Application Report
SLAA193 – February 2004

Efficient Resampling Filters for the AIC111

Sunil Kamath, Bharath Siravara, Pedro Gelabert, Neeraj Magotra High Performance Analog Group

ABSTRACT

The AIC111 is an ultralow-power, fixed sample rate, sigma-delta codec with a sample rate of 40 kHz and a usable bandwidth of 10 kHz. Typically, most applications process data at sample rates of 8 kHz, 16 kHz, and 20 kHz. This requires that the DSP or microprocessor connected to the AIC111 decimate the incoming data from 40 kHz down to the required sampling frequency, and interpolate the data back up to 40 kHz. This document outlines several methods and examples of resampling incoming data from the AIC111 at 40 kHz to the required processing sampling rate between 8 kHz and 20 kHz using decimation and interpolation techniques on a digital signal processor (DSP) or a microprocessor. In many situations, it may also be possible to exploit characteristics of supplementary signal processing stages in the system to relax filter specifications or even remove certain resampling stages for additional savings in computation. The document also provides some suggestions on ways to exploit these characteristics. The document also addresses how to use the H-bridge output on the AIC111 to drive speakers with varying impedance and bandwidth. Example routines have been developed specifically for use with the AIC111 audio codec and the Texas Instruments (TI) TMS320C54x™ (C54x™) DSP for speech/audio applications.

Contents

Figures

1 Introduction

The TI AIC111 is a low power DSP compatible voice band codec specifically designed for devices with stringent power consumption constraints such as hearing aids or headset applications. The AIC111 has a fixed sampling rate of 40-kilo samples per second (ksps) with a distortion free bandwidth of 10 kHz, as shown in Figure 1. The ADC on the AIC111 converts the incoming audio signal into 16-bit digital data. The serial interface on the AIC111 outputs a 32-bit data word consisting of the 16-bit audio data and a 16-bit control word at 40 ksps. Hence, it is necessary to decimate the data down to a maximum of 20 kHz to get a flat input spectrum. Additionally, most voice band applications work within the speech/audio bandwidth of 4 kHz to 10 kHz, and hence it is necessary to support a variable sample rate between 8 kHz and 20 kHz.

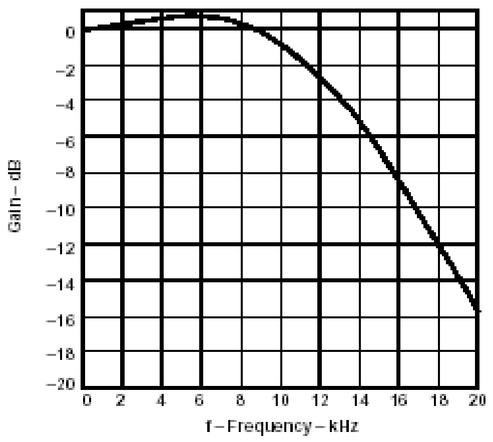


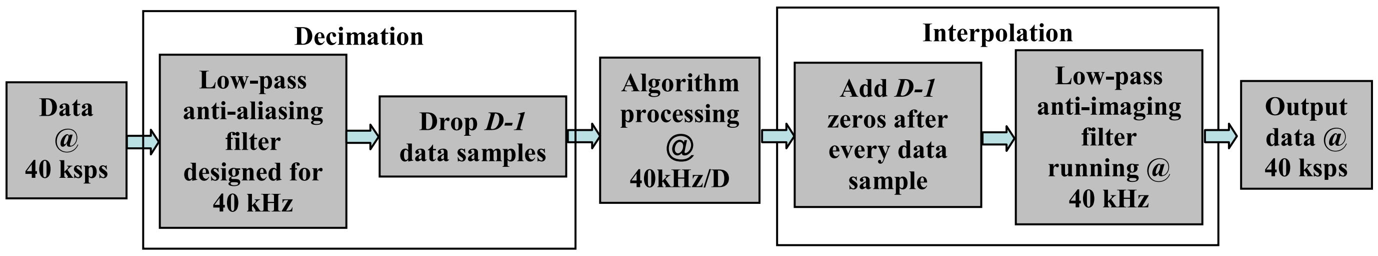
Figure 1. AIC111 Input Channel Frequency Response With HPF Bypassed

Correspondingly, the data supplied to the DAC on the AIC111 is a 32-bit word consisting of 16 or 20 bits of audio data and 16/12 bits of control. The AIC111 expects a fixed output rate of 40 ksps. Therefore, the data sent to the AIC111 needs to be packed with the audio and control bits and upsampled to 40 ksps. This process of decimation and interpolation needs to be performed on the DSP interfaced to the AIC111.

The complete resampling process involves decimating the data received from the codec to the appropriate processing rate and interpolating the processed data to the fixed AIC111 rate before outputting to the codec. Appendix C has a more detailed explanation on the theory behind the resampling process. Figure 2 shows the block diagram of the various steps necessary to interface the AIC111 with a device processing the data at lower sampling rates. This application note assumes that the resampled frequency is an integer divisor of 40 kHz. Resampling to frequencies that are not an integral ratio of 40 kHz is a nontrivial effort that requires a sample rate converter and is beyond the scope of this document.

Section 2 summarizes several classical DSP techniques for designing decimation and interpolation filters.

Section 3 describes AIC111-specific resampling subsystems for implementation on a DSP or a microprocessor. This description shows the downsampling and upsampling for an AIC111 interfaced to a TMS320C54x serial port. Appendix B shows the example C54x DSP. The development platform is comprised of the AIC111 and the DSP-codec EVM interfaced to the TMS320C5416 (C5416) DSP starter kit (DSK) along with Code Composer Studio™ (CCS).


Decimation
Interpolation
Data @ 40 ksps
Low-pass anti-aliasing filter designed for 40 kHz
Drop D-1 data samples
Algorithm processing @ 40kHz/D
Add D-1 zeros after every data sample
Low-pass anti-imaging filter running @ 40 kHz
Output data @ 40 ksps

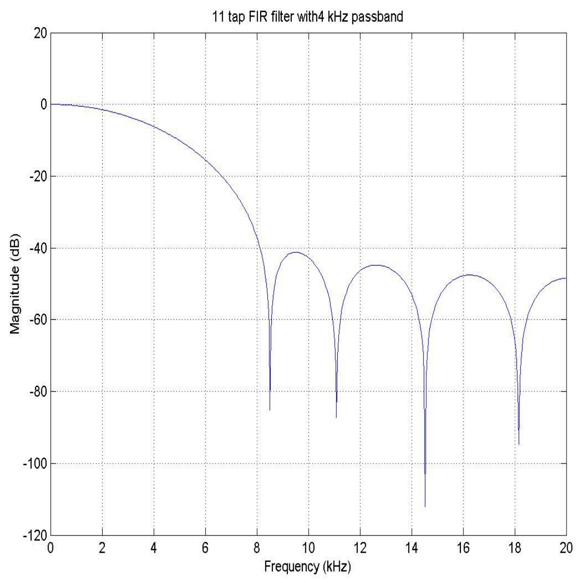
D = 40 kHz / required sampling frequency
(e.g., for 8-kHz sampling, D = 40/8 = 5)

Figure 2. Block Diagram of the Resampling Process

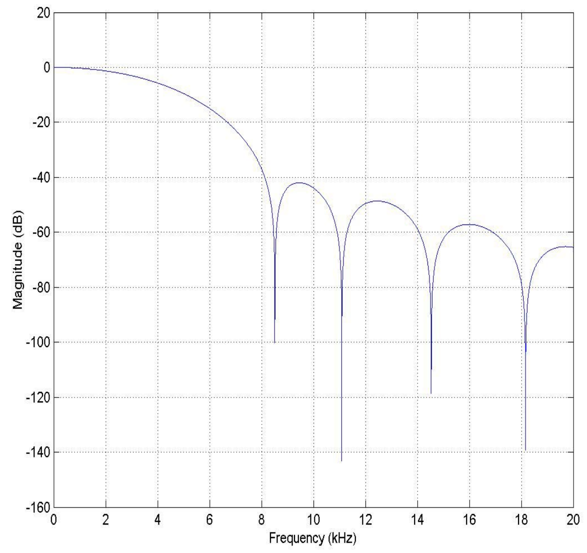
Furthermore, the AIC111 has an integrated headphone amplifier driven by an H-bridge speaker driver. This H-bridge speaker driver introduces noise above the 10-kHz frequency, reference [10]. In many cases, this high-frequency noise component is filtered out by the speaker/receiver itself. However, if the bandwidth of the speaker is greater than 10 kHz, an external analog filter can be used to filter out the high-frequency noise. Section 4 describes in detail the H-bridge output driver and gives examples of external analog filters that can be used.

2 Decimation/Interpolation Filters

In a fixed-precision environment like the C54x DSP, the correct choice for the type and structure of the decimation/interpolation filters is crucial. Choosing a filter implementation that is appropriate for the targeted application helps ensure proper operation at minimum cost with respect to the computation and memory storage required for the processing. Figure 3 through Figure 9 show FIR filters with different orders and passband frequencies. Appendix A details the floating point and the quantized filter coefficients for these filters. See references [4,5,6,7,8,9] for further information on designing FIR, IIR, and halfband filters.

