Motorola
Digital Signal
Processors

Conference Bridging in the Digital Telecommunications Environment Using the Motorola DSP56001/2

by
Ralph Weir
DSP Applications– East Kilbride, Scotland

Table of Contents

Illustrations

List of Tables

SECTION 1

Introduction

"To illustrate the performance available from this software, a 20-MHz DSP56000/ DSP56001 is capable of implementing 10 six-way conference bridges, a single 71-way bridge, or even 17 three-way bridges."

This application note describes one possible implementation of a conference bridge for a modern digital telephone exchange. Conference bridging is a feature available on many modern analog or digital telephone exchanges. It allows conversation among three or more subscribers and provides an arbitration scheme between subscribers who are involved in the conference.

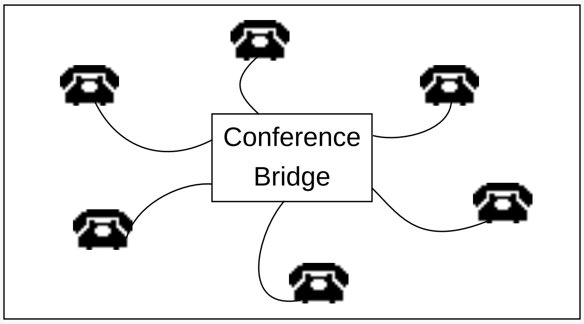


Figure 1-1 *A Six-Way Conference Bridge*

Although Figure 1-1 shows the conference bridge as a separate entity; it is usually implemented as part of a larger system, such as a PABX. This concept allows a conversation to occur among several subscribers in a controlled manner.

There are a number of possible schemes for implementing this concept. In the analog world, a common approach is to simply sum all incoming signals, giving one output. However, this method degrades the signal quality, because the signal-to-noise ratio (SNR) of the signal is significantly reduced.

A digital system can emulate this approach. However, a small conference bridge commonly uses a 'single speaker' algorithm that passes only one input signal to the listeners. In this case, the speaker is not a listener and no signal passes back; signal paths within a handset are relied on to provide a 'comfort level' of signal feedback.

This approach has many advantages; the most significant one is that the speaker's signal passes through unaltered, so the SNR does not change. This method is common for small bridges of no more than 10 channels. Larger bridges may use a multiple speaker algorithm.

A multiple speaker algorithm must decide which subscriber should be the speaker. There are a number of choices; most algorithms use the 'loudest speaker' algorithm, which designates the loudest speaking subscriber as the current speaker. The implementation of this approach is very straightforward.

To determine the loudest speaker, software uses an integrator to find the power level of the incoming signal. This power level is then used as the 'loudness', rather than the instantaneous level of the signal which slows the bridge's switching rate, giving a more aurally-pleasing result.

One unusual feature of the implemented software is the provision of a 'chairman' facility. One channel of the bridge is given priority over all others, giving the conference leader the ability to control the conversation without becoming the loudest speaker.

1.1 System Overview

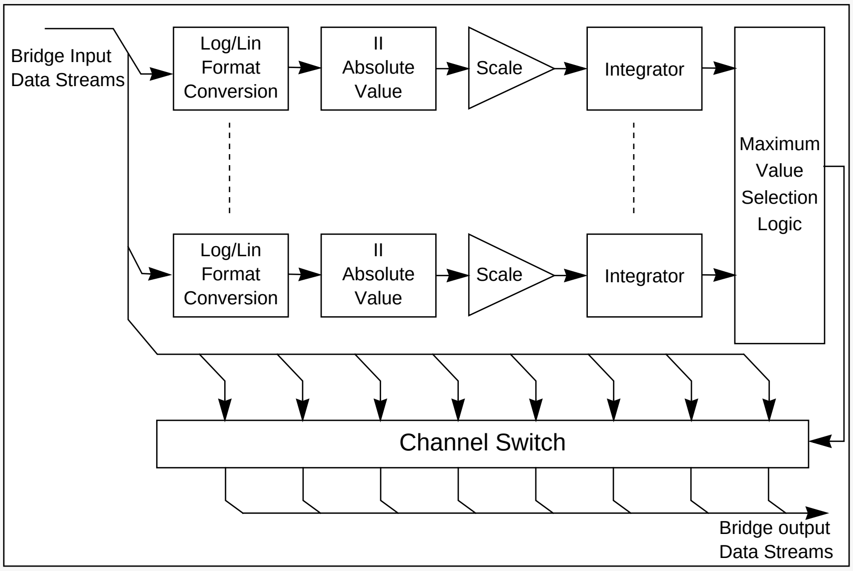
This application note describes a software implementation of a conference bridge written for the DSP56001/2. This device has a 24-bit architecture, a feature which greatly simplifies the software required. This processor is also available in different speed ratings.

The implemented software is comprised of various simple functional blocks, as shown in Figure 1-2. A detailed description of each stage is presented; it should be noted that some stages are merged to improve overall system throughput.

One variable of global importance is the number of channels, n_chans. This variable affects all sections of the bridge since it controls the number of channels implemented by the software.

