**Freescale Semiconductor, Inc.**
Application Note

Document Number: AN4847
Rev. 2, 07/2015

# Using the FFT on Sigma-Delta ADCs

*by: Ludek Slosarcik*

## 1 Introduction

This application note deals with two topics in digital signal processing and metering, namely the Fast Fourier Transform (FFT) and sigma delta Analog-to-Digital Converters (SD ADCs).

First, the FFT is a mathematical technique for transforming a time-domain digital signal into a frequency-domain representation of the relative amplitudes of the different frequency regions in the signal. The FFT is extremely important in the area of frequency (spectrum) analysis. Second, SD ADCs are high resolution, high integration, and low-cost ADCs for applications such as metering, process control, and monitoring.

Both topics are outlined in depth in this application note, and some interested parties may find more details on relevant web pages or reference sources. The main purpose of this application note is to describe some methods for properly using the FFT in cooperation with the SD ADCs, especially in power metering applications.

**Contents**

# 2 Basic problem description

Using the FFT in power (energy) metering applications has some specific requirements, which are described in *FFT-Based Algorithm for Metering Applications* (document AN4255). The basic requirement is to have a power-of-two number of input samples during one input signal period. To fulfill this requirement, know the input signal frequency to be able to compute the correct period of time between the neighboring samples. This is due to having the power-of-two number of samples during the signal period. This assumes that the time between each ADC sample may be slightly adjusted by a Programmable Delay Block (PDB) or by another special timer which is able to provide a controllable interval tick to the hardware trigger of the ADC used. Strictly speaking, we need to synchronize the input signal frequency with the ADC sampling rate. For example, this technique may be simply used on the ADCs based on the Successive Approximation Register (SAR), which may be hardware-triggered by the PDB. On the other hand, there are also other types of ADCs and applications where this described technique cannot be used. This problem appears in the sigma-delta ADCs, whose sampling interval cannot be modified slightly. There are some use cases for bypassing this issue. Therefore, the current application note is a logical continuation of the original AN4255, which doesn't resolve this issue.

# 3 Use case 1 – synchronous processing

The simplest way to get the power-of-two samples from the SD ADC for a subsequent FFT computation is to synchronize both processes; the signal frequency and the ADC sampling rate. This process is described in AN4255. It requires knowledge of the frequency of the measured signal and being able to run the SD ADC in the single conversion mode too. The detection of the signal frequency (period) may be performed by the Zero-Crossing Detection (ZCD) technique, described in AN4255, Section 3.2.1. Although the SD ADCs are not very suitable for the single-conversion configuration due to a longer start-up time, we may use this AD conversion mode in some cases. The main limiting factor in using the SD ADCs in this mode is the start-up time, which may be several times higher than the time for normal AD conversion. This is due to a latency of the decimation filter. For example, the start-up time for the SD ADC used in the Freescale MKM34Z128 MCU (ARM® Cortex®-M0+ core)[2.] is typically three times higher in comparison to its normal conversion time. This feature limits the use of this mode to higher output sample rates (OSR) in cooperation with the subsequent FFT computation.

# 4 Use case 2 – asynchronous processing

The other way to resolve the issue with using the FFT on the SD ADCs is to use an oversampling. In practical terms, the SD ADC collects more samples during the signal period than are really needed. The next step of this process is to use some type of re-computing technique, which transforms the original non-power-of-two ADC samples into the power-of-two samples required by the FFT. We may simply term this whole process as an interpolation of the input signal. This process is graphically represented in Figure 1. For simplification, there are only eight FFT points used and 12 measured samples (ratio is 1.5). Theoretically, it would be possible to use an undersampling method with the same subsequent process (interpolation to the power-of-two samples), but practically it is better to have a higher number of input samples than the FFT really needs. This is due to having a minimum computation error of the interpolated signal. Therefore, the ratio between the number of input samples and the power-of-two samples should be higher than 1. The upper bound of this ratio is limited by the power of the MCU used. The general rule is: a higher ratio, a lesser interpolation error, but a higher MCU (ADC) workload.

