AN13577

Building an I3C Sensor Network Using LPC553x/LPC55S3x

Rev. 1 — 04 May 2022 Application Note

Contents

1 Introduction

Some latest NXP MCUs (i.MX RT6xx, i.MX RT5xx, LPC553x/LPC55S3x) have integrated MIPI I3C controller to support faster, more power efficient, standardized sensor communication, while still maintaining backward compatible with legacy I2C. This major improvement is expected to be used in many mobile and Internet-of-Things application.

The I3C bus protocol supports:

- In-band command codes (Common Command Codes (CCC))
- Dynamic Address Assignment (DAA)
- In-band interrupts (IBI): interrupts can go from slave to master without extra wires that slave can take temporary control of the bus and report certain event to the master.
- Multi-master / multi-drop
- Hot-Join
- Compatible with legacy I2C but without clock stretching

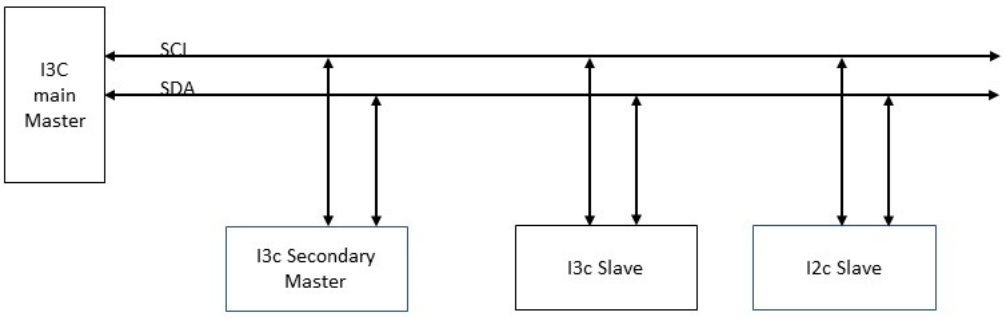

Figure 1. MIPI I3C Standardize Sensor Connectivity

This application note describes how to use the LPC553x/LPC55S3x I3C controller as a master, take full advantage of some key features of the I3C such as DAA and IBI, how to build a sensor network, and communicate with these slave sensors on the network.

It starts with a general description of the basic functionality of I3C and then describes how to use NXP SDK APIs to communicate with I3C slave sensors through CCC commands, assign slave addresses dynamically, and exchange data with these sensors directly, finally, it gives some examples how to use callbacks provided by SDK to handle In-band interrupts from these slave sensors.

The S/W package is tested on the LPC553x/LPC55S3x EVK board communicating with an onboard InvenSense ICM42688 motion-tracking sensor and connected with two P3T11xx temperature sensors on an NXP temperature sensor module board externally through the I3C bus.

2 MIPI I3C Basic

This section gives basic information about the features of the LPC553x/LPC55S3x MCU I3C controller and Common Command Code (CCC).

2.1 Features

LPC553x/LPC55S3x MCU I3C controller supports:

- 2-wire multi-drop bus capable of 12 MHz clock speeds, with up to 11 devices.
- Uses standard pads (I2C uses special pads) with 4 mA drive.
- Slave does not require a static address and its address can be dynamically assigned by the master. However, slaves may have an I2C static address assigned at start-up, so that the slave can operate naturally on an I2C-bus.
- In-Band interrupts (IBI), which allow slaves to notify a master without requiring an external pin.

2.2 Common Command Code (CCC)

MIPI I3C master can use Common Command Code (CCC) broadcast commands to control multiple I3C devices at once or direct commands to control each individual device, while I3C slave devices listen to and support a number of CCC commands to control and report certain device features and status, for example, bus reset, report device provisional ID (PID), bus characteristics register (BCR), device characteristics register (DCR), enabled/disable slave events such as in-band interrupts or renew the slave dynamic address.

There are many CCC commands in the MIPI I3C spec. Some are broadcast and some are direct. In order to support Dynamic Address Assignment (DAA), only a few most important commands related to this application note are listed here. These CCC commands are:

- RSTDAA Reset dynamic addresses on all the slave devices on the bus.
- ENTDAA Force all I3C slave devices (if they do not already have a dynamic address assigned) to go into address assignment mode.
- SETDASA Set dynamic address from static address. Note: this SETDASA CCC command support is not required by all I3C slave devices.
- SETNEWDA Set new dynamic address.
- ENEC/DISEC Enable/Disable slave events.

