Micromega Corporation

Application Note 40

Frequency Analysis Using ADC and FFT

Introduction

This application note provides a simple demonstration of frequency analysis using the FFT instruction. A function generator is used to provide an input sine wave to analyze. The demo program samples the analog signal using one of the ADC channels on the uM-FPU V3.1 chip, performs an FFT, then analyzes the magnitudes of the frequency spectrum to determine the frequency band of the input sine wave.

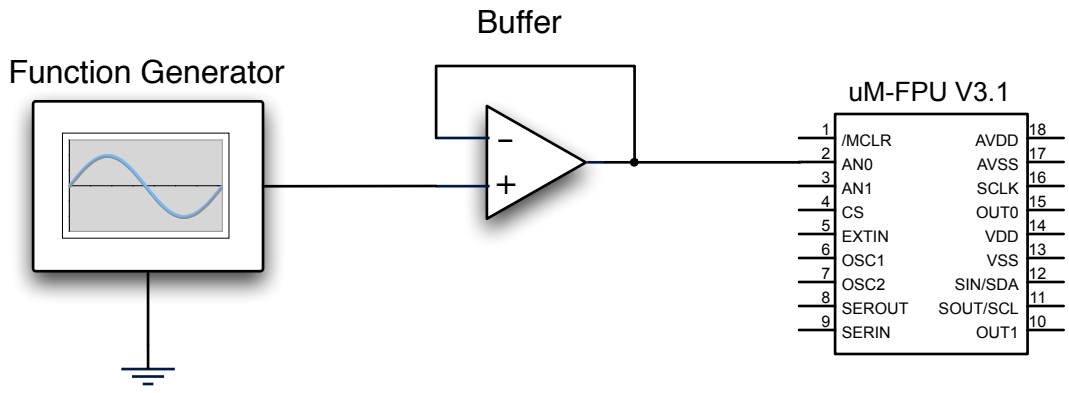
Connecting the Analog Signal

The uM-FPU V3.1 chip has two analog input pins (AN0, AN1) that are connected to a 12-bit ADC. The input range of the analog signals is from AVSS (analog ground) to AVDD (analog VDD). The supply voltage used in this demo if 5V, so the input range is from 0V to 5V.

The function generator is set for sine wave output, and the amplitude is adjusted for a peak-to-peak voltage of 3V, with an offset of 2V, so that the waveform falls in the range 0.5V to 3.5V. The output from the signal generator is passed through a unity gain op-amp that acts as a buffer, before connecting the signal to the AN0 pin on the uM-FPU V3.1 chip.

> **Note:** The uM-FPU V3.1 chip can be damaged if a voltage outside the acceptable limits is applied to an input pin. (See the Absolute Maximum Ratings in the uM-FPU V3.1 Datasheet). If not adjusted properly, most general purpose function generators can produce voltages well outside the acceptable limits, so a unity gain op-amp is recommended to provide an added level of protection.

