AN10895

LPC3143/LPC3154 secure boot process

Rev. 01 — 5 January 2010

Application note

Document information

Info	Content
Keywords	LPC3143, LPC3154, Secure boot, OTP, AES
Abstract	This application note describes the secure boot process implemented by boot ROMs on LPC3143 & LPC3154 secure parts.

Revision history

Rev	Date	Description
01	20100105	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

Both LPC3143 and LPC3154 chips have a hardware based 128-bit AES-CBC decryption engine and support secure booting. This application note describes the following features associated with the secure boot process, secure image creation and execution, and secure data on LPC3143 and LPC3154.

- Secure boot implemented by LPC3143/LPC3154 boot ROM
- On-chip 128-bit AES-CBC decryption engine
- On-chip One Time Programmable (OTP) fuses
- Image and data encryption tools provided by NXP
- Development process for products using LPC3143/LPC3154
- Production line process for products using LPC3143/LPC3154

2. Secure boot process

LPC3143/LPC3154 top level boot process is illustrated in Fig 1 "Main boot flow". The boot ROM reads the OTP poly-fuses into the data register as described in the OTP chapter in LPC3143/LPC3154 user manual. Based on the values of the security fuses the JTAG access to the chip can be enabled. By default the JTAG access to the chip is disabled at reset. As shown in the picture, the boot ROM determines the boot mode based on the reset state of the pins GPIO0, GPIO1 and GPIO2. The boot ROM indicates any error during boot process by toggling the GPIO2 pin; hence it is advised to connect this pin to an LED to get visual feedback. The boot ROM copies/downloads up to a 128 KB image to internal SRAM at location 0x11029000 and jumps to that location (sets ARM's program counter register to 0x11029000) after image verification. Hence the images for LPC3143/LPC3154 should be compiled with entry point at 0x11029000. On LPC3143/LPC3154 the image and header are validated using a 160-bit SHA1 hash algorithm.

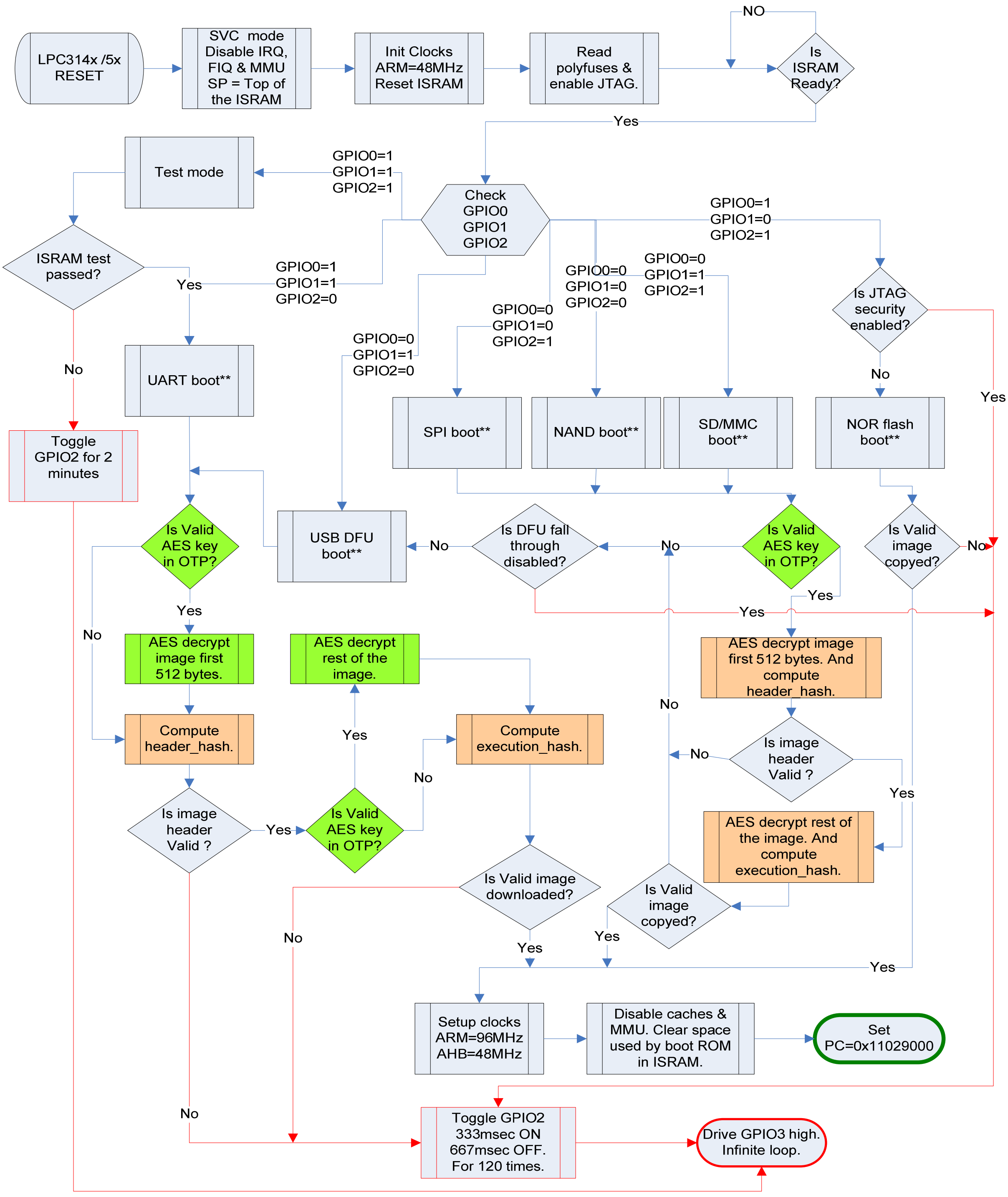
As part of image header validation, the following checks are done:

- 160 bit SHA1 hash digests of header (bytes 0x00 to 0x6C of the image) should match the value present in '`header_hash`' field (offset 0x6C) of the image header.
- The '`magic`' field (offset 0x04) in the header should have the value 0x41676D69.
- The '`image_type`' field (offset 0x1C) should match the boot interface specified by mode pins (GPIO0, GPIO1 & GPIO2) at reset.
- The '`sbzBootParameter`' field (offset 0x2C) should be zero.
- The '`imageLength`' field (offset 0x20) should be a multiple of 512 bytes and should be less than 128KB in value.

