AN14184

Using SmartDMA for Keyscan on MCX N Series MCU

Rev. 2 — 15 May 2024 Application note

Document information

Information	Content
Keywords	Keyscan, FRDM-MCXN947, SmartDMA, FRDM-MCXN236
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.

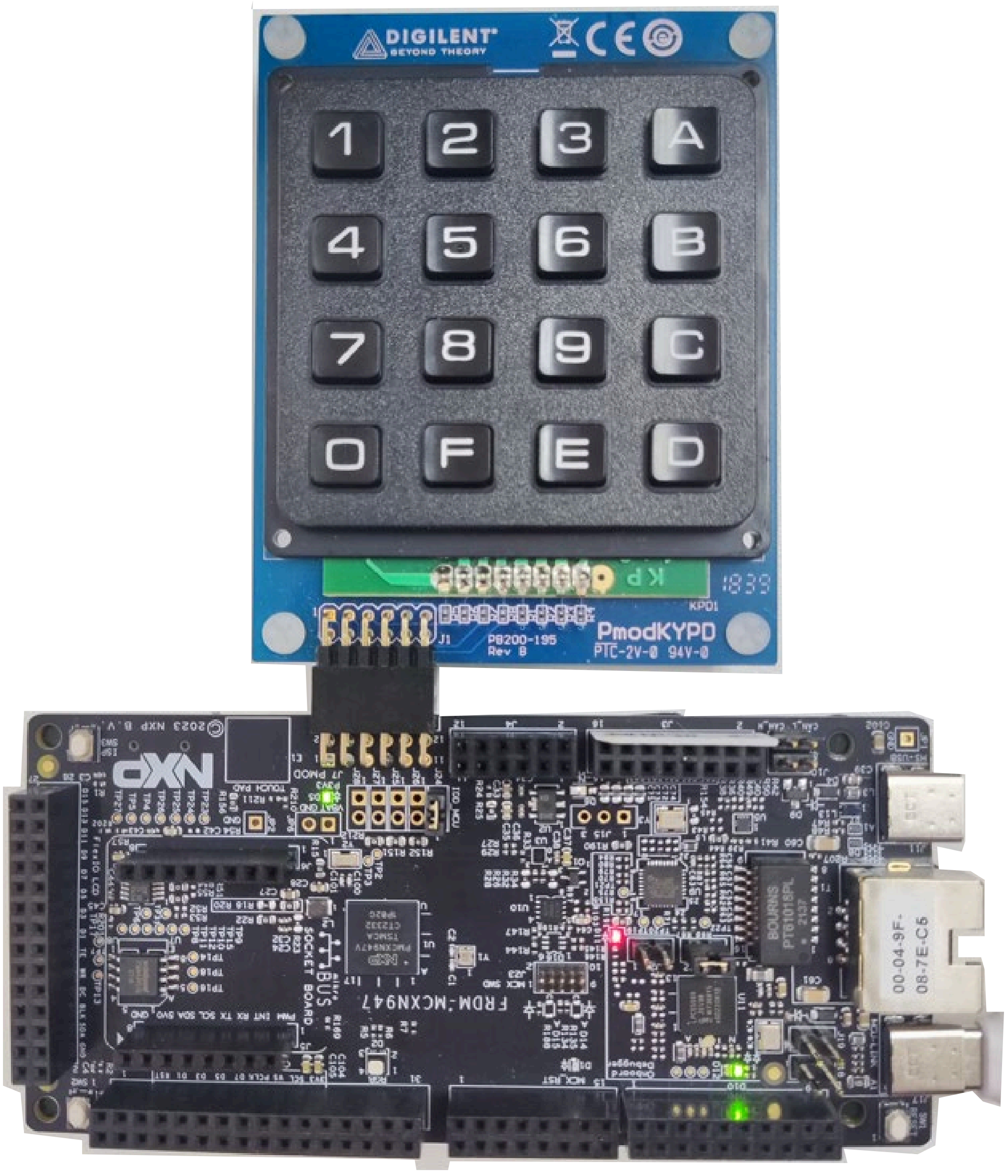


Figure 1. Demo hardware

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.

