AN12291

IP Binding on NXP LPC MCUs featuring on-chip Flash

Rev. 1.0 — 06 November 2018 Application note

Document information

Info	Content
Keywords	PUF, security, IP binding, secure boot, key storage
Abstract	This application note describes how to develop applications on NXP LPC MCUs to provide strong protection for stored information and protect against reverse engineering and cloning. The protection of this software Intellectual Property (IP) is built on the use of an SRAM Physically Unclonable Function (PUF). The SRAM PUF uses entropy in uninitialized SRAM to create device unique keys that bind cryptographically protected software IP to a specific device.

Revision history

Rev	Date	Description
1.0	11062018	Initial version

Contact information

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

1. Introduction

This application note describes techniques to protect embedded applications on LPC 32-bit ARM microcontrollers (MCUs) from reverse engineering, unauthorized firmware tampering, overproduction and counterfeiting. The protection is provided by binding the firmware and configuration information to the unique physical properties of the SRAM of MCU. The software Intellectual Property (IP) is bound to a specific device by an encryption process that uses cryptographic keys derived from a Physically Unclonable Function (PUF). Since these keys are unclonable and device-unique, the encrypted information that is stored is readable only by the authorized device. This *device-unique* encryption makes it difficult for an attacker to reverse engineer or modify the firmware. The overall process supports:

- A device-unique unclonable identity that can be cryptographically authenticated.
- Code authentication to ensure firmware may only be installed and updated by the OEM.
- Device-unique encryption keys to protect stored information from cloning, reverse engineering or modification.
- Optional version number validation to ensure that only new firmware updates are installed to prevent *roll-back attacks.*
- Additional confidential information can be protected to support application services. This information may be additional keys, trust roots or configuration information. This sensitive data has the same strong device-unique protection as the firmware and is strongly protected from extraction.
- Additional secret keys may also be generated by the PUF APIs to support strong device authentication using symmetric keys, public keys and/or public key certificates.

The process to design and field a secure embedded application using PUF is described. The solution is software-based but has implications on the system design and the manufacturing test process. Software modules are provided for integration with production test fixtures and applications. An IP binding *Tool Suite* is available to protect firmware and is used to create both factory installed images and images for over-the-air upgrades.

1.1 Physically unclonable functions

Like a snowflake, the atomic structure of every semiconductor device is different. These differences in atomic structure are expressed in the power-up state of uninitialized SRAM which forms a unique pattern for every device, like a *silicon fingerprint*. Keys derived from the SRAM PUF are not stored *on the chip* but are extracted *from the chip*, only when needed. They are only present in the chip during a short time window. When the SRAM is not powered, there is no key present on the chip. It makes the solution secure from reverse engineering and key extraction.

1.2 IP binding overview

The *binding* of software IP to a device uses a Unique-Device Secret (UDS) to encrypt stored information. The UDS key is derived from the SRAM PUF and provides the foundation of the security assurances. The UDS may also be used to derive a unique identifier for device identification and authentication. A wide range of keys, both symmetric and asymmetric, can be derived from this initial Root of Trust (RoT) for use by embedded applications. This application note focuses on the use of the UDS in an IP binding solution that protects OEM firmware and configuration information. Developers may also use the PUF based security for other security requirements within their applications.