The software currently supports up to eight channels per bridge without modification; obviously, all system variables are common between bridges. Modifying the software for larger bridges or for dynamically configurable bridges according to the requirements of the system controller is relatively simple. The only requirement for more bridges is additional internal or external memory for data storage and I/O.

To illustrate the performance available from this software, a 20-MHz DSP56000/DSP56001 is capable of implementing 10 six-way conference bridges, a single 71-way bridge, or even 17 three-way bridges. Obviously, it is possible to mix and match; thus the system may include different sized bridges or be reconfigured dynamically according to system requirements.



Bridge Input Data Streams
Log/Lin Format Conversion
II Absolute Value
Scale
Integrator
Log/Lin Format Conversion
II Absolute Value
Scale
Integrator
Maximum Value Selection Logic
Channel Switch
Bridge output Data Streams

Figure 1-2 *System Block Diagram*

1.2 Input/Output (I/O) Considerations

Input is taken from tables of data, representing the I/O data streams to the subscribers, held in logarithmic format. In the DSP56001/2 implementation, these tables may be updated from the SSI, a powerful serial interface capable of direct connection to the standard 32-slot serial data streams commonly used for communications within telephone exchanges. Alternatively, an external parallel serial conversion device such as the Mitel 8920 may be used, which has the advantage of adding additional serial ports to the DSP. Also, the MT8920 is table driven, resulting in a reduction in the number of interrupts the system must handle. This reduction may be advantageous in situations where the DSP is performing other tasks, such as dual-tone multi-frequency (DTMF) detection or generation.

The output data tables share the I/O memory space with the input tables; data is returned to these tables in logarithmic format for retransmission. One data table is required for each bridge, comprising one word per bridge channel. ■

SECTION 2

Channel Processors

2.1 Log/Linear Conversion

"Using the DSP56001/2 to implement conference bridging results in an additional advantage — the system controller may then set up the bridge to the exchange's specific requirements..."

Data within a digital telecommunications network is held in 8-bit logarithmic format to reduce the number of bits required to be transmitted while still giving an acceptable signal-to-noise (SNR) ratio. Two standards are used: Europe uses A-law coding and the US uses Mu-law coding. The formats are similar, but have differences that must be considered when converting to linear data.

The data conversion is actually performed using the lookup tables implemented in the DSP56001/2ROM. There are two tables, one for each coding technique; these are selected by using the system variable 'ctable', which should be $100 for Mu-law and $180 for A-law.

2.2 Absolute Value Calculation

The calculation of absolute value is performed within the log/linear conversion code. Both A-law and Mu-law are 'sign+magnitude' formats, where the most significant bit represents sign and the remaining seven bits form the magnitude. By ignoring the sign bit during format conversion, a positive value is always generated, allowing the quickest possible conversion to an absolute linear value.

2.3 Scaling

By applying a scaling factor to each individual channel of incoming data, different priorities can be assigned to each channel. Also, the integrator used has a very large gain and requires an attenuation stage on its input to prevent arithmetic overflow. These functions are combined in the scale block.

A different scale factor may be used for each channel of a bridge; however, with the current software configuration, equivalent channels in separate bridges are assigned the same priority. For example, if channel 1 of bridge 1 of a two-bridge system is selected as the highest priority channel, it will also be the highest priority channel on bridge 2.

The scale factors are set up in the consecutive system memory locations scale 1. . . scale *N*, where scale *N* corresponds to the scale factor applied to channel *N*.

2.4 Integration

Integration is performed using a two-pole infinite impulse response (IIR) filter. This technique allows the system designer to easily tailor the envelope detector function to the system's requirements, using one of the many standard IIR design techniques.

The coefficients of the filter are held in the memory locations coeff. . . coeff+3. These memory locations must be held in a modulo four memory area; which is any memory block with a base address divisible by four. The integrator requires two words of storage per bridge channel to store the intermediate data values generated within the filter stage.

2.5 Maximum Value Selection Logic

This block controls which of the input channels is actually selected as the current speaker. It has no system variables controlling it and is incorporated on a per-channel basis with the channel processors.

2.6 Channel Switch

This block controls the signal routing after the channel processors choose the current speaker. The channel processor returns a pointer indicating this channel; data from this channel is used as output for all other channels, except itself. The current speaker is, in fact, passed silence; for this, the system variable 'silence' is accessed. This variable contains the constant logarithmic value for silence according to the data format in use at the time. Note that the technique of sharing the I/O table between input and output data allows use of the original unaltered data. The input signal passes through unaltered; thus, the current speaker experiences no degradation in SNR.

2.7 System Performance

The computational load imposed by the bridge is comprised of two parts; one section is an absolute overhead for initializing the bridge; the second part is a per-channel overhead comprised of the computation involved in each channel processor. This is expressed in terms of Icyc (instruction cycles) when referring to the 8-kHz sampling rate used in telecommunications: 1687 Icyc are available with the standard 27-MHz processor, whereas 2500 Icyc are available with the 40-MHz DSP56002. In other words, an 8-kHz sample rate corresponds to a 1250 µs sample period and the above devices have 74.1 and 50 ns instruction cycle times, respectively. (See Table 2-1.)

