Freescale Semiconductor, Inc.
Application Note

Document Number: AN4255
Rev. 4, 07/2015

FFT-Based Algorithm for Metering Applications

by: Ludek Slosarcik

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 FFT is a method for doing this process very efficiently. It may be computed using a relatively short excerpt from a signal.

The FFT is one of the most important topics in Digital Signal Processing. It 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 energy computing in power meters. The critical task in a metering application is an accurate computation of energies, which are sometimes referred to as billing quantities. Their computation must be compliant with the international standard for electronic meters. The remaining quantities are calculated for informative purposes and they are referred to as non-billing.

Contents

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:

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

$$0 \le k < N$$

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 $x(n)$ amplitude. Their vector summation for n = 0,1,...,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:

$$x(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X(k) \cdot e^{j\frac{2\pi nk}{N}} \qquad \textit{Eqn. 2}$$

$$0 \le n < N$$

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

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 $N\cdot(N\text{-}1)$ complex additions.

3 FFT implementation

With regards to the derived equations in Section 2, "DFT basics," it is good to introduce the following substitution:

$$W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \quad \textit{Eqn. 3}$$

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:

$$DFT[x(n)] = X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk} \quad \textit{Eqn. 4}$$

$$IDFT[X(k)] = x(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X(k) \cdot W_N^{-nk} \quad \textit{Eqn. 5}$$

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

Symmetral property:

$$W_N^{nk+N/2} = -W_N^{nk} \quad \textit{Eqn. 6}$$

Periodicity property:

$$W_N^{nk} = W_N^{nk+N} = W_N^{nk+2N} = \ldots \quad \textit{Eqn. 7}$$

Recursion property:

$$W_{N/2}^{nk} = W_N^{2nk} \quad \textit{Eqn. 8}$$

These properties arise from the graphical representation of the twiddle factor (Equation 4) 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, made possible using the properties described in the previous section. The 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:

- $a(m) = x(2m)$, that is, samples of $x(n)$ for $n = 2m$
- $b(m) = x(2m + 1)$, that is, samples of $x(n)$ for $n = 2m + 1$

where *m* is an integer in the range of $0 \le m < N/2$.

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 two; 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:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk} = \sum_{m=0}^{N/2-1} x(2m) \cdot W_N^{2mk} + \sum_{m=0}^{N/2-1} x(2m+1) \cdot W_N^{(2m+1)k} = \quad \textit{Eqn. 9}$$

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

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

$$X(k) = \sum_{m=0}^{N/2-1} a(m) \cdot W_{N/2}^{mk} + W_N^{k} \sum_{m=0}^{N/2-1} b(m) \cdot W_{N/2}^{mk} = A(k) + W_N^{k} B(k) \quad \textit{Eqn. 10}$$

$$0 \le k < N$$

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 of the DFT (Equation 7), the outputs for $N/2 \le k < N$ from a DFT of length *N*/2 are identical to the outputs for $0 \le k < N/2$. That is, $A(k + N/2) = A(k)$ and $B(k + N/2) = B(k)$ for $0 \le k < N/2$. In addition, the factor $W_N^{k+N/2} = -W_N^{k}$ thanks the to symmetral property (Equation 6). Thus, the whole DFT can be calculated as follows:

$$X(k) = A(k) + W_N^{k} B(k)$$

$$X(k + N/2) = A(k) - W_N^{k} B(k) \quad \textit{Eqn. 11}$$

$$0 \le k < N/2$$

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 “butterfly” in this context (see Figure 1).

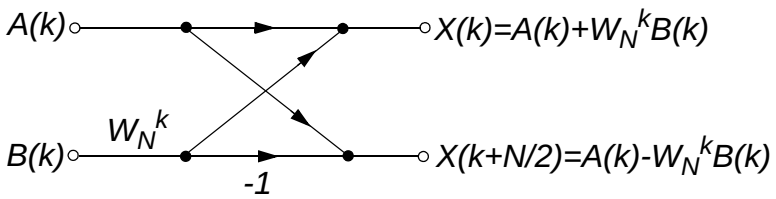

Figure 1. Basic butterfly computation in the DIT FFT algorithm

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.

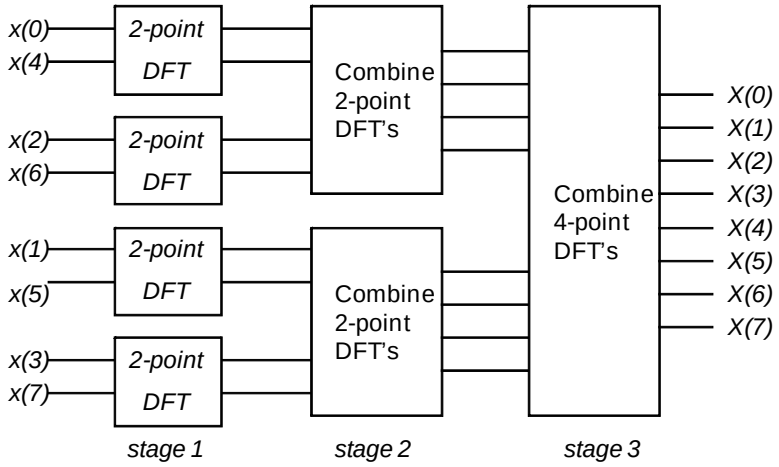

Figure 2. Decomposition of an 8-point DFT

For illustrative purposes, Figure 2 depicts the computation of an $N = 8$-point DFT. We observe that the 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 consists of *N*/2 butterfly computations.The combination of the smaller DFTs to form the larger DFT for $N = 8$ is illustrated in Figure 3.

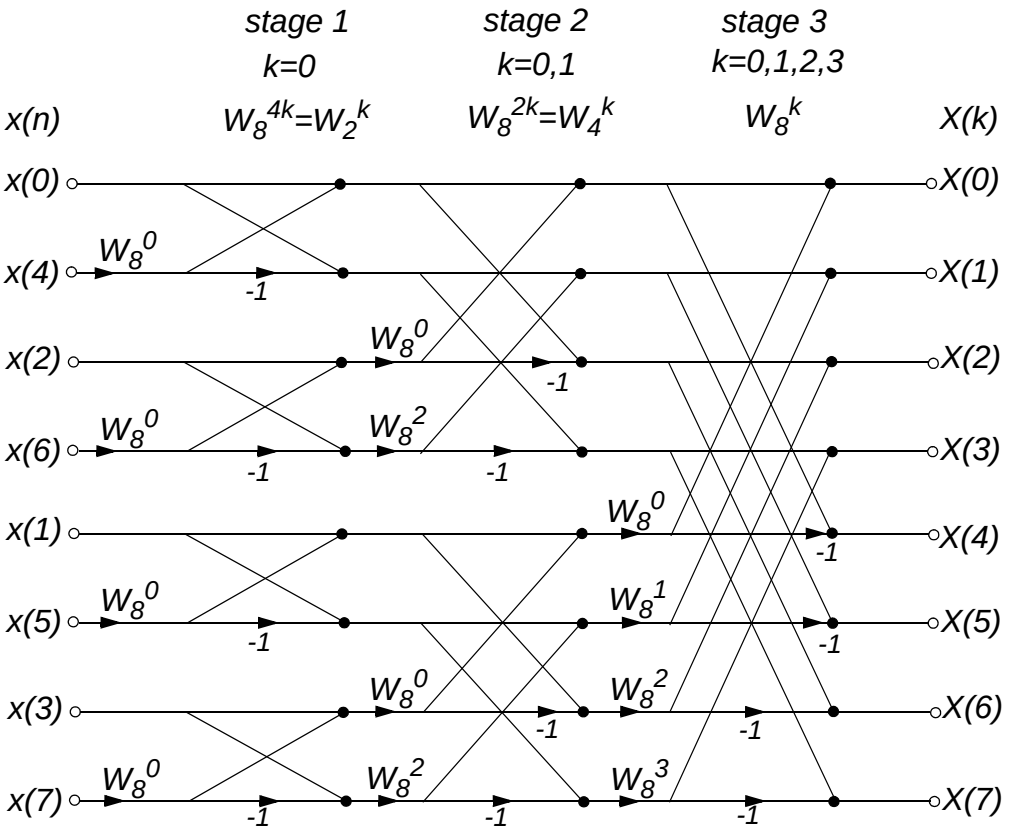

Figure 3. 8-point radix-2 DIT FFT algorithm data flow

Each dot represents a complex addition and each arrow represents a complex multiplication, as shown 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, and so on. It is also possible to represent it uniformly as a power of N (W_N), where N is the size of the input sequence $x(n)$. The context between both expressions is shown in 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 re-ordering of the input data sequence (see also the left side of Figure 2 and 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 a bit-reversal with an 8-point input sequence.

Table 1. Bit reversal with an 8-point input sequence

Decimal number	0	1	2	3	4	5	6	7
Binary equivalent	000	001	010	011	100	101	110	111
Bit reversed binary	000	100	010	110	001	101	011	111
Decimal equivalent	0	4	2	6	1	5	3	7

It is important to note that this FFT algorithm is of an “in-place” type, 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 may have a slightly different shape after the application of window functions 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 the frequency of the input signal to be well-known. In metering applications, this is accomplished by measuring a period of line voltage.

The detection of a signal (mains) period may be performed by a zero-crossing 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-crossings is a method used for frequency measurement of an input signal (the line voltage).
For example, the ZCD circuit may be realized using an analog 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 the zero-crossings is measured using a timer in the software. The 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-crossing 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).

