AN15087

Implementing Three I2C Target Interfaces Using SmartDMA on the MCX N947

Rev. 1.0 — 9 July 2026 Application note

Document information

Information	Content
Keywords	AN15087, SmartDMA, MCX N947, I2C, FRDM-MCXN947
Abstract	This application note demonstrates the implementation of three simultaneous I2C target interfaces on the MCX N947 using SmartDMA.

1 Introduction

This demo implements three virtual I2C target ports on MCX N947 using SmartDMA-assisted GPIO-style signaling, without dedicating three hardware LPI2C target peripherals. The key design points are:

- SmartDMA firmware handles timing-sensitive low-level bus state detection and byte handoff.
- The Arm application uses an SDK-style LPI2C target callback model for address, RX, TX, and completion events.
- Three virtual target channels are enabled from a single SmartDMA service instance. Each channel can be mapped to separate SmartDMA-capable pins for independent I2C connections.

The demo uses the FRDM-MCXN947 board as the SmartDMA I2C target. Three more FRDM-MCXN947 boards serve as simple LPI2C controller testers. Each board repeatedly writes a known 32-byte frame to address 0x7E, reads the target response, and compares the payload bytes.

2 Features

The following features are supported:

- Three virtual I2C targets: LPI2C0, LPI2C1, and LPI2C2 base pointers are used as virtual instance identifiers. The underlying bus engine is SmartDMA firmware rather than the hardware LPI2C target block; the physical bus signaling is implemented through SmartDMA PIO bit-banging.
- 7-bit target address configuration: address0 is mapped to the address register of each channel; only the lower 7 bits are used for matching.
- Start/Stop detection: The SmartDMA state machine detects I2C Start and Stop conditions and notifies the Arm core through interrupts.
- Controller write/target receive: When the controller writes, SmartDMA receives the bytes, returns ACK, and delivers the data to the Arm-side buffer.
- Controller read/target transmit: When the controller reads, the Arm application provides a TX buffer, and SmartDMA transmits data byte by byte while detecting controller ACK/NACK.
- Clock stretching: SCL is held low during the address phase, RX data handoff, and TX data loading while waiting for Arm-side processing.
- ACK/NACK control: Supports ACKSTALL mode. During the address phase, a callback to the Arm application is triggered, allowing the application to call SMARTDMA_LPI2C_SlaveSetAddressAck() to determine ACK or NACK.
- Callback events: ReceiveEvent and TransmitEvent are supported, with optional support for AddressMatchEvent, TransmitAckEvent, and CompletionEvent.
- SMBus Alert Response Address (ARA): Supports SMBus ARA 0x0C; the response byte is the own address of the device shifted left by one bit.
- Transfer completion statistics: A completion event is triggered after a Stop condition, carrying transferredCount and status information.
- SDK-style wrapper API: Provides TargetGetDefaultConfig, TargetInit, TargetDeinit, TransferCreateHandle, TransferNonBlocking, and an IRQ handler.

3 Hardware connection

Figure 1 shows the hardware connection block diagram. The three controller boards connect to the target board via three independent I2C buses (I2C0, I2C1, and I2C2). All four boards share a common ground, and the three controllers share a common reset pin and power supply, which allows them to be reset simultaneously. The three virtual I2C targets are all implemented on a single FRDM-MCXN947 board.

