AN10934

Using M3 DSP library filter functions

Rev. 01 — 14 May 2010 Application note

Document information

Info	Content
Keywords	M3, LPC1300, LPC1700, DSP, FIR, IIR, Biquad, Filter
Abstract	This application note and associated source code examples demonstrate how to use the filter functions contained within NXP's M3 DSP library.

Revision history

Rev	Date	Description
01	20100514	Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The NXP M3 DSP library contains a set of commonly used signal processing functions that have been designed and optimized for use with the NXP Cortex-M3 LPC1700 and LPC1300 family of products.

This application note describes, with the aid of software examples, how to use the filter functions contained within the library.

Note that the DSP Library supplied with application note AN10913[2] must be installed in order to build these software examples.

2. Filter examples

The filter example projects provided with this application note demonstrate how to use the FIR and IIR filter functions contained within the NXP M3 DSP Library. The projects are designed to take a waveform (represented as a table of samples) and execute a filter on this data to recover one of the original components. The filter results are output on the DAC allowing the user to observe the recovered waveform.

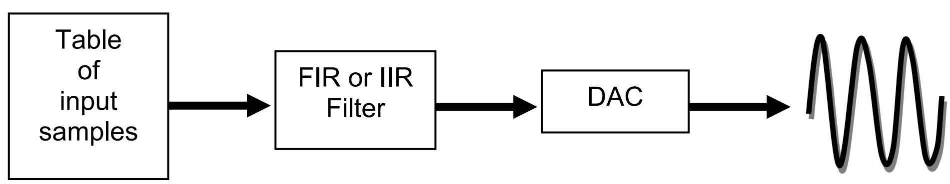

Fig 1. Filter example block diagram

This application note makes use of the LPC1700's memory-to-DAC DMA transfer functionality to output the filtered wave form. More information about this feature can be found in application note AN10917[3].

The filter input waveform is a combination of sine waves at three different frequencies: 50 Hz, 250 Hz and 500 Hz. The component sine waves can be seen in the top diagram of Fig 2. The resulting filter input waveform (an addition of the sine waves) can be seen in the bottom diagram. A table of input samples was created from this waveform, based on a sampling frequency of 10 kHz. This table can be found in the file waveform.c.

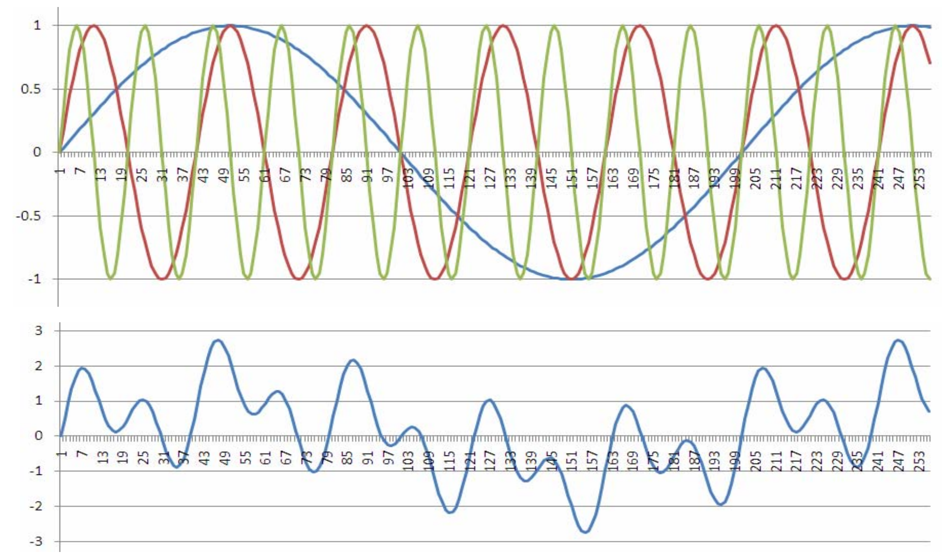

Fig 2. Filter input waveform

Software examples are provided for the following tool-chains and target hardware:

- Keil uVision 4.10 and a MCB1700 evaluation board.
- IAR Embedded Workbench 5.41 and a LPC1766-SK evaluation board.
- LPCXpresso 3.4 and a CodeRed RDB1768 evaluation board.

