Freescale Semiconductor, Inc.

MOTOROLA

Digital Signal Processing Division

This document has been designed for users who have little or no experience with DSP development tools. It provides a step-by-step guide to the installation and execution of the supplied demo(s). It also contains a number of worked examples showing how to write, assemble, and debug some simple programs using the hardware and software provided.

For full understanding of the DSP56002 and of the DSP56002EVM, this document should be read in conjunction with the manuals provided.

MOTOROLA

Digital Signal Processing Division

1.   EQUIPMENT

The following section will give a brief summary of the equipment required to use the EVM, some of which will be supplied with the EVM, and some of which will have to be supplied by the user.

1.1   WHAT YOU GET WITH THE EVM

- EVM board
- 3.5" disk titled 'Debug - EVM'
- Debug - EVM Manual
- DSP56002 User's Manual
- DSP56000 Family Manual
- DSP56002 Data Sheet
- CS4215 Data Sheet
- EVM Schematics
- Quick Start document
- The main board documentation is in the form of a READ.ME file on the EVM Software disk.

1.2   WHAT YOU NEED TO SUPPLY

- A PC (386 class or higher) with 2Mbytes of memory, a 3.5" floppy disk drive, and a serial port capable of 19,200 baud
- An RS-232 cable (DB-9 male to DB-9 female)
- A power supply, between 7 V and 9 V @ 700 mA (a.c. or d.c.)
- In order to use the demo, an audio source, headphones, and cables (with 1/8" stereo plugs) to connect into the audio part of the board.

2.   INSTALLATION PROCEDURE

Insert the "Debug - EVM' disk into the floppy disk drive, ensure that you are have selected the correct drive, and type '**install**'

You will be asked to specify the source drive, which will be your floppy disk drive, and to specify the destination drive, which should be your hard drive.

A directory will then be created on the hard drive called 'EVM'. This directory will contain all the files you require to use the DSP56002EVM.

The EVM software is now installed.

3. RUNNING THE DEMO

The EVM software includes a demonstration program. This demonstration is designed to show the advantage of 24-bit DSPs over 16-bit DSPs. The following section gives step by step instructions on how to run this demo.
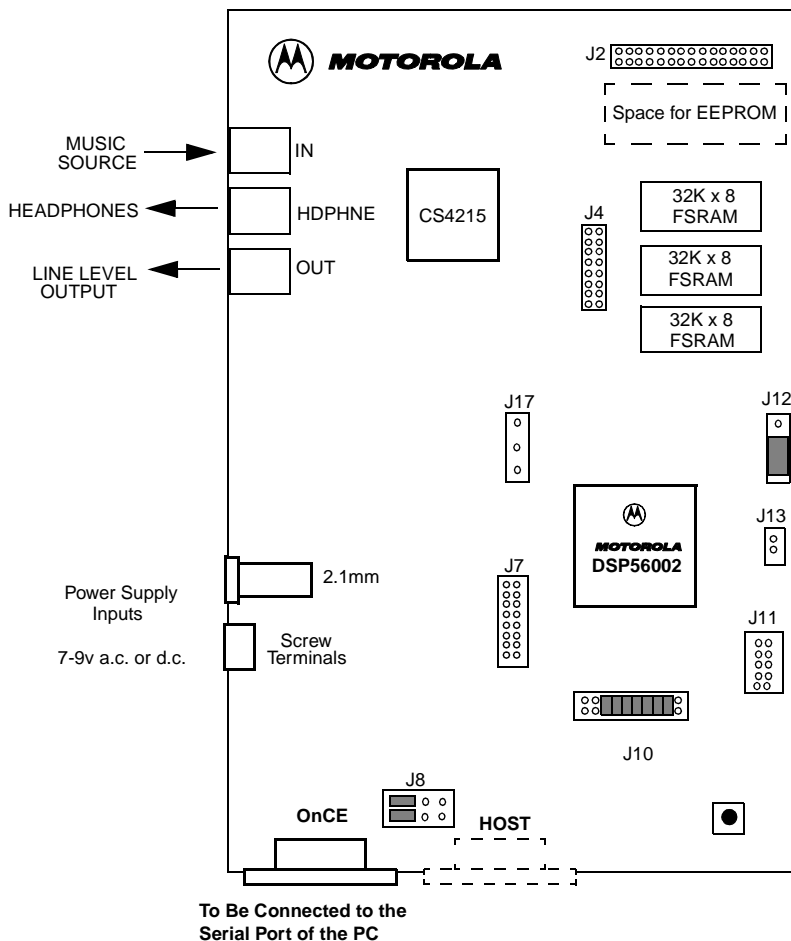

Figure 1: MAIN FEATURES OF THE EVM

3.1 CONNECTING THE BOARD FOR THE DEMO

When you receive the board some jumpers should already have been fitted on J8, J10, and J12. Please ensure that these jumpers have been fitted as shown in the diagram above.

