FFT in Metering Applications Using KM3x / KM3x_256 MCUs (ARM® Cortex® -M0+)

Luděk Šlosarčík | System Application Engineer

August.27.2015

External Use

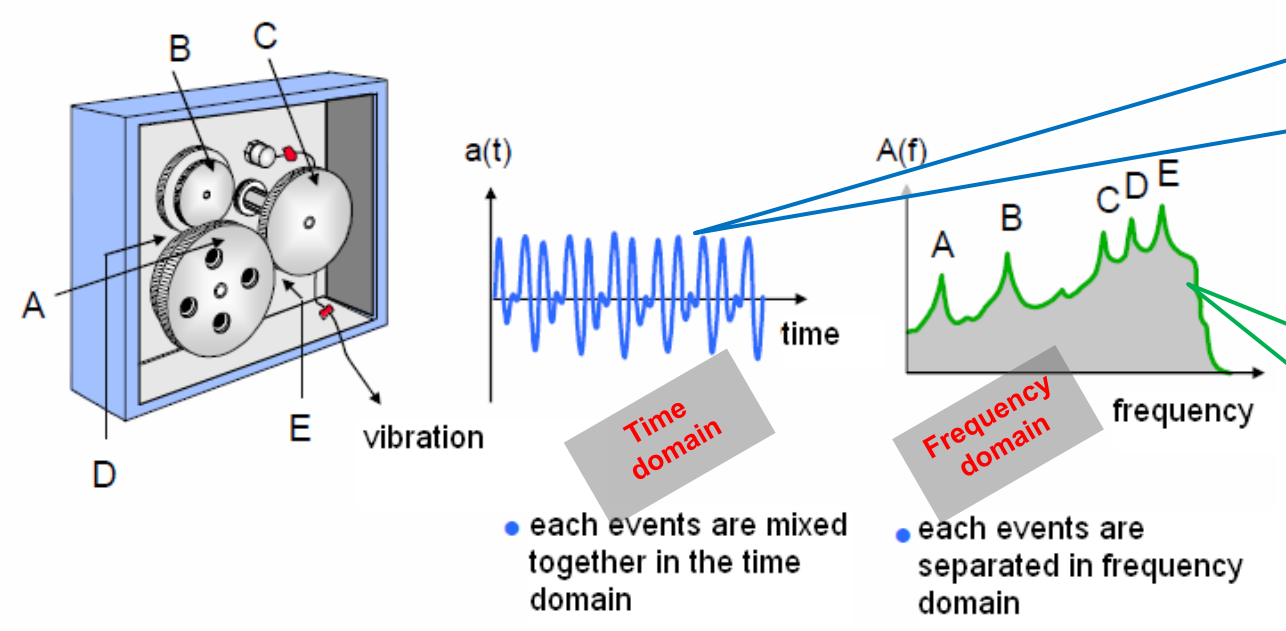
Agenda

- Digital signal processing theory
- Introduction to Freescale FFT metering library
- Using KM3x_256 MCU with FFT algorithm (examples)
- Using Freescale FFT metering library in C-code (tips and tricks)
- Measuring energy errors on real hardware
- FFT-based metering library (conclusion)

Digital signal processing theory

Time vs. frequency domain

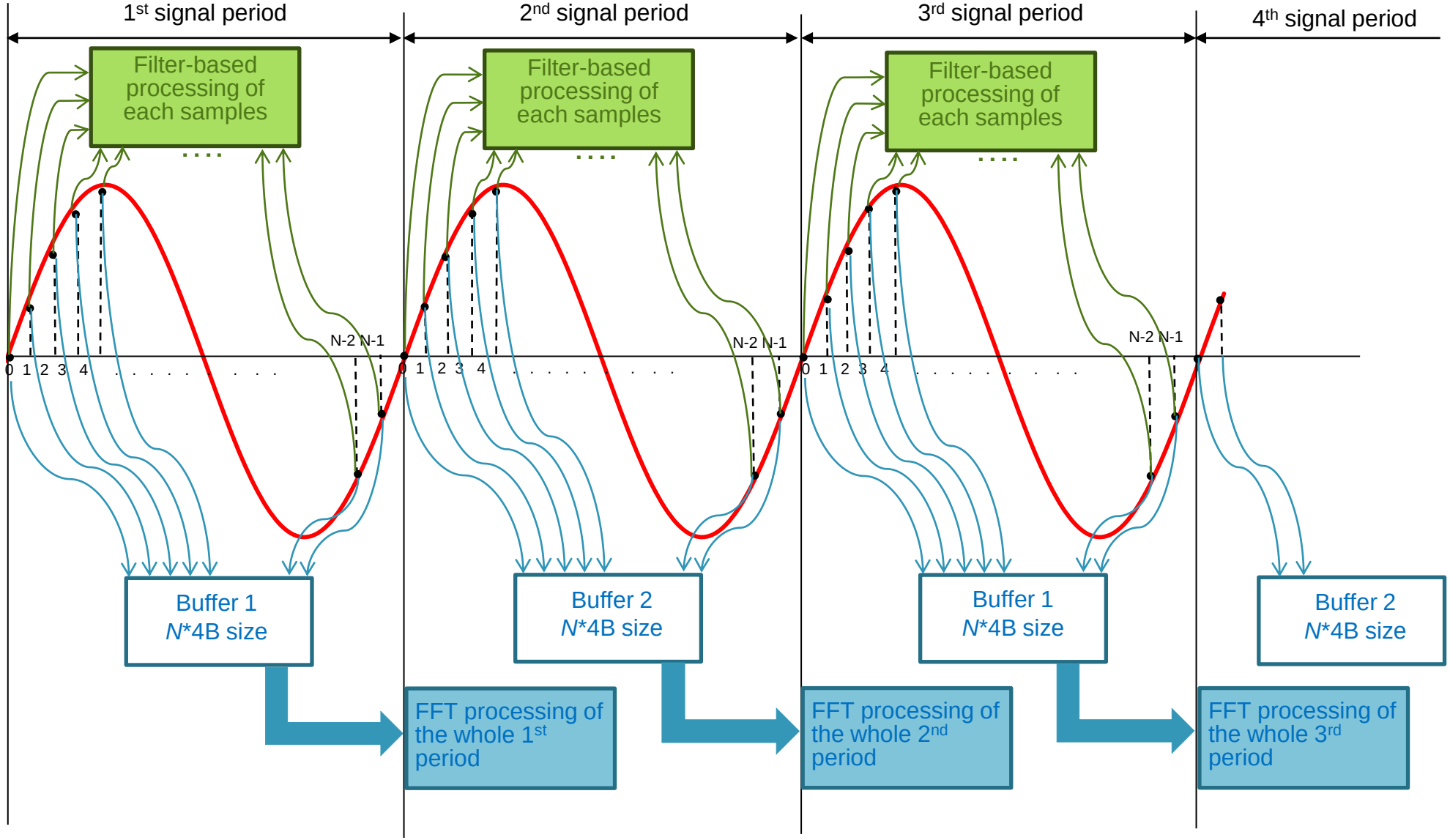
- Freescale offers algorithms that calculates metering quantities in either time or frequency domain:



Filter-based metering algorithm
- *processes signals in the time domain*
- *leverages IIR filters*
- *based on elementary 32/64-bit fractional arithmetic*
- *See also AN4265*

FFT-based metering algorithm
- *processes signals in the frequency domain*
- *leverages DFT computation*
- *based on elementary 32/64-bit fractional arithmetic*
- *See also AN4255, AN4847*

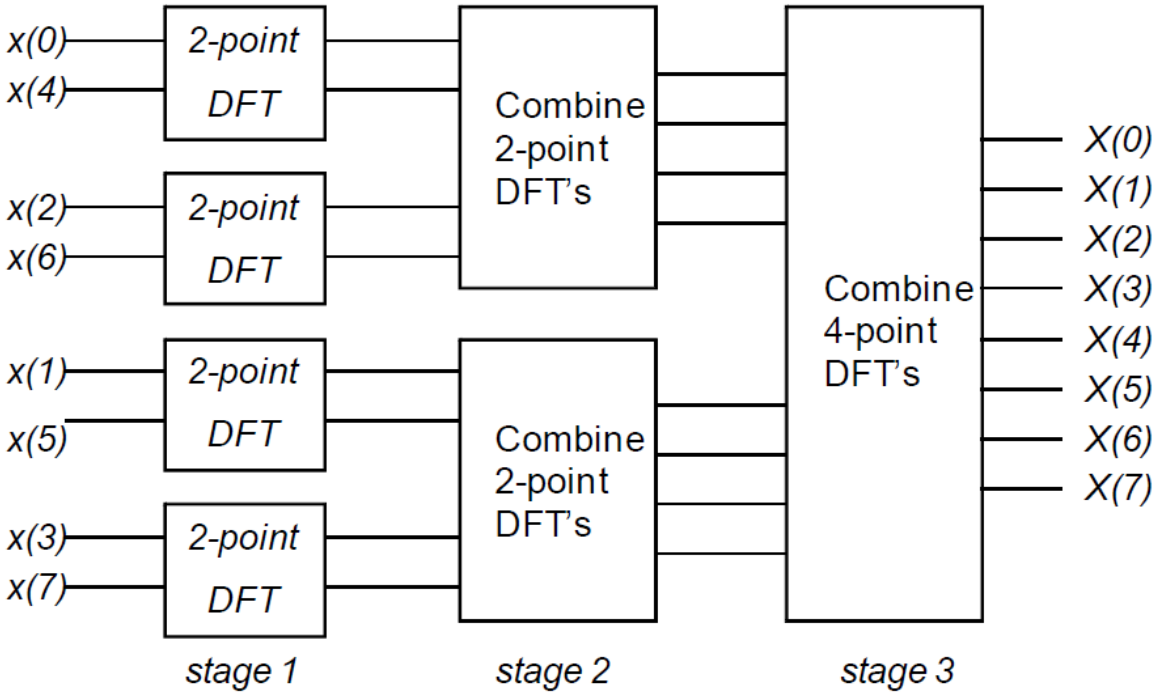
Filter-based vs. FFT-based data processing