The boot ROM will deem that an image downloaded from UART/USB or copied from NAND/SPI-NOR/SD/MMC is valid if all the header checks pass and the SHA1 hash digest of the 'execution part' (offset 0x80 onwards of the image) of the image matches with the '`execution_hash`' field (offset 0x08) in the header.

LPC3143/LPC3154 supports secure booting from SPI flash, NAND flash, MCI device (SD/SDHC/MMC/eMMC cards), UART, and USB (DFU class) interfaces.

LPC3143/LPC3154 supports non-secure boot from UART and USB (DFU class) interfaces during development. Once the AES key is programmed in the OTP, only secure boot is allowed through UART and USB.



Color Legend

🟩 In LPC3130/31/41/52 boot flow these steps are not present.

🟧 In LPC3130/31/41/52 boot flow CRC32 algorithm is used. While in LPC3143/54 SHA1 hash algorithm is used.

** See “LPC314x ISROM/Boot ROM” chapter in user manual for more details on individual boot modes.

Fig 1. Main boot flow

2.1 Secure image format

LPC3143/LPC3154 boot ROM expects the boot image be compiled with an entry point at 0x11029000 and conforming to the layout described in Table 1 (except for "Parallel NOR flash" boot mode).

Table 1. LPC3143/LPC3154 Image format

Field Name	Offset	Size in bytes	Description
Image Header			
`Vector`	0x00	4	Valid ARM instruction. Usually this will be a branch instruction to entry point of the image.
`Magic`	0x04	4	This field is used by boot ROM to detect a valid image header. This field should always be set to 0x41676d69
`execution_hash`	0x08	20	SHA1 hash of execution part of the image (offset 0x80 onwards).
`imageType`	0x1C	4	Specifies from which interface the image is loaded. The image type and the interface from which the image is copied should match or else boot ROM will treat the image as an invalid image. 0x00000000 – Booting plain DFU image from USB 0x00000001 – Booting plain image from UART 0x00000002 – Booting AES encrypted DFU image from USB 0x00000003 – Booting AES encrypted image from UART 0x00000004 – Booting AES encrypted image from SPI-NOR 0x00000005 – Booting AES encrypted image from NAND 0x00000006 – Reserved 0x00000007 – Booting AES encrypted image from SD/MMC
`imageLength`	0x20	4	Total image length including header rounded-up to the nearest 512 byte boundary. In C language the field can be computed as: `imageLength = (Actual length + 511) & ~0x1FF`
`releaseID`	0x24	4	Release or version number of the image. Note, this field is not used by boot ROM but is provided to track the image versions.
`buildTime`	0x28	4	Time (expressed in EPOC time format) at which image is built. Note, this field is not used by boot ROM but is provided to track the image versions.
`sbzBootParameter`	0x2C	4	Should be zero.
`cust_reserved`	0x30	60	Reserved for customer use
`header_hash`	0x6C	20	SHA1 hash of image header excluding this field (i.e. bytes 0x00 to 0x6C of the image).
Execution part			
`Program code`	0x80	Max.	Program code. The maximum size of the image allowed by

Field Name	Offset	Size in bytes	Description
		`(128KB - 128)`	boot ROM is 128 KB (including header). The final image has to be padded to the nearest 512 byte boundary.

2.2 Creating secure images

NXP provides WinXP based command line tool “SecImgCr.exe” to create secure images. The tool assumes that the first 128 bytes (except the ‘vector’ field) of the image are left blank for the tool to fill-in. The ‘`vector`’ field should be set with ‘branch to ARM reset handler’ instruction op-code. For example, if the `reset_handler` is at offset 0x80 of the image then the `vector` field should be 0xEA00001E. Also the tool doesn’t change the “`cust_reserved`” field in the header and hence customer could use this space to store any other data of their interest. Creating bootable images for LPC3143/LPC3154 is a multi-step process.

1. Signing: To make images tamper proof, images are signed using hash algorithms to detect any bit changes in the original published image. LPC3143 & LPC3154 employ a 160 bit SHA1 hash algorithm to sign images. The following steps are performed on the image as part of signing process:

    a. Sets proper “`image_type`” field based on the command line options provided to the tool. Note, boot ROM will not boot an image if “`image_type`” and the boot mode selected by GPIO0, GPIO1 & GPIO2 pins don’t match.

    b. Pads the image to next 512 byte boundary and then updates the “`imageLength`” field in the header.

    c. Sets the magic number value “`magic`” field of the header.

    d. Sets the “`sbzBootParameter`” field to zero if it not zero already.

    e. Calculates SHA1 hash digest for the “execution” part of the image (i.e. image from offset 0x80 to end of file). Updates the “`execution_hash`” field with the computed hash value.

    f. Sets the current system time in epoch format in “`buildTime`” field of the header.

    g. Now computes the SHA1 hash digest for the image header part (i.e. image from offset 0x00 to 0x80). Updates the “`header_hash`” field with the computed hash value.

2. AES encryption: Once AES key is programmed in OTP, the LPC3143/LPC3154 will boot secure images only. That means the boot images should be encrypted using the same key programmed in OTP. If AES keys are not programmed in OTP then this step must be skipped (use –pu or –pd options). This step is performed when “-ad”, “-au”, “-as”, “-an” or “-am” option is used for image type.

    a. As part of AES decryption process, the boot ROM uses the following 128 bit value as the initial vector.

        i. `NandAESIV1 = 0xD9C7AE91;`

        ii. `NandAESIV2 = 0xCECABFDC;`

        iii. `NandAESIV3 = 0x3F3F857F;`

```
iv. NandAESIV4 = 0x0CF9F7ED;
```

b. As described in Decryption Engine section of this document, the AES engine on LPC3143/LPC3154 reads key and data in little-endian form. Hence for customers who are not using “SecImgCr.exe” tool should use the algorithm explained in section Encrypting Data for AES Decryption to encrypt their images.

