# AN10995

## LPC1100 secondary bootloader

Rev. 1 — 13 October 2010 Application note

**Document information**

| Info | Content |
|---|---|
| **Keywords** | M0, LPC1100, IAP, Bootloader, XMODEM-1K |
| **Abstract** | This application note and associated example source code demonstrates how to create a secondary bootloader for a LPC1100 Microcontroller. |

**Revision history**

| Rev | Date | Description |
|---|---|---|
| 1 | 20101013 | 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

All devices within the LPC1100 family of Microcontrollers contain on-chip flash memory for the storage of application code and data. There are a number of mechanisms built into these devices that allow the contents of this memory to be updated in the field:

- In-System Programming (ISP) - Allows reprogramming of the on-chip flash memory using the devices primary bootloader and UART0. The primary bootloader is firmware that resides in the Microcontroller's ROM memory and is executed at power-up or when the device is reset.
- In-Application Programming (IAP) - Functions contained within the ROM allow the on-chip flash memory to be programmed or erased under control of the user's application.
- Serial Wire Debug (SWD) – The on-chip flash can be reprogrammed using the 2-pin debug interface.

In addition to these mechanisms a secondary bootloader can be created. This is a piece of software, residing in on-chip flash, which allows new application code to be downloaded using interfaces other than ISP/UART0 or SWD.

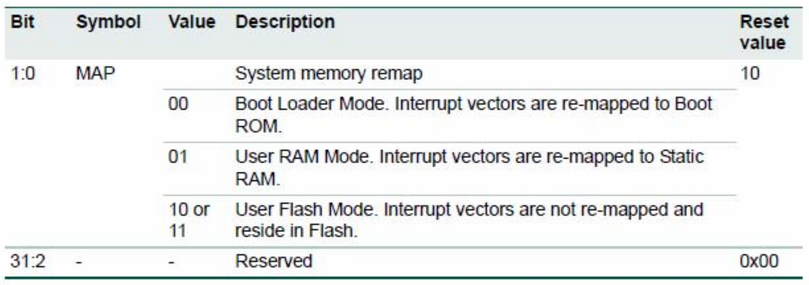
This application note describes how to create a secondary bootloader for the LPC1100. An example is provided that uses UART0 and the XMODEM-1K protocol to download new application firmware. Note that the software has been designed so that it is easily modified to use a different interface/protocol.

## 2. Background

This section presents background information about the LPC1100 that is relevant to the operation of a secondary bootloader.

### 2.1 Startup Sequence

Before a secondary bootloader can be designed the startup sequence of the ARM Cortex-M0 based LPC1100 must be understood. Following a power-on or reset the Cortex-M0 processor begins execution at address 0 in the memory map. Note that the LPC1100 is able to remap the memory that is located at address 0. This functionality is controlled by the SYSMEMREMAP register, see Fig 1.



| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 1:0 | MAP | | System memory remap | 10 |
| | | 00 | Boot Loader Mode. Interrupt vectors are re-mapped to Boot ROM. | |
| | | 01 | User RAM Mode. Interrupt vectors are re-mapped to Static RAM. | |
| | | 10 or 11 | User Flash Mode. Interrupt vectors are not re-mapped and reside in Flash. | |
| 31:2 | - | - | Reserved | 0x00 |

**Fig 1. System remap register bit description (SYSMEMREMAP, address 0x4004 8000)**

Initially the interrupt vectors are mapped to the Boot ROM causing the primary bootloader to be executed. Once this is complete the primary bootloader maps the interrupt vectors to on-chip flash and execution of the application begins.

The Cortex-M0 core expects the code located at the start address to be organized as illustrated in Fig 2. The first location contains the address of the top of the stack; this value is loaded into the stack pointer register by the processor. The next location contains the address of the reset handler; this is the address from which the processor will start executing code. Note that LSB of this address must be set to 1, indicating to the processor that this is Thumb code.

