1 Introduction

The Fast Fourier Transform (FFT) is a mathematical technique for transforming a time-domain digital signal into a frequency-domain representation of the relative amplitude of different frequency regions in the signal. The Fast Fourier Transform is a method for doing this process very efficiently. The FFT may be computed using a relatively short excerpt from a signal.

The Fast Fourier Transform is one of the most important topics in Digital Signal Processing. The FFT is extremely important in the area of frequency (spectrum) analysis: for example, voice recognition, digital coding of acoustic signals for data stream reduction in the case of digital transmission, detection of machine vibration, signal filtration, solving partial differential equations, and so on.

This application note describes how to use the FFT in metering applications, especially for power and energy computing in power meters.
2 DFT basics

For a proper understanding of the next sections, it is important to clarify what a Discrete Fourier Transform (DFT) is. The DFT is a specific kind of discrete transform, used in Fourier analysis. It transforms one function into another, which is called the frequency domain representation of the original function (a function in the time domain). The input to the DFT is a finite sequence of real or complex numbers, making the DFT ideal for processing information stored in computers. The relationship between the DFT and the FFT is as follows: DFT refers to a mathematical transformation or function, regardless of how it is computed, whereas the FFT refers to a specific family of algorithms for computing a DFT.

The DFT of a finite-length sequence of size $N$ is defined as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j \frac{2\pi nk}{N}} = \sum_{n=0}^{N-1} \left[ x(n) \cdot \cos\left(\frac{2\pi nk}{N}\right) - j \cdot x(n) \cdot \sin\left(\frac{2\pi nk}{N}\right) \right], \; 0 \leq k < N \quad \text{Eqn. 1}$$

Where:

- $X(k)$ is the output of the transformation
- $x(n)$ is the input of the transformation (the sampled input signal)
- $j$ is the imaginary unit

Each item in Equation 1 defines a partial sinusoidal element in complex format with a $kF_0$ frequency, with $(2\pi nk/N)$ phase, and with an $x(n)$ amplitude. Their vector summation for $n=0,1,\ldots,N-1$ (see Equation 1) and for the selected $k$-item, represents the total sinusoidal item of spectrum $X(k)$ in complex format for the $kF_0$ frequency. Note, that $F_0$ is the frequency of the input periodic signal. In the case of non-periodic signals, $F_0$ means the selected basic period of this signal for DFT computing.

The Inverse Discrete Fourier Transform (IDFT) is given by:

$$x(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X(k) \cdot e^{j \frac{2\pi nk}{N}}, \; 0 \leq n < N \quad \text{Eqn. 2}$$

Thanks to Equation 2, it is possible to compute discrete values of $x(n)$ retrospectively from the spectrum items of $X(k)$.

In these two equations, both $X(k)$ and $x(n)$ can be complex, so $N$ complex multiplications and $(N-1)$ complex additions are required to compute each value of the DFT if we use Equation 1 directly. Computing all $N$ values of the frequency components requires a total of $N^2$ complex multiplications and $N(N-1)$ complex additions.

3 FFT implementation

With regards to the derived equations in the previous chapter, it is good to introduce the following substitution:
FFT implementation

\[
W_N^{nk} = e^{-j \frac{2\pi nk}{N}}
\]

Eqn. 3

The \(W_N^{nk}\) element in this substitution is also called the “twiddle factor”. With respect to this substitution we may rewrite the equation for computing the DFT and IDFT into these formats:

\[
DFT[x(n)] = X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk}
\]

Eqn. 4

\[
IDFT[X(k)] = x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot W_N^{-nk}
\]

Eqn. 5

To improve efficiency in computing the DFT, some properties of \(W_N^{nk}\) are exploited. They are described as follows:

Symmetrical property: \(W_N^{nk+N/2} = -W_N^{nk}\)  

Eqn. 6

Periodicity property: \(W_N^{nk} = W_N^{nk+N} = W_N^{nk+2N} = \ldots\)  

Eqn. 7

Recursion property: \(W_N^{nk/N/2} = W_N^{2nk}\)  

Eqn. 8

These properties arise from the graphical representation of the twiddle factor (Equation 3) by the rotational vector for each \(nk\) value.