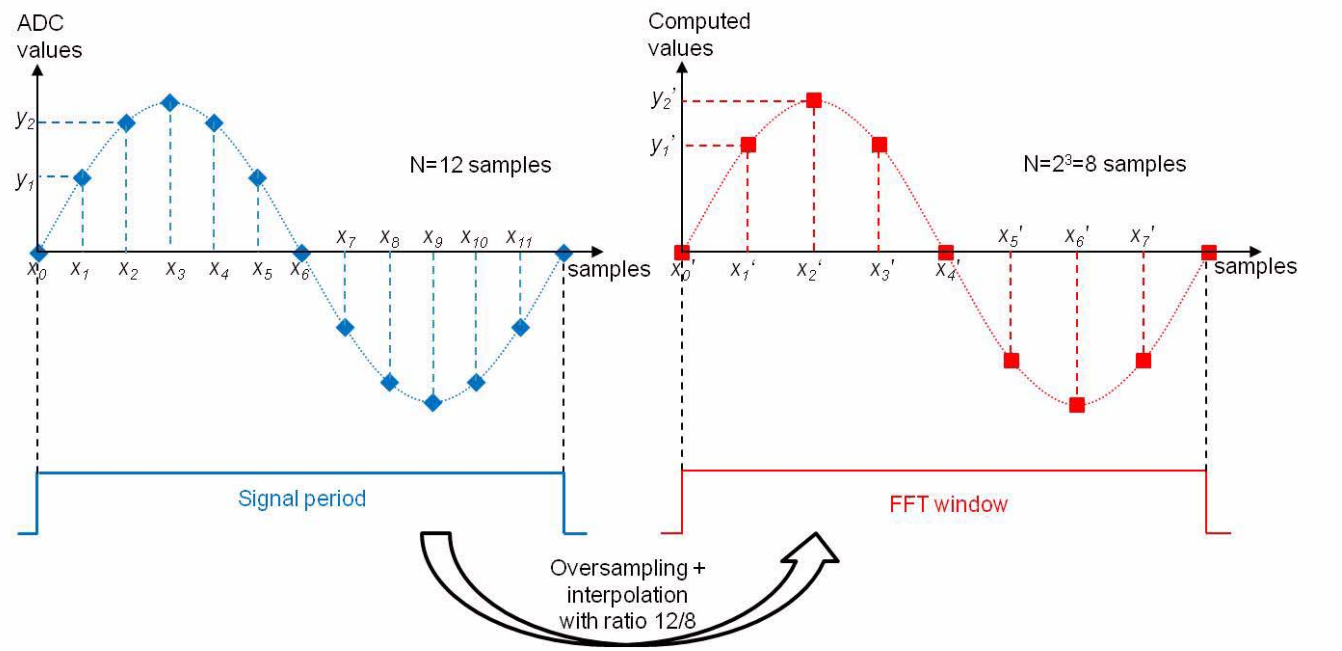



**Figure 1. FFT oversampling use case**

Generally, this computing technique supposes that both processes, the signal frequency, and the ADC sampling rate, are mostly purely asynchronous. Thanks to this computing technique we may let the SD ADC run in a continuous conversion mode, which is much more natural for this type of ADCs than the single conversion mode (see Section 3, “Use case 1 – synchronous processing,” on page 2).

As mentioned above, the key point for this use case is the interpolation. In the mathematical field of numerical analysis, interpolation is a method of constructing new data points within the range of a discrete set of known data points. With respect to Figure 1, the new data points are on its right-hand side (red data set), whereas the known data points are on its left-hand side (blue data set). While the known data points are measured by the ADC, the new data points are computed by the MCU. Therefore, the interpolation provides a means of estimating the function at intermediate known (for example measured) points.

There are plenty of known interpolation methods, some of which are described below. These methods were tested practically on the two-phase metering reference design based on the MKM34Z128 MCU.

## 4.1 Linear interpolation

Linear interpolation is the simplest method of getting values at positions in between the data points. The points are simply joined by straight line segments. Each segment (bounded by two neighboring data points) can be interpolated independently. With respect to the example in Figure 1, we will have the set of data points $(x_0,y_0)$, $(x_1,y_1)$,...,$(x_{11},y_{11})$. Linear interpolation on this set of data points is defined as the concatenation of linear interpolants between each pair of data points. A linear interpolant is the straight line between neighboring points. The detail of the interpolation of one segment is pictured in Figure 2. The color representation is the same as in Figure 1, that is the blue points are measured values, whereas the red points are computed values.

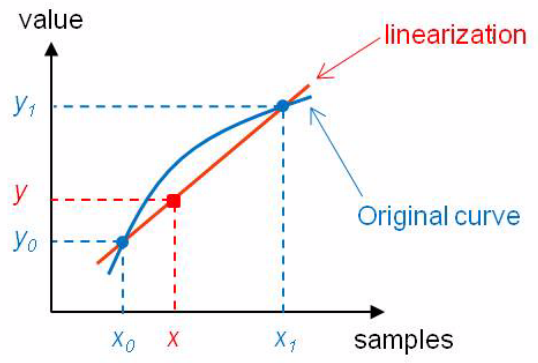



**Figure 2. Example of linear interpolation**

For a value $x$ in the interval $(x_0,x_1)$, the value $y$ along the straight line is given by the following equation:

$$\frac{y-y_0}{x-x_0} = \frac{y_1-y_0}{x_1-x_0} \qquad \textit{Eqn. 1}$$

Solving this equation for $y$, which is the unknown value at $x$, gives:

$$y = (y_1-y_0)\frac{x-x_0}{x_1-x_0} + y_0 = (ax+b) \qquad \textit{Eqn. 2}$$

Where $a$ and $b$ are linear coefficients; $a$ is the gain, while $b$ is the offset.

Linear interpolation is a special case of a polynomial interpolation with the degree of one. It is quick and easy, but it is not very precise, especially when there are higher harmonics in the input signal and a low ratio between the input samples and the computed points. We may resolve this issue by extending this ratio, if possible.