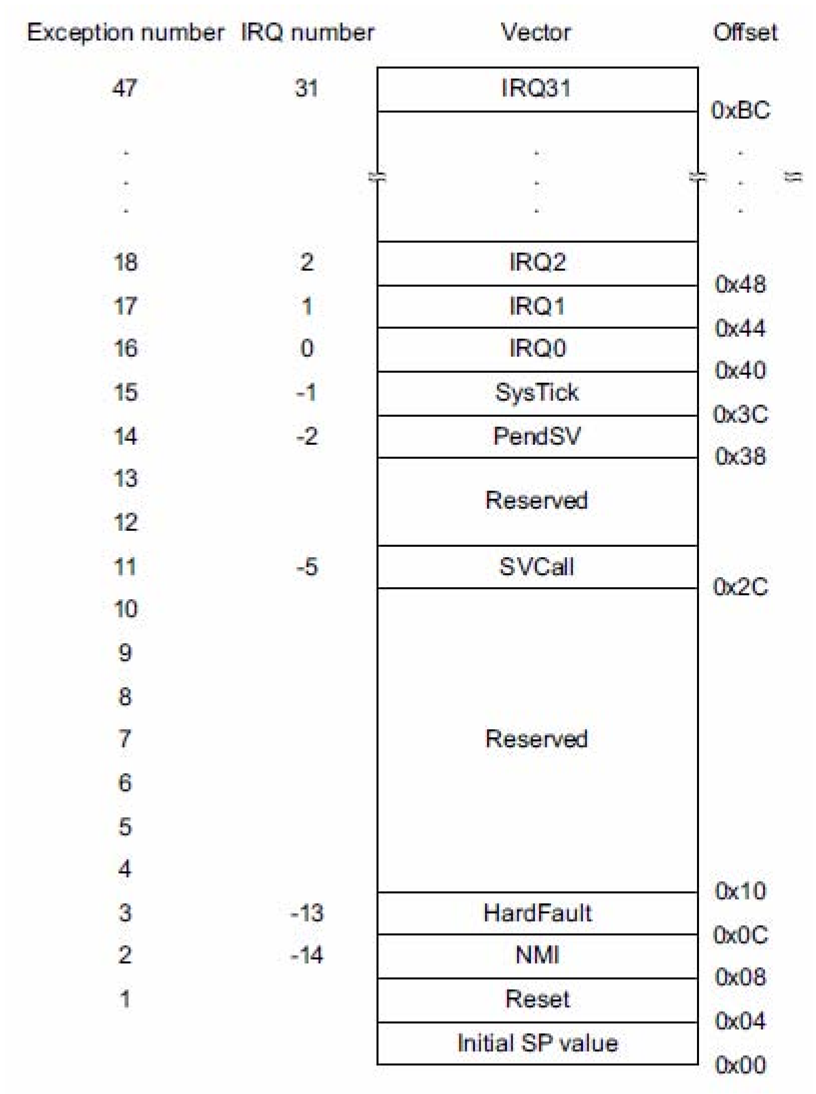


**Fig 2. Interrupt vector table**

### 2.2 Handling interrupts

The LPC1100 contains a NVIC (Nested Vectored Interrupt Controller) that handles all interrupts. When an interrupt occurs the processor uses the vector table to locate the address of the handler. The organization of the vector table can be seen in Fig 2.

### 2.3 In-Application Programming (IAP)

The boot ROM contains a number of routines that allow the on-chip flash memory to be programmed and erased. These routines are called In-Application Programming (IAP) functions and are provided so that user application code can erase and write to the on-chip flash memory. Note that flash memory is not accessible during a write or erase operation. Therefore IAP commands, which result in a flash write/erase operation, use 32 bytes of space in the top portion of the on-chip RAM for execution. The user application should not use this space if IAP flash programming is used. Further details regarding IAP can be found in the LPC1100 User Manual [2].

