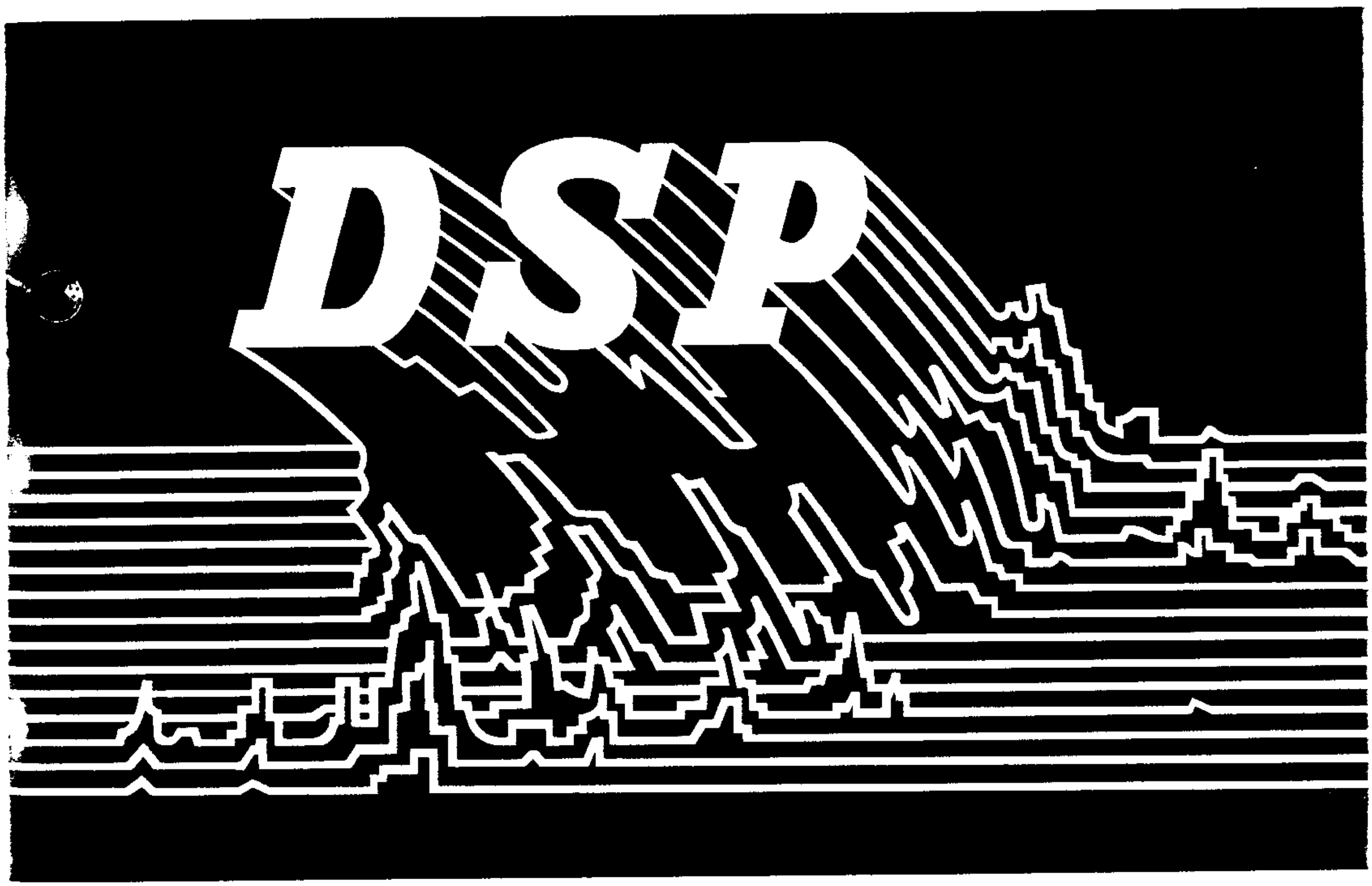
APR12/D

Twin CODEC Expansion Board for the DSP56000 Application Development System



MOTOROLA

Motorola Digital Signal Processors

Twin CODEC Expansion Board for the DSP56000 Application Development System

Prepared by Ralph Weir
DSP Applications
East Kilbride, Scotland
Eric Cheval
DSP Applications
Austin, Texas

 APR12/D

INTRODUCTION

The CODEC is an integrated analog-to-digital converter (ADC), digital-to-analog converter (DAC), and filter intended for use in telecommunications applications. It has been designed for a sample rate of 8 kHz, the standard telecommunications sampling frequency, and has a serial interface that may be used in a time-division multiplexed (TDM) system.

The filter implemented in the CODEC tailors the incoming analog signal for transmission through a telephone channel. It has a bandpass characteristic, cutting off at 300 Hz and 3.4 kHz; this also performs the anti-aliasing function for the ADC.

This application report describes a twin CODEC board designed to facilitate the development of telecommunication applications around the DSP56000 Family members. The board is intended for any situation where a DSP module is required to link two analog lines, as in the case of a line repeater or a telephone handset; it can also be used to develop applications requiring a single CODEC.

The CODEC board is designed to interface directly to the DSP, using the synchronous serial interface (SSI). The SSI is capable of generating all signals required for a serial CODEC, creating an interface with no glue logic. It is possible to create systems with many CODECs, all under the control of the DSP using 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 make conversion between the various data formats available from CODECs (linear, A-law, and Mu-law are the three main formats). It is essential that data be in a linear format before DSP processing. The conversion routines are listed in Appendix E; for more details, consult application note ANE408 or the Dr. BuB bulletin board. Printed circuit board (PCB) artwork for the expansion board is included in Appendix F.

DSP56000/DSP56001 SSI

The DSP56000 SSI is a powerful serial interface that may be used with many existing serial CODECs, COFIDECs (monocircuits), and serial ADCs/DACs. It is also a suitable medium for building serial networks of DSPs based on a TDM access protocol. A complete description of the interface can be found in DSP56000UM/AD Rev.2 *DSP56000/DSP56001 Digital Signal Processor User's Manual*, Section 7. A short description of the interface is presented in the following paragraphs.

The SSI is a six-pin interface that may be configured for synchronous or asynchronous exchange, continuous or gated clock, and normal or network mode.

Clocks may be generated internally and output or may be input from the external host system. The six pins are not necessarily used in all configurations, and unused lines may remain as general-purpose I/O.

Like every DSP56000/DSP56001 on-chip peripheral, the SSI is a full-duplex, double-buffered, memory-mapped peripheral, mapped into the peripheral area at the top of internal X-memory.

DATA REGISTERS

The SSI interface inputs and outputs the data using two 24-bit data registers mapped at X:$FFEF:

Write-Only Transmit Data Register (TX)

The transmit shift register is associated with the TX register. The transmit shift register shifts out the data written to the TX register onto the STD pin; when empty, it reads the data in the TX 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 occurs or may poll the SSI status flags.

Read-Only Receive Data Register (RX)

The receive shift register is associated with the RX register. The receive shift register formats the serial data read from the SRD pin; when full, it transfers the data to the RX register and sets a flag to indicate data is available. (If the data in the RX register is unread by the time the receive 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 occurs or may poll the SSI status flags.

CONTROL REGISTERS

Four control registers are associated with the SSI:

1. Control Register A (CRA)
2. Control Register B (CRB)
3. SSI Status Register (SSISR)
4. Time Slot Register (TSR)

CRA (16 Low Bits of X:$FFEC; Read/Write)

This register controls clock and frame sync generation, word length, and number of words per frame.

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

PSR — Prescaler Range
PSR = 1 enables a divided-by-8 prescaler in the clock generator.

WL1–WL0 — Word Length Control
Selects the number of bits per word (00 for 8-bit data).

DC4–DC0 — Frame Rate Divider Control
Controls the frame divider rate in network mode. In normal mode, these bits control the word transfer rate.

PM7–PM0 — Prescale Modulus Select
Selects the divide ratio for the clock generator.

According to the DSP crystal clock frequency and the required SSI clock rate, the various standard telecommunications frequencies may be generated. The following table details the values of PM7–PM0 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 and SC2, bits DC4–DC0 of CRA and the FSL bit of CRB have to be configured accordingly. For the MC145503 monocircuit, SC2 may be connected directly to TDE/RDE; since these lines should be high during word exchange, the FSL bit in CRB should be cleared.

The CODEC TDE line should be cycled at 8 kHz to provide the sampling rate clock. This defines DC4–DC0 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 16 kHz when the line is gated between two devices; thus, the DC4–DC0 value should be halved.

CRB (16 Low Bits of X:$FFED; Read/Write; Cleared by RESET)

This register controls interrupts from the SSI, enabling of the SSI, and clock frame sync formats. It also contains the control bits for serial control lines SC0–SC2.

RIE — Receive Interrupt Enable
Enables the RX interrupt; the DSP will be interrupted if data is read from the SSI RX register.

TIE — Transmit Interrupt Enable
Enables the TX interrupt; the DSP will be interrupted if data is written to the TX register for SSI transmission.

RE — Receive Enable
Enables the SSI receiver. If this bit is not set, the SSI will never receive any data.

TE — Transmit Enable
Enables the SSI transmitter. If this bit is not set, the SSI will never transmit any data.

MOD — Mode Select
Selects normal mode when clear, network mode when set.

GCK — Gated Clock Control
Selects continuous clock when clear, gated clock when set.

SYN — Synchronous/Asynchronous
Selects asynchronous mode when clear, synchronous mode when set.

FSL1 and FSL0 — Frame Sync Length Flags 1 and 0
Selects frame sync length — word length when clear, bit length when set.

SHFD — Shift Direction Flag
Transmit shift register shifts MSB first when clear.

SCKD — Clock Source
Selects external clock when clear, internal clock when set.

SCD2, SCD1 and SCD0 — Serial Control Direction Flags 2, 1 and 0
Controls the direction of the SC2, SC1 and SC0 lines; clear for input.

OF1 and OF0 — Serial Output Flags 1 and 0
Output data for SC1 and SC0 when configured as an output.

SR (8 Low Bits of X:$FFEE; Read-Only)

The SR provides 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 the last word has been fully transmitted since 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 condition 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 condition may cause an error with many serial devices; it 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 received frame occurs during the reception of a word when in network mode. This indicates the first time slot.

TFS — Transmit Frame Sync
Set when a received frame occurs during the transmission of a word when in network mode. This indicates the first time slot.

IF1 and IF0 — Serial Input Flags 1 and 0
These flags contain the data on the SC1 and SC0 lines when configured as input. They are latched from SC1 and SC0 during reception of the MSB of each incoming word.

TSR (X:$FFEE)

This 8-bit write-only time slot register is used in network mode; it behaves like an alternative TX data register that is written during unused time slots. In this case, rather than transmitting data, the STD pin will be three-stated during that time slot.

PORT C

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).

PCC (X:$FFE1; Read/Write)

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/0
PC6	SCK	Bidirectional	Serial Clock
PC7	SRD	Input	Receive Data
PC8	STD	Output	Transmit Data

Initializing the SSI

The recommended procedure for SSI initialization is as follows:

1. Reset the device. This can be a hardware reset, performed by driving the $\overline{\text{RESET}}$ pin low (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. Configure 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 into the CODEC interface. This flexibility has been achieved, but at the expense of the hardware simplicity, which is possible when using the SSI.

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 2.048 MHz have been included on the board. The frame sync generator is jumper configurable, allowing either an 8-kHz or a 16-kHz 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. It is therefore possible to have split communication rates, with transmission and reception clocks being generated by the DSP (SSI asynchronous mode) and external clock, respectively, if required.

CODEC SELECTION

One of the more involved parts of the circuit is the CODEC selection circuitry. This circuitry allows 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, since these are multifunction lines, it is possible that some applications will require the use of them for frame synchronization or asynchronous clock inputs. Thus, the board was not restricted to using them alone for CODEC selection.

A better solution allows the option of using one of the serial control lines or one of a pair of unused port lines; the lines chosen were PB0 and PC2. These lines were chosen assuming that the user would not simultaneously require the use of all features of all three peripherals; at least one of the lines that may be used for CODEC selection should be available. PC2 is the SCI serial clock line; since many SCI applications do not use the synchronous mode, this line is almost always available. On the other hand, PB0 will always be used when the host port is required.

The CODEC select line must be synchronized to the serial data streams to prevent data corruption. Additional circuitry is not required for SC0, the serial control line used, since it is internally synchronized to the transmit frame sync signal; however, PC2 and PB0 may change asynchronously with respect to the serial data streams and thus require external synchronization. Synchronization is the function of U5B; this is a D-type that synchronizes the select line used to the rising edge of the transmit frame sync. A 74F74 is recommended for use since other types (e.g., 74HC74) will introduce an excessive propagation delay, leading to data corruption.

A second complication results in the fact that some applications require separate transmit and receive frame synchronization. This requirement has resulted in the gating of the TDE and RCE signals with the select line separately, allowing the option of splitting the frame sync signals.

POWER SUPPLY

Since the CODEC board has been specifically designed to interface to the application development system, power may be taken from this: however, –5 V must be generated locally by using a 7660-V inverter.

JUMPER OPTIONS

The following is a list of the jumpers and their functions:

J1	Selects External Clock Generation (2.048 MHz)
J2	Selects External Frame Sync Generation
J3	Selects Synchronous Receive/Transmit Frame Sync
J4	Selects Synchronous Receive/Transmit Data Clocks
J5	Disables DSP Receive Frame Sync
J6	Selects Operating Mode of CODECs
J7	Disables DSP 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 8-kHz or 16-kHz Frame Sync Generation

The jumper setting for the example software included in this document are detailed in Figure 1.

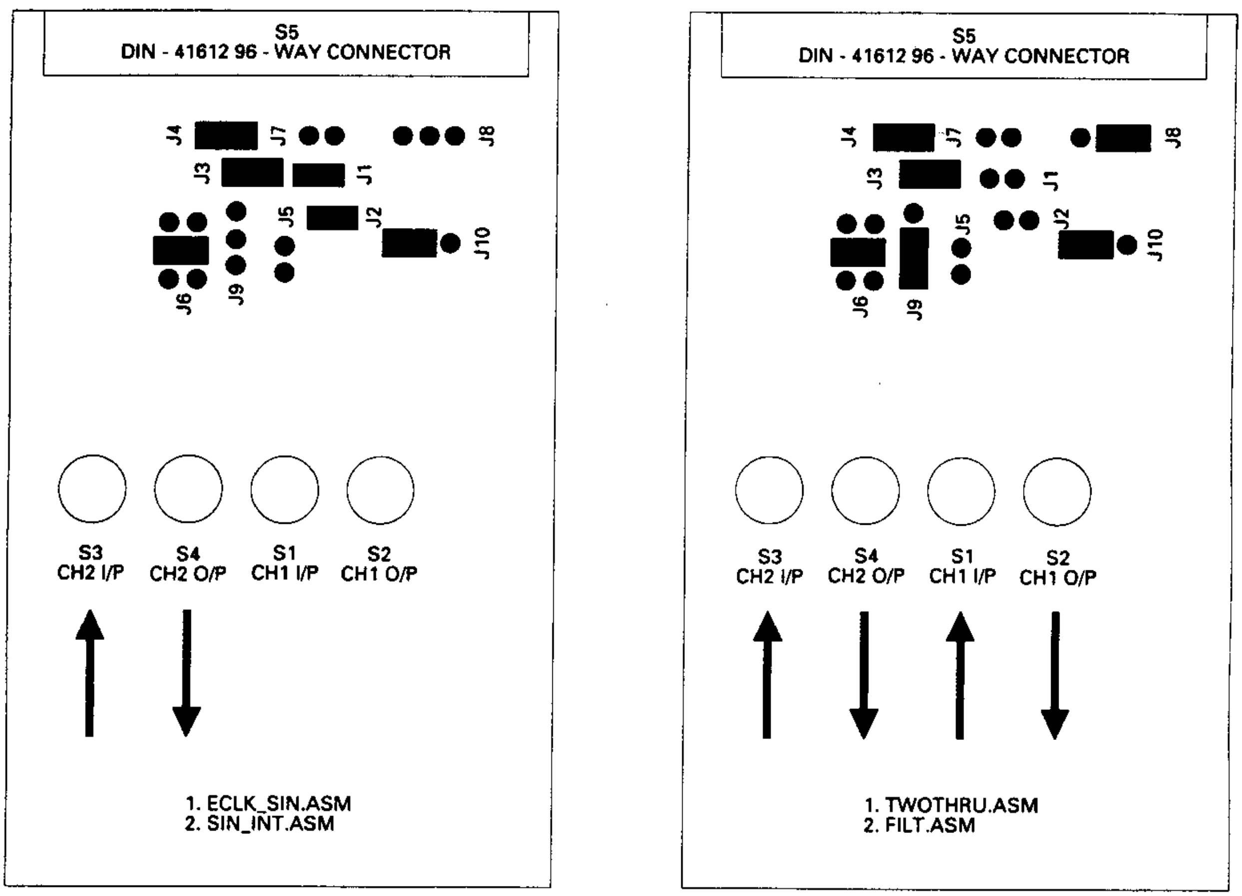
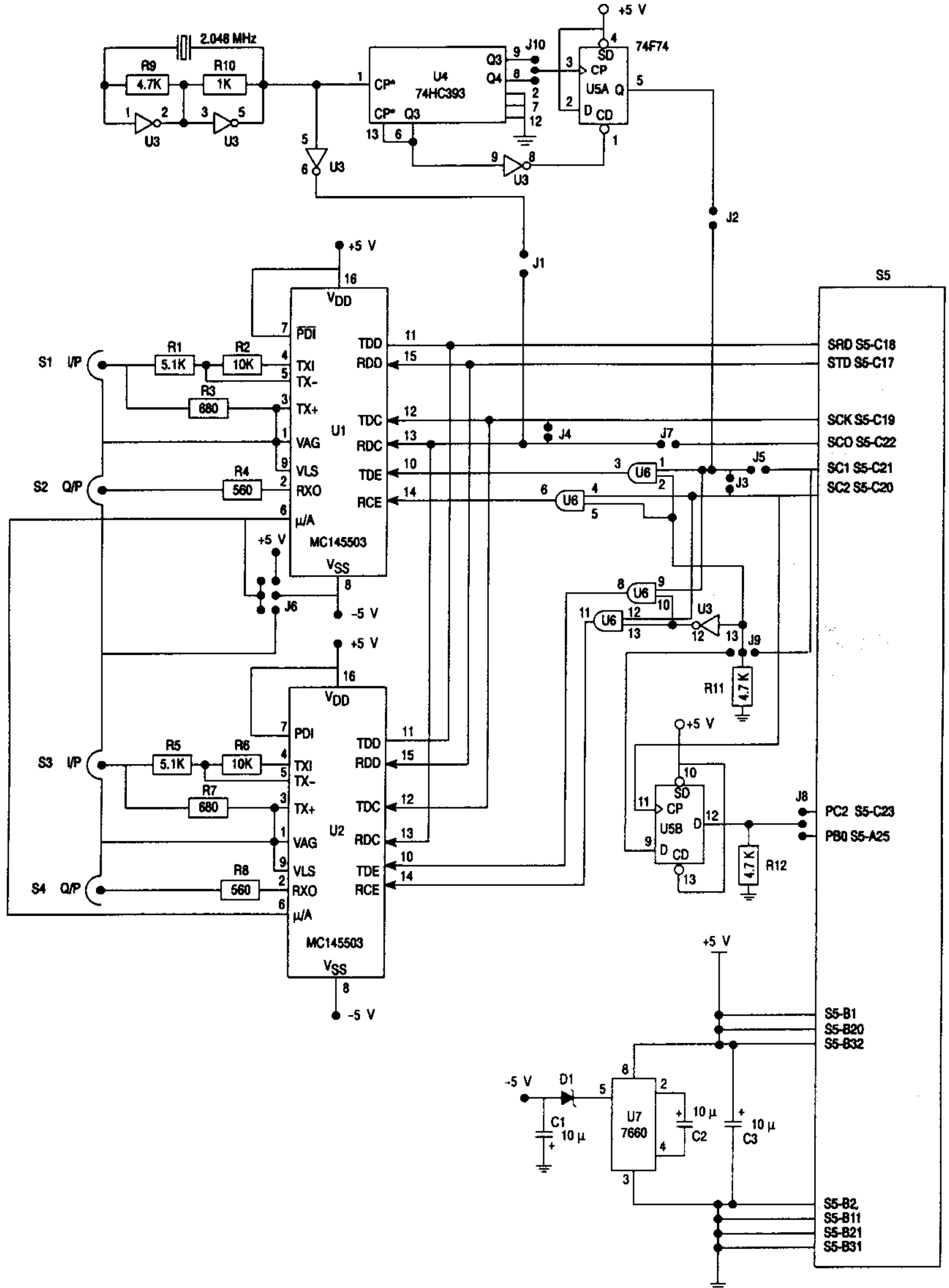


Figure 1. Configuration of the Twin CODEC Board for Example Software

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	10 kΩ
R2,6	5.1 kΩ
R3,7	680 Ω
R4,8	560 Ω
R9,11,12	4.7 kΩ
R10	1 kΩ
C1,2,3	10 µF tantalum
CD	0.1 µF (decouplers)
D1	Schottky Diode
X1	2.048 MHz 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     #0,x:M_CRA                      ; PSR=0 , WL=0 , DC4-0=$13 ,PM7-0=1
          movep     #$f200,x:M_CRB ; RIE=0,TIE=0, RE,TE = 1
          movep     #$1f8,x:M_PCC  ; set CC(8:3) as SSI pins
          andi      #$fe,mr                         ; 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 implemented around the following two software macros. 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 forms of FIR filter. They cover both initialization and execution of the filter.

The first macro, INIT_FIR, is passed various parameters indicating to the assembler which register to use for the filter, how many taps are in the filter, and what memory area to use; it then initializes 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
```

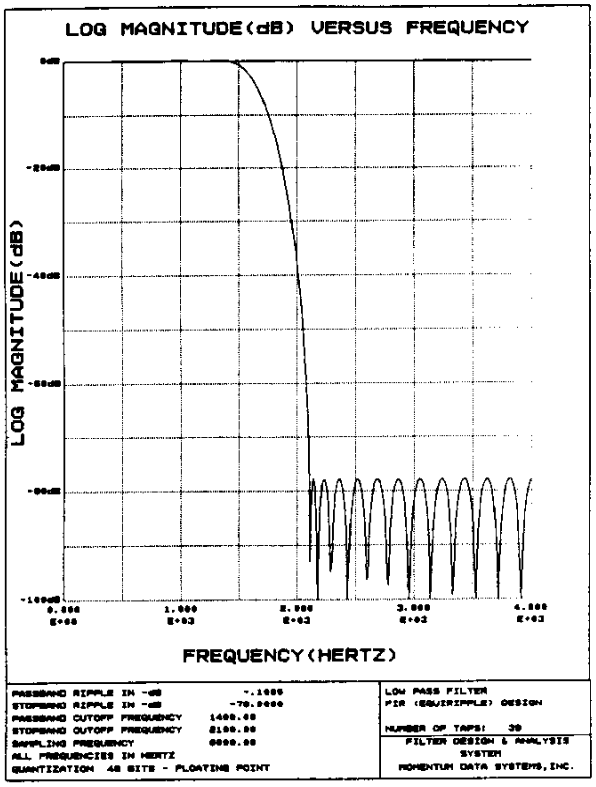
Filter 1

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 as follows:

Sample Rate	8 kHz
Filter Type	LPF
Upper Limit of Passband	1.4 kHz
Lower Limit of Stopband	2.1 kHz
Passband Ripple	-0.1 dB
Stopband Ripple	-78 dB
Number of Taps	39

This gives the filter of Figure D-1(a); this transfer function has been evaluated using extended floating-point arithmetic and is thus the closest achievable theoretically ideal filter. However, few DSPs 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 DSP56000, the word length is 24 bits; Figure D-1(b) is the realizable transfer function when the coefficients are quantized 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.



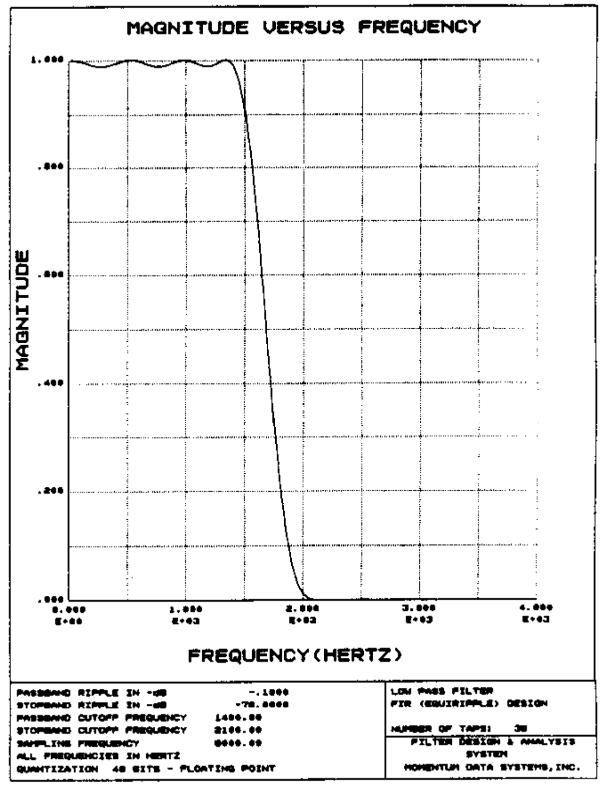


Figure D-1(a). Theoretical Log and Magnitude Plots Lowpass Filter

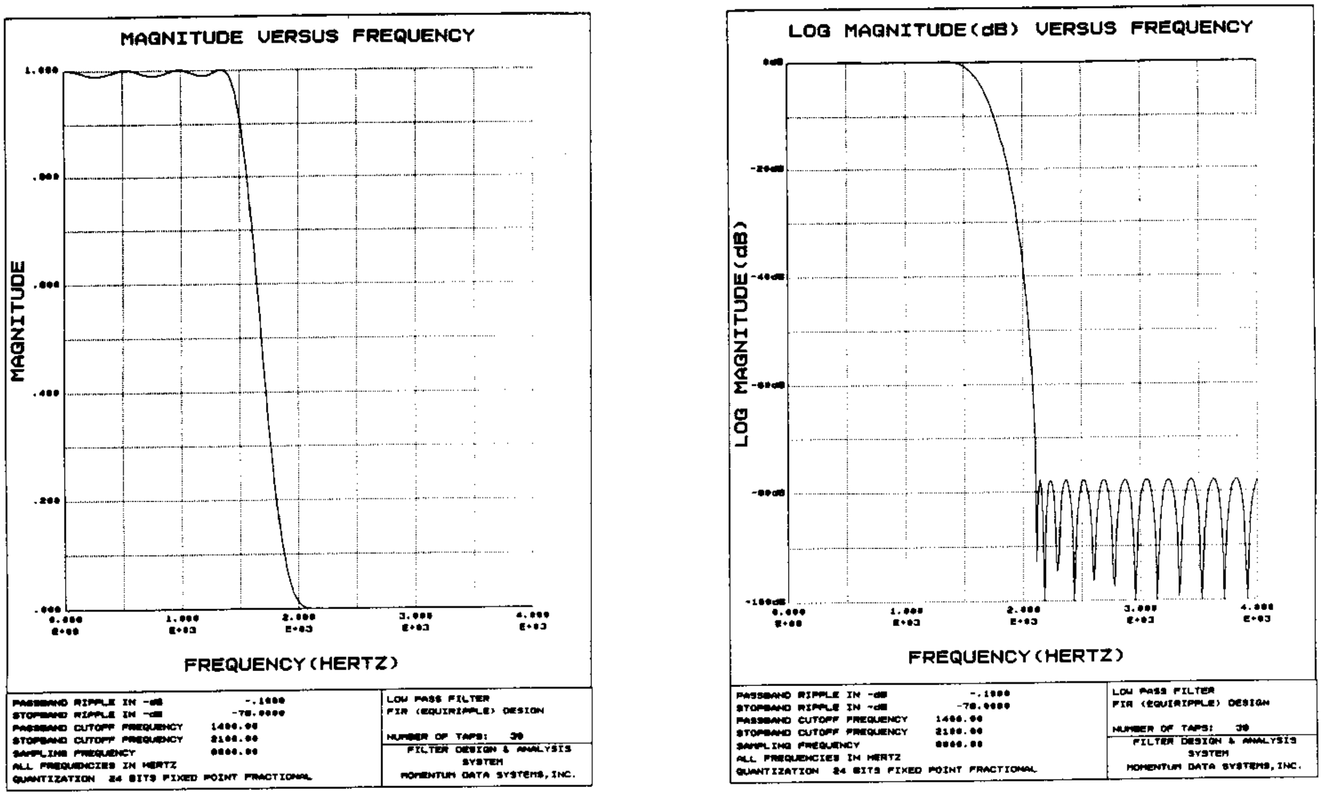


Figure D-1(b). Realizable Log and Magnitude Plots Lowpass Filter

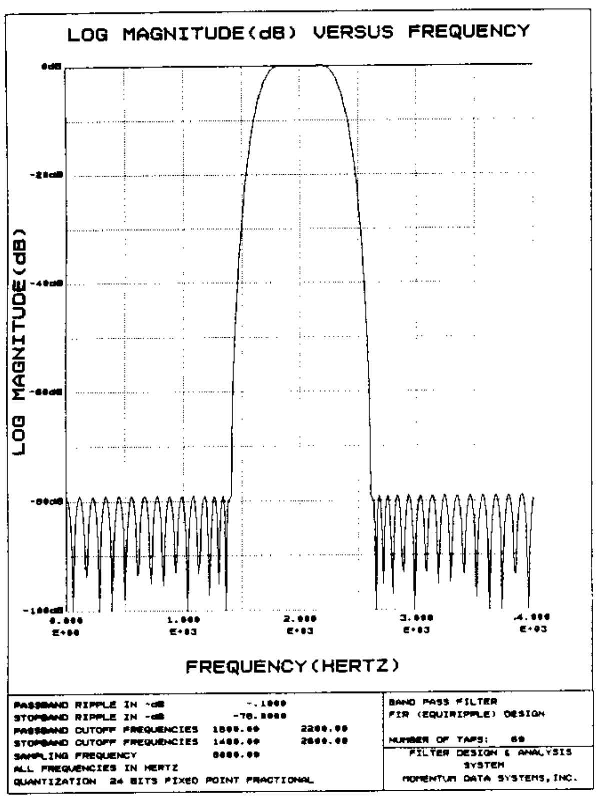
Filter 2

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 as follows:

Sample Rate	8 kHz
Filter Type	BPF
Upper Limit of Passband	2.2 kHz
Lower Limit of Passband	1.8 kHz
Upper Limit of Stopband	1.4 kHz
Lower Limit of Stopband	2.6 kHz
Passband Ripple	-0.1 dB
Stopband Ripple	-78 dB
Number of Taps	69

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

It should be noted that when using a CODEC, only approximately 78 dB of resolution is available. These filters were designed with that fact in mind; the DSP56000 will support filters with cutoffs of –144 dB. In this application, such a filter would be excessively powerful.



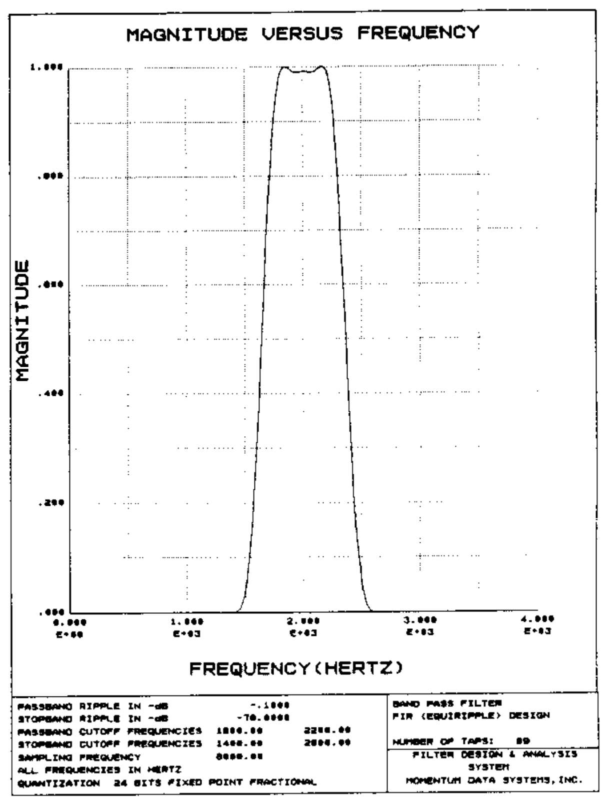


Figure D-2. Realizable Log and Magnitude Plots Bandpass Filter

APPENDIX E
LOG/LIN CONVERSION ROUTINES

```
;
; 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            al,b            ;save input
        lsl     a       #>_shift,x0     ;shift out sign bit, get shift constant
        lsr     a       #_atable,xl     ;shift in zero, get table base
        tfr     xl,a    al,xl           ;swap table base and offset
        mac     xl,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

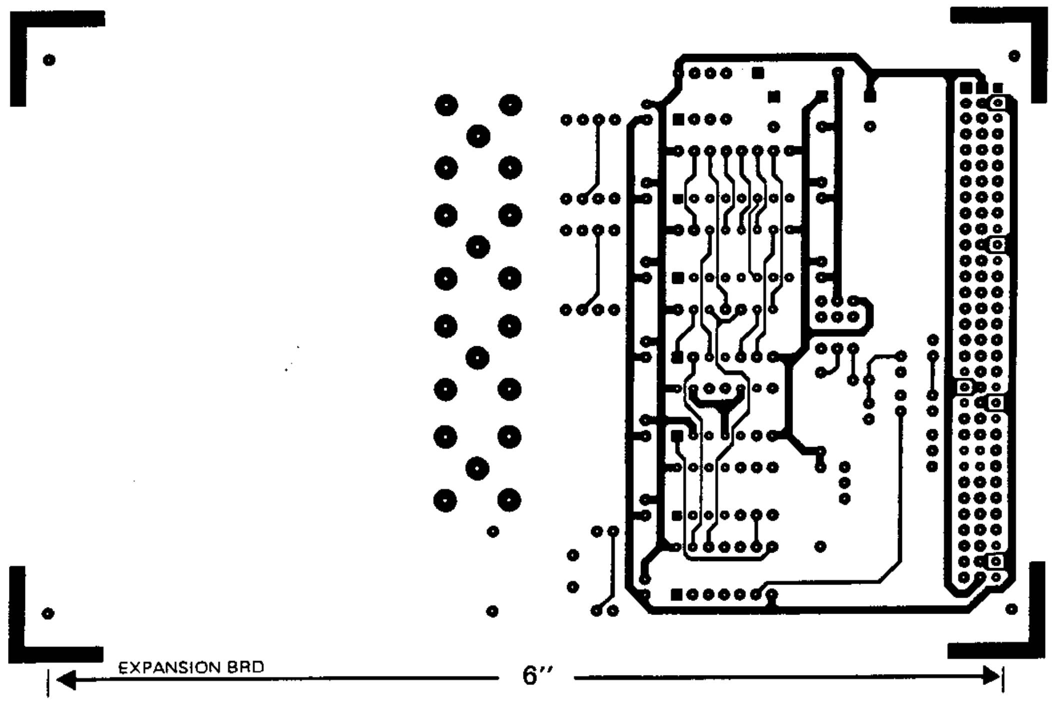


Figure F-1(a). PCB Artwork
Component Side

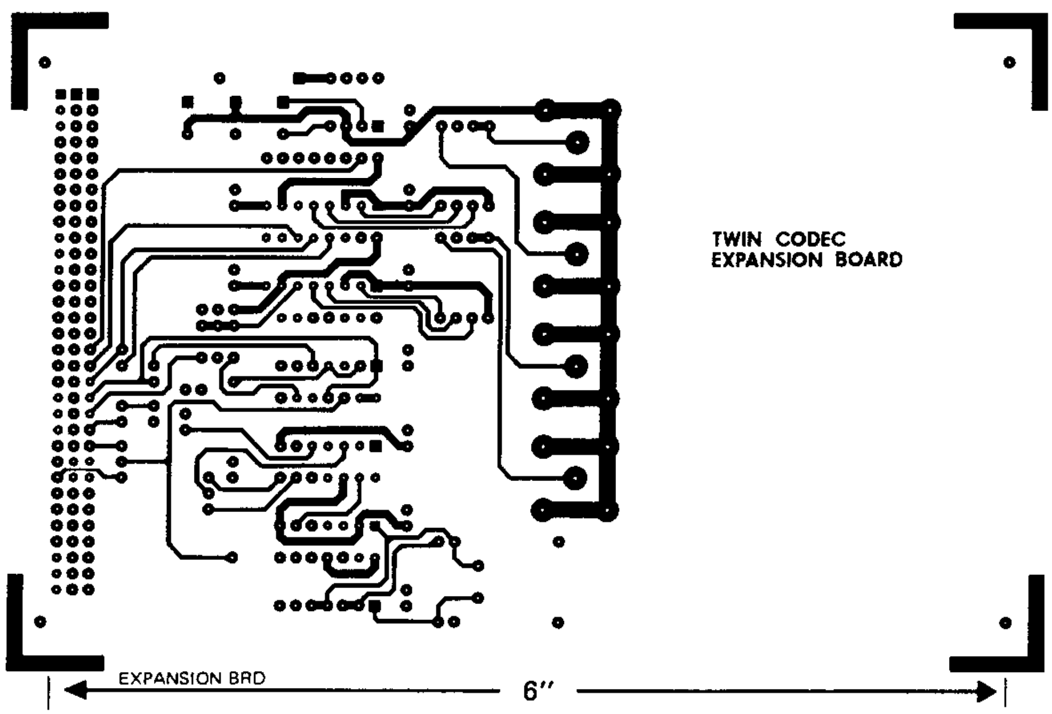


Figure F-1(b). PCB Artwork
Solder Side

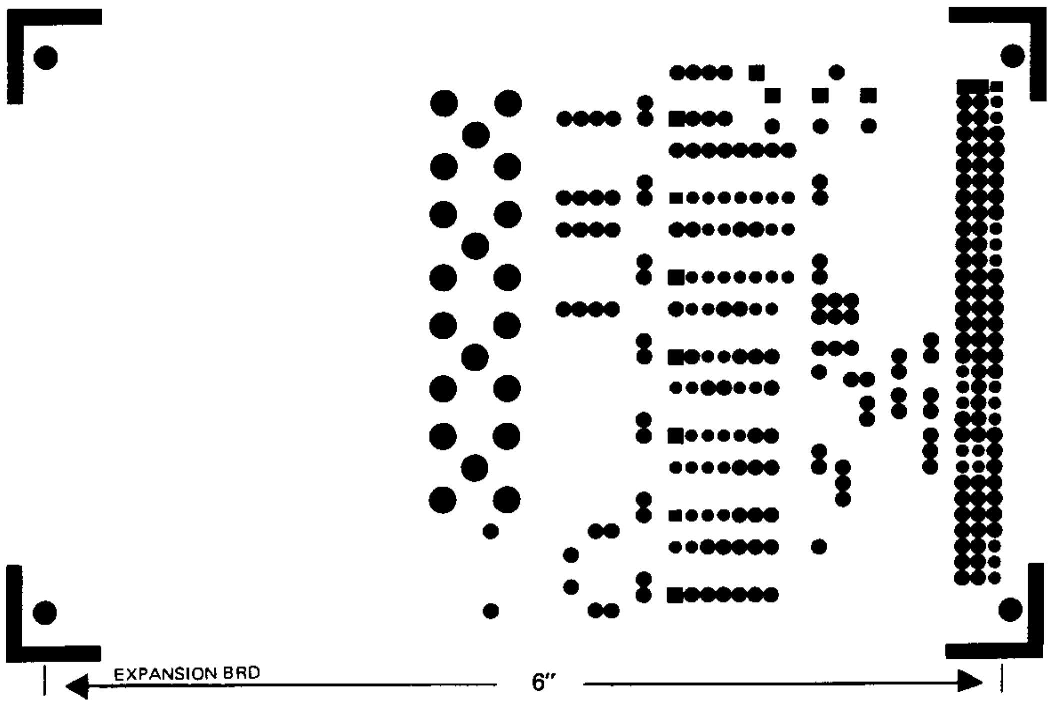


Figure F-1(c). Solder Resist Mask

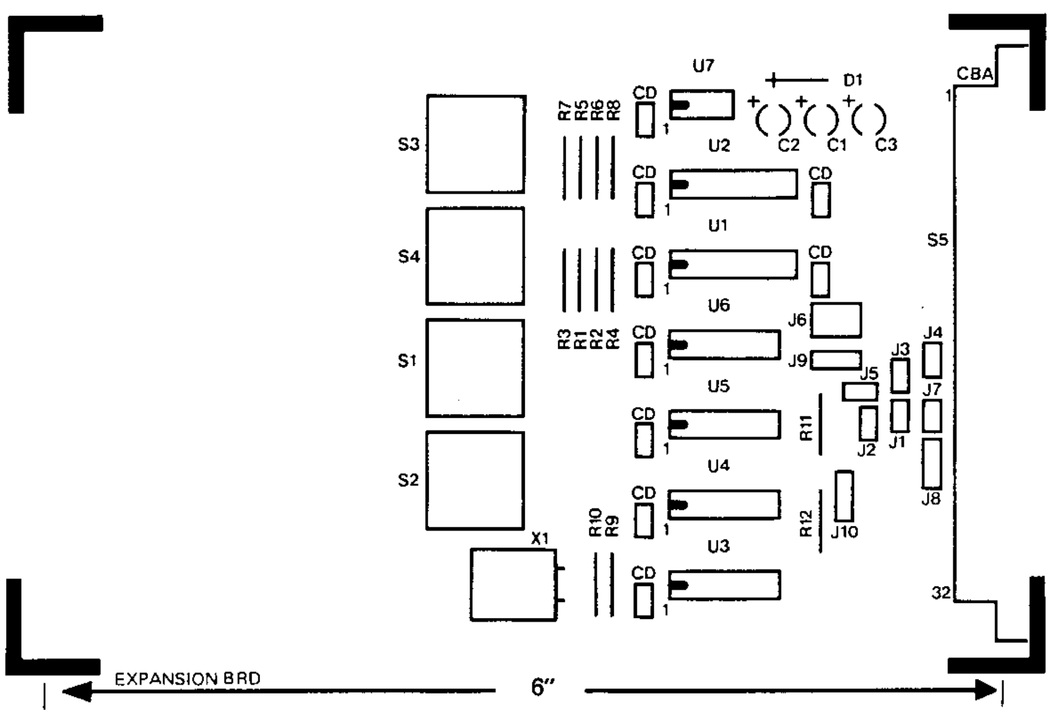


Figure F-1(d). Screen Printing

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

ATX30137-2 PRINTED IN USA 5/95 IMPERIAL LITHO 12938 3,000 DSP YGAVAA

APR12/D