The DAC output on these boards can be monitored at the following locations:

- ***MCB1700 Evaluation Board:*** Pin 1 on the jumper labeled 'SPK'.
- ***LPC1766-SK Evaluation Board:*** Pin 1 on the switch labeled 'S2'.
- ***RDB1768v2 Evaluation Board:*** Pin 1 on the headphone jack.

3. FIR filter

The Finite Impulse Response (FIR) filter is one of the most commonly used digital filter functions. The output of such a filter is dependent only upon the input samples and the filter coefficients. It can be described as follows:

$$y[n] = b_0x[n] + b_1x[n-1] + \dots b_Nx[n-N]$$

$y[n]$ represents the nth output sample

$x[n]$ represents the nth input sample

b_i represents the filter coefficients

N represents the number of taps

Fig 3. FIR filter difference equation

Because the filter uses only multiplication and addition it can be very efficiently implemented on LPC1700 and LPC1300 devices by making use of the built-in hardware Multiply and Accumulate (MAC) unit.

FIR filters have a number of desirable properties making them suitable for a wide range of filtering applications:

- The lack of feedback (no previous output samples are used in the calculation of the current output sample) makes them inherently stable.
- They can be easily designed to ensure that the phase difference between input and output is proportional to frequency, i.e., the delay through the filter is equal at all frequencies.
- They are insensitive to coefficient quantization error. Because no output terms are used in the filter calculation, rounding errors that occur when coefficients are converted from floating to fixed-point format are not compounded over multiple iterations.

The response of a FIR filter (low-pass, high-pass, cut-off frequency, stop band attenuation, etc.) is determined by the number of taps and the values used for the coefficients. There are a number of tools available that allow calculation of FIR filter coefficients based upon a desired response. The coefficients used in the example were generated using a tool called WinFilter[5].

3.1 FIR parameter limits

Internally, the FIR filter function uses 32-bit multiplication and addition to generate a 32-bit result. This means that input values have to be scaled appropriately to avoid overflow, i.e., scaled down to 16-bit values. Any gain or attenuation introduced by the filter can be removed by appropriately adjusting the magnitude out the output samples.

To maximize performance the filter has been implemented using a ‘block-FIR’ algorithm. The algorithm reduces the number of memory accesses by computing several output samples in each loop iteration. In this way, the input data and the coefficients can be re-used multiple times before reading more data from memory.

Because the block FIR function computes several output samples in each loop iteration the number of input samples passed to the function, and the number of taps used, must be a multiple of 4.

3.2 FIR filter example

This example demonstrates how create a FIR filter that will recover the 50 Hz sine wave component of the input waveform. In order to recover this component a low-pass filter with the following characteristics was designed:

- Sampling Frequency – 10 kHz
- Cut-off Frequency – 100 Hz
- Number of Taps – 128 (the filter order plus one)

A PC-based application (WinFilter[5]) was used to generate the coefficients necessary to implement this filter using the `vF_dspl_blockfir32` function. The frequency response that can be expected when using these coefficients is illustrated in Fig 4.