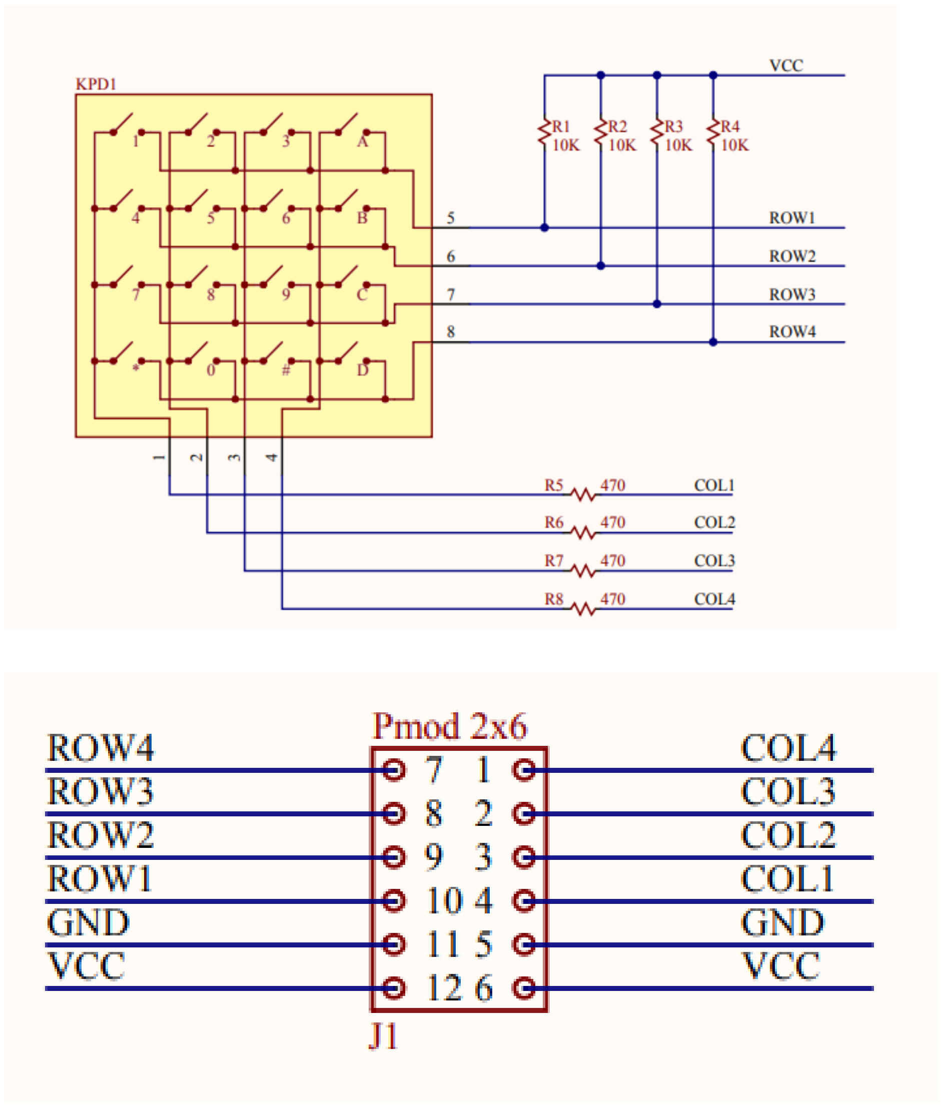


Figure 2. Keyscan schematic

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

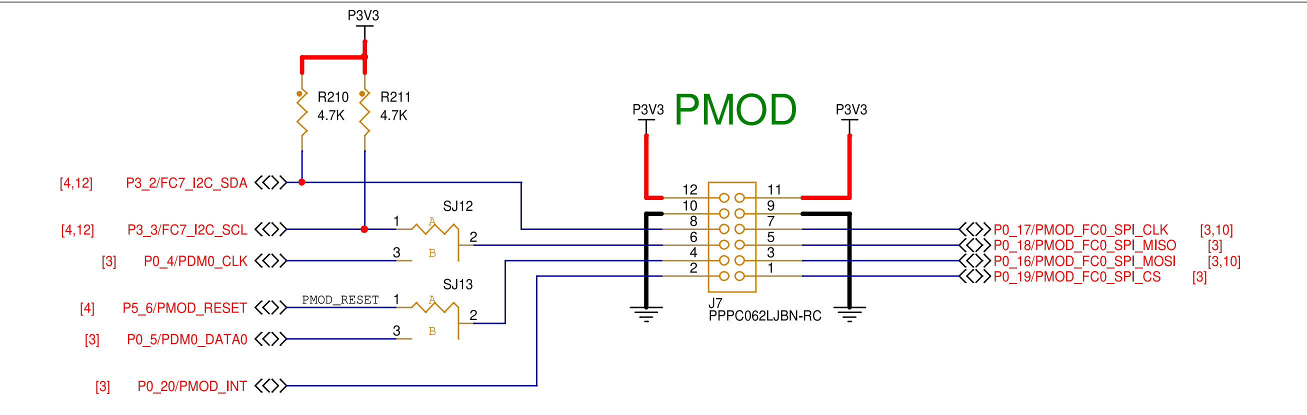


Figure 3. Pmod header schematic

