AN14900

Using eDMA and Ping-Pong buffer to Deserialize Multi-channel ADC Result FIFO

Rev. 1.0 — 29 December 2025 Application note

Document information

Information	Content
Keywords	AN14900, eDMA/ADC
Abstract	This application note describes how to use eDMA to tackle the ADC Result FIFO and Deserialize each channel data in FIFO to respectively buffer for each channel.

1 Introduction

This application note describes how to use eDMA to tackle the Analog-to-Digital Converter (ADC) result First-In First-Out (FIFO) and deserialize each channel data in FIFO to respective buffer for each channel. It is useful for high-speed and multi-channel ADC result process by reducing CPU loading and improving data processing speed.

The MCX Nx4x series microcontrollers combine the Arm Cortex-M33 TrustZone core with a CoolFlux BSP32, a PowerQuad DSP Co-processor, and multiple high-speed connectivity options running at 150 MHz. It delivers exceptional processing power and advanced integration.

Enhanced Direct Memory Access (eDMA) empowers efficient data transfers between memory and peripherals, alleviating CPU workload and enhancing system performance. MCX Nx4x series microcontrollers offer a versatile eDMA controller that can be configured to meet a wide range of data transfer requirements.

This demo uses one FRDM-MCXN947 board. The corresponding demo code can be found on NXP AppCodeHub.

2 Overview

Figure 1 shows the functions, which this application must realize. The buffer size of each channel depends on the application.

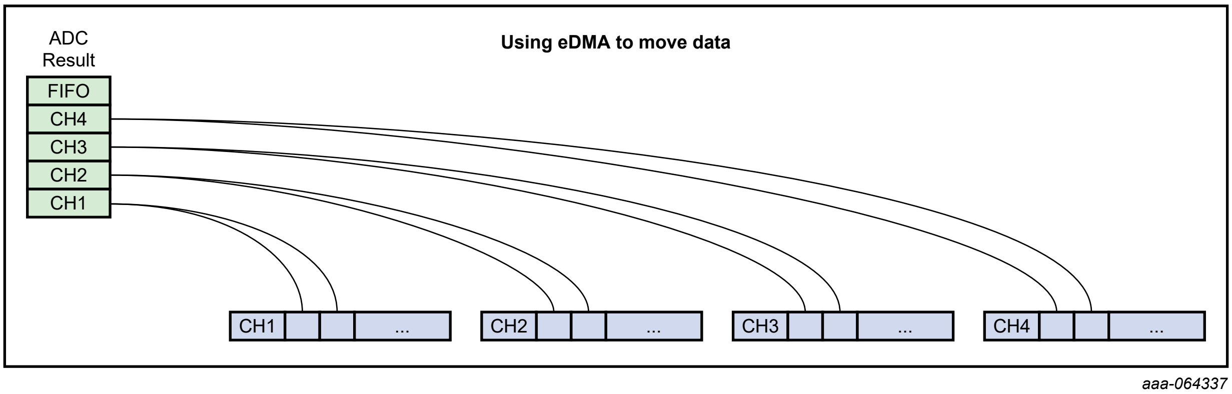

Figure 1. Demo process diagram

2.1 MCX Nx4x ADC block description

This section describes the MCX Nx4x ADC block.

2.1.1 Overview

The MCX Nx4x has two instances of 16-bit ADC.

The 16-bit ADC is a dual successive approximation ADC, which is designed for operation within an integrated microcontroller system-on-a-chip. Figure 2 shows the ADC diagram.

