APR403

Conference Bridging in the Digital Telecomms Environment, using the Motorola DSP56000

Prepared by
Ralph Weir, DSP Applications Engineer, Motorola Ltd., East Kilbride

SECTION 1
INTRODUCTION

This document describes one possible implementation of a conference bridge for a modern digital telephone exchange. Conference bridging is a feature which is available on many modern analogue or digital telephone exchanges. It allows calls of three or more subscribers to be set up, and provides an arbitration scheme between subscribers involved in the conference.

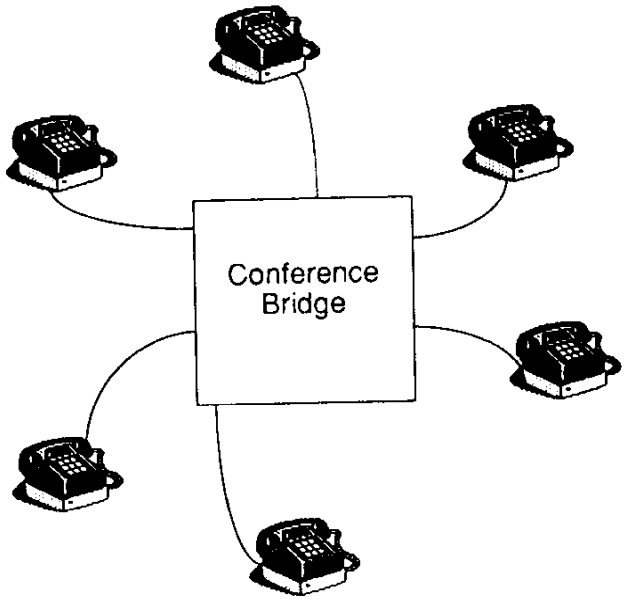

Figure 1-1. A six-way conference bridge

In Figure 1-1, the conference bridge is shown as a separate entity; it is usually implemented as part of a larger system, such as a PABX. The concept is to allow a conversation to take place between several subscribers in a controlled manner.

There are a number of possible schemes for implementing this. In the analogue world, it is common to simply sum all the incoming signals, giving one output; however, this degrades the signal quality, as the SNR of the signal is significantly reduced.

A digital system can emulate this approach. However, it is more common for a small conference bridge to use a 'single speaker' algorithm, where only one input signal is passed to the listeners. In this case, the speaker is not a listener, and is not passed any signal back; signal paths within his handset are relied on to provide a 'comfort level' of signal feedback.

This approach has many advantages; the most significant is that the speaker's signal is passed through unaltered, and so no change in SNR will be experienced. It is commonly used for small bridges, of up to perhaps 10 channels. Larger bridges may use a multiple speaker algorithm.

In this case, some algorithm must be used to decide which subscriber should be made the speaker. There are a number of choices; most algorithms use the 'loudest speaker' algorithm, which takes the loudest speaking subscriber as the current speaker. This is straightforward to implement, and is the scheme implemented here.

To determine the loudest speaker, the software described here uses an integrator to find the power level of the incoming signal. This is then used as the 'loudness', rather than the instantaneous level of the signal; this slows the bridge's switching rate, and gives a more aurally satisfying result.

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

1.1 OVERVIEW OF THE SYSTEM

This Application Note describes a software implementation of a conference bridge, written for the DSP56000/1 signal processor. This device has a 24-bit architecture, a feature which greatly simplifies the software required, and is available in different speed ratings.

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

One variable of global importance is n_chans. This affects all sections of the bridge, as 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. It is relatively simple to modify the software for larger bridges, or for dynamically configurable bridges according to the requirements of the system controller. The only requirement for additional bridges is additional memory for data storage and I/O – this may be internal or external.

To give some idea of the performance available from this software, a 20MHz DSP56000/1 is capable of implementing ten six-way conference bridges, or a single 71-way bridge, or even 17 three-way bridges. Obviously, it is possible to mix and match, so the system may include different sized bridges, or be reconfigured dynamically according to system requirements.