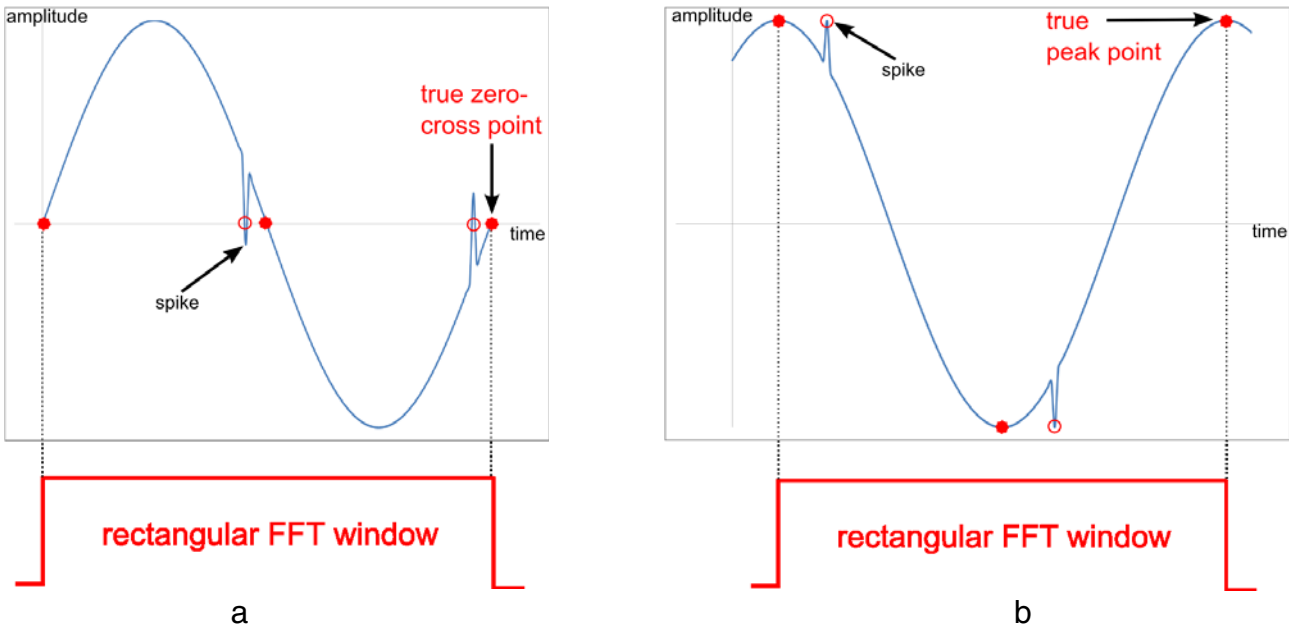

Figure 4. Zero-crossing point vs. peak point detection

It is also useful 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-crossings or peak detection.

In a practical implementation, it is better to measure the time between several true zero-crossings 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 useful algorithms to do the same thing as the DFT, but in much shorter time. Where the DFT needs N^2 complex multiplications (see 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 also becomes very large. Therefore, the FFT is simply a faster 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 for 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. However, this application note describes only the radix-2 DIT FFT algorithm.

4 Using 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 real part and 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 the imaginary part as the vertical component (see Figure 5).

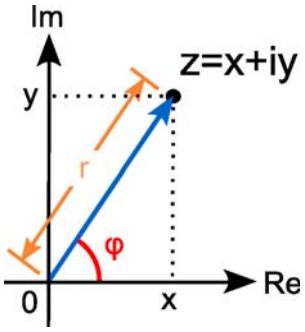

Figure 5. A graphical representation of a complex number

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:

$$r = |z| = \sqrt{x^2 + y^2} \qquad \textit{Eqn. 12}$$

The argument or phase of *z* is defined as:

$$\varphi = \arg(z) = \operatorname{atan}\left(\frac{y}{x}\right) \qquad \textit{Eqn. 13}$$

Together, *r* and φ 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 with each harmonic. Regarding Equation 12, the total RMS values of current and voltage in the frequency domain are defined as:

$$I_{RMS} = \sqrt{\sum_{k=0}^{\frac{N}{2}-1} (I_{RE}^2(k) + I_{IM}^2(k))} \quad \textit{Eqn. 14}$$

$$U_{RMS} = \sqrt{\sum_{k=1}^{\frac{N}{2}-1} (U_{RE}^2(k) + U_{IM}^2(k))} \quad \textit{Eqn. 15}$$

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

NOTE

Voltage offset (zero harmonic) is not included in the U_{RMS} computing. This simplification can be used because of the zero offset in the mains.

4.3 Complex power computing

The 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 each other in a trigonometric form. This is called a power triangle (see Figure 6).

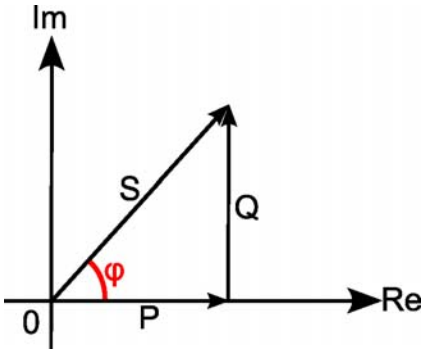

Figure 6. Power triangle

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

$$\bar{S} = P + jQ = \bar{U} \cdot \bar{I^*} \quad \textit{Eqn. 16}$$

Where $\overline{U}$ is a voltage vector ($\overline{U} = U_{RE} + jU_{IM}$) and $\overline{I^*}$ is a complex conjugate current vector ($\overline{I^*} = I_{RE} - jI_{IM}$), both separately for each harmonic.

Regarding Equation 12, the length of a complex power (|S|) is actually the apparent power (VA). In terms of current and voltage phasors (FFT outputs), and in terms of Equation 16, the complex power in Cartesian form can be finally expressed as:

$$\bar{S} = P + jQ = \bar{U} \cdot \bar{I^*} = \sum_{k=1}^{\frac{N}{2}-1} (U_{RE}(k) + jU_{IM}(k)) \cdot (I_{RE}(k) - jI_{IM}(k)) =$$

$$= \sum_{k=1}^{\frac{N}{2}-1} (\underbrace{(I_{RE}(k) \cdot U_{RE}(k) + I_{IM}(k) \cdot U_{IM}(k))}_{\text{real part of complex power = active power (P)}} + \underbrace{j(U_{IM}(k) \cdot I_{RE}(k) - U_{RE}(k) \cdot I_{IM}(k))}_{\text{imaginary part of complex power = reactive power (Q)}}) \qquad \textit{Eqn. 17}$$

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 also be expressed in polar form as:

$$\bar{S} = \sum_{k=1}^{\frac{N}{2}-1} (\underbrace{(|I(k)| \cdot |U(k)| \cdot \cos(U_{\varphi}(k) - I_{\varphi}(k)))}_{\text{real part of complex power = active power (P)}} + j(\underbrace{|I(k)| \cdot |U(k)| \cdot \sin(U_{\varphi}(k) - I_{\varphi}(k)))}_{\text{imaginary part of complex power = reactive power (Q)}}) \qquad \textit{Eqn. 18}$$

Where:

- $|I(k)|$, $|U(k)|$ are magnitudes of k^{th} harmonics of current and voltage
- $I\varphi(k)$, $U\varphi(k)$ are phase shifts of k^{th} harmonics of current and voltage (with regards to the FFT window origin)

Note that the 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 missing in both formulas, because the current values ($I_{RE}(0)$, $I_{IM}(0)$, $|I(0)|$) are multiplied by zero.

The magnitude of the complex power (|S|) is the apparent power (volt-ampere). In a pure sinusoidal system with no higher harmonics, the apparent power calculation gives the correct result. After harmonics are encoutered in the system, the apparent power calculation looses accuracy. In this case, it is better to use the total apparent power (volt-ampere). The total apparent power is defined as a product of the RMS values of voltage and current:

$$S_{tot} = U_{RMS} \cdot I_{RMS} \quad \textit{Eqn. 19}$$

Where:

- U_{RMS} is the RMS value of the line voltage [V]
- I_{RMS} is the RMS value of the line current [A]

4.4 Energy computing

Both the active and reactive energies are computed from the powers (active, reactive) by accumulating these powers per time unit (mostly per one hour). The computing formula is then expressed as:

$$\Delta\text{Energy} = \frac{\text{Power}}{(\text{Frequency} \cdot 3600)} \quad \textit{Eqn. 20}$$

Where:

- *ΔEnergy* is the active or reactive energy increment per one computing cycle [Wh/VARh]
- *Power* is the instantaneous active or reactive power measured during one cycle [W/VAR]
- *Frequency* is the line frequency [Hz]
- 3600 is a ‘hour’ coefficient

4.5 Power factor computing

In electrical engineering, the power factor of an AC electrical power system is defined as the ratio of the real (active) power flowing to the load, to the apparent power in the circuit, and it is a dimensionless number ranging between −1 and 1.

$$PF = \frac{P}{S} \quad \textit{Eqn. 21}$$

Where:

- *P* is the instantaneous active (real) power [W]
- *S* is the instantaneous apparent power [VA]

Real power is the capacity of the circuit for performing work in a particular time. Apparent power is the product of the current and voltage of the circuit. Due to energy stored in the load and returned to the source, or due to a non-linear load that distorts the wave shape of the current drawn from the source, the apparent power will be greater than the real power. A negative power factor can occur when the device which is normally the load generates power, which then flows back towards the device which is normally considered the generator (see Table 2).

Table 2. Power factor range vs. energy flow direction

Quadrant	Power factor range	Powers sign	Load mode	I to U phase shift
I	0...1	+P, +Q	Motor mode with inductive load	Lagging current
II	-1...0	-P, +Q	Inductive acting generator mode	Leading current
III	-1...0	-P, -Q	Capacitive acting generator mode	Lagging current
IV	0...1	+P, -Q	Motor mode with capacitive load	Leading current

4.6 Total Harmonic Distortion computing

In electrical engineering, the Total Harmonic Distortion (THD) is the ratio of the RMS amplitude of a set of higher harmonic frequencies to the RMS amplitude of the first harmonic, or fundamental, frequency. Therefore, the THD is an indicator of the signal distortion. For voltage and current signals, the THD calculation formulas, using frequency components, are as follows:

$$THD_U = \sqrt{\sum_{k=2}^{\frac{N}{2}-1} \frac{(U_{RE}^2(k) + U_{IM}^2(k))}{(U_{RE}^2(1) + U_{IM}^2(1))}} \cdot 100 \qquad \textit{Eqn. 22}$$

$$THD_I = \sqrt{\sum_{k=2}^{\frac{N}{2}-1} \frac{(I_{RE}^2(k) + I_{IM}^2(k))}{(I_{RE}^2(1) + I_{IM}^2(1))}} \cdot 100 \qquad \textit{Eqn. 23}$$

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

The end result of previous equations is a percentage. The higher the percentage, the higher the signal distortion is.

