Freescale Semiconductor
Application Note

Document Number: AN4765
Rev. 1, 08/2013

MPC57xx: Configuring and Using the eDMA Controller

by: Martin Vaupel and David McMenamin

The Qorivva MPC57xx family of multicore 32-bit microcontrollers is initially intended for automotive applications. It is based on e200 cores built on Power Architecture®.

Table Of Contents

1 Introduction

The MPC57xx family of microcontrollers features Freescale's enhanced Direct Memory Access (eDMA) controller.

This application note provides a working knowledge of the MPC57xx eDMA controller by covering the following topics: introduction and overview of DMA controllers, features of the MPC57xx 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 DMA configurations.

A ZIP file containing all of the examples used within this document is available for download from freescale.com.

1.1 DMA controller overview

A DMA controller provides the ability to move data from one memory mapped location to another without CPU intervention. Once configured and initiated, the DMA 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 reduced CPU loading and a corresponding increase in system performance. Figure 1 illustrates the functionality provided by a DMA controller.

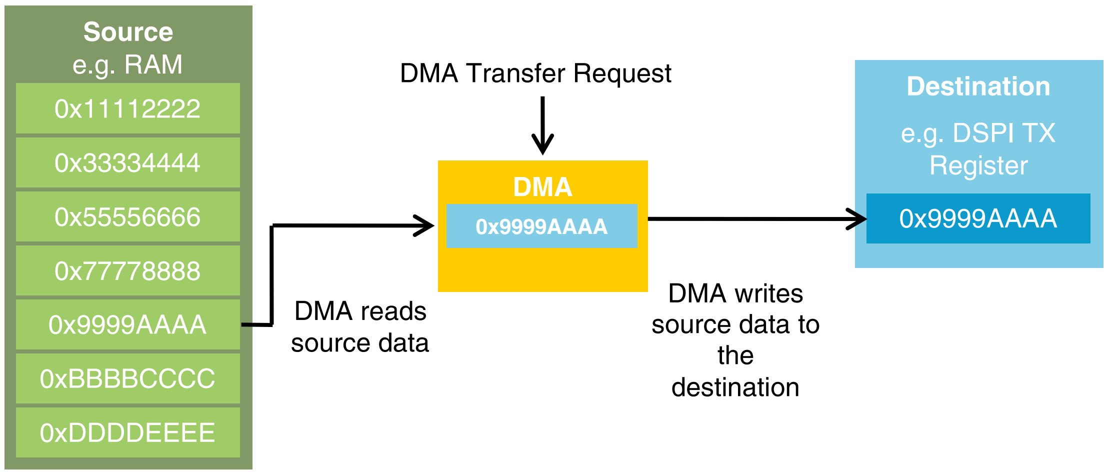

Figure 1. DMA operational overview

1.2 MPC57xx eDMA controller features

All MPC57xx devices feature a 64-channel eDMA controller. Each channel can be independently configured with the details of the transfer sequence that is to be executed. These details are specified in the channel Transfer Control Descriptor (TCD) registers.

eDMA transfers can be activated in 3 ways:

1. Events occurring in peripheral modules and off-chip can assert a DMA transfer request
2. Software activation
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 64 channels. This feature allows a channel to self-load a new TCD when it has performed the transfer for its current configuration. Using this feature allows for far greater than 64 transfer sequences to be defined and used.

1.3 eDMA architectural integration

To allow the eDMA, CPUs, and other masters to operate simultaneously, a multi-master bus architecture is implemented. The MPC57xx 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 MPC57xx family feature a dual-crossbar architechture; in this case, data passes from one crossbar to the next if a master requires acccess 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. Arbitration settings for the crossbar switch can be configured in the XBAR module registers. Refer to the microcontroller reference manual for more details.

The crossbar switch and interaction between bus masters and slave devices is illustrated in a simplified version in Figure 2. 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.