3. TEA encryption: For additional security for images transferred over USB, TEA encryption is used for USB-DFU images. The TEA algorithm uses a 128 bit key and encodes 128 bits of data at a time. This step is needed when USB-DFU mode is selected as boot mode. If AES key is programmed (use “-ad” option) in OTP then the signed and AES encrypted image is used as the input image for this step. If AES key is not programmed (use “-pd” option) then just the signed image is used as the input image for this step. The LPC3143/LPC3154 boot ROM has 64 built-in 128 bit keys. The SecImgCr.exe tool uses the same TEA keys when creating the image so customers can use one of the 64 keys to encrypt their images. This is accomplished by passing the key index (a value from 0-63) as an argument to “SecImgCr.exe” using the “-K” (upper case K) option.

The following sub-sections show the command line options to create images for various boot modes.

2.2.1 Creating plain DFU image (image_type = 0x00)

To create images during development (i.e. no AES key is programmed in the poly-fuses) to be loaded using USB-DFU mode, use the following command line options:

SecImgCr.exe –pd –K <Tea key index> -o <output dir> -i <input file name>

- <TEA key Index>: LPC31xx boot ROM has 64 built-in TEA keys to decrypt DFU images. The “-K” (upper-case K) parameter specifies the TEA key index. A value between 0 and 63 is valid for this parameter.
- <output dir name>: The tool will create the signed and TEA encrypted image in this directory. The output file name is same as the input file name with extension “.rom”.
- <input file name>: The image file to be signed and TEA encrypted for USB-DFU boot mode.

Example:

```
C:\software\csps\lpc313x\bsps\ea3131\examples\otpdemo>secimgcr -pd -o ./ -K 0 -i
otp_demo.bin
LPC3143/LPC3154 Secure Image Creator Utility v1.1
...Checking image type parameter

Opening otp_demo.bin
WARNING: Mismatch of the image type. Overwriting with requested value!
TEA Key table index:    0
```

In the above example a file named otp_demo.rom is created in the current directory. This file should be used when the chip is set to boot in USB-DFU boot mode with no AES keys programmed in the poly-fuses.

2.2.2 Creating plain UART image (image_type = 0x01)

To create images during development (i.e., no AES key is programmed in the poly-fuses) to be loaded using UART boot mode use the following command line options:

Secimgcr.exe –pu -o <output dir> -i <input file name>

- <output dir name>: The tool will create the signed image in this directory. The output file name is same as the input file name with extension “.rom”.
- <input file name>: The image file to be signed for plain UART boot mode.

Example:

```
C:\software\csps\lpc313x\bsps\ea3131\examples\otpdemo>secimgcr -pu -o ./ -i
otp_demo.bin
LPC3143/LPC3154 Secure Image Creator Utility v1.1
...Checking image type parameter

Opening otp_demo.bin
WARNING: Mismatch of the image type. Overwriting with requested value!
```

In the above example a file named otp_demo.rom is created in the current directory. This file should be used when the chip is set to boot in plain UART boot mode.

2.2.3 Creating secure DFU image (image_type = 0x02)

To create secure images (i.e. AES key is programmed in the poly-fuses) to be loaded using USB-DFU mode use the following command line options:

Secimgcr.exe –ad –K <Tea key index> -k <AES key filename> -o <output dir> -i <input file name>

- <TEA key Index>: LPC31xx boot ROM has 64 built-in TEA keys to decrypt DFU images. The “-K” (upper-case K) parameter specifies the TEA key index. A value between 0 & 63 is valid for this parameter.
- <AES key filename>: File containing a 128 bit AES key (16 bytes) with byte 0 appearing first and byte 15 of the key at the end of the file. For example if the AES key registers are to be loaded with following values.
  - `NandAESKey1 = 0x0FC14139; NandAESKey2 = 0x00215B47; NandAESKey3 = 0xAF9E139D; NandAESKey4 = 0x1650EA23;`
  - Then OTP fuses data should be:
    - `OTP_data4 = 0x0FC14139 (fuses 159 to 128; i.e. fuses 131 to 128 are 1 0 0 1)`
    - `OTP_dat5 = 0x00215B47 (fuses 191 to 160)`
    - `OTP_data6 = 0xAF9E139D (fuses 223 to 192)`
    - `OTP_data7 = 0x1650EA23 (fuses 255 to 224)`
  - Then the key file should contain data in following order:
    - `0x39, 0x41, 0xC1, 0x0F, 0x47, 0x5B, 0x21, 0x00, 0x9D, 0x13, 0x9E, 0xAF, 0x23, 0xEA, 0x50, 0x16`
- <output dir name>: The tool will create a signed, AES encrypted and TEA encrypted image in this directory. The output file name is same as the input file name with extension “.rom”.

- <input file name>: The image file to be signed, AES encrypted and TEA encrypted for USB-DFU boot mode.

Example:

```
C:\software\csps\lpc313x\bsps\ea3131\examples\otpdemo>secimgcr -ad -K 0 -k encrypt.key
-o ./ -i otp_demo.bin
LPC3143/LPC3154 Secure Image Creator Utility v1.1
...Checking image type parameter

Opening otp_demo.bin
WARNING: Mismatch of the image type. Overwriting with requested value!
TEA Key table index:    0
```

In the above example a file named otp_demo.rom is created in the current directory. This file should be used when the chip is programmed with an AES key and is set to boot in USB-DFU boot mode.

