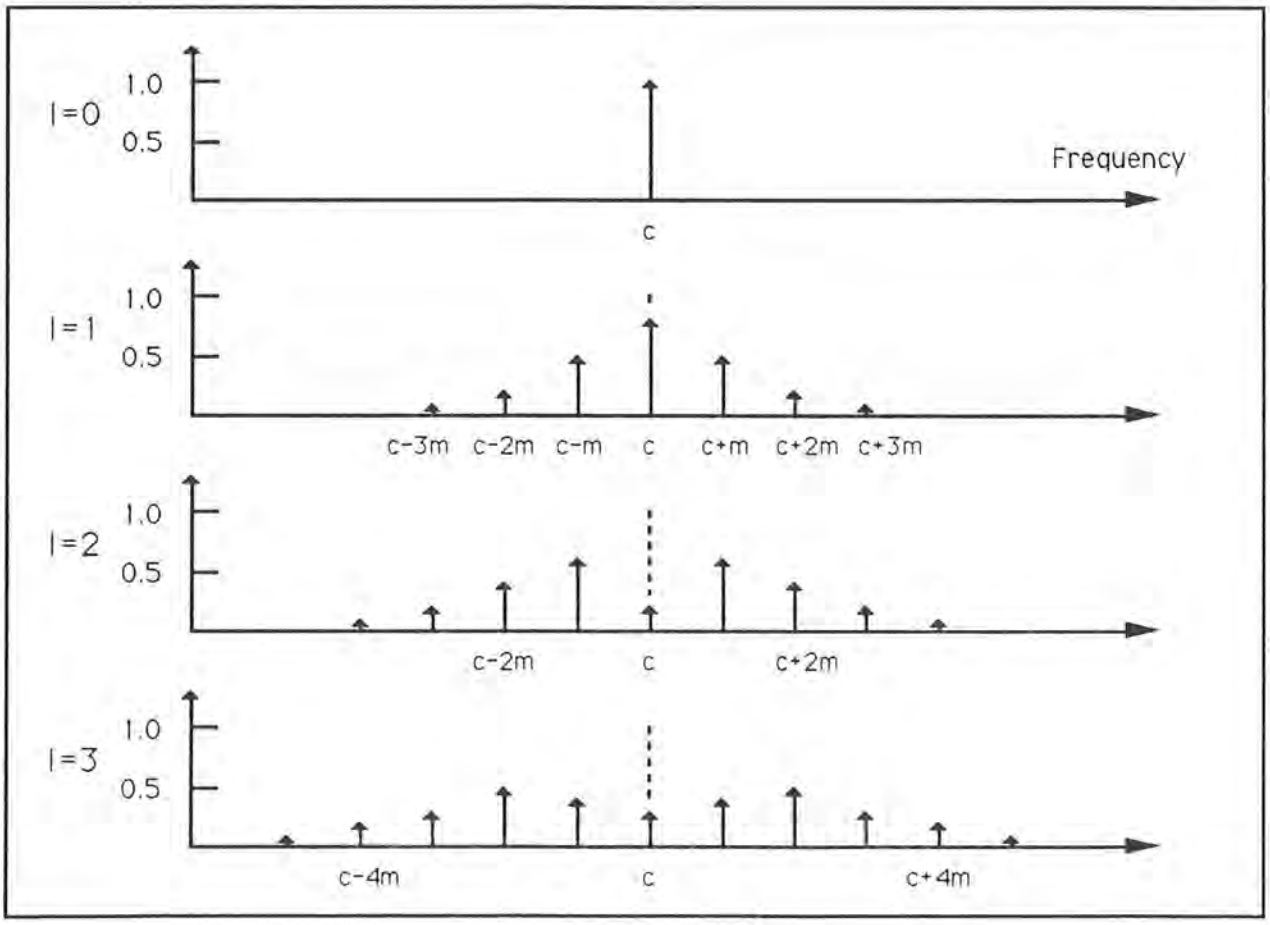



**Figure 2.** Illustration of increasing bandwidth as the modulation index is increased. c=carrier frequency, m=modulation frequency

changes with time providing a rich, time-varying spectrum. The progression of the spectral components with index can be complicated when the effects of the folded negative sideband are taken into account. After examining the shape of the Bessel functions, it is not hard to see that the evolution of an FM signal generally has a certain amount of "ripple" in it. That is, as the index increases, the amplitude of any particular component will not increase smoothly, but instead will alternately increase and decrease, sometimes going to zero. The amount of ripple is somewhat proportional to the maximum value of the modulation index. For example, figure 4 on the next page, plots the time-varying spectrum produced by the instrument in that figure with the parameter values indicated.

Unlike additive synthesis, frequency modulation allows only certain types of spectral evolutions. An effective strategy for using FM to synthesize a timbre is to select the spectral envelope that will realize the desired evolution of the overall richness, the bandwidth, of the spectrum. Because time evolution

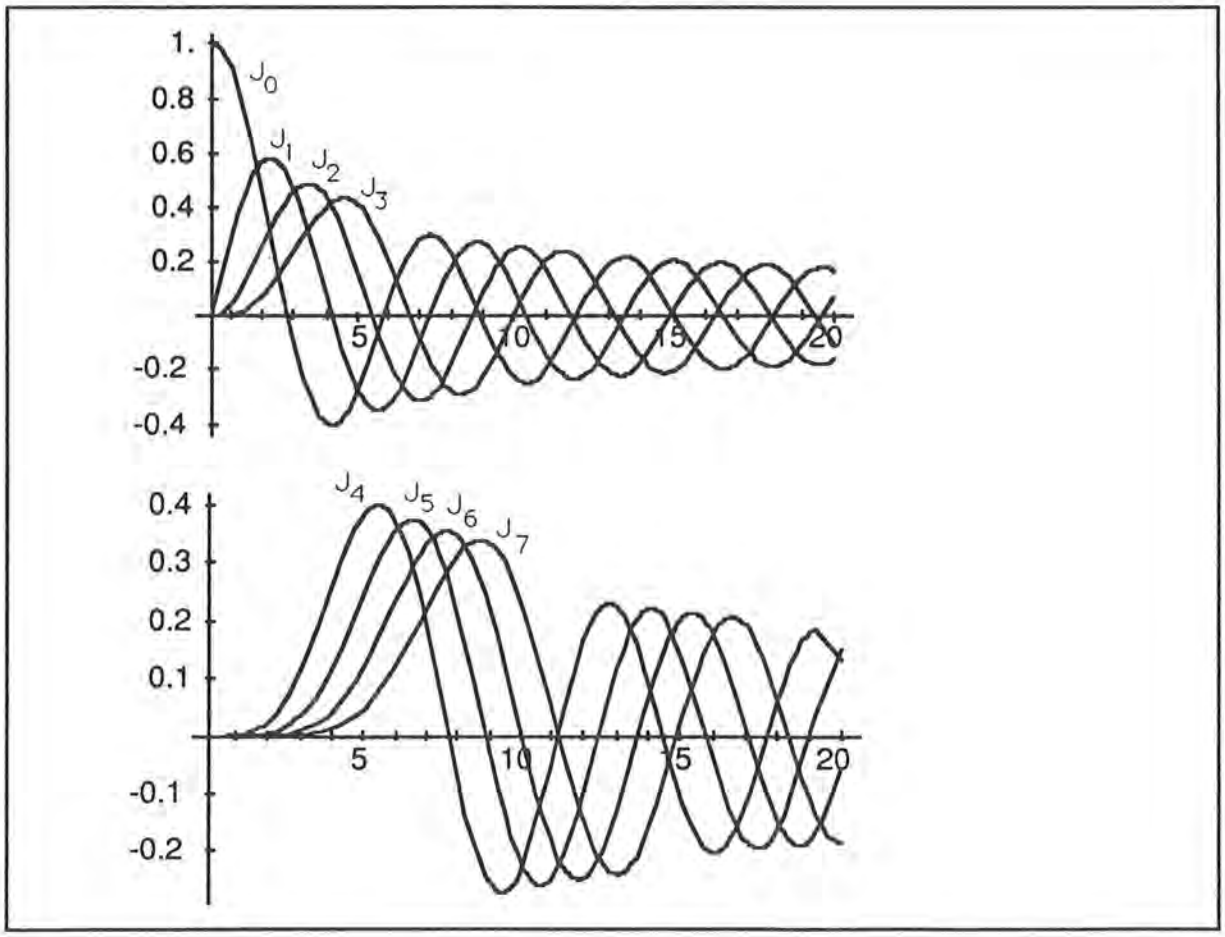



**Figure 3.** Illustration of first eight Bessel functions vs. modulation index

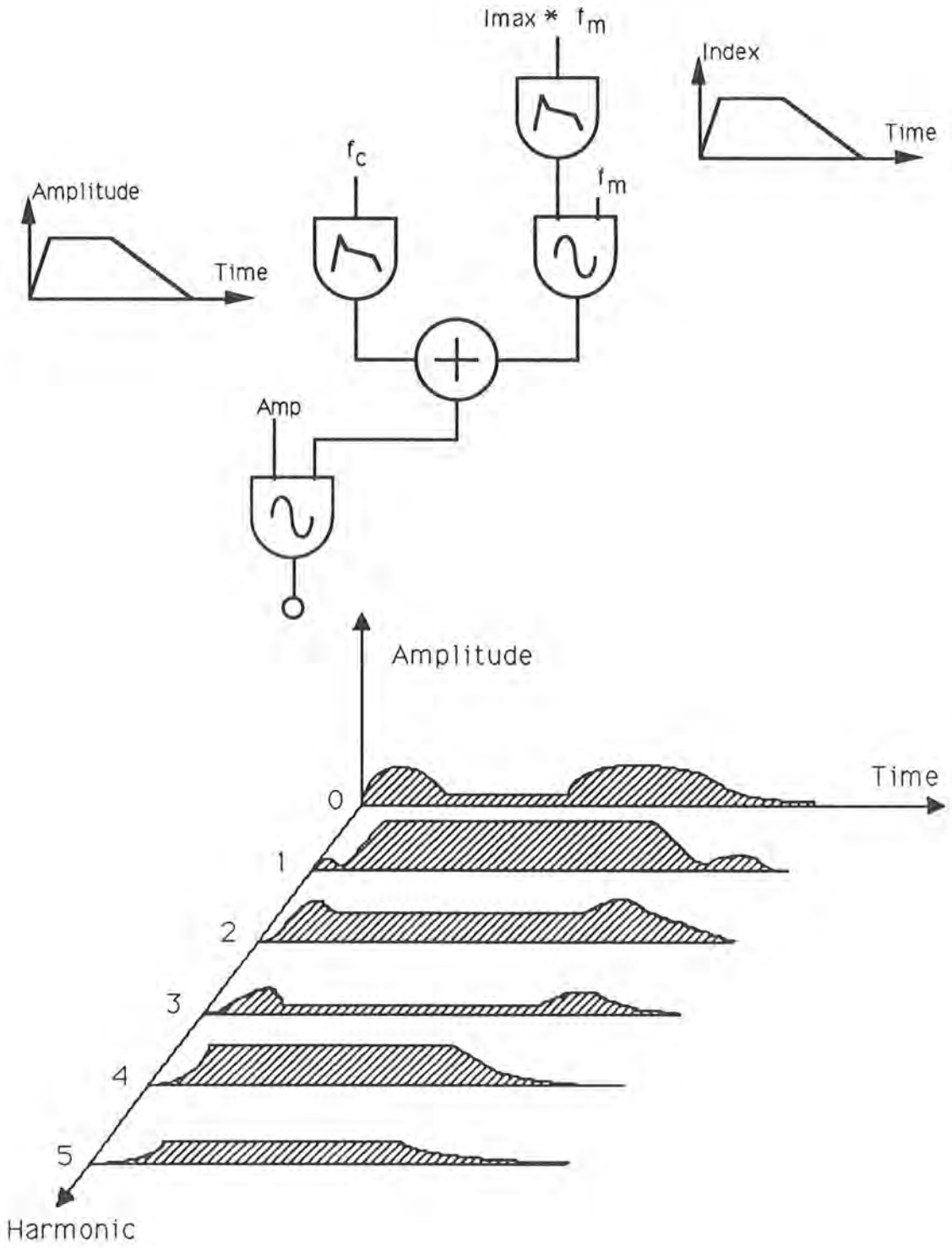



**Figure 4.** Simple FM synthesis instrument, and the dynamic spectrum produced by it. In this example the modulation frequency = carrier frequency.

of the richness of the spectrum is an important element in the perception of timbre, a wide variety of tones can be synthesized by this technique.

The following is an edited excerpt from a note by Marc Le Brun which describes a common bug when implementing FM synthesis with digital oscillators.

Each oscillator of the FM synthesis algorithm is represented by a number PHASE, giving the current phase of the oscillator. The pseudo-code to get the next sample from an oscillator is given as:

```
OSCILLATOR(PHASE, AMP, INCREMENT)
BEGIN
        SIGNAL = SINETABLE[PHASE];
        PHASE = PHASE + INCREMENT;
        RETURN(SIGNAL * AMP);
END;
```

The INCREMENT argument corresponds in the discrete oscillator to the frequency in a continuous oscillator, and it is commonly thought of as the frequency parameter despite the fact that usually some arithmetic must be done, such as multiplying by a constant in order to convert from frequency in Hertz to the sort of increment appropriate for the table lookup. In the following we will assume that this argument can be provided directly in frequency units.

Given the above format it is easy to take an expression, such as

a*sin(w*t),

describing an oscillator with amplitude a and frequency w (t being time), and implement it directly with the expression

OSCILLATOR (PHASE, a, w).

Given Chowning's FM formula, the obvious implementation would use two oscillators, each with a corresponding phase and parameters:

```
FMBUG (CARPHASE, MODPHASE, AMP, MODINDEX, CARINC, MODINC)
BEGIN
        MODSIGNAL = OSCILLATOR(MODPHASE, MODINDEX, MODINC);
        CARSIGNAL = OSCILLATOR(CARPHASE, AMP, CARINC + MODSIGNAL);
```

```
        RETURN(CARSIGNAL);
END;
```

Although this looks quite reasonable, this naive implementation does not compute the above FM expression! Let us look more closely to find the reason. The part of the expression that gives the instantaneous phase, p, of the outer sine (ie, the carrier oscillator) is:

$$p = \partial t + I\sin(\beta t).$$

To convert this phase into the instantaneous frequency, w, we take the derivative with respect to time:

$$w = dp/dt = \partial + I\beta\cos(\beta t).$$

Comparing this with the above routine we find that there is nothing corresponding to the multiplication by ß, the modulating frequency. This means that if we attempt to use the above routine to synthesize an FM signal we wind up getting a modulation index that is effectively 1/ß times what we expect. Thus the bandwidth will be decreased by a factor proportional to the modulating frequency.

Of course, we would expect the spectral profile to remain the same no matter what the modulating frequency was, as long as the index and ratio of carrier to modulating frequency remained the same. Thus this is a bug.

To fix this problem one need simply difference the modulating term with its value from the previous sample:

```
FM (CARPHASE, MODPHASE, AMP, MODINDEX, CARINC, MODINC, OLDMOD)
BEGIN
        MODSIGNAL = OSCILLATOR(MODPHASE, MODINDEX, MODINC);
        CARSIGNAL = OSCILLATOR(CARPHASE, AMP, CARINC+MODSIGNAL-OLDMOD);
        OLDMOD = MODSIGNAL;
        RETURN(CARSIGNAL);
END;
```

Various other methods, such as multiplying by the modulating frequency are both more expensive and may be incorrect when the frequencies are not constant.

It should be noted that we are dealing here with a discrete system, and the use of differentiation in the above discussion is not quite valid. For our purposes here, 1/ß is a sufficiently good approximation

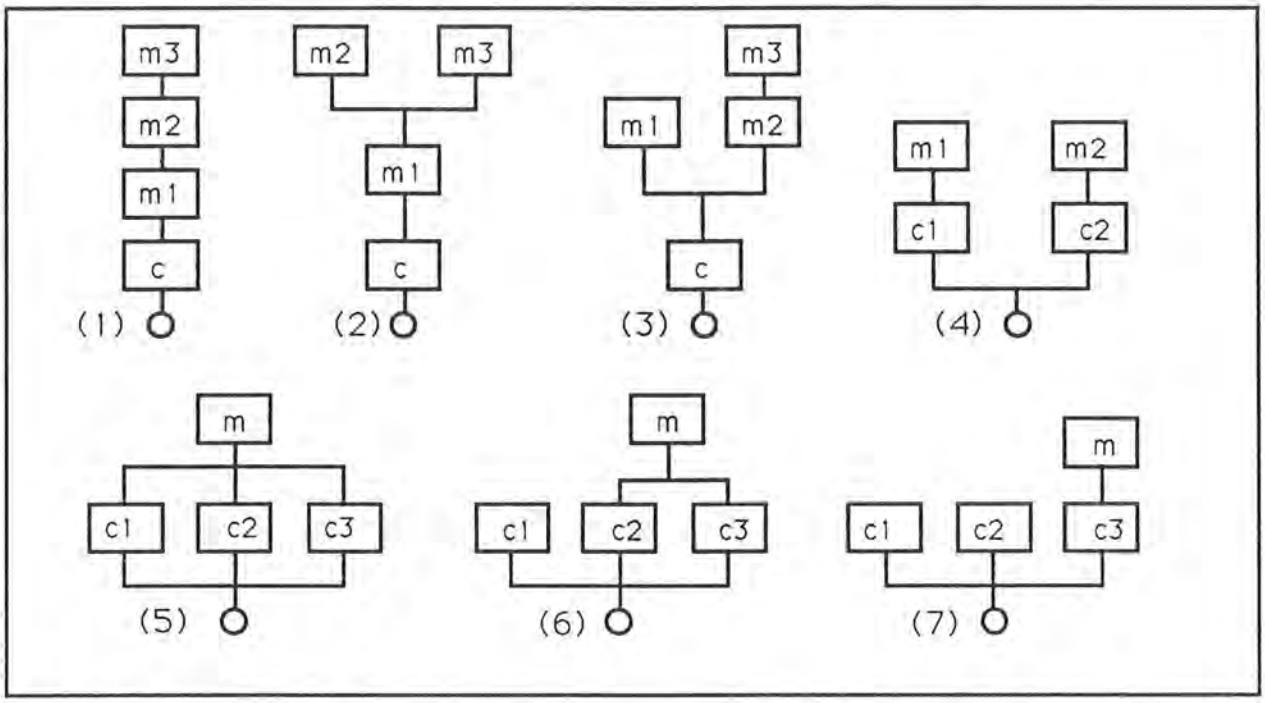



**Figure 5.** Diagram of the 7 FM configurations available to the user in the system. The c's correspond to carrier oscillators and the m's correspond to modulation oscillators.

to the actual quantity.

## 3.2 Implementation

This system allows the user to choose from seven configurations each consisting of four oscillators. The diagrams for these configurations is shown in figure 5. In the diagram, an oscillator that is above and connected to another oscillator modulates the lower one. Oscillators that are on the same level

(1) Asin(c + (I1sin(m1 + I2sin(m2 + I3sin(m3)))))
(2) Asin(c + I1sin(m1 + (I2sin(m2) + I3sin(m3)))))
(3) Asin(c + (I1sin(m1) + (I2sin(m2 + I3sin(m3)))))
(4) (A1sin(c1 + I1sin(m1))) + (A2sin(c2 + I2sin(m2)))
(5) (A1sin(c1 + I1sin(m))) + (A2sin(c2 + I2sin(m))) + (A3sin(c3 + I3sin(m)))
(6) A1sin(c1) + (A2sin(c2 + I1sin(m))) + (A3sin(c3 + I1sin(m)))
(7) A1sin(c1) + A2sin(c2) + (A3sin(c3 + I1sin(m)))

**Table 1.** Formulas for the 7 FM configurations.

and connected have their signals summed. The formulas for the configurations are shown on the previous page in table 1. Note that the A's are amplitude envelopes applied to the carrier oscillators, and the I's are the envelopes applied to the modulation index of the modulating oscillators. These seven configurations allow a wide variety of sounds to be synthesized by specifying only a few parameters.

The user is able to draw the modulation index envelope for modulating oscillators and the amplitude envelope for carrier oscillators. The frequency of the oscillators is also specified by the user. More information on this can be found in section 6.

## 3.3 Future Research

It would be interesting to apply an envelope to the frequency of the carrier and modulation oscillators. One could also study the effects of shifting the phase of the oscillators, or using waveforms other than a sinusoid for either the carrier, the oscillator, or both.

# 4. Karplus/Strong Algorithm for Plucked String Timbres

## 4.1 Theory

The Karplus/Strong algorithm for plucked string timbres is a modification of a standard wavetable synthesis technique. Wavetable synthesis consists of repeating a number of samples over and over thus producing a purely periodic signal. If we let $Y_t$ be the value of the $t^{th}$ sample, the algorithm can be written mathematically as

$$Y_t=Y_{t-p}$$

The parameter p is called the wavetable length or periodicity parameter. It represents the period of the tone (in samples). The initial conditions of the recurrence relation completely determine the resulting timbre. Normally, a sine wave, triangle wave, square wave, or other simple waveform is calculated and loaded into the wavetable before the note is played. With a sampling frequency of $f_s$, the frequency of the tone is $f_s/p$.

Alex Strong invented a technique to change this fixed harmonic content periodic signal into a time-varying harmonic rich sound by averaging two successive samples in the wave table. This can be written mathematically as

$$Y_t=\frac{1}{2}(Y_{t-p}+Y_{t-p-1})$$

It turns out that this averaging process produces a slow decay of the waveform. The resulting tone of this algorithm has a pitch that corresponds to a period of p + 1/2 samples (frequency = $f_s/(p + 1/2)$, and sounds remarkably like the decay of a plucked string[5]. The recurrence can be viewed as a digital filter without inputs as in Figure 6. Initially filling the wavetable with random values creates a spectrum that is rich with harmonics, which is consistent with that of a string sound. With the delay produced by the averaging, the higher harmonics decay rapidly leaving almost a pure sine wave.

## 4.2 Implementation

As suggested by Karplus and Strong I use a two-level randomness in the initial wave table. This produces a signal about 5 dB louder than uniform random numbers, and can be described

mathematically as

$$Y_t = \begin{cases} +A \text{ probability } \frac{1}{2} \\ -A \text{ probability } \frac{1}{2} \end{cases}$$

For the Karplus/Strong algorithm, the system generates a random wavetable length which corresponds to a random pitch. A wave table of the appropriate length is created, two-level random values are loaded into it, and the algorithm generates the number of samples necessary to produce a sound of the desired length.

# 5. Spectrum Analysis

## 5.1 Theory

A signal can be described either by its pattern of amplitude versus time (its waveform or time domain) or by its distribution of energy versus frequency (its spectrum or frequency domain). Either form of this dual representation is sufficient to describe the signal completely. The Fast Fourier Transform (FFT) is the bridge between these two representations. Since we are dealing with sampled data (ie the time domain data are at discrete instants of time) we will be using the discrete Fourier transform. One important feature of the discrete transform is that the data is subdivided into fixed sized blocks called records, each consisting of a number of samples. The transformation assumes that the samples in a record represent exactly one cycle of a periodic signal. This assumption is almost never true; to compensate for this we must choose a record size which is large enough to span several cycles of the lowest sampled frequency. This reduces the error introduced by the partial cycle at the beginning and ending of record. Further, we apply an amplitude envelope to the sampled data to force continuity between the start and end of the record.

Another consideration when choosing a record size is the fact that the FFT returns an average of the spectrum content for the time frame within the record. Therefore, if the sound being analyzed has a rapidly changing spectrum and we use a very large record size, details of the spectrum could be lost. So there is a trade-off between choosing a large record size for good frequency resolution and a smaller record size to follow a rapidly changing spectrum. The problem can be dealt with quite adequately by using overlap between the records of the analysis thus achieving a high spectral sample rate while maintaining a larger record size for better frequency resolution.

The first step in preparation for the FFT is choosing the record size. Since the spectrum is averaged over the duration of the record, the time resolution of the FFT can be no better than the record duration. And since the FFT gives harmonic frequencies that are integral multiples of the reciprocal of the record duration, the frequency resolution can be no better than this either. An audio sampling rate of 24kHz is used throughout the system, so assuming a record size of 256, the frequency resolution is

$$24000 \text{ Hz}/256 = 93.7 \text{ Hz}$$

Using a 2:1 overlap to insure that all parts of the waveform are seen, we have a spectral sample rate of 187.4 Hz, which means we will be able to view 187 different frames of the spectrum for every second of sound. The time resolution for the analysis is given as 1/frequency resolution which is

1/93.7 Hz = 10.7 msec.

When an FFT is used on data that is non–continuous (ie the first sample value is different than the last sample value) the result will contain transform "harmonics" which reduce the validity of analysis. To counteract this, an amplitude envelope is applied to the data prior to the FFT which forces continuity within the record. A "good" envelope to accomplish this will attenuate frequencies far removed from the center frequency by 40 to 80dB while leaving the center frequency relatively unchanged. One such envelope is a Hamming window, which uses the formula

Hamming(X) = .54-.46(cos(2πX)).

The shape of this envelope is shown in figure 6. Note that all frequencies which are farther than 172

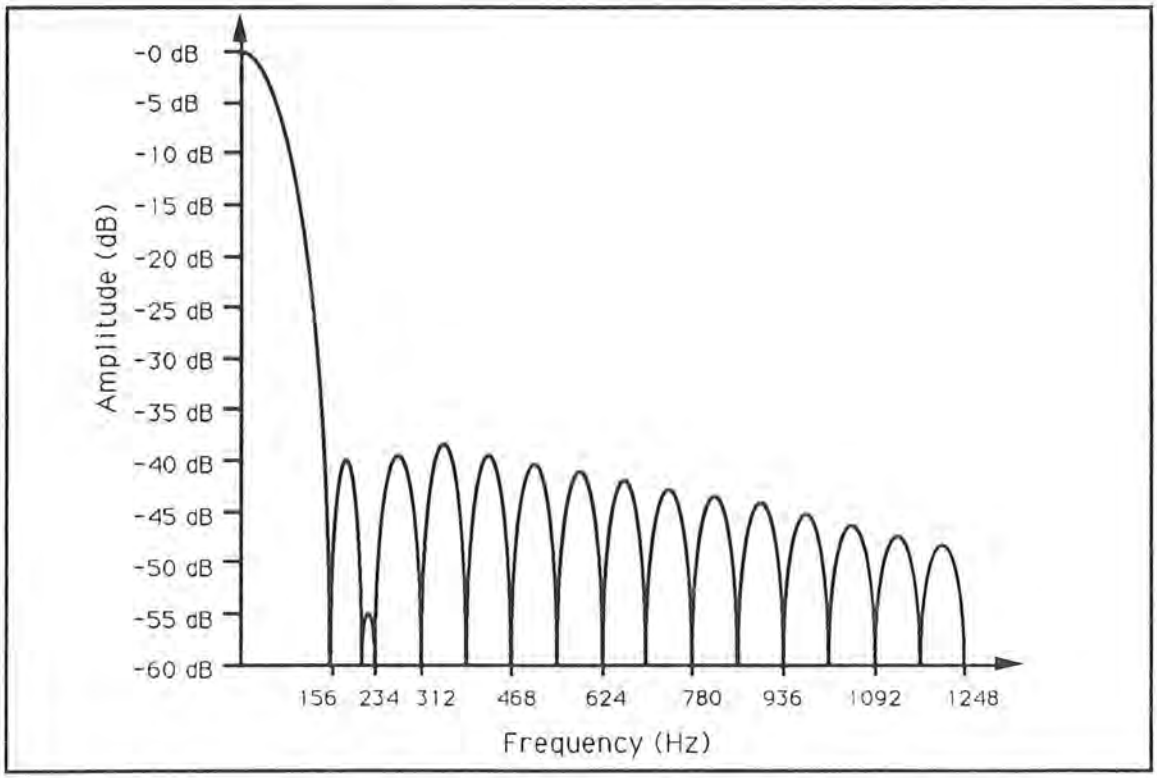



**Figure 6.** Amplitude envelope of the Hamming window used on each of the FFT records. (Hamming(X) = .54-.46(cos(2πX)))

Hz from the center are attenuated by 40dB or more.

## 5.2 Implementation

The analysis tool is probably the most innovative as it allows the user to "playback" the spectrum of the sound. The user is presented with controls similar to a tape deck (play, stop, rewind, etc.) and is able to view the changing spectrum at speeds of approximately 4 frames/second. At any time the user can stop the playback, step through the changing spectrum frame by frame, or use the fast forward or rewind buttons to take them quickly through the sound (skipping intermediate frames). More information about the user interface of the analysis tool can be found in section 6.3.

The analysis is done by maintaining a pointer within the sound data which keeps track of the location of the next analysis record. The 256 data samples are then copied into the another location in memory so the integrity of the original sound will be maintained. The DSP then applies the Hamming amplitude envelope and Fast Fourier Transform to the data. Finally, the FFT results are read by the CPU and displayed on the screen. In this manner both the DSP and the CPU are kept busy with their respective tasks, so maximum throughput is achieved.

# 6. User Interface Design

The purpose of this project was to take these synthesis and analysis algorithms out of the "theoretical" world and place them into the hands of people who have little or no understanding of the underlying mechanics and empower them with the tools to explore the possibilities.

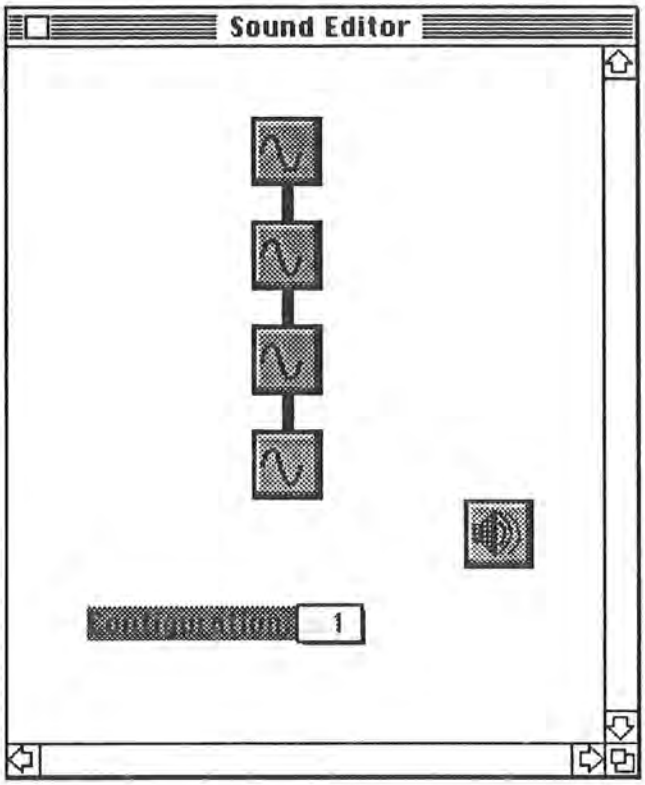
The user interacts with the system through 3 windows; the "Sound Editor" window allows the user to specify the type of synthesis to be used and set the parameters by clicking on the oscillator buttons, the "Sound View" window allows the user to view the resulting sound samples and zoom in for more detail, and the "Analysis" window allows the user to view the spectrum analysis of the sound. The windows are activated via the "Window" menu.

## 6.1 Sound Editor Window

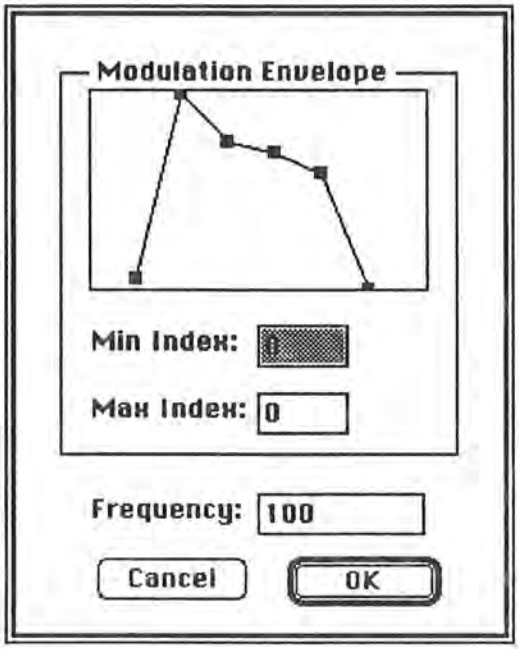
The Sound Editor Window lets the user specify the parameters for the generated sound. In the case of both FM and additive synthesis, four oscillators are graphically represented as buttons. When a user clicks on one of the oscillator buttons a dialog box is displayed which has all the parameters for that oscillator.

### 6.1.1 FM Synthesis

The relative position of the oscillators is significant in that two oscillators aligned vertically have the property of the top oscillator modulating lower oscillator. Two oscillators aligned horizontally will have their outputs summed.




For example, in the previous diagram the oscillators are all stacked vertically. In this case, the output from the top oscillator is used to modulate the oscillator below it, the result is used to modulate the third oscillator, and finally that result is used to modulate the bottom oscillator (or carrier). Clicking on any of the modulating oscillators will bring up the parameter dialog for the oscillator:

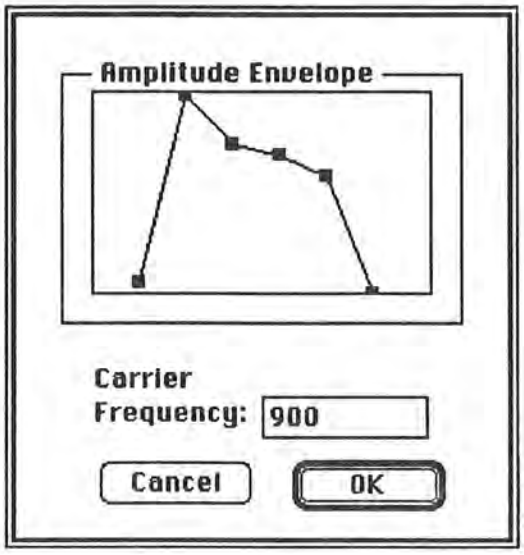



The user is then able to set the frequency and time–variant modulation index for the oscillator. The modulation index at time t is given as:

Index = MinIndex + ((MaxIndex - MinIndex) * Envelope(t))

Where Envelope(t) is defined as a linear function that follows the shape of the modulation envelope in the parameter dialog. The user is able to change the shape of the envelope by dragging the square "handles" up or down in the envelope view window. In this manner, the user is presented with a very intuitive way to interact with the system, providing an order of magnitude of "user–friendliness" over conventional methods of typing in numbers by hand.

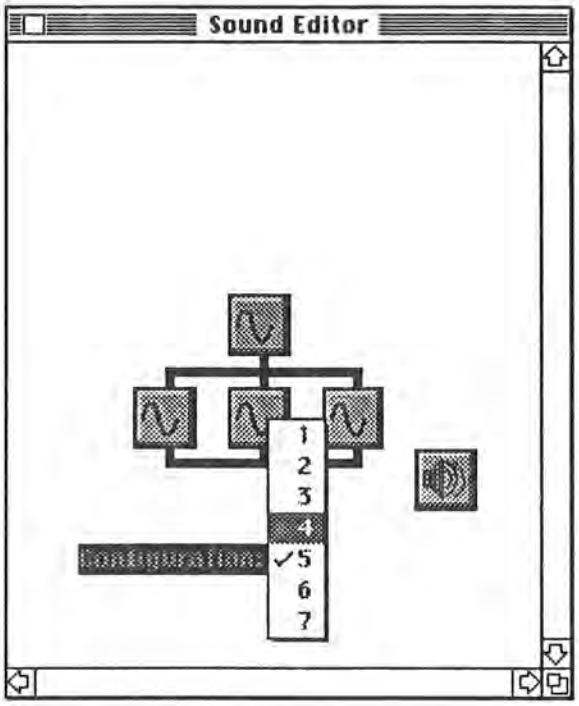
For the carrier oscillator the parameter dialog looks like this:




In this case the envelope corresponds to the output amplitude. The closer to the top of dialog the envelope values are, the louder the output signal will be. In the instance shown above the signal will start out at 0 and rapidly increase to the maximum possible amplitude and then gradually decrease down to 0 again.

The frequency of the oscillator can be typed into the dialog box.

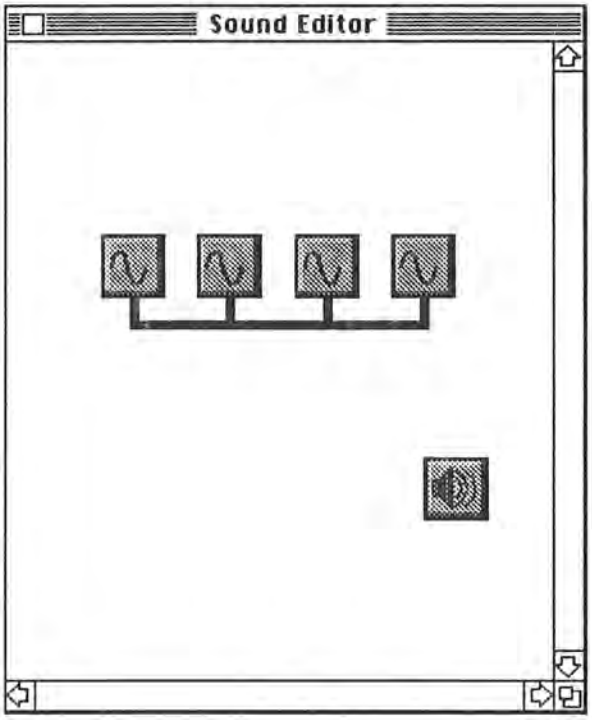
The "Configuration" pop–up menu allows the user to change between seven different configurations of the oscillators.



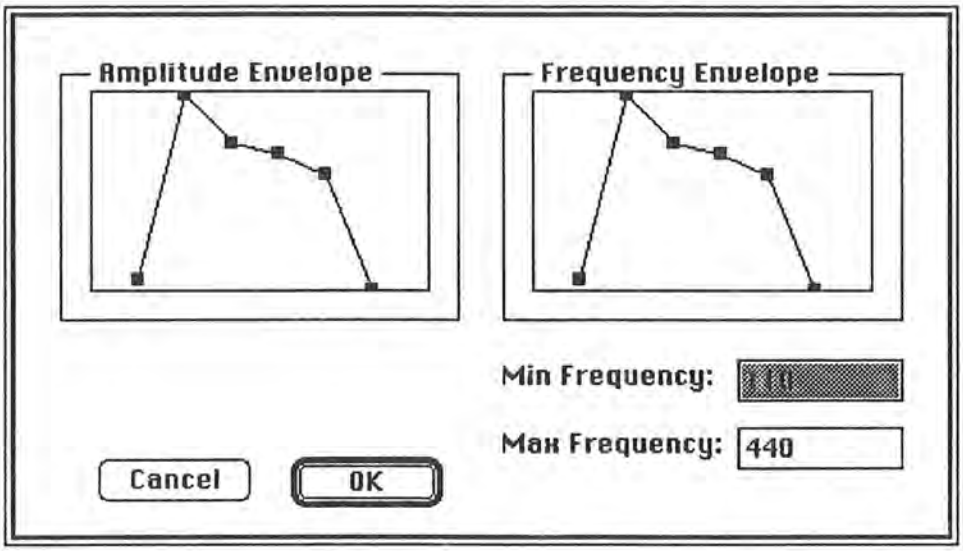

In this case the user is changing from configuration 5 to configuration 4. The different configurations allow the user to experiment with different arrangements of the oscillators. It is very simple to switch between all possible configurations without changing any of the oscillator parameters just to hear how each configuration affects the sound.

### 6.1.2 Additive Synthesis

Choosing "Additive" from the "Synthesis" menu changes the system into additive synthesis mode and changes the Sound Editor window to reflect this.




Note that the "Configuration" pop–up menu disappears since there is only one configuration in which the output of all four oscillators is summed to form the output signal. Clicking on any of the oscillator buttons will bring up the parameter dialog for the oscillator which allows the user to set the time–variant frequency and amplitude envelopes.



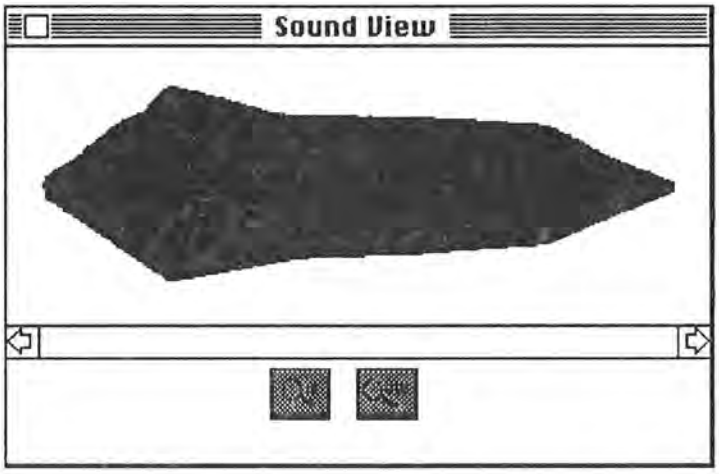

In the case shown above, the frequency of the oscillator will start out at 110Hz (Min Frequency) and quickly increase to 440Hz (Max Frequency) and then will gradually decline back to 110Hz at the end of the sound.

### 6.1.3 Karplus/Strong Synthesis

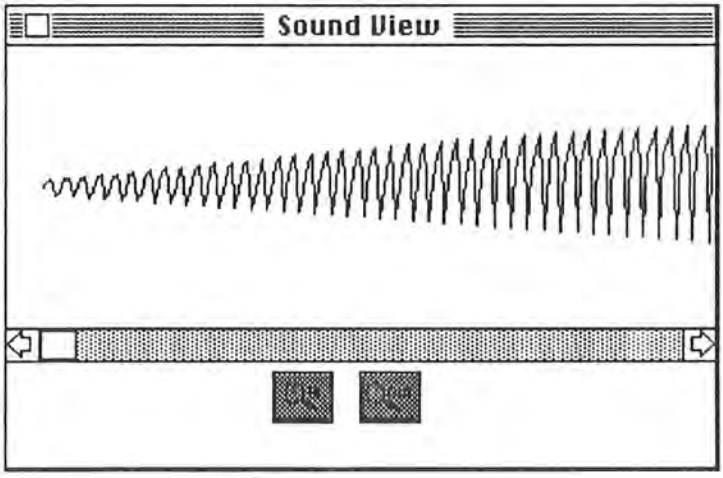
In this system there are no parameters for the Karplus/Strong synthesis and when the user presses the generate button  the system generates a random pitch.

## 6.2 Sound View Window

The Sound View window displays the waveform of the current sound. It is updated automatically when any of the sound generation parameters change.



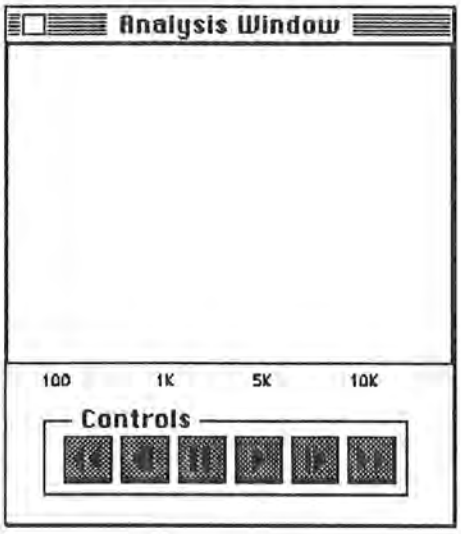

Initially, the waveform is scaled to fit completely inside the window. Two buttons are provided to zoom in and zoom out on the waveform to analyze it in more detail.




The example above shows the same waveform after the "zoom–in" button has been pressed several times.

## 6.3 Analysis Window

The Analysis Window provides a way for the user to view the spectrum analysis as it changes through time. As mentioned before, the user interface is designed around a set of tape deck like controls that allow the user to "play" through the sound and see the frequency information. Using a 256 sample window the system provides a resolution of 93.72 Hz and a 10.67 msec record duration (24kHz sample rate).




The buttons have the following functions:


Rewind: Quickly scan backward through the sound.


Step Back: Back up one frame and pause.

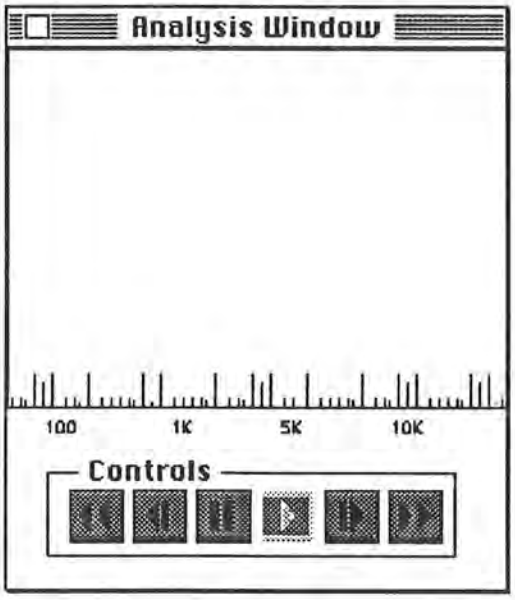

Pause: Halt the play back of the spectrum.


Play: Begin playing the spectrum and update the display as quickly as possible.

Step Forward: Move forward one frame and pause.

Fast Forward: Quickly scan forward through the sound.

When the spectrum is being analyzed the window will be updated with the frequency information.

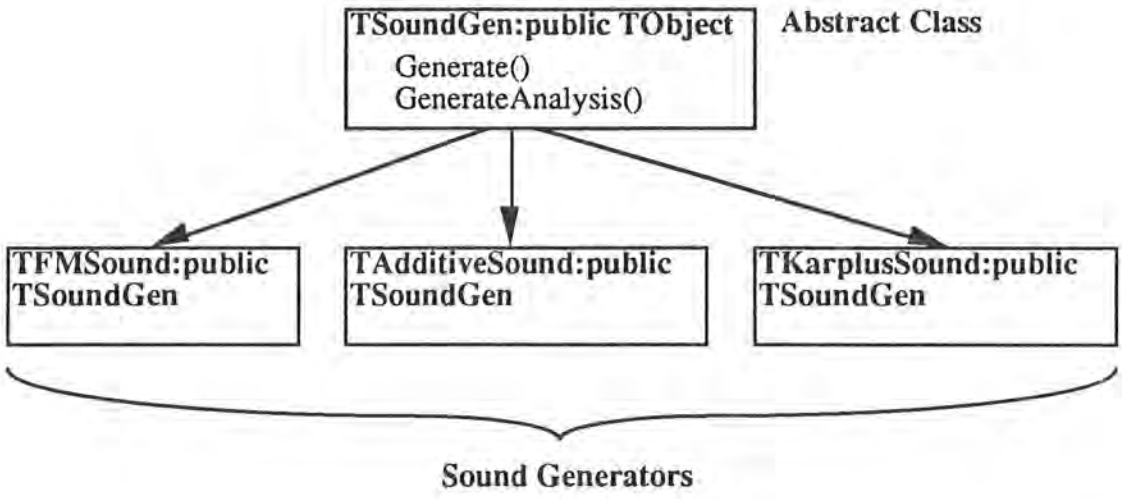



Each vertical line in the display corresponds to a harmonic of the analysis frequency (93.7Hz). The taller the line, the more energy for that particular frequency. This tool is very useful in viewing the effects that the parameters have on the generated sound; in particular, the bandwidth of an FM synthesized sound is proportional to the modulation index and frequency (as mentioned earlier). This can be observed directly in the analysis window.