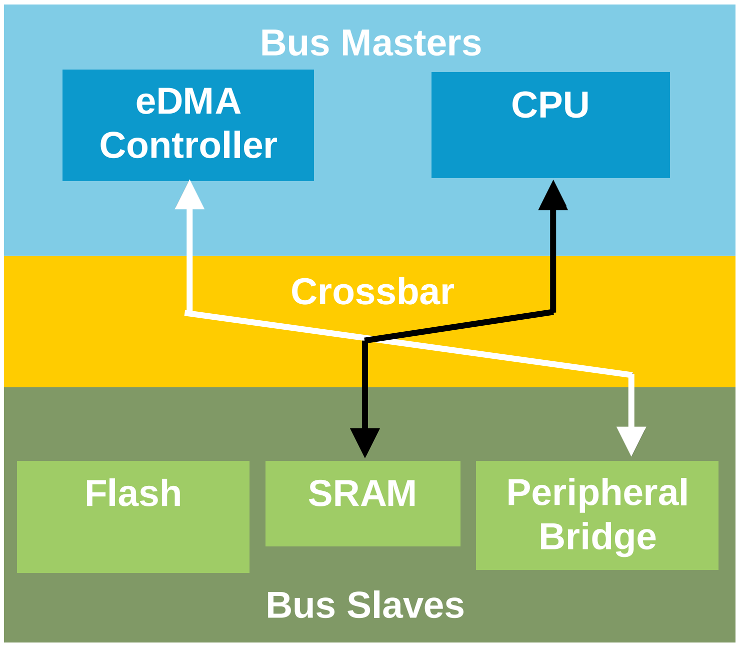

Figure 2. Multi-master bus architecture

2 Activating eDMA Transfers

2.1 Activation sources

- 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 channels' TCDs 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 DMA multiplexer

The DMA multiplexer is used to route the numerous peripheral DMA sources to individual DMA channels. There are six DMA mux instances available, each of which is able to route sources to eight DMA channels as described in Figure 3 and Table 1.

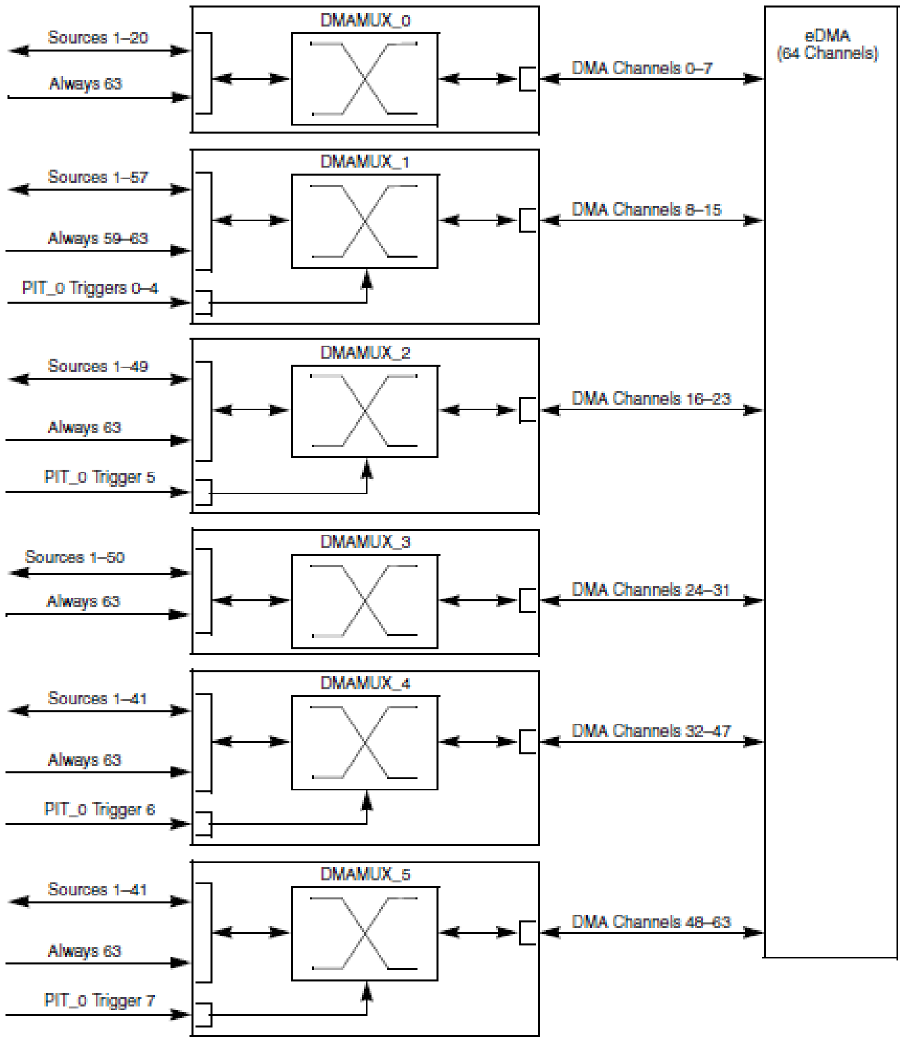

Figure 3. DMA channel multiplexing

Table 1. eDMA transfer request sources

DMAMUX instance	DMA channel
0	0–7
1	8–15
2	16–23
3	24–31
4	32–47
5	48–63

The DMA multiplexer (DMA_Mux) performs the task of routing the peripheral DMA request sources to the desired channel. It also provides the ability to gate a transfer request with the Periodic Interrupt Controller (PIT), on selected MUX implementations. This functionality is discussed further in section 2.3.

