AN12292

Secure Storage with SRAM PUF on NXP LPC54S0xx

Rev. 1.0 — 06 November 2018 Application note

Document information

Info	Content
Keywords	LPC54S0xx,PUF, Security
Abstract	This application note describes how the NXP LPC54S0xx family of ARM Cortex-M4 based microcontrollers can be used to develop secure embedded applications that provides strong protection for stored information.

Revision history

Rev	Date	Description
1.0	1106 2018	Initial version

Contact information

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

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

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2018. All rights reserved.

1. Introduction

The LPC54S0xx is a family of ARM© Cortex-M4 based microcontrollers for embedded applications featuring a rich peripheral set with very low power consumption and enhanced security features. This family of microcontrollers include a Physically Unclonable Function (PUF) controller based on SRAM that enables the secure generation of a unique device fingerprint and device-unique cryptographic keys. The SRAM PUF mechanism is tightly integrated into the LPC54S0xx family enabling keys from the PUF to be directly used by the device's internal AES-256 encryption engine. The unique and unclonable keys provide significant security benefits over other means of key injection or storage. The PUF keys reduce the threat of *break once repeat everywhere* attacks by basing the foundation of the security on device-unique unclonable keys.

This application note describes how the NXP LPC54S0xx family of ARM Cortex-M4 based microcontrollers can be used to develop secure embedded applications that provide very strong protection for stored information using SRAM PUF. The encryption may be used to protect both data and the boot image from external SPIFI, EMC, and SPI NOR flash devices or downloaded via serial slave interface (UART, SPI, I^2C).

1.1 SRAM based PUF key derivation

Due to deep submicron manufacturing process variations, every transistor in an Integrated Circuit (IC) has slightly different physical properties. These lead to small but measurable differences in terms of electronic properties like transistor threshold voltages and gain factor. Since these process variations are not fully controllable during manufacturing, these physical device properties cannot be copied or cloned.

It turns out that every SRAM cell has its own preferred state every time the SRAM is powered resulting from the random differences in the threshold voltages. This preference is independent from the preference of the neighboring cells and independent of the location of the cell on the chip or on the wafer.

Hence an SRAM region yields a unique and random pattern of 0's and 1's. This pattern can be called an SRAM fingerprint, since it is unique per SRAM and hence per chip. It can be used as a PUF.

Keys that are derived from the SRAM PUF are not stored *on the chip* but they are extracted *from the chip*, only when they are needed. In this way they are only present in the chip during a very short time window. When the SRAM is not powered there is no key present on the chip making the solution very secure from reverse engineering and key extraction.

1.2 LPC54S0xx SRAM PUF features

The SRAM PUF hardware constructs a 256-bit strength device-unique root key using the digital fingerprint of a device derived from uninitialized SRAM and error correction data called the Activation Code (AC). The Activation Code is generated during an *enrollment* process which takes place during the provisioning/personalization of the device at the manufacturer; see Section 2.1 – Device provisioning.

All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

The LPC54S0xx PUF hardware supports:

1. Generation, storage and reconstruction of device-unique keys based on PUF.
2. Routing of a PUF generated key to the AES encryption engine to support the encryption and authentication of external flash during device boot.
3. Secure storage and reconstruction of user keys with key sizes from 64-bits to 4096-bits.

Each generated PUF key is assigned a 4-bit index value to identify its usage. Keys with non-zero index are available through a register interface.

Keys that are assigned index value zero are output through a dedicated bus to the AES engine such that they cannot be accessed by software. The MCU may be configured to boot directly from external flash that has been protected by the PUF generated index zero key.

The PUF hardware supports the protection of additional application secrets using Key Codes (KC). A Key Code (KC) is a package of data that is encrypted by a PUF generated key. The encryption allows the confidential information to be secure stored outside of a processor. While normally used for the secure storage of cryptographic keys a key code may be used to for any sensitive application data. The key code data may be between 64-bits and 4096-bits in size. The 4-bit index values are used to identify the keys used for the encryption and decryption of the key codes.

The encrypted information stored in external NV memory cannot be transferred to any other device. The encryption process is unique to a specific processor and the encrypted information will not decrypt correctly if moved to another processor. The encryption process includes integrity checks that also prevent any modification of the information outside of the protected confines of the MCU.

1.3 Scope

This application note covers the LPC54S0xx use of SRAM PUF to:

- Protect the device firmware when stored on external flash devices.
- Create and use device-unique keys and identifiers.
- Protect provided keying material that may be used by applications.
- Ensure overall system security when using SRAM PUF.