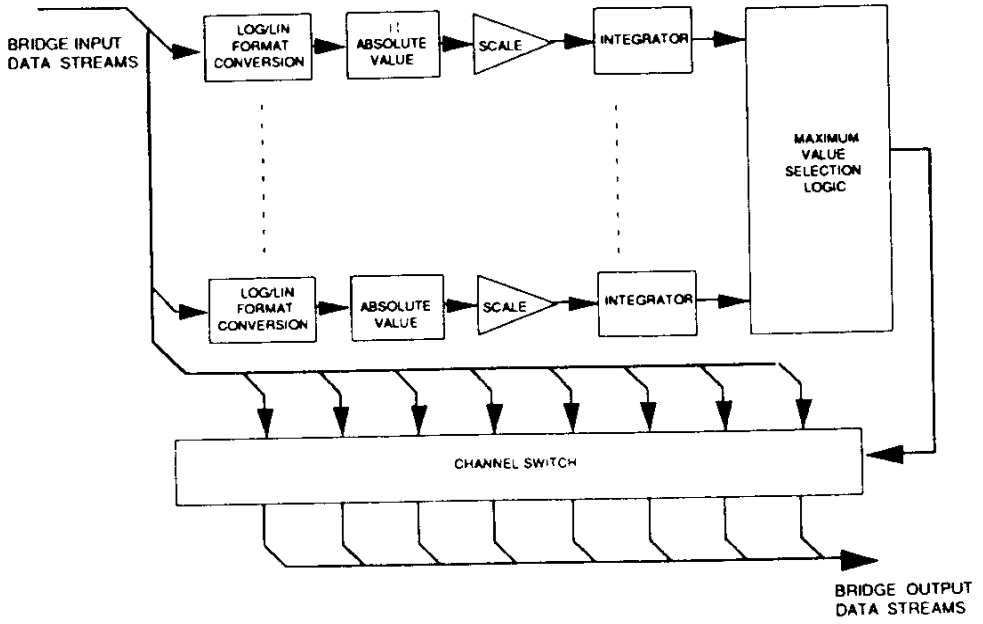

Figure 1-2. Block diagram of the system

1.2 INPUT/OUTPUT CONSIDERATIONS

Input is taken from tables of data, representing the I/O data streams to the subscribers, held in logarithmic format. In the 56000 implementation, these tables may be updated from the SSI interface; this is 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. This has the advantage of adding additional serial ports to the DSP itself; also the MT8920 is table driven, resulting in a reduction in the number of interrupts the system must handle. This may be advantageous where the DSP is performing other tasks, like 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 re-transmission.

One data table is required for each bridge, comprising one word per bridge channel.

SECTION 2
CHANNEL PROCESSORS

2.1 LOG/LIN CONVERSION

Data within a digital telecomms network is held in an 8-bit logarithmic format, to reduce the number of bits required to be transmitted while still giving an acceptable signal to noise ratio. Two standards are used; Europe uses A-law coding, while the US uses μ-law coding. The formats are similar, but detail differences exist, and must be taken account of when converting to linear data.

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

2.2 ABSOLUTE VALUE CALCULATION

The calculation of absolute value is performed within the log/lin conversion code. Both A-law and μ-law are 'sign+magnitude' formats, where the most significant bit represents sign, and the remaining seven form the magnitude By ignoring the sign bit during format conversion, a positive value is always generated. This allows 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 will be 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 scale1... scale8, where scaleN corresponds to the scale factor applied to channel N.

2.4 INTEGRATION

Integration is performed using a two-pole recursive (IIR) filter. This technique allows the system designer to easily tailor the envelope detector function to his system's requirements, using one of the many standard IIR design techniques. This is dealt with in more detail in a later section.

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; this simply means any memory block with base address divisible by 4.

The integrator requires two words of storage per bridge channel. This is used 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 have chosen 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 simply 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 us to use the original data unaltered. The input signal is passed through unaltered; thus, for the current speaker, absolutely no degradation in SNR will be experienced.

2.7 SYSTEM PERFORMANCE

The computational load imposed by the bridge is made up of two parts; one section is an absolute overhead for initialising the bridge, the second part is a per channel overhead, made up of the computation involved in each channel processor. This has been expressed in terms of Icyc, or instruction cycles; for reference, with the 8 KHz sampling rate in use in telecomms, 1280 Icyc are available from the standard 20.5 MHz processor, while 1687 Icyc are available with the 27 MHz processor. This is equivalent to saying that an 8 KHz sample rate corresponds to a 1250 µs sample period, and that the two devices have 97.5 and 74.1ns instruction cycle times respectively.