## 4.2 Polynomial interpolation

Polynomial interpolation is the interpolation of a given data set by a polynomial. Let us assume that we have $n$+1 discrete data points $(x_0,y_0)$, $(x_1,y_1)$,....,$(x_n,y_n)$ with different $x$-coordinates. The main objective is to find a polynomial function of degree $n$ that passes through these $n$+1 points. This polynomial is called an interpolating polynomial. There are plenty of known methods for solving this assignment. One of the simplest solution methods is using the Lagrange interpolating polynomial. This is a well-known technique in numerical analysis. The general equation for the Lagrange interpolating polynomial is:

$$y = L(x) = \sum_{i=0}^{n} y_i \cdot L_i(x) \qquad \textit{Eqn. 3}$$

Where $n$ is the degree of the polynomial, and $L_i(x)$ are Lagrange basic polynomials, expressed as:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)} \qquad \textit{Eqn. 4}$$

The basic polynomial $L_i(x)$ has the property that:

$$L_i(x_j) = \begin{cases} 0 & j \neq i \\ 1 & j = i \end{cases} \qquad \textit{Eqn. 5}$$

Polynomial interpolation is a generalization of linear interpolation, but it is more precise than the linear interpolation. Its precision depends on the degree of the polynomial ($n$-size in Equation 3). It gives a better accuracy for functions with larger variations. On the other hand, calculation of the interpolating polynomial is computationally expensive compared to linear interpolation, mainly for real-time applications (for example power metering). Due to this, the following text focuses only on the polynomial interpolation with a maximum degree of three. In other words, we will not compute one interpolating polynomial of degree $n$, but we will divide this task by finding several elementary interpolating polynomials of a lower degree ($n$=2 or $n$=3 at maximum). In practical terms, we will compute the $y$ value, which is the unknown value at $x$, using three or four neighboring data points. The same process will be repeated for each pair of $(x,y)$ values using several new neighbouring data points $(x_i,y_i)$. This process is described in the following subheads.

### 4.2.1 Lagrange quadratic interpolation

Solving the basic Equation 3 of the Lagrange form of the interpolating polynomial for $n$=2 for general value $y$, which is the unknown value at $x$, gives:

$$y = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2 = \qquad \textit{Eqn. 6}$$

$$= y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x)$$

Where $L_i(x)$ are basic Lagrange quadratic polynomials.

There is supposed to be three pairs (generally, $n$+1) of input data points $(x_0,y_0)$,$(x_1,y_1)$, and $(x_2,y_2)$ for solving this equation. The new $x$-value should be between one of these values, $x_0$ - $x_1$ or $x_1$ - $x_2$. This new $x$-value is in fact a recalculated value (see also Figure 1 for new $x_i$' values). It is the multiple of the ratio between the number of the ADC values to the number of the power-of-two FFT points.

The example of the interpolation for $n$=2 is pictured in Figure 3. The three input points (measured values) are shown in blue. The interpolating function passing through them is the red parabola. Generally, the Lagrange basic polynomials $L_i(x)$ of degree two are quadratic functions. Their curve is a parabola.

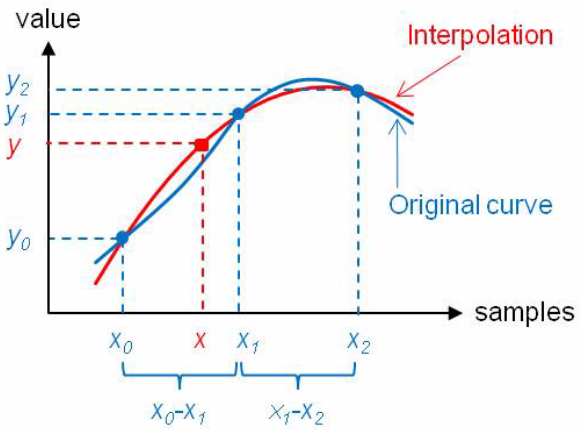



**Figure 3. Example of quadratic interpolation**

In general, the three *x*-values do not need to be spaced evenly ($x_0$ - $x_1$ may differ from $x_1$ - $x_2$). However, a uniform distribution of points is a typical use case for ADC periodic sampling intervals, that is, for metering applications. Thanks to this simplification, we may rewrite Equation 6 to the following form:

$$y = \frac{(x-x_0-1)(x-x_0-2)}{2}y_0 - (x-x_0)(x-x_0-2)y_1 + \frac{(x-x_0)(x-x_0-1)}{2}y_2 \qquad \textit{Eqn. 7}$$

## 4.2.2 Lagrange cubic interpolation

Solving the basic Equation 3 of the Lagrange form of the interpolating polynomial for $n$=3 for general value $y$, which is the unknown value at $x$, gives:

$$y = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}y_0 + \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}y_2 +$$

*Eqn. 8*

$$+ \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}y_3 = y_0L_0(x) + y_1L_1(x) + y_2L_2(x) + y_3L_3(x)$$

Where $L_i(x)$ are basic Lagrange cubic polynomials.

Given four discrete input data points $(x_0,y_0)$,$(x_1,y_1)$,$(x_2,y_2)$, and $(x_3,y_3)$, we can find each individual cubic polynomial $L_i(x)$. The new *x*-value should be between one of these three intervals, $x_0$ - $x_1$, $x_1$ - $x_2$, or $x_2$ - $x_3$. This new *x*-value is in fact a recalculated value (see also Figure 1 for new $x_i$' values). It is the multiple of the ratio between the number of the ADC values to the number of the power-of-two FFT points.

The example of the interpolation for $n$=3 is pictured in Figure 4. The four input points (measured values) are shown in blue. The interpolating function passing through them is the red cubic curve. Generally, the Lagrange basic polynomials $L_i(x)$ of degree three are cubic functions.