Use the RS-232 cable to connect the PC's serial port to the DB-9 connector labelled 'OnCE' on the EVM. This will enable the board to be controlled from the PC.

To run the demo you need a music source (with a phones output) which must be connected to the stereo input port labelled 'IN' and a pair of headphones connected to the

port labelled 'HDPHNE'.

Connect the power supply to the board using either the 2.1mm jack plug, or the screw terminals.

When you switch on the power supply, the green LED on the board should light.

3.2 STARTING THE DEMO

To start the demo, first start the music source and put on the headphones. Then simply type '**demo**' from the EVM directory. Full details of how to work the demo, and what it demonstrates will be displayed on screen.

After the instruction page, the demo will start, and the graphical user interface (GUI) will appear on the screen. Details on the GUI will follow in Section 4.3.

NOTE: If a message appears to say that the GUI cannot communicate with the board, try changing the position of the jumpers on J8, as shown in the diagram below.

Figure 2: ALTERNATIVE CONFIGURATION OF J8

A number of commands will then be executed. You will notice that the red LED is lit during the execution of these commands. This indicates that the DSP is in DEBUG mode.

The demo can then be controlled using the methods described in the instruction page. i.e.
    To hear the input signal with the added 60 Hz tone, briefly connect J17 pin 1 to ground.
    To hear the result of the filter with the 24-bit coefficients, briefly connect J17 pin 3 to ground.
    To hear the result of the filter with the same coefficients, but rounded to 16-bits, briefly connect J17 pin 2 to ground.
It is suggested that you touch the relevant pin on J17 to pin 16 (bottom right hand corner) of J7.

Freescale Semiconductor, Inc.

J17

J7

1 ($\overline{\text{IRQA}}$)
2 ($\overline{\text{IRQB}}$)
3 ($\overline{\text{NMI}}$)

16 (GND)

Figure 3: DETAILS OF PINS REQUIRED TO RUN DEMO

The reason that the contrast between the two filters is so vast is due to the fact that with 16-bit coefficients, it is impossible to place the notch of the filter exactly on 60 Hz. It is slightly offset and will therefore miss the noise. With the 24-bit coefficients, it is possible to place the filter on exactly 60 Hz.

3.3   STOPPING THE DEMO

To stop the execution of the demo, type **force r** in the COMMAND window in the bottom left corner of the screen. To exit the GUI, type **quit**.

4.   A SIMPLE PROGRAM - WORKED EXAMPLE 1

The following section contains a worked example detailing how to develop a very simple program for the DSP. It will demonstrate the form of assembly programs, give instructions on how to assemble programs, and show how the GUI can be used to verify the operation of, and, if necessary, to debug the program.

4.1   WRITING THE PROGRAM

The program can be edited using one of a large number of standard editors, e.g MS-DOS edit, Turbo C editor, EMACS, etc. It is also possible to use a word processor if it has a 'save as text' option.

The following program will perform the very simple task of adding two numbers together.

> NOTE: It is important to remember that the 56000 family of processors use fractional arithmetic. Please read **DSP56000 Family Manual Section 3.3: Data Representation and rounding** if you are unfamiliar with the DSP56000 family.

> NOTE: A semi-colon (;) comments to the end of the line.
>   Labels must be left justified

```
;*********************************************************************
;A SIMPLE PROGRAM
;*********************************************************************
;THIS SIMPLE PROGRAM WILL ADD TWO NUMBERS
;*********************************************************************
;Y MEMORY
;*********************************************************************
            org  y:$0              ;instructs the assembler that we
                                   ;are referring to Y memory starting
                                   ;at location 0
input1      dc   $1234             ;y:input1 is defined as a $1234
                                   ;$ indicates a hexadecimal value
result      ds   1                 ;reserve a single word of space in
                                   ;y memory. label it 'result'
;*********************************************************************
;X MEMORY
;*********************************************************************
            org  x:$0              ;instruct the assembler that we are
                                   ;now referring to X data memory,
                                   ;starting at location 0
input2      dc   $2345             ;x:input2 is defined as $2345

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