The total load is given by:-

$$Bcyc = 22 + 17*n\_chans$$

where Bcyc 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

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 µ-law, $180 for Variable

Variable	X/Y	Description
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 will depend 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 bridge implemented, giving the number of channels in each bridge.
IO	Y	This is the I/O area for a bridge; it requires one word per bridge channel.

2.9 CONTROL OF THE BRIDGE BY THE SYSTEM CONTROLLER

Using the 56000 to implement conference bridging brings an additional advantage; that the system controller may set up the bridge to the exchange's specific requirements, whether these be decided at power-on, or be instantaneous requirements.

The only requirement for this is that the system should support some form of access to the 56000's data RAM; this can be accomplished through the host port, or even using some of the unused slots of the SSI interface.

SECTION 3
THE INTEGRATOR – THEORY AND DESIGN

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 here; to understand why, we must examine the nature of a speech signal. A speech signal is made up 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 made up of a number of these elements.

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

Some idea of the resulting signal can be obtained from Figures 3-1 to 3-3. Figure 3-1 shows a pure tone. Figure 3-2 shows a low-frequency sine wave representing the modulating signal. Figure 3-3, shows the modulated signal; while this is very simplistic, it shows the complex nature of the speech signal.

It would plainly be useless to use instantaneous power as a basis for the 'loudest speaker' decision algorithm. Due to the various envelope functions the speaker applies to his speech, the instantaneous power is a rapidly-changing function; this would result in the bridge switching rapidly between speakers. Some means of damping the bridge's response is required.

This is the function of the integrator. It attempts to filter out the envelope functions, leaving only a slowly-changing function which can successfully be used in the decision logic.

Unfortunately, a tradeoff exists in the choice of integration coefficients. 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 tradeoff, it was decided to use an adapted IIR filter in preference to a true integrator. This allows the system designer to modify the integration coefficients quite easily, using a filter design methodology like the Bilinear Transform, described later in this section.