Table 2-1 *Computational Load for a Conference Bridge*

Processor	Speed	Icycle Times	Icycles Required
DSP56001	20.5 MHz	97.5 ns	1280
DSP56001	27 MHz	74.1 ns	1687
DSP56001	33 MHz	60.6 ns	2062
DSP56002	40 MHz	50 ns	2500

The total load is given by the Eqn. 2-1:

$$B_{cyc} = 22 + 17 * n\_chans \qquad \text{Eqn. 2-1}$$

where: B_{cyc} *is the number of processor instruction cycles required to implement the bridge*

Thus, for a 20-MHz processor, a six-channel bridge would require (22 + 17*6) = 124 instruction cycles or 12.4 µs. Obviously, various bridge configurations are possible, and most applications will require other functions (such as DTMF detection) to be integrated onto the DSP.

2.8 System Variable Summary

Table 2-2 gives a summary of the variables.

Table 2-2 *Variable Summary*

Variable	X/Y	Description
data	X	A block of memory intended for use as storage of the integrator's intermediate data terms. Two words are required per channel of each bridge.
coeff	Y	A four-word modulo storage area containing the coefficients for the integration filter.
ctable	Y	One word containing the base address of the appropriate conversion table. Set to $100 for Mu-law, $180 for A-law.
silence	Y	Silence Code for appropriate logarithmic data format. This is the code equivalent to DC zero.
scale	Y	An eight-word block of scaling coefficients used for the data scalers. Their value depends on channel priority and the integration coefficients; the bigger the value, the higher the priority of the channel.
n_chans	Y	One data word per implemented bridge, giving the number of channels in each bridge.
IO	Y	This is the I/O area for a bridge; requiring one word per channel.

2.9 Control of the Bridge by the System Controller

Using the DSP56001/2 to implement conference bridging results in an additional advantage — the system controller may then set up the bridge to the exchange's specific requirements, regardless of whether these requirements are decided at power-on or are instantaneous. The only requirement is that the system should support some form of access to the DSP56001 data RAM, which can be accomplished through the host port or the unused slots of the SSI. ■

SECTION 3

The Integrator — Theory and Design

3.1 Theory

"The integrator attempts to filter out the envelope functions, leaving only a slowly changing function that can successfully be used in the decision logic."

The function of the integrator is to provide a signal whose output is proportional to the power in the input. However, instantaneous power is of little use because of the nature of a speech signal. A speech signal is composed of two basic components — the speech tone and a modulating envelope. The interaction of these two form the basic elements of speech, called phonemes; every spoken word is comprised of a number of these elements.

Additionally, a sentence is a linguistic structure formed from several words. The speaker tends to apply a secondary envelop to a sentence, causing amplitude differences between word and word gaps; the actual envelope applied depends on dialect and language spoken.

Some idea of the resulting signal is illustrated in Figure 3-1 through Figure 3-3. Figure 3-1 shows a pure tone, Figure 3-2 shows a low-frequency sine wave representing the modulating signal, and Figure 3-3 shows the modulated signal. Although these figures are very simplistic, they show the complex nature of the speech signal.