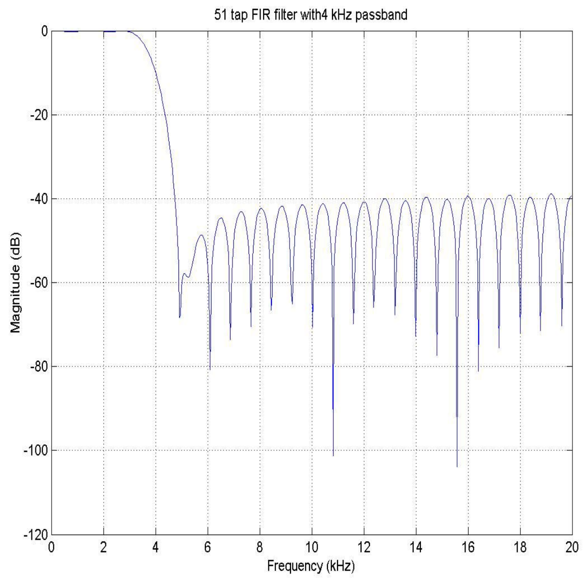


(a) Magnitude Response of FIR Filter

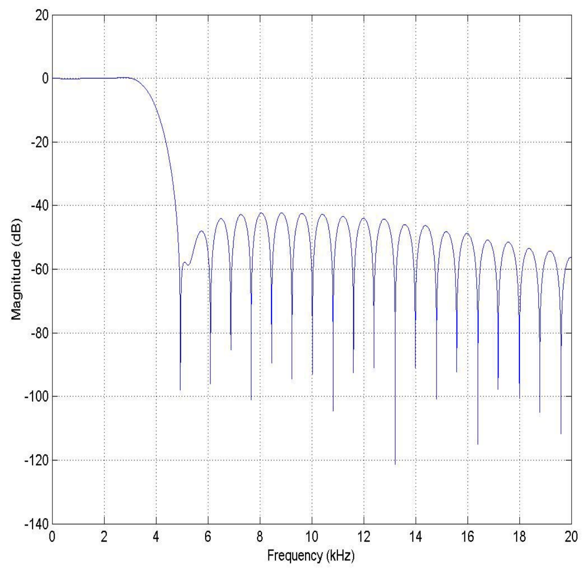


(b) Combined Magnitude Response of FIR Filter and ADC Channel

Figure 3. 11-Tap FIR Filter With 4-kHz Passband

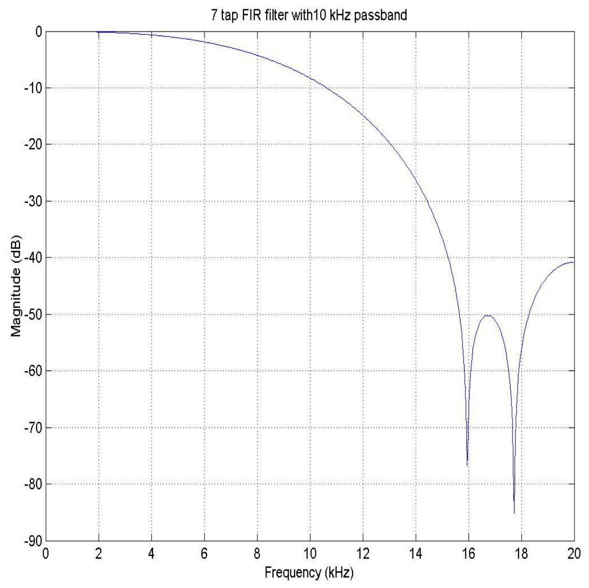


(a) Magnitude Response of FIR Filter

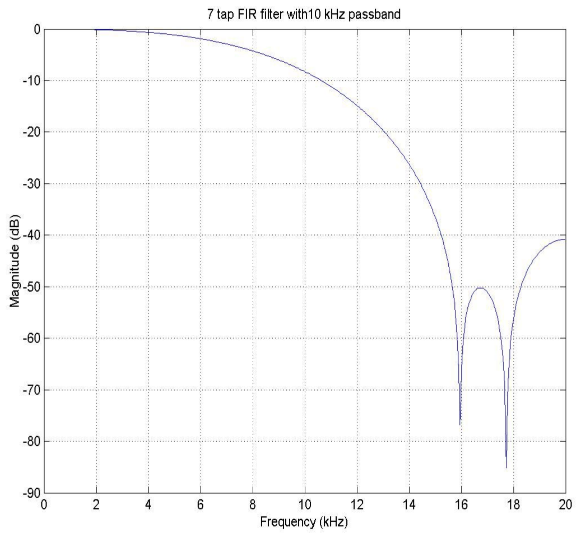


(b) Combined Magnitude Response of FIR Filter and ADC Channel

Figure 4. 51-Tap FIR Filter With 4-kHz Passband

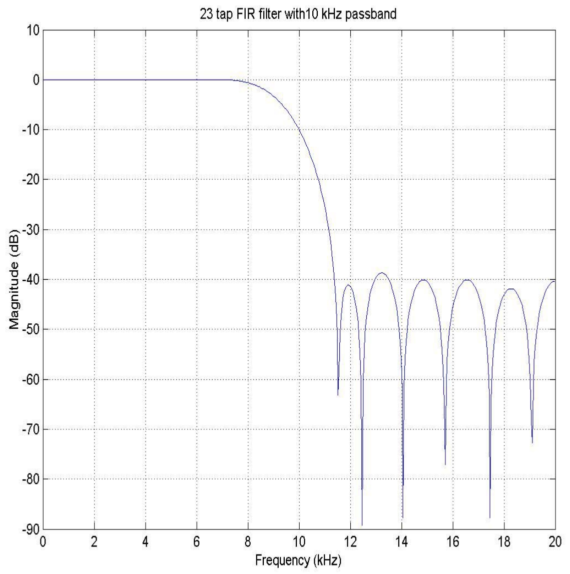


(a) Magnitude Response of FIR Filter

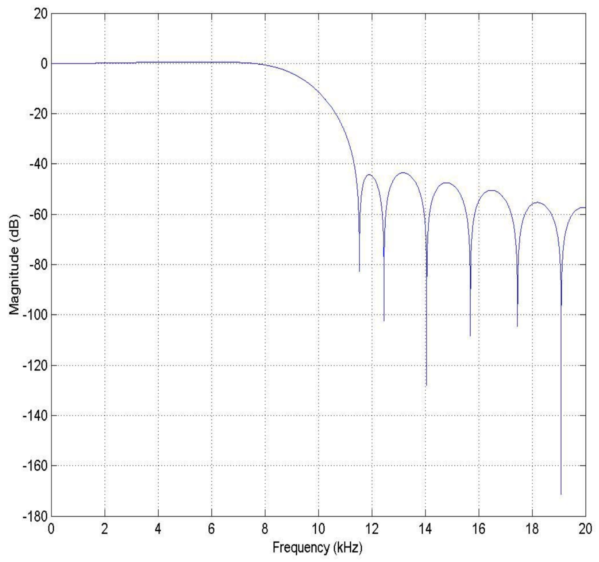


(b) Combined Magnitude Response of FIR Filter and ADC Channel

Figure 5. 7-Tap FIR Filter With 10-kHz Passband

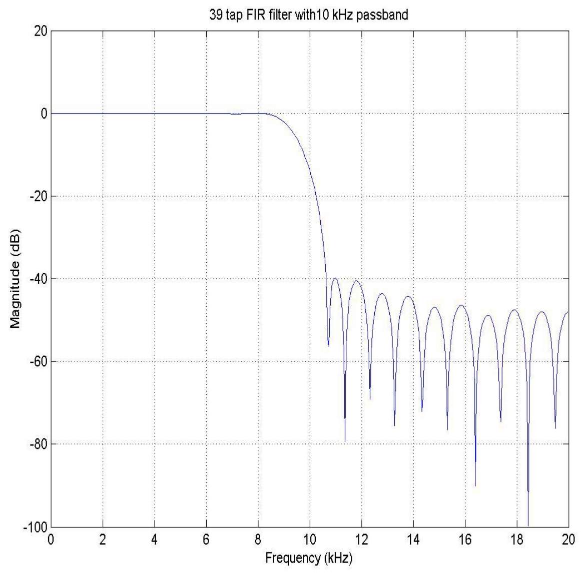


(a) Magnitude Response of FIR Filter

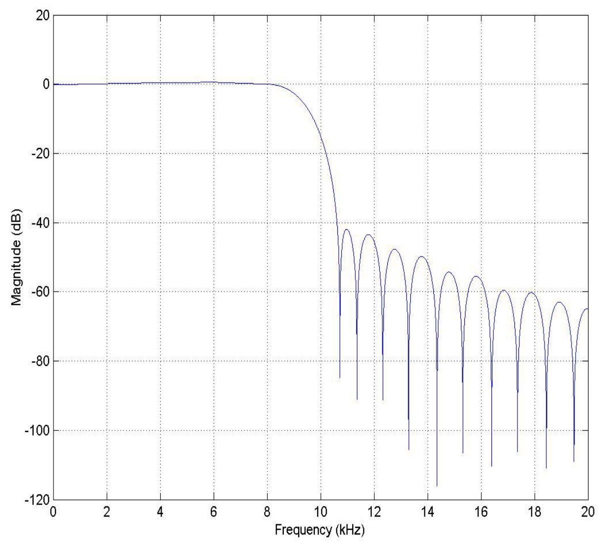


(b) Combined Magnitude Response of FIR Filter and ADC Channel

Figure 6. 23-Tap FIR Filter With 10-kHz Passband

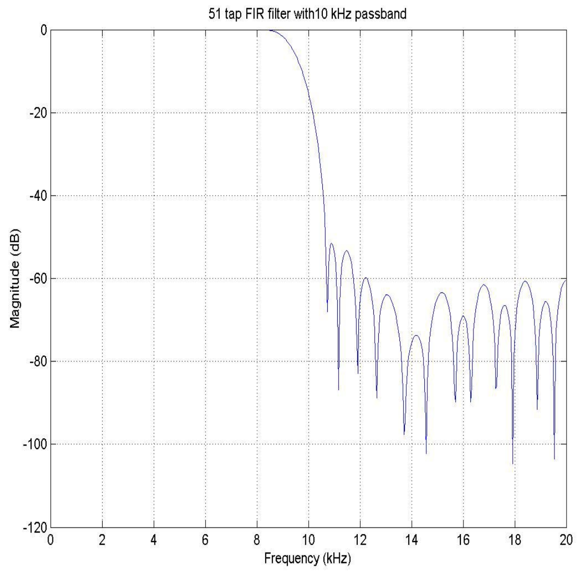


(a) Magnitude Response of FIR Filter

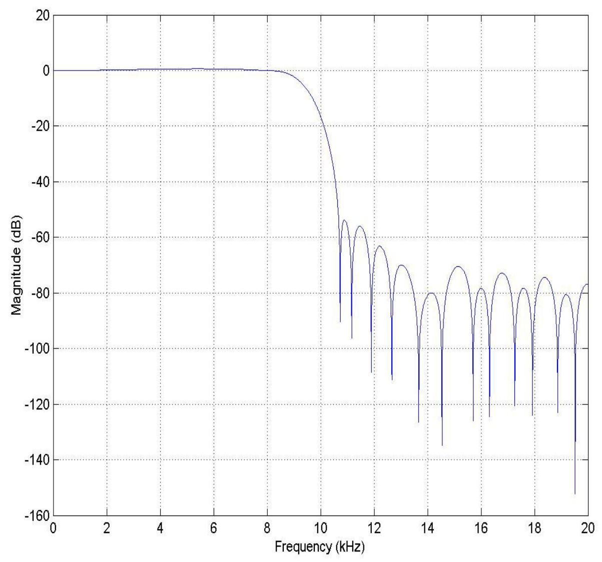


(b) Combined Magnitude Response of FIR Filter and ADC Channel

Figure 7. 39-Tap FIR Filter With 10-kHz Passband

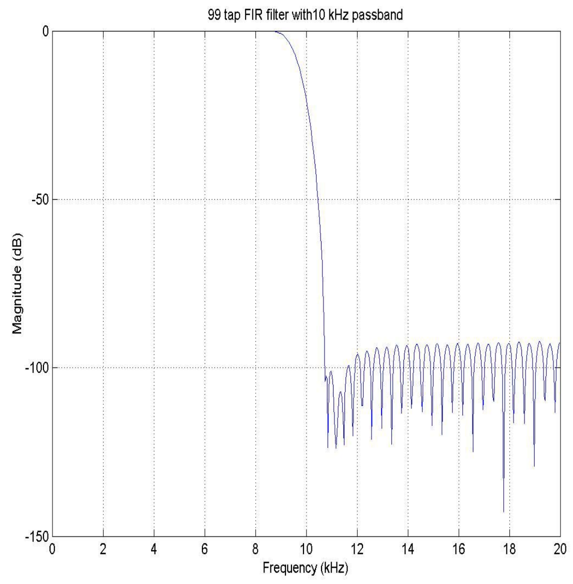


(a) Magnitude Response of FIR Filter

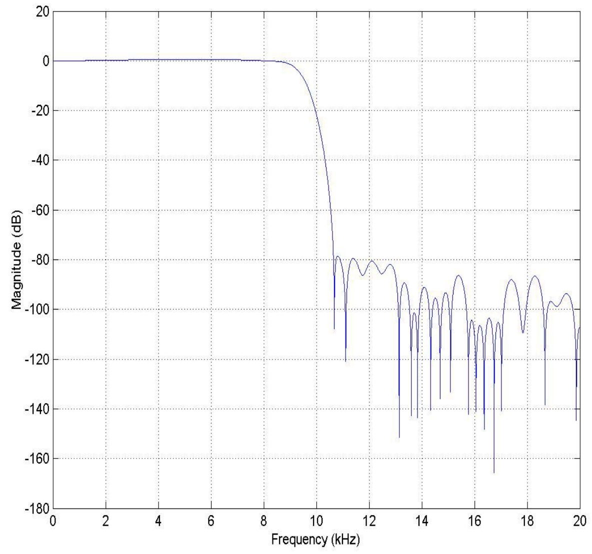


(b) Combined Magnitude Response of FIR Filter and ADC Channel

Figure 8. 51-Tap FIR Filter With 10-kHz Passband



(a) Magnitude Response of FIR filter



(b) Combined magnitude response of FIR filter and ADC channel

Figure 9. 99-Tap FIR Filter With 10-kHz Passband

3 Efficient Implementation of the Resampling Process

This section illustrates methods to efficiently implement the resampling subsystem by exploiting the transfer function characteristics of other signal processing subcomponents.

After the decimation filter, *D-1* samples are discarded. This downsampling process can be incorporated into an FIR decimation filter by only computing every D^{th} output sample. This technique saves computation required for filtering *F*(D-1)/D* samples every second for a saving of approximately *(D-1)/D %* instead of running the filter at full rate. This is equivalent to having a reduced data rate in the first place.

However, it is necessary that every input sample has to be stored in the filter feedforward delay buffer. Failing to store these values is equivalent to dropping input samples before band-limiting the data. This would result in taking frequency content above the Nyquist frequency and aliasing into the baseband, and thus introducing noise into the system.

Unlike FIR filters, IIR filters have a feedback loop where previous outputs are convolved with the feedback coefficients to compute later output samples. Because every output sample has to be computed, the filter cannot be run at the reduced sampling rate.