Discrete Fourier Transform vs. Fast Fourier Transform

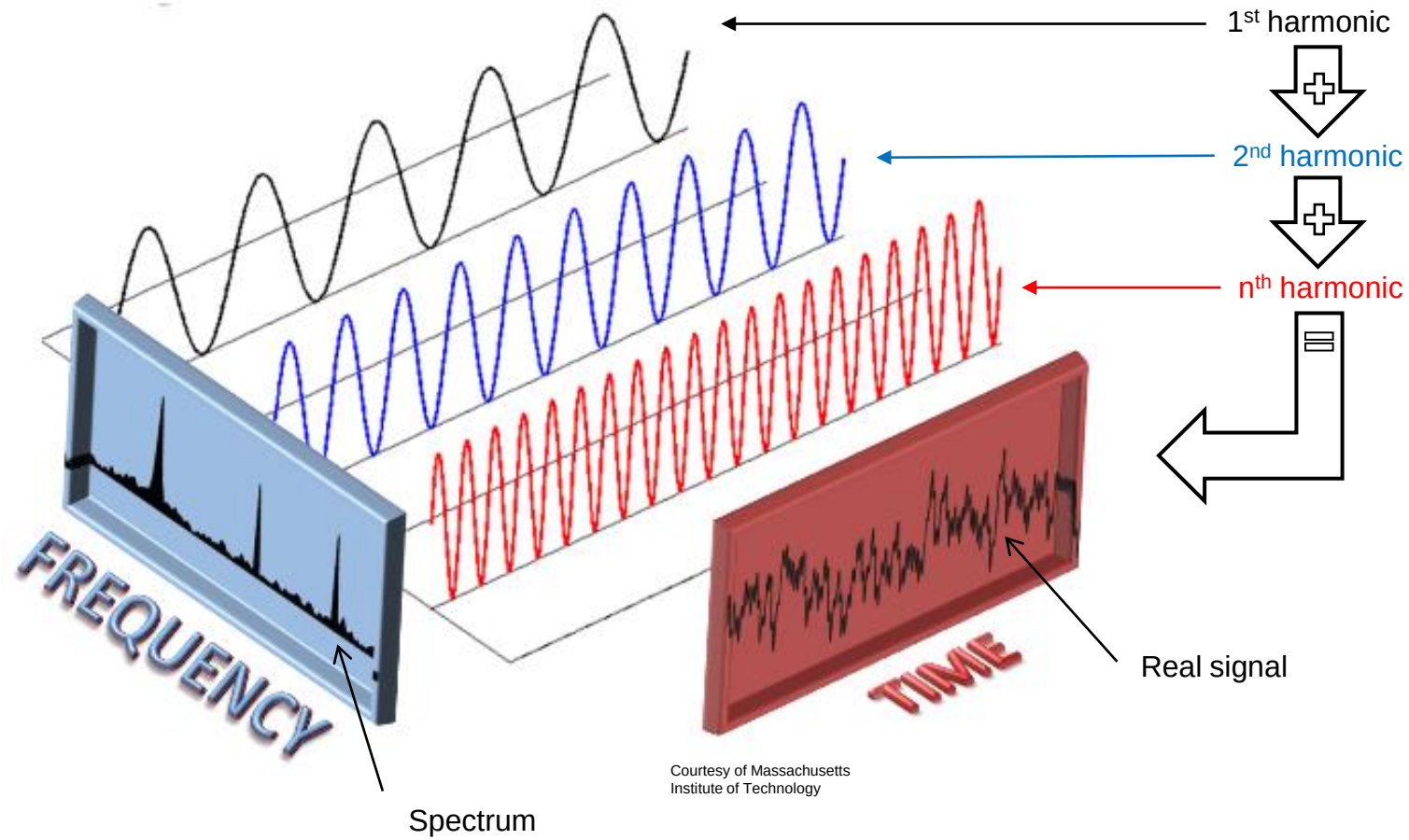
- **DFT** = special kind of the discrete transform which converts a time-domain function into its frequency domain representation or vice versa (IDFT). DFT or IDFT refers to a mathematical transformation, regardless of how it is computed. The DFT *N*-points calculation in the naive way takes O(N^2) arithmetical operations.

- **FFT** = specific family of algorithms for computing the DFT very effectively. The basic idea of the FFT is to decompose the DFT of a time domain sequence length *N* into successively smaller DFTs whose calculation require less arithmetic instruction. The FFT *N*-points calculation takes only O($N \cdot \log N$) arithmetical operations.
- **DFT/FFT** computing ratio is: $N / \log N$



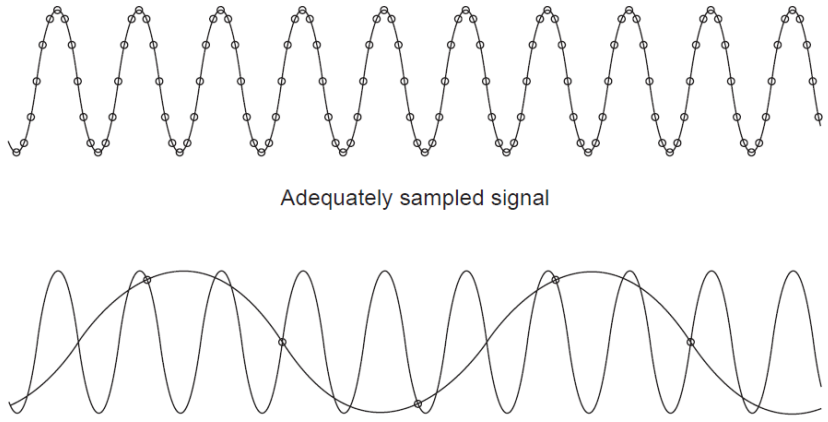
Understanding Fast Fourier Transformation

- The basic idea is: each periodic signal can be substituted by a finite sum of harmonic signals, each with a different frequency.



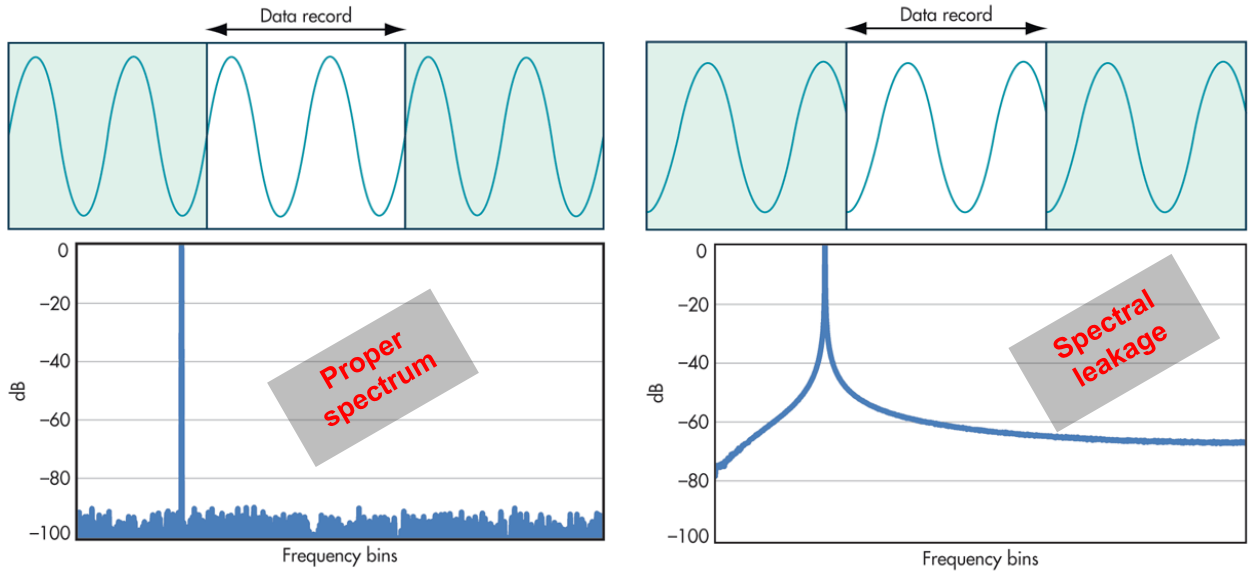
Signal processing – known issues

- **Aliasing (wrap-around error):**
  Aliasing results when the sampling does not occur fast enough.

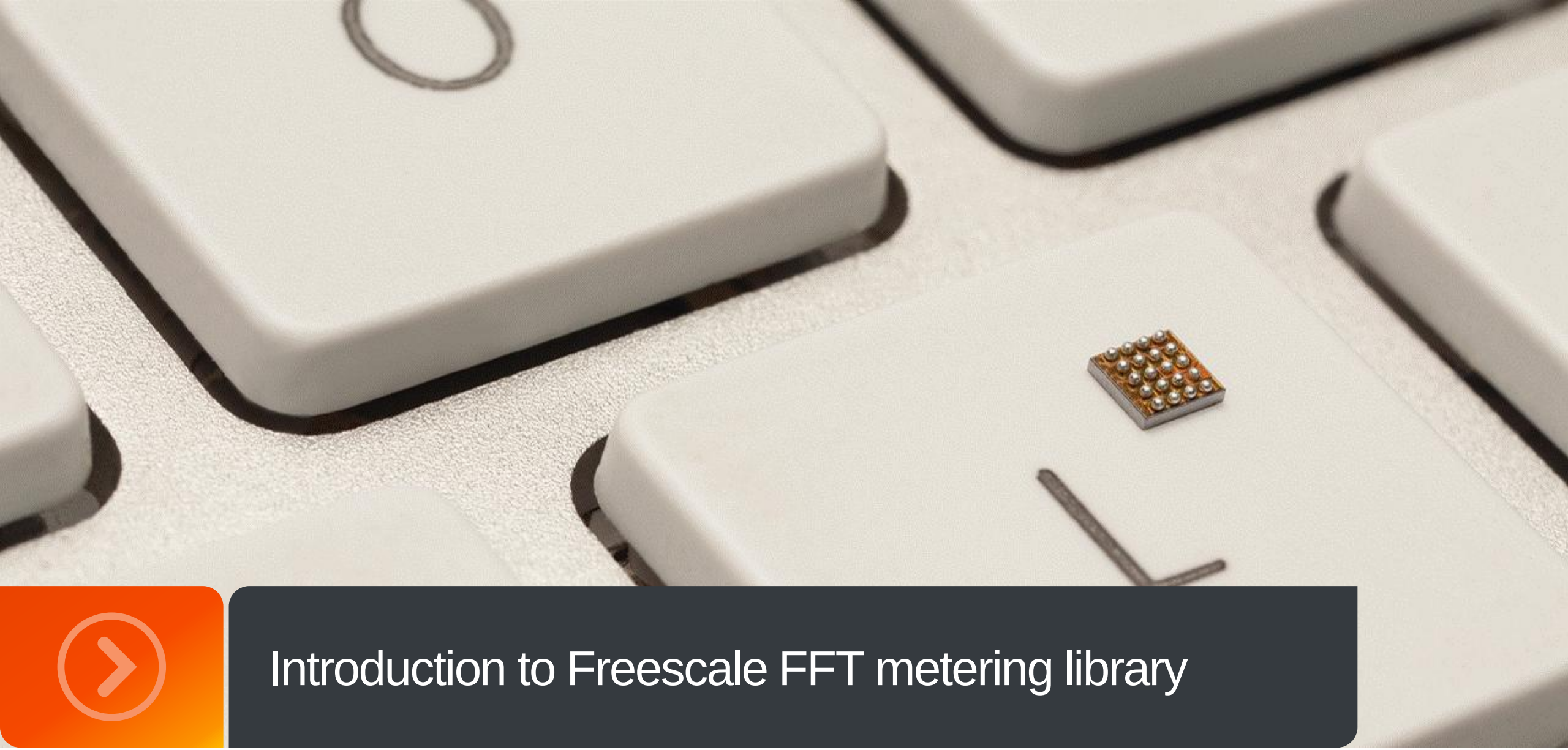


Courtesy of National Instruments

- **Leakage:**
  When the measured signal is not periodic in the sample interval, incorrect estimates of the amplitude and frequency occur. This error is referred to as leakage. For solving this issue, oversampling with interpolation is used. Alternatively, weighting functions called windows, can be also used.