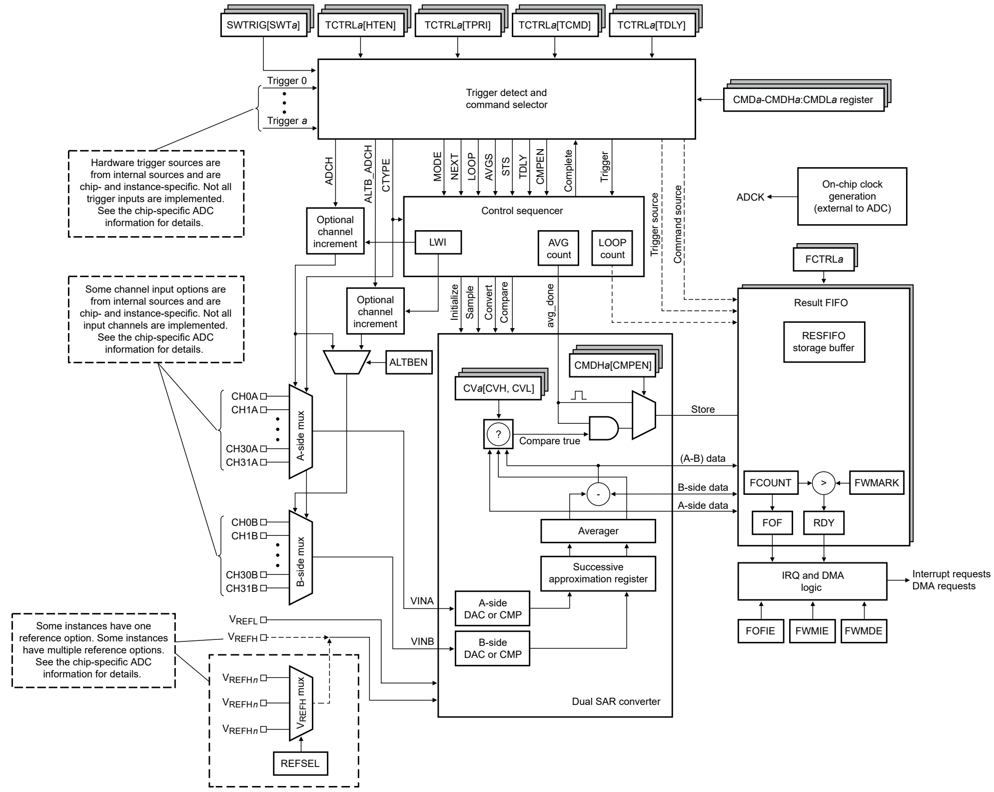

Figure 2. ADC block diagram

2.1.2 ADC features

ADC features are listed below:

- Linear successive approximation algorithm
  - Differential operation with 16-bit or 13-bit resolution
  - Single-ended operation with 16-bit or 12-bit resolution
  - Support for two simultaneous single-ended conversions
- Configurable analog input sample time
- Configurable speed options to accommodate operation in low-power modes of SoC
- Trigger detection with up to four trigger sources with priority level configuration. Software or hardware trigger option for each.
- 15 command buffers, to allow independent options selection and channel sequence scanning
- Automatic comparisons for less-than, greater-than, within range, or out-of-range with **store on true** and **repeat until true** options
- Two independent result FIFOs, each containing 16 entries. Each FIFO has configurable watermark and overflow detection.
- Interrupt, Direct Memory Access (DMA), or polled operation
- Linearity and gain adjustment calibration logic

2.1.3 Functional description

ADC performs analog-to-digital conversions on any of the software-selectable analog input channels via a successive approximation algorithm.

The ADC module can average the result of multiple conversions on a channel before storing the calculated result. The hardware average function is enabled by setting `CMDHn[AVGS]` to a non-zero value. The function operates in any conversion mode or configuration.

When the conversion and averaging loops finish, the resulting data is placed in one of the two available FIFO data buffers. The data includes tag information associated with the result. When the number of stored data words exceeds the setting, a configurable watermark level supports interrupts or DMA requests. Interrupts can also be enabled to indicate when FIFO overflow errors occur.

The module initializes to its lowest power state during reset.

ADC includes multiple command buffers to provide flexibility for channel scanning and independent channel selections for different trigger sources.

2.1.4 Result FIFO operation

ADC includes two 16-entry FIFOs in which the result of ADC conversions are stored. In addition, a valid indicator bit, the trigger source, the source command, and the loop count are also stored with the data. `FCTRLn[FCOUNT]` indicates how many valid data words are stored in each RESFIFO.

A programmable watermark threshold supports configurable notification of data availability. When `FCTRLn[FCOUNT]` is greater than `FCTRLn[FWMARK]`, the associated RDY flag is asserted. When IE[FWMIEn] = 1, a watermark interrupt request is issued. When DE[FWMDEn] = 1, a DMA request is issued. Reading RESFIFO provides the oldest unread data word entry in the FIFO and decrements `FCTRLn[FCOUNT]`. When `FCTRLn[FCOUNT]` falls equal to or below `FCTRLn[FWMARK]`, the RDY flag is cleared.

Each FIFO can be emptied by successive reads of `RESFIFOn`. When `RESFIFOn[VALID]` is 1, the associated FIFO entry is valid. Reading `RESFIFOn` when the FIFO is empty (when RESFIFOn[VALID] = 0 and FCTRLn[FCOUNT] = 0h) provides an undefined data word. All FIFOs are reset by writing 1b to `CTRL[RSTFIFOn]`.

If ADC attempts to store a data word to the FIFO when the FIFO is full, the FIFO overflow flag (`FCTRLn[FOF]`) is set. When IE[FOFIEn] = 1, an overflow interrupt request is issued. The FOF flag is cleared by writing **1** to `STAT[FOFn]`. When overflow events occur, no new data is stored and the data associated with the storage event that triggered the overflow is lost.