For more information about the CCC commands, check http://mipi.org.

3 Setup an I3C Sensor Network

In this sensor network example, three I3C slave sensors are used:

- InverSense ICM42688 Motion Tracking sensor
- NXP P3T1175 Temperature Sensor
- NXP P3T1108 Temperature Sensor

On LPC553x/LPC55S3x IOCON, these three pins are used for I3C communication.

`PIO0_9 I3C_SCL` (I3C clock)

`PIO0_24 I3C_SDA` (I3C data)

`PIO0_28 I3C_PUR` (I3C pull-up control on SDA line).

On the LPC553x/LPC55S3x EVK board, these pins are connected to the onboard motion tracking sensor ICM42688P directly. Externally, the EVK board also connects to the NXP P3T11xx temperature module where both P3T1175 and P3T1108 are on the same I3C bus. The connection between the two boards is shown in the table below:

Table 1. The connection between LPC553x/LPC55S3x EVK board and NXP P3T11xx temperature module

Pin function	LPC553x/LPC55S3x EVK board	NXP P3T11xx Temperature Module
I3C_SCL	J9-20	J13-1
I3C_SDA	J9-18	J13-2
GND	J9-14	J13-4
Power 3.3V	J10-8	J4-4
GND	J10-12	J4-6

NOTE

A known issue was described at the end of this application note explaining why the onboard Arduino connector on the NXP P3T11xx Temperature Module cannot be used.

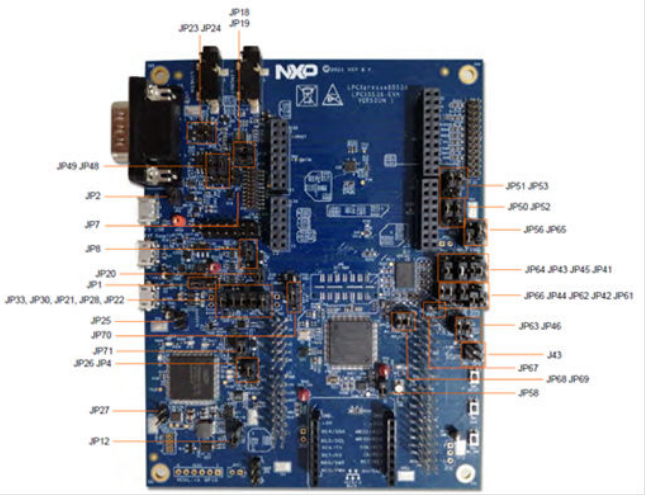

Figure 2. LPC55S3x EVK board

For more information on the jumper setting, refer to LPC553x/LPC55S3x-EVK Board User Manual.

P3T11xx Temperature Module Jumper setting (Using default setting from the schematics if not mentioned below):

JP2 1-2 close I3C SCL

JP3 1-2 close I3C SDA

NOTE

Since JP1 2-3 close that temperature module uses 3.3 V power supply as default, JP1, voltage selector, on LPC553x/LPC55S3x EVK board, 1-2 should be closed (3.3 V) accordingly.