Note that the interpolation filter is always run at the full sampling rate (40 kHz). For lower decimation factors, e.g., 2 or 3, the sampling overhead of the interpolation filter can be reduced by using a polyphase implementation, reference [7], or by using the method in Section 3.3.

Appendix B includes the C54x DSP code for the implementation of these filters.

3.1 Integrating the Decimation Filter Into Subsequent Algorithm Processing

Under some conditions, it is possible to incorporate the decimation task into another filtering operation and avoid this overhead. In this case, the subsequent filter coefficients are recalculated for the original sampling frequency instead of the lower sampling frequency. However, this might require a higher filter order, but the filter is run at the reduced sample rate. For example, the next few paragraphs depict this technique in a hearing aid application.

In a hearing aid application, the main task of the algorithm shapes the speech spectrum to compensate for the user's hearing loss profile. The shaped spectrum is basically the inverse of the hearing profile, reference [11]. This algorithm essentially implements a specially designed filter. Normally, the filter is designed for a sampling frequency of 8 kHz or 20 kHz and corresponding bandwith of 4 kHz or 10 kHz, respectively.

To incorporate the decimation filter into the spectral shaping filter, the spectral shaping filter is designed for a sampling frequency of 40 kHz while maintaining the original (desired) cutoff. The stopband now includes frequencies from the cutoff to 20 kHz. If the algorithm is run at the reduced rate as described in Section 5.1, no additional processing is required for the decimation process.

3.2 Eliminating the Interpolation Filter

Figure 10 depicts the block diagram of a DSP-based hearing aid/headset device equipped with the AIC111 codec. The figure also illustrates the signal condition (analog or digital), passband bandwidth (*fp*) and sampling rate (*F*) at different points in the system.

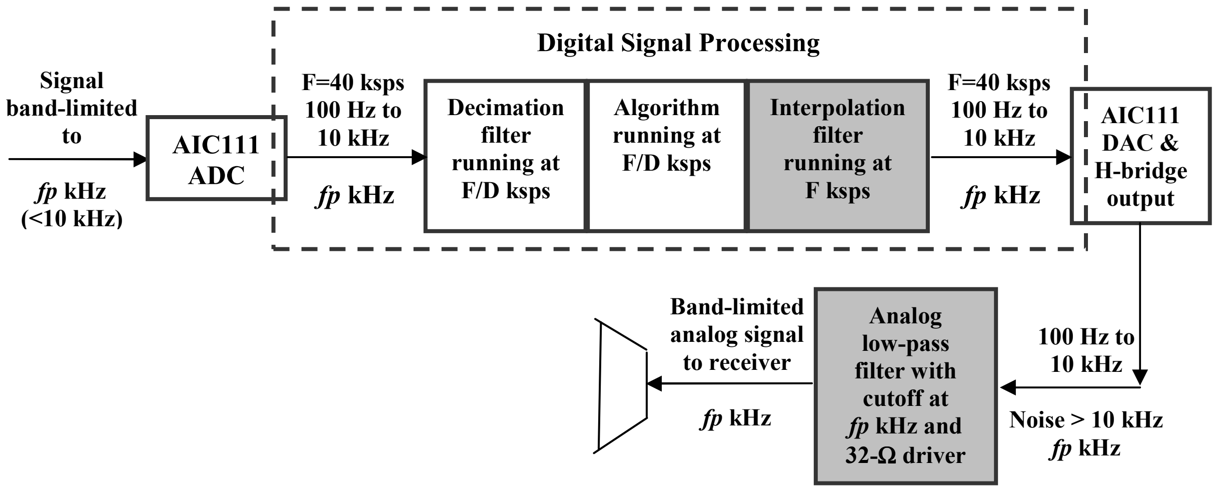

Digital Signal Processing
Signal band-limited to
fp kHz (<10 kHz)
AIC111 ADC
F=40 ksps 100 Hz to 10 kHz
fp kHz
Decimation filter running at F/D ksps
Algorithm running at F/D ksps
Interpolation filter running at F ksps
F=40 ksps 100 Hz to 10 kHz
fp kHz
AIC111 DAC & H-bridge output
100 Hz to 10 kHz
Noise > 10 kHz
fp kHz
Analog low-pass filter with cutoff at *fp* kHz and 32-Ω driver
Band-limited analog signal to receiver
fp kHz

Figure 10. Block Diagram of an AIC111-Equipped Hearing Aid/Headset Device

As mentioned in Section 1, the AIC111 outputs data samples at a rate of 40 ksps to the DSP with a bandwidth of 100 Hz to 10 kHz. The signal to the AIC111 is assumed to be band-limited to *fp* kHz. Downsampling data from the AIC111 by a factor of *D* to *F/D* ksps drastically reduces the computational requirements of subsequent signal processing algorithms.

