AN14300

MCX Nx4x: Unleashing the Power of eDMA Controller

Rev. 1.0 — 23 September 2024 Application note

Document information

Information	Content
Keywords	AN14300, eDMA, bus master, DMA, MCX Nx4x
Abstract	This application note provides a working knowledge by covering the following topics: introduction and overview of eDMA controllers, features of the MCX Nx4x eDMA module, interaction between the eDMA and DMA multiplexer (DMAMUX) and configuration advice for applications along with examples.

1 Introduction

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, which makes it ideal for demanding applications from motor control and industrial automation to audio processing and communication systems. However, efficient data management is crucial for these applications to achieve optimal performance.

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 application note provides a working knowledge by covering the following topics: introduction and overview of eDMA controllers, features of the MCX Nx4x eDMA module, interaction between the eDMA and DMA multiplexer (DMAMUX), and configuration advice for applications. Examples are used throughout this document to demonstrate increasingly complex eDMA configurations.

1.1 eDMA controller overview

An eDMA controller supports to move data from one memory- mapped location to another without CPU intervention. Once configured and initiated, the eDMA controller operates in parallel to the Central Processing Unit (CPU), performing data transfers, that would otherwise have been handled by the CPU. This results in a reduced CPU loading and a corresponding increase in system performance. Figure 1 illustrates the functionality provided by a DMA controller.

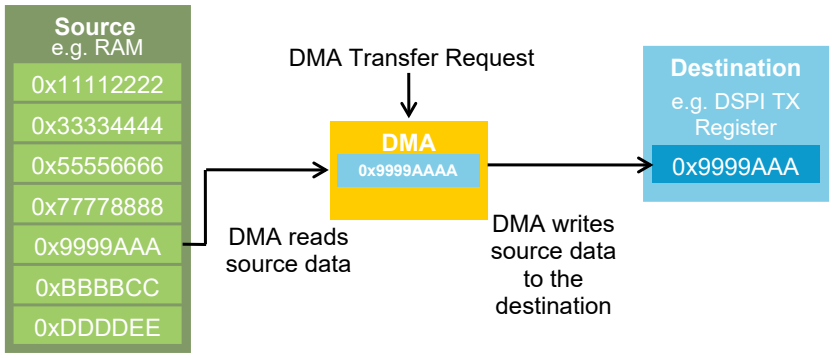

Figure 1. eDMA operational overview

1.2 MCX Nx4x eDMA block description

The eDMA module is partitioned into two major modules (Figure 2):

1. The eDMA Engine
2. Local memory containing Transfer Control Descriptors (TCDs) for each of the channel.

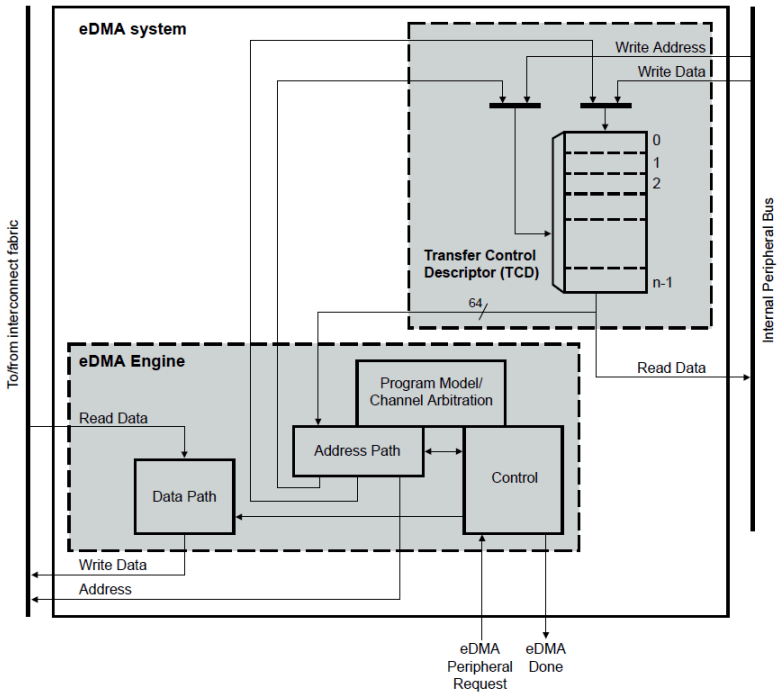

Figure 2. eDMA block diagram

Details of the further partitioned submodules are as below.

1.2.1 Address path

- Implements the primary and secondary channel.
- Responsible for generating source and destination memory addresses for each transfer.
- Allows the data transfer associated with one channel to be pre-empted by a higher priority channel.
- Supports various addressing modes like incrementing, decrementing, fixed, and scatter-gather.
- Allows configuration of transfer size and alignment for optimal performance.

1.2.2 Data path

- Implements the bus master read/write data path.
- Internal read, write data bus are the primary input and output.
- Integrates hardware flow control to prevent data overflows or underflows.

1.2.3 Program/Channel arbitration

- Manages access to the eDMA engine by multiple channels simultaneously.
- Uses priority levels to ensure that the critical data gets processed first.
- Includes round-robin scheduling for fair allocation of resources.

1.2.4 Control

- Orchestrates the entire eDMA control functions, including initiating transfers, managing interrupts, and handling errors.
- Provides various control registers for configuration and monitoring.

1.2.5 TCD

- Each eDMA channel has a dedicated Transfer Control Descriptor (TCD) module of 32 bytes.
- TCD is further partitioned into memory controller and memory array:
  - The memory controller manages the access both from the eDMA engine and from the internal peripheral bus.
  - The memory array stores each channel transfer profile like source, destination address, offset, and attributes. These details are specified in the channel TCD registers.

1.3 MCX Nx4x eDMA controller features

MCX Nx4x devices feature a two 16-channel DMA controller. Each channel can be independently configured with the details of the transfer sequence that is to be executed. The DMAMUX for MCX Nx4x series allows up to 128 DMA request signals (six unused signals are reserved) to be mapped to each channel.

eDMA transfers can be activated in three ways:

1. Peripheral paced hardware requests.
2. Software initiation.
3. Channel-to-channel linking – On completion of a transfer, one channel activates another.

Each channel can generate interrupts to indicate that it has partially completed or fully completed a transfer. Interrupts can also be generated to indicate that a transfer error has occurred.

Scatter/Gather processing is supported by each of the 16 channels. This feature allows a channel to self-load a new TCD when it has performed the transfer for its current configuration. This feature enables extensive transfer sequences beyond 16 via dynamically defining and employing transfer sequences.

1.4 eDMA architectural integration

To allow the eDMA, CPUs, and other masters to operate simultaneously, a multi-master bus architecture is implemented. The MCX Nx4x chips feature multiple bus masters: for example, cores, fast Ethernet controller, and LFAST.

The crossbar switch (XBAR) forms the heart of this multi-master architecture. It links each master to the required slave device. Many devices in the MCX Nx4x family feature a dual-crossbar architecture. In this case, data passes from one crossbar to the next if a master requires access to a slave that is not on the same crossbar as itself. If two or more masters attempt joint access to the same slave, an arbitration scheme commences, eliminating the risk of bus contention. Both fixed-priority and round-robin arbitration schemes are available.

The crossbar switch and interaction between bus masters and slave devices is illustrated in a simplified version in Figure 3. In this example, the eDMA controller is accessing one of the peripherals on the IP bus while the CPU is concurrently accessing the SRAM memory. The crossbar switch has formed the appropriate connections for this situation.

