AN14151

MCX Nx4x MICFIL interface

Rev. 1 — 20 January 2024 Application note

Document information

Information	Content
Keywords	Nirvana MCX Nx4x, audio, micfil interface
Abstract	This application note provides details on how to configure and use the MICFIL interface using different mechanisms.

1 Introduction

This section provides general information about the product.

1.1 Device overview

The MCX Nx4x series microcontrollers combine two Arm Cortex M33 cores with a CoolFlux BSP32, a PowerQuad DSP coprocessor, an NPU, and multiple high-speed connectivity options running at 150 MHz. To support a wide variety of applications, the MCX N-series includes advanced serial peripherals, timers, high-precision analog, and state-of-the-art security features. All MCX Nx4x products include dual-bank flash that supports read while write operation from internal flash. The MCX Nx4x series also supports large external serial memory configurations.

The MCX Nx4x series is a dual-core microcontroller family. CPU0 is the primary Cortex-M33 (ver r0p4-00rel0) processor, which supports TrustZone-M, Floating Point Unit (FPU), and Memory Protection Unit (MPU).

The MCX Nx4x device also includes a second instance of Cortex-M33, CPU1, which is the secondary CM33 intended to offload work from the main processor to support special dedicated applications. The configuration of this instance does not include MPU, FPU, DSP, ETM, Trustzone-M, Secure Attribution Unit (SAU) or coprocessor interface. SYSTICK is supported on both cores.

Cortex-M33 implements a modified Harvard memory architecture using two 32-bit bus interfaces: the Code and System buses. The bus interfaces are activated by address range and can include both instruction fetches and operand data references on a given bus port. (A traditional Harvard architecture strictly separates instruction fetches and operand data references onto specific bus ports regardless of access address.) The Code bus is typically used for instruction fetching and data accesses of PC-relative data, while the system bus is typically used for operand data references to the on-chip and off-chip memories and peripheral accesses. The bus structure fully supports concurrent instruction fetch and data accesses, but the Cortex-M33 implementations can generate both types of references on each bus.

1.2 Micfil interface

The MCX Nx4x has one instance of the MICFIL module supporting for up to 2 pairs of microphones, indicating that 4 channels are available.

MICFIL delivers audio from microphones to the processor in several applications, such as mobile telephones. As current digital audio systems use a multibit audio signal (also known as multibit PCM) to represent the signal, this module implements the required digital interface (a series of filters) to transform a pulse density modulated (PDM) microphone bitstream into a 24-bit PCM signal in the audio band, at a configurable output sample rate. The MICFIL architecture is designed to save gate count and minimize power consumption. MICFIL can work in a multichannel mode. All channels have the same configuration, but one could independently turn on or turn off each input channel.

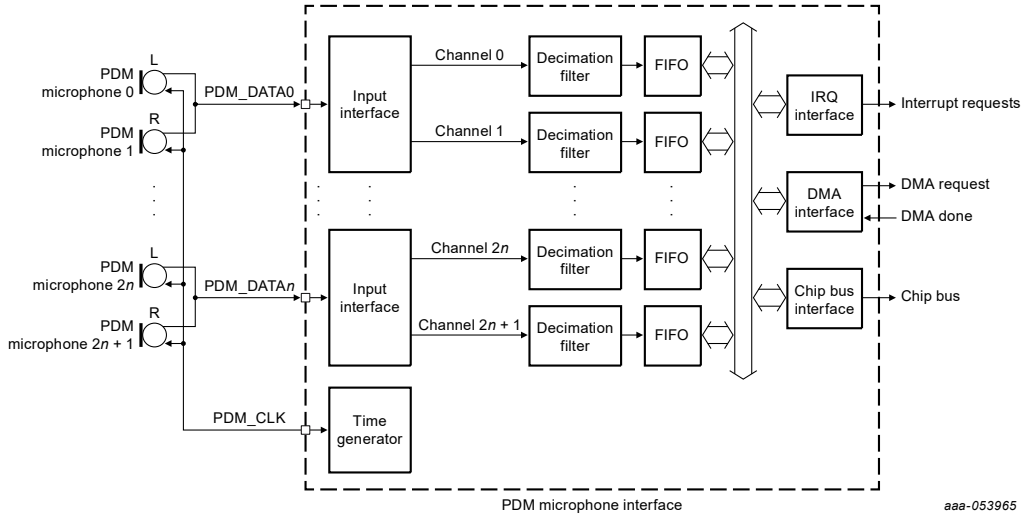

Figure 1. MICFIL block diagram

Here are some features available:

- Decimation filters:
  - Fixed-point filtering.
  - 24-bit PCM audio output.
  - Internal clock divider for a programmable PDM clock generation: clock divider bypass capability for low-frequency operations.
  - Frame synchronization.
  - Full or partial set of channel operations with individual enable controls.
  - Programmable decimation rate.
  - Programmable DC remover at output.
  - Range adjustment capability.
- FIFOs with interrupt and eDMA capability: each FIFO having a length of 16 entries.

