Exploring Motorola's DSP56307EVM:

Creating a Voice Mailbox

Adam Dziedzic
&
Duane Laviniere

Microprocessor Systems
12/10/99

Table of Contents

Abstract	Page 5
Introduction	Page 6
Materials and Methods	Page 7
Results	Page 12
Discussion	Page 13
Appendix	Page 14
Bibliography	Page 16

List of Tables

Table 1	Page 11

List of Figures

Figure 1 Page 14

Abstract:

The DSP56303 Voice Message system is a low-cost voice mailbox solution for small offices or homes. The product makes use of the Motorola DSP56303, as well as the Motorola HC12. The goal of this project was to create a voice message system with two voice mailboxes. The user would control the messaging system through the DSP. The user could press a button to record a message to either mailbox, and then press another button to playback the audio.

We wrote the audio recording software, as well as the filters used to remove signal noise. All of this was written in assembly for the DSP56303. When running, the software stored incoming audio to the 64kB of internal memory. This amounted to roughly eight seconds worth of sound data.

The HC12 was used to interface external memory. We discovered late that the DSP56303 lacks a real interface for external memory, so we decided to interface it to an HC12 board and perform memory expansion on that. Our plan was to use the internal memory of the HC12 as the secondary voice mailbox.

The interface for the voice message system used the two main toggles on the DSP56303. When one button was depressed, the DSP recorded all audio streaming into the Line Input port on the board. When the second button was depressed, the DSP played the recorded audio back through the output ports.

Overall, the DSP56303 proved a healthy, and welcome challenge. This project has further increased our knowledge of microprocessors.

Introduction:

This document details the creation of a voice mailbox using the Motorola DSP56307EVM, along with the implementation of a low-pass filter and interfacing additional memory to the evaluation module. Also included in the report is information on compiling assembly programs with the Motorola assembler and linker.

The voice mailbox system created is capable of recording audio at a sampling rate of 8kHz for about eight seconds with the 64k of SRAM on the evaluation module. Alternatively, a higher sampling rate can be selected for a higher quality sound sample, but at the expense of the length of the recorded sample. The DSP can be interfaced to up to 16M of external memory, although the evaluation module does not support additional memory beyond the 64k. For this reason, we had to find an alternate method of interfacing memory to the DSP without the use of the bus and control signals for adding external memory to the DSP.

The solution we found was to interface the DSP to the HC12 evaluation module. The boards were interfaced through the host port on the DSP. We used the internal memory on the HC12 for the second voice mailbox for our voice messaging system.

For an additional exercise and experimentation with the DSP, a low-pass filter was implemented. Originally, we expected that it would be necessary to create a low-pass filter so that we could sample the audio at a lower rate, but this was not necessary since there is one implemented on the CS4218 which eliminates any aliasing that might occur.

Since this is the first project done using the DSPs, this report is written in a manner that is intended to be informative and capable of passing on the information we found, and what information is needed for working with the DSP56307EVM boards.

Materials and Methods:

Software Creation

1) The voice mailbox program

For the programs created, the example code came in very handy. There were three example programs that were referenced for solving various problems. The program that was used as a starting point was the *echo* program. This was easily modified to provide a direct audio input to output without any signal processing. From there, the code was modified to store a sound sample in the internal memory of the DSP, play the recorded sample back, and then repeat the process. From there, the program was modified to utilize the external memory on the evaluation module. And finally, the interrupts from the buttons were utilized along with the timer interrupt for the flashing red LED.

For getting started writing programs for the DSP in assembly, there are a couple things that are very useful implemented in the DSP. The first notable part of the DSP is how the memory is organized. There are three memory spaces that the DSP uses: labeled X, Y, and P. The P, or program, memory is where the program code is usually stored. The two data spaces labeled X and Y are very convenient for manipulating two audio channels.

The next interesting part of the DSP is the address generation unit. The AGU is divided into two identical halves, each with its own ALU. There are four sets of three registers for each half that provide a means of accessing memory quickly and easily. The three registers are the address register, the offset register, and the modifier register. The address register (Rn, where n = 1..7) is the pointer to the memory location. The Offset register (Nn) has two uses: to offset the location in memory to make it possible to address higher memory locations than the 16 bit address register can alone, or to increment the address by more than one. And the third register, the modifier register (Mn) does a modulus on the address to create a circular buffer of length Mn. The circular buffer was used in the low-pass filter program, and the offset was used in the voice mailbox program. Detailed information about the AGU and sample code for each use can be found in the DSP56000 Family Manual, section 4.

Parallel data moves make it possible to speed up signal processing code greatly. While the processor is executing an instruction, it can have two parallel data moves executing at the same time getting the data ready for future instructions. For the instructions that allow parallel data moves, the assembly code is written with the instruction followed by the source and destination operands, then up to two data moves in the X, Y, and/or P memory segments.

The voice mailbox program is rather simple, once everything was figured out. To enable the external memory on the EVM, there are only two variables that need to be set. The address attribute register M_AAR0 needs to be set up for how the memory is to be interfaced. The other variable that needs to be initialized for the external memory is M_BCR. AAR0 tells the DSP the address where the memory is located and what memory spaces to use it with, while M_BCR controls the timing for the memory. After the memory is set up, the interrupts for the A and B buttons are enabled along with the timer interrupt for the flashing LED. This is done by setting M_IPCR as one of the example programs showed. The next step is to set up the stack, the operating mode, and the buffer for storing the audio. The audio buffer is initialized by writing the start of the external memory to the offset register, N4, using M4=-1 to signal that the AGU should use linear addressing, and setting R4 to 0 so that (R4+N4) points to the beginning of the buffer. The rest of the program simply waits for an interrupt to tell it to record or play back a sound sample. The interrupts are set up by writing jump instructions to the specified lines of the jump table that correspond to the interrupts A and D. For this program, the jump instructions go to the routines to record or play back a sound sample. The other files that are included did not require any modification from how they are distributed with the echo sample program.

2) The Low-Pass Filter Program

There are two low pass filters implemented that are almost identical. The first filter created was for a 5500Hz cut off frequency. The filter was designed with an online web page (http://www-users.cs.york.ac.uk/~fisher/cgi-bin/mkfscript). For this filter, the parameters entered were a *Second order Lowpass* filter type of *Butterworth* with a sampling rate of *44100* samples per second, and a cut off frequency of *5500*. With that information, the online script creates the recurrence relation and a sample of the C code that could be used to implement the filter, along with plots of the phase and frequency and some other plots. The main interest, if the frequency response looks like the one desired, is the code for implementing the filter. The next step was to translate the C code into assembly, which took quite a long time. It is very difficult to effectively use the parallel data moves to shrink the code. It is however theoretically possible to shrink the filtering code to one line per term in the filter equation using the multiply and accumulate instruction. Once the code was implemented, it took a while to make sure that it was actually filtering out some frequencies, because the high frequencies made very subtle differences to how the audio sounded.