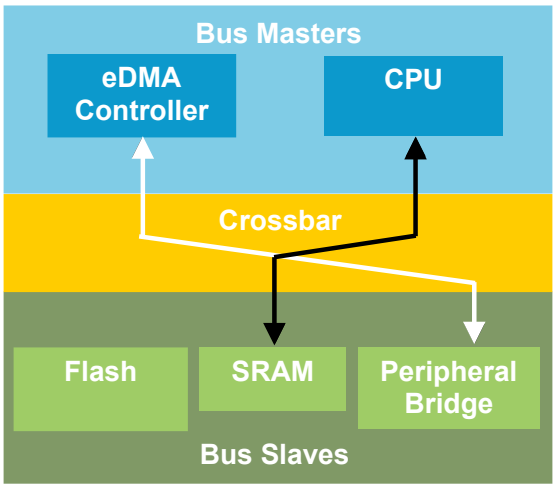

Figure 3. Multi-master bus architecture

2 eDMA data Flow

The basic flow of data transfer can be partitioned into three segments, which are explained as below:

- **Initiating the Transfer**
  - **Requesting Service**
    As shown in Figure 4, the selected channel requests service by asserting the eDMA peripheral request signal. This trigger is like the software-initiated flow where the `TCDn_CSR[START]` field is used.
  - **Routing the Request**
    The internal eDMA engine receives the request and routes it through the control module, program model, and channel arbitration unit.
  - **Channel Arbitration**
    In the next cycle, a fixed-priority algorithm, optionally combined with round-robin, chooses the activated channel.
  - **Accessing TCD Descriptor**
    The selected channel number is then used to access and read the corresponding TCD descriptor from the 64-bit wide TCD memory. This descriptor is loaded into primary or secondary channel execution registers of the address path.

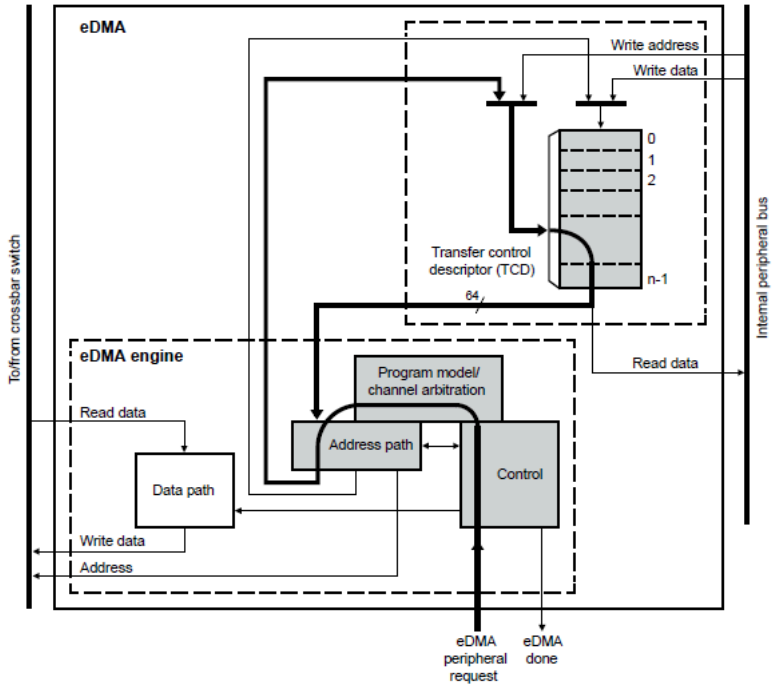

Figure 4. Initializing the transfer

- **Performing the data transfer**
  - **Data movement**

    As presented in Figure 5, modules like the address path, data path, and control unit work together to perform the actual data transfer as defined in the TCD.

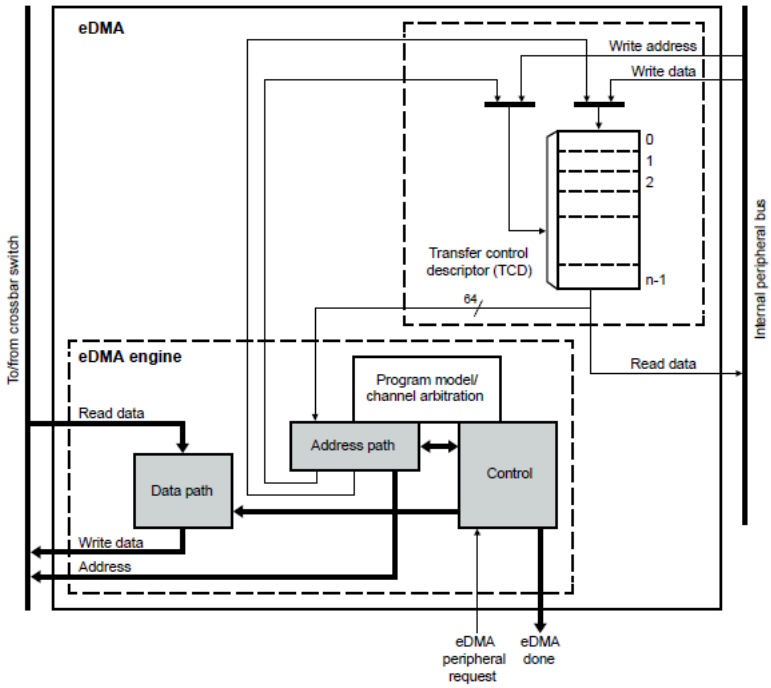

Figure 5. eDMA data transfer

  - **Source reads**

Source reads are initiated, and the fetched data is temporarily stored in the data path block before being transferred to the destination.

– **Destination writes**
This source read/destination write cycle continues until all NBYTES of data are transferred.

- **Completing the transfer**
  – **TCD updates**
  As described in Figure 6, after NBYTES are transferred, the address path logic updates specific fields in the TCD based on the operation, like SADDR, DADDR, and CITER.
  – **Major iteration completion**
  If the major iteration count reaches its limit, more operations occur, including final address adjustments and reloading BITER into CITER.
  – **Optional interrupt and scatter/gather**
  An optional interrupt request might be asserted. If the scatter/gather is enabled, a new TCD could be fetched from memory using the provided pointer.

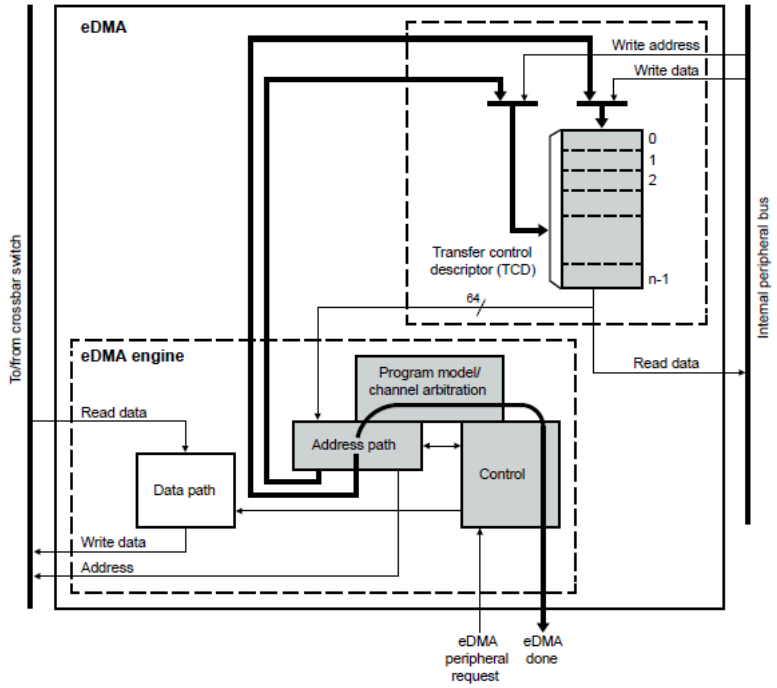

Figure 6. Completing the eDMA transfer

2.1 Channel activation

- Events occurring within other peripheral modules can be enabled to activate eDMA transfers. In many modules, event flags can be asserted as either eDMA or interrupt requests. Due to the high number of sources for those requests, a configurable multiplexer (DMAMUX) is implemented to route peripheral DMA requests to DMA channels.
- Channels may also be activated by software. The TCDs of the channels provide a `START` bit that activates the channel when asserted. This makes it possible to activate each channel in software. The `START` bit also provides a useful tool for testing and debugging the TCD, making it possible to assess if the channel performed the expected transfers each time it is activated.
- Channel linking provides the means for one channel to assert the `START` bit of another channel. The linked channel can be activated at stages of the transfer or on completion of the transfer.