The resampling subsystem normally includes an interpolation filter (shown as a shaded block) as the anti-imaging filter, described in Section 4.2. The filter cutoff is around *fp* kHz. Even though the speech bandwidth is limited to *fp* kHz, the H-bridge output contains noise above 10 kHz as mentioned in Section 1. Low-pass filtering to extract the desired analog signal can be done in either of the following ways,

1. The speaker/receiver with a suitable response could act as the low-pass filter with a cutoff at 10 kHz or lower as required. These types of speaker/receivers are typically found in hearing aid devices and can be directly driven by the AIC111 output H-bridge.
2. Headset applications usually use 32-Ω or 16-Ω receivers/speakers that support bandwidth around 16-kHz. Hence, these types of speakers do not normally filter the higher frequency image components introduced by the resampling process, nor the DAC-induced high-frequency noise. To remove the noise components, an analog filter (shown as shaded block) is employed to low-pass filter the H-bridge output to cut off at the desired frequency. Section 4 describes the H-bridge output driver and the output analog filter along with example filters that have 4 kHz and 10 kHz cutoffs.

Under these conditions, either the receiver or the external analog filter can also double as the anti-imaging filter rendering the interpolation filter redundant. The remaining *D-1* output samples can be zero-filled or duplicated from the previous output value, even though this might result in image components in the higher frequencies reproduced by the speaker. Therefore, the receiver or analog filter must be designed appropriately to attenuate these high-frequency components.

The filter specification for the decimation filter can be relaxed depending on the frequency response of the microphone or input data. For example, if the microphone input has a frequency range of only 4 kHz, the decimation filter can have a gradual rolloff starting at 4 kHz. Relaxing the filter specification reduces the order of the filter and the number of computations required.

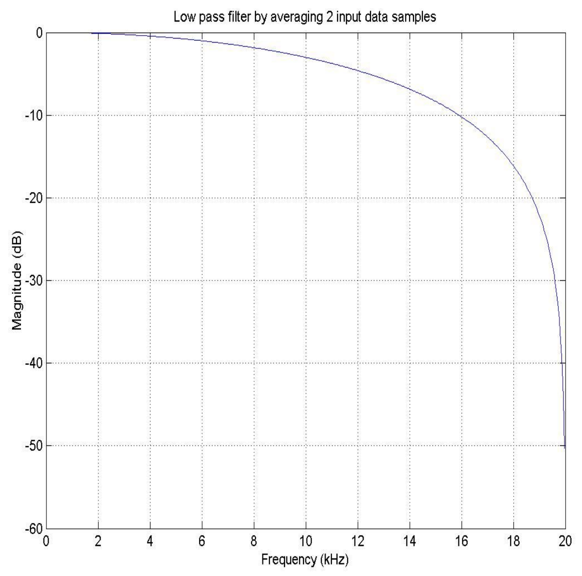
3.3 Reducing the Number of DSP Interrupts by Using Direct Memory Access (DMA)

The DSP is interfaced to the AIC111 via the multichannel-buffered serial port (McBSP). The McBSP is usually configured to interrupt the DSP on every sample received from the AIC111. However, the signal processing algorithms only process every D^{th} sample. Using the DMA reduces the number of CPU interrupts.

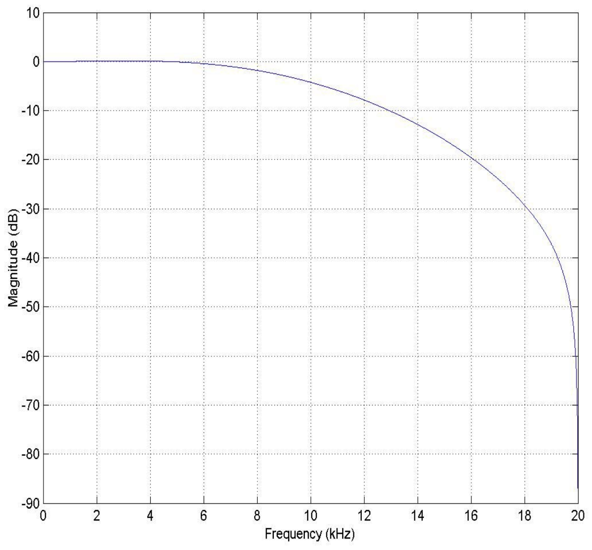
The McBSP can be configured for the DMA to interrupt the DSP after acquiring *D* input samples from the codec. This minimizes computations spent in context, saving and executing the interrupt service routine (ISR) for every input sample.

3.4 Decimation by Dropping Samples

In typical voice band applications, the microphone limits the bandwidth of the input signal to the AIC111. For these band-limited inputs with oversampled data, the DSP can decimate the data by discarding samples, or preferably by averaging the data samples. While this method results in some degradation of the processed speech/audio signal, empirical data shows that the distortion is hardly perceptible to the human ear for small decimation by factors (i.e., resampling to a sampling frequency of 20 kHz or 13.33 kHz). Figure 11 illustrates the low-pass filtering effect of averaging 2 samples, and Figure 12 illustrates the effect of averaging 5 samples. As we can see from Figure 13, averaging 2 samples results in a low-pass filter with a very gradual rolloff around 10 kHz. However, averaging a large number of samples to decimate down to lower sampling frequencies such as 8 kHz results in significant aliasing which distorts the quality of the signal in the desired frequency range. Therefore, using this technique to decimate by larger factors should be avoided.

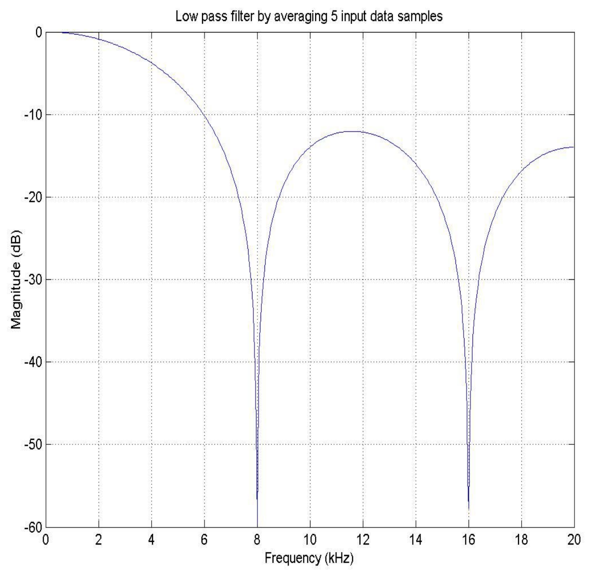


(a) Magnitude Response of FIR Filter

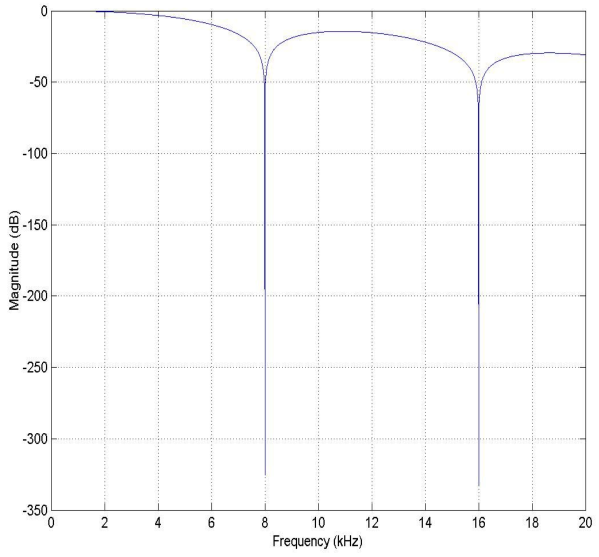


(b) Combined Magnitude Response of FIR Filter and ADC Channel

Figure 11. 10-kHz Low-Pass Filter by Averaging 2 Samples



(a) Magnitude Response of FIR Filter



(b) Combined Magnitude Response of FIR Filter and ADC Channel

Figure 12. 4-kHz Low-Pass Filter by Averaging 5 Samples

4 Output Analog Filter

The AIC111 has a 32-times oversampled three-level (-1, 0, 1) delta-sigma DAC that drives an H-bridge output driver. Typically, a two-level output is used since it is ideally linear. However, this limits the performance of the delta-sigma modulator. To increase the inband performance, a three-level output is used. This implies that the speakers need to be connected differentially across the output of the H-bridge. Connecting the speakers single-endedly halves the amplitude of the signal going into the speaker. Additionally, in the adaptive quantization mode, a pulse width modulation scheme is used which yields a further 3 dB in SNR. The H-bridge output driver uses a clock that is 4 times the DAC clock. Hence, there are 4 clock pulses for every output bit. The adaptive quantization mode takes advantage of this to vary the width of the output pulse between one and four clock pulses depending on the amplitude of the signal. All of these factors ensure excellent inband performance resulting in 90 dB of dynamic range with a THD of 0.05%. However, these factors also result in significant out-of-band noise as illustrated by Figure 13.

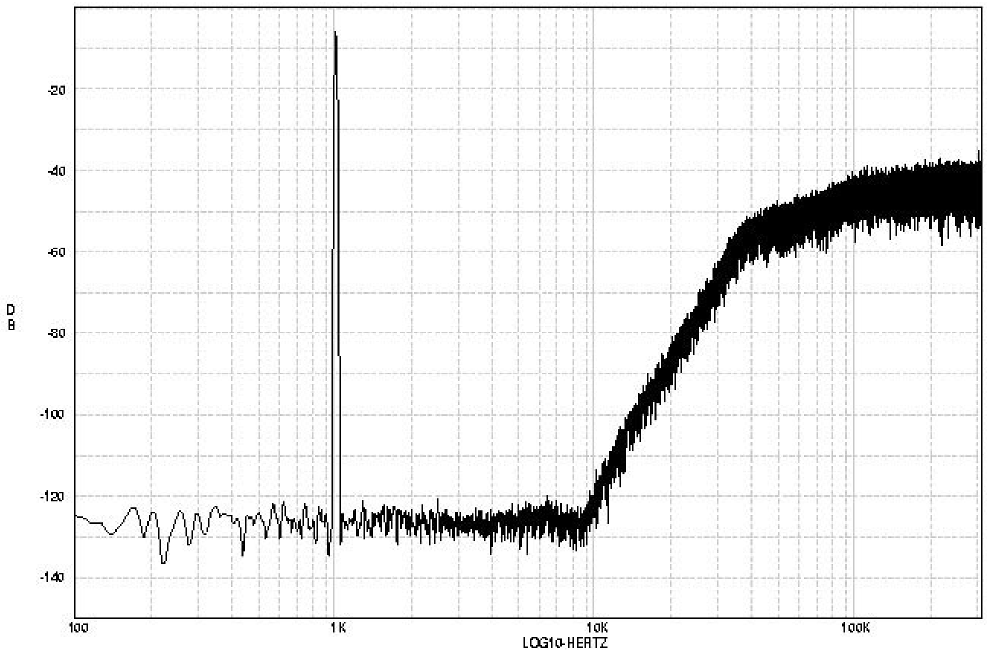


Figure 13. PSD of DAC Output for a 1-kHz Sine Wave.