MICFIL can deliver an interrupt request or eDMA request. Then, the chip or eDMA, respectively, could access the filter results stored in 16 entries FIFOs via the internal bus interface.

In this application note, one will see based on examples how to leverage the MICFIL coupled with eDMA, interrupts to send the audio stream to the SRAM for postprocessing.

Examples are based on the SDK version 2.13.1, using MCUXpresso v 11.7.

2 Interrupts based MICFIL

When enabled, an interrupt is triggered to indicate that filtering results are stored in the FIFO and ready to be processed. This interrupt is triggered when the FIFO of the enabled channel surpasses the watermark configurable level.

An error interrupt request can also be generated when an exceptional condition happens, such as an overflow or underflow in a channel output, or an overflow or underflow of a FIFO buffer.

The following example demonstrates how to leverage the interrupt mechanism to send data to the CPU.

In this example, MICFIL is clocked at 24 MHz, inherited from the FRO_HF (48 MHz) then divided by 2. However, many configurations are possible depending on the use case, the MICFIL clock can be attached to FRO_12M, PLL0, CLK_IN, PLL1, SAI0 for instance. Different clock sources (MICFIL_CLK_ROOT rate)

deliver different quality, and the MICFIL clock divider (CLKDIV) must be set accordingly to the targeted MICFIL frequency (output rate).

$$CLKDIV = \frac{MICFIL\_CLK\_ROOT\ rate}{8 \times OSR \times (output\ rate)}$$

Also only the right channel is enabled, with a FIFO watermark level of 4 and a sample rate of 16000. At the end of the example, the data stored in the SRAM array is displayed in the terminal.

Here is how to implement it:

- Set the MICFIL clock divider with the desired value and attach the FRO_HF clock to the MICFIL.

```
/* attach FRO HF to MICD */
CLOCK_SetClkDiv(kCLOCK_DivMicfilFClk, 2u);
CLOCK_AttachClk(kFRO_HF_to_MICFILF);
```

Figure 2. Example code to attach and configure the MICFIL clock and divider

- PDM initialization: PDM peripherals, PDM channels, PDM divider by writing to the CTRL_1 register.

```
int main(void)
{
    uint32_t i = 0U, j = 0U;

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

    /* attach FRO HF to MICD */
    CLOCK_SetClkDiv(kCLOCK_DivMicfilFClk, 2u);
    CLOCK_AttachClk(kFRO_HF_to_MICFILF);

    BOARD_InitPins();
    BOARD_PowerMode_OD();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    PRINTF("PDM interrupt example started! \n\r");

    memset(txBuff, 0U, sizeof(txBuff));

    /* Set up pdm */
    PDM_Init(DEMO_PDM, &pdmConfig);

    PDM_SetChannelConfig(DEMO_PDM, DEMO_PDM_ENABLE_CHANNEL_LEFT, &channelConfig);
    PDM_SetChannelConfig(DEMO_PDM, DEMO_PDM_ENABLE_CHANNEL_RIGHT, &channelConfig);
    if (PDM_SetSampleRateConfig(DEMO_PDM, DEMO_PDM_CLK_FREQ, DEMO_AUDIO_SAMPLE_RATE) != kStatus_Success)
    {
        PRINTF("PDM configure sample rate failed.\r\n");
        return -1;
    }
```