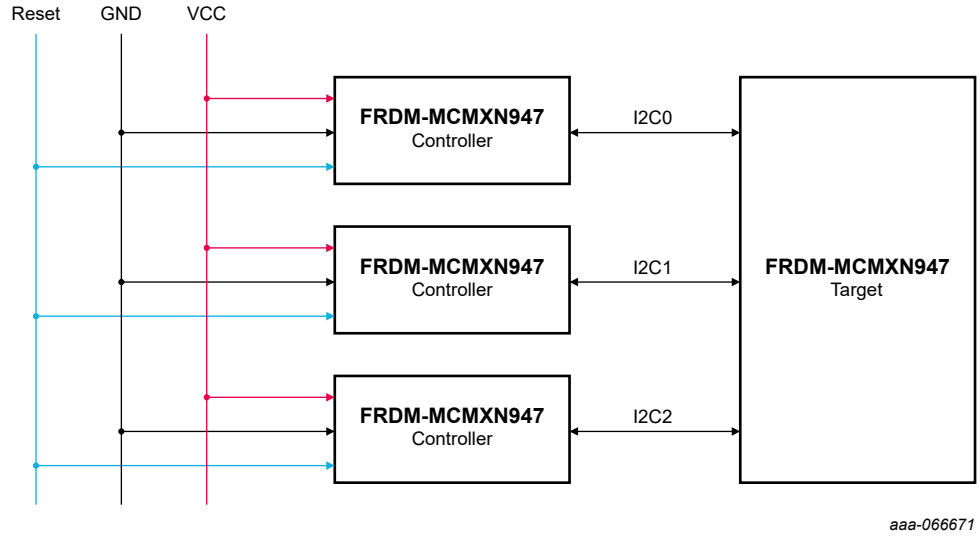

Figure 1. Hardware connection block diagram

Figure 2 shows the physical hardware connection. The hardware setup consists of three FRDM-MCXN947 controller boards on the left and one FRDM-MCXN947 target board on the right.

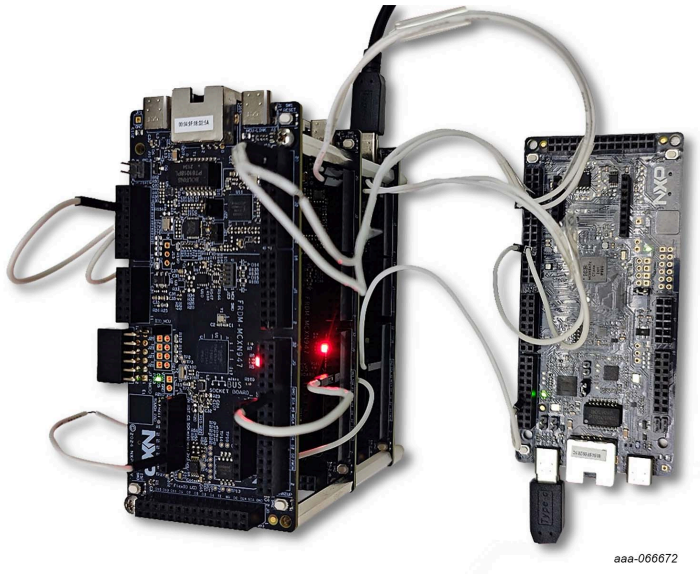

Figure 2. Physical hardware connection

4 Software introduction

The demo software is designed to stay compatible with the existing SDK hardware LPI2C target API. This compatibility makes it easier for users to understand, migrate, and integrate the implementation.

Table 1 summarizes the software layout.

Table 1. Software Layout

Layer	Files	Responsibility
Application entry	source/main.c	Initializes board hardware and status LED, starts the SmartDMA I2C demo, then runs polling/LED ticks forever.
Demo application	source/smartdma_i2c_demo.c/.h	Defines the three virtual target instances, target address, RX/TX buffers, application callback, and startup sequence.
SmartDMA LPI2C wrapper	i2c_driver/fsl_smartdma_lpi2c_slave.c/.h	Exposes an SDK-style LPI2C target API backed by SmartDMA firmware and a shared event buffer.
SmartDMA IRQ glue	source/smartdma_irq.c	Routes SMARTDMA_IRQHandler() to the wrapper IRQ handler.
Board setup	board/hardware_init.c, board/pin_mux.c, board/clock_config.c	Configures debug console, PLL clock, GPIOs, and SmartDMA/I2C pins.

4.1 Transaction support

The SmartDMA wrapper converts SmartDMA channel state updates into the LPI2C target transactional events expected by the application. Table 2 summarizes the implemented transaction behaviors.