3.1 The Radix-2 decimation in time FFT description

The basic idea of the FFT is to decompose the DFT of a time domain sequence of length \(N\) into successively smaller DFTs whose calculations require less arithmetic operations. This is known as a divide-and-conquer strategy, one made possible by those properties described in the previous section. Decomposition into shorter DFTs may be performed by splitting an \(N\)-point input data sequence \(x(n)\) into two \(N/2\)-point data sequences \(a(m)\) and \(b(m)\), corresponding to the even-numbered and odd-numbered samples of \(x(n)\), respectively, that is:

- \(a(m) = x(2m)\), i.e. samples of \(x(n)\) for \(n=2m\)
- \(b(m) = x(2m+1)\), i.e. samples of \(x(n)\) for \(n=2m+1\)

where \(m\) is an integer ranging in \(0 \leq m < N/2\).

This process of splitting the time domain sequence into even and odd samples is what gives the algorithm its name, “Decimation In Time (DIT)”. Thus, \(a(m)\) and \(b(m)\) are obtained by decimating \(x(n)\) by a factor of 2; hence, the resulting FFT algorithm is also called “radix-2”. It is the simplest and most common form of the Cooley-Tukey algorithm \([1]\).
Now, the $N$-point DFT (see Equation 1) can be expressed in terms of DFTs of the decimated sequences as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk} = \sum_{m=0}^{N/2-1} x(2m) \cdot W_N^{2mk} + \sum_{m=0}^{N/2-1} x(2m+1) \cdot W_N^{(2m+1)k}$$

Eqn. 9

$$= \sum_{m=0}^{N/2-1} x(2m) \cdot W_N^{2mk} + W_N^k \sum_{m=0}^{N/2-1} x(2m+1) \cdot W_N^{mk}$$

With the substitution given by Equation 8, the Equation 9 can be expressed as:

$$X(k) = \sum_{m=0}^{N/2-1} a(m) \cdot W_{N/2}^{mk} + W_N^k \sum_{m=0}^{N/2-1} b(m) \cdot W_{N/2}^{mk} = A(k) + W_N^k B(k) \quad , \quad 0 \leq k < N$$

Eqn. 10

These two summations represent the $N/2$-point DFTs of the sequences $a(m)$ and $b(m)$, respectively. Thus, DFT[$a(m)$]=$A(k)$ for even-numbered samples, and DFT[$b(m)$]=$B(k)$ for odd-numbered samples. Thanks to the periodicity property (Equation 7) of the DFT, the outputs for $0 \leq k < N/2$ from a DFT of length $N/2$ are identical to the outputs for $0 \leq k < N/2$. That is, $A(k+N/2)=A(k)$ and $B(k+N/2)=B(k)$ for $0 \leq k < N/2$. In addition, the factor $W_N^{k+N/2} = -W_N^k$ thanks the to symmetrical property (Equation 6). Thus, the whole DFT can be calculated as follows:

$$X(k) = A(k) + W_N^k B(k) \quad , \quad 0 \leq k < N/2$$

Eqn. 11

$$X(k+N/2) = A(k) - W_N^k B(k) \quad , \quad 0 \leq k < N/2$$

This result, expressing the DFT of length $N$ recursively in terms of two DFTs of size $N/2$, is the core of the radix-2 DIT FFT. Note, that final outputs of $X(k)$ are obtained by a +/- combination of $A(k)$ and $B(k)W_N^k$, which is simply a size 2 DFT. These combinations can be demonstrated by a simply oriented graph, sometimes called a “butterfly” in this context (see Figure 1).