Figure 3. Example code to initialize PDM peripheral and channels

- Enable PDM interrupt, and PDM peripherals. Both error interrupt and FIFO interrupt are enabled by writing in the CTRL_1 register.

```
    PDM_EnableInterrupts(DEMO_PDM, kPDM_ErrorInterruptEnable | kPDM_FIFOInterruptEnable);

    EnableIRQ(PDM_EVENT_IRQn);
#if !(defined FSL_FEATURE_PDM_HAS_NO_INDEPENDENT_ERROR_IRQ && FSL_FEATURE_PDM_HAS_NO_INDEPENDENT_ERROR_IRQ)
    EnableIRQ(PDM_ERROR_IRQn);
#endif
    PDM_Enable(DEMO_PDM, true);

    /* wait data read finish */
    while (!s_dataReadFinishedFlag)
    {
#if (defined(FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ) && (FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ == 1U))
        if (s_lowFreqFlag)
        {
            s_lowFreqFlag = false;
            PRINTF("PDM root clock freq too low, please switch to a higher value\r\n");
        }
#endif
    }

    PRINTF("PDM recieve data:\n\r");
    for (i = 0U; i < SAMPLE_COUNT; i++)
    {
        PRINTF("%6x ", txBuff[i]);
        if (++j > 32U)
        {
            j = 0U;
            PRINTF("\r\n");
        }
    }

    PDM_Deinit(DEMO_PDM);
```

Figure 4. Example code to enable PDM interrupts

- Implement PDM interrupt service routine.

The PDM interrupt service routine, triggered when the FIFO level surpasses the configured watermark parameter, retrieves the FIFO data and stores it in an SRAM array. To do so:

- Check if the channel's FIFO has surpassed its watermark value by reading into the STAT[3-0] registers. If the value is 1, data is available in the DATACHn register.
- If it is the case, the data is stored in an SRAM array using a for loop reading the watermark level times the FIFO. Therefore, DATACHn is read consecutively, providing a word value stored at the top of the channel's n FIFO.

```
void PDM_EVENT_IRQHandler(void)
{
    uint32_t i = 0U, status = PDM_GetStatus(DEMO_PDM);
    /* recieve data */
    if ((1U << DEMO_PDM_ENABLE_CHANNEL_LEFT) & status)
    {
        for (i = 0U; i < DEMO_PDM_FIFO_WATERMARK; i++)
        {
            if (s_readIndex < SAMPLE_COUNT)
            {
                txBuff[s_readIndex] = PDM_ReadData(DEMO_PDM, DEMO_PDM_ENABLE_CHANNEL_LEFT);
                s_readIndex++;
            }
        }
    }

    if ((1U << DEMO_PDM_ENABLE_CHANNEL_RIGHT) & status)
    {
        for (i = 0U; i < DEMO_PDM_FIFO_WATERMARK; i++)
        {
            if (s_readIndex < SAMPLE_COUNT)
            {
                txBuff[s_readIndex] = PDM_ReadData(DEMO_PDM, DEMO_PDM_ENABLE_CHANNEL_RIGHT);
                s_readIndex++;
            }
        }
    }
    PDM_ClearStatus(DEMO_PDM, status);
    if (s_readIndex >= SAMPLE_COUNT)
    {
        s_dataReadFinishedFlag = true;
        PDM_Enable(DEMO_PDM, false);
    }
    __DSB();
```

Figure 5. Example code of the PDM interrupt service routine

Interrupt can also be generated when an exceptional condition happens. Therefore, handling this hypothesis in the interrupt service routine is implemented by reading the FIFO_STAT register. A value different to 0 can inform that a FIFO overflow or underflow has occurred.

In the PDM ISR:

```
    /* handle PDM error status */
#if (defined FSL_FEATURE_PDM_HAS_NO_INDEPENDENT_ERROR_IRQ && FSL_FEATURE_PDM_HAS_NO_INDEPENDENT_ERROR_IRQ)
    pdm_error_irqHandler();
#endif

static void pdm_error_irqHandler(void)
{
    uint32_t status = 0U;

#if (defined(FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ) && (FSL_FEATURE_PDM_HAS_STATUS_LOW_FREQ == 1U))
    if (PDM_GetStatus(DEMO_PDM) & PDM_STAT_LOWFREQF_MASK)
    {
        PDM_ClearStatus(DEMO_PDM, PDM_STAT_LOWFREQF_MASK);
        s_lowFreqFlag = true;
    }
#endif

    status = PDM_GetFifoStatus(DEMO_PDM);
    if (status != 0U)
    {
        PDM_ClearFIFOStatus(DEMO_PDM, status);
        s_fifoErrorFlag = true;
    }
#if defined(FSL_FEATURE_PDM_HAS_RANGE_CTRL) && FSL_FEATURE_PDM_HAS_RANGE_CTRL
    status = PDM_GetRangeStatus(DEMO_PDM);
    if (status != 0U)
    {
        PDM_ClearRangeStatus(DEMO_PDM, status);
    }
#else
    status = PDM_GetOutputStatus(DEMO_PDM);
    if (status != 0U)
    {
        PDM_ClearOutputStatus(DEMO_PDM, status);
    }

#endif
}

#if !(defined FSL_FEATURE_PDM_HAS_NO_INDEPENDENT_ERROR_IRQ && FSL_FEATURE_PDM_HAS_NO_INDEPENDENT_ERROR_IRQ)
void PDM_ERROR_IRQHandler(void)
{
    pdm_error_irqHandler();
    __DSB();
}
#endif
```