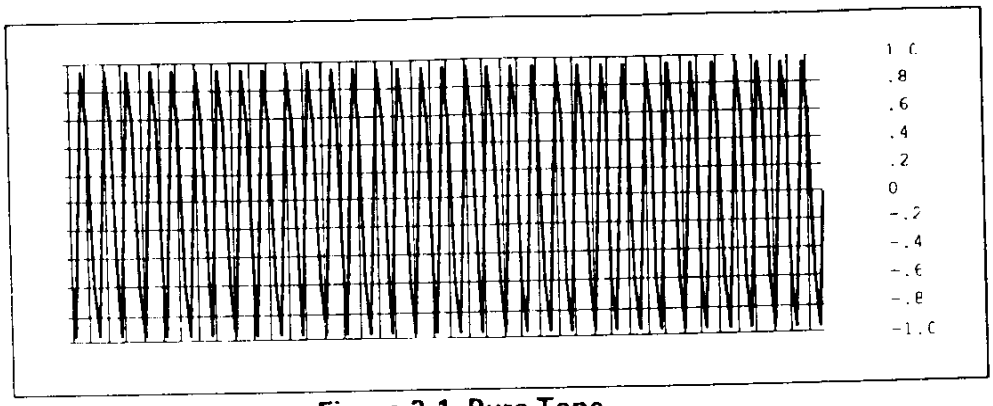

Figure 3-1. Pure Tone

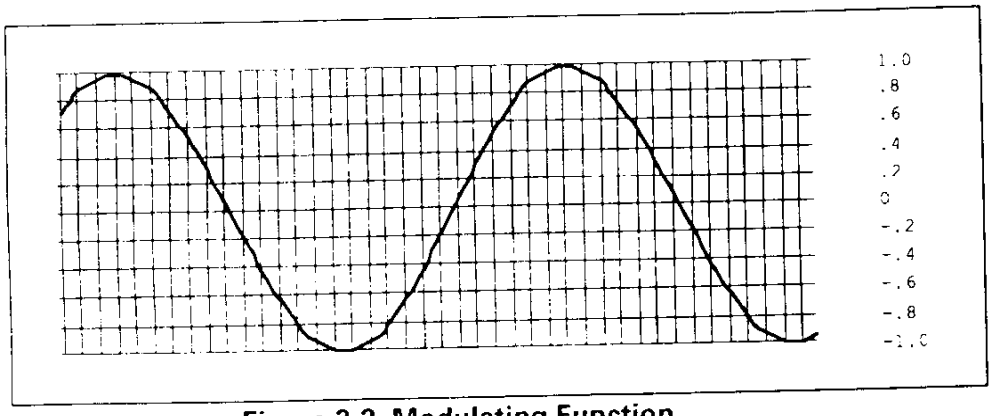

Figure 3-2. Modulating Function

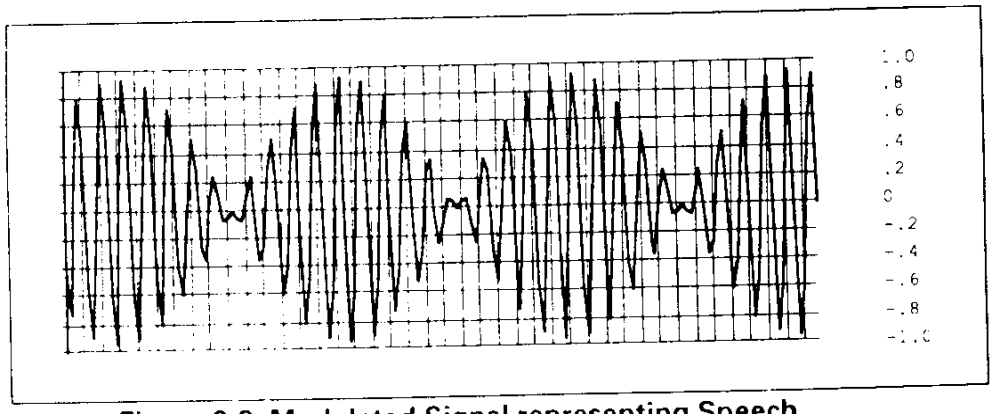

Figure 3-3. Modulated Signal representing Speech

3.1 THEORY BEHIND THE INTEGRATING FILTER

Integration of the incoming waveform is performed using a two-pole IIR filter; the transfer function H(z) for this filter is given below:-

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

From this transfer function, a signal flow graph can be synthesised, and is given in Figure 3-4.

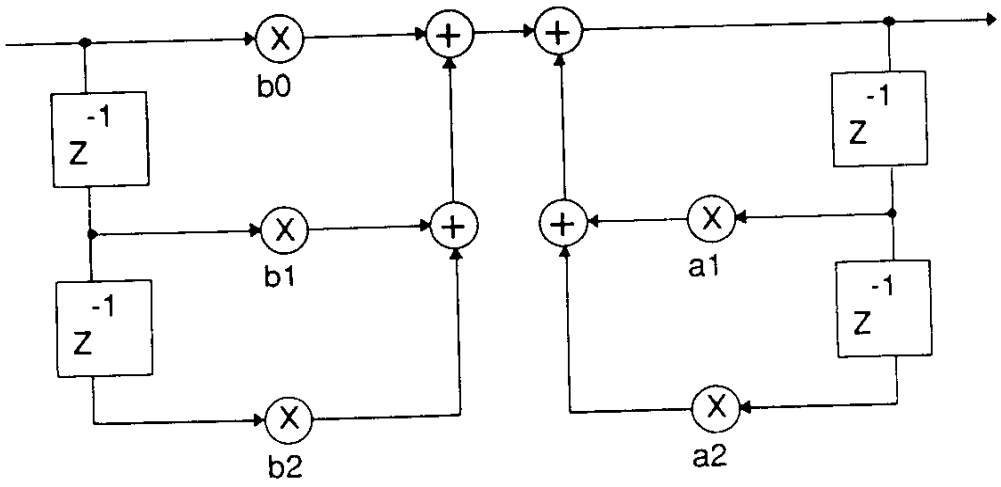

Figure 3-4. Direct Form I Filter

The filter actually implemented has been simplified to require only four MAC operations and two storage stages; the simplification procedure will now be considered.

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 3-5. This is an intermediate step of the simplification process; as can be seen, no overall simplification has been performed, but we now have the delay stages for the a and b stages adjacent.