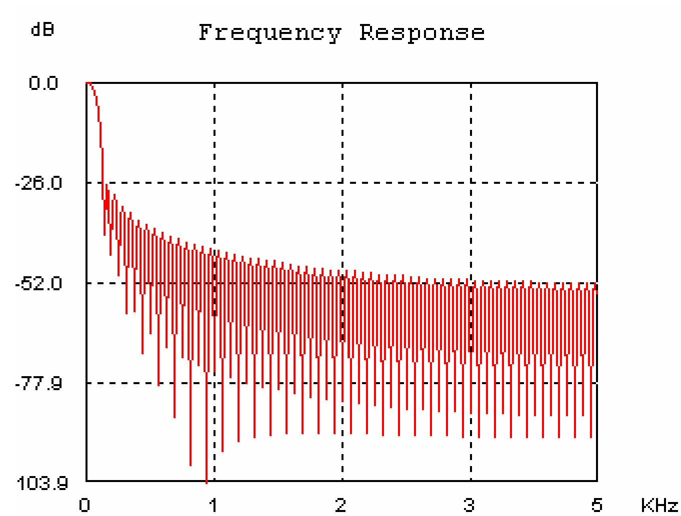

Fig 4. Low-pass FIR filter frequency response

The coefficients generated by the WinFilter application are fractional numbers in the range −1.0 to +1.0. They must be converted into 16-bit signed integers for use with the block FIR filter function. This process is achieved simply by multiplying them by 2^{15}. The example project contains a macro to perform this conversion, allowing fractional coefficients to be copied directly into the source file and then converted at build time.

The generated coefficients are stored in an array as follows (the SCALE_F2S16 macro is used to convert them in signed 16-bit integers):

```
int iFIR_Coeff[FIR_NB_TAPS] =
{
  SCALE_F2S16(-0.00248711688516133980),
  SCALE_F2S16(-0.00248994820340432470),
  ..
  ..
  SCALE_F2S16(-0.00247085376479650640)
};
```

Fig 5. FIR filter coefficient storage (fir_coeff.c)

The number of taps and a pointer to the filter coefficients are stored in a data structure that is passed (by reference) to the filter function every time it is called. This structure is initialized as follows:

```
tS_blockfir32_Coeff sFirCoeff =
{
  &iFIR_Coeff [0],
  FIR_NB_TAPS,
};
```

Fig 6. FIR filter parameter storage (fir_example.c)

When the FIR filter function is called, multiple input samples can be passed to it; when complete, it will return the same number of output samples.

```
vF_dspl_blockfir32((int *)&au32Output[0],
                   (int *)pu32WaveForm,
                   &sFirCoeff,
                   u16WaveForm_GetLen());
```

Fig 7. FIR filter function parameters (fir_example.c)

Both the original input wave form and the filtered version can be output on the DAC simply by changing the `FILTER_DATA` definition.