# 7. Class Design

Since the project was designed using the Object–Oriented programming methodology, it was necessary to design custom classes to provide a flexible and upgradeable structure. Starting with Apple's *MacApp* application class library simplified the standard user interface issues (menus, windows, dialogs, etc.).

The first custom class is *TSoundGen* which is an abstract class from which all sound generators are derived. Using this design, the application has an instance variable (fGenerator) of type TSoundGen* that points to the current sound generator. When the application is required to generate a sound it calls fGenerator->Generate() which will call the appropriate Generate() method depending on the current synthesis mode. Extending the system for new synthesis methods consists of creating a new subclass of TSoundGen and filling in the methods with the proper code.




The next custom class, *TDsp*, was designed to handle all interaction with the DSP. The DSP requires that certain structures be set up by the application and then passed on to the DSP. The TDsp class includes methods to create and maintain these shared data structures. Other support classes were designed for the project and can be viewed in the source code.

# 8. Conclusion

As stated before, the focus of the project was to put a "wrapper" around some difficult sound synthesis techniques and provide a responsive environment for a novice user to experiment. The DSP has provided the processing power needed to achieve this and a user interface designed around the Macintosh user interface guidelines invites the user to explore. The goal was to achieve an easy way to test and experiment with theories about the synthesis methods. With this system it is very simple, for example, to take Chowning's formulas for generating particular timbres with FM synthesis and try them out. There no longer exists the need to hand type in the parameters or wait for the computer to generate a disk file to be sent to the D/A converter. All operations are done in real–time with instantaneous feedback.

The project has given me the opportunity to explore and extend my knowledge of many areas within the computer science discipline:

**Real–time processing**
**Multi–processor systems**
**Object oriented design**
**Shared data structures**
**Pipelined processor architectures**

All "multi–media" applications developers (incorporating sound, video, etc.) must face similar challenges since the typical micro computer structure of one CPU cannot handle the requirements for real–time systems; many computer manufacturers are incorporating high speed auxiliary processors to handle the real–time tasks while relying on the CPU for the "traditional" tasks.

# References


[1] Chamberlin, Hal. 1980. *Musical Applications of Microprocessors*. New Jersey: Hayden.

[2] Chowning, John M. 1973. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation." *Journal of the Audio Engineering Society* 21(7).

[3] Dodge, Charles, and Jerse, Thomas A. 1985. *Computer Music*. New York: Schirmer.

[4] Grey, John. "An Exploration of Musical Timbre." Doctoral dissertation, Stanford University, 1975.

[5] Karplus, Kevin, and Strong, Alex. 1983. "Digital Synthesis of Plucked–String and Drum Timbres." *Computer Music Journal* 7(2).

[6] Scaletti, C.A., and Johnson, R.E. 1988. "An Interactive Environment for Object–oriented Music Composition and Sound Synthesis." *OOPSLA '88 Proceedings* September 25-30, 1988.

# Appendix A – Source Code

# MSound.cp

```
//========================================================================
//
// NAME
//      MSound.cp
//
// SYNOPSIS
//      This file implements the main program for the Sound project.
//      It initializes the tool box and MacApp, it then creates the
//      global application object and tells it to run.
//
//      Author Jeff Boone, Oregon State University
//
//
//========================================================================


//============================ Main Program ====================================

#include    "USound.h"

    const           SPLASH_ID = 1006;           // Resource ID of the splash screen
    DialogPtr       theDialog;


#pragma segment Main

void main()
{
    InitToolBox();                  // Essential toolbox and utilities

    PullApplicationToFront();
    if (ValidateConfiguration(&gConfiguration))     // Make sure we can run
    {
        theDialog = GetNewCenteredDialog(SPLASH_ID, nil, nil);
        if (theDialog != nil)
            DrawDialog(theDialog);                      // Show splash screen
        InitUMacApp(8);                                 // initialize MacApp; 8 calls to MoreMasters
        InitUDialog();                                  // so TDialogView, TButton, etc. not stripped
        InitUTEView();

        gSoundApplication = new TSoundApplication;      // create an application object
        FailNIL(gSoundApplication);                     // make sure enough memory
        gSoundApplication->ISoundApplication(kFileType);        // initialize the application
        DisposDialog(theDialog);
        gSoundApplication->Run();                       // run the application
    }
    else
        StdAlert(phUnsupportedConfiguration);   // tell user we can't run
}
```

# USound.cp

```
//============================================================================
//
// NAME
//      USound.h
//
// SYNOPSIS
//      This header file defines all the classes, structures and global
//      constants for the project.
//
// COPYRIGHT
//      Author Jeff Boone, Oregon State University
//
//============================================================================

#ifndef __TSOUND__
#define __TSOUND__

#include "SoundDump.h"
#include <Types.h>
#include "dsptaskdispatcher.h"


#define     kSignature          'JDB1'          // Application signature
#define     kFileType           'JDB2'          // File-type code used for document files
                                                //   created by this application

//----------------------------------------------------------------------------
//------------------------   Global Constants     ----------------------------
//----------------------------------------------------------------------------
const short             kEditorWindow       = 1001;             // Resource ID of editor view
const short             kAnalysisWindow     = 1002;             // Resource ID of analysis view
const short             kViewWindow         = 1003;             // Resource ID of sound view
const short             kFMSynthesis        = 1004;             // Resource ID of FM synthesis menu
const short             kKSSynthesis        = 1005;             // Resource ID of K/S synthesis menu
const short             kAddSynthesis       = 1006;             // Resource ID of Additve synthesis menu
const short             kMySaveAs           = 1007;             // Resource ID of Save as… menu
const short             kQuit               = 36;               // Resource ID of Quit item
const short             kAboutApp           = 1;                // Resource ID of About item
const unsigned long     kIDConstant         = 538976305;        // Const that is added to the base identifier
const unsigned long     kViewIt             = 0x00000001;       // "Just viewing" mask
const unsigned long     kHostReady          = 0x00000002;       // "Host ready" mask
const unsigned long     kDoFFT              = 0x00000004;       // "Do an FFT" mask
const unsigned long     kNeedToInit         = 0x00000008;       // "Need to initialize" mask
const short             kDSPParamBlock      = 9;                // Parameter data block
const short             kMaxVoices          = 1;                // Maximum number of voices
const short             kIndex              = 59;               // Parameter number of the index
const short             kReturnValues       = 65;               // Parameter number of the return values
const short             kNumParams          = 58;               // Number of params passed to DSP
const short             kNumSamplesIndex    = 0;                // Parameter number of the number of samples
const short             kBlockSize          = 240;              // Frame size
const short             kFFTSize            = 258;              // FFT Data record (2 flags)
const short             kParamSize          = 320;              // Parameter block size
const short             kFFTRecordSize      = 128;              // Size of 1 FFT record
const short             kNumEnvelopeValues  = 6;                // Number of Envelope values for Osc parameters
const Address           kNilAddress         = ( nil, nil );     // A nil DSP Address
const Boolean           kNow                = true;             // Install task now
const Boolean           kLater              = false;            // Frame synchronous install


    typedef enum {FM, Additive, KarplusStrong} SoundType;
    typedef enum {Modulator, Carrier} FMOpType;
    typedef enum {Play, Stop, StepF, StepB, FF, Rew} AnalysisMode;

    typedef struct {
            float           param[ kParamSize ];
            unsigned long   flags;
        } Parameters;

    typedef struct {
            float           sample[ kFFTSize ];         // 2 Flags + 256 FFT values
        } FFTData;

    typedef struct {
            float           sample[ kBlockSize ];
        } SampleBlock;

    typedef struct {
            int             envValues[ kNumEnvelopeValues ];
```

```
            long            frequency, minEnv, maxEnv;
            FMOpType        opType;
        } FMOperator;

    typedef struct {
            int             ampEnvValues[ kNumEnvelopeValues ], freqEnvValues[ kNumEnvelopeValues ];
            long            minFreq, maxFreq;
        } AdditiveOp;

    void pStrCpy(Str31 outstr, Str31 instr);


// ***************** C L A S S   D E F I N I T I O N S  ***********************

class TDspCodeBlock;        // Forward declarations
class Tdsp;

//-------------------------------------------------------------------------------
//-----------------------   TDspDataBlock Class  -------------------------------
//-------------------------------------------------------------------------------
    class TDspDataBlock {
      public:
        TDspDataBlock( Size blokSize );
        ~TDspDataBlock();

        Address         WhereHeader();
        Address         WhereData();
        Size            Size();
        Boolean         Status( short bitNum);
        void            SetStatus(short bitNum);
        void            ResetStatus( short bitNum);
        void            InstallIntVec( ProcPtr intRoutine);
        long            UserVar();
        void            UserVar( long newVar);
        unsigned long   DataType();
        void            DataType(unsigned long newVar);
        void            Name(Str31 newName);
        Str31           Name();
        void            ReadIndex(unsigned long newIndex);
        void            WriteIndex(unsigned long newIndex);
        unsigned long   ReadIndex();
        unsigned long   WriteIndex();

      protected:
        DataHeader              *fDataHeader;
    };

//-------------------------------------------------------------------------------
//---------------------------   TDspPRB Class  ----------------------------------
//-------------------------------------------------------------------------------
    class TDspPRB : public TDspDataBlock {
        public:
            TDspPRB( short whichPort );
            ~TDspPRB();

            short       WhichPRB();

        private:
            short               fPRBnum;
    };

//-------------------------------------------------------------------------------
//-----------------------   TDspParamBlock Class  -------------------------------
//-------------------------------------------------------------------------------
    class TDspParamBlock {
      public:
        TDspParamBlock( Size blokSize );
        ~TDspParamBlock();

        Address         Where();
        Size            BlockSize();

      private:
        Address                 fParamBlock;
        Size                    fSize;
    };

PortID          thePortID;
```

```
//--------------------------------------------------------------------------------
//--------------------------   TDspTask Class  -----------------------------------
//--------------------------------------------------------------------------------
    class TDspTask {
      public:
                        TDspTask();
                        ~TDspTask();    /* will free the code blocks*/
        void            InstallCodeBlock( TDspCodeBlock* newCodeBlock );
        void            Name(Str31 newName);
        Str31           Name();
        Address         Where();
        unsigned long   BusCycles();
        void            BusCycles( unsigned long  newBusCycleCount );
        unsigned long   DSPCycles();
        void            DSPCycles( unsigned long  newDSPCycleCount );
        TDspCodeBlock   *FirstCodeBlock();
        TDspTask        *NextTaskHeader();
        void            NextTaskHeader( TDspTask* newNextTask );
        void            WhichPort( PortID daPort ) { fPortID = daPort; };
        PortID          WhichPort() { return fPortID; };

      private:
        TaskHeader                  *fTaskHeader;
        TDspCodeBlock               *fFirstTCodeBlock;
        TDspTask                    *fNextTask;
        PortID                      fPortID;

      protected:
        void            FirstCodeBlock( TDspCodeBlock* newFirstCodeBlock );
    };

//--------------------------------------------------------------------------------
//-----------------------   TDspCodeBlock Class  -------------------------------
//--------------------------------------------------------------------------------
    class TDspCodeBlock {
      public:
                        TDspCodeBlock( TDspTask* myTask );
                        ~TDspCodeBlock( );  /* will free ifunc, vfunc, not data memory space, also itself */
        OSErr           AddVFunc( Str31 vfuncname );
        OSErr           AddIFunc( Str31 ifuncname );
        void            AddDataBlock( short whichDBH, TDspDataBlock* whichDB );
        void            AddDataBlock( TDspParamBlock* whichPB );
        Ptr             ParamBlock();
        Address         Where();
        TDspCodeBlock   *NextCodeBlock();
        void            NextCodeBlock(TDspCodeBlock* newCBSet);

      protected:
        Address         GetTrueDSPFResource( Str31 resname);

      private:
        CodeHeader                  *fCodeHeader;
        Ptr                         fLastVFunc;
        TDspCodeBlock               *fNextCodeBlock;
        TDspTask                    *fMyTask;

    };

//--------------------------------------------------------------------------------
//----------------------------   TDsp Class  -----------------------------------
//--------------------------------------------------------------------------------
    class TDsp {
        public:
            TDsp();
            ~TDsp();

            OSErr InstallTask(ProcessType theType, TDspTask* newTask, PortID whichPort, ReferenceQualifier where);
            void SuspendTask(TDspTask* theTask, Boolean immed);
            void ResumeTask(TDspTask* theTask, Boolean immed);
            OSErr RemoveTask(ProcessType theType, TDspTask* theTask);  // Does not kill
            OSErr KickIt();
    };

//--------------------------------------------------------------------------------
//-------------------------   TDspSound Class  -----------------------------------
//--------------------------------------------------------------------------------
    class TDspSound {
```

```
        public:
            TDspSound(Str31 whichAlg);
            ~TDspSound();

            TDspParamBlock          *fSoundParams;
            TDspDataBlock           *fDataBlock;
            TDspTask                *fTask;
            Boolean                 fPlaying;

        private:
            TDspPRB*                fPRB;
    };

//---------------------------------------------------------------------------
//-----------------------   TVoiceManager Class  ----------------------------
//---------------------------------------------------------------------------
    class TVoiceManager {
        public:
            TVoiceManager();
            ~TVoiceManager();

            OSErr           InstallVoices(Str31 whichAlg);
            void            PlayVoice(Parameters theParams);
            void            CheckVoices();
            void            GetSamples(SampleBlock* theSamples);

        private:
            TDspSound               *fVoices[kMaxVoices];
    };


//---------------------------------------------------------------------------
//-----------------------   TAnalysisTask Class  ----------------------------
//---------------------------------------------------------------------------
    class TAnalysisTask {
        public:
            TAnalysisTask(Str31 whichAlg);
            ~TAnalysisTask();

            void            Start();
            void            Stop();
            void            SetHostReady();
            void            SetParams(Parameters theParams);
            void            SetIndex(float newIndex);
            void            AdjustIndex(float offset);
            float           GetIndex();
            void            GetData(FFTData* samples);

            TDspParamBlock          *fSoundData;
            TDspTask                *fGenerateTask;
            TDspTask                *fAnalyzeTask;
    };


//===========================================================================
//--- The classes below must interact with MacApp, thus the "virtual pascal" ----
//--- declarations of the methods.                                          ----
//===========================================================================

//---------------------------------------------------------------------------
//-----------------------   TEditorView Class  ----------------------------
//---------------------------------------------------------------------------
    class TEditorView : public TView {
        public:
            virtual pascal void     DrawContents();
    };


//---------------------------------------------------------------------------
//-----------------------   TAnalysisView Class  ----------------------------
//---------------------------------------------------------------------------
    class TAnalysisView : public TView {
        public:
            virtual pascal void     Draw(Rect *area);
            virtual pascal Boolean  DoIdle(IdlePhase phase);
            virtual pascal void     SetViewFlag();
            virtual pascal void     ClearViewFlag();
            virtual pascal void     Restart( );
```

```
        TStaticText             *fTime;
        TStaticText             *fPercent;
    protected:
        TAnalysisTask           *fGenerator;
        Boolean                 fShowData;
};

//-----------------------------------------------------------------------------
//---------------------  TAnalysisButton Class  -------------------------------
//-----------------------------------------------------------------------------
class TAnalysisButton : public TPicture {
    public:
        virtual pascal TCommand     *DoMouseCommand(Point *theMouse, EventInfo *info,
                                        Point *hysteresis);
    protected:
        TAnalysisTask           *fGenerator;
};


//-----------------------------------------------------------------------------
//--------------------------  TAnalysis Class ---------------------------------
//-----------------------------------------------------------------------------
class TAnalysis : public TObject {
    public:
        virtual pascal void         IAnalysis();
        virtual pascal void         Free();
        virtual pascal void         SetAlg(Str31 whichAlg);
        virtual pascal void         OpenWindow();
        virtual pascal void         Restart();

        AnalysisMode            fMode;
        TAnalysisTask           *fAnalysisTask;
        TAnalysisView           *fAnalysisView;
        TWindow                 *fWindow;

    protected:
        TAnalysisButton         *fPlayButton;
        TAnalysisButton         *fStopButton;
        TAnalysisButton         *fRewButton;
        TAnalysisButton         *fFFButton;
        TAnalysisButton         *fStepFButton;
        TAnalysisButton         *fStepBButton;
};


//-----------------------------------------------------------------------------
//----  The SoundGen Class is an abstract class which is subclassed by all  ---
//----  sound generators.                                                    ---
//-----------------------------------------------------------------------------

class TSoundGen : public TObject {
    public:
        virtual pascal void         Generate(Boolean showIt);
        virtual pascal void         SetView(TEditorView *theView);
        virtual pascal void         GenerateAnalysis();
        virtual pascal void         Draw();
        virtual pascal void         DoRead( short aRefNum );
        virtual pascal void         DoWrite( short aRefNum );
        virtual pascal void         ShowIt(Boolean isShown);

        Str31                   fAlgorithm;
    protected:
        TEditorView     *fEditorView;               // The view which contains the op buttons
        Parameters      fParams;                    // Shared parameter block
};

//-----------------------------------------------------------------------------
//--------------------------  TFMSound Class  ---------------------------------
//-----------------------------------------------------------------------------
class TFMSound : public TSoundGen {
    public:
        virtual pascal void         IFMSound();
        virtual pascal void         Draw();
        virtual pascal void         Generate(Boolean showIt);
        virtual pascal void         SetConfig(int theConfig);
        virtual pascal int          GetConfig();
        virtual pascal void         SetOscParams(int whichOsc);
```

```
        virtual pascal void         ShowIt(Boolean isShown);
        virtual pascal void         GenerateAnalysis();
        virtual pascal void         DoRead( short aRefNum );
        virtual pascal void         DoWrite( short aRefNum );
    protected:
        FMOperator fOps[4];                     // 4 operator FM synthesis
        int fConfiguration;                     // The current configuration (1-8)
};

//---------------------------------------------------------------------------
//-----------------------   TKarplusSound Class  ----------------------------
//---------------------------------------------------------------------------
class TKarplusSound : public TSoundGen {
    public:
        virtual pascal void         IKarplusSound();
        virtual pascal void         Draw();
        virtual pascal void         Generate(Boolean showIt);
        virtual pascal void         GenerateAnalysis();
        virtual pascal void         ShowIt(Boolean isShown);
};

//---------------------------------------------------------------------------
//-----------------------   TAdditiveSound Class  ---------------------------
//---------------------------------------------------------------------------
class TAdditiveSound : public TSoundGen {
    public:
        virtual pascal void         IAdditiveSound();
        virtual pascal void         Draw();
        virtual pascal void         Generate(Boolean showIt);
        virtual pascal void         ShowIt(Boolean isShown);
        virtual pascal void         SetOscParams(long whichOsc);
        virtual pascal void         GenerateAnalysis();
        virtual pascal void         DoRead( short aRefNum );
        virtual pascal void         DoWrite( short aRefNum );

    protected:
        AdditiveOp  fOps[4];                    // 4 operator additive synthesis
};

//---------------------------------------------------------------------------
//-----------------------   TSoundDocument Class  ---------------------------
//---------------------------------------------------------------------------
class TSoundDocument : public TDocument {
    public:
        virtual pascal void         ISoundDocument();
        virtual pascal void         DoRead( short aRefNum, Boolean rsrcExists, Boolean forPrinting );
        virtual pascal void         DoWrite( short aRefNum, Boolean makingCopy );
        virtual pascal void         DoMakeViews( Boolean forPrinting );
};

//---------------------------------------------------------------------------
//----------------------   TSoundApplication Class  -------------------------
//---------------------------------------------------------------------------
class TSoundApplication : public TApplication {
    public:
        virtual pascal void         ISoundApplication(OSType itsMainFileType);
        virtual pascal void         HandleFinderRequest();
        virtual pascal TCommand     *DoMenuCommand(CmdNumber aCmdNumber);
        virtual pascal void         DoSetupMenus();
        virtual pascal void         SetSoundGen(SoundType theMode);
        virtual pascal SoundType    GetSoundGen();
        virtual pascal void         Terminate();
        virtual pascal void         Idle(IdlePhase phase);
        virtual pascal void         PlaySound(Boolean showIt);
        virtual pascal void         UpdateViewWindow()          { fViewWindow->ForceRedraw(); }
        virtual pascal TDocument    *DoMakeDocument( CmdNumber itsCmdNumber );

        TSoundGen       *fGenerator;            // Current sound generator
        TVoiceManager   *fVoiceManager;         // The voice manager (future multi-voice)
        TAnalysis       *fAnalysis;             // The analysis object
        TFMSound        *fFM;                   // The FM Sound generator
        TKarplusSound   *fKarplus;              // The Karplus/Strong sound generator
        TAdditiveSound  *fAdditive;             // The Additive synthesis generator
        TDsp            *fDsp;                  // The DSP object (interacts w/Bass-O-Matic)

    protected:
        SoundType       fSoundGen;              // Keeps track of current sound generation technique
        TWindow         *fEditorWindow;         // Ptr back to main window
```

```
        TWindow         *fViewWindow;           // Ptr to the Sound View window
        TWindow         *fAnalysisWindow;       // Ptr to the analysis window
        TSoundDocument  *fDocument;             // Ptr to current document
    };


//--------------------------------------------------------------------------
//----------------------   TEnvelopeView Class  ----------------------------
//--------------------------------------------------------------------------
    class TEnvelopeView : public TView {
        public:
            virtual pascal void         Draw(Rect *area);
            virtual pascal TCommand     *DoMouseCommand(Point *theMouse, EventInfo *info,
                                        Point *hysteresis);
            virtual pascal void         SetEnvValues(int theEnvValues[6]);
            virtual pascal void         GetEnvValues(int theEnvValues[6]);

        protected:
            int fEnvValues[6];                  // The current envelope values
    };

//--------------------------------------------------------------------------
//----------------------   TEnvelopeCommand Class  -------------------------
//--------------------------------------------------------------------------
    class TEnvelopeCommand  : public TCommand {
        public:
            virtual pascal void         IEnvelopeCommand(TEnvelopeView *itsView);
            virtual pascal void         TrackFeedback(VPoint *anchorPoint, VPoint *nextPoint,
                                            Boolean turnItOn, Boolean mouseDidMove);
            virtual pascal TCommand     *TrackMouse(TrackPhase aTrackPhase, VPoint *anchorPoint,
                                        VPoint *previousPoint, VPoint *nextPoint,
                                            Boolean mouseDidMove);
            virtual pascal void         TrackConstrain(VPoint *anchorPoint, VPoint *previousPoint, VPoint *nextPoint);
        protected:
            TEnvelopeView *fEnvelopeView;
    };


//--------------------------------------------------------------------------
//-------------------------   TSoundView Class  ----------------------------
//--------------------------------------------------------------------------
    class TSoundView : public  TView {
        public:
            virtual pascal void         Draw(Rect *area);
    };


//--------------------------------------------------------------------------
//-------------------------   TOscButton Class  ----------------------------
//--------------------------------------------------------------------------
    class TOscButton : public TIcon {
        public:
            virtual pascal TCommand     *DoMouseCommand(Point *theMouse, EventInfo *info, Point *hysteresis);
    };


//--------------------------------------------------------------------------
//-------------------------   TMyPopup Class  ------------------------------
//--------------------------------------------------------------------------
    class TMyPopup  : public TPopup {
        public:
            virtual pascal void         DoChoice(TView *origView, short itsChoice);
    };


//--------------------------------------------------------------------------
//-------------------------   TZoomButton Class  ---------------------------
//--------------------------------------------------------------------------
    class TZoomButton : public TPicture {
        public:
            virtual pascal TCommand     *DoMouseCommand(Point *theMouse, EventInfo *info, Point *hysteresis);
    };

//--------------------------------------------------------------------------
//----------------------   TSoundScroller Class  -------------------------
//--------------------------------------------------------------------------
    class TSoundScroller : public TScroller {
        public:
```

```
        virtual pascal void         DoScroll(VPoint *delta, Boolean redraw);
    };



//------    Global application object    ----------------
    TSoundApplication   *gSoundApplication;     // Global Application object


#endif
```

```
//========================================================================
//
// NAME
//      USound.cp
//
// SYNOPSIS
//      This file implements the TSoundApplication class and a few other
//      small classes that didn't seem to fit anywhere else.
//
// COPYRIGHT
//      Author Jeff Boone, Oregon State University
//
//  HISTORY
//
//========================================================================
#include    "USound.h"
#include    <Packages.h>

//-----------------------------------------------------------
//  Some utility functions
//-----------------------------------------------------------
extern void pStrCpy( Str31 outstr,  Str31 instr);

int strlen(char *s)
{
    int n;

    for (n=0; *s != '\0'; s++)
        n++;
    return(n);
}

void C2PStrCpy(char *Cstr, Str255 Pstr)
{
    short i, len = strlen(Cstr);

    for(i=len; i>0; i--) {
        Pstr[i] = Cstr[i-1];
    }
    Pstr[i] = len;

}


#pragma segment AInit

// ************* S O U N D   A P P L I C A T I O N   C L A S S  *************
//========================================================================
//  TSoundApplication class:
//      This is the application class that is a decendant of the
//      MacApp TApplication class. It handles all interaction with
//      the user.
//========================================================================

//========================================================================
//  ISoundApplication:
//      This method initializes the TSoundApplication object and should
//      be called immediately after instantiating a TSoundApplication
//      object. It creates all the windows that we will use throughout
//      the application.
//========================================================================
pascal void TSoundApplication::ISoundApplication(OSType itsMainFileType)
{
    const short     kEditorID       = 1001;
    const short     kViewID         = 1002;
    const short     kAnalysisID     = 1003;

    this->IApplication(itsMainFileType);

    //-------------------------------------------------------------------
    //------- This is a hack to suppress dead-stripping of --------------
    //------- our classes.  The variable "gDeadSuppression" -------------
    //------- is actually a MacApp constant that is always false --------
    //-------------------------------------------------------------------
    if (gDeadStripSuppression)
        {
            TSoundGen           *aSoundGen;
            TEnvelopeView       *anEnvelope;
```

```
        TEditorView         *anEditor;
        TSoundView          *aSound;
        TOscButton          *anOscButton;
        TMyPopup            *aPopup;
        TAnalysisView       *aView;
        TAnalysisButton     *aButton;
        TZoomButton         *zoomButton;
        TSoundScroller      *aScroller;

        aSoundGen       = new TSoundGen;
        anEnvelope      = new TEnvelopeView;
        anEditor        = new TEditorView;
        aSound          = new TSoundView;
        anOscButton     = new TOscButton;
        aPopup          = new TMyPopup;
        aView           = new TAnalysisView;
        aButton         = new TAnalysisButton;
        zoomButton      = new TZoomButton;
        aScroller       = new TSoundScroller;
    )

//----------------- Now the real work… -------------------

//----------------- Create the DSP object ------------------
fDsp = new TDsp();
FailNIL(fDsp);

//----------------- Create the Document object ------------------
fDocument = new TSoundDocument();
FailNIL(fDocument);
fDocument->ISoundDocument();

//----------------- Create the Sound Manager object ------------------
fVoiceManager = new TVoiceManager();
FailNIL(fVoiceManager);

//----------------- Create the FM generator object ------------------
fFM = new TFMSound;
FailNIL(fFM);
fFM->IFMSound();                // Initialize the FM generator object

//----------------- Create the Karplus/Strong object ------------------
fKarplus = new TKarplusSound;
FailNIL(fKarplus);
fKarplus->IKarplusSound();      // Initialize the Karplus/Strong sound generator

//----------------- Create the Additive sound generator object ------------------
fAdditive = new TAdditiveSound;
FailNIL(fAdditive);
fAdditive->IAdditiveSound();    // Initialize the Additive sound generator

//----------------- Initialize our fields ------------------
fSoundGen = FM;                             // Default to FM synthesis
fGenerator = (TSoundGen *)fFM;              // Point fGenerator to the FM generator object
fVoiceManager->InstallVoices("\pFMAlg1");   // Install the voices on the DSP
pStrCpy( fGenerator->fAlgorithm , "\pFMAnalysis1");

//----------------- Create the Analysis object & window ------------------
fAnalysisWindow = NewTemplateWindow(kAnalysisID, nil);
FailNIL(fAnalysisWindow);

fAnalysis = new TAnalysis();
FailNIL(fAnalysis);
fAnalysis->fWindow = fAnalysisWindow;
fAnalysis->IAnalysis();         // Initialize the Analysis object
fAnalysis->SetAlg("\pFMAnalysis1");

//----------------- Create the Sound view window ------------------
fViewWindow = NewTemplateWindow(kViewID, nil);
FailNIL(fViewWindow);

//----------------- Create the Sound editor window ------------------
fEditorWindow = NewTemplateWindow(kEditorID, nil);
FailNIL(fEditorWindow);

//----------------- Set the views for the generator objects ------------------
fFM->SetView ((TEditorView *)(fEditorWindow->FindSubView('view')));
fKarplus->SetView ((TEditorView *)(fEditorWindow->FindSubView('view')));
```

```
    fAdditive->SetView ((TEditorView *) (fEditorWindow->FindSubView('view')));

    //---------------- Open the editor window ------------------
    fEditorWindow->Open();

}


#pragma segment AOpen

pascal void TSoundApplication::HandleFinderRequest()
{
    // Don't want to do anything when starting from Finder
}


//======================================================================
//  Terminate:
//      This method is called by MacApp when the user is about to
//      terminate the application. It is used to do any last minute
//      clean up. We use it to delete the voice manager object, the
//      DSP object, and the analysis object.
//======================================================================
pascal void TSoundApplication::Terminate()
{
    //---------------- Delete the voice manager ------------------
    if (fVoiceManager != nil)
    {
        delete fVoiceManager;
        fVoiceManager = nil;
    }

    //---------------- Delete the DSP object ------------------
    if (fDsp != nil)
    {
        delete fDsp;
        fDsp = nil;
    }

    //---------------- Delete the analysis object ------------------
    if (fAnalysis != nil)
    {
        fAnalysis->Free();
        delete fAnalysis;
        fAnalysis = nil;
    }

    //---------------- Delete the analysis object ------------------
    if (fDocument != nil)
    {
        delete fDocument;
        fDocument = nil;
    }

    inherited::Terminate();
}



#pragma segment AOpen
//======================================================================
//  DoMenuCommand:
//      This method is called by MacApp when the user selects a
//      menu item. We figure out which menu item was selected and
//      do the appropriate behavior.
//======================================================================
pascal TCommand *TSoundApplication::DoMenuCommand(CmdNumber aCmdNumber)
{
    const short     SPLASH_ID   = 1006;

    DialogPtr       theDialog;
    EventRecord     theEvent;

    //---------------- See if we're in our range ------------------
    if ((aCmdNumber >= kEditorWindow) && (aCmdNumber <= kMySaveAs))
    {
        switch (aCmdNumber) {
            //---------------- User selected the Editor Window item ------------------
```

```
        case kEditorWindow:         // Show the editor window if it isn't already up
            {
                if (!(this->fEditorWindow->IsShown()))
                    this->fEditorWindow->Open();
                this->fEditorWindow->Select();      // Bring the window to the front
                break;
            }

        //---------------- User selected the Analysis window item ------------------
        case kAnalysisWindow:
            {
                if (!(this->fAnalysisWindow->IsShown()))
                    this->fAnalysisWindow->Open();
                this->fAnalysisWindow->Select();        // Bring the window to the front
                break;
            }


        //---------------- User selected the View window item ------------------
        case kViewWindow:       // Show the sound view window if it isn't already up )
            {
                if (!(this->fViewWindow->IsShown()))
                    this->fViewWindow->Open();
                this->fViewWindow->Select();        // Bring the window to the front
                break;
            }


        //---------------- User selected the FM synthesis item ------------------
        case kFMSynthesis:
            {
                this->fSoundGen = FM;                       // Set the current synth method
                this->fGenerator = (TSoundGen *)fFM;        // Point the generator to FM generator
                fAdditive->ShowIt(false);                   // Let Additive hide
                fKarplus->ShowIt(false);                    // Let Karplus/Strong hide
                fFM->ShowIt(true);                          // Show the FM synth buttons
                fViewWindow->Update();                      // Update the sound view
                fAnalysis->Restart();                       // Start the analysis over
                break;
            }

        //---------------- User selected the Karplus/Strong synthesis item ------------------
        case kKSSynthesis:
            {
                this->fSoundGen = KarplusStrong;            // Set the current synth method
                this->fGenerator = (TSoundGen *)fKarplus;   // Point the generator to K/S generator
                fAdditive->ShowIt(false);                   // Let Additive hide
                fKarplus->ShowIt(true);                     // Show the K/S button
                fFM->ShowIt(false);                         // Let FM hide
                fViewWindow->Update();                      // Update the sound view
                fAnalysis->Restart();                       // Start the analysis over
                break;
            }

        //---------------- User selected the Additive synthesis item ------------------
        case kAddSynthesis:
            {
                this->fSoundGen = Additive;                 // Set the current synth method
                this->fGenerator = (TSoundGen *)fAdditive;  // Point the generator to Additive generator
                fAdditive->ShowIt(true);                    // Show Additive buttons
                fKarplus->ShowIt(false);                    // Let K/S hide
                fFM->ShowIt(false);                         // Let FM hide
                fViewWindow->Update();                      // Update the sound view
                fAnalysis->Restart();                       // Start the analysis over
                break;
            }

        //---------------- User selected the save...  item ------------------
        case kMySaveAs:
            {
                fDocument->Save( kMySaveAs, true, false );
                break;
            }

    }
return gNoChanges;

}
```

```
    //---------------- See if user selected "About" item ------------------
    else if (aCmdNumber == kAboutApp)
        {
            //---------------- Get the dialog from resource file ------------------
            theDialog = GetNewCenteredDialog(SPLASH_ID, nil, nil);
            if (theDialog != nil)
            {
                BringToFront(theDialog);
                DrawDialog(theDialog);      // Show splash screen
            }

            //---------------- Wait for user to click the mouse ------------------
            do
                if (GetNextEvent(mDownMask,&theEvent))  {}
            while (theEvent.what == nullEvent);

            //---------------- Free up memory used by dialog ------------------
            DisposDialog(theDialog);
        }

    else
        return (inherited::DoMenuCommand(aCmdNumber));
}



#pragma segment ARes
//=========================================================================
//  DoSetupMenus:
//      This method is called by MacApp to set up the menu bar for
//      the user. This is where you enable/disable menus or set
//      check marks on menu items.
//=========================================================================
pascal void TSoundApplication::DoSetupMenus()
{
    int     i;

    inherited::DoSetupMenus();

    //---------------- Enable all our menus ------------------
    for (i = kEditorWindow; i<= kMySaveAs; i++)
        Enable(i, true);

    //------------ Put a check by the current synth method ---------------
    switch (this->fSoundGen) {
        case FM:            EnableCheck(kFMSynthesis, false, true); break;
        case KarplusStrong: EnableCheck(kKSSynthesis, false, true); break;
        case Additive:      EnableCheck(kAddSynthesis, false, true); break;
    }

    //-------- Put a check on window menu if window is open -------------
    if (this->fEditorWindow->IsShown())
        EnableCheck (kEditorWindow, true, true);
    if (this->fViewWindow->IsShown())
        EnableCheck (kViewWindow, true, true);
    if (this->fAnalysisWindow->IsShown())
        EnableCheck (kAnalysisWindow, true, true);

}



//=========================================================================
//  SetSoundGen:
//      This method sets the current sound generation mode for the
//      application. Valid values are "FM", "Additive", and "KarplusStrong"
//=========================================================================
pascal void TSoundApplication::SetSoundGen(SoundType theMode)
{
    this->fSoundGen = theMode;
    switch (theMode)
    {
        case FM:            fGenerator = (TSoundGen *)fFM; break;
        case KarplusStrong: fGenerator = (TSoundGen *)fKarplus; break;
        case Additive:      fGenerator = (TSoundGen *)fAdditive; break;
    }
}
```

```
//=============================================================================
//  GetSoundGen:
//      This method returns the current sound generation mode for the
//      application. Valid values are "FM", "Additive", and "KarplusStrong"
//=============================================================================
pascal SoundType  TSoundApplication::GetSoundGen()
{
    return (this->fSoundGen);
}


//=============================================================================
//  Idle:
//      This method is called by MacApp during the idle phase of our
//      application. We use it to check to see if there are any voices
//      on the DSP that need to be cleaned up.
//=============================================================================
pascal void TSoundApplication::Idle(IdlePhase phase)
{
    fVoiceManager->CheckVoices();           // See if we need to remove voices
    inherited::Idle(phase);
}


#pragma segment Main
//=============================================================================
//  PlaySound:
//      This method is called by our application to play a voice of
//      the currently installed sound generator.
//=============================================================================
pascal void TSoundApplication::PlaySound(Boolean showIt)
{
    fGenerator->Generate(showIt);
}


#pragma segment Main
//=============================================================================
//  DoMakeDocument:
//      This method is called by our application to play a voice of
//      the currently installed sound generator.
//=============================================================================
pascal TDocument *TSoundApplication::DoMakeDocument( CmdNumber /* itsCmdNumber */ )
{
    return (TDocument *)fDocument;
}



// ****************  E D I T O R   V I E W   C L A S S  ****************** )
//=============================================================================
//  TEditorView class:
//      This class is used to handle the sound editor window in which
//      the oscillator buttons are presented to the user.
//=============================================================================

//=============================================================================
//  DrawContents:
//      This method draws the contents of the editor window. It is
//      called by MacApp whenever the screen needs to be updated.
//      We just give the current sound generator a chance to redraw
//      itself.
//=============================================================================
#pragma segment ADraw
pascal void TEditorView::DrawContents()
{
    inherited::DrawContents();

    if (this->IsFocused())
        gSoundApplication->fGenerator->Draw();      // Let generator redraw itself
}
```

```
// ********************  O S C B U T T O N   C L A S S  **********************
//=========================================================================
//  TOscButton class:
//      This class implements the oscillator buttons that user can
//      click on to set the parameters for the oscillator.
//=========================================================================

#pragma segment ASelCommand
//=========================================================================
//  DoMouseCommand:
//      This method handles the user clicking on one of the oscillator
//      buttons. It calls the appropriate method to set the parameters.
//=========================================================================
pascal TCommand *TOscButton::DoMouseCommand(Point *theMouse, EventInfo *info,
                                            Point *hysteresis)
{
    unsigned long       selectedOsc;

    this->Hilite();         // Hilight the button
    while (WaitMouseUp())  ;
    this->Hilite();         // UnHilight the button

    selectedOsc = this->fIdentifier - kIDConstant;

    switch (selectedOsc) {
        case 0:case 1:case 2:case 3:    gSoundApplication->fFM->SetOscParams ((int)selectedOsc); break;
        case 4:case 5:case 6:case 7:    gSoundApplication->fAdditive->SetOscParams (selectedOsc-4); break;
        case 8:     break;
        case 9: gSoundApplication->PlaySound(false); break;
    }

    return (inherited::DoMouseCommand(theMouse, info, hysteresis));
}


// ******************  S O U N D   V I E W   C L A S S  *******************
//=========================================================================
//  TSoundView class:
//      This class implements displaying the sound wave form to the
//      user in a scrollable window.
//=========================================================================

#pragma segment ADraw
//=========================================================================
//  Draw:
//      This method draws the sound wave on the screen for the user
//      to view. It lets the DSP generate the waveform samples.
//=========================================================================
pascal void TSoundView::Draw(Rect * /* area */)
{
    const short     kStep   = 20;   // Number of samples to skip each step

    int         numSamples;
    SampleBlock *theSamples;
    int         width, height, i, viewSize;
    Rect        visRect;

    if (this->IsFocused())
    {
        //------------ Create room for the sample block ---------------
        theSamples = (SampleBlock*)(NewPtr(sizeof(SampleBlock)));

        //------------ Start the DSP generating samples ---------------
        gSoundApplication->PlaySound(true);         // Set the ViewIt Flag
        numSamples = 1;

        //------------ Move to the left/center of the window ----------------
        MoveTo(0,(short)(this->fSize.v / 2));
        this->GetVisibleRect(&visRect);
        viewSize = int (this->fSize.v / 2);

        do
        {
            //------------ Get a frames worth of data (240 samples) ----------------
            gSoundApplication->fVoiceManager->GetSamples(theSamples);

            i=1;
            do
```

```
            {
                //------------ Draw the 240 samples ---------------
                width = int (this->fSize.h * ((numSamples+1)/24960.0));
                if ((width < visRect.right) && (width > visRect.left))      // Only draw visible stuff
                {
                    height = int (theSamples->sample[i]*this->fSize.v);
                    LineTo(width,(short)(viewSize + height/2));
                }
                i += kStep;
            }
            while (i < 240);

            numSamples += 240;
        }
        while (numSamples < 25000);     // Loop until we draw all the samples

        DisposPtr(Ptr(theSamples));
    }
}


// ****************** E N V E L O P E   V I E W   C L A S S   ********************
//=========================================================================
//  TEnvelopeView class;
//      This class implements displaying the 6 envelope values to the
//      user and allowing them to change them by dragging the "handles"
//      up or down.
//=========================================================================

#pragma segment ADraw
//=========================================================================
//  Draw:
//      This method draws the envelope values on the screen into the
//      modal dialog.
//=========================================================================
pascal void TEnvelopeView::Draw(Rect *area)
{
    const   short   kXstep = 22;

    int     count;
    Rect    pointRect;

    if (this->Focus())
    {
        //------------ Put a box around the whole 9 yards ---------------
        FrameRect( area );

        //------------ Paint the handles ---------------
        for (count=0; count<6; count++)
        {
            SetRect (&pointRect, (count+1)*kXstep-3, 100-fEnvValues[count]-3, (count+1)*kXstep+3, 100-fEnvValues[count]+3);
            PaintRect (&pointRect);
        }

        //------------ Draw the lines in between env values ---------------
        MoveTo (kXstep, 100-fEnvValues[0]);                     // Move to first env value
        for (count=1; count<6; count++)                         // Paint the lines
            LineTo ((count+1)*kXstep, 100-fEnvValues[count]);   // Draw lines to other env values
    }
}



#pragma segment ASelCommand

//=========================================================================
//  DoMouseCommand:
//      This method creates a TCommand object whenever the user drags
//      one of the envelope handles up or down.
//=========================================================================
pascal TCommand *TEnvelopeView::DoMouseCommand(Point * /*theMouse*/, EventInfo * /*info*/, Point * /*hysteresis*/)
{
    TEnvelopeCommand *aCommand;

    //------------ Create the new command object ---------------
    aCommand = new TEnvelopeCommand;
    FailNIL(aCommand);
```

```
    //------------ Initialize the command object ---------------
    aCommand->IEnvelopeCommand(this);

    return ((TCommand *)(aCommand));
}


#pragma segment ADoCommand
//=====================================================================
//  SetEnvValues:
//      This method sets the 6 envelope values to those passed in via
//      the array.
//=====================================================================
pascal void TEnvelopeView::SetEnvValues(int theEnvValues[6])
{
    int     count;

    for (count=0; count<6; count++)
        fEnvValues[count] = theEnvValues[count];
}


#pragma segment ADoCommand
//=====================================================================
//  GetEnvValues:
//      This method returns the 6 envelope values in the array.
//=====================================================================
pascal void TEnvelopeView::GetEnvValues(int theEnvValues[6])
{
    int     count;

    for (count=0; count <6; count++)
        theEnvValues[count] = fEnvValues[count];
}


// ********************* E N V E L O P E   C O M M A N D   C L A S S  **********************
//=====================================================================
//  TEnvelopeCommand class:
//      This class implements a TCommand object that is used to handle
//      the user dragging the envelope handles up or down to change
//      their value.
//=====================================================================

#pragma segment ADoCommand
//=====================================================================
//  IEnvelopeCommand:
//      This method initializes the TEnvelopeCommand object and should
//      be called immediately after instantiating a TEnvelopeCommand
//      object.
//=====================================================================
pascal void TEnvelopeCommand::IEnvelopeCommand(TEnvelopeView *itsView)
{
    this->fEnvelopeView = itsView;
    this->ICommand (0, nil, (TView *)(itsView), nil);       // Let superclass initialize
    this->fConstrainsMouse = true;                          // We constrain the mouse
}


#pragma segment ADoCommand

//=====================================================================
//  TrackFeedback:
//      This method is called by MacApp as the user drags the mouse
//      around with the button down. It handles redrawing the envelope
//      values on the screen as the user changes them.
//=====================================================================
pascal void TEnvelopeCommand::TrackFeedback(VPoint *anchorPoint, VPoint *nextPoint,
                                    Boolean /*turnItOn*/, Boolean mouseDidMove)
{
    const   short   kXstep      = 22;
    const   short   kHandleSize = 3;

    Rect    tempRect;
    int     xLoc, value, theIndex, theEnvValues[6];

    if (mouseDidMove)
    {
```

```
    if (fView->Focus())
    {
      //------------ Paint using X-Or mode ---------------
      PenMode(patXor);
      anchorPoint->v = 100 - anchorPoint->v;

      xLoc = (int)((((anchorPoint->h-11) / kXstep) + 1) * kXstep - 2);
      if (xLoc < 0)
          xLoc = kXstep;
      else
          if (xLoc > 135)
              xLoc = 132;

      fEnvelopeView->GetEnvValues( theEnvValues );                    // Get the env values from envelope view object

      //------------ Figure out which env value to change ---------------
      theIndex = ((xLoc-11) / kXstep);

      //------------ Repaint the handle ---------------
      SetRect (&tempRect, xLoc-kHandleSize, (short)nextPoint->v-kHandleSize, xLoc+kHandleSize, (short)nextPoint->v+kHandle
      PaintRect (&tempRect);

      switch (theIndex) {
          //------------ We're changing 1st point, only redraw line to 2nd ---------------
          case 0:  {
                  MoveTo (xLoc, (short)(nextPoint->v));
                  LineTo (kXstep * 2, 100-(theEnvValues[1]));     // Line to 2nd point
                  break;
              }

          //------------ We're changing 6th point, only redraw line to 5th ---------------
          case 5:  {
                  MoveTo (xLoc, (short)(nextPoint->v));
                  LineTo (kXstep * 5, (short)(100-(theEnvValues[4])));    // Line to 5th point
                  break;
              }

          //------------ We're changing a middle point, redraw surrounding lines ---------------
          case 1:case 2:case 3:case 4:
              {
                  MoveTo (xLoc, (short)(nextPoint->v));               // Lines to surrounding points
                  LineTo (xLoc - kXstep, 100-(theEnvValues[theIndex-1]));
                  MoveTo (xLoc, (short)(nextPoint->v));
                  LineTo (xLoc + kXstep, 100-(theEnvValues[theIndex+1]));
                  break;
              }
      }

      //------------ Calculate and change the value for the current point ---------------
      value = (int)(100 - (nextPoint->v));
      theEnvValues[theIndex] = value;
      fEnvelopeView->SetEnvValues(theEnvValues);          // Update the envelope values
    }
  }
}



#pragma segment ADoCommand
//--------------------------------------------------------------------
//  TrackMouse:
//      This method is called by MacApp when the user is dragging the
//      mouse around. We use it to force a redraw when the user finally
//      finishes dragging the mouse.
//--------------------------------------------------------------------
pascal TCommand *TEnvelopeCommand::TrackMouse(TrackPhase aTrackPhase, VPoint * /*anchorPoint*/,
                                    VPoint */*previousPoint*/, VPoint * /*nextPoint*/, Boolean /*mouseDidMove*/)
{
    if (aTrackPhase == trackRelease)        // User is finished
        fView->ForceRedraw();               // Force a redraw of the screen
    return (this);
}



//--------------------------------------------------------------------
//  TrackConstrain:
//      This method is called by MacApp to constrain the motion as
```

```
//      the user drags the mouse around. In our case we constrain the
//      user to the visible part of our dialog.
//========================================================================
pascal void TEnvelopeCommand::TrackConstrain(VPoint * /*anchorPoint*/, VPoint * /*previousPoint*/, VPoint *nextPoint)
{
    const   short   kTop        = 0;
    const   short   kBottom     = 100;

    if (nextPoint->v < kTop)
        nextPoint->v = kTop;
    else
        if (nextPoint->v > kBottom)
            nextPoint->v = kBottom;
}



// ********************** S O U N D  G E N  C L A S S  **********************
//========================================================================
//  TSoundGen class:
//      This class is an abstract class from which all the sound
//      generators are derived. The methods here are the bare minimum
//      for a sound generator to implement.
//========================================================================

#pragma segment ADoCommand
//========================================================================
//  SetView:
//      This method sets the view for the sound generator to be the
//      one passed as a parameter.
//========================================================================
pascal void     TSoundGen::SetView(TEditorView *theView)
{
    fEditorView = theView;
}


#pragma segment Main
//========================================================================
//  Generate:
//      This method generates a sound on the DSP.
//========================================================================
pascal void     TSoundGen::Generate(Boolean showIt)
{
    // ------- Must be overridden -----------
}


//========================================================================
//  Generate:
//      This method generates analysis data for the FFT analysis
//      object.
//========================================================================
pascal void     TSoundGen::GenerateAnalysis()
{
    // ------- Must be overridden -----------
}


//========================================================================
//  Generate:
//      This method draws whatever is necessary for the sound
//      generator object.
//========================================================================
pascal void     TSoundGen::Draw()
{
    // ------- Must be overridden -----------
}


//========================================================================
//  DoRead:
//      This method should be filled in by the subclass to do
//      whatever is necessary to read in the data from a file.
//========================================================================
pascal void     TSoundGen::DoRead( short aRefNum )
{
    // ------- Must be overridden -----------
```

```
}


//--------------------------------------------------------------------
//  DoWrite:
//      This method should be filled in by the subclass to do
//      whatever is necessary to write the data to a file.
//--------------------------------------------------------------------
pascal void     TSoundGen::DoWrite( short aRefNum )
{
    // ------- Must be overridden -----------
}


//--------------------------------------------------------------------
//  ShowIt:
//      This method hides or shows the current sound generation
///     technique.
//--------------------------------------------------------------------
pascal void     TSoundGen::ShowIt( Boolean isShown )
{
    // ------- Must be overridden -----------
}


// ********************  M Y   P O P U P   C L A S S  **********************
//--------------------------------------------------------------------
//  TMyPopup class:
//      This class implements the popup menu for changing the algorithm
//      within FM synthesis.
//--------------------------------------------------------------------

#pragma segment ADoCommand
//--------------------------------------------------------------------
//  DoChoice:
//      This method sets the FM synthesis algorithm whenever the user
//      changes it using the popup menu.
//--------------------------------------------------------------------
pascal void TMyPopup::DoChoice(TView *origView, short itsChoice)
{
    gSoundApplication->fFM->SetConfig(this->fCurrentItem);      // Set the FM configuration
    inherited::DoChoice(origView, itsChoice);                   // Give the SuperClass a chance
}



// ****************  Z O O M  B U T T O N   C L A S S  *********************
//--------------------------------------------------------------------
//  TZoomButton class:
//      This class is used by the sound display for zooming in and out
//      on the waveform.
//--------------------------------------------------------------------

#pragma segment ASelCommand
//--------------------------------------------------------------------
//  DoMouseCommand:
//      This method figures out which button was pressed (zoom in or
//      zoom out) and does the appropriate behavior.
//--------------------------------------------------------------------
pascal TCommand *TZoomButton::DoMouseCommand(Point *theMouse, EventInfo *info, Point *hysteresis)
{
    TSoundView* theView;

    this->Hilite();             // Hilight the button
    while (WaitMouseUp())  ;    // Wait for user to let up the button
    this->Hilite();             // UnHilight the button

    theView = (TSoundView*)this->fSuperView->FindSubView('view');       // Find the view that has the waveform

    //-------------- User pressed the zoom in button ----------
    if (this->fIdentifier == 'ZmIn')
    {
        theView->Resize(theView->fSize.h*2, theView->fSize.v, true);    // Double the width of the view
    }
    else
    //-------------- User pressed the zoom out button ----------
    {
```

```
        theView->Resize(theView->fSize.h/2, theView->fSize.v, true);    // Shrink the view by 1/2
    }

    theView->ForceRedraw();     // Redraw the screen

    return (inherited::DoMouseCommand(theMouse, info, hysteresis));
}


// ***************  S O U N D  S C R O L L E R  C L A S S  ********************
//-----------------------------------------------------------------------
//  TSoundScroller class:
//      This class handles scrolling the waveform display window.
//-----------------------------------------------------------------------

#pragma segment ASelCommand
//-----------------------------------------------------------------------
//  DoScroll:
//      This method scrolls the window by letting the superclass do
//      the scroll and then forcing a redraw. We need to do this so
//      that we will redraw the whole view.
//-----------------------------------------------------------------------
pascal void         TSoundScroller::DoScroll(VPoint *delta, Boolean redraw)
{
    inherited::DoScroll(delta, redraw);

    this->FindSubView('view')->ForceRedraw();
}


// **************  S O U N D  D O C U M E N T  C L A S S  ******************
//-----------------------------------------------------------------------
//  TSoundDocument class:
//      This class implements saving and restoring the parameters for
//      the sound synthesis system.
//-----------------------------------------------------------------------

#pragma segment AReadFile
//-----------------------------------------------------------------------
//  DoRead:
//      This method reads the sound synthesis method, sets the
//      current synth method and call the generator to read its
//      data.
//-----------------------------------------------------------------------
pascal void TSoundDocument::DoRead( short aRefNum, Boolean /* rsrcExists */, Boolean /* forPrinting */ )
{
    long        readBytes;
    SoundType   theGenerator;

    readBytes = sizeof(SoundType);
    FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &theGenerator));    // Read the name of the generator

    gSoundApplication->fGenerator->ShowIt(false);                   // Hide the old sound gen method
    gSoundApplication->SetSoundGen(theGenerator);                   // Set current sound gen method
    gSoundApplication->fGenerator->DoRead( aRefNum );               // Let sound generator read
    gSoundApplication->fGenerator->ShowIt(true);                    // Show the new sound gen method
}


#pragma segment AWriteFile
//-----------------------------------------------------------------------
//  DoWrite:
//      This method writes the sound synthesis method and calls the
//      generator to write its data.
//-----------------------------------------------------------------------
pascal void TSoundDocument::DoWrite( short aRefNum, Boolean /* makingCopy */ )
{
    long        writeBytes;
    SoundType   theGenerator;

    writeBytes = sizeof(SoundType);
    theGenerator = gSoundApplication->GetSoundGen();
    FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &theGenerator));      // Write current sound gen method

    gSoundApplication->fGenerator->DoWrite( aRefNum );                  // Let sound generator write
}

#pragma segment ARes
```

```
//----------------------------------------------------------------------
//  ISoundDocument:
//      This method initializes the document and should be called
//      immediately after instantiating a TSoundDocument.
//----------------------------------------------------------------------
pascal void TSoundDocument::ISoundDocument( )
{
    this->IDocument( kFileType, kSignature, kUsesDataFork, !kUsesRsrcFork, !kDataOpen, !kRsrcOpen);
}

#pragma segment ARes
//----------------------------------------------------------------------
//  DoMakeViews:
//      Override MacApp's standard DoMakeViews so we don't create
//      any views.
//----------------------------------------------------------------------
pascal void TSoundDocument::DoMakeViews( Boolean /* forPrinting */ )
{
    // Don't create any views
}
```