```
;PROGRAM
;*************************************************************************

                org  p:0                    ;put following program in program
                                             ;memory starting at location 0
                jmp  begin                   ;p:0 is the reset vector i.e where
                                             ;the DSP looks for instructions
                                             ;after a reset

                org  p:$40                   ;start the main program at p:$40
                                             ;above the main interrupt vectors
begin           move y:input1,y0             ;load input1 into register y0
                move x:input2,a              ;move input2 into accumulator a
                add  y0,a                    ;add input1 to input 2
                move a,y:result              ;store the result at location
                                             ;y:result

                jmp  *                       ;this is equivalent to
                                             ;label        jmp   label
                                             ;and is therefore a never-ending,
                                             ;empty loop

;*************************************************************************
;END OF THE SIMPLE PROGRAM
;*************************************************************************
```

> NOTE: For more information on interrupt vectors refer to **DSP56000 Family Manual, section 7.3: Exception Processing State (Interrupt Processing)**

Once you have typed in this file, save it as add2num.asm and quit the editor.

4.2   ASSEMBLING THE PROGRAM

To assemble the program you have written, type **asm56000 -a -b -l add2num.asm**
Provided there are no errors, this will create 2 additional files:

```
add2num.cld
add2num.lst
```

The .cld file is the assembled version of the program, and this is what will be downloaded onto the device. The .lst file is the list file which gives full details of where program and data will be placed in the DSP memory.

If errors are reported, recheck the source (.asm) file.

4.3   INTRODUCTION TO THE GUI

This section will give a brief introduction to the GUI, detailing only that which is required to work through the example. Full details of the GUI can be found in the Debug - EVM manual.

![MOTOROLA logo]

Freescale Semiconductor, Inc.

MOTOROLA

Digital Signal Processing Division

To start up the GUI, type '**evm56K**'. The display you will see should be similar to the figure below.

```
 ┌DATA──┬HEX─┐                    ┌REGISTERS─┬HEX─┐
 │                  │                 │                    │
 │                  │                 │                    │
 │                  │                 │                    │
 │                  │                 │                    │
 └──────────────────┘                 │                    │
                                       │                    │
 ┌UNASSEMBLE─────────┐                │                    │
 │                  │                 │                    │
 │                  │                 │                    │
 │                  │                 │                    │
 │                  │                 │                    │
 │                  │                 │                    │
 │                  │                 │                    │
 └──────────────────┘                 │                    │
                                       │                    │
 ┌COMMAND──────────┐                  │                    │
 │EVM>              │                  │                    │
 │                  │                 │                    │
 └──────────────────┘                 └────────────────────┘