2.2 Channel arbitration

The peripheral request line for each channel of DMA is driven from a Direct Memory Access Multiplexer (DMAMUX). This is a flexible configuration that allows the user to select the appropriate peripheral to connect to each channel of the DMA controller. The DMA multiplexer is used to route the numerous peripheral DMA sources to individual DMA channels, and it allows up to 128 DMA request signals (six are reserved for future use). There are two DMA mux instances available, each of which is able to route sources to sixteen DMA channels.

Table 1. eDMA transfer request sources

DMAMUX instance	DMA channel
0	0–15
1	0-15

The DMA multiplexer (DMA_Mux) performs the task of routing the peripheral DMA request sources to the desired channel.

Table 2. Peripheral DMA requests on MCX Nx4x

DMAMUX number	DMAMUX0 - DMAMUX1 (Source descriptor)	DMAMUX0 - DMAMUX1 alias
0	Disabled	—
1	Receive event	FlexSPI0
2	Transmit event	FlexSPI0
3-6	INT0 - INT3	PINT0
7-8 to 15-16	DMAREQ_M0 - DMAREQ_M1	CTIMER0 to CTIMER4
17	Wake up event	WUU0
18	FIFO_request	MICFIL0
19-20	DMA0-DMA1	SCT0
21-22	FIFO A, FIFO B request	ADC0
23-24	FIFO A, FIFO B request	ADC1
25-27	FIFO_request	DAC0 to DAC2
28-30	DMA_request	CMP0 to CMP2
31-32 to 37-38	OUT0A-OUT0B to OUT3A-OUT3B	EVTG0
39-42	Req_Capt0 – Req_Capt3	PWM0
43-46	Req_val0 – Req_val3	PWM0
47-50	Req_capt0 – Req_capt3	PWM1
51-54	Req_val0 – Req_val3	PWM1
55, 56	—	Reserved
57, 58	Counter match event	LPTMR0 to LPTMR1
59, 60	DMA request	CAN0 to CAN1
61-68	Shifter(0-7) Status DMA request OR Timer(0-7) Status DMA request	FlexIO0
69-70 to 87-88	Receive and Transmit request	LP_FLEXCOMM0 to LP_FLEXCOMM9

Table 2. Peripheral DMA requests on MCX Nx4x*...continued*

DMAMUX number	DMAMUX0 - DMAMUX1 (Source descriptor)	DMAMUX0 - DMAMUX1 alias
89, 90	—	Reserved
91-92 to 93-94	Receive-Transmit request	EVMSIM0 to EVMSIM1
95-96 to 97-98	Receive-Transmit request	I3C0 to I3C1
99-100 to 101-102	Receive-Transmit request	SAI0 to SAI1
103-107	Ipd_req_sinc[0-4] or ipd_req_alt[0-4]	SINC0
108-109 to 118-119	Pin event request 0-Pin event request 1	GPIO0 to GPIO5
120	End of Scan	TSI0
121	Out of Range	TSI0

2.3 Modes of operation

The eDMA supports two modes, which are explained as below:

- **Debug mode**: In this mode, the DMA channel is disabled. Since DMA channels are disabled and enabled primarily via the DMA configuration registers, this mode is used mainly as the reset state for a DMA channel in the DMA Channel Mux. It may also be used to temporarily suspend a DMA channel while reconfiguration of the system takes place.
- **Normal mode**: In this mode, a DMA source is routed directly to the specified DMA channel. The operation of the DMA Mux in this mode is completely transparent to the system. A DMA source and a destination (such as a peripheral result or transmit buffer) requests/triggers the start of a DMA transfer when it is ready to receive or transfer data.

3 Transfer process

Prior to configuring the eDMA, it is useful to understand how the eDMA performs a transfer.

3.1 Handling multiple transfer requests

Only one channel can actively perform a transfer at a given time. Therefore, to handle multiple pending transfer requests the eDMA controller offers channel prioritization. Fixed-priority or round-robin prioritization can be selected.

In the fixed-priority scheme, each channel is assigned a priority level. When multiple requests are pending, the channel with the highest priority level performs its transfer first. By default, the fixed priority arbitration is implemented, with each channel being assigned a priority level equal to its channel number. Other priority levels can be assigned if required. Higher priority channels can preempt lower priority channels. Preemption occurs when a channel is performing a transfer while a transfer request is asserted to a channel of a higher priority. In this case, the lower priority channel halts its transfer and allows the channel of higher priority to carry out its transfer. The lower priority channel then resumes its transfer when the higher priority channel has completed its transfer. One level of preemption is supported. Preemption is an option and must be enabled on a per-channel basis if required.

In round-robin mode, the eDMA cycles through the channels in order, checking for a pending request. When a channel with a pending request is reached, it is allowed to perform its transfer. When the transfer has been completed, the eDMA continues to cycle through the channels looking for the next pending request.

3.2 Major and minor transfer loops

Each time a channel is activated and executes, several 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 few 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 7 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.

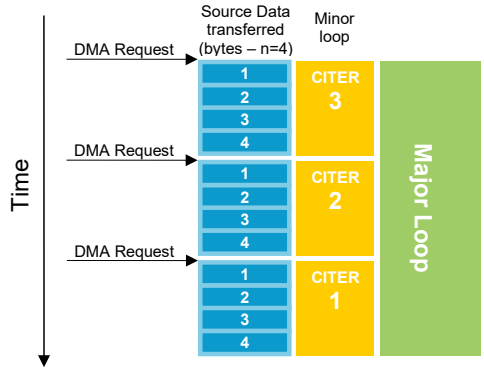

Figure 7. Major and minor transfer loops

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

3.3 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.

3.4 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

Note:

There are two ways to test the minor loop completion:

- ***Hardware***
- ***Software***

The TCD status fields execute the following sequence for a hardware-activated channel:

Table 3. TCD status field

Stage	TCDn_CSR field	CHn_CSR fields		State
	START	ACTIVE	DONE	
1	0	0	0	Initiate channel service request via hardware (peripheral request asserted).
2	0	1	0	The channel is executing.
3a	0	0	0	The channel has completed the minor loop and is idle.
3b	0	0	1	The channel has completed the major loop and is idle.

The best method to test for minor-loop completion when using hardware-initiated service requests is to read the TCDn_CITER field and test for a change.

Whereas in software initiated service request the TCDn_CSR[START] field is **1** at Stage 1.

For both activation types, the major-loop-complete status is explicitly indicated via the CHn_CSR[DONE] field.

The TCDn_CSR[START] field is cleared to **0** automatically when the channel begins execution, regardless of how the channel activates.

3.5 Channel linking

Channel linking (or chaining) is a mechanism in which one channel sets the TCDn_CSR[START] field of another channel (or itself), thus initiating a service request for that channel. When properly enabled, the eDMA engine automatically performs this operation at the major or minor loop completion.

4 Dynamic programming

This method is used to change the programming model during channel execution. To follow for changing the group or channel priority levels, perform the below steps:

- Halt the DMA by writing **1** to the CSR[HALT] field.
- Change the group or channel priorities as wanted.
- Enable normal DMA operations by writing 0 to the CSR[HALT] field.