The second filter implemented was the same as the first, only with a *2500Hz* cut-off frequency. The main difference between this program and the previous, other than the coefficients, is the fact the one of the coefficients is greater than one. Since the floating point representation of numbers in the DSP takes only decimal numbers, an additional line of code was needed to add the y[n-1] term to the lines that compute the output to get rid of the one in the coefficient. The difference in the sound of the audio was still very subtle, but it was more noticeable than the 5500Hz cut-off frequency.

Compiling the Programs

The first thing that needs to be done before compiling is either make sure the current directory is the one that contains the assembler or make sure that directory is in the path. To assemble the code after it was written, the asm56300.exe assembler was used, followed by the linker, dsplnk.exe. The assembler requires the *–B* option so that it creates a linkable *.cln* file. The line for compiling in general is as follows, where parts in [] are optional.

Asm56300 –B[output_object_file] source[.asm]

The code for compiling the voice mailbox program follows:

asm56300 -Brecplay.cln recplay.asm > err.txt

This explicitly created the default object file and sent all the information from compiling to the file err.txt so that the errors that were reported could be examined later. Once the code compiles successfully, there will be a file with the specified or default filename and an extension of ".cln" in the current directory. This file needs to be linked, which can be done by calling "dsplnk" followed by the name of the object file. Once this process is complete, there should be a file ending in ".cld" that can be uploaded to the DSP board using the program EVM30xW.

Hardware

The only hardware required for the above-mentioned programs is setting the jumpers for the sampling rate on the EVM. The jumpers are clearly labeled sampling rate, and set the rate according to the following table, from section 3 of the DSP56307EVM Technical Summary document.

Table 1: cs4218 Sampling Frequency Selection

J9 Pins 1-2 (MF6)	J9 Pins 3-4 (MF7)	J9 Pins 5-6 (MF8)	Sampling Rate (kHz)
Jumper	Jumper	Jumper	48.0
Jumper	Jumper	Open	32.0
Jumper	Open	Jumper	24.0
Jumper	Open	Open	19.2
Open	Jumper	Jumper	16.0
Open	Jumper	Open	12.0
Open	Open	Jumper	9.6
Open	Open	Open	8

The hardware for the memory expansion interface with the HC12 consisted of a 20pin connection between the Host Port (HI08) on the DSP to the J8 and J9 ports on the HC12. For this, we used a modified 60pin cable to slide onto the HI08 DIP.

Results:

In the end, we accomplished the goals we set out for ourselves. We learned to properly use the DSP56303. This was our main goal. We were not familiar with the device prior to the start of this project, and now we feel that we have a substancial understanding of the evaluation module.

We learned how to utilize the many features of the DSP56303. We were not content to simply get the voice message system up and running. We also wanted to learn the new features the board provides. We gained some more knowledge of the DSP by writing a high-frequency filter for the module as well as a 10-band equalizer, which we did not use for the final demonstration. This wealth of knowledge applies not only to the DSP, but also to microprocessor systems in general.

The audio recording software that we created also worked very well. When we discovered that the cross compiler documentation was inadequate, we decided to write the entire code in assembly. We were also worried about performance, so getting down to the metal was very important. The end result is not perfect, but it is well written code that performs the proper audio recording routines when we need them.

Unfortunately, our results are not quantifiable. However, we have appended some of our source code to the end of this report. The result is a voice mailbox system that records audio samples, and then plays it back for the listener.

Discussion:

Our final project differs from our initial goal in that the external memory interface had to be done through the HC12. Our initial expectations were for a compact package with only the DSP56303 and one megabyte of DRAM. However, since the DSP evaluation module that we received lacked an appropriate external address bus, we were forced to use the HC12.

If we were given more time, we would have been able to interface one megabyte of DRAM with the HC12, instead of using the internal memory of the HC12. However, the pin-outs for the DRAM chip were cryptic, and so any last minute changes could have been fatal to our design.

Appendix:

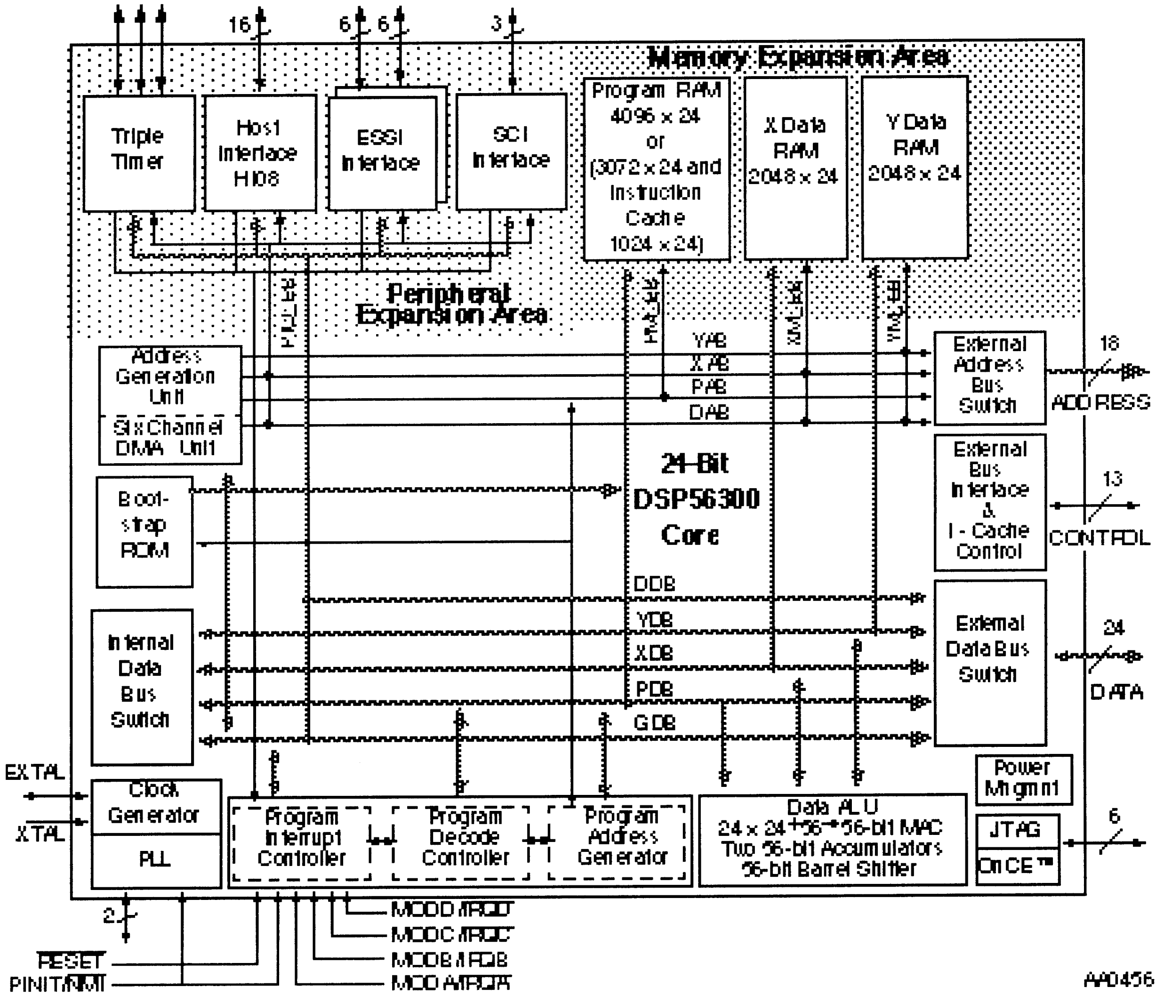


Figure 1: DSP56303 Block Diagram

```
;
; This program, originally available on the Motorola DSP bulletin board,
; is provided under a DISCLAIMER OF WARRANTY available from Motorola DSP
; Operation, 6501 William Cannon Drive, West, Austin, Texas  78735-8598.
;


        page 132,66,3,3,0


        org     x:$0
input   dsm     2
output  dsm     2
missc   ds      3
storer5 ds      1

        org     y:$0
average ds      2
temp    ds      2
storer6 ds      1
storer7 ds      1
coef    ds      3

        org     p:$0
        jmp     $40

        org     p:$0c
        jsr     ioprocess

        org     p:$0e
        jsr     ioprocess

start   equ     $40
half    equ     .5
alpha   equ     .99
beta    equ     1-alpha
oneos2  equ     .70711
maxcos  equ     2.56
coefa1  equ     .8803385
coefa2  equ     -.1985987
coefa0  equ     .3175231

sqrt    macro
        mpyr    x0,x0,b             y:(r5)+,y0
        mpy     x0,y0,b b,x1        y:(r5)+,y0
        macr    x1,y0,b             y:(r5)+,y0
        add     y0,b
        endm

divide  macro
        and     #$fe,ccr
        rep     #$18
        div     x0,a
        add     x0,a
        move    a0,b
        endm


        org     p:start
        or      #$03,mr
        movep   #$1a00,x:$ffed
        movep   #$3000,x:$ffff
        movep   #$4100,x:$ffec
        movep   #$ba00,x:$ffed
```

```
        movep   #$1ff,x:$ffe1
        movep   #$3,x:$ffe2
        move    #coef,r5
        move    #input,r1                               ;set r1 to point to
        move    #coefa1,b
        move    #coefa2,a
        move    b,y:(r5)+
        move    #coefa0,b
        move    a,y:(r5)+
        move    b,y:(r5)
        move    #1,m0                                           ;input
        move    #missc+2,r2                             ;set r2 to point to
        move    #2,n2
        move    #input,r0
        move    #output,r3
        move    n2,n1
        move    #0,n6
        move    m0,m3
                                                        ;bottom of misc.
        move    #average,r4                             ;set r4 to point at
                                                        ;average locations
        move    #temp,r5                                ;set r5 to the temporary
        move    n6,y:(r4)+
        move    n6,y:(r4)-
        andi    #$fc,mr

loop0   jmp     loop0

ioprocess
        movep   x:$ffef,x:(r0)+
        movep   x:(r3)+,x:$ffef
        move    n6,a
        move    r0,b
        cmp     a,b
        jseq    detect
        rti

                                                        ;I and Q locations
detect
        move    x:(r1)+,y0                              ;move input into  x0
        move    x:(r1)+n1,x1        y0,y:(r5)+          ;move x0 into temp.
                                                        ;move q input into x1
                                                        ;move q into temp
        move    x1,y:(r5)-
        move    #beta,y1                                ;move beta into y1 to
                                                        ;perform average
                                                        ;calculation
        mpy     y0,y1,a             #alpha,y0           ;multiply I input by beta
                                                        ;move alpha(1-beta)
                                                        ;into y0
        mpy     x1,y1,b             y:(r4)+,x0          ;multiply q input by beta
                                                        ;move past I average
                                                        ;xo to continue average
        mac     x0,y0,a             y:(r4)-,x1          ;acuumulate the new
                                                        ;I average and move
                                                        ;the old Q average
                                                        ;into x1
        mac     x1,y0,b             a,y:(r4)+           ;accumulate the new Q
                                                        ;average and move new
                                                        ;I average into memory
        sub     b,a       b,x1      b,y:(r4)-           ;subtract Q from I to
                                                        ;get carrier value and
                                                        ;move Q average to mem.
        move    a,x:(r2)-           y:(r5)+,a           ;move carrier into
                                                        ;memory an move I
                                                        ;into a to find bias
```

```
        sub     x1,a              #0,b              ;subtract bias from I
                                                    ;move Q into x0
        cmp     b,a               y:(r5)-,x0
        jne     start1
        move    (r2)+
        move    a1,x:(r1)-
        move    a1,x:(r1)-n1
        rts
start1  asr     a                 a,y:(r5)+         ;Shift I* right and
                                                    ;store I*
        tfr     x0,a              a,x0              ;transfer a and x0 so
                                                    ;bias can be subtracted
                                                    ;from q
        sub     x1,a              #0,r6             ;subtract bias from Q
                                                    ;move #0 into r6 to
                                                    ;count the number of
                                                    ;shift lefts needed
                                                    ;to make the number be
                                                    ;between .5<b,1
        asr     a                 a,y:(r5)-         ;Divide Q* in half
                                                    ;store Q* in memory
        mpy     x0,x0,b           a,y0              ;square I*/2
        mac     y0,y0,b           #half,y0          ;add to square Q*/2
                                                    ;Put .5 into y0 to
                                                    ;compare for sqrt

loop1   cmp     y0,b                                ;compare b to .5 to see
                                                    ;if it is greater than
                                                    ;.5 for the sqrt
                                                    ;algorithm
        jge     dosqrt                              ;jump if greater than
        asl     b       x:(r6)+,x1                  ;if b is less than .5
                                                    ;then shift b left and
                                                    ;increment r6
        jmp     loop1                               ;jump to compare again
dosqrt
        move    #coef,r5
        move    b,x0
        sqrt
        move    r6,x0
        move    #0,a
        cmp     x0,a    #temp,r5
        jeq     around1
        move    #oneos2,y0                          ;now multiply the out
                                                    ;put by 1/sqrt2 for
                                                    ;every shift left

        do      x0,enddo1
        move    b,x1                                ;move b into x0 to mult.
        mpy     x1,y0,b                             ;multiplly by 1/sqrt2
enddo1
around1 move    b,x:(r2)-         y:(r5),a          ;store h and recall I*
                                                    ;to get ready to divide
                                                    ;H by I* to ger 1/cos0

        abs     a       #1,r7
                                                    ;for the number of shift
                                                    ;lefts neede for
                                                    ;dividing H by I
loop2   cmp     a,b               y:(r5),y0         ;So compare I to H
        jlt     dodivide                            ;Do the divide if H<I
        asr     b       x:(r7)+,x1                  ;If H>I,shift H right
                                                    ;and increment the
                                                    ;counter

        jmp     loop2
dodivide
        tfr     b,a               a,x0
        divide
```

```
move    r7,x0
move    b,x:(r2)+n2                               ;store 1/cos0
move    x:(r2),a                                  ;move carrier into a and
                                                  ;store 1/cos0 in x1
rep     x0
asr     a                                         ;shift the carrier right
                                                  ;as many times as the
                                                  ;H was shifted for the
                                                  ;divide + one for the
                                                  ;shift right of I and Q
                                                  ;before the sum of
                                                  ;squares
                                                  ;save the new carrier
move    b,x1      y:(r5)+,y1
                                                  ;back to memory and
                                                  ;move I* back into x0
mpy     x1,y1,b a,x:(r2) y:(r5)-,y0               ;multiply I* by 1/cos0
                                                  ;and move Q* into y0
mpy     x1,y0,a           a,y1                    ;multiply Q* by 1/cos0
                                                  ;and move a into x0
sub     y1,b                                      ;subtract the carrrier
                                                  ;from I and store (L-R)
add     b,a               a,x0                    ;left
sub     x0,b
asl     a
asl     a
asl     b
asl     b       a,x:(r1)-         ;right
move    b,x:(r1)-n1
rts
```

