

# **Performing the Fast Fourier Transform with Microchip's dsPIC30F Series Digital Signal Controllers**

## **Application Note**

**Michigan State University  
Dept. of Electrical & Computer Engineering**

Author: Nicholas T Vogtmann

Friday, November 7<sup>th</sup>, 2008

### **Abstract**

This application note describes the basics of the Fast Fourier Transform and how to use a Digital Signal Processing Library. The library and design example use Microchip's dsPIC30F Series Digital Signal Controller, specifically the dsPIC30F3013. The Transform Analysis discusses the differences between the Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT), and how the FFT is generated easily using the C programming language.

### **Keywords**

Discrete Fourier Transform, DFT, Fast Fourier Transform, FFT, Radix-2, Decimation in Time, DIT, Digital Signal Controller, DSP, C Programming Language, Analog to Digital Converter, ADC, Microchip, dsPIC30F

## Introduction

The C Programming Language code examples included demonstrates the design of an embedded system that captures an audio signal from a microphone or signal generator and stores the 128-bit data stream into digital memory. Once the information is stored, the Digital Signal Processing Unit performs a 128-bit Fast Fourier Transform to locate the most predominate frequency. This signal is then classified into a low, middle, or high frequency range for the purpose of illuminating a corresponding Light Emitting Diode (LED). When the peak frequency is above 800 Hz, the microcontroller illuminates the Green LED. Likewise, when the peak frequency is between 400 Hz and 800 Hz, the microcontroller illuminates the Yellow LED. Otherwise, the microcontroller illuminates the Red LED. A block diagram of the system can be seen in *Figure 1*.

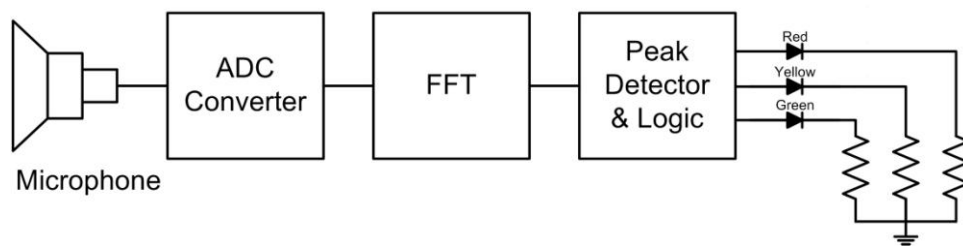


*Figure 1: Block Diagram*

## Background

Two main characteristics of audio signals are the frequency or frequencies of the sinusoidal wave(s) and the amplitude associated with them. Direct Current (DC) has a frequency of 0 Hz, and therefore the only information that can be analyzed is its voltage and current. With an Alternating Current (AC) signal such as a sound wave, the Voltage and Current change periodically and therefore renders a voltage or current at a specific point in time useless. The characteristics of a sound wave can be calculated if a sample of the sound is obtained that includes a full period of the lowest frequency present in the wave. The generic method of calculating frequencies within a signal is called Fourier analysis.

Deepening on the given signal, one of four different Fourier transforms is used to calculate the amplitude of the signal in regards to frequency, also known as the frequency domain. The four different signal types are Continuous & Periodic, Continuous & Aperiodic, Discrete & Periodic, and Discrete & Aperiodic. A continuous signal has a measurement for every point in time, where as a discrete signal only has measurements at predefined intervals. Since any digital system cannot store a continuous signal, the sound wave is sampled to create a discrete signal. A periodic signal repeats its original values after every T seconds in time, where as an aperiodic signal does not repeat any of its values in any particular sequence. Because the time that the audio signal will be captured makes the aperiodic effects from dampening insignificant, the microcontroller can calculate the signal as if it were a periodic signal. Therefore, the resulting transform that the Microchip dsPIC30F3013 will use is called the Discrete Fourier Transform.

The Discrete Fourier Transform shown in *Figure II (a)* can be simplified to the equations in *Figure II (b)* because the input signal is a function of only real numbers. These equations require the computation of  $2N^2$  multiplication and  $2N^2-2N$  addition functions. For 128 samples, this results in 32,768 multiplication and 32,512 addition functions. The Fast Fourier Transform a general term for a simplification upon the DFT and many different algorithms exist to accomplish this task. The most common algorithm is the radix-2 algorithm, which composes the DFT into smaller DFTs. The specific derivation of this algorithm, the radix-2 decimation in time (DIT), takes  $(N/2)\log_2(N)$  multiplication functions and  $(N)\log_2(N)$  addition functions, which brings the number of calculations for the 128-bit FFT down to 448 multiplication functions and 896 addition functions. There is further simplification, but comes at a cost of a more robust microprocessor for an increased computation speed.

$$X(f) = \sum_{t=0}^{N-1} x(t) e^{-j \frac{2\pi f t}{N}}, \text{ for } 0 \leq f \leq N-1 \quad (a)$$

$$\text{Re}\{X(f)\} = \sum_{t=0}^{N-1} \left[ \text{Re}\{x(t)\} \cos\left(\frac{2\pi f t}{N}\right) \right] \quad \text{Im}\{X(f)\} = \sum_{t=0}^{N-1} \left[ \text{Re}\{x(t)\} \sin\left(\frac{2\pi f t}{N}\right) \right] \quad (b)$$