# TAdditive.cp

```
//==========================================================================
//
// NAME
//      TAdditive.cp
//
// SYNOPSIS
//      This file implements the Additive synthesis sound class.
//
//      Author Jeff Boone, Oregon State University
//
//
//  HISTORY
//
//==========================================================================
#include    "USound.h"
#include    <Packages.h>


//--------------------------------------------------------------------
//  IAdditiveSound:
//      This method initializes the Additive sound object. It should be
//      called immediately after instantiating an Additive sound object.
//--------------------------------------------------------------------
pascal void TAdditiveSound::IAdditiveSound()
{
    int count;

    //-------------- Set the envelopes to something "reasonable" ---------------
    for (count=0; count<4; count++)
    {
        this->fOps[count].ampEnvValues[0] = this->fOps[count].freqEnvValues[0] = 10;
        this->fOps[count].ampEnvValues[1] = this->fOps[count].freqEnvValues[1] = 98;
        this->fOps[count].ampEnvValues[2] = this->fOps[count].freqEnvValues[2] = 75;
        this->fOps[count].ampEnvValues[3] = this->fOps[count].freqEnvValues[3] = 70;
        this->fOps[count].ampEnvValues[4] = this->fOps[count].freqEnvValues[4] = 60;
        this->fOps[count].ampEnvValues[5] = this->fOps[count].freqEnvValues[5] = 5;
    }

    //------- Set the oscillator frequencies to something "reasonable" -------------
    this->fOps[0].minFreq = 110;
    this->fOps[0].maxFreq = 440;

    this->fOps[1].minFreq = 110;
    this->fOps[1].maxFreq = 440;

    this->fOps[2].minFreq = 110;
    this->fOps[2].maxFreq = 440;

    this->fOps[3].minFreq = 110;
    this->fOps[3].maxFreq = 440;

}


#pragma segment Main

//--------------------------------------------------------------------
//  Generate:
//      This method sets up a parameter block and calls upon the
//      voice manager to play an Additive tone. It is called when the user
//      clicks on the "speaker" button when the system is set to
//      do Additive synthesis. The boolean parameter (showit) is set to true
//      whenever we are just viewing the sound on the screen.
//--------------------------------------------------------------------
pascal void TAdditiveSound::Generate(Boolean showIt)
{
    int count;

    //-------------- Number of samples ---------------
    fParams.param[0] = 24960.0;

    //-------------- Oscillator frequencies ---------------
    for (count=0; count<8; count+=2) {
        fParams.param[count+1] = float (this->fOps[count/2].minFreq);
        fParams.param[count+2] = float (this->fOps[count/2].maxFreq);
    }

    //-------------- Amplitude envelopes / 100 ---------------
```

```
    for (count=0; count<24; count++)
        fParams.param[count+9] = float (this->fOps[count/6].ampEnvValues[count%6])/100.0;

    //-------------- Frequency envelopes / 100 ---------------
    for (count=0; count<24; count++)
        fParams.param[count+33] = float (this->fOps[count/6].freqEnvValues[count%6])/100.0;

    //-------------- Flags ---------------
    fParams.flags = 0x00000000;         // Initialize the flags

    if (showIt)
        fParams.flags |= kViewIt;       // View flag set to true
    else
        fParams.flags &= ~kViewIt;      // View flag set to false

    fParams.flags |= kHostReady;        // Set the host ready flag
    fParams.flags &= ~kDoFFT;           // Clear the do FFT flag

    //-------- Tell voice manager to play it ------------
    gSoundApplication->fVoiceManager->PlayVoice(fParams);

}


//=============================================================================
//  GenerateAnalysis:
//      This procedure sets up a parameter block and passes these to
//      the analysis task. Note that nothing else occurs until the user
//      interacts with analysis control panel.
//=============================================================================
pascal void TAdditiveSound::GenerateAnalysis()
{
    int count;

    //-------------- Number of samples ---------------
    fParams.param[0] = 24960.0;

    //-------------- Oscillator frequencies ---------------
    for (count=0; count<8; count+=2) {
        fParams.param[count+1] = float (this->fOps[count/2].minFreq);
        fParams.param[count+2] = float (this->fOps[count/2].maxFreq);
    }

    //-------------- Amplitude envelopes / 100 ---------------
    for (count=0; count<24; count++)
        fParams.param[count+9] = float (this->fOps[count/6].ampEnvValues[count%6])/100.0;

    //-------------- Frequency envelopes / 100 ---------------
    for (count=0; count<24; count++)
        fParams.param[count+33] = float (this->fOps[count/6].freqEnvValues[count%6])/100.0;

    //------------------ Flags ----------------------
    fParams.flags = 0x00000000;         // Initialize the flags
    fParams.flags |= kHostReady;        // Set the host ready flag
    fParams.flags &= ~kDoFFT;           // Clear the do FFT flag

    //----------- Pass parameters to analysis task ---------------
    gSoundApplication->fAnalysis->fAnalysisTask->SetParams(fParams);
}



//=============================================================================
//  SetOscParams:
//      This method opens a dialog and allows the user to change
//      the parameters for the given oscillator.
//=============================================================================
pascal void TAdditiveSound::SetOscParams(long whichOsc)
{
    const           kOkButton   = 'Okbt';       // Consts for view ID's in resource file
    const           kMinimum    = 'ed01';
    const           kMaximum    = 'ed02';
    const           kDialog     = 'dlog';
    const           kAmpEnv     = 'env1';
    const           kFreqEnv    = 'env2';
    const short     kAddDialog  = 1005;

    TWindow         *aWindow;
```

```
    TDialogView     *aDialogView;
    IDType          dismisser;
    TEnvelopeView   *ampEnv, *freqEnv;
    TEditText       *minText, *maxText;
    Str255          tempText;
    long            tempNum;

    //------------------ Load the dialog resource -----------------
    aWindow = NewTemplateWindow (kAddDialog, nil);                  // Load the dialog from resource file

    //------------- Init dialog with current settings --------------
    aDialogView = (TDialogView *) (aWindow->FindSubView(kDialog));  // Find the dialog view
    ampEnv = (TEnvelopeView *) (aWindow->FindSubView(kAmpEnv));     // Find the amplitude evelope view
    freqEnv = (TEnvelopeView *) (aWindow->FindSubView(kFreqEnv));   // Find the frequency evelope view
    minText = (TEditText *)(aWindow->FindSubView(kMinimum));        // Find the minimum edit text
    maxText = (TEditText *) (aWindow->FindSubView(kMaximum));       // Find the maximum edit text

    ampEnv->SetEnvValues (this->fOps[whichOsc].ampEnvValues);       // Display current amp env values
    freqEnv->SetEnvValues (this->fOps[whichOsc].freqEnvValues);     // Display current freq env values
    NumToString(this->fOps[whichOsc].minFreq, tempText);            // Convert min to string
    minText->SetText(tempText, true);                               // Display in edit text, redraw
    NumToString(this->fOps[whichOsc].maxFreq, tempText);            // Convert max to string
    maxText->SetText(tempText, true);                               // Display in edit text, redraw

    //------------------ Display the dialog to the user -----------------
    dismisser = aDialogView->PoseModally();     // Display the dialog to the user
    aWindow->Close();

    //------------------ Store the new settings -----------------
    if (dismisser == kOkButton)                 // User clicked the OK button
    {
        ampEnv->GetEnvValues (this->fOps[whichOsc].ampEnvValues);       // Save changes to amplitude envelope
        freqEnv->GetEnvValues (this->fOps[whichOsc].freqEnvValues);     // Save changes to frequency envelope
        minText->GetText(tempText);                                     // Get the min edit text value
        StringToNum(tempText, &tempNum);                                // Convert to number, save
        this->fOps[whichOsc].minFreq = tempNum;
        maxText->GetText(tempText);                                     // Get the max edit text value
        StringToNum(tempText, &tempNum);                                // Convert to number, save
        this->fOps[whichOsc].maxFreq = tempNum;

        gSoundApplication->fAnalysis->Restart();                        // Restart the analysis tool
        gSoundApplication->UpdateViewWindow();                          // Redraw the sound view window
    }
}


//--------------------------------------------------------------------
//  ShowIt:
//      This procedure either hides or displays the Additive synthesis
//      buttons and installs the Additive voices if necessary. It is
//      called whenever the user changes synthesis methods.
//--------------------------------------------------------------------
pascal void TAdditiveSound::ShowIt(Boolean isShown)
{
    TOscButton *button;

    //------------- If "isShown" is true show the buttons ---------------
    button = (TOscButton *) (this->fEditorView->FindSubView('  5'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);
    button = (TOscButton *) (this->fEditorView->FindSubView('  6'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);
    button = (TOscButton *) (this->fEditorView->FindSubView('  7'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);
    button = (TOscButton *) (this->fEditorView->FindSubView('  8'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);

    //------------- Install the Additive Voices ---------------
    if (isShown)
    {
        pStrCpy( this->fAlgorithm , "\pAdditiveAnalysis");
        this->fEditorView->ForceRedraw();
        gSoundApplication->fVoiceManager->InstallVoices("\pAdditive");      // Install the voices on the DSP
        gSoundApplication->fAnalysis->SetAlg(fAlgorithm);                   // Update the analysis algorithm
        gSoundApplication->fAnalysis->Restart();                            // Restart the analysis tool
```

```
        gSoundApplication->UpdateViewWindow();                          // Redraw the sound view window
    }
}


//--------------------------------------------------------------------
//  Draw:
//      This method is used to draw the lines between the buttons
//      when using Additive synthesis. It is called anytime there is an
//      update to the screen when using Additive synthesis.
//--------------------------------------------------------------------
pascal void TAdditiveSound::Draw()
{
    PenSize (5,5);                  // Use a 5 pixel/5 pixel wide pen

    MoveTo (46, 112);               // Draw the lines between buttons
    LineTo (46, 123);
    MoveTo (94, 112);
    LineTo (94, 123);
    MoveTo (142, 112);
    LineTo (142, 123);
    MoveTo (190, 112);
    LineTo (190, 123);
    MoveTo (46, 123);
    LineTo (190, 123);

    PenNormal();                    // Set the pen back to normal
}

#pragma segment AReadFile
//--------------------------------------------------------------------
//  DoRead:
//      This method reads the parameter data from the disk.
//--------------------------------------------------------------------
pascal void TAdditiveSound::DoRead( short aRefNum )
{
    long        readBytes;
    short       count;

    readBytes = sizeof(float);
    //-------------- Oscillator frequencies ---------------
    for (count=0; count<8; count+=2) {
        FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fOps[count/2].minFreq));   // Read frequencies
        FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fOps[count/2].maxFreq));   // Read frequencies
    }

    //-------------- Amplitude envelopes ---------------
    for (count=0; count<24; count++)
        FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fOps[count/6].ampEnvValues[count%6 ]);        Read amp envelopes

    //-------------- Frequency envelopes ---------------
    for (count=0; count<24; count++)
        FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fOps[count/6].freqEnvValues[count%6 ]);     // Read freq envelopes


}


#pragma segment AWriteFile
//--------------------------------------------------------------------
//  DoWrite:
//      This method writes the parameter data to the disk.
//--------------------------------------------------------------------
pascal void TAdditiveSound::DoWrite( short aRefNum )
{
    long        writeBytes;
    short       count;

    writeBytes = sizeof(float);
    //-------------- Oscillator frequencies ---------------
    for (count=0; count<8; count+=2) {
        FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fOps[count/2].minFreq));          Write frequencies
        FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fOps[count/2].maxFreq));          Write frequencies
    }

    //-------------- Amplitude envelopes ---------------
    for (count=0; count<24; count++)
        FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fOps[count/6].ampEnvValues[count%6 ]);     // Write amp envelopes
```

```
    //-------------- Frequency envelopes / 100 ---------------
    for (count=0; count<24; count++)
        FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fOps[count/6].freqEnvValues[count%6]));  // Write freq envelopes
}
```

# TKarplus.cp

```
//-----------------------------------------------------------------------------
//
// NAME
//      TKarplus.cp
//
// SYNOPSIS
//      This file implements the Karplus/Strong synthesis sound class.
//
//      Author Jeff Boone, Oregon State University
//
//
//-----------------------------------------------------------------------------
#include    "USound.h"
#include    <Packages.h>


//-----------------------------------------------------------------------
//  IKarplusSound:
//      This method initializes the Karplus/Strong sound object. It should be
//      called immediately after instantiating an Karplus/Strong sound object.
//-----------------------------------------------------------------------
pascal void TKarplusSound::IKarplusSound()
{
        // Empty method; fill in any desired initialization
}


#pragma segment Main

//-----------------------------------------------------------------------
//  Generate:
//      This procedure sets up a parameter block and calls upon the
//      voice manager to play a Karplus/Strong tone. It is called when the
//      user clicks on the "speaker" button when the system is set to
//      do Karplus/Strong synthesis. The boolean parameter (showIt) is
//      set to true whenever we are just viewing the sound on the screen.
//-----------------------------------------------------------------------
pascal void TKarplusSound::Generate(Boolean showIt)
{

    //-------------- Number of samples ---------------
    fParams.param[0] = 25000.0;

    //-------------- Wave table size ---------------
    fParams.param[1] = float (Random() % 100) + 180.0;

    //-------------- Seed value for random table ---------------
    fParams.param[2] = float (Random());

     /-------------- Flags ---------------
    fParams.flags = 0x00000000;         // Initialize the flags

    if (showIt)
        fParams.flags |= kViewIt;       // View flag
    else
        fParams.flags &= ~kViewIt;      // View flag

    fParams.flags |= kHostReady;            // Set the host ready flag
    fParams.flags |= kNeedToInit;           // Set the need to init flag
    fParams.flags &= ~kDoFFT;               // Clear the DO_FFT flag

     /-------- Tell voice manager to play it ------------
    gSoundApplication->fVoiceManager->PlayVoice(fParams);
}


#pragma segment ASelCommand

//-----------------------------------------------------------------------
//  GenerateAnalysis:
//      This procedure sets up a parameter block and passes these to
//      the analysis task. Note that nothing else occurs until the user
//      interacts with analysis control pane..
//-----------------------------------------------------------------------
pascal void TKarplusSound::GenerateAnalysis()
{
    //-------------- Number of samples ---------------
    fParams.param[0] = 24960.0;
```

```
    //-------------- Wave table size ---------------
    fParams.param[1] = float (Random() % 100) + 180.0;

    //-------------- Seed value for random table ---------------
    fParams.param[2] = float (Random());

    //------------------ Flags ----------------------
    fParams.flags = 0x00000000;          // Initialize the flags
    fParams.flags |= kNeedToInit;        // Set the need to init flag
    fParams.flags |= kHostReady;         // Set the host ready flag
    fParams.flags &= ~kDoFFT;            // Clear the do FFT flag

    //----------- Pass parameters to analysis task ---------------
    gSoundApplication->fAnalysis->fAnalysisTask->SetParams(fParams);
}


//-----------------------------------------------------------------------
//  ShowIt:
//      This method either hides or displays the Karplus/Strong synthesis
//      button and installs the Karplus/Strong voices if necessary. It is
//      called whenever the user changes synthesis methods.
//-----------------------------------------------------------------------
pascal void TKarplusSound::ShowIt(Boolean isShown)
{
    TOscButton *button;

    //------------- If "isShown" is true show the buttons ---------------
    button = (TOscButton *) (this->fEditorView->FindSubView('    9'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);

    //------------- Install the Karplus/Strong voices ---------------
    if (isShown)
    {
        pStrCpy( this->fAlgorithm , "\pKarAnalysis");
        this->fEditorView->ForceRedraw();
        gSoundApplication->fVoiceManager->InstallVoices("\pKarplus");       // Install the voices on the DSP
        gSoundApplication->fAnalysis->SetAlg(fAlgorithm);                   // Update the analysis algorithm
        gSoundApplication->fAnalysis->Restart();                            // Restart the analysis tool
    }
}


//-----------------------------------------------------------------------
//  Draw:
//      This method can be filled if any special drawing needs to be
//      done for Karplus/Strong synthesis. (We don't do any)
//-----------------------------------------------------------------------
pascal void TKarplusSound::Draw()
{
        // Fill in any drawing code
}
```

# TFM.cp

```
//-----------------------------------------------------------------------------
//
// NAME
//      TFM.cp
//
// SYNOPSIS
//      This file implements the FM synthesis sound class.
//
//      Author Jeff Boone, Oregon State University
//
//
//  HISTORY
//
//-----------------------------------------------------------------------------
#include    "USound.h"
//#include  "TDsp.h"
#include    <Packages.h>


#pragma segment ASelCommand

//-----------------------------------------------------------------------
//  IFMSound:
//      This method initializes the FM sound object. It should be
//      called immediately after instantiating an FM sound object.
//-----------------------------------------------------------------------
pascal void IFMSound::IFMSound()
{
    int     count;

    //-------------- Set initial configuration ---------------
    this->fConfiguration = 1;

    //------- Initialize envelope values to something "reasonable" -------------
    for (count=0; count<4; count++)
    {
        this->fOps[count].envValues[0] = 10;
        this->fOps[count].envValues[1] = 98;
        this->fOps[count].envValues[2] = 75;
        this->fOps[count].envValues[3] = 70;
        this->fOps[count].envValues[4] = 60;
        this->fOps[count].envValues[5] = 5;
    }

    //-------- Set the oscillators to something "reasonable" ---------------
    this->fOps[0].frequency = 100;
    this->fOps[0].minEnv = 0;
    this->fOps[0].maxEnv = 0;
    this->fOps[0].opType = Modulator;

    this->fOps[1].frequency = 880;
    this->fOps[1].minEnv = 0;
    this->fOps[1].maxEnv = 0;
    this->fOps[1].opType = Modulator;

    this->fOps[2].frequency = 300;
    this->fOps[2].minEnv = 0;
    this->fOps[2].maxEnv = 2;
    this->fOps[2].opType = Modulator;

    this->fOps[3].frequency = 400;
    this->fOps[3].minEnv = 1;
    this->fOps[3].maxEnv = 3;
    this->fOps[3].opType = Carrier;

}

#pragma segment Main

//-----------------------------------------------------------------------
//  Generate:
//      This procedure sets up a parameter block and calls upon the
//      voice manager to play an FM tone. It is called when the user
//      clicks on the "speaker" button when the system is set to
//      do FM synthesis. The boolean parameter (showit) is set to true
//      whenever we are just viewing the sound on the screen.
//-----------------------------------------------------------------------
pascal void IFMSound::Generate(Boolean showIt)
```

```
}
    int count;

    //-------------- Number of samples ---------------
    fParams.param[0] = 24960.0;

    //-------------- Oscillator frequencies ---------------
    for (count=0; count<4; count++)
        fParams.param[count+1] = float (this->fOps[count].frequency);

    //-------------- Min/Max envelope values ---------------
    for (count=0; count<6; count+=2)  {
        fParams.param[count+5] = float (this->fOps[count/2].minEnv);
        fParams.param[count+6] = float (this->fOps[count/2].maxEnv);
    }

    //-------------- Envelop values/100 ---------------
    for (count=0; count<24; count++)
        fParams.param[count+11] = float (this->fOps[count/6].envValues[count%6])/100.0;

    //-------------- Flags ---------------
    fParams.flags = 0x00000000;          // Initialize the flags

    if (showIt)
        fParams.flags |= kViewIt;        // View flag set to true
    else
        fParams.flags &= ~kViewIt;       // View flag set to false

    fParams.flags |= kHostReady;         // Set the host ready flag
    fParams.flags &= ~kDoFFT;            // Clear the do FFT flag

    //-------- Tell voice manager to play it ------------
    gSoundApplication->fVoiceManager->PlayVoice(fParams);

}

#pragma segment ASelCommand

//--------------------------------------------------------------------
//  GenerateAnalysis:
//      This procedure sets up a parameter block and passes these to
//      the analysis task. Note that nothing else occurs until the user
//      interacts with analysis control panel.
//--------------------------------------------------------------------
pascal void TFMSound::GenerateAnalysis()
{
    int count;

    //-------------- Number of samples ---------------
    fParams.param[0] = 24960.0;

    //-------------- Oscillator frequencies ---------------
    for (count=0; count<4; count++)
        fParams.param[count+1] = float (this->fOps[count].frequency);

    //-------------- Min/Max envelope values ---------------
    for (count=0; count<6; count+=2)  {
        fParams.param[count+5] = float (this->fOps[count/2].minEnv);
        fParams.param[count+6] = float (this->fOps[count/2].maxEnv);
    }

    //-------------- Envelop values/100 ---------------
    for (count=0; count<24; count++)
        fParams.param[count+11] = float (this->fOps[count/6].envValues[count%6])/100.0;

    //-------------- Flags ---------------
    fParams.flags = 0x00000000;          // Initialize the flags
    fParams.flags |= kHostReady;         // Set the host ready flag
    fParams.flags &= ~kDoFFT;            // Clear the do FFT flag

    //---------- Pass parameters to analysis task ---------------
    gSoundApplication->fAnalysis->fAnalysisTask->SetParams(fParams);
}


//--------------------------------------------------------------------
//  SetConfig:
//      This procedure sets the current FM synthesis configuration (1-7)
```

```
//      to the one chosen by the user. It is called when the user
//      changes configuration using the pop-up menu. It basically
//      rearranges the buttons and installs the corresponding DSP
//      algorithm.
//--------------------------------------------------------------------
pascal void TFMSound::SetConfig(int theConfig)
{
    TOscButton *button1, *button2, *button3, *button4;
    Str31       theAlg;
    Str255      numString;

    this->fConfiguration = theConfig;

    //----------- Find all 4 buttons ---------------
    button1 = (TOscButton *)(this->fEditorView->FindSubView('   1'));
    button2 = (TOscButton *)(this->fEditorView->FindSubView('   2'));
    button3 = (TOscButton *)(this->fEditorView->FindSubView('   3'));
    button4 = (TOscButton *)(this->fEditorView->FindSubView('   4'));

    //-------------------------------------------------------------
    //---------------------- Move the buttons ---------------------
    //-------------------------------------------------------------
    switch (theConfig) {
        case 1:     {
                button1->Locate(96, 16, true);
                button2->Locate(96, 64, true);
                button3->Locate(96, 112, true);
                button4->Locate(96, 160, true);
                this->fOps[0].opType = Modulator;
                this->fOps[1].opType = Modulator;
                this->fOps[2].opType = Modulator;
                this->fOps[3].opType = Carrier;
                break;
            };
        case 2:     {
                button1->Locate(64, 64, true);
                button2->Locate(128, 64, true);
                button3->Locate(96, 112, true);
                button4->Locate(96, 160, true);
                this->fOps[0].opType = Modulator;
                this->fOps[1].opType = Modulator;
                this->fOps[2].opType = Modulator;
                this->fOps[3].opType = Carrier;
                break;
            };
        case 3:     {
                button1->Locate(128, 64, true);
                button2->Locate(128, 112, true);
                button3->Locate(64, 112, true);
                button4->Locate(96, 160, true);
                this->fOps[0].opType = Modulator;
                this->fOps[1].opType = Modulator;
                this->fOps[2].opType = Modulator;
                this->fOps[3].opType = Carrier;
                break;
            };
        case 4:     {
                button1->Locate(64, 112, true);
                button2->Locate(64, 160, true);
                button3->Locate(128, 112, true);
                button4->Locate(128, 160, true);
                this->fOps[0].opType = Modulator;
                this->fOps[1].opType = Carrier;
                this->fOps[2].opType = Modulator;
                this->fOps[3].opType = Carrier;
                break;
            };
        case 5:     {
                button1->Locate(96, 112, true);
                button2->Locate(48, 160, true);
                button3->Locate(96, 160, true);
                button4->Locate(144, 160, true);
                this->fOps[0].opType = Modulator;
                this->fOps[1].opType = Carrier;
                this->fOps[2].opType = Carrier;
                this->fOps[3].opType = Carrier;
                break;
            }
```

```
        case 6:     {
                button1->Locate(128, 112, true);
                button2->Locate(48, 160, true);
                button3->Locate(96, 160, true);
                button4->Locate(144, 160, true);
                this->fOps[0].opType = Modulator;
                this->fOps[1].opType = Carrier;
                this->fOps[2].opType = Carrier;
                this->fOps[3].opType = Carrier;
                break;
            }
        case 7:     {
                button1->Locate(144, 112, true);
                button2->Locate(48, 160, true);
                button3->Locate(96, 160, true);
                button4->Locate(144, 160, true);
                this->fOps[0].opType = Modulator;
                this->fOps[1].opType = Carrier;
                this->fOps[2].opType = Carrier;
                this->fOps[3].opType = Carrier;
                break;
            }
    } // Case

    //------------------ Redraw the screen -------------------------
    this->fEditorView->ForceRedraw();

    //------------------ Install the DSP algoritms -----------------
    NumToString(this->fConfiguration, numString);
    pStrCpy( theAlg,  "\pFMAlg ");                  // Build the DSP algorithm name (FMAlgx)
    theAlg[6] = numString[1];                       // Insert the number into the string
    gSoundApplication->fVoiceManager->InstallVoices(theAlg);        // Install the voices on the DSP

    //------------------ Update the analysis algorithm -----------------
    pStrCpy( this->fAlgorithm , "\pFMAnalysis ");
    this->fAlgorithm[11] = numString[1];
}


//=========================================================================
//  SetOscParams:
//      This method opens a dialog and allows the user to change
//      the parameters for the given oscillator.
//=========================================================================
pascal void TFMSound::SetOscParams(int whichOsc)
{
    const   kOkButton   = 'Okbt';       // Consts for view ID's in resource file
    const   kMinimum    = 'ed01';
    const   kMaximum    = 'ed02';
    const   kFreq       = 'ed03';
    const   kDialog     = 'dlog';
    const   kEnvView    = 'envi';
    const   kModDialog  = 1000;
    const   kCarDialog  = 1004;

    TWindow         *aWindow;
    TDialogView     *aDialogView;
    IDType          dismisser;
    TEnvelopeView   *envView;
    TEditText       *minText, *maxText, *freqText;
    Str255          tempText;
    Long            tempNum;

    //------------------ Load the dialog resource -----------------
    if (this->fOps[whichOsc].opType == Modulator)
        aWindow = NewTemplateWindow (kModDialog, nil);      // Load the modulator dialog from resource file
    else
        aWindow = NewTemplateWindow (kCarDialog, nil);      // Load the carrier dialog from resource file

    //------------- Init dialog with current settings ---------------
    aDialogView = (TDialogView *) (aWindow->FindSubView(kDialog));   // Find the dialog view
    envView = (TEnvelopeView *) (aWindow->FindSubView(kEnvView));    // Find the evelope view

    freqText = (TEditText *) (aWindow->FindSubView(kFreq));         // Find the frequency edit text
    envView->SetEnvValues (this->fOps[whichOsc].envValues);         // Display current env values
    NumToString(this->fOps[whichOsc].frequency, tempText);
    freqText->SetText(tempText, true);                              // Set the current frequency
```

```
    if (this->fOps[whichOsc].opType == Modulator)
    {
        minText = (TEditText *) (aWindow->FindSubView(kMinimum));   // Find the minimum edit text
        maxText = (TEditText *) (aWindow->FindSubView(kMaximum));   // Find the maximum edit text

        NumToString(this->fOps[whichOsc].minEnv, tempText);         // Convert min to string
        minText->SetText(tempText, true);                           // Display in edit text, redraw
        NumToString(this->fOps[whichOsc].maxEnv, tempText);
        maxText->SetText(tempText, true);
    }

    //------------------ Display the dialog to the user -----------------
    dismisser = aDialogView->PoseModally();
    aWindow->Close();                                               // Close the dialog

    //------------------ Store the new settings -----------------
    if (dismisser == kOkButton)                                     // User clicked the OK button
    {
        envView->GetEnvValues (this->fOps[whichOsc].envValues);     // Save changes to envelope
        freqText->GetText(tempText);                                // Save frequency
        StringToNum(tempText, &tempNum);                            // Convert to number, save
        this->fOps[whichOsc].frequency = tempNum;

        if (this->fOps[whichOsc].opType == Modulator)
        {
            minText->GetText(tempText);                             // Get the edit text value
            StringToNum(tempText, &tempNum);                        // Convert to number, save
            this->fOps[whichOsc].minEnv = tempNum;
            maxText->GetText(tempText);
            StringToNum(tempText, &tempNum);
            this->fOps[whichOsc].maxEnv = tempNum;
        }
        gSoundApplication->fAnalysis->Restart();                    // Restart the analysis tool
        gSoundApplication->UpdateViewWindow();                      // Redraw the sound view window
    }
}


//-----------------------------------------------------------------------
//  GetConfig:
//      Utility function to return the current configuration.
//-----------------------------------------------------------------------
pascal int  TFMSound::GetConfig()
{
    return fConfiguration;
}


//-----------------------------------------------------------------------
//  ShowIt:
//      This method either hides or displays the FM synthesis
//      buttons and installs the Additive voices if necessary. It is
//      called whenever the user changes synthesis methods.
//-----------------------------------------------------------------------
pascal void TFMSound::ShowIt(Boolean isShown)
{
    TOscButton  *button;
    TMyPopup    *popup;
    Str31       theAlg;
    Str255      numString;

    //------------- If "isShown" is true show the buttons ---------------
    button = (TOscButton *)(this->fEditorView->FindSubView('  1'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);
    button = (TOscButton *) (this->fEditorView->FindSubView('  2'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);
    button = (TOscButton *) (this->fEditorView->FindSubView('  3'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);
    button = (TOscButton *) (this->fEditorView->FindSubView('  4'));
    button->fShown = isShown;
    button->ViewEnable(isShown, false);
    popup = (TMyPopup *) (this->fEditorView->FindSubView('pop1'));
    popup->fShown = isShown;
    button->ViewEnable(isShown, false);
```

```
    //------------- Install the FM voices ---------------
    if (isShown)
    {
        NumToString(this->fConfiguration,numString);
        pStrCpy( theAlg,  "\pFMAlg ");
        theAlg[6] = numString[1];
        pStrCpy( this->fAlgorithm , "\pFMAnalysis ");
        this->fAlgorithm[11] = numString[1];

        gSoundApplication->fAnalysis->SetAlg(fAlgorithm);              // Update the analysis algorithm
        gSoundApplication->fAnalysis->Restart();                       // Restart the analysis tool
        gSoundApplication->fVoiceManager->InstallVoices(theAlg);       // Install the voices on the DSP
        this->fEditorView->ForceRedraw();
    }

}


//-----------------------------------------------------------------------
//  Draw:
//      This method is used to draw the lines between the buttons
//      when using FM synthesis. It is called anytime there is an
//      update to the screen when using FM synthesis.
//-----------------------------------------------------------------------
pascal void TFMSound::Draw()
{
    PenSize (5,5);                      // Use a 5 pixel/5 pixel pen size
    switch (this->fConfiguration) {
        case 1: {
                MoveTo (110, 48);       // Draw the lines between the buttons
                LineTo (110, 59);
                MoveTo (110, 96);
                LineTo (110, 107);
                MoveTo (110, 144);
                LineTo (110, 155);
                break;
            };
        case 2: {
                MoveTo (80, 96);
                LineTo (80, 99);
                LineTo (144, 99);
                LineTo (144, 96);
                MoveTo (110, 99);
                LineTo (110, 107);
                MoveTo (110, 144);
                LineTo (110, 155);
                break;
            };
        case 3:     {
                MoveTo (144, 96);
                LineTo (144, 107);
                MoveTo (80, 144);
                LineTo (80, 147);
                LineTo (144, 147);
                LineTo (144, 144);
                MoveTo (110, 147);
                LineTo (110, 155);
                break;
            };
        case 4:     {
                MoveTo (80, 144);
                LineTo (80, 155);
                MoveTo (144, 144);
                LineTo (144, 155);
                MoveTo (80, 192);
                LineTo (80, 195);
                LineTo (144, 195);
                LineTo (144, 192);
                break;
            };
        case 5:     {
                MoveTo (112, 144);
                LineTo (112, 155);
                MoveTo (64, 155);
                LineTo (64, 192);
                LineTo (160, 192);
                LineTo (160, 155);
```

```
                MoveTo (64, 192);
                LineTo (64, 197);
                LineTo (160, 197);
                LineTo (160, 192);
                MoveTo (112, 192);
                LineTo (112, 192);
                break;
            };
        case 6:     {
                MoveTo (144, 144);
                LineTo (144, 149);
                MoveTo (112, 155);
                LineTo (112, 149);
                LineTo (160, 149);
                LineTo (160, 155);
                MoveTo (64, 192);
                LineTo (64, 197);
                LineTo (160, 197);
                LineTo (160, 192);
                MoveTo (112, 192);
                LineTo (112, 192);
                break;
            };
        case 7:     {
                MoveTo (160, 144);
                LineTo (160, 155);
                MoveTo (64, 192);
                LineTo (64, 197);
                LineTo (160, 197);
                LineTo (160, 192);
                MoveTo (112, 192);
                LineTo (112, 192);
                break;
            };
    } // Case
    PenNormal();
}


#pragma segment AReadFile
//--------------------------------------------------------------------
//  DoRead:
//      This method reads the sound synthesis method, sets the
//      current synth method and call the generator to read its
//      data.
//--------------------------------------------------------------------
pascal void TFMSound::DoRead( short aRefNum )
{
    long        readBytes;
    short       count;
    TMyPopup    *popup;

    readBytes = sizeof(int);
    FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fConfiguration));  // Read the name of the generator

    readBytes = sizeof(float);
    //-------------- Oscillator frequencies ---------------
    for (count=0; count<4; count++)
        FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fOps[count].frequency));    // Read frequencies

    //-------------- Min/Max envelope values ---------------
    for (count=0; count<6; count+=2)
        FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fOps[count/2].minEnv));     // Read the min/max envelopes
        FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fOps[count/2].maxEnv));
    }

    //-------------- Envelop values/100 ---------------
    for (count=0; count<24; count++)
        FailOSErr(FSRead(aRefNum, &readBytes, (Ptr) &fOps[count/6].envValues[count%6]));     // Read the env values


    SetConfig( fConfiguration );
    popup = (TMyPopup *) (fEditorView->FindSubView('popl'));        // Find the popup menu
    popup->SetCurrentItem( fConfiguration, true );                  // Update the popup menu
}


#pragma segment AWriteFile
```

```
//----------------------------------------------------------------------
//  DoWrite:
//      This method writes the sound synthesis method and calls the
//      generator to write its data.
//----------------------------------------------------------------------
pascal void TFMSound::DoWrite( short aRefNum )
{
    long        writeBytes;
    short       count;

    writeBytes = sizeof(int);
    FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fConfiguration));            // Write the name of the generator

    writeBytes = sizeof(float);
    //-------------- Oscillator frequencies --------------
    for (count=0; count<4; count++)
        FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fOps[count].frequency));     // Write frequencies

    //-------------- Min/Max envelope values ---------------
    for (count=0; count<6; count+=2)  {
        FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fOps[count/2].minEnv));      // Write the min/max envelopes
        FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fOps[count/2].maxEnv));
    }

    //-------------- Envelop values/100 ---------------
    for (count=0; count<24; count++)
        FailOSErr(FSWrite(aRefNum, &writeBytes, (Ptr) &fOps[count/6].envValues[count%6]));  // Write the env values
}
```