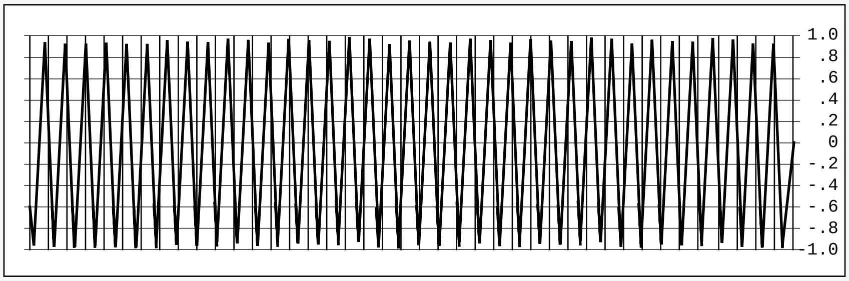


***Figure 3-1** Pure Tone*

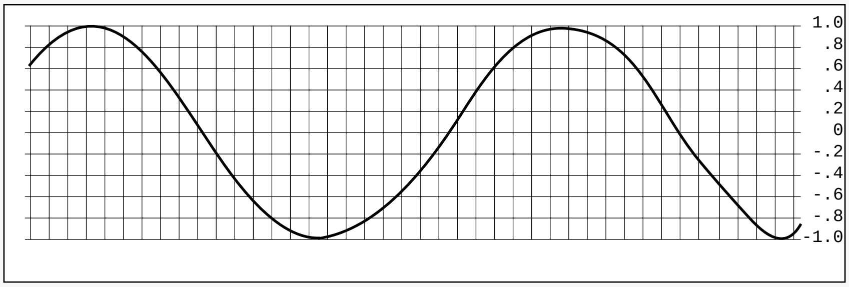


***Figure 3-2** Modulating Function*

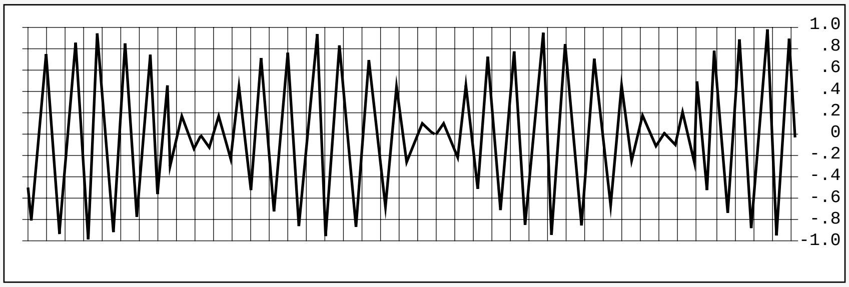


***Figure 3-3** Modulated Signal Representing Speech*

3.2 Design

It would be useless to use instantaneous power as a basis for the 'loudest speaker' algorithm. Due to the various envelope functions the speaker applies to his speech, the instantaneous power is a rapidly changing function, resulting in the bridge switching rapidly between speakers. Some means of damping the bridge's response is required, which is the function of the integrator.The integrator attempts to filter out the envelope functions, leaving only a slowly changing function that can successfully be used in the decision logic.

Unfortunately, choosing integration coefficients involves some trade-off. If the integrator uses too long a period, the bridge will switch too slowly, and an unacceptable switching delay will be introduced.

In view of this trade-off, the use of an adapted infinite impulse response (IIR) filter is preferable to the use of a true integrator. The system designer can then modify the integration coefficients quite easily, using a filter design methodology like the bilinear transform.

Integration of incoming waveform is performed using a two-pole IIR filter. The transfer function, H(z), for this filter is given in Eqn. 3-1:

$$H[z] = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=1}^{N} a_k z^{-k}} \qquad \text{Eqn. 3-1}$$

From this transfer function, a signal flow diagram can be synthesized as shown in Figure 3-4.

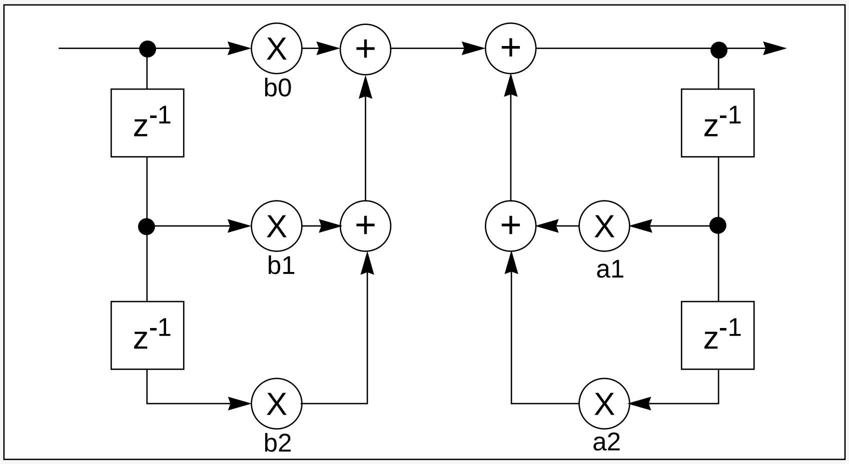


Figure 3-4 *Direct Form I Filter*

The implemented filter has been simplified to require only four multiply-accumulate (MAC) operations and two storage stages.

The first stage of the simplification is the reversal of the a and b stages of the filter, giving the signal flow diagram of Figure 3-4. This is an intermediate step of the simplification process; no overall simplification has been performed, but the delay stages for the a and b stages are adjacent.

Grouping the delay stages allows the delay elements to be combined, which was the object of grouping since no other simplification was gained by the grouping. The transfer function is unchanged by this operation.

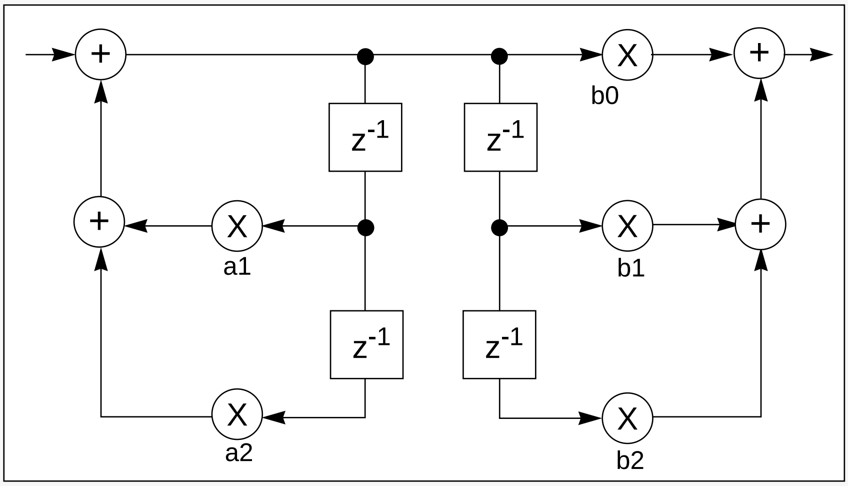


Figure 3-5 *Modified Direct Form I Filter*

This application deals with relative levels as the outputs from the integrator; there is no merit in producing the absolute value as an output. Thus, the integrator's gain is immaterial.