Figure 6. Example code of the PDM interrupt service routine in case of error detected

The example `mcxn5xxevk_pdm_interrupt` can be found in the MCUXpresso SDK for MCX Nx4x at `<SDK_LOCATION>\boards\<board_name>\driver_examples\pdm\.`

Figure 7 shows the result of running the project displayed in the terminal.

```
PDM interrupt example started!
PDM recieve data:
     0      0  1d800  7de00      0      0  1d800  7de00 30fc600 136fc000 10d99800 f5527400 30fc600 136fc000 10d99800 f5527400 390dc00 a74800 fd610a00 33aea00
 390dc00 a74800 fd610a00 33aea00 fcea1400 2856400 fe286800 1235200 fcea1400 2856400 fe286800 1235200 ff754200
 a7200 4b1600 ff7be000 ff754200  a7200 4b1600 ff7be000 969e00 ff690400 7ed400 ff9c4400 969e00 ff690400 7ed400 ff9c4400 3e6400 ffde1a00  3be00  c5800 3e6400 f
fde1a00  3be00  c5800 ffe40a00 1dc800 ffdee200 199c00 ffe40a00 1dc800 ffdee200 199c00 ffe92600  c4e00
ffff6f400 ffffe000 ffe92600  c4e00 ffff6f400 ffffe000   de00 fff8fa00  4c600 fff7ae00   de00 fff8fa00  4c600 fff7ae00  41000 fff9ba00  15400 ffffc9c00  41000 ff
f9ba00  15400 ffffc9c00 ffffeb200 ffffebc00 ffffd3200 ffff9c00 ffffeb200 ffffebc00 ffffd3200 ffff9c00 ffffce200 ffff8000 ffffd4a00
ffffeee00 ffffce200 ffff8000 ffffd4a00 ffffeee00 ffffde400 ffffe6000 ffffe5800 ffffe0e00 ffffde400 ffffe6000 ffffe5800 ffffe0e00 ffffe8a00 ffffdfa00 ffffe8400 ffffe1000 ffffe
8a00 ffffdfa00 ffffe8400 ffffe1000 ffffe6800 ffffe3000 ffffe4a00 ffffe4800 ffffe6800 ffffe3000 ffffe4a00 ffffe4800 ffffe3a00 ffffe5200 ffffe3600 ffffe5400
ffffe3a00 ffffe5200 ffffe3600 ffffe5400 ffffe3a00 ffffe4c00 ffffe4000 ffffe4600 ffffe3a00 ffffe4c00 ffffe4000 ffffe4600 ffffe4600 ffffe4400 ffffe4800 ffffe4200 ffffe4600 ffffe
4400 ffffe4800 ffffe4200 ffffe4800 ffffe4400 ffffe4600 ffffe4600 ffffe4800 ffffe4400 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600
ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe4600 ffffe
4600 ffffe4600 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800
ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe
4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4800 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4800 ffffe4a00 ffffe4a00
ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe
4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00 ffffe4a00
PDM interrupt example finished!
```

Figure 7. PDM interrupt example output on a serial terminal

3 eDMA based MICFIL

When enabled, the eDMA transfer is triggered when the PDM FIFO surpasses the watermark parameter configured. In general, if there is at least one enabled channel and all FIFOs of all enabled channels reach their watermark levels, the output interface makes a request for a eDMA transaction. It discharges the CPU of handling this process, and can be done even when the system is in low power.

The following example demonstrates how to leverage the eDMA mechanism to transfer data to the SRAM.

In this example the MICFIL is clocked at 24 MHz, inherited from the FRO_HF (48 MHz) then divided by 2, the same as the previous example.

Also only the right channel is enabled, with a FIFO watermark level of 4 and a sample rate of 16000. At the end of the example, data stored in the SRAM array is displayed in the terminal.