Use the following coherency model when executing a dynamic channel link request.

1. Write 1 to the TCDn_CSR[MAJORELINK] field.
2. Read back to the TCDn_CSR[MAJORELINK] field.
3. Test the TCDn_CSR[MAJORELINK] request status:
   - If TCDn_CSR[MAJORELINK] = 1, the dynamic link attempt was successful.
   - If TCDn_CSR[MAJORELINK] = 0, the attempted dynamic link did not succeed (the channel was already retiring).

4.1 Dynamic scatter/gather

Scatter/gather is the process of automatically loading a new TCD into a channel. It also allows a DMA channel to use multiple TCDs which:

- Enables using multiple data transfer configurations for a single DMA channel.
- Allows the **scattering** data to multiple destinations or the **gathering** data from multiple sources.
- Automatically loads a new transfer configuration (TCD) when the current one finishes.

A coherency model is needed as configuration can be changed during execution

Why Coherency?

- If a user wants to execute a dynamic scatter/gather operation by enabling the `TCDn_CSR[ESG]` field and at the same time the eDMA engine is retiring a channel then it is unclear whether the actual scatter/gather request is honored or not.

Recommended solutions

1. Force Zero on `TCDn_CSR[ESG]` after the completion:
   - Whenever users write to `TCDn_CSR`, make sure that `TCDn_CSR[ESG]` is cleared to **0** after the channel finishes its major loop (`CHn_CSR[DONE] = 1`).
2. Clear `DONE` before setting ESG:
   - To set `ESG`, clear the `DONE` field first.

Benefits

- Enables complex data transfers without needing multiple DMA channels.
- Improves flexibility and efficiency for specific DMA operations.

5 Configuring the eDMA

This section covers some of the important configuration steps and register fields. For full details of all the register fields, consult the reference manual of the microcontroller.

5.1 Configuration steps

To configure the eDMA, perform the following initialization steps:

1. Program the control register (`MP_CSR`). This step is necessary only if a configuration other than the default is required.
2. Configure the channel priority register (`CHn_PRI`) and group priority level (`CHn_GRPRI`) This step is necessary only if a configuration other than the default is required.
3. Enable error interrupts `CHn_CSR[EEI]`using either the `DMAEEI` or `DMASEEI` register. This step is necessary only if a configuration other than the default is required.
4. Write the transfer control descriptors (`eDMA_TCDn`) for all channels that are used. Configure TCDs for the scatter/gather mechanism if required.
5. Configure the appropriate peripheral module and configure the DMAMUX to route the activation signal to the appropriate channel.

5.2 Transfer Control Descriptors (TCD)

All transfer attributes for a channel are defined in the unique TCD of the channel. Each TCD is stored in the local SRAM of the eDMA controller. Only the DONE, ACTIVE, and STATUS fields are initialized at reset. All other TCD fields are undefined at reset and must be written to by software before the channel is activated. Failure to do this may result in unpredictable behavior of the channel. Figure 8 shows the TCD memory map.

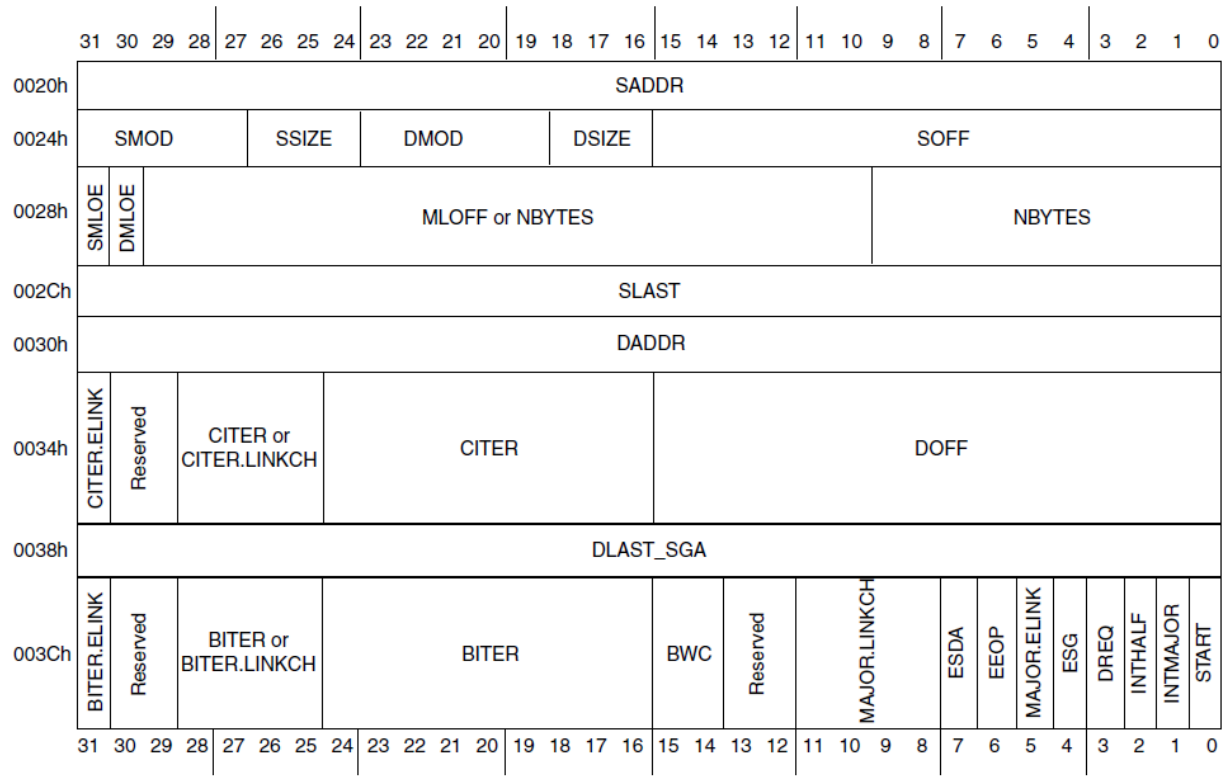

Figure 8. TCD memory map

Table 4 describes the elements of TCD and their functions.

Table 4. TCD field descriptions

Field	Description
SADDR[31:0]	Source address Memory address of the transferred source data. It allows any area of the memory map to be selected. As the eDMA performs transfers, this field is automatically updated for the next transfer.
SMOD[15:11]	Source address modulo It simplifies the implementation of a circular data queue. A circular buffer is created as the lower address fields wrap to their original value while the upper fields remain fixed. 00000 – The source address modulo feature is disabled. xxxxx – The number of lower source address bits that are allowed to increment.
SSIZE[10:8]	Source data transfer size It defines the read data size for the eDMA engine. It does not define the amount of data transferred per channel activation. 000 – 8-bit 001 – 16-bit 010 – 32-bit 011 – Reserved 100 – 16-byte burst 101 – Reserved 110 – Reserved