The mapping of the peripheral DMA requests to the DMAMUX input sources differs across the microcontrollers in the MPC57xx family. As an example, Table 2 lists the source mapping for MPC5746M.

Table 2. Peripheral DMA requests on MPC5746M

Source	Peripheral DMA requests					
	DMAMUX_0	DMAMUX_1	DMAMUX_2	DMAMUX_3	DMAMUX_4	DMAMUX_5
0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
1	ADC_SAR_0 EOC	DSPI_12 RX	ADC_SAR_2 EOC	ADC_SAR_3 EOC	ADC_SAR_4 EOC	ADC_SAR_7 EOC
2	ADC_SAR_1 EOC	DSPI_12 TX	ADC_SD_1 EOC	DSPI_2 RX	ADC_SAR_6 EOC	ADC_SD_4 EOC
3	ADC_SAR_B EOC	LINFlex_0 RX	DSPI_1 RX	DSPI_2 TX	ADC_SD_2 EOC	ADC_SD_5 EOC
4	ADC_SD_0 EOC	LINFlex_0 TX	DSPI_1 TX	LINFlex_2 RX	ADC_SD_3 EOC	DSPI_5 RX
5	DSPI_0 RX	LINFlex_1 RX	SENT_1 RX FAST	LINFlex_2 TX	DSPI_3 RX	DSPI_5 TX
6	DSPI_0 TX	LINFlex_1 TX	SENT_1 RX SLOW	I2C_0 RX	DSPI_3 TX	LINFlex_1 RX
7	DSPI_4 RX	LINFlex_14 RX	PSI5_0 CH0 RX PSI5	I2C_0 TX	LINFlex_0 RX	LINFlex_1 TX
8	DSPI_4 TX	LINFlex_14 TX	PSI5_0 CH0 RX SRX	PSI5_1 CH0 RX PSI5	LINFlex_0 TX	LINFlex_15 RX
9	Reserved	SENT_0 RX FAST	SIUL2 REQ2	PSI5_1 CH0 RX SRX	LINFlex_14 RX	LINFlex_15 TX
10	ADC_SAR_4 EOC	SENT_0 RX SLOW	SIUL2 REQ4	SIUL2 REQ5	LINFlex_14 TX	SENT_0 RX FAST
11	ADC_SD_3 EOC	SIPI CH0	GTM_PSM0_IRQ0	GTM_PSM0_IRQ4	PSI5_0 CH1 RX PSI5	SENT_0 RX SLOW
12	M_CAN_1	SIPI CH1	GTM_PSM0_IRQ1	GTM_PSM0_IRQ5	PSI5_0 CH1 RX SRX	SIPI CH2
13	M_CAN_2	SIPI CH2	GTM_PSM0_IRQ2	GTM_PSM0_IRQ6	SIPI CH0	SIPI CH3
14	SENT_0 RX FAST	SIPI CH3	GTM_PSM0_IRQ3	GTM_PSM0_IRQ7	SIPI CH1	SIUL2 REQ10
15	SENT_0 RX SLOW	SIUL2 REQ0	GTM_TIM1_IRQ0	GTM_TIM1_IRQ4	SIUL2 REQ9	GTM_TOM0_IRQ2
16	LINFlex_0 RX	SIUL2 REQ1	GTM_TIM1_IRQ1	GTM_TIM1_IRQ5	GTM_TIM0_IRQ0	GTM_TOM0_IRQ3
17	LINFlex_0 TX	GTM_TIM0_IRQ0	GTM_TIM1_IRQ2	GTM_TIM1_IRQ6	GTM_TIM0_IRQ1	GTM_TOM0_IRQ4
18	LINFlex_14 RX	GTM_TIM0_IRQ1	GTM_TIM1_IRQ3	GTM_TIM1_IRQ7	GTM_TOM0_IRQ0	GTM_TOM0_IRQ5
19	DSPI_0 CMD	GTM_TIM0_IRQ2	GTM_TOM1_IRQ0	GTM_TOM1_IRQ4	GTM_TOM0_IRQ1	GTM_PSM0_IRQ4
20	DSPI_4 CMD	GTM_TIM0_IRQ3	GTM_TOM1_IRQ1	GTM_TOM1_IRQ5	GTM_PSM0_IRQ0	GTM_PSM0_IRQ5
21	Reserved	GTM_TIM0_IRQ4	GTM_TOM1_IRQ2	GTM_TOM1_IRQ6	GTM_PSM0_IRQ1	GTM_PSM0_IRQ6
22	Reserved	GTM_TIM0_IRQ5	GTM_TOM1_IRQ3	GTM_TOM1_IRQ7	GTM_PSM0_IRQ2	GTM_PSM0_IRQ7
23	Reserved	GTM_TIM0_IRQ6	GTM_ATOM1_IRQ0	GTM_MCS1_IRQ2	GTM_PSM0_IRQ3	GTM_TOM1_IRQ4
24	Reserved	GTM_TIM0_IRQ7	GTM_ATOM1_IRQ1	GTM_MCS1_IRQ3	GTM_TOM1_IRQ0	GTM_TOM1_IRQ5
25	Reserved	GTM_TOM0_IRQ0	GTM_MCS1_IRQ0	GTM_MCS1_IRQ4	GTM_TOM1_IRQ1	GTM_TIM3_IRQ4
26	Reserved	GTM_TOM0_IRQ1	GTM_MCS1_IRQ1	GTM_MCS1_IRQ5	GTM_TIM3_IRQ0	GTM_TIM3_IRQ5

