AN10913

DSP library for LPC1700 and LPC1300

Rev. 3 — 11 June 2010 **Application note**

Document information

Info	Content
Keywords	LPC1700, LPC1300, DSP library
Abstract	This application note describes how to use the DSP library with the LPC1700 and LPC1300 products

Revision history

Rev	Date	Description
3	20100611	Updated Section 3.
2	20100401	Added performance tables throughout.
1	20100210	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 DSP library has been developed as a commonly used set of DSP functions optimized for the NXP Cortex-M3 LPC1700 and LPC1300 family products. Most functions have been implemented in Thumb-2 assembler unless there was little or no performance benefit in doing so.

The library is supplied as a static library project with source code in LPCXpresso (Code Red), Keil and IAR versions, and can also be linked into any ARM-EABI tool chain as a binary library.

1.1 DSP library functions

- Biquad filter
- Fast Fourier transform
- Dot product
- Vector manipulation
- FIR filter
- Resonator
- PID controller
- Random number generator

1.2 Convention used in function names and variable names

Each variable and function is prefixed with letters giving hints to the types. Some examples are shown below:

- vF: void Function
- iF: integer Function
- pi_x: a pointer to integer
- psi_x: a pointer to a signed integer
- si_x: a signed integer
- i_x: an integer
- pS_x: a pointer to a structure

1.3 Cortex-M3 for DSP

The Cortex-M3 has several attributes that make it deliver excellent DSP performance:

- 1-cycle 32x32 -> 32 signed multiplication
- 2-cycle (32x32)+32 -> 32 signed multiply accumulate
- Cortex-M3 is Harvard in having a separate data port to memory and instruction port to memory

Notes:

- Any load with base register update such as LDR <rt>,[<rn>],#<imm8> takes 2-cycles due to register bank write port conflict with the register write in the following instruction. The few exceptions are when it is followed by instructions that do not write results back to the register bank.

- In the library there are 16- and 32-bit variants of many algorithms. Because the Cortex-M3 is fundamentally a 32-bit architecture, there is often little or no performance improvement in the 16-bit implementation. Some other ARM cores have the ‘E’ DSP extensions or the V6 SIMD extensions that effectively allow the 32-bit registers to be used as a pair of 16-bit registers.
- One performance benefit from the 16-bit implementations could be the reduced data memory footprint of the coefficients and data for the algorithm and also the reduced memory system bandwidth which at the very least should save some power.
- The algorithms are all implemented using the native C types of ‘int’ and ‘short int’. A 32x32 multiply will overflow the 32-bit result if the inputs are not scaled appropriately by the user of the library.

2. Biquad filter

The biquad is a commonly used 2nd order filter section that can be cascaded to build any order of filter.

Biquad discrete-time function:

$$y(n) = b_0 \cdot x(n) + b_1 \cdot x(n-1) + b_2 \cdot x(n-2) + a_1 \cdot y(n-1) + a_2 \cdot y(n-2)$$

The Z-domain transfer function is:

$$H(z) = \frac{(b_o + b_1 \cdot Z^{-1} + b_2 \cdot Z^{-2})}{(1 - a_1 \cdot Z^{-1} - a_2 \cdot Z^{-2})}$$

The implementation is a Direct Form II (see Ref. [3]) which uses a shared 2-element state vector.

2.1 Function calling details

```
void vF_dspl_biquad32(int *pi_Output, int *pi_Input, tS_biquad32_StateCoeff
*pS_StateCoeff, int i_NSamples);


typedef struct
{
   short int psi_Coeff[5];
   short int psi_State[2];
}tS_biquad32_StateCoeff;
```

psi_Coeff are '2.14' format fractional values.

psi_State are '2.14' format fractional values that can be zero initialized for the first call but are updated by the routine to allow repeated calling of the filter with a stream of data.

pi_x and pi_y are '4.28' format fractional values.

2.2 Biquad filter performance

Table 1. Biquad filter

Biquad	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
32 samples	626	31.300	631	15.775	636	10.600

Biquad	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
32 samples	641	8.013	648	6.480	648	5.400

[1] 120 MHz only available on LPC1759 and LPC1769.

3. Fast Fourier transform

The Discrete Time Fourier Transform (DFT) is a commonly used transform in communications, audio signal processing, speech signal processing, instrumentation signal processing and image processing.