If the numerator of the transfer function in Eqn. 3-1 is factored by b0 the result is the transfer function shown in Eqn. 3-2.

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{b_0 \sum_{k=1}^{N} a_k z^{-k}} \qquad \text{Eqn. 3-2}$$

Other than in terms of gain, this is equivalent to the original filter and may be implemented by the signal flow diagram of Figure 3-4. As illustrated in Figure 3-4, the filter only requires four multiplies now, but implements a gain stage of 1/b0. Since only relative levels are of interest, this gain stage canbeignored,givingtheoptimalformoftheintegrator. ■

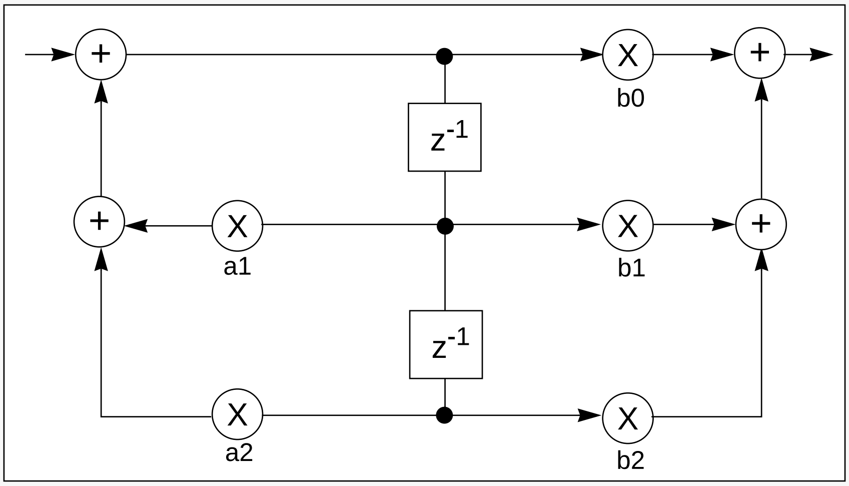


Figure 3-6 *Direct Form II Filter*

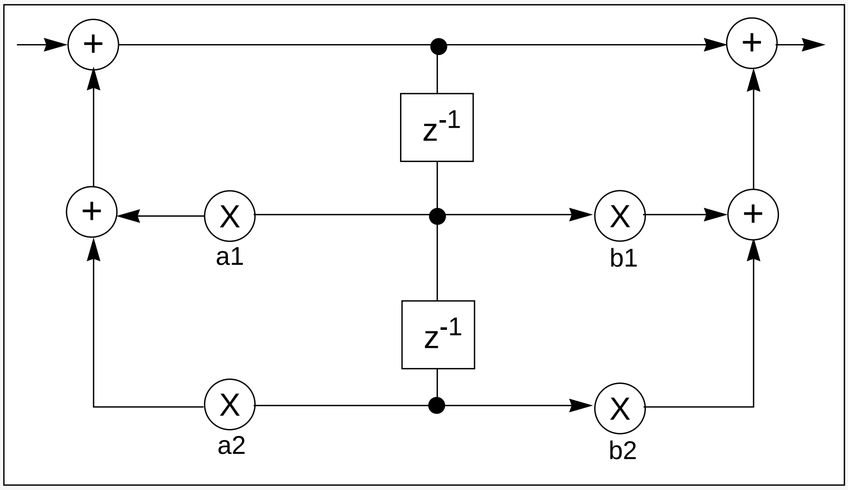


Figure 3-7 *Scaled Direct Form II Filter*

SECTION 4

Calculating the Integrator Coefficients

"...if an analog filter can be specified to perform the integration function, this function can be mapped into the digital domain for implementation by the DSP56001/2."

The signal flow diagram of Figure 3-4 will implement any filter; the final function of the block depends only upon the coefficients used by the filter.

The coefficients of the integrator were produced using a standard design technique—the bilinear transform. This technique converts a standard analog filter to the digital domain. Any standard filter approximation may be used.

The basis of the bilinear transform is the fact that the s-domain, used in the analysis of analog filters, may be mapped onto the z-domain using some mathematical relationship. A number of CAD packages are available for this; however, for those who do not have such a package, an explanation of the procedure will follow.

The s-domain may be mapped as shown in Figure 4-1. In this case, filters (of which the integrator is an example) must have their poles and zeros within the shaded area of the diagram, the left half plane (LHP).

The z-domain is represented in Figure 4-2. The z-plane is not divided into the LHP and RHP, but is represented as a unit circle. Filters must have poles and zeros within the unit circle (the shaded area).

