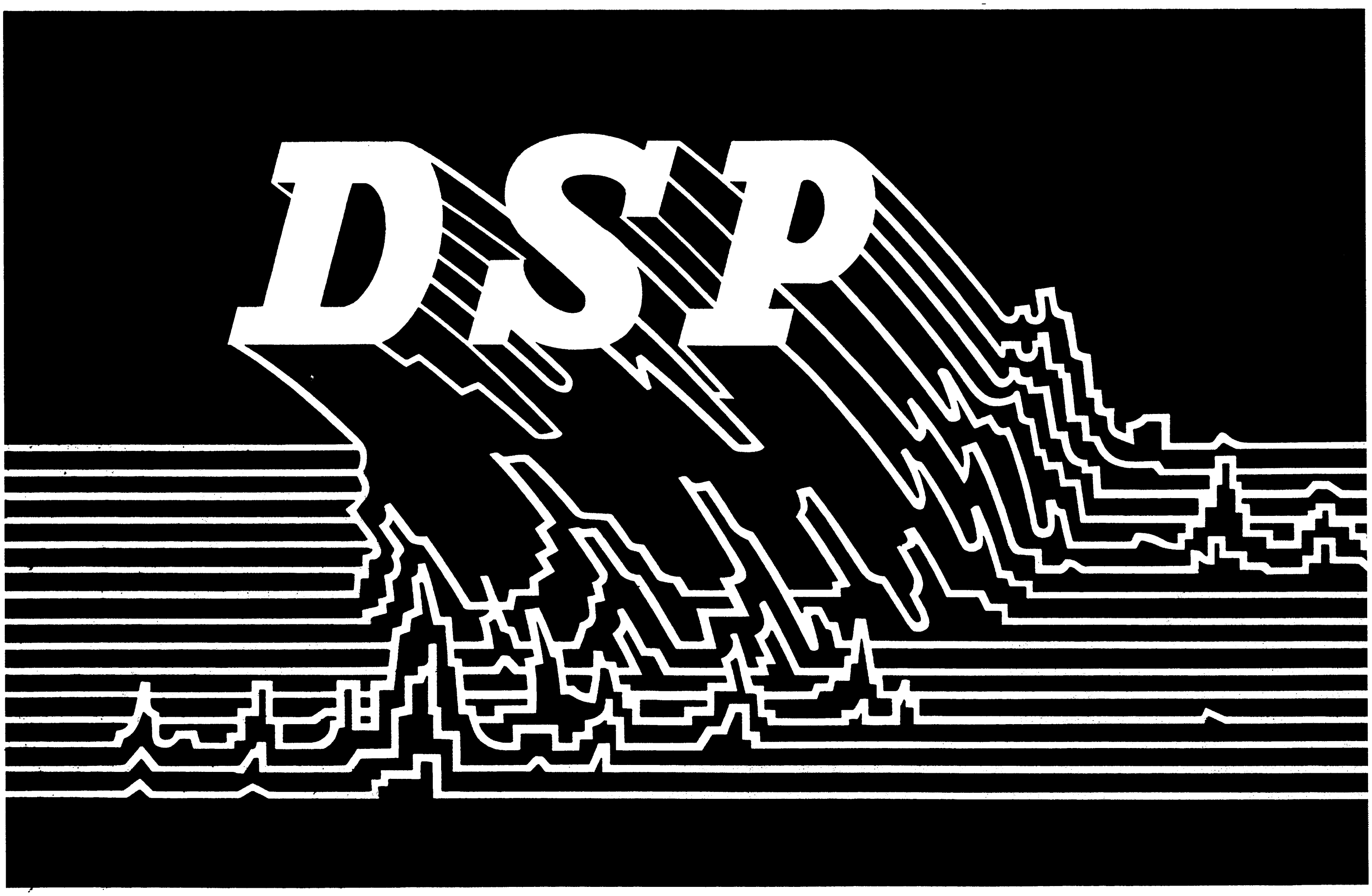
APR401/D

Twin CODEC Expansion Board for the DSP5600 Application Development System



MOTOROLA

Twin CODEC Expansion Board for the DSP5600 Application Development System

Prepared by: Ralph Weir, DSP Applications Engineer, Motorola Ltd., East Kilbride, Scotland
Eric Cheval, DSP Applications Engineer, Motorola Inc., Oakhill, Texas

INTRODUCTION

The codec is an integrated ADC, DAC and filter intended for use in telecommunications applications. As such, it has been designed for a sample rate of 8KHz, the standard telecomms sampling frequency, and has a serial interface which may be used in a TDM system.

The filter implemented in the codec tailors the incoming analogue signal for transmission through a telephone channel. It has a band-pass characteristic, cutting off at 300Hz and 3.4KHz; this also performs the anti-aliasing function for the ADC.

This document describes a twin codec board designed to facilitate the development of telecomms applications around the DSP56000 family processors. The board is intended for any situation where a DSP module is required to link two analogue lines, as in the case of a line repeater, or indeed a telephone handset; there is no reason why it should not be used to develop applications requiring a single codec.

The codec board is designed to interface to the DSP directly, using the SSI interface; this is capable of generating all signals required for a serial codec, creating an interface with no glue logic whatsoever. It is possible to create systems with many codecs, all under the control of one DSP processor over the SSI communications link. The board described here is a simple example of such a system.

Various software routines are available, giving the DSP the ability to convert between the various data formats available from codecs (linear, A-law and μ-law are the three main formats; it is essential that data be in a linear format before DSP processing). The conversion rotuines are listed in Appendix E, but for more details, consult application note ANE008, or the Dr Bub bulletin board. PCB artwork for the expansion board is included in Appendix F at the end of this document.

DSP56000/1 SERIAL SYNCHRONOUS INTERFACE (SSI)

The DSP56000 SSI is a powerful serial interface which may be used with many existing serial codecs, cofidecs (monocircuits) and serial ADC's and DAC's. It is also a suitable medium for building serial networks of DSP's based on a time division multiplexed (TDM) access protocol. A complete description of the interface will be found in DSP56000 UM/AD user's manual, section 7; what follows is a short description of the interface.

The SSI is a 6 pin interface which may be configured for synchronous or asynchronous exchange, continuous or gated clock, and normal or network mode. Clocks may be generated either internally and output, or input from the external host system. The 6 pins are not necessarily used in all configurations, and unused lines may remain as general purpose I/O.

Like every DSP56000/1 on-chip peripheral, this is a full duplex, double buffered, memory mapped peripheral, mapped into the peripheral area at the top of internal X-memory. This interface inputs and outputs the data using two 24bit data registers mapped at X:$FFEF :

- TX write-only Transmit Data Register
The transmit shift register is associated with this register. This register shifts out the data written to the TX register onto the STD pin; when empty, it reads the data in the Transmit Data Register if any is available (a transmitter underrun error will occur if no fresh data is present). The DSP may be programmed for an interrupt when the transfer takes place, or may poll the SSI status flags.

- RX read-only Receive Data Register
The receive shift register is associated with this register. This register formats the serial data read from the SRD pin; when full, it transfers the data to the Receive Data Register, and sets a flag to indicate data is available (if the data in the RX register is unread by the time the shift register is again full, a receiver overrun error will occur). Unused bits are written as zeros; the DSP may be programmed for an interrupt when the transfer takes place, or may poll the SSI status flags.

There are four control registers associated with the SSI. These are :

- CRA (X:$FFEC)
16bit read/write control register governing word length, frame rate and clock speed.

- CRB (X:$FFED)
16bit read/write control register governing interrupt control, peripheral enables and clock/frame sync formats.

- SSISR (X:$FFEE)
8bit read only peripheral status register containing all device status and error flags.

- TSR (X:$FFEE)
8bit write only time slot register used in network mode; behaves like an alternative Tx data register which is written during unused time slots. In this case, rather than transmitting data, the STD pin will be tri-stated during that time slot.

CRA (16 low bits of X:$FFEC; Read/Write register)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSR	WL1	WL0	DC4	DC3	DC2	DC1	DC0	PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

This register controls clock and frame sync generation, word length and number of words per frame. It is used as follows:

- **PSR** Prescaler Range. PSR=1 enables a divide by 8 prescaler in the clock generator.
- **WL1/0** Selects the number of bits per word (00 for 8 bit data).
- **DC4/0** Control the frame divider rate. In normal mode they control the word transfer rate.
- **PM7/0** Select the divide ratio for the clock generator.

According to the DSP's crystal clock frequency and the required SSI clock rate, the various standard telecomms frequencies may be generated. The table below details the values of PM7-0 for this; note that if the prescaler is used, these values should be divided by 8:

FOSC (MHz)	Maximum bit clock	128 KHz	1.536 MHz	1.544 MHz	2.048 MHz	2.56 MHz
16.384	4.096	32	2.67	2.65	2	1.6
18.432	4.608	36	3	2.99	2.25	1.8
20.48	5.12	40	3.33	3.31	2.5	2

When the frame sync has to be generated by the DSP on pins SC1/2, bits DC4-0 of CRA and the FSL bit of CRB have to be set up accordingly. For the MC145503 monocircuit, SC2 may be connected directly to TDE/RDE; these lines should be high during word exchange, so the FSL bit in CRB should be cleared.
The codec's TDE line should be cycled at 8KHz to provide the sampling rate clock. This defines DC4-0 for a single codec in normal mode, according to the selected bit clock SCK. Note that the value for a twin codec system is based on cycling TDE at 16KHz, as the line is gated between two devices; thus the DC4-0 value should be halved.