The out-of-band noise above 10 kHz needs to be filtered out by either the speaker/receiver or by using an output analog low-pass filter. Receivers used in hearing aids tend to have a high input impedance in the order of 120 Ω, and also have a sharp rolloff around 10 kHz. Thus, the out-of-band noise of the AIC111 has no influence on such a system. Commercially available speakers used in headset applications have lower impedance on the order of 16–32 Ω and also tend to have a bandwidth between 16–22 kHz. Using such a speaker directly with the H-bridge results in the load across the H-bridge being much lower than expected, leading to suboptimal performance. Additionally, some of the high-frequency noise generated by the H-bridge is now audible as it falls in the upper regions of the bandwidth of the speaker. This scenario can be easily rectified by connecting a passive RLC filter across the H-bridge output and connecting the speakers differentially across the output of the filter. The inductor in the filter blocks the high-frequency noise, and the load connected to the H-bridge is now the sum of the impedance of the filter and the speakers. This higher impedance value is thus more optimal, being closely matched to the H-bridge output driver impedance. Note that it is also feasible to connect an RC filter across the H-bridge. However, in such a scenario, the RC filter does not block the high-frequency noise, but dissipates it across the resistance and capacitance. This results in slightly higher power consumption. Figure 14 shows an example analog filter with a passband centered around 4 kHz.

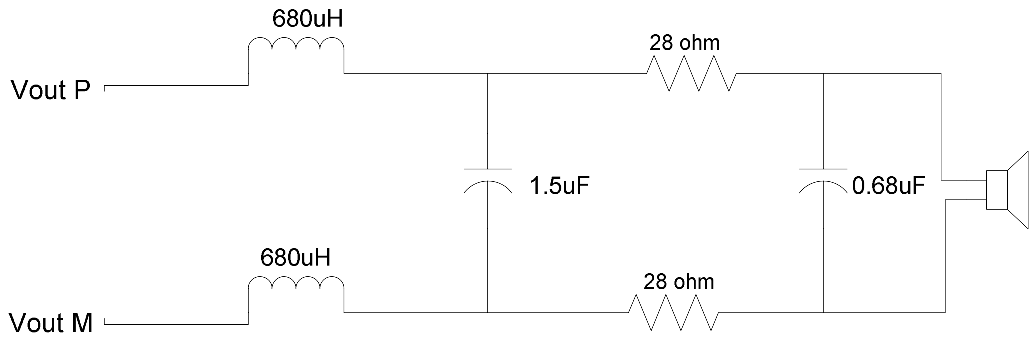


Figure 14. Third-Order Output Filter With 4-kHz Passband

5 References

1. *TMS320C54x DSP CPU and Peripherals Reference Set Volume 1* (SPRU131)
2. *AIC111 IC Design Specification, 1.3-V microPower DSP/µC Voice Band Audio Codec* (SLAS382)
3. TMS320C54x, LC54x, VC54x Fixed-Point Digital Signal Processors (SPRS039)
4. D. F. Elliott, *Handbook of Digital Signal Processing Engineering Applications*, Academic Press, 1987.
5. D. J. DeFatta, et al., *Digital Signal Processing: A Systems Design Approach*, John Wiley & Sons, 1988.
6. A. V. Oppenheim, R. W. Schafer, *Digital Signal Processing*, Prentice Hall, 1975.
7. P. P. Vaidyanathan, *Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial*, Proceedings of the IEEE, vol.78, Iss.1, pp. 56-93, Jan 1990.
8. L. Gazsi, *Explicit Formulas for Lattice Wave Digital Filters*, IEEE Trans. Circuits & Systems, vol. CAS-32, no. 1, Jan 1985.
9. A. Fettweis, et al., *Wave Digital Lattice Filters*, Int. J. Circuit Theory Appl., vol. 2, no. 2, pp. 203-211, June 1974.
10. G. Gata, et al., *A 1.1–V 270-µA Mixed-Signal Hearing Aid Chip*, IEEE J. Solid-State Circuits, vol. 37, No.12, Dec 2002.
11. T. Stetzler, et al., *Low Power Real-Time Programmable DSP Development Platform For Digital Hearing Aids*, ICASSP, Vol.4, pp.:2339-2342, Mar 1999.

Appendix A. Low-Pass Filter Coefficients

A.1 Resampled frequency = 8 kHz

A.1.1 Number of taps = 11

Floating Point	Quantized - S16Q15
0.00136649	45
0.02295194	752
0.06755080	2214
0.12730742	4172
0.18014170	5903
0.20136331	6598
0.18014170	5903
0.12730742	4172
0.06755080	2214
0.02295194	752
0.00136649	45

A.1.2 Number of taps = 51

Floating Point	Quantized - S16Q15
0.00608580	199
0.00249113	82
0.00341018	112
0.00223568	73
-0.00015719	-5
-0.00349888	-115
-0.00646796	-212
-0.00782592	-256
-0.00617263	-202
-0.00134929	-44
0.00565379	185
0.01249695	410
0.01636109	536
0.01454560	477
0.00605594	198
-0.00776348	-254
-0.02298622	-753
-0.03399716	-1114
-0.03465491	-1136
-0.02057063	-674
0.00930934	305
0.05186317	1699
0.10022042	3284
0.14480221	4745
0.17607795	5770
0.18767002	6150
0.17607795	5770
0.14480221	4745
0.10022042	3284
0.05186317	1699
0.00930934	305
-0.02057063	-674
-0.03465491	-1136
-0.03399716	-1114
-0.02298622	-753
-0.00776348	-254
0.00605594	198
0.01454560	477
0.01636109	536
0.01249695	410
0.00565379	185
-0.00134929	-44
-0.00617263	-202
-0.00782592	-256
-0.00646796	-212
-0.00349888	-115
-0.00015719	-5
0.00223568	73
0.00341018	112
0.00249113	82
0.00608580	199

A.2 Resampled Frequency = 20 kHz

A.2.1 Number of taps = 7

Floating Point	Quantized - S16Q15
-0.02816184	-923
0.02975524	975
0.27588835	9040
0.44503650	14583
0.27588835	9040
0.02975524	975
-0.02816184	-923

A.2.2 Number of taps = 23

Floating Point	Quantized - S16Q15
-0.00534764	-175
0.00630256	207
0.01017296	333
-0.01096614	-359
-0.02191395	-718
0.01596771	523
0.04419754	1448
-0.02046218	-671
-0.09293975	-3045
0.02365750	775
0.31151339	10208
0.47193872	15464
0.31151339	10208
0.02365750	775
-0.09293975	-3045
-0.02046218	-671
0.04419754	1448
0.01596771	523
-0.02191395	-718
-0.01096614	-359
0.01017296	333
0.00630256	207
-0.00534764	-175

A.2.3 Number of taps = 39

Floating Point	Quantized - S16Q15
-0.00074808	-25
0.00396001	130
0.00120830	40
-0.00629853	-206
-0.00316842	-104
0.00902645	296
0.00639472	210
-0.01191950	-391
-0.01151998	-377
0.01508148	494
0.01946143	638
-0.01807934	-592
-0.03176816	-1041
0.02077693	681
0.05298754	1736
-0.02291997	-751
-0.09863958	-3232
0.02425235	795
0.31290754	10253
0.47043692	15415
0.31290754	10253
0.02425235	795
-0.09863958	-3232
-0.02291997	-751
0.05298754	1736
0.02077693	681
-0.03176816	-1041
-0.01807934	-592
0.01946143	638
0.01508148	494
-0.01151998	-377
-0.01191950	-391
0.00639472	210
0.00902645	296
-0.00316842	-104
-0.00629853	-206
0.00120830	40
0.00396001	130
-0.00074808	-25

Appendix B. C54x DSP Implementation

This section contains the source code listing for AIC111 loopback on a C5416 DSK. To build and execute the code, create a project and add all of the files listed below to the project. Also include the Chip Support Library (CSL – csl5416.lib) and the run time support library (rts.lib). Sample routines to implement an FIR and an IIR filter from TI's DSPLIB are also provided. Please refer to the C54x DSPLIB User's Guide (SPRU518) for more information on these routines as well as other implementations.

