MICROCHIP

AN540

Implementing IIR Digital Filters

INTRODUCTION

This application note describes the implementation of various digital filters using the PIC17C42, the first member of Microchip's 2nd generation 8-bit microcontrollers. The PIC17C42 is a very high speed 8-bit microcontroller with an instruction cycle time of 250ns (@ 16 MHz input clock). Even though PIC17C42 is an 8-bit device, it's high speed and efficient instruction set allows implementation of digital filters for practical applications. Traditionally digital filters are implemented using expensive Digital Signal Processors (DSPs). In a system the DSP is normally a slave processor being controlled by either an 8- or 16-bit microcontroller. Where sampling rates are not high (esp. in mechanical control systems), a single chip solution is possible using the PIC17C42.

This application note provides a few examples of implementing digital filters. Example code for 2nd order Infinite Impulse Response (IIR) filters is given. The following type of filters are implemented:

- Low Pass
- High Pass
- Band Pass
- Band Stop (notch) filter

This application note does not explain how to design a filter. Filter design theory is well established and is beyond the scope of this application note. It is assumed that a filter is designed according to the desired specifications. The desired digital filters may be designed using either standard techniques or using commonly available digital filter design software packages.

Finite Impulse Response (FIR) filters have many advantages over IIR filters, but are much more resource intensive (both in terms of execution time and RAM). On the other hand, IIR filters are quite attractive for implementing with the PIC17C42 resources. Especially where phase information is not so important, IIR filters are a good choice (FIR filters have a linear phase response). Of the various forms used for realizing digital filters (like, Direct form, Direct II form, Cascade form, Parallel, Lattice structure, etc.) the Direct II form is used in this application note. It is easy to understand and simple macros can be built using these structures.

THEORY OF OPERATION

Digital filters in most cases assume the following form of relationship between the output and input sequences.

$$y(n) = -\sum_{i=o}^{M} a_i y(n-i) + \sum_{j=o}^{N} b_j x(n-j)$$

The above equation basically states that the present output is a weighted sum of the past inputs and past outputs. In case of FIR filters, the weighted constants ai=0 and in case of IIR filters, at least one of the ai constant is non zero. In case of IIR, the above formula may be re written in terms of Z transform as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k Z^{-k}}{1 + \sum_{k=1}^{N} a_k Z^{-k}}$$

The above equation can further be rewritten in difference equation format as follows:

$$y(n) = -\sum_{i=1}^{M} a_i y(n-i) + \sum_{j=o}^{N} b_j x(n-j)$$

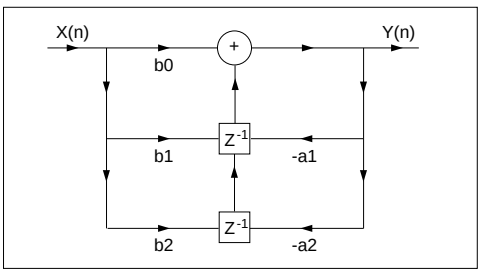
Realization of the above equation is called as the Direct Form II structure. For example, in case of a second order structure, M=N=2, gives the following difference equations :

$$d(n) = x(n) + a_1 d(n-1) + a_2 d(n-2) \quad (1)$$

$$y(n) = b_0 d(n) + b_1 d(n-1) + b_2 (d(n-2) \quad (2)$$

The above difference equations may be represented as shown in Figure 1.

FIGURE 1 - 2ND ORDER DIRECT FORM II STRUCTURE (TRANSPOSED)



The structure as shown in Figure 1 may be cascaded to attain a higher order filter. For example, if two stages are cascaded together, a 4th Order IIR Filter is obtained. This way, the output of the 1st stage becomes the input to the second stage. Multiple order filters are thus implemented by cascading a 2nd order filter structure as shown in Figure 1.

IMPLEMENTATION

A 4th order IIR Filter is implemented by cascading two structures shown in Figure 1. The output Y (output of each filter stage) is computed by direct implementation of Equations (1) & (2). Since each stage is similar algorithmically, it is implemented as a Macro using Microchip's Assembler/Linker for PIC17C42. This Macro (labelled **"BIQUAD"**) is called twice for implementing a 4th order filter. The output of the 1st stage is directly fed to the input of the second stage without any scaling.

Scaling is required depending on the particular application. The user can modify the code very easily without any penalty on speed. Also, saturation arithmetic is not used. Overflows can be avoided by limiting the input sequence amplitude. All numbers are assumed to be 16 bits in Q15 format (15 decimal points, MSB is sign bit). Thus the user must scale and sign extend the input sequence accordingly. For example, if the input is from a 12-bit A/D converter, the user must sign extend the 12-bit input if bit 11 is a one.

The BIQUAD macro is a generic macro and can be used for all IIR filters whether it is Low Pass, High Pass, Band Pass or Band Stop. A general purpose 16x16 multiplier routine is also provided. This routine is implemented as a straight line code for speed considerations.

The 4th order IIR filter implemented is a Low Pass Filter with the specifications as shown in Table 1.

TABLE 1 - FILTER CONSTANTS

	BAND1	BAND2
Lower Band Edge	0.0	600 Hz
Upper Band Edge	500 Hz	1 Khz
Nominal Gain	1.0	0.0
Nominal Ripple	0.01	0.05
Maximum Ripple	0.00906	0.04601
Ripple in dB	0.07830	-26.75
Sampling Frequency = 2 Khz		

The Low Pass Filter is designed using a digital filter design package (DFDP by Atlanta Signal Processors Inc.). The filter package produces filter constants of the structure shown in Table 1. Table 2 shows the filter co-efficients that are obtained for the above Low Pass filter specification.

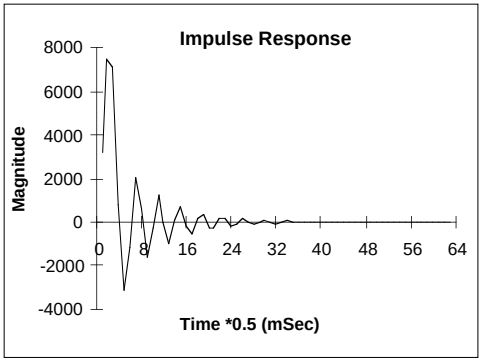
TABLE 2 - FILTER COEFFICIENTS

Co-efficients	a1	a2	b0	b1	b2
Stage 1	-0.133331	0.167145	0.285431	0.462921	0.285431
Stage 2	0.147827	0.765900	0.698273	0.499908	0.698273

The above filter co-efficients (5 per stage) are quantized to Q15 format (i.e they are multiplied by 32768) and saved in program memory (starting at label **"_coeff_lpass"**). The constants for both the stages are read into data memory using TLRD and TABLRD instructions in the Initialization Routine (labelled **"initFilter"**). The user may read the coefficients of only one stage at a time and save some RAM at the expense of speed.

The sample 4th order Low Pass IIR Filter is tested by analyzing the impulse response of the filter. An impulse signal is fed as input to the filter. This is simulated by forcing the input to the filter by a large quantity (say 7F00h) on the first input sample, and the all zeros from the 2nd sample onwards. The output sequence is the filter's impulse response and is captured into the PICMASTER's (Microchip's Universal In-Circuit Emulator) real time trace buffer. This captured data from PICMASTER is saved to file and analyzed. Analysis was done using MathCad for Windows and DSP Analysis program from Burr-Brown (DSPLAY). The Fourier Transform of this Impulse response of the filter should display the Filter's frequency response, in this case being a Low Pass type. The plots of the impulse response and the frequency response are shown in figures below.

FIGURE 2 - IMPULSE RESPONSE CAPTURED FROM PIC-MASTER



DSPLAY is a trademark of Burr-Brown
DFDP is a trademark of Atlanta Signal Processing Inc.
Windows is a trademark of Microsoft Corporation
MathCad is a registered trademark of MathSoft, Inc.

FIGURE 3 - SPECTRUM COMPUTED FROM IMPULSE RESPONSE

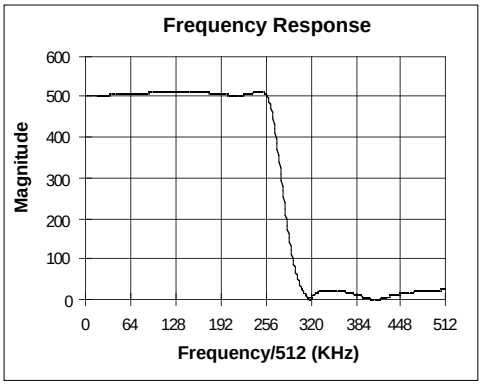
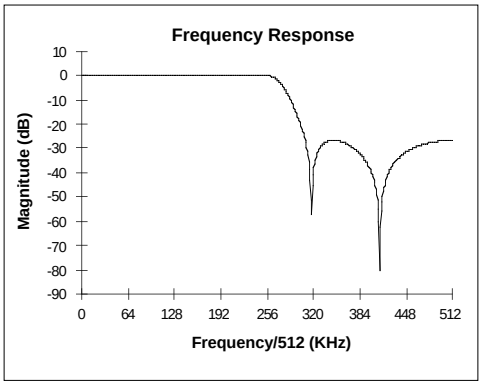


FIGURE 4 - LOG MAGNITUDE SPECTRUM OF IMPULSE RESPONSE



4

PERFORMANCE

The resource requirements for filter implementations using PIC17C42 is given below. These numbers can be used to determine whether a higher order filter can be executed in real time. The same information may be used to determine the highest sampling rate possible.

FILTER APPLICATIONS

Digital filters find applications in many areas especially involving processing of real world signals. In some applications like ABS systems in an automobile, digital filtering becomes a must. In this case elimination of noise (especially glitches and false readings of sensors) is very critical and thus the requirement of digital signal processing.

Digital filters are also needed in Process Control where notch filters and low pass filters are desired because the signals from sensors are transmitted over long lines, especially in a very noisy environment. In this case typically a notch filter (centering 50Hz or 60Hz) is used. In case of eliminating background noise, a band stop filter (e.g. 40Hz to 120Hz) is used. The sample code given in this application note can be used to design a feedback control system's digital compensator. For example, a typical DC Motor's digital compensator (like dead beat compensator) is of second order and has the same filter structure as is implemented in this application note.

TABLE 3 - RESOURCE REQUIREMENTS

Timing (Cycles)†	#of Filter Stages*775 + 16
Program Memory (locations)†	#of Filter Stages*68 + 290
RAM (File Registers)	#of Filter Stages*16+16

†: The above numbers do not include the initialization routine.

Author: Amar Palacherla
Logic Products Division

TABLE 4 - RESOURCE REQUIREMENTS

Filter Order	Cycles	Real Time (@ 16 MHz)	Maximum Sampling	Program Memory†	RAM
2	791	197.75 uSec	5.05 Khz	358	32
4	1566	391.5 uSec	2.55 Khz	426	48
6	2341	585.25 uSec	1.7 Khz	494	64
8	3116	779.0 uSec	1.28 Khz	562	80
10	3891	972.75 uSec	1.0 Khz	630	96

†: The above numbers do not include the initialization routine.

APPENDIX A: IIR.LST

```
MPASM B0.54                                                          PAGE  1
Digital IIR Filter Using PIC17C42



                        TITLE   “Digital IIR Filter Using PIC17C42”
                        LIST     P=17C42, C=120, T=ON, R=DEC, N=0

                    ;
                    include “17c42.h”



                    ;
                    include “17c42.mac”

                    ;
                    include “17c42iir.mac”
                    ;*****************************************************************
                    ;                       PIC17C42 MACRO
                    ;
                    ;               Macro For A Bi-Quad IIR Filter
                    ;       2nd order Direct Form (Transposed) Type
                    ;
                    ; Filter co-efficients B0 & B2 are assumed equal
                    ;
                    ; The difference equations for each cascade section is given by :
                    ;         Y(n) = B0*D(n) + B1*D(n-1) + B2*D(n-2)
                    ;         D(n) = X(n) - A1*D(n-1) -  A2*D(n-2)
                    ;    where X(n) = input sample, Y(n) = output of filter
                    ;    and A1, A2, B0, B1, B2 are the Filter Co-efficients
                    ;
                    ;  The above difference equations are only for 1 section of a
                    ; 2nd order Direct_Form II Filter structure (IIR)
                    ;
                    ;   NOTE :
                    ;            It is possible to design the above structures
                    ;       such that the co-efficients B0 = B2. If this is the
                    ;       case,
                    ;               Y(n) = B0*[D(n) + D(n-2)] = B2*[D(n) + D(n-2)]
                    ;       This way, one multiplication can be avoided
                    ;
                    ;  If a 4th order filter is to be implemented, the output of
                    ; the 1st structure should be input to the 2nd cascade section
                    ;
                    ;       Timing (WORST CASE) :
                    ;                               59+4*179 = 775 Cycles
                    ;                               (194 uS @ 16 Mhz)
                    ;       Program Memory  :
                    ;                               63 locations
                    ;
                    ;*****************************************************************
                    ;       The sample filters are desined so that B0=B2
                    ;       This saves 1 multiplication
                    ;
0001                B0_EQUALS_B2    equ     TRUE
                    ;
                    ;*****************************************************************
                    ;  Parameters to BIQUAD Macro :
                    ;               Filter Constants A1, A2, B0, B1, B2
                    ;               & D(n), D(n-1), D(n-2), filter stage #
                    ;
                    BIQUAD  MACRO   Ax1,Ax2,Bx0,Bx1,Dn,Dn_1,Dn_2,stage

                    ;
                    ; Compute Ax2*D(n-2)
```

```
;
MOVFP16  Dn_2,AARG              ; D(n-2) = multiplier
MOVFP16  Ax2,BARG               ; A2 = multiplicand
call     DblMult                ; (ACCd,ACCc) = A2*D(n-2)
;
; Add product to output of 1st section
; Save result in 32 bit Accumulator
;
ADD32    DPX,ACC
;
;  Compute A1*D(n-1)
;
MOVFP16  Dn_1,AARG              ; AARG = D(n-2) = multiplier
MOVFP16  Ax1,BARG               ; BARG = A2 = multiplicand
call     DblMult                ; (ACCd,ACCc) = A1*D(n-1)
;
; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
;       multiplications already done, so simply perform a 32 bit add
;       of previously obtained multiplication results
;
ADD32   DPX,ACC                 ; ACC = A1*D(n-1)+A2*D(n-2)+(output of 1st
;
;
; save the upper 16 bits  of D(n) from the 32 bit accumulator
; left shift the result by 1, to adjust the decimal point after
; a Q15*Q15 multiplication
;
rlcf     ACC+B1,w
rlcf     ACC+B2,w
movwf    Dn
rlcf     ACC+B3,w               ; decimal adjust ( mult by 2)
movwf    Dn+B1
;
; Compute B2 * [D(n) + D(n-2)]
;
if B0_EQUALS_B2
ADD16ACC Dn_2,Dn,AARG           ; AARG = Dn + D(n-2) = multiplier
MOVFP16  Bx0,BARG               ; BARG = A2 = multiplicand
call     DblMult                ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
MOVPF32  DPX,ACC

else
MOVFP16  Bx0,BARG
MOVFP16  Dn,AARG
call     DblMult                ; B0*D(n)
MOVPF32  DPX,ACC

MOVFP16  Bx2,BARG
MOVFP16  Dn_2,AARG
call     DblMult                ; B2*D(n-2)
ADD32    DPX,ACC
endif
;
; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
; This way in the next iteration D(n-2) is equal to the present D(n-1)
;
movfp    Dn_1,AARG+B0
movpf    AARG+B0,Dn_2           ; Shift down D(n-1)
movfp    Dn_1+B1,AARG+B1
movpf    AARG+B1,Dn_2+B1        ; AARG = D(n-1) = multiplier
MOVFP16  Bx1,BARG               ; BARG = B1 = multiplicand
call     DblMult                ; (ACCd,ACCc) = B1*D(n-1)
;
; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
;                  = B1*D(n-1) + B0*[D(n) + D(n-2)]
; Since all multiplications are already done, simply perform a
; 32 bit addition
;
ADD32   DPX,ACC             ; ACC = B1*D(n-1) + B2*D(n-2) + B0*D(n)
```

```
;
; Shift down D(n) to D(n-1) so that in the next iteration, the new
; D(n-1) is the present D(n)
;
MOV16    Dn,Dn_1            ; Shift down D(n) to D(n-1)
;
         ENDM


;
;*****************************************************************
;               Second Order Direct Form IIR Filter
;
;
; In the code given below, a 4th order IIR Elliptic Lowpass Filter
; is implemented. Other order filters may be implemented by
; taking the following example code as a basis.
;
;       The specifications of the filter are :
;
;       Sampling Frequency = 2.0 Khz
;
;       Filter Type = 4th Order Elliptic Lowpass Filter
;
;                                Band1           Band2
;       Lower Band Edge          0.0             600 Hz
;       Upper Band Edge          500 Hz          1 Khz
;       Nominal Gain             1.0             0.0
;       Nominal Ripple           0.01            0.05
;       Maximum Ripple           0.00906         0.04601
;       Ripple in dB             0.07830         -26.75
;
;  The Filter Co-efficients for the above specifications
; of the filter are computed as follows :
;
;      1st Section :
;                        A11 = -0.133331
;                        A12 = 0.167145
;                        B10 = 0.285431
;                        B11 = 0.462921
;                        B12 = 0.285431
;      2nd Section
;                        A21 = 0.147827
;                        A22 = 0.765900
;                        B20 = 0.698273
;                        B21 = 0.499908
;                        B22 = 0.698273
;
;
;       Performance (WORST Case):
;
;                        Cycles    = #of Filter Stages*775 + 16
;                                  = 2*775+16 = 1566 Cycles
;                                        ( 391 uSec)
;                                per each sample. Initialization
;                                time after reset is not counted
;                                Timing measured with B0_EQUALS_B2
;                                set to TRUE (see BIQUAD Macro for
;                                explanation
;
;                        Program Memory :
;                                        = 16+ # of FilterStages * (BIQUAD
;                                                + filter co-efficients)
;                                                + multiplier
;                                        = 16+2*(63+5)+274 = 421 locations
;                                          (excluding initialization)
;
;                        RAM usage = 48 file registers
;                        RAM usage/each additional stage = 16 file regs
;
```

```
                        ;
                        ;                       This time is less than 2 Khz (500 uSec),
                        ;                       which means real time filtering is possible
                        ;
                        ;*****************************************************************

                        ;*****************************************************************
                        ;
                        ;
                                CBLOCK  0
0000 0004               B0,B1,B2,B3
                                ENDC
                        ;*****************************************************************
                        ;
                                CBLOCK  0x18
0018 0004               DPX,DPX1,DPX2,DPX3          ; arithmetic accumulator
001C 0004               AARG,AARG1,BARG,BARG1       ; multiply arguments
                                ENDC
                        ;
                                CBLOCK
0020 0002               Dn1, Dn1_Hi
0022 0002               Dn1_1, Dn1_1_Hi
0024 0002               Dn1_2, Dn1_2_Hi
0026 0000
0026 0002               Dn2, Dn2_Hi
0028 0002               Dn2_1, Dn2_1_Hi
002A 0002               Dn2_2, Dn2_2_Hi
                                ENDC

                                CBLOCK
002C 0002               A11, A11_Hi
002E 0002               A12, A12_Hi
0030 0002               B10, B10_Hi
0032 0002               B11, B11_Hi                 ; 1st Section Filter Co-efficients
0034 0002               B12, B12_Hi
0036 0000
0036 0002               A21, A21_Hi
0038 0002               A22, A22_Hi
003A 0002               B20, B20_Hi
003C 0002               B21, B21_Hi                 ; 2nd Section Filter Co-efficients
003E 0002               B22, B22_Hi
                                ENDC

                                CBLOCK
0040 0002               X, X1                       ; 16 bits of input stream
0042 0002               Y, Y1                       ; 16 bits of filter output
0044 0000
0044 0004               ACC, ACC1, ACC2, ACC3       ; 32 bit accumulator for computations
                                ENDC
                        ;
0002                    FltStage        .set     2
000A                    NumCoeff        equ     (5*FltStage) ; 5 Co-eff per stage
                        ;
0001                    LPASS   .set    TRUE
0000                    HPASS   .set    FALSE
0000                    BPASS   .set    FALSE               ; select the desired filter type
0000                    BSTOP   .set    FALSE
                        ;
0001                    SIGNED  equ     TRUE                ; Set This To 'TRUE' for signed multi
                        ;                                   ; and 'FALSE' for unsigned.
                        ;
                        ;*****************************************************************
                        ;               Test Program For Low Pass Filter
                        ;*****************************************************************

                               ORG     0x0000
                        ;
                        start
0000 E00D                       call    initFilter
```

```
0001 B000                       movlw   0x00
0002 0140                       movwf   X
0003 B07F                       movlw   0x7f                    ; set initial Xn = X(0) = 0x7f00
0004 0141                       movwf   X+B1                    ; test for impulse response
                        ;
                        NextPoint
0005 E022                       call    IIR_Filter
0006 A442                       tlwt     _LOW,Y
                        tracePoint
0007 AE43                       tablwt   _HIGH,0,Y+B1
0008 0000                       nop
0009 2940                       clrf     X                      ; set X(n) = 0 , n <> 0
000A 2941                       clrf     X+B1                   ; for simulating an Impulse
000B C005                       goto     NextPoint
                        ;
000C C00C               self    goto     self
                        ;
                        ;*****************************************************************
                        ;

                        initFilter

                        ;
                        ;   At first read the Filter Co-efficients from Prog. Mem to Data RAM
                        ;
                                if      LPASS
000D B0C1                               movlw    _coeff_lpass
000E 010D                               movwf    tblptrl
000F B001                               movlw    page    _coeff_lpass
0010 010E                               movwf    tblptrh
                                endif
                                if     HPASS
                                        movlw    _coeff_hpass
                                        movwf    tblptrl
                                        movlw    page    _coeff_hpass
                                        movwf    tblptrh
                                endif
                                if     BPASS
                                        movlw    _coeff_bpass
                                        movwf    tblptrl
                                        movlw    page    _coeff_bpass
                                        movwf    tblptrh
                                endif
                                if     BSTOP
                                        movlw    _coeff_bstop
                                        movwf    tblptrl
                                        movlw    page    _coeff_bstop
                                        movwf    tblptrh
                                endif

                        ;
0011 B02C                       movlw    A11
0012 0101                       movwf     fsr0
0013 8404                       bsf      _fs0
0014 8D04                       bcf      _fs1                   ; auto increment
                        ;
                        ; Read Filter Co-efficients from Program Memory
                        ;
0015 B00A                       movlw    NumCoeff
0016 A92C                       tablrd   _LOW,_INC,A11          ; garbage
                        NextCoeff
0017 A000                       tlrd     _LOW,indf0
0018 AB00                       tablrd   _HIGH,_INC,indf0
0019 1700                       decfsz   wreg
001A C017                       goto     NextCoeff
                        ;
                        ; Initilize “Dn”s to zero
                        ;
001B B020                       movlw    Dn1
```

```
001C 0101                       movwf    fsr0
001D B00C                       movlw    6*FltStage
                        NextClr
001E 2900                       clrf     indf0
001F 1700                       decfsz   wreg
0020 C01E                       goto     NextClr
                        ;
0021 0002                       return
                        ;
                        ;*****************************************************************
                        ;                       1st Cascade Section
                        ;*****************************************************************
                        ;

                        IIR_Filter

                        ;
                        ; Compute  D(n) = X(n) + A1*D(n-1) + A2*D(n-2)
                        ;          Since the filter constants are computated in Q15 format,
                        ;          X(n) must be multiplied by 2**15 and then added to the
                        ;          other terms.
                        ;
                        ; Move Input to accumulator after proper scaling
                        ;
0022 8804                       bcf      _carry
0023 1941                       rrcf     X+B1
0024 1940                       rrcf     X
0025 2900                       clrf     wreg                    ; Scale the input X
0026 1900                       rrcf     wreg
0027 0145                       movwf    ACC+B1
0028 6040                       movfp    X,wreg
0029 0146                       movwf    ACC+B2
002A 6041                       movfp    X+B1,wreg
002B 0147                       movwf    ACC+B3                  ; ACC = scaled input : X*(2**15)
                        ;
                        ; 1st Biquad filter section
                        ;
                        BIQUAD  A11,A12,B10,B11,Dn1,Dn1_1,Dn1_2,1

                        ;
                        ; Compute A12*D(n-2)
                        ;


002C 7C24                       MOVFP    Dn1_2+B0,AARG+B0        ; move Dn1_2(B0) to AARG(B0)
002D 7D25                       MOVFP    Dn1_2+B1,AARG+B1        ; move Dn1_2(B1) to AARG(B1)



002E 7E2E                       MOVFP    A12+B0,BARG+B0          ; move A12(B0) to BARG(B0)
002F 7F2F                       MOVFP    A12+B1,BARG+B1          ; move A12(B1) to BARG(B1)

0030 E0AF                       call     DblMult                 ; (ACCd,ACCc) = A2*D(n-2)
                        ;
                        ; Add product to output of 1st section
                        ; Save result in 32 bit Accumulator
                        ;


0031 6018                       MOVFP    DPX+B0,WREG             ; get lowest byte of DPX into w
0032 0F44                       ADDWF    ACC+B0                  ; add lowest byte of ACC, save in ACC(B0)
0033 6019                       MOVFP    DPX+B1,WREG             ; get 2nd byte of DPX into w
0034 1145                       ADDWFC   ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0035 601A                       MOVFP    DPX+B2,WREG             ; get 3rd byte of DPX into w
0036 1146                       ADDWFC   ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
```

```
0037 601B                   MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
0038 1147                   ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)

                    ;
                    ;  Compute A1*D(n-1)
                    ;


0039 7C22                   MOVFP   Dn1_1+B0,AARG+B0        ; move Dn1_1(B0) to AARG(B0)
003A 7D23                   MOVFP   Dn1_1+B1,AARG+B1        ; move Dn1_1(B1) to AARG(B1)



003B 7E2C                   MOVFP   A11+B0,BARG+B0          ; move A11(B0) to BARG(B0)
003C 7F2D                   MOVFP   A11+B1,BARG+B1          ; move A11(B1) to BARG(B1)

003D E0AF                   call    DblMult                 ; (ACCd,ACCc) = A1*D(n-1)
                    ;
                    ; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
                    ;       multiplications already done, so simply perform a 32 bit add
                    ;       of previously obtained multiplication results
                    ;


003E 6018                   MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
003F 0F44                   ADDWF   ACC+B0                  ; add lowest byte of ACC,save in ACC(B0)
0040 6019                   MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
0041 1145                   ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0042 601A                   MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
0043 1146                   ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
0044 601B                   MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
0045 1147                   ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)

                    ;
                    ;
                    ; save the upper 16 bits  of D(n) from the 32 bit accumulator
                    ; left shift the result by 1, to adjust the decimal point after
                    ; a Q15*Q15 multiplication
                    ;
0046 1A45                   rlcf     ACC+B1,w
0047 1A46                   rlcf     ACC+B2,w
0048 0120                   movwf    Dn1
0049 1A47                   rlcf     ACC+B3,w               ; decimal adjust ( mult by 2)
004A 0121                   movwf    Dn1+B1
                    ;
                    ; Compute B2 * [D(n) + D(n-2)]
                    ;
                            if B0_EQUALS_B2



004B 6024                   movfp   Dn1_2+B0,wreg
004C 0E20                   addwf   Dn1+B0,w
004D 011C                   movwf   AARG+B0
004E 6025                   movfp   Dn1_2+B1,wreg
004F 1021                   addwfc  Dn1+B1,w
0050 011D                   movwf   AARG+B1
0051 7E30                   MOVFP   B10+B0,BARG+B0          ; move B10(B0) to BARG(B0)
0052 7F31                   MOVFP   B10+B1,BARG+B1          ; move B10(B1) to BARG(B1)
0053 E0AF                   call    DblMult                 ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
0054 5844                   MOVPF   DPX+B0,ACC+B0           ; move DPX(B0) to ACC(B0)
0055 5945                   MOVPF   DPX+B1,ACC+B1           ; move DPX(B1) to ACC(B1)
0056 5A46                   MOVPF   DPX+B2,ACC+B2           ; move DPX(B2) to ACC(B2)
0057 5B47                   MOVPF   DPX+B3,ACC+B3           ; move DPX(B3) to ACC(B3)
```

```
                        else
                                MOVFP16 B10,BARG
                                MOVFP16 Dn1,AARG
                                call    DblMult         ; B0*D(n)
                                MOVPF32 DPX,ACC
                                MOVFP16 Bx2,BARG
                                MOVFP16 Dn1_2,AARG
                                call    DblMult         ; B2*D(n-2)
                                ADD32   DPX,ACC
                         endif
                ;
                ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
                ; This way in the next iteration D(n-2) is equal to the present D(n-1)
                ;
0058 7C22               movfp   Dn1_1,AARG+B0
0059 5C24               movpf   AARG+B0,Dn1_2           ; Shift down D(n-1)
005A 7D23               movfp   Dn1_1+B1,AARG+B1
005B 5D25               movpf   AARG+B1,Dn1_2+B1        ; AARG = D(n-1) = multiplier
005C 7E32               MOVFP   B11+B0,BARG+B0          ; move B11(B0) to BARG(B0)
005D 7F33               MOVFP   B11+B1,BARG+B1          ; move B11(B1) to BARG(B1)
005E E0AF               call    DblMult                 ; (ACCd,ACCc) = B1*D(n-1)
                ;
                ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                ;                   = B1*D(n-1) + B0*[D(n) + D(n-2)]
                ; Since all multiplications are already done, simply perform a
                ; 32 bit addition
                ;
005F 6018               MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
0060 0F44               ADDWF   ACC+B0                  ; add lowest byte of ACC, save in ACC(B0)
0061 6019               MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
0062 1145               ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0063 601A               MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
0064 1146               ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
0065 601B               MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
0066 1147               ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)
                         ;
                ; Shift down D(n) to D(n-1) so that in the next iteration, the new
                ; D(n-1) is the present D(n)
                ;
0067 6020               MOVFP   Dn1+B0,WREG             ; get byte of Dn1 into w
0068 0122               MOVWF   Dn1_1+B0                ; move to Dn1_1(B0)
0069 6021               MOVFP   Dn1+B1,WREG             ; get byte of Dn1 into w
006A 0123               MOVWF   Dn1_1+B1                ; move to Dn1_1(B1)
                ;
                ;
                ; 2nd Biquad filter section
                ;
                BIQUAD  A21,A22,B20,B21,Dn2,Dn2_1,Dn2_2,2
                ;
                ; Compute A22*D(n-2)
                ;
006B 7C2A               MOVFP   Dn2_2+B0,AARG+B0        ; move Dn2_2(B0) to AARG(B0)
006C 7D2B               MOVFP   Dn2_2+B1,AARG+B1        ; move Dn2_2(B1) to AARG(B1) 006D 7E38
                        MOVFP   A22+B0,BARG+B0          ; move A22(B0) to BARG(B0)
006E 7F39               MOVFP   A22+B1,BARG+B1          ; move A22(B1) to BARG(B1)
006F E0AF               call    DblMult                 ; (ACCd,ACCc) = A2*D(n-2)
                ;
                ; Add product to output of 1st section
                ; Save result in 32 bit Accumulator
                ;


0070 6018               MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
0071 0F44               ADDWF   ACC+B0                  ; add lowest byte of ACC, save in ACC(B0)
0072 6019               MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
0073 1145               ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0074 601A               MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
0075 1146               ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
0076 601B               MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
```

```
0077 1147                   ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)

                    ;
                    ;  Compute A1*D(n-1)
                    ;


0078 7C28                   MOVFP   Dn2_1+B0,AARG+B0        ; move Dn2_1(B0) to AARG(B0)
0079 7D29                   MOVFP   Dn2_1+B1,AARG+B1        ; move Dn2_1(B1) to AARG(B1)



007A 7E36                   MOVFP   A21+B0,BARG+B0          ; move A21(B0) to BARG(B0)
007B 7F37                   MOVFP   A21+B1,BARG+B1          ; move A21(B1) to BARG(B1)

007C E0AF                   call    DblMult                 ; (ACCd,ACCc) = A1*D(n-1)
                    ;
                    ; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
                    ;       multiplications already done, so simply perform a 32 bit add
                    ;       of previously obtained multiplication results
                    ;


007D 6018                   MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
007E 0F44                   ADDWF   ACC+B0                  ; add lowest byte of ACC, save in ACC(B0)
007F 6019                   MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
0080 1145                   ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0081 601A                   MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
0082 1146                   ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
0083 601B                   MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
0084 1147                   ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)

                    ;
                    ;
                    ; save the upper 16 bits  of D(n) from the 32 bit accumulator
                    ; left shift the result by 1, to adjust the decimal point after
                    ; a Q15*Q15 multiplication
                    ;
0085 1A45                   rlcf    ACC+B1,w
0086 1A46                   rlcf    ACC+B2,w
0087 0126                   movwf   Dn2
0088 1A47                   rlcf    ACC+B3,w               ; decimal adjust ( mult by 2)
0089 0127                   movwf   Dn2+B1
                    ;
                    ; Compute B2 * [D(n) + D(n-2)]
                    ;
                            if      B0_EQUALS_B2



008A 602A                            movfp   Dn2_2+B0,wreg
008B 0E26                            addwf   Dn2+B0,w
008C 011C                            movwf   AARG+B0
008D 602B                            movfp   Dn2_2+B1,wreg
008E 1027                            addwfc  Dn2+B1,w
008F 011D                            movwf   AARG+B1



0090 7E3A                            MOVFP   B20+B0,BARG+B0          ; move B20(B0) to BARG(B0)
0091 7F3B                            MOVFP   B20+B1,BARG+B1          ; move B20(B1) to BARG(B1)

0092 E0AF                           call    DblMult         ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
0093 5844                            MOVPF   DPX+B0,ACC+B0   ; move DPX(B0) to ACC(B0)
0094 5945                            MOVPF   DPX+B1,ACC+B1   ; move DPX(B1) to ACC(B1)
0095 5A46                            MOVPF   DPX+B2,ACC+B2   ; move DPX(B2) to ACC(B2)
0096 5B47                            MOVPF   DPX+B3,ACC+B3   ; move DPX(B3) to ACC(B3)

                     else
```

```
                                MOVFP16 B20,BARG
                                MOVFP16 Dn2,AARG
                                call    DblMult         ; B0*D(n)
                                MOVPF32 DPX,ACC
                                MOVFP16 Bx2,BARG
                                MOVFP16 Dn2_2,AARG
                                call    DblMult         ; B2*D(n-2)
                                ADD32   DPX,ACC
                        endif
                        ;
                        ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
                        ; This way in the next iteration D(n-2) is equal to the present D(n-1)
                        ;
0097 7C28                       movfp    Dn2_1,AARG+B0
0098 5C2A                       movpf    AARG+B0,Dn2_2          ; Shift down D(n-1)
0099 7D29                       movfp    Dn2_1+B1,AARG+B1
009A 5D2B                       movpf    AARG+B1,Dn2_2+B1       ; AARG = D(n-1) = multiplier
009B 7E3C                       MOVFP   B21+B0,BARG+B0          ; move B21(B0) to BARG(B0)
009C 7F3D                       MOVFP   B21+B1,BARG+B1          ; move B21(B1) to BARG(B1)
009D E0AF                       call    DblMult                 ; (ACCd,ACCc) = B1*D(n-1)
                        ;
                        ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                        ;                   = B1*D(n-1) + B0*[D(n) + D(n-2)]
                        ; Since all multiplications are already done, simply perform a
                        ; 32 bit addition
                        ;
009E 6018                       MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
009F 0F44                       ADDWF   ACC+B0                  ; add lowest byte of ACC, save in ACC(B0)
00A0 6019                       MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
00A1 1145                       ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
00A2 601A                       MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
00A3 1146                       ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
00A4 601B                       MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
00A5 1147                       ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)
                        ;
                        ; Shift down D(n) to D(n-1) so that in the next iteration, the new
                        ; D(n-1) is the present D(n)
                        ;
00A6 6026                       MOVFP   Dn2+B0,WREG             ; get byte of Dn2 into w
00A7 0128                       MOVWF   Dn2_1+B0                ; move to Dn2_1(B0)
00A8 6027                       MOVFP   Dn2+B1,WREG             ; get byte of Dn2 into w
00A9 0129                       MOVWF   Dn2_1+B1                ; move to Dn2_1(B1)
                        ;


                        ;
                        ; The filter output is now computed
                        ; Save the Upper 16 Bits of 32 bit Accumulator into Y after
                        ; adjusting the decimal point
                        ;

                        MOV16   ACC+B2,Y

00AA 6046                       MOVFP   ACC+B2+B0,WREG          ; get byte of ACC+B2 into w
00AB 0142                       MOVWF   Y+B0                    ; move to Y(B0)
00AC 6047                       MOVFP   ACC+B2+B1,WREG          ; get byte of ACC+B2 into w
00AD 0143                       MOVWF   Y+B1                    ; move to Y(B1)

                        ;
00AE 0002                       return                          ; Output Y(n) computed
                        ;
                        ;*****************************************************************
                        ;  Set SIGNED/UNSIGNED Flag Before Including 17c42MPY.mac
                        ;
                        include         “17c42MPY.mac”

                        ;*****************************************************************
                        ;  Low Pass Filter Co-efficients
                        ;
                        ;
```

```
                    ;                       ELLIPTIC     LOWPASS FILTER
                    ;
                    ;                FILTER ORDER =    4
                    ;                Sampling frequency =   2.000  KiloHertz
                    ;
                    ;                        BAND  1        BAND  2
                    ;
                    ;
                    ; LOWER BAND EDGE          .00000         .60000
                    ; UPPER BAND EDGE          .50000        1.00000
                    ; NOMINAL GAIN            1.00000         .00000
                    ; NOMINAL RIPPLE           .01000         .05000
                    ; MAXIMUM RIPPLE           .00906         .04601
                    ; RIPPLE IN dB             .07830      -26.74235
                    ;
                    ;  I        A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                    ;
                    ;  1      -.133331       .167145       .285431       .462921       .285431
                    ;  2       .147827       .765900       .698273       .499908       .698273
                    ;

                    _coeff_lpass                            ; co-efficients for 1st Cascade
01C1 1111                   data     4369                   ; -A11
01C2 EA9B                   data     -5477                  ; -A12
01C3 2489                   data     9353                   ; B10
01C4 3B41                   data     15169                  ; B11
01C5 2489                   data     9353                   ; B12
                    ;                                       ; co-efficients for 2nd Cascade
01C6 ED14                   data    -4844                   ; -A21
01C7 9DF7                   data    -25097                  ; -A22
01C8 5961                   data     22881                  ; B20
01C9 3FFD                   data     16381                  ; B21
01CA 5961                   data     22881                  ; B22
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ;*****************************************************************
                    ;  High Pass Filter Co-efficients
                    ;
                    ;
                    ;                       ELLIPTIC    HIGHPASS FILTER
                    ;
                    ;                FILTER ORDER =    4
                    ;                Sampling frequency =   2.000  KiloHertz
                    ;
                    ;                        BAND  1        BAND  2
                    ;
                    ;
                    ; LOWER BAND EDGE          .00000         .50000
                    ; UPPER BAND EDGE          .40000        1.00000
                    ; NOMINAL GAIN             .00000        1.00000
                    ; NOMINAL RIPPLE           .04000         .02000
                    ; MAXIMUM RIPPLE           .03368         .01686
                    ; RIPPLE IN dB          -29.45335         .14526
                    ;
                    ;  I        A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                    ;
                    ;  1       .276886       .195648       .253677      -.411407       .253677
                    ;  2      -.094299       .780396       .678650      -.485840       .678650
                    ;
                    ;
                    _coeff_hpass                    ; co-efficients for 1st Cascade section
01CB DC8F                   data     -9073          ; -A11
01CC E6F5                   data     -6411          ; -A12
01CD 2079                   data     8313           ; B10
01CE CB57                   data     -13481         ; B11
01CF 2079                   data     8313           ; B12
                    ;                               ; co-efficients for 2nd Cascade section
```

```
01D0 0C12                       data    3090            ; -A21
01D1 9C1C                       data    -25572          ; -A22
01D2 56DE                       data    22238           ; B20
01D3 C1D0                       data    -15920          ; B21
01D4 56DE                       data    22238           ; B22
                        ;
                        ;*****************************************************************
                        ;
                        ;*****************************************************************
                        ;  Band Pass Filter Co-efficients
                        ;
                        ;
                        ;                       ELLIPTIC     BANDPASS FILTER
                        ;
                        ;               FILTER ORDER =    4
                        ;               Sampling frequency =   2.000  KiloHertz
                        ;
                        ;                        BAND  1        BAND  2        BAND  3
                        ;
                        ;
                        ; LOWER BAND EDGE         .00000         .30000         .90000
                        ; UPPER BAND EDGE         .10000         .70000        1.00000
                        ; NOMINAL GAIN            .00000        1.00000         .00000
                        ; NOMINAL RIPPLE          .05000         .05000         .05000
                        ; MAXIMUM RIPPLE          .03644         .03867         .03641
                        ; RIPPLE IN dB         -28.76779         .32956      -28.77647
                        ;
                        ;
                        ;  I        A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                        ;
                        ;  1      -.936676       .550568       .444000      -.865173       .444000
                        ;  2       .936707       .550568       .615540      1.199402       .615540
                        ;
                        _coeff_bpass                    ; co-efficients for 1st Cascade section
01D5 3BF2                       data     30693/2        ; -A11
01D6 DCC4                       data     -18041/2       ; -A12
01D7 1C6A                       data     14549/2        ; B10
01D8 C8A1                       data     -28350/2       ; B11
01D9 1C6A                       data     14549/2        ; B12
                        ;                               ; co-efficients for 2nd Cascade section
01DA C40D                       data    -30694/2        ; -A21
01DB DCC4                       data    -18041/2        ; -A22
01DC 2765                       data     20170/2        ; B20
01DD 4CC3                       data     39302/2        ; B21
01DE 2765                       data     20170/2        ; B22
                        ;
                        ;*****************************************************************
                        ;
                        ;*****************************************************************
                        ;  Band Stop Filter Co-efficients
                        ;
                        ;
                        ;                      ELLIPTIC    BANDSTOP FILTER
                        ;
                        ;              FILTER ORDER =    4
                        ;              Sampling frequency =   2.000  KiloHertz
                        ;
                        ;                       BAND  1        BAND  2        BAND  3
                        ;
                        ;
                        ;LOWER BAND EDGE         .00000         .45000         .70000
                        ;UPPER BAND EDGE         .30000         .55000        1.00000
                        ;NOMINAL GAIN           1.00000         .00000        1.00000
                        ;NOMINAL RIPPLE          .05000         .05000         .05000
                        ;MAXIMUM RIPPLE          .03516         .03241         .03517
                        ;RIPPLE IN dB            .30015      -29.78523         .30027
                        ;
                        ;
```

```
                    ;  I        A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                    ;
                    ;  1       .749420       .583282       .392685       .087936       .392685
                    ;  2      -.749390       .583282      1.210022      -.270935      1.210022
                    ;
                    _coeff_bstop                    ; co-efficients for 1st Cascade section
01DF D00A                   data     -24557/2       ; -A11
01E0 DAAC                   data     -19113/2       ; -A12
01E1 1922                   data     12868/2         ; B10
01E2 05A0                   data     2881/2          ; B11
01E3 1922                   data     12868/2         ; B12
                                                    ; co-efficients for 2nd Cascade section
01E4 2FF6                   data    24556/2          ; -A21
01E5 DAAC                   data    -19113/2         ; -A22
01E6 4D71                   data    39650/2          ; B20
01E7 EEA9                   data    -8878/2          ; B21
01E8 4D71                   data    39650/2          ; B22
                    ;
                    ;*****************************************************************
                    '

                           END
 Errors   :    0
 Warnings :    0
```

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200 Fax: 602 786-7277
Technical Support: 602 786-7627
Web: http://www.mchip.com/microhip

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770 640-0034 Fax: 770 640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508 480-9990 Fax: 508 480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 708 285-0071 Fax: 708 285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 214 991-7177 Fax: 214 991-8588

Dayton
Microchip Technology Inc.
35 Rockridge Road
Englewood, OH 45322
Tel: 513 832-2543 Fax: 513 832-2841

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 455
Irvine, CA 92715
Tel: 714 263-1888 Fax: 714 263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516 273-5305 Fax: 516 273-5335

AMERICAS (continued)

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408 436-7950 Fax: 408 436-7955

ASIA/PACIFIC

Hong Kong
Microchip Technology
Unit No. 3002-3004, Tower 1
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T. Hong Kong
Tel: 852 2 401 1200 Fax: 852 2 401 3431

Korea
Microchip Technology
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku,
Seoul, Korea
Tel: 82 2 554 7200 Fax: 82 2 558 5934

Singapore
Microchip Technology
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65 334 8870 Fax: 65 334 8850

Taiwan
Microchip Technology
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2 717 7175 Fax: 886 2 545 0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44 0 1628 851077 Fax: 44 0 1628 850259

France
Arizona Microchip Technology SARL
2 Rue du Buisson aux Fraises
91300 Massy - France
Tel: 33 1 69 53 63 20 Fax: 33 1 69 30 90 79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Muenchen, Germany
Tel: 49 89 627 144 0 Fax: 49 89 627 144 44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Pegaso Ingresso No. 2
Via Paracelso 23, 20041
Agrate Brianza (MI) Italy
Tel: 39 039 689 9939 Fax: 39 039 689 9883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81 45 471 6166 Fax: 81 45 471 6122

9/22/95