**Figure 1. Basic butterfly computation in the DIT FFT algorithm**

The procedure of computing the discrete series of an $N$-point DFT into two $N/2$-point DFTs may be adopted for computing the series of $N/2$-point DFTs from items of $N/4$-point DFTs. For this purpose, each $N/2$-point sequence should be divided into two sub-sequences of even and odd items and computing their DFTs consecutively. The decimation of the data sequence can be repeated again and again until the resulting sequence is reduced to one basic DFT.
Figure 2. Decomposition of an 8-point DFT

For illustrative purposes, Figure 2 depicts the computation of an $N = 8$-point DFT. We observe that computation is performed in three stages ($3 = \log_2 8$), beginning with the computations of four 2-point DFTs, then two 4-point DFTs, and finally, one 8-point DFT. Generally, for an $N$-point FFT, the FFT algorithm decomposes the DFT into $\log_2 N$ stages, each of which consist of $N/2$ butterfly computations. The combination of the smaller DFTs to form the larger DFT is illustrated in Figure 3 for $N=8$.
Note, that each dot represents a complex addition and each arrow represents a complex multiplication in Figure 3. The $W_N^k$ factors in Figure 3 may be presented as a power of two ($W_2$) at the first stage, as a power of four ($W_4$) at the second stage, as a power of eight ($W_8$) at the third stage, etc. It is also possible to represent it uniformly as a power of $N$ ($W_N$), where $N$ is the size of the input sequence $x(n)$. Context between both of these expressions gives Equation 8.

### 3.2 The Radix-2 decimation in time FFT requirements

For effective and optimal decomposition of the input data sequence into even and odd sub-sequences, it is good to have the power-of-two input data samples (...64,128, and so on).

The first step before computing the radix-2 FFT is a re-ordering of the input data sequence (see also the left side of Figure 2 or Figure 3). This means that this algorithm needs a bit-reversed data ordering; that is, the MSBs become LSBs, and vice versa. Table 1 shows an example of bit-reversal with an 8-point input sequence.

<table>
<thead>
<tr>
<th>Decimal number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary equivalent</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>Bit reversed binary</td>
<td>000</td>
<td>100</td>
<td>010</td>
<td>110</td>
<td>001</td>
<td>101</td>
<td>011</td>
<td>111</td>
</tr>
<tr>
<td>Decimal equivalent</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

It is important to note that this type of FFT algorithm is an “in place”, which means that the outputs of each butterfly throughout the computation can be placed in the same memory locations from which the inputs were fetched, resulting in an in-place algorithm that requires no extra memory to perform the FFT.

#### 3.2.1 Window selection

The FFT computation assumes that a signal is periodic in each data block; that is, it repeats over and over again. Most signals aren’t periodic, and even a periodic one might have an unknown period. When the FFT of a non-periodic signal is computed, then the resulting frequency spectrum suffers from leakage. To resolve this issue, it is good to take $N$ samples of the input signal and make them periodic. This may be generally performed by window functions (Barlett, Blackman, Kaiser-Bessel, and so on). Considering that the resulting spectrum after the application of some window function may have a slightly different shape in comparison to the frequency spectrum of a pure periodic signal without windowing, it is better not to use a special window function in a metering application too, or to use a simple rectangular window (a function with a coherent gain of 1.0). This requires that the frequency of the input signal is well known, of course. In metering applications, this is accomplished thanks to measuring a period of line voltage.

The detection of a signal (mains) period may be performed by a zero-cross detection (ZCD) technique. Zero-crossing is the instantaneous point at which there is no voltage present (see Figure 4a). In a line voltage wave, or other simple waveform, this normally occurs twice during each cycle. Counting the zero-crossing is a method used for frequency measurement of an input
signal (the line voltage). For example, the ZCD circuit may be realized by using an analogue comparator inside the MCU, where the first channel is connected to the reference voltage and the second channel is connected to the line through a simple voltage divider. Finally, the change in logic level from this comparator is interpreted by software as a zero-crossing of the mains. The time between zero-crossings is measured by a timer in the software. These zero-crossings also define the start and end-points of a simple rectangular FFT window (Figure 4a). Technically, it is not necessary to measure the frequency of an input signal by zero-cross points, but it is possible to use any other two points of the input signal that may be simply recognized - peak points, for example (see Figure 4b) - with a similar result (magnitudes are the same, phases are uniformly shifted).