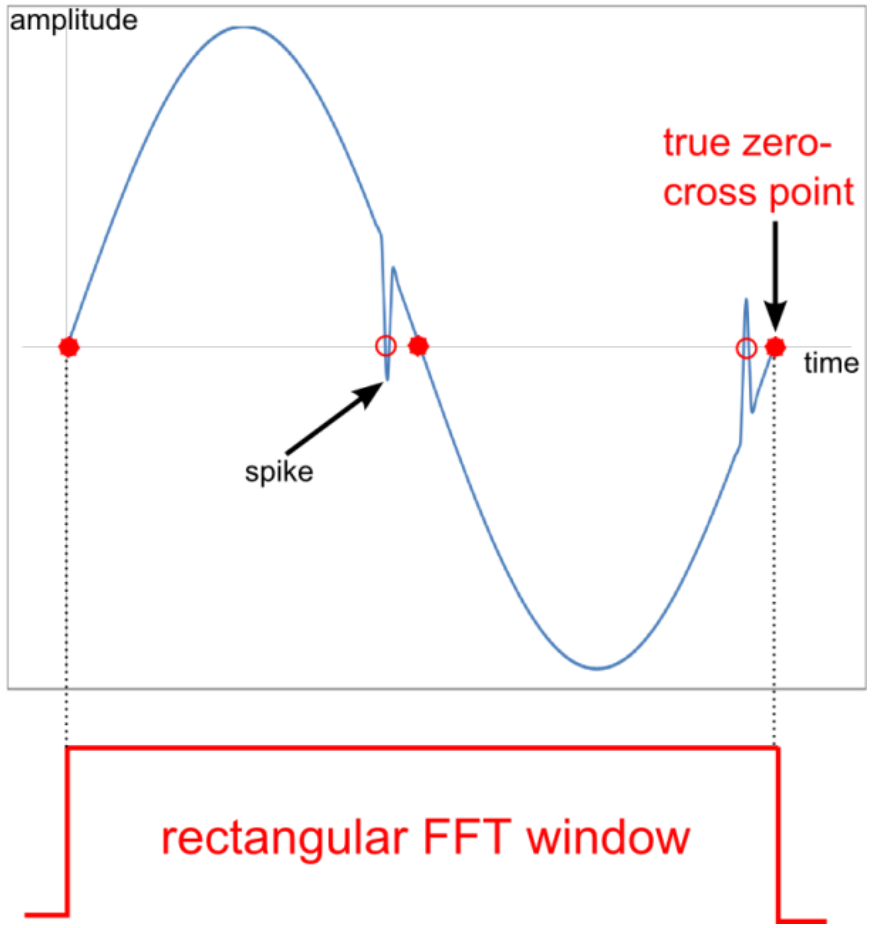
Courtesy of TI



Introduction to Freescale FFT metering library

FFT – basic requirements

- Number of samples (*N*) must be power-of-two (32,64,128,…) – valid for the best known FFT algorithm called Radix-2.
- Ensures that processed samples represent full signal period.
- Use some interpolation technique to generate exactly power-of-two samples on a metering device with SD ADCs for entire signal period.
- Sampling frequency must be at least 2x or higher than the maximum frequency included in the input signal (Nyquist theorem).
- Anti-aliasing filter (low-pass analog) should be used to prevent aliasing.



FFT – basic computing formulas

Complex power vector (in Cartesian form) is defined as:

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

Real part of the complex power = Active power

Imaginary part of the complex power = Reactive power

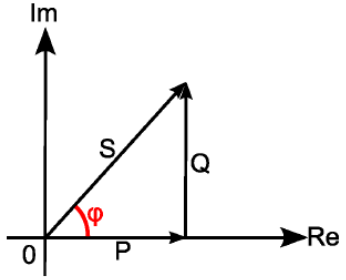
Apparent power = Complex power magnitude:

For the whole spectrum

$$S_{tot} = U_{RMS} \cdot I_{RMS}$$

For the fundamental frequency only

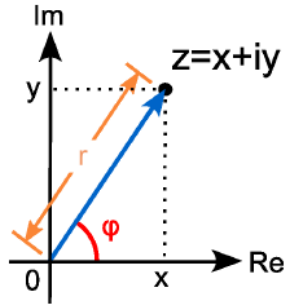
$$S = \sqrt{\left(P^2 + Q^2\right)}$$



Root Mean Square computing (in Cartesian form) is defined as:

$$I_{RMS} = \sqrt{\sum_{k=0}^{\frac{N}{2}-1} \left(I^2_{RE}(k) + I^2_{IM}(k)\right)} \quad \text{or} \quad I_{RMS} = S / U_{RMS}$$

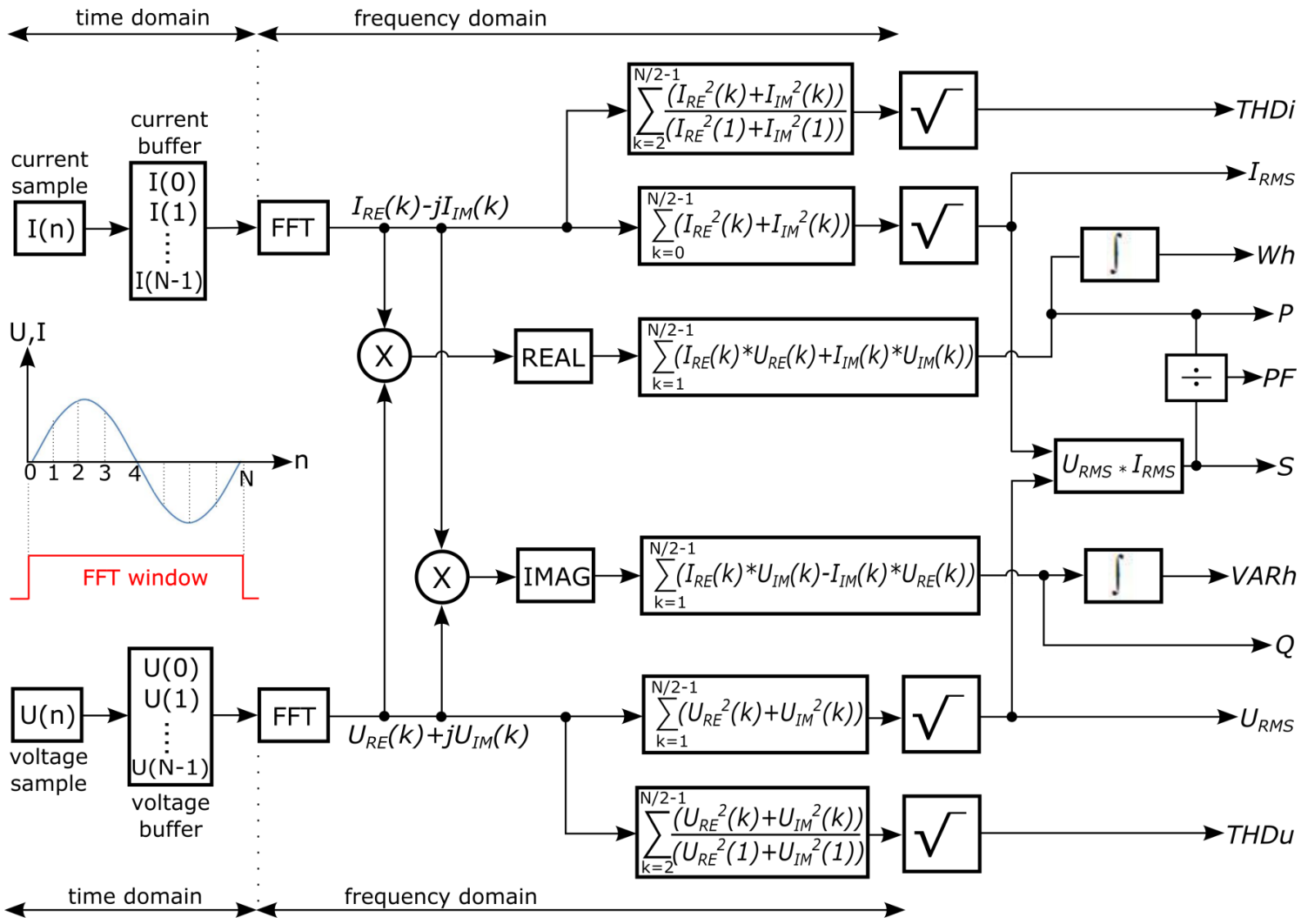
$$U_{RMS} = \sqrt{\sum_{k=1}^{\frac{N}{2}-1} \left(U^2_{RE}(k) + U^2_{IM}(k)\right)}$$



Where: $I_{RE}(k), U_{RE}(k)$ are real parts of k^{th} harmonics of input current/voltage
$I_{IM}(k), U_{IM}(k)$ are imaginary parts of k^{th} harmonics of input current/voltage

For more information see AN4255

Freescale FFT-based metering library block diagram



For more information see AN4255

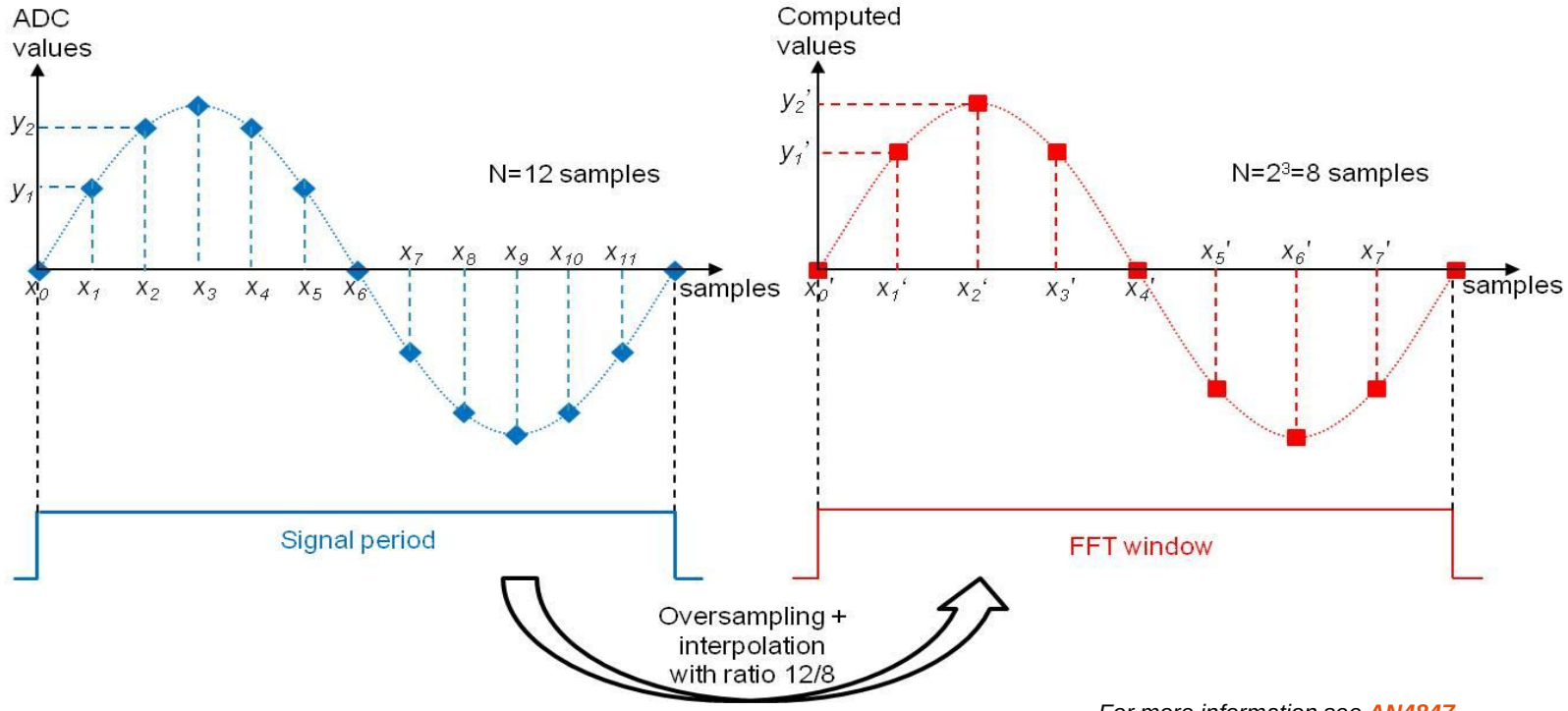
FFT use case – SAR ADCs vs. SD ADCs

- **Using the FFT on SD ADCs:**

  There are two asynchronous processes, Mains and ADC, which cannot be slightly adjusted. In this case we use **oversampling** with final counting the right samples (**Interpolation**).