## 3. Secondary boot loader design

The LPC1100 on-chip flash memory is divided into 4 kB sectors. These sectors represent the minimum amount of memory that can be individually erased. The secondary bootloader occupies the first sector of flash memory, starting at address 0. The remaining sectors are available for the storage of the application code and data. The location of the secondary bootloader means that, following a reset or power-on, it is executed first (before the application). The secondary bootloader operation is summarized in Fig 3.

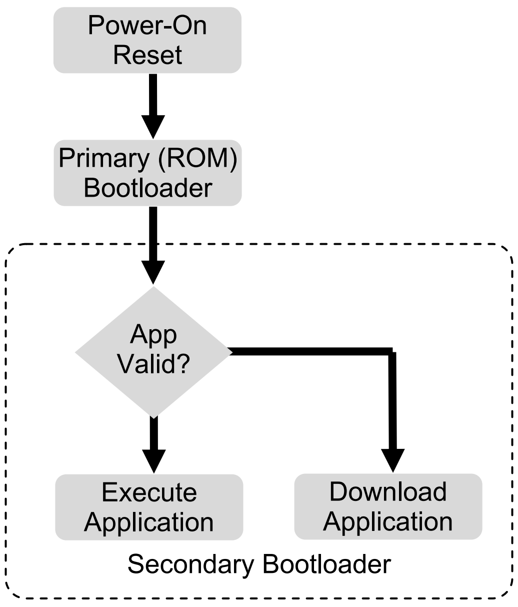


**Fig 3. Secondary bootloader flow chart**

Upon startup the secondary bootloader checks if there is a valid application stored in the flash memory, and if so this application is executed. Section 0 contains details of this integrity check. If no application is found, then the secondary bootloader attempts to download one using UART0 and the XMODEM-1K protocol.

## 3.1 Handling interrupts

When an application is developed, the contents of the interrupt vector table are created at build time. The vector table is then placed in memory at a fixed location (known to the processor) when the application is programmed into the flash.

On the LPC1100 the vector table is located in the same area of flash memory as the secondary bootloader. The secondary bootloader is designed to be permanently resident in flash memory and therefore it is not possible to update the contents of the vector table every time a new application is downloaded.

The Cortex-M3 core allows the vector table to be remapped; however this is not the case with the Cortex-M0. Because of this, the secondary bootloader has been designed to redirect the processor to the handler listed in a vector table located in the application area of flash memory, see Fig 4.