 ┌MENU─┐
```
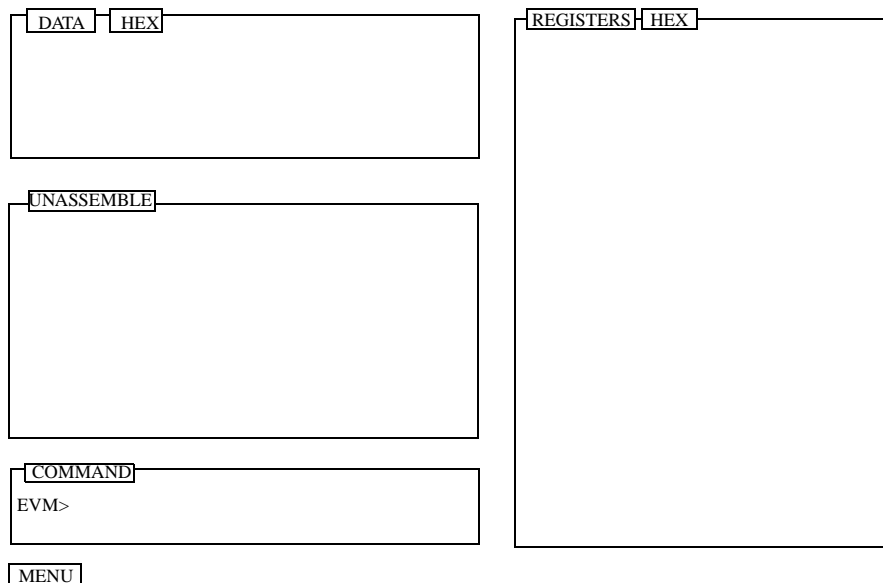
Figure 4: THE GUI

The DATA window, shown in the top left corner displays the data. To display the contents of X data memory, starting at location x:0, click in the COMMAND window and type: **display x:0**.

The radix in which the data is shown can be changed by clicking the box which is shown as containing the word HEX in the diagram above. Data can also be displayed in a graphical form. To do this type: **display x:0 -graph**. To change back to text type: **display x:0 -text**.

The UNASSEMBLE window shows an unassembled version of the contents of program memory. The next instruction to be executed will be highlighted.

As already shown the COMMAND window is where OnCE commands (i.e the controlling commands) are entered.

The REGISTERS window shows the contents of the registers of the ALU (Arithmetic Logic Unit) and the AGU (Address Generation Unit).

4.4   VERIFYING AND DEBUGGING PROGRAMS

To load the add2num program developed earlier, click in the command window and type: **load add2num**.

NOTE: The previous contents of the memory which are not overwritten will remain unchanged

The instruction at P:0 will be highlighted as this will be the first instruction to be executed. However, before we start to execute the program we should check that the values we expect to be in data memory are there. To do this type:

display y:input1
and then
display x:input2

NOTE: The data you have asked to be displayed will be the first value in the data window. If it looks incorrect, check the radix.

To step through the program, type **step** at the command prompt.

SHORTCUTS: Instead of typing in the entire command, type the start of the command, and by pressing the space bar, the GUI will complete the remainder. To repeat the last command, simply press return.

As you step through the code, you will see the instructions having an effect on the registers shown in the REGISTERS window.

Once you have stepped through the program, ensure that the program has executed correctly, by checking that y:result contains the value $3579.

Stepping through the program like this is good for short programs, but is impractical for large complex programs. The way to debug large programs is to set **breakpoints**. These are user defined points at which execution of the code will stop allowing the user to step through the section of interest.

To set a breakpoint in the add2num to check the result in accumulator a before it is moved into Y memory, the command is: **break p:$43**. You will see the breakpoint indicated in the unassemble window.

To point the DSP back to the start point of the program, type: **change pc 0**. This changes the program counter such that it is pointing to the reset vector. To start the program running, the command is **go**.

The DSP will stop when it reaches the breakpoint, you will then be able to step through the remainder of the code.

4.5   EXITING THE GUI

To exit from the GUI, type quit at the command prompt.

5. ANOTHER SIMPLE EXAMPLE - WORKED EXAMPLE 2

The following section will give details of another worked example. This one is slightly more complicated than the previous one and will demonstrate the use of the addressing modes, and how to configure the external memory.

5.1 WRITING THE PROGRAM

This file will take two lists of data, one in internal X memory, one in external Y memory, and calculate the sum of the products of the two lists. Calculating the sum of products is the basis for many DSP functions, therefore the DSP has a special instruction (the mac instruction) which multiples two values and adds the result to the contents of an accumulator.

NOTE: You do not need to type this program, it is provided as QS_EX2.ASM

```
;**********************************************************************
;WORKED EXAMPLE 2
;**********************************************************************
PBASE           EQU     $200            ;instruct the assembler to replace
                                        ;every occurrence of PBASE with $200
XBASE           EQU     $0              ;used to define the position of the
                                        ;data in X memory
YBASE           EQU     $200            ;used to define the position of the
                                        ;data in Y memory

BCR             EQU     $FFFE           ;address, in X memory, of the
                                        ;bus control register

                org     x:XBASE         ;internal x memory
list1           dc      $475638,$738301,$92673A,$898978,$091271,$F25067
                dc      $987153,$3A8761,$987237,$34B852,$734623,$233763
                dc      $F76756,$423423,$324732,$F40029

                org     y:$YBASE        ;external y memory
list2           dc      $F98734,$800000,$FEDCBA,$487327,$957572,$369856
                dc      $247978,$8a3407,$734546,$344787,$938482,$304F82
                dc      $123456,$657784,$567123,$675634

                org     p:0
                jmp     begin           ;reset vector

                org     p:PBASE         ;external program memory
begin           clr     a               ;clear accumulator a
                move    #list1,r0       ;pointer to the start of list1
                move    #list2,r4       ;pointer to the start of list2
                movep   #0,x:BCR        ;want zero wait state accesses to
                                        ;the external memory
                                        ;movep (move to peripheral) allows
                                        ;you to move immediate data into a
```