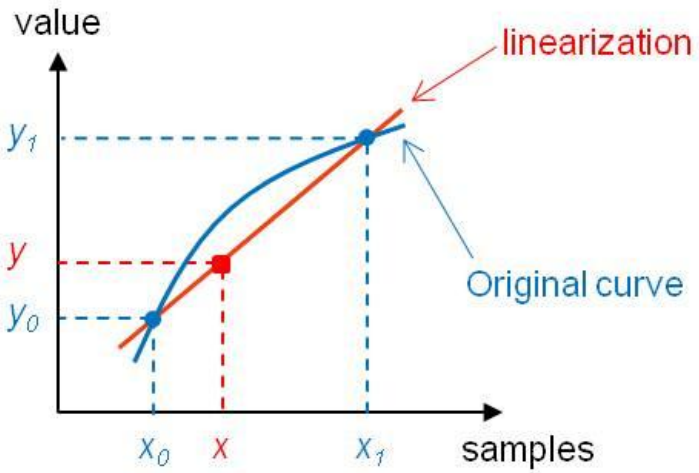
- **Using the FFT on SAR ADCs:**

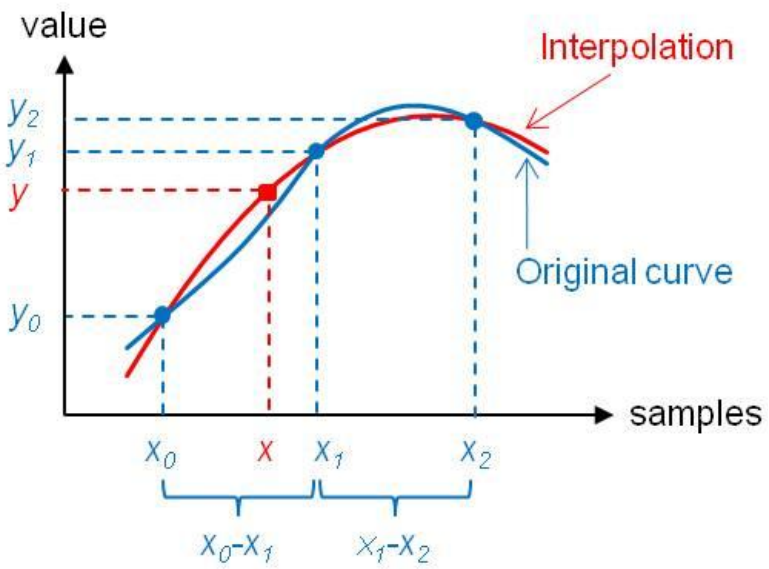
  Sampling rate is synchronized with the mains due to adjusting the sampling rate slightly by the **PDB** (a special timer for HW-triggering).

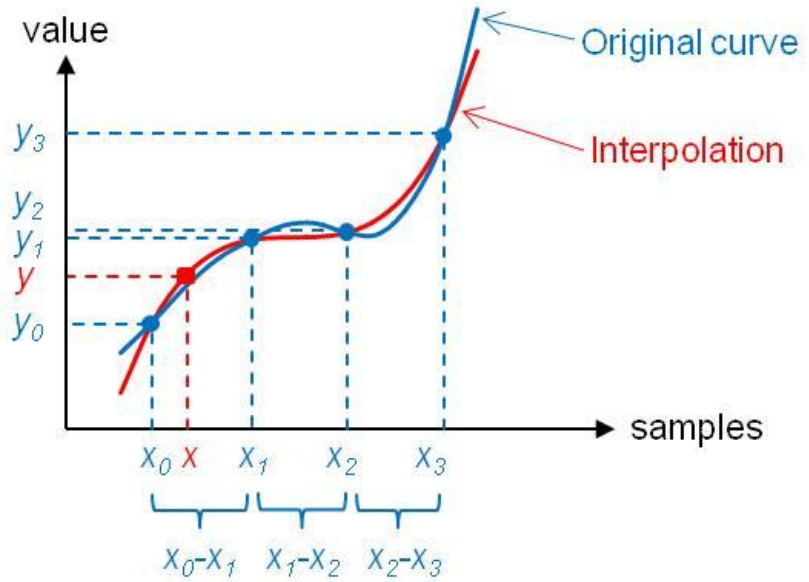


For more information see AN4847

Typical interpolation use cases for SD ADC







1st order (Linear): high speed, worse precision for higher harmonics

2nd order (Quadratic): good rate speed/precision

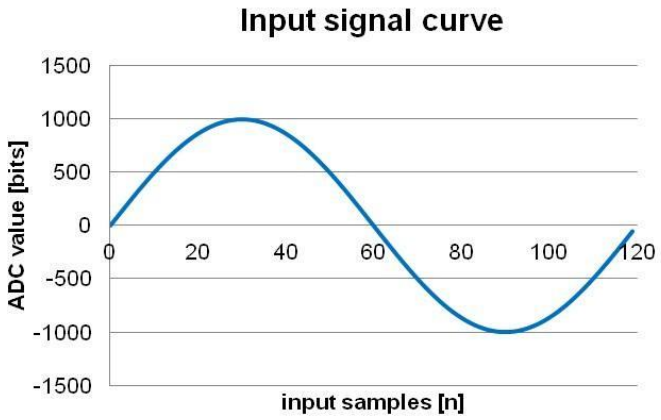
3rd order (Cubic): very good precision

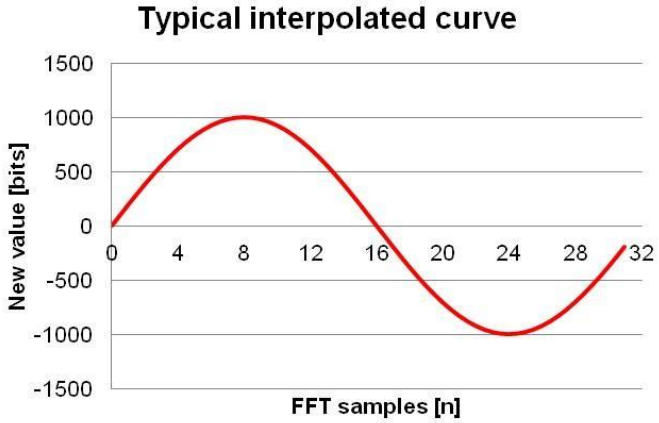
Note: the 4rd order and higher is not much effective for the real time computing.

For more information see AN4847

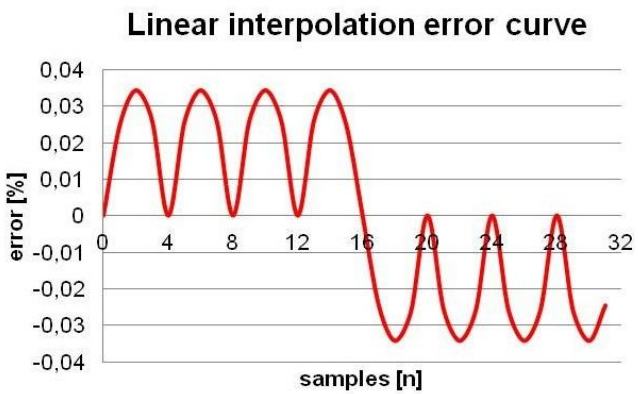
FFT – interpolation error curves (example #1)

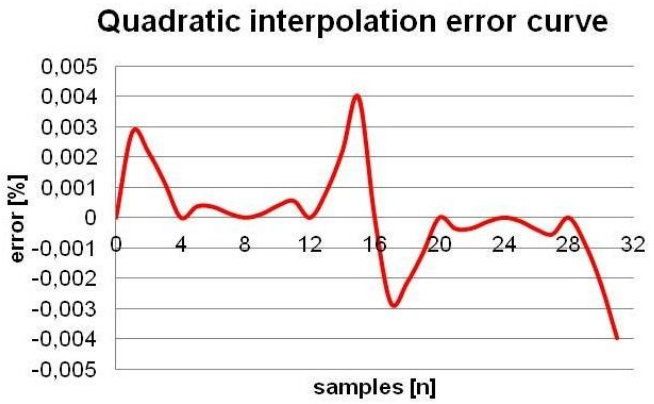
- Input signal: 1st harmonic only
- Interpolation ratio: 120/32
- Equivalent metering use case: f_{ADC} =6.144MHz, OSR=1024⇒f_s=6kHz, f_{INP} =50Hz

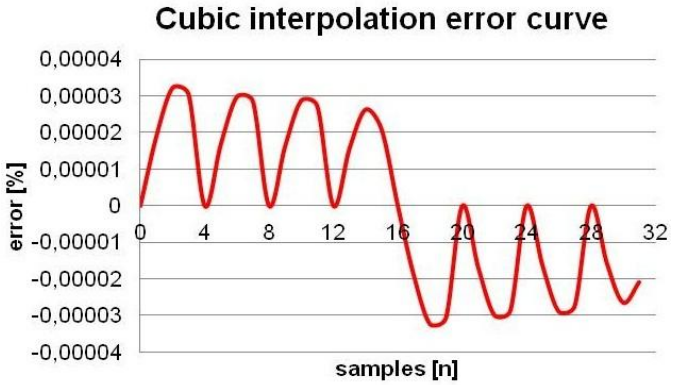




Ideal case!