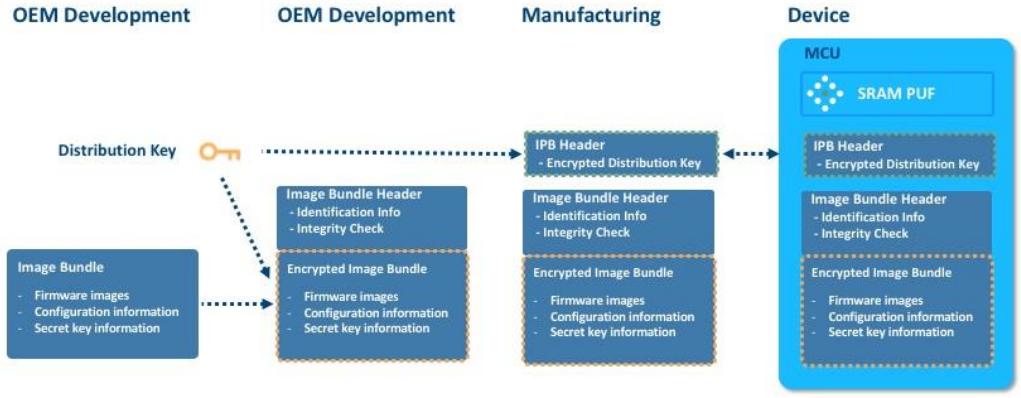

Fig 1. IP binding and the IP binding header

The encryption process that binds an image bundle to a device is illustrated in Fig 1. The OEM creates a firmware image and may bundle the image with additional secrets and configuration information. The image is protected by the OEM using a symmetric distribution key. The encryption process adds a header to the image bundle that identifies the protected bundle and provides integrity checks that are validated when decrypted. At the manufacturing facility, each device is uniquely authorized to decrypt the bundle by the creation of an IP binding header. This header contains the distribution key encrypted in a key that is uniquely generated from the SRAM PUF of the device. For protection of threats in manufacturing the distribution key should be securely stored and managed in a Hardware Security Module (HSM). Since the header is created with a PUF-derived secret, its encapsulated key is protected from extraction or use by any device other than the bound device.

The distribution key in the header will be set to the same key for all devices of the same model. It allows the same encrypted image bundle to be installed into a series of devices. The installation and distribution of the IP in the images is protected and the same distribution key may subsequently be used to create upgrade images.

The headers in the image bundle include a version number. The upgrade process may check this version number and include enforcement of monotonically increasing version numbers to prevent roll-back attacks.

The IP binding header may also be configured to include a unique distribution key per device. It provides the ability to control the distribution of images to individual devices.

2. Implementing IP binding

This section provides additional details and example code snippets to help developers create and deploy IP binding solutions.

2.1 Design

PUF device-unique keys provide a strong Root-of-Trust (RoT) for a system design. To use PUF, an embedded application must accommodate the use of uninitialized SRAM during the boot process to initialize and extract secret keys. An example of the SRAM memory allocation with PUF is shown in Fig 2.