Similarly to the previous example, we assume an uniform distributions of *x*-points in the metering applications. Therefore, the Equation 8 may be rewritten in the following form:

$$y = \frac{(x-x_0-1)(x-x_0-2)(x-x_0-3)}{-6}y_0 + \frac{(x-x_0)(x-x_0-2)(x-x_0-3)}{2}y_1 =$$

$$= \frac{(x-x_0)(x-x_0-1)(x-x_0-3)}{-2}y_2 + \frac{(x-x_0)(x-x_0-1)(x-x_0-2)}{6}y_3 \quad \textit{Eqn. 9}$$

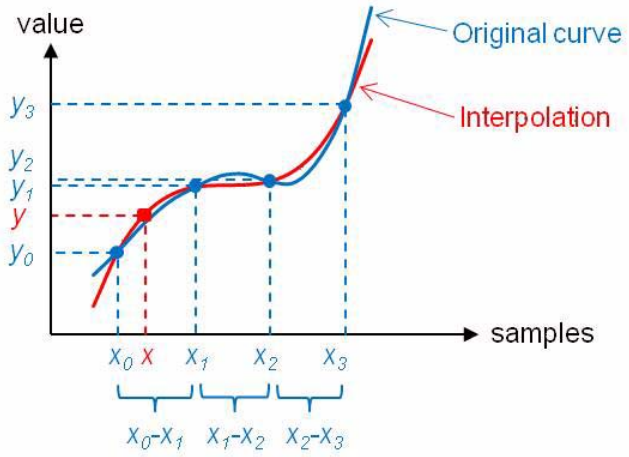



**Figure 4. Example of cubic interpolation**

# 5 Practical implementation

This chapter describes the implementation of the FFT resampling and interpolation techniques used in the C-code, and the simulation of these techniques in Excel®. It contains a description of the application programming interface (API) of three C-functions (for most frequent power meter topologies) for interpolation of the measured AD samples before the final FFT computation. The API for the subsequent FFT computation is described in a separate application note AN4255[1.]. The interpolation functions are part of the FFT-based metering library called METERLIBFFT, which is used for the power computation in some of Freescale electricity meter reference designs [4.] and [5.].

## 5.1 METERLIBFFT_Interpolation

These three functions are used for interpolation of the original input curve given by unsigned integer samples to the curve given by power-of-two samples, required by the FFT function. The source C-code for each type of algorithm is included in the *fft.c* file of the METERLIBFFT library. These functions support both oversampling and undersampling. The example of using the interpolation function in the main C-code is in the Section Appendix A, "Example of using the interpolation function in the single-phase user code." There are also other functions used in this example. These functions, whose API is described in the AN4255[1.], are also part of the metering library, and are used for power computation based on the FFT.

### 5.1.1 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);
```

### 5.1.2 Arguments

**Table 1. 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 2 |
| unsigned long | i_ord | in | Current interpolation order – see Table 2 |
| unsigned long | samples_inp | in | Input samples number can be higher or lower than the required power-of-two FFT samples |

**Table 2. Interpolation order defines**

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

### 5.1.3 Return

These functions return one of the following error codes valid only for undersampling use, for input samples lower than FFT samples:

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

### 5.1.4 Calling order

These functions should be used only if the interpolation processing is required. In that case, these functions should be called periodically in a defined interval, which depends on the line frequency. 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 calling these functions.

### 5.1.5 Performance

**Table 3. 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 4. 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

## 5.2 Simulation results

The results of the simulation for four examples are in the Appendices B, C, D, and E. These are typical cases of metering use. All examples were simulated in Excel. Each example has its own parametric table that describes the conditions of simulation. This is followed by two signal curves for a practical demonstration of how both the input and interpolated output signals look like. Finally, there are three error curves, each for a particular interpolation method. These error curves represent a percentage computational error between the ideal (non-interpolated) and the interpolated signal for each point during one period.

# 6 Summary

This application note describes several types of interpolation techniques for proper use of the FFT algorithm in special types of metering applications where the ADC periodic sampling intervals cannot be simply modified, for example, on the sigma-delta ADCs. The described interpolation techniques, which are not the only techniques, were selected with respect to their effective computing in a real-time application. The Lagrange quadratic interpolation method with ratio between two and three seems to be a good compromise with respect to its computing performance and precision. The best interpolation method (with regards to precision) supported by this library is the cubic interpolation, especially when used for the

current signal interpolation, due to its possibly bigger distortion. There are plenty of other known interpolation methods commonly used in math that can be used for this purpose. The limiting factor in their use in metering applications is computing performance and final precision.

# 7 References


1. *"FFT-Based Algorithm for Metering Applications"* (document AN4255)
2. *"Kinetis M reference manual"* (document MKMxxZxxACxx5RM)
3. Wikipedia articles "Interpolation", "Linear interpolation", "Polynomial interpolation", "Lagrange polynomial" available at en.wikipedia.org
4. *"Kinetis-M Two-Phase Power Meter Reference Design"* (document DRM149)
5. *"MKM34Z256 One-Phase Power Meter Reference Design"* (document DRM163)


# 8 Revision history

**Table 5. Revision history**

| Revision number | Date | Substantial changes |
|---|---|---|
| 0 | 12/2013 | Initial release |
| 1 | 11/2014 | Upgraded Section 5, "Practical implementation"<br>Upgraded Section 6, "Summary" |
| 2 | 07/2015 | Upgraded Section 5.1, "METERLIBFFT_Interpolation" |

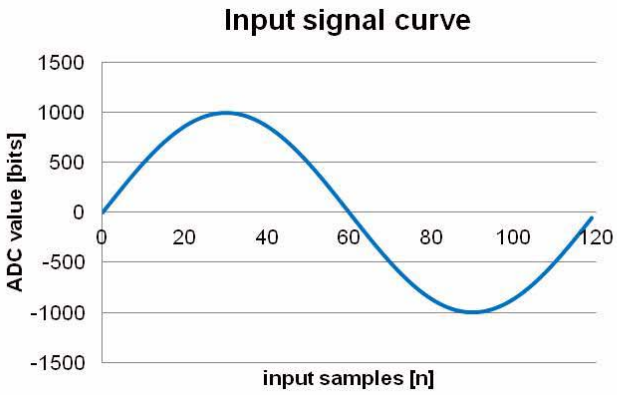
# Appendix A Example of using the interpolation function in the single-phase user code

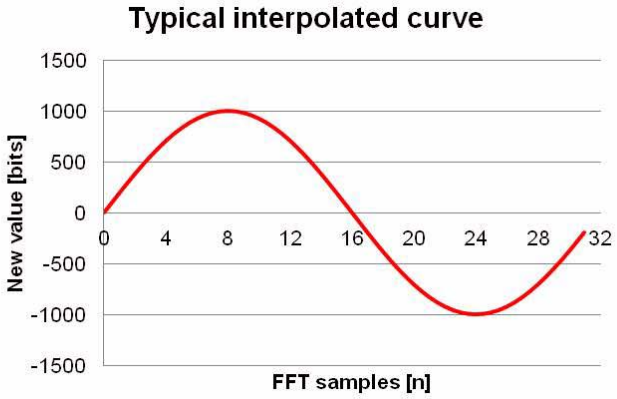
```
/*******************************************************************************
 * (c) Copyright 2015, Freescale Semiconductor Inc.
 * ALL RIGHTS RESERVED.
*******************************************************************************/
#include "fraclib.h"           /* fractional library header */
#include "meterlibfft.h"       /* metering library header */
#include "inputdata.h"         /* library of a typical periodic signals (LUT) */