B.1 Loopback Code

```
/////////////////////////////////////////////////////////////////
//
// Loopback code to intitialize the AIC111 Codec
// Author: Bharath Siravara
// Date: 05/14/03
// Texas Instruments Inc.
// File Name: Loopback.c
// Carries out loopback. No external parameters.
//
/////////////////////////////////////////////////////////////////

#include "loopback.h"

MCBSP_Handle hMcbsp1;

Uint32 temp, data_out;
Uint16 data_in, control_set, control_word;

short fir_coefs[NUM_COEFS] = {-923, 975, 9040, 14583, 9040,975,-923,};

void main()
{
int i;
asm("            NOP");           //probe point for spectral shaping coefficients
asm("            NOP");
//////////////////////////////////////////////////////
// Initialize the Chip Support Library (CSL)
// This is a onetime only initialization of the CSL
// that must be done before calling any CSL module API.
//////////////////////////////////////////////////////
CSL_init();
//////////////////////////////////////////////////////
//  Disable all interrupts
//////////////////////////////////////////////////////
IRQ_globalDisable();
//////////////////////////////////////////////////////////////////
//  The C5416 DSK has a 16MHz crystal.
//  Set up the PLL so as to set the CPU clock to 160MHz.
//  Since the McBSP will be set up in slav mode, it will use
//  the CPU clock to synchronize with the external FS(R/X) and
//  CLK(R/X). Setting the CPU clock low can derail the synchronization
//  Use the PLL module of the CSL to configure the PLL appropriately.
//////////////////////////////////////////////////////////////////
PLL_configArgs(PLL_MODE_MUL, 9, 100, 0);
//////////////////////////////////////////////////////
//  Set the base address of the interrupt vector table
//  IRQ_setVecs() will left shift the value that is passed
//  into the function by 7. This is a bug.
//  As a temporary work around, 0x00FF is passed to the function
//  which when left shifted by 7 becomes 0x7F80 which
//  is the desired value.
//////////////////////////////////////////////////////
i = IRQ_setVecs(0x00FF);
//////////////////////////////////////////////////////
//  Open and Configure the McBSP port
//////////////////////////////////////////////////////
mcbsp_cfg();
IRQ_globalEnable();
init_aic111();
while ( 1 )
{
i++;
}
} /* End of main*/
```

```
/////////////////////////////////////////////////////////////////
//
// Initialize McBSP1 of the C5416 DSP.
// Uses the TI Chip Support Library (CSL)
// Author: Bharath Siravara
// Date: 05/14/03
// Texas Instruments Inc.
// File Name: mcbsp_cfg.c
//
/////////////////////////////////////////////////////////////////
#include "loopback.h"
extern MCBSP_Handle hMcbsp1;
//////////////////////////////////////////////////////////
//  Assign and Fill up the config structure to
//  Configure the McBSP0 peripheral.
//////////////////////////////////////////////////////////
MCBSP_Config hMcbsp1_cfg = {
                 0x0020,//SPCR1
                 //0001000000100000b
                 //0~~~~~~~~~~~~~~~ DLB            0 = IDigital LoopBack mode disabled
                 //~00~~~~~~~~~~~~~ RJUST          00= right justify and zero fill MSBs in DRR[1,2]
                 //~~~00~~~~~~~~~~~ CLKSTP         0x= Clock Stop mode disabled
                 //~~~~~000~~~~~~~~ Reserved
                 //~~~~~~~~0~~~~~~~ DXENA          0 = Transmit is Disabled
                 //~~~~~~~~~0~~~~~~ ABIS           0 = A-bis mode is disabled
                 //~~~~~~~~~~10~~~~ RINTM          10= RINT driven by Frame Sync
                 //~~~~~~~~~~~~0~~~ RSYNCERR 0= No Synchronization error
                 //~~~~~~~~~~~~~0~~ RFULL          0 = RBR is not in overrun condition
                 //~~~~~~~~~~~~~~0~ RRDY           0 = Receiver is not ready
                 //~~~~~~~~~~~~~~~0 RRST!          0 = Serial Port Receiver is disabled and in reset

                 0x0020,          //SPCR2
                 //0000000000100000
                 //000000~~~~~~~~~~ Reserved
                 //~~~~~~0~~~~~~~~~ FREE           0 = Free Running Mode is disabled
                 //~~~~~~~0~~~~~~~~ SOFT           0 = Soft Mode is disabled
                 //~~~~~~~~0~~~~~~~ FRST!          0 = Frame Synchronization logic is reset
                 //~~~~~~~~~0~~~~~~ GRST!          0 = Sample Rate Generator is reset
                 //~~~~~~~~~~10~~~~ XINTM          10= XINT generated by Frame Sync.
                 //~~~~~~~~~~~~0~~~ XSYNCERR 0= Transmit Synchronization error
                 //~~~~~~~~~~~~~0~~ XEMPTY         0 = Transmit shift register XSR is empty
                 //~~~~~~~~~~~~~~0~ XRDY           0 = Transmitter is not ready
                 //~~~~~~~~~~~~~~~0 XRST!          0 = Serial port transmitter disabled and in reset state

                 0x00A0,          //RCR1
                 //0000000010100000
                 //0~~~~~~~~~~~~~~~                Reserved
                 //~0000000~~~~~~~~                RFRLEN1          000 0000 = 1 word per frame
                 //~~~~~~~~101~~~~~                RWDLEN1          101 = 32 bits
                 //~~~~~~~~~~~00000                Reserved

                 0x0000,          //RCR2
                 //0000000000000100
                 //0~~~~~~~~~~~~~~~                RPHASE 0 = Single-phase frame
                 //~0000000~~~~~~~~                RFRLEN2          000 0000 = 1 word per frame
                 //~~~~~~~~000~~~~~                RWDLEN1          000 = 8 bits
                 //~~~~~~~~~~~00~~~                RCOMPAND 00 = No Companding
                 //~~~~~~~~~~~~~0~~                RFIG 0 = Receive Frame Sync after first resets transfer
                 //~~~~~~~~~~~~~~00                RDATDLY 00 = 0-bit delay

                 0x00A0,          //XCR1
                 //0000000010100000
                 //0~~~~~~~~~~~~~~~                Reserved
                 //~0000000~~~~~~~~                XFRLEN1          000 0000 = 1 word per frame
                 //~~~~~~~~101~~~~~                XWDLEN1          101 = 32 bits
                 //~~~~~~~~~~~00000                Reserved

                 0x0000, //XCR2
                 //0000000000000100
                 //0~~~~~~~~~~~~~~~                XPHASE 0 = Single-phase frame
                 //~0000000~~~~~~~~                XFRLEN2          000 0000 = 1 word per frame
                 //~~~~~~~~000~~~~~                XWDLEN1          000 = 8 bits
```

```
                 //~~~~~~~~~~~00~~~                XCOMPAND 00 = No Companding
                 //~~~~~~~~~~~~~0~~                XFIG 0 = Transmit frame sync after first resets transfer
                 //~~~~~~~~~~~~~~00                XDATDLY 00 = 0-bit delay

                 0x0000,          //SRGR1
                 0x0000,          //SRGR2
                 0x0000,          //MCR1
                 0x0000,          //MCR2
                 0x0002,          //PCR
                 //0000000000000010
                 //00~~~~~~~~~~~~~~ Reserved
                 //~~0~~~~~~~~~~~~~ XIOEN          0 = DX, FSX and CLKX are configured as serial port pins
                 //~~~0~~~~~~~~~~~~ RIOEN          0 = DR, FSR and CLKR are configured as serial port pins
                 //~~~~0~~~~~~~~~~~ FSXM 0 = Transmit Frame Synchronization signal from an external source
                 //~~~~~0~~~~~~~~~~ FSRM 0 = Receive Frame Synchronization signal from an external source
                 //~~~~~~0~~~~~~~~~ CLKXM          0 = Transmitter Clock driven from an external source
                 //~~~~~~~0~~~~~~~~ CLKRM   0 = Receiver Clock driven from an external source
                 //~~~~~~~~0~~~~~~~ Reserved
                 //~~~~~~~~~0~~~~~~ CLKS_STAT 0 = CLKS pin status
                 //~~~~~~~~~~0~~~~~ DX_STAT        0 = DX pin status
                 //~~~~~~~~~~~0~~~~ DR_STAT        0 = DR pin status
                 //~~~~~~~~~~~~0~~~ FSXP           0 = Frame Synchronization pulse FSX is active high
                 //~~~~~~~~~~~~~0~~ FSRP           0 = Frame Synchronization pulse FSR is active high
                 //~~~~~~~~~~~~~~1~ CLKXP          1 = Trasnmit data sampled on falling edge of CLKX
                 //~~~~~~~~~~~~~~~0 CLKRP          0 = Receive data sampled on falling edge of CLKR
};               // End MCBSP_Config structure


void mcbsp_cfg()
{
                 /////////////////////////////////////////////////////////
                 //  Open and reset the McBSP0 peripheral. This call will open the
                 //  port and assign the handle to hMcbsp1 defined previously
                 /////////////////////////////////////////////////////////
                 hMcbsp1 = MCBSP_open(MCBSP_PORT1, MCBSP_OPEN_RESET);
                 MCBSP_reset(hMcbsp1);
                 /////////////////////////////////////////////////////////
                 //  Setup the MCBSP using the configuration structure
                 //  prefiously defined
                 /////////////////////////////////////////////////////////
                 MCBSP_config(hMcbsp1, &hMcbsp1_cfg);
                 MCBSP_start(hMcbsp1, MCBSP_XMIT_START | MCBSP_RCV_START , 0x200);
                 MCBSP_start(hMcbsp1, MCBSP_SRGR_START | MCBSP_SRGR_FRAMESYNC , 0x200);
                 IRQ_clear(IRQ_EVT_RINT1);
                 IRQ_enable(IRQ_EVT_RINT1);
}
```

```
///////////////////////////////////////////////////////////////
//
// Initialize AIC111 registers
// Author: Bharath Siravara
// Date: 05/14/03
// Texas Instruments Inc.
// File Name: init_aic111.c
//
///////////////////////////////////////////////////////////////
#include "loopback.h"
extern Uint16 control_set, control_word;
void init_aic111()
{
                 /////////////////////////////////////////////////////////
                 //  FORMAT0