5.6 Pin description

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

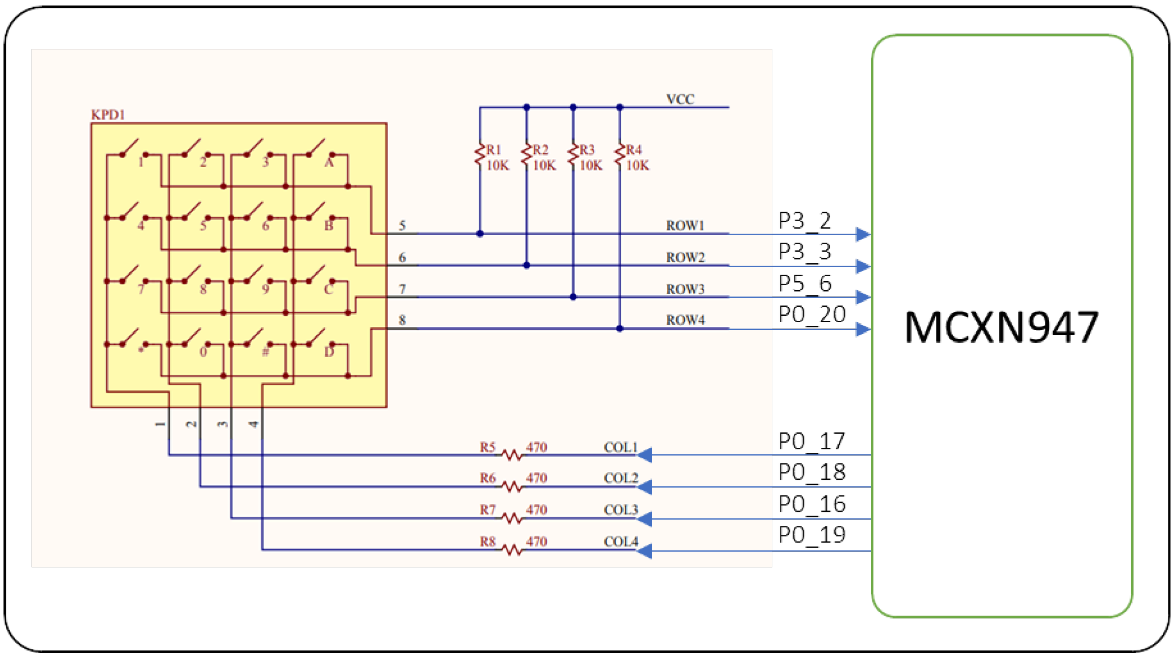


Figure 4. Hardware connection

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.