Voice Mailbox page 1 of 2 Lacks a program header!

```
;************************************************************************

      nolist
      include 'ioequ.asm'
      include 'intequ.asm'
      include 'ada_equ.asm'
      include 'vectors.asm'
        list

;*************************************************************************

Y_SIZE          EQU     $010000         ;     64K Y: WORDS
Y_START         EQU     $040000                     ; start address of external memory

LINEAR          EQU     $FFFFFF          ;Linear addressing mode
AAR0V           EQU     $040831          ;Value programmed into AAR0
                                      ;Compare 8 most significant bits
                                      ;Look for a match with address
                                      ;Y:0000 0100 xxxx xxxx xxxx xxxx
                                      ;No packing, no muxing, X, Y, and
                                      ;P enabled, AAR0 pin active low
                                      ;Asynchronous SRAM access

BCRV             EQU   $012421              ;Value programmed into BCR
                                      ;1 wait state for all AAR regions

;---Buffer for talking to the CS4218

       org    x:$0
RX_BUFF_BASE   equu     *
RX_data_1_2    ds      1        ; data time slot 1/2 for RX ISR (left audio)
RX_data_3_4    ds      1        ; data time slot 3/4 for RX ISR (right audio)

TX_BUFF_BASE   equ      *
TX_data_1_2    ds      1        ; data time slot 1/2 for TX ISR (left audio)
TX_data_3_4    ds      1        ; data time slot 3/4 for TX ISR (right audio)

RX_PTR          ds      1       ; Pointer for rx buffer
TX_PTR          ds      1       ; Pointer for tx buffer

CTRL_WD_12      equ     MIN_LEFT_ATTN+MIN_RIGHT_ATTN+LIN2+RIN2
CTRL_WD_34      equ     MIN_LEFT_GAIN+MIN_RIGHT_GAIN

; --- jumps for using the buttons ---
               org            p:$10
               jmp            rec_loop                                ; IRQA--Record

               org            p:$16                           ; IRQD--Play back
               jmp            play_loop


;---here's the program
        org     p:$100
START
main
        movep   #$040006,x:M_PCTL     ; PLL 7 X 12.288 = 86.016MHz
        movep   #AAR0V,x:M_AAR0         ;AAR0 as shown above
        movep   #BCRV,x:M_BCR           ;One ext. wait state for async srams
       movep   #$000E07,X:M_IPRC       ;IRQA/IRQD/SSI level 3 int edge sensitive
        ori     #3,mr                          ; mask interrupts
        movec   #0,sp                          ; clear hardware stack pointer
        move    #0,omr                         ; operating mode 0
        move    n0,r0                          ; Load start address of P into r0
        move    #$40,r7                                ; initialize stack pointer
        move    #-1,m7                         ; linear addressing
        jsr     ada_init                       ; initialize codec

        move    #Y_START,n4         ; offset of buffer - external memory
        move    #$FFFF,m4           ; use linear addressing
               move    #0,r4
```