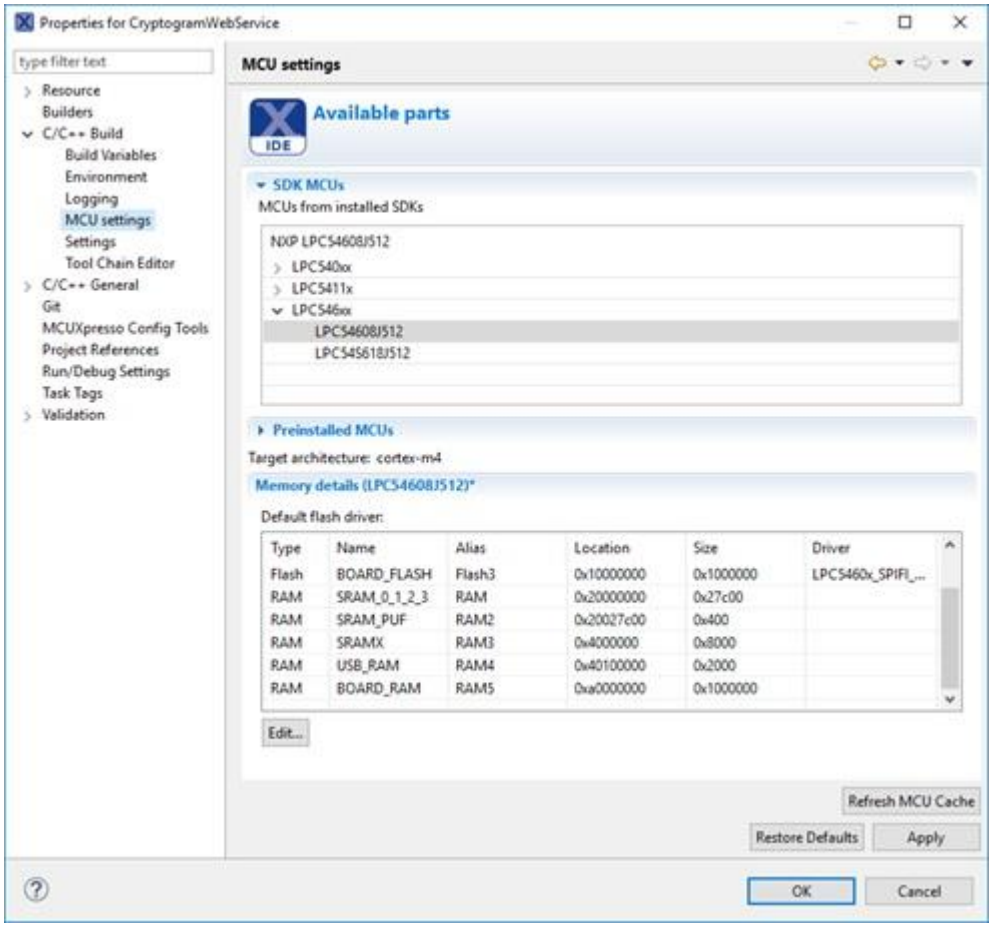

Fig 2. SRAM PUF memory allocation

The design must also ensure that the memory usage and OTP settings for the MCU are appropriate for the security design. Devices are only as secure as their overall configuration and designers should ensure that the locking mechanisms provided by NXP LPC MCUs are used correctly to protect the system. SWD and debug features must be disabled by the appropriate OTP settings.

2.2 Creation of a protected image bundle

Images are created and tested using the normal build process. Before protection of the image, it may be bundled with another configuration and secret key information. It allows application-unique roots of trust and authentication keys to be securely transmitted and installed with the firmware image.

The image bundle is encrypted with a distribution key using the IP Binding (IPB) tool. The IP binding tool suite is a command-line tool used to protect firmware images and provision additional keys. It is used in the software tool chain in the build/release of OEM process. Additionally, it can be used at an over-the-air update server for individual on-the-fly protection of the updated code images.

The tool suite uses an IPB_HEADER from an individual chip to extract its configured protection profile (authentication, encryption, anti-rollback) in order to select the protection options to be applied. It then takes an unprotected code image and optional additional keys and protects the code and keys using the distribution key to result in a protected code image. The latter is processed by the IPB firmware. When anti-rollback is active, the tool reads a 32-bit image version number from a file and increments the version number inside that file.

2.3 OEM device provisioning

A product is personalized in the OEM facility using a manufacturing tool that is used to test and program the device; see Fig 3. The cryptographic processing using the master distribution key should be within a Hardware Security Module (HSM) when the manufacturing facility is not fully trusted to protect this key.

The manufacturing tool starts the personalization process by injecting and running an *enrollment image* into a device. This code is transient and only runs once.

The functions of the enrollment image include:

- Device test functions.
- PUF initialization and device-unique key creation.
- Extraction of device unique public keys.
- Creation and storage of the IP binding header device in external flash.
- Loading the protected image bundle into flash.
- Obtaining a wrapped firmware distribution key and writing the device unique IPB header to NV memory.
- Locking down the device to prevent use of SWD and boundary scans and preventing any subsequent provisioning of the device.

The enrollment image is specific to an MCU and may include application specific testing and initialization. The PUF initialization and IP binding header functionalities must be integrated with the existing enrollment image software. Sample code for the PUF initialization and IP binding functions are provided for this integration in Section 6.1.

The PUF unique keys are used to create a unique identifier for the device. It may be extracted and stored for subsequent tracking of the product.