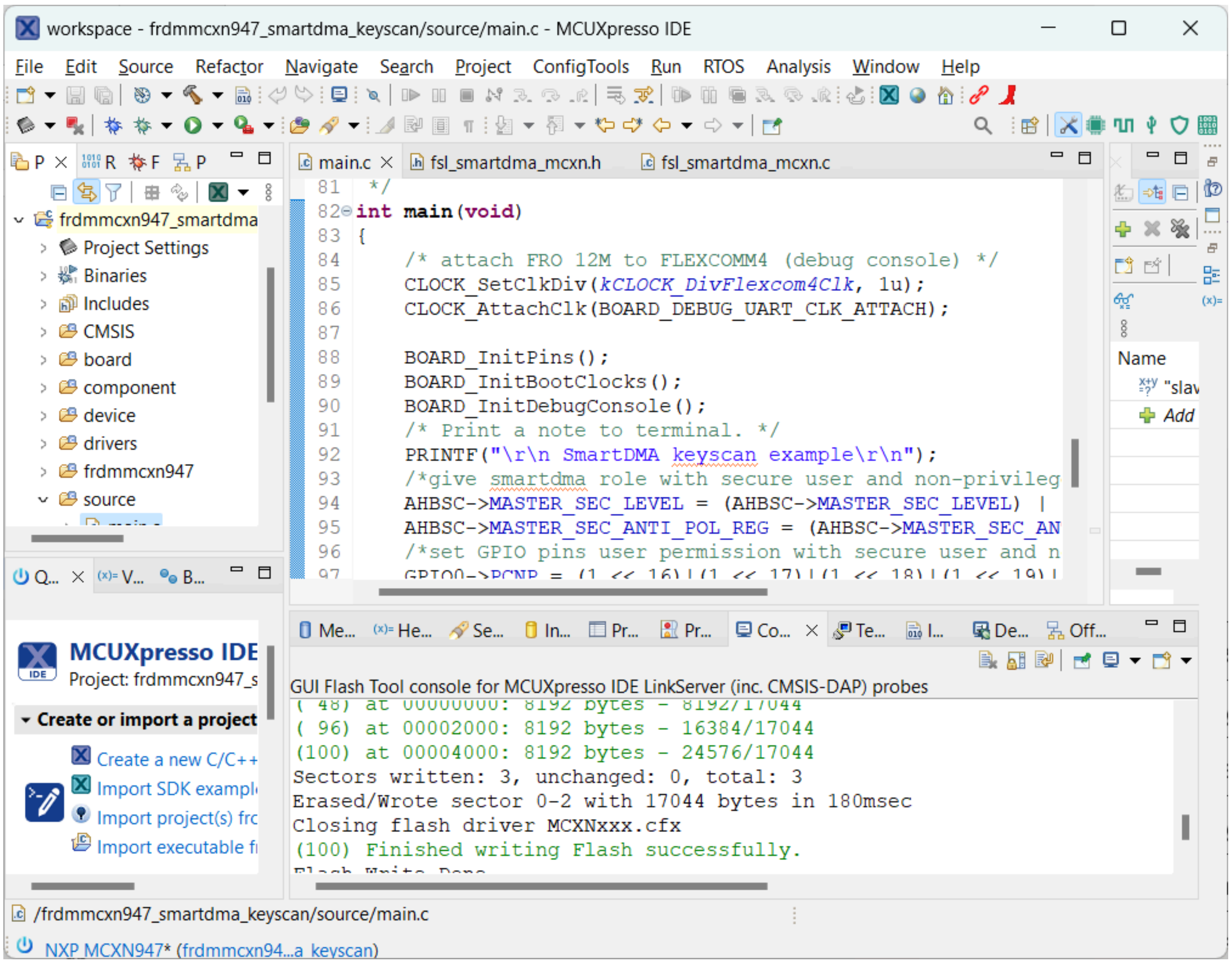
7 Download and run demo based on FRDM-MCXN947

This section describes how to prepare and run the demo that is based on the FRDM-MCXN947 board.

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.


workspace - frdmmcxn947_smartdma_keyscan/source/main.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run RTOS Analysis Window Help
main.c
fsl_smartdma_mcxn.h
fsl_smartdma_mcxn.c
frdmmcxn947_smartdma
Project Settings
Binaries
Includes
CMSIS
board
component
device
drivers
frdmmcxn947
source
MCUXpresso IDE
Project: frdmmcxn947_s
Create or import a project
Create a new C/C++
Import SDK exampl
Import project(s) fro
Import executable fi
Name
"slav
Add
GUI Flash Tool console for MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
/frdmmcxn947_smartdma_keyscan/source/main.c
NXP MCXN947* (frdmmcxn94...a_keyscan)

```
  */
int main(void)
{
    /* attach FRO 12M to FLEXCOMM4 (debug console) */
    CLOCK_SetClkDiv(kCLOCK_DivFlexcom4Clk, 1u);
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);

    BOARD_InitPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();
    /* Print a note to terminal. */
    PRINTF("\r\n SmartDMA keyscan example\r\n");
    /*give smartdma role with secure user and non-privileg
    AHBSC->MASTER_SEC_LEVEL = (AHBSC->MASTER_SEC_LEVEL) |
    AHBSC->MASTER_SEC_ANTI_POL_REG = (AHBSC->MASTER_SEC_AN
    /*set GPIO pins user permission with secure user and n
    GPIO0->PCNP = (1 << 16) | (1 << 17) | (1 << 18) | (1 << 19) |
```

```
( 48) at 00000000: 8192 bytes - 8192/17044
( 96) at 00002000: 8192 bytes - 16384/17044
(100) at 00004000: 8192 bytes - 24576/17044
Sectors written: 3, unchanged: 0, total: 3
Erased/Wrote sector 0-2 with 17044 bytes in 180msec
Closing flash driver MCXNxxx.cfx
(100) Finished writing Flash successfully.
Flash Write Done
```

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 Download and run demo based on FRDM-MCXN236

This section describes how to prepare and run the demo that is based on the FRDM-MCXN236 board.