                 /////////////////////////////////////////////////////////
                 control_word =   0x0846;
                 //0000100001000110b;
                 //0000~~~~~~~~~~~~                D3-D0 - These bits are set to appropriate values in the ISR
                 //~~~~1~~~~~~~~~~~                R/W              1 = WRITE
                 //~~~~~000~~~~~~~~                A2-A0            001 = FORMAT0 register address
                 //~~~~~~~~0~~~~~~~                PGAC_READ_MODE 0 = Read Format0 register contents
                 //~~~~~~~~~1000110                PGAC_GAIN 0x46 = 34dB (default)
                 control_set = 1;
                 while(control_set) {  };
                 /////////////////////////////////////////////////////////
                 //  FORMAT1
                 /////////////////////////////////////////////////////////
                 control_word =   0x0A01;
                 D3-D0 - These bits are set to appropriate values in the ISR
                 R/W              1 = WRITE
                 A2-A0            002 = FORMAT1 register address
```

```
                 DBUF_EN 0 = disable weak digital I/O buffer
                 HPF_CTL - Control bits for highpass filter 00 = normal mode
                 SHIFT - 000 = No Shift
                 DAC_MODE - 16 bit input goes through shifter
                 control_set = 1;
                 while(control_set) {  };
                 //////////////////////////////////////////////////////////
                 //  FORMAT2
                 //////////////////////////////////////////////////////////
                 control_word =   0x0B40;
                 D3-D0 - These bits are set to appropriate values in the ISR
                 //~~~~1~~~~~~~~~~~                R/W              1 = WRITE
                 //~~~~~011~~~~~~~~                A2-A0            003 = FORMAT2 register address
                 //~~~~~~~~0~~~~~~~                DAC_ADAPTIVE_Q 0 = Fixed Quantization
                 //~~~~~~~~~1~~~~~~                HB_OUT_EN 1 = H-Bridge output Enable
                 //~~~~~~~~~~0~~~~~                HB_DRIVE 0 = 40ohm
                 //~~~~~~~~~~~0~~~~                HIST_TIMEOUT_SEL 0 = 50ms PGAC Hysterisis Timeout
                 //~~~~~~~~~~~~00~~                PGAC_GAIN_MODE 00 = Automatic, Dual Rate
                 //~~~~~~~~~~~~~~0~                MIC_VSUP_PD 0 = Power Down MIC_VSUP
                 //~~~~~~~~~~~~~~~0                FRONTEND_PD 0 = Power Down PGAC + ADC
                 control_set = 1;
                 while(control_set) {  };
                 //////////////////////////////////////////////////////////
                 //  FORMAT3
                 //////////////////////////////////////////////////////////
                 control_word =   0x0CF7;
                 //0000110011110111b;
                 //0000~~~~~~~~~~~~                D3-D0 - These bits are set to appropriate values in the ISR
                 //~~~~1~~~~~~~~~~~                R/W              1 = WRITE
                 //~~~~~100~~~~~~~~                A2-A0            004 = FORMAT3 register address
                 //~~~~~~~~1111~~~~                ATTACK - 1111 = Attack Rate = 80000dB/s
                 //~~~~~~~~~~~~0111                RELEASE - 0111 = ?
                 control_set = 1;
                 while(control_set) {  };
                 //////////////////////////////////////////////////////////
                 //  FORMAT4
                 //////////////////////////////////////////////////////////
                 control_word =   0x0D42;
                 //0000110101000010b;
                 //0000~~~~~~~~~~~~                D3-D0 - These bits are set to appropriate values in the ISR
                 //~~~~1~~~~~~~~~~~                R/W              1 = WRITE
                 //~~~~~101~~~~~~~~                A2-A0            005 = FORMAT4 register address
                 //~~~~~~~~0100~~~~                ATTACK - 0100 = ?
                 //~~~~~~~~~~~~0010                RELEASE - 0010 = ?
                 control_set = 1;
                 while(control_set) {  };
}
```

```
/////////////////////////////////////////////////////////////////
//
// Interrupt handler for the McBSP Receive intterupt.
// Uses the TI Chip Support Library (CSL)
// Author: Bharath Siravara
// Date: 05/14/03
// Texas Instruments Inc.
// File Name: aic111_interrupt.c
//
/////////////////////////////////////////////////////////////////
#include "loopback.h"
extern MCBSP_Handle hMcbsp1;
extern Uint32 temp, data_out;
extern Uint16 data_in, control_set, control_word;
extern short fir_coefs[NUM_COEFS];
/********************
* Transmit Interrupt*
********************/
interrupt
void c_int22( void )
{

                 short *dbuf[NUM_COEFS];
                 short temp_out;
    temp = MCBSP_read32(hMcbsp1);
    temp = temp>>16;
    data_in = temp;
    //Add call to low pass filter here
    //Decimate data from 40kHz to required sampling frequency.
    data_out = (Uint32)temp_out<<16;
```

```
    if (control_set) /* check to see if register r/w is pending */
    {
        data_out = data_out | (Uint32)control_word;
        control_set = 0;
    }
    MCBSP_write32(hMcbsp1, data_out);
}
```

```
************************************************************************
** Interrupt Vector table
** File Name: C5416Vec.asm
************************************************************************
** Include Statements
************************************************************************
      .global    _c_int00
      .global    _c_int22
      .sect   ".vectors"
************************************************************************
**  Unmaskable Interrupts
************************************************************************
RESET:      BD    _c_int00        ; HW/SW RESET vector
            NOP
            NOP
NMI:        RETE                  ; ~NMI, Non-Maskable interrupt
            NOP
            NOP
            NOP
************************************************************************
**  S/W Interrupts
************************************************************************
SINT17:     RETE
            NOP
            NOP
            NOP
SINT18:     RETE
            NOP
            NOP
            NOP
SINT19:     RETE
            NOP
            NOP
            NOP
SINT20:     RETE
            NOP
            NOP
            NOP
SINT21:     RETE
            NOP
            NOP
            NOP
SINT22:     RETE
            NOP
            NOP
            NOP
SINT23:     RETE
            NOP
            NOP
            NOP
SINT24:     RETE
            NOP
            NOP
            NOP
SINT25:     RETE
            NOP
            NOP
            NOP
SINT26:     RETE
            NOP
            NOP
            NOP
SINT27:     RETE
            NOP
            NOP
            NOP
SINT28:     RETE
            NOP
            NOP
            NOP
SINT29:     RETE
```

```
            NOP
            NOP
            NOP
SINT30:     RETE
            NOP
            NOP
            NOP
************************************************************************
**   Rest of the Interrupts
************************************************************************
INT0:       RETE                 ; external user interrupt #0
            NOP
            NOP
            NOP
INT1:       RETE                 ; external user interrupt #1
            NOP
            NOP
            NOP
INT2:       RETE                 ; external user interrupt #2
            NOP
            NOP
            NOP
TINT0:      RETE                 ; Timer0 interrupt
            NOP
            NOP
            NOP
BRINT0:     RETE                 ; McBSP#0 receive interrupt
            NOP
            NOP
            NOP
BXINT0:     RETE                 ; McBSP#0 transmit interrupt
            NOP
            NOP
            NOP
DMAC0:      RETE                 ; DMA channel0 interrupt
            NOP
            NOP
            NOP
TINT1:      RETE                 ; Timer1 interrupt
            NOP
            NOP
            NOP
INT3:       RETE                 ; external user interrupt #3
            NOP
            NOP
            NOP
HPINT:      RETE                 ; HPI interrupt
            NOP
            NOP
            NOP
BRINT1:     BD    _c_int22         ; McBSP#1 receive interrupt
            NOP
            NOP
BXINT1:     RETE                 ; McBSP#1 transmit interrupt
            NOP
            NOP
DMAC4:      RETE                 ; DMA channel4 interrupt
            NOP
            NOP
            NOP
DMAC5:      RETE                 ; DMA channel5 interrupt
            NOP
            NOP
            NOP
        .end
************************************************************************
**  End of File -- C5402Vec.asm
************************************************************************
```

```
/////////////////////////////////////////////////////////////////
//
// CMD file for Loopback project
// Author: Bharath Siravara
// Date: 05/14/03
// Texas Instruments Inc.
// File Name: C5416.cmd
//
/////////////////////////////////////////////////////////////////
```

```
MEMORY
{
    PAGE 0: DARAM0:               origin = 0x0080 length = 0x1f00
    PAGE 0: DARAM1:               origin = 0x2000 length = 0x1f00
    PAGE 0: DARAM2:               origin = 0x4000 length = 0x1000
    PAGE 0: DARAM3:               origin = 0x6000 length = 0x1f00
    PAGE 0: MYVECT: origin = 0x7F80 length = 0x80
}
SECTIONS
{
    .vectors:  > MYVECT            PAGE 0
    .text:     > DARAM0            PAGE 0
    .cinit:    > DARAM0            PAGE 0
    .bss:      > DARAM1            PAGE 0
    .stack:    > DARAM1            PAGE 0
    .const:         > DARAM1       PAGE 0
    .sine_table_1 > DARAM2 PAGE 0
    .csldata > DARAM1 PAGE 0
}
```