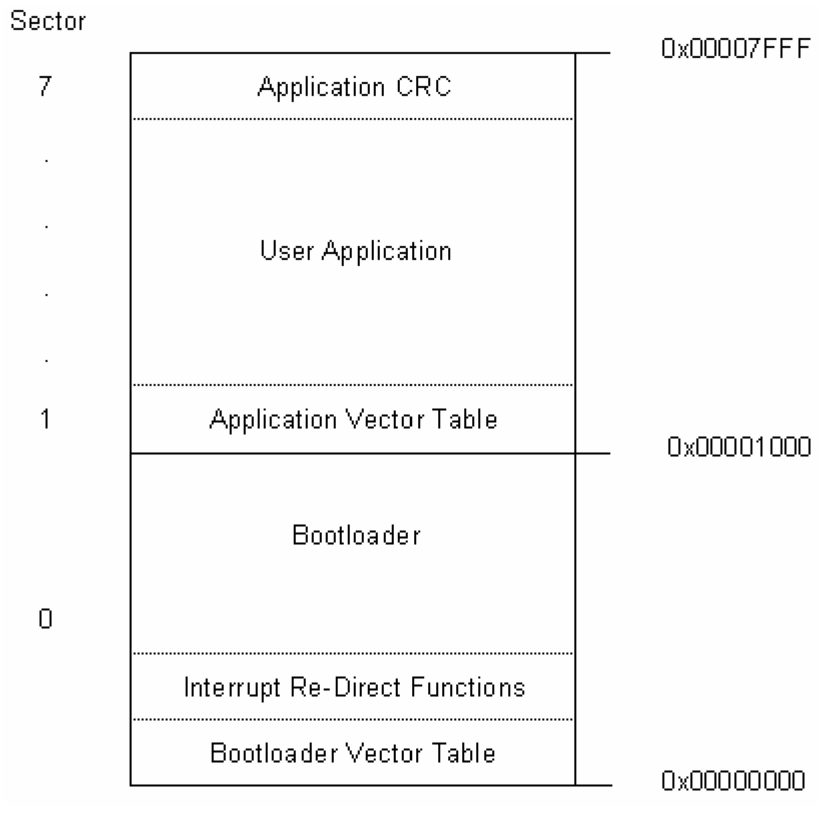


Fig 4. Interrupt Redirection

When an interrupt occurs the CPU gets the address of the interrupt handler from the vector table located in sector 0 (this is the bootloader vector table shown in 0). The addresses contained in this table point to a series of re-direct functions. These functions obtain the address of the interrupt handler from the vector table located in the application area (the Application Vector Table shown in Fig 4). An example of one of these functions is shown in Fig 5.

```
void SysTick_Handler(void)
{
    /* Re-direct interrupt, get handler address from
       application vector table */
    asm volatile("ldr r0, =0x103C");
    asm volatile("ldr r0, [r0]");
    asm volatile("mov pc, r0");
}
```

**Fig 5. Interrupt re-direct function**

When an interrupt occurs, the handler pointed to by the relevant entry in the application vector table is executed.

## 3.2 Application integrity check

Following a reset the secondary bootloader checks if there is a valid image contained within the application flash sectors. It does this by generating a 16-bit CRC of the application flash sectors and comparing it to the value stored in the last 2 bytes of flash memory. If the two values match then the secondary bootloader executes the application, if not the process of downloading a new application is started.

## 3.3 XMODEM-1K protocol

XMODEM-1K is a simple file transfer protocol based on the original XMODEM protocol. The primary difference between the two is that the 1K variant supports both 128 and 1024 byte packets (the original XMODEM protocol supported only 128 byte packets).

The secondary bootloader acts as a client and the equipment to which it is connected (usually a PC) acts as the server. The client initiates a transfer by sending the ASCII character ‘C’ to the server. Once the initialization phase is complete the server transfers the file data to the client one packet at a time. Upon reception the client checks the packet CRC and sends an ACK or NACK back to the server. This process continues until the entire file has been transferred.

# 4. Application firmware development

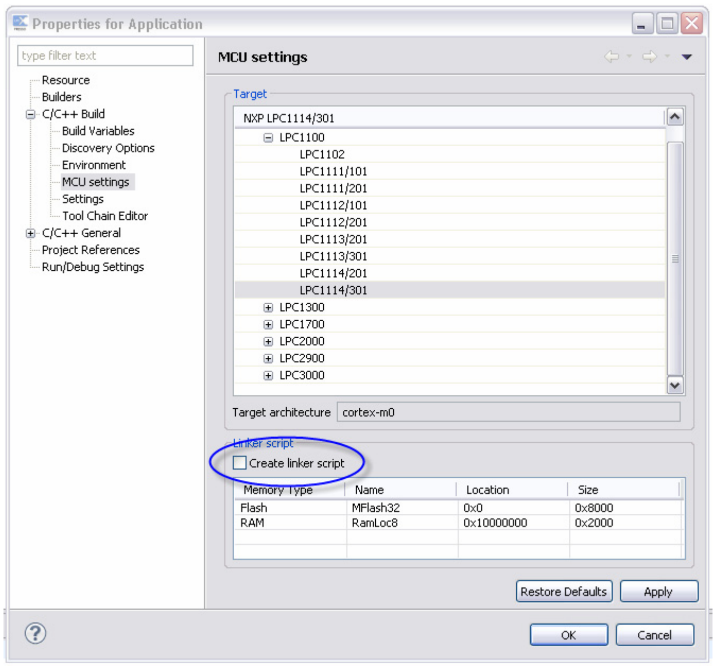
When building an application that will reside in flash memory alongside the secondary bootloader, the linker settings must be modified so that it is placed at the correct location. The following section details how this can be achieved.

## 4.1 LPCXpresso application development

The most straightforward way to modify the linker settings when developing an application using LPCXpresso is to replace the following files with those from the example application:

LinkerFiles\Application\Release\application_Release_mem.ld
LinkerFiles\Application\Debug\application_Debug_mem.ld

The project settings then need to be altered so that these linker files are used instead of those generated by the toolchain. To do this open the application project using LPCXpresso, right click on the project and select properties. Then ensure that the “MCU Settings” and “Tool Settings” match those shown in Fig 6 and Fig 7.



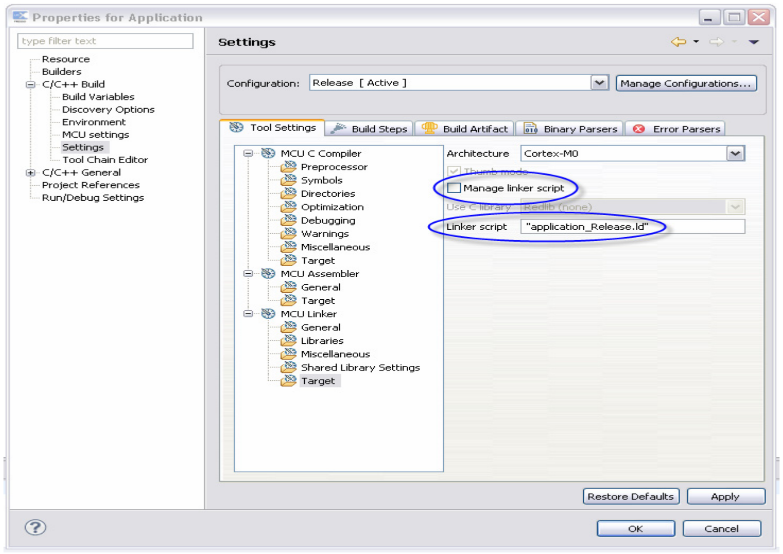
Target
NXP LPC1114/301
LPC1100
LPC1102
LPC1111/101
LPC1111/201
LPC1112/101
LPC1112/201
LPC1113/201
LPC1113/301
LPC1114/201
LPC1114/301
LPC1300
LPC1700
LPC2000
LPC2900
LPC3000

Target architecture cortex-m0

Linker script
☐ Create linker script

| Memory Type | Name | Location | Size |
|---|---|---|---|
| Flash | MFlash32 | 0x0 | 0x8000 |
| RAM | RamLoc8 | 0x10000000 | 0x2000 |

**Fig 6. LPCXpresso linker script control**



Configuration: Release [ Active ]  Manage Configurations...

Architecture: Cortex-M0

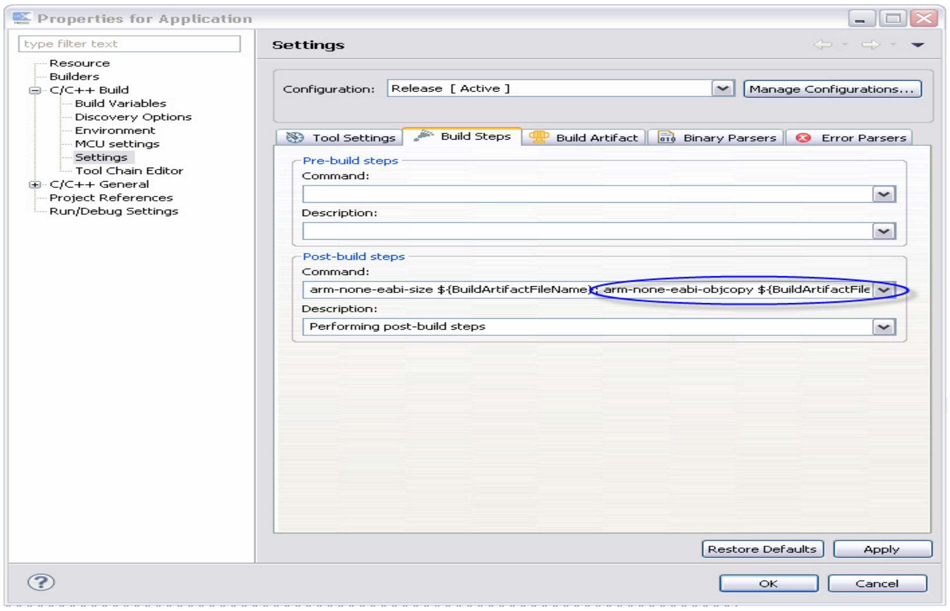
Manage linker script [ ]

Linker script: "application_Release.ld"

Fig 7. LPCXpresso linker script selection

In addition the application project settings should be modified to ensure that a binary file is created. To do this simple add the following command to the post build steps shown in Fig 8:

arm-none-eabi-objcopy -O binary ${BuildArtifactFileName} ${BuildArtifactFileBaseName}.bin;



Configuration: Release [ Active ]  Manage Configurations...

Pre-build steps
Command:
Description:

Post-build steps
Command:
arm-none-eabi-size ${BuildArtifactFileName}; arm-none-eabi-objcopy ${BuildArtifactFile
Description:
Performing post-build steps

Fig 8. LPCXpresso post build steps

## 5. LPCXpresso demonstration

The software accompanying this application note consists of a secondary bootloader and an example application. The following set of steps describes how to use this software to demonstrate the operation of the secondary bootloader (note that this demonstration requires the use of an LPCXpresso target board and an Embedded Artists base board):

1. Build the release target of both the Bootloader and Application projects using the LPCXpresso toolchain. This should result in the creation of the following binary files:
   \Application\Release\Application.bin
   \Bootloader\Release\Bootloader.bin
2. Program the bootloader binary file into the target hardware using a tool such as Flash Magic. Alternatively the object file (Bootloader.axf) maybe programmed into the hardware using the LPCXpresso toolchain.
3. Connect the Base Board to a PC via the mini-USB connector X3. Start a terminal emulator application on the PC that is capable of communication using the XMODEM-1K protocol (HyperTerminal is used in this example). The terminal emulator should be configured to communicate at 9600-8-N-1.
4. Reset the target hardware and it should then start periodically transmitting the ‘C’ character to the PC – indicating it is ready to begin a transfer using the XMODEM-1K protocol. Note that when using HyperTerminal the board should be reset before it is connected to the PC – otherwise it may enter ISP mode.
5. Now the application binary file can be transmitted to the target hardware, where the secondary bootloader will program it into flash. To start this process use HyperTerminal to send the file Application.bin using the XMODEM-1K protocol.
6. Once the transfer is complete the target hardware should be reset and LED2 (on the LPCXpresso board) will begin to flash (if using HyperTerminal remember to disconnect before pressing the base board reset button – otherwise the LPC1100 will enter ISP mode).

The application code can initiate a new download simply by invalidating the CRC that is stored in the last location of flash memory. This can be done by writing all zero’s to this location using the IAP calls and then re-invoking the secondary bootloader by resetting the device. The example application contains a function that implements this when pin P0.1 is taken low (this is connected to SW3 on the Embedded Artists base board). Pressing this button when the application is running (LED2 is flashing) will re-invoke the secondary bootloader and allow a new application to be downloaded.

## 6. References

[1] Cortex-M0 Devices Generic User Guide DUI0497A, ARM Limited

[2] LPC111x User Manual UM10398, NXP Semiconductors

## 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.

### 7.3 Trademarks

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

## 8. Contents

**Date of release: 13 October 2010**

**Document identifier: AN10995**