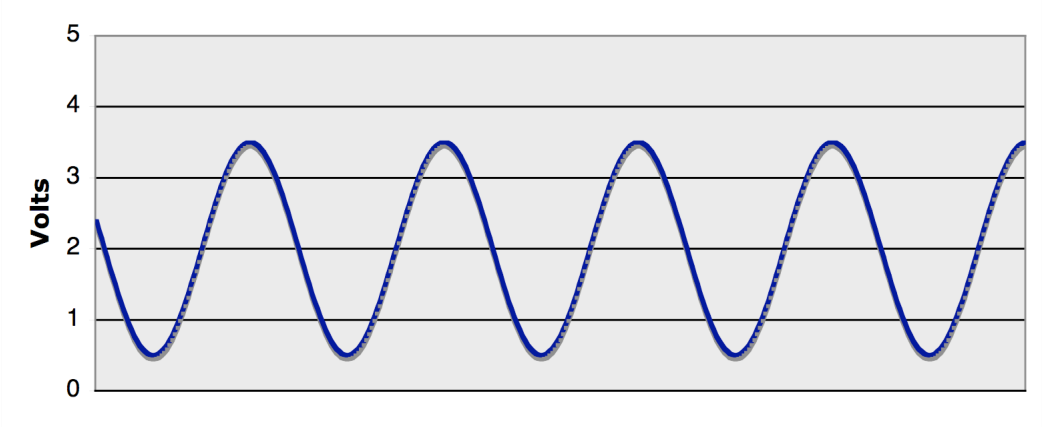
Figure 1 - Connecting the Analog Signal



Analog to Digital Conversion Using the uM-FPU V3.1

The input waveform from the function generator is a continuous sine wave (see Figure 2). The 12-bit ADC converts the continuos analog waveform into a series of discrete digital values. The Nyquist rate is the minimum sampling rate required to avoid aliasing (i.e. to ensure that enough samples are collected to adequately represent the waveform). For a sine wave, the Nyquist rate is equal to twice the frequency of the waveform. The maximum sampling rate of the uM-FPU ADCs is 10 kHz, so the maximum frequency of the input sine wave is 5 kHz.

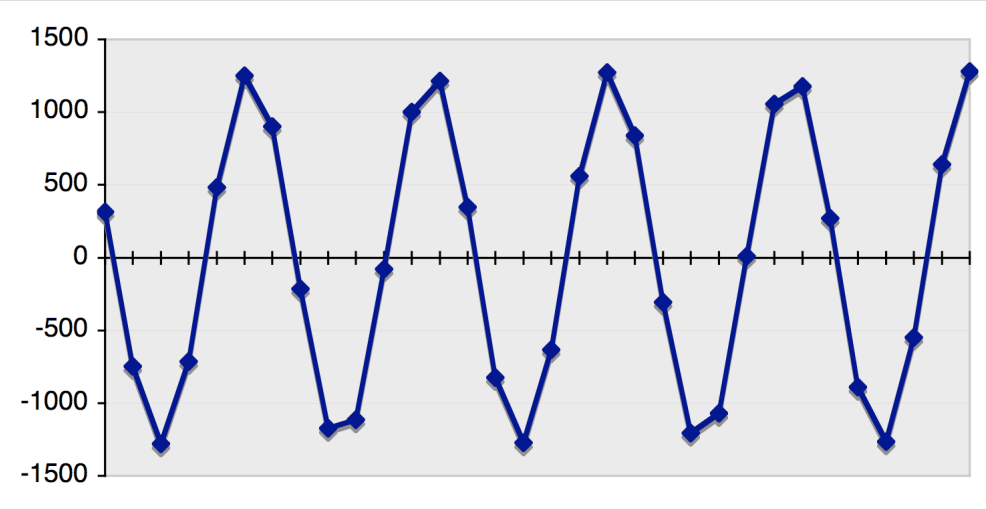
Figure 2 - Input Waveform (1.53 kHz sine wave)



The `ADCMODE` instruction defines the trigger type and repeat count for the ADC. Since the sine wave is continuously changing, we want to sample at precise time intervals, so the timer trigger is used with one sample per trigger. The period for the timer is the inverse of the sampling frequency and is specified in microseconds. The `ADCSCALE` instruction is used to set the scaling value for converting raw ADC values to floating point. Once the ADC is enabled with the `ADCMODE` instruction, the `ADCWAIT` instruction is used to wait until the next sample is ready. The `ADCLOAD` instruction is then used to get the sample value in floating point (using the scaling value).

Since a 32 point FFT will be calculated, 32 input samples are collected and stored in the uM-FPU registers. As the samples are collected, the minimum and maximum values are calculated. The average of these values is used to center the samples around zero. Figure 3 shows an example of the input sampling after the samples have been normalized.

Figure 3 - Input Sampling (1.53 kHz sine wave)