Table 4. TCD field descriptions*...continued*

Field	Description
	111 – Reserved
DMOD[7:3]	Destination address modulo It can be used to implement a circular data destination. As per SMOD but for the destination address.
DSIZE[2:0]	Destination data transfer size It defines the data write size for the eDMA engine. As per SSIZE.
SOFF[15:0]	Source address signed offset Signed offset that is added to the current source address, upon completion of a minor loop, to calculate the new source address value.
NBYTES[29:0]/[9:0]	The number of bytes to be transferred in each service request of the channel. The number of bytes to be transferred upon each activation of the channel. The length of the field varies depending on enabling/disabling minor offset.
SMLOE[31]	Source minor loop offset enable 0 – The minor loop offset is not applied to the SADDR. 1 – The minor loop offset is applied to the SADDR.
DMLOE[30]	Destination minor loop offset enable 0 – The minor loop offset is not applied to the DADDR. 1 – The minor loop offset is applied to the DADDR.
MLOFF[29:10]	If SMLOE or DMLOE is set, this field represents a sign-extended offset applied to the source or destination address to form the next-state value after the minor loop completes.
SLAST_SDA[31:0]	Last source address adjustment Signed offset that is added to the source address upon completion of the major loop, to calculate the new source address value. It can be used to restore the source address to the original value or to adjust the source address to the next data structure.
DADDR[31:0]	Destination address Memory address pointing to the destination data.
CITER_ELINK	Enable channel linking on completion of a minor loop 0 – Channel Linking on completion of a minor loop is disabled. 1 – Channel Linking on completion of a minor loop is enabled. ***Note:*** *This field must be equal to the* `BITER_E_LINK` *field or a configuration error is reported.*
LINKCH[5:0]	Minor loop link channel number If channel-to-channel linking is enabled (ELINK = 1), then after the minor loop is exhausted, the eDMA engine initiates a channel service request to the channel defined by this field by writingthe `TCDn_CSR[START]` field of that channel to **1**.
CITER[14:0] or CITER[8:0]	Current iteration count This 9-bit (ELINK = 1) or 15-bit (ELINK = 0) count represents the current major loop count for the channel. It is decremented each time the channel finishes a service request and is written back to the TCD memory. After the major iteration count is exhausted, the channel performs a number of operations — for example, final source and destination address calculations — and optionally generates an interrupt to signal channel completion before reloading the CITER field from the Beginning Iteration Count (BITER) field.
DOFF[15:0]	Destination address signed offset Signed offset that is added to the current destination address upon completion of a minor loop to calculate the next destination address.
DLAST_SGA[31:0]	Last destination address adjustment or memory address for the next TCD

Table 4. TCD field descriptions*...continued*

Field	Description
	If scatter/gather is disabled (ESG = 0), then the value contained in this field performs the same task as the SLAST field for the destination address. When scatter/gather is enabled (ESG = 1), this field is used as a pointer to a 0-modulo-32 region that contains the next TCD for this channel.
ELINK	Enable link 0b – Channel-to-channel linking disabled. 1b – Channel-to-channel linking enabled.
LINKCH[12:9]	Link channel number If channel-to-channel linking is enabled (ELINK = 1), then after the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by this field by setting that channel's TCDn_CSR[START] field.
BITER[14:0] or BITER[8:0]	Beginning major iteration count As the transfer control descriptor is first loaded by software, this 9-bit (ELINK = 1) or 15-bit (ELINK = 0) field must be set equal to the value in the CITER field. As the major iteration count is exhausted, eDMA reloads the contents of this field into the CITER field. If the channel is configured to execute a single service request, the initial values of BITER and CITER must be 0x0001.
BWC[15:14]	Bandwidth control It provides a means of controlling the amount of bus bandwidth the eDMA uses. 00 – No stalls-consume 100 % bandwidth. 01 – Reserved. 10 – eDMA stalls for four cycles after each read/write. 11 – eDMA stalls for eight cycles after each read/write.
MAJORLINKCH[11:8]	Major loop link channel number If (MAJORELINK = 0) then: No channel-to-channel linking, or chaining, is performed after the major loop counter is exhausted. Otherwise: After the major loop counter is exhausted, the eDMA engine initiates a channel service request.
ESDA	Enable Store Destination Address 0b – Ability to store destination address to system memory disabled. 1b – Ability to store destination address to system memory enabled.
EEOP	Enable End-Of-Packet Processing 0b – End-of-packet operation disabled. 1b – End-of-packet hardware input signal enabled.
MAJORLINK[5]	Enable channel linking on completion of a major loop 0 – Channel linking on completion of a major loop is disabled. 1 – Channel linking on completion of a major loop is enabled.
ESG[4]	Enable scatter/gather processing 0 – Scatter/Gather processing is disabled. 1 – Scatter/Gather processing is enabled.
DREQ[3]	Disable request If set when the channel completes a major loop, the eDMA clears the corresponding DMAERQ, disabling the transfer request. 0 – The DMAERQ bit of the channel is not affected. 1 – The DMAERQ bit of the channel is cleared upon completion of a major loop.

Table 4. TCD field descriptions*...continued*

Field	Description
INTHALF[2]	Generate interrupt when major loop is half-complete. When CITER = BITER ÷ 2, the eDMA asserts an interrupt request in the DMAINT register. `0` – The major loop half complete interrupt is disabled. `1` – The major loop half complete interrupt is enabled.
INTMAJOR[1]	Generate an Interrupt on Completion of a Major Loop. When CITER = 0, the eDMA asserts an interrupt request in the DMAINT register. `0` – The major loop complete interrupt is disabled. `1` – The major loop complete interrupt is enabled.
START[0]	Channel start Writing this bit as a **1** explicitly activates the channel and a minor loop transfer is performed.

If the TCD descriptor of a channel is configured with an illegal value or an illegal combination of values, a channel error is reported in the DMAERR register.

6 Example eDMA configurations

This section details two example eDMA configurations, based on FRDM-MCXN947 SDK, starting with a simple configuration and progressing to the more advanced feature and function of the eDMA at an application level:

1. Basic transfer (including channel linking)
2. Scatter/gather

6.1 Example configuration 1: basic transfer

This example configures the eDMA for a basic software-triggered eDMA transfer and channel linking on completion of the major loop of the first channel.

6.1.1 Requirements

Four data arrays are created in an internal SRAM. The first, `srcAddr[]` is of 4-byte size while three arrays `destAddr0[]`, `destAddr1[]`, and `destAddr3[]` of 4 bytes each to demonstrate different setups for the TCDs.

- AT_NONCACHEABLE_SECTION_INIT(uint32_t srcAddr[BUFFER_LENGTH]) = {0x01U, 0x02U, 0x03U, 0x04U}
- AT_NONCACHEABLE_SECTION_INIT(uint32_t destAddr0[BUFFER_LENGTH]) = {0x00U, 0x00U, 0x00U, 0x00U}
- AT_NONCACHEABLE_SECTION_INIT(uint32_t destAddr1[BUFFER_LENGTH]) = {0x00U, 0x00U, 0x00U, 0x00U}
- AT_NONCACHEABLE_SECTION_INIT(uint32_t destAddr2[BUFFER_LENGTH]) = {0x00U, 0x00U, 0x00U, 0x00U}

When the channel performing the transfer is activated by software, the first 32-bit piece of data in the sequence is moved from the source to the destination.

On completion of the major loop, the next channel is triggered automatically with channel linking and 32 bits are moved by the second channel, and so on.

When this transfer has been completed, the channel is not used again, making it unnecessary to restore or prepare the channel for future transfers.

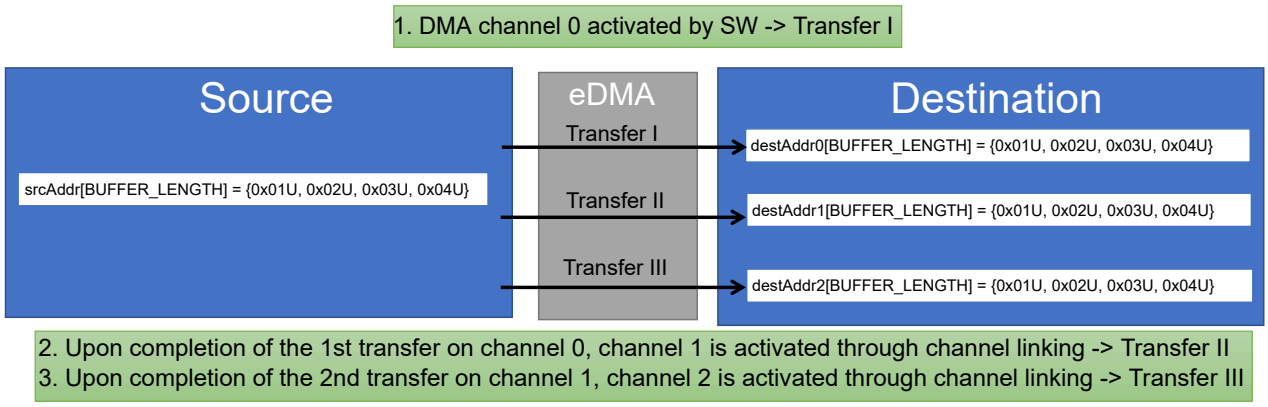

Figure 9. Example 1 requirements

6.1.2 Module configuration

This example uses software channel activation for the first TCD and activation by channel linking for the second TCD and then for the third TCD. Configuring the DMAMUX or the eDMA module registers is not required. It is only necessary to load the source data before configuring and activating the channel via the TCDs.