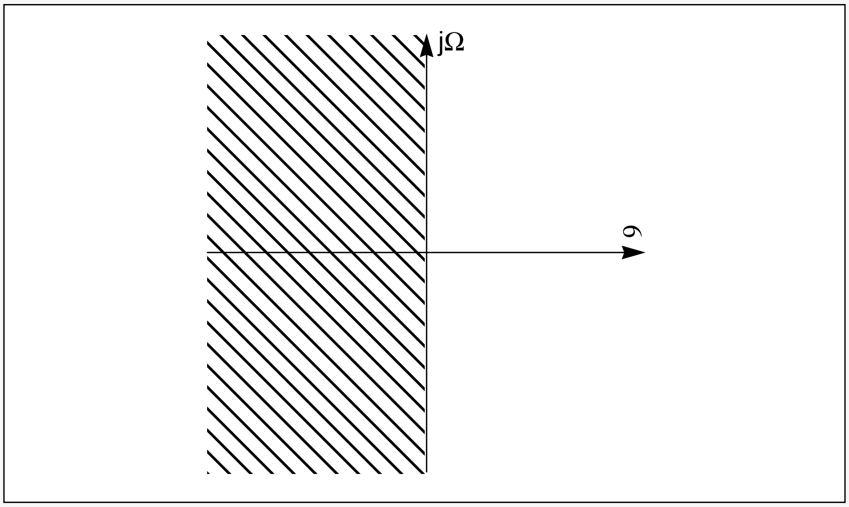


***Figure 4-1** S-Domain*

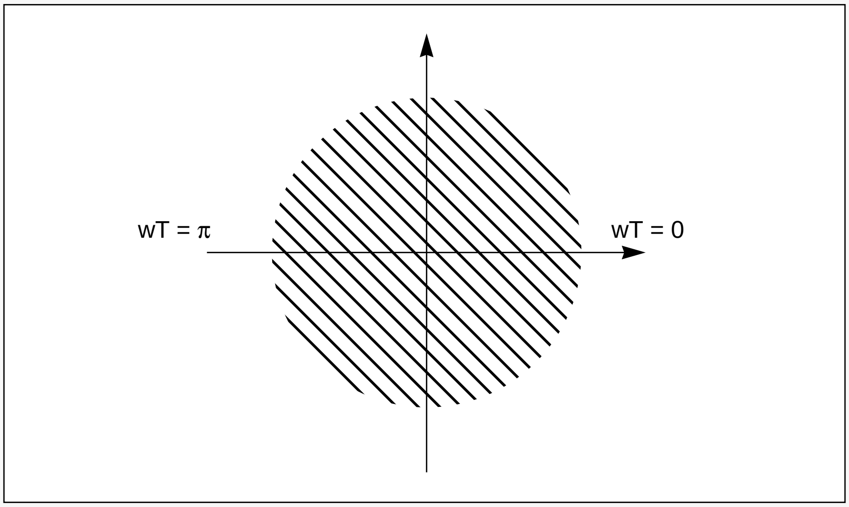


***Figure 4-2** Z-Domain*

Thus, stable digital filters map the LHP of the s-domain into the unit circle of the z-domain. An algebraic mapping for this procedure is shown in Eqn. 4-1:

$$z = \frac{\left(\frac{2}{T}\right) + s}{\left(\frac{2}{T}\right) - s} \qquad \text{Eqn. 4-1}$$

or, equivalently, the reverse mapping is shown in Eqn. 4-2:

$$s = \frac{2(1 - z^{-1})}{T(1 + z^{-1})} \qquad \text{Eqn. 4-2}$$

In the above equations, *T* represents the sampling period of the digital system; for example, *T* might equal 125 µs. To evaluate the frequency response of an analog filter, set *s = jω*; to evaluate the same function for a digital filter, set *z = e jωT*. Thus, if Ω represents frequency in the analog domain, a relationship between the digital frequency variable ω and the analog frequency variable Ω, can be obtained by substituting them in Eqn. 4-1:

$$jW = \frac{2(1 - e^{-j\omega T})}{T(1 + e^{-j\omega T})} = \frac{2\left(e^{\frac{j\omega T}{2}} - e^{-\frac{j\omega T}{2}}\right)}{T\left(e^{\frac{j\omega T}{2}} + e^{-\frac{j\omega T}{2}}\right)} \qquad \text{Eqn. 4-3}$$

Eqn. 4-3 gives:

$$\Omega = \frac{2\tan\left(\frac{\omega T}{2}\right)}{T} \qquad \text{Eqn. 4-4}$$

This relationship is used to distort the analog filter before applying the mapping of Eqn. 4-1 to preserve the frequency response of the analog filter; a procedure often known as 'prewarping'.

The completed design procedure is as follows:

1. Specify the filter in terms of sampling frequency, 3-dB point, passband/stopband ripple, etc.

2. Prewarp the filter by applying the relationship in Eqn. 4-4 to distort the original filter's frequency response.

3. Obtain an s-domain equation describing the warped filter. One of the standard filter approximations, such as the Butterworth LPF, may be used. The LPF is demonstrated in the following example.

4. Use the bilinear transform to map this analog equation into the digital domain.

To summarize, if an analog filter can be specified to perform the integration function, this function can be mapped into the digital domain for implementation by the DSP56001/2. The demonstration coefficients are calculated in the following example.

