Implementing a 10-Band Stereo Equalizer on the DSP56311 EVM Board

By James M. Montgomery

This document describes the development and implementation of a 10-band stereo equalizer programming example on the DSP56311 evaluation module (EVM). It provides an example of how to use readily available development tools to develop complex code for the DSP56311EVM. It also discusses how to program the enhanced filter coprocessor (EFCOP) in Multichannel mode.

The DSP56311EVM is a low-cost hardware platform that serves as a hardware reference design for system and board designers using the DSP56311. It is also a very flexible platform for developing DSP56311 code. Software engineers can download software to on-device or on-board RAM, then run and debug it.

The DSP56311EVM features:

- DSP56311 24-bit digital signal processor
- FSRAM for expansion memory and flash memory for stand-alone operation
- 16-bit CD-quality audio codec
- Command converter circuitry

For details on the DSP56311EVM, consult the DSP56311EVM Product Preview (DSP56311EVMP) and the *DSP56311EVM User's Manual* (DSP56311EVMUM). These documents are available on the Freescale web site listed on the back cover of this document.

CONTENTS

1 Filter Design

The 10-band stereo equalizer is constructed using 10 digital IIR bandpass filters in parallel for each stereo audio channel. The on-board codec samples the incoming audio stream at 48,000 Hz. The center frequencies for these filters lie between 0 Hz to $f_s/2$ (where f_s is the sample frequency of 48,000 Hz).

Figure 1 shows the passive RCL circuit forming a bandpass filter. The digital IIR Filter discussed later in this application note is based on this circuit. The s-domain analysis of the second-order bandpass analog filter is also shown.

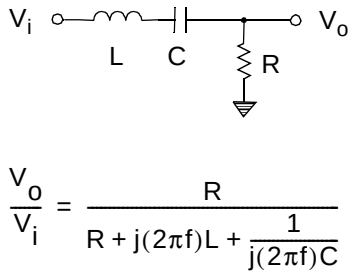

Figure 1. Analog Bandpass Filter and Voltage Divider Analysis

Equation 1 shows the s-domain transfer function of the circuit. H(s) is derived from the voltage divider analysis of the RCL network to be:

$$H(s) = \frac{V_o}{V_i} = \frac{Rs}{Rs + Ls^2 + \frac{1}{C}} \qquad \textbf{Equation 1}$$

where $s = j(2\pi f)$.

Equation 2 shows the bilinear transformation between the s-plane and the z-plane:

$$s = \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) \qquad \textbf{Equation 2}$$

where $z = e^{j\theta}$, $\theta = \omega T = (2\pi f)(1/f_s)$, and T is the sample period $(1/f_s)$.

Using **Equation 2**, the z-plane transfer function is found from **Equation 1**:

$$H(z) = \frac{\alpha(1 - z^{-2})}{\frac{1}{2} - \gamma z^{-1} + \beta z^{-2}} \qquad \textbf{Equation 3}$$

With **Equation 3**, the coefficients for each filter are calculated using the following three equations:

$$\beta = \frac{1}{2}\frac{1 - \tan\left(\frac{\theta_o}{2Q}\right)}{1 + \tan\left(\frac{\theta_o}{2Q}\right)} \qquad \textbf{Equation 4}$$

$$\gamma = \left(\frac{1}{2} + \beta\right)\cos\theta_o \qquad \textbf{Equation 5}$$

$$\alpha = \left(\frac{1}{2} - \beta\right)/2 \qquad \textbf{Equation 6}$$

where $Q = f_o/(f_2 - f_1)$ and $\theta_o = 2\pi(f_o/f_s)$. The value f_o is the center frequency of the bandpass filter, f_1 and f_2 are the half-power points (where the gain is equal to $1/(\sqrt{2})$), and f_s is the sample frequency. These equations are approximations for center frequencies less than $f_s/8$ (or 6000 Hz). To implement the transfer function from **Equation 3** as a digital IIR Filter, it must be transformed to a difference equation in the discrete time domain. **Equation 7** shows this difference equation, and **Figure 2** shows its representation as a network diagram.

$$y(n) = 2\{\alpha[x(n) - x(n-2)] + \gamma y(n-1) - \beta y(n-2)\} \qquad \textbf{Equation 7}$$

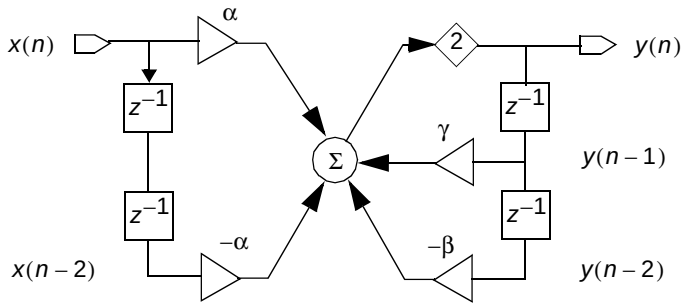

Figure 2. Bandpass IIR Filter Network Diagram

At each sample period, a left and right sound byte is fed to the 10 filters in parallel (see **Figure 3**). After each respective bandpass filter eliminates the frequencies not in its range, each output ($y_1(n) \rightarrow y_{10}(n)$) is scaled by an output gain. This gain value ranges from 0 to 1. The results of the ten filters are then summed together and outputted. This process allows one to selectively remove, or limit, the gain of a particular frequency range from the sound source.

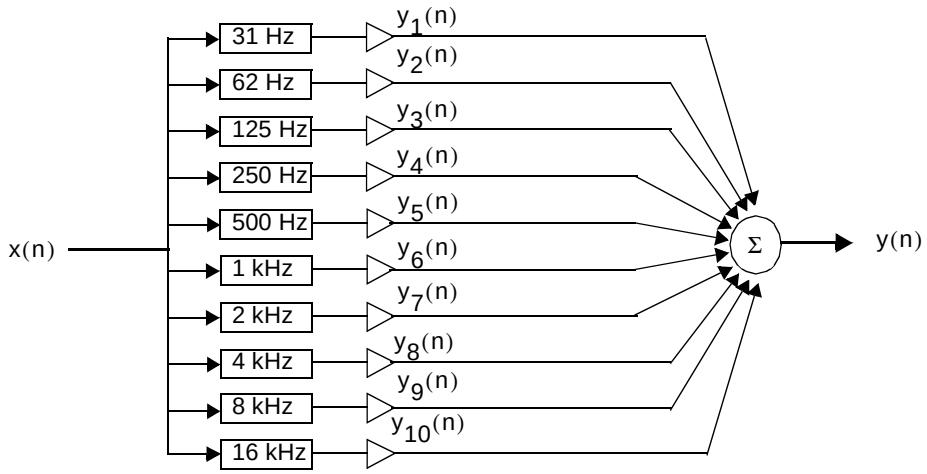

Figure 3. IIR Equalizer Data-Flow Diagram

Table 1 shows the 10 center frequencies chosen for this programming example. The coefficients for the center frequencies less than $f_s/8$ (or 6000 Hz) were found using equations 4–6 in **Section 1**. The coefficients for the center frequencies above 6000 Hz were found using more exact equations. Q is chosen to be 1.4.

Table 1. Digital IIR Bandpass Coefficients

Center Frequency	α	β	γ
31 Hz	0.000723575	0.49855285	0.998544628
62 Hz	0.001445062	0.497109876	0.997077038
125 Hz	0.002904926	0.494190149	0.994057064
250 Hz	0.005776487	0.488447026	0.987917799
500 Hz	0.011422552	0.477154897	0.975062733
1000 Hz	0.02234653	0.455306941	0.947134157
2000 Hz	0.04286684	0.414266319	0.88311345
4000 Hz	0.079552886	0.340894228	0.728235763
8000 Hz	0.1199464	0.2601072	0.3176087
16000 Hz	0.159603	0.1800994	-0.4435172

2 Development Environment

This section describes the development environment for the 10-band stereo equalizer (see **Figure 4**). It outlines the hardware and software requirements; describes how to establish the physical connection between the PC and the DSP56311 EVM board; and lists the steps for compiling, downloading, and running code on the DSP56311 EVM board. Once you complete these steps, you are ready to implement the 10-band stereo equalizer.

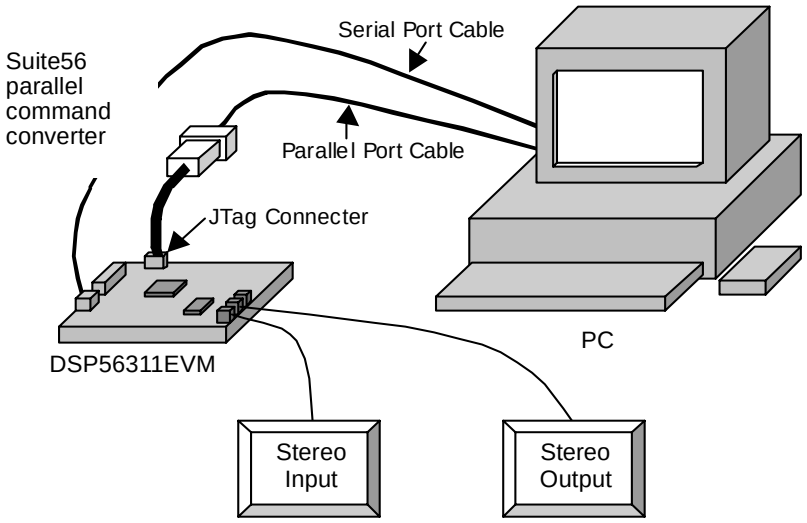

Figure 4. Development Setup

2.1 Personal Computer Running Windows NT® 4.0

The following programs should be running on your personal computer:

- *Codewright for Windows*. Programmer's text editor used to create and modify files. Note that other text editors can also be used.
- *Command Prompt.* DOS-style terminal used to run the *asm56300* compiler.
- *Suite56 DSP56300 Software Development Tools.* Free Freescale DSP tools to compile and link DSP assembly code. The hardware debugger, *ads56300,* has a GUI interface that communicates with the EVM Board through the parallel port command converter. It also downloads and executes code on the DSP56311EVM.
- *Visual Basic® 4.0.* Programming language for creating the equalizer GUI that allows you to change the gain values for the various equalizer bands.

2.2 Suite56 Parallel Command Converter

The parallel command converter provides the physical connection between the PC and the DSP56311EVM. Its parallel port interface connects to the PC. In addition, its female 14-pin header connects the device to the *JTAG/OnCE* Port (J2) on the DSP56311EVM. For details on this device, consult the *Suite56 Parallel Port Command Converter User's Manual* (DSPCOMMPARALLELUM).

The jumper settings on the DSP56311EVM are listed in **Table 2**. For details on how to set up the jumpers for the desired functionality, see the *DSP56311EVM User's Manual* (DSP56311EVMUM).

Table 2. Jumper Setting on the DSP56311 EVM Board

Number	Function	Description
J1	Boot Mode Select	HI08 bootstrap in MC68302 bus mode
J3	FSRAM Memory Configuration Option	Unified memory map
J4	SCI Header Pinout	Connects serial port connector signals RxD and TxD to the DSP SCI port

Table 2. Jumper Setting on the DSP56311 EVM Board (Continued)

Number	Function	Description
J5	SCI Port Clock	Connects on-board 156.3 kHz oscillator to the SCI port SCLK input (used for baud rate generation)
J6	On-board JTAG Enable/Disable Option	On-board command converter disabled
J7	ESSI0 Header Pinout	Selects the DSP ESSI0 port interface for use with an on-board codec
J8	CS4218 Sampling Frequency Selection	Selects 48 kHz sample rate for the codec
J9	ESSI1 Header Pinout	Selects DSP ESSI1 port interface for use with an on-board codec
J10	Core Current Measurement Jumper	Connected jumper that applies power to the DSP core

The *Line IN* jack on the DSP56311EVM connects to the headphone jack of the PC. The PC provides the sound source for the DSP56311EVM. A pair of headphones or stereo speakers can connect to the *Headphone OUT/Line OUT* jack to listen to the filtered sound source.