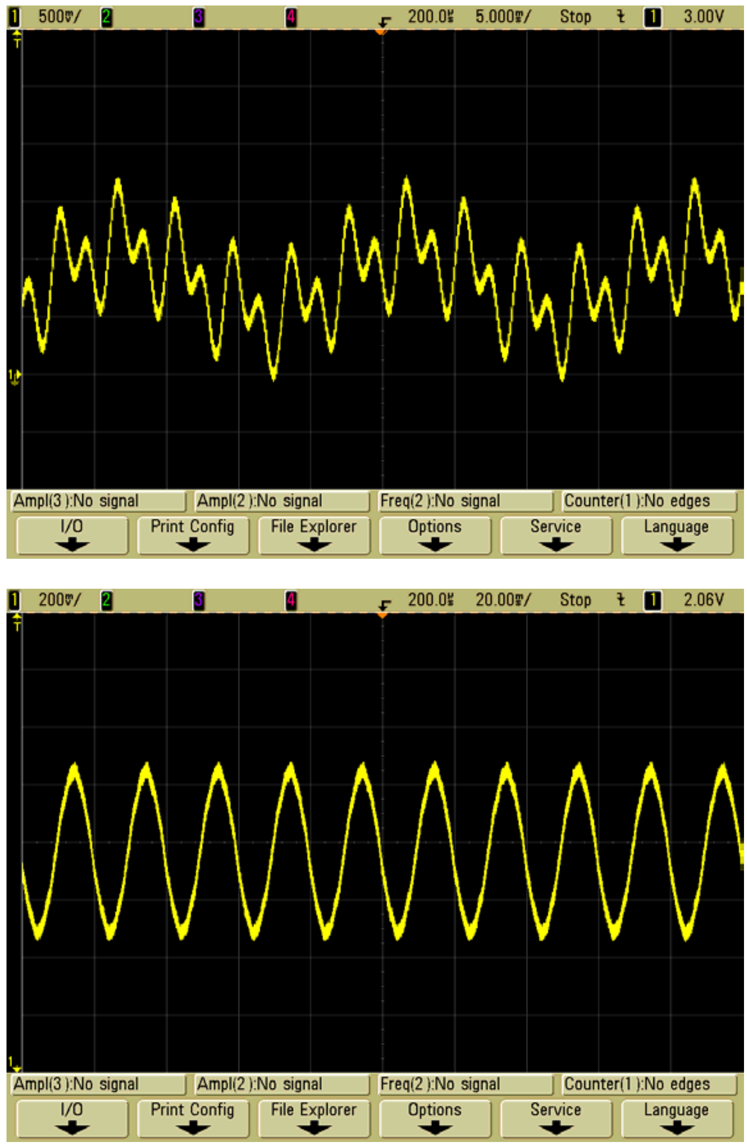

Fig 8. Low-pass FIR filter input/output

The output from the DAC as a result of running the example software is shown in Fig 8. The top waveform represents the filter input data (output by the DAC when `FILTER_DATA = 0`). The bottom waveform represents the filter output (generated by the DAC when `FILTER_DATA = 1`).

4. IIR filter

The Infinite Impulse Response (IIR) filter is another commonly used digital filter function. Unlike the FIR filter, an IIR filter uses feedback; its output is dependent upon the input samples, previous output values and filter coefficients.

An IIR filter can achieve the same response as a FIR filter with fewer operations and less memory usage. However, the use of feedback means that high order IIR filters can be sensitive to coefficient quantization (because rounding errors are compounded) and are more prone to instability. To overcome this problem high order filters are usually implemented as a number of cascaded second order IIR filters. The NXP DSP Library contains a second order (Biquad) IIR filter function (Direct Form II implementation) which can be described as follows:

$$y[n] = b_0v[n] + b_1v[n-1] + b_2v[n-2]$$

where

$$v[n] = x[n] - a_1v[n-1] - a_2v[n-2]$$

y[n] represents the nth output sample

x[n] represents the nth input sample

a_i represents the feedback filter coefficients

b_i represents the feedforward filter coefficients

Fig 9. IIR filter (direct form II implementation) difference equation

As with FIR filters, IIR filters use only multiplication and addition and therefore can be very efficiently implemented on LPC1700 and LPC1300 devices by making use of the built-in hardware MAC (multiply and accumulate) unit.

4.1 IIR filter example

This example demonstrates how to create an IIR filter that will recover the 50 Hz sine wave component of the input waveform. In order to recover this component a low-pass filter with the following characteristics was designed:

- Sampling Frequency – 10 kHz
- Cut-off Frequency – 100 Hz
- Order – 2

A PC-based application (WinFilter[5]) was used to generate the coefficients necessary to implement this filter using the `vF_dspl_biquad32` function. The frequency response that can be expected when using these coefficients is illustrated in Fig 10.

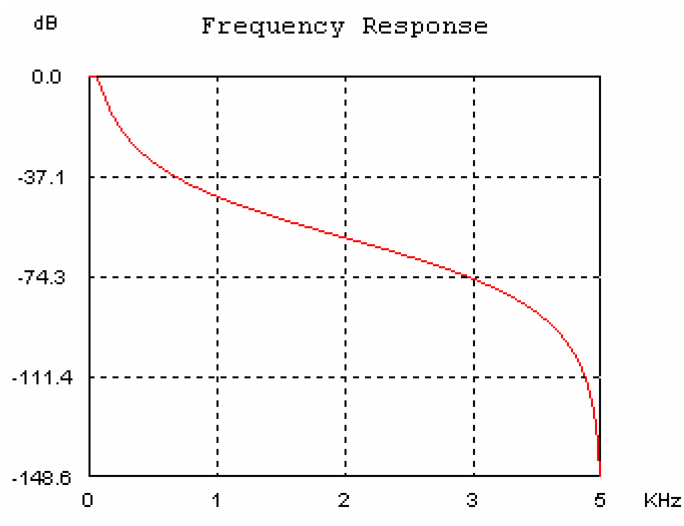

Fig 10. Low-pass IIR filter frequency response

The coefficients generated by this application are fractional numbers in the range –2.0 to +1.999999. They must be converted into 2.14 fixed-point numbers for use with the Biquad filter function; see section 5 for more details regarding this number format. The example project contains a macro to do this conversion, which will allow fractional coefficients to be copied directly into the source file and then converted at build time.