/*******************************************************************************
 * Buffers definitions                                                         *
*******************************************************************************/
/* multiplexed mandatory buffers (time domain / frequency domain in the Cartesian form) */
Frac24 u_re[120];     /* U-ADC output buffer/FFT real part output buffer */
Frac24 i_re[120];     /* I-ADC output buffer/FFT real part output buffer */

/* dedicated mandatory buffers (frequency domain in the Cartesian form) */
Frac24 u_im[SAMPLES64];     /* U-FFT imaginary part output buffer */
Frac24 i_im[SAMPLES64];     /* I-FFT imaginary part output buffer */

/*******************************************************************************
 * Variables definitions                                                       *
*******************************************************************************/
tMETERLIBFFT1PH_DATA ui;              /* 1-PH main metering structure */
long fcn_out;                         /* metering function output or function error state */
long *pu,*pi;                         /* pointers to LUTs */

/*******************************************************************************
 * Main                                                                        *
*******************************************************************************/
void main (void)
{
    /* Mandatory initialization section - for main FFT calculation */
    fcn_out = METERLIBFFT1PH_InitParam(&ui, SAMPLES64, SENS_PROP, IMP5000, IMP5000, EN_RES10);
    METERLIBFFT1PH_InitMainBuff(&ui, u_re, i_re, u_im, i_im, NULL);
    fcn_out = METERLIBFFT1PH_SetCalibCoeff(&ui, 325.27, 141.422, NULL, 0, 0);

    /* ADC sampling simulation (function fills-up both U and I buffers) */
    pu = sin_120s_6e6_5h_10p; /* set pointer to the beginning of U-LUT */
    pi = sin_120s_4e6_5h_40p; /* set pointer to the beginning of I-LUT */
    for (unsigned long cnt = 0; cnt < 120; cnt++)
    {
        u_re[cnt] = (*pu++);        /* copy U-binary values from LUT to the U-output buffer */
        i_re[cnt] = (*pi++);        /* copy I-binary values from LUT to the I-output buffer */
    }

    /* performs interpolation only for asynchronous processing */
    METERLIBFFT1PH_Interpolation(&ui, ORD2, ORD2, 120);