# TDsp.cp

```
//-----------------------------------------------------------------------------
//
// NAME
//      TDsp.cp
//
// SYNOPSIS
//      This file implements the DSP specific functions such as loading
//      an algorithm onto the DSP, etc. Since these methods don't directly
//      interact with MacApp, I have chosen to use the standard C++
//      constructor and destructor methods (as opposed to the IMyClass() and
//      Free() MacApp methods).
//
// COPYRIGHT
//      Author Jeff Boone, Oregon State University
//
//  HISTORY
//
//-----------------------------------------------------------------------------

#include <Events.h>
#include <OSUtils.h>
#include <Strings.h>
#include <StdLib.h>
#include <stdio.h>
#include <strings.h>
#include <string.h>
#include <Memory.h>
#include <Resources.h>
#include <Errors.h>

#include "dspTaskDispatcher.h"
#include "USound.h"

#define kNoName "\pNot Named Yet"

#pragma segment dsp_code

//-----------------------------------------------------------------------
//  NewTrueAddress:
//      This is a utility method that creates a block of memory that
//      is to be shared by both the DSP and the host.
//-----------------------------------------------------------------------
Address NewTrueAddress( Size theSize)
{
    Ptr theNewPtr;              // Logical pointer to the data
    Ptr theNewPPtr;             // Physical pointer to the data
    Address tAddress;

    if (theSize != 0)
        if ( dspNewTruePtr( theSize, &theNewPtr, false) != noErr)   // Ask DSPManager for memory block
        {
            tAddress = kNilAddress;     // Error creating the memory
        }
        else
        {
            dspLogical2Physical( theNewPtr, &theNewPPtr );      // Convert the logical address to physical
            tAddress.logical = theNewPtr;                       // Store the pointers
            tAddress.physical = theNewPPtr;
        }
    else
        tAddress = kNilAddress;         // User passed 0 for size

    return tAddress;
}


//-----------------------------------------------------------------------
//  pStrCat & pStrCpy:
//      These are utility functions that make a copy of a pascal string
//      and convert a pascal string to a C string, respectively.
//-----------------------------------------------------------------------
void pStrCat( Str31 outstr, Str31 instr)
{
    short i;
    for (i=1; i <= instr[0] ; i++)
        outstr[ outstr[0] + i ] = instr[i];
    outstr[0] += instr[0];
}
```

```
void pStrCpy( Str31 outstr,  Str31 instr)
{
    short i;
    for (i=0; i <= instr[0]; i++)
        outstr[i] = instr[i];
    outstr[i] = '\0';
}


/******************************  D S P   C L A S S  ************************************/
//----------------------------------------------------------------------
//  DSP Class:
//      This is the class that deals directly with the DSPManager. It
//      handles creating new tasks for the DSP to execute.
//----------------------------------------------------------------------

//----------------------------------------------------------------------
//  TDsp:
//      This is the constructor for the DSP class.
//----------------------------------------------------------------------
TDsp::TDsp()
{
    // We don't need anything yet
}

//----------------------------------------------------------------------
//  ~TDsp:
//      This is the destructor for the DSP class.
//----------------------------------------------------------------------
TDsp::~TDsp()
{
    // We don't need anything yet
}



#pragma segment Main

//----------------------------------------------------------------------
//  InstallTask:
//      This method handles installing a task on the DSP. Note that
//      the task is not activated; this must be done explicitly.
//----------------------------------------------------------------------
OSErr   TDsp::InstallTask(ProcessType theType, TDspTask* newTask,
                          PortID whichPort, ReferenceQualifier where)
{
    OSErr           tOSErr;
    TaskHeader*     tPtr;

    //---------------- Check for nil task ---------------
    if (newTask == nil)
        debugstr("Oops, trying to install a nil task.");

    //---------------- Insert the task ---------------
    tPtr = (TaskHeader *)newTask->Where().logical;
    tOSErr = dspInsertTask( theType, whichPort, tPtr, where);

    //---------------- Set up the port ---------------
    if (tOSErr == noErr)
        newTask->WhichPort( whichPort);

    return tOSErr;
}


//----------------------------------------------------------------------
//  SuspendTask:
//      This method handles deactivating an active task. Note that the
//      task will still be allocated and in the task list.
//----------------------------------------------------------------------
void TDsp::SuspendTask(TDspTask* theTask, Boolean immed)
{
    if (immed)
        //---------------- Turn the task off kNow ----------------
        ((TaskHeader *)theTask->Where().logical)->th_flags &= ~dspTaskActive;
    else
        //---------------- Synchro-stop the task ----------------
```

```
        ((TaskHeader *)theTask->Where().logical)->th_flags |= dspTaskNxtStateInActive;
}


//--------------------------------------------------------------------
//  ResumeTask:
//      This method handles activating an inactive task.
//--------------------------------------------------------------------
void TDsp::ResumeTask(TDspTask* theTask, Boolean immed)
{
    if (immed)
        //--------------- Turn the task on kNow -----------------
        ((TaskHeader *)theTask->Where().logical)->th_flags |= dspTaskActive;
    else
        //--------------- Synchro-start the task -----------------
        ((TaskHeader *)theTask->Where().logical)->th_flags |= dspTaskNxtStateActive;
}


//--------------------------------------------------------------------
//  RemoveTask:
//      This method removes a task from the DSP list of tasks. Note that
//      it is still taking up memory after this call.
//--------------------------------------------------------------------
OSErr TDsp::RemoveTask(ProcessType theType, TDspTask* theTask)
{
    OSErr tOSErr;

    //----------------- Remove the task from the list --------------
    tOSErr = dspRemoveTask( theType, theTask->WhichPort(), (TaskHeader *)theTask->Where().logical );
    tOSErr = noErr;
    return tOSErr;
}


//--------------------------------------------------------------------
//  KickIt:
//      This method tells the DSP to look at the tasks that we've set
//      up. This should be the last method called when you have built,
//      installed, and activated a task.
//--------------------------------------------------------------------
OSErr TDsp::KickIt()
{
    return dspUpdateList(InterruptProcessID);
}


//************************  D S P   T A S K   C L A S S  ********************************
//--------------------------------------------------------------------
//  TDspTask Class:
//      This class is used for building a DSP task. The idea is to
//      build the task in Macintosh memory and then passing it off
//      to the DSP to execute it.
//--------------------------------------------------------------------

#pragma segment Task
//--------------------------------------------------------------------
//  TDspTask:
//      This is the constructor for the TDspTask class.
//--------------------------------------------------------------------
TDspTask::TDspTask()
{
    //------------ Allocate the memory for the header -----------
    dspNewTrapPtr(TaskHeaderSize, (Ptr*)&fTaskHeader, false);
    if (fTaskHeader == nil )
        debugstr("Couldn't allocate TaskHeader");

    //------------ Initialize the fields -----------
    fTaskHeader->th_flags = 0x0;
    fTaskHeader->th_firstCodeHeader = kNilAddress;
    fTaskHeader->th_nextTaskHeader = kNilAddress;
    fTaskHeader->th_RealTimeRequirements.rtr_hostBusCyclesPerSample  = 0;
    fTaskHeader->th_RealTimeRequirements.rtr_dspCPUCyclesPerSample  = 0;
    pStrCpy(fTaskHeader->th_name, kNoName);
    fFirstTCodeBlock = nil;
```

```
//--------------------------------------------------------------------
//  ~TDspTask:
//      This is the destructor for the TDspTask class. It destroys
//      the code blocks and the task header.
//--------------------------------------------------------------------
TDspTask::~TDspTask()
{
    TDspCodeBlock*  tCB;

    //--------- Destroy all the code block headers -----------------
    while (fFirstTCodeBlock != nil)
    {
        tCB = FirstCodeBlock()->NextCodeBlock();
        delete FirstCodeBlock();
        FirstCodeBlock( tCB );
    }

    //--------- Destroy the task header -----------------
    dspDisposeTruePtr( (Ptr)fTaskHeader );
}


//--------------------------------------------------------------------
//  InstallCodeBlock:
//      This method installs a code block into the task. Normally,
//      you read in the DSP code from the resource file and then
//      install it into the task.
//--------------------------------------------------------------------
void TDspTask::InstallCodeBlock( TDspCodeBlock* newCodeBlock )
{
    TDspCodeBlock*  tempTDspCodeBlock;

    tempTDspCodeBlock = FirstCodeBlock();           // Get first code block
    FirstCodeBlock( newCodeBlock );                 // Install the new code block
    if (tempTDspCodeBlock != nil)
        FirstCodeBlock()->NextCodeBlock( tempTDspCodeBlock );       // Patch in the old first code block
}


//--------------------------------------------------------------------
//  Name:
//      This utility method either returns the name of the task (no
//      arguments) or sets the name of the task (pass name as an
//      agrument).
//--------------------------------------------------------------------
void TDspTask::Name(Str31 newName)
{
    pStrCpy(fTaskHeader->th_name, newName);
}

Str31 TDspTask::Name()
{
    return fTaskHeader->th_name;
}



#pragma segment Main
//--------------------------------------------------------------------
//  Where:
//      This is a commonly used method that returns the Address of
//      the dspTask.
//--------------------------------------------------------------------
Address TDspTask::Where()
{
    Address     tAddress;

    tAddress.logical = (Ptr)fTaskHeader;                            // Retrieve the logical ptr from field
    dspLogical2Physical( tAddress.logical, &tAddress.physical);     // Convert to physical address
    return tAddress;
}

#pragma segment Task


//--------------------------------------------------------------------
//  BusCycles:
//      This method either returns the current number of DSP bus cycles
```

```
//      used by the task (no argument) or sets the number of DSP bus
//      cycles (pass cycles as argument). These figures are used by
//      the DSP to figure out if it has enough real time bandwith to
//      handle processing the task.
//-----------------------------------------------------------------------
unsigned long TDspTask::BusCycles()
{
    //------------- Return the current bus loading -------------
    return fTaskHeader->th_RealTimeRequirements.rtr_hostBusCyclesPerSample;
}

void TDspTask::BusCycles( unsigned long  newBusCycleCount )
{
    //------------- Set the new bus loading -------------
    fTaskHeader->th_RealTimeRequirements.rtr_hostBusCyclesPerSample = newBusCycleCount;
}


//-----------------------------------------------------------------------
//  DSPCycles:
//      This method either returns the current number of DSP cycles
//      used by the task (no argument) or sets the number of DSP
//      cycles (pass cycles as argument). These figures are used by
//      the DSP to figure out if it has enough real time bandwith to
//      handle processing the task.
//-----------------------------------------------------------------------
unsigned long TDspTask::DSPCycles()
{
    //------------- Return the current cpu loading -------------
    return fTaskHeader->th_RealTimeRequirements.rtr_dspCPUCyclesPerSample;
}

void TDspTask::DSPCycles( unsigned long  newDSPCycleCount )
{
    //------------- Return the current cpu loading -------------
    fTaskHeader->th_RealTimeRequirements.rtr_dspCPUCyclesPerSample = newDSPCycleCount;
}



//-----------------------------------------------------------------------
//  FirstCodeBlock:
//      This method either returns the first code block for the task
//      (no arguments) or sets the first code block (pass the new code
//      block as an argument).
//-----------------------------------------------------------------------
TDspCodeBlock*  TDspTask::FirstCodeBlock()
{
    return fFirstTCodeBlock;
}

void    TDspTask::FirstCodeBlock( TDspCodeBlock* newFirstCodeBlock )
{
    if (newFirstCodeBlock  != nil)
        fTaskHeader->th_firstCodeHeader = newFirstCodeBlock->Where();
    else
        fTaskHeader->th_firstCodeHeader = kNilAddress;

    fFirstTCodeBlock = newFirstCodeBlock;
}



//-----------------------------------------------------------------------
//  NextTaskHeader:
//      This method either returns the header of the next task
//      (no arguments) or sets the header of the next task (pass the
//      new header as an argument).
//-----------------------------------------------------------------------
TDspTask*       TDspTask::NextTaskHeader()
{
    return fNextTask;
}

void    TDspTask::NextTaskHeader( TDspTask* newNextTask )
{
    if (newNextTask != nil)
        fTaskHeader->th_nextTaskHeader = newNextTask->Where();
```

```
    else
        fTaskHeader->th_nextTaskHeader = kNilAddress;

    fNextTask = newNextTask;
}


/************************  D S P   D A T A   B L O C K   C L A S S  ************************/
//=========================================================================
//  TDspDataBlock Class:
//      The DSPManager has a set of routines that allow easy management
//      of data buffers. The TDspDataBlock class places a wrapper around
//      these routines.
//=========================================================================

#pragma segment DataBlock
//=========================================================================
//  TDspDataBlock:
//      This is the constructor for the TDspDataBlock class. It must be
//      passed the size of the data block that you wish to create. It
//      will then initialize all the fields to reasonable values.
//=========================================================================
TDspDataBlock::TDspDataBlock( Size blokSize )
{
    //------------- Create a new data block header ----------------
    dspNewTruePtr(DataHeaderSize, (Ptr*)&fDataHeader, false);

    if (fDataHeader == nil ) return;            // There was an error

    //------------- Create the data block ----------------
    fDataHeader->dh_haddr = NewTrueAddress( blokSize );
    if (fDataHeader->dh_haddr.logical == nil )
        fDataHeader->dh_hsize = nil;            // There was an error
    else
        fDataHeader->dh_hsize = blokSize;

    //------------- Initialize the fields ----------------
    fDataHeader->dh_write           = 0x0;
    fDataHeader->dh_read            = 0x0;
    fDataHeader->dh_flags           = 0x0;
    fDataHeader->dh_datatype        = 'bugr';
    fDataHeader->dh_interruptvec    = nil;
    fDataHeader->dh_userinfo        = nil;
}



//=========================================================================
//  ~TDspDataBlock:
//      This is the destructor for the TDspDataBlock class. It disposes
//      the memory used by the data block and the header.
//=========================================================================
TDspDataBlock::~TDspDataBlock()
{
    //---------- Dispose the data block -------------
    if (fDataHeader->dh_haddr.logical != nil)
        dspDisposeTruePtr( fDataHeader->dh_haddr.logical );

    //---------- Dispose the header -------------
    dspDisposeTruePtr( (Ptr)fDataHeader );
}



//=========================================================================
//  WhereHeader:
//      This method returns the Address of the header of the data
//      block.
//=========================================================================
Address TDspDataBlock::WhereHeader()
{
    Address     tAddress;

    tAddress.logical = (Ptr)fDataHeader;
    dspLogical2Physical( tAddress.logical, &tAddress.physical );
    return tAddress;
}
```

```
//----------------------------------------------------------------------
//  WhereData:
//      This method returns the Address of the data in the data
//      block.
//----------------------------------------------------------------------
Address TDspDataBlock::WhereData()
{
    Address     tAddress;

    tAddress.logical = fDataHeader->dh_haddr.logical;
    dspLogical2Physical( tAddress.logical, &tAddress.physical );
    return tAddress;
}



//----------------------------------------------------------------------
//  Size:
//      This method returns the size of the data block.
//----------------------------------------------------------------------
Size    TDspDataBlock::Size()
{
    return fDataHeader->dh_hsize;
}


//----------------------------------------------------------------------
//  Status:
//      This method returns one flag from the dh_flags field in the
//      data block (pass the flag that you're interested in).
//----------------------------------------------------------------------
Boolean TDspDataBlock::Status( short bitNum )
{
    return ( (((long)0x1 << bitNum) & fDataHeader->dh_flags) != 0);
}


//----------------------------------------------------------------------
//  SetStatus:
//      This method sets one flag in the dh_flags field in the
//      data block (pass the flag that you want to set).
//----------------------------------------------------------------------
void    TDspDataBlock::SetStatus( short bitNum )
{
    fDataHeader->dh_flags |= ( (long)0x01 << bitNum );
}


//----------------------------------------------------------------------
//  ResetStatus:
//      This method clears one flag (or all flags in the dh_flags field
//      in the data block (pass the flag that you want to clear or -1 for
//      all flags).
//----------------------------------------------------------------------
void    TDspDataBlock::ResetStatus( short bitNum )
{
    if ( bitNum < 0 )
        fDataHeader->dh_flags = 0;
    else
        fDataHeader->dh_flags &= !( (long)0x01 << bitNum );
}


//----------------------------------------------------------------------
//  InstallIntVec:
//      This method sets up the interrupt vector for the data block.
//      The DSPManager will call this routine in accordance with the
//      flags that you have established for the data block.
//----------------------------------------------------------------------
void    TDspDataBlock::InstallIntVec( ProcPtr intRoutine )
{
    fDataHeader->dh_interruptvec = intRoutine;


//----------------------------------------------------------------------
```

```
//  UserVar:
//      This method either sets the dh_userinfo field of the data block
//      (pass the value to set) or returns the current setting (no
//      arguments).
//---------------------------------------------------------------------
long    TDspDataBlock::UserVar()
{
    return fDataHeader->dh_userinfo;
}

void    TDspDataBlock::UserVar( long newVar )
{
    fDataHeader->dh_userinfo = newVar;
}


//---------------------------------------------------------------------
//  DataType:
//      This method either sets the data type of the data block
//      (pass the type to set) or returns the current data type (no
//      arguments).
//---------------------------------------------------------------------
unsigned long   TDspDataBlock::DataType()
{
    return fDataHeader->dh_datatype;
}

void    TDspDataBlock::DataType( unsigned long newVar )
{
    fDataHeader->dh_datatype = newVar;
}


//---------------------------------------------------------------------
//  Name:
//      This method either sets the name of the data block
//      (pass the name to set) or returns the current name (no
//      arguments).
//---------------------------------------------------------------------
void    TDspDataBlock::Name(Str31 newName)
{
    pStrCpy(fDataHeader->dh_name, newName);
}

Str31   TDspDataBlock::Name()
{
    return fDataHeader->dh_name;
}


//---------------------------------------------------------------------
//  ReadIndex:
//      This method either sets the read index of the data block
//      (pass the index to set) or returns the current read index (no
//      arguments).
//---------------------------------------------------------------------
void    TDspDataBlock::ReadIndex(unsigned long newIndex)
{
    fDataHeader->dh_read = newIndex;
}

unsigned long   TDspDataBlock::ReadIndex()
{
    return fDataHeader->dh_read;
}


//---------------------------------------------------------------------
//  WriteIndex:
//      This method either sets the write index of the data block
//      (pass the index to set) or returns the current write index (no
//      arguments).
//---------------------------------------------------------------------
void    TDspDataBlock::WriteIndex(unsigned long newIndex)
{
    fDataHeader->dh_write = newIndex;
}
```

```
unsigned long   TDspDataBlock::WriteIndex()
{
    return fDataHeader->dh_write;
}

/*************************  D S P  P R B  C L A S S  ***************************/
//-----------------------------------------------------------------------
//  TDspPRB Class:
//      The DSPManager uses "Partial Result Buffers" which allow many
//      applications to sum their sound data into. At the end of every
//      frame, the DSP takes the data from these PRB's and packs it
//      into the correct format for the DMA buffer to the D/A converter.
//      The TDspPRB class places a wrapper around the routines to manage
//      these buffers locally.
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
//  TDspPRB:
//      This is the constructor for the TDspPRB class. Pass the ID of
//      the port in which you wish to sum into.
//-----------------------------------------------------------------------
TDspPRB::TDspPRB(short whichPort) : TDspDataBlock(0)
{
    OSErr   err;

    //--------------- Get the address of the PRB for the desired port --------------
    err = dspGetPartialResultBufferAddress ( InterruptProcessID, whichPort, (Ptr *)&(fDataHeader->dh_haddr.physical));

    if (err != noErr)
        debugstr("Couldn't get PRB Address");

    fPRBnum = whichPort;
}


//-----------------------------------------------------------------------
//  ~TDspPRB:
//      This is the destructor for the TDspPRB class.
//-----------------------------------------------------------------------
TDspPRB::~TDspPRB()
{
    // Fill in any desired clean up code...
}


/*******************  D S P  P A R A M  B L O C K  C L A S S  ********************/
//-----------------------------------------------------------------------
//  TDspParamBlock Class:
//      This is a class which allows you to set up a block of data
//      to share between the DSP and host code without all the
//      overhead associated with buffer management (no header). Normally
//      this is used to share a few parameters between the host and DSP.
//-----------------------------------------------------------------------

#pragma segment ParamBlock
//-----------------------------------------------------------------------
//  TDspParamBlock:
//      This is the constructor for the TDspParamBlock class. Pass the
//      size of the parameter block. It will allocate the storage for
//      it.
//-----------------------------------------------------------------------
TDspParamBlock::TDspParamBlock(Size blokSize)
{
    if ( blokSize != 0 )
        fParamBlock = NewTrueAddress( blokSize );
    else
        fParamBlock = kNilAddress;

    fSize = blokSize;
}


//-----------------------------------------------------------------------
//  ~TDspParamBlock:
//      This is the destructor for the TDspParamBlock class. It will
//      dispose of the memory used by the parameter block.
//-----------------------------------------------------------------------
TDspParamBlock::~TDspParamBlock()
```

```
{
    if ( fParamBlock.logical != nil )
        dspDisposeTruePtr( fParamBlock.logical );
}


//-----------------------------------------------------------------------
//  Where:
//      This method returns a pointer to the parameter block. Note: this
//      will be the actual data since there is no header information.
//-----------------------------------------------------------------------
Address TDspParamBlock::Where()
{
    return fParamBlock;
}


//-----------------------------------------------------------------------
//  BlockSize:
//      This method returns the size of the the parameter block.
//-----------------------------------------------------------------------
Size TDspParamBlock::BlockSize()
{
    return fSize;
}



/**********************  D S P   C O D E   B L O C K   C L A S S  ************************/
//-----------------------------------------------------------------------
//  TDspCodeBlock Class:
//      This is a class which holds DSP code resources to be executed
//      by the DSP. There are methods to add Initialization functions,
//      Visible functions, data blocks and parameter blocks.
//-----------------------------------------------------------------------

#pragma segment DspCode
//-----------------------------------------------------------------------
//  TDspCodeBlock:
//      This is the constructor for the TDspCodeBlock class. Pass a
//      pointer to the task that you want to install the code block
//      into.
//-----------------------------------------------------------------------
TDspCodeBlock::TDspCodeBlock( TDspTask* myTask )
{
    short i;

    fMyTask = myTask;

    //--------------- Create the code header -------------
    dspNewTruePtr(CodeHeaderSize, (Ptr*)&fCodeHeader, false);
    if ( fCodeHeader == nil )
        debugstr("Couldn't allocate new CodeHeader");

    //--------------- Initialize the code header -------------
    fCodeHeader->cd_dspglobals.logical = (Ptr)dspGlobalsRead();
    dspLogical2Physical(fCodeHeader->cd_dspglobals.logical, &fCodeHeader->cd_dspglobals.physical);

    fCodeHeader->cd_firstVFuncHeader = kNilAddress;
    fCodeHeader->cd_ifunc = kNilAddress;
    fCodeHeader->cd_nextCodeHeader = kNilAddress;
    fCodeHeader->cd_numdbh = 0;

    for (i = 0; i < MAX_DBHS; i--)
        fCodeHeader->cd_dbh[i] = kNilAddress;

    pStrCpy(fCodeHeader->cd_name, kNoName);
}


//-----------------------------------------------------------------------
//  ~TDspCodeBlock:
//      This is the destructor for the TDspCodeBlock class. It unloads
//      the ifunc and all the vfunc's for the code block and then
//      deallocates the memory for the code block.
//-----------------------------------------------------------------------
TDspCodeBlock::~TDspCodeBlock()
{
```

```
    Address         tAddress;
    Ptr             tPtr;
    TaskHeaderPtr   theTask;


    //--------------- Kill the Ifunc --------------
    theTask = (TaskHeaderPtr)fMyTask->Where().logical;
    if (dspUnloadIFunction (  theTask, fCodeHeader) != noErr)
        debugstr("Problem unloading IFunc");

    //--------------- Kill all Vfuncs --------------
    while ( fCodeHeader->cd_firstVFuncHeader.logical != nil )
    {
        tAddress = ((VFunctionHeader*)fCodeHeader->cd_firstVFuncHeader.logical)->vf_nextVFuncHeader;

        if (dspUnloadVFunction ( theTask, (VFunctionHeaderHandle)&(fCodeHeader->cd_firstVFuncHeader.logical)) != noErr)
            debugstr("Trouble unloading vfunc");

        //------------ Reconnect the linked list ---------
        fCodeHeader->cd_firstVFuncHeader = tAddress;
    }

    //--------------- Dispose the parameter block --------------
    tPtr = fCodeHeader->cd_dbh[kDSPParamBlock].logical;
    if (tPtr != nil)
        dspDisposeTruePtr( tPtr );

    //--------------- Dispose the code header --------------
    dspDisposeTruePtr( (Ptr)fCodeHeader );
}


//-----------------------------------------------------------------------
//  AddIFunc:
//      This method adds an initialization function to a code block.
//      The Ifunc must be stored as a resource in the application.
//      Pass the name of the Ifunc resource to add.
//-----------------------------------------------------------------------
OSErr   TDspCodeBlock::AddIFunc( Str31 ifuncname )
{
    OSErr           tErr;
    TaskHeaderPtr   theTask;

    //--------------- Get ptr to the task --------------
    theTask = (TaskHeaderPtr)fMyTask->Where().logical;

    //--------------- Load the Ifunction --------------
    tErr = dspLoadIFunction ( theTask, fCodeHeader, ifuncname, false);
    if (tErr != noErr)
        debugstr("Bad dspLoadIFunction");

    return tErr;
}


//-----------------------------------------------------------------------
//  AddVFunc:
//      This method adds a "visible" function to a code block.
//      The Vfunc must be stored as a resource in the application.
//      Pass the name of the Vfunc resource to add.
//-----------------------------------------------------------------------
OSErr   TDspCodeBlock::AddVFunc( Str31 vfuncname )
{
    Address         tVFunc;
    Address         tAddress;
    OSErr           tErr;
    TaskHeaderPtr   theTask;

    //--------------- Get ptr to the task --------------
    theTask = (TaskHeaderPtr)fMyTask->Where().logical;

    //--------------- Load the Vfunction --------------
    tErr = dspLoadVFunction ( theTask, (VFunctionHeaderHandle)&tVFunc.logical, vfuncname, false);
    if (tErr != noErr)
        debugstr("Bad dspLoadVFunction");

    //--------------- Insert into list ----------------
    tAddress = fCodeHeader->cd_firstVFuncHeader;
```

```
    fCodeHeader->cd_firstVFuncHeader = tVFunc;
    ((VFunctionHeader*)tVFunc.logical)->vf_nextVFuncHeader = tAddress;

    return tErr;
}


//-----------------------------------------------------------------------
//  AddDataBlock:
//      This method either adds a data block (pass the number of the
//      data block and a pointer to it) or a parameter block (pass
//      a pointer to it) to be used by the task.  NOTE: The data block
//      must already be allocated.
//-----------------------------------------------------------------------
void    TDspCodeBlock::AddDataBlock( short whichDBH, TDspDataBlock* whichDB )
{
    fCodeHeader->cd_dbh[whichDBH] = whichDB->WhereHeader();
    fCodeHeader->cd_numdbh++;
}

void    TDspCodeBlock::AddDataBlock( TDspParamBlock* whichPB )
{
    fCodeHeader->cd_dbh[kDSPParamBlock] = whichPB->Where();
}


//-----------------------------------------------------------------------
//  ParamBlock:
//      This method returns a pointer to the parameter block for
//      the code block.
//-----------------------------------------------------------------------
Ptr TDspCodeBlock::ParamBlock()
{
    return fCodeHeader->cd_dbh[kDSPParamBlock].logical;
}


//-----------------------------------------------------------------------
//  Where:
//      This method returns a pointer to the code block.
//-----------------------------------------------------------------------
Address TDspCodeBlock::Where()
{
    Address tAddress;

    tAddress.logical = (Ptr)fCodeHeader;
    dspLogical2Physical(tAddress.logical, &tAddress.physical);

    return tAddress;
}


//-----------------------------------------------------------------------
//  NextCodeBlock:
//      This method either returns a pointer to the next code block
//      in the list of code blocks (pass no parameters) or sets the
//      the next code block (pass a pointer to the desired code block).
//-----------------------------------------------------------------------
TDspCodeBlock*  TDspCodeBlock::NextCodeBlock()
{
    return fNextCodeBlock;
}

void    TDspCodeBlock::NextCodeBlock(TDspCodeBlock* newCBSet)
{
    if (newCBSet != nil)
        fCodeHeader->cd_nextCodeHeader = newCBSet->Where();
    else
        fCodeHeader->cd_nextCodeHeader = kNilAddress;

    fNextCodeBlock = newCBSet;
}


//-----------------------------------------------------------------------
//  GetTrueDSPFResource:
//      This method loads a resource into a block of memory created
//      with a NewTrueAddress call and returns a Ptr to it.
```

```
//--------------------------------------------------------------------
Address TDspCodeBlock::GetTrueDSPFResource( Str31 resname )
{
    Handle      tHandle;
    long        i;
    Address     tAddress;
    Size        tResSize;
    Str31       tString;
    Ptr         tPtr;

    //--------------- Get handle to resource --------------
    pStrCpy(tString, resname);
    tHandle = GetNamedResource( 'DSPF', tString );

    //--------------- Get ptr to resource --------------
    HLock(tHandle);
    tPtr = StripAddress( *tHandle );

    if (tHandle != nil)
    {
        //--------------- Copy the resource --------------
        tResSize = SizeResource( tHandle );
        tAddress = NewTrueAddress( tResSize );
        if (tAddress.logical != nil)
            for (i = 0; i < tResSize; i++)
                (tAddress.logical)[i] = tPtr[i];
        ReleaseResource(tHandle);

        return tAddress;
    }

    return kNilAddress;
}

/*********************  D S P   S O U N D   C L A S S  ***************************/
//--------------------------------------------------------------------
//  TDspSound Class:
//      The TDspSound class uses the TDspCodeBlock, TDspPRB, and
//      TDspTask classes to create something that actually runs
//      on the DSP.
//--------------------------------------------------------------------

#pragma segment DspSound
//--------------------------------------------------------------------
//  TDspSound:
//      This is the constructor for the TDspSound class. Pass the name
//      of the DSP code resource to load.
//--------------------------------------------------------------------
TDspSound::TDspSound(Str31 whichAlg)
{
    TDspCodeBlock*  tCB;

    //----------- Create a new parameter block -------------
    fSoundParams = new TDspParamBlock( sizeof(Parameters));

    //--------- Create a new partial result buffer ---------
    fPRB = new TDspPRB(dspSystemMonoPort);
    fPRB->Name("\pSystem Mono PRB");

    //------------------ Create a new task -----------------
    fTask = new TDspTask();
    fTask->Name("\pSynth Sound task");

    //--------------- Create a new code block --------------
    tCB = new TDspCodeBlock(fTask);
    if (tCB->AddVFunc( whichAlg ) != noErr)
    {
        DebugStr ("\pCouldn't add code block.");
        return;
    }

    //--------- Load the initialization function -----------
    if (tCB->AddIFunc("\pIfunc") != noErr)
    {
        DebugStr ("\pCouldn't add Ifunc.");
        return;
    }
```

```
    //------- Add the parameter block and the PRB -----------
    tCB->AddDataBlock( fSoundParams );
    tCB->AddDataBlock( 0, fPRB );

    //---------------- Install the code block ---------------
    fTask->InstallCodeBlock( tCB );

    //--------------- Finally, install the task -------------
    if (gSoundApplication->fDsp->InstallTask(InterruptProcessID, fTask, dspSystemMonoPort, InsertHeadID) != noErr)
    {
        DebugStr ("\pCouldn't Install Task.");
        return;
    }
};


//-----------------------------------------------------------------------
//  ~TDspSound:
//      This is the destructor for the TDspSound class. It removes
//      the task from the DSP and frees the memory used by it.
//-----------------------------------------------------------------------
TDspSound::~TDspSound()
{
    if (gSoundApplication->fDsp->RemoveTask(InterruptProcessID, fTask) != noErr)
        debugstr("Couldn't Remove Task");

    delete fTask;
    if (fPRB != nil)
        delete fPRB;
    if (fSoundParams != nil)
        delete fSoundParams;
};



/********************** V O I C E   M A N A G E R   C L A S S ***************************/
//-----------------------------------------------------------------------
//  TVoiceManager Class:
//      The TVoiceManager class is used by the application to install
//      a number (kMaxVoices) of voices onto the DSP. Once these voices
//      are installed, the application can play a voice by calling the
//      PlayVoice method.
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
//  TVoiceManager:
//      This is the constructor for the TVoiceManager class. It just
//      initializes all the voices to be nil.
//-----------------------------------------------------------------------
TVoiceManager::TVoiceManager()
{
    int i;

    for (i=0; i<kMaxVoices; i++)
        fVoices[i] = nil;
};


//-----------------------------------------------------------------------
//  ~TVoiceManager:
//      This is the destructor for the TVoiceManager class. It deletes
//      any allocated (playing) voices.
//-----------------------------------------------------------------------
TVoiceManager::~TVoiceManager()
{
    int i;

    for (i=0; i<kMaxVoices; i++)
    {
        if (fVoices[i] != nil)      // This voice is playing
        {
            delete fVoices[i];      // Delete it (automatically stops it)
            fVoices[i] = nil;
        }
    }
};
```

```
//--------------------------------------------------------------------
//  InstallVoices:
//      This method allows the application to install a number (kMaxVoices)
//      of voices of the desired algorithm onto the DSP. Pass the name
//      of the DSP code resource that you wish to load.
//--------------------------------------------------------------------
OSErr   TVoiceManager::InstallVoices(Str31 whichAlg)
{
    int i;

    for (i=0; i<kMaxVoices; i++)
    {
        if (fVoices[i] != nil)          // This voice is playing
        {
            delete fVoices[i];          // Remove the old voice (even if playing!)
        }
        fVoices[i] = new TDspSound (whichAlg);      // Install new voice
        fVoices[i]->fPlaying = false;
    }
    return noErr;
};


#pragma segment Main
//--------------------------------------------------------------------
//  PlayVoice:
//      This method forces the DSP to start playing a voice of the
//      currently installed algorithm. Pass the parameters that should
//      be used to play the voice.
//--------------------------------------------------------------------
void    TVoiceManager::PlayVoice(Parameters theParams)
{
    int         i,j;
    Boolean     done;
    Parameters* tempPtr;

    i=0;
    done=false;
    while (!done)
    {
        if (!fVoices[i]->fPlaying)
        {
            // -------  Initialize the paramblock  ------
            tempPtr = (Parameters*)(fVoices[i]->fSoundParams->Where().logical);
            for (j=0; j<64; j++)
                tempPtr->param[j] = theParams.param[j];

            tempPtr->flags = theParams.flags;

            // -------  Start the task on the DSP  ------
            gSoundApplication->fDsp->ResumeTask(fVoices[i]->fTask, kNow);

            done=true;
            fVoices[i]->fPlaying = true;
        }
        i=i+1;
        if (i >= kMaxVoices-1)  done=true;      // No available voices
    }
};



//--------------------------------------------------------------------
//  CheckVoices:
//      This method is called at idle time by the application to see
//      if there are any "straggling" voices that are finished
//      executing on the DSP. It will suspend the task if it is
//      finished and update its fPlaying flag.
//--------------------------------------------------------------------
void    TVoiceManager::CheckVoices()
{
    int i;

    for (i=0; i<kMaxVoices; i++)
    {
        if (fVoices[i]->fPlaying)       // This voice is playing
```

```
        {
            // -------  Check to see if there are more samples to output  ------
            if (((Parameters*)(fVoices[i]->fSoundParams->Where().logical))->param(kNumSamplesIndex) <= 0.0)
            {
                gSoundApplication->fDsp->SuspendTask(fVoices[i]->fTask, kNow);      // If not, suspend the task…
                fVoices[i]->fPlaying = false;                                       // and update the fPlaying flag
            }
        }
    }
};



//--------------------------------------------------------------------
//  GetSamples:
//      This method is called by the application to return a block
//      of samples (240). These are used to draw a visual representation
//      of the sound. While the host is drawing the samples, the DSP
//      is busy calculating the next block.
//--------------------------------------------------------------------
void    TVoiceManager::GetSamples(SampleBlock* theSamples)
{
    int         i;
    Parameters* tempPtr;

    tempPtr = (Parameters*)(fVoices[0]->fSoundParams->Where().logical);

    //--------------- Copy the 240 samples  ----------
    for (i=0; i<kBlockSize; i++)
        theSamples->sample[i] = tempPtr->param[i+kReturnValues];

    //---------- Turn on the host ready flag ---------
    tempPtr->flags |= kHostReady;
};


/********************** A N A L Y S I S   C L A S S ***************************/
//--------------------------------------------------------------------
//  TAnalysisTask Class:
//      The TAnalysisTask class is used by the application to perform
//      an FFT analysis on a generated sound.
//--------------------------------------------------------------------

//--------------------------------------------------------------------
//  TAnalysisTask:
//      This is the constructor for the TAnalysisTask class. It sets
//      up all the DSP code to perform the FFT analysis. Pass the name
//      of the code resource which generates the sound data to analyze.
//      It will create a DSP task to do the data generation and a DSP
//      task to do the FFT analysis.
//--------------------------------------------------------------------
TAnalysisTask::TAnalysisTask(Str31 whichAlg)
{
    TDspCodeBlock*  tAnalysisCB;
    TDspCodeBlock*  tGenerateCB;

    //---------- Create the parameter block ---------
    fSoundData = new TDspParamBlock( sizeof(Parameters));

    //---------- Create the data generating task ----------
    fGenerateTask = new TDspTask();
    fGenerateTask->Name("\pSynth Generate task");

    //---------- Create the FFT analysis task ----------
    fAnalyzeTask = new TDspTask();
    fAnalyzeTask->Name("\pSynth Analysis task");

    //---------- Create code block for analysis task ----------
    tAnalysisCB = new TDspCodeBlock(fAnalyzeTask);

    //---------- Install the analysis Vfunc ---------
    if (tAnalysisCB->AddVFunc( "\pAnalysis" ) != noErr)
    {
        debugstr ("Couldn't add analysis code block.");
        return;
    }

    //---------- Install the analysis Ifunc ---------
```

```
    if (tAnalysisCB->AddIFunc("\pifunc") != noErr)
    {
        debugstr ("Couldn't add analysis Ifunc.");
        return;
    }

    //--------- Add the buffer data block ---------
    tAnalysisCB->AddDataBlock( fSoundData );

    //--------- Create the generating code block ---------
    tGenerateCB = new TDspCodeBlock(fGenerateTask);

    //--------- Install the generating Vfunc ---------
    if (tGenerateCB->AddVFunc( whichAlg ) != noErr)
    {
        debugstr ("Couldn't add generate code block.");
        return;
    }

    //--------- Install the generating Ifunc ---------
    if (tGenerateCB->AddIFunc("\pifunc") != noErr)
    {
        debugstr ("Couldn't add generate Ifunc.");
        return;
    }

    //--------- Add the parameter block for generate task ---------
    tGenerateCB->AddDataBlock( fSoundData );

    //--------- Install both code blocks ---------
    fAnalyzeTask->InstallCodeBlock( tAnalysisCB );
    fGenerateTask->InstallCodeBlock( tGenerateCB );

    //--------- Install the Analysis task ---------
    if (qSoundApplication->fDsp->InstallTask(InterruptProcessID, fAnalyzeTask, dspSystemMonoPort, InsertHeadID) != noErr)
    {
        DebugStr ("\pCouldn't Install Analysis Task.");
        return;
    }

    //--------- Install the Generate task ---------
    if (qSoundApplication->fDsp->InstallTask(InterruptProcessID, fGenerateTask, dspSystemMonoPort, InsertTailID) != noErr)
    {
        DebugStr ("\pCouldn't Install Generate Task.");
        return;
    }

    //--------- Turn off host ready flag ---------
    ((Parameters*)(fSoundData->Where().logical))->flags &= ~kHostReady;
};



//==========================================================================
//  ~TAnalysisTask:
//      This is the destructor for the TAnalysisTask class. It removes
//      the generating and analysis tasks from the DSP. It also frees
//      the memory used by the tasks.
//==========================================================================
TAnalysisTask::~TAnalysisTask()
{
    //--------- Remove the tasks ---------
    if (qSoundApplication->fDsp->RemoveTask(InterruptProcessID, fGenerateTask) != noErr)
        DebugStr("\pCouldn't Remove Task");
    if (qSoundApplication->fDsp->RemoveTask(InterruptProcessID, fAnalyzeTask) != noErr)
        DebugStr("\pCouldn't Remove Task");

    //--------- Free the tasks ---------
    delete fGenerateTask;
    delete fAnalyzeTask;

    //--------- Free the parameter block ---------
    if (fSoundData != nil)
        delete fSoundData;
};
```

```
//--------------------------------------------------------------------
//  Start:
//      This method activates both the generating and the analysis
//      tasks on the DSP.
//--------------------------------------------------------------------
void    TAnalysisTask::Start()
{
    //--------- Set the host ready flag ---------
    ((Parameters*)(fSoundData->Where().logical))->flags |= kHostReady;

    //--------- Start both tasks ---------
    gSoundApplication->fDsp->ResumeTask(fGenerateTask, kNow);
    gSoundApplication->fDsp->ResumeTask(fAnalyzeTask, kNow);

};


//--------------------------------------------------------------------
//  Stop:
//      This method deactivates both the generating and the analysis
//      tasks on the DSP.
//--------------------------------------------------------------------
void    TAnalysisTask::Stop()
{
    //--------- Clear the host ready flag ---------
    ((Parameters*)(fSoundData->Where().logical))->flags &= ~kHostReady;
};



//--------------------------------------------------------------------
//  SetHostReady:
//      This method sets the host ready flag so the DSP will execute
//      the generate and analysis tasks. The DSP will clear the flag
//      when it finsishes executing the tasks.
//--------------------------------------------------------------------
void    TAnalysisTask::SetHostReady()
{
    //--------- Set the host ready flag ---------
    ((Parameters*)(fSoundData->Where().logical))->flags |= kHostReady;
};


//--------------------------------------------------------------------
//  SetParams:
//      This method sets up the parameter block for the generate task.
//      these parameters will be used by the DSP algorithm when it
//      generates the sound data.
//--------------------------------------------------------------------
void    TAnalysisTask::SetParams(Parameters theParams)
{
    Parameters* tempPtr;
    int         count;

    //--------- Set the parameters ---------
    tempPtr = (Parameters*)(fSoundData->Where().logical);
    for (count=0; count<kNumParams; count++)
        tempPtr->param[count] = theParams.param[count];

    tempPtr->flags = theParams.flags;
};


//--------------------------------------------------------------------
//  SetIndex:
//      This method sets the index that is used by the analysis algorithm
//      as a pointer to the current analysis record.
//--------------------------------------------------------------------
void    TAnalysisTask::SetIndex(float newIndex)
{
    ((Parameters*)(fSoundData->Where().logical))->param[kIndex] = newIndex;
};


//--------------------------------------------------------------------
//  AdjustIndex:
//      This method adjusts the index that is used by the analysis
//      algorithm by the offset that is passed.
```

```
//--------------------------------------------------------------------------
void    TAnalysisTask::AdjustIndex(float offset)
{
    ((Parameters*)(fSoundData->Where().logical))->param(kIndex) += offset;
};


//--------------------------------------------------------------------------
//  AdjustIndex:
//      This method returns the current index that is used by the analysis
//      algorithm.
//--------------------------------------------------------------------------
float   TAnalysisTask::GetIndex()
{
    return ((Parameters*)(fSoundData->Where().logical))->param(kIndex);
};


//--------------------------------------------------------------------------
//  GetData:
//      This method returns the current FFT record from the analysis
//      task.
//--------------------------------------------------------------------------
void    TAnalysisTask::GetData(FFTData* samples)
{
    Parameters* tempPtr;
    int i;

    tempPtr = (Parameters*)(fSoundData->Where().logical);

    for (i=0; i<kFFTRecordSize; i++)
        samples->sample[i] = tempPtr->param[i+kReturnValues];
};
```