As the interrupt mechanism is replaced by the eDMA, the PDM ISR is not implemented and only the interrupt in case of FIFO error is enabled. The PDM ISR handling the possible FIFO underflow/overflow errors is the same as the previous example.

```
#define DEMO_PDM                PDM
#define DEMO_PDM_ERROR_IRQn     PDM_EVENT_IRQn

PDM_EnableInterrupts(DEMO_PDM, kPDM_ErrorInterruptEnable);
EnableIRQ(DEMO_PDM_ERROR_IRQn);
```

Figure 8. Example code to enable PDM error interrupt

The MCX Nx4x provides two 16 channel eDMA controllers, and Channel 0 of controller 0 is used. This channel is configured to be connected with the MICFIL, by writing into the DMA0->CH0_MUX register the value of the MICFIL source number.

Table 1. DMAMUX0 request assignments

DMAMUX number	Alias	Source description
9	CTIMER1	DMAREQ_M0
10	CTIMER1	DMAREQ_M1
11	CTIMER2	DMAREQ_M0
12	CTIMER2	DMAREQ_M1
13	CTIMER3	DMAREQ_M0
14	CTIMER3	DMAREQ_M1
15	CTIMER4	DMAREQ_M0
16	CTIMER4	DMAREQ_M1
17	wuuo	Wakeupevent
18	MICFIL0	FIFO_request

eDMA peripheral must be as well initialized, and an eDMA handle for DMA0 channel 0 must be create to store callback function and parameters. Here, the default configuration is used.

eDMA initialization is done by writing into the MP_CSR register. The Management Page Control (MP_CSR) register defines the basic operating configuration of the DMA.

```
#define DEMO_EDMA               DMA0
#define DEMO_PDM_EDMA_CHANNEL   kDmaRequestMuxMicfil0FifoRequest
#define DEMO_EDMA_CHANNEL       0
#define DEMO_AUDIO_SAMPLE_RATE 16000
#define BUFFER_SIZE (256)

AT_NONCACHEABLE_SECTION_ALIGN(edma_handle_t dmaHandle, 4);
    /* Create EDMA handle */
    /*
     * dmaConfig.enableRoundRobinArbitration = false;
     * dmaConfig.enableHaltOnError = true;
     * dmaConfig.enableContinuousLinkMode = false;
     * dmaConfig.enableDebugMode = false;
     */
    EDMA_GetDefaultConfig(&dmaConfig);
    EDMA_Init(DEMO_EDMA, &dmaConfig);
    EDMA_CreateHandle(&dmaHandle, DEMO_EDMA, DEMO_EDMA_CHANNEL);
#if defined(FSL_FEATURE_EDMA_HAS_CHANNEL_MUX) && FSL_FEATURE_EDMA_HAS_CHANNEL_MUX
    EDMA_SetChannelMux(DEMO_EDMA, DEMO_EDMA_CHANNEL, DEMO_PDM_EDMA_CHANNEL);
#endif
```

Figure 9. Example code of eDMA initialization

The PDM must be initialized accordingly by:

- Initializing the PDM peripheral.

```
#define DEMO_PDM                    PDM
#define DEMO_PDM_FIFO_WATERMARK     (4)
#define DEMO_PDM_QUALITY_MODE       kPDM_QualityModeHigh
#define DEMO_PDM_CIC_OVERSAMPLE_RATE (0U)
static const pdm_config_t pdmConfig = {
    .enableDoze        = false,
    .fifoWatermark     = DEMO_PDM_FIFO_WATERMARK,
    .qualityMode       = DEMO_PDM_QUALITY_MODE,
    .cicOverSampleRate = DEMO_PDM_CIC_OVERSAMPLE_RATE,
};
    /* Set up pdm */
    PDM_Init(DEMO_PDM, &pdmConfig);
```

Figure 10. Example code of PDM initialization and configuration

- Initializing the PDM Rx eDMA handle and link it to the eDMA handle to transmit the PDM data. A callback is defined as well, that will be called once the DMA transfer is completed.

```
static void pdmEdmallback(PDM_Type *base, pdm_edma_handle_t *handle, status_t status, void *userData);

AT_NONCACHEABLE_SECTION_ALIGN(pdm_edma_handle_t pdmRxHandle, 4);

PDM_TransferCreateHandleEDMA(DEMO_PDM, &pdmRxHandle, pdmEdmallback, NULL, &dmaHandle);
```

Figure 11. Example code of PDM eDMA interrupt service routine

- Installing the EDMA descriptor memory (TCD i.e Transfer Control Descriptor) of the PDM Rx EDMA handle. The TCD is a structure for each eDMA channel containing the information of the eDMA transfer and its attributes.

```
AT_QUICKACCESS_SECTION_DATA_ALIGN(edma_tcd_t s_edmaTcd[1], 32U);
PDM_TransferInstallEDMATCDMemory(&pdmRxHandle, s_edmaTcd, 1);
```