2.2.4 Creating secure UART image (image_type = 0x03)

To create encrypted images (i.e. AES key is programmed in the poly-fuses) to be loaded using UART mode, use the following command line options:

Secimgcr.exe –au -k <AES key filename> -o <output dir> -i <input file name>

- <AES key filename>: File containing a 128 bit AES key (16 bytes) with byte 0 appearing first and byte 15 of the key at the end of the file. For example if the AES key registers are to be loaded with following values.
  - `NandAESKey1 = 0x0FC14139; NandAESKey2 = 0x00215B47; NandAESKey3 = 0xAF9E139D; NandAESKey4 = 0x1650EA23;`
  - Then OTP fuses data should be:
    - `OTP_data4 = 0x0FC14139 (fuses 159 to 128; i.e. fuses 131 to 128 are 1 0 0 1)`
    - `OTP_dat5 = 0x00215B47 (fuses 191 to 160)`
    - `OTP_data6 = 0xAF9E139D (fuses 223 to 192)`
    - `OTP_data7 = 0x1650EA23 (fuses 255 to 224)`
  - Then the key file should contain data in following order:
    - `0x39, 0x41, 0xC1, 0x0F, 0x47, 0x5B, 0x21, 0x00, 0x9D, 0x13, 0x9E, 0xAF, 0x23, 0xEA, 0x50, 0x16`
- <output dir name>: The tool will create a signed and AES encrypted image in this directory. The output file name is same as the input file name with extension ".rom".
- <input file name>: The image file to be signed and AES encrypted for UART boot mode.

Example:

```
C:\software\csps\lpc313x\bsps\ea3131\examples\otpdemo>secimgcr -au -k encrypt.key -o
./ -i otp_demo.bin
LPC3143/LPC3154 Secure Image Creator Utility v1.1
...Checking image type parameter
```

```
Opening otp_demo.bin
WARNING: Mismatch of the image type. Overwriting with requested value!
```

In the above example a file named otp_demo.rom is created in the current directory. This file should be used when the chip is programmed with an AES key and set to boot in UART boot mode.

2.2.5 Creating secure SPI-NOR image (image_type = 0x04)

To create encrypted images (i.e. AES key is programmed in the poly-fuses) to be loaded from SPI-NOR flash device, use the following command line options. Note, this boot mode is not supported if an AES key is not programmed in OTP.

Secimgcr.exe –as -k <AES key filename> -o <output dir> -i <input file name>

- <AES key filename>: File containing a 128 bit AES key (16 bytes) with byte 0 appearing first and byte 15 of the key at the end of the file. For example if the AES key registers are to be loaded with following values.
    - NandAESKey1 = 0x0FC14139; NandAESKey2 = 0x00215B47; NandAESKey3 = 0xAF9E139D; NandAESKey4 = 0x1650EA23;
    - Then OTP fuses data should be:
        - OTP_data4 = 0x0FC14139 (fuses 159 to 128; i.e. fuses 131 to 128 are 1 0 0 1)
        - OTP_dat5 = 0x00215B47 (fuses 191 to 160)
        - OTP_data6 = 0xAF9E139D (fuses 223 to 192)
        - OTP_data7 = 0x1650EA23 (fuses 255 to 224)
    - Then the key file should contain data in following order:
        - 0x39, 0x41, 0xC1, 0x0F, 0x47, 0x5B, 0x21, 0x00, 0x9D, 0x13, 0x9E, 0xAF, 0x23, 0xEA, 0x50, 0x16
- <output dir name>: The tool will create a signed and AES encrypted image in this directory. The output file name is same as the input file name with extension “.rom”.
- <input file name>: The image file to be signed and AES encrypted for SPI-NOR boot mode.

Example:

```
C:\software\csps\lpc313x\bsps\ea3131\examples\otpdemo>secimgcr -as -k encrypt.key -o
./ -i otp_demo.bin
LPC3143/LPC3154 Secure Image Creator Utility v1.1
...Checking image type parameter

Opening otp_demo.bin
WARNING: Mismatch of the image type. Overwriting with requested value!
```

In the above example a file named otp_demo.rom is created in the current directory. This file should be used when the chip is programmed with an AES key and set to boot from SPI-NOR flash device.

2.2.6 Creating secure NAND image (image_type = 0x05)

To create encrypted images (i.e. AES key is programmed in the poly-fuses) to be loaded from a NAND flash device, use the following command line options. Note, this boot mode is not supported if an AES key is not programmed in OTP.

Secimgcr.exe –an -k <AES key filename> -o <output dir> -i <input file name>

- <AES key filename>: File containing a 128 bit AES key (16 bytes) with byte 0 appearing first and byte 15 of the key at the end of the file. For example if the AES key registers are to be loaded with following values.
    - NandAESKey1 = 0x0FC14139; NandAESKey2 = 0x00215B47; NandAESKey3 = 0xAF9E139D; NandAESKey4 = 0x1650EA23;
    - Then OTP fuses data should be:
        - OTP_data4 = 0x0FC14139 (fuses 159 to 128; i.e. fuses 131 to 128 are 1 0 0 1)
        - OTP_dat5 = 0x00215B47 (fuses 191 to 160)
        - OTP_data6 = 0xAF9E139D (fuses 223 to 192)
        - OTP_data7 = 0x1650EA23 (fuses 255 to 224)
    - Then the key file should contain data in following order:
        - 0x39, 0x41, 0xC1, 0x0F, 0x47, 0x5B, 0x21, 0x00, 0x9D, 0x13, 0x9E, 0xAF, 0x23, 0xEA, 0x50, 0x16
- <output dir name>: The tool will create a signed and AES encrypted image in this directory. The output file name is same as the input file name with extension ".rom".
- <input file name>: The image file to be signed and AES encrypted for NAND boot mode.

Example:

```
C:\software\csps\lpc313x\bsps\ea3131\examples\otpdemo>secimgcr -an -k encrypt.key -o
./ -i otp_demo.bin
LPC3143/LPC3154 Secure Image Creator Utility v1.1
...Checking image type parameter

Opening otp_demo.bin
WARNING: Mismatch of the image type. Overwriting with requested value!
```

In the above example a file named otp_demo.rom is created in the current directory. This file should be used when the chip is programmed with an AES key and set to boot from NAND flash device.

2.2.7 Creating secure MCI (SD/MMC card) image (image_type = 0x07)

To create encrypted images (i.e. AES key is programmed in the poly-fuses) to be loaded from MCI interface (SD/MMC card or a managed NAND flash such as eMMC, eSD & moviNAND devices) device, use the following command line options. Note, this boot mode is not supported if an AES key is not programmed in OTP.

Secimgcr.exe –am -k <AES key filename> -o <output dir> -i <input file name>

- <AES key filename>: File containing a 128 bit AES key (16 bytes) with byte 0 appearing first and byte 15 of the key at the end of the file. For example if the AES key registers are to be loaded with following values.
  - `NandAESKey1 = 0x0FC14139; NandAESKey2 = 0x00215B47; NandAESKey3 = 0xAF9E139D; NandAESKey4 = 0x1650EA23;`
  - Then OTP fuses data should be:
    - `OTP_data4 = 0x0FC14139 (fuses 159 to 128; i.e. fuses 131 to 128 are 1 0 0 1)`
    - `OTP_dat5 = 0x00215B47 (fuses 191 to 160)`
    - `OTP_data6 = 0xAF9E139D (fuses 223 to 192)`
    - `OTP_data7 = 0x1650EA23 (fuses 255 to 224)`
  - Then the key file should contain data in following order:
    - `0x39, 0x41, 0xC1, 0x0F, 0x47, 0x5B, 0x21, 0x00, 0x9D, 0x13, 0x9E, 0xAF, 0x23, 0xEA, 0x50, 0x16`
- <output dir name>: The tool will create a signed and AES encrypted image in this directory. The output file name is same as the input file name with extension ".rom".
- <input file name>: The image file to be signed and AES encrypted for MCI boot mode.

Example:

```
C:\software\csps\lpc313x\bsps\ea3131\examples\otpdemo>secimgcr -au -k encrypt.key -o
./ -i otp_demo.bin
LPC3143/LPC3154 Secure Image Creator Utility v1.1
...Checking image type parameter

Opening otp_demo.bin
WARNING: Mismatch of the image type. Overwriting with requested value!
```

In the above example a file named otp_demo.rom is created in the current directory. This file should be used when the chip is programmed with an AES key and set to boot from SD/MMC card or a managed NAND flash (eMMC, moviNAND) device.

3. Decryption engine

LPC3143 and LPC3154 chips have on-chip 128 bit AES decryption engines to support secure booting. The decryption engine employs 128 bit AES with 512 byte CBC algorithm. That means the AES decryption engines uses a 128 bit key and a 128 bit initialization vector and encodes 128 bits of data at time. The resultant 128 bit ciphered output is used as the initial value for encoding the next 128 bits of data and this cycle continues for 512 bytes, after that the original 128 bit initial value is used.

It is assumed that most of the systems designed using LPC3143/LPC3154 will use NAND flash to store program code. To decrypt the program code with an algorithm like AES, a hardware implementation is needed. To do this without increasing the load on the AHB bus, it is essential that the AES decryption is integrated in the NAND controller module so that ECC and AES can be performed efficiently. In the NAND flash controller, an AES module is connected to the main control module. This module is kicked off after error correction in decode mode. In the read flow, the encrypted data is first read from the flash, error corrected, decrypted in hardware and stored back in the SRAM (NAND buffers present at 0x70000000). Even though the AES decryption engine is part of the NAND flash controller block, the engine can be accessed by ARM using AES_FROM_AHB mode. The AES engine can be set to AES_FROM_AHB mode by setting the 'mode' bit in AES_FROM_AHB register (address 0x1700 087C).

This module needs 11 to 12 clock cycles per 128 bits to process. Before the module can be used, an AES key and initial values need to be programmed. This is done via writing four registers, NandAESKey1 (address 0x1700 0854) – NandAESKey4 (address 0x1700 0860) to build the 128-bit AES key value. Similarly NandAESIV1 (address 0x1700 0864) – NandAESIV4 (address 0x1700 0870) registers must be written with the 128-bit initial vector value. Upon writing the fourth word the NAND flash controller automatically programs the AES key and initial value into the AES module. The first register (address 0x1700 0854) value is the least significant 32 bit field in the 128bit word. The last register (address 0x1700 0860) value is the most significant 32 bit field in the 128 bit word.

The data is read from the NAND buffer (SRAM at 0x70000000) in chunks of four words, processed in the AES module and sent decrypted back into the NAND buffer. To pipeline this as much as possible the AES controller reads the next chunk of four words before it is actually needed and stores it in a 128 bit register. In this way the time to decrypt 512 bytes is reduced to around 400 clock cycles.

The data is decrypted in little-endian mode. This means that the first byte read from flash is integrated into the AES codeword as the least significant byte. The 16th byte read from flash is the most significant byte of the first AES codeword. Most AES libraries on a PC are implemented with big-endian format hence byte reversal of key and data is needed while encrypting the data on a PC. To reduce confusion NXP provides image and data encryption tools which take care of the endianess issue. Hence it is highly recommended that customers use these tools to encrypt image/data for the LPC3143/LPC3154 to decrypt properly.

3.1 Accessing AES from AHB

To access AES decryption engine from ARM, do the following.

- Enable all essential clocks needed. Since the AES engine is integrated with the NAND controller, the following clocks should be enabled for the AES engine to function. NANDFLASH_S0_CLK(10), NANDFLASH_AES_CLK (12), NANDFLASH_NAND_CLK(13) and NANDFLASH_PCLK(14) clocks should be enabled.
- Write the initial vector to NandAESIV1 (address 0x1700 0864) – NandAESIV4 (address 0x1700 0870) registers.
- Write the AES key to NandAESKey1 (address 0x1700 0854) – NandAESKey4 (address 0x1700 0860) registers.
- Set the ‘mode’ bit in AES_FROM_AHB register (address 0x1700 087C). By writing the value 0x00000008 to the AES_FROM_AHB register.
- Now copy 512 bytes of encrypted data to one of the NAND buffers, either RAM0 (address 0x7000 0000) or to RAM1 (address 0x7000 0400) using ARM instructions or by memory-to-memory DMA operation.
- Once copying is done set the appropriate ‘decryptRAMx’ bit in AES_FROM_AHB register. i.e. if data in RAM0 has to be decrypted set ‘decryptRam0’ (bit 0). By writing AES_AHB_REG = 0x00000009. This starts the hardware decryption engine.

- The completion of the decryption process is indicated by hardware in the NandIRQSatus1 register. Check “RAM0 AES done” (bit 0) in NandIRQStatus1 register (address 0x1700 0800). Similarly check “RAM1 AES done” (bit 1) when ‘decryptRAM1’ is set.
- Once the “RAMx AES done” bit is set, the data present in the NAND buffer RAM0/RAM1 is decrypted data. Once the bit is set it can only be cleared by writing a ‘1’ to this bit in NandIRQStatus1 register.
- Note, when AES engine is accessed through AHB, NAND operations using hardware flow control are not possible as the NAND buffers are used by the AES engine.

3.2 Accessing AES during NAND read operations

The AES decryption engine can be invoked during NAND read operations only when hardware flow control is used to read external NAND chips. To access the AES decryption engine during NAND read operations set the “AES on” bit (Bit 2) in the NandConfig register (address 0x1700 080C).

3.3 Encrypting data for AES decryption

As described in the previous section, the AES decryption engine on the LPC3143/LPC3154 is designed assuming little-endian format across 16 bytes of data. Hence the data encrypted by external systems such as a PC should be aware of this artifact and perform the byte reversal of data during the encryption process. NXP provides WindowsXP based SecImCr.exe and aes_enc.exe tools to encrypt boot images and data. These tools take care of the byte reversing issue. Customers who are not using these tools should use the following pseudo-code as an example for encrypting their images/data.

1. Do byte reversal of 16 bytes AES key (i.e. `rev_key [0] = key[15], rev_key[1] = key[14] …`).
2. Read 16 bytes of initial vector in `cbc_iv`.
3. Read 16 bytes of data and XOR with initial vector (`cbc_iv`).
4. Do byte reversal of 16 bytes resultant obtained in step 3. (i.e. `rev_data [0] = xor_data[15], rev_data[1] = xor_data[14] …`).
5. Now use `rev_key` and `rev_data` as inputs to AES-CBC encrypt routine.
6. Now byte reverse the resultant obtained in step 5 and save the reversed `rev_enc` data in the output file. (i.e. `rev_enc[0] = enc[15], rev_enc[1] = enc[14], …`).
7. Copy `rev_enc` to `cbc_iv` as initial vector for next iteration. Go to step 3 until 512 bytes of data is encrypted.
8. Once 512 bytes are encrypted reset the `cbc_iv` value to the original initial vector. i.e. go to step 2.
9. Repeat steps 2 to 8 until the complete data is encrypted. Since the decryption engine operates on 512 byte blocks it is required that the data be padded to next 512 byte boundary before encryption.

3.3.1 Using SecImgCr.exe for encrypting images

Refer to sections 2.2.3 through 2.2.7 above.

3.3.2 Using aes_enc.exe for encrypting data

To create encrypted data to be decrypted by the AES engine, use the aes_enc.exe tool with the following command line options:

aes_enc.exe -k <AES key filename> -c < CBC IV filename> -o <output dir> -i <input file name>

- <AES key filename>: File containing 128 bit AES key (16 bytes) with byte 0 appearing first and byte 15 of the key at the end of the file. For example, if the AES key registers are to be loaded with following values:

  ```
  NandAESKey1 = 0x0FC14139; NandAESKey2 = 0x00215B47; NandAESKey3 = 0xAF9E139D;
  NandAESKey4 = 0x1650EA23;
  ```

  Then the key file should contain data in following order:

  ```
          0x39, 0x41, 0xC1, 0x0F, 0x47, 0x5B, 0x21, 0x00, 0x9D, 0x13, 0x9E,
          0xAF, 0x23, 0xEA, 0x50, 0x16
  ```

- <CBC IV filename>: File containing 128 bit CBC initial vector (16 bytes) with byte 0 appearing first and byte 15 of the vector at the end of the file. For example, if the CBC IV registers are to be loaded with following values:

  ```
  NandAESIV1 = 0xDEADBEEF; NandAESIV2 = 0xFADCABED; NandAESIV3 = 0xABADDEED;
  NandAESIV4 = 0x12345678;
  ```

  Then the CBC_IV file should contain data in the following order:

  ```
          0xEF, 0xBE, 0xAD, 0xDE, 0xED, 0xAB, 0xDC, 0xFA, 0xED, 0xDE, 0xAD,
          0xAB, 0x78, 0x56, 0x34, 0x12
  ```

- <output dir name>: The tool will create the encrypted image in this directory. The output file name is same as input file name with extension ".enc".
- <input file name>: Binary file containing data to be encrypted.

Example:

```
C:\dev\svn\nxpmcu\software\csps\lpc313x\bsps\ea3131\examples\aesfromahb>aes_enc
-k encrypt.key -c iv.cbc -o ./ -i plain_data.dat
LPC3143/54 AES encode Utility v1.1
...Checking image type parameter

Opening plain_data.dat

Opening encrypt.key

Opening iv.cbc
```

In the above example the file plain_data.enc is created in the current directory. The contents of this file can be used as encrypted data for the chip to decrypt.

4. OTP fuses

LPC3143/LPC3154 has 512 one-time-programmable (OTP) poly-fuses on chip to store security keys, unique ID, USB VID & PID, JTAG security level flags and custom data.

4.1 OTP fuse map

Refer to the “OTP fuse map” section of the “LPC314x/LPC315x One-Time Programmable memory (OTP)” chapter in the LPC314x/LPC315x user manual. Fuses which are marked reserved in the map should not be programmed. Programming reserved fuses will be treated as a security hack by the boot ROM and will lock the processor permanently. So customers should take caution when programming the poly-fuses.

4.1.1 Unique ID fuses

A 128-bit unique ID is programmed in fuses fuse_128 to fuse_0 by NXP during chip production.

4.1.2 AES key fuses

The boot ROM on LPC3143/LPC3154 assumes that the AES key is programmed in fuse_128 to fuse_255 for secure booting. The boot ROM reads the AES key from the fuse block only if the “AES key valid” fuse (fuse_504) is programmed. Hence must program fuse_504 once a valid AES key is programmed in fuse_128 to fuse_255.

4.1.3 JTAG security fuses

A JTAG interface is provided for boundary scan and to debug software. Using a debugger program on a PC and a JTAG probe, users could control the ARM core on these chips and also dump registers and memory contents for debug purposes. However, such access to chip internal contents would compromise the code security (assuming the customer is using the secure boot process). Hence to safeguard the code security, the JTAG block on the LPC3143/LPC3154 chips is controlled by “JTAG security fuses”. The JTAG security fuses define four levels of access control as described below:

- Level 0: The JTAG block is at this level when fuse_511, fuse_510 and fuse_509 are not programmed. At this level JTAG access is enabled all the time. This level is used during the development phase. This is the factory default state for all chips.
- Level 1: The JTAG block is at this level when fuse_509 is programmed and fuse_510 & fuse_511 are left untouched. At this level, JTAG access can be enabled by software (already running on the chip) by setting the sticky bit (the bit can be un-set only by chip reset) 'JTAG_EN' (bit_31) in OTP_con register (address 0x13005000). Customer software could implement a password mechanism (such as multi-pin state combination or a hardware event) to set the ‘JTAG_EN’ bit. Hence this level is also referred as “password protected level”.
- Level 2: The JTAG block is at this level when fuse_509 and fuse_510 are programmed and fuse_511 is left untouched. At this level, JTAG access can be enabled by special test equipment. This level should be used only if the customer expects NXP to support “Returned Material Analysis” for parts which fail in the customer’s system.
- Level 3: The JTAG block is at this level when fuse_511, fuse_510 and fuse_509 are programmed. At this level, JTAG access is completely and permanently disabled.

Customers using secure boot should set the JTAG security level to anything other than Level 0 to completely protect their code. Customers using external SDRAM should devise other schemes to protect data exchanged across the SDRAM interface. LPC3143/LPC3154 boot ROM disables parallel NOR-flash boot if the JTAG level is set to anything other than level 0. Hence parallel NOR-flash boot based designs are not recommended for LPC3143/LPC3154.

4.1.4 USB VID PID fuses

In USB-DFU boot mode, by default the boot ROM reports VID=0x0471 (Philips ID) and PID=0xDF55 during USB enumeration. Customers who plan to use USB-DFU boot mode for field upgrades, have to use their own VID and PIDs. To facilitate customized VID & PID numbers, the boot ROM reads the values from OTP fuses. Customers should program the following fuses to use this feature.

Table 2. USB VID and PID fuses

Field Name	Fuses	Size in bits	Description
`VID`	`Fuse_479 to Fuse_464`	16	16 bits containing Vendor Identification number assigned by usb.org.
`PID`	`Fuse_463 to Fuse_448`	16	16 bits containing product identification number registered with usb.org for this product family.
`PID_VID valid`	`Fuse_503`	1	Customer should program this fuse if a valid PID and VID are programmed in Fuse_479 to Fuse_448. Boot ROM will read PID & VID from OTP only if Fuse_503 is programmed.

4.1.5 DFU fall through fuse

As shown in Fig 1 “Main boot flow”, the boot ROM on LPC3143/LPC3154 will switch to USB-DFU boot mode if a valid image is not found on SPI-NOR, NAND or MCI devices in their respective boot modes. This feature is provided for customer production-line programming for systems in which boot mode selector switches/jumpers are not available. Once the device enters USB-DFU boot mode, customers can then download their test/program image which programs the boot device with final production firmware and also programs the OTP fuse block. Customers who want to disable this feature in their final end product should program fuse_502 to disable this “DFU fall through” feature.

4.1.6 Customer fuses

All unmarked fuses in the fuse map (see the User Manual for the fuse map) are available for customers to use them as OTP memory.

4.2 Programming OTP fuses

OTP fuses can be programmed only once and only one at time. An un-programmed fuse when read will return a zero while a programmed fuse will return one. In other words, customers should program only fuses which they want to set to one. For example, if we need to program 0x9 (1001’b) in fuses 131 to 128. We should program fuse_131 and fuse_128 and leave fuse_130 and fuse_129 untouched.

Use following steps to program fuses:

1. Enable OTP_PCLK (clock id = 38) in CGU block.

2. Disable “external enable” feature of the clock until programming is done. i.e. clear bit 3 in PCR38 register (address 0x13004158).
3. Set OTP_PCLK clock rate to a value between 100 KHz and 500 KHz. This can be achieved by setting a larger divider value for the fractional divider associated with OTP_PCLK. Programming fuses is possible only when the OTP_PCLK is less than 500 KHz.
4. Apply write voltage of 3.3V to VPP pins.
5. Remove the write protection for the corresponding 32 bit fuse block. The 512 fuses on this chip are divided into 16 blocks of 32 fuses each. The OTP_wport register (address 0x13005008) has a bit for each of the 16 blocks which controls the write protection for that block. If the bit (bit_0 to bit_15) in OTP_wport register is set then the 32 fuses associated with that bit can’t be programmed. For example, for programming fuse_131 and fuse_128, bit 4 in OTP_wrport register should be cleared. Note, do not set the ‘lock’ bit (bit 31) in this register. The lock bit is sticky in nature, which causes the OTP_wrport register to freeze its state until the chip is reset. This feature could be used in production software for additional security.
6. Make sure the fuse to be programmed is in idle state. To set the fuse in idle state, program the fuse address in the ‘ADRS’ field (bits 8:0) and set the ‘mode’ field (bits 17:16) to 00’b the in OTP_con register (address 0x13005000). For example, to put fuse_128 in idle state, write 0x00000080 to OTP_con register.
7. Leave the fuse in this state for at least 6 micro-seconds.
8. Now blow the fuse by setting it in write state by programming the fuse address in ‘ADRS’ field (bits 8:0) and setting ‘mode’ field (bits 17:16) to 10’b in OTP_con register (address 0x13005000). For example to program fuse_128, write 0x00020080 to OTP_con register.
9. Leave the fuse in this state for at least 6 micro-seconds.
10. Now set the fuse back in idle state by programming OTP_con register. For example to put fuse_128 in idle state, write 0x00000080 to OTP_con register.
11. Repeat steps 5 to 10 for all the fuses to be programmed.
12. Once fuses are programmed apply read voltage of 1.2V on VPP pins. Applying write voltage level on VPP pins during normal operation could cause fuses to be blown randomly, leading to processor lock-up. Hence write voltage level (anything above 2.7V) should be applied only for the period of write operation. For all other remaining periods, a read voltage level of 1.2V should be applied on VPP pins for the chip to operate.
13. Set the OTP_PCLK speed back to 6MHz.
14. Enable ‘external enable’ feature of the OTP_PCLK to conserve power. By setting this bit the OTP_PCLK is enabled only during read/write operation of OTP registers. When no read/write accesses are performed on the OTP register block, the clock is gated automatically by the CGU block.

5. Suggested development process

NXP recommends the following two development methods for customers using LPC3143 and LPC3154 chips:

1. Customers should use USB-DFU plain-text mode or UART plain-text mode as boot methods during their development phase. Once the firmware is finalized customers should program the boot device (NAND flash, SPI-NOR flash or MCI device) with encrypted images and program AES keys in OTP.
    - Set the board in USB-DFU boot mode or UART boot mode.
    - Use SecImgCr.exe to signing the image for USB-DFU plain-text boot or UART plain-text boot.
    - Test the board and firmware in this mode.
    - Now create the final signed and encrypted image for final boot device (NAND flash, SPI-NOR flash or MCI device).
    - Program the boot device with encrypted image.
    - Program AES key and AES Key valid fuses in the OTP block.
    - Change the boot mode resistors/jumpers to final production boot mode.
    - Do the final testing in this configuration.
2. Customers could populate their prototype boards with LPC3141 (for LPC3143 products) or with LPC3152 (for LPC3154 products) chips since the LPC3141 is pin and feature (except security engine) compatible with LPC3143. Similarly the LPC3152 is pin and feature (except security engine) compatible with LPC3154.
    - Once the board and firmware are thoroughly tested replace LPC3141 with LPC3143 chip.
    - Set the board in USB-DFU boot mode or UART boot mode.
    - Use SecImgCr.exe to signing the image for USB-DFU plain-text boot or UART plain-text boot.
    - Test the board and firmware in this mode.
    - Now create the final signed and encrypted image for the final boot device (NAND flash, SPI-NOR flash or MCI device).
    - Program the boot device with the encrypted image.
    - Program AES key in the OTP block.
    - Change the boot mode resistors/jumpers to final production boot mode.
    - Do the final testing in this configuration.

6. Suggested production process

The production process for end products using LPC3143/LPC3154 chips can be divided into multiple steps as below.

1. Running test & diagnostics software: In this step, test firmware is loaded into the device to test that various memories and interfaces on the board are working properly and to detect any production/assembly errors. For example, memory bit pattern tests are done to detect whether all data and address lines are properly connected to the memory device. If there are any errors then the software should indicate which bit/line is not connected properly.
2. Programming boot device: In this step the boot device (NAND flash, SPI-NOR flash or MCI device) is programmed with the encrypted firmware.

3. Programming OTP: In this step OTP fuses are programmed with the AES key, USB VID/PID, etc.
4. Functional test: In this step any desired functional tests are done on the product.
5. Programming security fuses: This step should be done at the very end since disabling the JTAG interface disables the debug interface of the chip. Also in this step the customer may choose to disable the DFU fall-through mechanism by blowing fuse_502.

6.1 Using DFU method

This method is recommended for designs which have USB device (standard-B, mini-B, micro-B, micro-AB) connectors. As described in section “4.1.5 DFU fall through fuse” the boot ROM will switch to USB-DFU mode if a proper image is not programmed in the boot device (NAND, SPI-NOR or MCI device). Since a freshly assembled board (assuming no pre-programmed components are used) will be in this state, an operator could then connect the board to a PC using the USB interface. The device will enumerate as a DFU Class device for firmware download. Batch applications could be developed on a PC which download multiple images to the board for test & diagnostics, OTP programming, and boot device programming. For example, below is the list of steps which could be carried out in a production flow:

- Typically after all components are assembled on the PCB. Open and short tests are done.
- An operator connects the board using a USB cable to a PC and powers up the board.
- LPC3143/LPC3154 will enumerate as DFU class device with the PC.
- The batch software running on the PC will download the “test & diagnostics” firmware to do functional tests on the board. The firmware downloaded in this step must be a signed image, but not encrypted (use secImgCr.exe –pd option).
- The test firmware signals (using onboard LEDs or another mechanism) that tests have passed and does a self reset of the board using the watch dog reset mechanism.
- The boot ROM will re-enumerate with the PC as a DFU class device since no valid firmware is programmed in the boot device.
- On this second iteration the batch software on the PC will download a “Firmware programmer” program to the device. The programmer will program the boot device and also program the OTP with the AES key. If the end product firmware is not embedded in the programmer code, then it can re-enumerate with PC as a DFU class device to download the end product firmware to be programmed in the boot device.
- Now the operator can disconnect and test the device for final functional validation of the product.
- Once the device is tested for end product functionality the firmware should provide a mechanism to program the JTAG fuses. If comprehensive tests are done as part of “test & diagnostics” step, then programming of security fuse can be combined with the previous step and skip the functional validation step.

Note, depending on the size of “test & diagnostics” firmware and “firmware programmer” code, they can be combined into one image, reducing the number of steps during production.

6.2 Using UART boot method

This method could be used for designs which have a serial port in their final product. Instead of USB-DFU downloads, the firmware has to be downloaded using UART ports. All other steps remain the same in this method also.

6.3 Using DFU and SD/MMC boot method

This method could be used for designs which have an SD/MMC slot in their final product. In this method, AES keys should be programmed in OTP first using DFU or UART or JTAG interfaces. Once the AES keys are programmed the test/diagnostics/programmer firmware can be loaded from SD/MMC cards.

7. Additional reference

TBD.

8. Legal information

8.1 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.

8.2 Disclaimers

General — 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.

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 medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a 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 for 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.

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.

8.3 Trademarks

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

9. Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.

For more information, please visit: http://www.nxp.com
For sales office addresses, email to: salesaddresses@nxp.com

Date of release: 5 January 2010

Document identifier: AN10895_1