Table 2. Data based on the P3T11xx module default jumper setting

NXP Temperature Sensor	Slave Static Address
P3T1175	0x4C
P3T1108	0x48

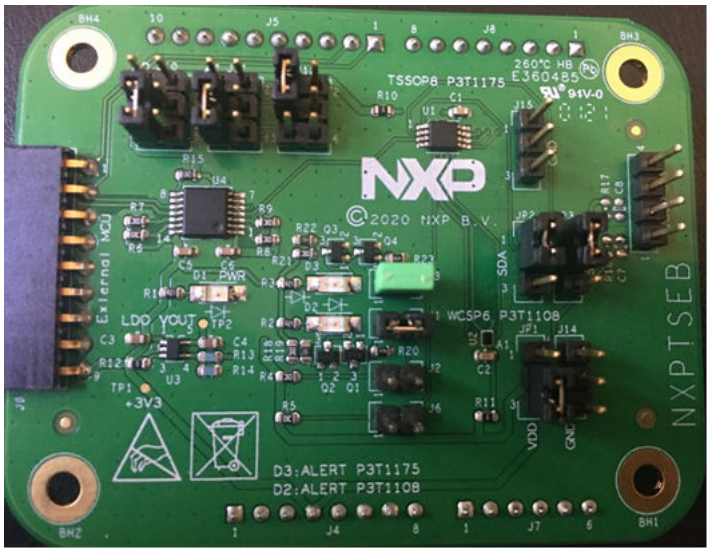

Figure 3. NXP P3T11xx Temperature Module

4 I3C DAA and sensor network discovery

This section describes the steps of usage the existing I3C driver APIs from SDK to configure the I3C controller as the I3C master, to use CCC commands and assign dynamic addresses to these sensors on the network, to insert callbacks to listen to the IBI interrupts from the slaves, and finally, to read sensor data continuously and response to IBI interrupts.

Along with the application note, there is the complete source code for the I3C sensor network example compiled in Keil, IAR, and MCUXpresso IDEs. Two zip files are included, one for Keil and IAR and the other is for MCUXpresso. The Keil are IAR projects are in the directory `\boards\lpcxpresso55s36\driver_examples\i3c\i3c_sensor_network`.

4.1 I3C master initialization and DAA

Our I3C master initialization in this example, `i3c_sensor_network.c`, includes the following:

- Depending on the system clock setting, you might need a clock divider for the I3C clock.
  - The master needs an accurate clock, capable of the frequency that is a multiple of the I3C clock. The higher baud rate you need to achieve, the higher clock frequency you must consider for the I3C controller. For example, this example sets a 50 MHz clock to the I3C controller which ideally supports both 2.5 MHz and 10 MHz I3C clocks.
  - Due to various device characteristics of the I3C slave connected to the I3C bus, a slower push-pull baud rate must be used before DAA. This example sets PPBAUD to 2 Mbit/s and ODBAUD to 400 Kbps initially. Once the address assignment is done, a higher push-pull baud rate can be used for data communication. The SDK I3C driver `API I3C_MasterSetBaudRate()` can be used to set a higher baud rate for sensor data reading.
  - The divider for the PPBAUD is a 4-bit field in MCONFIG register. It is not possible to set a very fast I3C clock with very low PPBAUD. If some devices do not support higher PPBAUD during DAA, it is possible to set a slower I3C clock along with lower PPBAUD before DAA, then set a faster I3C clock along with higher PPBAUD.
- Create a device address table for all sensors on the I3C bus, make all these sensors listen to the broadcast commands from the master.
- Create an I3C secondary master as the first node on the bus and assign a dynamic address to this secondary master with the NXP vendor ID and some pseudodevice characteristics.
- The `I3C_BusMasterCreate()` API is the single most important engine for creating a master structure and assigning dynamic addresses to the devices on the bus.

The master structure creation must include not only all the resource capabilities of the masters but also consider all the possible devices, I3C or I2C, connected to the bus, therefore, all the baud rates, I2C baud rate, I3C push-pull baud rate, and I3C open-drain baud rate must be considered and configured here.

- Issue first the RSTDAA CCC broadcast command to all slaves on the bus to ensure a clean start.
- Issue the DISEC CCC broadcast command to all the slaves to disable slave events during DAA to prevent events such as slave in-band interrupts.
- The master operates with a built-in Enter Dynamic Address Assignment (ENTDAA) mechanism to simplify assignment of DAs to slaves. This built-in mechanism also takes care of those slave devices that do not support the SETDASA CCC command. In the LPC553x/LPC55S3x User Manual, Section “Assigning dynamic addresses to I3C devices” describes the DAA mechanism in detail. In SDK for LPC553x/LPC55S3x, `API I3C_BusMasterDoDAA()` in the I3C driver does exactly that to assign dynamic address to slave devices.
- During the DAA process, the master also reads the unique device information and capabilities of the slave, such as part number, Vendor ID(VID), BCR, DCR registers and saves them in the device information data structure for each device.
- Issue the ENEC CCC broadcast command to all the slaves to re-enable slave events after DAA finishes.

Once DAA is done, inside the `i3c_sensor_network.c` file, the main routine, `demo_i3cBus.i3cDevList` contains a list of the devices found on the I3C bus. The unique device information data is used as the identifier of all the devices on the list.