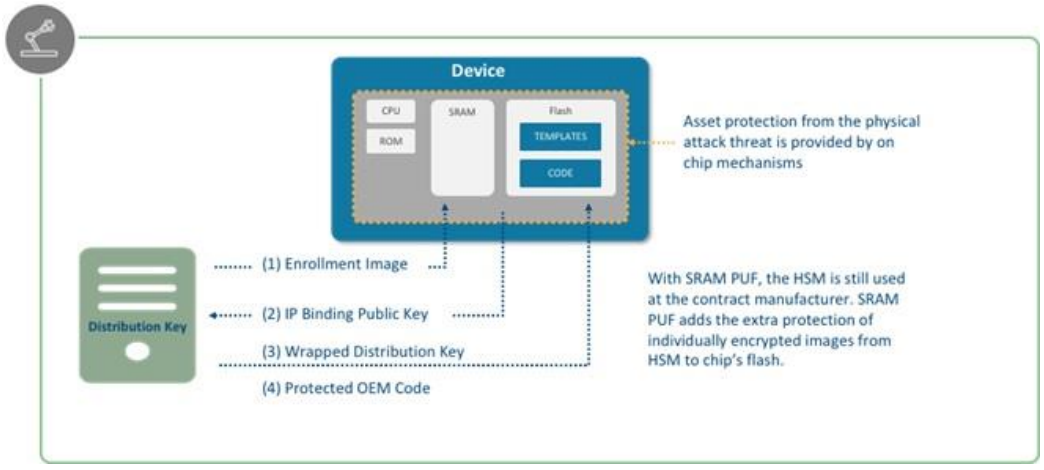

Fig 3. OEM device provisioning

2.4 Secure boot using SRAM PUF

On boot, the device creates the device unique secret keys using the SRAM PUF. This device firmware encryption key is then used to decrypt the IP binding header which provides the distribution key to decrypt and authenticate the image bundle and then run the unwrapped firmware. The integration of these capabilities is supported by an IP binding firmware module that provides all of the required PUF and encryption functionality, as described in Section 4.

Example code illustrating how to integrate a secure boot process is provided in Section 6.2.

3. Format specifications

The format of the IP binding header and image bundle header is described in this section. The IP binding header is encrypted with a PUF-derived key that is unique to a single device. The image bundle header is unique for an encrypted firmware image. The encrypted image bundle may be used with multiple devices and each device is enabled to use the bundle by giving it a unique IP binding header which contains a common distribution key encrypted for the specific device.

3.1 IP binding header

The IP binding header is a device unique sequence of octets that enables the decryption of a provided firmware image. The header contains three fields:

- Security options that describes if anti-rollback protection is required.
- Encrypted distribution key.
- A Message Authentication Code (MAC), used to provide a strong integrity check of the security.

The format of these fields is described below:

```
IPB_HEADER
{
  uint32_t security_options;  /* anti_rollback flags */
  uint8_t encrypted_distribution_key[32];
  uint8_t IPB_HDR_MAC[32];  /* MAC of prior two fields using PUF derived key*/
}
```

The options are set when the IP binding header is created. The options include the roll-back policy and the ability to support null encrypt for use in debug and development. Example code and usage are provided in Section 6.

3.2 Image bundle header format

The format of image bundle header is described below:

```
IMG_HEADER
{
    security_version[4]
    version_counter[4]
    image_size[4]
    additional_keys_size[4]
    jump_offset[4]
    hash_encrypted_additional_keys[32]  /* SHA256 */
    hash_encrypted_image[32]            /* SHA256 */
    IMG_HDR_MAC[32]  /* MAC of everything up to here */
}
```

The additional keys stored in the protected image bundle are available to the developer to use for protecting trust roots and application-specific secrets.

4. IP binding firmware

The IP binding firmware is the device side of the solution, implemented in a single firmware module compiled to a binary library for the targeted platform. The IP binding firmware authenticates, decrypts to SRAM, and protects from rollback a binary code image and additional keys. Authentication, decryption, rollback protection are the features that can be individually activated or deactivated with the help of the IP binding tool; see Section 5.1 and the resulting IPB_HEADER data structure programmed into the device. The firmware image is internally interfaced to a device-unique key source.

5. IP binding tool description

5.1 IP binding tool suite

The IP binding tool suite v1.0 is a command-line tool used to protect firmware images and provision additional keys. It is used in the software tool chain in the build/release process of OEM. Additionally, it can be used at an over-the-air update server for individual on-the-fly protection of the updated code images.

The tool suite uses an IPB_HEADER from an individual chip to extract its configured protection profile (authentication, encryption, anti-rollback) in order to select the protection options to be applied. It then takes an unprotected code image and optional additional keys and protects the code and keys using the distribution key to result in a protected code image. The latter is processed by the IP binding firmware. When anti-rollback is active, the tool reads a 32-bit image version number from a file and increments the version number inside that file.

6. Example code

The following examples provide code suitable for use in the development of a secure MCU product utilizing the SRAM PUF to bind software IP to the device's PUF device secret keys.

6.1 Code example for the enrollment image

The following code supports the initialization and use of PUF based keys during the initial device enrollment.

Enrollment involves the following:

- Programming a firmware image integrated with IPB firmware into the individual chip's NVM.
- Booting the chip and calling bk_init(), then bk_enroll() to produce the chip-unique and enrollment-unique Activation Code (AC).
- Calling ipb_bind_header() with the security profile configuration options in order to produce the chip-unique IPB_HEADER.

Note: Enrolling again would produce a different AC, which cannot be used with a mismatching IPB_HEADER – ipb_bind_header() would need to be repeated with the new AC

- The AC and the IPB_HEADER is stored in on-chip NVM.
- Optionally, the AC and IPB_HEADER is sent outside the chip for storage in a database for the purpose of backup recovery or server based activation without NVM.
- Use case-appropriate hardware access control mechanisms are permanently enabled. Examples of such are SWD/debug interface disablement, external flash write and erase disablement, changing an NVM variable to disable the command implementing the enrollment flow on the chip.

The code below should be integrated with the test and initialization software injected into a device by the manufacturing test system.

```
18  include/iidbroadkey.h
19  #define BK_SRAM_PUF_SIZE_BYTES              (1024)
20  #define BK_AC_SIZE_BYTES                    (788)
21
22  /*
23        *  REQUIREMENT:
24        *    Modify the Memory Map to allocate SRAM slice (RAM2)
25        *    (identify/allocate/isolate SRAM memory)
26        */
27          static __attribute__ ((section(".noinit.$RAM2"))) uint8_t
    sram_puf[BK_SRAM_PUF_SIZE_BYTES];
28      static uint8_t activation_code[BK_AC_SIZE_BYTES]__attribute__ ((aligned
    (4)));
29
30
31  APP_PUF_store_activation_code( activation_code ) - Application dependent code to
    write Activation Code (ex.  Non-volatile memory)
32
33  APP_IPB_load_distribution_key( distribution_key ) - Application dependent
    function to provide Application dependent Distribution Key
34
35  APP_IPB_store_counter( 0 )   -  Initialize Monotonic Counter
36
37
38
39          /*  Initialize SRAM dedicated for PUF */
40          ret_code = bk_init( sram_puf, BK_SRAM_PUF_SIZE_BYTES );
41          if (IID_SUCCESS != ret_code)   break;
42
43          ret_code = bk_enroll( activation_code );
44          if (IID_SUCCESS != ret_code)   break;
45
46          /*  Application-Specific method for storage (i.e. flash, OTP)
47          ret_code = APP_PUF_store_activation_code( activation_code );
48          if (IID_SUCCESS != ret_code)   break;
49
50          /*  Application-Specific method for Distribution Key management
    (cryptogram)
51          ret_code = APP_IPB_load_distribution_key( distribution_key );
52          if (IID_SUCCESS != ret_code)   break;
53
54          ipb_header_t   ipb_header;
55          ret_code = ipb_bind_header(
56                                  (IP_BIND_AUTHENTICATE | IP_BIND_ENCRYPT |
    IP_BIND_NON_ROLLBACK),
57                                  distribution_key,
58                                  (ipb_header_t * const )&ipb_header );
59          if (IID_SUCCESS != ret_code)   break;
60
61
```

```
62        /* Initialize Monotonic counter
63         *  Application-Specific method for storage (i.e. NVRAM)
64         */
65        ret_code = APP_IPB_store_counter( 0 );
```

6.2 Code example for secure boot

The following code demonstrates how to use and integrate the IP binding firmware. This code is integrated into the boot process to support the PUF initialization process and the decryption and validation of the protected firmware.

```
66    static __attribute__ ((section(".noinit.$RAM2"))) uint8_t
      sram_puf[BK_SRAM_PUF_SIZE_BYTES];
67
68      #typedef struct PUF_boot_params {
69          const ipb_header_t * const ipb_header;
70          const uint8_t      * const image_address;
71                uint8_t      * const image_target_address;
72                uint8_t      * const additional_keys_target_address;
73                uint32_t     * const version_counter;
74                uintptr_t    * const execution_address;
75      } PUF_boot_params_t;
76
77      PUF_boot_params_t   params;
78      Bool                bool_counter_updated;
79
80
81
82    APP_IPB_load_params( &params ) - Application dependent code to load boot params
83
84    APP_PUF_load_activation_code( activation_code ) - Application dependent code to
      read Activation Code (ex: Non-volatile memory)
85
86    APP_PUF_test_activation_code( activation_code ) - Application dependent code to
      validate activation code.  For example, the activation code might be stored in
      flash and thus it might have initial clear values (i.e. zero or all-ones).
87
88
89       /*  Load variables required by ipb_process_image()
90         *    Application-Specific method for storage (nvram, header, etc.)
91         *
92         *  Flash Storage:
93         *     IPB_header - stored in OTP by Provisioning Image
94         *     bound_image - stored in Flash; ipb_process_image output to SRAM
95         *     counter - stored in NOR Flash
96         *
97         *  Flashless Storage:
98         *     IPB_header - stored in OTP by Provisioning Image
99         *     bound_image - stored in SRAM at boot; ipb_process_image no output(XIP)
100        *     counter - stored in NOR Flash
```

```
101       */
102      ret_code = APP_IPB_load_params( &params );
103      if (IID_SUCCESS != ret_code)   break;
104
105      /*  Initialize SRAM dedicated for PUF */
106      ret_code = bk_init( sram_puf, BK_SRAM_PUF_SIZE_BYTES );
107      if (IID_SUCCESS != ret_code)   break;
108
109      /* Load Activation code
110       *  Application-Specific method for storage
111       */
112      ret_code = APP_PUF_load_activation_code( activation_code );
113      if (IID_SUCCESS != ret_code)   break;
114
115      /* Test if Activation Code is Valid
116       *  Application-Specific Method for Test (i.e. not all 0x0 or 0xFF)
117       *   Start Only, Enroll is during Enrollment
118       */
119      ret_code = APP_PUF_test_activation_code( activation_code );
120      if (IID_SUCCESS != ret_code)   break;
121
122      /* System has valid Activation Code;  Enroll would alter IPBindingHeader */
123      ret_code = bk_start( activation_code );
124      if (IID_SUCCESS != ret_code)   break;
125      ret_code = ipb_process_image(
126                          params->ipb_header,
127                          params->image_address,
128                          params->image_target_address,
129                          params->additional_keys_target_address,
130                          params->version_counter,
131                          &bool_counter_updated,
132                          params->execution_address );
133      if (IID_SUCCESS != ret_code)   break;
```

6.3 Secure update details

The process for a secure update follows a similar procedure as the secure boot. Encrypted firmware images are created in an identical manner as the initial image used for the device. The firmware distribution process must also create and distribute a new IP binding header for each device authorized to use the firmware. Once the protected firmware bundle and binding header are delivered to a device, the bundle is decrypted in the same manner as the initial image. When the image security options require roll-back protection, the target device must maintain a version number in non-volatile memory. The stored version number is checked when roll-back protection is required and only loads the new image when it contains a larger version number. On acceptance of the new image, the device must update its stored reference counter.

```
134      /* Store Monotonic counter
135       *  Application-Specific method for storage (i.e. NVRAM)
136       */
137      if( counter_updated == true )
138          ret_code = APP_IPB_store_counter( params->version_counter );
```

6.4 Additional protected information retrieval

The protected image bundle may include additional protected secrets. These secrets may be device unique configuration information or secret keys used for applications. This block of information is first in the image data and is of size *additional_keys_size* as provided by the header information.

7. Conclusion

This application note describes techniques to protect embedded applications using the PUF technology available on the NXP LPC54S0xx MCU family.

The physically unclonable keys derived from the PUF provide a strong root-of-trust for the identification of devices and the secure distribution of information to and from the device. IP binding uses device unique public keys as a foundation for the protection of firmware and configuration information. The PUF based root of trust may also be applied to other cryptographic services within a device. PUF derived keys may also be used to securely derive device unique keys for protocols like TLS, IPsec a SSH.

NXP has licensed the PUF technology from Intrinsic-ID. Addition information on the application of PUF technologies and IP binding may be obtained from Intrinsic-ID by accessing their website at https://www.intrinsic-id.com/ or by emailing info@intrinsic-id.com

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

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. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

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 and its suppliers accept 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 competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

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.

8.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

8.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

8.5 Trademarks

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

<Name> — is a trademark of NXP B.V.

9. List of figures

10. Contents

For more information, visit: http://www.nxp.com

Date of release: 06 November 2018
Document identifier: AN12291