Table 2. Transaction behaviors

Transaction/Event	Implemented behavior
Address phase with the ACK STALL	SmartDMA forwards the received address byte to the Arm. The application callback calls SMARTDMA_LPI2C_SlaveSetAddressAck(). The current demo policy ACKs every received address.
I2C write	After an accepted write address, the wrapper raises ReceiveEvent. The application supplies a 32-byte rxBuf. RXDATAEVENT stores each byte and CompletionEvent reports the transferred byte count.
I2C read	After an accepted read address, the wrapper raises TransmitEvent. The application supplies txBuf. TXDATAEVENT loads successive bytes; if data is unavailable, the wrapper sends 0xFF.
SMBus ARA read	When address 0x0C is received with read direction, the application reports an ARA event and returns one byte: target address << 1.
Completion	The SmartDMA status register acts as the transaction-complete indication. The wrapper clears it and reports CompletionEvent with status and count.

4.2 Implemented demo functions

Table 3 summarizes the implemented demo functions.

Table 3. Demo functions

Function	Code location	Behavior
Three virtual I2C targets	source/smartdma_i2c_demo.c	Initializes virtual LPI2C0, LPI2C1, and LPI2C2 target instances at address 0x7E.
SDK-compatible target wrapper	i2c_driver/fsl_smartdma_lpi2c_slave.c	Maps LPI2C-style APIs to SmartDMA event buffer operations and callback dispatch.
SmartDMA service boot	i2c_driver/fsl_smartdma_lpi2c_slave.c	Installs s_smartdmaI2cSlaveFirmware at 0x04000000 and boots API kSMARTDMA_I2CSlave.

Table 3. Demo functions*...continued*

Function	Code location	Behavior
Status indication	source/status_led.c	Toggles green LED every 250 ms while the target main loop is running.
Controller stress test	mcxn947_i2c_master_tester/source/i2c_testcase_default.c	Runs repeated 32-byte write and read/compare loops.
Tester error reporting	mcxn947_i2c_master_tester/source/tester_error.c	Reports start failures, NAKs, transfer failures, timeouts, and data mismatches over UART.
Interrupt priority	i2c_driver/fsl_smartdma_lpi2c_slave.c	Configures SMARTDMA_IRQn priority and enables SmartDMA interrupt handling for channel event dispatch.

4.3 SmartDMA service implementation

Table 4 summarizes the SmartDMA service implementation details.

Table 4. SmartDMA service implementation

Item	Value/Behavior
Firmware memory address	SMARTDMA_I2C_SLAVE_MEM_ADDR = 0x04000000
Firmware image	s_smartdmaI2cSlaveFirmware from drivers/mcxn/fsl_smartdma_fw.c
Shared buffer	100 32-bit words used as SmartDMA stack and channel event mailbox.
Channel mailbox layout	CH0 uses word offsets 0 to 3, CH1 uses 8 to 11, and CH2 uses 16 to 19 for state/data/address/status.
ACKSTALL bit	Address register bit 24 enables Arm-owned address ACK/NACK decision.
Interrupt priority	SMARTDMA_IRQn is enabled with priority 1.
AHB priority	SYSCON->AHBMATPRIO is updated before booting the SmartDMA service. Increasing the priority of SmartDMA helps improve the timeliness of SmartDMA responses.
System clock	Setting the system clock to the maximum frequency helps improve the timeliness of SmartDMA responses.

4.4 Startup flow

At reset, main() calls BOARD_InitHardware(), initializes the status LED, reads the core clock, and calls SmartdmaI2cDemo_Start(). The demo initializes per-channel buffers, configures each virtual target, starts nonblocking target transfers, and finally boots the SmartDMA I2C target service.

Figure 3 shows the startup flow.