2. Using SRAM PUF hardware

The process flow to use the SRAM PUF hardware on LPC54S0xx provides benefits throughout the lifecycle of a product. The use of the SRAM PUF hardware is described for:

- Device provisioning during OEM manufacturing.
- Secure boot from encrypted flash.
- Derivation of device-unique keys for use by applications.
- Protection of application secrets during device operation.

The device provisioning process initializes the security of the device and establishes a unique identifier for each device. The provisioning process stores an encrypted version of the firmware in external NV memory.

All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

On each boot, the decryption of the firmware uses a PUF device key, index zero key, to decrypt the firmware into *on chip* RAM. In operation, the firmware is never available unencrypted external to the device.

The PUF key generation may be used to create additional device unique keys for use by applications. The keys may be symmetric secret keys or asymmetric public/private key pairs. For example, the application software may use the key generation interfaces to create an elliptic curve public key pair for use by the TLS protocol. It enables the strong storage of per-device keys that may be used to remotely authenticate a device.

A system may have additional secrets that need to be configured into the device. Such secrets may use a PUF generated key to locally encrypt and store this information. This *wrapped* information may be securely stored in external NV memory since it is encrypted.

2.1 Device provisioning

A product is personalized in the OEM facility using a manufacturing tool that is used to test and program the device. 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 extraction.
- Optional extraction of a Unique Device Identifier (UDI) and additional device-unique keys for the protection of other data or secrets.
- Writing the PUF Activation Code to NV memory.
- At the end of the device provisioning process all debug and boundary interfaces must be disabled.

The enrollment image is specific to an MCU type and may include application specific testing and initialization. The PUF enrollment should be integrated with the existing enrollment image software.

The PUF initialization generates secret keys that never leave the device. The first key, index zero key, is dedicated to encryption and decryption of external NV memory and used during the boot process.

2.2 SRAM PUF hardware and the AES engine

2.2.1 Usage for secure boot

LPC54S0xx has no internal flash for code and data storage. The application images must reside in external devices like quad SPI and parallel flash memory. The AES engine may be configured to decrypt the boot image from external flash (SPI, QSPI, or parallel flash) or the serial ports (SPI, I^2C, UART). The device-unique PUF key makes any externally stored encrypted information only useable by the device on which the PUF has been enrolled.

The key with index 0 created by SRAM PUF may be directly used by the LPC54S0xx's AES engine. The use of PUF, as compared to OTP, provides considerable benefits for system security. The unclonable PUF keys are unique to a device and guarantees that encrypted information can only be used with that one device.

All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

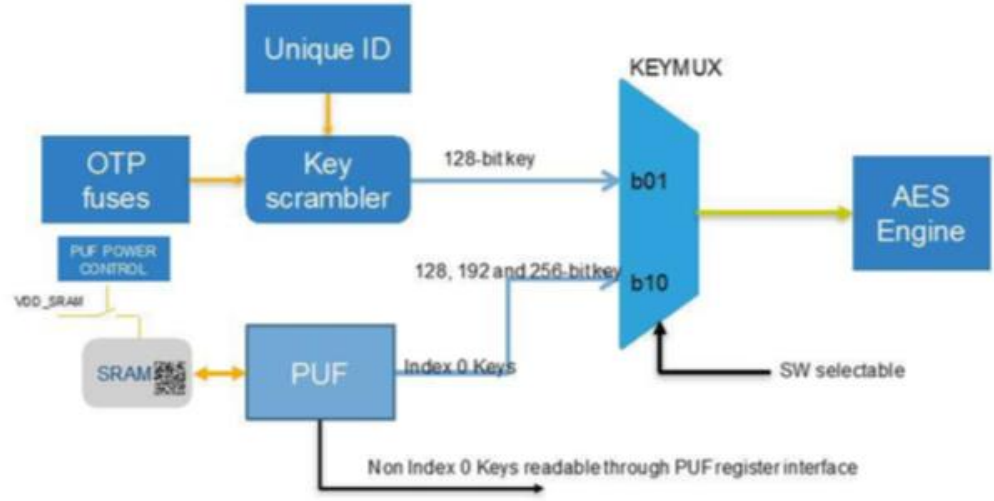

Fig 1. SRAM PUF enables the encryption of information with a device-specific unclonable key

The PUF hardware can be used to generate keys and key identifiers for applications. These keys are never stored and are device-unique and unclonable. The keys may be from 64-bits to 4096-bits in size and may be used by the application as symmetric or asymmetric secrets.