2.3 Interfacing the PC to the DSP56311EVM

Following are the steps to compile, download and run the code on the DSP56311EVM. It is assumed that you are using the GUI version of ADS56300 (part of the Suite56 DSP software development tools):

1. The `stereo.asm` file is the main assembly file of the project. Using the command prompt, change to the directory where the project files are stored.
2. At the prompt, type: `asm56300 -a -b -l stereo.asm`.

   Two output files are created. `Stereo.lst` contains a listing of the code, and `stereo.cld` is the executable to be downloaded to the DSP56311EVM. There may be a few warning when you compile the code. These warnings tell you of pipeline stalls located in the code. They have no effect on the operation of the code.
3. Using the ADS56300 GUI, reset the 56311EVM.
4. Under File → **Load** → **Memory COFF**, select the desired file (`stereo.cld`). Press **Apply** to load the file into memory.
5. Select **GO** (or type `go` into the Command window).

You should know about the following ADS56300 GUI windows:

- *Command.* Allows the user to type line commands.
- *Core Registers.* Displays the state of the core registers. The values can also be modified.
- *EFCOP Registers.* Displays the state of the EFCOP registers. The values can also be modified.
- *Assembly.* Displays the assembly code loaded in the DSP56311 program memory.
- *X Memory.* Displays the X-data Memory in the DSP56311.
- *Y Memory*. Displays the Y-data Memory in the DSP56311.

2.4 Useful Debugging Techniques

The breakpoint feature can be very useful. Software breakpoints stop at a particular instruction in program memory. Hardware breakpoints allow you to examine the effects of the DSP56311. For example, a breakpoint can be set up when a DMA channel writes data to one of the EFCOP registers in Y memory. This allows you to view

the state of the EFCOP after each sample is written to it. Hardware breakpoints are particularly helpful when EFCOP operation needs to be verified.

3 Implementation of 10-Band Stereo Equalizer

There are numerous ways to implement the 10-band stereo equalizer using the DSP56311EVM. The examples in this section show how to implement two versions of the equalizer. The overall functionality of both versions is identical. The main differences between the versions lie in how the DSP56311 resources are used. The two versions of the equalizer are compared in terms of:

- *Program Flow and Functionality.* The general flow serves as a template for designing each specific implementation. Pertinent information includes how the DSP is initialized, what the main interrupt sources are, how they are configured, and how they are handled. After examining these features, you should have a good idea at how the overall program is structured.
- *DSP56311 Core Implementation.* How to process the 10 bandpass filters using the DSP56300 core. The memory map and register usage must also be considered.
- *EFCOP and DMA Implementation.* How to process the 10 bandpass filters using the EFCOP in Multichannel mode and the DMA controller. The DSP56300 core is minimally used to set up the peripherals. The memory map, register usage, and peripheral setup must also be considered.

3.1 Program Flow and Functionality

The general program flow of the 10-band stereo equalizer occurs in 14 stages (see **Figure 5**). All but four of these stages (4, 6, 9, and 11) generally apply to any implementation. The first six stages initialize the DSP56311EVM hardware and software buffers in memory:

- Stage 1: Initialize the DSP56311. Set the clock frequency and bus interface.
- Stage 2: Initialize ESSI0 and ESSI1 to interface with the codec.
- Stage 3: Initialize the SCI to interface with an RS-232 port.
- Stage 4:
  - **a.** (DSP56311 core implementation): Set up the *Data Sample* and *Filter Coefficient* memory buffers. These data buffers reside in X and Y memory, respectively
  - **b.** (EFCOP and DMA Implementation): Set up EFCOP memory and initialize the DMA controllers.
- Stage 5: Set the equalizer knob values to a preset level.
- Stage 6: Set up the values in registers R0 to R7.

The last eight stages are part of an infinite loop to process the data received and transmitted through the codec:

- Stage 7: Frame sync for the codec.
- Stage 8: Get voice data from the receive buffer.
- Stage 9: Process the LEFT voice data using the 10 Bandpass filters.
- Stage 10: Store the LEFT voice data to transmit buffer.
- Stage 11: Process the RIGHT voice data using the 10 Bandpass filters.
- Stage 12: Store the RIGHT voice data to transmit buffer.

- Stage 13: Using the equalizer knob values, adjust the gain values for each of the 10 bandpass filters.
- Stage 14: Using the equalizer knob value that sets the main volume, adjust the main volume settings of the codec.

3.1.1 Equalizer Filter and Volume Gain

After each of the 10 digital IIR bandpass filters in the 10-band stereo equalizer eliminates the frequencies not in its range, the output is scaled by its respective output gain. The 10 gain values (and main volume) are determined by user-settable values called *equalizer knob values*.[1] The equalizer knob values consist of eleven 8-bit values received through the Serial Communication Interface (SCI) and placed into a predetermined space in Y memory (see **Figure 6**). The least significant five bits of the first ten knob values are used to select one out of 32 values in the filter gain table, ranging from –0.2 to 0.999 (see **Example 16** on page -31). The least significant five bits of the last knob value are used to select 1 out of 32 values in the volume gain table, which are used to configure the volume setting in the codec (see **Example 17** on page -32). Using interrupts, the SCI constantly updates the equalizer knob values in Y memory. In Stage 13, the knob values are used to update the runtime gain values. The 5-bit knob values function as indexes into the filter and volume Gain tables. The run-time gain values are then used in Stages 9 and 11 when the voice data is filtered. This process continually repeats.

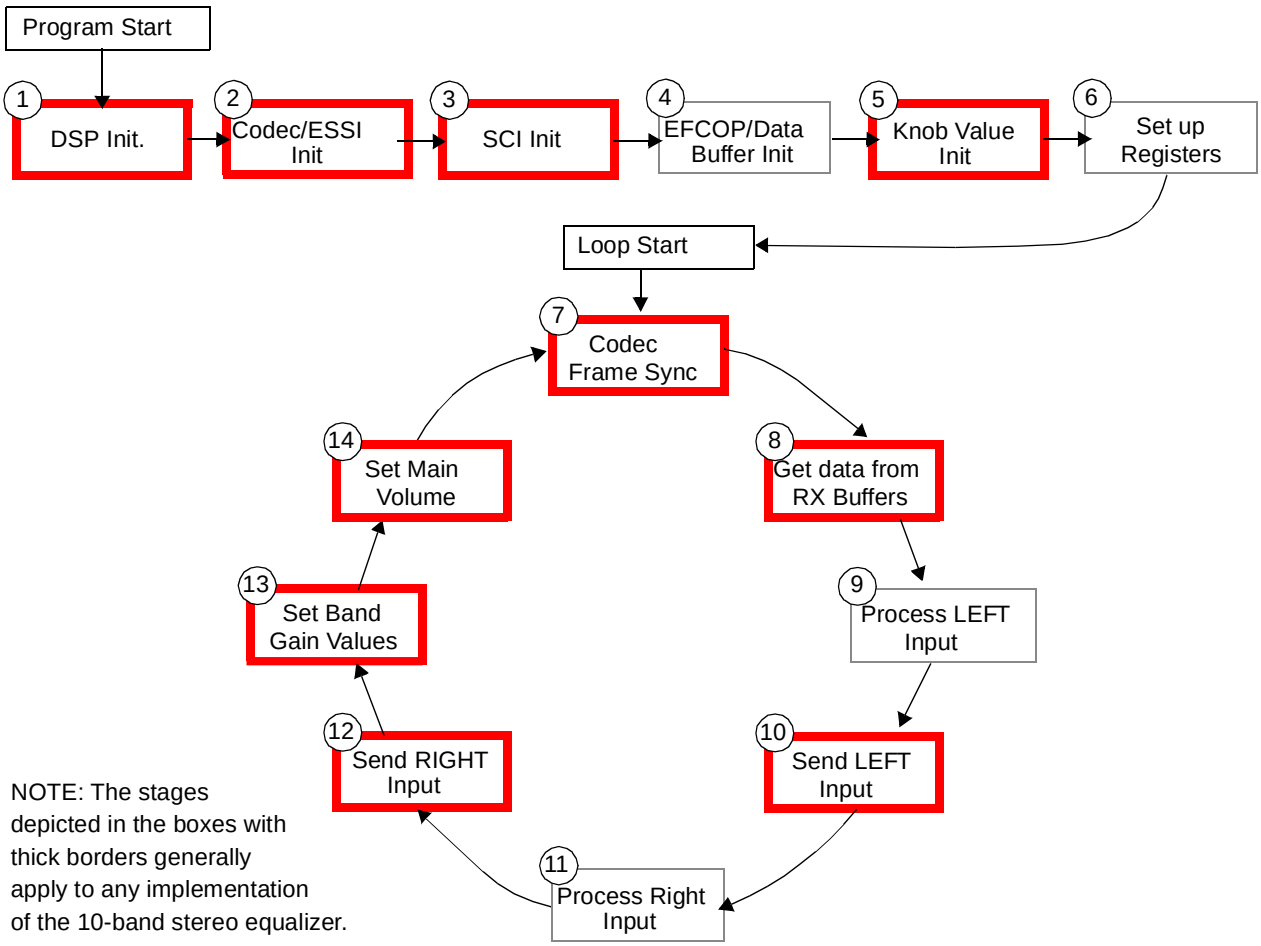

Figure 5. General Program Flow

1. The equalizer GUI is used to set the equalizer knob values.

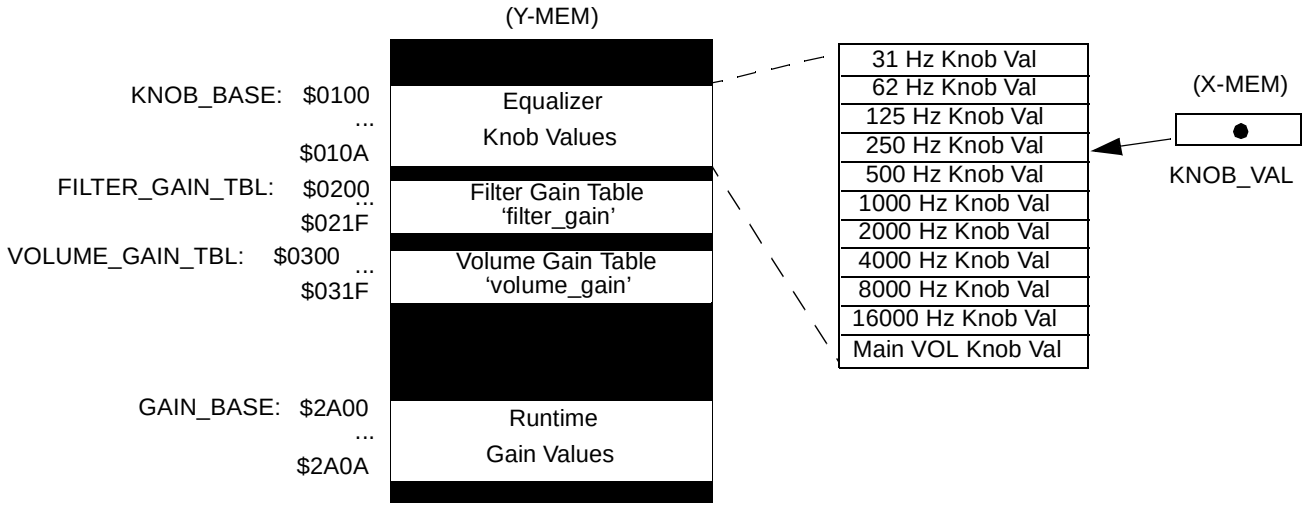

Figure 6. Knob and Gain Memory Areas

3.1.2 Stage 1: DSP Initialization

The first stage of the code, shown in **Example 1**, initializes the following DSP memory address locations:

- *INIT_PCTL*. Written to the Phase Lock Loop Control Register (PCTL) to set up the DSP56300 core operating frequency.
- *INIT_BCR*. Written to the Bus Control Register (BCR) to control the external bus activity and bus interface unit operation.
- *START*. Marks the start of the program in program memory.