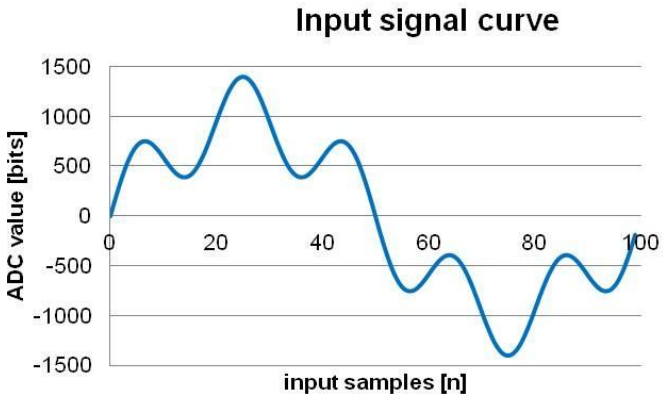


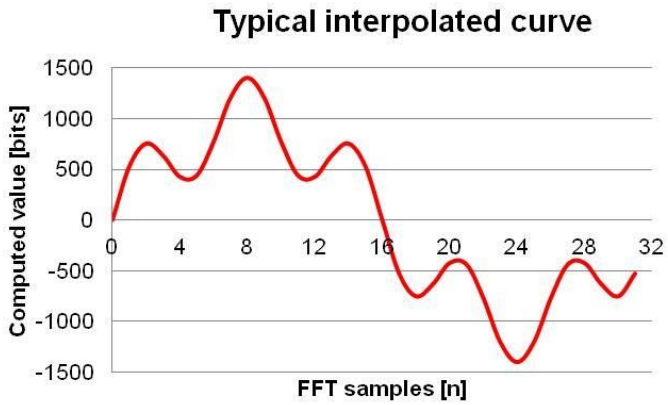


Note: average value of these errors are approx. zero during one signal period.

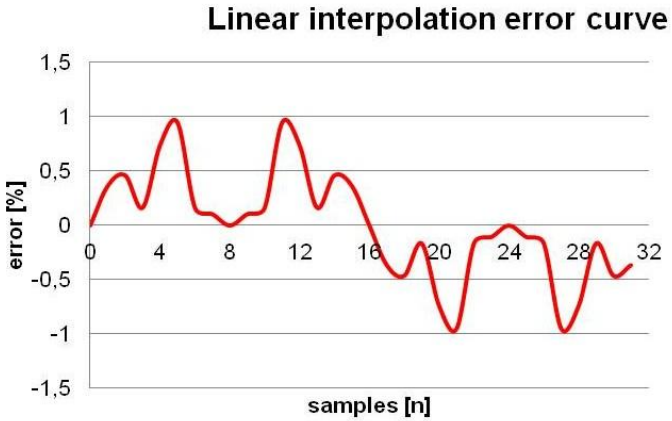
FFT – interpolation error curves (example #2)

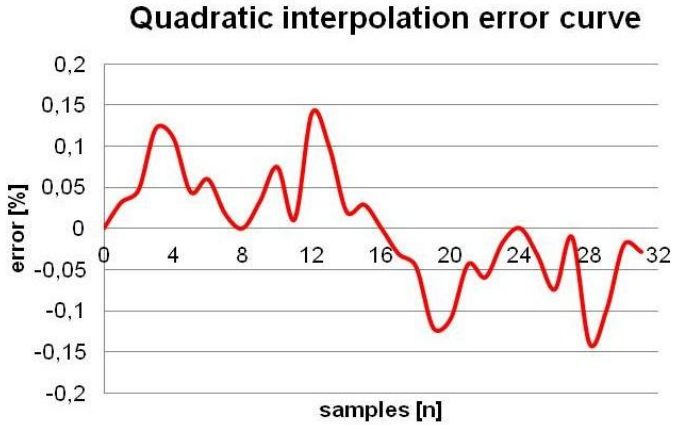
- Input signal: 1st harmonic + 5th harmonic with 40% depth of modulation
- Interpolation ratio: 100/32
- Equivalent metering use case: f_{ADC} =6.144MHz, OSR=1024⇒f_s=6kHz, f_{INP} =60Hz

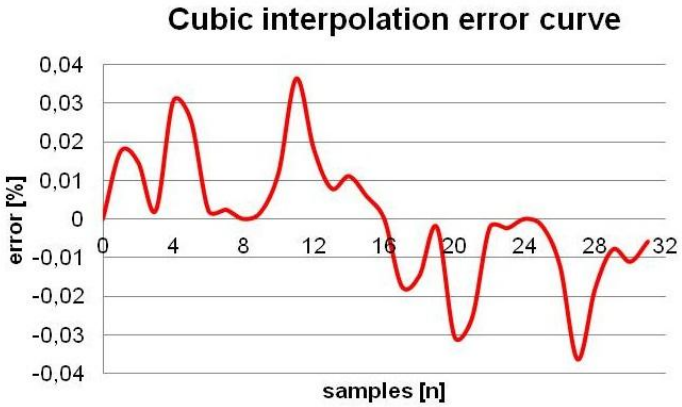




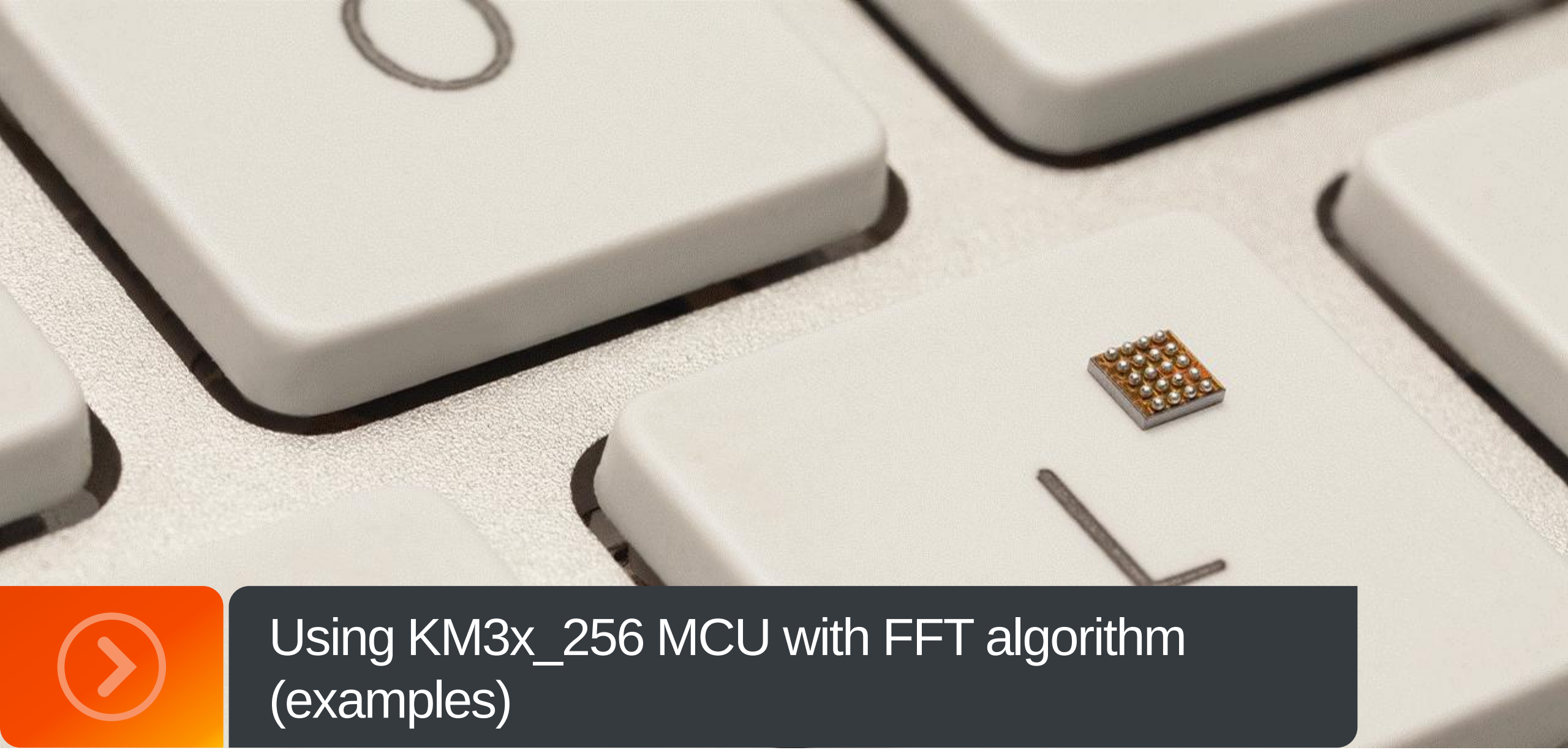
Worse case!





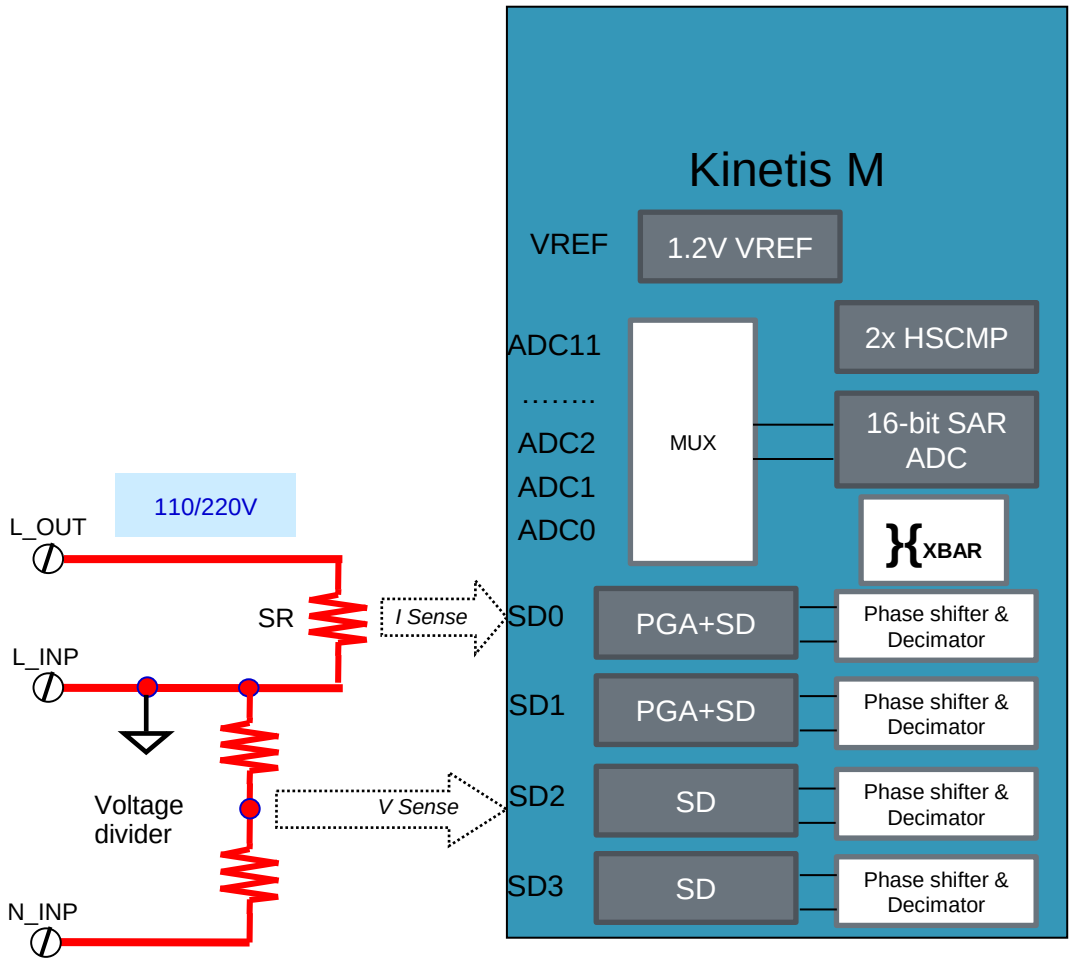


Note: average value of these errors are approx. zero during one signal period.