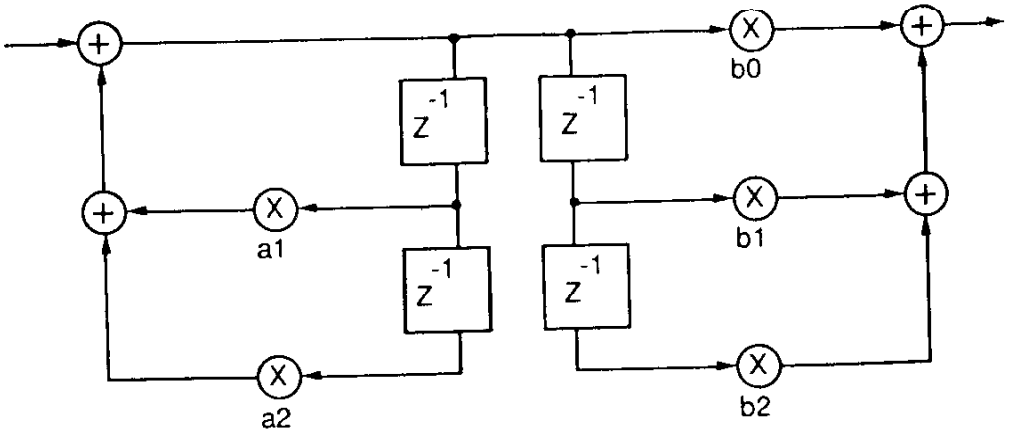

Figure 3-5. Modified Direct Form I Filter

Grouping the delay stages allows us to combine the delay elements; this was the object of grouping, as no other simplification was gained by the group. The transfer function is unchanged by this operation.

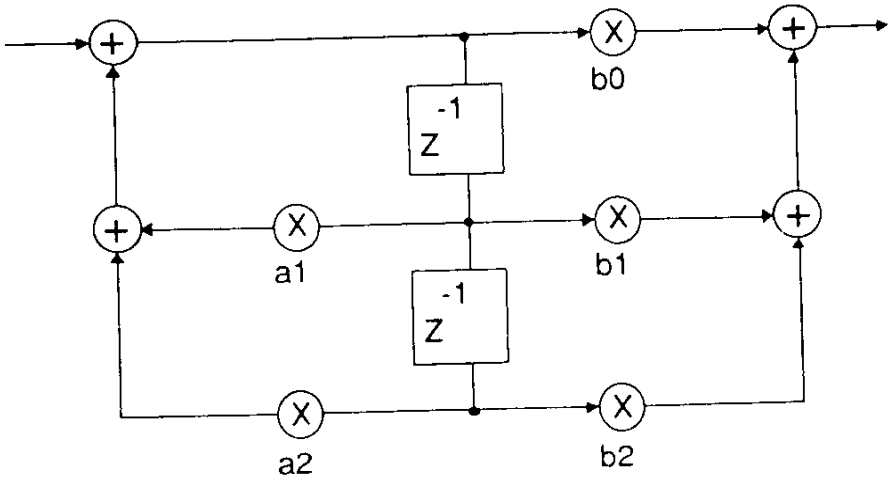

Figure 3-6. Direct Form II Filter

At this stage, it is worth noting that our 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 we factor the numerator of the transfer function (3-1) by b0, we have the transfer function (3-2).

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

Other than in terms of gain, this is equivalent to the original filter, and may be implemented by the signal flow graph of Figure 3-7 it is apparent that the filter now only requires four multiplies, but implements a gain stage of 1/b0. As we are interested only in relative levels, this gain stage can be ignored; this gives us the optimal form of the integrator.