Example 1. DSP Initialization

```
;*************************************************************************
       nolist
       include 'ioequ.asm'
       include 'intequ.asm'
       include 'ada_equ.asm'
       include 'vectors.asm'
       list

INIT_PCTL      EQU    $040006 ; Fcore=fcrystal*MF=12.288MHz*7=86 MHz
INIT_BCR       EQU    $012421

       org     p:$400
START
;--------------------------
; DSP Initialization
;--------------------------
       movep          #INIT_PCTL,x:M_PCTL   ; PLL 7 X 12.288 = 86.016MHz
       movep          #INIT_BCR,x:M_BCR     ; AARx - 1 wait state
       ori            #3,mr                 ; mask interrupts
       movec          #0,sp                 ; clear hardware stack pointer
       move           #0,omr                ; operating mode 0
```

3.1.3 Stage 2: Codec/ESSI Initialization and Operation

The second stage of the code, shown in **Example 2**, sets up the codec on the DSP56311 EVM Board. The receive and transmit buffer pointers are set up first. Then the *ada_init* procedure sets up the codec by initializing the ESSI0 and ESSI1 registers. For details, refer to *Programming the CS4218 CODEC for Use With DSP56300 Devices* (AN1790/D).

Example 2. Codec/ESSI Initialization

```
;--------------------------
; Codec Initialization
;--------------------------
       move          #RX_BUFF_BASE,x0
       move          x0,x:RX_PTR          ; Initialize the rx pointer
       move          #TX_BUFF_BASE,x0
       move          x0,x:TX_PTR          ; Initialize the tx pointer
       jsr           ada_init             ; Initialize codec
```

The codec is configured to sample incoming data at a rate of 48,000 Hz. **Figure 7** shows the data format between the ESSI interface and the codec. ESSI0 is configured for Network mode with two time slots. Slot 0 of the 32-bit frame always contains the Left Channel Word, while Slot 1 always contains the Right Channel Word.

Six ESSI0 interrupt service routines (ISRs) handle the ESSI0 receive and transmit interrupts. These ISRs are located in the `ada_init.asm` file. A receive interrupt occurs at the end of each time slot, after each channel is serially shifted into the ESSI0 Receive Shift Register and then transferred to the Receive Data Register. The receive interrupt service routines place each word in the receive buffers into X memory (see **Figure 7**). The 16-bit channel words are placed into the 16 most significant bits of the 24-bit memory word. The lower 8 bits are cleared. A transmit interrupt occurs at the beginning of each time slot. The transmit interrupt service routines place each word into the ESSI0 Transmit Data Register, where it is then transferred to the Transmit Shift Register and then serially shifted out of the DSP.

Stages 7 through 12 (**Figure 5**) take the left and right channel words from the Receive Data Buffer (RX_BUFF_BASE), process them, and place the result into the Transmit Data Buffer (TX_BUFF_BASE) to be transmitted during the next codec data frame. T1 and T2 in **Figure 7** represents the *maximum* time allowed to process each channel in order to be transmitted correctly during the following time frame. The Left Channel (**T1**) must be processed from the rising edge of the frame sync to the falling edge of the frame sync and placed into the Transmit Data Buffer in order to be transmitted on time. The Right Channel (**T2**) must be processed from the rising edge of the frame sync to the end of slot 0 of the next time frame and placed into the Receive Data Buffer. The restriction of the processing time for each channel is due to the design. With a core frequency of 86.016 MHz and a sampling rate of 48 kHz, **T1** = 1794 DSP clocks and **T2** = 59,198 DSP clocks.

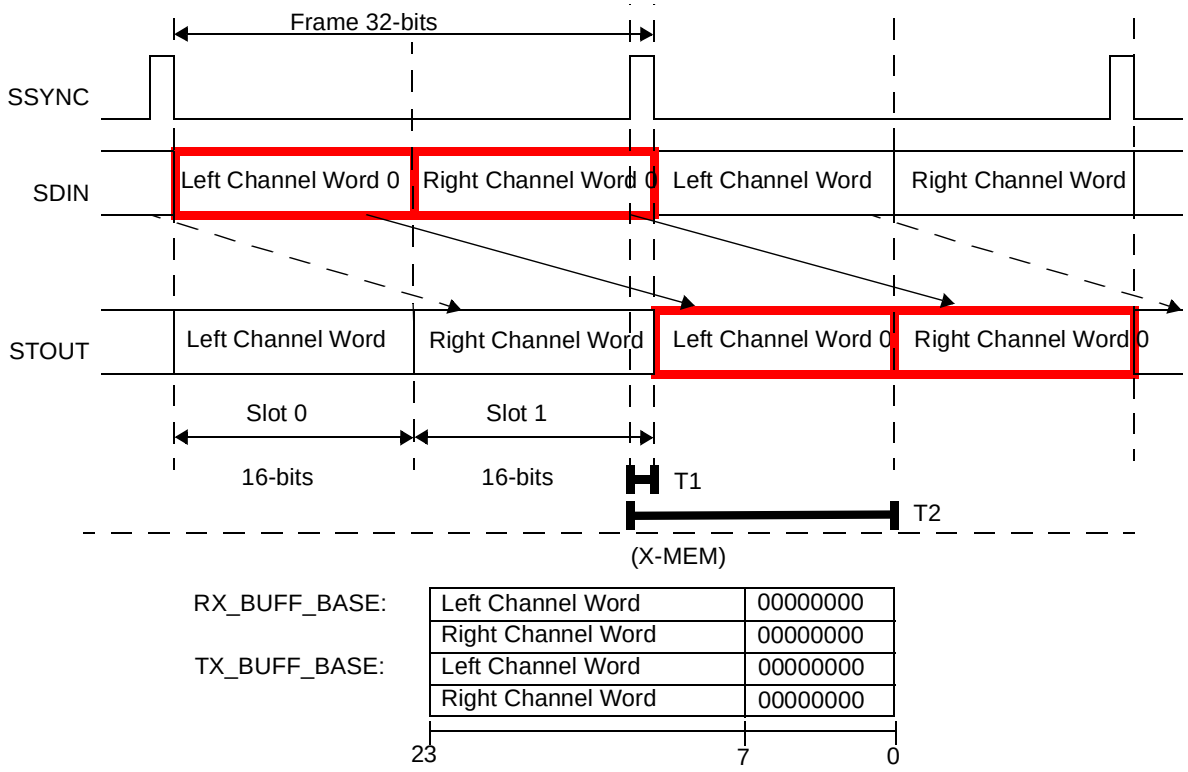

Figure 7. Codec Data Format and Layout in Memory

3.1.4 Stage 3: SCI Initialization and Operation

Stage 3 of the code, shown in **Example 3**, initializes the following memory address locations:

- *INIT_SCCR*. Written to the SCI Clock Control Register (SCCR) to set up the baud rate.
- *INIT_SCR*. Written to the SCI Control Register (SCR) to control the serial interface operation.

The Port E register (PCRE) is also configured to enable the SCI lines. Finally the pointer (KNOB_VAL) to the equalizer knob values in Y memory is initialized.

Example 3. SCI Initialization

```
INIT_SCCR       EQU     $002010 ; baud = Fcore/[64*(7(SCP)+1)*(CD+1)]
INIT_SCR        EQU     $000b02

;--------------------------
; SCI Initialization
;--------------------------
       movep           #INIT_SCCR,x:M_SCCR          ; Initialize SCI
       movep           #INIT_SCR,x:M_SCR
       movep           #$7,x:M_PCRE
       move            #KNOB_BASE,r0                ; Initialize the SCI pointer.
       move            r0,x:KNOB_VAL  ;
```

The SCI is configured to receive the equalizer knob values from the external PC (or other source). One ISR handles the SCI receive interrupt. A receive interrupt occurs when a byte is shifted into the SCI Receive Shift Register and then transferred to the Receive Data Register. The ISR places this byte in one of the equalizer knob value memory

locations to which KNOB_VAL points (see **Figure 6**). The operation of the SCI ISR is very simple (as shown in the top half of **Figure 8**).

1. It saves a few core registers to the system stack.
2. It reads the data byte from the SCI Receive Data Register.
3. The character “Enter” (hex value 0xd) is used to reset the table pointer (KNOB_VAL) to the equalizer knob value base (KNOB_BASE).
4. If the character “Enter” is read, KNOB_VAL is set to equal KNOB_BASE. Else if another character is read, then that character is put at the Y-Memory location pointed to by KNOB_VAL.
5. The core registers are then restored, and the interrupt exits.

In this programming example, the equalizer knob values come from the COM1 or COM2 port of a PC. The Equalizer GUI that allows the user to set the knob values sends the data pattern shown in the bottom half of **Figure 8**.

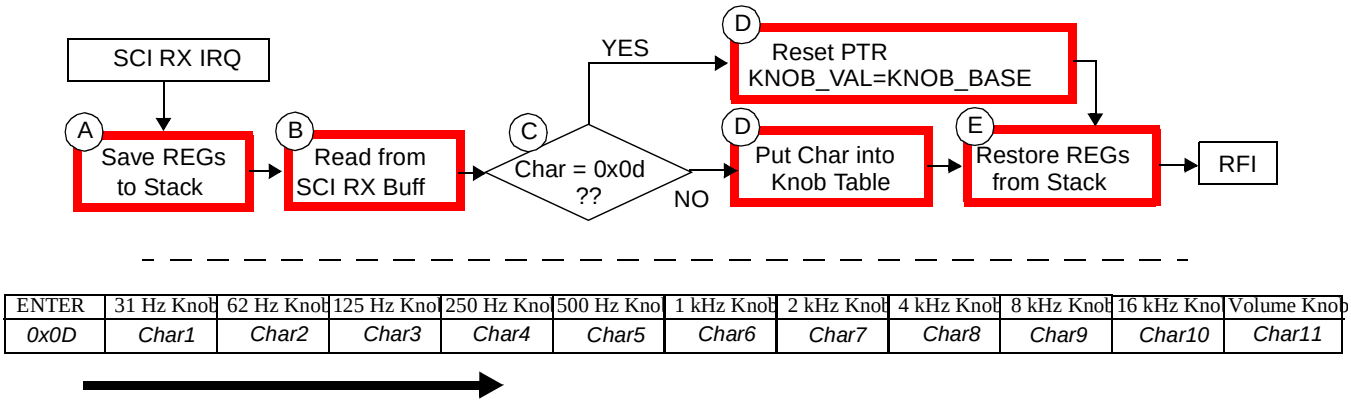

Figure 8. SCI Interrupt Service Routine and Incoming Data Pattern

3.1.5 Stage 4: EFCOP Memory Initialization and DMA Setup

Stage 4 of the code, shown in **Example 4**, sets up the X and Y memories for this version of the implementation. The code in program memory does two things.