The startup sequence proceeds as follows:

1. BOARD_InitHardware(): Configures pins, clocks, and the debug console.
2. StatusLed_Init(): Initializes the status LED.
3. SmartdmaI2cDemo_Start(coreClock): Enters the demo initialization sequence:
   - Initializes demo state: txBuf = 0x00 to 0x1F, address table = 0x7E.
   - Creates three virtual targets: LPI2C0, LPI2C1, and LPI2C2 with ACKSTALL enabled.
   - Calls SMARTDMA_LPI2C_ServiceStart() to install firmware, enable the IRQ, and boot the SmartDMA service.
4. While (1) main loop: Runs continuously, executing two parallel tasks:
   - SmartdmaI2cDemo_Poll(): Optionally prints RX results.

- StatusLed_Tick(): Toggles the green LED every 250 ms.

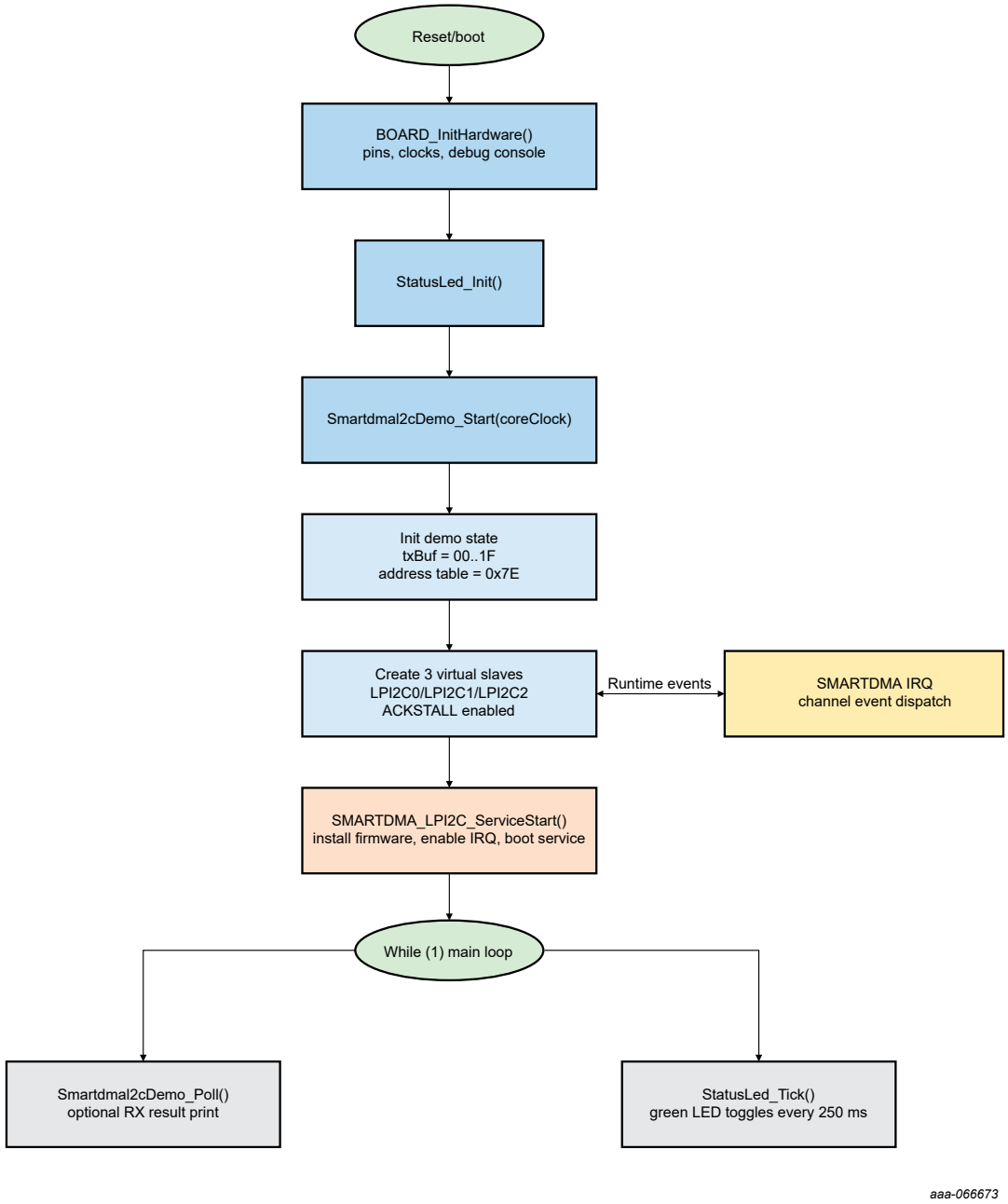

Figure 3. Startup flow

4.5 SmartDMA LPI2C wrapper

The wrapper keeps the public programming style close to the MCUXpresso LPI2C target transactional API. The application calls the following functions:

- SMARTDMA_LPI2C_SlaveGetDefaultConfig()
- SMARTDMA_LPI2C_SlaveInit()
- SMARTDMA_LPI2C_SlaveTransferCreateHandle()
- SMARTDMA_LPI2C_SlaveTransferNonBlocking()

Internally, the wrapper maps virtual LPI2C base pointers to SmartDMA channel indexes and translates SmartDMA state codes into LPI2C target callback events.

Table 5 shows the channel-to-register and pin mapping.

Table 5. Channel mailbox register and pin mapping

Channel	State register	Data register	Address register	Status register	Pins
CH0/LPI2C0	0	1	2	3	SDA P1_6/PIO2, SCL P1_7/PIO3
CH1/LPI2C1	8	9	10	11	SDA P3_4/PIO4, SCL P3_5/PIO5
CH2/LPI2C2	16	17	18	19	SDA P1_22/PIO18, SCL P1_23/PIO19

The address register stores the configured 7-bit target address in bits [7:0]. Bit 24 enables ACKSTALL mode. In this demo, ACKSTALL is enabled on all three channels, which allows the Arm callback to decide whether each address phase must be ACKed.

4.6 Callback events and data path

The application callback manages address match, receive setup, transmit setup, and completion. On a normal read, it returns the per-channel txBuf initialized to 0x00 to 0x1F. On a write, it provides a 32-byte rxBuf and records the received byte count when the completion event arrives.

Figure 4 shows the callback event and data path flow.