The code to configure the TCDs on channel 0, 1, and 2 is given below (includes enabling channel linking on channel 1 and 2):

```
/* Configure CH0 */
DMA_0.TCD[0].SADDR = (int)&srcAddr[0];
/*DMA_0.TCD[0].ATTR = 0x02 which is defining SMOD, SSIZE, DMOD & DSIZE as
 below:*/

DMA_0.TCD[0].SMOD = 0;
DMA_0.TCD[0].SSIZE = 0x2;         /* 32-bit */
DMA_0.TCD[0].DMOD = 0;
DMA_0.TCD[0].DSIZE = 0x2;         /* 32-bit */
DMA_0.TCD[0].SOFF = 0x4;
/*TCD0_NBYTES_MLOFFNO & TCD0_NBYTES_MLOFFYES defines TCD Transfer Size without
 or with minor loop offset where SMLOE, DMLOE are disabled*/

DMA_0.TCD[0].NBYTES = 8;
DMA_0.TCD[0].SLAST_SDA  = 0;
DMA_0.TCD[0].DADDR = 0x20000030;
/*TCD0_CITER_ELINKNO & TCD0_CITER_ELINKYES which indicates the Current Major
 loop channel linking enabled or disabled:*/

DMA_0.TCD[0].ELINK = 1;           /*Enable Link*/
DMA_0.TCD[0].LINKCH = 1;
DMA_0.TCD[0].CITER = 2;           /*Current Major iteration count*/
DMA_0.TCD[0].DOFF = 0x4;
DMA_0.TCD[0].DLAST_SGA = 0;
/*TCD0_BITER_ELINKNO & TCD0_BITER_ELINKYES which indicates the Beginning Major
 loop channel linking enabled or disabled:*/

DMA_0.TCD[0].ELINK = 1;            /*Enable Link*/
DMA_0.TCD[0].LINKCH = 1;
DMA_0.TCD[0].BITER = 2;            /*Beginning Major iteration count*/
/*DMA_0.TCD[0].CSR as shown below: */
DMA_0.TCD[0].BWC = 0;
DMA_0.TCD[0].MAJORLINKCH = 1;       /* Link to channel 1 */
```

```
DMA_0.TCD[0].MAJORELINK = 1;        /* Enable channel linking*/
DMA_0.TCD[0].ESG = 0;
DMA_0.TCD[0].DREQ = 1;
DMA_0.TCD[0].INTHALF = 0;
DMA_0.TCD[0].INTMAJ = 0;

/* Configure CH1 */
DMA_0.TCD[1].SADDR = (int)&srcAddr[0];
/*DMA_0.TCD[1].ATTR = 0x02 which is defining SMOD, SSIZE, DMOD & DSIZE as
 below:*/
DMA_0.TCD[1].SMOD = 0;
DMA_0.TCD[1].SSIZE = 0x2;            /* 32-bit */
DMA_0.TCD[1].DMOD = 0;
DMA_0.TCD[1].DSIZE = 0x2;            /* 32-bit */
DMA_0.TCD[1].SOFF = 0x4;
/*TCD1_NBYTES_MLOFFNO & TCD1_NBYTES_MLOFFYES defines TCD Transfer Size without
 or with minor loop offset where SMLOE, DMLOE are disabled*/
DMA_0.TCD[1].NBYTES = 16;            /* 4x32-bits */
DMA_0.TCD[1].SLAST_SDA  = 0;
DMA_0.TCD[1].DADDR = 0x20000040;
/*TCD1_CITER_ELINKNO & TCD1_CITER_ELINKYES which indicates the Current Major
 loop channel linking enabled or disabled:*/

DMA_0.TCD[1].ELINK = 0;              /*Enable Link*/
DMA_0.TCD[1].LINKCH = 0;
DMA_0.TCD[1].CITER = 1;              /*Current Major iteration count*/
DMA_0.TCD[1].DOFF = 0x4;
DMA_0.TCD[1].DLAST_SGA = 0;
/*TCD1_BITER_ELINKNO & TCD1_BITER_ELINKYES which indicates the Beginning Major
 loop channel linking enabled or disabled:*/

DMA_0.TCD[1].ELINK = 0;               /*Enable Link*/
DMA_0.TCD[1].LINKCH = 0;
DMA_0.TCD[1].BITER = 1;               /*Beginning Major iteration count*/
/*DMA_0.TCD[1].CSR as shown below:*/
DMA_0.TCD[1].BWC = 0;
//DMA_0.TCD[1].MAJORLINKCH = 0;
DMA_0.TCD[1].MAJORELINK = 0;
DMA_0.TCD[1].ESG = 0;
DMA_0.TCD[1].DREQ = 1;
DMA_0.TCD[1].INTHALF = 0;
DMA_0.TCD[1].INTMAJ = 0;
/* Configure CH2 */
DMA_0.TCD[2].SADDR = (int)&data_array1[0];
/*DMA_0.TCD[2].ATTR = 0x02 which is defining SMOD, SSIZE, DMOD & DSIZE as
 below:*/
DMA_0.TCD[2].SMOD = 0;
DMA_0.TCD[2].SSIZE = 0x2;              /* 32-bit */
DMA_0.TCD[2].DMOD = 0;
DMA_0.TCD[2].DSIZE = 0x2;              /* 32-bit */
DMA_0.TCD[2].SOFF = 0x4;
/*TCD2_NBYTES_MLOFFNO & TCD2_NBYTES_MLOFFYES defines TCD Transfer Size without
 or with minor loop offset where SMLOE, DMLOE are disabled*/
DMA_0.TCD[2].NBYTES = 16;              /* 4x32-bits */
DMA_0.TCD[2].SLAST_SDA  = 0;
DMA_0.TCD[2].DADDR = 0x20000050;
/*TCD2_CITER_ELINKNO & TCD2_CITER_ELINKYES which indicates the Current Major
 loop channel linking enabled or disabled: */

DMA_0.TCD[2].ELINK = 0;                /*Enable Link*/
```

```
DMA_0.TCD[2].LINKCH = 0;
DMA_0.TCD[2].CITER = 1;                    /*Current Major iteration count*/
DMA_0.TCD[2].DOFF = 0x4;
DMA_0.TCD[2].DLAST_SGA = 0;
/*TCD2_BITER_ELINKNO & TCD2_BITER_ELINKYES which indicates the Beginning Major
 loop channel linking enabled or disabled:*/

DMA_0.TCD[2].ELINK = 0;                     /*Enable Link*/
DMA_0.TCD[2].LINKCH = 0;
DMA_0.TCD[2].BITER = 1;                     /*Beginning Major iteration count*/
/*DMA_0.TCD[2].CSR as shown below:*/
DMA_0.TCD[2].BWC = 0;
//DMA_0.TCD[2].MAJORLINKCH = 0;
DMA_0.TCD[2].MAJORELINK = 0;
DMA_0.TCD[2].ESG = 0;
DMA_0.TCD[2].DREQ = 1;
DMA_0.TCD[2].INTHALF = 0;
DMA_0.TCD[2].INTMAJ = 0;
```

Note: *Bit fields that are commented out are shown so that all TCD fields can be viewed. If a bit field is commented out, its value is set to 0.*

Channel linking is enabled by setting MAJORELINK = 1.