```
                                     ;memory location
          move     x:(r0)+,x0  y:(r4)+,y0
                                     ;load value in X memory pointed to
                                     ;by the contents of r0 into x0 and
                                     ;post-increment r0
                                     ;load value in Y memory pointed to
                                     ;by r4 into y0 and post-increment
                                     ;r4
          do       #15,endloop
          mac      x0,y0,a   x:(r4)+,x0    y:(r0)+,y0
                                     ;parallelism allows arithmetic
                                     ;instruction, TWO data loads
                                     ;and two register post-increments
                                     ;to be done in one instruction
                                     ;cycle.
endloop         jmp       *
;***********************************************************************
```

NOTE: The `jmp *` instruction is NOT inside the loop.
NOTE: There is a bug in this program.

5.2   EXTERNAL MEMORY CONFIGURATION

There is an option to configure the external memory of the board in two different ways. Full details of the corresponding memory maps are in the READ.ME file.

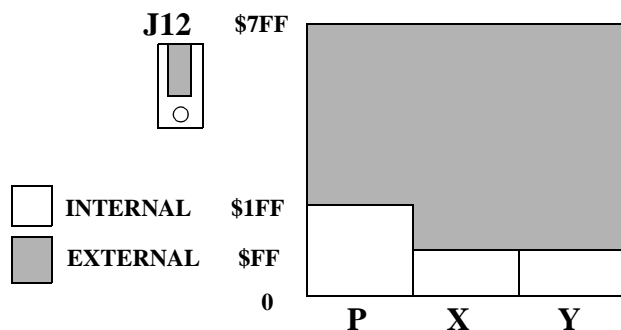The external memory map is controlled by the position of the jumper on J12.

Figure 5: UNIFIED MEMORY MAP

J12

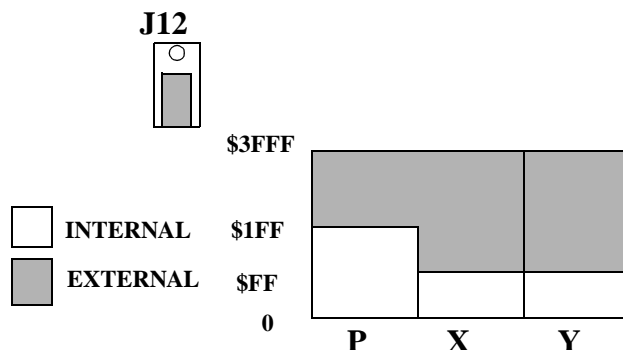|	
INTERNAL	$1FF
EXTERNAL	$FF

$3FFF

0

P   X   Y

Figure 6: PARTITIONED MEMORY MAP

NOTE: For more details on memory maps see **Section 3.2** in the **DSP56002 User's Manual**.

With the memory configured as shown in Figure 5, external memory is regarded as one large block. There is no separation between X, Y, or P. In this configuration X:$1000, Y:$1000, and P:$1000 are treated as the same memory cell.
In the second configuration, the memory is divided into two areas. Half of the external memory is mapped to Y, and the remainder is unified between P and X memory.

In this example we want to make use of external Y memory, and external P memory, therefore we require the partitioned memory map. Place the jumper on J12 accordingly.

5.3   DEBUGGING THE PROGRAM

Assemble and run the program as before. i.e:
asm56000 -a -b -l QS_EX2.asm
evm56k
     then
          put a breakpoint in the code at the `jmp  *` instruction
          and type **go** at the COMMAND prompt.

If the program was working correctly the result in accumulator a would be :
     $ FE 9F2051 6DFCC2
However there is a bug in the program, which you may have already spotted. Debug the program!

     NOTE: If you can't find the bug, see the solution at the bottom of the next page.

MOTOROLA

Freescale Semiconductor, Inc.

Digital Signal Processing Division

SOLUTION: Registers r0 and r4 are used incorrectly inside the do loop. r0 should be used to point to the list which is in X memory, r4 should be used to point to the list in Y memory.

MOTOROLA

Digital Signal Processing Division

6. FILTERING AUDIO - WORKED EXAMPLE 3

The following section contains a more complex worked example. This will show how to develop an application using the on-board codec, and the codec configuration file supplied with the EVM software.

6.1 APPLICATION REQUIREMENTS

The end product of this worked example will be an assembly program which will take an audio signal, through the codec, subject it to a simple low pass filter, and output the result back through the codec. Each step of the development will be documented.

6.2 CONFIGURING THE CODEC

The CS4215 is a sophisticated device and is therefore relatively complex to configure. An attempt has been made to isolate the user of the EVM from this complexity by including the files 'ada_init.asm' and 'txrx_isr.asm' with the software.

The file 'ada_init' can be used to initialize the codec. It has been set up such that the parameters can be changed by the user easily by changing one of a few control words. The code will currently be in the format necessary for the demo with the 60 Hz filter. It will remain the same for this demonstration.

> NOTE: When including this file in your program, you MUST ensure that there is no conflict in memory. e.g. The ada_init.asm program uses locations x:0..9, the users program should therefore not use these locations.

6.3 VERIFYING THE INPUT AND OUTPUT OF DATA

The best and easiest way to verify that the codec is being configured correctly and that the data is being received and transmitted by the DSP correctly, is to simply pass the data straight through without any processing at all. The following program (which you will find as QS6_3.asm) will do this.