# TAnalysis.cp

```
//----------------------------------------------------------------------------
//
// NAME
//      TAnalysis.cp
//
// SYNOPSIS
//      This file implements the classes which support doing FFT frequency
//      analysis.
//
//      Author Jeff Boone, Oregon State University
//
//
//  HISTORY
//
//----------------------------------------------------------------------------
//#include  "TDsp.h"
#include    "USound.h"
#include    <Packages.h>


// ************  A N A L Y S I S  V I E W  C L A S S  **************
//----------------------------------------------------------------------------
//  AnalysisView class:
//      This class is used to implement the display of the FFT
//      records for the user.
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
//  Draw:
//      This method just draws a box around the display.
//----------------------------------------------------------------------------
pascal  void TAnalysisView::Draw(Rect *area)
{
    if (this->Focus())
    {
        this->GetQDExtent( area );              // get the rect of the view
        FrameRect( area );                      // Put a box around the whole 9 yards
    }
}


//----------------------------------------------------------------------------
//  SetViewFlag:
//      This method turns on the view flag. This flag determines whether
//      or not the FFT display screen gets updated.
//----------------------------------------------------------------------------
pascal  void TAnalysisView::SetViewFlag()
{
    fShowData = true;
}


//----------------------------------------------------------------------------
//  ClearViewFlag:
//      This method turns off the view flag. This flag determines whether
//      or not the FFT display screen gets updated.
//----------------------------------------------------------------------------
pascal  void TAnalysisView::ClearViewFlag()
{
    fShowData = false;
}


//----------------------------------------------------------------------------
//  DoIdle:
//      All FFT drawing gets done at idle time as our application runs.
//      This method checks the fShowData flag to see if it needs to update
//      the FFT display; if necessary it handles getting the FFT
//      record from the DSP and displaying the record on screen.
//----------------------------------------------------------------------------
pascal Boolean TAnalysisView::DoIdle(IdlePhase phase)
{
    const float     kStepFIndex =   -128.0;     // Constants for adjusting the index
    const float     kStepBIndex =   -384.0;
    const float     kFastFIndex =   256.0;
    const float     kRewIndex   =   -768.0;

    Rect            qdRect;
```

```
int             i;
FFTData         *samples;           // The FFT record from DSP
float           index, percent;
Str255          timeStr;

samples = (FFTData*)(NewPtr(sizeof(FFTData)));                  // Create room for the FFT record

if (this->Focus())
{
    if (fShowData)                                              // We need an update
    {
        switch (gSoundApplication->fAnalysis->fMode)            // Figure out which mode we're in (play, step, etc)
        {
            //-------------- Step Forward --------------
            case StepF:
            {
                gSoundApplication->fAnalysis->fAnalysisTask->AdjustIndex(kStepFIndex);
                break;
            }
            //-------------- Step Backward --------------
            case StepB:
            {
                gSoundApplication->fAnalysis->fAnalysisTask->AdjustIndex(kStepBIndex);
                break;
            }
            //-------------- Fast Forward --------------
            case FF:
            {
                gSoundApplication->fAnalysis->fAnalysisTask->AdjustIndex(kFastFIndex);
                break;
            }
            //-------------- Rewind --------------
            case Rew:
            {
                gSoundApplication->fAnalysis->fAnalysisTask->AdjustIndex(kRewIndex);
                break;
            }
        }

        //--------------  Update the time display --------------
        index = gSoundApplication->fAnalysis->fAnalysisTask->GetIndex();
        NumToString(long(index), timeStr);
        fTime->Focus();
        fTime->SetText(timeStr,true);

        //--------------  Update the percent display --------------
        percent = (index / 25000) * 100;                    // Index / num samples * 100
        sprintf( (char *)timeStr, " %3.1f%%", percent);
        timeStr[0] = 5;
        fPercent->Focus();
        fPercent->SetText(timeStr,true);
        this->Focus();

        //--------------  Get current FFT data --------------
        gSoundApplication->fAnalysis->fAnalysisTask->GetData(samples);      // Get current FFT data

        //--------------  Let DSP do another record --------------
        gSoundApplication->fAnalysis->fAnalysisTask->SetHostReady();        // Let DSP do another FFT

        //--------------  If we're stepping, stop DSP --------------
        if ((gSoundApplication->fAnalysis->fMode == StepF) || (gSoundApplication->fAnalysis->fMode == StepB))
        {
            gSoundApplication->fAnalysis->fAnalysisTask->Stop();
            fShowData = false;
        }

        //--------------  Erase the display --------------
        this->GetQDExtent(&qdRect);
        EraseRect(&qdRect);
        FrameRect(&qdRect);     // Put a box around the whole 9 yards

        //--------------  Draw the FFT data --------------
        for (i=0; i<128; i++)
        {
            MoveTo (i*2, (short)this->fSize.v);
            LineTo (i*2, (short)(this->fSize.v-( (short)(samples->sample[i]) /10 )));
        }
```

```
            if ((index >= 24960) || (index < 0))        // End of the sound
                gSoundApplication->fAnalysis->Restart();
        }
    }

    //--------------  Free the FFT data memory --------------
    DisposPtr(Ptr(samples));
    return (inherited::DoIdle(phase));
}


//======================================================================
//  Restart:
//      This method is called whenever the FFT analysis needs to be
//      started over.
//======================================================================
pascal void TAnalysisView::Restart( )
{
    gSoundApplication->fAnalysis->fAnalysisTask->Stop();                // Stop the task if it's running
    gSoundApplication->fAnalysis->fAnalysisTask->SetIndex( 0.0 );       // Start from beginning

    if (fTime->Focus());                                                // Update the sample index display
        fTime->SetText("\p0",true);

    if (fPercent->Focus());                                             // Update the percent display
        fPercent->SetText("\p0%",true);

    fShowData = false;                                                  // Don't update the display

}



// ***************  A N A L Y S I S   B U T T O N   C L A S S  ********************
//======================================================================
//  AnalysisButton class:
//      This class is used to implement the play, rewind, fast forward,
//      etc. buttons that are used to navigate through the FFT
//      analysis of a sound.
//======================================================================

#pragma segment ASelCommand

//----------------------------------------------------------------------
//  DoMouseCommand:
//      This method handles when the user clicks on any of the analysis
//      buttons.
//----------------------------------------------------------------------
pascal TCommand *TAnalysisButton::DoMouseCommand(Point *theMouse, EventInfo *info,
                                    Point *hysteresis)
{
    TAnalysisButton* tempButton;

    if (!this->fHilite)             // If this button isn't highlighted...
        this->Hilite();             // Hilight the button
    while (WaitMouseUp())  ;        // Wait for user to let up the mouse

    switch (this->fIdentifier)      // Figure out which button was pressed

        //------------------------------------------------------------
        //--  User clicked the play button                          --
        //------------------------------------------------------------
        case 'play':                // User clicked the play button
        {
            if (!this->fHilite)     // If we aren't already playing
            {
                //-------------- Start playing the FFT analysis ---------------
                gSoundApplication->fGenerator->GenerateAnalysis();              // Generate the sound data
                gSoundApplication->fAnalysis->fAnalysisTask->Start();           // Start the FFT analysis task
                gSoundApplication->fAnalysis->fAnalysisView->SetViewFlag();     // Tell the view to update
                gSoundApplication->fAnalysis->fMode = Play;                     // Put us in play mode
            }
            break;
        }

        //------------------------------------------------------------
        //--  User clicked the stop button                          --
```

```
//-----------------------------------------------------------------
case 'stop':
{
    //-------------- Stop playing the FFT analysis ---------------
    gSoundApplication->fAnalysis->fAnalysisTask->Stop();                    // Stop the FFT analysis task
    gSoundApplication->fAnalysis->fAnalysisView->ClearViewFlag();           // Tell the view to stop updating
    this->Hilite();                                                         // UnHilight stop button

    switch (gSoundApplication->fAnalysis->fMode)                            // Figure out which mode we were in
    {
        case Play:
        {   tempButton = (TAnalysisButton*)this->fSuperView->FindSubView('play');   // Find play button
            tempButton->Focus();                                                    // Set the focus rect
            tempButton->Hilite();                                                   // UnHilight play button
            break;
        }
        case Rew:
        {   tempButton = (TAnalysisButton*)this->fSuperView->FindSubView('rew ');   // Find rewind button
            tempButton->Focus();                                                    // Set the focus rect
            tempButton->Hilite();                                                   // UnHilight rewind button
            break;
        }
        case FF:
        {   tempButton = (TAnalysisButton*)this->fSuperView->FindSubView('ffwd');   // Find fast forward button
            tempButton->Focus();                                                    // Set the focus rect
            tempButton->Hilite();                                                   // UnHilight fast forward button
            break;
        }
    }
    gSoundApplication->fAnalysis->fMode = Stop;                             // Update the mode
    break;
}

//-----------------------------------------------------------------
//--  User clicked the rewind button                              --
//-----------------------------------------------------------------
case 'rew ':            // Rewind
{
    gSoundApplication->fGenerator->GenerateAnalysis();                      // Generate the sound samples
    gSoundApplication->fAnalysis->fAnalysisTask->Start();                   // Start the FFT analysis task
    gSoundApplication->fAnalysis->fAnalysisView->SetViewFlag();             // Tell the view to update
    gSoundApplication->fAnalysis->fMode = Rew;                              // Update the mode
    break;
}

//-----------------------------------------------------------------
//--  User clicked the step backward button                       --
//-----------------------------------------------------------------
case 'stbk':            // Step back
{
    gSoundApplication->fGenerator->GenerateAnalysis();                      // Generate the sound samples
    gSoundApplication->fAnalysis->fAnalysisTask->Start();                   // Start the FFT analysis task
    gSoundApplication->fAnalysis->fAnalysisView->SetViewFlag();             // Tell the view to update
    this->Hilite();                                                         // UnHilight step back button
    gSoundApplication->fAnalysis->fMode = StepB;                            // Update the mode
    break;
}

//-----------------------------------------------------------------
//--  User clicked the step forward button                        --
//-----------------------------------------------------------------
case 'stfw':            // Step forward
{
    gSoundApplication->fGenerator->GenerateAnalysis();                      // Generate the sound samples
    gSoundApplication->fAnalysis->fAnalysisTask->Start();                   // Start the FFT analysis task
    gSoundApplication->fAnalysis->fAnalysisView->SetViewFlag();             // Tell the view to update
    this->Hilite();                                                         // UnHilight step forward button
    gSoundApplication->fAnalysis->fMode = StepF;                            // Update the mode
    break;
}

//-----------------------------------------------------------------
//--  User clicked the fast forward button                        --
//-----------------------------------------------------------------
case 'ffwd':            // Fast forward
{
    gSoundApplication->fGenerator->GenerateAnalysis();                      // Generate the sound samples
    gSoundApplication->fAnalysis->fAnalysisTask->Start();                   // Start the FFT analysis task
```

```
            gSoundApplication->fAnalysis->fAnalysisView->SetViewFlag();          // Tell the view to update
            gSoundApplication->fAnalysis->fMode = FF;                            // Update the mode
            break;
        }
    }

    return (inherited::DoMouseCommand(theMouse, info, hysteresis));

}

// ****************** A N A L Y S I S   C L A S S  ********************
//=========================================================================
//  Analysis class:
//      This is the main class for doing FFT analysis of the sound
//      data. It handles maintaining the current synthesis method
//      for analysis and the analysis window.
//=========================================================================


//-------------------------------------------------------------------------
//  IAnalysis:
//      This method is used to initialize the analysis object and should
//      be called immediately after instantiating an analysis object.
//      It creates the window that will display the FFT analysis.
//-------------------------------------------------------------------------
pascal void TAnalysis::IAnalysis()
{
    //------------------ Initialize the FFT view -------------------
    fAnalysisView = (TAnalysisView*)(fWindow->FindSubView('view'));
    fAnalysisView->fIdleFreq = 0;                                        // Allow idle processing
    fAnalysisView->ClearViewFlag();
    fAnalysisView->fTime = (TStaticText*)fWindow->FindSubView('time');
    fAnalysisView->fPercent = (TStaticText*)fWindow->FindSubView('pcnt');

    //------------------ Initialize the buttons -------------------
    fPlayButton = (TAnalysisButton*)(fWindow->FindSubView('play'));
    fStopButton = (TAnalysisButton*)(fWindow->FindSubView('stop'));
    fRewButton = (TAnalysisButton*)(fWindow->FindSubView('rew '));
    fFFButton = (TAnalysisButton*)(fWindow->FindSubView('ffwd'));
    fStepFButton = (TAnalysisButton*)(fWindow->FindSubView('stfw'));
    fStepBButton = (TAnalysisButton*)(fWindow->FindSubView('stbk'));

    //--------------------- Set the task to nil -------------------
    fAnalysisTask = nil;
}


//-------------------------------------------------------------------------
//  ~TAnalysis:
//      This is the destructor for the Analysis class. It will delete
//      the DSP task if it exists.
//-------------------------------------------------------------------------
// TAnalysis::~TAnalysis()
pascal void TAnalysis::Free()
{
    if (fAnalysisTask != nil)
    {
        delete fAnalysisTask;       // Trash the task
        fAnalysisTask = nil;
    }
}


//-------------------------------------------------------------------------
//  SetAlg:
//      This method is used specify which synthesis algorithm to
//      load when doing the analysis.
//-------------------------------------------------------------------------
pascal void TAnalysis::SetAlg(Str31 whichAlg)
{
    //----------- If we already have a task, delete it -------------
    if (fAnalysisTask != nil)
    {
        delete fAnalysisTask;
        fAnalysisTask = nil;
    }

    //----------- Create a new task with the specified algorithm -------------
```

```
    fAnalysisTask = new TAnalysisTask(whichAlg);
    fAnalysisTask->SetIndex(0.0);
}


//----------------------------------------------------------------------
//  OpenWindow:
//      This method opens the analysis window. It is called whenever
//      the user chooses "Analysis" from the window menu.
//----------------------------------------------------------------------
pascal void TAnalysis::OpenWindow()
{
    if (!(this->fWindow->IsShown()))
        this->fWindow->Open();          // Open the window if it isn't already
    this->fWindow->Select();            // Bring the window to the front
}


//----------------------------------------------------------------------
//  Restart:
//      This method is called by the application when there is a need
//      to update the analysis window.
//----------------------------------------------------------------------
pascal void TAnalysis::Restart()
{
    switch (this->fMode)            // Figure out which mode we're in (play, step, etc)
    {
        //-------------- Play --------------
        case Play:
        {
            if (this->fPlayButton->Focus())
                this->fPlayButton->Hilite();
            break;
        }
        //-------------- Fast Forward --------------
        case FF:
        {
            if (this->fFFButton->Focus())
                this->fFFButton->Hilite();
            break;
        }
        //-------------- Rewind --------------
        case Rew:
        {
            if (this->fRewButton->Focus())
                this->fRewButton->Hilite();
            break;
        }
    }
    this->fMode = Stop;
    fAnalysisView->Restart();

}
```

# Sound.r

```
//--------------------------------------------------------------------------------
//
// NAME
//      Sound.r
//
// SYNOPSIS
//      This is the resource description file for the Sound application.
//      It contains the menu, dialog, window, etc. resources for the
//      application.
//
// COPYRIGHT
//      Author Jeff Boone, Oregon State University
//
//  HISTORY
//
//--------------------------------------------------------------------------------

include "Views.rsrc";       // Include the ViewEdit views

//--------- Include the requirements for this source ----------
#ifndef __TYPES.R__
#include "Types.r"
#endif

#ifndef __MacAppTypes__
#include "MacAppTypes.r"
#endif

#ifndef __ViewTypes__
#include "ViewTypes.r"
#endif

//---------------------- Menu Numbers ---------------
#define mPopup          235         // FM Configuration pop-up menu
#define mWindows        4
#define mSynthesis      5

//------------- Menu Item Command Numbers ------------
#define cSoundEditor    1001
#define cAnalysis       1002
#define cSoundView      1003

#define cFMSynthesis    1004
#define cKSSynthesis    1005
#define cAddSynthesis   1006

#define cMySaveAs       1007

// #define cQuit        36      // Defined in MacAppTypes.r


// Get the application's CODE.  Note that this is done this way so that the application can
// be linked and/or rezzed separately.  Also, Rez currently (MPW 3.0) does not support
// -align longword simultaneously with the -append option (but it only tells you if you're
// getting -p progress indication).
include $$Shell("ObjApp")$$Shell("XAppName") 'CODE';

//--------------------------------------------------------------
// Basic resources for various purposes.
// Self evident from their names.
//--------------------------------------------------------------
include "MacApp.rsrc";
include "Dialog.rsrc";
include "Printing.rsrc";

#if qDebug
include "Debug.rsrc";
#endif


//--------------------------------------------------------------
// Now include the "default" resources
// Note: you can pick and choose to selective,y replace defaults
//       in your own Rez source.
//--------------------------------------------------------------
include "Defaults.rsrc"  'ALRT' (phAboutApp);
include "Defaults.rsrc"  'cmnu' (mBuzzwords);
include "Defaults.rsrc"  'cmnu' (mApple);
```

```
include "Defaults.rsrc"  'cmnu' (mEdit);


//-------------------------------------------------------------
//----  File menu resource
//-------------------------------------------------------------
resource 'cmnu' (mFile) {
    mFile,
    textMenuProc,
    0x7FFFFBBB,
    enabled,
    "File",
     {
/* [1] */   "Open…",            noIcon, "O",    noMark, plain, cOpen;
/* [2] */   "Save…",            noIcon, "S",    noMark, plain, cMySaveAs;
 /*  */     "-",                noIcon, noKey,  noMark, plain, nocommand;
/* [3] */   "Quit",     noIcon, "Q",    noMark, plain, cQuit
     }
};

//-------------------------------------------------------------
//----  Window menu resource
//-------------------------------------------------------------
resource 'cmnu' (mWindows) {
    mWindows,
    textMenuProc,
    0x7FFFFFBD,
    enabled,
    "Windows",
     {
/* [1] */   "Sound Editor", noIcon, noKey, noMark, plain, cSoundEditor;
/* [2] */   "Analysis",     noIcon, noKey, noMark, plain, cAnalysis;
/* [3] */   "Sound View",   noIcon, noKey, noMark, plain, cSoundView
     }
};

//-------------------------------------------------------------
//----  Synthesis menu resource
//-------------------------------------------------------------
resource 'cmnu' (mSynthesis) {
    mSynthesis,
    textMenuProc,
    0x7FFFFFBD,
    enabled,
    "Synthesis",
     {
/* [1] */   "FM",               noIcon, noKey, noMark, plain, cFMSynthesis;
/* [2] */   "Karplus/Strong",   noIcon, noKey, noMark, plain, cKSSynthesis;
/* [3] */   "Additive",         noIcon, noKey, noMark, plain, cAddSynthesis
     }
};


//-------------------------------------------------------------
//---- Menu bar resource for the appliation
//-------------------------------------------------------------
resource 'MBAR' (kMBarDisplayed) { {mApple; mFile; mEdit; mWindows; mSynthesis;} };

include "Defaults.rsrc"  'DITL' (phAboutApp);
include "Defaults.rsrc"  'STR#' (kDefaultCredits);
include "Defaults.rsrc"  'WIND' (kDefaultWindowID);

// Get the default MacApp® application icon and necessary bundling rsrcs
include "Defaults.rsrc"  'MApp' (0);

// Get the default Version resources
include "Defaults.rsrc"  'vers' (1);        // Application or file specific
include "Defaults.rsrc"  'vers' (2);        // Overall package


//-------------------------------------------------------------
//----  Pop up menu resource for the FM configuration
//-------------------------------------------------------------
resource 'cmnu' (mPopup) {
    mPopup,
    textMenuProc,
    allEnabled,
    enabled,
```

```
    "Configuration:",
    {
/* [1] */   "1",    noIcon, noKey, noMark, plain, nocommand;
/* [2] */   "2",    noIcon, noKey, noMark, plain, nocommand;
/* [3] */   "3",    noIcon, noKey, noMark, plain, nocommand;
/* [4] */   "4",    noIcon, noKey, noMark, plain, nocommand;
/* [5] */   "5",    noIcon, noKey, noMark, plain, nocommand;
/* [6] */   "6",    noIcon, noKey, noMark, plain, nocommand;
/* [7] */   "7",    noIcon, noKey, noMark, plain, nocommand;
    }
};

//----------------------------------------------------------
//----  Menu color table for the pop up menu
//----------------------------------------------------------
resource 'mctb' (mPopup) {{
        mPopup, 0,
        {
        0x9999, 0x0000, 0x9999;         /* title color */
        0xFFFF, 0x6666, 0x0000;         /* "menu bar" color */
        0x0000, 0x6666, 0xFFFF;         /* default item color */
        0xFFFF, 0xFFFF, 0x0000          /* default menu background color */
        };

}};

resource 'MBAR' (kMBarNotDisplayed) {{mPopup}};


//----------------------------------------------------------
//----  "About" dialog item list resource
//----------------------------------------------------------
resource 'DITL' (1006, preload) {
    {   /* array DITLarray: 1 element */
        /* [1] */
        {0, 0, 336, 320},
        Picture {
            enabled,
            1001
        },
    }
};


//----------------------------------------------------------
//---- "About" dialog resource
//----------------------------------------------------------
resource 'DLOG' (1006, preload) {
    {50, 50, 386, 370},
    altDBoxProc,
    visible,
    noGoAway,
    0x0,
    1006,
    "Splash"
};


//----------------------------------------------------------
//----  Application size resource
//----------------------------------------------------------
resource 'SIZE' (-1) {
    saveScreen,
    acceptSuspendResumeEvents,
    enableOptionSwitch,
    canBackground,
    MultiFinderAware,
    backgroundAndForeground,
    dontGetFrontClicks,
    ignoreChildDiedEvents,
    is32BitCompatible,
    reserved, reserved, reserved, reserved, reserved, reserved, reserved,
    1024 * 1024,
    1024 * 1024
};

// ------------------- Bundle Stuff ---------------------------------
```

```
type 'JDB1' as 'STR ';

resource 'JDB1' (0) {                // Creator (or Signature)
    "Sound V0.9d1";
};


// ------------------------------------------------------------------------
resource 'FREF' (128) {
    'APPL', 0, ""       // Show files of type 'APPL' with ICN# with local ID 0
};

resource 'FREF' (129) {
    'JDB2', 1, ""       // Show files of type 'Bug1' with ICN# with local ID 1
};

// ------------------------------------------------------------------------
resource 'BNDL' (128) {
    'JDB1',     // Application signature (creator)
    0,          // Resource ID of version data
    {
        'ICN#', // This is how to map local ICN# ids into resource IDs
        {
            0, 128; // Local ICN# 0 is in resource 128
            1, 129  // Local ICN# 1 is in resource 129
        };
        'FREF', // This is how to map local FREF ids into resource IDs
        {
            0, 128; // Local FREF 0 (arbitrary) is in resource 128
            1, 129  // Local FREF 1 (also arbitrary) is in resource 129
        }
    }
};
```

# Analysis.s

```
/*-------------------------------------------------------------------------*/
/*
/* NAME
/*      Analysis.s
/*
/* SYNOPSIS
/*      This module performs an FFT analysis on the data that is pointed
/*      to by r17e + RETURN_VALUES. Normally these values are placed there
/*      by one of the generating routines. When this routine finishes, it
/*      turns off the HOST_READY flag, the host reads all 256 FFT values,
/*      turns the flag back on again, and finally the sound generation DSP
/*      module generates 256 more samples.
/*
/*      Author Jeff Boone, Oregon State University
/*          The FFT code is authored by AT&T
/*
/*-------------------------------------------------------------------------*/

#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "Analysis_Offsets.h"
#include "mac32c.a.h"

.rsect ".prog"
        r10e = r17                      /* Data Buffer address          */
        nop

        r7e = r17 + FLAGS
        r2e = *r7                       /* r2 contains flags from host  */
        nop                             /* latent instruction           */
        r2e & DO_FFT_FLAG               /* test the do fft bit          */
        if (eq) pcgoto End_It           /* host is not ready            */

        r8e = BUFFER
        r9e = r10 + RETURN_VALUES

        r7e = 256-2                     /* Copy over the data samples   */
loop:   *r8++ = a1 = dsp(*r9++)
        if (r7-- >= 0) pcgoto loop
        nop

        r11e = BUFFER

_fftc:
        *r14++ = r10e                   /* Push r10 */
        *r14++ = r12e                   /* Push r12 */
        *r14++ = r13e                   /* Push r13 */
        *r14++ = r18e                   /* Push r18 */
        *r14++ = r17e                   /* Push r17 */
        *r14 = r11e                     /* Push r11 */

/*-------------------------------------------------------------------------*/
/*--------------------  APPLY HAMMING WINDOW     --------------------*/
/*-------------------------------------------------------------------------*/

_hamm: r1e = 256             /* length of array, N  */
       r4e = BUFFER          /* pointer to top of array data */

       r2 = r1 * 2
       r2 = r2 * 2           /* 4*N */
       r2 = r2 - 4           /* 4*(N-1) */
       r2e = r2 + r4         /* pointer to end of array *

       r3e = _hammC      /* pointer to constant 0.5 */
                         /* initialization of a2 and a3 for n=0 */
       a2 = -*r3         /* a2 = -0.46*cos(2*pi*n/(N-1))+0.46 */
       a3 = a2 + *r3++   /* a3 = -0.46*sin(2*pi*n/(N-1))+ 0.0 */
       a1 = a2 + *r3     /* first window value = 0.08 */

       r1 = r1/2
       r1 = r1 - 2           /* used as loop counter */

_hammA: *r4++ = a0 = a1 * *r4   /* store (data * window) value */
        *r2-- = a0 = a1 * *r2   /* store (data * window) value */

       a0 = a2 * *r14++        /* a2 * cos(2*pi/(N-1)) */
       a2 = a3 - a3 * *r14     /* updated a2 for (n+1) */
       a0 = a2 * *r14--        /* a2 * sin(2*pi/(N-1)) */
```

```
      a3 = a0 + a3 * *r14     /* updated a3 for (n+1) */
      a1 = a2 + *r3           /* updated window value */

      if(r1-- >= 0) pcgoto _hammA
      nop

/*-------------------------------------------------------------------------*/
/*--------------------        FFT SECTION          --------------------*/
/*-------------------------------------------------------------------------*/

_fftb: r10e = 256             /* N */
      r7e = 8               /* M */
      r2e = BUFFER              /* points to _fftR */
      r17e = _fftI              /* points to _fftI */
      r6e = BUFFER              /* points to _fftR */
      r9e = r10 * 2             /* N * 2 */

/* In place bit reversal ---------------------------- */
      r17e = r17 - r2
      r16e = -r17
      r1e = r2                  /* points to _fftR */

      r5 = r10 - 3                  /* NM3 = N -3 */

_fftbA:r1 - r2
      if (ge) goto _fftb5
      r4e = r1                      /* temp. pointer for i */
                                    /* Begin complex exchange */
      *r1++r17 = a0 = *r2++r17      /* _fftR(i) = _fftR(j) */
      *r1++r16 = a0 = *r2++r16      /* _fftI(i) = _fftI(j) */
      *r2++r17 = a0 = *r4++r17      /* _fftR(j) = _fftR(i) */
      *r2++r16 = a0 = *r4++r16      /* _fftI(j) = _fftI(i) */

_fftb5:r3e = r9                      /* N2 = 2 * N */
      r2e = r2-r6
_fftb6:r3 - r2
      if (gt) goto _fftb7
      nop

      r2e = r2 - r3
      goto _fftb6
      r3e = r3/2

_fftb7:r2e = r2 + r6
      r2e = r2 + r3
      if (r5-- >= 0) goto _fftbA
      r1e = r1 + 4

/* Begin FFT calculation ---------------------------------- */

      r9 = 2
      r7 = r7 - 2             /* M = 2 */
      r8 = 1
      r4e = _fftbW
      r6e = _fftbtwo          /* points to 2.0 */
      r13e = _fftbtwo + 4     /* points to 1.0 */

_fftb30:r10e= r10/2
      r9e = r9*2
      r15e = r9*2
      r15e = r15-r17
      r2e = BUFFER                  /* _fftR */
                                    /* initialize twiddle factors */
      a2 = *r13++                   /* Ur = a2 = 1.0 */
      a3 = *r13--                   /* Ui = a3 = 0.0 */
      r18 = r9 - 2
      r8 = r8*2

_fftb20:r1e= r2
      r11e = r1

      r3e = r1
      r3e = r3 + r9
      r12e = r3
      r5e = r10 - 2

/*Butterfly ---------------------------------------- */
```

```
_fftb10:
  a0 = *r1++r17 + a2 * *r3++r17     /* a0=Re(i)+Ur*Re(k) */
 *r11++r17=a0=a0 - *r3++r16 * a3    /* Re(i)=a0=a0-Ui*Im(k) */
  a1 = *r1++r16 + a3 * *r3++r17     /* a1=Im(i)+Ui*Re(k) */
 *r12++r17=a0=-a0+ *r1++r17 * *r6   /* Re(k)=-a0+2*Re(i) */
 *r11++r15=a1=a1+ *r3++r15 * a2     /* Im(i)=a1=a1+Ur*Im(k) */
  if (r5-- >= 0) goto _fftb10
 *r12++r15=a1=-a1+ *r1++r15 * *r6   /* Im(k)=-a1+2*Im(i) */

      a0 = a2 * *r4++           /* Compute new twiddle */
      a2 = a0 - a3 * *r4        /* U = U * _fftbW */
      a0 = a2 * *r4--
      a3 = a0 + a3 * *r4
      if (r18 -- >= 0) pcgoto _fftb20
      r2e = r2 + 4

      if (r7-- >= 0) pcgoto _fftb30
      r4e = r4 + 8

/* ------------------------------------------------- */


        r11e = *r14--                   /* Pop r11 (BUFFER address)         */
        r17e = *r14--                   /* Pop r17 (Host Data address)      */
        r18e = *r14--                   /* Pop r18 (Return Address)         */
        r13e = *r14--                   /* Pop r13 (Needed by BOM)          */
        r12e = *r14--                   /* Pop r12 (Needed by BOM)          */
        r10e = *r14                     /* Pop r10 (Needed by BOM)          */
        nop

        r3e = BUFFER
        r4e = _fftI

        r5e = (128) - 2
here:   a0 = *r3 * *r3                  /* Put (a^2 + b^2) into the table   */
        *r3++ = a0 = a0 + *r4 * *r4++
        if (r5-->=0 )   pcgoto here
        nop

        r2e = BUFFER
        r8e = r17 + RETURN_VALUES

        r5e = (128) - 2
here2:  *r8++ = a0 = ieee(*r2++)        /* Move the data to host memory     */
        2*nop
        if (r5-->=0 )   pcgoto here2
        nop

        r8e = r17 + FLAGS               /* r8 points to host flags          */
        r2e = *r8                       /* r2 contains flags from host      */
        nop                             /* latent instruction               */
        r2e &= ~DO_FFT_FLAG             /* Clear the do_fft flag            */
        *r8 = r2e                       /* Write flags to memory            */

End_It: return(r18)
        nop

_fftI:          256*float 0.0

BUFFER:         256 * float
TEMP:           float
SCRATCH:        2 * float

.rsect ".tab"

/*Constant table ---------------------------------- */
_sqrtC: float 0.4237288136, 0.89
_sqrtD: float 1.4074074347, 0.81, 2.27424702, -.263374728
        float 0.5, 1.414213562

_hammC: float 0.46, 0.54

_fftbtwo:float 2.0, 1.0, 0.0

/*Twiddle table ---------------------------------- */
.rsect ".var"

_fftbW:
```

```
float -1.0000000,  0.0000000     /* cos(pi/1),-sin(pi/1)   */
float  0.0000000, -1.0000000     /* cos(pi/2),-sin(pi/2)   */
float  0.7071068, -0.7071068     /* cos(pi/4),-sin(pi/4)   */
float  0.9238795, -0.3826834     /* cos(pi/8),-sin(pi/8)   */
float  0.9807853, -0.1950903     /* cos(pi/16),-sin(pi/16) */
float  0.9951847, -0.0980171     /* cos(pi/32),-sin(pi/32) */
float  0.9987955, -4.9067674e-2 /* cos and -sin(pi/64)    */
float  0.9996988, -2.4541228e-2 /* cos and -sin(pi/128)   */
float  0.9999247, -1.2271538e-2 /* cos and -sin(pi/256)   */
float  0.9999812, -6.1358846e-3 /* cos and -sin(pi/512)   */

float  0.9999952938, -3.067956691e-3 /* cos and -sin(pi/1024) */
float  0.9999988234, -1.533980094e-3 /* cos and -sin(pi/2048) */
```

```
/*-------------------------------------------------------------------------------------------------------*/
/*      Analysis_Offsets.h                                                                               */
/*                                                                                                       */
/*                  Offsets for host/DSP shared parameters                                               */
/*                                                                                                       */
/*-------------------------------------------------------------------------------------------------------*/

#define RETURN_VALUES           260     /* param[65-320] (256 floats)   */
#define FLAGS                   1280    /* host/dsp flags               */

#define DO_FFT_FLAG             0x00000004
```

# An Interactive Workbench for Sound Experimentors

Vol. II

Jeff Boone

1991

# FMAlg1.s

```
/*-------------------------------------------------------------------------------*/
/*      FM_Offsets.h                                                             */
/*                                                                               */
/*                  Offsets for host/DSP shared parameters                       */
/*                                                                               */
/*      struct FMParams {                                                        */
/*          int     numSamples;                                                  */
/*          float   freq1, freq2, freq3, freq4;                                  */
/*          float   minIndex1, maxIndex1, minIndex2, maxIndex2, minIndex3, maxIndex3;  */
/*          float   env1[6], env2[6], env3[6], env4[6];                          */
/*          float   viewFlag, hostReady                                          */
/*  37 floats                                                                    */
/*-------------------------------------------------------------------------------*/

#define FM_SAMPLES              0
#define FM_FREQ1                4
#define FM_FREQ2                8
#define FM_FREQ3                12
#define FM_FREQ4                16
#define FM_MININDEX1            20
#define FM_MAXINDEX1            24
#define FM_MININDEX2            28
#define FM_MAXINDEX2            32
#define FM_MININDEX3            36
#define FM_MAXINDEX3            40
#define FM_ENV1                 44
#define FM_ENV2                 68
#define FM_ENV3                 92
#define FM_ENV4                 116


/*---------- These should be consistent with other synth methods --------------*/
#define FM_FLAGS                1280
#define FM_INDEX                236

#define FM_RETURN_DATA          260

#define VIEW_FLAG               0x00000001
#define HOST_READY_FLAG         0x00000002
#define DO_FFT_FLAG             0x00000004
#define SECOND_HALF             0x00000001

#define INIT_FLAG   0x00000001
```

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis algorithm 1.
/*
/*  This algorithm gets loaded onto the DSP when the user plays a voice
/*  of FM synthesis when the configuration is set to "1."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*              -------
/*              |     |
/*              |  1  |     Oscillator 1
/*              |     |
/*              -------
/*                 |
/*                 V
/*              -------
/*              |     |
/*              |  2  |     Oscillator 2
/*              |     |
/*              -------
/*                 |
/*                 V
/*              -------
/*              |     |
/*              |  3  |     Oscillator 3
/*              |     |
/*              -------
/*                 |
/*                 V
/*              -------
/*              |     |
/*              |  4  |     Oscillator 4
/*              |     |
/*              -------
/*                 |
/*                 O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "FM_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"


.rsect  ".prog"
            r10e = r16 + DH_HADDR + PHYSICAL    /* PRB or Sample buffer      */
            r10e = *r10
            r9e = r17                           /* host parameters           */

            r9e = PARAMS
            r7e = 35                            /* number of params - 2      */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1         /* copy over all params      */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2

loop2:      *r8++ = a0 = *r2 * *r9++            * set up INC1 - INC4         */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + FM_FLAGS
            r2e = *r7                           * r2 contains flags from host */
            nop                                 /* latent instruction        */
            r2e & VIEW_FLAG                     /* test the view bit         */
            if (eq) pcgoto Next                 * we want to hear it        */
            nop

            r10e = r17 + FM_RETURN_DATA         /* point r10 to param block  */
            r2e & HOST_READY_FLAG               /* test the host_ready bit   */
            if (eq) pcgoto EndIt                /* host is not ready         */
```

```
Next:       r8e = r17
            a0 = dsp(*r8)                   /* NumSamples                          */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int24(a0)            /* convert to an integer               */
            3*nop
            r2e = *r2                       /* r2 = NumSamples                     */
            nop

            r2e = r2 - 240                  /* Subtract the frame size             */
            r2 - 0                          /* See if we need more samples         */
            if (ge) pcgoto Next2            /* need more samples                   */

            r3e = INDEX                     /* don't need samples                  */
            r2e = ZERO                      /* zero out our VAR parameters         */
            r7e = 7                         /* there are 9 of them                 */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters    */
            *r3++ = a0 = *r2                /* Zero out the parameters             */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag           */

            pcgoto EndIt                    /* exit the module                     */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp       */
            nop
            a0 = float(*r4)                 /* convert to float                    */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining            */

            r11e = INDEX
            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                         */
            a2 = a1 - *r6                   /* Subtract 0.5 for round              */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                          */
            3*nop
            r2e = *r2                       /* r2 = Start                          */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 =     * 4 (for float size)       */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env(start)                          */
            r4e = r3 + 4                    /* Env(finish)                         */
            a0 = *r4 - *r3                  /* a0 = Env(finish) - Env(start)       */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1              */

            r3e = r2 + FM_ENV2              /* Env(start)                          */
            r4e = r3 + 4                    /* Env(finish)                         */
            a0 = *r4 - *r3                  /* a0 = Env(finish) - Env(start)       */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2              */

            r3e = r2 + FM_ENV3              /* Env(start)                          */
            r4e = r3 + 4                    /* Env(finish)                         */
            a0 = *r4 - *r3                  /* a0 = Env(finish) - Env(start)       */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3              */

            r3e = r2 + FM_ENV4              /* Env(start)                          */
            r4e = r3 + 4                    /* Env(finish)                         */
            a0 = *r4 - *r3                  /* a0 = Env(finish) - Env(start)       */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4              */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit
```

```
        r2e &= ~INIT_FLAG
        *r8 = r2e                       /* Turn off init flag           */

        r3e = PARAMS + FM_ENV1          /* Set up initial indexes       */
        r4e = INDEX1
        *r4 = a0 = *r3

        r3e = PARAMS + FM_ENV2
        r4e = INDEX2
        *r4 = a0 = *r3

        r3e = PARAMS + FM_ENV3
        r4e = INDEX3
        *r4 = a0 = *r3

        r3e = PARAMS + FM_ENV4
        r4e = INDEX4
        *r4 = a0 = *r3

NoInit: r7e = 238                       /* Loop count (frameSize-2)     */
        r9e = SINE_SIZE

Mainloop: r1e = PARAMS + FM_MAXINDEX1
        r5e = PARAMS + FM_MININDEX1
        a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1 */
        r6e = INDEX1_INC
        r4e = INDEX1
        a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN */
        *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC */

        r2e = PHASE1
        r4e = TEMP_INT
        *r4 = a2 = int24 (*r2)
        3*nop
        r4e = *r4                       /* r4 = PHASE1                  */
        r1e = MOD1
        r4 = r4*2                       /* Multiply by 4 for float      */
        r4 = r4*2

        r5e = r4 + SINE_TABLE           /* Need to do interpolation     */
        *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1  */

        r3e = INC1
        *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1       */
        3*nop

        r3e = TEMP_INT
        *r3 = a0 = int24(*r2)           /* r3 = PHASE1                  */
        3*nop
        r3e = *r3
        nop
        r3 - 256                        /* See if we're past end of table */
        if (lt) pcgoto less1
        nop
        *r2 = a0 = *r2 - *r9            /* Wrap to start of table       */

less1:  r5e = PARAMS + FM_MININDEX2
        r1e = PARAMS + FM_MAXINDEX2
        a3 = *r1 - *r5                  /* a3 = MAX_INDEX2 - MIN_INDEX2 */
        r6e = INDEX2_INC
        r4e = INDEX2
        a1 = *r5 + a3 * *r4             /* a1 = (INDEX2 * (MAX-MIN)) + MIN */
        *r4 = a0 = *r4 + *r6

        r1e = PHASE2
        r4e = TEMP_INT
        *r4 = a2 = int24 (*r2)
        3*nop
        r4e = *r4
        r1e = MOD2
        r4 = r4*2                       /* Multiply by 4 for float      */
        r4 = r4*2
        r5e = r4 + SINE_TABLE           /* Need to do interpolation     */
        *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE2] * INDEX2  */

        r3e = INC2
        a1 = *r2 + *r3                  /* Phase2 = Phase2 + Inc2       */
```

```
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + (Mod1*Inc1)    */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok2
            nop

less2:      r3 - 0
            if (gt) pcgoto ok2
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok2:        r5e = PARAMS + FM_MININDEX3
            r1e = PARAMS + FM_MAXINDEX3
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX3 - MIN_INDEX3     */
            r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX3 * (MAX-MIN)) + MIN  */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24  *r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3           */
            r4e = MOD2
            r5e = INC2
            *r2 = a0 = a1 + *r4 * *r5       /* Phase3 = Phase3 + (mod2 * inc2)  */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE3                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok3
            nop

less3:      r3 - 0
            if (gt) pcgoto ok3
            nop
            *r2 = a0 = *r2 - *r9            /* Handle negative case             */

ok3:        r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                        /* a1 = INDEX4                      */
            *r4 = a0 = *r4 + *r6            /* INDEX4 = INDEX4 + INDEX4_INC     */

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                       /* Multiply by 4 for float          */
```

```
            r5e = r1 + SINE_TABLE           /* r5 is index into sine table      */
            r6e = MOD3

            r8e = r17 + FM_FLAGS            /* r8 points to host flags          */
            r3e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r3e & VIEW_FLAG                 /* test the view bit                */
            if (eq) pcgoto Sound            /* we want to hear it               */
            nop

            a0 = a1 * *r5                   /* Just viewing                     */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer         */
            pcgoto Cont
            nop

Sound:      *r10++ = a0 = *r10 + a1 * *r5   /* Sum into PRB                     */

Cont:       r3e = INC4
            r5e = INC3
            a1 = *r2 + *r3                  /* PHASE4 + INC4                    */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD3*INC3)      */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD3 - OLD_SIG3  */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags          */
            r2e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag        */
            *r8 = r2e                       /* Write flags to memory            */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
MAX_AMP:        float   (1.0)
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
```