Using KM3x_256 MCU with FFT algorithm (examples)

1-phase power meter using KM3x_256 MCU



MCU resources:

- PGA+SD (current measurement)
- SD (voltage measurement)
- Phase Shifter (for U-I phase shift compensation)
- Comparator (zero-cross detection)
- Peripheral Crossbar (optional)
- Internal Voltage reference (for AFE and CMP)
- PLL 12MHz (for AFE, CMP and core)
- TMR (for synchronous processing mode only)

Data processing (async. IRQ mode):

- Reading U+I data periodically every OSR/f_{ADC} [s] and saving these into buffers – IRQs are caused by the slowest channel.
- When a new ZCD occurs, both the FFT calculation process for the previous period and a new sampling process for the current period are started-up, etc.

1-phase power meter using KM3x_256 MCU (continued)

Use case #1 (sync. processing):

AFE settings :
- 24-bit SD ADC
- f_{ADC}=6.144MHz,OSR=512⇒f_s=12kHz
- HW-triggered single conversion mode

FFT settings :
- f_{inp}~50Hz
- No interpolation
- 64 input samples (32 harmonics)

MCU workload (@12MHz):
- *METERLIBFFT1PH_CalcMain* =4.15ms

RAM requirement (@4B AFE result):
- U-buffer: 2x4Bx64samples=512B
- I-buffer: 2x4Bx64samples=512B
- Total: 1024B

Use case #2 (async. processing):

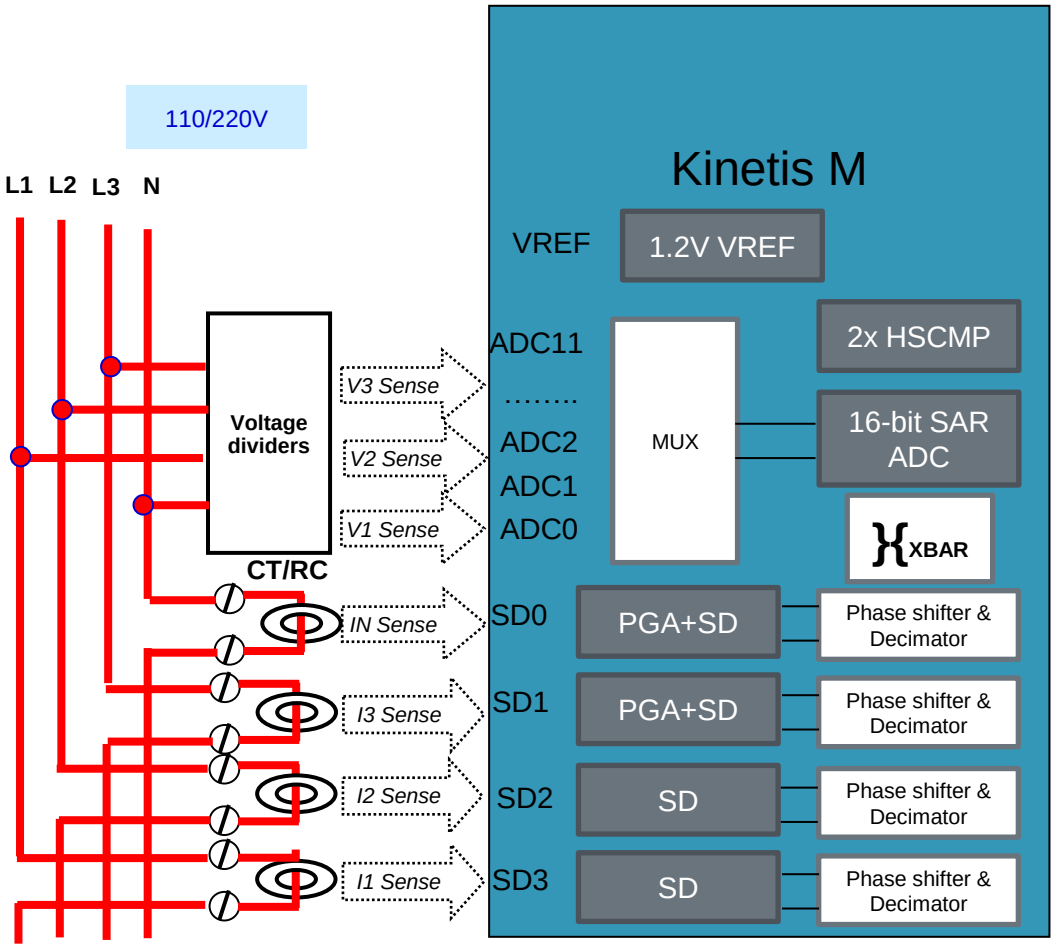
AFE settings :
- 24-bit SD ADC
- f_{ADC}=6.144MHz,OSR=1024⇒f_s=6kHz
- SW-triggered continuous conversion mode

FFT settings:
- f_{inp}~50Hz ⇒120 input samples
- Cubic 3rd order interpolation (ratio 120/64)
- 64 output samples (32 harmonics)

MCU workload (@12MHz):
- *METERLIBFFT1PH_CalcMain* + *METERLIBFFT1PH_Interpolation* = 4.15ms + 2.35ms=6.5ms

RAM requirement (@4B AFE result):
- U-buffer: 2x4Bx120samples=960B
- I-buffer: 2x4Bx120samples=960B
- Total: 1920B

3-phase power meter using KM3x_256 MCU



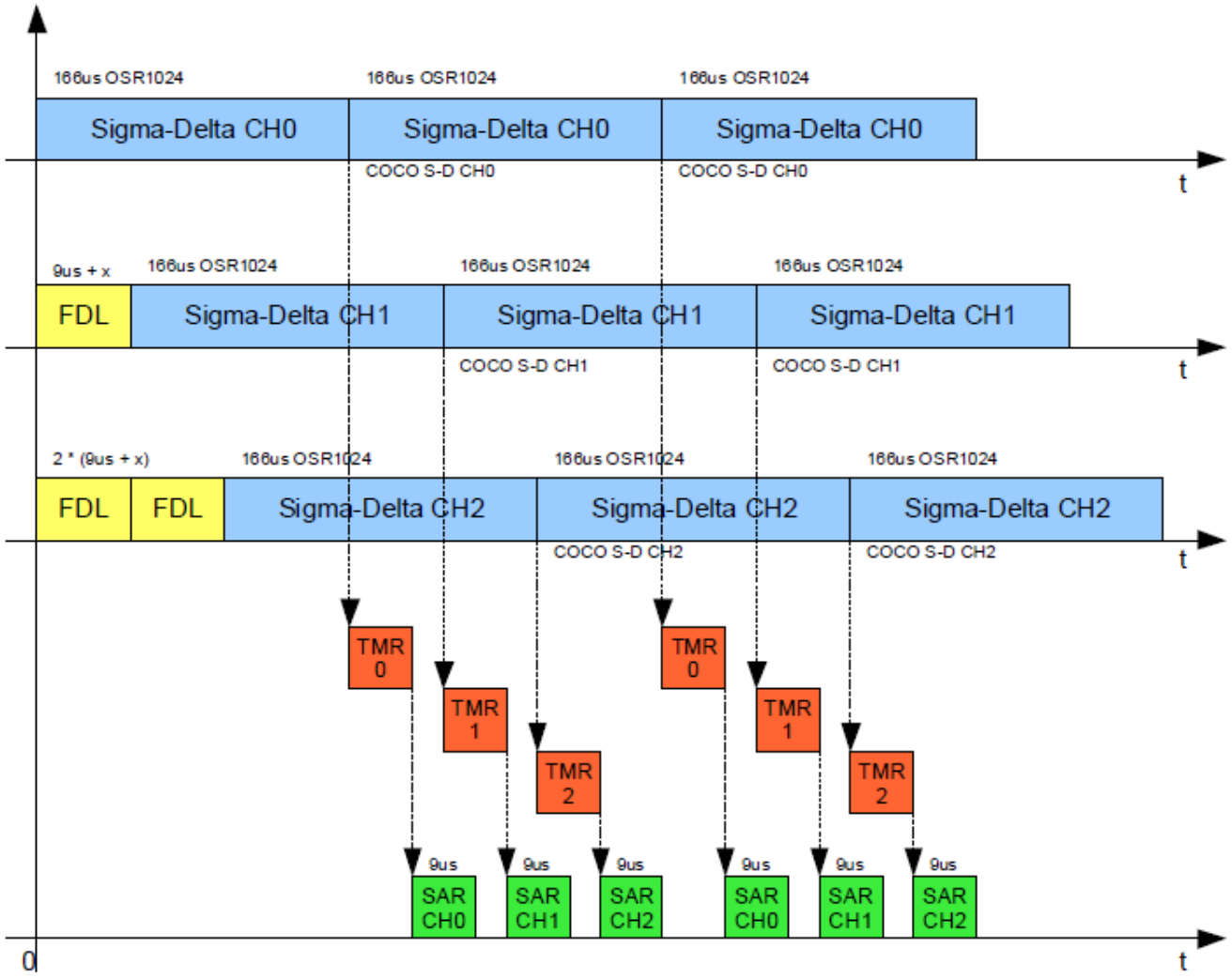
MCU resources:

- 4 x SD ADCs (for current measurement)
- SAR ADC (3 channels for voltage measurement)
- Comparator (3 channels for zero-cross detection)
- Peripheral Crossbar (AFE->TMR->SAR)
- Internal Voltage reference (for AFE,ADC,CMP)
- PLL 12MHz (AFE,ADC,CMP), FLL 48MHz (core)
- Phase Shifter (fix delay between channel)
- 3 x TMR (for U-I phase shift compensation)
- 1 x TMR (for synchronous processing mode only)

Data processing (async. IRQ mode):

- AFE COCx flag starts TMRx through XBAR every OSR/f_{ADC} [s]. SAR ADCx is HW triggered by TMRx – each U+I data pair is read in SARx IRQ routine and saved into the buffer (DMA may be also used instead of SARx IRQ).
- When a new ZCD occurs, both the FFT calculation process for the previous period and a new sampling process for the current period are started-up, etc.