```
;************************************************************************
;VERIFYING THE OPERATION OF THE CODEC AND THE SSI
;THIS PROGRAM WILL CONFIGURE THE CODEC, ACCORDING TO THE PARAMETERS IN THE
;ADA_INIT.ASM FILE, AND RECEIVE AND THEN TRANSMIT AUDIO DATA WITHOUT
;AFFECTING IT.
;************************************************************************

START           EQU  $40

                ORG  P:0
                jmp  START

                ORG  p:$000c
```

```
        jsr  ssi_rx_isr        ;SSI receive data
        jsr  ssi_rx_isr        ;SSI receive data with exception
        jsr  ssi_tx_isr        ;SSI transmit data
        jsr    ssi_tx_isr      ;SSI transmit data with exception
                               ;These interrupt service routines
                               ;contained in the file
                               ;tx_rx_isr.asm.
                               ;See 56002 manual for information
                               ;on interrupts


        ORG    P:START

        movep #$261009,x:PLL   ;these labels are defined in the
        movep #0,x:BCR         ;ada_init.asm program
        ori   #3,mr            ;set bit0 and bit1 in the mode
                               ;register i.e. mask interrupts
        movec #0,sp            ;clear hardware stack pointer
        move  #0,omr           ;mode 0;enabl int. PRAM;rst=0000
        move  #$40,r6          ; initialize stack pointer
        move  #-1,m6           ; linear addressing
                               ; these are used by the isrs

        include  'ada_init.asm'
                               ;initialize the codec
TONE_OUTPUT   EQU   HEADPHONE_EN+LINEOUT_EN+(4*LEFT_ATTN)+(4*RIGHT_ATTN)
TONE_INPUT    EQU   MIC_IN_SELECT+(15*MONITOR_ATTN)

loop_1
        jset  #2,x:SSISR,*     ;wait for frame sync. to pass
        jclr  #2,x:SSISR,*     ;wait for frame sync

        move  x:RX_BUFF_BASE,a ;move left sample into acc. a
        move  x:RX_BUFF_BASE+1),b
                               ;move right sample into acc. b

        jsr    process_stereo  ;jump to the subroutine which will
                               ;process the samples

        move  a,x:TX_BUFF_BASE ;move a into position from which
                               ;it can be transmitted
        move  b,x:TX_BUFF_BASE+1
                               ;move b into the position from
                               ;which it can be transmitted
        move  #TONE_OUTPUT,y0  ;set up control words
        move  y0,x:TX_BUFF_BASE+2
        move  #TONE_INPUT,y0
        move  y0,x:TX_BUFF_BASE+3