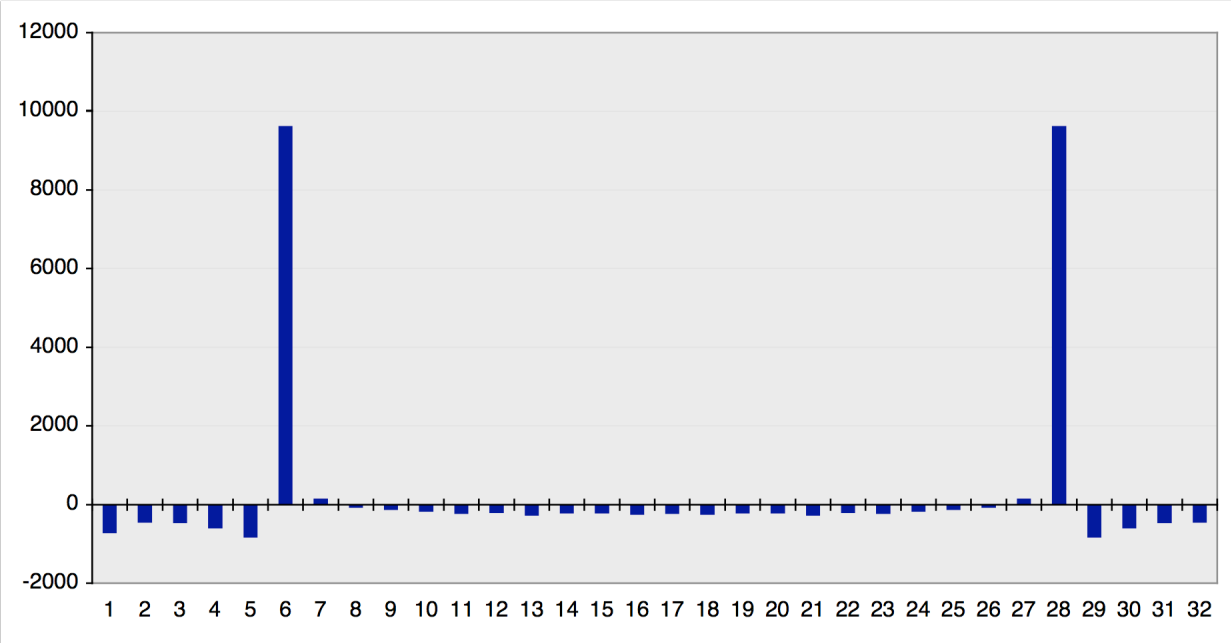


The Fast Fourier Transform

The Fast Fourier Transform (FFT) calculates the discrete Fourier transform for the input samples. This provides the frequency spectrum of the input signal. If all data points fit in the FPU registers, the FFT can be calculated with a single `FFT` instruction. If a larger number of data points are required, the `FFT` instruction is used repeatedly as part of a multi-stage algorithm. (See Application Note 35 - Fast Fourier Transforms Using the FFT Instruction.) We use 32 sample points in this application note, so a single `FFT` instruction calculates the FFT.

The `FFT` instruction takes an *n* by 2 matrix (where *n* is a power of 2) as input and stores the results back in the same matrix. Each row of the matrix represents one data point, with the first column being the real part of a complex number, and the second column being the imaginary part of a complex number. In our example, the 32 samples from the ADC are stored in the first column of the matrix, and zeroes are stored in the second column. Figure 4 shows the magnitude values for the frequency spectrum after the FFT is calculated.