```
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAlg2.s

```
/*-----------------------------------------------------------------------------*/
/*  FM Synthesis algorithm 2.
/*
/*  This algorithm gets loaded onto the DSP when the user plays a voice
/*  of FM synthesis when the configuration is set to "2."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*              -------                     -------
/*              |     |                     |     |
/*              |  1  | Oscillator 1        |  2  | Oscillator 2
/*              |     |                     |     |
/*              -------                     -------
/*                 |--------------- + ----------|
/*                                  |
/*                                  V
/*                              -------
/*                              |     |
/*                              |  3  |     Oscillator 3
/*                              |     |
/*                              -------
/*                                 |
/*                                 V
/*                              -------
/*                              |     |
/*                              |  4  |     Oscillator 4
/*                              |     |
/*                              -------
/*                                 |
/*                                 O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------------*/
#include <dspregs.h>
#include "FM_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"

.rsect  ".prog"
            r10e = r16 + DH_HADDR + PHYSICAL    /* PRB or Sample buffer        */
            r10e = *r10
            r9e = r17                           /* Host parameters             */

            r8e = PARAMS
            r7e = 35                            /*  Number of params - 2       */

loop1:      *r8++ = a2 = dsp  *r9++)
            if (r7-- >= 0) pcgoto loop1         /*  copy over all params       */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2

loop2:      *r8++ = a0 = *r2 * *r9++            /* Set up INC1 - INC4          */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + FM_FLAGS
            r2e = *r7                           /* r2 contains flags from host */
            nop                                 /* latent instruction          */
            r2e & VIEW_FLAG                     /* test the view bit           */
            if (eq) pcgoto Next                 /* we want to hear it          */
            nop

            r10e = r17 + FM_RETURN_DATA         /* point r10 to param block    */
            r2e & HOST_READY_FLAG               /* test the host_ready bit     */
            if (eq) pcgoto Exit                 /* host is not ready           */

Next:       r8e = r17
            a0 = dsp(*r8)                       /* NumSamples                  */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int2( a0)
            3*nop
            r2e = *r2                           /* r2 = NumSamples             */
```

```
            nop

            r2e = r2 - 240              /* Subtract the frame size          */
            r2 - 0                      /* Need to output more samples?     */
            if (ge) pcgoto Next2

            r3e = INDEX                 /* don't need samples               */
            r2e = ZERO                  /* zero out our VAR parameters      */
            r7e = 7                     /* there are 9 of them              */
here:       if (r7-- >= 0) pcgoto here  /* Re-initailize the VAR parameters */
            *r3++ = a0 = *r2            /* Zero out the parameters          */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                   /* Turn on need to init flag        */

            pcgoto EndIt                /* exit the module                  */

Next2:      r4e = TEMP_INT
            *r4 = r2                    /* put samples remaining in temp    */
            nop
            a0 = float(*r4)             /* convert to float                 */
            2*nop
            *r8 = a1 = ieee(a0)         /* Update samples remaining         */

            r11e = INDEX
            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7             /* I * COMPARE                      */
            a2 = a1 - *r6               /* Subtract 0.5 for round           */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)          /* a2 = Start                       */
            3*nop
            r2e = *r2                   /* r2 = Start                       */
            nop
            r2 = r2*2
            r2 = r2*2                   /* r2 = r2 * 4 (for float size)     */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1          /* Env[start]                       */
            r4e = r3 + 4                /* Env[finish]                      */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 1           */       /* INDEX_INC1 = a0 * COMPARE

            r3e = r2 + FM_ENV2          /* Env[start]                       */
            r4e = r3 + 4                /* Env[finish]                      */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 2           */

            r3e = r2 + FM_ENV3          /* Env[start]                       */
            r4e = r3 + 4                /* Env[finish]                      */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 3           */

            r3e = r2 + FM_ENV4          /* Env[start]                       */
            r4e = r3 + 4                /* Env[finish]                      */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 4           */

            r8e = NEED_TO_INIT
            r1e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                   /* Turn off init flag               */

            r3e = PARAMS + FM_ENV1      /* Set up initial indexes           */
            r4e = INDEX1
            *r4 = a0 = *r3
```

```
            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 238                       /* Loop count (frameSize-2)          */
            r9e = SINE_SIZE

Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1      */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN   */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC      */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                       */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation          */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1       */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1            */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */

less1:      r5e = PARAMS + FM_MININDEX2
            r1e = PARAMS + FM_MAXINDEX2
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX2 - MIN_INDEX2      */
            r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX2 * (MAX-MIN)) + MIN   */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation          */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE2] * INDEX2       */

            r3e = INC2
            a1 = *r2 + *r3                  /* Phase2 = Phase2 + Inc2            */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + (Mod1*Inc1)     */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                       */
```

```
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less2:      r5e = PARAMS + FM_MININDEX3
            r1e = PARAMS + FM_MAXINDEX3
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX3 - MIN_INDEX3     */
            r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX3 * (MAX-MIN)) + MIN  */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3           */
            r4e = MOD2
            r5e = INC2
            r1e = MOD1
            r8e = INC1
            a0 = a1 + *r4 * *r5             /* Phase3 + Inc3 + (mod2 * inc2)    */
            2*nop
            *r2 = a0 = a0 + *r1 * *r8       /* Add MOD1 * INC1                  */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE3                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok3
            nop

less3:      r3 - 0
            if (gt) pcgoto ok3
            nop
            *r2 = a0 = *r2 - *r9            /* Handle negative case             */

ok3:        r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                        /* a1 = INDEX4                      */
            *r4 = a0 = *r4 - *r6

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                       /* Multiply by 4 for float          */
            r5e = r1 + SINE_TABLE           /* r5 is index into sine table      */
            r6e = MOD3

            r8e = r17 + FM_FLAGS            /* r8 points to host flags          */
            r2e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r1e & VIEW_FLAG                 /* test the view bit                */
            if (eq) pcgoto Sound            /* we want to hear it               */
            nop
```

```
            a0 = a1 * *r5                   /* Just viewing                 */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer     */
            pcgoto Cont
            nop

Sound:      *r10++ = a0 = *r10 + a1 * *r5   /* Sum into PRB                 */

Cont:       r3e = INC4
            r5e = INC3
            a1 = *r2 + *r3                  /* PHASE4 + INC4                */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD3*INC3)  */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD3 - OLD_SIG3 */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                  */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table       */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case         */

ok4:        r2e = ONE
            *r11 = a1 = *r11 - *r2          /* Increment the index          */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame       */
            nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags      */
            r2e = *r8                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag    */
            *r8 = r2e                       /* Write flags to memory        */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
MAX_AMP:        float   (1.0)
TEMP_FLOAT:     float
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
```

```
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492992, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAlg3.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis algorithm 3.
/*
/*  This algorithm gets loaded onto the DSP when the user plays a voice
/*  of FM synthesis when the configuration is set to "3."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*                                      -------
/*                                      |     |
/*                                      |  2  | Oscillator 2
/*                                      |     |
/*                                      -------
/*                                         |
/*                                         V
/*              -------                 -------
/*              |     |                 |     |
/*              |  1  | Oscillator 1    |  3  |     Oscillator 3
/*              |     |                 |     |
/*              -------                 -------
/*                  |------------ + ---------|
/*                                |
/*                               V
/*                            -------
/*                            |     |
/*                            |  4  |     Oscillator 4
/*                            |     |
/*                            -------
/*                               |
/*                               O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "FM_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"

.rsect  ".prog"
            r10e = r16 + DH_HADDR + PHYSICAL    /* PRB or Sample buffer      */
            r10e = *r10
            r9e = r17                           /* host parameters           */

            r8e = PARAMS
            r7e = 35                            /* number of params - 2      */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1         /* copy over all params      */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2

loop2:      *r8++ = a0 = *r2 * *r9++            /* Set up INC1 - INC4        */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + FM_FLAGS
            r2e = *r7                           /* r2 contains flags from host */
            nop                                 /* latent instruction        */
            r2e & VIEW_FLAG                     /* test the view bit         */
            if (eq) pcgoto Next                 /* we want to hear it        */
            nop

            r10e = r17 + FM_RETURN_DATA         /* point r10 to param block  */
            r2e & HOST_READY_FLAG               /* test the host_ready bit   */
            if (eq) pcgoto EndIt                /* host is not ready         */

Next:       r8e = r17
            a0 = dsp(*r8)                       /* NumSamples                */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int24(a0)                /* convert to an integer     */
            3*nop
            r2e = *r2                           /* r2 = NumSamples           */
```

```
        nop

        r2e = r2 - 240              /* Subtract the frame size          */
        r2 - 0                      /* See if we need more samples      */
        if (ge) pcgoto Next2        /* need more samples                */

        r3e = INDEX                 /* don't need samples               */
        r2e = ZERO                  /* zero out our VAR parameters      */
        r7e = 7                     /* there are 9 of them              */
here:   if (r7-- >= 0) pcgoto here  /* Re-initailize the VAR parameters */
        *r3++ = a0 = *r2            /* Zero out the parameters          */
        r4e = NEED_TO_INIT
        r2e = INIT_FLAG
        *r4 = r2e                   /* Turn on need to init flag        */

        pcgoto EndIt                /* exit the module                  */

Next2:  r4e = TEMP_INT
        *r4 = r2                    /* put samples remaining in temp    */
        nop
        a0 = float(*r4)             /* convert to float                 */
        2*nop
        *r8 = a1 = ieee(a0)         /* Update samples remaining         */

        r11e = INDEX
        r7e = COMPARE
        r6e = PT_FIVE
        a1 = *r11 * *r7             /* I * COMPARE                      */
        a2 = a1 - *r6               /* Subtract 0.5 for round           */
        r2e = TEMP_INT
        nop
        *r2 = a2 = int(a2)          /* a2 = Start                       */
        3*nop
        r2e = *r2                   /* r2 = Start                       */
        nop
        r2 = r2*2
        r2 = r2*2                   /* r2 = r2 * 4 (for float size)     */
        r2e = PARAMS + r2
        r3e = r2 + FM_ENV1          /* Env[start]                       */
        r4e = r3 + 4                /* Env[finish]                      */
        a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]    */
        r5e = INDEX1_INC
        nop
        *r5 = a1 = a0 * *r7         /* Save Index increment 1           */      /* INDEX_INC1 = a0 * COMPARE

        r3e = r2 + FM_ENV2          /* Env[start]                       */
        r4e = r3 + 4                /* Env[finish]                      */
        a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]    */
        r5e = INDEX2_INC
        nop
        *r5 = a1 = a0 * *r7         /* Save Index increment 2           */

        r3e = r2 + FM_ENV3          /* Env[start]                       */
        r4e = r3 + 4                /* Env[finish]                      */
        a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]    */
        r5e = INDEX3_INC
        nop
        *r5 = a1 = a0 * *r7         /* Save Index increment 3           */

        r3e = r2 + FM_ENV4          /* Env[start]                       */
        r4e = r3 + 4                /* Env[finish]                      */
        a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]    */
        r5e = INDEX4_INC
        nop
        *r5 = a1 = a0 * *r7         /* Save Index increment 4           */

        r8e = NEED_TO_INIT
        r2e = *r8
        nop
        r2e & INIT_FLAG
        if (eq) pcgoto NoInit

        r2e &= ~INIT_FLAG
        *r8 = r2e                   /* Turn off init flag               */

        r3e = PARAMS + FM_ENV1      /* Set up initial indexes           */
        r4e = INDEX1
        *r4 = a0 = *r3
```

```
            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 238                       /* Loop count (frameSize-2)          */
            r9e = SINE_SIZE

Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1      */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN   */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC      */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                       */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation          */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1       */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1            */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */

less1:      r5e = PARAMS + FM_MININDEX2
            r1e = PARAMS + FM_MAXINDEX2
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX2 - MIN_INDEX2      */
            r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX2 * (MAX-MIN)) + MIN   */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation          */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE2] * INDEX2       */

            r3e = INC2
            a1 = *r2 + *r3                  /* Phase2 = Phase2 + Inc2            */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + (Mod1*Inc1)     */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                       */
```

```
            3*nop
            r3e = *r3
            nop
            r3 - 256                    /* See if we're past end of table   */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9        /* Wrap to start of table           */
            pcgoto ok2
            nop

less2:      r3 - 0
            if (gt) pcgoto ok2
            nop
            *r2 = a0 = *r2 + *r9        /* Handle negative case             */

ok2:        r5e = PARAMS + FM_MININDEX3
            r1e = PARAMS + FM_MAXINDEX3
            a3 = *r1 - *r5              /* a3 = MAX_INDEX3 - MIN_INDEX3     */
            r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r5 + a3 * *r4         /* a1 = (INDEX3 * (MAX-MIN)) + MIN  */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                   /* Multiply by 4 for float          */
            r4 = r4*2
            r5e = r4 + SINE_TABLE       /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1         /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3              /* Phase3 = Phase3 + Inc3           */
            r4e = MOD2
            r5e = INC2
            *r2 = a0 = a1 + *r4 * *r5   /* Phase3 = Phase3 + (mod2 * inc2)  */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)       /* r3 = PHASE3                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                    /* See if we're past end of table   */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9        /* Wrap to start of table           */

less3:      r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                    /* a1 = INDEX4                      */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                   /* Multiply by 4 for float          */
            r5e = r1 + SINE_TABLE       /* r5 is index into sine table      */
            r6e = MOD3

            r8e = r17 + FM_FLAGS        /* r8 points to host flags          */
            r3e = *r8                   /* r2 contains flags from host      */
            nop                         /* latent instruction               */
            r3e & VIEW_FLAG             /* test the view bit                */
            if (eq) pcgoto Sound        /* we want to hear it               */
            nop

            a0 = a1 * *r5               /* Just viewing                     */
            2*nop
            *r10++ = a0 = ieee(a0)      /* Place into sample buffer         */
            pcgoto Cont
```

```
            nop

Sound:      *r10++ = a0 = *r10 + a1 * *r5   /* Sum into PRB                      */

Cont:       r3e = INC4
            r5e = INC3
            r1e = INC2
            r8e = MOD2
            a1 = *r2 + *r3                  /* PHASE4 + INC4                     */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD3*INC3)       */
            2*nop
            a2 = a2 + *r8 * *r1             /* Add Inc2 * Mod2                   */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD3 - OLD_SIG3   */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case              */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index               */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame            */
            nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags           */
            r2e = *r8                       /* r2 contains flags from host       */
            nop                             /* latent instruction                */
            r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag         */
            *r8 = r2e                       /* Write flags to memory             */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
MAX_AMP:        float   (1.0)
TEMP_FLOAT:     float
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
```

```
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAlg4.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis algorithm 4.
/*
/*  This algorithm gets loaded onto the DSP when the user plays a voice
/*  of FM synthesis when the configuration is set to "4."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*              -------                     -------
/*              |     |                     |     |
/*              |  1  | Oscillator 1        |  2  | Oscillator 2
/*              |     |                     |     |
/*              -------                     -------
/*                 |                           |
/*                 V                           V
/*              -------                     -------
/*              |     |                     |     |
/*              |  3  | Oscillator 3        |  4  |     Oscillator 4
/*              |     |                     |     |
/*              -------                     -------
/*                 |------------ + ---------|
/*                               |
/*                               O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "FM_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"

.rsect  ".prog"
            r10e = r16 + DH_HADDR + PHYSICAL    /* PRB or Sample buffer         */
            r10e = *r10
            r9e = r17                           /* host parameters              */

            r8e = PARAMS
            r7e = 35                            /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1         /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2

loop2:      *r8++ = a0 = *r2 * *r9++            /* Set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + FM_FLAGS
            r2e = *r7                           /* r2 contains flags from host  */
            nop                                 /* latent instruction           */
            r2e & VIEW_FLAG                     /* test the view bit            */
            if (eq) pcgoto Next                 /* we want to hear it           */
            nop

            r10e = r17 + FM_RETURN_DATA         /* point r10 to param block     */
            r2e & HOST_READY_FLAG               /* test the host_ready bit      */
            if (eq) pcgoto EndIt                /* host is not ready            */

Next:       r8e = r17
            a0 = dsp(*r8)                       /* NumSamples                   */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int24(a0)                /* convert to an integer        */
            3*nop
            r2e = *r2                           /* r2 = NumSamples              */
            nop

            r2e = r2 - 240                      /* Subtract the frame size      */
            r2 - 0                              /* See if we need more samples  */
            if (ge) pcgoto Next2                /* need more samples            */

            r3e = INDEX                         /* don't need samples           */
```

```
            r2e = ZERO                      /* zero out our VAR parameters      */
            r7e = 7                         /* there are 9 of them              */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters */
            *r3++ = a0 = *r2                /* Zero out the parameters          */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag        */

            pcgoto EndIt                    /* exit the module                  */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp    */
            nop
            a0 = float(*r4)                 /* convert to float                 */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining         */

            r11e = INDEX
            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                      */
            a2 = a1 - *r6                   /* Subtract 0.5 for round           */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                       */
            3*nop
            r2e = *r2                       /* r2 = Start                       */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)     */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1           */          /* INDEX_INC1 = a0 * COMPARE

            r3e = r2 + FM_ENV2              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2           */

            r3e = r2 + FM_ENV3              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3           */

            r3e = r2 + FM_ENV4              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4           */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag               */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes           */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
```

```
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 238                       /* Loop count (frameSize-2)          */
            r9e = SINE_SIZE

Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1      */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN   */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC      */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                       */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation          */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1) * INDEX1       */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1            */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */

less1:      r5e = PARAMS + FM_MININDEX2
            r1e = PARAMS + FM_MAXINDEX2
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX2 - MIN_INDEX2      */
            r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX2 * (MAX-MIN)) + MIN   */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation          */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE2) * INDEX2       */

            r3e = INC2
            a1 = *r2 + *r3                  /* Phase2 = Phase2 + Inc2            */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + (Mod1*Inc1)     */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */
```

```
less2:      r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r4                        /* a1 = INDEX3                          */
            *r4 = a0 = *r4 + *r6            /* INDEX3 = INDEX3 + INDEX4_INC         */

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                       /* Multiply by 4 for float              */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* r5 points to sine table              */
            *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3               */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase3 + Inc3 + (mod1 * inc1)        */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)
            3*nop
            r3e = *r3                       /* r3 = PHASE3                          */
            nop
            r3 - 256                        /* See if we're past end of table       */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table               */
            pcgoto ok3
            nop

less3:      r3 - 0
            if (gt) pcgoto ok3
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case                 */

ok3:        r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                        /* a1 = INDEX4                          */
            *r4 = a0 = *r4 + *r6            /* INDEX4 = INDEX4 + INDEX4_INC         */

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                       /* Multiply by 4 for float              */
            r5e = r1 + SINE_TABLE
            r6e = MOD3

            a0 = *r6 + a1 * *r5             /* Mod3 + (Envelope * Sine[PHASE4] */
            r1e = PT_FIVE
            nop
            a0 = a0 * *r1                   /* Divide by 2                          */
            2*nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags              */
            r3e = *r8                       /* r2 contains flags from host          */
            nop                             /* latent instruction                   */
            r3e & VIEW_FLAG                 /* test the view bit                    */
            if (eq) pcgoto Sound            /* we want to hear it                   */
            nop

            a0 = a1 * *r5                   /* Just viewing                         */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer             */
            pcgoto Cont
            nop

Sound:      *r10++ = a0 = *r10 + a0         /* Sum into PR3                         */
```

```
Cont:       r3e = INC4
            r5e = INC2
            r6e = MOD2
            a1 = *r2 + *r3                  /* PHASE4 + INC4                    */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD2*INC2)      */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD3 - OLD_SIG3  */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags          */
            r2e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag        */
            *r8 = r2e                       /* Write flags to memory            */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
```

```
INDEX4:         float   0.0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAlg5.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis algorithm 5.
/*
/*  This algorithm gets loaded onto the DSP when the user plays a voice
/*  of FM synthesis when the configuration is set to "5."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*                          -------
/*                          |     |
/*                          |  1  | Osc 1
/*                          |     |
/*                          -------
/*                             |
/*                ------------------------------
/*               V              V               V
/*           -------        -------         -------
/*           |     |        |     |         |     |
/*           |  2  | Osc 2  |  3  | Osc 3   |  4  | Osc 4
/*           |     |        |     |         |     |
/*           -------        -------         -------
/*              |---------------|---------------|
/*                              +
/*                              O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "FM_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"

.rsect  ".prog"
            r10e = r16 + DH_HADDR + PHYSICAL    /* PRB or Sample buffer         */
            r10e = *r10
            r9e = r17                           /* host parameters              */

            r8e = PARAMS
            r7e = 35                            /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1         /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2

loop2:      *r8++ = a0 = *r2 * *r9++            /* Set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + FM_FLAGS
            r2e = *r7                           /* r2 contains flags from host  */
            nop                                 /* latent instruction           */
            r2e & VIEW_FLAG                     /* test the view bit            */
            if (eq) pcgoto Next                 /* we want to hear it           */
            nop

            r10e = r17 + FM_RETURN_DATA         /* point r10 to param block     */
            r2e & HOST_READY_FLAG               /* test the host_ready bit      */
            if (eq) pcgoto EndIt                /* host is not ready            */

Next:       r8e = r17
            a0 = dsp(*r8)                       /* NumSamples                   */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int24(a0)                /* convert to an integer        */
            3*nop
            r2e = *r2                           /* r2 = NumSamples              */
            nop

            r2e = r2 - 240                      /* Subtract the frame size      */
            r2 - 0                              /* See if we need more samples  */
            if (ge) pcgoto Next2                /* need more samples            */
```

```
            r3e = INDEX                     /* don't need samples                */
            r2e = ZERO                      /* zero out our VAR parameters       */
            r7e = 7                         /* there are 9 of them               */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters  */
            *r3++ = a0 = *r2                /* Zero out the parameters           */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag         */

            pcgoto EndIt                    /* exit the module                   */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp     */
            nop
            a0 = float(*r4)                 /* convert to float                  */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining          */

            r11e = INDEX
            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                       */
            a2 = a1 - *r6                   /* Subtract 0.5 for round            */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                        */
            3*nop
            r2e = *r2                       /* r2 = Start                        */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)      */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env[start]                        */
            r4e = r3 + 4                    /* Env[finish]                       */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1            */      /* INDEX_INC1 = a0 * COMPARE

            r3e = r2 + FM_ENV2              /* Env[start]                        */
            r4e = r3 + 4                    /* Env[finish]                       */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2            */

            r3e = r2 + FM_ENV3              /* Env[start]                        */
            r4e = r3 + 4                    /* Env[finish]                       */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3            */

            r3e = r2 + FM_ENV4              /* Env[start]                        */
            r4e = r3 + 4                    /* Env[finish]                       */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4            */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag                */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes            */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
```

```
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 238                       /* Loop count (frameSize-2)          */
            r9e = SINE_SIZE

Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1      */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN   */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC      */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                       */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation          */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1       */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1            */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */

less1:      r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r4                        /* a1 = INDEX2                       */
            *r4 = a0 = *r4 + *r6            /* INDEX2 = INDEX2 + INDEX2_INC      */

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                       */
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* r5 is index into sine table       */
            r3e = PT_FIVE
            *r1 = a0 = a1 * *r5             /* SINE_TABLE[PHASE2] * INDEX2       */

            r3e = INC2
            a1 = *r2 + *r3                  /* a1 = Phase2 + Inc2                */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + Inc2 + (MOD1*INC1) */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */
```

```
less2:      r3 - 0
            if (gt) pcgoto ok2
            nop
            *r2 = a0 = *r2 + *r9          /* Handle negative case          */

ok2:        r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r4                      /* a1 = INDEX3                   */
            *r4 = a0 = *r4 + *r6          /* INDEX3 = INDEX3 + INDEX3_INC  */

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                     /* Multiply by 4 for float       */
            r4 = r4*2
            r5e = r4 + SINE_TABLE         /* Need to do interpolation      */
            *r1 = a0 = *r5 * a1           /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3                /* Phase3 = Phase3 + Inc3        */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5     /* Phase3 + Inc3 + (mod1 * inc1) */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)         /* r3 = PHASE3                   */
            3*nop
            r3e = *r3
            nop
            r3 - 256                      /* See if we're past end of table */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9          /* Wrap to start of table        */
            pcgoto ok3
            nop

less3:      r3 - 0
            if (gt) pcgoto ok3
            nop
            *r2 = a0 = *r2 + *r9          /* Handle negative case          */

ok3:        r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                      /* a1 = INDEX4                   */
            *r4 = a0 = *r4 + *r6          /* INDEX4 = INDEX4 + INDEX4_INC  */

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                     /* Multiply by 4 for float       */
            r5e = r1 + SINE_TABLE
            r6e = MOD3
            a0 = *r6 + a1 * *r5           /* Mod3 + Carrier                */
            r3e = MOD2
            nop
            a0 = a0 + *r3                 /* Mod3 + Carrier + Mod2         */
            r1e = PT_THREE
            nop
            a0 = a0 * *r1                 /* Divide by 3                   */
            2*nop

            r8e = r17 + FM_FLAGS          /* r8 points to host flags       */
            r3e = *r8                     /* r2 contains flags from host   */
            nop                           /* latent instruction            */
            r3e & VIEW_FLAG               /* test the view bit             */
            if (eq) pcgoto Sound          /* we want to hear it            */
            nop

            a0 = a1 * *r5                 /* Just viewing                  */
```

```
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer        */
            pcgoto Cont
            nop

Sound:      *r10++ = a0 = *r10 + a1 * *r5   /* Sum into PRB                    */

Cont:       r3e = INC4
            a1 = *r2 + *r3                  /* PHASE4 + INC4                   */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* PHASE4 = PHASE4 + (MOD1*INC1)   */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case            */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index             */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame          */
            nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags         */
            r2e = *r8                       /* r2 contains flags from host     */
            nop                             /* latent instruction              */
            r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag       */
            *r8 = r2e                       /* Write flags to memory           */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
PT_THREE:       float   0.3933333
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
```

```
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAlg6.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis algorithm 6.
/*
/*  This algorithm gets loaded onto the DSP when the user plays a voice
/*  of FM synthesis when the configuration is set to "6."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*                                 -------
/*                                |       |
/*                                |   1   | Osc 1
/*                                |       |
/*                                -------
/*                                   |
/*                              ----------------
/*                              V               V
/*          -------          -------          -------
/*          |     |          |     |          |     |
/*          |  2  | Osc 2    |  3  | Osc 3    |  4  | Osc 4
/*          |     |          |     |          |     |
/*          -------          -------          -------
/*             |---------------|---------------|
/*                             +
/*                             O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "FM_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"

.rsect  ".prog"
            r10e = r16 + DH_HADDR + PHYSICAL    /* PRB or Sample buffer         */
            r10e = *r10
            r9e = r17                           /* host parameters              */

            r8e = PARAMS
            r7e = 35                            /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1         /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2

loop2:      *r8++ = a0 = *r2 * *r9++            /* Set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + FM_FLAGS
            r2e = *r7                           /* r2 contains flags from host  */
            nop                                 /* latent instruction           */
            r2e & VIEW_FLAG                     /* test the view bit            */
            if (eq) pcgoto Next                 /* we want to hear it           */
            nop

            r10e = r17 + FM_RETURN_DATA         /* point r10 to param block     */
            r2e & HOST_READY_FLAG               /* test the host_ready bit      */
            if (eq) pcgoto EndIt                /* host is not ready            */

Next:       r8e = r17
            a0 = dsp(*r8)                       /* NumSamples                   */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int24(a0)                /* convert to an integer        */
            3*nop
            r2e = *r2                           /* r2 = NumSamples              */
            nop

            r2e = r2 - 240                      /* Subtract the frame size      */
            r2 - 0                              /* See if we need more samples  */
            if (ge) pcgoto Next2                /* need more samples            */
```

```
            r3e = INDEX                 /* don't need samples                */
            r2e = ZERO                  /* zero out our VAR parameters       */
            r7e = 7                     /* there are 9 of them               */
here:       if (r7-- >= 0) pcgoto here  /* Re-initaillize the VAR parameters */
            *r3++ = a0 = *r2            /* Zero out the parameters           */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                   /* Turn on need to init flag         */

            pcgoto EndIt                /* exit the module                   */

Next2:      r4e = TEMP_INT
            *r4 = r2                    /* put samples remaining in temp     */
            nop
            a0 = float(*r4)             /* convert to float                  */
            2*nop
            *r8 = a1 = ieee(a0)         /* Update samples remaining          */

            r11e = INDEX
            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7             /* I * COMPARE                       */
            a2 = a1 - *r6               /* Subtract 0.5 for round            */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)          /* a2 = Start                        */
            3*nop
            r2e = *r2                   /* r2 = Start                        */
            nop
            r2 = r2*2
            r2 = r2*2                   /* r2 = r2 * 4 (for float size)      */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1          /* Env[start]                        */
            r4e = r3 + 4                /* Env[finish]                       */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 1            */          /* INDEX_INC1 = a0 * COMPARE

            r3e = r2 + FM_ENV2          /* Env[start]                        */
            r4e = r3 + 4                /* Env[finish]                       */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 2            */

            r3e = r2 + FM_ENV3          /* Env[start]                        */
            r4e = r3 + 4                /* Env[finish]                       */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 3            */

            r3e = r2 + FM_ENV4          /* Env[start]                        */
            r4e = r3 + 4                /* Env[finish]                       */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 4            */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                   /* Turn off init flag                */

            r3e = PARAMS + FM_ENV1      /* Set up initial indexes            */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
```

```
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 238                       /* Loop count (frameSize-2)        */
            r9e = SINE_SIZE

Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1    */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC    */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                     */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation        */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1     */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1          */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */

less1:      r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r4                        /* a1 = INDEX2                     */
            *r4 = a0 = *r4 + *r6            /* INDEX2 = INDEX2 + INDEX2_INC    */

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                     */
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* r5 is index into sine table     */
            r3e = PT_FIVE
            *r1 = a0 = a1 * *r5             /* SINE_TABLE[PHASE2] * INDEX2     */

            r3e = INC2
            *r2 = a1 = *r2 + *r3            /* Phase2 = Phase2 + Inc2          */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */

less2:      r6e = INDEX3_INC
            r4e = INDEX3
```

```
          a1 = *r4                        /* a1 = INDEX3                          */
          *r4 = a0 = *r4 + *r6            /* INDEX3 = INDEX3 + INDEX3_INC         */

          r2e = PHASE3
          r4e = TEMP_INT
          *r4 = a2 = int24 (*r2)
          3*nop
          r4e = *r4
          r1e = MOD3
          r4 = r4*2                       /* Multiply by 4 for float              */
          r4 = r4*2
          r5e = r4 + SINE_TABLE           /* Need to do interpolation             */
          *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

          r3e = INC3
          a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3               */
          r4e = MOD1
          r5e = INC1
          *r2 = a0 = a1 + *r4 * *r5       /* Phase3 + Inc3 + (mod1 * inc1)        */
          2*nop

          r3e = TEMP_INT
          *r3 = a0 = int24(*r2)           /* r3 = PHASE3                          */
          3*nop
          r3e = *r3
          nop
          r3 - 256                        /* See if we're past end of table       */
          if (lt) pcgoto less3
          nop
          *r2 = a0 = *r2 - *r9            /* Wrap to start of table               */
          pcgoto ok3
          nop

less3:    r3 - 0
          if (gt) pcgoto ok3
          nop
          *r2 = a0 = *r2 + *r9            /* Handle negative case                 */

ok3:      r6e = INDEX4_INC
          r4e = INDEX4
          a1 = *r4                        /* a1 = INDEX4                          */
          *r4 = a0 = *r4 + *r6            /* INDEX4 = INDEX4 + INDEX4_INC         */

          r2e = PHASE4
          r1e = TEMP_INT
          *r1 = a2 = int(*r2)
          3*nop
          r1e = *r1
          nop
          r1 = r1*2
          r1 = r1*2                       /* Multiply by 4 for float              */
          r5e = r1 + SINE_TABLE
          r6e = MOD3
          a0 = *r6 + a1 * *r5             /* Mod3 + Carrier                       */
          r1e = PI_THREE
          r3e = MOD2
          a0 = a0 + *r3                   /* Mod3 + Carrier + Mod2                */
          2*nop
          a0 = a0 * *r1                   /* Divide by 3                          */
          2*nop

          r8e = r17 + FM_FLAGS            /* r8 points to host flags              */
          r3e = *r8                       /* r2 contains flags from host          */
          nop                             /* latent instruction                   */
          r3e & VIEW_FLAG                 /* test the view bit                    */
          if (eq) pcgoto Sound            /* we want to hear it                   */
          nop

          a0 = a1 * *r5                   /* Just viewing                         */
          2*nop
          *r10++ = a0 = ieee(a0)          /* Place into sample buffer             */
          pcgoto Cont
          nop

Sound:    *r10++ = a0 = *r10 + a1 * *r5   /* Sum into PRB                         */

Cont:     r3e = INC4
          r5e = INC1
```

```
            r6e = MOD1
            a1 = *r2 + *r3                  /* PHASE4 + INC4                    */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD1*INC1)      */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD1 - OLD_SIG3  */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags          */
            r2e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag        */
            *r8 = r2e                       /* Write flags to memory            */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
PT_THREE:       float   0.3933333
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
```

```
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAlg7.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis algorithm 7.
/*
/*  This algorithm gets loaded onto the DSP when the user plays a voice
/*  of FM synthesis when the configuration is set to "7."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*                                        -------
/*                                        |     |
/*                                        |  1  | Osc 1
/*                                        |     |
/*                                        -------
/*                                           |
/*                                           V
/*          -------         -------         -------
/*          |     |         |     |         |     |
/*          |  2  | Osc 2   |  3  | Osc 3   |  4  | Osc 4
/*          |     |         |     |         |     |
/*          -------         -------         -------
/*             |---------------|---------------|
/*                             +
/*                             O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "FM_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"

.rsect  ".prog"
            r10e = r16 + DH_HADDR + PHYSICAL    /* PRB or Sample buffer         */
            r10e = *r10
            r9e = r17                           /* host parameters              */

            r8e = PARAMS
            r7e = 35                            /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1         /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2

loop2:      *r8++ = a0 = *r2 * *r9++            /* Set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + FM_FLAGS
            r2e = *r7                           /* r2 contains flags from host  */
            nop                                 /* latent instruction           */
            r2e & VIEW_FLAG                     /* test the view bit            */
            if (eq) pcgoto Next                 /* we want to hear it           */
            nop

            r10e = r17 + FM_RETURN_DATA         /* point r10 to param block     */
            r2e & HOST_READY_FLAG               /* test the host_ready bit      */
            if (eq) pcgoto EndIt                /* host is not ready            */

Next:       r8e = r17
            a0 = dsp(*r8)                       /* NumSamples                   */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int24(a0)                /* convert to an integer        */
            3*nop
            r2e = *r2                           /* r2 = NumSamples              */
            nop

            r2e = r2 - 240                      /* Subtract the frame size      */
            r2 - 0                              /* See if we need more samples  */
            if (ge) pcgoto Next2                /* need more samples            */

            r3e = INDEX                         /* don't need samples           */
```

```
            r2e = ZERO                      /* zero out our VAR parameters      */
            r7e = 7                         /* there are 9 of them              */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters */
            *r3++ = a0 = *r2                /* Zero out the parameters          */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag        */

            pcgoto EndIt                    /* exit the module                  */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp    */
            nop
            a0 = float(*r4)                 /* convert to float                 */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining         */

            r11e = INDEX
            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                      */
            a2 = a1 - *r6                   /* Subtract 0.5 for round           */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                       */
            3*nop
            r2e = *r2                       /* r2 = Start                       */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)     */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1           */          /* INDEX_INC1 = a0 * COMPARE

            r3e = r2 + FM_ENV2              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2           */

            r3e = r2 + FM_ENV3              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3           */

            r3e = r2 + FM_ENV4              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4           */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag               */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes           */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
```

```
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 238                       /* Loop count (frameSize-2)          */
            r9e = SINE_SIZE

Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1      */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN   */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC      */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                       */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation          */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1       */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1            */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */

less1:      r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r4                        /* a1 = INDEX2                       */
            *r4 = a0 = *r4 + *r6            /* INDEX2 = INDEX2 + INDEX2_INC      */

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE2                       */
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* r5 is index into sine table       */
            r3e = PT_FIVE
            *r1 = a0 = a1 * *r5             /* SINE_TABLE[PHASE2] * INDEX2       */

            r3e = INC2
            *r2 = a1 = *r2 + *r3            /* Phase2 = Phase2 + Inc2            */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */

less2:      r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r4                        /* a1 = INDEX3                       */
```

```
            *r4 = a0 = *r4 + *r6            /* INDEX3 = INDEX3 + INDEX3_INC     */

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* r5 is index into sine table      */
            *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            *r2 = a1 = *r2 + *r3            /* Phase3 = Phase3 + Inc3           */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE3                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less3:      r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                        /* a1 = INDEX4                      */
            *r4 = a0 = *r4 + *r6            /* INDEX4 = INDEX4 + INDEX4_INC     */

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                       /* Multiply by 4 for float          */
            r5e = r1 + SINE_TABLE
            r6e = MOD3
            a0 = *r6 + a1 * *r5             /* Mod3 + Carrier                   */
            r1e = PT_THREE
            r3e = MOD2
            a0 = a0 + *r3                   /* Mod3 + Carrier + Mod2            */
            2*nop
            a0 = a0 * *r1                   /* Divide by 3                      */
            2*nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags          */
            r3e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r3e & VIEW_FLAG                 /* test the view bit                */
            if (eq) pcgoto Sound            /* we want to hear it               */
            nop

            a0 = a1 * *r5                   /* Just viewing                     */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer         */
            pcgoto Cont
            nop

Sound:      *r10++ = a0 = *r10 + a1 * *r5   /* Sum into PRB                     */

Cont:       r3e = INC4
            r5e = INC1
            r6e = MOD1
            a1 = *r2 + *r3                  /* PHASE4 + INC4                    */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD1*INC1)      */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD1 - OLD_SIG3  */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                      */
```

```
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + FM_FLAGS            /* r8 points to host flags          */
            r2e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag        */
            *r8 = r2e                       /* Write flags to memory            */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
MAX_AMP:        float   (1.0)
TEMP_FLOAT:     float
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
PT_THREE:       float   0.3933333
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
```

```
float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
float   -0.024646, 0.0000000
```

# FMAnalysis1.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis FFT data generator algorithm 1.
/*
/*  This algorithm gets loaded onto the DSP when the user opens the
/*  analysis window when using FM synthesis with the configuration set to "1."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*              -------
/*              |     |
/*              |  1  |     Oscillator 1
/*              |     |
/*              -------
/*                 |
/*                 V
/*              -------
/*              |     |
/*              |  2  |     Oscillator 2
/*              |     |
/*              -------
/*                 |
/*                 V
/*              -------
/*              |     |
/*              |  3  |     Oscillator 3
/*              |     |
/*              -------
/*                 |
/*                 V
/*              -------
/*              |     |
/*              |  4  |     Oscillator 4
/*              |     |
/*              -------
/*                 |
/*                 O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "FM_Offsets.h"


.rsect  ".prog"
            r10e = r17                      /* Data Buffer                  */

            r7e = r17 + FM_FLAGS
            r2e = *r7                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e & HOST_READY_FLAG           /* test the host ready bit      */
            if (eq) pcgoto EndIt            /* host is not ready            */

            r9e = r17                       /* host parameters              */
            r8e = PARAMS
            r7e = 35                        /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1     /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2
loop2:      *r8++ = a0 = *r2 * *r9++        /* set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop


Next:       r8e = r17
            a0 = dsp(*r8)
            r2e = TEMP_INT                  /* NumSamples                   */
            nop
            *r2 = a1 = int24(a0)
            3*nop
```

```
        r2e = *r2                       /* r2 = NumSamples                  */
        nop
        r2e = r2 - 128                  /* subtract the frame size          */
        r2 - 0                          /* need to output more samples?     */
        if (ge) pcgoto Next2

        r3e = INDEX                     /* don't need samples               */
        r2e = ZERO                      /* zero out our VAR parameters      */
        r7e = 7                         /* there are 9 of them              */
here:   if (r7-- >= 0) pcgoto here      /* Re-initaillize the VAR parameters */
        *r3++ = a0 = *r2                /* Zero out the parameters          */
        r4e = NEED_TO_INIT
        r2e = INIT_FLAG
        *r4 = r2e                       /* Turn on need to init flag        */

        pcgoto EndIt                    /* exit the module                  */

Next2:  r4e = TEMP_INT
        *r4 = r2                        /* put samples remaining in temp    */
        nop
        a0 = float(*r4)                 /* convert to float                 */
        2*nop
        *r8 = a1 = ieee(a0)             /* Update samples remaining         */

        r10e += FM_RETURN_DATA          /* r10 points to output buffer      */

        r4e = FIRST_HALF                /* see if we're doing 1st half      */
        r2e = *r4
        nop
        r2e & SECOND_HALF
        if (eq) pcgoto Next3            /* doing 1st half                   */
        nop

        r10e = r10 + 508                /* doing 2nd half, add offset       */
        r8e = r17 + FM_FLAGS
        r2e &= ~SECOND_HALF
        *r4 = r2e                       /* Toggle FIRST_HALF                */

        r2e = *r8                       /* r2 contains flags from host      */
        nop                             /* latent instruction               */
        r2e &= ~HOST_READY_FLAG         /* clear the host_ready flag        */
        r2e |= DO_FFT_FLAG              /* set the do_fft flag              */
        *r8 = r2e                       /* write flags to memory            */

        pcgoto Next4
        nop

Next3:  r2e |= SECOND_HALF
        *r4 = r2e                       /* Toggle FIRST_HALF                */

Next4:  r11e = INDEX                    /* Read index from host             */
        r2e = r17 + FM_INDEX
        *r11 = a0 = dsp(*r2)

        r7e = COMPARE
        r6e = PT_FIVE
        a1 = *r11 * *r7                 /* I * COMPARE                      */
        a2 = a1 - *r6                   /* Subtract 0.5 for round           */
        r2e = TEMP_INT
        nop
        *r2 = a2 = int(a2)              /* a2 = Start                       */
        3*nop
        r2e = *r2                       /* r2 = Start                       */
        nop
        r2 = r2*2
        r2 = r2*2                       /* r2 = r2 * 4 (for float size)     */
        r2e = PARAMS + r2
        r3e = r2 + FM_ENV1              /* Env[start]                       */
        r4e = r3 + 4                    /* Env[finish]                      */
        a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
        r5e = INDEX1_INC
        nop
        *r5 = a1 = a0 * *r7             /* Save Index increment 1           */

        r3e = r2 + FM_ENV2              /* Env[start]                       */
        r4e = r3 + 4                    /* Env[finish]                      */
        a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
        r5e = INDEX2_INC
```

```
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2          */

            r3e = r2 + FM_ENV3              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3          */

            r3e = r2 + FM_ENV4              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4          */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag              */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes          */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 126                       /* Loop count (frameSize-2)        */
            r9e = SINE_SIZE


Mainloop:   r1e = PARAMS + FM_MAXINDEX1
            r5e = PARAMS + FM_MININDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1    */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC    */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                     */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation        */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1     */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1          */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less1
            nop
```

```
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */

less1:      r5e = PARAMS + FM_MININDEX2
            r1e = PARAMS + FM_MAXINDEX2
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX2 - MIN_INDEX2    */
            r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX2 * (MAX-MIN)) + MIN */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation        */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE2] * INDEX2     */

            r3e = INC2
            a1 = *r2 + *r3                  /* Phase2 = Phase2 + Inc2          */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + (Mod1*Inc1)   */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */
            pcgoto ok2
            nop

less2:      r3 - 0
            if (gt) pcgoto ok2
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case            */

ok2:        r5e = PARAMS + FM_MININDEX3
            r1e = PARAMS + FM_MAXINDEX3
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX3 - MIN_INDEX3    */
            r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX3 * (MAX-MIN)) + MIN */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation        */
            *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3          */
            r4e = MOD2
            r5e = INC2
            *r2 = a0 = a1 + *r4 * *r5       /* Phase3 = Phase3 + (mod2 * inc2) */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE3                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less3
```

```
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */
            pcgoto ok3
            nop

less3:      r3 - 0
            if (gt) pcgoto ok3
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case            */

ok3:        r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                        /* a1 = INDEX4                     */
            *r4 = a0 = *r4 + *r6            /* INDEX4 = INDEX4 + INDEX4_INC    */

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                       /* Multiply by 4 for float         */
            r5e = r1 + SINE_TABLE           /* r5 is index into sine table     */
            r6e = MOD3

            a0 = a1 * *r5                   /* a0 = Index4 * sine_table[r5]    */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer        */

            r3e = INC4
            r5e = INC3
            a1 = *r2 + *r3                  /* PHASE4 + INC4                   */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD3*INC3)     */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD3 - OLD_SIG3 */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case            */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index             */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame          */
            nop

            r8e = r17 + FM_INDEX
            *r8 = a0 = ieee(*r11)           /* Write Index to host             */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
```

```
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
FIRST_HALF:     fltbits 0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
```