Voice Mixer page [illegible] 2

```
        move     #LINEAR,m0

rec_loop
        movep    #$21,x:M_TCSR0                 ;Timer mode 2 (FLASH LED)
                 move    #0,r4

        do               #$F,rec2
                 do              #$FFF,rec                      ; repeat recording #$FFFF times

        jset     #3,x:M_SSISR0,*     ; wait for rx frame sync
        jclr     #3,x:M_SSISR0,*     ; wait for rx frame sync
                 clr a
                 clr b
                 move     x:RX_BUFF_BASE,a    ; receive left
;        move     x:RX_BUFF_BASE+1,b ; receive right

                 nop
                 move    a,y:(r4+n4)                    ; save new sample in buffer
                 move    y:(r4)+,b                      ; increment buffer

                 nop
                 move     a,x:TX_BUFF_BASE    ; transmit left
;        move     b,x:TX_BUFF_BASE+1 ; transmit right

rec
                 nop
rec2

        movep    #$11,x:M_TCSR0                 ;Select timer mode 1 (LED OFF)
                 jmp      rec

play_loop
        movep    #$11,x:M_TCSR0                 ;Select timer mode 1 (LED OFF)
                 move    #0,r4

                 do              #$F,play2
                 do              #$FFF,play                     ; repeat recording #$FFFF times

        jset     #3,x:M_SSISR0,*     ; wait for rx frame sync
        jclr     #3,x:M_SSISR0,*     ; wait for rx frame sync
                 clr a
                 clr b
                 move     x:RX_BUFF_BASE,a    ; receive left
;        move     x:RX_BUFF_BASE+1,b ; receive right

                 nop
                 move     y:(r4+n4),a                   ; recall oldest sample in buffer
                 move     y:(r4)+,b                     ; increment buffer

                 nop
;                move     b,x:TX_BUFF_BASE    ; transmit left
        move     a,x:TX_BUFF_BASE+1 ; transmit right

play
                 nop
play2
                 jmp      play

                 include 'ada_init.asm' ; used to include codec initialization routines

        end
```

```
;       Author:        Adam Dziedzic
;       Filename:      LPfilter.asm
;       Description:   implements a second order Butterworth Lowpass filter
;               sampling rate:         44.1 kHz
;               cut-off frequency:     5.5 kHz
;               Nyquist frequency:     22.05 kHz
;
;****************************************************************************

    nolist
      include 'ioequ.asm'
      include 'intequ.asm'
      include 'ada_equ.asm'
      include 'vectors.asm'
        list

;*******************************************************************************


;---Buffer for talking to the CS4218
       org    x:$0
RX_BUFF_BASE   equ      *
RX_data_1_2    ds       1       ; data time slot 1/2 for RX ISR (left audio)
RX_data_3_4    ds       1       ; data time slot 3/4 for RX ISR (right audio)

TX_BUFF_BASE   equ      *
TX_data_1_2    ds       1       ; data time slot 1/2 for TX ISR (left audio)
TX_data_3_4    ds       1       ; data time slot 3/4 for TX ISR (right audio)

RX_PTR          ds       1      ; Pointer for rx buffer
TX_PTR          ds       1      ; Pointer for tx buffer

CTRL_WD_12      equ     MIN_LEFT_ATTN+MIN_RIGHT_ATTN+LIN2+RIN2
CTRL_WD_34      equ     MIN_LEFT_GAIN+MIN_RIGHT_GAIN


        org     p:$190

;Low-pass filter coeffs
;NZEROS                 equ             2
;NPOLES                 equ             2
;GAIN                   equ             1.028071822
; INV_GAIN
                                dc              0.9726946878
; COEF1
                                dc              0.3341260413
; COEF2
                                dc              0.9450481661
;INV_GAIN               equ     *
;INV_GAIN_d             equ     0.9726946878
;COEF1                  equ     *
;COEF1_d                        equ     0.6905989232
;COEF2                  equ     *
;COEF2_d                        equ     1.6329931618

        org     p:$100
START
main
              ;Do initialization stuff
        movep   #$040006,x:M_PCTL  ; PLL 7 X 12.288 = 86.016MHz
        ori     #3,mr              ; mask interrupts
        movec   #0,sp              ; clear hardware stack pointer
        move    #0,omr             ; operating mode 0
              move    #$40,r7            ; initialize stack pointer
        move    #-1,m7             ; linear addressing
        jsr     ada_init           ; initialize codec

              ;Filter buffers
              move    #$0400,r3                  ; xv[] buffer from filter code - start at
$400
              move    #3,m3                      ; length of 3 samples
```

```
        move    #$0420,r4                       ; yv[] buffer
        move    #$3,m4                          ; make filter buffer 3 deep

                ; Coefficients
                move    #$190,r5                        ; coefficient buffer
                move    #$3,m5                          ; 3 deep already set up (with dc)

        clr     a                               ; clear a
        rep     #3                              ; clear the filter buffer
        move    a,l:(r4)+

filter_loop
                ;Get a sound sample
        jset    #3,x:M_SSISR0,*                                         ; wait for rx frame
sync
        jclr    #3,x:M_SSISR0,*                                         ; wait for rx frame
sync

                clr             b
                move    x:RX_BUFF_BASE,x0                               ; new input
                move    p:$190,y0                                               ; load 1/GAIN
into y0
                mpy             x0,y0,b         x:(r3)+,x0      y:(r4)+,y0      ; b = new_input/GAIN,
x0 = x[n-2], y0 = y[n-1]
                move    b,x:(r3)+                                               ; store x[n]
                add             x0,b                                                    ; b +=
x[n-2]
                move    p:$192,x0                                               ; x0 = COEF2
                nop
                mac             -x0,y0,b                        y:(r4)+,y0      ; b += -COEF2
* y[n-1], y0 = y[n-2]
                move    p:$191,x0                                               ; x0 = COEF1
                mac             x0,y0,b                                                 ; b +=
COEF1 * y[n-2]
                move    x:(r3),a                                                ; a = x[n-1]
                addl    b,a                                                             ; b =
2*b + a
                nop
                move    a,y:(r4)                                                ; store output

                ;transmit sound sample - same to both channels
                move    a,x:TX_BUFF_BASE        ; right channel
                move    b,x:TX_BUFF_BASE+1      ; left channel

                jmp             filter_loop

                include 'ada_init.asm' ; used to include codec initialization routines
filter
                end
```

Low-pass filter (2.5 kHz)

```
;       Author:         Adam Dziedzic
;       Filename:       LPfilter.asm
;       Description:    implements a second order Butterworth Lowpass filter
;               sampling rate:          44.1 kHz
;               cut-off frequency:      2.5 kHz
;               Nyquist frequency:      22.05 kHz
;
;***************************************************************************

    nolist
      include 'ioequ.asm'
      include 'intequ.asm'
      include 'ada_equ.asm'
      include 'vectors.asm'
        list

;****************************************************************************


;---Buffer for talking to the CS4218
       org     x:$0
RX_BUFF_BASE   equ      *
RX_data_1_2    ds       1       ; data time slot 1/2 for RX ISR (left audio)
RX_data_3_4    ds       1       ; data time slot 3/4 for RX ISR (right audio)

TX_BUFF_BASE   equ      *
TX_data_1_2    ds       1       ; data time slot 1/2 for TX ISR (left audio)
TX_data_3_4    ds       1       ; data time slot 3/4 for TX ISR (right audio)

RX_PTR          ds       1      ; Pointer for rx buffer
TX_PTR          ds       1      ; Pointer for tx buffer

CTRL_WD_12      equ     MIN_LEFT_ATTN+MIN_RIGHT_ATTN+LIN2+RIN2
CTRL_WD_34      equ     MIN_LEFT_GAIN+MIN_RIGHT_GAIN


        org     p:$1A0

;======================
;Low-pass filter coeffs
;NZEROS                 equ             2
;NPOLES                 equ             2
;GAIN                   equ             1.028071822
; INV_GAIN
                                dc              0.51761146              ;0.9726946878
; COEF1
                                dc              0.6043997995            ;0.3341260413
; COEF2
                                dc              0.5036953413            ;0.9450481661
;----------------------

        org     p:$100
START
main
               ;Do initialization stuff
        movep   #$040006,x:M_PCTL  ; PLL 7 X 12.288 = 86.016MHz
        ori     #3,mr              ; mask interrupts
        movec   #0,sp              ; clear hardware stack pointer
        move    #0,omr             ; operating mode 0
               move    #$40,r7            ; initialize stack pointer
        move    #-1,m7             ; linear addressing
        jsr     ada_init           ; initialize codec

               ;Filter buffers
               move    #$0400,r3                  ; xv[] buffer from filter code - start at $400
               move    #3,m3                      ; length of 3 samples
        move    #$0420,r4                  ; yv[] buffer
        move    #$3,m4                     ; make filter buffer 3 deep

        clr     a                          ; clear a
        rep     #3                         ; clear the filter buffer
```

```
        move    a,l:(r4)+

filter_loop
                ;Get a sound sample
        jset    #3,x:M_SSISR0,*                                         ; wait for rx frame sync
        jclr    #3,x:M_SSISR0,*                                         ; wait for rx frame sync

                clr             b
                move    x:RX_BUFF_BASE,x0                               ; new input
                move    p:$1A0,y0                                               ; load 1/GAIN
into y0
                mpy             x0,y0,b         x:(r3)+,x0      y:(r4)+,y0      ; b = new_input/GAIN, x0
= x[n-2], y0 = y[n-1]
                nop
                move    b,x:(r3)+                                               ; store x[n]
                add             x0,b                                                    ; b +=
x[n-2]
                move    p:$1A2,x0                                               ; x0 = COEF2
                nop
                add             y0,b
                nop
                mac             -x0,y0,b                        y:(r4)+,y0      ; b += -COEF2 *
y[n-1], y0 = y[n-2]
                move    p:$1A1,x0                                               ; x0 = COEF1
                mac             x0,y0,b                                                 ; b +=
COEF1 * y[n-2]
                move    x:(r3),a                                                ; a = x[n-1]
                addl    b,a                                                     ; b = 2*b
+ a
                nop
                move    a,y:(r4)                                                ; store output

                ;transmit sound sample - same to both channels
                move    a,x:TX_BUFF_BASE        ; right channel
                move    a,x:TX_BUFF_BASE+1      ; left channel

                jmp             filter_loop

                include 'ada_init.asm' ; used to include codec initialization routines
filter
                end
```