![Figure 4. Zero-cross point vs. peak point detection](image)

It is also good to know that this software technique for measuring the signal frequency must contain some kind of sophisticated algorithm for removing possible voltage spikes (see Figure 4). These spikes may appear in the line as a product of interference from a load (motor, contactor, and so on) and may cause false zero-crossing or peak detection.

In a practical implementation, it is better to measure the time between several true zero-cross or peak points. Finally, an arithmetic mean must be performed to compute the correct signal frequency. Each period of input signals (voltage and current) is then sampled with a frequency, which is \( N \) times higher than the measured frequency of the line voltage, where \( N \) is the number of samples. When the sampling frequency is different from this, the resulting frequency spectrum may suffer from leakage.

### 3.3 The Radix-2 decimation in time FFT conclusion

The radix-2 FFT utilizes some clever algorithms to do the same thing as the DFT, but in much less time. Whereas the DFT needs \( N^2 \) complex multiplications (see at Section 2, “DFT basics”), the FFT takes only \( N/2 \cdot \log_2 N \) complex multiplications and \( N \cdot \log_2 N \) complex additions. Therefore,
the ratio between the DFT computation and the FFT computation for the same \( N \) is proportional to \( 2N / \log_2 N \). In cases where \( N \) is small, this ratio is not very significant, but when \( N \) becomes large, this ratio gets very large. Therefore, the FFT is simply a fast way to calculate the DFT.

The radix-2 FFT algorithm is generally defined as a radix-\( r \) FFT algorithm, where the \( N \)-point input sequence is split into \( r \)-subsequences to raise computation efficiency, for example radix-4 or radix-8. Thus, the radix is the size of the FFT decomposition.

Similarly the DIT algorithm is sometimes used Decimation In Frequency (DIF) algorithm (also called the Sande-Tukey algorithm), which decomposes the sequence of DFT coefficients \( X(k) \) into successively smaller sub-sequences\[^3\]. However, this application note describes only the radix-2 DIT FFT algorithm.

4 Using an FFT for power computing

4.1 Conversion between Cartesian and polar forms

The FFT implementation in power meters requires complex number computing, because the mathematical formulas describing the DFT or FFT in previous chapters suppose that each item in these formulas (in graphical format these are \( X(k) \) in Figure 3) contains a complex number.

A complex number is a number consisting of a real and an imaginary part. This number can be represented as a point or position vector in a two-dimensional Cartesian coordinate system called the complex plane. The numbers are conventionally plotted using the real part as the horizontal component, and imaginary part as the vertical (see Figure 5).

Another way of encoding points in the complex plane, other than using the \( x \) - and \( y \)-coordinates, is to use the distance of a point \( z \) to \( O \), the point whose coordinates are \((0,0)\), and the angle of the line through \( z \) and \( O \). This idea leads to the polar form of complex numbers. The absolute value (or magnitude) of a complex number \( z = x + iy \) is

\[
r = |z| = \sqrt{x^2 + y^2}
\]  

Eqn. 12

The argument or phase of \( z \) is defined as:

\[
\varphi = \arg(z) = \tan^{-1}\left(\frac{y}{x}\right)
\]  

Eqn. 13
Together, \( r \) and \( \phi \) show another way of representing complex numbers, the polar form, as the combination of modulus and argument fully specify the position of a point on the plane.

### 4.2 Root Mean Square computing

In electrical engineering, the Root Mean Square (RMS) is a fundamental measurement of the magnitude of an AC signal. The RMS value assigned to an AC signal is the amount of DC required to produce an equivalent amount of heat in the same load. In a complex plane, the RMS value of the current (\( I_{RMS} \)) and the voltage (\( U_{RMS} \)) is the same as the summation of their magnitudes (see vector \( r \) in Figure 5) associated for each harmonic. Regarding Equation 12, the total RMS values of current and voltage in the frequency domain are defined as:

\[
I_{RMS} = \sqrt{\frac{1}{N-1} \sum_{k=0}^{N/2-1} (I_{RE}(k) + I_{IM}(k))^2}
\]

\[
U_{RMS} = \sqrt{\frac{1}{N-1} \sum_{k=0}^{N/2-1} (U_{RE}(k) + U_{IM}(k))^2}
\]

Where:

- \( I_{RE}(k), U_{RE}(k) \) are real parts of \( k^{th} \) harmonics of current and voltage.
- \( I_{IM}(k), U_{IM}(k) \) are imaginary parts of \( k^{th} \) harmonics of current and voltage.