    /* main calculation (FFT, I-signal conditioning, scaling, averaging) */
    METERLIBFFT1PH_CalcMain(&ui);
    while (1);
}
/*******************************************************************************
 * End of module                                                               *
*******************************************************************************/
```

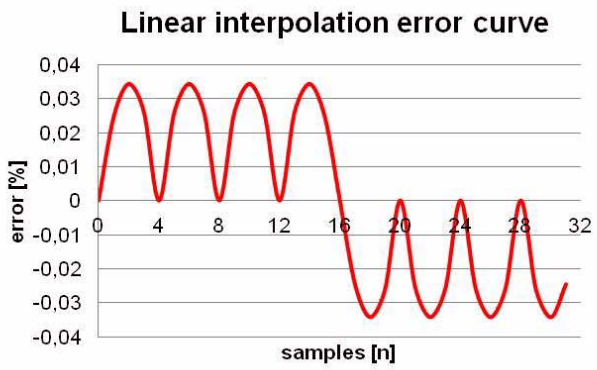
# Appendix B Results of simulation – Example 1

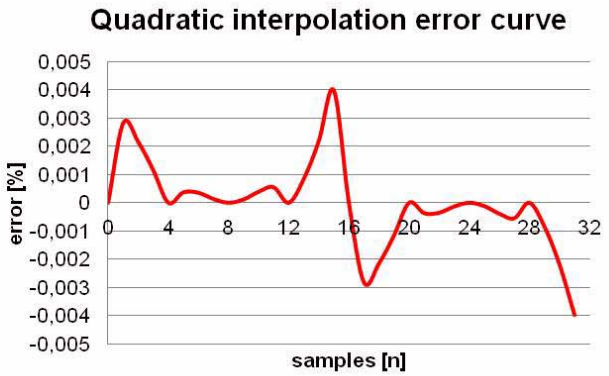
**Table 6. Parameter table for example 1**

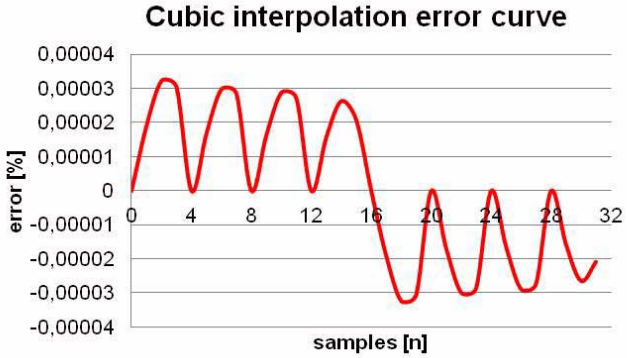
| | |
|---|---|
| Input signal | sinus 1st harmonics only |
| Input samples | 120 |
| Reguired FFT samples | 32 |
| Interpolation ratio | 120/32 = 3.75 |
| Equivalent metering use case | fADC = 6.144 MHz, OSR = 1024, sampling rate = 6144000/1024 = 6KHz, fINP = 50 Hz |



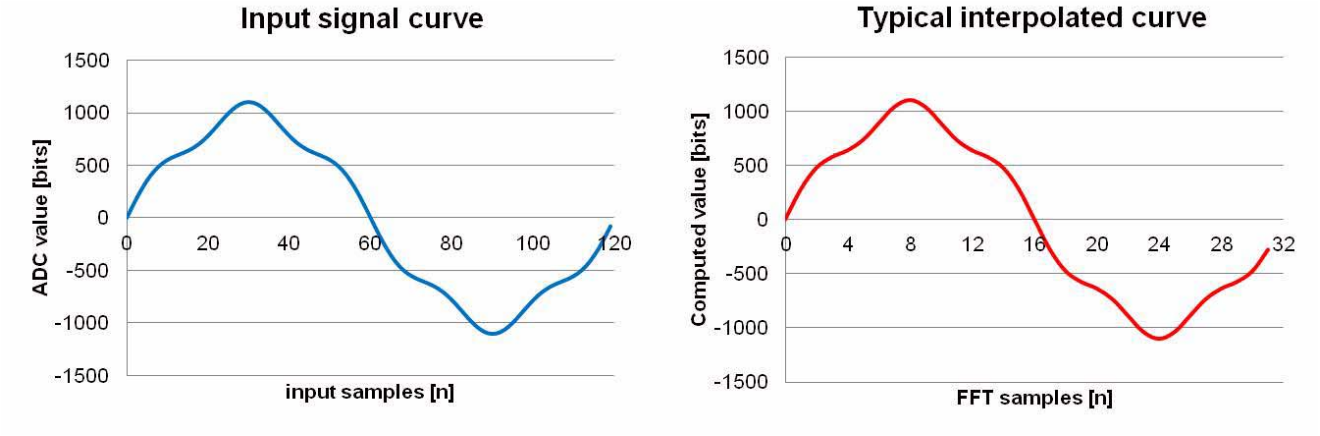