Filter Design Results

Generated by: http://www-users.cs.york.ac.uk/~fisher/mkfilter

Summary

You specified the following parameters:

filtertype = Butterworth
passtype = Lowpass
ripple =
order = 2
samplerate = 44100
corner1 = 5500
corner2 =
adzero =
logmin =

Results

```
Command line: /www/usr/fisher/helpers/mkfilter -Bu -Lp -o 2 -a 1.2471655329e-01 0.C
raw alpha1     =   0.1247165533
raw alpha2     =   0.1247165533
warped alpha1 =   0.1315163158
warped alpha2 =   0.1315163158
gain at dc    :   mag = 1.028071822e+01   phase =   0.0000000000 pi
gain at centre:   mag = 7.269565568e+00   phase =  -0.5000000000 pi
gain at hf    :   mag = 0.000000000e+00

S-plane zeros:

S-plane poles:
         -0.5843115954 + j   0.5843115954
         -0.5843115954 + j  -0.5843115954

Z-plane zeros:
         -1.0000000000 + j   0.0000000000       2 times

Z-plane poles:
          0.4725240831 + j   0.3329369794
          0.4725240831 + j  -0.3329369794

Recurrence relation:
y[n] = (  1 * x[n- 2])
     + (  2 * x[n- 1])
     + (  1 * x[n- 0])

     + ( -0.3341260413 * y[n- 2])
     + (  0.9450481661 * y[n- 1])
```

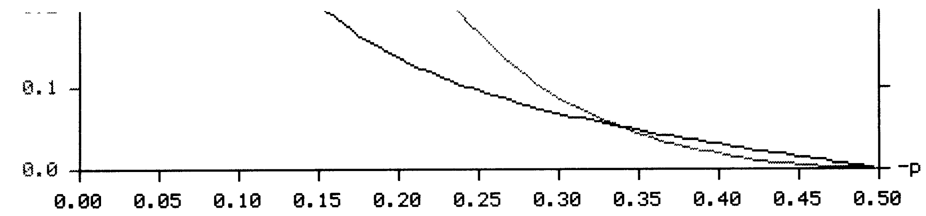
Ansi ``C'' Code

```
/* Digital filter designed by mkfilter/mkshape/gencode   A.J. Fisher
   Command line: /www/usr/fisher/helpers/mkfilter -Bu -Lp -o 2 -a 1.2471655329e-01

#define NZEROS 2
#define NPOLES 2
#define GAIN   1.028071822e+01

static float xv[NZEROS+1], yv[NPOLES+1];

static void filterloop()
  { for (;;)
      { xv[0] = xv[1]; xv[1] = xv[2];
        xv[2] = next input value / GAIN;
        yv[0] = yv[1]; yv[1] = yv[2];
        yv[2] =   (xv[0] + xv[2]) + 2 * xv[1]
                     + ( -0.3341260413 * yv[0]) + (  0.9450481661 * yv[1]);
        next output value = yv[2];
      }
  }
```

Download code and/or coefficients: TERSE VERBOSE

Magnitude (red) and phase (blue) vs. frequency

- *x* axis: frequency, as a fraction of the sampling rate (i.e. 0.5 represents the Nyquist frequency, which is 22050 Hz)
- *y* axis (red): magnitude (linear, normalized)
- *y* axis (blue): phase

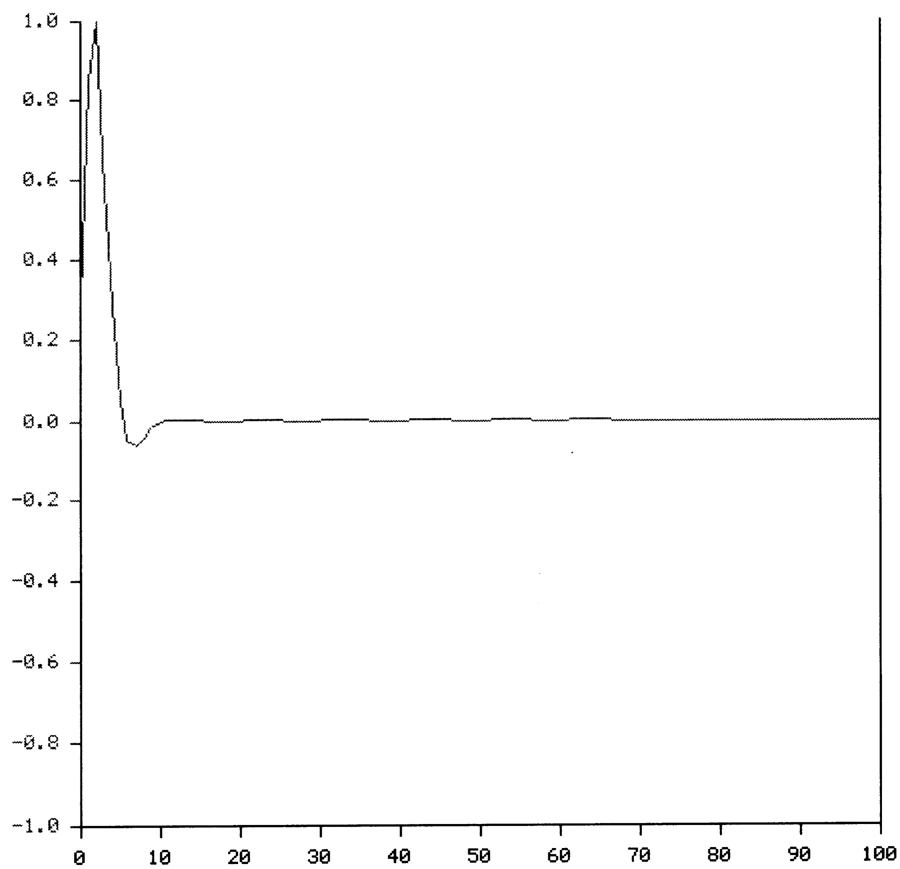


For an expanded view, enter frequency limits (as a fraction of the sampling rate) here:

Lower limit: Upper limit: zoom

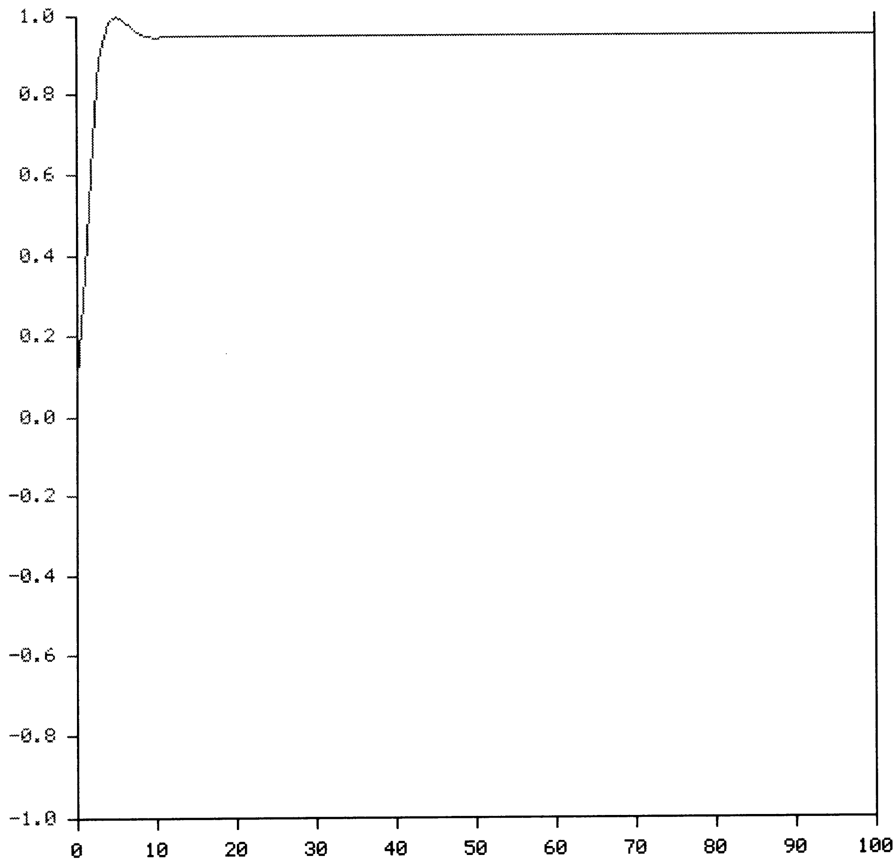
Impulse response

- *x* axis: time, in samples (i.e. 44100 represents 1 second)
- *y* axis (red): filter response (linear, normalized)



Step response

- *x* axis: time, in samples (i.e. 44100 represents 1 second)
- *y* axis (red): filter response (linear, normalized)



For a view on a different scale, enter upper time limit (integer number of samples) here:

Upper limit: [ ] zoom

Tony Fisher fisher@minster.york.ac.uk

Filter Design Results

Generated by: http://www-users.cs.york.ac.uk/~fisher/mkfilter

Summary

You specified the following parameters:

filtertype	= Butterworth
passtype	= Lowpass
ripple	=
order	= 2
samplerate	= 44100
corner1	= 2500
corner2	=
adzero	=
logmin	=

Results

```
Command line: /www/usr/fisher/helpers/mkfilter -Bu -Lp -o 2 -a 5.6689342404e-02 0.C
raw alpha1    =   0.0566893424
raw alpha2    =   0.0566893424
warped alpha1 =   0.0572963984
warped alpha2 =   0.0572963984
gain at dc    :   mag = 3.972018787e+01   phase =   0.0000000000 pi
gain at centre:   mag = 2.808641419e+01   phase =  -0.5000000000 pi
gain at hf    :   mag = 0.000000000e+00

S-plane zeros:

S-plane poles:
        -0.2545611911 + j   0.2545611911
        -0.2545611911 + j  -0.2545611911

Z-plane zeros:
        -1.0000000000 + j   0.0000000000      2 times

Z-plane poles:
         0.7518476706 + j   0.1978001002
         0.7518476706 + j  -0.1978001002

Recurrence relation:
y[n] = (  1 * x[n- 2])
     + (  2 * x[n- 1])
     + (  1 * x[n- 0])

     + ( -0.6043997995 * y[n- 2])
     + (  1.5036953413 * y[n- 1])
```

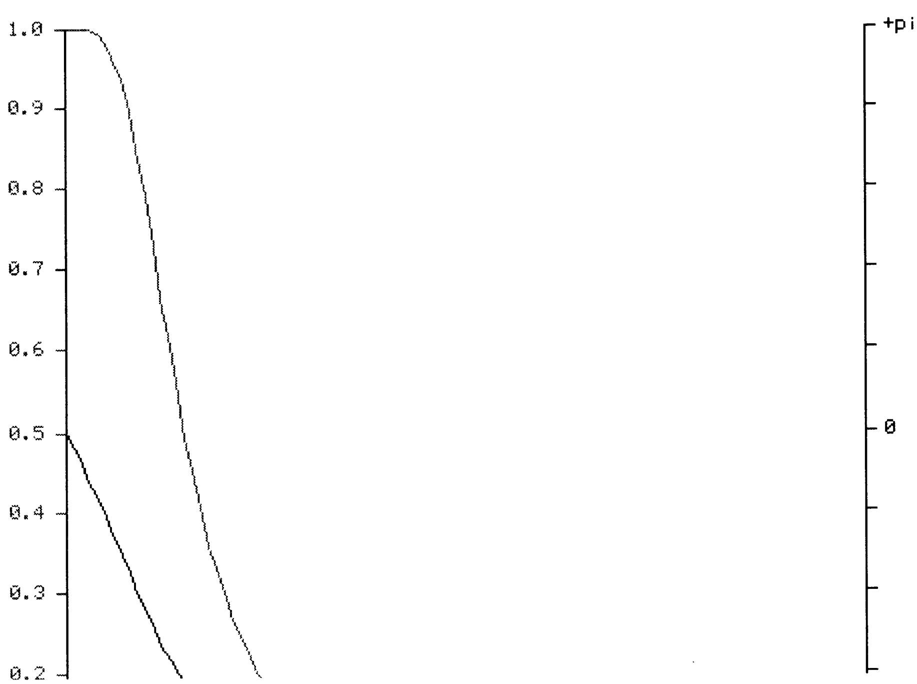
Ansi ``C'' Code

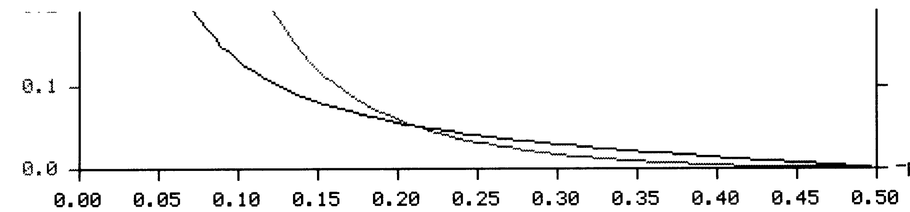
```
/* Digital filter designed by mkfilter/mkshape/gencode   A.J. Fisher
   Command line: /www/usr/fisher/helpers/mkfilter -Bu -Lp -o 2 -a 5.6689342404e-02

#define NZEROS 2
#define NPOLES 2
#define GAIN   3.972018787e+01

static float xv[NZEROS+1], yv[NPOLES+1];

static void filterloop()
  { for (;;)
      { xv[0] = xv[1]; xv[1] = xv[2];
        xv[2] = next input value / GAIN;
        yv[0] = yv[1]; yv[1] = yv[2];
        yv[2] =   (xv[0] + xv[2]) + 2 * xv[1]
                     + ( -0.6043997995 * yv[0]) + (  1.5036953413 * yv[1]);
        next output value = yv[2];
      }
  }
```