5 Metering library

This section describes the metering library implementation of the FFT-based metering algorithm. This application note is delivered together with the metering library and test applications. The library comprises several functions with a unique Application Programming Interface (API) for most frequent power meter topologies; that is, one-phase, two-phase (Form-12S), and three-phase. The library is provided in object format (*.*a* and/or *.*lib* files) and the test applications in C-source code. The function prototypes, as well as internal data structures, are declared in the *meterlibfft.h* header file. A simple block diagram of the whole FFT computing process in a typical one-phase power meter application is depicted in Figure 7. For a practical implementation of this computing process into the real power meter, see Freescale reference designs [4], [5], or [6] in Section 7, “References.”

NOTE

The IAR Embedded Workbench® for ARM® (version 7.40.1) tool was used to obtain performance data for all library functions (see performance tables below). The code was compiled with full optimization of execution speed for the MKM34Z128 target (ARM Cortex®-M0+ core) and MKM34Z256 target (ARM Cortex-M0+ core with MMAU). The device was clocked at 24 MHz using the Frequency-Locked Loop (FLL) module operating in FLL Engaged External (FEE) mode, driven by an external 32.768 kHz crystal. The measured execution times were recalculated to core clock cycles.

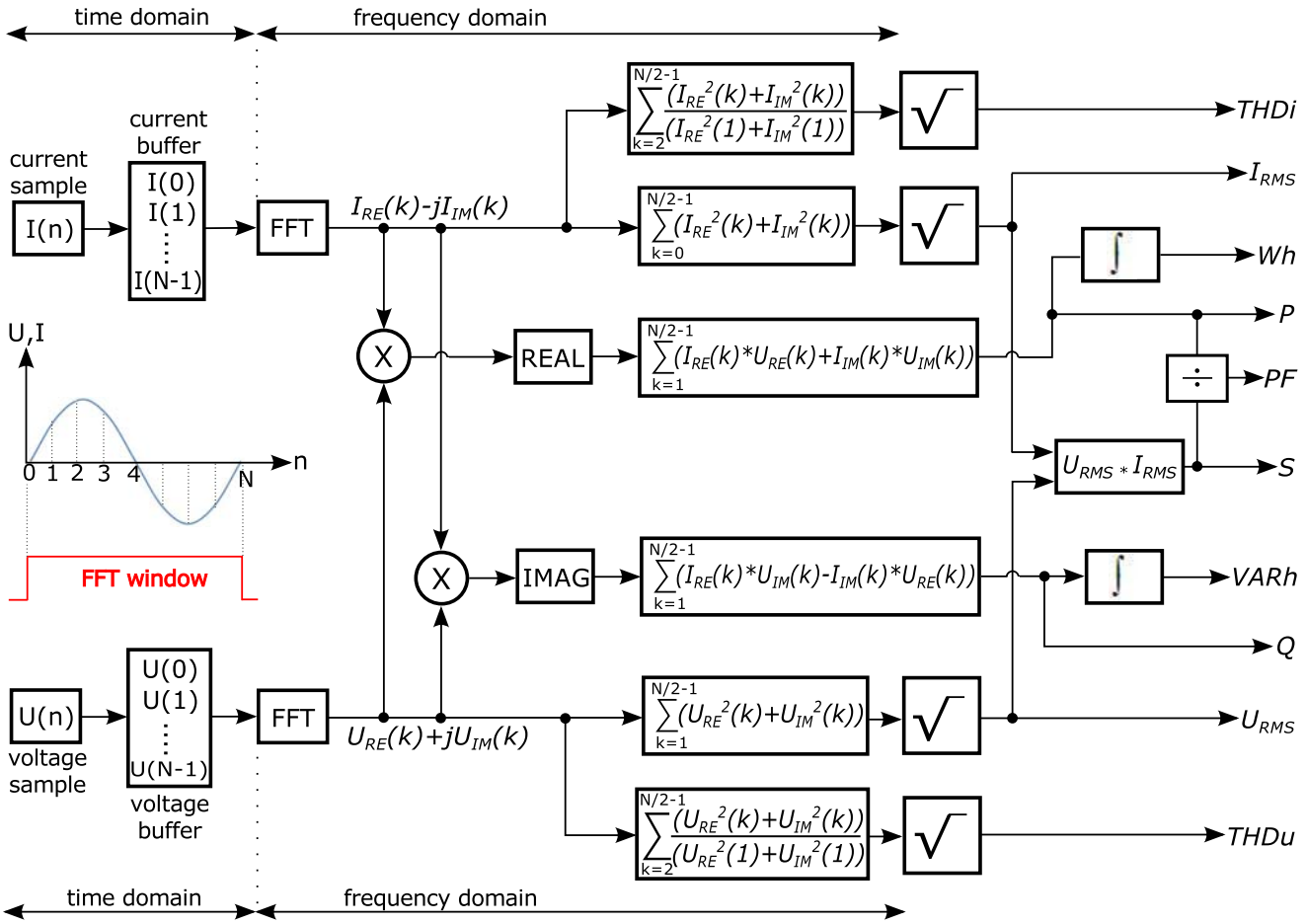

Figure 7. Block diagram of the one-phase power meter computing process based on the FFT

5.1 Core architecture and compiler support

This high-performance FFT-based metering library supports ARM Cortex-M0+ and Cortex-M4 cores. In addition to standard cores, the library also supports the Memory-Mapped Arithmetic Unit (MMAU), a hardware math module designed by Freescale to accelerate the execution of specific metering algorithms.

The default installation folder of the metering library is *C:\Freescale\METERLIBFFT_R4_0_0*. Table 3 lists all the necessary header files, library files, and their locations, relative to the default installation folder.

Add these files and paths into your project workspace to successfully integrate this metering library into your application.

Table 3. FFT-based metering library integration

Include files and libraries			METERLIBFFT		
			Cortex-M0+ w/o MMAU	Cortex-M0+ w/ MMAU	Cortex-M4
include	files	iar	fraclib.h meterlibfft.h		
		armcc			
		gcc			
	paths	iar	..\lib\fraclib\inc ..\lib\fraclib\inc\cm0p ..\lib\meterlibFFT\inc	..\lib\fraclib\inc ..\lib\fraclib\inc\cm0p_mmau ..\lib\fraclib\inc\cm0p_mmau\iar ..\lib\meterlibFFT\inc	..\lib\fraclib\inc ..\lib\fraclib\inc\cm4 ..\lib\meterlibFFT\inc
		armcc		..\lib\fraclib\inc ..\lib\fraclib\inc\cm0p_mmau ..\lib\fraclib\inc\cm0p_mmau\armcc ..\lib\meterlibFFT\inc	
		gcc		..\lib\fraclib\inc ..\lib\fraclib\inc\cm0p_mmau ..\lib\fraclib\inc\cm0p_mmau\gcc ..\lib\meterlibFFT\inc	
library	files	iar	fraclib_cm0p_iar.a meterlibFFT_cm0p_iar.a	fraclib_cm0p_mmau_iar.a meterlibFFT_cm0p_mmau_iar.a	fraclib_cm4_iar.a meterlibFFT_cm4_iar.a
		armcc	fraclib_cm0p_armcc.lib meterlibFFT_cm0p_armcc.lib	fraclib_cm0p_mmau_armcc.lib meterlibFFT_cm0p_mmau_armcc.lib	fraclib_cm4_armcc.lib meterlibFFT_cm4_armcc.lib
		gcc	fraclib_cm0p_gcc.a meterlibFFT_cm0p_gcc.a	fraclib_cm0p_mmau_gcc.a meterlibFFT_cm0p_mmau_gcc.a	fraclib_cm4_gcc.a meterlibFFT_cm4_gcc.a
	paths	iar	..\lib\fraclib ..\lib\meterlibFFT		
		armcc			
		gcc			

5.2 Function API summary

The following subsections describe functions API defined in the FFT-based metering library. Prototypes of all functions and internal data structures are declared in the *meterlibfft.h* header file.

5.2.1 One-phase power meter API

- void **METERLIBFFT1PH_CalcMain** (tMETERLIBFFT1PH_DATA *p)
  FFT Calculation and Signal-Conditioning Processing Function
- long **METERLIBFFT1PH_CalcVarHours** (tMETERLIBFFT1PH_DATA *p, unsigned long *varh_i, unsigned long *varh_e, unsigned long frequency)
  Reactive Energy Calculation Function

- long **METERLIBFFT1PH_CalcWattHours** (tMETERLIBFFT1PH_DATA *p, unsigned long *wh_i, unsigned long *wh_e, unsigned long frequency)

  Active Energy Calculation Function
- void **METERLIBFFT1PH_GetAvrgValues** (tMETERLIBFFT1PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Averaged Variables Reading Function
- void **METERLIBFFT1PH_GetInstValues** (tMETERLIBFFT1PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Instantaneous Variables Reading Function
- void **METERLIBFFT1PH_GetMagnitudes** (tMETERLIBFFT1PH_DATA *p, unsigned long magn_fft)

  Harmonic Magnitudes Calculation Function
- void **METERLIBFFT1PH_GetPhases** (tMETERLIBFFT1PH_DATA *p, unsigned long ph_fft)

  Harmonic Phase Shifts Calculation Function
- void **METERLIBFFT1PH_InitAuxBuff** (tMETERLIBFFT1PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long *ph_u, long *ph_i)

  Auxiliary Buffers Initialization Function
- void **METERLIBFFT1PH_InitMainBuff** (tMETERLIBFFT1PH_DATA *p, Frac24 *u_re, Frac24 *i_re, Frac24 *u_im, Frac24 *i_im, long *shift)

  Main Buffers Initialization Function
- long **METERLIBFFT1PH_InitParam** (tMETERLIBFFT1PH_DATA *p, unsigned long samples, unsigned long sensor, unsigned long kwh_cnt, unsigned long kvarh_cnt, unsigned long en_res)

  Parameters Initialization Function
- long **METERLIBFFT1PH_Interpolation** (tMETERLIBFFT1PH_DATA *p, unsigned long u_ord, unsigned long i_ord, unsigned long samples_inp)

  Interpolation Function
- long **METERLIBFFT1PH_SetCalibCoeff** (tMETERLIBFFT1PH_DATA *p, double u_max, double i_max, Frac24 *i_offs, double p_offs, double q_offs)

  Set Calibration Coefficients Function

5.2.2 Two-phase power meter API

- void **METERLIBFFT2PH_CalcMain** (tMETERLIBFFT2PH_DATA *p)

  FFT Calculation and Signal-Conditioning Processing Function
- long **METERLIBFFT2PH_CalcVarHours** (tMETERLIBFFT2PH_DATA *p, unsigned long *varh_i, unsigned long *varh_e, unsigned long frequency)