Figure 12. Example code to create a PDM eDMA descriptor memory

- Configuring the PDM channel, here only the channel left (0) is used. The configuration is user-defined, Decimation Filter Output Gain, DC remover cut off frequency, PDM output DC remover cut-off frequency are configurable and is done by writing in the PDM CTRL_1 register.

```
#define DEMO_PDM_ENABLE_CHANNEL_LEFT  (0U)
static const pdm_channel_config_t channelConfig = {
#if (defined(FSL_FEATURE_PDM_HAS_DC_OUT_CTRL) && (FSL_FEATURE_PDM_HAS_DC_OUT_CTRL))
    .outputCutOffFreq = kPDM_DcRemoverCutOff40Hz,
#else
    .cutOffFreq = kPDM_DcRemoverCutOff152Hz,
#endif
#ifdef DEMO_PDM_CHANNEL_GAIN
    .gain = DEMO_PDM_CHANNEL_GAIN,
#else
    .gain       = kPDM_DfOutputGain4,
#endif
};
PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &pdmRxHandle, DEMO_PDM_ENABLE_CHANNEL_LEFT, &channelConfig);
```

Figure 13. Example code to configure the PDM channel

- Configuring the PDM sample rate by writing to the PDM CTRL_2 register.

```
#define DEMO_PDM_CLK_FREQ            CLOCK_GetMicfilClkFreq()
#define DEMO_AUDIO_SAMPLE_RATE 16000
if (PDM_SetSampleRateConfig(DEMO_PDM, DEMO_PDM_CLK_FREQ, DEMO_AUDIO_SAMPLE_RATE) != kStatus_Success)
{
    PRINTF("PDM configure sample rate failed.\r\n");
    return -1;
}
```

Figure 14. Example code to configure the PDM sample rate

- Initializing and performing an eDMA transfer. This mechanism is flexible and allows multiple transfer possibilities. For instance, multi pdm channel data can be sent (in an interleave manner per channel or per block), it support static/dynamic scatter gather cases using the link transfer feature. In this example a simple transfer is configured, where the destination is an array of size 256 storing 2-bytes data.

```
AT_NONCACHEABLE_SECTION_ALIGN(static int16_t rxBuff[BUFFER_SIZE], 4);
xfer.data          = (uint8_t *)rxBuff;
xfer.dataSize      = BUFFER_SIZE * 2U;
xfer.linkTransfer = NULL;

PDM_TransferReceiveEDMA(DEMO_PDM, &pdmRxHandle, &xfer);
```

Figure 15. Example code to initialize and configure the eDMA transfer

The DMA transfer is then triggered only when the PDM FIFO surpasses the watermark parameter. When the DMA is completed, the PDM eDMA callback previously defined will be called.

The example `mcxn5xxevk_pdm_edma_transfer` can be found in the MCUXpresso SDK for MCX Nx4x at `<SDK_LOCATION>\boards\<board_name>\driver_examples\pdm\`.

Figure 16 shows the result of running the project displayed in the terminal.

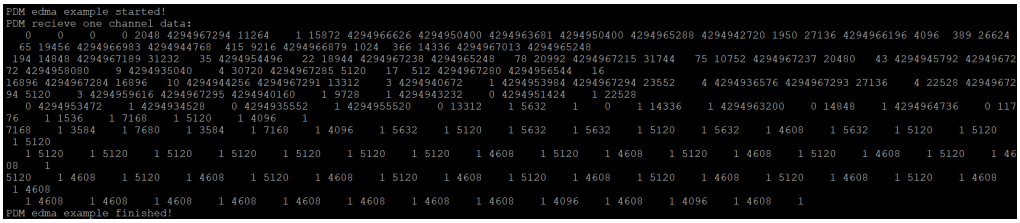

Figure 16. PDM with eDMA transfer example output on a serial terminal

4 Conclusion

This application note provides a quick look at the MICFIL module in the MCX Nx4x device. It briefly introduces how to leverage the MICFIL interrupts or eDMA to handle data management to the memory based on MCUX Nx4x SDK examples. These mechanisms can be used in many use cases more or less complex, for instance sending data to a codec using the SAI interface with I2S, therefore bypassing the CPU.

5 Note about the source code in the document

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

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN

ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6 Revision history

Table 2. Revision history

Document ID	Release date	Description
AN14151 v.1.0	20 January 2024	Initial version

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.

Contents

For more information, please visit: https://www.nxp.com

Date of release: 20 January 2024
Document identifier: AN14151