TB3141

Implementing the Fast Fourier Transform (FFT) on dsPIC® Digital Signal Controllers

Author: *Fergus O'Kane*
Microchip Technology Inc.

INTRODUCTION

The Microchip dsPIC® Digital Signal Controllers (DSC) have unique features for implementing DSP algorithms, such as the Fast Fourier Transform (FFT) in embedded systems. This Technical Brief explains how to implement the FFT on dsPIC® DSCs using the DSP library supplied with the MPLAB® XC16 C Compiler.

FFT LIBRARY FUNCTIONS

The MPLAB® XC16 C Compiler provides a DSP library with functions for vector math, matrix math, digital filters, window and transform functions, including the Fast Fourier Transform (FFT). Most of the library functions are written in optimized assembly to be as efficient as possible. The functions also make use of the hardware DSP features of the dsPIC® DSC controllers.

When using the DSP library in a project, the user must include the header file, `dsp.h`, and add the library file, `libdsp-elf.a` (or `libdsp-coff.a` for a project using the COFF debug file format), to the project. Both files are supplied with the compiler.

The DSP library function for implementing the 16-bit FFT is `FFTComplexIP()`.

EXAMPLE 1: `FFTComplexIP()`

```
fractcomplex* FFTComplexIP (
   int log2N,
   fractcomplex* srcCV,
   fractcomplex* twidFactors,
   int factPage
);
```

`log2N` informs the function of what size FFT to use (e.g., `log2N = 8` implies 2^8 = 256-point FFT). `srcCV` is a pointer to an array containing the input data. `twidFactors` is a pointer to an array containing the twiddle factors used by the FFT and `factPage` is an address of the page in memory containing the twiddle factors if they are stored in program memory. If `twidFactors` is stored in RAM instead of program memory, then `factPage` should be set to 0xFF00 (defined as `COEFFS_IN_DATA` in the `dsp.h` file).

When using twiddle factors stored in program memory and accessed using PSV, the compiler functions, `__builtin_psvoffset()` and `__builtin_psvpage()`, can be used to get the address and page information needed for the last two parameters of the `FFTComplexIP()` function. An example is shown in Example 2.

EXAMPLE 2: EXECUTING THE FFT

```
// using FFTComplexIP with twiddle factors stored
// in flash program memory
FFTComplexIP(LOG2N, (fractcomplex*)&fftBuffer[0],
(fractcomplex*)__builtin_psvoffset
(&psvTwiddleFactors[0]),
__builtin_psvpage(&psvTwiddleFactors[0]));
```

The data type, `fractcomplex`, is defined in header file, `dsp.h`. The type definition is shown in Example 3.

EXAMPLE 3: `fractcomplex` DATA TYPE

```
typedef    struct {
fractional real;
fractional imag;
} fractcomplex;
```

A `fractcomplex` variable, therefore, is simply a structure containing two members of type fractional, which represent the real and imaginary components of a data point. A variable of type, `fractcomplex`, can be declared and accessed as follows:

EXAMPLE 4: USING `fractcomplex`

```
// declare variable 'inputData' to be type /
// fractcomplex
fractcomplex   inputData;

// access members of 'inputData'
inputData.real = Q15(0.5);
inputData.imag = Q15(0.25);
```

In this example, `inputData` represents a fractional complex number: 0.5 + 0.25j. Note the imaginary operator j is not present in the code or values used, but is implied in the variable being the imaginary component.

The FFTComplexIP() function expects both the input data and twiddle factors to be in a fractcomplex array and returns a pointer to the fractcomplex array containing the FFT output. This function performs the FFT in-place, which means the FFT output is placed in the same array that contained the input data (i.e., when the function completes, the output data has overwritten the input data). Performing the FFT operation in-place saves RAM, as only one array is needed for the input and output data, instead of two. However, there is a version of the function, if needed, which does not compute the result in-place, called FFTComplex().