  Reactive Energy Calculation Function
- long **METERLIBFFT2PH_CalcWattHours** (tMETERLIBFFT2PH_DATA *p, unsigned long *wh_i, unsigned long *wh_e, unsigned long frequency)

  Active Energy Calculation Function

- void **METERLIBFFT2PH_GetAvrgValuesPh1** (tMETERLIBFFT2PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Averaged Variables Phase 1 Reading Function
- void **METERLIBFFT2PH_GetAvrgValuesPh2** (tMETERLIBFFT2PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Averaged Variables Phase 2 Reading Function
- void **METERLIBFFT2PH_GetInstValuesPh1** (tMETERLIBFFT2PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Instantaneous Variables Phase 1 Reading Function
- void **METERLIBFFT2PH_GetInstValuesPh2** (tMETERLIBFFT2PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Instantaneous Variables Phase 2 Reading Function
- void **METERLIBFFT2PH_GetMagnitudesPh1** (tMETERLIBFFT2PH_DATA *p, unsigned long magn_fft)

  Harmonic Magnitudes Calculation Function for the Phase 1
- void **METERLIBFFT2PH_GetMagnitudesPh2** (tMETERLIBFFT2PH_DATA *p, unsigned long magn_fft)

  Harmonic Magnitudes Calculation Function for the Phase 2
- void **METERLIBFFT2PH_GetPhasesPh1** (tMETERLIBFFT2PH_DATA *p, unsigned long ph_fft)

  Harmonic Phase Shifts Calculation Function for the Phase 1
- void **METERLIBFFT2PH_GetPhasesPh2** (tMETERLIBFFT2PH_DATA *p, unsigned long ph_fft)

  Harmonic Phase Shifts Calculation Function for the Phase 2
- void **METERLIBFFT2PH_InitAuxBuffPh1** (tMETERLIBFFT2PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long *ph_u, long *ph_i)

  Auxiliary Buffers Initialization Function for the Phase 1
- void **METERLIBFFT2PH_InitAuxBuffPh2** (tMETERLIBFFT2PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long *ph_u, long *ph_i)

  Auxiliary Buffers Initialization Function for the Phase 2
- void **METERLIBFFT2PH_InitMainBuffPh1** (tMETERLIBFFT2PH_DATA *p, Frac24 *u_re, Frac24 *i_re, Frac24 *u_im, Frac24 *i_im, long *shift)

  Main Buffers Initialization Function for the Phase 1
- void **METERLIBFFT2PH_InitMainBuffPh2** (tMETERLIBFFT2PH_DATA *p, Frac24 *u_re, Frac24 *i_re, Frac24 *u_im, Frac24 *i_im, long *shift)

  Main Buffers Initialization Function for the Phase 2
- long **METERLIBFFT2PH_InitParam** (tMETERLIBFFT2PH_DATA *p, unsigned long samples, unsigned long sensor, unsigned long kwh_cnt, unsigned long kvarh_cnt, unsigned long en_res)

Parameters Initialization Function

- long **METERLIBFFT2PH_Interpolation** (tMETERLIBFFT2PH_DATA *p, unsigned long u_ord, unsigned long i_ord, unsigned long samples_inp)

  Interpolation Function
- long **METERLIBFFT2PH_SetCalibCoeffPh1** (tMETERLIBFFT2PH_DATA *p, double u_max, double i_max, Frac24 *i_offs, double p_offs, double q_offs)

  Set Calibration Coefficients Function for the phase 1
- long **METERLIBFFT2PH_SetCalibCoeffPh2** (tMETERLIBFFT2PH_DATA *p, double u_max, double i_max, Frac24 *i_offs, double p_offs, double q_offs)

  Set Calibration Coefficients Function for the phase 2

5.2.3 Three-phase power meter API

- void **METERLIBFFT3PH_CalcMain** (tMETERLIBFFT3PH_DATA *p)

  FFT Calculation and Signal-Conditioning Processing Function
- long **METERLIBFFT3PH_CalcVarHours** (tMETERLIBFFT3PH_DATA *p, unsigned long *varh_i, unsigned long *varh_e, unsigned long frequency)

  Reactive Energy Calculation Function
- long **METERLIBFFT3PH_CalcWattHours** (tMETERLIBFFT3PH_DATA *p, unsigned long *wh_i, unsigned long *wh_e, unsigned long frequency)

  Active Energy Calculation Function
- void **METERLIBFFT3PH_GetAvrgValuesPh1** (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Averaged Variables Phase 1 Reading Function
- void **METERLIBFFT3PH_GetAvrgValuesPh2** (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Averaged Variables Phase 2 Reading Function
- void **METERLIBFFT3PH_GetAvrgValuesPh3** (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Averaged Variables Phase 3 Reading Function
- void **METERLIBFFT3PH_GetInstValuesPh1** (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Instantaneous Variables Phase 1 Reading Function
- void **METERLIBFFT3PH_GetInstValuesPh2** (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

  Non-Billing (U,I,P,Q,S,PF,THD) Instantaneous Variables Phase 2 Reading Function
- void **METERLIBFFT3PH_GetInstValuesPh3** (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms, double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i)

Non-Billing (U,I,P,Q,S,PF,THD) Instantaneous Variables Phase 3 Reading Function

- void **METERLIBFFT3PH_GetMagnitudesPh1** (tMETERLIBFFT3PH_DATA *p, unsigned long magn_fft)

  Harmonic Magnitudes Calculation Function for the Phase 1
- void **METERLIBFFT3PH_GetMagnitudesPh2** (tMETERLIBFFT3PH_DATA *p, unsigned long magn_fft)

  Harmonic Magnitudes Calculation Function for the Phase 2
- void **METERLIBFFT3PH_GetMagnitudesPh3** (tMETERLIBFFT3PH_DATA *p, unsigned long magn_fft)

  Harmonic Magnitudes Calculation Function for the Phase 3
- void **METERLIBFFT3PH_GetPhasesPh1** (tMETERLIBFFT3PH_DATA *p, unsigned long ph_fft)

  Harmonic Phase Shifts Calculation Function for the Phase 1
- void **METERLIBFFT3PH_GetPhasesPh2** (tMETERLIBFFT3PH_DATA *p, unsigned long ph_fft)

  Harmonic Phase Shifts Calculation Function for the Phase 2
- void **METERLIBFFT3PH_GetPhasesPh3** (tMETERLIBFFT3PH_DATA *p, unsigned long ph_fft)

  Harmonic Phase Shifts Calculation Function for the Phase 3
- void **METERLIBFFT3PH_InitAuxBuffPh1** (tMETERLIBFFT3PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long *ph_u, long *ph_i)

  Auxiliary Buffers Initialization Function for the Phase 1
- void **METERLIBFFT3PH_InitAuxBuffPh2** (tMETERLIBFFT3PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long *ph_u, long *ph_i)

  Auxiliary Buffers Initialization Function for the Phase 2
- void **METERLIBFFT3PH_InitAuxBuffPh3** (tMETERLIBFFT3PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long *ph_u, long *ph_i)

  Auxiliary Buffers Initialization Function for the Phase 3
- void **METERLIBFFT3PH_InitMainBuffPh1** (tMETERLIBFFT3PH_DATA *p, Frac24 *u_re, Frac24 *i_re, Frac24 *u_im, Frac24 *i_im, long *shift)

  Main Buffers Initialization Function for the Phase 1
- void **METERLIBFFT3PH_InitMainBuffPh2** (tMETERLIBFFT3PH_DATA *p, Frac24 *u_re, Frac24 *i_re, Frac24 *u_im, Frac24 *i_im, long *shift)

  Main Buffers Initialization Function for the Phase 2
- void **METERLIBFFT3PH_InitMainBuffPh3** (tMETERLIBFFT3PH_DATA *p, Frac24 *u_re, Frac24 *i_re, Frac24 *u_im, Frac24 *i_im, long *shift)

  Main Buffers Initialization Function for the Phase 3
- long **METERLIBFFT3PH_InitParam** (tMETERLIBFFT3PH_DATA *p, unsigned long samples, unsigned long sensor, unsigned long kwh_cnt, unsigned long kvarh_cnt, unsigned long en_res)

Parameters Initialization Function

- long **METERLIBFFT3PH_Interpolation** (tMETERLIBFFT3PH_DATA *p, unsigned long u_ord, unsigned long i_ord, unsigned long samples_inp)

  Interpolation Function
- long **METERLIBFFT3PH_SetCalibCoeffPh1** (tMETERLIBFFT3PH_DATA *p, double u_max, double i_max, Frac24 *i_offs, double p_offs, double q_offs)

  Set Calibration Coefficients Function for the phase 1
- long **METERLIBFFT3PH_SetCalibCoeffPh2** (tMETERLIBFFT3PH_DATA *p, double u_max, double i_max, Frac24 *i_offs, double p_offs, double q_offs)

  Set Calibration Coefficients Function for the phase 2
- long **METERLIBFFT3PH_SetCalibCoeffPh3** (tMETERLIBFFT3PH_DATA *p, double u_max, double i_max, Frac24 *i_offs, double p_offs, double q_offs)

  Set Calibration Coefficients Function for the phase 3
- long **METERLIBFFT3PH_GetRotation**(tMETERLIBFFT3PH_DATA *p, double *u12_ph, double *u13_ph, double *u23_ph)

  Angle and Rotation Calculation Function

5.2.4 Common power meter API

- void **METERLIBFFT_SetEnergy** (p, whi, whe, varhi, varhe)

  Common Set/Clear Energy Counters Define

5.3 METERLIBFFT_CalcMain

Firstly, these functions execute the main FFT calculation processing for both voltage and current signals; 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 data are computed internally in the Cartesian data format.

Secondly, these functions execute the additional current signal conditioning processing, including software phase shift correction (if needed), and signal integration (if needed) for derivative type of current sensors. All this additional processing can eliminate the current sensor inaccuracies and sensor features using software computing in the frequency domain.

Finally, these functions execute additional postprocessing, such as scaling to engineering units and averaging of all non-billing values, with saving them to the internal data structure. All previous actions are done separately for each phase.

For a proper calculation, these functions need instantaneous voltage and current samples to be measured periodicaly, and saved to the “time domain” buffers during previous signal period. Both these buffers, addressed by *u_re* and *i_re* pointers, will be rewritten by the frequency domain real part values after the calculation, while the imaginary frequency domain parts of the result are saved to separate buffers set by *u_im* and *i_im* pointers. Pointers to all these buffers must be initialized by functions described in Section 5.12, “METERLIBFFT_InitParam.” The first FFT buffers’ position matches the zero harmonic, and so on.