SCK	128 KHZ	1.536 MHZ	2.048 MHZ
DC4-0 One CODEC	2	$17	$1F
DC4-0 Two CODEC	1	$B	$10

CRB (16 low bits of X:$FFED; Read/Write register; cleared by RESET)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RIE	TIE	RE	TE	MOD	GCK	SYN	FSL	*	*	SKD	SD2	SD1	SD0	OF1	OF0

This register controls interrupts from the SSI, enabling of the SSI and clock/frame sync formats. It also contains the control bits for the serial control lines SC0/1/2, and is used as follows:

- **RIE** Enables the Rx interrupt; the DSP will be interrupted when data may be read from the SSI's RX register.
- **TIE** Enables the Tx interrupt; the DSP will then be interrupted when data may be written to the TX register for SSI transmission.
- **RE** Enables the SSI receiver. If this bit is not set, the SSI will never receive any data.
- **TE** Same as RE, but for the SSI transmitter.
- **MOD** Selects normal mode when clear, network mode when set.
- **GCK** Selects continuous clock when clear, gated clock when set.
- **SYN** Selects asynchronous mode when clear, synchronous mode when set.
- **FSL** Selects frame sync length - word length when clear, bit length when set.
- **SKD** Selects external clock when clear, internal clock when set.
- **SD2** Controls the direction of the SC2 line; clear for input.
- **SD1/0** Controls the direction of the SC1 and SC0 lines; clear for input.
- **OF1/0** Output data for SC1 and SC0 when configured as an output

SR (8 low bits of X:$FFEE; Read only register)

7	6	5	4	3	2	1	0
RDF	TDE	ROE	TUE	RFS	TFS	IF1	IF0

The status register provides the following status information to the DSP:

- **RDF** Receive Data Full - set when the RX register contains valid received data which may be read by the DSP.
- **TDE** Transmit Data Empty - set when the TX register is empty, and ready to receive another word for transmission. Note that this does not mean the last word has been fully transmitted, as the SSI port is a buffered interface.
- **ROE** Receiver Overrun Error - set when the DSP overwrites valid, but unread, data in the RX register. This would occur if the SSI was receiving a stream of serial data, and for some reason the DSP did not read one word. This condition may be used to provide an alternative interrupt to the DSP, or may be polled to check for the error condition.
- **TUE** Transmitter Underrun Error - set when a frame sync occurs, but the SSI has no data to transmit. This may cause an error with many serial devices; this may be used to switch to the transmit exception interrupt vector, or may be polled to detect the error.
- **RFS** Receive Frame Sync - set when a receive frame sync occurs during the reception of a word, when in network mode. This indicates the first time slot.
- **TFS** Transmit Frame Sync - same as RFS, but for transmission.
- **IF1/0** Input Flag 1/0 - this flag contains the data on the SC1/0 line, when configured as an input. It is latched from SC1/0 during reception of the MSB of each incoming word.

Some or all of the lines allocated from Port C to the SSI must be configured as dedicated on-chip peripheral pins, by setting the corresponding bits of the Port C control register:

PCC (X:$FFE1 Port C Control register - read/write register)

The Port C lines are used as follows:

-PC3	SC0	bidirectional	Serial Control line
-PC4	SC1	bidirectional	Serial Control line
-PC5	SC2	bidirectional	Frame Sync I/O
-PC6	SCK	bidirectional	Serial Clock
-PC7	SRD	input	receive data
-PC8	STD	output	transmit data

Initialising the SSI

The recommended procedure for SSI initialisation is given in the user's manual, and is as follows:

1. RESET the device. This can be a hardware reset, performed by driving the RESET* pin low, as on power-on reset, or a software reset, performed by executing the RESET instruction, which resets the on-chip peripherals.

2. Program the SSI control registers CRA and CRB by writing to their locations in X-memory

3. Set at least one SSI pin as not general purpose I/O by setting the corresponding control bit in the PCC register.

Hardware for the Codec Board

As the twin codec board is intended as a development tool for use in a wide variety of applications, a great deal of flexibility had to be built in to the codec interface. This has been achieved, but at the expense of the hardware simplicity which is possible when using the SSI interface.

The codec used for this board is the MC145503, one of the MC14550X range; the variety available from this range allows the user to select a device with as few, or as many, features as required. The MC145503 is the standard codec, with the addition of complete access to the on-chip op-amp; it is pin-compatible with the older MC14403 codec, which may be used equally well in the board.

Clock Generation

A clock and frame sync generator capable of supplying 2.048MHz have been included on the board. The frame sync generator is jumper configurable, allowing either an 8KHz or a 16KHz frame sync, for use in single and dual codec applications respectively.

A further option exists with the serial communications clock and frame sync signals; they may be generated by the DSP processor. It is therefore possible to have split communication rates, with transmission and reception clocks being generated by the DSP processor and external clock respectively if required.

Codec Selection

One of the more involved parts of the circuit is the codec selection circuitry. This must allow either of the two codecs to be addressed; in many applications, it would be possible to accomplish this using the serial control lines, SC0 and SC1. However, these are multi-function lines. It is possible that some applications will require the use of them for frame synchronisation, or asynchronous clock inputs. In view of this, it was decided not to restrict the board to using them alone for codec selection.

A better answer was felt to be allowing the option of using one of the the serial control lines, or one of a pair of unused port lines; the lines chosen were PB0 and PC2. These were chosen in the hope that the user would not require the use of all the features of all three peripherals simultaneously; it is expected that at least one of the lines which may be used for codec selection will be available. PC2 is the SCI's serial clock line; as many applications of the SCI do not use the SCI in its synchronous mode, this line is almost always available. On the other hand, PB0 will always be in use where the host port is required.

The codec select line used must be synchronised to the serial data streams to prevent data corruption. Additional circuitry is not required for SC0, the serial control line used, as it is internally synchronised to the Transmit Frame Sync signal; however, PC2 and PB0 may change asynchronously with respect to the serial data streams, and thus require external synchronisation. This is the function of U5B; this is a D-type which synchronises the select line used to the rising edge of the transmit frame sync. It is recommended a 74F74 is used for this, as other types (eg 74HC74) will introduce an excessive propagation delay which will lead to data corruption.

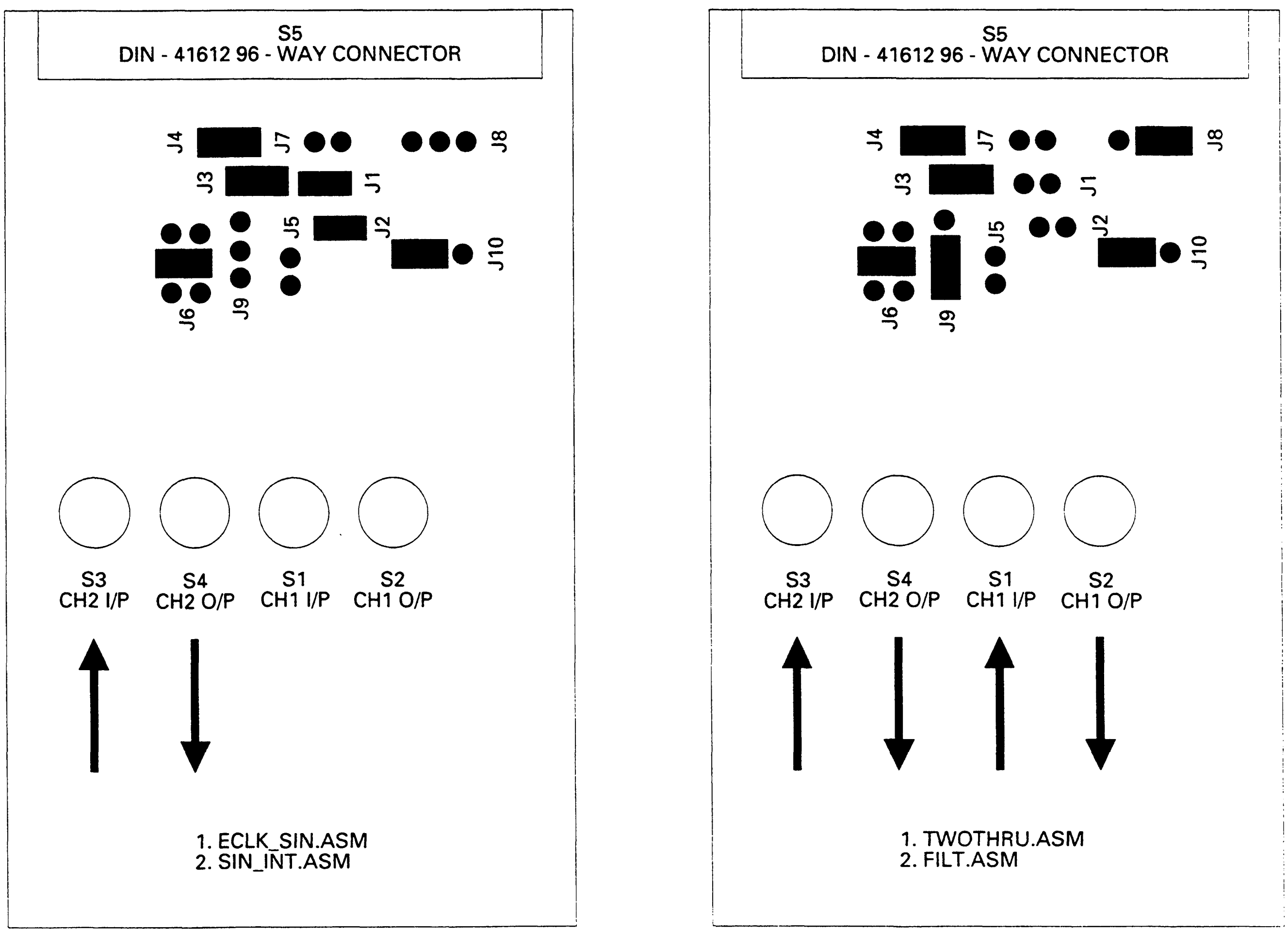
A second complication lies in the fact that some applications will require separate transmit and receive frame synchronisation. This has resulted in the gating of the TDE and RCE signals with the select line, separately; this allows the option of splitting the frame sync signals.