*Figure II: (a) Discrete Fourier Transform (b) Discrete Fourier Transform with real input*

The radix-2 algorithm first rearranges the bits within the array of values, as shown in *Table I* for a 16-bit FFT. This arrangement can be achieved during the ADC stage and prevents rearrangement of values after computation. The remaining operations on the samples originate from one type of computation, called the butterfly computation. This computation acquired its name from the cross that represents the flow of additions and subtractions that compute its result, as seen in *Figure III*. One of the major features of the DIT is that there are half as many multiplications by prioritizing the multiplication, called a twiddle factor, on the second value,  $x[b]$ .

| inputArray[x] |        | bitReversalArray[x] |         |
|---------------|--------|---------------------|---------|
| decimal       | binary | binary              | decimal |
| [0]           | 0000   | 0000                | [0]     |
| [1]           | 0001   | 1000                | [8]     |
| [2]           | 0010   | 0100                | [4]     |
| [3]           | 0011   | 1100                | [12]    |
| [4]           | 0100   | 0010                | [2]     |
| [5]           | 0101   | 1010                | [10]    |
| [6]           | 0110   | 0110                | [6]     |
| [7]           | 0111   | 1110                | [14]    |
| [8]           | 1000   | 0001                | [1]     |
| [9]           | 1001   | 1001                | [9]     |
| [10]          | 1010   | 0101                | [7]     |
| [11]          | 1011   | 1101                | [13]    |
| [12]          | 1100   | 0011                | [3]     |
| [13]          | 1101   | 1011                | [11]    |
| [14]          | 1110   | 0111                | [7]     |
| [15]          | 1111   | 1111                | [15]    |

*Table I: Input bit reversal for correct order of FFT output*

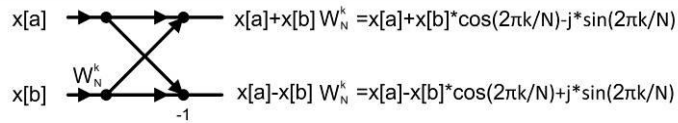


Figure III: Butterfly Computation

These butterfly computations are computed in stages, as shown in Figure IV. All of these stages allow every sample to contribute to its final value as the crosses span across all of the samples. The unique way ordering these calculations allows for reduced memory space because prior values of the same row can be overwritten with its new value because its prior value will not be used again.