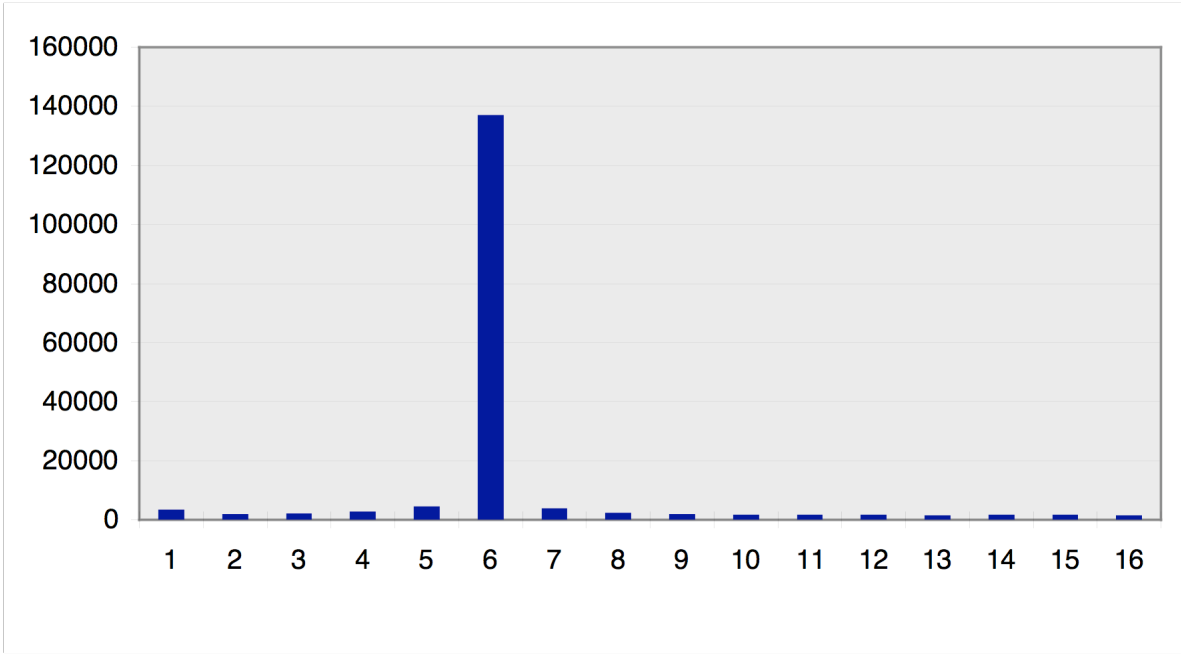
Figure 4 - FFT Output (1.53 kHz sine wave)



Notice that the second half of the results are a mirror image of the first. Only the first 16 values need to be analyzed to determine the dominant frequency. Since individual samples are susceptible to noise, we will make ten sampling passes and accumulate the magnitude values. This reduces the noise component and increases accuracy. Figure 5 shows the accumulated magnitude values after the ten sampling passes.

The bandwidth of each point in the result is equal to the sampling frequency divided by the number of points in the FFT (except for the first and last points which have only half the bandwidth of the other points). In Figure 5, the sample frequency is 10 kHz and there are 32 points in the FFT, so the bandwidth of points 2 through 15 is 312.5 Hz, and the bandwidth of points 1 and 16 is 156.25 Hz. The frequency band for point 6 is therefore 1.40625 Hz to 1.71875 Hz.

Figure 5 - Magnitude Values (1.53 kHz)



Point 6 shows significantly more magnitude than the other points. The frequency of the input sine wave was 1.53 kHz, which falls in the frequency band for point 6.

The dominant frequency is determined by checking the 16 points to find the maximum accumulated magnitude. Since some input frequencies can have magnitudes that are split between adjacent points, the sample program increases the frequency width to include half of the bandwidth for any adjacent point that has more than half the accumulated magnitude of the maximum point. Figures 6a, 6b and 6c show the results for other input frequencies.

Figure 6a - Magnitude Values (1.73 kHz sine wave)

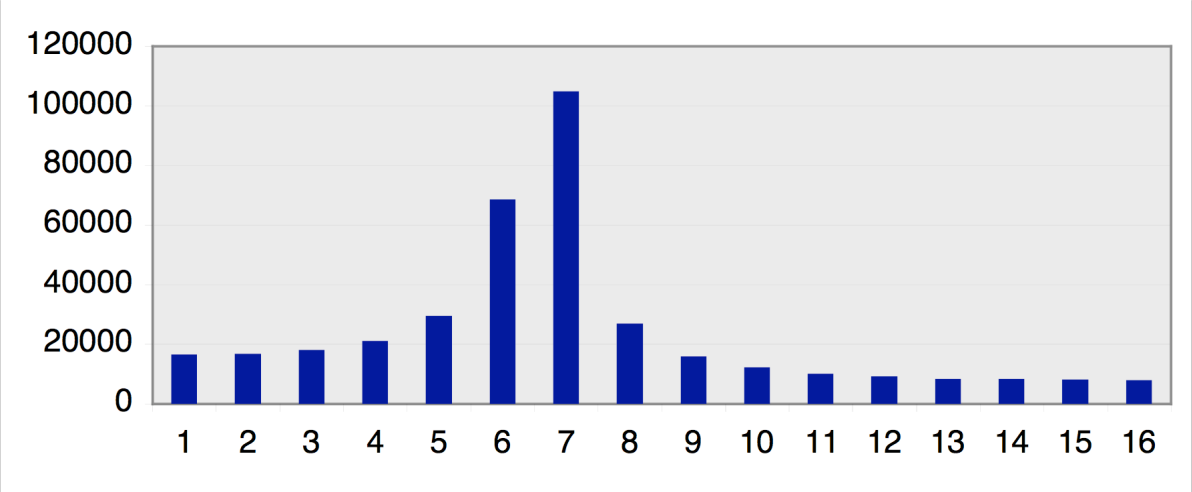


Figure 6b - Magnitude Values (3.33 kHz sine wave)