The device handle is extracted based on the unique part number and device VID and used for peer-to-peer communication between the master and the sensor device.

Table 3. Sensor-Specific information before and after DAA

Device Name	Static Address, based on external pull-up/down	Dynamic Address after DAA	Part Number after DAA	VID after DAA	Part Number References
ICM42688	0x69	0xB	N/A	0x235	
P3T1175	0x4C	0xA	0x152A	0x11B	P3T1175 Datasheet Section 7.4.2
P3T1108	0x48	0x9	0x1529	0x11B	P3T1108 Datasheet Section 7.4.2

MIPI Alliance Manufacturer ID or vendor ID information can be found on the https://mid.mipi.org/ website. For example, All NXP I3C devices have the same Vendor ID 0x11B.

NOTE

Static address is not required during DAA. Static addresses are listed in the S/W example as they are useful if you want to use a static address to communicate with the sensor directly regardless communication type I2C or I3C. SDK has several APIs to do that, `I3C_MasterTransferBlocking()` and `I3C_MasterTransferNonBlocking()`.

4.2 I3C In-Band Interrupt (IBI)

In-Band Interrupt (IBI) is a method whereby a slave device can emit its address into the arbitrated Address header on the I3C Bus to notify the controller of the interrupt.

The slave device can take temporary control of the bus and report a certain event to the master. IBI is over the serial bus rather than requiring separate pins. It is a major improvement over I2C, where an interrupt from the slave device requires an additional pin.

Every I3C slave device must have a read-only Bus Characterization Register (BCR) and a Device Characterization Register (DCR). These register information can be read using one of the CCC commands. Not every I3C slave device can support IBI though, bit 1 of the BCR register indicates whether IBI request is capable or not.

During DAA, the master device uses the built-in CCC command mechanism to collect device information and retrieve BCR and DCR register information from the slave device.

4.2.1 Sensor specific IBI configuration

IBIs are various from sensor to sensor and must be configured before they can be used.

In this example, on ICM42688 sensor, the tap detect interrupt is routed to IBI.

On NXP P3T1175 and P3T1198 sensor, the temperature alert interrupt is routed to IBI. The alert is a transition above and then below upper threshold or lower threshold.

Information on how to configure an interrupt and route it to the IBI is device-specific. The generic device register access APIs and device-specific IBI configuration S/W examples are included in `fsl_icm42688p.c/.h and fsl_p3t11xx.c/.h` files. For more device-specific information, refer to the data sheets of these sensors.

4.2.2 Sensor IBI registration

SDK provides an API, `I3C_BusMasterRegisterDevIBI()`, to register device IBI to the master.

Device IBI registration and a corresponding callback routine are needed to handle IBI interrupt when generated.

The example is provided in this demo software package.

4.3 I3C Sensor Network Demo Operation

The section describes how this sensor network demo works in details.

Due to the design limitation on LPC553x/LPC55S3x that the SWDIO pin, PIO0_9, is shared with the I3C_SCL pin, onboard MCULink debug probe cannot be used for debugging or programming once this I3C demo software is programmed and is running. To disable SWD while keeping the MCULink Virtual Communication(VCOM) port running, JP27 should be closed once the programming is done.

To use the MCULink VCOM feature from J1, JP12 must be closed (default) and JP26 must be open (default).

Once the I3C demo code is running, MCULink cannot be used to program the flash anymore. The NXP blhost utility must be used to erase the whole flash first if you want to make some change and rerun this demo or run another application. Jumper setting change is required from programming to code execution and back to programming.

4.3.1 Jumper setting and operation for programming

To ensure a clean start, erase the whole flash first. Here is the sequence of the operation:

1. Disconnect COMx from “Tera Term” used for debugging printout if any.
2. Change the boot mode on jumper J43, 1-2 Open and 3-4 closed: ISP0 high and ISP1 low, target LPC553x/LPC55S3x MCU going to ISP boot mode.
3. Close JP27 to force MCULink VCOM to be used as the UART ISP COM port.
4. Power the board through J1 (MCULink).
5. Press the **Reset** button to ensure LPC553x/LPC55S3x MCU is in the ISP boot mode.
6. Open a command window to run blhost: `blhost -p COMx -- flash-erase-all`, where COMx is the MCULink VCOM port shown on the Windows Device Manager.