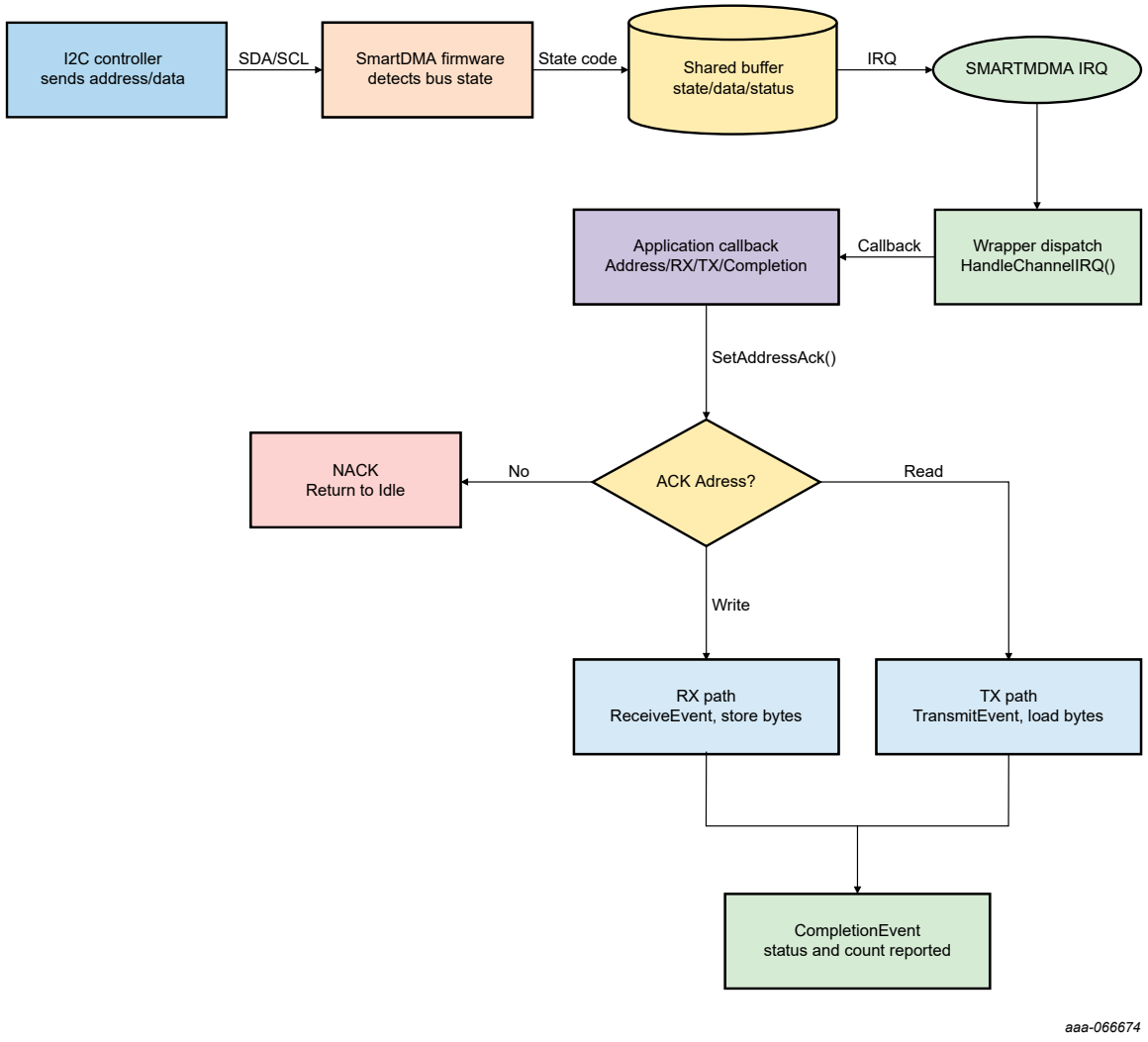

Figure 4. Callback event and data path flow

Table 6 summarizes the application behavior for each callback event.

Table 6. Callback event application behavior

Event	Application behavior
kLPI2C_SlaveAddressMatchEvent	Detects SMBus ARA read address 0x0C and updates araResponseByte. When ACKSTALL is enabled, calls SMARTDMA_LPI2C_SlaveSetAddress Ack().
kLPI2C_SlaveReceiveEvent	Clears rxBuf, resets rxCount, and assigns a 32-byte receive buffer to transfer->data.
kLPI2C_SlaveTransmitEvent	For SMBus ARA, returns one byte: address << 1. For normal reads, returns txBuf with 32 bytes.
kLPI2C_SlaveCompletionEvent	For write transactions that complete successfully, record transferredCount and set rxDone.

The current SmartdmaI2cDemo_ShouldAckAddress() implementation returns true without checking the received address. This implementation is convenient for bring-up, but it causes the ACKSTALL path to ACK

every address byte. For production-style address filtering, compare ((receivedAddress >> 1) & 0x7F) with the configured app->address, and handle SMBus ARA separately if required.

5 Hardware setup

As described in Section 3, the setup uses three FRDM-MCXN947 controller boards and one target board. Because all three target channels share the default address 0x7E, each controller board connects to one dedicated target channel. It allows the three channels to be tested simultaneously on independent I2C buses. Table 7 lists the required pin connections.

Table 7. Controller-to-Target pin connections

Signal	Controller 0 → Target CH0	Controller 1 → Target CH1	Controller 2 → Target CH2
SDA	LPI2C2 P4_0 → P1_6/J9[10]/ SMARTDMA_PIO2	LPI2C2 P4_0 → P3_4/J9[12]/ SMARTDMA_PIO4	LPI2C2 P4_0 → P1_22/J9[24]/ SMARTDMA_PIO18
SCL	LPI2C2 P4_1 → P1_7/J9[9]/ SMARTDMA_PIO3	LPI2C2 P4_1 → P3_5/J9[11]/ SMARTDMA_PIO5	LPI2C2 P4_1 → P1_23/J9[23]/ SMARTDMA_PIO19
GND	Common ground	Common ground	Common ground
Power/debug	USB debug port	USB debug port	USB debug port