The generated coefficients are stored in an array as follows (the `QFORMAT 16` macro is used to convert them into 2.14 format signed 16-bit integers):

```
tS_biquad32_StateCoeff sIIR_Coeff =
{
  /* Fs = 10kHz Fc = 100Hz 2nd Order */
  QFORMAT_16(14,  1.91119706742607360000), /* psi_Coeff[0] - a1 */
  QFORMAT_16(14, -0.91497583480143418000), /* psi_Coeff[1] - a2 */
  QFORMAT_16(14,  0.00094547653094439164), /* psi_Coeff[2] - b0 */
  QFORMAT_16(14,  0.00189095306188878330), /* psi_Coeff[3] - b1 */
  QFORMAT_16(14,  0.00094547653094439164), /* psi_Coeff[4] - b2 */
  0,                                          /* psi_State[0] */
  0                                           /* psi_State[1] */
};
```

Fig 11. IIR filter coefficient storage (iir_coeff.c)

See section 4.2 for further information regarding the format of the a_1 and a_2 coefficients.

When the IIR filter function is called, multiple input samples can be passed to it; when complete, it will return the same number of output samples.

```
vF_dspl_biquad32(&aiOutput[0],
                 &aiInput[0],
                 &sIIR_Coeff,
                 u16WaveForm_GetLen());
```

Fig 12. IIR filter function parameters (iir_example.c)

Both the original input wave form and the filtered version can be output on the DAC by simply changing the `FILTER_DATA` definition.

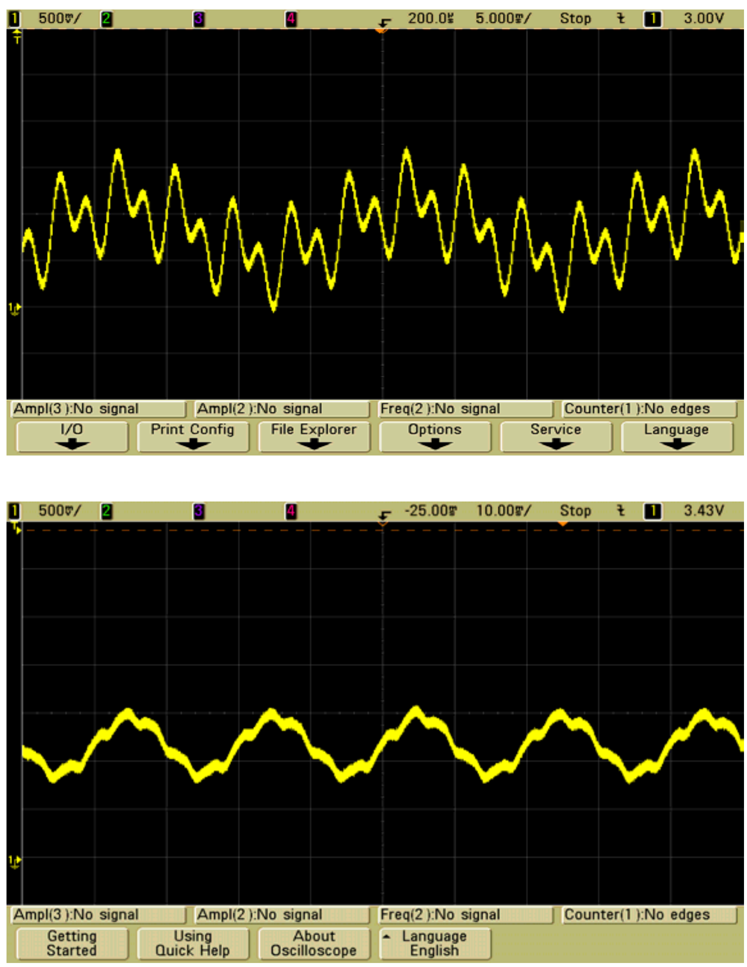

Fig 13. Low-pass IIR filter input/output

The output from the DAC as a result of running the example software, with (bottom) and without (top) filtering enabled, is shown in Fig 13.