Conversion results can be steered to any FIFO in the design. `TCTRLn[FIFO_SEL_A]` and `TCTRLn[FIFO_SEL_B]` determine into which FIFO the final result is written. Depending on which trigger is executing, the results can be steered to different locations. Depending on the type of conversion selected, the FIFO destination register fields are interpreted differently. During either differential or single-ended mode (CMDLn[CTYPE] != 3h) only one result is produced. The destination during these modes is determined from `TCTRLn[FIFO_SEL_A]`. In dual-single-ended mode, both `TCTRLn[FIFO_SEL_A]` and `TCTRLn[FIFO_SEL_B]` determine the Channel A and Channel B destinations respectively.

2.1.4.1 FIFO control register (FCTRL0 - FCTRL1)

Offset

Register	Offset
FCTRL0	E0h
FCTRL1	E4h

Function

Contains control and status fields for each FIFO in the design. A programmable watermark can be set for each FIFO, which can be used to trigger an interrupt. In addition, the number of entries stored in each FIFO can be monitored by reading FCTRLn[FCOUNT].

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0												FWMARK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0											FCOUNT				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field

Field	Function
31-20 —	Reserved
19-16 FWMARK	Watermark Level Selection Selects the storage threshold for the ADC Result FIFO. When the number of data words stored in the FIFO is greater than this value, the STAT[RDY0] flag is asserted. When IE[FWMIEn] = 1, an interrupt request is generated. When DE[FWMDEn] = 1, a DMA request is generated
15-5 —	Reserved
4-0 FCOUNT	Result FIFO Counter Indicates the number of data words stored in the result FIFO. This value may be used with PARAM[FIFOSIZE] to calculate how much room is left in the result FIFO. This field is incremented with each storage of new data into the result FIFO and decremented with each read of the result FIFO. The FIFO is reset by writing to CTRL[RSTFIFOn], which initializes FCTRLn[FCOUNT] to 0h.

2.1.4.2 Data result FIFO register (RESFIFO0 - RESFIFO1)

Offset

Register	Offset
RESFIFO0	300h
RESFIFO1	304h

Function

Stores the data result of ADC conversions in a 16-entry FIFO. Several tag fields of source command and trigger information are stored with the data. FCTRLn[FCOUNT] indicates how many valid data words are stored in the RESFIFO. Reading RESFIFO provides the oldest unread data word entry in the FIFO and decrements FCTRLn[FCOUNT]. The FIFO can be emptied by successive reads of RESFIFO. The FIFO is reset by writing 0b1 to CTRL[RSTFIFOn].

The following table describes the format of data in the result FIFO in different modes of operation. The sign bit is the MSB in signed 2's complement modes. For example, when configured for 12-bit single-ended mode, D[15] and D[2:0] become 0. When configured for 13-bit differential mode, D[15] is the sign bit, and D[2:0] becomes 0.

Conversion mode	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	Format
16-bit differential	S[1]	D[2]	D	D	D	D	D	D	D	D	D	D	D	D	D	D	Signed 2's complement
16-bit single-ended	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	Unsigned, 16-bit magnitude
13-bit differential	S	D	D	D	D	D	D	D	D	D	D	D	D	0	0	0	Signed 2's complement, left justified, zero extended
12-bit single-ended	0	D	D	D	D	D	D	D	D	D	D	D	D	0	0	0	Unsigned, zero in D[15] and D[2:0]

[1] Sign bit
[2] Data, 2's complement data when indicated

Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	VALID	0			CMDSRC				LOOPCNT				0		TSRC	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	D															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Function
31 VALID	FIFO Entry is Valid Indicates whether the FIFO entry is valid, which determines what happens to reads from RESFIFO. 0b - FIFO is empty. Discard any read from RESFIFO. 1b - FIFO contains data. FIFO record read from RESFIFO is valid.
30-28 —	Reserved
27-24	Command Buffer Source
CMDSRS	Indicates the executed command buffer that generated this result. 0000b - Not a valid value CMDSRC value for a data word in RESFIFO. 0h is only found in the initial FIFO state, prior to the storage of an ADC conversion result into a RESFIFO buffer. 0001b - CMD1 0010b-1110b - Corresponding command buffer used as control settings for this conversion. 1111b - CMD15
23-20	Loop Count Value

Field	Function
LOOPCNT	Indicates the loop count value during the command that generated this result. When CMDHn[LOOP] is non-zero, results are stored multiple times during command execution at the loop boundary. 0000b - Result is from initial conversion in command. 0001b - Result is from second conversion in command. 0010b-1110b - Result is from (LOOPCNT + 1) conversion in command. 1111b - Result is from 16th conversion in command.
19-18 —	Reserved
17-16 TSRC	Trigger Source Indicates the trigger source that initiated a conversion and generated this result. When multiple commands are chained together using CMDHn[NEXT], this field indicates the trigger source that started the command sequence. 00b - Trigger source 0 01b - Trigger source 1 10b - Trigger source 2 11b - Trigger source 3
15-0 D	Data Result Contains the result of an ADC conversion. The following for the data in D is summarized in Table 370.