### 4.3 Complex power computing

AC power flow has three components: real or true power (P) measured in watts (W), apparent power (S) measured in volt-amperes (VA), and reactive power (Q) measured in reactive volt-amperes (VAR). These three types of power - active, reactive, and apparent - relate to one another in a trigonometric form. This is called a power triangle (see Figure 6).

\[
S = P + jQ = U \cdot I^* \]

Angle \( \phi \) in this picture is the phase of voltage relative to current. A complex power is then defined as:

\[
S = P + jQ = U \cdot I^* \]

---

**Figure 6. Power triangle**

**Eqn. 14**

**Eqn. 15**
Using an FFT for power computing

Where $U$ is a voltage vector ($U=U_{RE}+jU_{IM}$) and $I$ is a current vector ($I=I_{RE}+jI_{IM}$). Note, that $I^*$ is a complex conjugate current vector.

Regarding Equation 12, the length of a complex power ($|S|$) is the apparent power (VA) actually. In terms of current and voltage phasors (FFT outputs), and in terms of Equation 15, the complex power in Cartesian form can be finally expressed as:

$$S = \sum_{k=1}^{N/2-1} (U_{RE}(k) + jU_{IM}(k)) \cdot (I_{RE}(k) - jI_{IM}(k))$$

Eqn. 16

Where:

$I_{RE}(k)$, $U_{RE}(k)$ are real parts of $k^{th}$ harmonics of current and voltage.

$I_{IM}(k)$, $U_{IM}(k)$ are imaginary parts of $k^{th}$ harmonics of current and voltage.

In terms of Equation 12 and Equation 13, both parts of the total complex power (P and Q) can be also expressed in polar form as:

$$S = \sum_{k=1}^{N/2-1} \left( \left| I(k) \right| \cdot \left| U(k) \right| \cdot \cos(U_{\phi}(k) - I_{\phi}(k)) \right) + j \left( \left| I(k) \right| \cdot \left| U(k) \right| \cdot \sin(U_{\phi}(k) - I_{\phi}(k)) \right)$$

Eqn. 17

Where:

$|I(k)|$, $|U(k)|$ are magnitudes of $k^{th}$ harmonics of current and voltage.

$I_{\phi}(k)$, $U_{\phi}(k)$ are phase shifts of $k^{th}$ harmonics of current and voltage (with regards to the FFT window origin).

Note, that inputs for these equations are Fourier items of current and voltage (in Cartesian or polar form). For a graphical interpretation of these items, see $X(k)$ in Figure 3.

There are two basic simplifications used in the previous formulas:

- Thanks to the symmetry of the FFT spectrum, only $N/2$ items are used for complex power computing.
- It is expected that voltage in the mains has no DC offset. Therefore the 0-harmonic is missed in both formulas because of multiplication of the current values ($I_{RE}(0)$, $I_{IM}(0)$, $|I(0)|$) with zero.

Total apparent power may be also computed from the RMS values of voltage and current as:
5 Practical implementation

This section describes implementation of the FFT algorithm in C-code. Here is described the application programming interface (API) of two main C-functions, one for FFT computing and the other for power computing. These two functions are the basis of the metering library, which is currently used for power computing in three electricity meter reference designs. All of these reference designs are described in the following Freescale design reference manuals:

- DRM121 - describes hardware design of the single-phase electricity meter based on the MCF51EM256 silicon (ColdFire® V1 core).
- DRM122 - describes hardware design of the single-phase electricity meter based on the MK30X256 silicon (ARM® Cortex®-M4 core).
- Design reference manual that describes the two-phase electricity meter based on the MKM34Z128 silicon (ARM Cortex-M0+ core).

\[
S = U_{RMS} \cdot I_{RMS}
\]

Figure 7. A Block diagram of the power meter measurement process based on the FFT
In true power meters, the energies (active, reactive) are computed from the powers consecutively by accumulation of these powers per time unit. A simple block diagram of this computing process in a typical power meter is depicted in Figure 7.