        jmp    loop_1

process_stereo
```

```
            nop
            nop
            nop
            rts
```

lementation of digital filters can be found in most DSP textbooks.

The filter described below will be implemented in the QSFILTER.ASM.

Filter order: M=2
No. of sections   N=2

Transpose Direct Form I cascade of 2nd order sections:

$$H(z) = \prod_{i=1}^{N} \frac{b_{i0} + b_{i1}z^{-1} + b_{i2}z^{-2}}{1 + a_{i1}z^{-1} + ai2z^{-2}}$$
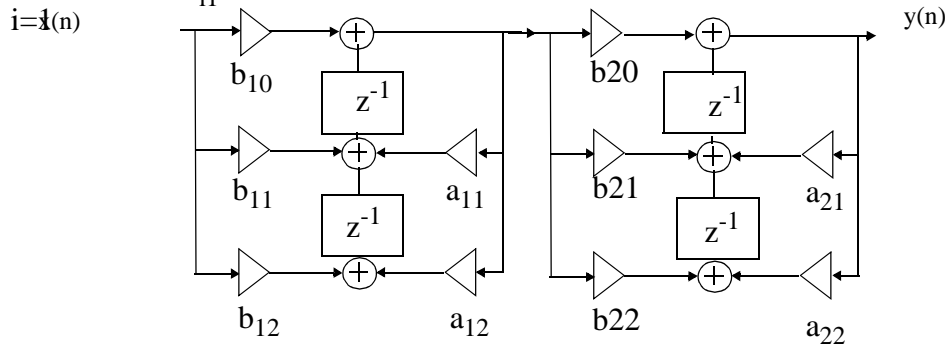

Figure 7: EXAMPLE OF A DIGITAL FILTER

For a low pass filter with a cut-off of 1 kHz, the coefficients by the filter design package are:

```
            b10 =   0.00371753
            b11 =   0.00741518
            a11 =   0.83384359
            b12 =   0.00370753
            a12 = -0.34867418

            b20 =   0.00485158
            b21 =   0.00970316
            a21 =   0.86615109
            b22 =   0.00485158
            a22 = -0.38555753
```

For More Information On This Product,
Go to: www.freescale.com

6.4   IMPLEMENTING THE FILTER IN THE DSP

The code which implements the filter is QSFILTER.ASM. The actual implementation is such that the audio is passed straight through for about the first ten seconds, then is filtered for ten seconds, then straight through, then filtered, and cycles like this until execution of the program is halted.

```
;************************************************************************
;QSFILTER : LOW PASS FILTERING DEMO FOR DSP56002EVM.
;WILL PASS AUDIO STRAIGHT THROUGH FOR APPROXIMATELY TEN SECONDS, THEN FILTER
;FOR TEN SECONDS, THEN STRAIGHT THROUGH, ETC..ETC..ETC
;************************************************************************
START           EQU $40


                ORG  X:$10
coefd           dc   0.00370753       ;b10
                dc   0.5              ;scaling factor
                dc   0.00741518       ;b11
                dc   0.83384359       ;a11
                dc   0.00370753       ;b12
                dc   -0.34867418      ;a12

                dc   0.00485158       ;b20
                dc   0.5              ;scaling factor
                dc   0.00970316       ;b21
                dc   0.86615109       ;a21
                dc   0.00485158       ;b22
                dc   -0.38555753      ;a22

                ORG  y:$10
rtdelay         bsc  4,$0             ;define 4 locations for rtdelay,
                                      ;initialize as 0
ltdelay         bsc  4,$0             ;define 4 locations for ltdelay,
                                      ;initialize as 0
tempstore       ds   1               ;define a single location for the
                                      ;temporary storage.

                ORG  P:0
                jmp  START

                ORG p:$000c
                jsr  ssi_rx_isr       ;SSI receive data
                jsr  ssi_rx_isr       ;SSI receive data with exception
                jsr  ssi_tx_isr       ;SSI transmit data
                jsr    ssi_tx_isr     ;SSI transmit data with exception
                                      ;These interrupt service routines
                                      ;contained in the file
                                      ;tx_rx_isr.asm.
                                      ;See 56002 manual for information
                                      ;on interrupts
```

```
                    ORG     P:START


                    movep   #$261009,x:PLL    ;these labels are defined in the
                    movep   #0,x:BCR          ;ada_init.asm program
                    ori     #3,mr             ;set bit0 and bit1 in the mode
                                              ;register i.e. mask interrupts
                    movec   #0,sp             ;clear hardware stack pointer
                    move    #0,omr            ;mode 0;enabl int. PRAM;rst=0000
                    move    #$40,r6           ; initialize stack pointer
                    move    #-1,m6            ; linear addressing
                                              ; these are used by the isrs


                    include  'ada_init.asm'
                                              ;initialize the codec