Note that the FFTComplexIP()/FFTComplex() function has a requirement that input data be in the range, -0.5 to +0.5, to prevent overflow during the operation. If the data range is higher, it can be scaled by shifting the data to the right. The DSP library has a VectorScale() function that can be used for data scaling.

The complementary Inverse FFT (IFFT) library function is IFFTComplexIP().

EXAMPLE 5: IFFTComplexIP()

```
fractcomplex* IFFTComplexIP (
   int log2N,
   fractcomplex* srcCV,
   fractcomplex* twidFactors,
   int factPage
);
```

The usage is the same as the FFTComplexIP() function, except that the twiddle factors are different for the FFT and IFFT.

Twiddle Factors

Twiddle factors can be generated using the DSP library function, TwidFactorInit().

EXAMPLE 6: TwidFactorInit()

```
fractcomplex* TwidFactorInit (
   int log2N,
   fractcomplex* twidFactors,
   int conjFlag
);
```

The first argument of the function, log2N, informs the function of what size FFT is being used (e.g., log2N = 8 implies 2^8 = 256-point FFT). twidFactors is a pointer to an array which is used to store the twiddle factors and conjFlag is a value that tells the function to generate twiddle factors for either the FFT or Inverse FFT (conjFlag = 0 for FFT, conjFlag = 1 for IFFT).

An example of using this function is shown below.

EXAMPLE 7: GENERATING TWIDDLE FACTORS

```
TwidFactorInit(LOG2N,&twiddleFactors[0], 0);
```

Where twiddleFactors is an array of type, fractcomplex, defined by the user, only N/2 twiddle factors are needed for an FFT.

The twiddle factors generated by this function are stored in an array in RAM. The array must be in X-memory. Since the twiddle factor values only need to be generated once and will not change, they could be precalculated beforehand by the user and stored in program memory. This would be appropriate if RAM is limited in the application and needed for other purposes.

To store precalculated twiddle factors in program memory, an array in PSV memory can be declared. This allows the dsPIC® DSC instructions to access the data in program memory as if it were in RAM.

EXAMPLE 8: TWIDDLE FACTORS IN PROGRAM MEMORY

```
const fractcomplex twiddleFactors[FFT_SIZE/2]
__attribute__((space(auto_psv)))=
{
        //.. enter twiddle factor values
here
};
```

The twiddle factors are of type, fractcomplex, containing both real and imaginary components.

Bit Reversal

The radix-2 Decimation-In-Frequency (DIF) FFT algorithm changes the order of the data during processing and a bit reversal function must be called to re-order the data afterwards.

The BitReverseComplex() DSP library function handles this bit-reversed sorting of data (the function parameters, log2N and srcCV, are as previously explained for the other functions).

EXAMPLE 9: BitReverseComplex()

```
extern fractcomplex* BitReverseComplex (
   int log2N
   fractcomplex* srcCV
);
```

Note that the hardware support for Bit-Reversed Addressing in the dsPIC® DSC used by this function requires that srcCV be aligned in a certain way in memory. The address of the array must be a multiple of the array size in bytes. For example, a 512-point FFT having a fractcomplex array of 512 elements, with each fractcomplex element being two words (four bytes), would require srcCV to be aligned to a 512x2 = 1024-word or 2048-byte boundary. More information on alignment and placing data in memory is explained later.

Complex Squared Magnitude

The most useful way to view the Fourier Transform output is as the squared magnitude. After the bit-reversed operation is complete, the real and imaginary output of the FFT is stored in the same `fractcomplex` array on which the FFT was performed. The DSP library function, `SquareMagnitudeCplx()`, can compute the squared magnitude from this data.

