AN14184

Using SmartDMA for Keyscan on MCX N Series MCU

Rev. 1 — 20 January 2024 Application note

Document information

Information	Content
Keywords	Keyscan, FRDM-MCXN947, SmartDMA
Abstract	This application note introduces Keyscan solution for MCX N series MCU.

1 Introduction

This application note mainly introduces the Keyscan solution for MCX N series MCU. It includes the introduction of the Keyscan solution, its features and API routines, and a demo.

All MCX N series MCUs include a SmartDMA coprocessor, which can effectively reduce the load on the Arm core and perform fast I/O operations.

2 Target application

As the name suggests, the Keyscan solution is used in key scanning applications, such as in a computer keyboard. However, it can also be used in other scenarios, such as in a requirement where continuous scanning of IO port input is required.

3 Keyscan interfaces

The Keyscan scheme has no fixed interface. It can be used for matrix scanning and for row or column scanning. The number of scanned keys can be one or from one to two hundreds.

If it is a commonly used computer keyboard, it is generally 101 keys, 104 keys, or 87 keys. If it is a small keyboard, it is generally 4x4 to get 16 keys. It can also be used for irregular button matrix. In short, the button layout and the number of buttons can be customized.

This application note uses a 4x4 matrix keyboard, however, the interface is not fully compatible, it only supports the determination of 2x4 with a total of 8 keys. To achieve the determination of 16 keys, the hardware should be reworked by some wires.

4 Features of Keyscan solution

The features of the Keyscan solution are as follows:

- 4 x 4 Keyscan
- Superfast key scan without Arm core intervention (>= 8 kHz report rate)
- Programmable debounce time
- Easy to support 8 x 16 or other size
- Easily portable to other platforms

5 Functional description

This section describes the functional description of the Keyscan solution.

5.1 Keyscan engine

SmartDMA that serves as a coprocessor of MCX N series MCU, is characterized by efficient instruction execution. It can quickly and efficiently complete key scanning operations. During the scanning process of button operations, there is no need for the Arm core to intervene. An interrupt is sent to the Arm core only when there is a change in the key value. The Arm core only needs to read the key value from the RAM.

5.2 Keyscan driver lib

The instructions of SmartDMA use the type of machine code. The code implements the functions of the Keyscan solution and is released in a C array. Some API routines are provided in this application. You can use API routines to initialize the engine and configure the pins, start, or stop Keyscan.

5.3 System clock

The Keyscan engine shares the system clock with the Arm core. Lowering the system clock frequency reduces the speed at which SmartDMA executes code.

5.4 Memory usage

The code of SmartDMA must be loaded and executed at a fixed location, which in this application is 0x04000000. Changing the execution instruction location requires regeneration of the instruction code array.

5.5 Hardware description

Connect the PmodKYPD board with the FRDM-MCXN947 board as shown in Figure 1.

Figure 1. Demo hardware

Note: *The PmodKYPD board can be purchased through the following website:*

https://store.digilentinc.com/pmod-kypd-16-button-keypad

Figure 2 shows the PmodKYPD schematic.