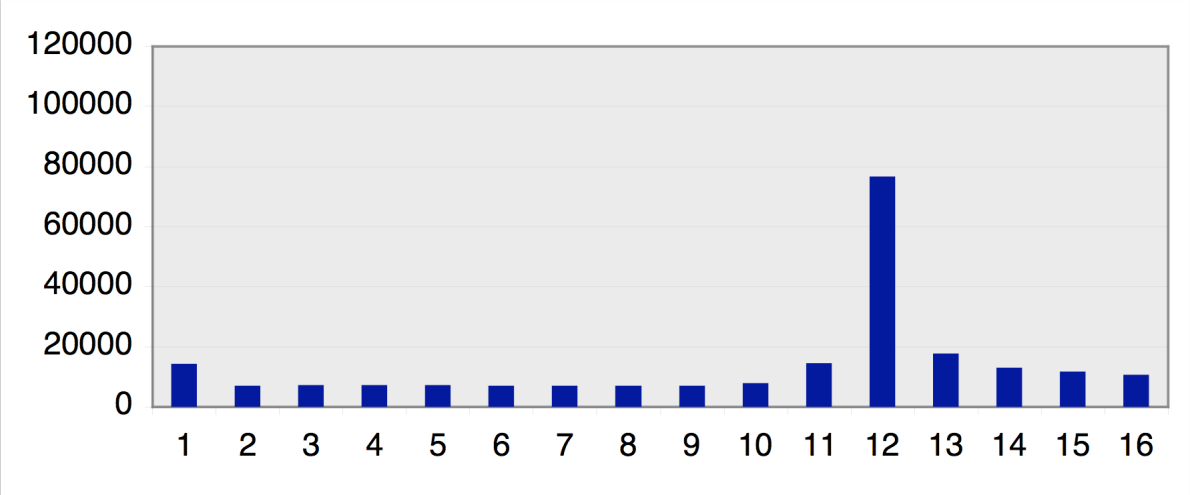
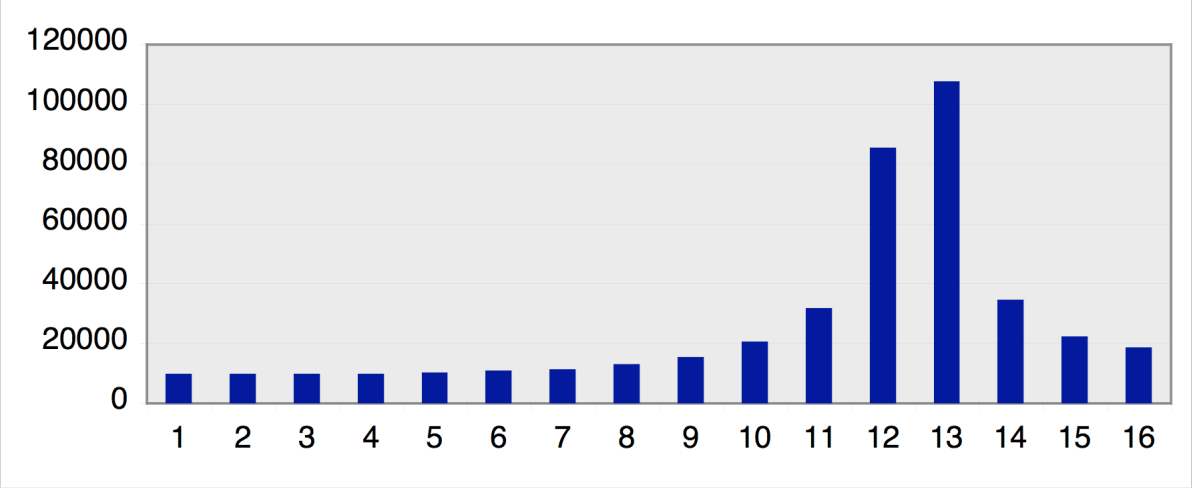


Figure 6c - Magnitude Values (3.57 kHz sine wave)



Demonstration Program

Figure 7 shows the output from the demonstration program. The majority of the code is contained in uM-FPU V3.1 user-defined functions that are described in the next section.