2.2 MCX Nx4x eDMA block description

This section describes the MCX Nx4x eDMA block.

2.2.1 Overview

The eDMA is a highly programmable data-transfer engine optimized to minimize any required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known and is not defined within the transferred data itself.

The enhanced direct memory access (eDMA) controller can perform complex data transfers with minimal intervention from a host processor. The hardware microarchitecture includes:

- A DMA engine that performs:
  - Source address and destination address calculations
  - Data-movement operations
- Local memory containing transfer control descriptors for each of the 16 channels

Figure 3 illustrates the components of the eDMA system, including the eDMA module (engine).

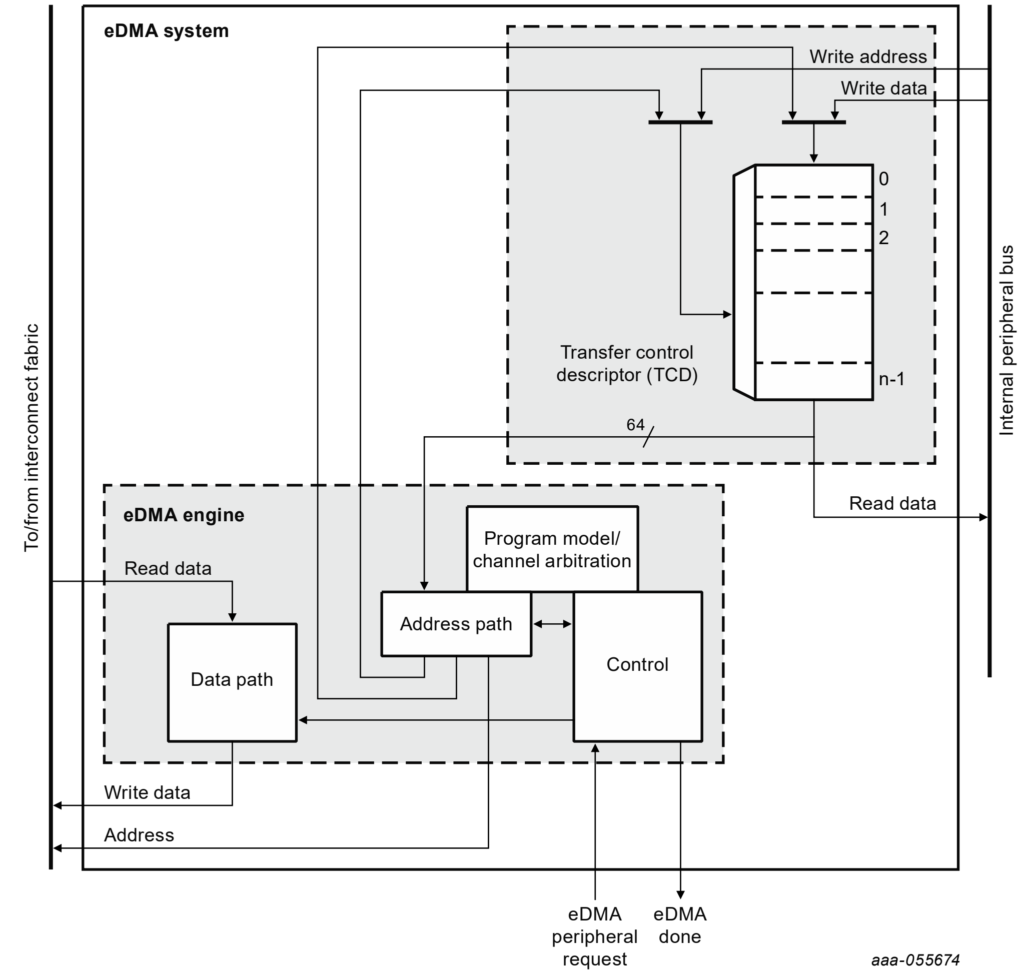

Figure 3. eDMA block diagram

For more Information, see the *MCX Nx4x Reference Manual* (document MCXNX4XRM) or MCX Nx4x: Unleashing the Power of eDMA Controller (document AN14300).