Power Supply
As the codec board has been designed specifically to interface to the ADS development system, power may be taken from this; however, -5V must be generated locally. This is accomplished using a 7660 voltage inverter.

Jumper Options
The following is a list of the jumpers provided, along with their function.

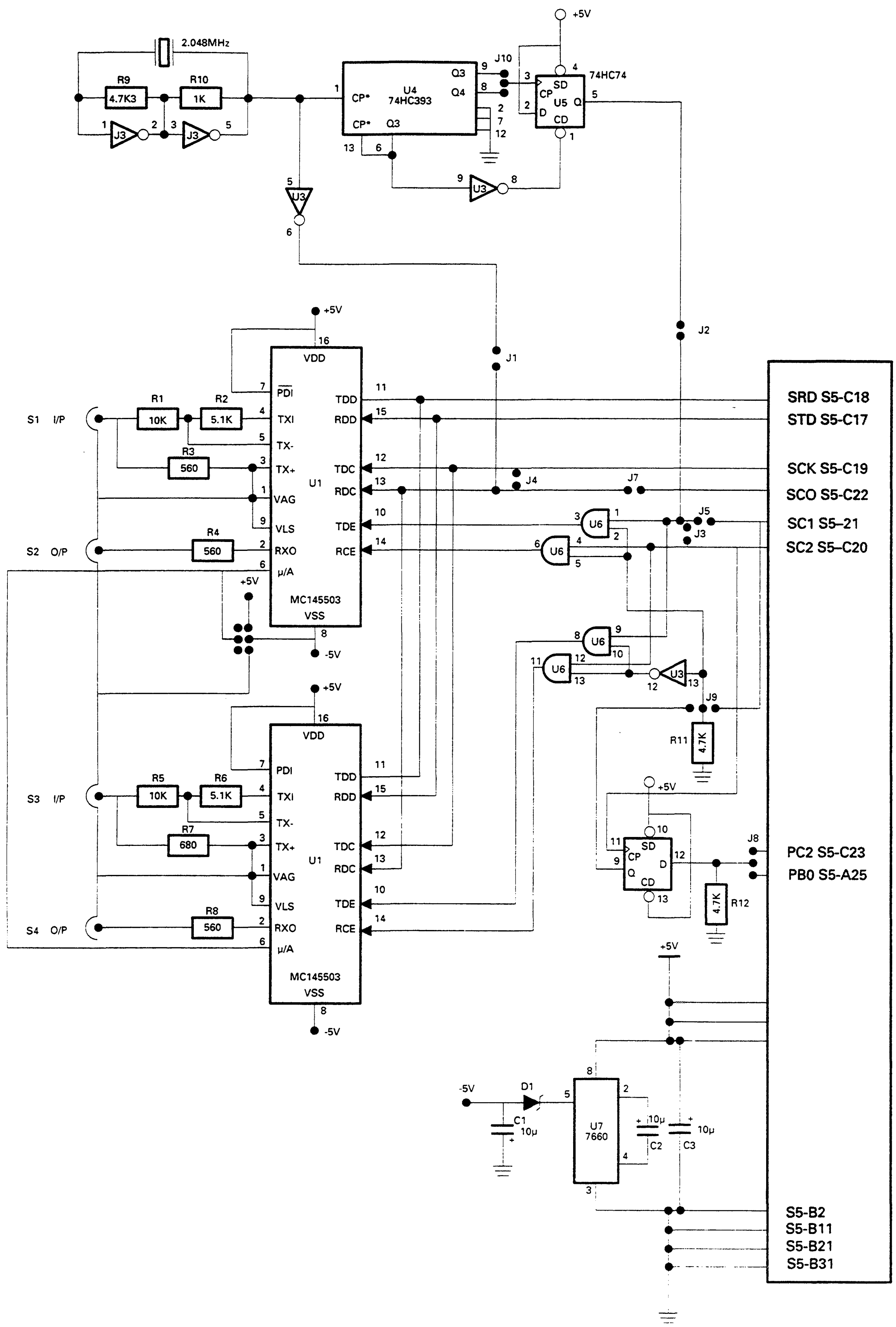
Jumper	Function
J1	Selects External Clock Generation (2.048MHz)
J2	Selects External Frame Sync Generation
J3	Selects Synchronous Receive/Transmit Frame Sync
J4	Selects Synchronous Receive/Transmit Data Clocks
J5	Disables DSP's Receive Frame Sync
J6	Selects operating mode of codecs
J7	Disables DSP's serial clock
J8	Select either PC2 or PB0 for codec selection
J9	Select either SC0 or the output of J8 for codec selection.
J10	Select either 8KHz or 16KHz frame sync generation

The jumper settings for the example software included in this document are detailed below:-



Configuration of the Twin Codec Board for Example Software

Appendix A Twin Codec Expansion Board Schematics



Appendix B Twin Codec Expansion Board Parts List

U1,2	MC145503 PCM Codec
U3	74HC04
U4	74HC393
U5	74F74
U6	74HC08
U7	Maxim 7660
R1,5	10KΩ
R2,6	5.1KΩ
R3,7	680Ω
R4,8	560Ω
R9,11,12	4.7KΩ
R10	1KΩ
C1,2,3	10µ tantalum
C4-	0.1µ (decouplers)
D1	Schottky Diode
X1	2.048MHz Crystal
S1,2,3,4	'Square Pad' BNC Connectors
S5	DIN41612 96-way connector

Appendix C
Demo Software for Twin Codec Board

EXAMPLE 1

```
;*****************************************************************
;
; FILENAME: ECLK_SIN.ASM
;
; FUNCTION: CODE INITIALISES SSI TO INTERFACE TO A SINGLE CODEC, WITH
; SYNCHRONOUS Rx/Tx SECTIONS.
;
; J1 - ON      J6 - -5V
; J2 - ON      J7 - OFF
; J3 - ON      J8 - OFF
; J4 - ON      J9 - OFF
; J5 - OFF
;
; J10 SHOULD BE SET FOR 8KHz FRAME SYNC
; (THIS IS THE POSITION AWAY FROM THE J10 LETTERING ON THE PCB)
;
;*****************************************************************

            include     '\dsp\demo\ioequ'       ;look for IOEQU.ASM
            org         p:$40

;*****************************************************************
; program code
;*****************************************************************

start       move        #M_SR,r2                ; set up r2 for often-used register
            reset

; SETUP FOR EXTERNAL CLOCK

            movep       #0,x:M_CRA              ; PSR=0 , WL=0 , DC4-0=$13 ,PM7-0=1
            movep       #$3200,x:M_CRB ; RIE=0,TIE=0, RE,TE = 1
            movep       #$1f8,x:M_PCC  ; set CC(8:3) as SSI pins

;*****************************************************************
; wait for transmission and reception
;*****************************************************************

wait
wtde        jclr        #M_TDE,x:(r2),wtde      ; wait for tde
            movep       x0,x:M_TX               ; write data to TX reg.
wrdf        jclr        #M_RDF,x:(r2),wrdf      ; wait for rdf
            movep       x:M_RX,x0               ; read data from RX reg.
            jmp         wait
```

EXAMPLE 2

```
;*****************************************************************
;
; FILENAME: SIN_INT.ASM WRITTEN : 13/4/88
;
; FUNCTION: CODE INITIALISES SSI TO INTERFACE TO A SINGLE CODEC, WITH
; SYNCHRONOUS Rx/Tx SECTIONS. DATA TRANSFERS USE FAST INTERRUPT
;
; J1 - ON   J6 - -5V
; J2 - ON   J7 - OFF
; J3 - ON   J8 - OFF
; J4 - ON   J9 - OFF
; J5 - OFF
;

; J10 SHOULD BE SET FOR 8KHz FRAME SYNC
; (THIS IS THE POSITION AWAY FROM THE J10 LETTERING ON THE PCB)
;
;
;*****************************************************************

          include    '\dsp\demo\ioequ'                ;look for IOEQU.ASM

;*****************************************************************
; reset vector
;
; not normally required for the ADS; however, this will allow the user to load and
; run this file directly, without changing the PC from the ADS's default.
;
;*****************************************************************

          org        p:$00
reset     jmp        start


;*****************************************************************
; interrupt routines
;*****************************************************************

          org        p:$0C
ssi_rx    movep      x:M_RX,a
          nop

          org        p:$10
ssi_tx    movep      a,x:M_TX
          nop

          org        p:$40

;*****************************************************************
; program code
;*****************************************************************

start     move       #M_SR,r2                        ; set up r2 for often-used register
          reset
          movep      #$3000,x:M_IPR ; enable SSI interrupts on level 2
          movep      #[illegible],x:M_CRA            ; PSR=0 , WL=0 , DC4-0=$13 ,PM[illegible]-0=[illegible]
          movep      #$[illegible],x:M_CRB ; RIE=[illegible],TIE=[illegible], RE,TE = 1
          movep      #$1[illegible],x:M_[illegible] ; set [illegible](8:3) as SSI pins
          [illegible] #[illegible],[illegible]      ; enable interrupts
wait      jmp        wait                            ; wait for interrupt
```

EXAMPLE 3

```
          PAGE      132,66,3,3

;*****************************************************************
;
; FILENAME: TWOTHRU.ASM WRITTEN : 13/4/88
; HISTORY : THE BEGINNING
;
; FUNCTION: CODE INITIALISES SSI TO INTERFACE TO TWO CODECS, WITH
; SYNCHRONOUS Rx/Tx SECTIONS. CODEC SELECTION IS PERFORMED
; USING PC2 OR PB0
; DATA IS READ FROM EACH CODEC, AND OUTPUT TO THE SAME CODEC.
;
; BOARD CONFIGURATION
;
; J1        OFF                     J6      -5V
; J2        OFF                     J7      OFF
; J3        ON                      J8      END NEAREST PCB LETTERING
; J4        ON                      J9      END NEAREST PCB LETTERING
; J5        OFF                     J10     DON'T CARE
;
;***************************************************************

          include   '\dsp\demo\ioequ'     ;look for IOEQU.ASM
          org       p:$40

;***************************************************************
; program code
;***************************************************************

start     move      #M_SR,r2              ; set up r2 for often-used register
          reset

; SETUP FOR INTERNAL CLOCK

          movep     #$1301,x:M_CRA ; PSR=0 , WL=0 , DC4-0=$13 ,PM7-0=1
          movep     #$3234,x:M_CRB ; RIE=0,TIE=0, RE,TE = 1
          movep     #$1f8,x:M_PCC  ; set CC(8:3) as SSI pins
          movep     #$1,x:M_PBDDR  ; port B as I/O lines

;***************************************************************
; wait for transmission and reception
;***************************************************************
wait
wtde1     jclr      #M_TDE,x:(r2),wtde1   ; wait for tde
          bset      #0,x:M_PBD            ; set PB0 line for codec one
          movep     X0,x:M_TX             ; write data to TX reg.
wrdf1     jclr      #M_RDF,x:(r2),wrdf1   ; wait for rdf
          movep     x:M_RX,x0             ; read data from RX reg.

wtde2     jclr      #M_TDE,x:(r2),wtde2   ; wait for tde
          bclr      #0,x:M_PBD            ; clear PB0 line for codec two
          movep     X1,x:M_TX             ; write data to TX reg.
wrdf2     jclr      #M_RDF,x:(r2),wrdf2   ; wait for rdf
          movep     x:M_RX,X1             ; read data from RX reg.
          jmp       wait
```

EXAMPLE 4

```
          PAGE      132,66,3,3
          OPT       MEX

;*************************************************************
;
; FILENAME: FILT.ASM WRITTEN : 13/4/88
; HISTORY : THE BEGINNING
;
; FUNCTION: CODE INITIALISES SSI TO INTERFACE TO TWO CODECS, WITH
; SYNCHRONOUS Rx/Tx SECTIONS. CODEC SELECTION IS PERFORMED
; USING PC2 OR PB0
; DATA IS READ FROM EACH CODEC, AND OUTPUT TO THE SAME CODEC.
;
; BOARD CONFIGURATION
;
; J1        OFF                 J6        -5V
; J2        OFF                 J7        OFF
; J3        ON                  J8        END NEAREST PCB LETTERING
; J4        ON                  J9        END NEAREST PCB LETTERING
; J5        OFF                 J10       DON'T CARE
;
;*************************************************************

          include   '\dsp\demo\ioequ'        ;look for IOEQU.ASM
          maclib    '\dsp\macros\filter'
          maclib    '\dsp\macros\compand'

          org       p:$40

;*************************************************************
; data for filters
;*************************************************************

xfilt_ad  equ       0                        ; address for x filter storage
n_pts_x   equ       39                       ; number of points in x filter
codx_op   equ       xfilt_ad+n_pts_x         ; temporary store for x filter output

yfilt_ad  equ       128                      ; address for y filter storage
n_pts_y   equ       99                       ; number of points in y filter
cody_op   equ       yfilt_ad+n_pts_y         ; temporary store for y filter output


;*************************************************************
; program code
;*************************************************************

start     move      #M_SR,r2        ; set up r2 for often-used register
          move      #4,omr          ; set data ROM's on for log/linear data access
          reset                     ; reset on-chip peripherals in case not already done

; set up filter data

          init_fir  0,4,n_pts_x,xfilt_ad     ; initialise filter 1
          init_fir  3,7,n_pts_y,yfilt_ad     ; initialise filter 2
```

```
; SETUP FOR SSI INTERNAL CLOCK, SET UP SSI

          movep     #$1,x:M_PBDDR ; port B as I/O lines
          movep     #$1301,x:M_CRA ; PSR=0 , WL=0 , DC4-0=$13 ,PM7-0=1
          movep     #$3234,x:M_CRB ; RIE=0,TIE=0, RE,TE = 1
          movep     #$1f8,x:M_PCC  ; set CC(8:3) as SSI pins

; transmit initial data for codecs

w1        jclr      #M_TDE,x:(r2),w1          ; wait for tde
          bset      #0,x:M_PBD                ; set PB0 line for codec one
          movep     #$d5,x:M_TX               ; write first data to TX reg.
w2        jclr      #M_TDE,x:(r2),w2          ; wait for tde
          bclr      #0,x:M_PBD                ; clear PB0 line for codec two
          movep     #$d5,x:M_TX               ; write first data to TX reg.

;****************************************************************
; wait for transmission and reception
;****************************************************************

wait
          bclr      #0,x:M_PBD                ; set PB0 line for codec one
wtde1     jclr      #M_TDE,x:(r2),wtde1       ; wait for tde
          movep     x:codx_op,x:M_TX          ; write data to TX reg.
wrdf1     jclr      #M_RDF,x:(r2),wrdf1       ; wait for rdf
          movep     x:M_RX,x0                 ; read data from RX reg.
          allin                               ; convert to linear data
          fir_filt  r0,r4,n_pts_x ; filter using filter 1
          linal                               ; back to logarithmic data for o/p
          move      a1,x:codx_op              ; store

          bset      #0,x:M_PBD                ; clear PB0 line for codec two
wtde2     jclr      #M_TDE,x:(r2),wtde2       ; wait for tde
          movep     x:cody_op,x:M_TX          ; write data to TX reg.
wrdf2     jclr      #M_RDF,x:(r2),wrdf2       ; wait for rdf
          movep     x:M_RX,x0                 ; read data from RX reg.
          allin                               ; convert to linear for processing
          fir_filt  r3,r7,n_pts_y ; filter in filter 2
          linal                               ; back to log for output
          move      a1,x:cody_op              ; store
end       jmp       wait


;*****************************************************************************
; filter coefficients
;
; FILT1 is a LPF, 3dB cutoff at 2KHz
; FILT2 is a BPF, 3dB points at 1.6KHz and 2.4KHz
;
;*****************************************************************************
```

```
                radix   16
                org     y:xfilt_ad

FILT1           DC      00060B,002157,003775,FFF959,FF9A4E,FFF841,00C389,00232D
                DC      FEA110,FFB51B,0257F6,007B50,FC1572,FF51F1,06A75B,00DAF9
                DC      F36611,FF0606,2848B8,40CD26,2848B8,FF0606,F36611,00DAF9
                DC      06A75B,FF51F1,FC1572,007B50,0257F6,FFB51B,FEA110,00232D
                DC      00C389,FFF841,FF9A4E,FFF959,003775,002157,00060B

                org     y:yfilt_ad

FIR64_2         DC      000000,00016A,000000,FFFAD4,000000,0009BF,000000,FFF5BF
                DC      000000,000000,000000,0018A7,000000,FFC5F6,000000,0051D8
                DC      000000,FFBA7B,000000,000000,000000,007B83,000000,FEFC0E
                DC      000000,014DD5,000000,FEF9C5,000000,000000,000000,019ED4
                DC      000000,FCB7BA,000000,041DEB,000000,FCC911,000000,000000
                DC      000000,057118,000000,F3C20B,000000,12EAFE,000000,E8322E
                DC      000000,1998C5,000000,E8322E,000000,12EAFE,000000,F3C20B
                DC      000000,057118,000000,000000,000000,FCC911,000000,041DEB
                DC      000000,FCB7BA,000000,019ED4,000000,000000,000000,FEF9C5
                DC      000000,014DD5,000000,FEFC0E,000000,007B83,000000,000000
                DC      000000,FFBA7B,000000,0051D8,000000,FFC5F6,000000,0018A7
                DC      000000,000000,000000,FFF5BF,000000,0009BF,000000,FFFAD4
                DC      000000,00016A,000000
```

EXAMPLE 5

```
;*****************************************************************************
;
; AM Modulator
;
; This example uses the twin codec board to acquire two signals.
; One is output without alteration, and also to modulate the second.
;
; BOARD CONFIGURATION
;
; J1        ON                J6     -5V
; J2        ON                J7     OFF
; J3        ON                J8     END NEAREST PCB LETTERING
; J4        ON                J9     END NEAREST PCB LETTERING
; J5        OFF               J10    END NEAREST PCB LETTERING
;
;*****************************************************************************


          include  '\dsp\demo\ioequ'        ;look for IOEQU.ASM
          maclib   '\dsp\macros\compand'

; following is the reset vector
          org      p:0
          jmp      start

; these are the SSI interrupt vectors.  Only one pair are used
          org      p:$c
          jsr      interpt
          jsr      interpt

; x memory reservations
          org      x:
out1      ds       1             ; storage for data for codec one o/p
out2      ds       1             ; storage for data for codec two o/p
in1       ds       1             ; data received from codec 1
in2       ds       1             ; data received from codec 2

;**********************************************************************
; Start of Program
; First Step - Initialisation of hardware and software
;**********************************************************************

          org      p:$40
start     reset                     ; clear out processor
          movep    #$128,x:M_CRA ; PSR=0 , WL=0 , DC4-0=$13 ,PM7-0=1
          movep    #$b200,x:M_CRB   ; enable Tx interrupt, external clock
          movep    #$1ff,x:M_PCC ; set CC(8:3) as SSI pins
          move     #6,omr           ; enable data ROM's
          movep    #$2300,x:M_BCR
          movep    #$1,x:M_PBDDR ; port B as I/O lines
          movep    #$3000,x:M_IPR   ; set SSI interrupts to level 2
          move     #2,sr            ; and enable interrupts

runtime   jmp      runtime          ; and continue waiting for more data
```

```
;*************************************************************************
; this is the interrupt routine
;*************************************************************************

interpt     jset    #0,x:M_PBD,proc     ; if PB0 line set, do frame processing
            bset    #0,x:M_PBD          ; if not, set PB0 line for codec one
            movep   x:out1,X:M_TX       ; and write output data to codec one
            movep   X:M_RX,x:in2        ; read input data read from codec two
            rti

proc        bclr    #0,x:M_PBD          ; if not, clear PB0 line for codec two
            movep   x:out2,X:M_TX       ; and write output data to codec two
            movep   X:M_RX,x:in1        ; read input data read from codec one
            andi    #$fc,mr             ; re-enable interrupts to allow I/O during processing
crdd        jsr     process             ; process data
            rti

;*************************************************************************
; This routine performs signal processing tasks on the data
; As an example, one channel is used to modulate the other
;*************************************************************************

process     move    x:in1,a             ; read sample from channel 1
            move    a,x:out1            ; output to same channel
            allin                       ; convert channel1 data to linear
            move    a1,y0               ; and transfer for multiply
            move    x:in2,a             ; read sample from channel 2
            allin                       ; convert channel 2 to linear
            move    a1,x0               ; transfer back for multiply

            mpy     x0,y0,a             ; perform multiply for AM modulation
            linal                       ; convert result to log format
            move    a1,x:out2           ; output modulated result to channel 2
            rts
```

Appendix D Example Filter Description

Filter Implementation Techniques

The filters used as examples in the demo software are all implemented around two software macros, listed below. All are FIR filters, with different numbers of taps, and were designed using a proprietary digital filter CAD system.

The macros were implemented to allow the rapid creation of different froms of FIR filter. They cover both initialisation and execution of the filter.

The first macro, INIT_FIR, is passed various parameters indicating to the assembler which registers to use for the filter, how many taps are in the filter, and what memory area to use; it then initialises the DSP to perform this filter. Note that this generates filters with symmetrical memory usage; i.e. if the filter uses the first 100 locations of X-memory, it will also use the first 100 locations of Y-memory.

The second macro, FIR_FILT, performs one pass of the FIR filter algorithm. It must be passed the register pair used for data and coefficient access.

MACRO - INIT_FIR

```
;*****************************************************************************
; FIR Filter Initialisation macro
;
; Calling Procedure      :     init_fir  coeff,data,points,address
;
; Parameters             :     coeff  - number of coefficient address register set
;                              in the range 0-7 (ie R0/N0/M0 - R7/N7/M7)
;                        :     data   - number of data address register set
;                              in the range 0-7 (ie R0/N0/M0 - R7/N7/M7)
;                        :     points  - number of points in filter
;                        :     address - X/Y memory area to be used for data
;                              and coefficients
;
; Comments               :     coeff and data should not be in the same address register
;                              group; ie one may be in group 0-3, the other group 4-7
;                              This initialisation routine sets up the filter to use the same
;                              locations in X & Y memory.
;
;*****************************************************************************

init_fir       MACRO             COEFF,DATA,POINTS,ADDRESS      move
#?ADDRESS,r\COEFF
               move              #?ADDRESS,r\DATA
               move              #?POINTS-1,m\COEFF
               move              #?POINTS-1,m\DATA
               ENDM
```

MACRO - FIR_FILT

```
;*************************************************************************
; FIR filter macro ; ;   input linear data in a1 ;   output result in a1 ;
;*************************************************************************

fir_filt       MACRO             COEFF,DATA
               move              a1,x0
               clr               a            x0,x:(DATA)+    y:(COEFF)+,y0
               rep               #n_pts_1-1
               mac               x0,y0,a      x:(DATA)+,x0    y:(COEFF)+,y0             macr
x0,y0,a        (DATA)-
ENDM
```

Filter 1

The first filter was implemented using the FDAS filter design package, available from Momentum Data Systems. The filter was designed using the Parks-Maclellan design methodology; the initial specification for the filter is given below.

Sample Rate	8KHz
Filter Type	LPF
Upper Limit of Pass Band	1.4KHz
Lower Limit of Stop Band	2.1KHz
Pass Band Ripple	-0.1dB
Stop Band Ripple	-78dB
Number of Taps	39

This gives the filter of Figure 1A; this transfer function has been evaluated using extended floating point arithmetic, and is thus the closest we can achieve to the theoretically ideal filter. However, few DSP processors will work to this type of accuracy; the coefficients must be truncated to fit in the word length of the processor used. In the case of the 56000, the word length is 24 bits; Figure 1B is the realisable transfer function when the coefficients are quantised for this word length.

As can be seen, truncating the coefficients to 24 bits has had no serious effect on the filter's transfer function. This is not always the case; for example, truncating the coefficients to 16 bits significantly alters the stopband characteristics of the filter.

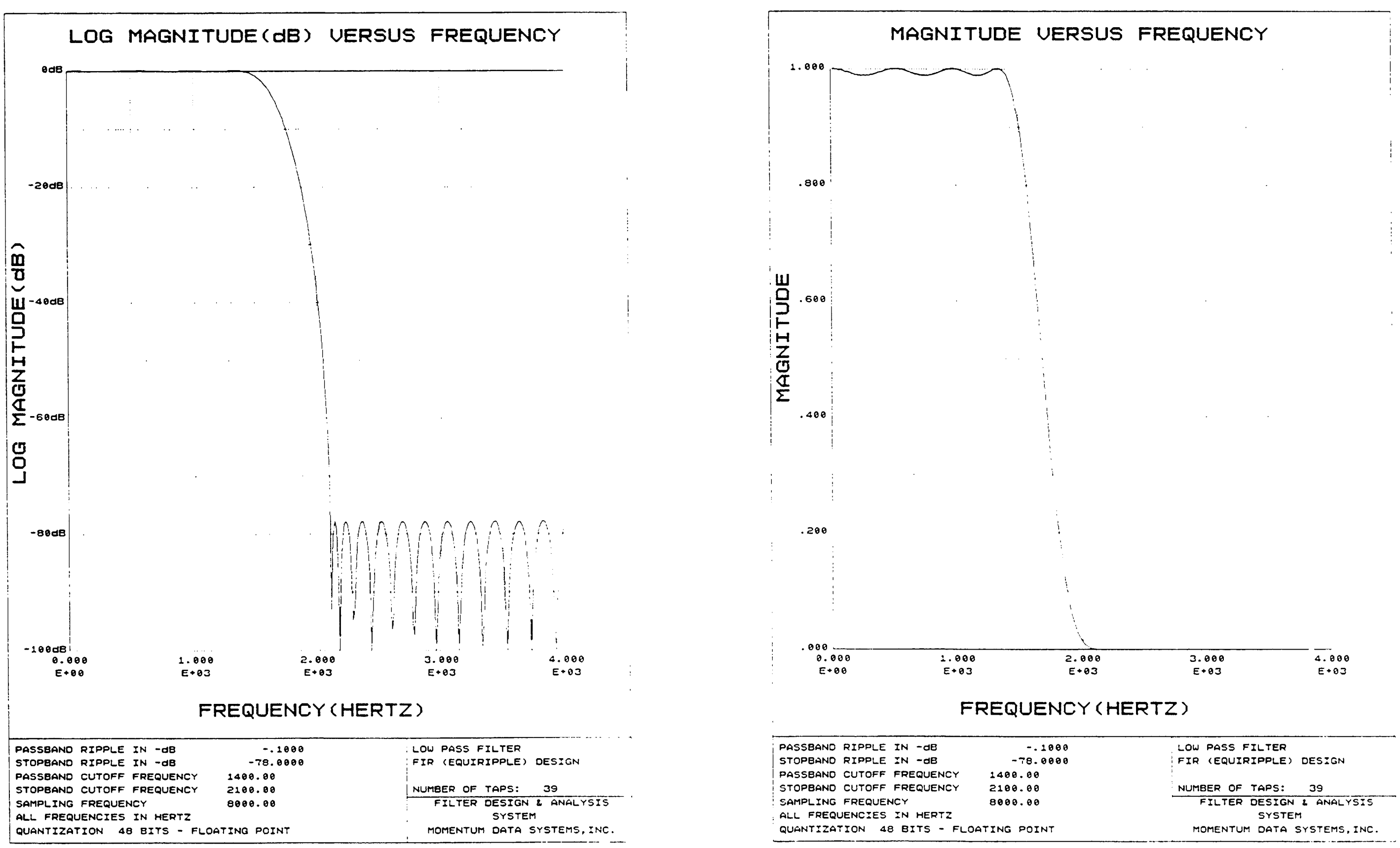
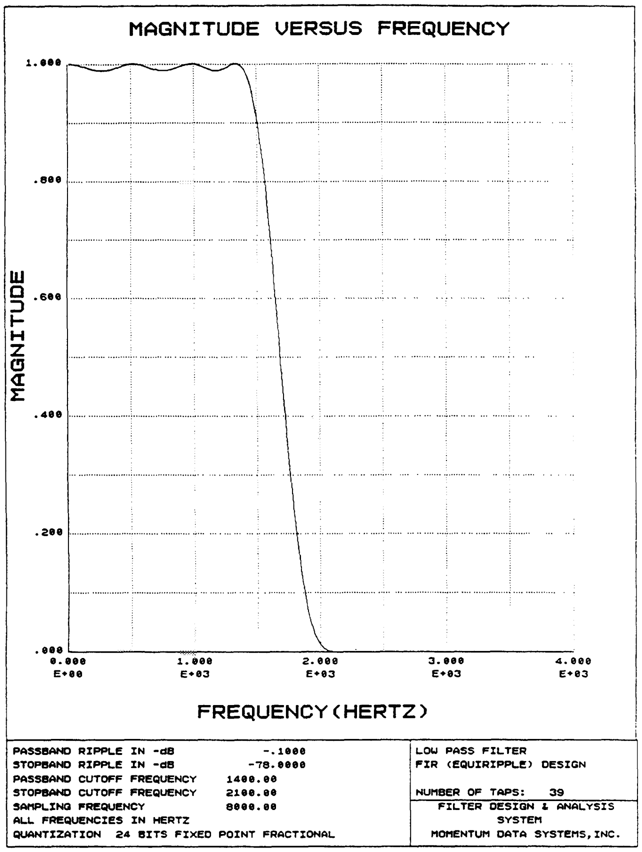


Figure 1A
Theroretical Log & Magnitude Plots
Low Pass Filter



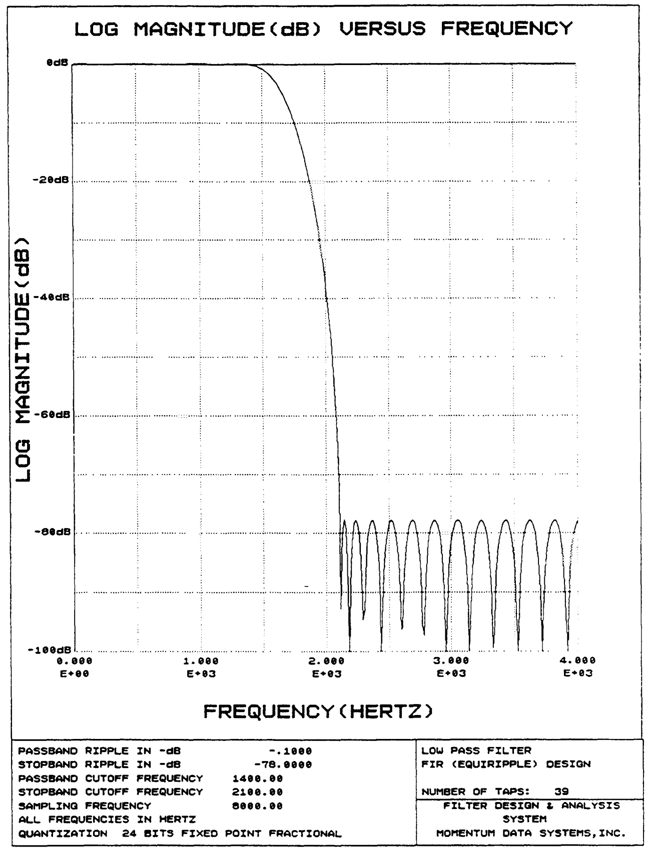


Figure 1B
Realisable Log & Magnitude Plots
Low Pass Filter

Filter 2

The second filter was again implemented using the FDAS filter design package, available from Momentum Data Systems. The filter was again designed using the Parks Maclellan design methodology; the initial specification for the filter is given below.

Sample Rate	8KHz
Filter Type	BPF
Upper Limit of Pass Band	2.2KHz
Lower Limit of Pass Band	1.8KHz
Upper Limit of Stop Band	1.4KHz
Lower Limit of Stop Band	2.6KHz
Pass Band Ripple	-0.1dB
Stop Band Ripple	-78dB
Number of Taps	69

This gives the filter of Figure 2; this is the 24-bit version of the filter.

It should be noted that when using a codec, only around 78dB of resolution is available. These filters were designed with that fact in mind; the 56000 will support filters with cut-offs of -144dB. In this application, such a filter would be excessively powerful.

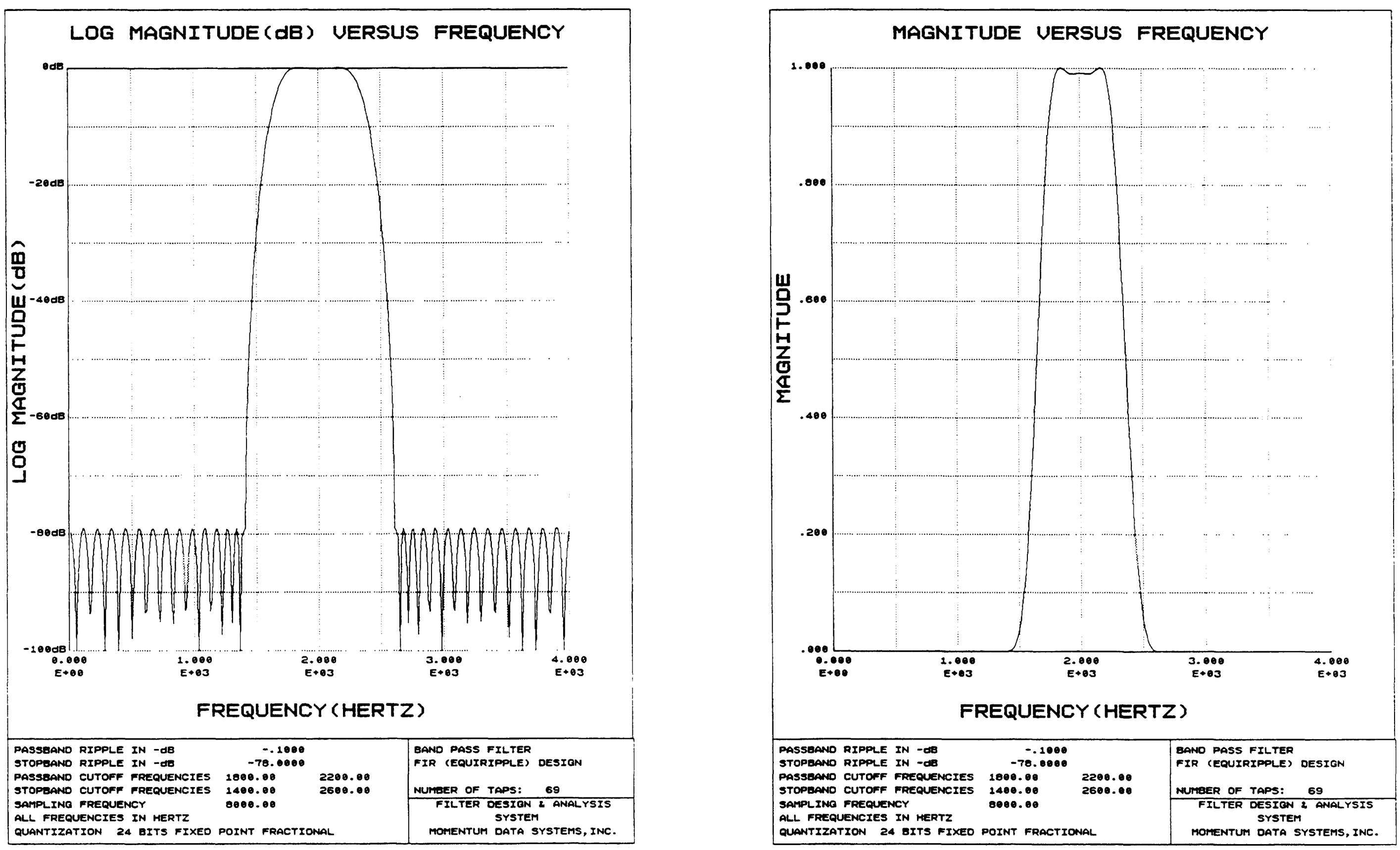


Figure 2
Realisable Log & Magnitude Plots
Band Pass Filter

```
;
; This program originally available on the Motorola DSP bulletin board.
; It is provided under a DISCLAMER OF WARRANTY available from
; Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
; Linear PCM to Companded CODEC Data Conversion Macros
;
; Last Update 20 Apr 87   Version 1.0
;
linlog  ident   1,0
;
; These macros convert 13 bit, linear fractional data into 8 bit companded
; data suitable for transmission to CODEC D/A converters used in
; telecommunications applications. Four companded formats are
; supported for the Motorola MC14400 CODEC series and similar devices.
;
; Macro Calls:         linsm - linear to sign magnitude conversion
;                              with mu-law companding.
;                      linmu - linear to mu-law companded conversion
;                              without zero code suppression.
;                      lind3d4 - linear to mu-law companded conversion
;                              with D3/D4 format zero code suppression.
;                      linal - linear to a-law companded conversion
;                              with CCITT (G7.12) format.
;
;                      No macro arguments are required.  However, these
;                      macros assume that the scaling modes are off
;                        (S1=0, S0=0).
;
; Input data is a 56 bit number in accumulator a.  Although any 56 bit
; number may be used, the 13 bit linear fraction is assumed to be in
; the most significant bits of a1.  Values outside this fractional range
; are automatically converted to a maximum positive or negative companded
; value (dynamic range limiting).
;
; Output data is in the 8 most significant bits of a1.  The 16 LSB's
; of a1 are zero.
;
; ______________________________
; | Sign |    Chord Number    |       Step Number        |
; | Bit  |                    |                          |
; |__23__|__22_____21_____20__|__19_____18_____17_____16__|
;
; Alters Data ALU Registers
;       x1      x0
;       a2      a1      a0      a
;       b2      b1      b0      b
;
; Alters Address Registers
;       r0
;
; Alters Program Control Registers
;       pc      sr
;
; Uses 0 locations on System Stack
;
; Latest Revision - April 15, 1987
; Tested and verified - April 20, 1987
;
```

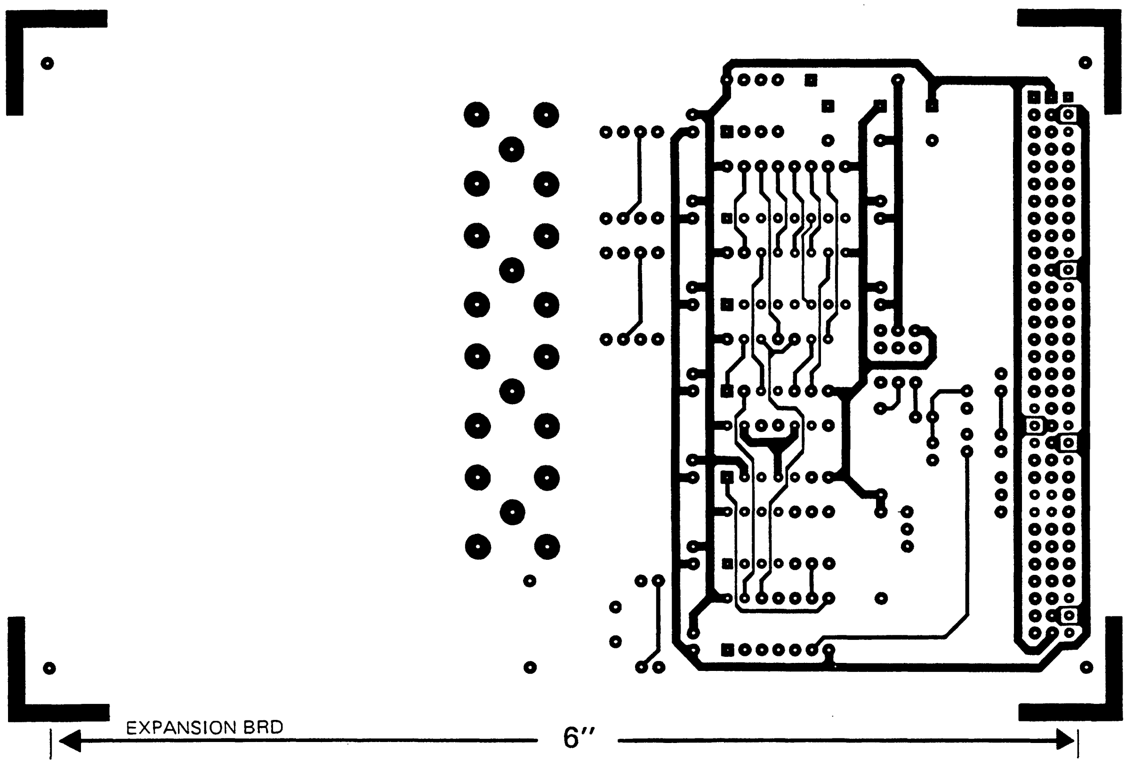
```
; linsm - linear to sign magnitude conversion
;
linsm   macro
_bias   equ     $008400                 ;absolute bias = 33
;
        tfr     a,b     a,a             ;save input sign, limit input data
        abs     a       #_bias,x0       ;form input magnitude, get bias
        add     x0,a    #7,r0           ;add bias to magnitude, get chord bar
        move            a,a             ;limit again
        rep     #7                      ;find chord number by normalizing
        norm    r0,a                    ; biased magnitude to get step number
        asl     a                       ;isolate step number
        asl     a       b,b             ;limit input again
        neg     b       r0,a2           ;invert sign bit, get chord number
        asr     a                       ;combine chord and step
        asr     a
        asr     a
        asl     b                       ;get sign bit
        ror     a       #<$ff,x0        ;combine sign, chord and step
        and     x0,a                    ;clear 16 LSB's
        endm
;
; linmu - linear to mu-law conversion
;
linmu   macro
_bias   equ     $008400                 ;absolute bias = 33
;
        tfr     a,b     a,a             ;save input sign, limit input data
        abs     a       #_bias,x0       ;form input magnitude, get bias
        add     x0,a    #7,r0           ;add bias to magnitude, get chord bar
        move            a,a             ;limit again
        rep     #7                      ;find chord number by normalizing
        norm    r0,a                    ; biased magnitude to get step number
        asl     a                       ;isolate step number
        asl     a       b,b             ;limit input again
        neg     b       r0,a2           ;invert sign bit, get chord number
        asr     a                       ;combine chord and step
        asr     a
        asr     a
        not     a                       ;invert 7 LSB's for mu-law
        asl     b                       ;get sign bit
        ror     a       #<$ff,x0        ;combine sign, chord and step
        and     x0,a                    ;clear 16 LSB's
        endm
;
; lind3d4 - linear to mu-law conversion with zero code suppression
;
lind3d4 macro
_bias   equ     $008400                 ;absolute bias = 33
;
        tfr     a,b     a,a             ;save input sign, limit input data
        abs     a       #_bias,x0       ;form input magnitude, get bias
        add     x0,a    #7,r0           ;add bias to magnitude, get chord bar
        move            a,a             ;limit again
        rep     #7                      ;find chord number by normalizing
        norm    r0,a                    ; biased magnitude to get step number
        asl     a                       ;isolate step number
        asl     a       b,b             ;limit input again
        neg     b       r0,a2           ;invert sign bit, get chord number
        asr     a                       ;combine chord and step
```

```
        asr     a
        asr     a
        not     a                       ;invert 7 LSB's for mu-law
        asl     b                       ;get sign bit
        ror     a       #<$ff,x0        ;combine sign, chord and step
        and     x0,a    #<$02,x0        ;clear 16 LSB's
        teq     x0,a                    ;suppress zero code
        endm
;
; linal - linear to a-law conversion
;
linal   macro
        tfr     a,b     a,a             ;save input sign, limit input data
        move            #1,a0           ;force to non-zero value
        abs     a       #7,r0           ;form input magnitude, get chord bar
        move            a,a             ;limit again
        rep     #6                      ;find chord number by normalizing
        norm    r0,a                    ; magnitude to get step number
        jnr     <_ok                    ;jump if normalized
        move            (r0)-           ;adjust for chord zero
_ok     asl     a                       ;isolate step number
        asl     a       b,b             ;limit input again
        neg     b       r0,a2           ;invert sign bit, get chord number
        asr     a                       ;combine chord and step
        asr     a
        asr     a
        asl     b                       ;get sign bit
        ror     a       #<$ff,x0        ;combine sign, chord and step
        and     x0,a    #<$55,x0        ;clear 16 LSB's
        eor     x0,a                    ;invert odd bits for a-law
        endm
;
; This program originally available on the Motorola DSP bulletin board.
; It is provided under a DISCLAMER OF WARRANTY available from
; Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
; Companded CODEC to Linear PCM Data Conversion Macros
;
; Last Update 20 Apr 87   Version 1.0
;
loglin  ident   1,0
;
; These macros convert 8 bit companded data received from CODEC A/D
; converters used in telecommunications applications to 13 bit, linear
; fractional data.  The internal mu/a-law lookup tables in the DSP56001
; X data ROM are used to minimize execution time.  Three companded
; formats are supported for the Motorola MC14400 CODEC series and
; similar devices.
;
; Macro Calls:          smlin - sign magnitude to linear conversion
;                               with mu-law companding.
;                       mulin - mu-law companded to linear conversion.
;                       allin - a-law companded to linear conversion
;                               with CCITT (G7.12) format.
;
;                       No macro arguments are required.  However, these
;                       macros assume that the scaling modes are off
;                         (S1=0, S0=0).
;
```

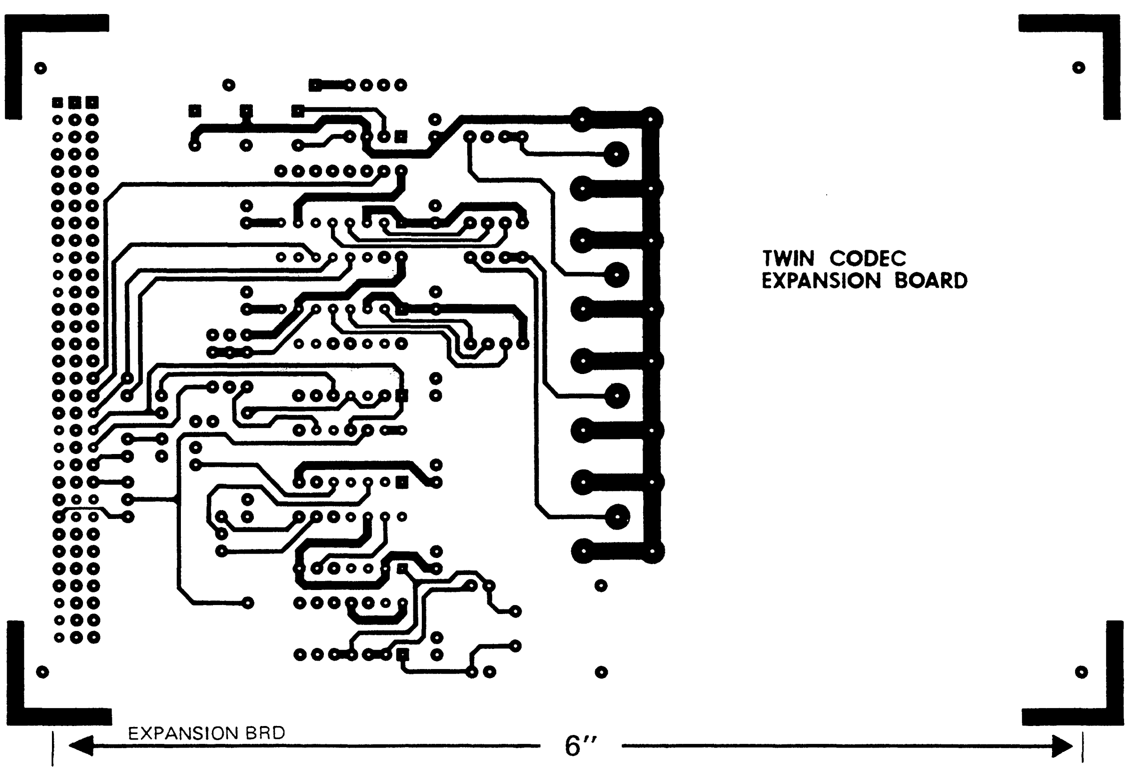
```
; Input data is in the 8 most significant bits of a1.  The remaining
; bits of a are ignored.
;
; _____________________________
; | Sign |    Chord Number     |       Step Number          |
; | Bit  |                     |                            |
; |__23__|__22_____21_____20__|__19_____18_____17_____16__|
;
; Output data is in the 56 bit accumulator a.  The linear fraction is
; in the 13 most significant bits of a1 and the 11 least significant
; bits are zero.
;
; Alters Data ALU Registers
;       x1      x0
;       a2      a1      a0      a
;       b2      b1      b0      b
;
; Alters Address Registers
;       r0
;
; Alters Program Control Registers
;       pc      sr
;
; Uses 0 locations on System Stack
;
; Latest Revision - April 15, 1987
; Tested and verified - April 20, 1987
;
; smlin - sign magnitude to linear conversion
;
smlin   macro
_shift  equ     $80                     ;shift constant
_mutable        equ     $100            ;base address of mu-law table
;
        not     a       a1,b            ;invert input bits, save input
        lsl     a       #>_shift,x0     ;shift out sign bit, get shift constant
        lsr     a       #_mutable,x1    ;shift in zero, get table base
        tfr     x1,a    a1,x1           ;swap table base and offset
        mac     x1,x0,a                 ;shift offset down and add to base
        move            a,r0            ;move to address register
        nop
        lsl     b       x:(r0),a        ;c=sign bit, lookup linear data
        neg     a       a,b             ;a=negative result, b=positive result
        tcs     b,a                     ;if pos sign, correct result
        endm
;
; mulin - mu-law to linear conversion
;
mulin   macro
_shift  equ     $80                     ;shift constant
_mutable        equ     $100            ;base address of mu-law table
;
        move            a1,b            ;save input
        lsl     a       #>_shift,x0     ;shift out sign bit, get shift constant
        lsr     a       #_mutable,x1    ;shift in zero, get table base
        tfr     x1,a    a1,x1           ;swap table base and offset
        mac     x1,x0,a                 ;shift offset down and add to base
        move            a,r0            ;move to address register
        nop
        lsl     b       x:(r0),a        ;c=sign bit, lookup linear data
```

```
        neg     a       a,b             ;a=negative result, b=positive result
        tcs     b,a                     ;if pos sign, correct result
        endm
;
; allin - a-law to linear conversion
; .
allin   macro
_shift  equ     $80                     ;shift constant
_atable equ     $180                    ;base address of a-law table
;
        move            a1,b            ;save input
        lsl     a       #>_shift,x0     ;shift out sign bit, get shift constant
        lsr     a       #_atable,x1     ;shift in zero, get table base
        tfr     x1,a    a1,x1           ;swap table base and offset
        mac     x1,x0,a                 ;shift offset down and add to base
        move            a,r0            ;move to address register
        nop
        lsl     b       x:(r0),a        ;c=sign bit, lookup linear data
        neg     a       a,b             ;a=negative result, b=positive result
        tcs     b,a                     ;if positive sign, correct result
        endm
```

Appendix F Twin Codec Expansion Board PCB Artwork

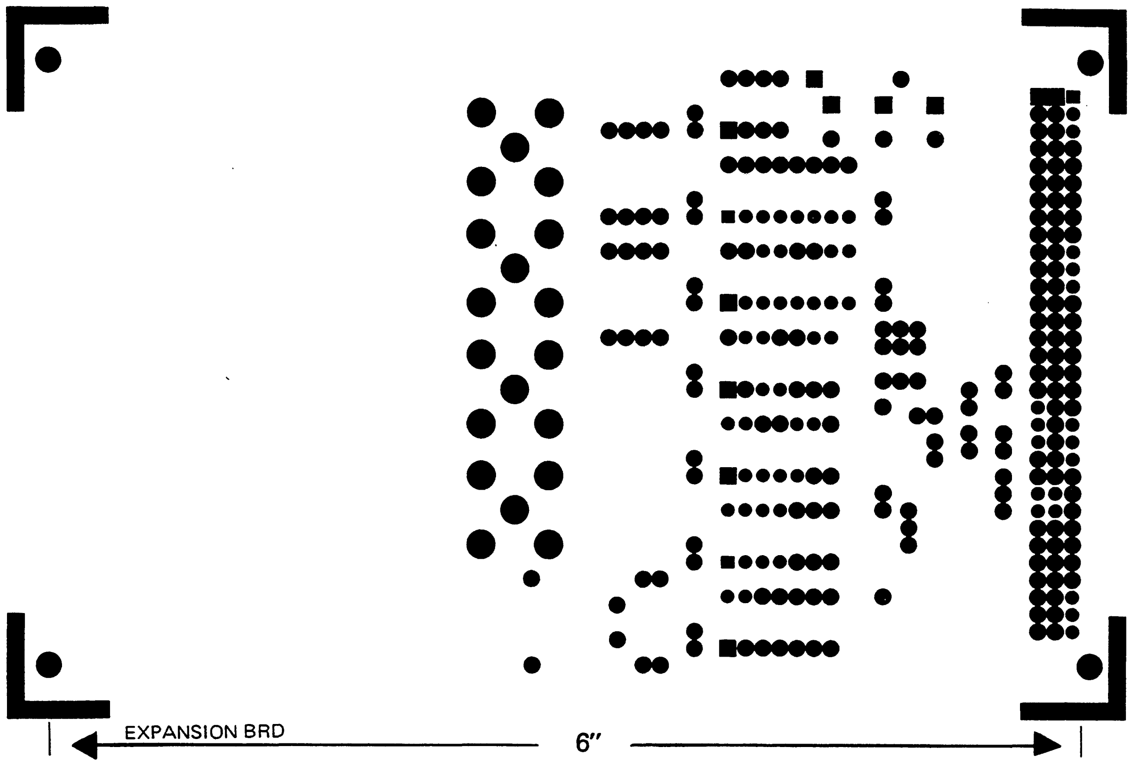


PCB Artwork
Component Side

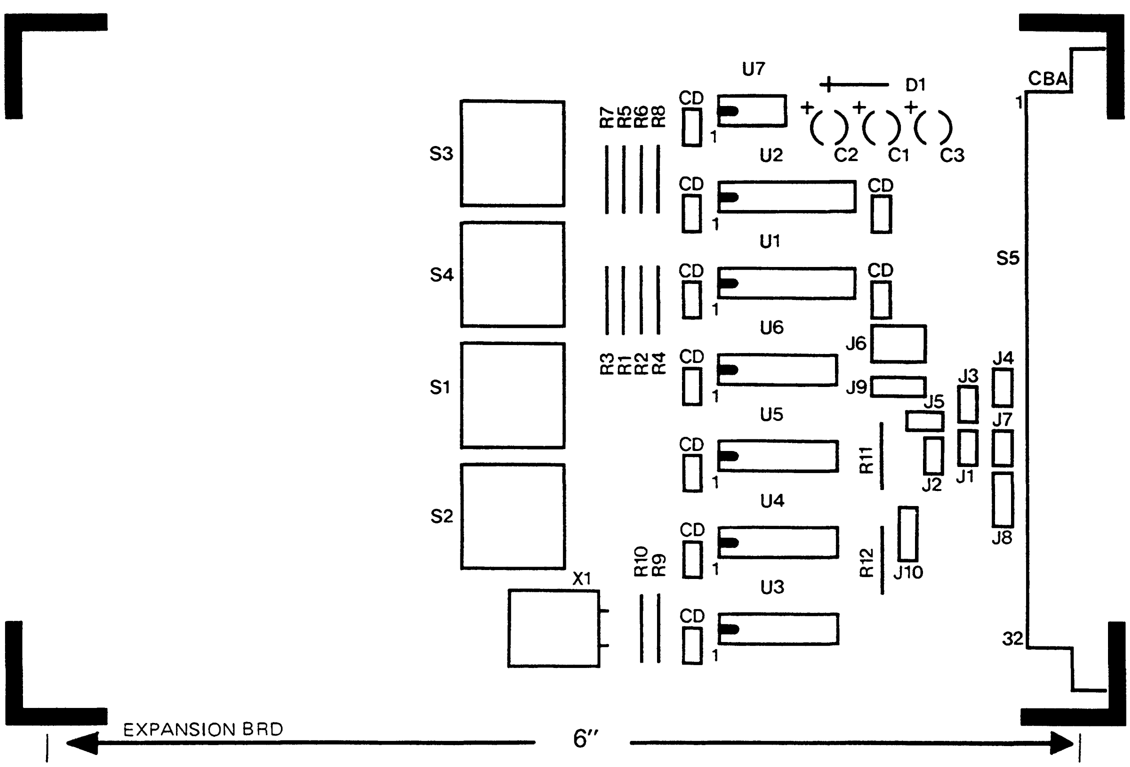


PCB Artwork
Solder Side

Appendix F Twin Codec Expansion Board PCB Artwork



Solder Resist Mask



Screen Printing

Literature Distribution Centers:
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.
ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

MOTOROLA
JIT PRINTED IN THE USA 1993 MPS
APR401/D