Note: The demo prints the frequency band of the input signal whenever if the values change. The frequency values are rounded before being displayed.

Figure 7: Sample Output from *getFreq.bs2* Demo Program

```
Frequency Analysis Demo
uM-FPU V3.1.0

Sampling Frequency:       5.0 kHz
Maximum FFT Frequency:    2.5 kHz
Bandwidth per FFT point:  156 Hz

 1.5 kHz  to  1.6 kHz
```

uM-FPU V3.1 User-defined Functions

The *getFreq.fpu* file contains uM-FPU V3.1 user-defined functions to do the following:

- set the sample frequency for the ADC
- read the ADC, normalize data, perform FFT, and accumulate the magnitude results
- determine the frequency band for the dominant frequency
- convert a value in Hertz to a formatted string

A summary of each uM-FPU V3.1 function is shown below.

getID

Returns an ID number that the microcontroller program can use as a simple check to confirm that the correct user-defined functions have been programmed on the uM-FPU V3.1 chip. Register 0 is set to zero before calling this function. If no functions have been programmed it will remain zero. If the correct `getID` function is programmed, a value of 40 is returned.

Input:

register 0	32-bit integer	0

Output:

register 0	32-bit integer	40

setSample

Sets the sampling frequency for the ADC. The sample frequency specified is converted to the sample period in microseconds and used by the `ADCMODE` instruction to set the period for the timer trigger. The minimum period is 100 microseconds. The period is converted back to the actual sample frequency, and the frequency width of the FFT points is calculated.

Input:

register 0	32-bit long	sample frequency in Hertz

Output:

sampleFreq	32-bit float	actual sample frequency (Hertz)
binFreq	32-bit float	frequency width of each FFT point (Hertz)
binFreq2	32-bit float	frequency width of each FFT point (Hertz) / 2

readADC

Makes ten sampling passes on the input waveform. For each pass, the input waveform is sampled, samples are normalized to be centered on zero, the FFT is calculated, and the magnitude values are accumulated.

Input:

sampleFreq	32-bit float	actual sample frequency (Hertz)

Output:

magArray	32-bit float	accumulated magnitude array

getFreq

Analyses the accumulated magnitude array to find the maximum magnitude and sets the lower and upper values for the frequency band. If the magnitude value for an adjacent point is greater than half the magnitude of the maximum magnitude, the frequency band is extended in that direction.

Input:

magArray	32-bit float	accumulated magnitude array

Output:

freq1	32-bit integer	lower range of frequency band
freq2	32-bit integer	upper range of frequency band

Z flag set in status byte if `freq1` or `freq2` changed since last call

insertHz

Converts a value in Hertz to a formatted string and stores it at the current string selection point. If the value is less than 1000, the value is displayed in Hz, otherwise the value is displayed in kHz.

Input:

register A	32-bit float	frequency value in Hertz

Output:

string selection	string	*nnnn Hz* or *nn.n kHz* e.g. `1.5 kHz`

Additional Files

There are additional files located on the Micromega website that accompany this application note. They include:

getFreq.fpu	contains uM-FPU V3.1 user-defined functions
getFreq.bs2	Basic Stamp demo program

Before running the demo program, the user-defined functions in *getFreq.fpu* must be programmed into the uM-FPU V3.1 chip using the uM-FPU V3 IDE software.

Further Information

See the Micromega website (http://www.micromegacorp.com) for additional information regarding the uM-FPU V3.1 floating point coprocessor, including:

uM-FPU V3.1 Datasheet
uM-FPU V3.1 Instruction Set
Using the uM-FPU V3 Integrated Development Environment (IDE)
Application Note 35 - Fast Fourier Transforms using the FFT Instruction