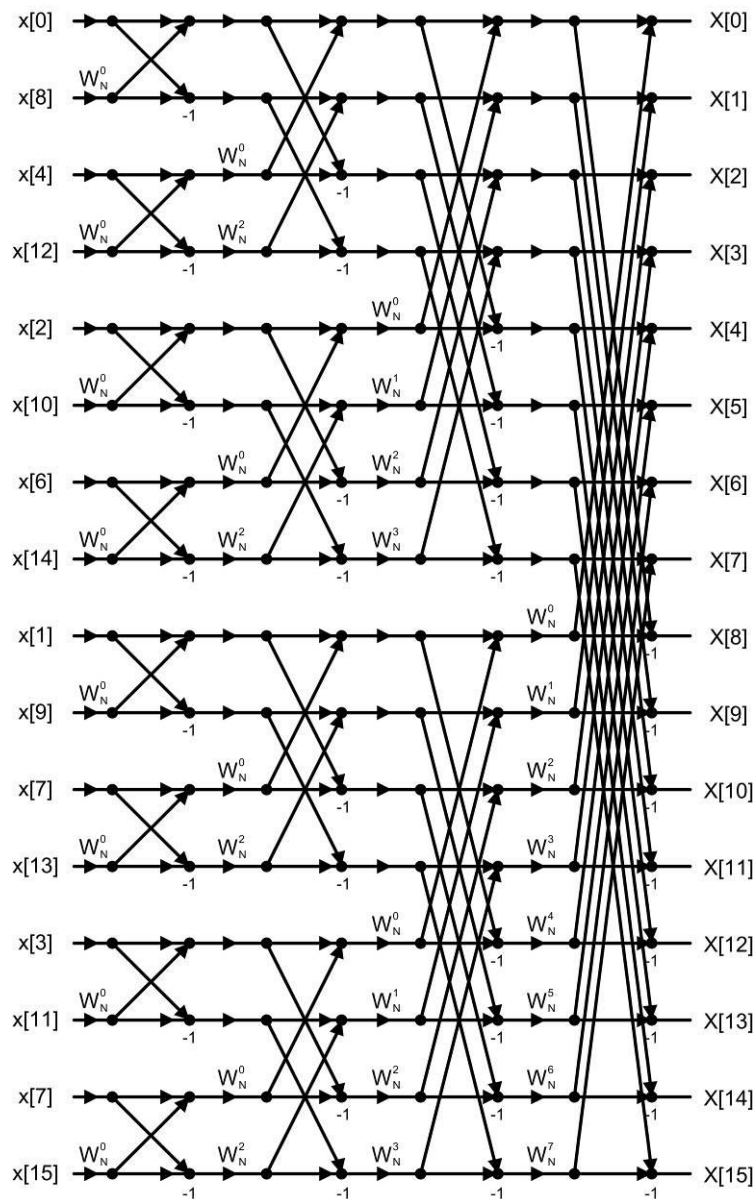


Figure IV: Radix-2 Decimation in Time Algorithm

## Procedure

To concentrate on the calculation of the FFT, the following C language code examples will give a starting point on how to implement the FFT on Microchip's dsPIC30F series Digital Signal Controllers. The analog data can be captured by the ADC or represented by a *fractcomplex* array that includes the samples. Using the predefined array, the code template in *Figure V* can be used. This 16 sample array has 32 indexes. The first 16 are the real part of the samples and the second 16 are the imaginary part of the samples, which will be zero values because all ADC captured data is only real numbers. The twiddle factors also need to be programmed into the microcontroller. These twiddle factors are also *fractcomplex* but will vary depending on what sample size is used.

```
fractcomplex sigCmpx[<Num of Samples>] __attribute__((section
(".ydata, data, ymemory"), aligned (<Num of Samples> * 2 *2))) =
{
0xSample01.Real, 0xSample04.Real, 0xSample03.Real, 0xSample04.Real,
0xSample05.Real, 0xSample06.Real, 0xSample07.Real, 0xSample08.Real,
0xSample09.Real, 0xSample10.Real, 0xSample11.Real, 0xSample12.Real,
0xSample13.Real, 0xSample14.Real, 0xSample15.Real, 0xSample16.Real,
0xSample01.Imag, 0xSample04.Imag, 0xSample03.Imag, 0xSample04.Imag,
0xSample05.Imag, 0xSample06.Imag, 0xSample07.Imag, 0xSample08.Imag,
0xSample09.Imag, 0xSample10.Imag, 0xSample11.Imag, 0xSample12.Imag,
0xSample13.Imag, 0xSample14.Imag, 0xSample15.Imag, 0xSample16.Imag
};
```

Source: CE018

*Figure V: 16 Sample fractcomplex array*

For the FFT function to execute, the input data must be in the fractional fixed point range of [-0.5, 0.5]. To do this, we need to shift the signal data to the right by one. Once that is completed, the following code in *Figure VI* can be executed. The <Num of Samples> needs to be replaced by the number of samples captured by the ADC converter as well as <Num of Butterfly Stages>, which corresponds to the amount of vertical columns as shown in *Figure IV*. The <Sampling Rate> must also be known to be able to scale the array numbers to the correct frequency.

```
FFTComplexIP (<Num of Butterfly Stages>, &sigCmpx[0],
&twiddleFactors[0], COEFFS_IN_DATA);

BitReverseComplex (<Num of Butterfly Stages>, &sigCmpx[0]);

SquareMagnitudeCplx(<Num of Samples>, &sigCmpx[0], &sigCmpx[0].real);

VectorMax(<Num of Samples>/2, &sigCmpx[0].real, &peakFrequencyBin);

peakFrequency = peakFrequencyBin*(<Sampling Rate>/<Num of Samples>);
```

Source: CE018

*Figure VI: 16 FFT Code to Convert Analog Data into a Peak Frequency*

## Conclusion

The Fast Fourier Transform is an efficient algorithm of calculating the Discrete Fourier Transform. This type of transform can find the oscillating frequencies within a sinusoidal signal, such as a sound wave. Microchip's dsPIC30F series Digital Signal Controllers have specific hardware and libraries that can easily compute the FFT from a sampled signal. This sampled signal is retrieved from the Analog to Digital Converter in the device and the amount of samples can be tweaked depending on the needs of the project. Further simplification of the Radix-2 algorithm can be achieved if bounds for the input are limited but the included implementation is able to calculate the transform from any real signal. From this information, the strongest frequency can be calculated and an output operation can take place.

## References

The Scientist and Engineer's Guide to Digital Signal Processing, copyright ©1997-1998 by Steven W. Smith. For more information visit the book's website at: [www.DSPguide.com](http://www.DSPguide.com)

Developing FFT Applications with Low-Power Microcontrollers, copyright © 2006 by Maxim Integrated Products. For more information, visit the website at:

[http://www.maxim-ic.com/appnotes.cfm/an\\_pk/3722](http://www.maxim-ic.com/appnotes.cfm/an_pk/3722)

Decimation-in-time (DIT) Radix-2 FFT, by Douglas L. Jones. For more information, visit the website at:

<http://cnx.org/content/m12016/latest/>

CE018 – Using the Fast Fourier Transform (FFT) for Frequency Detection. For more information, the code example can be downloaded at:

[http://ww1.microchip.com/downloads/en/DeviceDoc/CE018\\_FFT\\_DSPlib\\_101007.zip](http://ww1.microchip.com/downloads/en/DeviceDoc/CE018_FFT_DSPlib_101007.zip)