There are many algorithms to implement the DFT efficiently, but often the reality is that certain algorithms suit certain machine architectures. The ARM architecture in general, due to the register bank of 16, delivers the best FFT performance by using a Radix-4 transform and this is what has been implemented in this library.

3.1 DFT formula

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{kn}$$

where:

$$W_N = e^{-\frac{2\pi j}{N}}$$

3.2 DFT function prototype

```
void vF_dspl_fftR4b16N64(short int *psi_Y, short int *psi_x);
void vF_dspl_fftR4b16N256(short int *psi_Y, short int *psi_x);
void vF_dspl_fftR4b16N1024(short int *psi_Y, short int *psi_x);
void vF_dspl_fftR4b16N4096(short int *psi_Y, short int *psi_x);
```

3.3 FFT performance

Table 2. FFT performance (coefficients in flash memory)

FFT (coefficients in flash memory)	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (ms)	Cycles	Time (ms)	Cycles	Time (ms)
64 points	3895	0.195	4035	0.101	4202	0.070
256 points	21107	1.055	21719	0.543	22339	0.372
1024 points	107007	5.350	110161	2.754	113326	1.889
4096 points	518926	25.946	538209	13.455	557494	9.292

FFT (coefficients in Flash memory)	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (ms)	Cycles	Time (ms)	Cycles	Time (ms)
64 points	4384	0.055	4616	0.046	4616	0.038
256 points	22961	0.287	23884	0.239	23884	0.199
1024 points	116749	1.459	121657	1.217	121657	1.014
4096 points	578059	7.226	600694	6.007	600694	5.006

[1] 120 MHz only available on LPC1759 and LPC1769.

4. Dot product

This function implements a 32-bit dot-product (otherwise known as the scalar product) in assembler. The loop does one multiply per loop iteration for maximum vector length flexibility, but could be simply unrolled further for fixed lengths of vectors.

4.1 Dot product formula

$$z = \bar{y} \cdot \bar{x} = \sum_{i=0}^{N-1} x(i) \cdot y(i)$$

4.2 Dot product function prototype

```
int iF_dspl_dotproduct32(int *pi_x, int *pi_y, int i_VectorLen);
```

4.3 Dot product performance

N = Number of cycles

N = (8 * i_VectorLen) + 8

Assumes all instruction fetches are single cycle.

5. Vector add

5.1 VectAdd16

```
void vF_dspl_vectadd16(int *psi_z, int *psi_x, int *psi_y, int i_VectorLen);
```

5.2 VectAdd32

```
void vF_dspl_vectadd32(int *pi_z, int *pi_x, int *pi_y, int i_VectorLen);
```

5.3 Vector add performance

Table 3. Vector addition

Vector addition	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	340	17.000	343	8.575	346	5.767
32 bit	341	17.050	346	8.650	351	5.850

Vector addition	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	349	4.363	352	3.520	352	2.933
32 bit	357	4.463	363	3.630	363	3.025

[1] 120 MHz only available on LPC1759 and LPC1769.

6. Vector subtract

6.1 VectSub16

```
void vF_dspl_vectsub16(int *psi_z, int *psi_x, int *psi_y, int i_VectorLen);
```

6.2 VectSub32

```
void vF_dspl_vectsub32(int *pi_z, int *pi_x, int *pi_y, int i_VectorLen);
```

6.3 Vector Sub Performance

Table 4. Vector subtraction

Vector subtraction	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	309	15.450	313	7.825	317	5.283
32 bit	341	17.050	346	8.650	351	5.850

Vector subtraction	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	321	4.013	326	3.260	326	2.717
32 bit	358	4.475	365	3.650	365	3.042

[1] 120 MHz only available on LPC1759 and LPC1769.

7. Vector add constant

7.1 VectAddConst16

```
void vF_dspl_vectaddconst16(int *psi_y, int *psi_x, int si_c, int i_VectorLen);
```

7.2 VectAddConst32

```
void vF_dspl_vectaddconst32(int *pi_y, int *pi_x, int i_c, int i_VectorLen);
```

7.3 Vector Add Constant Performance

Table 5. Vector add constant

Vector add constant	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	274	13.700	278	6.950	282	4.700
32 bit	274	13.700	280	7.000	287	4.783

Vector add constant	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	287	3.588	292	2.920	292	2.433
32 bit	295	3.688	303	3.030	303	2.525

[1] 120 MHz only available on LPC1759 and LPC1769.