2.2.2 Major and minor

Each time a channel is activated and executes, a number of bytes, **NBYTES**, are transferred from the source to the destination. This is referred to as a minor transfer loop. A major transfer loop consists of a number of minor transfer loops. This number is specified within the TCD. As iterations of the minor loop are completed, the current iteration (CITER) TCD field is decremented. When the current iteration field has been exhausted, the channel has completed a major transfer loop.

Figure 4 shows the relationship between major and minor loops. In this example, a channel is configured so that a major loop consists of three iterations of a minor loop. The minor loop is configured to be a transfer of four bytes.

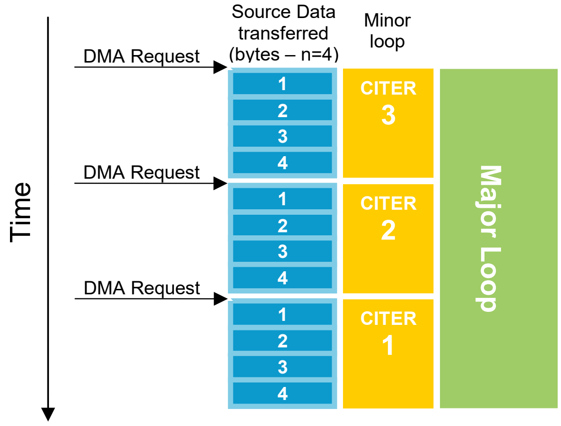

Figure 4. Major and minor transfer loops

The channel performs a selection of tasks upon completion of each minor and major transfer loop.

2.2.3 Completing a minor transfer loop

On completion of the minor loop, excluding the final minor loop, the eDMA carries out the following tasks:

- Decrementing the current iteration (CITER) counter.
- Updating the source address by adding the current source address to the signed source offset: SADDR = SADDR + SOFF (source address is updated automatically as transfers are performed. On completion of the minor loop, the source address contains the source address for the last piece of data that was read in the minor loop; offset is added to this value).
- Updating the destination address by adding the current destination address to the signed destination offset: DADDR = DADDR + DOFF.
- Updating channel status bits and requesting (enabled) interrupts.
- Asserting the start bit of the linked channel upon completion of a minor loop, if channel linking is enabled.

2.2.4 Completing a major transfer loop

On completion of the major/final minor loop, the eDMA performs the following:

- Updating the source address by adding the current source address to the last source address adjustment: SADDR = SADDR + SLAST
- Updating the destination address by adding the current destination address to the last destination address adjustment: DADDR = DADDR + DLAST
- Updating the channel status bits and requesting (enabled) interrupts
- Asserting the start bit of the linked channel upon completion of a minor loop, if the channel linking is enabled
- Reloading current iteration (CITER) from the beginning major iteration count (BITER) field

3 MCU features and peripheral settings

Figure 5 shows the eDMA data move procedure: 1->2…->5->6…, each Minor loop.

Finish 4ch Data (CH1- CH4) moving from ADC result FIFO into respective channel buffer. And the major loop finishes one block data (including 4ch data) moving (for example, one block contains all 4ch data and each channel contains 1024 data).

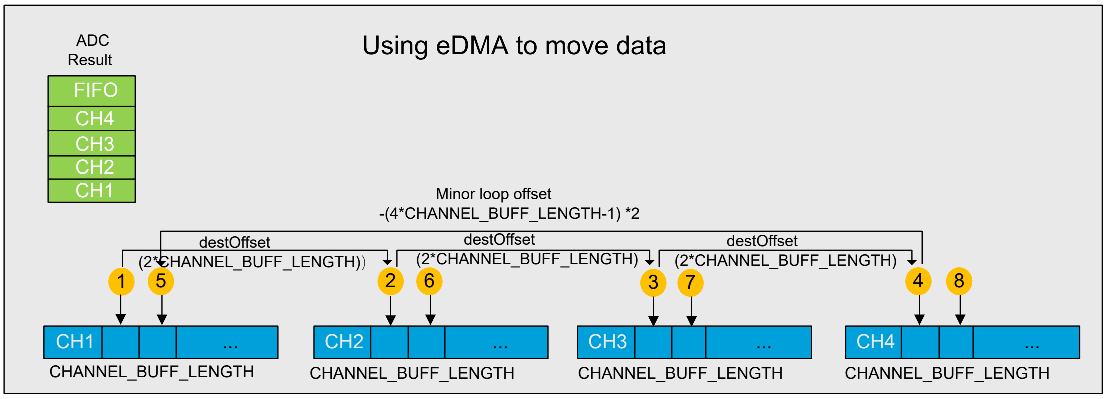

Figure 5. Data moving process diagram

3.1 MCU peripheral settings

Figure 6 shows using Ping-pong buffer to keep the sampling continuity. In Internal Service Routine (ISR), each channel data stored in a ping or pong buffer is copied to a permanent channel buffer.