5.1 API summary

This section describes the application programming interface (API) used in the metering library. There are defined two main functions (see Table 2) in the metering library: FFT_radix2 and PowerCalculation. There are also some additional mathematical functions used inside these main functions, such fractional, radix and arctangent computing functions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Output</th>
<th>Description</th>
<th>Code size(^1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT_radix2</td>
<td>n, input, result</td>
<td>void</td>
<td>Computes the FFT (radix-2 DIT algorithm)</td>
<td>514 bytes</td>
</tr>
<tr>
<td>PowerCalculation</td>
<td>voltage, current, powers, mU,mI</td>
<td>void</td>
<td>Computes the whole power vector</td>
<td>706 bytes</td>
</tr>
</tbody>
</table>

\(^1\) Valid for ARM Cortex-M0+ core.

5.2 FFT_radix2 function

This function computes the FFT of the input signal: that is, it transforms input data from the time domain into the frequency domain using the radix-2 DIT algorithm (see Section 3.1, “The Radix-2 decimation in time FFT description”). The output parameters (arguments) from this function are in Cartesian data format. Each real and imaginary item of the output spectrum has a width of 32 bits.

5.2.1 Synopsis

This subsection provides the header file that should be included within a source file that references the FFT_radix2 function. An appropriate declaration for this function is also shown below. This declaration is not included in your program; only the header file should be included:

```c
#include “fft2.h”
void FFT_radix2(unsigned short n,long input[N_SAMPLES],ComplexFFT result[N_SAMPLES])
```

5.2.2 Arguments

This subsection describes input and output arguments of the FFT_radix2 function.

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Data format</th>
<th>Used range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>In</td>
<td>unsigned 16-bits</td>
<td>8, 16, 32, 64, 128</td>
<td>Number of samples per one period of input signal</td>
</tr>
<tr>
<td>input</td>
<td>In</td>
<td>signed 32-bits</td>
<td>0x80000000 ... 0x7FFFFFFF</td>
<td>Input data buffer (size n) in time domain</td>
</tr>
<tr>
<td>result</td>
<td>Out</td>
<td>ComplexFFT</td>
<td>see Table 4</td>
<td>Output data buffer (size n) in frequency domain</td>
</tr>
</tbody>
</table>
5.3 **PowerCalculation function**

This function computes the complete power vector of the input signal from the frequency spectrum. The power vector contains these values: RMS value of the voltage, powers (active, reactive, apparent), and angle between the 1st harmonics of the current and voltage. The *PowerCalculation* function calls internally the *FFT_radix2* function two times, because it needs the output arguments (spectrum) from both the channels, voltage and current channel.

### 5.3.1 Synopsis

This subsection provides the header file that should be included within a source file that references the *PowerCalculation* function. There is also shown an appropriate declaration for this function. This declaration is not included in your program; only the header file should be included: 

```
#include “metering2.h”
```

```cpp
void PowerCalculation (long voltage[N_SAMPLES], long current[N_SAMPLES], Power_Vector *powers, unsigned long mU[MAX_FFT_SEND], unsigned long mI[MAX_FFT_SEND])
```

### 5.3.2 Arguments

This subsection describes input and output arguments of the *PowerCalculation* function.