- Clears the *x(n)* and *y(n)* data buffers at DATA_BASE_L and DATA_BASE_R.
- Sets up the data buffer pointers in memory using DATA_PTR.

Figure 9 shows the memory map for this implementation. The following areas of memory are specific to this implementation and have not been discussed in previous sections.

- *DATA_BASE_L and DATA_BASE_R*. These two areas in X memory hold the current data values for *x(n)* to *x(n-2)* and *y(n)* to *y(n-2)* for all 10 bandpass filters.
- *DATA_PTR*. It is necessary to store the pointers to the memory areas. Storing these values in X memory means that one register can be assigned to save and restore the four data pointers when they’re needed.
- *COEF_BASE*. This area of Y memory contains the α, β, and γ coefficients for each of the 10 bandpass filters.

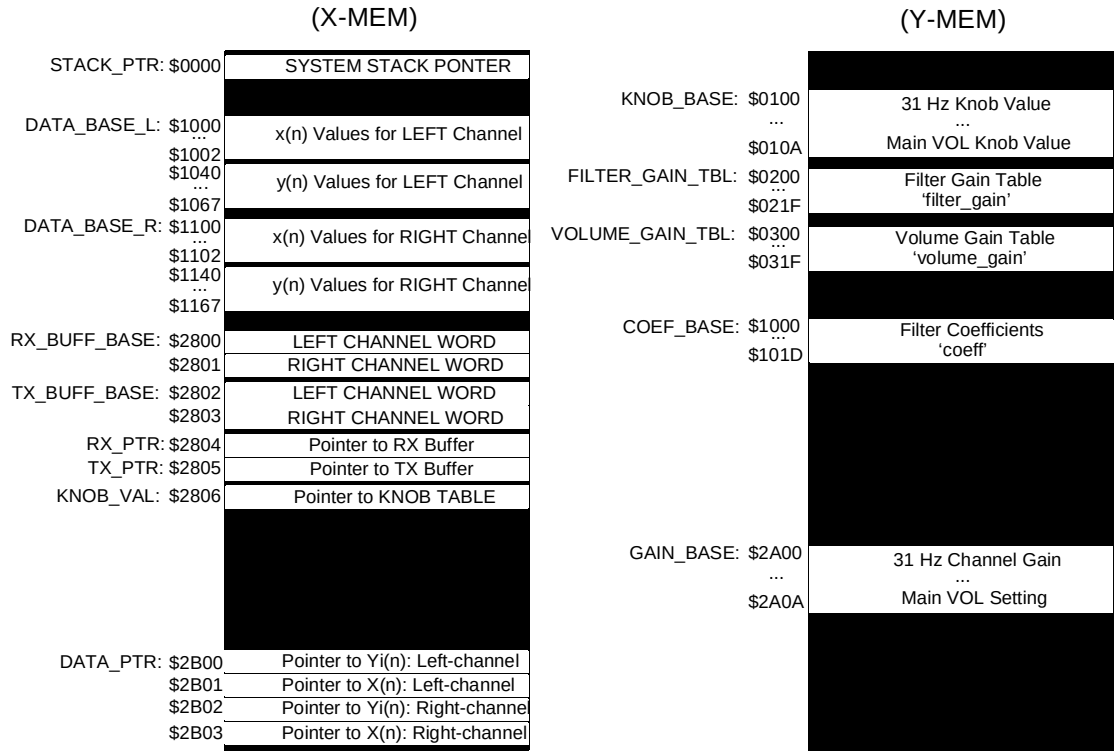

Figure 9. Memory Map for DSP56300 Core Implementation

Example 4. Filter Parameter Setup

```
        org             y:FILTER_GAIN_TBL
        include         'filter_gain'
        org             y:VOLUME_GAIN_TBL
        include         'volume_gain'
        org             y:COEF_BASE
        include         'coeff'


;;;;IN PROGRAM MEMORY;;;
;--------------------------
; Filter Parameter Setup
;--------------------------
        move    #0,r0

        ;Clear the x(n) and y(n) Data Buffers
        move            #DATA_BASE_L,r3
        rep             #$68
        move            r0,x:(r3)+
                        ; x:DATA_BASE_L+($00..$02) - x(n) left chan.
                        ; x:DATA_BASE_L+($40..$42) - y(n) left chan. 31 Hz
                        ; x:DATA_BASE_L+($44..$46) - y(n) left chan. 62 Hz
                        ;  ....
                        ; x:DATA_BASE_L+($64..$66) - y(n) left chan. 16 kHz

        move            #DATA_BASE_R,r3
        rep             #$68
        move            r0,x:(r3)+
```

```
                    ; x:DATA_BASE_R+($00..$02) - x(n) right chan.
                    ; x:DATA_BASE_R+($40..$42) - y(n) right chan. 31 Hz
                    ; x:DATA_BASE_R+($44..$46) - y(n) right chan. 62 Hz
                    ;  ....
                    ; x:DATA_BASE_R+($64..$66) - y(n) right chan. 16 kHz

; Setup Filter Data Buffers (and Pointers)
move            #DATA_PTR,r3
nop
move            #DATA_BASE_L+$40,r0   ; Yi(n):L-ch at x:DATA_BASE_L+$40
move            r0,x:(r3)+
move            #DATA_BASE_L,r0       ; X(n):L-ch  at x:DATA_BASE_L
move            r0,x:(r3)+
move            #DATA_BASE_R+$40,r0   ; Yi(n):R-ch at x:DATA_BASE_R+$40
move            r0,x:(r3)+
move            #DATA_BASE_R,r0       ; X(n):R-ch  at x:DATA_BASE_R
move            r0,x:(r3)
```

Next, we implement the 10-band stereo equalizer using the DSP56311 EFCOP to process the bandpass filters and DMA to transfer the DATA to/from the EFCOP. Stage 4 of the code, shown in **Example 5**, sets up the X and Y memories for this implementation version. The code in program memory does the following:

- Clears the four EFCOP data buffers at FIR_FDBA_L, FIR_FDBA_R, IIR_FDBA_L, and IIR_FDBA_R.
- Sets up the EFCOP data buffer pointers in memory using FDBA_PTR.

Figure 10 shows the memory map for this implementation.

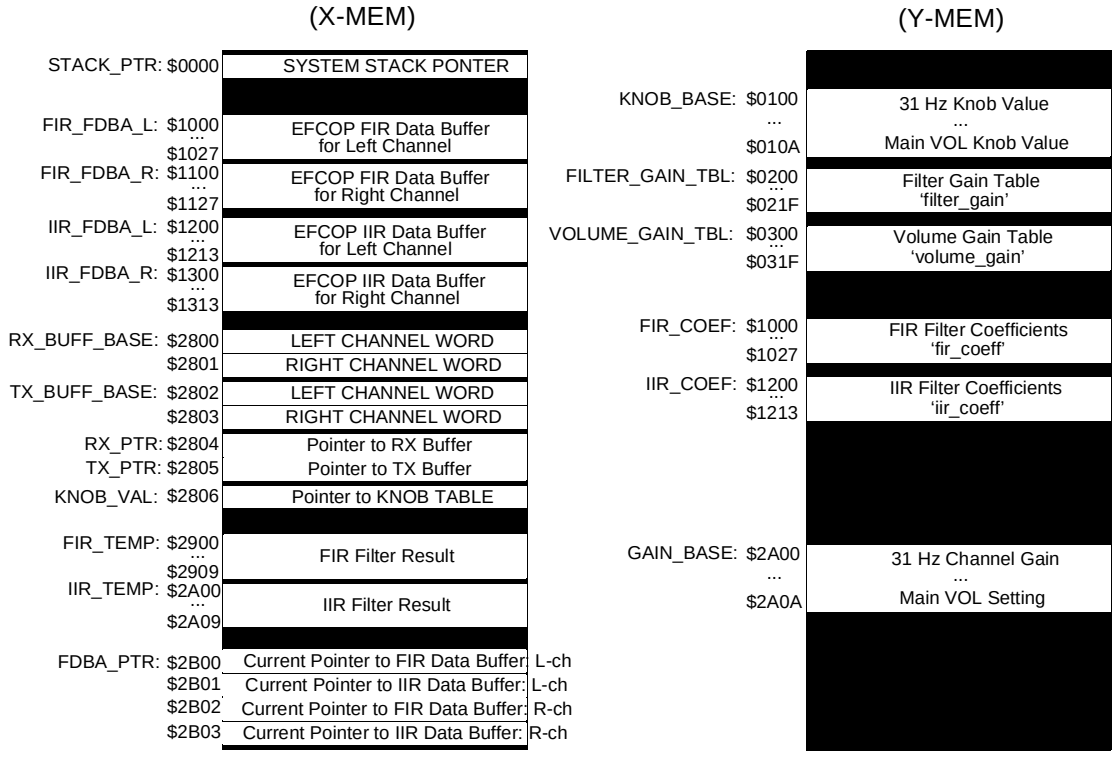

Figure 10. Memory Map for EFCOP/DMA Implementation

The following areas of memory are specific to this implementation and are not discussed in previous sections.

- *FIR_FDBA_L*, *FIR_FDBA_R*. These two areas in X memory hold the current left and right x(n) to x(n-2) data values for each of the 10 EFCOP filter channels.
- *IIR_FDBA_L*, *IIR_FDBA_R*. These two areas in X memory hold the current left and right y(n) to y(n-2) data values for each of the 10 EFCOP filter channels.
- *FIR_TEMP*. This area in X memory holds the result from the EFCOP after it has processed the FIR part of the IIR filter. This area is 10 words long (one word for each of the 10 channels).
- *IIR_TEMP*. This area in X memory holds the result from the EFCOP after it has processed the IIR part of the IIR filter. This area is 10 words long (one word for each of the 10 channels).
- *FDBA_PTR*. Due to the nature of the program, it is necessary to store the FDBA register of the EFCOP after each use. This pointer saves and restores the correct data pointer values to the EFCOP.
- *FIR_COEF*. This area of Y memory contains the α coefficients for each of the 10 EFCOP channels.
- *IIR_COEF*. This area of Y memory contains the β and γ coefficients for each of the 10 EFCOP channels.