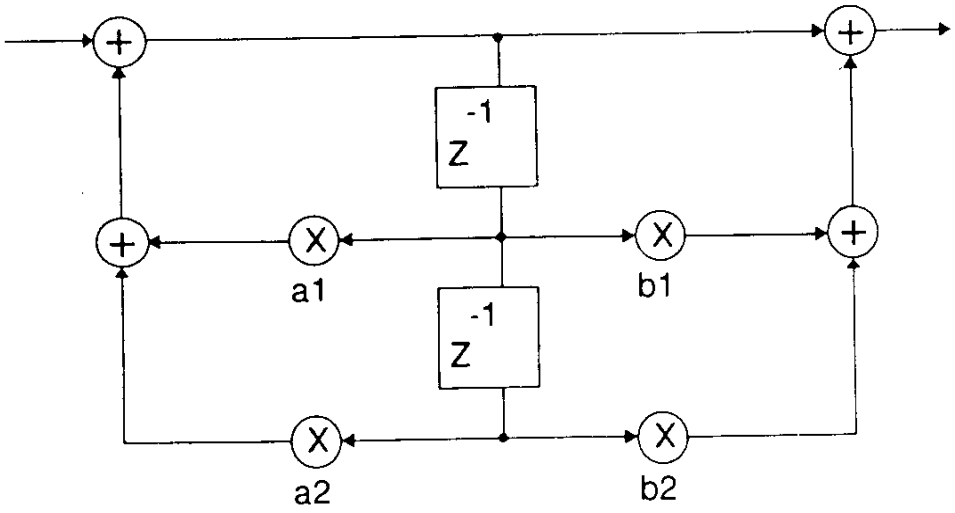

Figure 3-7. Scaled Direct Form II Filter

SECTION 4
CALCULATING THE INTEGRATOR COEFFICIENTS

The signal flow graph of Figure 3-7 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 converts a standard analogue filter to the digital domain; we have only considered Butterworth filters, but 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 analogue 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, the procedure will now be explained.

The s-domain may be mapped as 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, or LHP

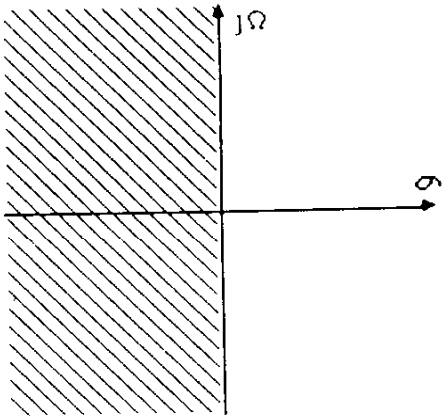

Figure 4-1. The S-domain

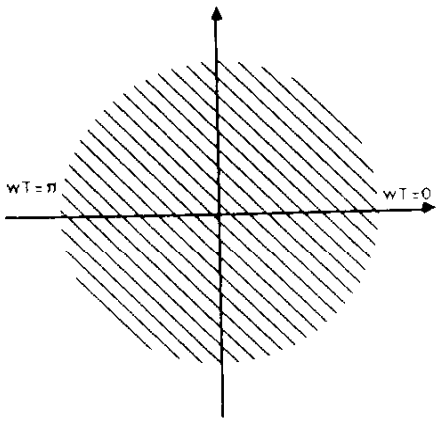

Figure 4-2. The Z-domain

The z-domain is represented in Figure 4-2. As can be seen, the z-plane is not divided into an LHP and RHP, but instead as a unit circle. Filters must have poles and zeros within the unit circle; this is again shaded.

Thus, for stable digital filters, we must map the LHP of the s-domain into the unit circle of the z-domain. An algebraic mapping for this exists:-

$$z = \frac{\left(\frac{2}{T}\right) + s}{\left(\frac{2}{T}\right) - s} \tag{4-3}$$

or, equivalently, the reverse mapping:-

$$s = \frac{2\left(1 - z^{-1}\right)}{T\left(1 + z^{-1}\right)} \tag{4-4}$$

In these relationships, T represents the sampling period of the digital system; for our purposes, it is 125μs.

If we required to evaluate the frequency response of an analogue filter, we would set $s = j\omega$; to evaluate the same function for a digital filter, we would set $z = e\ j\omega T$. Thus, if we use Ω to represent frequency in the analogue domain, we can obtain a relationship between the digital frequency variable ω and the analogue frequency variable Ω by substituting in (4-3):-

$$jW = \frac{2\left(1 - e^{-j\omega T}\right)}{T\left(1 + e^{-j\omega T}\right)} = \frac{2\left(e^{\frac{j\omega T}{2}} - e^{-\frac{j\omega T}{2}}\right)}{T\left(e^{\frac{j\omega T}{2}} + e^{-\frac{j\omega T}{2}}\right)} \tag{4-5}$$

From the relationship in (4-5):-

$$\Omega = \frac{2 \tan\left(\frac{\omega T}{2}\right)}{T} \tag{4-6}$$

This relationship is used to distort the analogue filter before applying the mapping of (4-3) to preserve the frequency response of the analogue filter, a procedure often known as 'pre-warping'

The completed design procedure is as follows:-

1) Specification of the Filter in terms of sampling frequency, 3 dB point, passband/stopband ripple etc.

2) Prewarp the filter. This is performed by applying the relationship of (4-6) 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. This is demonstrated in the following example.

4) Use the Bilinear Transform to map this analogue equation into the digital domain.