Download code and/or coefficients: TERSE VERBOSE

Magnitude (red) and phase (blue) vs. frequency

- *x* axis: frequency, as a fraction of the sampling rate (i.e. 0.5 represents the Nyquist frequency, which is 22050 Hz)
- *y* axis (red): magnitude (linear, normalized)
- *y* axis (blue): phase



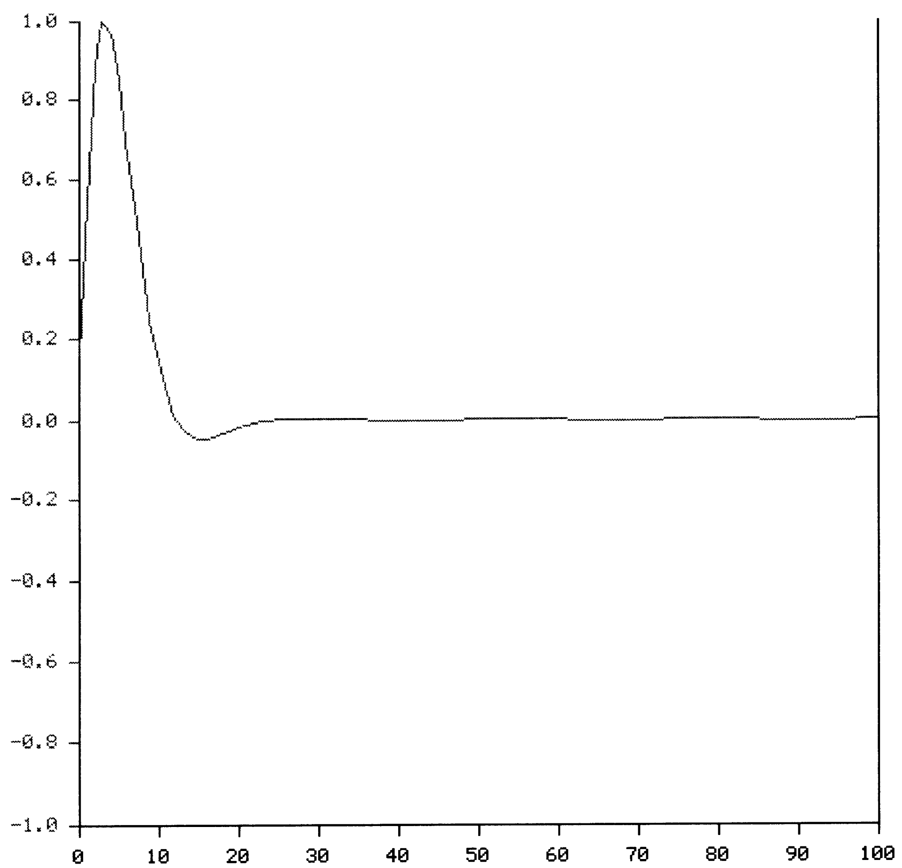


For an expanded view, enter frequency limits (as a fraction of the sampling rate) here:

Lower limit: Upper limit: zoom

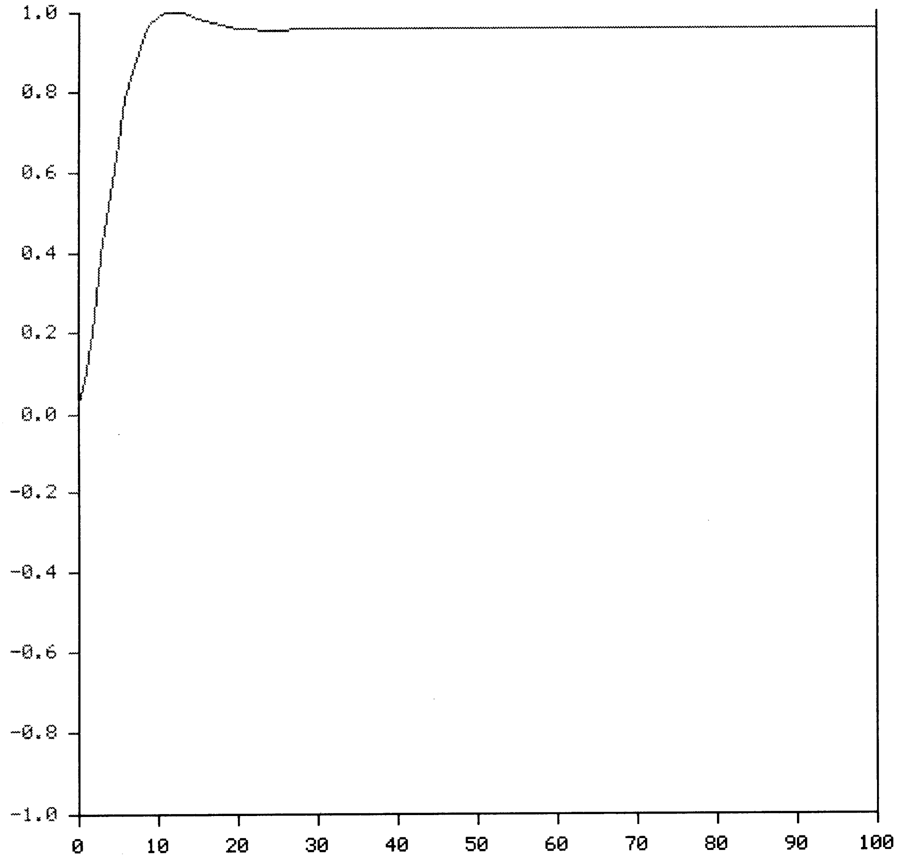
Impulse response

- *x* axis: time, in samples (i.e. 44100 represents 1 second)
- *y* axis (red): filter response (linear, normalized)



Step response

- *x* axis: time, in samples (i.e. 44100 represents 1 second)
- *y* axis (red): filter response (linear, normalized)



For a view on a different scale, enter upper time limit (integer number of samples) here:

Upper limit: zoom

Tony Fisher fisher@minster.york.ac.uk

Bibliography

Digital Stereo 10-Band Graphic Equalizer Using the DSP56001 Application Notes: Motorola Inc, 1988

DSP56000 Digital Signal Processor Family Manual, Motorola Inc, 1995

DSP56307EVM Users Manual, Motorola Inc, PDF version

http://www.dspguru.com/

http://www.mot-sps.com/

http://www-users.cs.york.ac.uk/~fisher/cgi-bin/mkfscript