Example 5. EFCOP Memory Initialization

```
;--------------------------
; EFCOP Initialization
;--------------------------
       move          #0,r0

       ; Clear the EFCOP Data Buffer
       move          #FIR_FDBA_L,r3        ; FIR Left Channel
       rep           #40
       move          r0,x:(r3)+

       move          #FIR_FDBA_R,r3        ; FIR Right Channel
       rep           #40
       move          r0,x:(r3)+

       move          #IIR_FDBA_L,r3        ; IIR Left Channel
       rep           #20
       move          r0,x:(r3)+

       move          #IIR_FDBA_R,r3        ; IIR Right Channel
       rep           #20
       move          r0,x:(r3)+

       ; Clear the Temporary Storage Areas
       move          #FIR_TEMP,r3
       rep           #CHANNELS
       move          r0,x:(r3)+

       move          #IIR_TEMP,r3
       rep           #CHANNELS
       move          r0,x:(r3)+

       ; Setup EFCOP Data Buffers (and Pointers)
       move          #FDBA_PTR,r3          ; Base pointer for FDBA values (X mem)
```

```
        move            #FIR_FDBA_L,r0
        move            r0,x:(r3)+
        move            #IIR_FDBA_L,r0
        move            r0,x:(r3)+
        move            #FIR_FDBA_R,r0
        move            r0,x:(r3)+
        move            #IIR_FDBA_R,r0
        move            r0,x:(r3)+

        ; Setup Channels in EFCOP
        movep           #CHANNELS-1,y:M_FDCH  ; # of EFCOP Channels
```

3.1.6 Stage 5: Equalizer Knob Value Initialization

Stage 5 of the code, shown in **Example 6**, clears the memory spaces corresponding to the 'Runtime' gain Values by writing a 0x0 to them. The equalizer knob values in memory are then set with the value 0x1F (for the filter gain values) and 0x10 (for the volume gain value).

Example 6. Knob Value Initialization

```
;--------------------------
; Knob Value Initialization
;--------------------------
        ; Clear the 'Runtime' Gain Values in memory
        move            #0,r0
        move            #GAIN_BASE,r3
        rep             #11
        move            r0,y:(r3)+

        ; Set equalizer knob values (for Filters)
        move            #$00001f,r0           ; Set index into Filter Gain Table
        move            x:KNOB_BASE,r3
        rep             #10
        move            r0,y:(r3)+

        ; Set equalizer knob values (for Volume)
        move            #$000010,r0           ; Set index into Volume Gain Table
        nop
        move            r0,y:(r3)+
```

3.1.7 Stage 6: Set up Registers R0 to R7

This implementation of the 10-band stereo equalizer uses all of the available DSP56300 core registers, as shown in **Example 7**.

Example 7. Register Usage

```
;--------------------------
; Setup Registers
;--------------------------
        ; R0 - IIR Coeff Pointer (30-word Buffer)
        move            #COEF_BASE,r0         ; IIR Coeff for Left/Right Chan.
        move            #29,m0
        ; R1 - Knob Value Pointer (11-word Buffer)
        move            #KNOB_BASE,r1
```

```
; R2 - Points to the Filter and Volume Gain Tables
move          #FILTER_GAIN_TBL,r2
; R3 - 'Runtime' Filter Gain Pointer (11-word Buffer)
move          #GAIN_BASE,r3
move          #10,m3
; R4 - Pointer to Yi(n) buffers (3-words * 10 Bands)
move          #2,m4                ; Set y(n) modulo for 3 words
move          #4,n4
; R5 - Pointer to X(n) buffer (3-words)
move          #2,m5
; R6 - User Stack Pointer
move          #STACK_PTR,r6        ; initialize stack pointer.
move          #-1,m6               ; linear addressing
; R7 - Holds Pointer Value for current Data Buffer (4-Words)
move          #DATA_PTR,r7         ; Base pointer for Data values (X mem)
move          #3,m7                ; Set the buffer to 4
```

The core register usage is as follows:

- *R0*. Pointer to filter coefficients in Y memory (30-word circular buffer)
- *R1*. Pointer to knob values in Y memory (11-word buffer)
- *R2*. Pointer to filter gain table and volume gain table in Y memory.
- *R3*. Pointer to filter gain values in Y memory (11-word buffer).
- *R4*. Pointer to Yi(n) data buffers in X memory. This register is used for both the left and right channels.
- *R5*. Pointer to X(n) data buffers in X memory. This register is used for both the left and right channels.
- *R6*. System stack pointer, primarily used for interrupt service routines. The routines can use this register to save and restore the state of regular code flow.
- *R7*. Pointer for the current data buffer pointers. This register helps store the *runtime* X(n) and Yi(n) data buffer pointer values in X memory (4-word buffer).

3.1.8 Stage 7, 8, 10, and 12: Codec Operation

The code for Stages 7, 8, 10, and 12 is shown in **Example 8**. Stages 7 through 14 make up an infinite loop that processes the left and right voice channels that are received. In Stage 7, the Receive Frame Sync bit (RFS) of the ESSI Status Register (SSISR) is used to start each loop. In Stage 8, the left and right voice data, stored at RX_BUFF_BASE, is moved to registers in the DSP. After the voice data from the left and right channels is processed, it is moved to TX_BUFF_BASE.

Example 8. Codec Code

```
;--------------------------
; START LOOP
;--------------------------
loop
       ; Get Left and Right Channel Data Bytes
       jset          #3,x:M_SSISR0,*             ; wait for RX frame sync
       jclr          #3,x:M_SSISR0,*             ; wait for RX frame sync

       move          x:RX_BUFF_BASE,x1           ; receive left
       move          x:RX_BUFF_BASE+1,y1         ; receive right
```

```
;;; PROCESS LEFT INPUT code ;;;
       move          a,x:TX_BUFF_BASE            ; transmit left data byte
;;; PROCESS RIGHT INPUT code ;;;
       move          b,x:TX_BUFF_BASE+1          ; transmit left data byte
```

3.1.9 Stage 9 and 11: Process Left/Right Input

Processing of the left and right voice data bytes is practically identical. The only difference is the codec data bytes that are filtered. For a complete IIR filter to be implemented, the voice data must be processed using the EFCOP FIR and IIR types of filters. These two filter types (shown in **Figure 11**) are used together to create two filtering phases. During the first phase, the FIR results for each of the 10 channels are calculated using the EFCOP. DMA 0 transfers the codec voice data sample to the EFCOP, and DMA 1 transfers the results to the FIR_TEMP buffer. During the second phase, the IIR results for each of the 10 Channels are calculated. DMA 2 transfers the FIR results to the EFCOP, and DMA 3 transfers the final results to the IIR_TEMP buffer in X memory (see **Example 9**). The results are then multiplied by their respective gain values and added together.

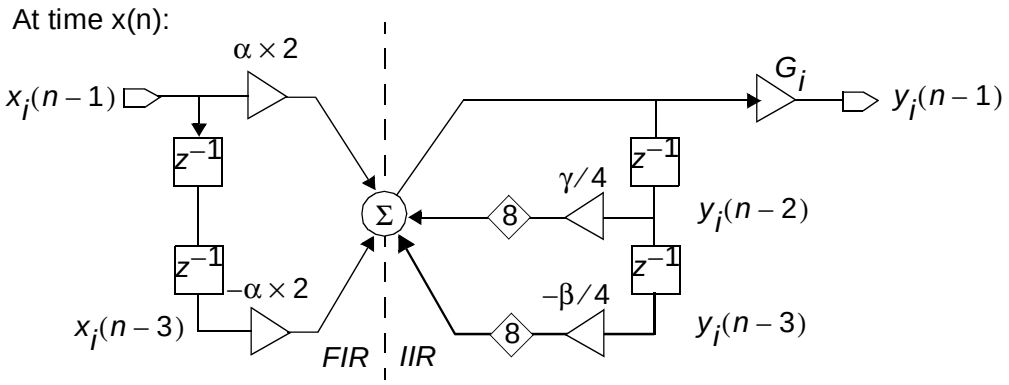

Figure 11. EFCOP IIR Block Diagram

The FIR coefficients (in the `fir_coeff` file *)* are multiplied by two. Similarly, the IIR coefficients (in the `iir_coeff` file*)* are divided by four. These operations produce the correct multiplication factor while the EFCOP is processing the data in the IIR phase. The EFCOP IIR block diagram for a single channel (**Figure 11**) shows the how the two EFCOP phases are related. The EFCOP in IIR mode is configured so that it scales the feedback terms by 8. The EFCOP also introduces a time delay when it is in FIR Multichannel mode. This is why at time x(n), x(n–1) is processed instead. The channel gain, *Gi*, can be set to have a value between –0.2 and 0.999 (see **Section 3.1.1**, *Equalizer Filter and Volume Gain*, on page 8). A 3-tap FIR filter is used during the FIR filtering phase, implemented as follows:

1. Set the filter count register (FCNT) to the length of the filter coefficients –1 (that is, 2).
2. Set the Data and Coefficient Base Address Pointers (FDBA, FCBA).
3. Clear the ALU control register (FACR).
4. Set the control and status register (FCSR):
   - FSCO = 0 (EFCOP filter coefficients are stored sequentially in memory)
   - FPRC = 1 (EFCOP starts processing with no state initialization)
   - FMLC = 1 (Multichannel Mode)
   - FOM = 00 (Real FIR filter)

— FLT = 0 (FIR filter)

— FEN = 1 (Enable EFCOP)

A 2-tap IIR filter is used during the IIR filtering phase, implemented as follows:

1. Set the filter count register (FCNT) to the length of the filter coefficients-1 (i.e. 1).
2. Set the Data and Coefficient Base Address Pointers (FDBA, FCBA).
3. Set the ALU control register (FACR).
   — FISL = 1 (Determines where Scaling is done)
   — FSCL = 01 (Scaling factor of 8)
4. Set the control and status register (FCSR):
   — FSCO = 0 (EFCOP filter coefficients are stored sequentially in memory)
   — FPRC = 1 (EFCOP starts processing with no state initialization)
   — FMLC = 1 (Multichannel Mode)
   — FLT = 1 (IIR filter)
   — FEN = 1 (Enable EFCOP)