4.1 Bilinear Transform Example

To design an LPF with a 3-dB cutoff at 5 Hz and a sampling frequency of 8 kHz, normalize this filter to the sampling frequency by dividing all frequencies by the sampling frequency. The new frequencies are as follows:

Sampling Frequency (normalized) = 1 Hz =2π rad/s

Cutoff Frequency (normalized) = $^5/_{8000}$ Hz = 6.25E^{-4}Hz = 3.927E^{-3} rad/s

The first step is to prewarp this frequency as follows:

$$\Omega_{3dB} = 2\tan\left(\frac{3.927E^{-3}}{2}\right)\text{Hz} = 3.927E^{-3}\text{rad/s}$$

Eqn. 4-5

NOTE: *Prewarping does not always result in a large change in value. In this case, because of the near linearity of the tangent function for small values, there is no effective change; for most filters, however, the normalized cutoff frequency will be much larger, and prewarping will cause a significant change.*

The Butterworth approximation for a second-order lowpass filter cutoff frequency, Ω_0 rad/s, is given by Eqn. 4-6:

$$H(s) = \frac{\Omega_0^2}{s^2 + \sqrt{2}\Omega_0 s + \Omega_0^2}$$

Eqn. 4-6

With Ω_0 set to the calculated 3-dB point, an s-domain equation defines the filter. The design process is completed by translating the s-domain to the z-domain using Eqn. 4-1.

After some calculation, this translation yields the following transfer function in the z-domain:

$$H(z) = 3.866E^{-6} \frac{1 + 2Z^{-1} + Z^{-2}}{1 - 1.9945Z^{-1} + 0.9945Z^{-2}} \qquad \text{Eqn. 4-7}$$

4.2 Coefficients for Design Example

In tabular form for the DSP56001/2, the coefficients are as follows:

a1	-1.9945
a2	0.9945
b1	2
b2	1

■

SECTION 5

DSP56001/2 Implementation of the Conference Bridge

```
OPT           cex, mex, md, mu, cc
section       bridge
include       'stddefs'
xdef          bridge
xdef          data 1
xdef          data 2
xdef          silence


;****************************************************************************
; Y-MEMORY VARIABLE DECLARATIONS
;****************************************************************************

    org y:
; tap storage area
; intermediate storage for integrators

data 1   ds      16       :tap storage, bridge 1
data 2   ds      16       :tap storage, bridge 2

;*****************************************************************************
; X-MEMORY VARIABLE DECLARATIONS
;*****************************************************************************

    org x:

; integration filter coefficients

; note these are in order (a1, a2, b1, b2) and are right-shifted coefficients
; ie, a1/2, a2/2, b1/2, b2/2


coeff    dc      (1.9928921/2.0)                      :a1/2
         dc      (-.99291730/2.0)                     :a2/2
         dc      0.9999999999                         :b1/2
         dc      0.5                                  :b2/2

; variables for conversion routines.
; These will be set on powerup for A-law or mu-law
```

Figure 5-1 *Implementation of the Conference Bridge (sheet 1 of 4)*

```
silence   dc   $d50000       :$d5 for A-law, $77 for mu-law

scale1    dc   $0000a0
scale2    dc   $0000a0
scale3    dc   $0000a0
scale4    dc   $0000a0
scale5    dc   $0000a0
scale6    dc   $0000a0
scale7    dc   $0000a0
scale8    dc   $0000a0

;*************************************************************************
; Conference Bridging Macro
;
; Last Update 21/6/88 Version 1.0
;
; bridge indent 1,0
;
; This macro performs a conference bridging function for telecomms switch
; applications. The bridge uses a single source algorithm for line
; arbitration,using the 'loudest speaker' algorithm.
;
; Data input is in the form of a table of logarithmic (A-law or mu-law format)
; data the algorithm, used converts this to a linear absolute value for
; processing.
;
; This (linear absolute) signal is then integrated to a given an estimation of
; the signal's power content. The coefficients of this integrator are variable
; for specific system needs, and are available to a host processor.
;
; After integration, compares are used to find the highest power signal on the
; bridge. This is then passed out to all outputs, except the source output. This
; receives silence.
;
; The bridge software can implement bridges of one or more channels, depending
; only on one variable. However, a bridge of less than three channels is
; meaningless in most telecomms applications.
;
;
;
; Macro Calls: Bridge - implement conference bridge with following parameters:
;         1) Data 1/0 table at location in r5
;         2) Integrator memory at location in rl
;         3) Number of channels in bridge at location x:(r2)
;         4) Base of appropriate conversion table in x:'ctable'
;            This should be $100 for mu-law, $180 for A-law
;         5) Value of SILENCE tone at x:silence
;
;
; From these parameters, the software can allocate memory and use the correct
; format conversion law.
;
;
```