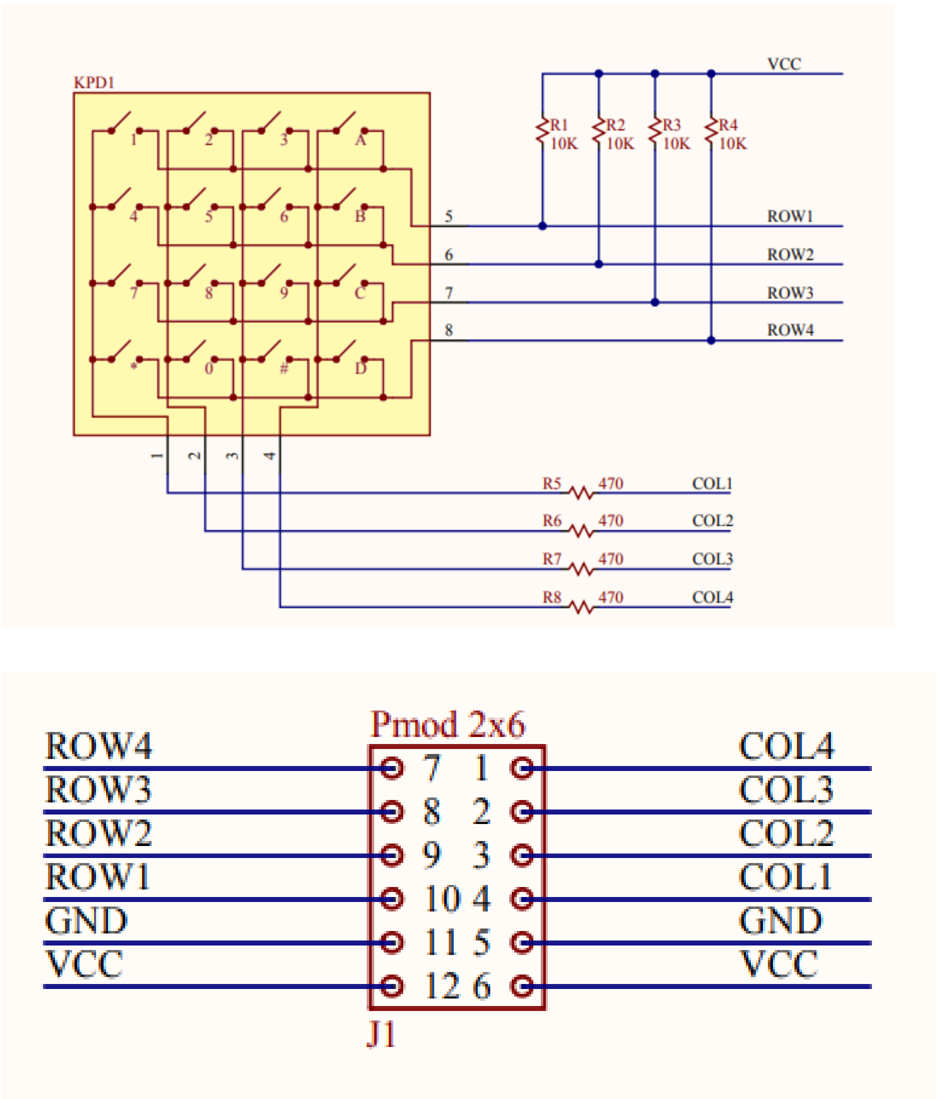

Figure 2. Keyscan schematic

Figure 3 shows the schematic of Pmod header on the FRDM-MCXN947 board.

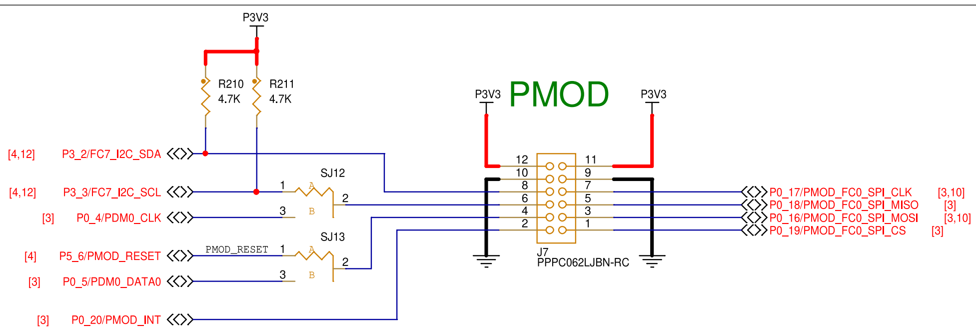

Figure 3. Pmod header schematic

5.6 Pin description

Figure 4 shows how to connect the PmodKYPD board with the FRDM-MCXN947 board.