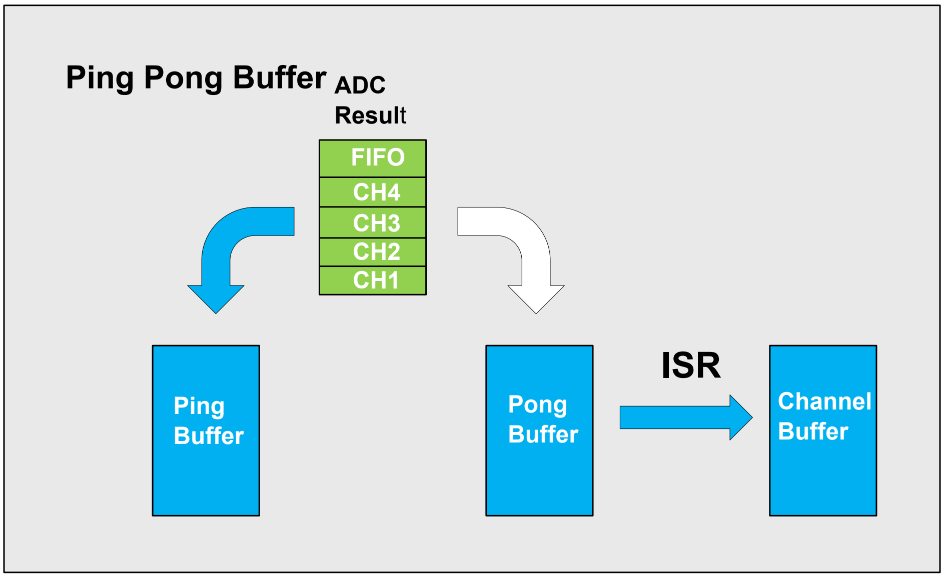

Figure 6. Ping Pong buffer process diagram

The example configuration code is listed as below.

```
static void EDMA0_Configuration(void)
{
    edma_transfer_config_t transferConfig[2];
    edma_channel_config_t lpadcDmaChnlConfig;
    edma_config_t userConfig;
```

```
    _destAddr_ping[0] = 0x5A5A;
    _destAddr_ping[1] = DEMO_CHANNEL_BUFF_LENGTH * 4;
    _destAddr_pong[0] = 0x5A5A;
    _destAddr_pong[1] = DEMO_CHANNEL_BUFF_LENGTH * 4;
    destAddr_ping = &_destAddr_ping[2];
    destAddr_pong = &_destAddr_pong[2];

    PRINTF("destAddr_ping = 0x%d, destAddr_ping[1] = 0x%d\r
\n",sizeof(destAddr_ping),sizeof(destAddr_ping[0]));

    lpadcDmaChnlConfig.channelDataSignExtensionBitPosition             = 0U;
    lpadcDmaChnlConfig.channelPreemptionConfig.enableChannelPreemption = false;
    lpadcDmaChnlConfig.channelPreemptionConfig.enablePreemptAbility    = true;
    lpadcDmaChnlConfig.channelRequestSource                            =
 DEMO_DMA_REQUEST;
    lpadcDmaChnlConfig.protectionLevel                                 =
 kEDMA_ChannelProtectionLevelUser;
#if !(defined(FSL_FEATURE_EDMA_HAS_NO_CH_SBR_SEC) &&
 FSL_FEATURE_EDMA_HAS_NO_CH_SBR_SEC)
    lpadcDmaChnlConfig.securityLevel = kEDMA_ChannelSecurityLevelNonSecure;
#endif /* !(defined(FSL_FEATURE_EDMA_HAS_NO_CH_SBR_SEC) &&
 FSL_FEATURE_EDMA_HAS_NO_CH_SBR_SEC) */

    /* Configure EDMA channel for one shot transfer */
    EDMA_GetDefaultConfig(&userConfig);
    EDMA_Init(DEMO_DMA_BASEADDR, &userConfig);

    EDMA_CreateHandle(&g_DMA0_Handle, DEMO_DMA_BASEADDR, DEMO_DMA_CHANNEL_0);
    EDMA_SetCallback(&g_DMA0_Handle, DMA0_Callback, NULL);
    EDMA_InstallTCDMemory(&g_DMA0_Handle, g_DMA0_Tcd, 2);

#if (defined(FSL_FEATURE_LPADC_FIFO_COUNT) && (FSL_FEATURE_LPADC_FIFO_COUNT ==
 2U))
    void *srcAddr = (uint32_t *)&(DEMO_LPADC_BASE->RESFIFO[0U]);
#else
    void *srcAddr = (uint32_t *)&(DEMO_LPADC_BASE->RESFIFO);
#endif /* (defined(FSL_FEATURE_LPADC_FIFO_COUNT) &&
 (FSL_FEATURE_LPADC_FIFO_COUNT == 2U)) */
    EDMA_PrepareTransfer(&transferConfig[0], srcAddr, sizeof(uint16_t),
 destAddr_ping, sizeof(destAddr_ping[0]), sizeof(destAddr_ping[0])*4,
    DEMO_CHANNEL_BUFF_LENGTH * 4 * 2, kEDMA_PeripheralToMemory);