The first DMA transfer is initiated by setting the start bit of TCD:

```
DMA_0.TCD[0].START = 1; /* Start transfer on channel 0 */
```

Channel 1 starts automatically via channel linking.

If possible, step through the code in a debugging environment and monitor the source and destination memory addresses as the channels are activated and the transfers performed. On completion of the major loop, the source and destination addresses are restored. Further activations of channel 0 therefore result in the transfer process being repeated.

With this configuration, each time one of the 32-bit values is transferred. A minor loop is completed. When transfers have been completed, the major loop is complete.

6.2 Example configuration 2: scatter/gather

This example configures the eDMA for a software-triggered eDMA transfer with a subsequent scatter/gather mechanism at completion of the first major loop.

6.2.1 Requirements

Two data arrays and one TCD array are created in an internal SRAM. The first, `srcAddr` is of a 8-byte size while `destAddr[]` is 8 bytes to demonstrate different set-ups for the TCDs.

The `tcdMemoryPoolPtr` structure contains the new TCD content that has to be loaded into the channels TCD on completion of the first major loop.

When the channel performing the transfer is activated by software, the first 32-bit piece of data in the sequence is moved from the source to the destination.

On completion of the first major loop, scatter/gather is performed and the data contained in the structure `tcdMemoryPoolPtr` is loaded into the TCD of the channel. The next transfer is triggered automatically as the START bit is set when the new TCD is loaded. When this second transfer has completed, the channel is not used again, making it unnecessary to restore or prepare the channel for future transfers.

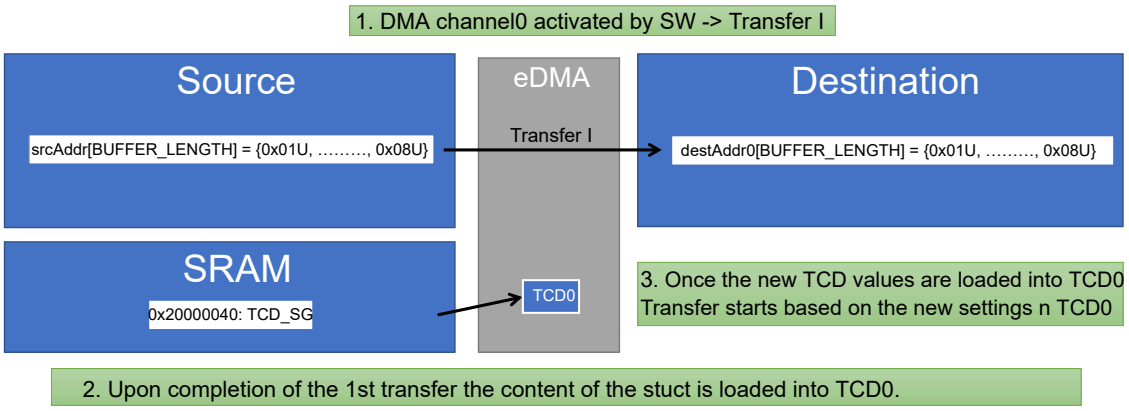

Figure 10. Example 2 requirements

6.2.2 Module configuration

This example uses software channel activation for the first transfer. The second transfer is started automatically when the new TCD is loaded as the START bit is set. Configuring the `DMA_Mux` or the eDMA module registers is not required. It is only necessary to load the source data before configuring and activating the channel via the TCDs.

The code to perform the transfer on channel 0 (includes enabling scatter/gather) is given below:

```
/* Configure CH0 */
DMA_0.TCD[0].SADDR = (int)&srcAddr[0];
DMA_0.TCD[0].SMOD = 0;
DMA_0.TCD[0].SSIZE = 0x2;         /* 32-bit */
DMA_0.TCD[0].DMOD = 0;
DMA_0.TCD[0].DSIZE = 0x2;         /* 32-bit */
DMA_0.TCD[0].SOFF = 0x4;
DMA_0.TCD[0].NBYTES = 16;         /* 2 structures of 16 bytes each will be
 created */
DMA_0.TCD[0].SLAST_SDA = 0;
/*TCD0_CITER_ELINKNO & TCD0_CITER_ELINKYES which indicates the Current
Major loop channel linking enabled or disabled: */

DMA_0.TCD[0].ELINK = 0;           /*Enable Link*/
DMA_0.TCD[0].LINKCH = 0;
DMA_0.TCD[0].CITER = 1;           /*Current Major iteration count*
DMA_0.TCD[0].DADDR = 0x20000040;
DMA_0.TCD[0].DOFF = 0x4;
DMA_0.TCD[0].DLAST_SGA = 0x20000020;
/*TCD0_BITER_ELINKNO & TCD0_BITER_ELINKYES which indicates the Beginning
Major loop channel linking enabled or disabled: */

DMA_0.TCD[0].ELINK = 0;           /*Enable Link*/
DMA_0.TCD[0].LINKCH = 0;
DMA_0.TCD[0].BITER = 1;           /*Beginning Major iteration count*/
/*DMA_0.TCD[0].CSR as shown below: */

/*For the transfer of 1st structure the register values will be*/
DMA_0.TCD[0].BWC = 0;
DMA_0.TCD[0].MAJORLINKCH = 0;
DMA_0.TCD[0].MAJORELINK = 0;      /* Disable channel linking */
DMA_0.TCD[0].ESG = 0;             /* Enable Sgatter/gather */
DMA_0.TCD[0].DREQ = 1;
```

```
DMA_0.TCD[0].INTHALF = 0;
DMA_0.TCD[0].INTMAJ = 0;

/*And for the second structure the values will be: */
DMA_0.TCD[0].BWC = 0;
DMA_0.TCD[0].MAJORLINKCH = 0;
DMA_0.TCD[0].MAJORELINK = 0;        /* Disable channel linking */
DMA_0.TCD[0].ESG = 1;               /* Enable Sgatter/gather */
DMA_0.TCD[0].DREQ = 0;
DMA_0.TCD[0].INTHALF = 0;
DMA_0.TCD[0].INTMAJ = 2;
```

Note: *Bit fields that are commented out are shown so that all the TCD fields can be viewed and differences are made more obvious.*

The code to perform the transfer on channel 0 (includes enabling sgatter/gather).

Note: *The 32 bytes of the TCD array used for scatter/gather has to be 32 byte aligned. The channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32-byte; otherwise, a configuration error is reported.*

The `tcdMemoryPoolPtr` structure is configured according to the TCD memory map (compare Figure 7). The first element of the array contains the source address (SADDR) and then points to the data array that is to be transferred. The last word is configured to set the START bit so that a transfer starts when the new TCD is loaded.

If possible, step through the code in a debugging environment and monitor the source and destination memory addresses as the channels are activated and the transfers performed. On completion of the first major loop, the values defined in the array `tcdMemoryPoolPtr` are loaded into `TCD[0]`.

7 Debugging tips

While this application note gives a solid grounding in using the eDMA, it is such a powerful module that there are many possible use case scenarios that cannot all be covered in this application note. To aid in debugging problems that may arise when developing applications, the eDMA includes the Error Status (ES) register, which can be a powerful tool for diagnosing problems with DMA transfers.

By effectively using these registers and understanding their bit fields, developers can:

- **Proactively identify and address errors**: Prompt error handling based on specific bit flags can prevent data corruption and system crashes.
- **Pinpoint the source of issues**: Differentiating between master and channel-specific errors enables targeted debugging and faster resolution.
- **Validate DMA configurations**: Checking for errors like invalid transfer sizes or descriptor table issues helps ensure a correct setup.
- **Enhance application robustness**: Incorporating error handling routines based on these registers improves application reliability and stability.