Example 9. Use EFCOP and DMA to Process the Left Channel

```
;;; PROCESS LEFT INPUT

; Initialize EFCOP for FIR stage of LEFT input
lfstart
       movep          #$000,y:M_FCSR        ; Reset the EFCOP

       movep          #FIR_LEN-1,y:M_FCNT   ; Set the counter for 3 Coeffs
       movep          x:(r7),y:M_FDBA       ; R7 = Current FIR Data Pointer
       movep          #FIR_COEF,y:M_FCBA    ; FIR Coeff Pointer
       movep          #$000,y:M_FACR        ; Clear the FACR
       movep          #$0C1,y:M_FCSR        ; Enable EFCOP

; Initialize DMA 0 (Data Samples -> EFCOP {FDIR Reg})
       movep          #RX_BUFF_BASE,x:M_DSR0; DMA source is the sound data buffer
       movep          #M_FDIR,x:M_DDR0      ; DMA Destination is the EFCOP (Y Mem)
       movep          #CHANNELS-1,x:M_DCO0  ; DMA Count in mode A
       movep          #$8eAA44,x:M_DCR0     ; Enable DMA Channel 0

; Initialize DMA 1 (EFCOP {FDOR Reg} -> FIR Temp Storage)
       movep          #M_FDOR,x:M_DSR1      ; DMA source is the EFCOP (Y Mem)
       movep          #FIR_TEMP,x:M_DDR1    ; DMA Destination is FIR_TEMP in X Mem
       movep          #CHANNELS-1,x:M_DCO1  ; DMA Count in mode A
       movep          #$8EB2C1,x:M_DCR1     ; Enable DMA Channel 1

; Wait for Completion of FIR Stage
       jclr           #0,x:M_DSTR,*         ; DMA 0 Finished
       jclr           #1,x:M_DSTR,*         ; DMA 1 Finished

       movep          y:M_FDBA,x:(r7)+      ; Update FIR Data Pointer, and
                                            ; Point to IIR Data Pointer
; Initialize EFCOP for IIR stage of Left Input
       movep          #$000,y:M_FCSR        ; Reset the EFCOP
```

```
        movep           #IIR_LEN-1,y:M_FCNT   ; Set the Counter to 2 Coeffs.
        movep           x:(r7),y:M_FDBA       ; R7 = Current IIR Data Pointer
        movep           #IIR_COEF,y:M_FCBA    ; IIR Coeff Pointer
        movep           #$041,y:M_FACR        ; Set up Scaling factor
        movep           #$0C3,y:M_FCSR        ; EFCOP enable

; Initialize DMA 2 (FIR Temp Storage -> EFCOP {FDIR Reg})
        movep           #FIR_TEMP,x:M_DSR2    ; DMA source is the sound data buffer
        movep           #M_FDIR,x:M_DDR2      ; DMA Destination is the EFCOP (Y Mem)
        movep           #CHANNELS-1,x:M_DCO2  ; DMA Count in mode A
        movep           #$8EAA54,x:M_DCR2     ; Enable DMA Channel 2

; Initialize DMA 3 (EFCOP {FDOR Reg} -> FIR Temp Storage)
        movep           #M_FDOR,x:M_DSR3      ; DMA source is the EFCOP (Y Mem)
        movep           #IIR_TEMP,x:M_DDR3    ; DMA Destination is FIR_TEMP in X Mem
        movep           #CHANNELS-1,x:M_DCO3  ; DMA Count in mode A
        movep           #$8EB2C1,x:M_DCR3     ; Enable DMA Channel 3
; Wait for Completion of IIR Stage
        jclr            #2,x:M_DSTR,*         ; DMA 2 Finished
        jclr            #3,x:M_DSTR,*         ; DMA 3 Finished

        movep           y:M_FDBA,x:(r7)+      ; Update IIR Data Pointer, and
                                              ; Point to FIR Data Pointer
                                              ;  (Right Channel)

; Send out sound byte
        move            #IIR_TEMP,r0          ; Pointer to IIR values
        move            #GAIN_BASE,r4         ; Pointer to Gain values

        clr             a               x:(r0)+,x0     y:(r4)+,y0

        do              #9,left_out
        mac             x0,y0,a         x:(r0)+,x0     y:(r4)+,y0
left_out

        macr            x0,y0,a

        move            a,x:TX_BUFF_BASE      ; transmit left data byte
```

3.1.10 Stage 13 and 14: Setting Knob and Main Volume Gain Values

Stages 13 and 14 are shown in **Example 10**. During Stage 13, the equalizer knob values (at KNOB_BASE) are used as indexes into the filter gain table. The gain values for the 10 filters come from this Table. During Stage 14, the last equalizer knob value is used as an index into the volume gain table. The codec controls the main volume for the system. If the knob value for the volume is between 0x0 to 0xF, then the output is attenuated (less sound). Attenuation is accomplished by reprogramming the upper control word (CTRL_WD_HI) for the codec. If the knob value for the volume is between 0x10 to 0x1F, then gain is added (more sound). Gain is accomplished by reprogramming the lower control word (CTRL_WD_LO) for the codec. The value programmed to the codec is in the volume gain table.

Example 10. Setting Gain Values

```
; GET and SET new Band Gain Values
```

```
bgain_s
       move    #KNOB_BASE,r1               ; Pointer to equalizer knob values.
       move    #FILTER_GAIN_TBL,r2         ; Pointer to Filter Gain Table.
       move    #GAIN_BASE,r3               ; Pointer to 'runtime' gain values.

       clr           a
       do            #10,bgain             ; 10 Knobs for 10 Filter Channels
       move          y:(r1)+,a             ; 1. Get Knob Value.
       and           #$00001F,a            ; 2. Mask for lowest 5 bits.
       move          a1,n2                 ; 3. Set index into Filter Gain Table.
       move          y:(r2+n2),r0          ; 4. Use index to get Filter Gain Value.
       move          r0,y:(r3)+            ; 5. Update 'runtime' Filter Gain Value
bgain
vgain_s
       move    #VOLUME_GAIN_TBL,r2; Pointer to Volume Gain Table
       move          y:(r1)+,a             ; 1. Get Knob Value.
       and           #$00001F,a            ; 2. Mask for lowest 5 bits.
       move          a1,n2                 ; 3. Set index into Volume Gain Table.
       move          y:(r2+n2),r0          ; 4. Use index to get Volume Gain Value.
       move          r0,y:(r3)+            ; 5. Update 'runtime' Volume Gain Value

       cmp           #$00000F,a            ; If index=0x0-0xF, attenuate the output
       jle           vol_atten             ; If index = 0x10-0x1F, add gain
       move          #$000300,r1
       move          r0,x:CTRL_WD_LO
       move          r1,x:CTRL_WD_HI
       jmp           vgain
vol_atten
       move          #$000000,r1
       move          r0,x:CTRL_WD_HI
       move          r1,x:CTRL_WD_LO
vgain
       jsr           init_codec            ; Send Control Word to CODEC
```

The KNOB TABLE discussed earlier sets the gain values for the 10 bandpass filters and the main volume. Four spaces in memory are used in the stages.

- *KNOB_BASE*. Base of the 11-word equalizer knob value area in Y memory. The values in this memory area are ASCII values sent from the COM port on the PC. These values are the indexes to the filter and volume gain tables.
- *FILTER_GAIN_TBL*. Filter gain values in Y memory. This table is contains 32 words ranging from the values of –0.2 to 0.999 (see **Section 6**, *Coefficients and Gain Table Files*, on page 28).
- *VOLUME_GAIN_TBL*. Volume setting values in Y memory. This table is made up of 32 words. The lower 16 words contain configuration settings for the lower 16 bits of codec control data. The upper 16 words contain configuration settings for the upper 16 bits of codec control data (see **Section 6**, *Coefficients and Gain Table Files*, on page 28).
- *GAIN_BASE*. 10-word Y memory table that contains the currently selected runtime gain values. The code in **Example 10** show how this table is updated.

4 Equalizer Graphical User Interface (GUI)

A simple GUI sets the gain values for each of the bandpass filters and the main volume. This GUI runs under the Windows NT/98 OS. **Figure 12** shows the initial state of the equalizer GUI. This section discusses the GUI operation and the development of this GUI using Microsoft Visual Basic®.

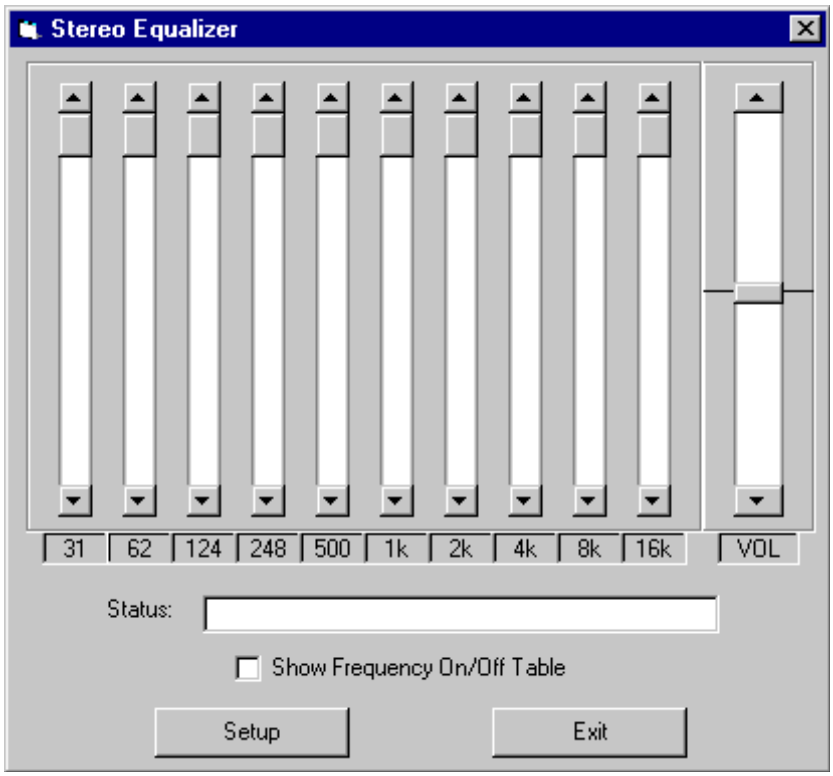

Figure 12. Equalizer Graphical User Interface

4.1 GUI Operation

The GUI interface consists of the following:

- Eleven equalizer knobs (represented as scroll bars) on the face of the GUI; 10 to set the gain values of the bandpass filters and 1 to set the main volume.
- A status line to display messages to the user. It is currently not used for anything
- Frequency table On/Off checkbox to bring up a small dialog box that allows the user to change the knob values of the 10 bandpass filters' gain. When a certain frequency is checked, the corresponding knob changes to the very top position (a gain of 1). The opposite happens when it is unchecked (see **Figure 13**).
- **Setup** and **Exit** buttons to bring up a dialog window that allows the user to change the communications port on the PC.

The bandpass filter gain ranges from 0 to 1, with 0 at the bottom position and 1 at the top. Each knob allows you to select from 1 of 16 different positions (gain values). Setting the scroll bar to the top causes the gain to be 1. Hence, the frequencies of that particular band passes through. Setting the scroll bar to the bottom causes the gain to be 0. Hence, the frequencies of that particular band is removed (or limited).

The main volume knob has 32 positions that can be selected. Sixteen of these positions gradually decrease the main volume, while 15 increase the main volume. There is one knob position that does not affect the main volume.

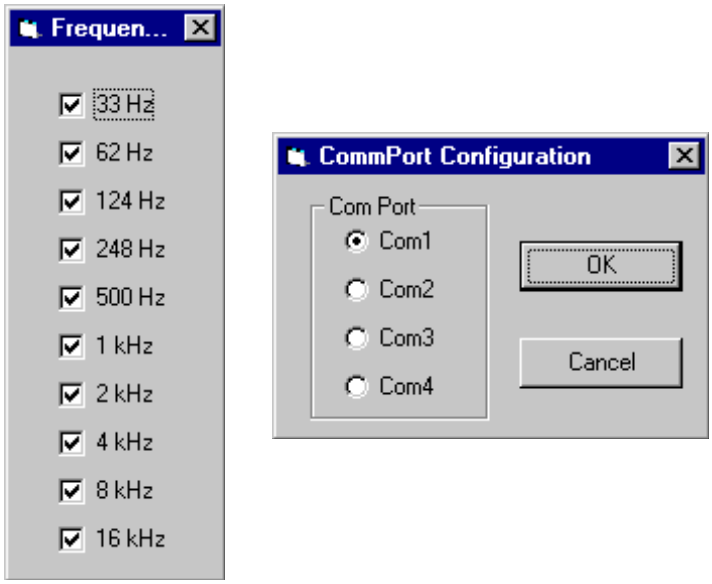

Figure 13. Frequency Table and COM Port Configuration

4.2 GUI Development

The equalizer GUI is implemented in Microsoft Visual Basic®, mainly because this very simple programming language provides good access to the communications port of a personal computer running Microsoft Windows. This section briefly describes the code written for the GUI. It is assumed that you know how to use Microsoft Visual Basic 4.0

4.2.1 Equalizer Form

The Equalizer Form is the main GUI form (see **Figure 12**). Several procedures are associated with this form, but only two are key to its functionality: Band_Change() and Send_Data(). **Example 11** shows these two procedures.