5.3.1 Syntax

```
#include “meterlibfft.h”
void METERLIBFFT1PH_CalcMain (tMETERLIBFFT1PH_DATA *p);
void METERLIBFFT2PH_CalcMain (tMETERLIBFFT2PH_DATA *p);
void METERLIBFFT3PH_CalcMain (tMETERLIBFFT3PH_DATA *p);
```

5.3.2 Arguments

Table 4. METERLIBFFT_CalcMain functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure

5.3.3 Return

These functions do not return any arguments.

5.3.4 Calling order

All of these functions should be called periodically in a defined interval, which depends on the line frequency and/or the multiplied ADC sampling rate (synchronous or asynchronous processing). The one-shot mandatory parameter initialization must be executed before calling these functions. The periodic interpolation processing must (may) be done closely before calling these functions. The energy calculation processing should be executed closely after calling these functions.

5.3.5 Performance

Table 5. METERLIBFFT_CalcMain functions performance for the CM0+ core

Function name	Code size[1] [B]	Clock cycles[2]		
		Basic mode	Integration mode	Phase shift correction mode
METERLIBFFT1PH_CalcMain	2524	95465	98343	95465+2880**harm*
METERLIBFFT2PH_CalcMain	2938	190450	196207	190450+6230**harm*
METERLIBFFT3PH_CalcMain	3376	285435	295030	285435+9600**harm*
Note: *harm* is a total number of shifted harmonics				

1 Code size of the sine lookup table (4 KB) isn’t included

2 Valid for 64 input samples, 32 output harmonics, test vector of U and I signals contained the 1st and the 5th harmonics

Table 6. METERLIBFFT_CalcMain functions performance for the CM0+ core with MMAU

Function name	Code size[1] [B]	Clock cycles[2]		
		Basic mode	Integration mode	Phase shift correction mode
METERLIBFFT1PH_CalcMain	3094	49891	52770	49891+840**harm*
METERLIBFFT2PH_CalcMain	3508	99782	105539	99782+1920**harm*
METERLIBFFT3PH_CalcMain	3946	149674	158309	149674+2640**harm*
Note: *harm* is a total number of shifted harmonics				

1 Code size of the sine lookup table (4 KB) isn't included

2 Valid for 64 input samples, 32 output harmonics, test vector of U and I signals contained the 1st and the 5th harmonics

5.4 METERLIBFFT_CalcVarHours

These functions perform reactive energies (import, export) calculation for all phases altogether. Both energies are computed from instantaneous reactive power increment of each phase by accumulating these powers per time unit. While the total import reactive energy is computed from the positive reactive powers increments, the total export reactive energy is computed from negative reactive powers increments. The output energy resolution (*varh_i* and *varh_e*) depends on *en_res* parameter, set by the function described in Section 5.12, "METERLIBFFT_InitParam."

5.4.1 Syntax

```
#include "meterlibfft.h"

long METERLIBFFT1PH_CalcVarHours (tMETERLIBFFT1PH_DATA *p, unsigned long *varh_i, unsigned long
*varh_e, unsigned long frequency);

long METERLIBFFT2PH_CalcVarHours (tMETERLIBFFT2PH_DATA *p, unsigned long *varh_i, unsigned long
*varh_e, unsigned long frequency);

long METERLIBFFT3PH_CalcVarHours (tMETERLIBFFT3PH_DATA *p, unsigned long *varh_i, unsigned long
*varh_e, unsigned long frequency);
```

5.4.2 Arguments

Table 7. METERLIBFFT_CalcVarHours functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
unsigned long	varh_i	out	Pointer to the LCD import reactive energy counter
unsigned long	varh_e	out	Pointer to the LCD export reactive energy counter
unsigned long	frequency	in	Line frequency [mHz], for example 50000 = 50.000 Hz

5.4.3 Return

When positive, the function returns the reactive energy LED flashing frequency resolution (in mHz) for the current line period (only one LED flashing per one period is allowed). This can be used for low-jitter pulse output generation using software and timer (patented method).

When negative, no output pulse generation is needed in the current period.

5.4.4 Calling order

All of these functions should be called periodically in a defined interval, which depends on the line frequency and/or the multiplied ADC sampling rate. Anyway, these functions must (may) be called closely after the main (FFT) calculation processing. The one-shot mandatory parameter initialization must be done before calling these functions.

5.4.5 Performance

Table 8. METERLIBFFT_CalcVarHours functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles	Code size [B]	Clock cycles
METERLIBFFT1PH_CalcVarHours	292	698	292	696
METERLIBFFT2PH_CalcVarHours	372	751	372	744
METERLIBFFT3PH_CalcVarHours	490	883	490	878

5.5 METERLIBFFT_CalcWattHours

These functions execute active energies (import, export) calculation for all phases altogether.
Both energies are computed from instantaneous active power increment of each phase by accumulating these powers per time unit. While the total import active energy is computed from the positive active power increments, the total export active energy is computed from negative active power increments. The output energy resolution (*wh_i* and *wh_e*) depends on *en_res* parameter set by the function described in Section 5.12, “METERLIBFFT_InitParam.”

5.5.1 Syntax

```
#include “meterlibfft.h”

long METERLIBFFT1PH_CalcWattHours (tMETERLIBFFT1PH_DATA *p, unsigned long *wh_i, unsigned long
*wh_e, unsigned long frequency);

long METERLIBFFT2PH_CalcWattHours (tMETERLIBFFT2PH_DATA *p, unsigned long *wh_i, unsigned long
*wh_e, unsigned long frequency);

long METERLIBFFT3PH_CalcWattHours (tMETERLIBFFT3PH_DATA *p, unsigned long *wh_i, unsigned long
*wh_e, unsigned long frequency);
```

5.5.2 Arguments

Table 9. METERLIBFFT_CalcWattHours functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
unsigned long	wh_i	out	Pointer to the LCD import active energy counter
unsigned long	wh_e	out	Pointer to the LCD export active energy counter
unsigned long	frequency	in	Line frequency [mHz], for example 50000 = 50.000 Hz

5.5.3 Return

When positive, the function returns active energy LED flashing frequency resolution (in MHz) for the current line period (only one LED flashing per one period is allowed). This can be used for low-jitter pulse output generation using software and timer (patented method). When negative, no output pulse generation is needed in the current period.

5.5.4 Calling order

All of these functions should be called periodically in a defined interval, which depends on the line frequency and/or the multiplied ADC sampling rate. Anyway, these functions should be called closely after the main (FFT) calculation processing. The one-shot mandatory parameter initialization must be done before calling these functions.

5.5.5 Performance

Table 10. METERLIBFFT_CalcWattHours functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles	Code size [B]	Clock cycles
METERLIBFFT1PH_CalcWattHours	292	698	292	696
METERLIBFFT2PH_CalcWattHours	372	751	372	744
METERLIBFFT3PH_CalcWattHours	490	883	490	878

5.6 METERLIBFFT_GetAvrgValues

These functions return all averaged non-billing values, scaled to engineering units for each phase separetely. These values can be used for LCD showing, remote data visualization, and so on.

5.6.1 Syntax

```
#include “meterlibfft.h”
```

```
void METERLIBFFT1PH_GetAvrgValues (tMETERLIBFFT1PH_DATA *p, double *urms, double *irms, double
*w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT2PH_GetAvrgValuesPh1 (tMETERLIBFFT2PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT2PH_GetAvrgValuesPh2 (tMETERLIBFFT2PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT3PH_GetAvrgValuesPh1 (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT3PH_GetAvrgValuesPh2 (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT3PH_GetAvrgValuesPh3 (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);
```

5.6.2 Arguments

Table 11. METERLIBFFT_GetAvrgValues functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
double	urms	out	Pointer to the average RMS line voltage value in volts
double	irms	out	Pointer to the average RMS line current value in amperes
double	w	out	Pointer to the average active power (P) value in watts
double	var	out	Pointer to the average reactive power (Q) value in volt-amperes-reactive
double	va	out	Pointer to the average unsigned apparent power value (S) in volt-amperes
double	pf	out	Pointer to the average power factor value (dimensionless quantity)
double	thd_u	out	Pointer to the average THD voltage value in percent
double	thd_i	out	Pointer to the average THD current value in percent

5.6.3 Return

These functions do not return any arguments.

5.6.4 Calling order

Calling frequency of these functions should be in the range < 0.004 Hz – line frequency >. In case of a lower calling frequency, the internal counters may overflow. In this case, the first dummy reading is neccessary for clearing all internal counters. In case of a higher calling frequency, all output values will equal zero. All output values are scaled to engineering units in a double precision form.

5.6.5 Performance

Table 12. METERLIBFFT_GetAvrgValues functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles[1]	Code size [B]	Clock cycles[1]
METERLIBFFT1PH_GetAvrgValues	296	6764	296	6764
METERLIBFFT2PH_GetAvrgValuesPh1	296	6764	296	6764
METERLIBFFT2PH_GetAvrgValuesPh2	298		298	
METERLIBFFT3PH_GetAvrgValuesPh1	296	6764	296	6764
METERLIBFFT3PH_GetAvrgValuesPh2	298		298	
METERLIBFFT3PH_GetAvrgValuesPh3	298		298	

1 An average is computed every 25th cycle (two times per second)

5.7 METERLIBFFT_GetInstValues

These functions return all instantaneous non-billing values, scaled to engineering units for each phase separetely.

5.7.1 Syntax

```
#include “meterlibfft.h”

void METERLIBFFT1PH_GetInstValues (tMETERLIBFFT1PH_DATA *p, double *urms, double *irms, double
*w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT2PH_GetInstValuesPh1 (tMETERLIBFFT2PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT2PH_GetInstValuesPh2 (tMETERLIBFFT2PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT3PH_GetInstValuesPh1 (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT3PH_GetInstValuesPh2 (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);

void METERLIBFFT3PH_GetInstValuesPh3 (tMETERLIBFFT3PH_DATA *p, double *urms, double *irms,
double *w, double *var, double *va, double *pf, double *thd_u, double *thd_i);
```

5.7.2 Arguments

Table 13. METERLIBFFT_GetInstValues functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
double	urms	out	Pointer to the instantaneous RMS line voltage value in volts
double	irms	out	Pointer to the instantaneous RMS line current value in amperes
double	w	out	Pointer to the instantaneous active power (P) value in watts
double	var	out	Pointer to the instantaneous reactive power (Q) value in volt-amperes-reactive
double	va	out	Pointer to the instantaneous unsigned apparent power value (S) in volt-amperes
double	pf	out	Pointer to the instantaneous power factor value (dimensionless quantity)
double	thd_u	out	Pointer to the instantaneous THD voltage value per cent
double	thd_i	out	Pointer to the instantaneous THD current value per cent

5.7.3 Return

These functions do not return any arguments.