    transferConfig[0].destOffset = DEMO_CHANNEL_BUFF_LENGTH*2;
    transferConfig[0].minorLoopOffset = (int32_t)(-1)*((DEMO_CHANNEL_BUFF_LENGTH
 * 4-1)*2);
    transferConfig[0].enableDstMinorLoopOffset = true;

    /* Used to change the destination address to the original value */
    transferConfig[0].dstMajorLoopOffset = (int32_t)((-1)
 * sizeof(destAddr_ping));

    EDMA_PrepareTransfer(&transferConfig[1], srcAddr, sizeof(uint16_t),
 destAddr_pong, sizeof(destAddr_pong[0]), sizeof(destAddr_pong[0])*4,
    DEMO_CHANNEL_BUFF_LENGTH * 4 * 2, kEDMA_PeripheralToMemory);

    transferConfig[1].destOffset = DEMO_CHANNEL_BUFF_LENGTH*2;
    transferConfig[1].minorLoopOffset = (int32_t)(-1)*((DEMO_CHANNEL_BUFF_LENGTH
 * 4-1)*2);
    transferConfig[1].enableDstMinorLoopOffset = true;
```

```
    /* Used to change the destination address to the original value */
    transferConfig[1].dstMajorLoopOffset = (int32_t)((-1)
  * sizeof(destAddr_pong));

    EDMA_SubmitLoopTransfer(&g_DMA0_Handle, transferConfig, 2);

    EDMA_InitChannel(DEMO_DMA_BASEADDR, DEMO_DMA_CHANNEL_0,
  &lpadcDmaChnlConfig);

    EDMA_EnableAutoStopRequest(DEMO_DMA_BASEADDR, DEMO_DMA_CHANNEL_0, false);

    EnableIRQ(DEMO_DMA_IRQ);
    EDMA_EnableChannelRequest(DEMO_DMA_BASEADDR, DEMO_DMA_CHANNEL_0);
}
```

4 Demo setup and CPU loading performance

This section describes the demo setup and CPU loading performance.

4.1 Demo setup

This section describes how to set up a demo.

4.1.1 Board connection

Connect FRDM-N947 boards to the analog Input source by **J1_M4** Arduino connector (the red part shows in the J1_M4). Connect **Channel 1 - Channel 4** to analog input source for 4CH analog input.

Use a USB type-C cable to connect to the FRDM board connector marked as MCU-Link. Download the code using the debug button in the tool bar after the compiler. Select **CMSIS-DAP** or **J-Link** in **Debug As** according to the firmware in your on-board debugger.

Figure 7 shows the board connection of this demo.

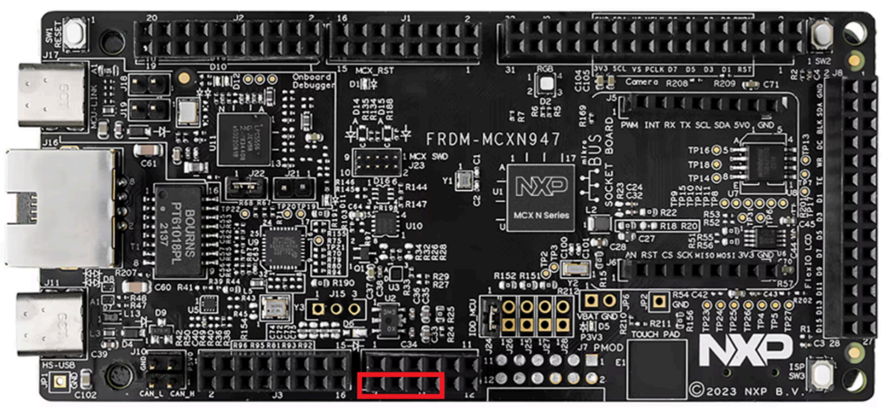

Figure 7. Demo connection

4.1.2 Project setup in MCUXpresso IDE

To clone the demo code from Application Code Hub in MCUXpresso IDE, perform the following steps:

1. Open MCUXpresso IDE. In the Quick Start Panel, choose **Import from Application Code Hub**.
2. Find the demo that you need by searching the name directly or selecting the tags you are interested in. Open the project, click the **GitHub link** and then **Next**.
3. Select the **main** branch and then click **Next**.
4. Select your local path for the repo in **Destination->Directory:** window. The MCUXpresso IDE clones the repo to the path that you have selected. Click **Next** after the clone process.
5. Select **Import existing Eclipse projects** in Wizard for the project import window and then **Next**.
6. Select the project in this repo (only one project in this repo) and then **Finish**.

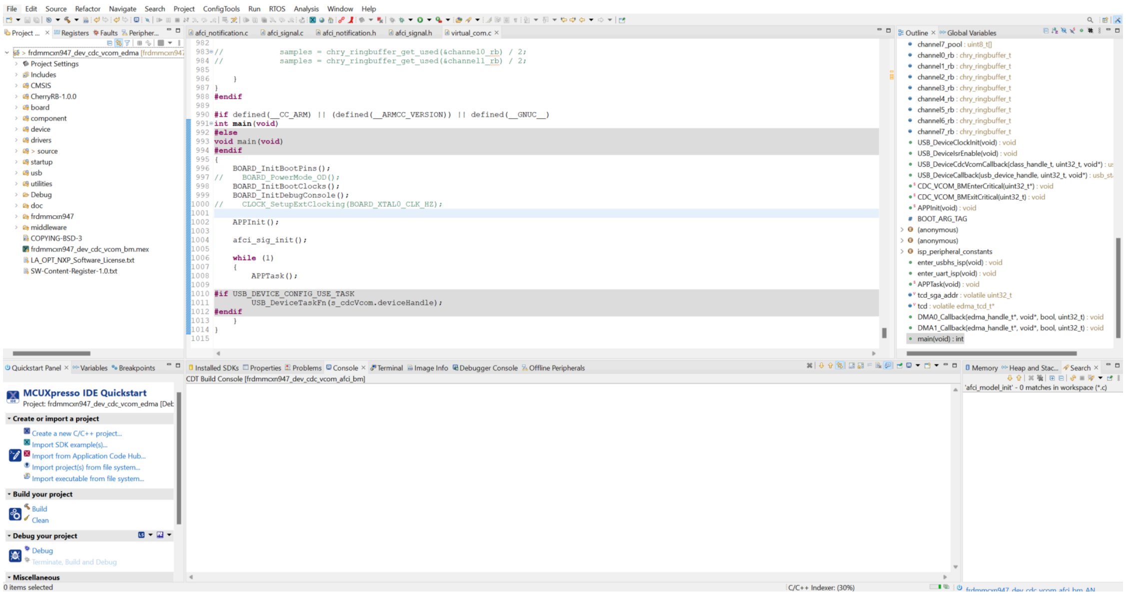

Figure 8. Import demo code

4.1.3 Run the demo

To reset the board, perform the following steps:

1. Press **SW3** on the FRDM board.
2. Debug the code and give the different analog input value.
3. Add to watch the channel data in the data buffer `channelX_pool` in the code.

5 Reference

See the following documents for more details:

- *MCX Nx4x Reference Manual* (document MCXNX4XRM)
- *MCX Nx4x Data Sheet* (document MCXNX4X-DS)
- *FRDM-MCXN947 Board User Manual* (document UM12018)
- *MCX Nx4x: Unleashing the Power of eDMA Controller* (document AN14300)
- *MPC57xx: Configuring and Using the eDMA Controller* (document AN4765)
- *Optimizing the S32K1xx eDMA for Performance Demanding Applications* (document AN12972)

Some of the documents listed above are available only under a non-disclosure agreement (NDA). To access such a document, contact a local NXP field applications engineer (FAE) or sales representative.

6 Note about the source code in the document

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

7 Revision history

Table 1 summarizes the revisions to this document.

Table 1. Revision history

Document ID	Release date	Description
AN14900 v.1.0	29 December 2025	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.

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.

CoolFlux — is a trademark of NXP B.V.

J-Link — is a trademark of SEGGER Microcontroller GmbH.

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

Contents

Date of release: 29 December 2025
Document identifier: AN14900