3-phase power meter using KM3x_256 MCU (continued)



Sampling signal chain:

AFE COCx

⇓

XBAR

⇓

TMRx

⇓

XBAR

⇓

SAR ADCx sample

3-phase power meter using KM3x_256 MCU (continued)

Use case #1 (sync. processing):

AFE settings :
- 24-bit SD ADC
- f_{ADC}=6.144MHz,OSR=512⇒f_s=12kHz
- HW-triggered single conversion mode

ADC settings :
- 16-bit SAR ADC
- HW-triggered single conversion mode

FFT settings:
- f_{inp}~50Hz, no interpolation
- 64 input samples (32 harmonics)

MCU workload (@48MHz):
- *METERLIBFFT3PH_CalcMain* =3.11ms

RAM requirement (@4B AFE result):
- U-buffer: 3x2x4Bx64samples=1536B
- I-buffer: 3x2x4Bx64samples=1536B
- Total: 3072B

Use case #2 (async. processing):

AFE settings :
- 24-bit SD ADC
- f_{ADC}=6.144MHz,OSR=1024⇒f_s=6kHz
- SW-triggered continuous conversion mode

ADC settings :
- 16-bit SAR ADC
- HW-triggered single conversion mode

FFT settings:
- f_{inp}~50Hz ⇒120 input samples
- Cubic 3rd order interpolation (ratio 120/64)
- 64 output samples (32 harmonics)

MCU workload (@48MHz):
- *METERLIBFFT3PH_CalcMain* + *METERLIBFFT3PH_Interpolation* = 3.11ms + 1.76ms=4.87ms

RAM requirement (@4B AFE result):
- U-buffer: 3x2x4Bx120samples=2880B
- I-buffer: 3x2x4Bx120samples=2880B
- Total: 5760B

Using Freescale FFT metering library in C-code (tips and tricks)

Example code for 1-phase FFT asynchronous processing

- Process 1: AFEx IRQ (@166.6µs)

```
static void afech2_callback (AFE_CH_CALLBACK_TYPE type, int32 result)
{
   if (type == COC_CALLBACK)
   {
      u24_sample = result;                          /* reading both AFE samples */
      i24_sample = AFE_ChRead (CH0);
      if (samples_cnt < (BUFF_LENGTH(F_MIN)))  /* buffers overflow protection */
      {
         u_re[buffer][samples_cnt] = u24_sample;
         i_re[buffer][samples_cnt] = i24_sample;
         samples_cnt++;                        /* samples counter incrementation */
      }
   }
}
```

- Process 3: LPTMR IRQ (@500 ms)

```
static void lptmr_callback (void)
{
   /* Mains frequency reading and averaging...                                  */
   if (TMR_ReadClrCaptReg(CH0, &tmp16)==TRUE) freq_avrg += (TMR2FREQ(tmp16));
   if (++freq_cnt == FREQ_PERIOD)
   {
      val.f = freq_avrg/FREQ_PERIOD;                    /* averaged frequency */
      freq_cnt = freq_avg = 0;
   }
   /* Other processing, such as showing values, etc. */
}
```

- Process 2: CMPx IRQ (@20ms)

```
static void cmp_callback (CMP_CALLBACK_SRC module, int8 status)
{
   if (module == CMP1_CALLBACK)                          /* Is module CMP1? */
   {
      if (status & (CMP_SCR_CFF_MASK))                   /* Is edge rising? */
      {
         if (samples_cnt > BUFF_LENGTH(F_MAX)) /* peak in the mains detected? */
         {
            samples = samples_cnt;  /* total number of samples per one period */
            samples_cnt = 0;       /* clearing the counter for the next cycle */
            METERLIBFFT1PH_InitMainBuff(&ui, u_re[buffer], i_re[buffer], u_im, i_im, NULL);
            buffer = buffer ^ 1;             /* swap buffer for next writting */
            data_processing ();                    /* call the FFT processing */
         }
      }
   }
}
```

FFT processing

meterlibFFT

```
static void data_processing (void)
{
    int32 tmr_cmp;
    uint16 i;
    /* performs the interpolation only for asynchronous processing        */
    METERLIBFFT1PH_Interpolation(&ui, ORD2, ORD3, samples);
    /* main calculation (FFT, I-signal post-conditioning, scaling, averaging) */
    METERLIBFFT1PH_CalcMain(&ui);
    /* calculates active energy increment and energy timer treshold value     */
    tmr_cmp = METERLIBFFT1PH_CalcWattHours(&ui, &ramcfg.energy.wh_i, &ramcfg.energy.wh_e, val.f);
    if (tmr_cmp > 0)
    {
       TMR_SetCntrVal(CH1, 0);
       TMR_SetCmp1Val(CH1, TMRCMPVAL(tmr_cmp, TMR1CLK));
       TMR_SetCountMode(CH1, COUNT_POSEDGE);
    }
    /* calculates reactive energy increment and energy timer treshold value   */
    tmr_cmp = METERLIBFFT1PH_CalcVarHours(&ui, &ramcfg.energy.varh_i, &ramcfg.energy.varh_e, val.f);
    if (tmr_cmp > 0)
    {
       TMR_SetCntrVal(CH2, 0);
       TMR_SetCmp1Val(CH2, TMRCMPVAL(tmr_cmp, TMR2CLK));
       TMR_SetCountMode(CH2, COUNT_POSEDGE);
    }
}
```

Freq. settings

Example code for 1-phase FFT synchronous processing

- Process 1: TMRx->XBAR->AFEx IRQ (@20ms/FFT_SAMPLES)

```
static void afech2_callback (AFE_CH_CALLBACK_TYPE type, int32 result)
{
   if (type == COC_CALLBACK)
   {
      u24_sample = result;                          /* reading both AFE samples */
      i24_sample = AFE_ChRead (CH0);
      u_re[buffer][samples_cnt] = u24_sample;
      i_re[buffer][samples_cnt] = i24_sample;
      if (++samples_cnt == FFT_SAMPLES) {
         samples_cnt = 0;
         METERLIBFFT1PH_InitMainBuff(&ui, u_re[buffer], i_re[buffer], u_im, i_im, NULL);
         buffer = buffer ^ 1;                  /* swap buffer for next writting */
         data_processing ();                        /* call the FFT processing */
      }
   }
}
```

meterlibFFT

FFT processing

```
static void data_processing (void)
{
    int32 tmr_cmp;
    uint16 i;
    /* main calculation (FFT, I-signal post-conditioning, scaling, averaging) */
    METERLIBFFT1PH_CalcMain(&ui);
    /* calculates active energy increment and energy timer treshold value     */
    tmr_cmp = METERLIBFFT1PH_CalcWattHours(&ui, &ramcfg.energy.wh_i, &ramcfg.energy.wh_e, val.f);
    if (tmr_cmp > 0)
    {
       TMR_SetCntrVal(CH1, 0);
       TMR_SetCmp1Val(CH1, TMRCMPVAL(tmr_cmp, TMR1CLK));
       TMR_SetCountMode(CH1, COUNT_POSEDGE);
    }
    /* calculates reactive energy increment and energy timer treshold value   */
    tmr_cmp = METERLIBFFT1PH_CalcVarHours(&ui, &ramcfg.energy.varh_i, &ramcfg.energy.varh_e, val.f);
    if (tmr_cmp > 0)
    {
       TMR_SetCntrVal(CH2, 0);
       TMR_SetCmp1Val(CH2, TMRCMPVAL(tmr_cmp, TMR2CLK));
       TMR_SetCountMode(CH2, COUNT_POSEDGE);
    }
}
```

- Process 2: LPTMR IRQ (@500 ms)

```
static void lptmr_callback (void)
{
   /* Mains frequency reading and averaging...                                */
   if (TMR_ReadClrCaptReg(CH0, &tmp16)==TRUE) freq_avrg += (TMR2FREQ(tmp16));
   if (++freq_cnt == FREQ_PERIOD)
   {
      val.f = freq_avrg/FREQ_PERIOD;                    /* averaged frequency */
      freq_cnt = freq_avg = 0;
      if (abs(val.f-last_f)) > F_CHANGE) {
         afe_chan_init();
         last_f = val.f;
      }
   }
   /* Other processing, such as showing values, etc. */
}
```

Freq. settings

AFE H/W triggering

XBAR

TMRx counting

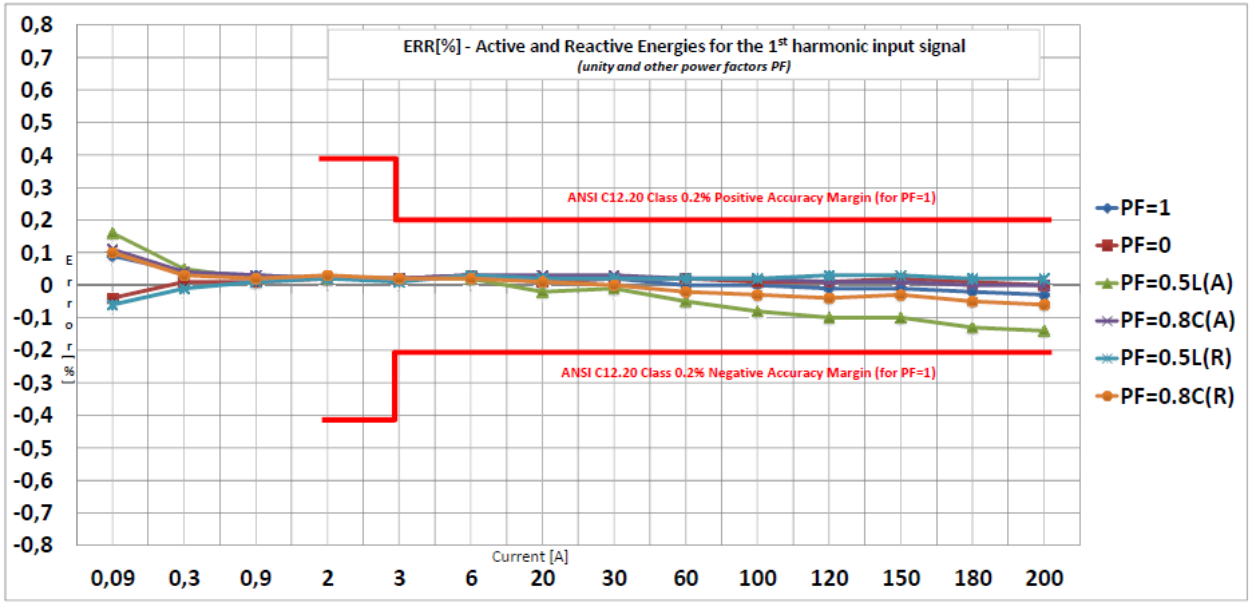
Measuring energy errors on real hardware

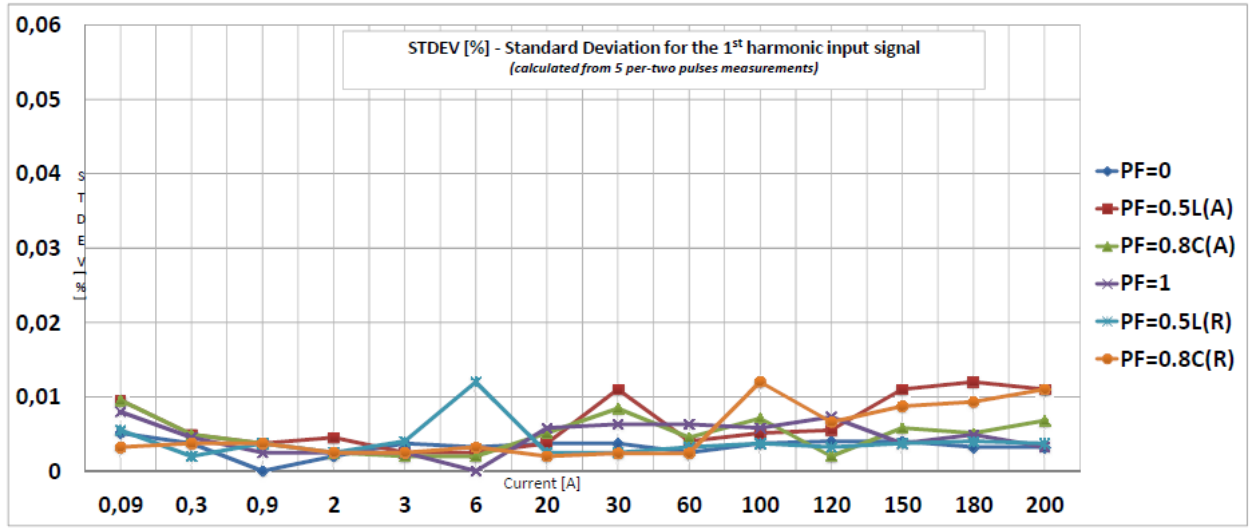
Typical basic harmonic measurement test report