4.2 IIR parameter limits

Internally, the IIR filter uses only multiplication and addition; therefore, the sign of the feedback (a_i) coefficients generated by WinFilter have to be reversed (if they are positive they have to be made negative and vice versa). The actual coefficients generated by the tool for the filter specified in section 4.2 are as follows:

a_1 = –1.91119706742607360000

a_2 = 0.91497583480143418000

However, when they are added to the IIR coefficient data structure, the sign is reversed so that they become:

a_1 = 1.91119706742607360000

a_2 = –0.91497583480143418000

As with the FIR filter, the IIR function uses 32-bit multiplication and addition to generate a 32-bit result. This means that input values have to be scaled appropriately to avoid overflow, i.e., scaled down to 16-bit values. Any gain or attenuation introduced by the filter can be removed by appropriately adjusting the magnitude out the output samples.

5. Fixed-point number representation

A fixed-point number format (often referred to as Q-Format) can be used to represent fractional numbers without having to use floating point data types.

A fixed-point number consists of three fields:

1. A 1-bit field representing the sign, an i-bit
2. 2's complement field representing the integer portion and an f-bit
3. 2's complement field representing the fractional part of the number

For example, a 16-bit data type can be used to represent a fractional number, where 1-bit represents the sign, 1-bit represents the integer and 14-bits represent the fraction, as follows:

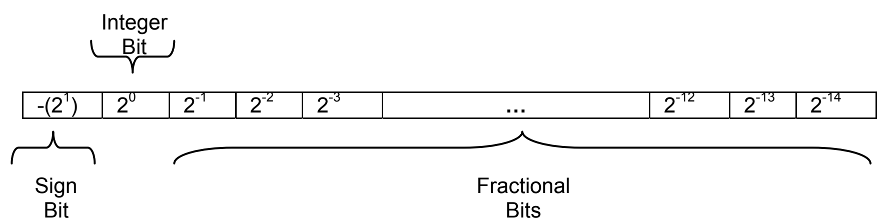

Fig 14. Fixed-point number example

The resolution is defined by the number of fractional bits. In this example there are 14 and therefore the resolution is 2^{-14} (0.00006103515625). The largest positive value that can be represented by this format is 1.999938964843750 (0x7FFF) and the largest negative value is –2.0 (0x8000).

The notation used to describe these numbers is Qi.f, e.g., a number containing 4 integer bits and 28 fractional bits (stored in a 32 bit data type) would be designated as a Q4.28 value.

5.1 Floating point to fixed-point conversion

Converting a floating point value into a Qi.f fixed-point representation can be achieved simply by multiplying the floating point number by 2^f and then rounding to the nearest integer. An example showing how to convert the floating point value 1.2345678 into a 2.14 fixed point representation is shown in Fig 15.

$$1.2345678 * 2^{14} = 20227.1588352 = \mathbf{20227\ (2.14\ Fixed\text{-}Point\ Format)}$$

Fig 15. Floating point to fixed-point conversion example

6. References

[1] Understanding Digital Signal Processing by Richard G. Lyons

[2] DSP library for LPC1700 and LPC1300 (AN10913)

[3] Memory to DAC transfers using the LPC1700’s DMA (AN10917)

[4] LPC17xx User Manual (UM10360)

[5] WinFilter Version 0.8 (http://www.nxp.com/redirect/winfilter.20m.com)

7. Legal information

7.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

7.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

8. Contents

Date of release: 14 May 2010

Document identifier: AN10934_1