EXAMPLE 10: `SquareMagnitudeCplx`

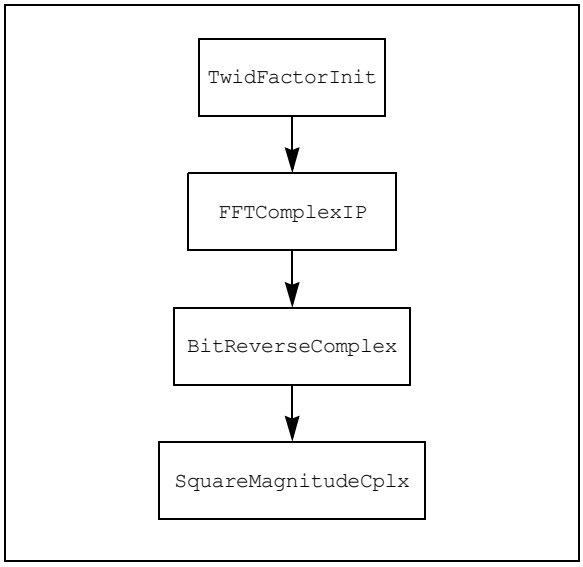
```
extern fractional* SquareMagnitudeCplx (
   int numElems,
   fractcomplex* srcV,
   fractional* dstV
);
```

The argument, `numElems`, is the number of elements in the input array, which should be the same as the FFT size, `srcV` is the input array and `dstV` is an array of type fractional to store the result.

Summary of FFT Library Functions

A summary of the 16-bit FFT library functions and the order in which they are used is shown in Figure 1. The `TwidFactorInit()` function is only needed if the twiddle factors are to be created during run time and stored in RAM.

FIGURE 1: FFT FUNCTION CALLS



Placing Data In Memory

The FFT library functions require data to be placed appropriately in memory for correct operation. The functions do not check for this correct placement as it would increase the execution time. It is up to the user to ensure data placement is correct. The `FFTComplexIP()` function requires the twiddle factors to be in X-memory or to be stored in a PSV section, which the compiler will map into X-memory. Similarly, the `TwidFactorInit()` function requires the `twidFactors` array to be in X-memory. The source vector, `srcV`, for the `FFTComplexIP()` function must be located in Y-memory. Having the twiddle factors in X-memory and the input data in Y-memory allows the dsPIC® DSC to fetch data from both memory blocks simultaneously. As previously explained, the `BitReverseComplex()` function requires data to be specially aligned according to the input array size. The input array used in `BitReverseComplex()` should be the same as the array used in `FFTComplexIP()`, so it must be in Y-memory. The `SquareMagnitudeComplex()` function would use the same array as the `BitReverseComplex()` and `FFTComplexIP()` functions, so this would have the same properties mentioned already.

To summarize, the array containing the twiddle factors must be either in X-memory in RAM or in a PSV section if stored in Flash program memory, and the array containing the input data/FFT result, which is used in functions, `FFTComplexIP()`, `BitReverseComplex()` and `SquareMagnitudeComplex()`, must be in Y-memory, and aligned to a multiple of the array size in bytes.

The MPLAB® XC16 C Compiler provides attributes that can be applied to variables to control where they are placed in memory. Relevant examples are provided in Example 11.

EXAMPLE 11: PLACING VARIABLES IN MEMORY

```
/* place array fftBuffer in Y-memory aligned
to the size of the array in bytes */
fractcomplex fftBuffer[FFT_SIZE]
__attribute__((space(ymemory),aligned
(FFT_SIZE*2*2)));

// place array twiddleFactors in X-memory
fractcomplex twiddleFactors[FFT_SIZE/2]
__attribute__((space(xmemory)));
```

Note that the position of the Y-memory block in RAM may be different depending on the dsPIC® DSC device used. The first 8 Kbytes of RAM, from address 0x0000 to 0x1FFF, is called “near memory”. Depending on the device, Y-memory may be within this 8-Kbyte near memory range or may be outside of it. The MPLAB XC16 C Compiler tries to place variables in near memory by default, known as the small data memory model. If addresses higher than 0x2000 are needed, or if Y-memory is above 0x2000, then the compiler large data memory model must be used, or the far attribute must be applied to the variable(s) which will be placed outside of near memory. The memory model option can be selected in the compiler settings. More detailed information on the memory models and attributes can be found in the compiler user guide.