5.7.4 Calling order

These functions can be called anytime, with the best time being after the main (FFT) calculation processing. All output values are scaled to engineering units in a double precision form.

5.7.5 Performance

Table 14. METERLIBFFT_GetInstValues functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles	Code size [B]	Clock cycles
METERLIBFFT1PH_GetInstValues	232	5925	232	5925
METERLIBFFT2PH_GetInstValuesPh1	232	5925	232	5925
METERLIBFFT2PH_GetInstValuesPh2	234		234	
METERLIBFFT3PH_GetInstValuesPh1	232	5949	232	5925
METERLIBFFT3PH_GetInstValuesPh2	234		234	
METERLIBFFT3PH_GetInstValuesPh3	234		234	

5.8 METERLIBFFT_GetMagnitudes

These functions convert voltage and current data (computed by the function described in Section 5.3, "METERLIBFFT_CalcMain") from Cartesian data form to polar data form, and return voltage and current harmonic magnitudes for each phase separately. These values can be used for additional postprocessing and visualization. Voltage and current magnitudes are available at the two buffers addressed by *mag_u* and *mag_i* pointers (initialized by functions described in Section 5.10, "METERLIBFFT_InitAuxBuff"). The first buffer position matches the zero harmonic, and so on.

5.8.1 Syntax

```
#include "meterlibfft.h"
void METERLIBFFT1PH_GetMagnitudes (tMETERLIBFFT1PH_DATA *p, unsigned long magn_fft);
void METERLIBFFT2PH_GetMagnitudesPh1 (tMETERLIBFFT2PH_DATA *p, unsigned long magn_fft);
void METERLIBFFT2PH_GetMagnitudesPh2 (tMETERLIBFFT2PH_DATA *p, unsigned long magn_fft);
void METERLIBFFT3PH_GetMagnitudesPh1 (tMETERLIBFFT3PH_DATA *p, unsigned long magn_fft);
void METERLIBFFT3PH_GetMagnitudesPh2 (tMETERLIBFFT3PH_DATA *p, unsigned long magn_fft);
void METERLIBFFT3PH_GetMagnitudesPh3 (tMETERLIBFFT3PH_DATA *p, unsigned long magn_fft);
```

5.8.2 Arguments

Table 15. METERLIBFFT_GetMagnitudes functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
unsigned long	magn_fft	in	Number of required harmonic magnitudes in the range of < 1 – half of the input samples >

5.8.3 Return

These functions do not return any arguments.

5.8.4 Calling order

These functions may be called after the main (FFT) calculation processing. The one-shot mandatory and auxiliary parameter initialization must be done before calling these functions.

5.8.5 Performance

Table 16. METERLIBFFT_GetMagnitudes functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles[1]	Code size [B]	Clock cycles[1]
METERLIBFFT1PH_GetMagnitudes	234	17702	444	3142
METERLIBFFT2PH_GetMagnitudesPh1	234	17702	444	3142
METERLIBFFT2PH_GetMagnitudesPh2	236		446	
METERLIBFFT3PH_GetMagnitudesPh1	234	17702	444	3142
METERLIBFFT3PH_GetMagnitudesPh2	236		446	
METERLIBFFT3PH_GetMagnitudesPh3	240		450	

[1] Valid for 32 harmonic magnitudes

5.9 METERLIBFFT_GetPhases

These functions convert voltage and current data (computed by the function described in Section 5.3, “METERLIBFFT_CalcMain”) from Cartesian data form to polar data form and return voltage and current harmonic phase shifts for each phase separately. These values can be used for additional postprocessing and visualization. Voltage and current phase shifts are available at the two buffers addressed by *ph_u* and *ph_i* pointers (initialized by functions described in Section 5.10, “METERLIBFFT_InitAuxBuff“). The first buffer position matches the zero-harmonic, and so on.

5.9.1 Syntax

```
#include “meterlibfft.h”
void METERLIBFFT1PH_GetPhases (tMETERLIBFFT1PH_DATA *p, unsigned long ph_fft);
void METERLIBFFT2PH_GetPhasesPh1 (tMETERLIBFFT2PH_DATA *p, unsigned long ph_fft);
void METERLIBFFT2PH_GetPhasesPh2 (tMETERLIBFFT2PH_DATA *p, unsigned long ph_fft);
void METERLIBFFT3PH_GetPhasesPh1 (tMETERLIBFFT3PH_DATA *p, unsigned long ph_fft);
void METERLIBFFT3PH_GetPhasesPh2 (tMETERLIBFFT3PH_DATA *p, unsigned long ph_fft);
void METERLIBFFT3PH_GetPhasesPh3 (tMETERLIBFFT3PH_DATA *p, unsigned long ph_fft);
```

5.9.2 Arguments

Table 17. METERLIBFFT_GetPhases functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
unsigned long	ph_fft	in	Number of required harmonic phase shifts in the range < 1 – half of the input samples >

5.9.3 Return

These functions do not return any arguments.

5.9.4 Calling order

This function may be called after the main (FFT) calculation processing. The one-shot mandatory and auxiliary parameter initialization must be performed before calling these functions.

5.9.5 Performance

Table 18. METERLIBFFT_GetPhases functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size[1] [B]	Clock cycles[2]	Code size[1] [B]	Clock cycles[2]
METERLIBFFT1PH_GetPhases	346	15591	338	6188
METERLIBFFT2PH_GetPhasesPh1	346	15591	338	6188
METERLIBFFT2PH_GetPhasesPh2	348		340	
METERLIBFFT3PH_GetPhasesPh1	346	15639	338	6188
METERLIBFFT3PH_GetPhasesPh2	348		340	
METERLIBFFT3PH_GetPhasesPh3	352		344	

1 Code size of the arctangent lookup table (8 KB) isn't included
2 Valid for 32 harmonic phase shifts

5.10 METERLIBFFT_InitAuxBuff

These functions are used to initialize the pointers to the voltage and current magnitudes and phase shifts buffers. This initialization is performed separately for each particular phase. As these buffers don't have to be used for the main FFT computing, these initializations are only auxiliary, and should be performed if additional harmonic magnitudes and phase shifts computing is required. These computations are performed by functions described in Section 5.8, "METERLIBFFT_GetMagnitudes" and Section 5.9, "METERLIBFFT_GetPhases." The position of the first buffers matches the zero harmonic, and so on. The length of these buffers is optional, but their maximal length cannot exceed the number of FFT

harmonics (half of the input samples set by *samples* parameter in Section 5.12, "METERLIBFFT_InitParam").

5.10.1 Syntax

```
#include "meterlibfft.h"

void METERLIBFFT1PH_InitAuxBuff (tMETERLIBFFT1PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long
*ph_u, long *ph_i);

void METERLIBFFT2PH_InitAuxBuffPh1(tMETERLIBFFT2PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long
*ph_u, long *ph_i);

void METERLIBFFT2PH_InitAuxBuffPh2(tMETERLIBFFT2PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long
*ph_u, long *ph_i);

void METERLIBFFT3PH_InitAuxBuffPh1(tMETERLIBFFT3PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long
*ph_u, long *ph_i);

void METERLIBFFT3PH_InitAuxBuffPh2(tMETERLIBFFT3PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long
*ph_u, long *ph_i);

void METERLIBFFT3PH_InitAuxBuffPh3(tMETERLIBFFT3PH_DATA *p, Frac24 *mag_u, Frac24 *mag_i, long
*ph_u, long *ph_i);
```

5.10.2 Arguments

Table 19. METERLIBFFT_InitAuxBuff functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
Frac24	mag_u	out	Pointer to the harmonic magnitudes voltage buffer in Q0.23 data format
Frac24	mag_i	out	Pointer to the harmonic magnitudes current buffer in Q0.23 data format
long	ph_u	out	Pointer to the harmonic phase shifts voltage buffer in 0.001°, for example 45000 = 45.000°
long	ph_i	out	Pointer to the harmonic phase shifts current buffer in 0.001°, for example 45000 = 45.000°

5.10.3 Return

These functions do not return any arguments.

5.10.4 Calling order

These functions should be called in the initialization section only.

5.10.5 Performance

Table 20. METERLIBFFT_InitAuxBuff functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles	Code size [B]	Clock cycles
METERLIBFFT1PH_InitAuxBuff	12	32		30
METERLIBFFT2PH_InitAuxBuffPh1	12	32	12	30
METERLIBFFT2PH_InitAuxBuffPh2	22		22	
METERLIBFFT3PH_InitAuxBuffPh1	12	32	12	30
METERLIBFFT3PH_InitAuxBuffPh2	22		22	
METERLIBFFT3PH_InitAuxBuffPh3	22		22	

5.11 METERLIBFFT_InitMainBuff

These functions are used to initialize pointers to three types of buffers: input time-domain buffers, output frequency-domain buffers, and phase-shift correction buffer. The position of first buffers matches the zero harmonic, and so on. All these initializations are performed separately for each particular phase.

Both the input voltage and current time-domain buffers, where the ADC values are saved after the sampling, are united with the output frequency-domain buffers, where the real FFT parts of the result will be saved after the computation. Therefore, the input time-domain data are rewritten by the real parts of the FFT result, while the imaginary parts of the FFT result are saved to the separate buffers. The length of time-domain buffers depends on the maximum input samples number, while the buffer length for frequency domain buffers is always power-of-two (set by *samples* parameter in the function described in Section 5.12, “METERLIBFFT_InitParam”). These functions also initialize the pointer to the U–I phase-shift correction buffer for software phase-shift correction in the frequency domain. The parasitic current sensor phase shifts are saved separately for each harmonic. These phase shifts can be compensated for by the function described in Section 5.3, “METERLIBFFT_CalcMain.” If the software phase shift correction is not required, this pointer must be set to NULL.