8. Vector element-by-element multiply

8.1 VectMulElement16

```
void vF_dspl_vectmulelement16(int *psi_z, int *psi_x, int *psi_y, int i_VectorLen);
```

8.2 VectMulElement32

```
void vF_dspl_vectmulelement32(int *pi_z, int *pi_x, int *pi_y, int i_VectorLen);
```

8.3 Vector Element-by-Element Multiply Performance

Table 6. Vector multiply

Vector multiply	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	277	13.850	280	7.000	283	4.717
32 bit	309	15.450	312	7.800	315	5.250

Vector multiply	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	286	3.575	290	2.900	290	2.417
32 bit	320	4.000	325	3.250	325	2.708

[1] 120 MHz only available on LPC1759 and LPC1769.

9. Vector multiply by constant

Multiply each vector element by a constant.

9.1 Vector Multiply by Constant formula

$$y = \bar{x} \cdot c$$

9.2 VectMulConst16 function prototype

```
void vF_dspl_vectmulconst16(short int *psi_y, short int *psi_x, short int si_c, int
i_VectorLen);
```

9.3 VectMulConst32 function prototype

```
void vF_dspl_vectmulconst32(int *pi_y, int *pi_x, int i_c, int i_VectorLen);
```

9.4 Vector Multiply by Constant Performance

Table 7. Vector multiply constant

Vector multiply constant	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	243	12.150	247	6.175	252	4.200
32 bit	274	13.700	277	6.925	280	4.667

Vector multiply constant	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
16 bit	257	3.213	264	2.640	264	2.200
32 bit	283	3.538	286	2.860	286	2.383

[1] 120 MHz only available on LPC1759 and LPC1769.

10. Vector sum of squares

10.1 Vector sum of squares formula

$$y = \sum_{i=0}^{N-1} x_i^2$$

10.2 VectSumSquares16 function prototype

```
int iF_dspl_vectsumofsquares16(short int *psi_x, int i_VectorLen);
```

10.3 VectSumSquares32 function prototype

```
int iF_dspl_vectsumofsquares32(int *pi_x, int i_VectorLen);
```

10.4 Vector Sum of Squares Performance

Table 8. Vector sum of squares

Vector sum of squares	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (µs)	Cycles	Time (µs)	Cycles	Time (µs)
16 bit	242	12.100	244	6.100	247	4.117
32 bit	242	12.100	245	6.125	249	4.150

Vector sum of squares	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (µs)	Cycles	Time (µs)	Cycles	Time (µs)
16 bit	250	3.125	254	2.540	254	2.117
32 bit	254	3.175	259	2.590	259	2.158

[1] 120 MHz only available on LPC1759 and LPC1769.

11. FIR filter

Unlike DSP processors, the Cortex-M3 cannot perform load operations in parallel with ALU operations, so each data load cycle is a cycle that cannot be used for performing filter arithmetic. FIR filters are basically a long sequence of multiply-accumulate operations with the output sample being produced by the accumulation of many coefficient by input-sample multiplies.

To maximize FIR filter performance on the Cortex-M3, we utilize what is known as 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 some more from memory.

11.1 FIR filter formula

$$y(n) = \sum_{i=0}^{N-1} x(n-i) \cdot h(i)$$

11.2 FIR32 calling details

```
typedef struct
{
   int *pi_Coeff;
   int NTaps;
}tS_blockfir32_Coeff;


void vF_dspl_blockfir32(int *pi_y, int *pi_x, tS_blockfir32_Coeff *pS_Coeff, int
i_nsamples);
```

Note that the number of sample ‘i_nsamples’ must be a multiple of 4.

11.3 FIR filter performance

Table 9. FIR filter

	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
32 samples and taps	3433	171.650	3445	86.125	3470	57.833

	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
32 samples and taps	3495	43.688	3520	35.200	3520	29.333

[1] 120 MHz only available on LPC1759 and LPC1769.

12. Resonator (oscillator)

The resonator function is used to very efficiently generate sinusoidal signal – i.e., no look up table or use of trigonometric approximations. Note that this algorithm is just a special case of the biquad filter section but with the numerator coefficients equal to zero and the two poles on the unit-circle so that it oscillates.