#### Table 5. PowerCalculation function arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Data format</th>
<th>Used range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage</td>
<td>In</td>
<td>signed 32-bits</td>
<td>0x80000000 ... 0x7FFFFFFF</td>
<td>Voltage input data buffer (size n) in frequency domain</td>
</tr>
<tr>
<td>current</td>
<td>In</td>
<td>signed 32-bits</td>
<td>0x80000000 ... 0x7FFFFFFF</td>
<td>Current input data buffer (size n) in frequency domain</td>
</tr>
<tr>
<td>powers</td>
<td>Out</td>
<td>Power_Vector</td>
<td>see Table 6</td>
<td>Power vector (W, VAr, VA, VRMS, angle)</td>
</tr>
<tr>
<td>mU</td>
<td>Out</td>
<td>unsigned 32-bits</td>
<td>0x00000000 ... 0x7FFFFFFF</td>
<td>Voltage magnitudes (used only for FreeMASTER visualization)</td>
</tr>
<tr>
<td>mI</td>
<td>Out</td>
<td>unsigned 32-bits</td>
<td>0x00000000 ... 0x7FFFFFFF</td>
<td>Current magnitudes (used only for FreeMASTER visualization)</td>
</tr>
</tbody>
</table>

#### Table 6. Power_Vector data type definition

<table>
<thead>
<tr>
<th>Name</th>
<th>Data format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>signed 64-bits</td>
<td>0x8000000000000000 ... 0x7FFFFFFFFFFFFF</td>
<td>Active power (not scaled)</td>
</tr>
<tr>
<td>Q</td>
<td>signed 64-bits</td>
<td>0x8000000000000000 ... 0x7FFFFFFFFFFFFF</td>
<td>Reactive power (not scaled)</td>
</tr>
<tr>
<td>S</td>
<td>unsigned 64-bits</td>
<td>0x0000000000000000 ... 0x7FFFFFFFFFFFFF</td>
<td>Apparent power (not scaled)</td>
</tr>
<tr>
<td>U</td>
<td>unsigned 32-bits</td>
<td>0x00000000 ... 0x7FFFFFFF</td>
<td>RMS value of voltage (not scaled)</td>
</tr>
<tr>
<td>Angle</td>
<td>signed 32-bits</td>
<td>0x80000000 ... 0x7FFFFFFF</td>
<td>Angle between the 1st harmonic of U and I</td>
</tr>
</tbody>
</table>
5.4 Computing performance

The metering library based on the FFT can be used with different hardware platforms (MCUs). This answers the MCU computational requirements for using this metering library on the ARM Cortex-M0+ core (MKM34Z128 MCU). Table 7 shows computational time for PowerCalculation function, which calls the FFT_radix2 function two times, once for voltage FFT computation and then for current FFT computation.

Table 7. Computing performance of the whole FFT metering library

<table>
<thead>
<tr>
<th>Number of FFT samples</th>
<th>Computing time [ms]</th>
<th>MCU execution cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.24</td>
<td>11513</td>
</tr>
<tr>
<td>16</td>
<td>0.55</td>
<td>26384</td>
</tr>
<tr>
<td>32</td>
<td>1.13</td>
<td>54208</td>
</tr>
<tr>
<td>64</td>
<td>2.42</td>
<td>116093</td>
</tr>
<tr>
<td>128</td>
<td>5.36</td>
<td>257131</td>
</tr>
</tbody>
</table>

1) CPUCLK=47.972352 MHz, Compiler optimization=high speed, f_{inp}=50 Hz, Cartesian form of the FFT, ARM Cortex-M0+ core.

6 Summary

This application note describes how to compute powers in a metering application using the FFT. A computing technique based on the FFT has some advantages and also disadvantages:

Advantages of realization:
- The same precision for both active and reactive energies
- Four quadrant active and reactive energy measurement
- Frequency analysis of the mains, ability to compute Total Harmonic Distortion (THD)
- Offset removal, because the 0-harmonic may be missed out for power computing

Disadvantages of realization:
- Adjustable sampling rate is necessary to compensate for the frequency changes in the mains
- Higher computational power of the MCU (a 32-bit MAC unit is required)

7 References


8 Revision history

<table>
<thead>
<tr>
<th>Revision number</th>
<th>Date</th>
<th>Substantial changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11/2011</td>
<td>Initial release</td>
</tr>
<tr>
<td>1</td>
<td>10/2013</td>
<td>Upgrade Section 5, “Practical implementation”</td>
</tr>
</tbody>
</table>