5.11.1 Syntax

```
#include “meterlibfft.h”

void METERLIBFFT1PH_InitMainBuff (tMETERLIBFFT1PH_DATA *p, Frac24 *u_re, Frac24 *i_re, Frac24
*u_im, Frac24 *i_im, long *shift);

void METERLIBFFT2PH_InitMainBuffPh1 (tMETERLIBFFT2PH_DATA *p, Frac24 *u_re, Frac24 *i_re,
Frac24 *u_im, Frac24 *i_im, long *shift);

void METERLIBFFT2PH_InitMainBuffPh2 (tMETERLIBFFT2PH_DATA *p, Frac24 *u_re, Frac24 *i_re,
Frac24 *u_im, Frac24 *i_im, long *shift);

void METERLIBFFT3PH_InitMainBuffPh1 (tMETERLIBFFT3PH_DATA *p, Frac24 *u_re, Frac24 *i_re,
Frac24 *u_im, Frac24 *i_im, long *shift);

void METERLIBFFT3PH_InitMainBuffPh2 (tMETERLIBFFT3PH_DATA *p, Frac24 *u_re, Frac24 *i_re,
Frac24 *u_im, Frac24 *i_im, long *shift);
```

```
void METERLIBFFT3PH_InitMainBuffPh3 (tMETERLIBFFT3PH_DATA *p, Frac24 *u_re, Frac24 *i_re,
Frac24 *u_im, Frac24 *i_im, long *shift);
```

5.11.2 Arguments

Table 21. METERLIBFFT_InitMainBuff functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
Frac24	u_re	in / out	Pointer to the input time-domain voltage buffer united with the output frequency-domain voltage buffer (real part) in Q0.23 data format
Frac24	i_re	in / out	Pointer to the input time-domain current buffer united with the output frequency-domain current buffer (real part) in Q0.23 data format
Frac24	u_im	out	Pointer to the frequency-domain voltage buffer (imaginary part) in Q0.23 data format
Frac24	i_im	out	Pointer to the frequency-domain current buffer (imaginary part) in Q0.23 data format
long	shift	in	Pointer to the U–I phase-shift correction buffer in 0.001°, for example 4500 = 4.500°. Set to NULL, if correction is not required.

5.11.3 Return

These functions do not return any arguments.

5.11.4 Calling order

As most of these buffers are used for main FFT computing (performed by function described in Section 5.3, “METERLIBFFT_CalcMain”), these initializations are mandatory and must be performed in the initialization section.

5.11.5 Performance

Table 22. METERLIBFFT_InitMainBuff functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles	Code size [B]	Clock cycles
METERLIBFFT1PH_InitMainBuff	26	57	26	55
METERLIBFFT2PH_InitMainBuffPh1	26	57	26	55
METERLIBFFT2PH_InitMainBuffPh2	28		28	
METERLIBFFT3PH_InitMainBuffPh1	26	57	26	55
METERLIBFFT3PH_InitMainBuffPh2	28		28	
METERLIBFFT3PH_InitMainBuffPh3	30		30	

5.12 METERLIBFFT_InitParam

These functions are used for initialization of parameters. All these initializations are valid for all phases together. Proper initialization is very important for receiving correct outputs from other functions.

5.12.1 Syntax

```
#include “meterlibfft.h”

long METERLIBFFT1PH_InitParam (tMETERLIBFFT1PH_DATA *p, unsigned long samples, unsigned long
sensor, unsigned long kwh_cnt, unsigned long kvarh_cnt, unsigned long en_res);

long METERLIBFFT2PH_InitParam (tMETERLIBFFT2PH_DATA *p, unsigned long samples, unsigned long
sensor, unsigned long kwh_cnt, unsigned long kvarh_cnt, unsigned long en_res);

long METERLIBFFT3PH_InitParam (tMETERLIBFFT3PH_DATA *p, unsigned long samples, unsigned long
sensor, unsigned long kwh_cnt, unsigned long kvarh_cnt, unsigned long en_res);
```

5.12.2 Arguments

Table 23. METERLIBFFT_InitParam functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
unsigned long	samples	in	Number of the required FFT samples – see Table 24
unsigned long	sensor	in	Current sensor type – see Table 25
unsigned long	kwh_cnt	in	Active energy impulse number – see Table 26
unsigned long	kvarh_cnt	in	Reactive energy impulse number – see Table 26
unsigned long	en_res	in	Active / reactive energy resolution – see Table 27

Table 24. Number of input samples defines

Define name	Description
SAMPLES8	8 input samples, 4 output harmonics
SAMPLES16	16 input samples, 8 output harmonics
SAMPLES32	32 input samples, 16 output harmonics
SAMPLES64	64 input samples, 32 output harmonics
SAMPLES128	128 input samples, 64 output harmonics
SAMPLES256	256 input samples, 128 output harmonics
SAMPLES512	512 input samples, 256 output harmonics

Table 25. Current sensor type defines

Define name	Description
SENS_DERIV	Derivative type of the current sensor (Rogowski Coil)
SENS_PROP	Proportional type of the current sensor (shunt, Current Transformer)

Table 26. Impulse number defines

Define name	Description	Define name	Description
IMP200	200 imp / kWh or 200 imp / kVARh	IMP5000	5000 imp / kWh or 5000 imp / kVARh
IMP250	250 imp / kWh or 250 imp / kVARh	IMP10000	10000 imp / kWh or 10000 imp / kVARh
IMP500	500 imp / kWh or 500 imp / kVARh	IMP12500	12500 imp / kWh or 12500 imp / kVARh
IMP1000	1000 imp / kWh or 1000 imp / kVARh	IMP20000	20000 imp / kWh or 20000 imp / kVARh
IMP1250	1250 imp / kWh or 1250 imp / kVARh	IMP25000	25000 imp / kWh or 25000 imp / kVARh
IMP2000	2000 imp / kWh or 2000 imp / kVARh	IMP50000	50000 imp / kWh or 50000 imp / kVARh
IMP2500	2500 imp / kWh or 2500 imp / kVARh	IMP100000	100000 imp / kWh or 100000 imp / kVARh

Table 27. Energy resolution defines

Define name	Description
EN_RES1	Energy resolution is 1 Wh / VARh, max. range is 4294.967 MWh / MVARh, supports Impulse numbers <= 1000
EN_RES10	Energy resolution is 0.1 Wh / VARh, max. range is 429.4967 MWh / MVARh, supports Impulse numbers <= 10000
EN_RES100	Energy resolution is 0.01 Wh / VARh, max. range is 42.94967 MWh / MVARh, supports all Impulse numbers

5.12.3 Return

These functions return one of the following error codes:

- FFT_ERROR (positive) – some of the input parameters are not right, the function output is not valid
- FFT_OK (zero) – all input parameters are right, the function output is valid

5.12.4 Calling order

These mandatory functions must be called primarily in the initialization section, or after changing some of its parameters during the program execution.

5.12.5 Performance

Table 28. METERLIBFFT_InitParam functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles	Code size [B]	Clock cycles
METERLIBFFT1PH_InitParam	1380	787	1380	782
METERLIBFFT2PH_InitParam	1414	837	1414	830
METERLIBFFT3PH_InitParam	1440	849	1440	847

5.13 METERLIBFFT_Interpolation

These functions are used to interpolate the original input curve, given by unsigned integer samples, to the curve given by power-of-two samples, required by the FFT function [7]. These functions support both oversampling and undersampling.

5.13.1 Syntax

```
#include “meterlibfft.h”

long METERLIBFFT1PH_Interpolation (tMETERLIBFFT1PH_DATA *p, unsigned long u_ord, unsigned long
i_ord, unsigned long samples_inp);

long METERLIBFFT2PH_Interpolation (tMETERLIBFFT2PH_DATA *p, unsigned long u_ord, unsigned long
i_ord, unsigned long samples_inp);

long METERLIBFFT3PH_Interpolation (tMETERLIBFFT3PH_DATA *p, unsigned long u_ord, unsigned long
i_ord, unsigned long samples_inp);
```

5.13.2 Arguments

Table 29. METERLIBFFT_Interpolation functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure
unsigned long	u_ord	in	Voltage interpolation order – see Table 30
unsigned long	i_ord	in	Current interpolation order – see Table 30
unsigned long	samples_inp	in	Input samples number, can be higher or lower than required by power-of-two FFT samples

Table 30. Interpolation order defines

Define name	Description
ORD1	The 1st order (linear) interpolation
ORD2	The 2nd order (quadratic) interpolation
ORD3	The 3rd order (cubic) interpolation

5.13.3 Return

These functions return one of the following error codes valid only for undersampling use, for input samples lower than FFT samples (set by *samples* parameter in Section 5.12, "METERLIBFFT_InitParam"):

- FFT_ERROR (positive) – FFT samples are higher than input samples, and FFT samples are higher than 128.
- FFT_OK (zero) – undersampling ratio is correct.

5.13.4 Calling order

These functions should be used only if interpolation processing is required [7]. In this case, these functions should be called periodically in a defined interval, which depends on the line frequency and/or the multiplied ADC sampling rate. These functions should be called closely before the main (FFT) calculation processing. The one-shot mandatory parameter initialization must be performed before calling these functions. Apart from other things, this parameter initialization function sets the number of required FFT points, and also initializes all pointers to the input buffers used by the interpolation functions.

NOTE

The original values in the input buffers (ADC values) will be rewritten by the new (interpolated) values after these functions are performed.

5.13.5 Performance

Table 31. METERLIBFFT_Interpolation functions performance for the CM0+ core

Function name	Code size [B]			Stack size [B][1]	Clock cycles[2]		
	1st order	2nd order	3rd order		1st order	2nd order	3rd order
METERLIBFFT1PH_Interpolation	506	842	1654	512	12521	35260	76996
METERLIBFFT2PH_Interpolation	586	922	1734		24946	70519	153991
METERLIBFFT3PH_Interpolation	682	1018	1830		37418	106019	231227

1 Due to the undersampling use case (input samples < FFT samples)
2 Number of input samples = 120, number of required FFT points = 64, the same interpolation order for both channels

Table 32. METERLIBFFT_Interpolation functions performance for the CM0+ core with MMAU

Function name	Code size [B]			Stack size [B][1]	Clock cycles[2]		
	1st order	2nd order	3rd order		1st order	2nd order	3rd order
METERLIBFFT1PH_Interpolation	940	1116	1560	512	7388	15543	28304
METERLIBFFT2PH_Interpolation	1020	1196	1640		14728	30942	56607
METERLIBFFT3PH_Interpolation	1116	1292	1736		22067	46533	84911

1 Due to the undersampling use case (input samples < FFT samples)
2 Number of input samples = 120, number of required FFT points = 64, the same interpolation order for both channels

5.14 METERLIBFFT_SetCalibCoeff

These functions are used to set up all voltage and current calibration coefficients (gain factors). The absolute value of these coefficients depends on hardware topology (sensor, AFE). Therefore, the final calculation precision of some library functions strictly depends on proper settings of these coefficients. These coefficients should be interpreted as maximum peak voltage or current value valid for maximum AFE range (24-bit AFE range). If the current offset correction is not used, this offset pointer should be assigned to NULL (non-true I_{RMS} computing in this case).