12.1 Resonator formula

12.1.1 Discrete time representation

$$y(n) = a_1 \cdot y(n-1) + a_2 \cdot y(n-2)$$

12.1.2 Z-domain representation

$$H(z) = \frac{1}{(1 - a_1 \cdot Z^{-1} - a_2 \cdot Z^{-2})}$$

12.2 Resonator calling details

```
typedef struct
{
   int i_Coeff_a1;
   int i_yn_1;
   int i_yn_2;
}tS_ResonatorStateCoeff;
void vF_dspl_resonator(int *psi_Output, void *pS_ResonatorStateCoeff, int i_NSamples);
```

Since the resonator is a recursive algorithm, care needs to be taken with the parameter scaling that is used. The coefficients and state of the resonator needs setting up as follows:

```
i_Coeff_a1 = 2.0 * cos(Omega) * pow(2.0,14)
```

To start the oscillation, the initial state should be set as follows:

```
i_yn_1 = 0;
i_yn_2 = -Amplitude * sin(Omega) * pow(2.0,14)
```

where:

Omega = frequency as a fraction of the sample rate

Amplitude = required amplitude of the wave – must be <2.0 due to the ‘2.14’ arithmetic

A numerical format of ‘2.14’ has been used because the a1 coefficient is larger than 1 and the single cycle multiply of the CM3 can only be guaranteed not to overflow if the multiplier and multiplicand inputs to the multiplier are 16-bits.

12.3 Resonator performance

Table 10. Resonator

Resonator	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (µs)	Cycles	Time (µs)	Cycles	Time (µs)
512 samples	5153	257.650	5157	128.925	5161	86.017

Resonator	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (µs)	Cycles	Time (µs)	Cycles	Time (µs)
512 samples	5166	64.575	5172	51.720	5172	43.100

[1] 120 MHz only available on LPC1759 and LPC1769.

13. PID controller

The ‘Proportional, Integral, Differential’ is a commonly used feedback control algorithm with very modest CPU usage.

13.1 PID controller discrete time formula

$$u(n) = K_p \cdot e(n) + K_i \cdot \sum_{k=0}^{n} e(k) + K_d \cdot (e(n) - e(n-1))$$

13.2 PID controller function calling details

```
typedef struct
{
   short int Kp;
   short int Ki;
   short int Kd;
   short int IntegratedError;
   short int LastError;
}tS_pid_Coeff;


short int vF_dspl_pid(short int si_Error, tS_pid_Coeff *pS_Coeff);
```

13.3 PID controller performance

Table 11. PID controller

	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
PID	47	2.350	49	1.225	52	0.867

	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
PID	56	0.700	60	0.600	60	0.500

[1] 120 MHz only available on LPC1759 and LPC1769.

14. Random number generator

The library implements an assembler version of a Linear Congruential random sequence generator best described in Ref. [1].

Note that if you only want less bits than the 32-bits returned by this function it is better to choose the upper bits of the word returned as these are 'more random' than the lower bits.

14.1 Random number formula

$$Y_n = (a \cdot X_n + c) \bmod m$$

Where:

Yn is the new number in the output sequence

Xn is a seed value (or the previous value in a sequence)

c is a well chosen constant – see Ref. [1]

a is a well chosen multiplier constant – Ref. [1]

m is a carefully chosen modulus for the arithmetic – Ref. [1]

In our implementation m=2^32 so that we simply use the native arithmetic.

c = 32767

a = 16644525

14.2 Random number function prototype

```
int iF_RandomNumber(int i_Seed);
```

Note: To produce a sequence of random numbers, use the previous result as the seed for the next call.

14.3 Random number performance

Table 12. Random number generator

	Flash access 1 CPU clocks (20 MHz max)		Flash access 2 CPU clocks (40 MHz max)		Flash access 3 CPU clocks (60 MHz max)	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
Random number	23	1.150	26	0.650	30	0.500

	Flash access 4 CPU clocks (80 MHz max)		Flash access 5 CPU clocks (100 MHz max)		Flash access 5 CPU clocks (120 MHz max)[1]	
	Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
Random number	34	0.425	38	0.380	38	0.317

[1] 120 MHz only available on LPC1759 and LPC1769.

15. References

[1] Knuth, The Art of Computer Programming Vol. 2, Semi-numerical Algorithms, Chapter 3 – Random Numbers

[2] Rabiner and Gold, Theory & Application of Digital Signal Processing

[3] Proakis and Manolakis, Digital Signal Processing, Principles, Algorithms, and Applications

16. Legal information

16.1 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.

16.2 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.

16.3 Trademarks

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

17. Contents

Date of release: 11 June 2010
Document identifier: AN10913