EXAMPLE 12: FAR MEMORY ATTRIBUTE

```
/* place array fftBuffer in Y-memory aligned
to the size of the array in bytes */
fractcomplex fftBuffer[FFT_SIZE]
__attribute__((space(ymemory), far, aligned
(FFT_SIZE*2*2)));
```

Furthermore, note that the dsPIC33E families of DSCs have a feature known as Extended Data Space (EDS), which must be taken into account. These devices have more RAM than the dsPIC33F families and use EDS to access the higher addresses. On some dsPIC33E devices, the Y-memory block may be above address, 0x7FFF. If it is, the EDS attribute must be applied to the variables to be located in Y-memory. Without the correct attribute, a linker error will occur when compiling the code.

EXAMPLE 13: EDS MEMORY ATTRIBUTE

```
/* place array fftBuffer in Y-memory aligned
to the size of the array in bytes */
fractcomplex fftBuffer[FFT_SIZE]
__attribute__((space(ymemory), eds, aligned
(FFT_SIZE*2*2)));
```

FFT Output

It is often needed to find the maximum frequency component in the output spectrum. To find this frequency, the data from the squared magnitude operation can be checked to find the frequency bin number where the maximum value occurs. The DSP library includes a `VectorMax()` function which can be used for this. Once the bin number is found, it can be translated to a frequency by using the formula in Equation 1. The sampling frequency, divided by the FFT size, is the resolution of the FFT, so the operation is essentially multiplying the frequency bin number by the FFT resolution. Note that since the FFT output is mirrored, only the first FFT_SIZE/2 elements of the output array need to be checked.

EQUATION 1: CALCULATE MAXIMUM FREQUENCY

$$F_{MAX} = bin \bullet (F_{SAMP}/FFT\_SIZE)$$

Where:

F_{MAX} is the maximum frequency component to be found

bin is the frequency bin number where the maximum value occurs

F_{SAMP} is the sampling frequency

FFT_SIZE is the size of the FFT

EQUATION 2: CALCULATION FOR BIN NUMBER 3

$$F_{MAX} = 3 \bullet (8192\ Hz/256) = 96\ Hz$$

As an example, considering an input signal sampled at 8,192 Hz, the FFT size is 256 points and the bin number where the maximum value occurs is 3, the resulting frequency would be 96 Hz.

EXAMPLE 14: FINDING MAXIMUM FREQUENCY COMPONENT FROM FFT OUTPUT

```
// variable definitions
fractional   squaredMagnitude[FFT_SIZE];
fractional   fftMaxValue;
int16_t      fftMaxValueBin;
uint16_t     peakFrequencyHz;

// find the max value in the magnitude vector and /
// which bin it is in
fftMaxValue = VectorMax(FFT_SIZE/2,
(fractional*)&squaredMagnitude[0],
(int16_t*)&fftMaxValueBin);

// Compute the frequency (in Hz) of the largest //
// spectral component
peakFrequencyHz = fftMaxValueBin *
(SAMPLING_FREQUENCY_HZ / FFT_SIZE);
```

CONCLUSION

The unique DSP features of the dsPIC® Digital Signal Controllers make them a good fit for applications which require an FFT. The DSP library supplied with the MPLAB® XC16 C Compiler makes it easy for the developer to use an FFT in their application.

REFERENCES

- *"dsPIC33E/PIC24E Family Reference Manual"*, **Section 2. "CPU"** (DS70359)
- *"16-Bit Language Tools Libraries Reference Manual"* (DS50001456)
- *"MPLAB® XC16 C Compiler User's Guide"* (DS50002071)
- *MPLAB® XC16 C Compiler DSP Library Help file*

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

ISBN: 978-1-63277-683-9

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Dusseldorf
Tel: 49-2129-3766400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Venice
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Poland - Warsaw
Tel: 48-22-3325737

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

07/14/15