MP_ES (Master Error Status) Register: Provides insights into errors affecting the entire eDMA controller, not specific channels.

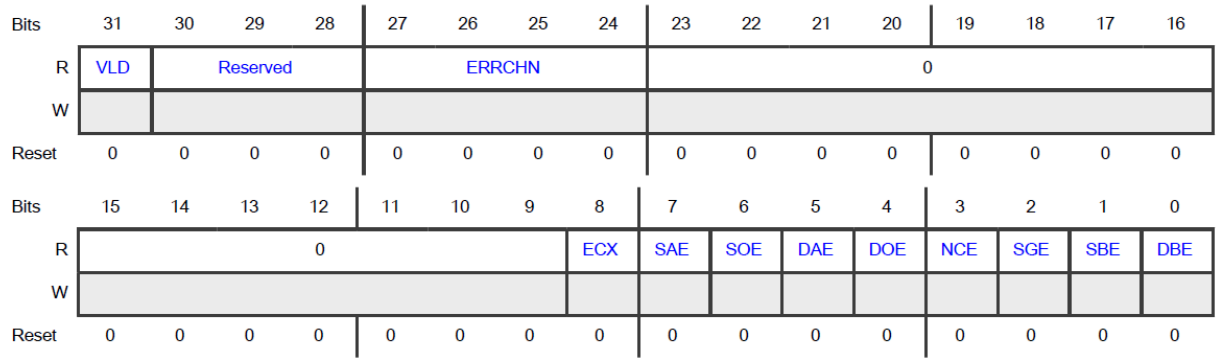

Figure 11. MP_ES (master error status) register

Key Bit Fields:

Table 5. Error Status Register field descriptions

Field	Description
VLD	`CH_ES[ERR]` status bits, indicating an error in one of the channels. The exact channel number can be determined by checking the Channel Error Status registers.
ERRCHN	Error channel number.
ECX	Transfer canceled via the error cancel transfer bit `DMA_CR[ECX]`.
SAE	Configuration error in the `TCD_SADDR` field (inconsistent with `TCD_ATTR[SSIZE]`).
SOE	Configuration error in the `TCD_SOFF` field (inconsistent with `TCD_ATTR[SSIZE]`)
DAE	Configuration error in the `TCD_DADDR` field (inconsistent with `TCD_ATTR[DSIZE]`)
DOE	Configuration error in the `TCD_DOFF` field (inconsistent with `TCD_ATTR[DSIZE]`)
NCE	Configuration error in the `TCD_NBYTES` or `TCD_CITER` field. Initially `TCD_CITER` must be programmed to be the same value as `TCD_BITER`.
SGE	Scatter/gather error, this indicates a configuration error in the `TCD_DLAST_SGA` field, which has to be on a 32-byte boundary when scatter/gather is enabled (`TCD_CSR[ESG]` = 1).
SBE	Bus error on source read.
DBE	Bus error on a destination write.

Channel Error Status Register: Captures errors specific to individual channels, aiding in pinpointing problematic transfers.

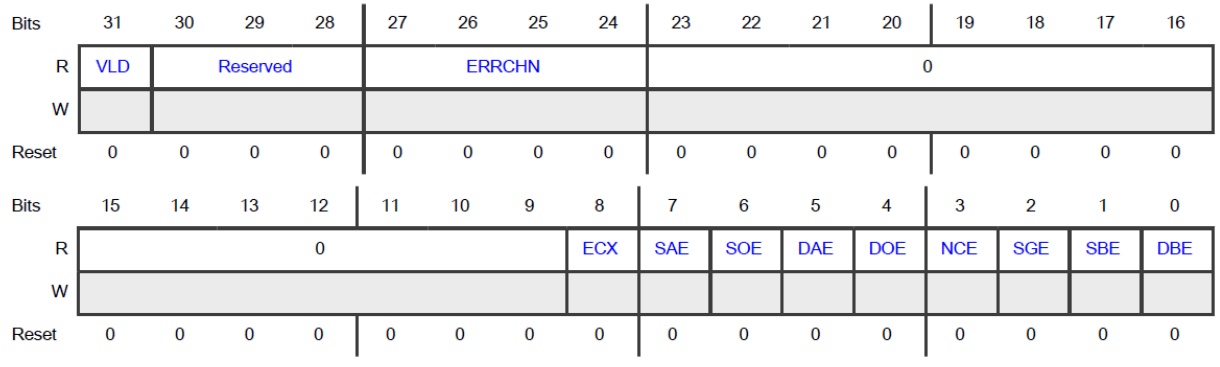

Figure 12. Channel error status register

Table 6. Error status register field descriptions

Field	Description
ERR	Enable if an error in the channel has occurred.
SAE	Configuration error in the TCD_SADDR field (inconsistent with TCD_ATTR[SSIZE]).
SOE	Configuration error in the TCD_SOFF field (inconsistent with TCD_ATTR[SSIZE])
DAE	Configuration error in the TCD_DADDR field (inconsistent with TCD_ATTR[DSIZE])
DOE	Configuration error in the TCD_DOFF field (inconsistent with TCD_ATTR[DSIZE])
NCE	Configuration error in the TCD_NBYTES or TCD_CITER field. Initially, TCD_CITER must be programmed to be the same value as TCD_BITER.
SGE	Scatter/gather error, this indicates a configuration error in the TCD_DLAST_SGA field which must be on a 32-byte boundary when scatter/gather is enabled (TCD_CSR[ESG] = 1).
SBE	Bus error on source read.
DBE	Bus error on a destination write.

When an error occurs in a DMA transaction, it is flagged in this ES register, depending on the type of error. The following table describes the errors indicated by the ES.

Table 7. Error Status Register field descriptions

Field	Description
VLD	CH_ES[ERR] status bits, indicating an error in one of the channels. The exact channel number can be determined by checking the Channel Error Status registers.
ERRCHN	Error channel number.
ECX	Transfer canceled via the error cancel transfer bit DMA_CR[ECX].
SAE	Configuration error in the TCD_SADDR field (inconsistent with TCD_ATTR[SSIZE]).
SOE	Configuration error in the TCD_SOFF field (inconsistent with TCD_ATTR[SSIZE])
DAE	Configuration error in the TCD_DADDR field (inconsistent with TCD_ATTR[DSIZE])
DOE	Configuration error in the TCD_DOFF field (inconsistent with TCD_ATTR[DSIZE])
NCE	Configuration error in the TCD_NBYTES or TCD_CITER field. Initially TCD_CITER must be programmed to be the same value as TCD_BITER.
SGE	Scatter/gather error, this indicates a configuration error in the TCD_DLAST_SGA field, which has to be on a 32-byte boundary when scatter/gather is enabled (TCD_CSR[ESG] = 1).
SBE	Bus error on source read.
DBE	Bus error on a destination write.

Another useful tool when debugging is the soft start bit, TCD_CSR[START]. It can be used to start any configuration in software and is a good method to check if the configuration is behaving as expected.

8 Conclusion

eDMA technology in MCx Nx4x Series microcontrollers unlocks significant performance benefits for various applications. This application note provides a good understanding and working knowledge of the MCX Nx4x eDMA controller. It enables the user to create eDMA configurations suitable for applications by effectively using these features. Developers can maximize data throughput, minimize CPU load, and create high-performance embedded systems. The source code provided along with this application note can be used as a basis for configurations.

For more information on the NXP MCX Nx4x family, visit nxp.com.

9 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.

10 Revision history

Table 8 summarizes the revisions to this document.

Table 8. Revision history

Document ID	Release date	Description
AN14300 v.1.0	23 September 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.

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.

Contents