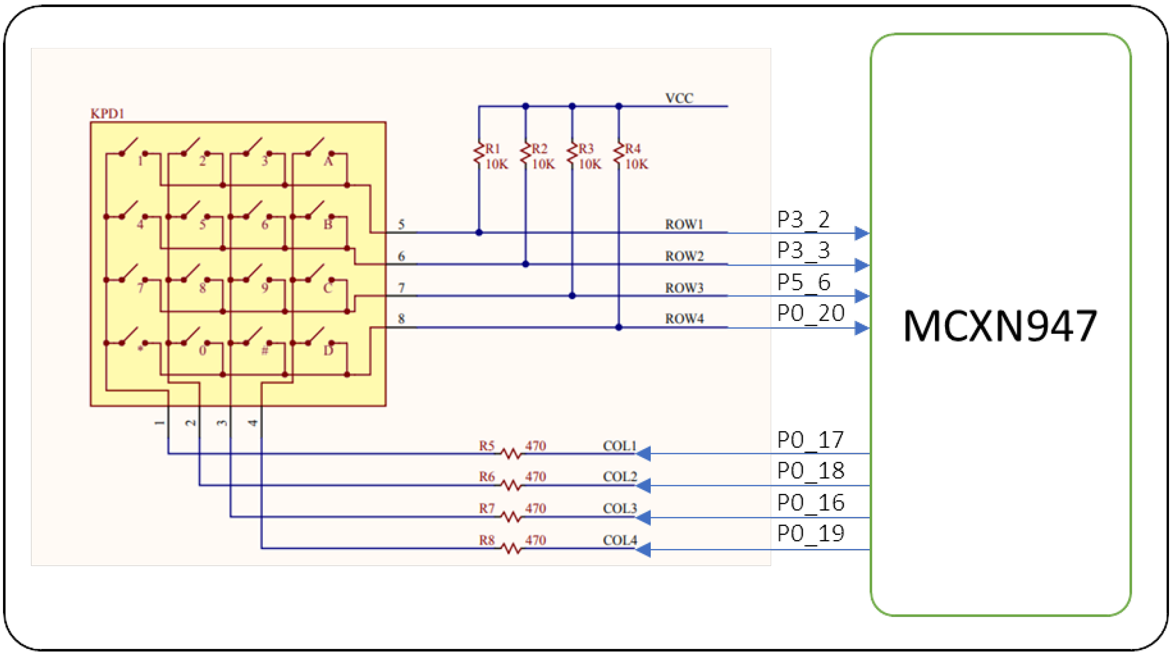

Figure 4. Hardware connection

5.7 Keyscan timings

When a key is not pressed, the SmartDMA continuously outputs waveforms on each column line. However, when a key is pressed, the row where the key is located will have the same waveform as the column where it is located. In this way, it can locate which button is pressed.

6 Software description

This section describes the SmartDMA Keyscan example and the functions to implement it.

6.1 Demo example introduction

The demo code in this example is generated by the config tool as a standalone "Hello World" project. Based on this project, I/O initialization code and SmartDMA driver code are added.

6.2 SmartDMA function array

The SmartDMA Keyscan API can be found in the file fsl_smartdma_mcxn.h.

```
/*!
* @brief The API index when using s_smartdmaKeyscanFirmware
*/
enum _smartdma_keyscan_api
{
/*!using Smartdma to control GPIO . */
kSMARTDMA_Keyscan_4x4 = 0U,
};
```

In the fsl_smartdma_mcxn.c file, there is an array called s_smartdmaKeyscanFirmware, which contains the implementation of SmartDMA Keyscan functions. The purpose of encapsulating the SmartDMA functions into an array is to reduce the SmartDMA research cost for users and allow them to directly use the module functions implemented, enabling faster implementation of application functions.

6.3 SmartDMA initialization

The following functions implement the SmartDMA initialization.

Table 1. API routines

Routine	Description
`SMARTDMA_InitWithoutFirmware`	Initialize the SmartDMA
`SMARTDMA_InstallFirmware`	Install the firmware
`SMARTDMA_InstallCallback`	Install the complete callback function
`SMARTDMA_Boot`	Boot the SmartDMA to run the program
`SMARTDMA_Deinit`	Deinitialize the SmartDMA
`SMARTDMA_Reset`	Reset the SmartDMA
`SMARTDMA_HandleIRQ`	SmartDMA IRQ
`SmartDMA_keyscan_callback`	SmartDMA interrupt callback

6.3.1 Init SmartDMA

To enable SmartDMA, perform the following operations.