   Once the flash has been erased successfully, message should show on Command Prompt:

```
C:\Data\blhost_2.6.6\blhost_2.6.6\bin\win>blhost -p COM4 -- flash-erase-all
Ping responded in 1 attempt(s)
Inject command 'flash-erase-all'
Successful generic response to command 'flash-erase-all'
Response status = 0 (0x0) Success.
```

Figure 4. Blhost Command Response

7. Once the flash is erased, open JP27. Now, MCULink debug probe must be active and ready to program the flash.
8. Recompile and program sensor network demo S/W onto the internal flash.

NOTE

The debugger must also work until the breakpoint hits where PIO0_9 is reconfigured as the I3C_SLK pin. Exit debugger once the flash programming is done.

4.3.2 Jumper setting and operation for code execution

For the demo free running, follow the steps below:

1. Close JP27 so that demo S/W printout uses the MCULink VCOM port.
2. Change the boot mode on J43, close 1-2 and 3-4: Both ISP0 and ISP1 are low. LPC553x/LPC55S3x Target MCU forces to internal flash boot.
3. Power the board through J1.
4. Re-enable UART connection on “Tera Term” for display.

If you have made code change and rebuilt in I3C sensor network demo, go back to the steps mentioned in Section 4.3.1 again.

4.3.3 Demo operation and generating IBI

Once this demo is up and running, it starts displaying messages on “Tera Term” over the MCULink VCOM port at 115200 bits/s (8N1) including the dynamic address of each sensor on the network, sensor data reading from each sensor continuously. The demo is also monitoring IBI interrupt from the slave if any.

NOTE

MCULink VCOM port is shared for both “blhost” utility and demo message display. If blhost is used to erase the flash, disconnect the “Tera Term” COMx port first, once it is done, reconnect before running the demo.

Once an IBI interrupt is generated, the demo exits, a reset to the board is required to restart.

The IBI interrupt from ICM42688 is a tap detector. The sensitivity of the tap detection on each ICM42688 may vary, any tap on the component or the board may generate an IBI interrupt.

The IBI interrupt from NXP P3T11xx sensor is an alert signal when temperature reading is a transition above and then below the upper threshold or lower threshold.

On the temperature sensors, the upper threshold is set to 30 degree C and lower threshold is set to 20 degree C, defined in the `p3t11xx.h` file. Depending on the working environment, it is necessary to adjust the thresholds accordingly or unwanted IBI may generate accidentally. The `fsl_p3t11xx.c` has the example how to program the upper and lower limit of the configuration registers of the sensor.

The display of the message can be seen below:

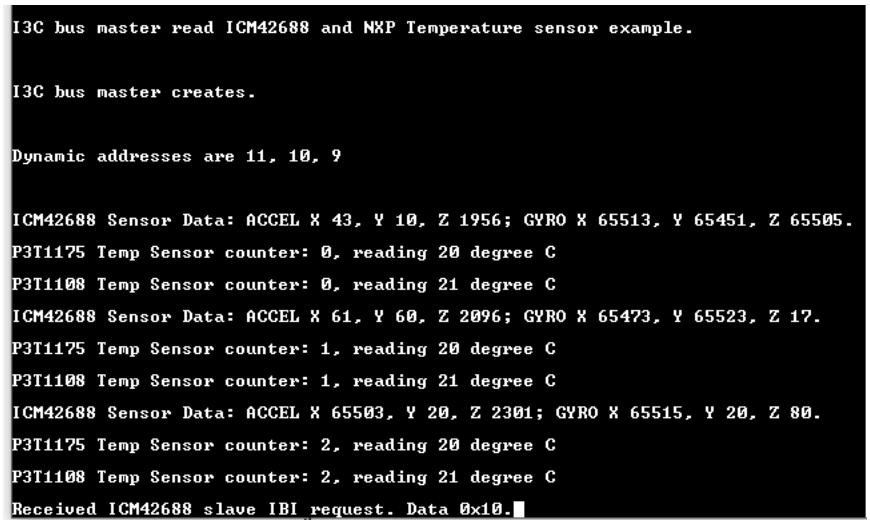

Figure 5. Sensor Network Demo Message Display

5 Known issues

RX data corruption was found on the master side due to RXFIFO ringing when a level shifter was added between the master and the slave. Many I3C slave devices can support wide range supply voltage, VCC, for example, P3T1108 is 1.4 V to 3.6 V and ICM42688 is 1.61 V to 3.6 V. Thus, level-shifter is not required between the master and most of the I3C slave parts.