Example 11. Baud_Change() and Send_Data() Procedures

```
Private Sub Band_Change(Index As Integer)
'    BandVal(Index).Caption = Format(31 - Band(Index).Value)
    Call Send_Data
End Sub

Private Sub Send_Data()
    ' Send Gain info to DSP
    If MSComm1.PortOpen Then
        ' First, Send Reset Character
        MSComm1.Output = Chr$(13)

        For Knob = 0 To 10 Step 1
            MSComm1.Output = Chr$((31 - Band(Knob).Value) + 32)
        Next Knob
    End If
End Sub
```

The 11 knobs (scroll bars) form an object array element named *Band*. The Band_Change() procedure is called when one of the equalizer knobs changes its value. This procedures invokes the Send_Data procedure, which uses the MSComm1 object to transmit ASCII characters out of the specified communications port. The value 0x0d is sent out first. Then the position value for each knob is read and sent out, starting with the 33 Hz knob (see **Example 8** on page -12).

4.2.2 Frequency Table Form

Example 12 shows the main procedure for this form.

Example 12. Check1_Click Procedure

```
Private Sub Check1_Click(Index As Integer)
    If Check1(Index).Value = 1 Then
        Form1.Band(Index).Value = 0
    ElseIf Check1(Index).Value = 0 Then
        Form1.Band(Index).Value = 15
    End If
End Sub
```

The 10 checkboxes form an object array element named ‘Check1’. The Check1_Click() procedure changes the equalizer knob values.

4.2.3 Communications Port Settings Form

The code for the communications port settings form changes the communications port value in the MSComm1 object.

5 Using the EFCOP in Multichannel Mode

The EFCOP peripheral module functions as a general-purpose, fully programmable filter. It has optimized modes of operation to perform real and complex impulse response (FIR) filtering, infinite impulse response (IIR) filtering, adaptive FIR filtering, and multichannel FIR filtering. As **Figure 14** shows, the EFCOP comprises these main functional blocks:

- Peripheral module bus (PMB) interface, including:
  - Data input buffer
  - Constant input buffer
  - Output buffer
  - Filter counter
- Filter data memory (FDM) bank
- Filter coefficient memory (FCM) bank
- Filter multiplier accumulator (FMAC) machine
- Address generation
- Control logic

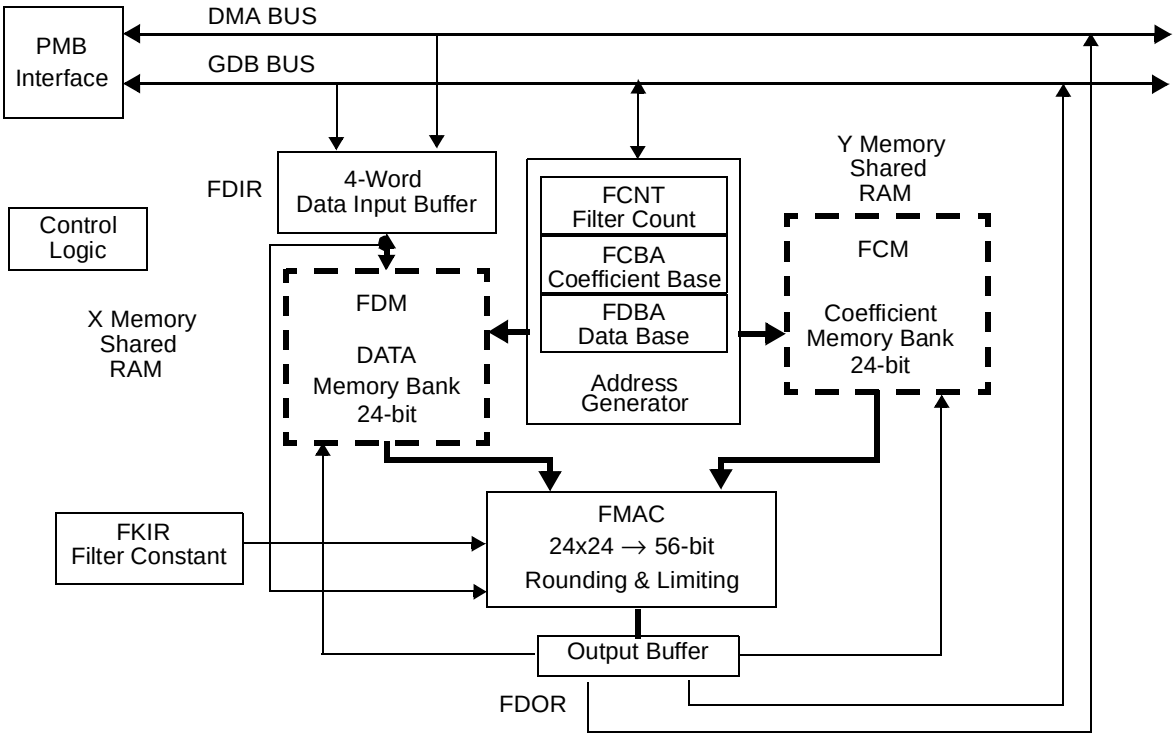

Figure 14. EFCOP Block Diagram

5.1 EFCOP Registers

This section documents the registers for configuring and operating the EFCOP (see **Table 3**). For details on these registers, consult the *DSP56311 User's Manual* (DSP56311UM/D)

Table 3. EFCOP Registers Accessible Through the PMB

Register Name	Description
Filter Data Input Register (**FDIR**)	A FIFO four words deep and 24-bits wide for DSP-to-EFCOP data transfers. Data from the FDIR is transferred to the FDM for filter processing.
Filter Data Output Register (**FDOR**)	A 24-bit wide register for EFCOP-to-DSP data transfers. Data is transferred to the FDOR after processing of all filter taps completes for a specific set of input samples.
Filter K-Constant Input Register (**FKIR**)	A 24-bit register for DSP-to-EFCOP constant transfers.
Filter Count (**FCNT**) Register	A 24-bit register that specifies the number of filter taps. The EFCOP address generation logic uses the count stored in the FCNT register to generate correct addressing to the FDM and FCM.
EFCOP Control Status Register (**FCSR**)	The DSP56300 core uses this 24-bit read/write register to program the EFCOP and to examine the status of the EFCOP module.
EFCOP ALU Control Register (**FACR**)	The DSP56300 core uses this 24-bit read/write register to program the EFCOP data ALU operating modes.
EFCOP Data Buffer Base Address (**FDBA**)	The DSP56300 core uses this 16-bit read/write register to indicate to the EFCOP the data buffer base start address pointer in FDM RAM.
EFCOP Coefficient Buffer Base Address (**FCBA**)	The DSP56300 core uses this 16-bit read/write register to indicate the EFCOP coefficient buffer base start address pointer in FCM RAM.

Table 3. EFCOP Registers Accessible Through the PMB (Continued)

Register Name	Description
Decimation/ Channel Count Register (**FDCH**)	A 24-bit register that sets the number of channels in Multichannel mode and the filter decimation ratio. The EFCOP address generation logic uses this information to supply the correct addressing to the FDM and FCM.

5.2 EFCOP Programming for Multichannel Mode

This section discusses how to program the EFCOP to process multiple channels (Multichannel mode) and shows how the filter coefficients should be set up in memory. EFCOP operation is determined by the control bits in the FCSR. Multichannel mode is selected by setting FCSR[FMLC]. The number of channels to process is one plus the number in the FDCH[FCHL] bits. Further filtering operations are enabled via the appropriate bits in the FACR. After the FCSR is configured, enable the EFCOP by setting FCSR[FEN]. To ensure proper EFCOP operation, most FCSR bits must not be changed while the EFCOP is enabled.

For each time period, the EFCOP receives the samples for each channel sequentially. This is repeated for consecutive time periods. Filtering is performed with the same filter or different filters for each channel using the FCSR[FSCO] bit. If FCSR[FSCO] is set, the same set of coefficients is used for all channels. If FSCO is clear, the coefficients for each filter are stored sequentially in memory for each channel.

5.2.1 FIR Filter Type

To select the FIR filter type, clear FCSR[FLT]. In single-channel mode, the EFCOP takes an input, *x(n)*, from the FDIR, saves the input while shifting the previous inputs down in the FDM, multiplies each input in the FDM by the corresponding coefficient, B_i, stored in the FCM, accumulates the multiplication results, and places the accumulation result, *w(n)*, in the FDOR. In Multichannel mode, the operation for FIR filtering is identical but the EFCOP takes the input *x(n – 1)* instead of *x(n)*. This is done for each sample input to the FDIR. See **Figure 15**.

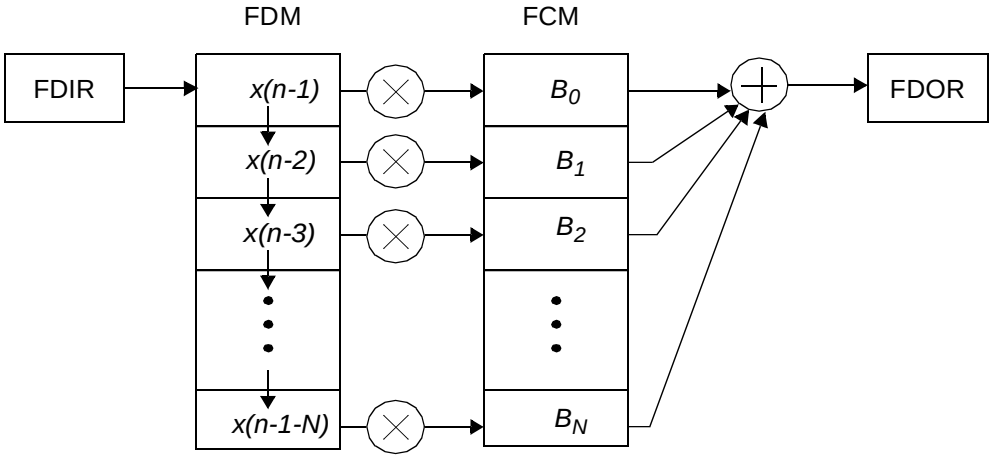

Figure 15. Multichannel FIR Filter Type Processing

5.2.2 IIR Filter Type

To select the IIR filter type, set the FCSR[FLT] bit. In Single and Multichannel modes, the EFCOP performs these steps:

1. Multiply each previous output value in the FDM by the corresponding coefficient, *A*, stored in the FCM.
2. Accumulate the multiplication results.
3. Add the input, *w(n)*, from the FDIR (which is optionally not scaled by S, depending on the FACR[FISL] bit setting).
4. Place the accumulation result, *y(n)*, in the FDOR.
5. Save the output while shifting the previous outputs down in the FDM.

This process repeats for each sample input to the FDIR. To process a complete IIR filter, a FIR filter type session followed by an IIR filter type session is needed.