- Clear reset of SmartDMA
- Enable the clock for SmartDMA
- Enable the IRQ for SmartDMA

6.3.2 Install SmartDMA firmware

The function module of SmartDMA must be placed at a fixed memory address to work properly. In this application, it must be placed at 0x04000000.

For example:

```
/*! @brief The firmware used for keyscan. */
extern const uint8_t s_smartdmaKeyscanFirmware[];
/*! @brief The s_smartdmaKeyscanFirmware firmware memory address. */
#define SMARTDMA_KEYSCAN_MEM_ADDR 0x04000000U
/*! @brief Size of s_smartdmacameraFirmware */
#define SMARTDMA_KEYSCAN_FIRMWARE_SIZE (s_smartdmaKeyscanFirmwareSize)
/*! @brief Size of s_smartdmacameraFirmware */
extern const uint32_t s_smartdmaKeyscanFirmwareSize;
```

The process of installing SmartDMA firmware is essentially copying the code array of SmartDMA function modules to a specified RAM address.

As the following code snippet:

```
SMARTDMA_InitWithoutFirmware();
SMARTDMA_InstallFirmware(SMARTDMA_KEYSCAN_MEM_ADDR,s_smartdmaKeyscanFirmware,
SMARTDMA_KEYSCAN_FIRMWARE_SIZE);
```

6.3.3 SmartDMA callback routine

SmartDMA can actively trigger an interruption in the Arm core, such as after the end of data transfer.

SmartDMA has a related interrupt number (`SMARTDMA_IRQHandler`) in the Arm vector table. In the configuration phase of SmartDMA, a callback function can be installed.

As the following code snippet:

```
SMARTDMA_InstallCallback(SmartDMA_keyscan_callback, NULL);
```

In the callback function, the Arm core can read the pressed key value and print the log.

6.3.4 Boot SmartDMA API

In the application, define a structure to set parameters related to SmartDMA. These parameters include the address of the data buffer, the length of data transfer, and the address of SmartDMA stack space. The most important thing is to find an API that must be executed from the SmartDMA function block code.

As the following code snippet:

```
smartdmaParam.smartdma_stack = (uint32_t*)g_samrtdma_stack;
smartdmaParam.p_gpio_reg = (uint32_t*)g_keyscan_gpio_register;
smartdmaParam.p_keyvalue = (uint32_t*)KeyValue;
smartdmaParam.p_keycan_interval = (uint32_t*)&g_keyscan_interval;
SMARTDMA_Boot(kSMARTDMA_Keyscan_4x4, &smartdmaParam, 0x2);
```

The process of boot is to give the address of the corresponding API to the program counter of SmartDMA, and then it begins to execute the function block.

7 Demo download and run

This section describes how to prepare and run the demo.

7.1 Prepare the demo

1. Connect the USB Type-C to micro-USB cable between the PC host and the USB port on the board.
2. Open a serial terminal on a PC for the serial device with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.