TONE_OUTPUT         EQU     HEADPHONE_EN+LINEOUT_EN+(4*LEFT_ATTN)+(4*RIGHT_ATTN)
TONE_INPUT          EQU     MIC_IN_SELECT+(15*MONITOR_ATTN)

loop_1
                    do      #$60,wait         ;do for approximately 10 seconds
                    do      #$FFF,wait_inner
                    jset    #2,x:SSISR,*      ;wait for frame sync. to pass
                    jclr    #2,x:SSISR,*      ;wait for frame sync

                    move    x:RX_BUFF_BASE,a  ;move left sample into acc. a
                    move    x:RX_BUFF_BASE+1),b
                                              ;move right sample into acc. b

                    move    a,x:TX_BUFF_BASE  ;move a into position from which
                                              ;it can be transmitted
                    move    b,x:TX_BUFF_BASE+1
                                              ;move b into the position from
                                              ;which it can be transmitted
                    move    #TONE_OUTPUT,y0   ;set up control words
                    move    y0,x:TX_BUFF_BASE+2
                    move    #TONE_INPUT,y0
                    move    y0,x:TX_BUFF_BASE+3
wait
                    nop
wait_inner


                    do      #$60,wait2        ;do for approximately 10 seconds
                    do      #$FFF,wait2_inner
                    jset    #2,x:SSISR,*      ;wait for frame sync. to pass
                    jclr    #2,x:SSISR,*      ;wait for frame sync

                    move    x:RX_BUFF_BASE,a  ;move left sample into acc. a
                    move    x:RX_BUFF_BASE+1),b
                                              ;move right sample into acc. b

                    jsr     process_stereo    ;jump to the subroutine which will
```

18

```
                                    ;process the samples

            move    a,x:TX_BUFF_BASE    ;move a into position from which
                                        ;it can be transmitted
            move    b,x:TX_BUFF_BASE+1
                                        ;move b into the position from
                                        ;which it can be transmitted
            move    #TONE_OUTPUT,y0     ;set up control words
            move    y0,x:TX_BUFF_BASE+2
            move    #TONE_INPUT,y0
            move    y0,x:TX_BUFF_BASE+3
wait2
            nop
wait2_inner

            jmp     loop_1




process_stereo
            move    #ltdelay,r0         ;set up pointer to left delay
                                        ;storage
            jsr     filter              ;filter the left sample
            move    a,y:tempstorage     ;filtered value in a, store it
            move    b,a                 ;move the right sample into acc. a
            move    #rtdelay,r0         ;set up pointer to right delay
                                        ;storage
            jsr     filter              ;filter the right sample
            move    a,b                 ;move filtered right sample into b
            move    y:tempstorage,a     ;move filtered left sample into a
            rts

filter
            move    #LINEAR,m0          ;set up addressing mode
            move    m0,m4               ;set up the addressing mode
            ori     #$08,mr             ;enable the scaling mode

            do      #2,stage
            asr     a  #<coefs,r4       ;set up pointer to the coefficients
                                        ;shift the sample right
            move    a,x0    y:(r0)+,a
                                        ;x0=ip a=delayed z-2
            asr     a       x:(r4)+,x1
                                        ;a= (z-2)/2   x1= bi0
            macr    x1,x0,a x:(r4)+,x1  y:(r0)-,y0
                                        ;a=(z-2)/2+(bi0*ip)=op
                                        ;x1 = 0.5 y0=delay z-1
            mpy     y0,x1,a a,y1        x:(r4)+,x1
                                        ;a=(z-1)/2  x1= =a11
            mac     x0,x1,a x:(r4)+,x1
                                        ;a=(z-1)/2+(bi1*ip) x1= ai1
```

19

```
        macr   y1,x1,a x:(r4)+,x1
                                ;a=(z-1)/2+(bi1*ip)+(ai1*op)
                                ;x1 = bi2
        mpy    x1,x0,a x:(r4)+,x1  a,y:(r0)+
                                ;a=(bi1*ip) x1=ai2 save previous a
        macr   y1,x1,a             ;a=(bi1*ip)+(ai2*op)
        tfr    y1,a    x:(r4)+,x0  a,y:(r0)+
stage
        and    #$F3,mr
        rts


        include 'txrx_isr.asm'

        end
```

7. CONCLUSION

You have now come to the end of the quick start document, and hopefully it has given you a good basic understanding of the EVM. For further information please refer to the manuals provided.