Conditions:

- KM34Z128CLL5 MCU
- 120V/60Hz
- 30(200)A
- ANSI 2-ph meter (12S)
- 2000 imp/kWh/kVArh (Kh=0.5)
- Current Transformer (CTR 2000:1)
- FFT based metering algorithm

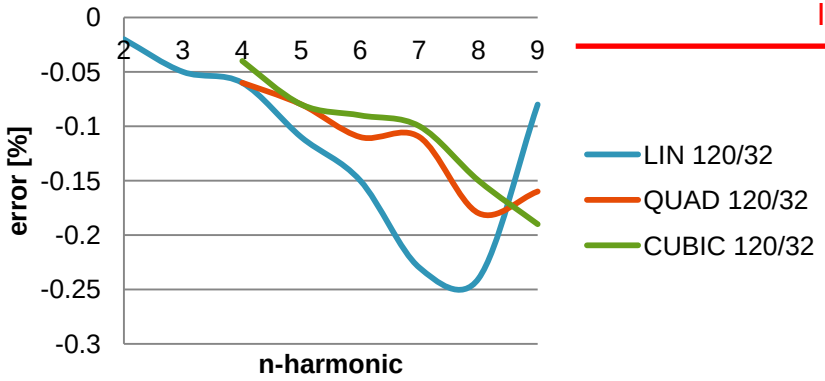
For more information see the Freescale's 2-ph reference design web page



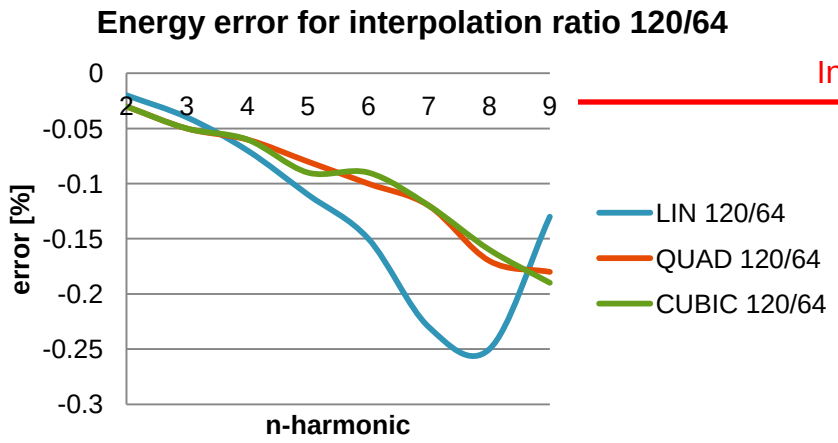


Higher harmonics measurement results

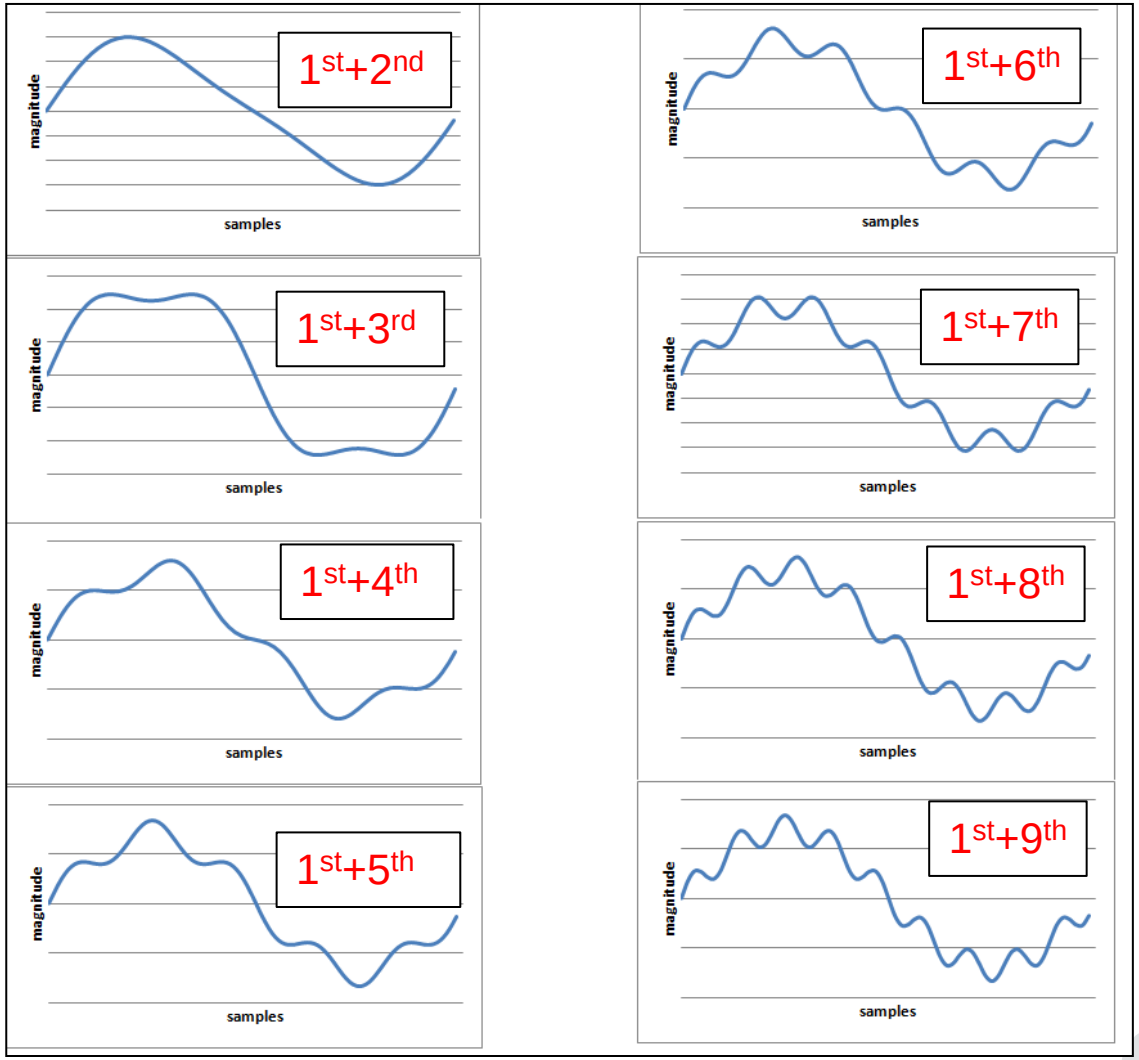
Conditions: 120V/100A/50Hz, 1-ph connection, 100% of the 1st harmonic + 20% of the nth harmonics (U and I)

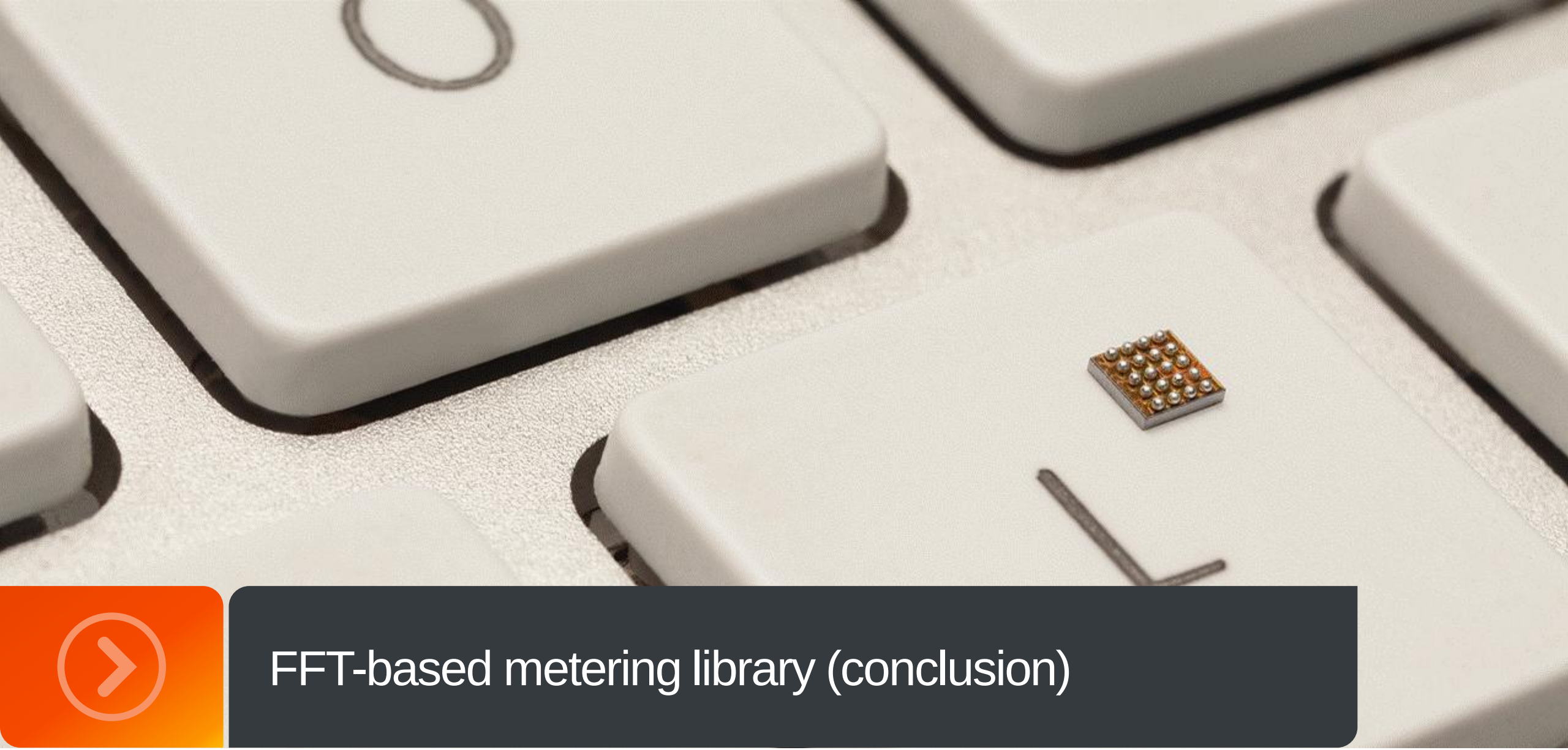


Input signals



Input signals





FFT-based metering library (conclusion)

FFT in metering applications

Advantages of using FFT:

- The same precision for both active and reactive energies (due to using one computing formula)
- Four-quadrant active and reactive energy measurement
- Frequency analysis of the input signal – ability to compute Total Harmonics Distortion (THD)
- Offset removal, because the zero-harmonic can be missing for power computing

Disadvantages of using FFT:

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

www.Freescale.com