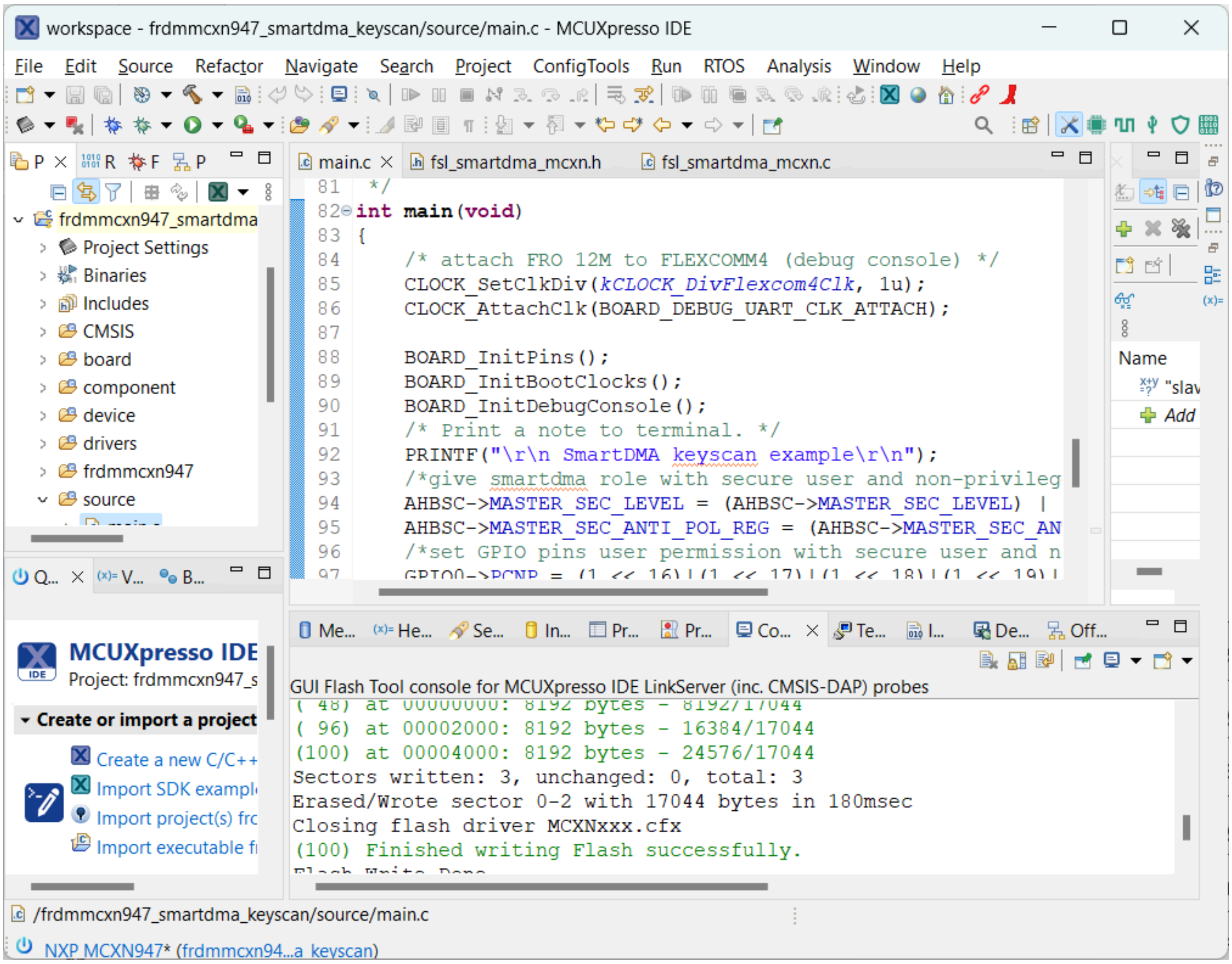

Figure 5. Project

4. Either press the reset button on your board or launch the debugger in your IDE to begin running the demo.

7.2 Run the demo

1. The following lines are printed to the serial terminal when the demo program is executed.

```
SmartDMA keyscan example
```

2. Press some button on PmodKYPD, the following lines are printed to the serial terminal:

```
Button 2 is pressed
Button 1 is pressed
Button B is pressed
Button 6 is pressed
Button 5 is pressed
```

8 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

9 Revision history

Table 2 summarizes revisions to this document.

Table 2. Revision history

Document ID	Release date	Description
AN14184 v.1	20 January 2024	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.

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.

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 and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) 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.

MCX — is a trademark of NXP B.V.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

Contents

1 Introduction 2
2 Target application 2
3 Keyscan interfaces 2
4 Features of Keyscan solution 2
5 Functional description 2
5.1 Keyscan engine 2
5.2 Keyscan driver lib 3
5.3 System clock 3
5.4 Memory usage 3
5.5 Hardware description 3
5.6 Pin description 5
5.7 Keyscan timings 6
6 Software description 6
6.1 Demo example introduction 6
6.2 SmartDMA function array 6
6.3 SmartDMA initialization 7
6.3.1 Init SmartDMA 7
6.3.2 Install SmartDMA firmware 7
6.3.3 SmartDMA callback routine 8
6.3.4 Boot SmartDMA API 8
7 Demo download and run 8
7.1 Prepare the demo 8
7.2 Run the demo 9
8 Note about the source code in the document 9
9 Revision history 10
Legal information 11

Date of release: 20 January 2024
Document identifier: AN14184