***Figure 5-1** Implementation of the Conference Bridge (sheet 2 of 4)*

```
; Note that the macro assumes that the scaling modes are off (S1=0, S2=0)
;
; Input/Output data is in the 8 most significant bits of the I/0 table: the
; remaining bits of data, if present, are ignored.
;
;
;         Sign      Chord  Number       Step Number
;         Bit
;         23        22  21  20         19  18  17  16
;
; Alters Data ALU Registers
;   y1    y0
;   x1    x0
;   a2    a1   a0   a
;   b2    b1   b0   b
;
;
; Alters Address Registers
;   r0    r1   r2   r3
;   r4    r5   r6   r7
;   n
;   m
;
;
; ; Alters Program Control Registers
;   pc    sr
;
; bridge selection
; Selection of a bridge is performed by initializing r5 to access the I/0 table
; for a given bridge, and r1 with the address for that bridge’s filter data.
; Note that this is not the coefficients for the filter, but the W(n-1) and
; W(n-2)terms.
;
;***************************************************************************
; This selection is Conference Bridge
;***************************************************************************
;
          org       p:
bridge

; bridge setup
; Set up some variables for the bridges. These are generic to any bridge
; implementation. The variables are the address of the scale table, and the
; address of the coefficients for the  integrator. (These coefficients may be
; downloaded by the host.)

    cir   b         #<scale1,r6        ; set up address of scale table
    move            #<oeff,r4          ; and address of integrator coefficients
    move            #3,m4              ; and modulo for integrator coefficients
    move            r5,n5              ; save r5 for future use
    move            r5,57              ; initialize r7 for no signal operation

; stage one of the bridge is conversion of the current channel data to absolute
; linear value. This is performed using the DSP56001's A/mu-law lookup tables in
; ROM; x:'ctable' is the address of the conversion table to be used. This is $180
; for A-law, $100 for mu-law.
```

Figure 5-1 *Implementation of the Conference Bridge (sheet 3 of 4)*

```
    do        #6,end_env
    move      x:(r5)+,a
    lsl       a          x:<cshift,y0   ;shift out sign bit, get_shift constant
    lsr       a          y:<tab_val,yl  ;shift in zero, get table base
    tfr       yl,a       al,yl          ;swap table base and offset
    mac       yl,y0,a    #<silence,r3   ;shift offset down and add to base
    move      a,r0                      ;move to address register


; r0 is a pointer to absolute value of the incoming signal, in linear format.
; Next stage is integration using recursive filter. Note the use of a scale factor
; to scale the input for calculations; also note that we are using the ALU's
; scalers to allow use of coefficients in the range +2 . . . -2.


envelope or #$8,mr                               ;set ALU scaling mode on
    move   x:(r0),y0
    move   x:(r6)+,xl                            ;read scaling coefficient for input
    mpy    xl,y0,a   y:(rl)+,y0  x:(r4)+,x0      ;scale input data, read W(n-1)&a1
mac y0,x0,ay:(rl),ylx:(r4)+,x0;read WW(n-2), a2
    mac    yl,x0,a   y0,y:(rl)-  x:(r4)+,x0      ;rite new W(n-2), get b1
    mac    y0,x0,a   a,y:(rl)+   x:(r4)+,x0      ;rite new W(n-1), get b2
    macr   yl,x0,a   (rl)+                       ;round output data, and
                                                 ;increment data pointer to next
                                                 ;integrator data
           and       #$f7,mr                     ;cancel the ALU scaling mode

; Finally, check to see how this slot's signal compares with the others. We also
; fetch the silence code into yl here: this is used later.

     cmp     b,a       x:(r3),y1        ;is this the biggest input channel?
     tgt     a,b       r5,r7            ;if yes. Transfer the data as a
                                        ;new maximum, and set up pointer

; This is the end of the bridge loop. On exit, the DSP will have calculated
; the channel with the largest magnitude, and stored the magnitude of that
; channel in B, and a pointer to that channel’s data in r7.

end_env                                 ;This is the end of the bridge loop



; Having found the largest channel, we must write that value to all output
; channels, except the source speaker, who should receive silence. This is
; performed by writing the output data to all channels, then overwriting this
; overwriting with silence for the speaker channel


output   move   n5,r5              ;read address of data table
         move   x:-(r7),a          ;read output and silence tone
         rep           #6          ;repeat for all channels
         move   a,x:(R5)+          ;write out biggest channel data to channels
         move   y1,x:(r7)          ;but overwrite source with silence
         rts
```

Figure 5-1 *Implementation of the Conference Bridge (sheet 4 of 4)*

```
;*******************************************************************
; Interrupt Service routine
; Called at 8kHz by the Frame Sync Interrupt
; Saves registers, then processes one frame
;********************************************************************

frame    REG_DUMP                    ;save all registers
         move          #lol,r5      ;set up pointer to start of lo data table
                                     ;for this bridge
         move          #,datal,rl   ;second pointer to stored tap data for this
                                     ;bridge
crbl     jsr           <bridge      ;call bridge macro for channel 1 (short
         move          #102,r5      ;jump forced)
         move          #<data2,rl   ;second pointer to stored tap data for this
                                     ;bridge
crb2     jsr           <bridge      ;call bridge macro for channel 2
                                     ;(short jump forced)
         REG_RES                     ;restore registers
         rti                         ;return to main handler
```

Figure 5-2 *Example code that shows the calling convention for conference bridges. This code should be run as an interrupt routine, or modified to fit into a polled environment.*