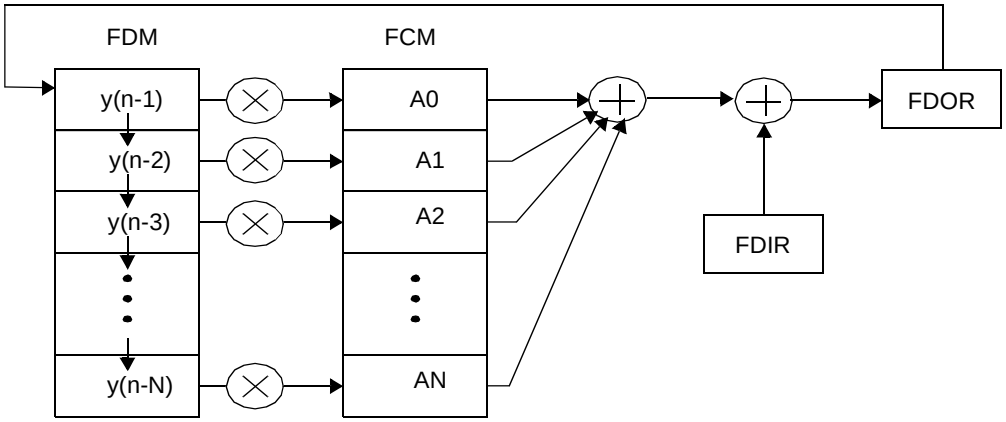

Figure 16. Multichannel IIR Filter Type Processing

5.2.3 Memory Configuration

The EFCOP uses two memory banks:

- *Filter Data Memory (FDM).* This 24-bit-wide memory bank is mapped as X memory and stores input data samples for EFCOP filter processing. The EFCOP Data Base Address (FDBA) register points to the EFCOP FDM bank.
- *Filter Coefficient Memory (FCM).* This 24-bit-wide memory bank is mapped as Y memory and stores filter coefficients for EFCOP filter processing. The EFCOP Coefficient Base Address (FCBA) register points to the EFCOP FCM bank.

The number of coefficients, M, used by each channel determines how the filter coefficients and data samples are stored in FCM and FDM, respectively. The value m = M-1 is stored in the Filter Count Register (FCNT) to select the number of filter taps that each channel will use. The base address (lower boundary) value of the FDM and FCM must have zeros in the k LSBs, where $2^k \geq M \geq 2^{k-1}$. The upper boundary is equal to the lower boundary plus (M –1). Since $M \leq 2^k$, once M is chosen (that is, FCNT[11–0] is assigned), a sequential series of data memory blocks (each of length 2^k) is created where multiple circular buffers for multichannel filtering can be located. If $M < 2^k$, there is a space between sequential circular buffers of $2^k - M$ (see **Figure 17**).

The data samples, D(n) are stored in each circular buffer of the FDM starting at the lower addresses. The EFCOP manages placement of sample data into FDM. The filter coefficients are stored in "reverse order," where H(N –1) is stored in each circular buffer of the FCM starting at the lower addresses. These values must be set up in Y memory before the EFCOP is enabled.

Figure 17 shows an example EFCOP memory configuration. The EFCOP is set in Multichannel mode. There are two filter channels and each channel has three coefficients. Before the EFCOP is enabled, the FCM must be initialized. The coefficients for the first channel are stored in reverse order starting from the FCM base address (0x0). Since each filter has three coefficients (k = 2), the coefficients for the next channel start at 0x4. After the EFCOP is enabled and initialized, the sample data is sent to the Filter Data Input Register (FDIR). The EFCOP transfers that data to the FDM. The EFCOP does not touch the 0x3 and 0x7 positions in FDM and FCM.

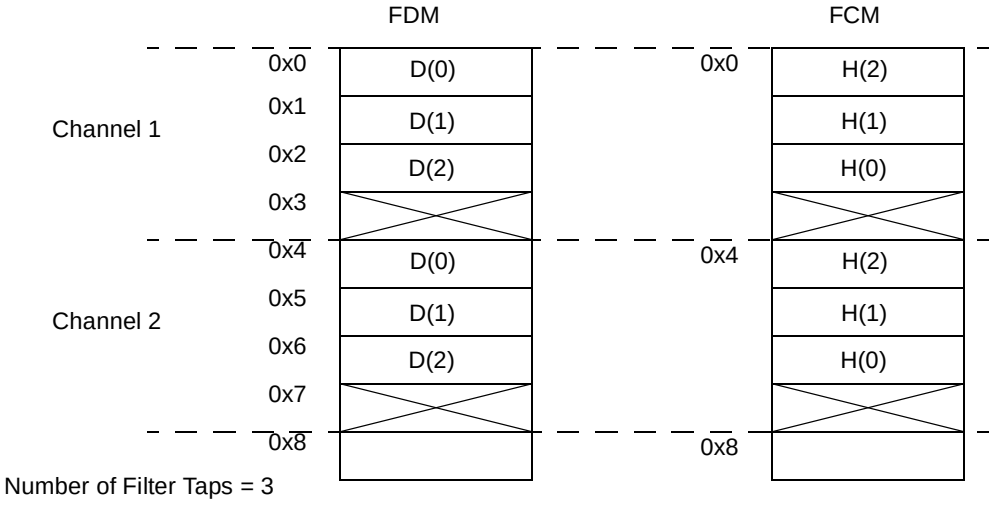

Figure 17. Memory Configuration Example

6 Coefficients and Gain Table Files

This section lists the coefficients used in both 10-band stereo equalizer implementations. It also lists the filter and volume gain tables.

Example 13. DSP56311 Core Implementation FIR and IIR Coefficients

```
;************************************************************************
;       COEFF.ASM
;       Digital Stereo 10-band Graphic Equalizer Using the 56311
;
;       Copyright (c) Freescale Semiconductor, 2000
;
;************************************************************************
************************************************************************
; IIR Coefficients for each of the 10 Bands
;************************************************************************
31 Hz
               DC              .49855285                       ;beta
               DC              .000723575                      ;alpha
               DC              .998544628                      ;gamma
; 62 Hz
               DC              .497109876
```

```
                DC              .001445062
                DC              .997077038
; 125 Hz
                DC              .494190149
                DC              .002904926
                DC              .994057064
; 250 Hz
                DC              .488447026
                DC              .005776487
                DC              .987917799
; 500 Hz
                DC              .477154897
                DC              .011422552
                DC              .975062733
; 1000 Hz
                DC              .455306941
                DC              .02234653
                DC              .947134157
; 2000 Hz
                DC              .414266319
                DC              .04286684
                DC              .88311345
; 4000 Hz
                DC              .340894228
                DC              .079552886
                DC              .728235763
; 8000 Hz
                DC              .2601072
                DC              .1199464
                DC              .3176087
; 16000 Hz
                DC              .1800994
                DC              .159603
                DC              -.4435172
```

Example 14. EFCOP and DMA Implementation FIR Coefficients

```
;************************************************************************
;       FIR_COEFF.ASM
;       Digital Stereo 10-band Graphic Equalizer Using the 56311
;
;       Copyright (c) Freescale Semiconductor, 2000
;
;************************************************************************

************************************************************************
; FIR Coefficients for each of the 10 Bands
;************************************************************************

 31 Hz
                DC              -.000723575*2           ; A2 = -alpha
                DC              $000000                 ; A1 = 0
                DC              .000723575*2            ; A0 = alpha
                DC              $000000                 ; Added space to line up Coeffs in memory
; 62 Hz
```

```
                DC              -.001445062*2
                DC              $000000
                DC              .001445062*2
                DC              $000000
; 125 Hz
                DC              -.002904926*2
                DC              $000000
                DC              .002904926*2
                DC              $000000
; 250 Hz
                DC              -.005776487*2
                DC              $000000
                DC              .005776487*2
                DC              $000000
; 500 Hz
                DC              -.011422552*2
                DC              $000000
                DC              .011422552*2
                DC              $000000
; 1000 Hz
                DC              -.02234653*2
                DC              $000000
                DC              .02234653*2
                DC              $000000
; 2000 Hz
                DC              -.04286684*2
                DC              $000000
                DC              .04286684*2
                DC              $000000
; 4000 Hz
                DC              -.079552886*2
                DC              $000000
                DC              .079552886*2
                DC              $000000
; 8000 Hz
                DC              -.1199464*2
                DC              $000000
                DC              .1199464*2
                DC              $000000
; 16000 Hz
                DC              -.159603*2
                DC              $000000
                DC              .159603*2
                DC              $000000
```

Example 15. EFCOP and DMA Implementation IIR Coefficients

```
;*************************************************************************
;       IIR_COEFF.ASM
;       Digital Stereo 10-band Graphic Equalizer Using the 56311
;
;       Copyright (c) Freescale Semiconductor, 2000
;
;*************************************************************************

*************************************************************************
```

```
; IIR Coefficients for each of the 10 Bands
;*****************************************************************************

 31 Hz
              DC            -.49855285/4;B2 = -beta
              DC            .998544628/4;B1 = gamma
; 62 Hz
              DC            -.497109876/4
              DC            .997077038/4
; 125 Hz
              DC            -.494190149/4
              DC            .994057064/4
; 250 Hz
              DC            -.488447026/4
              DC            .987917799/4
; 500 Hz
              DC            -.477154897/4
              DC            .975062733/4
; 1000 Hz
              DC            -.455306941/4
              DC            .947134157/4
; 2000 Hz
              DC            -.414266319/4
              DC            .88311345/4
; 4000 Hz
              DC            -.340894228/4
              DC            .728235763/4
; 8000 Hz
              DC            -.2601072/4
              DC            .3176087/4
; 16000 Hz
              DC            -.1800994/4
              DC            -.4435172/4
```

Example 16. Filter Gain Table

```
;*****************************************************************************
;       Filter_Gain.ASM
;       Digital Stereo 10-band Graphic Equalizer Using the 56311
;
;       Copyright (c) Freescale Semiconductor, 2000
;
;*****************************************************************************

*****************************************************************************
; Filter Gain (G) Coefficients
;*****************************************************************************

              DC            -0.200
              DC            -0.187
              DC            -0.171
              DC            -0.160
              DC            -0.150
              DC            -0.137
              DC            -0.114
              DC            -0.103
```

```
                DC              -0.092
                DC              -0.080
                DC              -0.067
                DC              -0.051
                DC              -0.039
                DC              -0.027
                DC              -0.015
                DC              0.000

                DC              0.000
                DC              0.030
                DC              0.060
                DC              0.090
                DC              0.120
                DC              0.150
                DC              0.180
                DC              0.210

                DC              0.250
                DC              0.290
                DC              0.340
                DC              0.380
                DC              0.460
                DC              0.540
                DC              0.750
                DC              0.999
```

Example 17. Volume Gain Table

```
;*************************************************************************
;       Volume_Gain.ASM
;       Digital Stereo 10-band Graphic Equalizer Using the 56311
;
;       Copyright (c) Freescale Semiconductor, 2000
;
;*************************************************************************

;*************************************************************************
; Volume Gain (V) Coefficients
;*************************************************************************

                DC              $1FFB00
                DC              $1CE300
                DC              $1AD300
                DC              $18C300
                DC              $16B300
                DC              $14A300
                DC              $129300
                DC              $108300

                DC              $0E7300
                DC              $0C6300
                DC              $0A5300
                DC              $084300
                DC              $063300
```

```
        DC      $042300
        DC      $021300
        DC      $000300

        DC      $000000
        DC      $110000
        DC      $220000
        DC      $330000
        DC      $440000
        DC      $550000
        DC      $660000
        DC      $770000

        DC      $880000
        DC      $990000
        DC      $AA0000
        DC      $BB0000
        DC      $CC0000
        DC      $DD0000
        DC      $EE0000
        DC      $FF0000
```

7 References

This application note refers to the following Freescale documents, which are available at the web site listed on the back cover of this document:

- *DSP56311 User's Manual*, DSP56311UM.
- DSP56311 technical data sheet, DSP56303.
- *Programming the CS4218 CODEC for use with DSP56300 Devices*, AN1790.

NOTES:

NOTES:

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Order No.: AN2110
Rev. 1
11/2005

freescale™
semiconductor