2.2.2 Derivation of device unique keys for applications

The PUF secrets may be used by applications.

2.2.3 Protection of user secrets

The PUF hardware supports the protection of user secrets. The secrets may be confidential configuration information or secret keys required for applications. This secret information would normally be provided at the factory part of the initial device provisioning. This protection may also be used in an operation device to protect selected secrets (for example, user configured passwords). The secrets are encrypted, and integrity protected in a format called a Key Code (KC). The Key Code (KC) format supports the protection of arbitrary user provided information. This information may be from 64-bits to 4096-bits in size. The device-unique key used to wrap these secrets are not readable by software, so the key codes may be safely stored on external non-volatile memory. The protected information must be word aligned.

3. Example software snippets

This section contains examples of code supporting the integration of PUF based security features on the LPC54S0xx to create secure applications.

3.1 Provisioning example code

The following example code should be integrated into the manufacturing provisioning process. Code examples are provided for:

All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

- The manufacturing time provisioning of PUF.
- The manufacturing time configuration of the LPC54S0xx AES engine to use a PUF generated key and encrypt the installed image.
- The manufacturing time configuration to lock down the debug and boundary scan interfaces.

This section provides high-level code that can be used as starting point for the development of the driver code. The example code uses status polling to control the flow.

Note:

1. The status polling method is used for clarity. For more efficient operation, an interrupt-driven architecture is recommended.
2. It is assumed that the key is going directly to the secure part of the design; see Fig 1. Therefore, no key output register is defined.

After PUF is reset (with or without a power cycle of SRAM) it needs time to initialize (indicated by busy asserted). It is assumed that the system waits for initialization to be finished before it starts issuing commands. The function puf_waitForInit can be used for this.

Note: This code does not include controls for powering on and off the SRAM. It is assumed that it is done in other parts of the system.

Glossary

CTRL	PUF control register
STAT	PUF status register
ALLOW	PUF allow register
CODEINPUT	PUF code input register
CODEOUTPUT	PUF code output register
ENROLL	PUF control register bit1 – begin enroll operation
START	PUF control register bit2 – begin start operation
BUSY	PUF status register bit0 - indicates that operation is in progress
SUCCESS	PUF status register bit1 - last operation was successful
ERROR	PUF status register bit2 – PUF is in ERROR state; no operations allowed.
CODEOUTAVAIL	PUF status register bit7 – next part of AC is available.
ALLOWENROLL	PUF allow register bit0 – enroll operation is allowed.
ALLOWSTART	PUF Allow register bit1 – start operation is allowed.
Initialize (target)	Empties the target data structure.
get_data (data, source)	Retrieves the next data word from the source structure puts it in data and removes it from the head.
append_data (data, target)	Appends the data in data to the end of target.

All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

3.1.1 PUF provisioning code examples

The following code should be executed during device provisioning by the enrollment image to obtain:

- The unique device id.
- The device Activation Code (AC).

The Activation Code (AC) must be subsequently stored in non-volatile memory for later use.

Note: The Activation Code (AC) does not need to be stored in NVRAM, any addressable memory.

3.1.1.1 PUF provisioning code example for initialize

```
status PUF_waitForInit() {

// wait until initialization has finished
while (*STAT & BUSY != 0) {}

// check that initialization has passed
if (*STAT & SUCCESS == 0) {
return ERROR
}
return OK
}
```

3.1.1.2 PUF provisioning code example for enroll

```
/* output: ACdata - byte array for Activation Code storage */
status_t  PUF_Enroll(
PUF_Type *base,
uint8_t *activationCode,
size_t activationCodeSize)
{
// clear the ACdata storage
initialize(ACdata)

// check if Enroll is allowed
if (*ALLOW & ALLOWENROLL == 0) {
return NOT_ALLOWED
}

// Make sure that the module that receives the key is initialized
// and can accept the key
// begin Enroll
*CTRL = ENROLL

// wait till command is accepted
while (*STAT & (BUSY | ERROR) == 0) {
```


All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

```
}
// while busy read AC
while (*STAT & BUSY != 0) {
if (*STAT & CODEOUTAVAIL != 0) {
tempData = *CODEOUTPUT
append_data(tempData, ACdata)
}
// During this loop the key is transported to the
// receiving module using the interlocked interface
} // while
// check result
if (*STAT & SUCCESS == 0) {
return ERROR
}
return OK
}
```