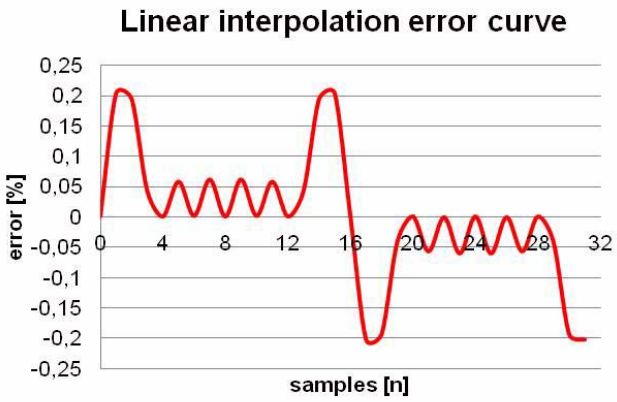



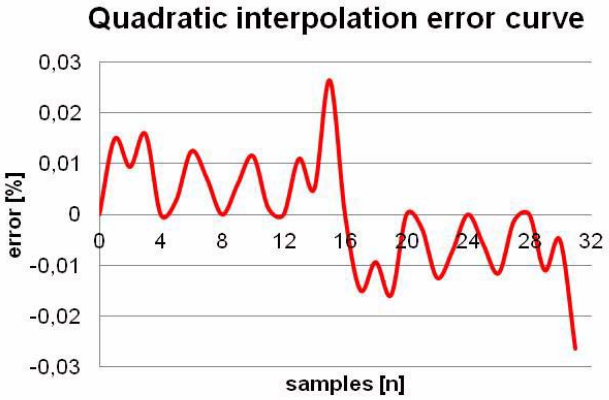
# Appendix C Results of simulation – Example 2

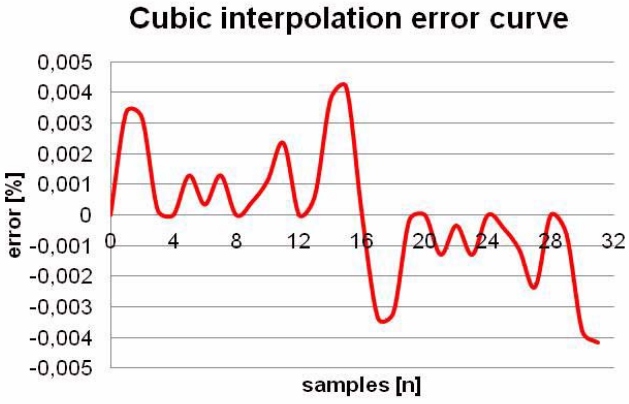
**Table 7. Parameter table for example 2**

| | |
|---|---|
| Input signal | sinus 1st harmonics + sinus 5th harmonics with 10% depth of modulation |
| Input samples | 120 |
| Reguired FFT samples | 32 |
| Interpolation ratio | 120/32 = 3.75 |
| Equivalent metering use case | fADC = 6.144 MHz, OSR = 1024, sampling rate = 6144000/1024 = 6KHz, fINP = 50 Hz |



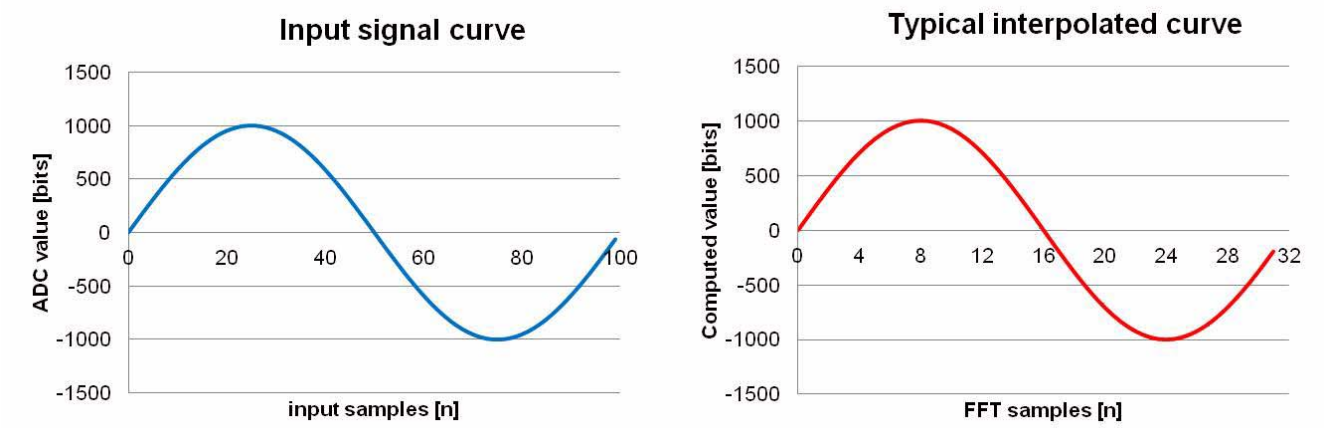



# Appendix D Results of simulation – Example 3

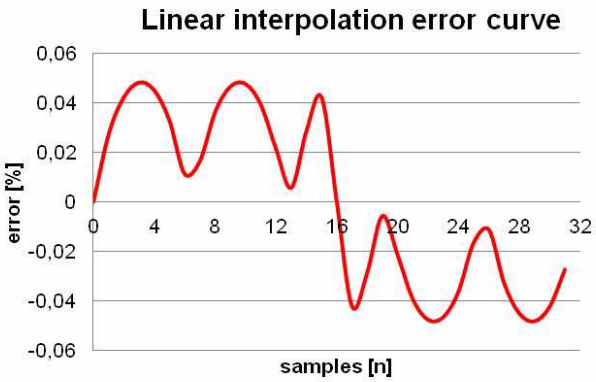
**Table 8. Parameter table for example 3**

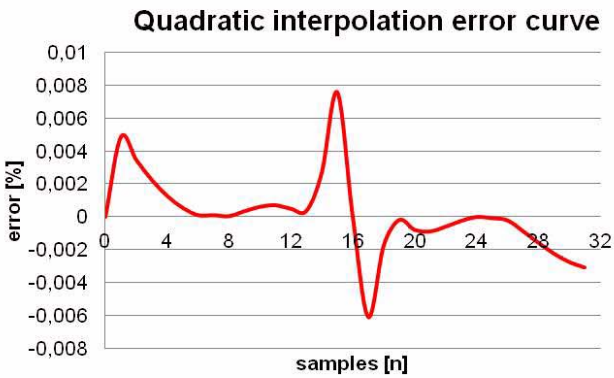
| | |
|---|---|
| Input signal | sinus 1st only |
| Input samples | 101 |
| Reguired FFT samples | 32 |
| Interpolation ratio | 101/32 = 3.15625 |