8.1 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.
4. Either press the reset button on your board or launch the debugger in your IDE to begin running the demo.

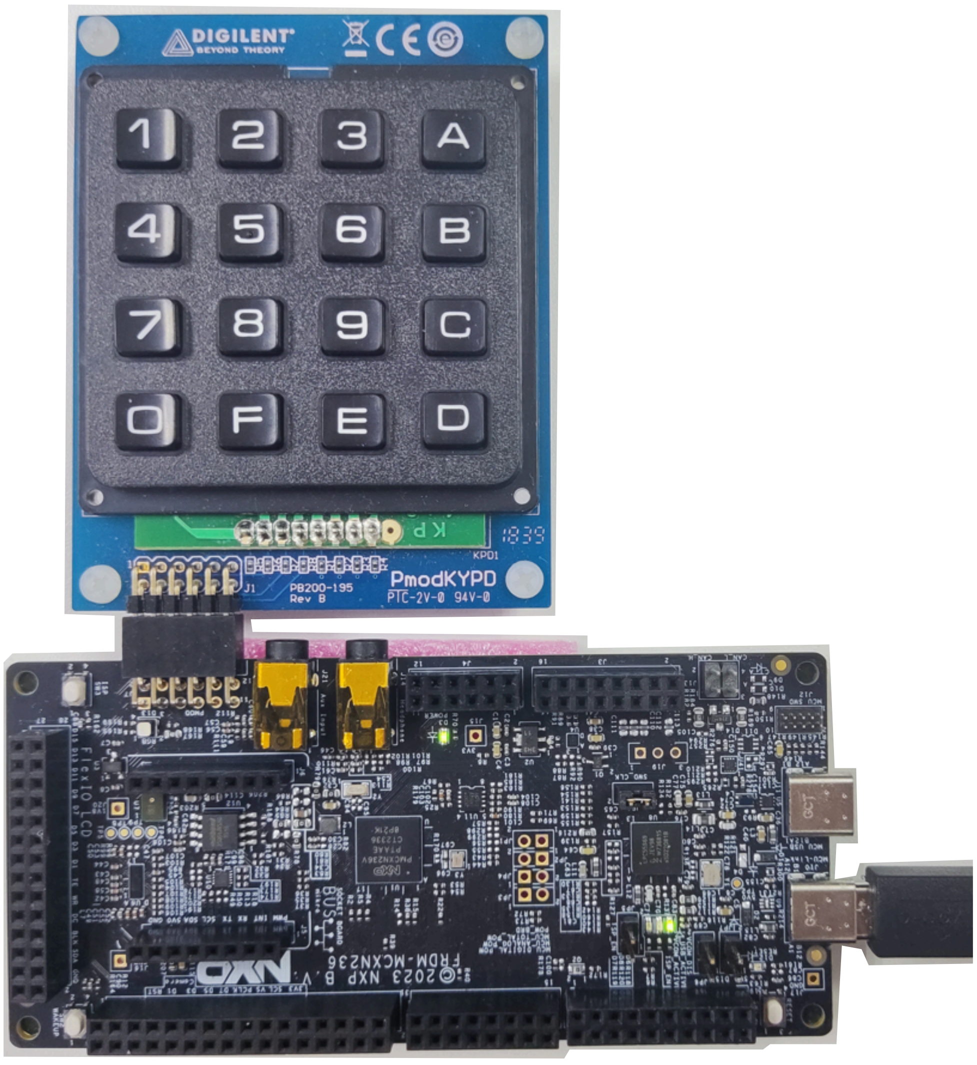


Figure 6. Project

8.2 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
```

9 Note about the source code in the document

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

10 Revision history

Table 2 summarizes revisions to this document.

Table 2. Revision history

Document ID	Release date	Description
AN14184 v.2	15 May 2024	• Added Section 8 section. • Added reference of FRDM-MCXN236 in the document.
AN14184 v.1	20 January 2024	Initial public release

Legal information

Contents

Date of release: 15 May 2024
Document identifier: AN14184