```
float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
float   -0.024646, 0.0000000
```

# FMAnalysis2.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis FFT data generator algorithm 2.
/*
/*  This algorithm gets loaded onto the DSP when the user opens the
/*  analysis window when using FM synthesis with the configuration set to "2."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*              -------                     -------
/*              |     |                     |     |
/*              |  1  | Oscillator 1        |  2  | Oscillator 2
/*              |     |                     |     |
/*              -------                     -------
/*                 |-------------- + ----------|
/*                               |
/*                               V
/*                            -------
/*                            |     |
/*                            |  3  |     Oscillator 3
/*                            |     |
/*                            -------
/*                               |
/*                               V
/*                            -------
/*                            |     |
/*                            |  4  |     Oscillator 4
/*                            |     |
/*                            -------
/*                               |
/*                               O    <---- Output
/*
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "FM_Offsets.h"


.rsect  ".prog"
            r10e = r17                      /* Data Buffer                  */

            r7e = r17 + FM_FLAGS
            r2e = *r7                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e & HOST_READY_FLAG           /* test the host ready bit      */
            if (eq) pcgoto EndIt            /* host is not ready            */

            r9e = r17                       /* host parameters              */
            r8e = PARAMS
            r7e = 35                        /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1     /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2
loop2:      *r8++ = a0 = *r2 * *r9++        /* set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop


Next:       r8e = r17
            a0 = dsp(*r8)
            r2e = TEMP_INT                  /* NumSamples                   */
            nop
            *r2 = a1 = int24(a0)
            3*nop
            r2e = *r2                       /* r2 = NumSamples              */
            nop
            r2e = r2 - 128                  /* subtract the frame size      */
            r2 - 0                          /* need to output more samples? */
            if (ge) pcgoto Next2
```

```
            r3e = INDEX                     /* don't need samples               */
            r2e = ZERO                      /* zero out our VAR parameters      */
            r7e = 7                         /* there are 9 of them              */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters */
            *r3++ = a0 = *r2                /* Zero out the parameters          */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag        */

            pcgoto EndIt                    /* exit the module                  */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp    */
            nop
            a0 = float(*r4)                 /* convert to float                 */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining         */

            r10e += FM_RETURN_DATA          /* r10 points to output buffer      */

            r4e = FIRST_HALF                /* see if we're doing 1st half      */
            r2e = *r4
            nop
            r2e & SECOND_HALF
            if (eq) pcgoto Next3            /* doing 1st half                   */
            nop

            r10e = r10 + 508                /* doing 2nd half, add offset       */
            r8e = r17 + FM_FLAGS
            r2e &= ~SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF                */

            r2e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r2e &= ~HOST_READY_FLAG         /* clear the host_ready flag        */
            r2e |= DO_FFT_FLAG              /* set the do_fft flag              */
            *r8 = r2e                       /* write flags to memory            */

            pcgoto Next4
            nop

Next3:      r2e |= SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF                */

Next4:      r11e = INDEX                    /* Read index from host             */
            r2e = r17 + FM_INDEX
            *r11 = a0 = dsp(*r2)

            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                      */
            a2 = a1 - *r6                   /* Subtract 0.5 for round           */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                       */
            3*nop
            r2e = *r2                       /* r2 = Start                       */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)     */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1           */

            r3e = r2 + FM_ENV2              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2           */

            r3e = r2 + FM_ENV3              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
```

```
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3           */

            r3e = r2 + FM_ENV4              /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4           */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= -INIT_FLAG
            *r8 = r2e                       /* Turn off init flag               */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes           */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 126                       /* Loop count (frameSize-2)         */
            r9e = SINE_SIZE

Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1     */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN  */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC     */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                      */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE(PHASE1) * INDEX1      */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1           */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less1:      r5e = PARAMS + FM_MININDEX2
            r1e = PARAMS + FM_MAXINDEX2
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX2 - MIN_INDEX2     */
            r6e = INDEX2_INC
```

```
            r4e = INDEX2
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX2 * (MAX-MIN)) + MIN  */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE2] * INDEX2      */

            r3e = INC2
            a1 = *r2 + *r3                  /* Phase2 = Phase2 + Inc2           */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + (Mod1*Inc1)    */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less2:      r5e = PARAMS + FM_MININDEX3
            r1e = PARAMS + FM_MAXINDEX3
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX3 - MIN_INDEX3     */
            r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX3 * (MAX-MIN)) + MIN  */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3           */
            r4e = MOD2
            r5e = INC2
            r1e = MOD1
            r8e = INC1
            a0 = a1 + *r4 * *r5             /* Phase3 + Inc3 + (mod2 * inc2)    */
            2*nop
            *r2 = a0 = a0 + *r1 * *r8       /* Add MOD1 * INC1                  */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE3                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok3
            nop

less3:      r3 - 0
            if (gt) pcgoto ok3
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */
```

```
ok3:        r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                        /* a1 = INDEX4                      */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                       /* Multiply by 4 for float          */
            r5e = r1 + SINE_TABLE           /* r5 is index into sine table      */
            r6e = MOD3

            a0 = a1 * *r5                   /* a0 = Index4 * sine_table[r5]     */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer         */

            r3e = INC4
            r5e = INC3
            a1 = *r2 + *r3                  /* PHASE4 + INC4                    */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD3*INC3)      */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD3 - OLD_SIG3  */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + FM_INDEX
            *r8 = a0 = ieee(*r11)           /* Write Index to host              */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
```

```
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
FIRST_HALF:     fltbits 0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAnalysis3.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis FFT data generator algorithm 3.
/*
/*  This algorithm gets loaded onto the DSP when the user opens the
/*  analysis window when using FM synthesis with the configuration set to "3."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*                                        -------
/*                                        |     |
/*                                        |  2  | Oscillator 2
/*                                        |     |
/*                                        -------
/*                                           |
/*                                           V
/*              -------                   -------
/*              |     |                   |     |
/*              |  1  | Oscillator 1      |  3  |     Oscillator 3
/*              |     |                   |     |
/*              -------                   -------
/*                  |------------ + ---------|
/*                                |
/*                                V
/*                             -------
/*                             |     |
/*                             |  4  |     Oscillator 4
/*                             |     |
/*                             -------
/*                                |
/*                                O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "FM_Offsets.h"


.rsect  ".prog"
            r10e = r17                  /* Data Buffer */

            r7e = r17 + FM_FLAGS
            r2e = *r7                   /* r2 contains flags from host      */
            nop                         /* latent instruction               */
            r2e & HOST_READY_FLAG       /* test the host ready bit          */
            if (eq) pcgoto EndIt        /* host is not ready                */

            r9e = r17                   /* host parameters                  */
            r8e = PARAMS
            r7e = 35                    /* number of params - 2             */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1 /* copy over all params             */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2
loop2:      *r8++ = a0 = *r2 * *r9++    /* set up INC1 - INC4               */
            if (r6-- >= 0) pcgoto loop2
            nop


Next:       r8e = r17
            a0 = dsp(*r8)
            r2e = TEMP_INT              /* NumSamples                       */
            nop
            *r2 = a1 = int24(a0)
            3*nop
            r2e = *r2                   /* r2 = NumSamples                  */
            nop
            r2e = r2 - 128              /* subtract the frame size          */
            r2 - 0                      /* need to output more samples?     */
            if (ge) pcgoto Next2
```

```
            r3e = INDEX                     /* don't need samples                */
            r2e = ZERO                      /* zero out our VAR parameters       */
            r7e = 7                         /* there are 9 of them               */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters  */
            *r3++ = a0 = *r2                /* Zero out the parameters           */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag         */

            pcgoto EndIt                    /* exit the module                   */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp     */
            nop
            a0 = float(*r4)                 /* convert to float                  */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining          */

            r10e += FM_RETURN_DATA          /* r10 points to output buffer       */

            r4e = FIRST_HALF                /* see if we're doing 1st half       */
            r2e = *r4
            nop
            r2e & SECOND_HALF
            if (eq) pcgoto Next3            /* doing 1st half                    */
            nop

            r10e = r10 + 508                /* doing 2nd half, add offset        */
            r8e = r17 + FM_FLAGS
            r2e &= ~SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF                 */

            r2e = *r8                       /* r2 contains flags from host       */
            nop                             /* latent instruction                */
            r2e &= ~HOST_READY_FLAG         /* clear the host_ready flag         */
            r2e |= DO_FFT_FLAG              /* set the do_fft flag               */
            *r8 = r2e                       /* write flags to memory             */

            pcgoto Next4
            nop

Next3:      r2e |= SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF                 */

Next4:      r11e = INDEX                    /* Read index from host              */
            r2e = r17 + FM_INDEX
            *r11 = a0 = dsp(*r2)

            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                       */
            a2 = a1 - *r6                   /* Subtract 0.5 for round            */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                        */
            3*nop
            r2e = *r2                       /* r2 = Start                        */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)      */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env[start]                        */
            r4e = r3 + 4                    /* Env[finish]                       */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1            */

            r3e = r2 + FM_ENV2              /* Env[start]                        */
            r4e = r3 + 4                    /* Env[finish]                       */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]     */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2            */

            r3e = r2 + FM_ENV3              /* Env[start]                        */
            r4e = r3 + 4                    /* Env[finish]                       */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]     */
```

```
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3          */

            r3e = r2 + FM_ENV4              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4          */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag              */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes          */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 126                       /* Loop count (frameSize-2)        */
            r9e = SINE_SIZE


Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1    */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC    */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                     */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation        */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1     */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1          */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */

less1:      r5e = PARAMS + FM_MININDEX2
            r1e = PARAMS + FM_MAXINDEX2
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX2 - MIN_INDEX2    */
            r6e = INDEX2_INC
```

```
        r4e = INDEX2
        a1 = *r5 + a3 * *r4             /* a1 = (INDEX2 * (MAX-MIN)) + MIN  */
        *r4 = a0 = *r4 + *r6

        r2e = PHASE2
        r4e = TEMP_INT
        *r4 = a2 = int24 (*r2)
        3*nop
        r4e = *r4
        r1e = MOD2
        r4 = r4*2                       /* Multiply by 4 for float          */
        r4 = r4*2
        r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
        *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE2] * INDEX2      */

        r3e = INC2
        a1 = *r2 + *r3                  /* Phase2 = Phase2 + Inc2           */
        r4e = MOD1
        r5e = INC1
        *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + (Mod1*Inc1)    */
        2*nop

        r3e = TEMP_INT
        *r3 = a0 = int24(*r2)           /* r3 = PHASE2                      */
        3*nop
        r3e = *r3
        nop
        r3 - 256                        /* See if we're past end of table   */
        if (lt) pcgoto less2
        nop
        *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
        pcgoto ok2
        nop

less2:  r3 - 0
        if (gt) pcgoto ok2
        nop
        *r2 = a0 = *r2 - *r9            /* Handle negative case             */

ok2:    r5e = PARAMS + FM_MININDEX3
        r1e = PARAMS + FM_MAXINDEX3
        a3 = *r1 - *r5                  /* a3 = MAX_INDEX3 - MIN_INDEX3     */
        r6e = INDEX3_INC
        r4e = INDEX3
        a1 = *r5 + a3 * *r4             /* a1 = (INDEX3 * (MAX-MIN)) + MIN  */
        *r4 = a0 = *r4 + *r6

        r2e = PHASE3
        r4e = TEMP_INT
        *r4 = a2 = int24 (*r2)
        3*nop
        r4e = *r4
        r1e = MOD3
        r4 = r4*2                       /* Multiply by 4 for float          */
        r4 = r4*2
        r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
        *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

        r3e = INC3
        a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3           */
        r4e = MOD2
        r5e = INC2
        *r2 = a0 = a1 + *r4 * *r5       /* Phase3 = Phase3 + (mod2 * inc2)  */
        2*nop

        r3e = TEMP_INT
        *r3 = a0 = int24(*r2)           /* r3 = PHASE3                      */
        3*nop
        r3e = *r3
        nop
        r3 - 256                        /* See if we're past end of table   */
        if (lt) pcgoto less3
        nop
        *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less3:  r6e = INDEX4_INC
        r4e = INDEX4
        a1 = *r4                        /* a1 = INDEX4                      */
```

```
            *r4 = a0 = *r4 + *r6

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                       /* Multiply by 4 for float          */
            r5e = r1 + SINE_TABLE           /* r5 is index into sine table      */
            r6e = MOD3

            a0 = a1 * *r5                   /* a0 = Index4 * sine_table[r5]     */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer         */

            r3e = INC4
            r5e = INC3
            r1e = INC2
            r8e = MOD2
            a1 = *r2 + *r3                  /* PHASE4 + INC4                    */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD3*INC3)      */
            2*nop
            a2 = a2 + *r8 * *r1             /* Add Inc2 * Mod2                  */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD3 - OLD_SIG3  */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + FM_INDEX
            *r8 = a0 = ieee(*r11)           /* Write Index to host              */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
```

```
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX:          float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
FIRST_HALF:     fltbits 0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAnalysis4.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis FFT data generator algorithm 4.
/*
/*  This algorithm gets loaded onto the DSP when the user opens the
/*  analysis window when using FM synthesis with the configuration set to "4."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*              -------                     -------
/*              |     |                     |     |
/*              |  1  | Oscillator 1        |  2  | Oscillator 2
/*              |     |                     |     |
/*              -------                     -------
/*                 |                           |
/*                 V                           V
/*              -------                     -------
/*              |     |                     |     |
/*              |  3  | Oscillator 3        |  4  |     Oscillator 4
/*              |     |                     |     |
/*              -------                     -------
/*                  |------------ + ---------|
/*                                |
/*                                O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "FM_Offsets.h"


.rsect  ".prog"
            r10e = r17                      /* Data Buffer                  */

            r7e = r17 + FM_FLAGS
            r2e = *r7                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e & HOST_READY_FLAG           /* test the host ready bit      */
            if (eq) pcgoto EndIt            /* host is not ready            */

            r9e = r17                       /* host parameters              */
            r8e = PARAMS
            r7e = 35                        /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1     /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2
loop2:      *r8++ = a0 = *r2 * *r9++        /* set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop


Next:       r8e = r17
            a0 = dsp(*r8)
            r2e = TEMP_INT                  /* NumSamples                   */
            nop
            *r2 = a1 = int24(a0)
            3*nop
            r2e = *r2                       /* r2 = NumSamples              */
            nop
            r2e = r2 - 128                  /* subtract the frame size      */
            r2 - 0                          /* need to output more samples? */
            if (ge) pcgoto Next2

            r3e = INDEX                     /* don't need samples           */
            r2e = ZERO                      /* zero out our VAR parameters  */
            r7e = 7                         /* there are 9 of them          */
here:       if (r7-- >= 0) pcgoto here      /* Re-initaillze the VAR parameters */
            *r3++ = a0 = *r2                /* Zero out the parameters      */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
```

```
            *r4 = r2e                       /* Turn on need to init flag       */

            pcgoto EndIt                    /* exit the module                 */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp   */
            nop
            a0 = float(*r4)                 /* convert to float                */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining        */

            r10e += FM_RETURN_DATA          /* r10 points to output buffer     */

            r4e = FIRST_HALF                /* see if we're doing 1st half     */
            r2e = *r4
            nop
            r2e & SECOND_HALF
            if (eq) pcgoto Next3            /* doing 1st half                  */
            nop

            r10e = r10 + 508                /* doing 2nd half, add offset      */
            r8e = r17 + FM_FLAGS
            r2e &= ~SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF               */

            r2e = *r8                       /* r2 contains flags from host     */
            nop                             /* latent instruction              */
            r2e &= ~HOST_READY_FLAG         /* clear the host_ready flag       */
            r2e |= DO_FFT_FLAG              /* set the do_fft flag             */
            *r8 = r2e                       /* write flags to memory           */

            pcgoto Next4
            nop

Next3:      r2e |= SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF               */

Next4:      r11e = INDEX                    /* Read index from host            */
            r2e = r17 + FM_INDEX
            *r11 = a0 = dsp(*r2)

            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                     */
            a2 = a1 - *r6                   /* Subtract 0.5 for round          */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                      */
            3*nop
            r2e = *r2                       /* r2 = Start                      */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)    */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1          */

            r3e = r2 + FM_ENV2              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2          */

            r3e = r2 + FM_ENV3              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3          */

            r3e = r2 + FM_ENV4              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
```

```
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4          */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag              */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes          */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 126                       /* Loop count (frameSize-2)        */
            r9e = SINE_SIZE

Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1    */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC    */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                     */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation        */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1     */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1          */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */

less1:      r5e = PARAMS + FM_MININDEX2
            r1e = PARAMS + FM_MAXINDEX2
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX2 - MIN_INDEX2    */
            r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX2 * (MAX-MIN)) + MIN */
            *r4 = a0 = *r4 + *r6

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
```

```
        r4e = *r4
        r1e = MOD2
        r4 = r4*2                       /* Multiply by 4 for float          */
        r4 = r4*2
        r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
        *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE2] * INDEX2      */

        r3e = INC2
        a1 = *r2 + *r3                  /* Phase2 = Phase2 + Inc2           */
        r4e = MOD1
        r5e = INC1
        *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + (Mod1*Inc1)    */
        2*nop

        r3e = TEMP_INT
        *r3 = a0 = int24(*r2)           /* r3 = PHASE2                      */
        3*nop
        r3e = *r3
        nop
        r3 - 256                        /* See if we're past end of table   */
        if (lt) pcgoto less2
        nop
        *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less2:  r6e = INDEX3_INC
        r4e = INDEX3
        a1 = *r4                        /* a1 = INDEX3                      */
        *r4 = a0 = *r4 + *r6            /* INDEX3 = INDEX3 + INDEX4_INC     */

        r2e = PHASE3
        r4e = TEMP_INT
        *r4 = a2 = int24 (*r2)
        3*nop
        r4e = *r4
        r1e = MOD3
        r4 = r4*2                       /* Multiply by 4 for float          */
        r4 = r4*2
        r5e = r4 + SINE_TABLE           /* r5 points to sine table          */
        *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

        r3e = INC3
        a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3           */
        r4e = MOD1
        r5e = INC1
        *r2 = a0 = a1 + *r4 * *r5       /* Phase3 + Inc3 + (mod1 * inc1)    */
        2*nop

        r3e = TEMP_INT
        *r3 = a0 = int24(*r2)
        3*nop
        r3e = *r3                       /* r3 = PHASE3                      */
        nop
        r3 - 256                        /* See if we're past end of table   */
        if (lt) pcgoto less3
        nop
        *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
        pcgoto ok3
        nop

less3:  r3 - 0
        if (gt) pcgoto ok3
        nop
        *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok3:    r6e = INDEX4_INC
        r4e = INDEX4
        a1 = *r4                        /* a1 = INDEX4                      */
        *r4 = a0 = *r4 + *r6            /* INDEX4 = INDEX4 + INDEX4_INC     */

        r2e = PHASE4
        r1e = TEMP_INT
        *r1 = a2 = int(*r2)
        3*nop
        r1e = *r1
        nop
        r1 = r1*2
        r1 = r1*2                       /* Multiply by 4 for float          */
        r5e = r1 + SINE_TABLE
```

```
            r6e = MOD3

            a0 = *r6 + a1 * *r5             /* Mod3 + (Envelope * Sine[PHASE4] */
            r1e = PT_FIVE
            nop
            a0 = a0 * *r1                   /* Divide by 2                     */
            2*nop

            a0 = a1 * *r5                   /* a0 = Index4 * sine_table[r5]    */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer        */

            r3e = INC4
            r5e = INC2
            r6e = MOD2
            a1 = *r2 + *r3                  /* PHASE4 + INC4                   */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD2*INC2)     */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD3 - OLD_SIG3 */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case            */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index             */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame          */
            nop

            r8e = r17 + FM_INDEX
            *r8 = a0 = ieee(*r11)           /* Write Index to host             */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
```

```
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
FIRST_HALF:     fltbits 0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAnalysis5.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis FFT data generator algorithm 5.
/*
/*  This algorithm gets loaded onto the DSP when the user opens the
/*  analysis window when using FM synthesis with the configuration set to "5."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*                          -------
/*                          |     |
/*                          |  1  | Osc 1
/*                          |     |
/*                          -------
/*                             |
/*                ------------------------------
/*                V             V              V
/*           -------         -------        -------
/*           |     |         |     |        |     |
/*           |  2  | Osc 2   |  3  | Osc 3  |  4  | Osc 4
/*           |     |         |     |        |     |
/*           -------         -------        -------
/*              |---------------|---------------|
/*                              +
/*                              O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "FM_Offsets.h"


.rsect  ".prog"
            r10e = r17                      /* Data Buffer                  */

            r7e = r17 + FM_FLAGS
            r2e = *r7                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e & HOST_READY_FLAG           /* test the host ready bit      */
            if (eq) pcgoto EndIt            /* host is not ready            */

            r9e = r17                       /* host parameters              */
            r8e = PARAMS
            r7e = 35                        /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1     /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2
loop2:      *r8++ = a0 = *r2 * *r9++        /* set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop


Next:       r8e = r17
            a0 = dsp(*r8)
            r2e = TEMP_INT                  /* NumSamples                   */
            nop
            *r2 = a1 = int24(a0)
            3*nop
            r2e = *r2                       /* r2 = NumSamples              */
            nop
            r2e = r2 - 128                  /* subtract the frame size      */
            r2 - 0                          /* need to output more samples? */
            if (ge) pcgoto Next2

            r3e = INDEX                     /* don't need samples           */
            r2e = ZERO                      /* zero out our VAR parameters  */
            r7e = 7                         /* there are 9 of them          */
here:       if (r7-- >= 0) pcgoto here      /* Re-initaillize the VAR parameters */
            *r3++ = a0 = *r2                /* Zero out the parameters      */
            r4e = NEED_TO_INIT
```

```
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag       */

            pcgoto EndIt                    /* exit the module                 */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp   */
            nop
            a0 = float(*r4)                 /* convert to float                */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining        */

            r10e += FM_RETURN_DATA          /* r10 points to output buffer     */

            r4e = FIRST_HALF                /* see if we're doing 1st half     */
            r2e = *r4
            nop
            r2e & SECOND_HALF
            if (eq) pcgoto Next3            /* doing 1st half                  */
            nop

            r10e = r10 + 508                /* doing 2nd half, add offset      */
            r8e = r17 + FM_FLAGS
            r2e &= ~SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF               */

            r2e = *r8                       /* r2 contains flags from host     */
            nop                             /* latent instruction              */
            r2e &= ~HOST_READY_FLAG         /* clear the host_ready flag       */
            r2e |= DO_FFT_FLAG              /* set the do_fft flag             */
            *r8 = r2e                       /* write flags to memory           */

            pcgoto Next4
            nop

Next3:      r2e |= SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF               */

Next4:      r11e = INDEX                    /* Read index from host            */
            r2e = r17 + FM_INDEX
            *r11 = a0 = dsp(*r2)

            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                     */
            a2 = a1 - *r6                   /* Subtract 0.5 for round          */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                      */
            3*nop
            r2e = *r2                       /* r2 = Start                      */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)    */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1          */

            r3e = r2 + FM_ENV2              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2          */

            r3e = r2 + FM_ENV3              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3          */

            r3e = r2 + FM_ENV4              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
```

```
            a0 = *r4 - *r3                  /* a0 = Env(finish) - Env(start)    */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4           */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag               */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes           */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 126                       /* Loop count (frameSize-2)         */
            r9e = SINE_SIZE


Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1     */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN  */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC     */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                      */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1      */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1           */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less1:      r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r4                        /* a1 = INDEX2                      */
            *r4 = a0 = *r4 + *r6            /* INDEX2 = INDEX2 + INDEX2_INC     */

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                      */
```

```
            r1e = MOD2
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* r5 is index into sine table      */
            r3e = PT_FIVE
            *r1 = a0 = a1 * *r5             /* SINE_TABLE[PHASE2] * INDEX2      */

            r3e = INC2
            a1 = *r2 + *r3                  /* a1 = Phase2 + Inc2               */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase2 = Phase2 + Inc2 + (MOD1*INC1) */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE2                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less2:      r3 - 0
            if (gt) pcgoto ok2
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok2:        r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r4                        /* a1 = INDEX3                      */
            *r4 = a0 = *r4 + *r6            /* INDEX3 = INDEX3 + INDEX3_INC     */

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2
            r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3                  /* Phase3 = Phase3 + Inc3           */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* Phase3 + Inc3 + (mod1 * inc1)    */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE3                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok3
            nop

less3:      r3 - 0
            if (gt) pcgoto ok3
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok3:        r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                        /* a1 = INDEX4                      */
            *r4 = a0 = *r4 + *r6            /* INDEX4 = INDEX4 + INDEX4_INC     */

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
```

```
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                       /* Multiply by 4 for float          */
            r5e = r1 + SINE_TABLE
            r6e = MOD3
            a0 = *r6 + a1 * *r5             /* Mod3 + Carrier                   */
            r3e = MOD2
            nop
            a0 = a0 + *r3                   /* Mod3 + Carrier + Mod2            */
            r1e = PT_THREE
            nop
            a0 = a0 * *r1                   /* Divide by 3                      */
            2*nop

            a0 = a1 * *r5                   /* a0 = Index4 * sine_table[r5]     */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer         */

            r3e = INC4
            a1 = *r2 + *r3                  /* PHASE4 + INC4                    */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5       /* PHASE4 = PHASE4 + (MOD1*INC1)    */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + FM_INDEX
            *r8 = a0 = ieee(*r11)           /* Write Index to host              */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
PT_THREE:       float   0.3933333
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
```

```
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
FIRST_HALF:     fltbits 0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAnalysis6.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis FFT data generator algorithm 6.
/*
/*  This algorithm gets loaded onto the DSP when the user opens the
/*  analysis window when using FM synthesis with the configuration set to "6."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*                                 -------
/*                                |       |
/*                                |   1   | Osc 1
/*                                |       |
/*                                -------
/*                                   |
/*                                ----------------
/*                               V                V
/*          -------          -------          -------
/*          |     |          |     |          |     |
/*          |  2  | Osc 2    |  3  | Osc 3    |  4  | Osc 4
/*          |     |          |     |          |     |
/*          -------          -------          -------
/*             |---------------|----------------|
/*                             +
/*                             O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "FM_Offsets.h"


.rsect  ".prog"
            r10e = r17                      /* Data Buffer                  */

            r7e = r17 + FM_FLAGS
            r2e = *r7                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e & HOST_READY_FLAG           /* test the host ready bit      */
            if (eq) pcgoto EndIt            /* host is not ready            */

            r9e = r17                       /* host parameters              */
            r8e = PARAMS
            r7e = 35                        /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1     /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2
loop2:      *r8++ = a0 = *r2 * *r9++        /* set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop


Next:       r8e = r17
            a0 = dsp(*r8)
            r2e = TEMP_INT                  /* NumSamples                   */
            nop
            *r2 = a1 = int24(a0)
            3*nop
            r2e = *r2                       /* r2 = NumSamples              */
            nop
            r2e = r2 - 128                  /* subtract the frame size      */
            r2 - 0                          /* need to output more samples? */
            if (ge) pcgoto Next2

            r3e = INDEX                     /* don't need samples           */
            r2e = ZERO                      /* zero out our VAR parameters  */
            r7e = 7                         /* there are 9 of them          */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters */
            *r3++ = a0 = *r2                /* Zero out the parameters      */
            r4e = NEED_TO_INIT
```

```
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag       */

            pcgoto EndIt                    /* exit the module                 */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp   */
            nop
            a0 = float(*r4)                 /* convert to float                */
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining        */

            r10e += FM_RETURN_DATA          /* r10 points to output buffer     */

            r4e = FIRST_HALF                /* see if we're doing 1st half     */
            r2e = *r4
            nop
            r2e & SECOND_HALF
            if (eq) pcgoto Next3            /* doing 1st half                  */
            nop

            r10e = r10 + 508                /* doing 2nd half, add offset      */
            r8e = r17 + FM_FLAGS
            r2e &= ~SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF               */

            r2e = *r8                       /* r2 contains flags from host     */
            nop                             /* latent instruction              */
            r2e &= ~HOST_READY_FLAG         /* clear the host_ready flag       */
            r2e |= DO_FFT_FLAG              /* set the do_fft flag             */
            *r8 = r2e                       /* write flags to memory           */

            pcgoto Next4
            nop

Next3:      r2e |= SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF               */

Next4:      r11e = INDEX                    /* Read index from host            */
            r2e = r17 + FM_INDEX
            *r11 = a0 = dsp(*r2)

            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                     */
            a2 = a1 - *r6                   /* Subtract 0.5 for round          */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                      */
            3*nop
            r2e = *r2                       /* r2 = Start                      */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)    */
            r2e = PARAMS + r2
            r3e = r2 + FM_ENV1              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1          */

            r3e = r2 + FM_ENV2              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 2          */

            r3e = r2 + FM_ENV3              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
            r5e = INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 3          */

            r3e = r2 + FM_ENV4              /* Env[start]                      */
            r4e = r3 + 4                    /* Env[finish]                     */
```

```
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4           */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag               */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes           */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 126                       /* Loop count (frameSize-2)         */
            r9e = SINE_SIZE


Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1     */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN  */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC     */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                      */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float          */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1      */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1           */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less1:      r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r4                        /* a1 = INDEX2                      */
            *r4 = a0 = *r4 + *r6            /* INDEX2 = INDEX2 + INDEX2_INC     */

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                      */
```

```
            r1e = MOD2
            r4 = r4*2                   /* Multiply by 4 for float          */
            r4 = r4*2

            r5e = r4 + SINE_TABLE       /* r5 is index into sine table      */
            r3e = PT_FIVE
            *r1 = a0 = a1 * *r5         /* SINE_TABLE[PHASE2] * INDEX2      */

            r3e = INC2
            *r2 = a1 = *r2 + *r3        /* Phase2 = Phase2 + Inc2           */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)       /* r3 = PHASE2                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                    /* See if we're past end of table   */
            if (lt) pcgoto less2
            nop
            *r2 = a0 = *r2 - *r9        /* Wrap to start of table           */

less2:      r6e = INDEX3_INC
            r4e = INDEX3
            a1 = *r4                    /* a1 = INDEX3                      */
            *r4 = a0 = *r4 + *r6        /* INDEX3 = INDEX3 + INDEX3_INC     */

            r2e = PHASE3
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4
            r1e = MOD3
            r4 = r4*2                   /* Multiply by 4 for float          */
            r4 = r4*2
            r5e = r4 + SINE_TABLE       /* Need to do interpolation         */
            *r1 = a0 = *r5 * a1         /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

            r3e = INC3
            a1 = *r2 + *r3              /* Phase3 = Phase3 + Inc3           */
            r4e = MOD1
            r5e = INC1
            *r2 = a0 = a1 + *r4 * *r5   /* Phase3 + Inc3 + (mod1 * inc1)    */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)       /* r3 = PHASE3                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                    /* See if we're past end of table   */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9        /* Wrap to start of table           */
            pcgoto ok3
            nop

less3:      r3 - 0
            if (gt) pcgoto ok3
            nop
            *r2 = a0 = *r2 + *r9        /* Handle negative case             */

ok3:        r6e = INDEX4_INC
            r4e = INDEX4
            a1 = *r4                    /* a1 = INDEX4                      */
            *r4 = a0 = *r4 + *r6        /* INDEX4 = INDEX4 + INDEX4_INC     */

            r2e = PHASE4
            r1e = TEMP_INT
            *r1 = a2 = int(*r2)
            3*nop
            r1e = *r1
            nop
            r1 = r1*2
            r1 = r1*2                   /* Multiply by 4 for float          */
            r5e = r1 + SINE_TABLE
            r6e = MOD3
            a0 = *r6 + a1 * *r5         /* Mod3 + Carrier                   */
```

```
            r1e = PT_THREE
            r3e = MOD2
            a0 = a0 + *r3                   /* Mod3 + Carrier + Mod2          */
            2*nop
            a0 = a0 * *r1                   /* Divide by 3                    */
            2*nop

            a0 = a1 * *r5                   /* a0 = Index4 * sine_table[r5]   */
            2*nop
            *r10++ = a0 = ieee(a0)          /* Place into sample buffer       */

            r3e = INC4
            r5e = INC1
            r6e = MOD1
            a1 = *r2 + *r3                  /* PHASE4 + INC4                  */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD1*INC1)    */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD1 - OLD_SIG3 */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                    */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table         */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case           */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index            */
            if (r7-- >= 0) pcgoto MainLoop  /* Loop for size of frame         */
            nop

            r8e = r17 + FM_INDEX
            *r8 = a0 = ieee(*r11)           /* Write Index to host            */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_THREE:       float   0.3933333
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
```

```
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
FIRST_HALF:     fltbits 0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# FMAnalysis7.s

```
/*-----------------------------------------------------------------------*/
/*  FM Synthesis FFT data generator algorithm 7.
/*
/*  This algorithm gets loaded onto the DSP when the user opens the
/*  analysis window when using FM synthesis with the configuration set to "7."
/*
/*  It has 4 oscillators in this arrangement:
/*
/*                                        -------
/*                                        |     |
/*                                        |  1  | Osc 1
/*                                        |     |
/*                                        -------
/*                                           |
/*                                           V
/*          -------         -------         -------
/*          |     |         |     |         |     |
/*          |  2  | Osc 2   |  3  | Osc 3   |  4  | Osc 4
/*          |     |         |     |         |     |
/*          -------         -------         -------
/*             |---------------|---------------|
/*                             +
/*                             O    <---- Output
/*
/* COPYRIGHT
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "FM_Offsets.h"


.rsect  ".prog"
            r10e = r17                      /* Data Buffer                  */

            r7e = r17 + FM_FLAGS
            r2e = *r7                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e & HOST_READY_FLAG           /* test the host ready bit      */
            if (eq) pcgoto EndIt            /* host is not ready            */

            r9e = r17                       /* host parameters              */
            r8e = PARAMS
            r7e = 35                        /* number of params - 2         */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1     /* copy over all params         */
            nop

            r9e = PARAMS + FM_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 2
loop2:      *r8++ = a0 = *r2 * *r9++        /* set up INC1 - INC4           */
            if (r6-- >= 0) pcgoto loop2
            nop


Next:       r8e = r17
            a0 = dsp(*r8)
            r2e = TEMP_INT                  /* NumSamples                   */
            nop
            *r2 = a1 = int24(a0)
            3*nop
            r2e = *r2                       /* r2 = NumSamples              */
            nop
            r2e = r2 - 128                  /* subtract the frame size      */
            r2 = 0                          /* need to output more samples? */
            if (ge) pcgoto Next2

            r3e = INDEX                     /* don't need samples           */
            r2e = ZERO                      /* zero out our VAR parameters  */
            r7e = 7                         /* there are 8 of them          */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters */
            *r3++ = a0 = *r2                /* Zero out the parameters      */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
```

```
        *r4 = r2e                       /* Turn on need to init flag       */

        pcgoto EndIt                    /* exit the module                 */

Next2:  r4e = TEMP_INT
        *r4 = r2                        /* put samples remaining in temp   */
        nop
        a0 = float(*r4)                 /* convert to float                */
        2*nop
        *r8 = a1 = ieee(a0)             /* Update samples remaining        */

        r10e += FM_RETURN_DATA          /* r10 points to output buffer     */

        r4e = FIRST_HALF                /* see if we're doing 1st half     */
        r2e = *r4
        nop
        r2e & SECOND_HALF
        if (eq) pcgoto Next3            /* doing 1st half                  */
        nop

        r10e = r10 + 508                /* doing 2nd half, add offset      */
        r8e = r17 + FM_FLAGS
        r2e &= ~SECOND_HALF
        *r4 = r2e                       /* Toggle FIRST_HALF               */

        r2e = *r8                       /* r2 contains flags from host     */
        nop                             /* latent instruction              */
        r2e &= ~HOST_READY_FLAG         /* clear the host_ready flag       */
        r2e |= DO_FFT_FLAG              /* set the do_fft flag             */
        *r8 = r2e                       /* write flags to memory           */

        pcgoto Next4
        nop

Next3:  r2e |= SECOND_HALF
        *r4 = r2e                       /* Toggle FIRST_HALF               */

Next4:  r11e = INDEX                    /* Read index from host            */
        r2e = r17 + FM_INDEX
        *r11 = a0 = dsp(*r2)

        r7e = COMPARE
        r6e = PT_FIVE
        a1 = *r11 * *r7                 /* I * COMPARE                     */
        a2 = a1 - *r6                   /* Subtract 0.5 for round          */
        r2e = TEMP_INT
        nop
        *r2 = a2 = int(a2)              /* a2 = Start                      */
        3*nop
        r2e = *r2                       /* r2 = Start                      */
        nop
        r2 = r2*2
        r2 = r2*2                       /* r2 = r2 * 4 (for float size)    */
        r2e = PARAMS + r2
        r3e = r2 + FM_ENV1              /* Env[start]                      */
        r4e = r3 + 4                    /* Env[finish]                     */
        a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
        r5e = INDEX1_INC
        nop
        *r5 = a1 = a0 * *r7             /* Save Index increment 1          */

        r3e = r2 + FM_ENV2              /* Env[start]                      */
        r4e = r3 + 4                    /* Env[finish]                     */
        a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
        r5e = INDEX2_INC
        nop
        *r5 = a1 = a0 * *r7             /* Save Index increment 2          */

        r3e = r2 + FM_ENV3              /* Env[start]                      */
        r4e = r3 + 4                    /* Env[finish]                     */
        a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
        r5e = INDEX3_INC
        nop
        *r5 = a1 = a0 * *r7             /* Save Index increment 3          */

        r3e = r2 + FM_ENV4              /* Env[start]                      */
        r4e = r3 + 4                    /* Env[finish]                     */
        a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
```

```
            r5e = INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4          */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag              */

            r3e = PARAMS + FM_ENV1          /* Set up initial indexes          */
            r4e = INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV2
            r4e = INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV3
            r4e = INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + FM_ENV4
            r4e = INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 126                       /* Loop count (frameSize-2)        */
            r9e = SINE_SIZE


Mainloop:   r5e = PARAMS + FM_MININDEX1
            r1e = PARAMS + FM_MAXINDEX1
            a3 = *r1 - *r5                  /* a3 = MAX_INDEX1 - MIN_INDEX1    */
            r6e = INDEX1_INC
            r4e = INDEX1
            a1 = *r5 + a3 * *r4             /* a1 = (INDEX1 * (MAX-MIN)) + MIN */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC    */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                     */
            r1e = MOD1
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* Need to do interpolation        */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1     */

            r3e = INC1
            *r2 = a1 = *r2 + *r3            /* Phase1 = Phase1 + Inc1          */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                     */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table  */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table          */

less1:      r6e = INDEX2_INC
            r4e = INDEX2
            a1 = *r4                        /* a1 = INDEX2                     */
            *r4 = a0 = *r4 + *r6            /* INDEX2 = INDEX2 + INDEX2_INC    */

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE2                     */
            r1e = MOD2
```

```
        r4 = r4*2                       /* Multiply by 4 for float          */
        r4 = r4*2

        r5e = r4 + SINE_TABLE           /* r5 is index into sine table      */
        r3e = PT_FIVE
        *r1 = a0 = a1 * *r5             /* SINE_TABLE[PHASE2] * INDEX2      */

        r3e = INC2
        *r2 = a1 = *r2 + *r3            /* Phase2 = Phase2 + Inc2           */
        2*nop

        r3e = TEMP_INT
        *r3 = a0 = int24(*r2)           /* r3 = PHASE2                      */
        3*nop
        r3e = *r3
        nop
        r3 - 256                        /* See if we're past end of table   */
        if (lt) pcgoto less2
        nop
        *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less2:  r6e = INDEX3_INC
        r4e = INDEX3
        a1 = *r4                        /* a1 = INDEX3                      */
        *r4 = a0 = *r4 + *r6            /* INDEX3 = INDEX3 + INDEX3_INC     */

        r2e = PHASE3
        r4e = TEMP_INT
        *r4 = a2 = int24 (*r2)
        3*nop
        r4e = *r4
        r1e = MOD3
        r4 = r4*2                       /* Multiply by 4 for float          */
        r4 = r4*2
        r5e = r4 + SINE_TABLE           /* r5 is index into sine table      */
        *r1 = a0 = *r5 * a1             /* mod3 = SINE_TABLE[PHASE3] * INDEX3 */

        r3e = INC3
        *r2 = a1 = *r2 + *r3            /* Phase3 = Phase3 + Inc3           */
        2*nop

        r3e = TEMP_INT
        *r3 = a0 = int24(*r2)           /* r3 = PHASE3                      */
        3*nop
        r3e = *r3
        nop
        r3 - 256                        /* See if we're past end of table   */
        if (lt) pcgoto less3
        nop
        *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */

less3:  r6e = INDEX4_INC
        r4e = INDEX4
        a1 = *r4                        /* a1 = INDEX4                      */
        *r4 = a0 = *r4 + *r6            /* INDEX4 = INDEX4 + INDEX4_INC     */

        r2e = PHASE4
        r1e = TEMP_INT
        *r1 = a2 = int(*r2)
        3*nop
        r1e = *r1
        nop
        r1 = r1*2
        r1 = r1*2                       /* Multiply by 4 for float          */
        r5e = r1 + SINE_TABLE
        r6e = MOD3
        a0 = *r6 + a1 * *r5             /* Mod3 + Carrier                   */
        r1e = PT_THREE
        r3e = MOD2
        a0 = a0 + *r3                   /* Mod3 + Carrier + Mod2            */
        2*nop
        a0 = a0 * *r1                   /* Divide by 3                      */
        2*nop

        a0 = a1 * *r5                   /* a0 = Index4 * sine_table[r5]     */
        2*nop
        *r10++ = a0 = ieee(a0)          /* Place into sample buffer         */
```

```
            r3e = INC4
            r5e = INC1
            r6e = MOD1
            a1 = *r2 + *r3                  /* PHASE4 + INC4                    */
            r4e = OLD_SIG3
            nop
            a2 = a1 + *r6 * *r5             /* PHASE4 + INC4 + (MOD1*INC1)      */
            2*nop
            *r2 = a3 = a2 - *r4             /* PHASE4 + INC4 + MOD1 - OLD_SIG3  */

            2*nop
            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE4                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table           */
            pcgoto ok4
            nop

less4:      r3 - 0
            if (gt) pcgoto ok4
            nop
            *r2 = a0 = *r2 + *r9            /* Handle negative case             */

ok4:        r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + FM_INDEX
            *r8 = a0 = ieee(*r11)           /* Write Index to host              */

EndIt:      return (r18)
            nop


FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
MOD1:           float   0.0
MOD2:           float   0.0
MOD3:           float   0.0
ONE:            float   1.0
PT_THREE:       float   0.3933333
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         37*float
OLD_SIG1:       float   0.0
OLD_SIG2:       float   0.0
OLD_SIG3:       float   0.0
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INDEX1_INC:     float
INDEX2_INC:     float
INDEX3_INC:     float
INDEX4_INC:     float

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
PHASE1:         float   0.0
PHASE2:         float   0.0
PHASE3:         float   0.0
PHASE4:         float   0.0
INDEX1:         float   0.0
INDEX2:         float   0.0
INDEX3:         float   0.0
INDEX4:         float   0.0
```

```
FIRST_HALF:     fltbits 0
NEED_TO_INIT:   fltbits 0x00000001

/*---------------------- This is the sine wave table ------------ */
.rsect ".tab"
SINE_TABLE: float   0.000000, 0.024541, 0.049067, 0.073563, 0.098016
            float   0.122409, 0.146728, 0.170959, 0.195087, 0.219098
            float   0.242976, 0.266708, 0.290280, 0.313677, 0.336884
            float   0.359889, 0.382677, 0.405235, 0.427548, 0.449604
            float   0.471389, 0.492891, 0.514095, 0.534990, 0.555562
            float   0.575800, 0.595691, 0.615223, 0.634384, 0.653164
            float   0.671550, 0.689531, 0.707097, 0.724238, 0.740942
            float   0.757199, 0.773001, 0.788337, 0.803198, 0.817576
            float   0.831460, 0.844845, 0.857720, 0.870078, 0.881913
            float   0.893216, 0.903981, 0.914202, 0.923872, 0.932986
            float   0.941537, 0.949522, 0.956934, 0.963770, 0.970026
            float   0.975697, 0.980781, 0.985274, 0.989173, 0.992477
            float   0.995182, 0.997289, 0.998794, 0.999698, 1.000000
            float   0.999699, 0.998797, 0.997292, 0.995187, 0.992483
            float   0.989181, 0.985283, 0.980791, 0.975709, 0.970039
            float   0.963784, 0.956949, 0.949538, 0.941555, 0.933004
            float   0.923892, 0.914223, 0.904004, 0.893240, 0.881938
            float   0.870104, 0.857747, 0.844873, 0.831490, 0.817606
            float   0.803230, 0.788369, 0.773034, 0.757234, 0.740977
            float   0.724274, 0.707135, 0.689569, 0.671589, 0.653204
            float   0.634425, 0.615264, 0.595733, 0.575843, 0.555606
            float   0.535034, 0.514140, 0.492936, 0.471436, 0.449651
            float   0.427596, 0.405283, 0.382726, 0.359938, 0.336934
            float   0.313727, 0.290330, 0.266759, 0.243027, 0.219149
            float   0.195139, 0.171011, 0.146780, 0.122461, 0.098068
            float   0.073616, 0.049119, 0.024593, 0.000053, -0.024488
            float   -0.049014, -0.073511, -0.097963, -0.122356, -0.146676
            float   -0.170907, -0.195035, -0.219046, -0.242925, -0.266658
            float   -0.290230, -0.313627, -0.336835, -0.359840, -0.382629
            float   -0.405187, -0.427501, -0.449557, -0.471343, -0.492845
            float   -0.514050, -0.534945, -0.555518, -0.575757, -0.595648
            float   -0.615181, -0.634344, -0.653124, -0.671511, -0.689493
            float   -0.707060, -0.724201, -0.740906, -0.757165, -0.772968
            float   -0.788305, -0.803167, -0.817545, -0.831431, -0.844816
            float   -0.857693, -0.870052, -0.881888, -0.893192, -0.903959
            float   -0.914181, -0.923852, -0.932967, -0.941519, -0.949505
            float   -0.956919, -0.963756, -0.970013, -0.975686, -0.980771
            float   -0.985265, -0.989165, -0.992470, -0.995177, -0.997285
            float   -0.998792, -0.999697, -1.000000, -0.999701, -0.998799
            float   -0.997296, -0.995193, -0.992489, -0.989188, -0.985292
            float   -0.980801, -0.975720, -0.970051, -0.963798, -0.956965
            float   -0.949555, -0.941573, -0.933023, -0.923912, -0.914245
            float   -0.904026, -0.893263, -0.881962, -0.870130, -0.857774
            float   -0.844901, -0.831519, -0.817636, -0.803261, -0.788402
            float   -0.773068, -0.757268, -0.741012, -0.724310, -0.707172
            float   -0.689608, -0.671628, -0.653244, -0.634466, -0.615306
            float   -0.595775, -0.575886, -0.555650, -0.535079, -0.514185
            float   -0.492982, -0.471482, -0.449698, -0.427644, -0.405331
            float   -0.382775, -0.359988, -0.336984, -0.313777, -0.290381
            float   -0.266810, -0.243078, -0.219200, -0.195190, -0.171063
            float   -0.146832, -0.122513, -0.098120, -0.073668, -0.049172
            float   -0.024646, 0.0000000
```

# Additive.s

```
/*=============================================================================================================*/
/*      Additive_Offsets.h                                                                                     */
/*                                                                                                             */
/*                  Offsets for host/DSP shared parameters                                                     */
/*                                                                                                             */
/*      struct AddParams {                                                                                     */
/*          float   numSamples;                                                                                */
/*          float   minFreq1, maxFreq1, minFreq2, maxFreq2, minFreq3, maxFreq3, minFreq4, maxFreq4;            */
/*          float   ampEnv1[6], ampEnv2[6], ampEnv3[6], ampEnv3[6];                                           */
/*          float   freqEnv1[6], freqEnv2[6], freqEnv3[6], freqEnv4[6];                                       */
/*  57 floats                                                                                                  */
/*=============================================================================================================*/

#define ADD_SAMPLES             0       /* param[0] */
#define ADD_MIN_FREQ1           4       /* param[1] */
#define ADD_MAX_FREQ1           8       /* param[2] */
#define ADD_MIN_FREQ2           12      /* param[3] */
#define ADD_MAX_FREQ2           16      /* param[4] */
#define ADD_MIN_FREQ3           20      /* param[5] */
#define ADD_MAX_FREQ3           24      /* param[6] */
#define ADD_MIN_FREQ4           28      /* param[7] */
#define ADD_MAX_FREQ4           32      /* param[8] */
#define ADD_AMP_ENV1            36      /* param[9-14] */
#define ADD_AMP_ENV2            60      /* param[15-20] */
#define ADD_AMP_ENV3            84      /* param[21-26] */
#define ADD_AMP_ENV4            108     /* param[27-32] */
#define ADD_FREQ_ENV1           132     /* param[33-38] */
#define ADD_FREQ_ENV2           156     /* param[39-44] */
#define ADD_FREQ_ENV3           180     /* param[45-50] */
#define ADD_FREQ_ENV4           204     /* param[51-56] */

/*----------- These should be consistent with other synth methods --------------*/
#define ADD_INDEX               236     /* param[59] */
#define ADD_RETURN_DATA         260     /* param[65-320] (256 floats) */
#define ADD_FLAGS               1280

#define VIEW_FLAG               0x00000001
#define HOST_READY_FLAG         0x00000002
#define DO_FFT_FLAG             0x00000004
#define SECOND_HALF             0x00000001

#define INIT_FLAG               0x00000001
```

```
/*=============================================================================*/
/*
/* NAME
/*     Additive.s
/*
/* SYNOPSIS
/*      This DSP module implements 4 sine wave oscillator additive synthesis.
/*      It expects the host to pass a parameter block of this form:
/*
/*      struct AddParams {
/*          float   numSamples;
/*          float   minFreq1, maxFreq1, minFreq2, maxFreq2, minFreq3, maxFreq3;
/*          float   ampEnv1[6], freqEnv1[6], ampEnv2[6], freqEnv2[6];
/*          float   ampEnv3[6], freqEnv3[6], ampEnv4[6], freqEnv4[6];
/*          unsigned long   flags;
/*      };
/*
/*      It will use these parameters for the 4 oscillators giving them time-
/*      varying freqencies and amplitudes.
/*
/*      Author Jeff Boone, Oregon State University
/*
/*=============================================================================*/
#include <dspregs.h>
#include "Additive_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"

.rsect  ".prog"
            r10e = r16 + DH_HADDR + PHYSICAL    /* PRB or Sample buffer         */
            r10e = *r10
            r9e = r17                           /* host parameters              */

            r8e = PARAMS
            r7e = 55                            /*  number of params - 2        */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1         /*  copy over all params        */
            nop

            r9e = PARAMS + ADD_MIN_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 6

loop2:      *r8++ = a0 = *r2 * *r9++            /* set up INC1 - INC8           */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + ADD_FLAGS
            r2e = *r7                           /* r2 contains flags from host  */
            nop                                 /* latent instruction           */
            r2e & VIEW_FLAG                     /* test the view bit            */
            if (eq) pcgoto Next                 /* we want to hear it           */
            nop

            r10e = r17 + ADD_RETURN_DATA        /* point r10 to param block     */
            r2e & HOST_READY_FLAG               /* test the host_ready bit      */
            if (eq) pcgoto EndIt                /* host is not ready            */

Next:       r8e = r17
            a0 = dsp(*r8)                       /* NumSamples                   */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int24(a0)
            3*nop
            r2e = *r2                           /* r2 = NumSamples              */
            nop
            r2e = r2 - 240                      /* Subtract the frame size      */
            r2 - 0                              /* need to output more samples? */
            if (ge) pcgoto Next2

            r3e = INDEX                         /* don't need samples           */
            r2e = ZERO                          /* zero out our VAR parameters  */
            r7e = 11                            /* there are 13 of them         */
here:       if (r7-- >= 0) pcgoto here          /* Re-initailize the VAR parameters */
            *r3++ = a0 = *r2                    /* Zero out the parameters      */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
```

```
            *r4 = r2e                   /* Turn on need to init flag       */

            pcgoto EndIt                /* exit the module                 */

Next2:      r4e = TEMP_INT
            *r4 = r2                    /* put samples remaining in temp   */
            nop
            a0 = float(*r4)
            2*nop
            *r8 = a1 = ieee(a0)         /* Update samples remaining        */


            r11e = INDEX
            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7             /* I * COMPARE                     */
            a2 = a1 - *r6               /* Subtract 0.5 for round          */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)          /* a2 = Start                      */
            3*nop
            r2e = *r2                   /* r2 = Start                      */
            nop
            r2 = r2*2
            r2 = r2*2                   /* r2 = r2 * 4 (for float size)    */
            r2e = PARAMS + r2
            r3e = r2 + ADD_AMP_ENV1     /* Env[start]                      */
            r4e = r3 + 4                /* Env[finish]                     */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]   */
            r5e = AMP_INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 1          */

            r3e = r2 + ADD_FREQ_ENV1    /* Env[start]                      */
            r4e = r3 + 4                /* Env[finish]                     */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]   */
            r5e = FREQ_INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 1          */

            r3e = r2 + ADD_AMP_ENV2     /* Env[start]                      */
            r4e = r3 + 4                /* Env[finish]                     */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]   */
            r5e = AMP_INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 2          */

            r3e = r2 + ADD_FREQ_ENV2    /* Env[start]                      */
            r4e = r3 + 4                /* Env[finish]                     */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]   */
            r5e = FREQ_INDEX2_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 2          */

            r3e = r2 + ADD_AMP_ENV3     /* Env[start]                      */
            r4e = r3 + 4                /* Env[finish]                     */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]   */
            r5e = AMP_INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 3          */

            r3e = r2 + ADD_FREQ_ENV3    /* Env[start]                      */
            r4e = r3 + 4                /* Env[finish]                     */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]   */
            r5e = FREQ_INDEX3_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 3          */

            r3e = r2 + ADD_AMP_ENV4     /* Env[start]                      */
            r4e = r3 + 4                /* Env[finish]                     */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]   */
            r5e = AMP_INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7         /* Save Index increment 4          */

            r3e = r2 + ADD_FREQ_ENV4    /* Env[start]                      */
            r4e = r3 + 4                /* Env[finish]                     */
            a0 = *r4 - *r3              /* a0 = Env[finish] - Env[start]   */
```

```
            r5e = FREQ_INDEX4_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 4          */

            r8e = NEED_TO_INIT
            r2e = *r8
            nop
            r2e & INIT_FLAG
            if (eq) pcgoto NoInit

            r2e &= ~INIT_FLAG
            *r8 = r2e                       /* Turn off init flag              */

            r3e = PARAMS + ADD_AMP_ENV1     /* Set up initial indexes          */
            r4e = AMP_INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + ADD_AMP_ENV2
            r4e = AMP_INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + ADD_AMP_ENV3
            r4e = AMP_INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + ADD_AMP_ENV4
            r4e = AMP_INDEX4
            *r4 = a0 = *r3

            r3e = PARAMS + ADD_FREQ_ENV1
            r4e = FREQ_INDEX1
            *r4 = a0 = *r3

            r3e = PARAMS + ADD_FREQ_ENV2
            r4e = FREQ_INDEX2
            *r4 = a0 = *r3

            r3e = PARAMS + ADD_FREQ_ENV3
            r4e = FREQ_INDEX3
            *r4 = a0 = *r3

            r3e = PARAMS + ADD_FREQ_ENV4
            r4e = FREQ_INDEX4
            *r4 = a0 = *r3

NoInit:     r7e = 238                       /* Loop count (frameSize-2)        */
            r9e = SINE_SIZE


Mainloop:   r6e = AMP_INDEX1_INC
            r4e = AMP_INDEX1
            a1 = *r4                        /* a1 = INDEX1                     */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC    */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                     */
            r1e = OUTPUT
            r4 = r4*2                       /* Multiply by 4 for float         */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* r5 is index into sine table     */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1] * INDEX1     */

            r6e = FREQ_INDEX1_INC
            r4e = FREQ_INDEX1
            a1 = *r4                        /* a1 = Index1                     */
            *r4 = a0 = *r4 + *r6            /* Index1 = Index1 + Index1_Inc    */

            r2e = PHASE1
            r3e = INC1                      /* r3 points to Inc1(MinFreq)      */
            r1e = INC2
            a0 = *r1 - *r3                  /* a0 = Inc2 - Inc1                */
            3*nop
            a0 = *r3 + a1 * a2              /* a0 = Index1 * (Inc2-Inc1) + Inc1 */
            3*nop
```