To summarise, if we can specify an analogue filter to perform the integration function, we can map this into the digital domain for implementation by the 56000. As an example, the demonstration coefficients are calculated below:-

4.1 EXAMPLE OF THE BILINEAR TRANSFORM

Supposing we wish to design an LPF with 3 dB cutoff at 5 Hz, sampling frequency 8 KHz. We can normalise this filter to the sampling frequency; this has no effect on the coefficients, but keeps the maths simple. This is accomplished by dividing all frequencies by the sampling frequency. The new frequencies are as follows:-

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

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

Referring back to the design procedure, our first step is to prewarp this frequency:-

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

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

The Butterworth Approximation for a 2nd order Low Pass Filter, cut off frequency Ω_0 rad/s, is given by:-

$$H(s) = \frac{\Omega_0^2}{s^2 + \sqrt{2}\Omega_0 s + \Omega_0^2} \qquad (4\text{-}7)$$

With Ω_0 set to our calculated 3 dB point, we now have an s-domain equation defining our filter. The design process is completed by translating this to the z-domain using (4-3).

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

$$H(z) = 3.866E^{-6}\frac{1 + 2Z^{-1} + Z^{-2}}{1 - 1.9945Z^{-1} + 0.9945Z^{-2}} \qquad (4\text{-}8)$$

4.2 COEFFICIENTS FOR DESIGN EXAMPLE

In tabular form for the 56000, the coefficients are:-

a1	-1.9945
a2	0.9945
b1	2
b2	1

SECTION 5
DSP56000 IMPLEMENTATION OF THE CONFERENCE BRIDGE

```
        OPT         cex,mex,md,mu,cc
        section     bridge
        include     'stddefs'
        xdef        bridge
        xdef        data1
        xdef        data2
        xdef        silence

;*****************************************************************
; Y-MEMORY VARIABLE DECLARATIONS
;*****************************************************************
        org y:

; tap storage area
; intermediate storage for integrators

data1   ds      16      ; tap storage, bridge 1
data2   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

silence dc      $d50000     $d5 for A-law, $?? for mu-law

; scale factors for channels
; set up on call initialisation or power up to reflect who's chairman

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  ident  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 give 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 Call's   bridge - implement conference bridge with following parameters:

;           1) Data I/O table at location in r5
;           2) Integrator memory at location in r1
;           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.

; 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/O 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 initialising r5 to access the I/O table for
; a given bridge, and r1 with the address of 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 section 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.)

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

; find biggest channel
; stage one of the bridge is conversion of the current channel data to absolute linear value.
; This is performed using the 56001'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.

    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,y1           ; shift in zero, get table base
    tfr     y1,a        a1,y1                   ; swap table base and offset
    mac     y1,y0,a     #<silence,r3            ; shift offset down and add to base
    move    a,r0                                ; move to address register

; r0 is a pointer to the 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         #$8,mr                                          ; set ALU scaling mode on
         move       x:(r0),y0
         move       x:(r6)+,x1                                      ; read scaling coefficient for input
         mpy        x1,y0,a     y:(r1)+,y0  x:(r4)+,x0              ; scale input data, read W(n-1) & a1
         mac        y0,x0,a     y:(r1),y1   x:(r4)+,x0              ; read W(n-2), a2
         mac        y1,x0,a     y0,y:(r1)-  x:(r4)+,x0              ; write new W(n-2), get b1
         mac        y0,x0,a     a,y:(r1)+   x:(r4)+,x0              ; write new W(n-1), get b2
         macr       y1,x0,a     (r1)+                               ; round output data, and
                                                                    ; increment data pointer to next
                                                                    ; integrator data
         and        #$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 y1 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
; with silence for the speaker channel
```

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

5.1 EXAMPLE CALLING FUNCTION

The following code is intended as an example to show the calling convention for the conference bridges. It should be run as an interrupt routine, or modified to fit into a polled environment.

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

frame   REG_DUMP                ; save all registers
        move    #io1,r5         ; set up pointer to start of io data table
                                ; for this bridge
        move    #<data1,r1      ; second pointer to stored tap data
                                ; for this bridge
crb1    jsr     <bridge         ; call bridge macro for channel 1
                                ; (short jump forced)
        move    #io2,r5         ; set up pointer to start of io data table
                                ; for this bridge
        move    #<data2,r1      ; 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
```