3.1.1.3 PUF provisioning code example for start

```
/* input: activationCode - byte array containing Activation Code */

status_t  PUF_Start(
PUF_Type *base,
const uint8_t *activationCode,
size_t activationCodeSize)
{
// check if Start is allowed
if (*ALLOW & ALLOWSTART == 0) {
return NOT_ALLOWED
}

// Make sure that the module that receives the key is initialized
// and can accept the key
// begin Start
*CTRL = START

// wait till command is accepted
while (*STAT & (BUSY | ERROR) == 0) {
}

// while busy send AC, read key
while (*STAT & BUSY != 0) {
if (*STAT & CODEINREQ != 0) {
get_data(tempData, activationCode)
*CODEINPUT = tempData
}

// During this loop the the key is transported to the
// receiving module using the interlocked interface
} // while
```


All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

```
80
81    // check result
82    if (*STAT & SUCCESS == 0) {
83    return ERROR
84    }
85    return OK
86    }
```

3.2 AES engine usage of PUF

The AES engine on the LPC54S0xx may be configured to directly use keys from the PUF engine. The following code snippets demonstrate the routing and usage of the PUF keys to the AES engine for encryption and decryption.

3.2.1 AES engine configuration for PUF usage.

The AES engine must be configured to use PUF keys rather than OTP keys. The following code provides the details of this configuration.

It shows how to select the PUF key with index 0

```
87    #define PUF_INTRINSIC_KEY_SIZE 16
88    status_t result;
89    uint8_t keyCode[PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(PUF_INTRINSIC_KEY_SIZE)];
90
91    result = PUF_SetIntrinsicKey(PUF, kPUF_KeyIndex_00, PUF_INTRINSIC_KEY_SIZE,
      keyCode, sizeof(keyCode));
92    if(result != kStatus_PUF_Success)   return result;
93
94    /* Get Intrinsic Key */
95    result = PUF_GetHwKey(PUF, keyCode, sizeof(keyCode), kPUF_KeySlot0, rand());;
96    if(result != kStatus_PUF_Success)   return result;
```

3.2.2 AES engine using PUF derived key for encryption

The use of a PUF generated device unique key for AES encryption may be integrated with the boot process to protect firmware images.

PUF generated keys can be used with AES for create secure storage of data:
Set KEY as shown in Section 3.2.1 (AES engine configuration for PUF usage).

```
97    /* Applicatin_Specific iv is 96-bit unique value */
98    uint8_t explicit_iv[12] =
      {0x58,0x9C,0x84,0xD1,0x7D,0x1B,0x43,0xBE,0x57,0xCB,0xD6,0xD9};
99
100   /* Application Specific/
101   uint8_t aad[] =
102   { 0xDF, 0x76, 0xB5, 0x5A, 0x48, 0x8E, 0x68, 0xF8,
103     0xF9, 0xCB, 0x82, 0x95, 0xC9, 0x1A, 0xCF, 0xEB };
104
105   uint8_t tag[16];
106
```


All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

```
status = AES_EncryptTagGcm( APP_AES, clear_data,encrypt_data, *image_length,
                            explicit_iv, sizeof(explicit_iv),
                           aad, sizeof(aad),
                           tag, sizeof(tag) );
```

3.2.3 AES engine using PUF for decryption

The following code example describes the use of a PUF derived key to decrypt external storage.

Set KEY as shown in Section 3.2.1 (AES engine configuration for PUF usage)

```
/* iv is 96-bit unique value */
uint8_t explicit_iv[12] =
{0x58,0x9C,0x84,0xD1,0x7D,0x1B,0x43,0xBE,0x57,0xCB,0xD6,0xD9};

uint8_t aad[] =
{ 0xDF, 0x76, 0xB5, 0x5A, 0x48, 0x8E, 0x68, 0xF8,
  0xF9, 0xCB, 0x82, 0x95, 0xC9, 0x1A, 0xCF, 0xEB };

uint8_t tag[16];

status = AES_DecryptTagGcm( APP_AES, encrypt_image, clear_image,  *image_length,
                           explicit_iv, sizeof(explicit_iv),
                            aad, sizeof(aad),
                            tag, sizeof(tag) );
```

3.3 User key protection examples

The following code examples are used to encrypt or decrypt user secrets using a PUF generated device unique secret. This key wrapping and unwrapping may be used by applications to securely store secret information.

3.3.1 Wrapping (Set Key)

The following code snippet demonstrates how an application's secret information may be encrypted. The device unique encryption allows the key code that is created to be stored off the device in any non-volatile storage.

KC : Key Code

```
status SetIntrinsicKey(KCdata, KeyIndex, KeySize) {
// clear the KCdata storage initialize(KCdata)
// check if Set Key is allowed
if (*ALLOW & ALLOWSETKEY == 0) {
return NOT_ALLOWED
}

// program the key size and index
*KEYSIZE = KeySize >> 6 // convert to 64-bit blocks
```


All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

```
*KEYINDEX = KeyIndex

// begin Set Key
*CTRL = GENERATEKEY

// wait till command is accepted
while (*STAT & (BUSY | ERROR) == 0) {
}
// while busy read KC
while (*STAT & BUSY != 0) {
if (*STAT & CODEOUTAVAIL != 0) {
tempData = *CODEOUTPUT
append_data(tempData, KCdata)
}
} // while

// check result
if (*STAT & SUCCESS == 0) {
return ERROR
}
return OK
}


status SetUserKey(KCdata, KeyIndex, UKdata) {
// clear the KCdata storage
initialize(KCdata)

// check if Set Key is allowed
if (*ALLOW & ALLOWSETKEY == 0) {
return NOT_ALLOWED
}

// detect key size
KeySize = length_in_bits(UKdata)

// program the key size and index
*KEYSIZE = KeySize >> 6 // convert to 64-bit blocks
*KEYINDEX = KeyIndex

// begin Set Key
*CTRL = SETUSERKEY

// wait till command is accepted
while (*STAT & (BUSY | ERROR) == 0) {}

// while busy write UK and read KC
while (*STAT & BUSY != 0) {
if (*STAT & KEYINREQ != 0) {
get_data(tempData, UKdata)
*KEYINPUT = tempData
```


All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

```
182  }
183  if (*STAT & CODEOUTAVAIL != 0) {
184  tempData = *CODEOUTPUT
185  append_data(tempData, KCdata)
186  }
187  } // while
188  // check result
189  if (*STAT & SUCCESS == 0) {
190  return ERROR
191  }
192  return OK
193  }
```

3.3.2 Unwrapping (Get Key)

The following code snippet demonstrates how an applications secret information may be encrypted. The device unique encryption allows the key code that is created to be stored off the device in any non-volatile storage.

Example for Get Key (UnWrap**)**

```
194  status GetKey( KCdata, KeyIndex, KeyData) {
195  // clear the KeyData storage
196  initialize(KeyData)
197  // put unused value in KeyIndex
198  KeyIndex = 255
199
200  // check if Get Key is allowed
201  if (*ALLOW & ALLOWGETKEY == 0) {
202  return NOT_ALLOWED
203  }
204
205  // begin Get Key
206  *CTRL = GETKEY
207
208  // wait till command is accepted
209  while (*STAT & (BUSY | ERROR) == 0) {
210  }
211
212  // while busy send KC, read key
213  while (*STAT & BUSY != 0) {
214  if (*STAT & CODEINREQ != 0) {
215  get_data(tempData, KCdata)
216  *CODEINPUT = tempData
217  }
218  if (*STAT & KEYOUTAVAIL != 0) {
219  KeyIndex = *KEYOUTINDEX
220  tempData = *KEYOUTPUT
221  append_data(tempData, KeyData)
222  }
223  } // while
```


All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

```
224
225  // check result
226  if (*STAT & SUCCESS == 0) {
227  return ERROR
228  }
229  return OK
230  }
```

4. Conclusion

LPC54S0xx devices provide PUF functionality to generate and use device-unique cryptographic keys. The keys may be used to support the secure storage of firmware images or other user data in non-volatile memory off-chip. The keys are also available to support application authentication services. This application note provides usage descriptions and detailed code examples to integrate PUF in the LPC54S0xx.

All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

5. Legal information

5.1 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.

5.2 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 competent authorities.

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

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

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

5.3 Licenses

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

5.4 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>

5.5 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.

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2018. All rights reserved.

6. List of figures

All information provided in this document is subject to legal disclaimers.

© NXP Semiconductors N.V. 2018. All rights reserved.

7. Contents

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

© NXP Semiconductors N.V. 2018. **All rights reserved.**

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

Date of release: 06 November 2018
Document identifier: AN12292