| Equivalent metering use case | $f_{ADC}$ = 6.144 MHz, OSR = 1024, sampling rate = 6144000/1024 = 6KHz, $f_{INP}$ = 59.4Hz |

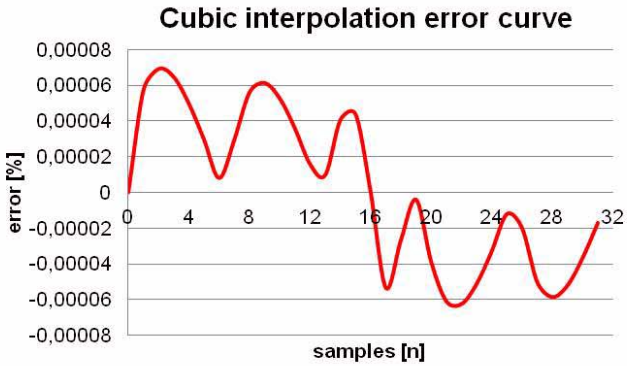








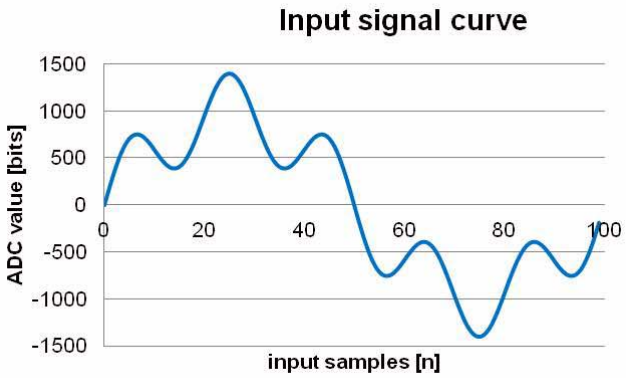



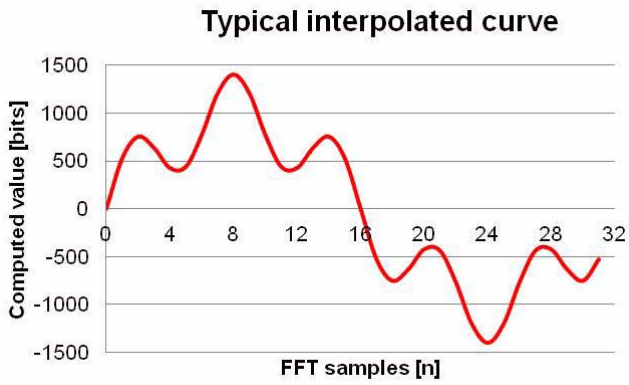



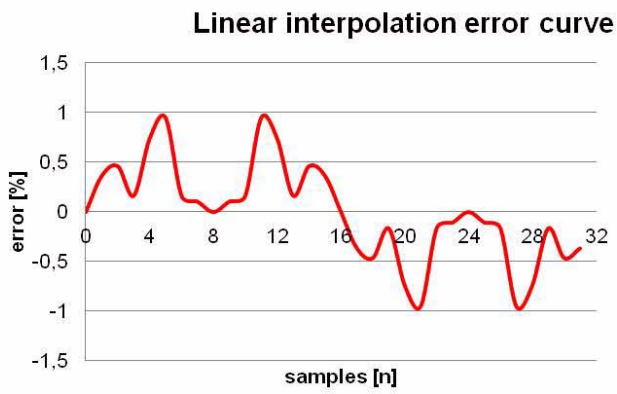
# Appendix E Results of simulation – Example 4

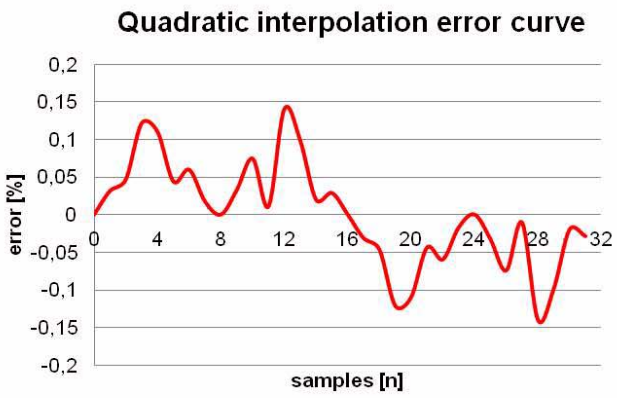
**Table 9. Parameter table for example 4**

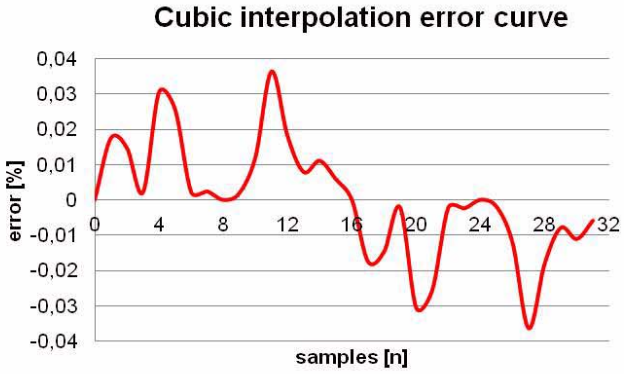
| | |
|---|---|
| Input signal | sinus 1st harmonics + sinus 5th harmonics with 40% depth of modulation |
| Input samples | 100 |
| Reguired FFT samples | 32 |
| Interpolation ratio | 100/32 = 3.125 |
| Equivalent metering use case | fADC = 6.144 MHz, OSR = 1024, sampling rate = 6144000/1024 = 6KHz, fINP = 60 Hz |



## *How to Reach Us:*

**Home Page:**
freescale.com

**Web Support:**
freescale.com/support



Document Number: AN4847
Rev. 2
07/2015