Some related issues were found on NXP P3T11xx Temperature Sensor Modules. Once the onboard level-shifter, NTS0304(U4), is used Dynamic Address Assignment fails. Thus, the Arduino connector on the temperature sensor module cannot be used. Similar issues have also been found on the MIMXRT6XX EVK Rev. E board. In this case, data corruption happens due to RX FIFO ringing of the I3C controller once the NXP NTS0102 level shifter is used.

Depending on the hardware environment, if you have high capacitance on the board such as longer trace due to layout or off board wiring, the addition of the level shifter could add another level of complexity.

Changing the SLEW bit (bit 6) of the IOCON register from fast mode to standard mode on `I3C_SCL` and `I3C_SDA` pins may be helpful.

If level-shifter is required, NXP recently launched P3A9606, a bidirectional I3C/I2C voltage-level translator, which is targeted for a I3C application.

6 References

Here is the list of the references related to this demo:

- LPC553x/LPC55S3x-EVK Board User Manual
- LPC553x/LPC55S3x-EVK Schematics
- P3T1108UK-P3T1175DP Temperature Sensor Daughter Card Schematics
- ICM-42688-P 6-Axis MEMS Motion Tracking Device Datasheet
- P3T1108UK I3C Digital Temperature Sensor Datasheet
- P3T1175 I3C Digital Temperature Sensor Datasheet
- MIPI Alliance I3C Specification

7 Conclusion

LPC553x/LPC55S3x has some enhanced features to support sensor communication. Its I3C controller can be used for numerous mobile and IoT applications.

The I3C controller can do many little things I2C cannot provide: DAA and IBI are two main advantages. This example has demonstrated how to take advantage of the existing I3C APIs provided by our SDK and build a real-world sensor application.

8 Revision history

Table 4. Revision history

Revision number	Date	Substantive changes
1	04 May 2022	Changes made: in the title, “LPC55S3x” is replaced with “LPC553x/LPC55S3x”; in the document:“LPC55S3x” is replaced with “LPC553x/LPC55S3x”, “LPC55S36” is replaced with “LPC553x/LPC55S3x”
0	18 February 2022	Initial release

Legal information

Definitions

Draft — A draft status on a document indicates that 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 in a draft version of a document and shall have no liability for the consequences of use of such information.

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.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

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.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

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

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

Trademarks

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

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Airfast — is a trademark of NXP B.V.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Cadence — the Cadence logo, and the other Cadence marks found at www.cadence.com/go/trademarks are trademarks or registered trademarks of Cadence Design Systems, Inc. All rights reserved worldwide.

CodeWarrior — is a trademark of NXP B.V.

ColdFire — is a trademark of NXP B.V.

ColdFire+ — is a trademark of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

EdgeScale — is a trademark of NXP B.V.

EdgeVerse — is a trademark of NXP B.V.

eIQ — is a trademark of NXP B.V.

FeliCa — is a trademark of Sony Corporation.

Freescale — is a trademark of NXP B.V.

HITAG — is a trademark of NXP B.V.

ICODE and I-CODE — are trademarks of NXP B.V.

Immersiv3D — is a trademark of NXP B.V.

I2C-bus — logo is a trademark of NXP B.V.

Kinetis — is a trademark of NXP B.V.

Layerscape — is a trademark of NXP B.V.

Mantis — is a trademark of NXP B.V.

MIFARE — is a trademark of NXP B.V.

MOBILEGT — is a trademark of NXP B.V.

NTAG — is a trademark of NXP B.V.

Processor Expert — is a trademark of NXP B.V.

QorIQ — is a trademark of NXP B.V.

SafeAssure — is a trademark of NXP B.V.

SafeAssure — logo is a trademark of NXP B.V.

StarCore — is a trademark of NXP B.V.

Synopsys — Portions Copyright © 2021 Synopsys, Inc. Used with permission. All rights reserved.

Tower — is a trademark of NXP B.V.

UCODE — is a trademark of NXP B.V.

VortiQa — is a trademark of NXP B.V.

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

Date of release: 04 May 2022
Document identifier: AN13577