B.2 DSPLib Code to Implement FIR and IIR Filters

```
;***********************************************************
; Version 2.20.01
;***********************************************************
;****************************************************************
;  Function:     fir
;  Description: delayed buffer finite impulse response filter
;
;  Copyright Texas instruments Inc, 1998
;----------------------------------------------------------------
;  Revision History:
;  1.00, Karen Baldwin 8/31/98. Original Beta Release.
;****************************************************************
                 .mmregs
; Far-mode adjustment
; -------------------
         .if __far_mode
OFFSET   .set  2
         .else
OFFSET   .set  1
         .endif

FRAME_SZ       .set 1
REG_SAVE_SZ    .set 2
PARAM_OFFSET   .set FRAME_SZ + REG_SAVE_SZ + OFFSET
; Register usage
; -------------
        .asg    0 + FRAME_SZ, SAVE_AR1
        .asg    0 + REG_SAVE_SZ + FRAME_SZ, RETURN_ADDR
        .asg    0 + PARAM_OFFSET, h
        .asg    1 + PARAM_OFFSET, r
        .asg    2 + PARAM_OFFSET, db
        .asg    3 + PARAM_OFFSET, nh
        .asg    4 + PARAM_OFFSET, nx
        .asg     0, nc
        .asg     AR2, r_ptr
        .asg     AR3, h_ptr
        .asg     AR4, x_ptr
        .asg     AR5, db_ptr
        .asg     BRC, rptb_cnt
;**************************************************************************
                 .global _fir
_fir
        PSHM    ST0                                 ; 1 cycle
        PSHM    ST1                                 ; 1 cycle
        RSBX    OVA                                 ; 1 cycle
        RSBX    OVB                                 ; 1 cycle
; Save contents of AR1
; And establish local frame
; Set sign extension mode
; Set FRCT bit:
;----------------------------------------------------------------
                 FRAME            #-(FRAME_SZ)                                  ; 1 cycle
        SSBX     SXM                             ; 1 cycle
        SSBX    FRCT                           ; 1 cycle
; Copy arguments to their local locations as necessary
;----------------------------------------------------------------
```

```
        STLM    A, x_ptr                            ; 1 cycle
        MVDK   *sp(h), h_ptr                        ; 2 cycles
                 MVDK   *sp(r), r_ptr                        ; 2 cycles
                 MVDK   *sp(db), db_ptr                      ; 2 cycles
;
; Set outer loop count by subtracting 1 from nsamps and
; storing into block repeat count register
;---------------------------------------------------------------
        LD     *sp(nx), A                        ; 1 cycle
        SUB     #1, A                            ; 2 cycles
        STLM    A, rptb_cnt                      ; 1 cycle
; Set pointer to delay buffer
;---------------------------------------------------------------
        LD      *db_ptr, A                       ; 1 cycle
        STLM    A, db_ptr                        ; 1 cycle
; Store length of coefficient vector/ delay buffer in BK
; register
;---------------------------------------------------------------
        LD      *sp(nh), A                      ; 1 cycle
        STLM    A, BK                           ; 1 cycle
        SUB      #3, A                                                 ; 2 cycles
        STL      A, *sp(nc)                                 ; 1 cycle
; Begin outer loop on # samples
;---------------------------------------------------------------
_start:
     RPTBD END_LOOP - 1                                       ; 2 cycles
;
; Store 0 to AR0, to use as circular addressing offset
;---------------------------------------------------------------
                 STM             #1, AR0                ; delay slot; 2 cycles
;
; Zero the accumulator before calculating next sum.
; Move next input sample into delay buffer
;---------------------------------------------------------------
       MVDD    *x_ptr+, *db_ptr                              ; 1 cycles
;
; Sum h * x for next y value
;---------------------------------------------------------------
                 MPY              *h_ptr+0%, *db_ptr+0%, A                            ; 1 cycle
                 RPT              *sp(nc)                                                     ; 2
cycle
                 MAC              *h_ptr+0% , *db_ptr+0%, A                           ; 1 cycle * ncoeffs-2
                 MACR             *h_ptr+0% , *db_ptr, A                              ; 1 cycle
;
; Store result
;---------------------------------------------------------------
                 STH              A, *r_ptr+                                                ; 1 cycle
END_LOOP:
_end:
;
; Reset FRCT bit to restore normal C operating environment
; Return overflow condition, OVA, in accumulator A
; Restore stack to previous value, FRAME, etc..
;---------------------------------------------------------------
RETURN:
        LDM      db_ptr, B                           ; 1 cycle
        MVDK     *sp(db), db_ptr                     ; 2 cycles
        LD       #0, A                               ; 1 cycle
        XC       1, AOV                              ; 1 cycle
        LD       #1, A                               ; 1 cycle
        FRAME  #(FRAME_SZ)                                                 ; 1 cycle
        POPM    ST1                         ; 1 cycle
        POPM    ST0                         ; 1 cycle
        .if __far_mode
           FRETD                                     ; 4 cycles
        .else
                  RETD                                        ; 3.0 cycles
        .endif
        NOP                                          ; delay slot 1 cycle
        STL      B, *db_ptr                                                ; delay slot 1 cycle
;END
;end of file. please do not remove. it is left here to ensure that no lines of code are removed by any editor
```

```
;************************************************************
; Version 2.20.01
;************************************************************
;*****************************************************************************
```

```
;  Function:      iircas51
;  Description:  cascaded IIR direct form I using 5-coefs per biquad
;
;  Copyright Texas instruments Inc, 1998
;----------------------------------------------------------------------------
; Revision History
; 1.00          R. Piedra, 8/31/98. Original version. Started from code by Jeff Hayes.
; 2.00  Li Yuan, 6/08/01. Fixed the problem of overflow.
; 3.00  Cesar Iovescu, 10/04/01. Use a1/2 insead of a1 to avoid overflow.
;****************************************************************************
        .mmregs
; Far-mode adjustment
                .if __far_mode
offset          .set 1                                  ; far mode uses one extra location for ret
addr  ll
                .else
offset          .set 0
                .endif
                .asg            (2), save_ar7                   ; stack description
                .asg            (3), save_ar6
                .asg            (4), save_ar1
                .asg            (5), ret_addr
                .asg            (6 + offset), arg_h
                .asg            (7 + offset), arg_y
                .asg            (8 + offset), arg_d
                .asg            (9 + offset), arg_nbiq
                .asg            (10 + offset), arg_n
                                                                                ; register usage
                                                                                ; AR0: circ addr index
                .asg            ar1, ar_x                       ;
                .asg            ar2, ar_d                       ;
                .asg            ar3, ar_h                       ;
                .asg            ar4, ar_y                       ;
                .asg            ar5, ar_count                   ;
                .asg            ar6, ar_hsave                   ;
                .asg            ar7, ar_dsave                   ;
;*****************************************************************************
                .def            _iircas51

_iircas51:

; Preserve registers
;------------------

                pshm            ar1                                             ; preserve registers
                                (1)
                pshm            ar6                                             ;
                                                (1)
                pshm            ar7                                             ;
                                                (1)
        PSHM    ST0                              ; 1 cycle
        PSHM    ST1                              ; 1 cycle
        RSBX    OVA                              ; 1 cycle
        RSBX    OVB                              ; 1 cycle

                ssbx            sxm                                             ; sign extension on
                                (1)
                ssbx            frct                                            ; fract on
                                (1)

; Get arguments
;--------------
                                                                                ; get arguments
                stlm            a, ar_x                         ; pointer to x
                (1)
                mvdk            *sp(arg_h),*(ar_hsave)          ; pointer to h
                (2)
                mvdk            *sp(arg_y),*(ar_y)              ; pointer to y
                (2)
                mvdk            *sp(arg_d),*(ar_dsave)          ; de-referencing                (2)
                mvdk            *ar_dsave,*(ar_dsave)           ; pointer to d
                (2)

                ld              *sp(arg_nbiq),a   ; a = nbiq                                    (2)
                sub             #1,a,b                                          ;
                                                (2)
                stl             b, *sp(arg_nbiq)  ; nbiq(stack) = nbiq-1                        (1)

                mvdk            *sp(arg_n),*(ar_count)          ; sample counter                (2)
```

```
                 mar              *ar_count-                         ; ar_count = nsamples-1           (1)
                 stm              #2,AR0                                              ; used for incrementing
ar_d             (2)
; Loop through N samples
;-----------------------
_start:
next_sample
                 ; Loop each sample through N biquads
                 ;------------------------------------
                 mvdk             *sp(arg_nbiq), BRC                 ; BRC <- (nbiq -1)
                 (2)
                 mvmm             ar_hsave,ar_h                      ; reinitialize pointer to top     (1)
                 mvmm             ar_dsave,ar_d                      ; reinitialize pointer to top     (1)

                 rptbd            eloop-1                            ;
                                  (2)
                 ld               *ar_x+, 16, a                      ; ah = x(n) = new sample          (1)
                 nop                                                                  ;
                 mpya             *ar_h+                                              ; B = (a= x(n))* b0
                                  (1)
                                                                                      ; T = b0
                 mac              *ar_d+,*ar_h+,b  ; B += x(n-1)*b1                   (1)
                                                                                      ; T = x(n-1)
                 mac              *ar_d-,*ar_h+,b  ; B += x(n-2)*b2                   (1)
                                                                                      ; T=x(n-2)

; B = x(n)*b0 + x(n-1)*b1 + x(n-2)*b2

                 delay            *ar_d                                               ; x(n-2) = x(n-1)
                 (1)
                 sth              a,*ar_d+0                          ; x(n-1) = x(n)                   (1)
                 mas              *ar_d,*ar_h,b    ; b -= a1/2*y(n-1)                                  (1)
                 mas              *ar_d+,*ar_h+,b  ; b -= a1/2*y(n-1)                                  (1)
                 mas              *ar_d-,*ar_h+,b,a; b -= a2*y(n-2)                   (1)
eloop

                 delay            *ar_d                                               ; y(n-2) = y(n-1)
                 (1)
                 banzd            next_sample, *ar_count- ; compute next example                       (2)
                 sth              a,*ar_d                            ; y(n-1) = y(n)                   (1)
                 sth              a,*ar_y+                           ; store y(n)
                 (1)


; Return
;-------
_end:
                 mvmm             ar_dsave,ar_d                      ; reinitialize pointer to top     (1)
                                                                                      ; to allow dual buffering
                                                                                      ; OJO : NOT REALLY NEEDED
                 .asg             ar_h, ar_temp                      ; ar_h not used anymore
                 mvdk             *sp(arg_d),*(ar_temp)              ;
                 mvkd             *(ar_d), *ar_temp; update new ar_d                  (2)
                 ld               #0,a                                                ;
                 xc               1, AOV                                              ; return overflow flag
                 ld               #1,a                                                ; if either a or b overflow
                 xc               1, BOV                                              ;
                 ld               #1,a                                                ;
                                                    (1)
        POPM    ST1                              ; 1 cycle
        POPM    ST0                              ; 1 cycle
                 popm             ar7                                                 ;
                 popm             ar6                                                 ;
                 popm             ar1                                                 ;
                 .if              __far_mode
                 fretd                                                                ;
                 .else
                 retd                                                                 ;
                                  .endif
                 nop
                 nop

;end of file. please do not remove. it is left here to ensure that no lines of code are removed by any editor
```

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless