

Implementation of Fast Fourier Transforms

INTRODUCTION

Fourier transforms are one of the fundamental operations in signal processing. In digital computations, Discrete Fourier Transforms (DFT) are used to describe, represent and analyze discrete-time signals. However, direct implementation of DFT is computationally very inefficient. Of the various available high speed algorithms to compute DFT, the Cooley-Tukey algorithm is the simplest and most commonly used. These efficient algorithms to compute DFTs are called Fast Fourier Transforms (FFTs).

This application note provides the source code to compute the FFT using PIC17C42. The theory behind the FFT algorithms is well established and described in the literature and hence not described in this application note. A Radix-2 Cooley-Tukey FFT is implemented with no limits on the length of FFT. The length is only limited by the amount of available program memory space. All computations are done using double precision arithmetic.

IMPLEMENTATION

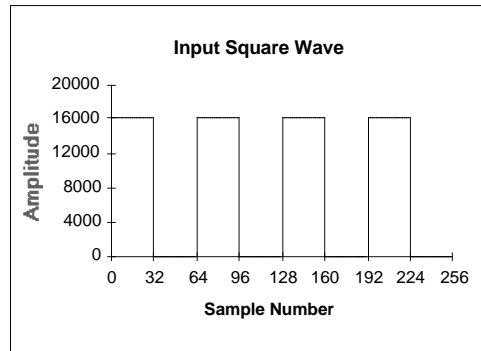
Since the PIC17C42 has only 232 x 8 general purpose RAM (equivalent of 116 x 16), at most a 32 point FFT (16-bit REAL & IMAGINARY data) can be implemented using on chip RAM. To compute higher point FFT, the data is stored in program memory space of PIC17C42. The PIC17C42 has instructions (TABLRD & TABLWT) to transfer data between program memory space and on chip file registers. In extended microcontroller mode, the PIC17C42 has 2K x 16 (0000h:07FFh) on chip program memory space and 62K x 16 (0800h:0FFFFh) of external program memory space. In this mode, the code (in this case, the FFT code) may reside on the on chip EPROM and the data to be analyzed may be stored in external RAMs (up to 62K). A suggested method of connecting external RAMs (appropriate EEPROMs may also be used) is shown in Figure 3.

If PIC17C42 is used in extended microcontroller mode and if all the code resides on chip, then the cost may further be reduced by using only one external SRAM instead of two. The block diagram is shown in Figure 2. The 16-bit data stored in the external RAM is organized as low byte followed by high byte. To achieve this, the code presented in this application note needs minor modifications, especially where TABLRD and TABLWR instructions are used. Address indexing must be incremented by two since 2 reads/writes must be performed to access a 16-bit data.

The FFT is implemented with Decimation In Frequency. Thus the input data before calling the FFT routine ("R2FFT") should be in normal order and the transformed data is in scrambled order. The original data is overwritten by the transformed data to conserve memory. This is achieved by use of in-place calculations. This in-place calculations causes the order of the DFT terms to be permuted. So at the end of the transform, all the data needs to be unscrambled to get the right order of the DFT terms. In some applications the order of terms is not necessary. Keeping this in mind, the unscrambling code is written as a separate subroutine ("Unscramble") and may be called if necessary.

Before implementing the FFT using PIC17C42, a "C" program was written and tested. This high level programming helps in writing the assembly code and the results of both programs can be compared against while debugging the assembly code. The "C" source code for the Radix-2 FFT is shown in Appendix A. The assembly code source file of the FFT program is shown in Appendix B. For a listing of the header file "17C42.h" and the macro definition file "17C42.mac" please refer to Appendices C and D respectively of the application note AN00540.

FIGURE 1 - TEST WAVE FORM



Implementing FFT

TESTING

The assembly code was developed and debugged using Microchip's PICMASTER™ In-Circuit Emulator System. A main program generates a test pattern (like a square wave) and calls the FFT routines "R2FFT" & "Unscramble". After the DFT terms are computed, the results are captured into PICMASTER's real time trace buffer by putting a trace point on a dummy TABLRD instruction and capturing only the 2nd cycle (data cycle of TABLRD) of the instruction. The data from Trace buffer was hot linked to Microsoft Excel™ using DDE and then the graphs were plotted and analyzed.

The code was tested on various wave forms (a rectangular pulse, a triangular wave, square wave and a sine wave) and using FFT lengths of 64, 256 & 1024. The results of a 256 Point FFT on a square wave is shown below. The test wave form is shown in Figure 1 and the frequency spectrum of it computed by PIC17C42 is shown in Figure 2. As expected, the spectra appears at the odd harmonics of the input waveform's fundamental frequency (At $N*256/64$, $N = 0, 1, 3, 5, \dots$).

PERFORMANCE

The performance of FFT using PIC17C42 is quite impressive noting that for an 8-bit machine with no hardware multiplier. Also note that all computations are performed using double precision arithmetic (16- and 32-bits) which is the case in most of the low end DSPs. Table 1 provides the real time performance in total number of Instruction cycles for both the "R2FFT" and "Unscramble" routines using 64, 256 and 1024 Point FFT. Note that the timings are in a worst case situation and in general will be a lot better than shown in the table. The worst case situation arises because the 16 x 16 software multiplier ("DbIMult") does not have a uniform timing and depends on the input data. The worst case timing of the multiplier is used in the computation of performance.

FIGURE 2 - FFT (MAGNITUDE SPECTRUM) OF FIGURE 1 COMPUTED BY PIC17C42

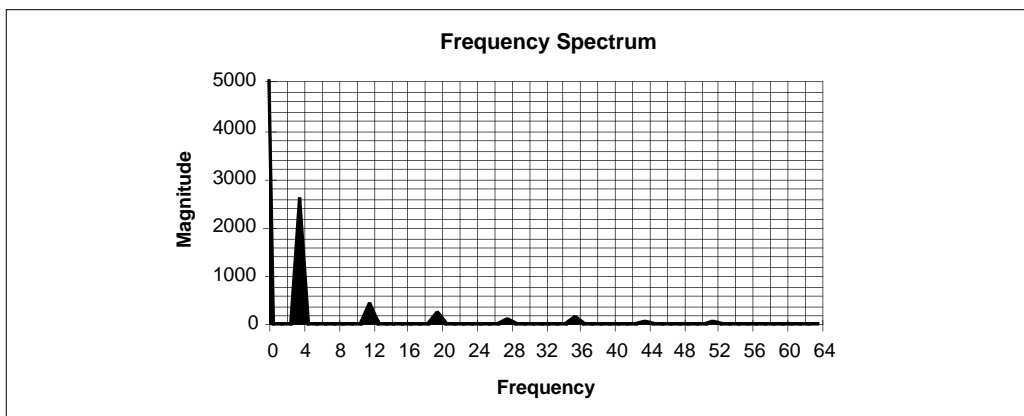


TABLE 1 - WORST CASE PERFORMANCE OF FFT IN INSTRUCTION CYCLES AND REAL TIME @ 25 MHZ

N (FFT Length)	64 Point	256 Point	1024 Point
R2FFT	34116 + 768*Mult = 171588	178024 + 4096*Mult= 911208	878752 + 20480*Mult = 4544672
Unscramble	5143	22495	93525
Total	176731 (28.28 mS)	933703 (149.39 mS)	4638197 (742.11 mS)

Table 2 shows the Program Memory and Data RAM requirements for an N Point FFT. The multiplier routine and other general purpose macro requirements are included in the memory requirements. The speed performance for the square wave test data differs from Table 1 since "worst case timings" is not used, and reflects a more reasonable data.

FFT APPLICATIONS

Although the FFT does not find a place in many microcontroller applications, it is very useful in providing a benchmark of the processor. As can be seen from table 2, the performance is very satisfactory, considering the fact that PIC17C42 is a Microcontroller and not a DSP. Also it should be borne in mind that all computations are performed in 16/32 bit arithmetic and that PIC17C42 is a low-cost 8-bit device unlike the DSPs which are relatively expensive.

In applications like Instrumentation, where real time FFT computation is not required, PIC17C42 can be used as a single chip solution instead of a Microcontroller and a Digital Signal Processor.

Suggested Reading :

- [1] Rabiner. L.R., and Gold, B., Theory and Application Of Digital Signal Processing, Englewood Cliffs, NJ : Prentice-Hall, 1975.
- [2] Burrs, C.S., and Parks, T.W., DFT/FFT and Convolution Algorithms, New York : Wiley, 1985.
- [3] Rodriguez, Jeffrey J., "An Improved FFT Digit-Reversal Algorithm," IEEE Transactions On Acoustics, Speech, And Signal Processing, Vol. 37, No. 8, Aug 1989.

Author: Amar Palacherla
Logic Products Division

TABLE 2 - REQUIREMENTS FOR RADIX-2 FFT @ 25 MHZ

N (FFT Length)	64 Point	256 Point	1024 Point
Code Space (locations)	$603 + 0.75 * N = 651$	$603 + 0.75 * N = 795$	$603 + 0.75 * N = 1371$
Data Storage in Program Memory Space	$2 * N = 128$	$2 * N = 512$	$2 * N = 2048$
Scratch RAM	49	49	49
Performance (Square Wave Input Data)	122384 (19.58 mS)	644416 (103.11 mS)	3192176 (510.75mS)

FIGURE 3 - EXTERNAL MEMORY CONNECTION

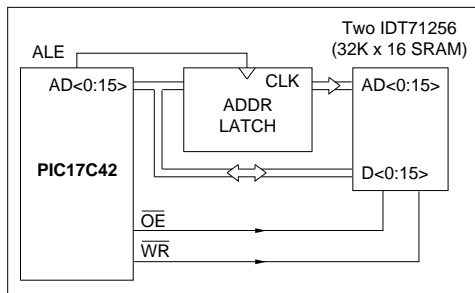
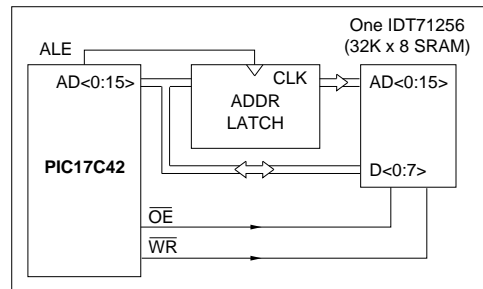


FIGURE 4 - ALTERNATE EXTERNAL MEMORY CONNECTION



Implementing FFT

APPENDIX A: FFT ALGORITHM

MPASM B0.54

PAGE 1

```
*****
;
;       A Cooley-Tukey Radix-2 DIF FFT
;
;       Radix-2 implementation
;       Decimation In Frequency
;       Single Butterfly
;       Table Lookup of Twiddle Factors
;       Complex Input & Complex Output
;
;       All data is assumed to be 16 bits and the intermediate
;       results are stored in 32 bits
;
;       Length Of FFT must be a Power Of 2
;       Max Length Possible is 2**15
;
;       The input/output complex data is organized as a single array
;       of real data followed by imaginary data
;       Data is stored in External Memory and is accessed by
;       TABLRD & TABLWT Instructions
;
;*****
;
;       LIST    P=17C42, C=120, T=ON, R=DEC, N=0
;       include "17c42.h"
;
;       include "17c42.mac"
;*****
;
;       RLC16AB
;
; DESCRIPTION:
;       16 bit rotate left A into B
;
; ARGUMENTS:
;       2*a => b
;
; TIMING (in cycles):
;       3
;
;
; RLC16AB MACRO    a,b
;
;       BCF        _carry
;       RLCF        a+B0,W
;       MOVWF       b+B0
;       RLCF        a+B1,W
;       MOVWF       b+B1
;       ENDM
;*****
;
;       TBLADDR
;
; DESCRIPTION:
;       Load 16 bit table pointer with specified label
;
; TIMING (in cycles):
;       4
;
;
; TBLADDR MACRO    label
;
;       MOVLW      (label) & 0xff
;       MOVWF      tblptrl
;       MOVLW      page    (label)
;       MOVWF      tblptrh
;       ENDM
;*****
```

```

;          ADDLBL
;
; DESCRIPTION:
;          Add A Label (16 bit constant) To A File Register (16 bit)
;
; TIMING (in cycles):
;          4
;
ADDLBL MACRO  label,f
        MOVLW  (label) & 0xff
        ADDWF  f+B0
        MOVLW  page    (label)
        ADDWFC f+B1
        ENDM
;*****
0100 FftLen .set 256 ; FFT Length
0008 Power .set 8 ; (2**Power = FftLen)
00EF DigitRevCount .set 239 ; (FftLen-1) - (2**((Power+1)/2))

0001 SCALE_BUTTERFLY .set TRUE ; intermediate scaling performed

0800 EXT_RAM_START_ADDR .set 0x0800 ; External Memory Data Storage Start Addr
;
;*****
;          CBLOCK 0
0000 0004 B0,B1,B2,B3 ; RAM offset constants
;          ENDC
;
;          CBLOCK 0x18
0018 0002 AARG,AARG1 ; 16 bit multiplier A
001A 0002 BARG,BARG1 ; 16 bit multiplicand B
001C 0004 DPX,DPX1,DPX2,DPX3 ; 32 bit multiplier result = A*B
;          ENDC
;
;          CBLOCK
0020 0004 ACC, ACC1, ACC2, ACC3 ; 32 bit accumulator for computa
;          ENDC
;
;          CBLOCK
0024 0002 count1,count11 ; N1
0026 0002 count2,count22 ; N2
0028 0002 QuartLen,QuartLen1 ; FftLen/4
;          ENDC
;
;          CBLOCK
002A 0002 TF_Offset,TF_Offset1
002C 0002 TF_Addr,TF_Addr1 ; twiddle factor address computa
002E 0002 Cos,Cos1,
0030 0002 Sin,Sin1
;          ENDC
;
;          CBLOCK
0032 0002 VarIloop,VarIloop1
0034 0002 VarJloop,VarJloop1
0036 0001 VarKloop
0037 0002 VarL,VarL1
;          ENDC
;
;          CBLOCK
0039 0002 Xi,Xi1
003B 0002 Yi,Yi1
003D 0002 Xl,Xl1
003F 0002 Yl,Yl1
;          ENDC
;

```

Implementing FFT

```
                                CBLOCK
0041 0002                        Xt,Xt1
0043 0002                        Yt,Yt1
                                ENDC
                                CBLOCK
0045 0004                        temp,temp1,temp2,temp3
0049 0002                        testCount,testCount1
004B 0002                        PulseCount, PulseCount1
                                ENDC
;
;
;*****
;                                Test Program For FFT Subroutine
;*****
;
                                ORG    0x0000
;
                                include "square.asm"           ; Generate Test Vector Data
;*****
;                                Test Routine For FFT
; FFT Of Square Wave Pulse
;*****

0008                                PulseWidthFactor    .set    8
;
testFft
                                MOVK16  2*PulseWidthFactor,testCount

0000 B010                        MOVLW   (2*PulseWidthFactor) & 0xff
0001 0149                        MOVWF  testCount+B0
0002 B000                        MOVLW   (2*PulseWidthFactor)/256
0003 014A                        MOVWF  testCount+B1
                                CLR16   Yi
0004 293B                        CLRF  Yi+B0
0005 293C                        CLRF  Yi+B1
                                TBLADDR  ExtRamAddr           ; load table pointers with data start addr
0006 B000                        MOVLW   (ExtRamAddr) & 0xff
0007 010D                        MOVWF  tblptrl
0008 B008                        MOVLW   page    (ExtRamAddr)
0009 010E                        MOVWF  tblptrh
;
nextPulse
                                MOVK   FftLen/PulseWidthFactor,PulseCount
000A B020                        MOVLW  FftLen/PulseWidthFactor
000B 014B                        MOVWF  PulseCount
                                MOVK   FftLen/PulseWidthFactor,PulseCount1
000C B020                        MOVLW  FftLen/PulseWidthFactor
000D 014C                        MOVWF  PulseCount1
;
                                MOVK16  0x3FFF,Xi
000E B0FF                        MOVLW  (0x3FFF) & 0xff
000F 0139                        MOVWF  Xi+B0
0010 B03F                        MOVLW  (0x3FFF)/256
0011 013A                        MOVWF  Xi+B1
                                LX1
0012 E034                        call   write
0013 174B                        decfsz PulseCount
0014 C012                        goto  LX1
;
                                CLR16  Xi
0015 2939                        CLRF  Xi+B0
0016 293A                        CLRF  Xi+B1
                                LX2
0017 E034                        call   write
0018 174C                        decfsz PulseCount1
0019 C017                        goto  LX2
;
                                DECL6  testCount
```

```

001A 2900          CLRF    WREG
001B 0749          DECF    testCount+B0
001C 034A          SUBWFB testCount+B1
                   TFSZ16  testCount
001D 6049          MOVFP  testCount+B0,WREG
001E 084A          IORWF  testCount+B1,W
001F 3300          TSTFSZ WREG
0020 C00A          goto   nextPulse
                   ;
0021 E039          call   R2FFT          ; Compute Fourier Transform
0022 E116          call   Unscramble      ; Digit Reverse the scrambled data
                   ;
                   ; Fourier Transform Completed
                   ;
                   ; capture data to PIC-MASTER Trace Buffer
                   MOVK16  FftLen*2,temp
0023 B000          MOVLW  (FftLen*2) & 0xff
0024 0145          MOVWF  temp+B0
0025 B002          MOVLW  (FftLen*2)/256
0026 0146          MOVWF  temp+B1
                   TBLADDR  ExtRamAddr      ; load table pointers with data start addr
0027 B000          MOVLW  (ExtRamAddr) & 0xff
0028 010D          MOVWF  tblptrl
0029 B008          MOVLW  page      (ExtRamAddr)
002A 010E          MOVWF  tblptrh
                   capture
002B A930          tablrd  0,1,Sin          ; table latch = mem(tblptr)
                   DEC16  temp
002C 2900          CLRF    WREG
002D 0745          DECF    temp+B0
002E 0346          SUBWFB temp+B1
                   TFSZ16  temp
002F 6045          MOVFP  temp+B0,WREG
0030 0846          IORWF  temp+B1,W
0031 3300          TSTFSZ WREG
0032 C02B          goto   capture
                   ;
0033 C033          self   goto   self
                   ;
                   write
0034 A439          tblwt  0,Xi
0035 AF3A          tblwt  1,1,Xi+B1          ; auto increment for Imag Data
0036 A43B          tblwt  0,Yi
0037 AF3C          tblwt  1,1,Yi+B1
0038 0002          return
                   ;
                   ;*****
                   ;
                   ;          RADIX-2 FFT
                   ;
                   ;          Decimation In Frequency
                   ;
                   ;          ; Input Data should be unscrambled
                   ;          ; Output Data at the end is in scrambled form
                   ;          ; To obtain the unscrambled form, the digit reverse counter
                   ;          ; subroutine, "Unscramble" should be called (see the example)
                   ;
                   ;*****
R2FFT
0039 B000          MOVK16  FftLen,count2      ; count2 = N
003A 0126          MOVLW  (FftLen) & 0xff
003B B001          MOVWF  count2+B0
003C 0127          MOVLW  (FftLen)/256
                   MOVWF  count2+B1
                   MOVK16  FftLen/4,QuartLen ; QuartLen = FftLen/4
003D B040          MOVLW  (FftLen/4) & 0xff
003E 0128          MOVWF  QuartLen+B0
003F B000          MOVLW  (FftLen/4)/256
0040 0129          MOVWF  QuartLen+B1

```

Implementing FFT

```
0041 292B                                clrf    TF_Offset+B1
                                           ; Init TF_Offset = 1
0042 B001                                MOVK    1,TF_Offset
0043 012A                                MOVLW  1
                                           MOVWF  TF_Offset
0044 B008                                MOVK    Power,VarKloop           ; Kloop
0045 0136                                MOVLW  Power
                                           MOVWF  VarKloop
                                           Kloop                               ; for K = 1 to Power-1
                                           MOV16  count2,count1             ; count1 = count2
0046 6026                                MOVFP  count2+B0,WREG           ; get byte of count2 into w
0047 0124                                MOVWF  count1+B0                ; move to count1(B0)
0048 6027                                MOVFP  count2+B1,WREG           ; get byte of count2 into w
0049 0125                                MOVWF  count1+B1                ; move to count1(B1)
                                           RRC16  count2                    ; count2 = count2/2
004A 1A27                                RLCF  count2+B1,W              ; move sign into carry bit
004B 1927                                RRCF  count2+B1
004C 1926                                RRCF  count2+B0
004D 2934                                CLR16  VarJloop                 ; J = 0
004E 2935                                CLRFB VarJloop+B0
                                           CLR16  TF_Addr                    ; TF_Addr = 0
004F 292C                                CLRFB TF_Addr+B0
0050 292D                                CLRFB TF_Addr+B1
                                           Jloop
;
; Read Twiddle factors from Sine/Cosine Table from Prog Mem
;
                                           MOVFP16 TF_Addr,tblptrl          ; load sine table address to table
0051 6D2C                                MOVFP  TF_Addr+B0,tblptrl+B0     ; move TF_Addr(B0) to tblptrl(B0)
0052 6E2D                                MOVFP  TF_Addr+B1,tblptrl+B1     ; move TF_Addr(B1) to tblptrl(B1)
                                           ADDLBL SineTable,tblptrl        ; add table offset
0053 B094                                MOVLW  (SineTable) & 0xff
0054 0F0D                                ADDWF  tblptrl+B0
0055 B002                                MOVLW  page(SineTable)
0056 110E                                ADDWFC tblptrl+B1
0057 A830                                tablrd 0,0,Sin                  ; Read Sine Value from lookup table
0058 A030                                tlrld 0,Sin
0059 A231                                tlrld 1,Sin+B1
                                           ADD16  QuartLen,tblptrl
005A 6028                                MOVFP  QuartLen+B0,WREG         ; get lowest byte of QuartLen into w
005B 0F0D                                ADDWF  tblptrl+B0                ; add lowest byte of tblptrl, save in
005C 6029                                MOVFP  QuartLen+B1,WREG         ; get 2nd byte of QuartLen into w
005D 110E                                ADDWFC tblptrl+B1                ; add 2nd byte of tblptrl, save in
005E A82E                                tablrd 0,0,Cos                  ; Read Cosine Value from table
005F A02E                                tlrld 0,Cos
0060 A22F                                tlrld 1,Cos+B1
;
                                           ADD16  TF_Offset,TF_Addr         ; TF_Addr = TF_Addr + TF_Offset
0061 602A                                MOVFP  TF_Offset+B0,WREG         ; get lowest byte of TF_Offset into w
0062 0F2C                                ADDWF  TF_Addr+B0                ; add lowest byte of TF_Addr, save in
0063 602B                                MOVFP  TF_Offset+B1,WREG         ; get 2nd byte of TF_Offset into w
0064 112D                                ADDWFC TF_Addr+B1                ; add 2nd byte of TF_Addr, save in
;
                                           RLC16AB VarJloop,VarIloop
0065 8804                                BCF   _carry
0066 1A34                                RLCF  VarJloop+B0,W
0067 0132                                MOVWF VarIloop+B0
0068 1A35                                RLCF  VarJloop+B1,W
0069 0133                                MOVWF VarIloop+B1
;
                                           Iloop
                                           RLC16AB count2,VarL             ; VarL = count2*2
006A 8804                                BCF   _carry
006B 1A26                                RLCF  count2+B0,W
006C 0137                                MOVWF VarL+B0
006D 1A27                                RLCF  count2+B1,W
006E 0138                                MOVWF VarL+B1
                                           ADD16  VarIloop,VarL             ; VarL = (I+count2)*2
```



```

006F 6032      MOVFP  VarIloop+B0,WREG      ; get lowest byte of VarIloop into w
0070 0F37      ADDWF  VarL+B0              ; add lowest byte of VarL, save in
0071 6033      MOVFP  VarIloop+B1,WREG      ; get 2nd byte of VarIloop into w
0072 1138      ADDWFC VarL+B1              ; add 2nd byte of VarL, save in VarL(B1)
;
; Get Real & Imag Data from external RAMS (Program Memory)
; load table pointers with data start addr
;
      MOVFP16 VarL,tblptrl      ; read data(L)
0073 6D37      MOVFP  VarL+B0,tblptrl+B0     ; move VarL(B0) to tblptrl(B0)
0074 6E38      MOVFP  VarL+B1,tblptrl+B1     ; move VarL(B1) to tblptrl(B1)
      ADDLBL ExtRamAddr,tblptrl ; add data addr offset
0075 B000      MOVLW  (ExtRamAddr) & 0xff
0076 0F0D      ADDWF  tblptrl+B0
0077 B008      MOVLW  page (ExtRamAddr)
0078 110E      ADDWFC tblptrl+B1
0079 A93D      tablrld 0,1,Xl          ; auto increment for Imag Data
007A A03D      tlrld  0,Xl
007B A23E      tlrld  1,Xl+B1          ; real data XL
007C A83F      tablrld 0,0,Yl
007D A03F      tlrld  0,Yl
007E A240      tlrld  1,Yl+B1          ; imag data YL
      MOVFP16 VarIloop,tblptrl  ; read data(I)
007F 6D32      MOVFP  VarIloop+B0,tblptrl+B0 ; move VarIloop(B0) to tblptrl(B0)
0080 6E33      MOVFP  VarIloop+B1,tblptrl+B1 ; move VarIloop(B1) to tblptrl(B1)
      ADDLBL ExtRamAddr,tblptrl ; add data addr offset
0081 B000      MOVLW  (ExtRamAddr) & 0xff
0082 0F0D      ADDWF  tblptrl+B0
0083 B008      MOVLW  page (ExtRamAddr)
0084 110E      ADDWFC tblptrl+B1
0085 A939      tablrld 0,1,Xi          ; auto increment for Imag Data
0086 A039      tlrld  0,Xi
0087 A23A      tlrld  1,Xi+B1          ; real data XI
0088 A83B      tablrld 0,0,Yi
0089 A03B      tlrld  0,Yi
008A A23C      tlrld  1,Yi+B1          ; imag data YI
;
; Real & Imag Data is fetched
; Compute Butterfly
;
      SUB16ACC      Xl,Xi,Xt      ; Xt = Xi - Xl
008B 603D      MOVFP  Xl+B0,WREG      ; get lowest byte of Xl into w
008C 0439      SUBWF  Xi+B0,W          ; sub lowest byte of Xi, save in Xi(B0)
008D 0141      MOVWF  Xt+B0
008E 603E      MOVFP  Xl+B1,WREG      ; get 2nd byte of Xl into w
008F 023A      SUBWFB Xi+B1,W          ; sub 2nd byte of Xi, save in Xi(B1)
0090 0142      MOVWF  Xt+B1
      ADD16      Xl,Xi          ; Xi = Xi + Xl
0091 603D      MOVFP  Xl+B0,WREG      ; get lowest byte of Xl into w
0092 0F39      ADDWF  Xi+B0          ; add lowest byte of Xi, save in Xi(B0)
0093 603E      MOVFP  Xl+B1,WREG      ; get 2nd byte of Xl into w
0094 113A      ADDWFC Xi+B1          ; add 2nd byte of Xi, save in Xi(B1)
      SUB16ACC      Yl,Yi,Yt      ; Yt = Yi - Yl
0095 603F      MOVFP  Yl+B0,WREG      ; get lowest byte of Yl into w
0096 043B      SUBWF  Yi+B0,W          ; sub lowest byte of Yi, save in Yi(B0)
0097 0143      MOVWF  Yt+B0
0098 6040      MOVFP  Yl+B1,WREG      ; get 2nd byte of Yl into w
0099 023C      SUBWFB Yi+B1,W          ; sub 2nd byte of Yi, save in Yi(B1)
009A 0144      MOVWF  Yt+B1
      ADD16      Yl,Yi          ; Yi = Yi + Yl
009B 603F      MOVFP  Yl+B0,WREG      ; get lowest byte of Yl into w
009C 0F3B      ADDWF  Yi+B0          ; add lowest byte of Yi, save in Yi(B0)
009D 6040      MOVFP  Yl+B1,WREG      ; get 2nd byte of Yl into w
009E 113C      ADDWFC Yi+B1          ; add 2nd byte of Yi, save in Yi(B1)
;
      if SCALE_BUTTERFLY
RRC16      Xi
009F 1A3A      RLCF  Xi+B1,W          ; move sign into carry bit
00A0 193A      RRCF  Xi+B1

```

Implementing FFT

```

00A1 1939          RRCF   Xi+B0
                  RRC16  Yi
00A2 1A3C          RLCF   Yi+B1,W          ; move sign into carry bit
00A3 193C          RRCF   Yi+B1
00A4 193B          RRCF   Yi+B0
                  RRC16  Xt
00A5 1A42          RLCF   Xt+B1,W          ; move sign into carry bit
00A6 1942          RRCF   Xt+B1
00A7 1941          RRCF   Xt+B0
                  RRC16  Yt
00A8 1A44          RLCF   Yt+B1,W          ; move sign into carry bit
00A9 1944          RRCF   Yt+B1
00AA 1943          RRCF   Yt+B0
                  endif

;

MOVFP16 Cos,AARG
00AB 782E          MOVFP  Cos+B0,AARG+B0      ; move Cos(B0) to AARG(B0)
00AC 792F          MOVFP  Cos+B1,AARG+B1      ; move Cos(B1) to AARG(B1)
                  MOVFP16 Yt,BARG
00AD 7A43          MOVFP  Yt+B0,BARG+B0      ; move Yt(B0) to BARG(B0)
00AE 7B44          MOVFP  Yt+B1,BARG+B1      ; move Yt(B1) to BARG(B1)
00AF E182          Call   DblMult ; COS*Yt
                  MOVFP32 DPX,ACC
00B0 5C20          MOVFP  DPX+B0,ACC+B0      ; move DPX(B0) to ACC(B0)
00B1 5D21          MOVFP  DPX+B1,ACC+B1      ; move DPX(B1) to ACC(B1)
00B2 5E22          MOVFP  DPX+B2,ACC+B2      ; move DPX(B2) to ACC(B2)
00B3 5F23          MOVFP  DPX+B3,ACC+B3      ; move DPX(B3) to ACC(B3)
                  MOVFP16 Sin,AARG
00B4 7830          MOVFP  Sin+B0,AARG+B0      ; move Sin(B0) to AARG(B0)
00B5 7931          MOVFP  Sin+B1,AARG+B1      ; move Sin(B1) to AARG(B1)
                  MOVFP16 Xt,BARG
00B6 7A41          MOVFP  Xt+B0,BARG+B0      ; move Xt(B0) to BARG(B0)
00B7 7B42          MOVFP  Xt+B1,BARG+B1      ; move Xt(B1) to BARG(B1)
00B8 E182          Call   DblMult ; SIN*Xt, Scale if necessary
                  ADD32  ACC,DPX
00B9 6020          MOVFP  ACC+B0,WREG      ; get lowest byte of ACC into w
00BA 0F1C          ADDWF  DPX+B0          ; add lowest byte of DPX, save in DPX(B0)
00BB 6021          MOVFP  ACC+B1,WREG      ; get 2nd byte of ACC into w
00BC 111D          ADDWFC DPX+B1          ; add 2nd byte of DPX, save in DPX(B1)
00BD 6022          MOVFP  ACC+B2,WREG      ; get 3rd byte of ACC into w
00BE 111E          ADDWFC DPX+B2          ; add 3rd byte of DPX, save in DPX(B2)
00BF 6023          MOVFP  ACC+B3,WREG      ; get 4th byte of ACC into w
00C0 111F          ADDWFC DPX+B3          ; add 4th byte of DPX, save in DPX(B3)
                  MOVFP16 DPX+B2,Y1
00C1 5E3F          MOVFP  DPX+B2+B0,Y1+B0      ; move DPX+B2(B0) to Y1(B0)
00C2 5F40          MOVFP  DPX+B2+B1,Y1+B1      ; move DPX+B2(B1) to Y1(B1)

;

MOVFP16 Yt,BARG
00C3 7A43          MOVFP  Yt+B0,BARG+B0      ; AARG = SIN, BARG = Yt
00C4 7B44          MOVFP  Yt+B1,BARG+B1      ; move Yt(B0) to BARG(B0)
00C5 E182          Call   DblMult ; SIN*Yt
                  MOVFP32 DPX,ACC
00C6 5C20          MOVFP  DPX+B0,ACC+B0      ; move DPX(B0) to ACC(B0)
00C7 5D21          MOVFP  DPX+B1,ACC+B1      ; move DPX(B1) to ACC(B1)
00C8 5E22          MOVFP  DPX+B2,ACC+B2      ; move DPX(B2) to ACC(B2)
00C9 5F23          MOVFP  DPX+B3,ACC+B3      ; move DPX(B3) to ACC(B3)
                  MOVFP16 Cos,AARG
00CA 782E          MOVFP  Cos+B0,AARG+B0      ; move Cos(B0) to AARG(B0)
00CB 792F          MOVFP  Cos+B1,AARG+B1      ; move Cos(B1) to AARG(B1)
                  MOVFP16 Xt,BARG
00CC 7A41          MOVFP  Xt+B0,BARG+B0      ; move Xt(B0) to BARG(B0)
00CD 7B42          MOVFP  Xt+B1,BARG+B1      ; move Xt(B1) to BARG(B1)
00CE E182          Call   DblMult ; COS*Xt, Scale if necessary
                  SUB32  ACC,DPX
00CF 6020          MOVFP  ACC+B0,WREG      ; DPX = COS*Xt - SIN*Yt
00D0 051C          SUBWF  DPX+B0          ; get lowest byte of ACC into w
00D1 6021          MOVFP  ACC+B1,WREG      ; sub lowest byte of DPX, save in DPX(B0)
00D2 031D          SUBWFB DPX+B1          ; get 2nd byte of ACC into w
                  ; sub 2nd byte of DPX, save in DPX(B1)

```

```

00D3 6022      MOVFP  ACC+B2,WREG      ; get 3rd byte of ACC into w
00D4 031E      SUBWFB DPX+B2          ; sub 3rd byte of DPX, save in DPX(B2)
00D5 6023      MOVFP  ACC+B3,WREG      ; get 4th byte of ACC into w
00D6 031F      SUBWFB DPX+B3          ; sub 4th byte of DPX, save in DPX(B3)
                                MOVFP16 DPX+B2,X1      ; X1 = COS*Xt - SIN*Yt, Scale if neces
00D7 5E3D      MOVFP  DPX+B2+B0,X1+B0 ; move DPX+B2(B0) to X1(B0)
00D8 5F3E      MOVFP  DPX+B2+B1,X1+B1 ; move DPX+B2(B1) to X1(B1)
;
;
; Store results of butterfly
;
                                DEC16  tblptr1      ; table pointer already loaded with I
00D9 2900      CLRFB  WREG
00DA 070D      DECF  tblptr1+B0
00DB 030E      SUBWFB tblptr1+B1
00DC A439      tlwt  0,Xi
00DD AF3A      tablwt 1,1,Xi+B1      ; auto increment for Imag Data
00DE A43B      tlwt  0,Yi
00DF AE3C      tablwt 1,0,Yi+B1      ; Xi & Yi stored
                                MOVFP16 VarL,tblptr1      ; read data(L)
00E0 6D37      MOVFP  VarL+B0,tblptr1+B ; move VarL(B0) to tblptr1(B0)
00E1 6E38      MOVFP  VarL+B1,tblptr1+B1 ; move VarL(B1) to tblptr1(B1)
                                ADDLBL  ExtRamAddr,tblptr1      ; add data addr offset
00E2 B000      MOVLW  (ExtRamAddr) & 0xff
00E3 0F0D      ADDWF  tblptr1+B0
00E4 B008      MOVLW  page      (ExtRamAddr)
00E5 110E      ADDWFC tblptr1+B1
00E6 A43D      tlwt  0,X1
00E7 AF3E      tablwt 1,1,X1+B1      ; auto increment for Imag Data
00E8 A43F      tlwt  0,Y1
00E9 AE40      tablwt 1,0,Y1+B1      ; X(L) & Y(L) stored
;
; Increment for next Iloop
;
                                RLC16AB count1,temp      ; temp = count1*2
00EA 8804      BCF  _carry
00EB 1A24      RLCF  count1+B0,W
00EC 0145      MOVWF  temp+B0
00ED 1A25      RLCF  count1+B1,W
00EE 0146      MOVWF  temp+B1
                                ADD16  temp,VarIloop      ; I = I + temp
00EF 6045      MOVFP  temp+B0,WREG      ; get lowest byte of temp into w
00F0 0F32      ADDWF  VarIloop+B0      ; add lowest byte of VarIloop,save in VarIloop(B0)
00F1 6046      MOVFP  temp+B1,WREG      ; get 2nd byte of temp into w
00F2 1133      ADDWFC VarIloop+B1      ; add 2nd byte of VarIloop,save in VarIloop(B1)
;
                                MOVK16 ((FftLen*2),temp
00F3 B000      MOVLW  ((FftLen*2)) & 0xff
00F4 0145      MOVWF  temp+B0
00F5 B002      MOVLW  ((FftLen*2))/256
00F6 0146      MOVWF  temp+B1
                                SUB16  VarIloop,temp      ; temp = 2*FftLen - I
00F7 6032      MOVFP  VarIloop+B0,WREG ; get lowest byte of VarIloop into w
00F8 0545      SUBWF  temp+B0          ; sub lowest byte of temp, save in temp(B0)
00F9 6033      MOVFP  VarIloop+B1,WREG ; get 2nd byte of VarIloop into w
00FA 0346      SUBWFB temp+B1          ; sub 2nd byte of temp, save in temp(B1)
                                DEC16  temp
00FB 2900      CLRFB  WREG
00FC 0745      DECF  temp+B0
00FD 0346      SUBWFB temp+B1
00FE 9746      btfs  temp+B1,MSB
00FF C06A      goto  Iloop          ; while I < 2*FftLen
;
; I Loop end

```

Implementing FFT

```
;
; increment for next J Loop
;

          INCL6   VarJloop           ; J = J + 1
0100 2900      CLRf   WREG
0101 1534      INCf   VarJloop+B0
0102 1135      ADDWFC VarJloop+B1
          MOVl6   count2,temp
0103 6026      MOVFP count2+B0,WREG ; get byte of count2 into w
0104 0145      MOVWF temp+B0        ; move to temp(B0)
0105 6027      MOVFP count2+B1,WREG ; get byte of count2 into w
0106 0146      MOVWF temp+B1        ; move to temp(B1)
          SUBl6   VarJloop,temp      ; temp = count2 - J
0107 6034      MOVFP VarJloop+B0,WREG ; get lowest byte of VarJloop into w
0108 0545      SUBWF temp+B0        ; sub lowest byte of temp, save in temp(B0)
0109 6035      MOVFP VarJloop+B1,WREG ; get 2nd byte of VarJloop into w
010A 0346      SUBWFB temp+B1       ; sub 2nd byte of temp, save in temp(B1)
          DEC16   temp
010B 2900      CLRf   WREG
010C 0745      DECF   temp+B0
010D 0346      SUBWFB temp+B1
010E 9746      btfs  temp+B1,MSB
010F C051      goto   Jloop         ; while J < count2
;
; J Loop end
;
; increment for next K Loop
;
          RLC16   TF_Offset         ; TF_Offset = 2 * TF_Offset
0110 8804      BCF   _carry
0111 1B2A      RLCF   TF_Offset+B0
0112 1B2B      RLCF   TF_Offset+B1
0113 1736      decfsz VarKloop
0114 C046      goto   Kloop        ; while K < Power
;
0115 0002      return              ; FFT complete
;
; K Loop End
; FFT Computation Over with data scrambled
; Descramble the data using "Unscramble" Routine
;
;*****
;      Unscramble Data Order Sequence
;      A digit reverse counter
;*****
include "reverse.asm"
;*****
;      A digit reverse counter
;
;      Unscramble Data Order Sequence Of Radix-2 FFT
; Length (must be a power of 2) is limited only by
; the amount of External RAM available and must be
; a number less than 2**15
;
;*****
Unscramble
          CLR16   VarJloop           ; J = 0
0116 2934      CLRf   VarJloop+B0
0117 2935      CLRf   VarJloop+B1
0118 2933      clrf  VarIloop+B1
          MOVK    1,VarIloop        ; I = 1
0119 B001      MOVLW 1
011A 0132      MOVWF VarIloop
nextI
          MOVK16  FftLen/2,VarKloop
011B B080      MOVLW (FftLen/2) & 0xff
011C 0136      MOVWF VarKloop+B0
011D B000      MOVLW (FftLen/2)/256
```

```

011E 0137          MOVWF  VarKloop+B1
011F C127          goto   testK

                KlessJ

                SUB16  VarKloop,VarJloop ; J = J - K
0120 6036          MOVFP  VarKloop+B0,WREG ; get lowest byte of VarKloop into w
0121 0534          SUBWF  VarJloop+B0 ; sub lowest byte of VarJloop,save in VarJloop(B0)
0122 6037          MOVFP  VarKloop+B1,WREG ; get 2nd byte of VarKloop into w
0123 0335          SUBWFB VarJloop+B1 ; sub 2nd byte of VarJloop,save in VarJloop(B1)
                RRC16  VarKloop ; K = K/2
0124 1A37          RLCF  VarKloop+B1,W ; move sign into carry bit
0125 1937          RRCF  VarKloop+B1
0126 1936          RRCF  VarKloop+B0

                testK

                MOV16  VarJloop,temp

0127 6034          MOVFP  VarJloop+B0,WREG ; get byte of VarJloop into w
0128 0145          MOVWF  temp+B0 ; move to temp(B0)
0129 6035          MOVFP  VarJloop+B1,WREG ; get byte of VarJloop into w
012A 0146          MOVWF  temp+B1 ; move to temp(B1)
                SUB16  VarKloop,temp ; temp = J - K
012B 6036          MOVFP  VarKloop+B0,WREG ; get lowest byte of VarKloop into w
012C 0545          SUBWF  temp+B0 ; sub lowest byte of temp, save in temp(B0)
012D 6037          MOVFP  VarKloop+B1,WREG ; get 2nd byte of VarKloop into w
012E 0346          SUBWFB temp+B1 ; sub 2nd byte of temp, save in temp(B1)
012F 9746          btfs  temp+B1,MSB
0130 C120          goto   KlessJ ; while K < J
                ADD16  VarKloop,VarJloop ; J = J + K
0131 6036          MOVFP  VarKloop+B0,WREG ; get lowest byte of VarKloop into w
0132 0F34          ADDWF  VarJloop+B0 ; add lowest byte of VarJloop,save in VarJloop(B0)
0133 6037          MOVFP  VarKloop+B1,WREG ; get 2nd byte of VarKloop into w
0134 1135          ADDWFC VarJloop+B1 ; add 2nd byte of VarJloop,save in VarJloop(B1)
;
; if (i < j) then swap data(i) & data(j)
;
                MOV16  VarJloop,temp
0135 6034          MOVFP  VarJloop+B0,WREG ; get byte of VarJloop into w
0136 0145          MOVWF  temp+B0 ; move to temp(B0)
0137 6035          MOVFP  VarJloop+B1,WREG ; get byte of VarJloop into w
0138 0146          MOVWF  temp+B1 ; move to temp(B1)
                SUB16  VarIloop,temp ; temp = J - I
0139 6032          MOVFP  VarIloop+B0,WREG ; get lowest byte of VarIloop into w
013A 0545          SUBWF  temp+B0 ; sub lowest byte of temp, save in temp(B0)
013B 6033          MOVFP  VarIloop+B1,WREG ; get 2nd byte of VarIloop into w
013C 0346          SUBWFB temp+B1 ; sub 2nd byte of temp, save in temp(B1)
                DEC16  temp
013D 2900          CLRf  WREG
013E 0745          DECF  temp+B0
013F 0346          SUBWFB temp+B1
0140 9F46          btfs  temp+B1,MSB
0141 C174          goto   inci

;
; swap data
; read data(i)
                RLC16AB VarIloop,tblptrl ; add twice the addr, since Real Data
0142 8804          BCF  _carry
0143 1A32          RLCF  VarIloop+B0,W
0144 010D          MOVWF  tblptrl+B0
0145 1A33          RLCF  VarIloop+B1,W
0146 010E          MOVWF  tblptrl+B1
                ADDLBL  ExtRamAddr,tblptrl ; is followed by Imag Data
0147 B000          MOVLW  (ExtRamAddr) & 0xff
0148 0F0D          ADDWF  tblptrl+B0
0149 B008          MOVLW  page (ExtRamAddr)
014A 110E          ADDWFC tblptrl+B1
014B A939          tablrd 0,1,Xi ; auto increment for Imag Data
014C A039          tlrld 0,Xi
014D A23A          tlrld 1,Xi+B1 ; real data XI
014E A83B          tablrd 0,0,Yi
014F A03B          tlrld 0,Yi

```

Implementing FFT

```
0150 A23C          tlrld    1,Yi+B1 ; imag data YI
;
; read data(j)
;
          RLC16AB VarJloop,tblptr1 ; add twice the addr, since Real Data
0151 8804          BCF     _carry
0152 1A34          RLCF    VarJloop+B0,W
0153 010D          MOVWF   tblptr1+B0
0154 1A35          RLCF    VarJloop+B1,W
0155 010E          MOVWF   tblptr1+B1
          ADDLBLE ExtRamAddr,tblptr1 ; is followed by Imag Data
0156 B000          MOVLW   (ExtRamAddr) & 0xff
0157 0F0D          ADDWF   tblptr1+B0
0158 B008          MOVLW   page (ExtRamAddr)
0159 110E          ADDWFC  tblptr1+B1
015A A93D          tablrd  0,1,X1 ; auto increment for Imag Data
015B A03D          tlrld   0,X1
015C A23E          tlrld   1,X1+B1 ; real data XL
015D A83F          tablrd  0,0,Y1
015E A03F          tlrld   0,Y1
015F A240          tlrld   1,Y1+B1 ; imag data YL
;
; Interchange data(I) & data(J)
;
; J addr already loaded into table pointers, bu autoincremented
;
          DEC16   tblptr1
0160 2900          CLRF    WREG
0161 070D          DECF    tblptr1+B0
0162 030E          SUBWFB  tblptr1+B1
0163 A439          tlwt    0,Xi
0164 AF3A          tablw   1,1,Xi+B1 ; auto increment for Imag Data
0165 A43B          tlwt    0,Yi
0166 AE3C          tablw   1,0,Yi+B1 ; X(I) & Y(I) stored
          RLC16AB VarIloop,tblptr1 ; add twice the addr, since Real Data
0167 8804          BCF     _carry
0168 1A32          RLCF    VarIloop+B0,W
0169 010D          MOVWF   tblptr1+B0
016A 1A33          RLCF    VarIloop+B1,W
016B 010E          MOVWF   tblptr1+B1
          ADDLBLE ExtRamAddr,tblptr1 ; is followed by Imag Data
016C B000          MOVLW   (ExtRamAddr) & 0xff
016D 0F0D          ADDWF   tblptr1+B0
016E B008          MOVLW   page (ExtRamAddr)
016F 110E          ADDWFC  tblptr1+B1
0170 A43D          tlwt    0,X1
0171 AF3E          tablw   1,1,X1+B1 ; auto increment for Imag Data
0172 A43F          tlwt    0,Y1
0173 AE40          tablw   1,0,Y1+B1 ; X(L) & Y(L) stored
;
; increment I
;
incI
          INC16   VarIloop
0174 2900          CLRF    WREG
0175 1532          INCF    VarIloop+B0
0176 1133          ADDWFC  VarIloop+B1
          MOVK16  DigitRevCount,temp
0177 B0EF          MOVLW   (DigitRevCount) & 0xff
0178 0145          MOVWF   temp+B0
0179 B000          MOVLW   (DigitRevCount)/256
017A 0146          MOVWF   temp+B1
          SUB16   VarIloop,temp ; temp = DigitRevCount - I
017B 6032          MOVFP   VarIloop+B0,WREG ; get lowest byte of VarIloop into w
017C 0545          SUBWF   temp+B0 ; sub lowest byte of temp, save in temp(B0)
017D 6033          MOVFP   VarIloop+B1,WREG ; get 2nd byte of VarIloop into w
017E 0346          SUBWFB  temp+B1 ; sub 2nd byte of temp, save in temp(B1)
017F 9746          btfs   temp+B1,MSB
0180 C11B          goto    nextI ; while i < DigitRevCount
```

```

0181 0002                return
;
; End digit reverse counter
;
;*****

;*****
;          Include Double Precision Multiplication Routine
;*****
0001    SIGNED    equ    TRUE

                include "l7c42mpy.mac"

;
;*****
;          Sine-Cosine Tables
;*****
;
                include "fft256.tbl"
;
;          256 Point FFT Sine Table
; coefficient table (size of table is 3n/4).
;
SineTable
;
0294 0000                data        0
0295 0324                data        804
0296 0648                data        1608
0297 096A                data        2410
0298 0C8C                data        3212
0299 0FAB                data        4011
029A 12C8                data        4808
029B 15E2                data        5602
029C 18F9                data        6393
029D 1C0B                data        7179
029E 1F1A                data        7962
029F 2223                data        8739
02A0 2528                data        9512
02A1 2826                data        10278
02A2 2B1F                data        11039
02A3 2E11                data        11793
02A4 30FB                data        12539
02A5 33DF                data        13279
02A6 36BA                data        14010
02A7 398C                data        14732
02A8 3C56                data        15446
02A9 3F17                data        16151
02AA 41CE                data        16846
02AB 447A                data        17530
02AC 471C                data        18204
02AD 49B4                data        18868
02AE 4C3F                data        19519
02AF 4EBF                data        20159
02B0 5133                data        20787
02B1 539B                data        21403
02B2 55F5                data        22005
02B3 5842                data        22594
02B4 5A82                data        23170
02B5 5CB3                data        23731
02B6 5ED7                data        24279
02B7 60EB                data        24811
02B8 62F1                data        25329
02B9 64E8                data        25832
02BA 66CF                data        26319
02BB 68A6                data        26790
02BC 6A6D                data        27245

```

Implementing FFT

```
02BD 6C23          data          27683
02BE 6DC9          data          28105
02BF 6F5E          data          28510
02C0 70E2          data          28898
02C1 7254          data          29268
02C2 73B5          data          29621
02C3 7504          data          29956
02C4 7641          data          30273
02C5 776B          data          30571
02C6 7884          data          30852
02C7 7989          data          31113
02C8 7A7C          data          31356
02C9 7B5C          data          31580
02CA 7C29          data          31785
02CB 7CE3          data          31971
02CC 7D89          data          32137
02CD 7E1D          data          32285
02CE 7E9C          data          32412
02CF 7F09          data          32521
02D0 7F61          data          32609
02D1 7FA6          data          32678
02D2 7FD8          data          32728
02D3 7FF5          data          32757
;
CosTable
;
02D4 7FFF          data          32767
02D5 7FF5          data          32757
02D6 7FD8          data          32728
02D7 7FA6          data          32678
02D8 7F61          data          32609
02D9 7F09          data          32521
02DA 7E9C          data          32412
02DB 7E1D          data          32285
02DC 7D89          data          32137
02DD 7CE3          data          31971
02DE 7C29          data          31785
02DF 7B5C          data          31580
02E0 7A7C          data          31356
02E1 7989          data          31113
02E2 7884          data          30852
02E3 776B          data          30571
02E4 7641          data          30273
02E5 7504          data          29956
02E6 73B5          data          29621
02E7 7254          data          29268
02E8 70E2          data          28898
02E9 6F5E          data          28510
02EA 6DC9          data          28105
02EB 6C23          data          27683
02EC 6A6D          data          27245
02ED 68A6          data          26790
02EE 66CF          data          26319
02EF 64E8          data          25832
02F0 62F1          data          25329
02F1 60EB          data          24811
02F2 5ED7          data          24279
02F3 5CB3          data          23731
02F4 5A82          data          23170
02F5 5842          data          22594
02F6 55F5          data          22005
02F7 539B          data          21403
02F8 5133          data          20787
02F9 4EBF          data          20159
02FA 4C3F          data          19519
02FB 49B4          data          18868
02FC 471C          data          18204
02FD 447A          data          17530
02FE 41CE          data          16846
```


02FF	3F17	data	16151
0300	3C56	data	15446
0301	398C	data	14732
0302	36BA	data	14010
0303	33DF	data	13279
0304	30FB	data	12539
0305	2E11	data	11793
0306	2B1F	data	11039
0307	2826	data	10278
0308	2528	data	9512
0309	2223	data	8739
030A	1F1A	data	7962
030B	1C0B	data	7179
030C	18F9	data	6393
030D	15E2	data	5602
030E	12C8	data	4808
030F	0FAB	data	4011
0310	0C8C	data	3212
0311	096A	data	2410
0312	0648	data	1608
0313	0324	data	804
0314	0000	data	0
0315	FCDC	data	-804
0316	F9B8	data	-1608
0317	F696	data	-2410
0318	F374	data	-3212
0319	F055	data	-4011
031A	ED38	data	-4808
031B	EA1E	data	-5602
031C	E707	data	-6393
031D	E3F5	data	-7179
031E	E0E6	data	-7962
031F	DDDD	data	-8739
0320	DAD8	data	-9512
0321	D7DA	data	-10278
0322	D4E1	data	-11039
0323	D1EF	data	-11793
0324	CF05	data	-12539
0325	CC21	data	-13279
0326	C946	data	-14010
0327	C674	data	-14732
0328	C3AA	data	-15446
0329	C0E9	data	-16151
032A	BE32	data	-16846
032B	BB86	data	-17530
032C	B8E4	data	-18204
032D	B64C	data	-18868
032E	B3C1	data	-19519
032F	B141	data	-20159
0330	AECD	data	-20787
0331	AC65	data	-21403
0332	AA0B	data	-22005
0333	A7BE	data	-22594
0334	A57E	data	-23170
0335	A34D	data	-23731
0336	A129	data	-24279
0337	9F15	data	-24811
0338	9D0F	data	-25329
0339	9B18	data	-25832
033A	9931	data	-26319
033B	975A	data	-26790
033C	9593	data	-27245
033D	93DD	data	-27683
033E	9237	data	-28105
033F	90A2	data	-28510
0340	8F1E	data	-28898
0341	8DAC	data	-29268
0342	8C4B	data	-29621
0343	8AFC	data	-29956

Implementing FFT

```
0344 89BF          data          -30273
0345 8895          data          -30571
0346 877C          data          -30852
0347 8677          data          -31113
0348 8584          data          -31356
0349 84A4          data          -31580
034A 83D7          data          -31785
034B 831D          data          -31971
034C 8277          data          -32137
034D 81E3          data          -32285
034E 8164          data          -32412
034F 80F7          data          -32521
0350 809F          data          -32609
0351 805A          data          -32678
0352 8028          data          -32728
0353 800B          data          -32757
;
;*****
;
;*****
;          FFT Input/Output Data Stored In External RAM
; Operate Processor In Extended Microcontroller Mode
; External Data Starts at Address 0x0800, with 2 bytes of
; Real Data followed by 2 bytes of Imaginary Data.
;*****
;          ORG          EXT_RAM_START_ADDR
;
ExtRamAddr
;
;          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

All rights reserved. © 1995, Microchip Technology Incorporated, USA.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights. The Microchip logo and name are registered trademarks of Microchip Technology Inc. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.