6 Build and flash

To build, flash, and verify the demo, perform the following steps:

1. Open workspace: Open the workspace in MCUXpresso IDE 25.6 or use a compatible MCUXpresso IDE version with the MCX N947 support installed.
2. Flash target board: Build the mcxn947_smartdma_3xi2c_target debug configuration and flash it to the target FRDM-MCXN947 board.
3. Flash controller tester boards: Build the mcxn947_i2c_controller_tester debug configuration and flash it to each controller tester FRDM-MCXN947 board.
4. Verify target UART: Open the target debug UART at 115200 bauds. Confirm the SmartDMA I2C target service startup log.
   Expected target-side startup logs include:

```
SmartDMA I2C target demo
CoreClock:<frequency>
Build Timestamp: <date> <time>
SMARTDMA virtual I2C0 target started at 0x7E (ACKSTALL=1)
SMARTDMA virtual I2C1 target started at 0x7E (ACKSTALL=1)
SMARTDMA virtual I2C2 target started at 0x7E (ACKSTALL=1)
SMARTDMA SWI2C target service started. Each target expects 32-byte writes and
 replies 00..1F.
```

5. Verify tester UART: Open the tester debug UART at 115200 bauds. Confirm the tester startup log and periodic success/fail heartbeat.
   Expected tester-side logs include:

```
…
[INFO] [status] success=11119700 fail=0
[INFO] [status] success=11119800 fail=0
[INFO] [status] success=11119900 fail=0
[INFO] [status] success=11120000 fail=0
…
```

```
SmartDMA I2C target demo
CoreClock:<frequency>
Build Timestamp: <date> <time>
SMARTDMA virtual I2C0 target started at 0x7E (ACKSTALL=1)
SMARTDMA virtual I2C1 target started at 0x7E (ACKSTALL=1)
SMARTDMA virtual I2C2 target started at 0x7E (ACKSTALL=1)
SMARTDMA SWI2C target service started. Each target expects 32-byte writes and
 replies 00..1F.


[INFO] I2C target=0x7e baud=450000 payload=31
[INFO] FRDM MCXN947 I2C master tester
[INFO] CoreClock:150000000Hz
[INFO] LPI2C uses FC2 pins P4_0/P4_1
[INFO] [status] success=11119700 fail=0
[INFO] [status] success=11119800 fail=0
[INFO] [status] success=11119900 fail=0
[INFO] [status] success=11120000 fail=0
[INFO] [status] success=100 fail=0
[INFO] [status] success=200 fail=0
…
```

7 Summary

This application note demonstrates the implementation of three simultaneous I2C target interfaces on the MCX N947 using SmartDMA. The solution provides a software-based approach that supports three independent 400 kHz I2C buses without dedicating hardware LPI2C target peripherals, while maintaining compatibility with the standard MCUXpresso SDK LPI2C target API.

8 Acronyms

Table 8 lists the acronyms used in this document.

Table 8. Acronyms

Acronym	Description
ACK	Acknowledgment
AHB	Advanced High-Performance Bus
API	Application Programming Interface
ARA	Alert Response Address
GND	Ground
GPIO	General-Purpose Input/Output
IDE	Integrated Development Environment
IRQ	Interrupt Request
LPI2C	Low-Power Inter-Integrated Circuit
NACK	Not Acknowledged
PLL	Phase Locked Loop
SCL	Serial Clock
SDA	Serial Data

Table 8. Acronyms...*continued*

Acronym	Description
SDK	Software Development Kit
SMBus	System Management Bus
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

9 Revision history

Table 9 summarizes revisions to this document.

Table 9. Revision history

Document ID	Release date	Description
AN15087 v.1.0	9 July 2026	Initial public 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 https://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 document 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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell 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.

Tables

Figures

Contents