5.14.1 Syntax

```
#include “meterlibfft.h”

long METERLIBFFT1PH_SetCalibCoeff (tMETERLIBFFT1PH_DATA *p, double u_max, double i_max, Frac24
*i_offs, double p_offs, double q_offs);

long METERLIBFFT2PH_SetCalibCoeffPh1 (tMETERLIBFFT2PH_DATA *p, double u_max, double i_max,
Frac24 *i_offs, double p_offs, double q_offs);

long METERLIBFFT2PH_SetCalibCoeffPh2 (tMETERLIBFFT2PH_DATA *p, double u_max, double i_max,
Frac24 *i_offs, double p_offs, double q_offs);

long METERLIBFFT3PH_SetCalibCoeffPh1 (tMETERLIBFFT3PH_DATA *p, double u_max, double i_max,
Frac24 *i_offs, double p_offs, double q_offs);

long METERLIBFFT3PH_SetCalibCoeffPh2 (tMETERLIBFFT3PH_DATA *p, double u_max, double i_max,
Frac24 *i_offs, double p_offs, double q_offs);

long METERLIBFFT3PH_SetCalibCoeffPh3 (tMETERLIBFFT3PH_DATA *p, double u_max, double i_max,
Frac24 *i_offs, double p_offs, double q_offs);
```

5.14.2 Arguments

Table 33. METERLIBFFT_SetCalibCoeff functions arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA	p	in	Pointer to the one-phase metering library data structure.
tMETERLIBFFT2PH_DATA	p	in	Pointer to the two-phase metering library data structure.
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure.
double	u_max	in	Peak line voltage [V] valid for AFE full-scale range.
double	i_max	in	Peak line current [A] valid for AFE full-scale range.
Frac24	i_offs	in	Pointer to the current offset in Q0.23 data format. Use NULL, if offset doesn’t have to be included in the I_{RMS} computing.
double	p_offs	in	Active power (P) offset correction value in Watts. Use zero, if P-offset doesn’t have to be included in the active power computing.
double	q_offs	in	Reactive power (Q) offset correction value in Volt-Amperes-reactive. Use zero, if Q-offset doesn’t have to be included in the reactive power computing.

5.14.3 Return

These functions return one of the following codes:

- FFT_ERROR (positive) – some of the input parameters are too large, overflow may occur; the right coefficients values should be kept as (u_max*i_max) < (2^{31}/10000)
- FFT_OK (zero) – all input parameters are correct

5.14.4 Calling order

These mandatory functions must be called in the initialization section primarily. They may be called also during or after the hardware calibration processing.

5.14.5 Performance

Table 34. METERLIBFFT_SetCalibCoeff functions performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size [B]	Clock cycles	Code size [B]	Clock cycles
METERLIBFFT1PH_SetCalibCoeff	146	2130	146	2130
METERLIBFFT2PH_SetCalibCoeffPh1	146	2130	146	2130
METERLIBFFT2PH_SetCalibCoeffPh2				
METERLIBFFT3PH_SetCalibCoeffPh1	146	2130	146	2130
METERLIBFFT3PH_SetCalibCoeffPh2				
METERLIBFFT3PH_SetCalibCoeffPh3				

5.15 METERLIBFFT_SetEnergy

This macro sets all energy counters and clears all reminders. There is only one macro performing the same actions for all types of metering topologies (one-phase, two-phase, three-phase). The energy resolution depends on *en_res* parameter, set by the function described in Section 5.12, “METERLIBFFT_InitParam.”

5.15.1 Syntax

```
#include “meterlibfft.h”

METERLIBFFT_SetEnergy( p,  whi,  whe,  varhi,  varhe);
```

5.15.2 Arguments

Table 35. METERLIBFFT_SetEnergy macro arguments

Type	Name	Direction	Description
tMETERLIBFFT1PH_DATA, tMETERLIBFFT2PH_DATA, tMETERLIBFFT3PH_DATA	p	in	Pointer to one of the metering library data structures
unsigned long	whi	in	Import active energy value (see Table 27 for the value resolution)
unsigned long	whe	in	Export active energy value (see Table 27 for the value resolution)
unsigned long	varhi	in	Import reactive energy value (see Table 27 for the value resolution)
unsigned long	varhe	in	Export reactive energy value (see Table 27 for the value resolution)

5.15.3 Return

This macro does not return any arguments.

5.15.4 Calling order

This macro should be used mainly in the initialization section.

5.16 METERLIBFFT_GetRotation

This function computes the phase angle between each individual phases (phase 2 to phase 1, phase 3 to phase 1, phase 3 to phase 2) and also returns the sense of rotation (forward, reverse). This is a specific function for the three-phase mains only.

5.16.1 Syntax

```
#include “meterlibfft.h”

long METERLIBFFT3PH_GetRotation(tMETERLIBFFT3PH_DATA *p, double *u12_ph, double *u13_ph, double
*u23_ph);
```

5.16.2 Arguments

Table 36. METERLIBFFT3PH_GetRotation function arguments

Type	Name	Direction	Description
tMETERLIBFFT3PH_DATA	p	in	Pointer to the three-phase metering library data structure.
double	u12_ph	out	Pointer to the phase 2 to the phase 1 angle in degree.
double	u13_ph	out	Pointer to the phase 3 to the phase 1 angle in degree.
double	u23_ph	out	Pointer to the phase 3 to the phase 2 angle in degree.

5.16.3 Return

The function returns one of the following output states:

- ROT_FORWARD (positive) – clockwise (or forward) sense of rotation, that is 1-2-3, 2-3-1, or 3-1-2.
- ROT_REVERSE (negative) – counter-clockwise (or reverse) sense of rotation, that is 2-1-3, 1-3-2, or 3-2-1.
- ROT_UNKNOWN (zero) – sense of rotation cannot be recognized due to phase losing.

5.16.4 Calling order

This function may be called after the main (FFT) calculation processing. The one-shot mandatory parameter initialization must be done before calling this function.

5.16.5 Performance

Table 37. METERLIBFFT3PH_GetRotation function performance

Function name	CM0+ core		CM0+ core with MMAU	
	Code size[1] [B]	Clock cycles	Code size[1] [B]	Clock cycles
METERLIBFFT3PH_GetRotation	532	3982	524	2926

[1] Code size of the arctangent lookup table (8 KB) isn't included.

6 Summary

This application note describes how to compute basic metering values in a metering application using the FFT. The presented algorithm is simple and highly accurate, it can be easily integrated into electronic meters, and requires only instantaneous phase voltage and current samples to be provided to their inputs. It has been designed for devices featuring sigma-delta or SAR converters, which have a fixed or adjustable measurement sample rate. The performance of the FFT-based metering library has been tested in several power meter reference designs [5] [6] with high accuracy for high-current dynamic ranges.

The computing technique based on FFT has the folllowing advantages and disadvantages in metering applications:

Advantages of realization:

- The same precision for both the active and reactive energies (due to using one computing formula).
- Four-quadrant active and reactive energy measurement.
- Frequency analysis of the mains, ability to compute the total harmonic distortion (THD).
- Offset removal, because the zero-harmonic can be missing for power computing.

Disadvantages of realization:

- Additional interpolation processing is needed when using fixed sample rate (sigma-delta ADCs).
- Higher computational power of the MCU (a 32-bit MAC unit is required).

7 References

1. J.W.Cooley and J.W.Tukey, An algorithm for the machine calculation of the complex Fourier series, Math. Comp., Vol. 19 (1965), pp. 297-301
2. Wikipedia articles “Cooley-Tukey FFT algorithm”, “Complex number”, “AC Power”, “Power Factor”, “Total Harmonic Distortion”, “Q (number format)”, available at en.wikipedia.org
3. Fast Fourier Transform (FFT) article, available at www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html
4. *MQX-enabled MK30X256 Single-Phase Electricity Meter Reference Design* (document DRM122)
5. *Kinetis-M Two-Phase Power Meter Reference Design* (document DRM149)
6. *MKM34Z256 One-Phase Power Meter Reference Design* (document DRM163)
7. *Using the FFT on the Sigma-Delta ADCs* (document AN4847)
8. Fractional and Integer Arithmetic – DSP56000 Family of General-Purpose Digital Signal Processors, (Motorola 1993, USA)

8 Revision history

Table 38. Revision history

Revision number	Date	Substantial changes
0	11/2011	Initial release
1	10/2013	Upgraded Section 5, “Metering library”
2	11/2014	Upgraded Section 4.2, “Root Mean Square computing” Upgraded Section 4.3, “Complex power computing” Added Section 4.4, “Energy computing” Added Section 4.5, “Power factor computing” Added Section 4.6, “Total Harmonic Distortion computing” Upgraded Section 5, “Metering library”
3	02/2015	Upgraded Section 4.2, “Root Mean Square computing” Upgraded Section 5.14, “METERLIBFFT_SetCalibCoeff” Upgraded Figure 7 Upgraded Clock cycles values in Table 5 - Table 32
4	07/2015	Upgraded Clock cycles values in Table 5 - Table 37 Upgraded Figure 7 Upgraded Section 5, “Metering library” Added Section 5.1, “Core architecture and compiler support Upgraded Section 5.12, “METERLIBFFT_InitParam” Upgraded Section 5.13, “METERLIBFFT_Interpolation Upgraded Section 5.14, “METERLIBFFT_SetCalibCoeff” Added Section 5.16, “METERLIBFFT_GetRotation” Upgraded Section 6, “Summary” Upgraded Section 7, “References”

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4255
Rev. 4
07/2015