```
          *r2 = a0 = *r2 + a0           /* Phase1 = Phase1 + Inc1 + (Index1 * (Inc2-Inc1)   */
          2*nop

          r3e = TEMP_INT
          *r3 = a0 = int24(*r2)         /* r3 = PHASE1                      */
          3*nop
          r3e = *r3
          nop
          r3 - 256                      /* See if we're past end of table   */
          if (lt) pcgoto less1
          nop
          *r2 = a0 = *r2 - *r9          /* Wrap to start of table           */

less1:    r6e = AMP_INDEX2_INC
          r4e = AMP_INDEX2
          a1 = *r4                      /* a1 = INDEX2                      */
          *r4 = a0 = *r4 + *r6          /* INDEX2 = INDEX2 + INDEX2_INC     */

          r2e = PHASE2
          r4e = TEMP_INT
          *r4 = a2 = int24 (*r2)
          3*nop
          r4e = *r4                     /* r4 = PHASE2                      */
          r1e = OUTPUT
          r4 = r4*2                     /* Multiply by 4 for float          */
          r4 = r4*2

          r5e = r4 + SINE_TABLE         /* r5 is index into sine table      */
          *r1 = a0 = *r1 + a1 * *r5     /* Sum into output                  */

          r6e = FREQ_INDEX2_INC
          r4e = FREQ_INDEX2
          a1 = *r4                      /* a1 = Index2                      */
          *r4 = a0 = *r4 + *r6          /* Index2 = Index2 + Index2_Inc     */

          r2e = PHASE2
          r3e = INC3                    /* r3 points to Inc3(MinFreq)       */
          r1e = INC4
          a0 = *r1 - *r3                /* a0 = Inc4(MaxFreq - Inc3         */
          3*nop
          a0 = *r3 + a1 * a0            /* a0 = Index2 * (Inc4-Inc3) + Inc3 */
          3*nop
          *r2 = a0 = *r2 + a0           /* Phase2 = Phase2 + Inc3 + (Index2 * (Inc4-Inc3)   */
          2*nop

          r3e = TEMP_INT
          *r3 = a0 = int24(*r2)         /* r3 = PHASE2                      */
          3*nop
          r3e = *r3
          nop
          r3 - 256                      /* See if we're past end of table   */
          if (lt) pcgoto less2
          nop
          *r2 = a0 = *r2 - *r9          /* Wrap to start of table           */

less2:    r6e = AMP_INDEX3_INC
          r4e = AMP_INDEX3
          a1 = *r4                      /* a1 = INDEX3                      */
          *r4 = a0 = *r4 + *r6          /* INDEX3 = INDEX3 + INDEX3_INC     */

          r2e = PHASE3
          r4e = TEMP_INT
          *r4 = a2 = int24 (*r2)
          3*nop
          r4e = *r4                     /* r4 = PHASE3                      */
          r1e = OUTPUT
          r4 = r4*2                     /* Multiply by 4 for float          */
          r4 = r4*2

          r5e = r4 + SINE_TABLE         /* r5 is index into sine table      */
          *r1 = a0 = *r1 + a1 * *r5     /* Sum into output                  */

          r6e = FREQ_INDEX3_INC
          r4e = FREQ_INDEX3
          a1 = *r4                      /* a1 = Index3                      */
          *r4 = a0 = *r4 + *r6          /* Index3 = Index3 + Index3_Inc     */

          r2e = PHASE3
```

```
            r3e = INC5                  /* r3 points to Inc5(MinFreq)       */
            r1e = INC6
            a0 = *r1 - *r3              /* a0 = Inc6(MaxFreq - Inc5         */
            3*nop
            a0 = *r3 + a1 * a0          /* a0 = Index3 * (Inc6-Inc5) + Inc5 */
            3*nop
            *r2 = a0 = *r2 + a0         /* Phase3 = Phase3 + Inc5 + (Index3 * (Inc6-Inc5)   */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)       /* r3 = PHASE3                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                    /* See if we're past end of table   */
            if (lt) pcgoto less3
            nop
            *r2 = a0 = *r2 - *r9        /* Wrap to start of table           */

less3:      r6e = AMP_INDEX4_INC
            r4e = AMP_INDEX4
            a1 = *r4                    /* a1 = INDEX4                      */
            *r4 = a0 = *r4 + *r6        /* INDEX4 = INDEX4 + INDEX4_INC     */

            r2e = PHASE4
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                   /* r4 = PHASE4                      */
            r1e = OUTPUT
            r4 = r4*2                   /* Multiply by 4 for float          */
            r4 = r4*2

            r5e = r4 + SINE_TABLE       /* r5 is index into sine table      */
            r1e = OUTPUT

            r8e = r17 + ADD_FLAGS       /* r8 points to host flags          */
            r3e = *r8                   /* r2 contains flags from host      */
            nop                         /* latent instruction               */
            r3e & VIEW_FLAG             /* test the view bit                */
            if (eq) pcgoto Sound        /* we want to hear it               */
            nop

            a0 = *r1 + a1 * *r5         /* Sum into output                  */
            2*nop
            *r10++ = a0 = ieee(a0)      /* Place into sample buffer         */
            pcgoto Cont
            nop

Sound:      a0 = *r1 + a1 * *r5         /* Sum into output                  */
            3*nop
            *r10++ = a0 = *r10 + a0     /* Divide by 4                      */

Cont:       r6e = FREQ_INDEX4_INC
            r4e = FREQ_INDEX4
            a1 = *r4                    /* a1 = Index4                      */
            *r4 = a0 = *r4 + *r6        /* Index4 = Index4 + Index4_Inc     */

            r2e = PHASE4
            r3e = INC7                  /* r3 points to Inc7(MinFreq)       */
            r1e = INC8
            a0 = *r1 - *r3              /* a0 = Inc8(MaxFreq) - Inc7        */
            3*nop
            a0 = *r3 + a1 * a0          /* a0 = Index4 * (Inc8-Inc7) + Inc7 */
            3*nop
            *r2 = a0 = *r2 + a0         /* Phase4 = Phase4 + Inc7 + (Index4 * (Inc8-Inc7)   */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)       /* r3 = PHASE4                      */
            3*nop
            r3e = *r3
            nop
            r3 - 256                    /* See if we're past end of table   */
            if (lt) pcgoto less4
            nop
            *r2 = a0 = *r2 - *r9        /* Wrap to start of table           */
```

```
less4:      r2e = ONE
            *r11 = a1 = *r11 + *r2          /* Increment the index              */
            if (r7-- >= 0) pcgoto Mainloop  /* Loop for size of frame           */
            nop

            r8e = r17 + ADD_FLAGS           /* r8 points to host flags          */
            r2e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag        */
            *r8 = r2e                       /* Write flags to memory            */

EndIt:      return (r18)
            nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
MAX_AMP:        float   (1.0)
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
OUTPUT:         float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
ZERO:           float   0.0

PARAMS:         57*float
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INC5:           float
INC6:           float
INC7:           float
INC8:           float
AMP_INDEX1_INC:         float
AMP_INDEX2_INC:         float
AMP_INDEX3_INC:         float
AMP_INDEX4_INC:         float
FREQ_INDEX1_INC:        float
FREQ_INDEX2_INC:        float
FREQ_INDEX3_INC:        float
FREQ_INDEX4_INC:        float

.rsect ".var"
INDEX:                  float   0.0
PHASE1:                 float   0.0
PHASE2:                 float   0.0
PHASE3:                 float   0.0
PHASE4:                 float   0.0
AMP_INDEX1:             float   0.0
AMP_INDEX2:             float   0.0
AMP_INDEX3:             float   0.0
AMP_INDEX4:             float   0.0
FREQ_INDEX1:            float   0.0
FREQ_INDEX2:            float   0.0
FREQ_INDEX3:            float   0.0
FREQ_INDEX4:            float   0.0
NEED_TO_INIT:           fltbits 0x00000001

.rsect ".tab"
SINE_TABLE: float   0.00000000, 0.00613525, 0.01226675, 0.01839075, 0.024504
            float   0.03060225, 0.036682, 0.04273975, 0.04877175, 0.0547745
            float   0.06074400, 0.066677, 0.07257, 0.07841925, 0.084221
            float   0.08997225, 0.09566925, 0.10130875, 0.106887, 0.112401
            float   0.11784725, 0.12322275, 0.12852375, 0.1337475, 0.1388905
            float   0.14395000, 0.14892275, 0.15380575, 0.158596, 0.163291
            float   0.16788750, 0.17238275, 0.17677425, 0.1810595, 0.1852355
            float   0.18929975, 0.19325025, 0.19708425, 0.2007995, 0.204394
            float   0.20786500, 0.21121125, 0.21443, 0.2175195, 0.22047825
            float   0.22330400, 0.22599525, 0.2285505, 0.230968, 0.2332465
            float   0.23538425, 0.2373805, 0.2392335, 0.2409425, 0.2425065
            float   0.24392425, 0.24519525, 0.2463185, 0.24729325, 0.24811925
            float   0.24879550, 0.24932225, 0.2496985, 0.2499245, 0.25
            float   0.24992475, 0.24969925, 0.249323, 0.24879675, 0.24812075
            float   0.24729525, 0.24632075, 0.24519775, 0.24392725, 0.24250975
            float   0.24094600, 0.23923725, 0.2373845, 0.23538875, 0.233251
            float   0.23097300, 0.22855575, 0.226001, 0.22331, 0.2204845
```

```
float   0.217526, 0.21443675, 0.21121825, 0.2078725, 0.2044015
float   0.2008075, 0.19709225, 0.1932585, 0.1893085, 0.18524425
float   0.1810685, 0.17678375, 0.17239225, 0.16789725, 0.163301
float   0.15860625, 0.153816, 0.14893325, 0.14396075, 0.1389015
float   0.1337585, 0.128535, 0.123234, 0.117859, 0.11241275
float   0.106899, 0.10132075, 0.0956815, 0.0899845, 0.0842335
float   0.07843175, 0.0725825, 0.06668975, 0.06075675, 0.05478725
float   0.04878475, 0.04275275, 0.036695, 0.03061525, 0.024517
float   0.018404, 0.01227975, 0.00614825, 0.00001325, -0.006122
float   -0.0122535, -0.0183778, -0.0244908, -0.030589, -0.036669
float   -0.0427268, -0.0487588, -0.0547615, -0.0607313, -0.0666645
float   -0.0725575, -0.0784068, -0.0842088, -0.08996, -0.0956573
float   -0.1012968, -0.1068753, -0.1123893, -0.1178358, -0.1232113
float   -0.1285125, -0.1337363, -0.1388795, -0.1439393, -0.148912
float   -0.1537953, -0.158586, -0.163281, -0.1678778, -0.1723733
float   -0.176765, -0.1810503, -0.1852265, -0.1892913, -0.193242
float   -0.1970763, -0.2007918, -0.2043863, -0.2078578, -0.211204
float   -0.2144233, -0.217513, -0.220472, -0.223298, -0.2259898
float   -0.2285453, -0.230963, -0.2332418, -0.2353798, -0.2373763
float   -0.2392298, -0.240939, -0.2425033, -0.2439215, -0.2451928
float   -0.2463163, -0.2472913, -0.2481175, -0.2487943, -0.2493213
float   -0.249698, -0.2499243, -0.25, -0.2499253, -0.2496998
float   -0.249324, -0.2487983, -0.2481223, -0.247297, -0.246323
float   -0.2452003, -0.24393, -0.2425128, -0.2409495, -0.2392413
float   -0.2373888, -0.2353933, -0.2332558, -0.230978, -0.2285613
float   -0.2260065, -0.2233158, -0.2204905, -0.2175325, -0.2144435
float   -0.2112253, -0.2078798, -0.204409, -0.2008153, -0.1971005
float   -0.193267, -0.189317, -0.185253, -0.1810775, -0.176793
float   -0.172402, -0.167907, -0.163311, -0.1586165, -0.1538265
float   -0.1489438, -0.1439715, -0.1389125, -0.1337698, -0.1285463
float   -0.1232455, -0.1178705, -0.1124245, -0.106911, -0.1013328
float   -0.0956938, -0.089997, -0.084246, -0.0784443, -0.0725953
float   -0.0667025, -0.0607695, -0.0548, -0.0487975, -0.0427658
float   -0.036708, -0.0306283, -0.02453, -0.018417, -0.012293
float   -0.0061615, 0.000000
```

# AdditiveAnalysis.s

```
/*------------------------------------------------------------------------*/
/*
/* NAME
/*     AdditiveAnalysis.s
/*
/* SYNOPSIS
/*      This DSP module uses 4 sine wave oscillators to generate sound
/*      data for the FFT analysis. It expects the host to pass a parameter
/*      block of this form:
/*
/*      struct AddParams {
/*          float   numSamples;
/*          float   minFreq1, maxFreq1, minFreq2, maxFreq2, minFreq3, maxFreq3;
/*          float   ampEnv1[6], freqEnv1[6], ampEnv2[6], freqEnv2[6];
/*          float   ampEnv3[6], freqEnv3[6], ampEnv4[6], freqEnv4[6];
/*          unsigned long   flags;
/*      };
/*
/*      It will use these parameters for the 4 oscillators giving them time-
/*      varying freqencies and amplitudes.
/*
/*      Author Jeff Boone, Oregon State University
/*
/*------------------------------------------------------------------------*/
#include <dspregs.h>
#include "Additive_Offsets.h"
#include "dspTaskDispatcherOffsets.a.h"

.rsect  ".prog"
            r10e = r17
            r9e = r17                       /* host parameters              */

            r7e = r17 + ADD_FLAGS
            r2e = *r7                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e & HOST_READY_FLAG           /* test the host ready bit      */
            if (eq) pcgoto EndIt            /* host is not ready            */

            r8e = PARAMS
            r7e = 55                        /*  number of params - 2        */

loop1:      *r8++ = a2 = dsp (*r9++)
            if (r7-- >= 0) pcgoto loop1     /*  copy over all params        */
            nop

            r9e = PARAMS + ADD_MIN_FREQ1
            r8e = INC1
            r2e = FREQ_MULT
            r6e = 6

loop2:      *r8++ = a0 = *r2 * *r9++        /* set up INC1 - INC8           */
            if (r6-- >= 0) pcgoto loop2
            nop

            r7e = r17 + ADD_FLAGS
            r2e = *r7                       /* r2 contains flags from host  */
            nop                             /* latent instruction           */
            r2e & VIEW_FLAG                 /* test the view bit            */
            if (eq) pcgoto Next             /* we want to hear it           */
            nop

            r10e = r17 + ADD_RETURN_DATA    /* point r10 to param block     */
            r2e & HOST_READY_FLAG           /* test the host_ready bit      */
            if (eq) pcgoto EndIt            /* host is not ready            */

Next:       r8e = r17
            a0 = dsp(*r8)                   /* NumSamples                   */
            r2e = TEMP_INT
            nop
            *r2 = a1 = int24(a0)
            3*nop
            r2e = *r2                       /* r2 = NumSamples              */
            nop
            r2e = r2 - 128                  /* Subtract the frame size      */
            r2 - 0                          /* need to output more samples? */
            if (ge) pcgoto Next2

            r3e = INDEX                     /* don't need samples           */
```

```
            r2e = ZERO                      /* zero out our VAR parameters      */
            r7e = 11                        /* there are 13 of them             */
here:       if (r7-- >= 0) pcgoto here      /* Re-initailize the VAR parameters */
            *r3++ = a0 = *r2                /* Zero out the parameters          */
            r4e = NEED_TO_INIT
            r2e = INIT_FLAG
            *r4 = r2e                       /* Turn on need to init flag        */

            pcgoto EndIt                    /* exit the module                  */

Next2:      r4e = TEMP_INT
            *r4 = r2                        /* put samples remaining in temp    */
            nop
            a0 = float(*r4)
            2*nop
            *r8 = a1 = ieee(a0)             /* Update samples remaining         */

            r10e += ADD_RETURN_DATA

            r4e = FIRST_HALF                /* see if we're doing 1st half      */
            r2e = *r4
            nop
            r2e & SECOND_HALF
            if (eq) pcgoto Next3            /* doing 1st half                   */
            nop

            r10e = r10 + 508                /* doing 2nd half, add offset       */
            r8e = r17 + ADD_FLAGS
            r2e &= ~SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF                */

            r2e = *r8                       /* r2 contains flags from host      */
            nop                             /* latent instruction               */
            r2e &= ~HOST_READY_FLAG         /* clear the host_ready flag        */
            r2e |= DO_FFT_FLAG              /* set the do_fft flag              */
            *r8 = r2e                       /* write flags to memory            */

            pcgoto Next4
            nop

Next3:      r2e |= SECOND_HALF
            *r4 = r2e                       /* Toggle FIRST_HALF                */

Next4:      r11e = INDEX                    /* Read index from host             */
            r2e = r17 + ADD_INDEX
            *r11 = a0 = dsp(*r2)
            r7e = COMPARE
            r6e = PT_FIVE
            a1 = *r11 * *r7                 /* I * COMPARE                      */
            a2 = a1 - *r6                   /* Subtract 0.5 for round           */
            r2e = TEMP_INT
            nop
            *r2 = a2 = int(a2)              /* a2 = Start                       */
            3*nop
            r2e = *r2                       /* r2 = Start                       */
            nop
            r2 = r2*2
            r2 = r2*2                       /* r2 = r2 * 4 (for float size)     */
            r2e = PARAMS + r2
            r3e = r2 + ADD_AMP_ENV1         /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = AMP_INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1           */

            r3e = r2 + ADD_FREQ_ENV1        /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = FREQ_INDEX1_INC
            nop
            *r5 = a1 = a0 * *r7             /* Save Index increment 1           */

            r3e = r2 + ADD_AMP_ENV2         /* Env[start]                       */
            r4e = r3 + 4                    /* Env[finish]                      */
            a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]    */
            r5e = AMP_INDEX2_INC
            nop
```

```
*r5 = a1 = a0 * *r7             /* Save Index increment 2          */

r3e = r2 + ADD_FREQ_ENV2        /* Env[start]                      */
r4e = r3 + 4                    /* Env[finish]                     */
a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
r5e = FREQ_INDEX2_INC
nop
*r5 = a1 = a0 * *r7             /* Save Index increment 2          */

r3e = r2 + ADD_AMP_ENV3         /* Env[start]                      */
r4e = r3 + 4                    /* Env[finish]                     */
a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
r5e = AMP_INDEX3_INC
nop
*r5 = a1 = a0 * *r7             /* Save Index increment 3          */

r3e = r2 + ADD_FREQ_ENV3        /* Env[start]                      */
r4e = r3 + 4                    /* Env[finish]                     */
a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
r5e = FREQ_INDEX3_INC
nop
*r5 = a1 = a0 * *r7             /* Save Index increment 3          */

r3e = r2 + ADD_AMP_ENV4         /* Env[start]                      */
r4e = r3 + 4                    /* Env[finish]                     */
a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
r5e = AMP_INDEX4_INC
nop
*r5 = a1 = a0 * *r7             /* Save Index increment 4          */

r3e = r2 + ADD_FREQ_ENV4        /* Env[start]                      */
r4e = r3 + 4                    /* Env[finish]                     */
a0 = *r4 - *r3                  /* a0 = Env[finish] - Env[start]   */
r5e = FREQ_INDEX4_INC
nop
*r5 = a1 = a0 * *r7             /* Save Index increment 4          */

r8e = NEED_TO_INIT
r2e = *r8
nop
r2e & INIT_FLAG
if (eq) pcgoto NoInit

r2e &= ~INIT_FLAG
*r8 = r2e                       /* Turn off init flag              */

r3e = PARAMS + ADD_AMP_ENV1     /* Set up initial indexes          */
r4e = AMP_INDEX1
*r4 = a0 = *r3

r3e = PARAMS + ADD_AMP_ENV2
r4e = AMP_INDEX2
*r4 = a0 = *r3

r3e = PARAMS + ADD_AMP_ENV3
r4e = AMP_INDEX3
*r4 = a0 = *r3

r3e = PARAMS + ADD_AMP_ENV4
r4e = AMP_INDEX4
*r4 = a0 = *r3

r3e = PARAMS + ADD_FREQ_ENV1
r4e = FREQ_INDEX1
*r4 = a0 = *r3

r3e = PARAMS + ADD_FREQ_ENV2
r4e = FREQ_INDEX2
*r4 = a0 = *r3

r3e = PARAMS + ADD_FREQ_ENV3
r4e = FREQ_INDEX3
*r4 = a0 = *r3

r3e = PARAMS + ADD_FREQ_ENV4
r4e = FREQ_INDEX4
*r4 = a0 = *r3
```

```
NoInit:     r7e = 126                       /* Loop count (frameSize-2)          */
            r9e = SINE_SIZE


Mainloop:   r6e = AMP_INDEX1_INC
            r4e = AMP_INDEX1
            a1 = *r4                        /* a1 = INDEX1                       */
            *r4 = a0 = *r4 + *r6            /* INDEX1 = INDEX1 + INDEX1_INC      */

            r2e = PHASE1
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE1                       */
            r1e = OUTPUT
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* r5 is index into sine table       */
            *r1 = a0 = *r5 * a1             /* SINE_TABLE[PHASE1) * INDEX1       */

            r6e = FREQ_INDEX1_INC
            r4e = FREQ_INDEX1
            a1 = *r4                        /* a1 = Index1                       */
            *r4 = a0 = *r4 + *r6            /* Index1 = Index1 + Index1_Inc      */

            r2e = PHASE1
            r3e = INC1                      /* r3 points to Inc1(MinFreq)        */
            r1e = INC2
            a0 = *r1 - *r3                  /* a0 = Inc2 - Inc1                  */
            3*nop
            a0 = *r3 + a1 * a0              /* a0 = Index1 * (Inc2-Inc1) + Inc1  */
            3*nop
            *r2 = a0 = *r2 + a0             /* Phase1 = Phase1 + Inc1 + (Index1 * (Inc2-Inc1)   */
            2*nop

            r3e = TEMP_INT
            *r3 = a0 = int24(*r2)           /* r3 = PHASE1                       */
            3*nop
            r3e = *r3
            nop
            r3 - 256                        /* See if we're past end of table    */
            if (lt) pcgoto less1
            nop
            *r2 = a0 = *r2 - *r9            /* Wrap to start of table            */

less1:      r6e = AMP_INDEX2_INC
            r4e = AMP_INDEX2
            a1 = *r4                        /* a1 = INDEX2                       */
            *r4 = a0 = *r4 + *r6            /* INDEX2 = INDEX2 + INDEX2_INC      */

            r2e = PHASE2
            r4e = TEMP_INT
            *r4 = a2 = int24 (*r2)
            3*nop
            r4e = *r4                       /* r4 = PHASE2                       */
            r1e = OUTPUT
            r4 = r4*2                       /* Multiply by 4 for float           */
            r4 = r4*2

            r5e = r4 + SINE_TABLE           /* r5 is index into sine table       */
            *r1 = a0 = *r1 + a1 * *r5       /* Sum into output                   */

            r6e = FREQ_INDEX2_INC
            r4e = FREQ_INDEX2
            a1 = *r4                        /* a1 = Index2                       */
            *r4 = a0 = *r4 + *r6            /* Index2 = Index2 + Index2_Inc      */

            r2e = PHASE2
            r3e = INC3                      /* r3 points to Inc3(MinFreq)        */
            r1e = INC4
            a0 = *r1 - *r3                  /* a0 = Inc4(MaxFreq - Inc3          */
            3*nop
            a0 = *r3 + a1 * a0              /* a0 = Index2 * (Inc4-Inc3) + Inc3  */
            3*nop
            *r2 = a0 = *r2 + a0             /* Phase2 = Phase2 + Inc3 + (Index2 * (Inc4-Inc3)   */
            2*nop
```

```
          r3e = TEMP_INT
          *r3 = a0 = int24(*r2)          /* r3 = PHASE2                        */
          3*nop
          r3e = *r3
          nop
          r3 - 256                       /* See if we're past end of table     */
          if (lt) pcgoto less2
          nop
          *r2 = a0 = *r2 - *r9           /* Wrap to start of table             */

less2:    r6e = AMP_INDEX3_INC
          r4e = AMP_INDEX3
          a1 = *r4                       /* a1 = INDEX3                        */
          *r4 = a0 = *r4 + *r6           /* INDEX3 = INDEX3 + INDEX3_INC       */

          r2e = PHASE3
          r4e = TEMP_INT
          *r4 = a2 = int24 (*r2)
          3*nop
          r4e = *r4                      /* r4 = PHASE3                        */
          r1e = OUTPUT
          r4 = r4*2                      /* Multiply by 4 for float            */
          r4 = r4*2

          r5e = r4 + SINE_TABLE          /* r5 is index into sine table        */
          *r1 = a0 = *r1 + a1 * *r5      /* Sum into output                    */

          r6e = FREQ_INDEX3_INC
          r4e = FREQ_INDEX3
          a1 = *r4                       /* a1 = Index3                        */
          *r4 = a0 = *r4 + *r6           /* Index3 = Index3 + Index3_Inc       */

          r2e = PHASE3
          r3e = INC5                     /* r3 points to Inc5(MinFreq)         */
          r1e = INC6
          a0 = *r1 - *r3                 /* a0 = Inc6(MaxFreq - Inc5           */
          3*nop
          a0 = *r3 + a1 * a0             /* a0 = Index3 * (Inc6-Inc5) + Inc5   */
          3*nop
          *r2 = a0 = *r2 + a0            /* Phase3 = Phase3 + Inc5 + (Index3 * (Inc6-Inc5)   */
          2*nop

          r3e = TEMP_INT
          *r3 = a0 = int24(*r2)          /* r3 = PHASE3                        */
          3*nop
          r3e = *r3
          nop
          r3 - 256                       /* See if we're past end of table     */
          if (lt) pcgoto less3
          nop
          *r2 = a0 = *r2 - *r9           /* Wrap to start of table             */

less3:    r6e = AMP_INDEX4_INC
          r4e = AMP_INDEX4
          a1 = *r4                       /* a1 = INDEX4                        */
          *r4 = a0 = *r4 + *r6           /* INDEX4 = INDEX4 + INDEX4_INC       */

          r2e = PHASE4
          r4e = TEMP_INT
          *r4 = a2 = int24 (*r2)
          3*nop
          r4e = *r4                      /* r4 = PHASE4                        */
          r1e = OUTPUT
          r4 = r4*2                      /* Multiply by 4 for float            */
          r4 = r4*2

          r5e = r4 + SINE_TABLE          /* r5 is index into sine table        */
          r1e = OUTPUT

          r4e = PT_TWO_FIVE

          a0 = *r1 + a1 * *r5            /* Sum into output                    */
/*        3*nop
          a0 = a0 * *r4                  /* Divide by 4                        */
          2*nop
          *r10++ = a0 = leee(a0)         /* Place into sample buffer           */

          r6e = FREQ_INDEX4_INC
```

```
          r4e = FREQ_INDEX4
          a1 = *r4                          /* a1 = Index4                          */
          *r4 = a0 = *r4 + *r6              /* Index4 = Index4 + Index4_Inc         */

          r2e = PHASE4
          r3e = INC7                        /* r3 points to Inc7(MinFreq)           */
          r1e = INC8
          a0 = *r1 - *r3                    /* a0 = Inc8(MaxFreq) - Inc7            */
          3*nop
          a0 = *r3 + a1 * a0                /* a0 = Index4 * (Inc8-Inc7) + Inc7 */
          3*nop
          *r2 = a0 = *r2 + a0               /* Phase4 = Phase4 + Inc7 + (Index4 * (Inc8-Inc7)   */
          2*nop

          r3e = TEMP_INT
          *r3 = a0 = int24(*r2)             /* r3 = PHASE4                          */
          3*nop
          r3e = *r3
          nop
          r3 - 256                          /* See if we're past end of table       */
          if (lt) pcgoto less4
          nop
          *r2 = a0 = *r2 - *r9              /* Wrap to start of table               */

less4:    r2e = ONE
          *r11 = a1 = *r11 + *r2            /* Increment the index                  */
          if (r7-- >= 0) pcgoto Mainloop    /* Loop for size of frame               */
          nop

          r8e = r17 + ADD_INDEX
          *r8 = a0 = ieee(*r11)             /* Write Index to host                  */

EndIt:    return (r18)
          nop

FREQ_MULT:      float   (256.0/24000.0)     /*  Sine table size/ Sample rate */
COMPARE:        float   (1.0/(24960.0/5.0))
MAX_AMP:        float   (1.0)
TEMP_INT:       int24
SINE_SIZE:      float   (256.0)
NUMBER_SAMP:    float   24960.0
A_FIFTH:        float   (1.0/5.0)
OUTPUT:         float   0.0
ONE:            float   1.0
PT_FIVE:        float   0.5
PT_TWO_FIVE:    float   0.25
ZERO:           float   0.0

PARAMS:         57*float
INC1:           float
INC2:           float
INC3:           float
INC4:           float
INC5:           float
INC6:           float
INC7:           float
INC8:           float
AMP_INDEX1_INC:         float
AMP_INDEX2_INC:         float
AMP_INDEX3_INC:         float
AMP_INDEX4_INC:         float
FREQ_INDEX1_INC:        float
FREQ_INDEX2_INC:        float
FREQ_INDEX3_INC:        float
FREQ_INDEX4_INC:        float

.rsect ".var"
INDEX:                  float   0.0
PHASE1:                 float   0.0
PHASE2:                 float   0.0
PHASE3:                 float   0.0
PHASE4:                 float   0.0
AMP_INDEX1:             float   0.0
AMP_INDEX2:             float   0.0
AMP_INDEX3:             float   0.0
AMP_INDEX4:             float   0.0
FREQ_INDEX1:            float   0.0
FREQ_INDEX2:            float   0.0
```

```
FREQ_INDEX3:            float   0.0
FREQ_INDEX4:            float   0.0
FIRST_HALF:             fltbits 0
NEED_TO_INIT:           fltbits 0x00000001

.rsect ".tab"
SINE_TABLE: float   0.00000000, 0.00613525, 0.01226675, 0.01839075, 0.024504
            float   0.03060225, 0.036682, 0.04273975, 0.04877175, 0.0547745
            float   0.06074400, 0.066677, 0.07257, 0.07841925, 0.084221
            float   0.08997225, 0.09566925, 0.10130875, 0.106887, 0.112401
            float   0.11784725, 0.12322275, 0.12852375, 0.1337475, 0.1388905
            float   0.14395000, 0.14892275, 0.15380575, 0.158596, 0.163291
            float   0.16788750, 0.17238275, 0.17677425, 0.1810595, 0.1852355
            float   0.18929975, 0.19325025, 0.19708425, 0.2007995, 0.204394
            float   0.20786500, 0.21121125, 0.21443, 0.2175195, 0.22047825
            float   0.22330400, 0.22599525, 0.2285505, 0.230968, 0.2332465
            float   0.23538425, 0.2373805, 0.2392335, 0.2409425, 0.2425065
            float   0.24392425, 0.24519525, 0.2463185, 0.24729325, 0.24811925
            float   0.24879550, 0.24932225, 0.2496985, 0.2499245, 0.25
            float   0.24992475, 0.24969925, 0.249323, 0.24879675, 0.24812075
            float   0.24729525, 0.24632075, 0.24519775, 0.24392725, 0.24250975
            float   0.24094600, 0.23923725, 0.2373845, 0.23538875, 0.233251
            float   0.23097300, 0.22855575, 0.226001, 0.22331, 0.2204845
            float   0.217526, 0.21443675, 0.21121825, 0.2078725, 0.2044015
            float   0.2008075, 0.19709225, 0.1932585, 0.1893085, 0.18524425
            float   0.1810685, 0.17678375, 0.17239225, 0.16789725, 0.163301
            float   0.15860625, 0.153816, 0.14893325, 0.14396075, 0.1389015
            float   0.1337585, 0.128535, 0.123234, 0.117859, 0.11241275
            float   0.106899, 0.10132075, 0.0956815, 0.0899845, 0.0842335
            float   0.07843175, 0.0725825, 0.06668975, 0.06075675, 0.05478725
            float   0.04878475, 0.04275275, 0.036695, 0.03061525, 0.024517
            float   0.018404, 0.01227975, 0.00614825, 0.00001325, -0.006122
            float   -0.0122535, -0.0183778, -0.0244908, -0.030589, -0.036669
            float   -0.0427268, -0.0487588, -0.0547615, -0.0607313, -0.0666645
            float   -0.0725575, -0.0784068, -0.0842088, -0.08996, -0.0956573
            float   -0.1012968, -0.1068753, -0.1123893, -0.1178358, -0.1232113
            float   -0.1285125, -0.1337363, -0.1388795, -0.1439393, -0.148912
            float   -0.1537953, -0.158586, -0.163281, -0.1678778, -0.1723733
            float   -0.176765, -0.1810503, -0.1852265, -0.1892913, -0.193242
            float   -0.1970763, -0.2007918, -0.2043863, -0.2078578, -0.211204
            float   -0.2144233, -0.217513, -0.220472, -0.223298, -0.2259898
            float   -0.2285453, -0.230963, -0.2332418, -0.2353798, -0.2373763
            float   -0.2392298, -0.240939, -0.2425033, -0.2439215, -0.2451928
            float   -0.2463163, -0.2472913, -0.2481175, -0.2487943, -0.2493213
            float   -0.249698, -0.2499243, -0.25, -0.2499253, -0.2496998
            float   -0.249324, -0.2487983, -0.2481223, -0.247297, -0.246323
            float   -0.2452003, -0.24393, -0.2425128, -0.2409495, -0.2392413
            float   -0.2373888, -0.2353933, -0.2332558, -0.230978, -0.2285613
            float   -0.2260065, -0.2233158, -0.2204905, -0.2175325, -0.2144435
            float   -0.2112253, -0.2078798, -0.204409, -0.2008153, -0.1971005
            float   -0.193267, -0.189317, -0.185253, -0.1810775, -0.176793
            float   -0.172402, -0.167907, -0.163311, -0.1586165, -0.1538265
            float   -0.1489438, -0.1439715, -0.1389125, -0.1337698, -0.1285463
            float   -0.1232455, -0.1178705, -0.1124245, -0.106911, -0.1013328
            float   -0.0956938, -0.089997, -0.084246, -0.0784443, -0.0725953
            float   -0.0667025, -0.0607695, -0.0548, -0.0487975, -0.0427658
            float   -0.036708, -0.0306283, -0.02453, -0.018417, -0.012293
            float   -0.0061615, 0.000000
```

# Karplus.s

```
/*-----------------------------------------------------------------------------*/
/*
/* NAME
/*      Karplus.s
/*
/* SYNOPSIS
/*      This is the DSP module that performs the Karplus/Strong plucked
/*      string synthesis. It expects the host to pass it a parameter block
/*      of this form:
/*
/*      struct KSParams {
/*          float   numSamples;
/*          float   tableSize;
/*          float   randomSeed;
/*          unsigned long flags;
/*      };
/*
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "Karplus_Offsets.h"

.rsect ".prog"

        r10e = r16 + DH_HADDR + PHYSICAL    /* -- PRB Address --        */
        r10e = *r10

        r8e = r17
        a0 = dsp(*r8)                   /* NumSamples                   */
        r3e = TEMP_INT
        nop
        *r3 = a1 = int24(a0)            /* convert to an integer        */
        3*nop
        r3e = *r3                       /* r2 = NumSamples              */
        nop

        r3e - 0                         /* See if we need more samples  */
        if (lt) pcgoto end              /* Need more samples            */

More:   r7e = r17 + KAR_FLAGS
        r2e = *r7                       /* r2 contains flags from host  */
        nop                             /* latent instruction           */
        r2e & VIEW_FLAG                 /* test the view bit            */
        if (eq) pcgoto Next             /* we want to hear it           */
        nop

        r10e = r17 + KAR_RETURN_DATA    /* point r10 to param block     */
        r2e & HOST_READY_FLAG           /* test the host_ready bit      */
        if (eq) pcgoto end              /* host is not ready            */

Next:   r4e = TEMP_INT
        r3e = r3 - 240                  /* Subtract the frame size      */
        *r4 = r3e                       /* put samples remaining in temp*/
        nop
        a0 = float(*r4)                 /* convert to float             */
        2*nop
        *r8 = a1 = ieee(a0)             /* Update samples remaining     */

        r2e & NEED_TO_INIT              /* test the need to init bit    */
        if (eq) pcgoto noInit           /* host is not ready            */

        r2e &= ~NEED_TO_INIT            /* Turn off Need_To_Init flag   */
        *r7 = r2e                       /* Write flag back to memory    */

    *-----------------------------------------------------------------------*/
    *  This section loads the initial random values into the wave table for  */
    *  the Karplus/Strong synthesis.                                         */
    *-----------------------------------------------------------------------*/

        r11e = r17 + KAR_SIZE           /* TableSize addr               */
        r6e = TEMP_INT
        a0 = dsp(*r11)                  /* Size of wave table           */
        *r6 = a1 = int24(a0)
        nop
        r5e = r17 + KAR_SEED            /* Seed value                   */
        r1e = TEMP_FLOAT
```

```
        r6e = *r6                       /* r6 = tableSize                  */
        r2e = waveTable                 /* Pointer to output               */
        r6e = r6 - 2
        *r1 = a0 = dsp(*r5)
        r7e = MINUS_ONE
        r8e = TWO

ran:    r3e = ranmul
        nop
ranA:   a0 = *r1 * *r3++                /* ranmul * seed                   */
        a0 = a0 + *r3++
        r4e = ranmod
        nop
        a1 = *r3 + a0 * *r4++
        a1 = int(a1)
        a1 = float(a1)
        2*nop

        *r1 = a0 = a0 - a1 * *r4--      /* Update the seed value           */
        3*nop
        a0 = a0 * *r4
        2*nop
        *r2++ = a1 = *r7 + a0 * *r8

stop:   nop
        if (r6-- >= 0 ) pcgoto ran      /* Loop for numSamples             */
        nop

  /*-----------------------------------------------------------------*/
  /*  Now use the table to create the output values.                 */
  /*-----------------------------------------------------------------*/


noInit: r7e = OFFSET
        r7e = *r7

        r11e = r17 + KAR_SIZE           /* TableSize addr                  */
        r9e = TEMP_INT
        a0 = dsp(*r11)                  /* Size of wave table              */
        *r9 = a1 = int24(a0)            /* r9 points to wave table size    */

        r6e = 240-2                     /* r6 is the counter(frameSize-2)  */
        r2e = r7 + waveTable            /* Pointer to wave table           */
        r4e = POINT5                    /* Constant 0.5                    */
        r9e = *r9                       /* r9 = tableSize                  */

        r7e = GAIN
        r9e = r9 * 2
        r9e = r9 * 2                    /* Multiply by 4 for float size    */
        r5e = r9 + waveTable            /* Address of end of table         */

loop:   r3e = r2 - 4                    /* Let r3 trail r2 by one sample   */
        r3 - waveTable                  /* See if we've fallen off front   */
        if (ge) pcgoto skip1
        nop
        r3e += r9                       /* r3 = r3 + length of table       */

skip1:  r2 - r5                         /* See if we reached end of table  */
        if (lt) pcgoto skip2
        nop
        r2e = waveTable                 /* Start at front again            */

skip2:  a1 = *r2 + *r3                  /* Add two samples                 */
        3*nop

        *r2++ = a2 = a1 * *r4           /* Average                         */

        r8e = r17 + KAR_FLAGS           /* r8 points to host flags         */
        r3e = *r8                       /* r2 contains flags from host     */
        nop                             /* latent instruction              */
        r3e & VIEW_FLAG                 /* test the view bit               */
        if (eq) pcgoto Sound            /* we want to hear it              */
        nop

        a0 = a2 * *r7
        2*nop
        *r10++ = a0 = ieee(a0)          /* Place into sample buffer        */
        pcgoto Cont
```

```
        nop

Sound:  *r10++ = a0 = *r10 + a2 * *r7   /* Output result                    */

Cont:   if (r6-- >= 0) pcgoto loop
        nop                             /* Latent instruction               */
        r7e = OFFSET                    /* Store the last position          */
        r2e = r2 - waveTable
        *r7 = r2

        r8e = r17 + KAR_FLAGS           /* r8 points to host flags          */
        r2e = *r8                       /* r2 contains flags from host      */
        nop                             /* latent instruction               */
        r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag        */
        *r8 = r2e                       /* Write flags to memory            */

end:    return (r18)
        nop

ranmul:         float   25173., 13849., -0.5
ranmod:         float   (1.0/32768.0), 32768.0      /* 2^(-15), 2^15 */
_seed:          float   2378.0
TEMP_FLOAT:     float
.align 4
TEMP_INT:       int
.align 4
POINT5:         float   0.500000
GAIN:           float   (1.7)
MINUS_ONE:      float   (-1.0)
TWO:            float   2.0
ZERO:           float   0.0

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
OFFSET:         int24   0
.align 4
waveTable:      512 * float
```

# KarAnalysis.s

```
/*-----------------------------------------------------------------------------*/
/*
/* NAME
/*      Karplus.s
/*
/* SYNOPSIS
/*      This is the DSP module that performs the Karplus/Strong plucked
/*      string synthesis. It expects the host to pass it a parameter block
/*      of this form:
/*
/*      struct KSParams {
/*          float   numSamples;
/*          float   tableSize;
/*          float   randomSeed;
/*          unsigned long flags;
/*      };
/*
/*      Author Jeff Boone, Oregon State University
/*
/*-----------------------------------------------------------------------------*/
#include <dspregs.h>
#include "dspTaskDispatcherOffsets.a.h"
#include "Karplus_Offsets.h"

.rsect ".prog"
        r10e = r17                      /* Data buffer                  */

Around: r7e = r17 + KAR_FLAGS
        r2e = *r7                       /* r2 contains flags from host  */
        nop                             /* latent instruction           */
        r2e & HOST_READY_FLAG           /* test the host ready bit      */
        if (eq) pcgoto end              /* host is not ready            */
        nop

        a0 = dsp(*r10)                  /* NumSamples                   */
        r3e = TEMP_INT
        nop
        *r3 = a1 = int24(a0)            /* convert to an integer        */
        3*nop
        r3e = *r3                       /* r3 = NumSamples              */
        nop

        r3e - 0                         /* See if we need more samples  */
        if (lt) pcgoto end              /* Need more samples            */

Next:   r4e = TEMP_INT
        r3e = r3 - 256                  /* Subtract the frame size      */
        *r4 = r3e                       /* put samples remaining in temp*/
        nop
        a0 = float(*r4)                 /* convert to float             */
        2*nop
        *r10 = a1 = ieee(a0)            /* Update samples remaining     */

        r2e & NEED_TO_INIT              /* test the need to init bit    */
        if (eq) pcgoto noInit           /* host is not ready            */

        r2e &= ~NEED_TO_INIT            /* Turn off Need_To_Init flag   */
        *r7 = r2e                       /* Write flag back to memory    */

  /*-------------------------------------------------------------------------*/
  /*  This section loads the initial random values into the wave table for  */
  /*  the Karplus/Strong synthesis.                                          */
  /*-------------------------------------------------------------------------*/

        r11e = r17 + KAR_SIZE           /* TableSize addr               */
        r6e = TEMP_INT
        a0 = dsp(*r11)                  /* Size of wave table           */
        *r6 = a1 = int24(a0)
        nop
        r5e = r17 + KAR_SEED            /* Seed value                   */
        r1e = TEMP_FLOAT
        r6e = 256                       /* r6 = tableSize               */
        r2e = waveTable                 /* Pointer to output            */
        r6e = r6 - 2
        *r1 = a0 = dsp(*r5)
        r7e = MINUS_ONE
        r8e = TWO
```

```
ran:    r3e = ranmul
        nop
ranA:   a0 = *r1 * *r3++                /* ranmul * seed                */
        a0 = a0 + *r3++
        r4e = ranmod
        nop
        a1 = *r3 + a0 * *r4++
        a1 = int(a1)
        a1 = float(a1)
        2*nop

        *r1 = a0 = a0 - a1 * *r4--      /* Update the seed value        */
        3*nop
        a0 = a0 * *r4
        2*nop
        *r2++ = a1 = *r7 + a0 * *r8

stop:   nop
        if (r6-- >= 0 ) pcgoto ran      /* Loop for table size          */
        nop

  /*---------------------------------------------------------------*/
  /*  Now use the table to create the output values.               */
  /*---------------------------------------------------------------*/


noInit: r10e = r17 + KAR_RETURN_DATA    /* r10 points to return data    */
        r7e = OFFSET                    /* load the offset              */
        r7e = *r7

        r11e = r17 + KAR_SIZE           /* TableSize addr               */
        r9e = TEMP_INT
        a0 = dsp(*r11)                  /* Size of wave table           */
        *r9 = a1 = int24(a0)            /* r9 points to wave table size */

        r6e = 256-2                     /* r6 is the counter(frameSize-2) */
        r2e = r7 + waveTable            /* Pointer to wave table        */
        r4e = POINT5                    /* Constant 0.5                 */
        r9e = 256                       /* r9 = tableSize               */

        r7e = GAIN
        r9e = r9 * 2
        r9e = r9 * 2                    /* Multiply by 4 for float size */
        r5e = r9 + waveTable            /* Address of end of table      */

loop:   r3e = r2 - 4                    /* Let r3 trail r2 by one sample */
        r3 - waveTable                  /* See if we've fallen off front */
        if (ge) pcgoto skip1
        nop
        r3e += r9                       /* r3 = r3 + length of table    */

skip1:  r2 - r5                         /* See if we reached end of table */
        if (lt) pcgoto skip2
        nop
        r2e = waveTable                 /* Start at front again         */

skip2:  a1 = *r2 + *r3                  /* Add two samples              */
        3*nop

        *r2++ = a2 = a1 * *r4           /* Average                      */
        2*nop

        a0 = a2 * *r7                   /* Multiply by gain             */
        a0 = *r2++ * *r7                /* Multiply by gain             */
        3*nop
        *r10++ = a0 = ieee(a0)          /* Place into sample buffer     */

Cont:   if (r6-- >= 0) pcgoto loop
        nop                             /* Latent instruction           */
        r7e = OFFSET                    /* Store the last position      */
        r2e = r2 - waveTable
        *r7 = r2

endit:  r8e = r17 + KAR_FLAGS           /* r8 points to host flags      */
        r2e = *r8                       /* r2 contains flags from host  */
        nop                             /* latent instruction           */
        r2e |= DO_FFT_FLAG              /* set the do_fft flag          */
        r2e &= ~HOST_READY_FLAG         /* Clear the host_ready flag    */
```

```
        *r8 = r2e                       /* Write flags to memory            */

        r2e = r17 + KAR_INDEX
        a0 = dsp(*r2)
        2*nop
        r3e = FRAME
        a0 = a0 + *r3
        3*nop
        *r2 = a0 = ieee(a0)             /* Write Index to host              */

end:    return (r18)
        nop

ranmul:         float   25173., 13849., -0.5
ranmod:         float   (1.0/32768.0), 32768.0      /* 2^(-15), 2^15 */
_seed:          float   2378.0
TEMP_FLOAT:     float
.align 4
TEMP_INT:       int
.align 4
POINT5:         float   0.500000
GAIN:           float   (40.7)
MINUS_ONE:      float   (-1.0)
TWO:            float   2.0
ZERO:           float   0.0
FRAME:          float   256.0

/*---------------------- These are the variable parameters ------------ */
.rsect ".var"
INDEX:          float   0.0
OFFSET:         int24   0
.align 4
waveTable:      512 * float
```

# ifunc.s

```
/*============================================================================*/
/*
/* NAME
/*      ifunc.s
/*
/* SYNOPSIS
/*      This is the initialization function for all of the DSP modules.
/*      it is executed befor the module and sets up registers r16e and
/*      r17e to point to the PRB and the user params, respectively. It
/*      also sets up register r13e to point to the Vfunction header and
/*      calls the Vfunction routine (main module routine).
/*
/*      Author Jeff Boone, Oregon State University
/*
/*============================================================================*/

.rsect  ".host"                         /* Execute out of host memory          */
#include "dspTaskDispatcherOffsets.h"

ifunc:

        goto pc+4
        nop
        oldr18:nop

        r2e=pc-12                       /* r2 points to oldr18                 */
        *r2 = r18e

        r1e=r12

        r2e = r1 + CD_NUMDBH_0 + PHYSICAL       /* load data stucture address */
        r16e = *r2                      /* 1st data structure address (in this case: PRB)   */
        r2e = r1 + CD_NUMDBH_9 + PHYSICAL
        r17e = *r2                      /* 9th data structure address  (parameter block)    */

        r13e = r1 + CD_FIRSTVFUNCHEADER + PHYSICAL          /* r13 points to vfunc header      */

        r3e = r1 + CD_DSPGLOBALS + PHYSICAL             /* find out vcache function location */
        r3e = *r3
        nop
        r3e += G_VCACHE + PHYSICAL
        r3e = *r3
        nop
        call r3(r18)                            /* Call the Vfunction (main module)        */
        nop

        r2e =pc - 76                            /* point to oldr18                         */
        r18e = *r2
        nop
        return  (r18)
        nop
```