Source	Peripheral DMA requests					
	DMAMUX_0	DMAMUX_1	DMAMUX_2	DMAMUX_3	DMAMUX_4	DMAMUX_5
27	Reserved	GTM_TOM0_IRQ2	GTM_MCS1_IRQ2	GTM_MCS1_IRQ6	GTM_TIM3_IRQ1	GTM_TIM3_IRQ6
28	Reserved	GTM_TOM0_IRQ3	GTM_MCS1_IRQ3	GTM_MCS1_IRQ7	GTM_TIM3_IRQ2	GTM_TIM3_IRQ7
29	Reserved	GTM_TOM0_IRQ4	GTM_TIM2_IRQ0	GTM_TIM2_IRQ4	GTM_TIM3_IRQ3	GTM_MCS3_IRQ4
30	Reserved	GTM_TOM0_IRQ5	GTM_TIM2_IRQ1	GTM_TIM2_IRQ5	GTM_MCS3_IRQ0	GTM_MCS3_IRQ5
31	Reserved	GTM_TOM0_IRQ6	GTM_TIM2_IRQ2	GTM_TIM2_IRQ6	GTM_MCS3_IRQ1	GTM_MCS3_IRQ6
32	Reserved	GTM_TOM0_IRQ7	GTM_TIM2_IRQ3	GTM_TIM2_IRQ7	GTM_MCS3_IRQ2	GTM_MCS3_IRQ7
33	Reserved	GTM_ATOM0_IRQ0	GTM_ATOM2_IRQ0	GTM_ATOM2_IRQ2	GTM_MCS3_IRQ3	DSPI_5 CMD
34	Reserved	GTM_ATOM0_IRQ1	GTM_ATOM2_IRQ1	GTM_ATOM2_IRQ3	DSPI_3 CMD	M_CAN_1
35	Reserved	GTM_ATOM0_IRQ2	GTM_MCS2_IRQ0	GTM_ATOM2_IRQ4	ADC_SD_1 EOC	M_CAN_2
36	Reserved	GTM_ATOM0_IRQ3	GTM_MCS2_IRQ1	GTM_ATOM2_IRQ5	ADC_SD_4 EOC	DSPI_3 RX
37	Reserved	GTM_MCS0_IRQ0	GTM_MCS2_IRQ2	GTM_ATOM2_IRQ6	ADC_SD_5 EOC	DSPI_3 TX
38	Reserved	GTM_MCS0_IRQ1	GTM_MCS2_IRQ3	GTM_ATOM2_IRQ7	ADC_SAR_0 EOC	ADC_SAR_3 EOC
39	Reserved	GTM_MCS0_IRQ2	GTM_ATOM3_IRQ0	GTM_ATOM3_IRQ2	DSPI_0 CMD	LINFlex_2 RX
40	Reserved	GTM_MCS0_IRQ3	GTM_ATOM3_IRQ1	GTM_ATOM3_IRQ3	DSPI_0 RX	LINFlex_2 TX
41	Reserved	GTM_MCS0_IRQ4	ADC_SD_2 EOC	SIUL2 REQ8	DSPI_0 TX	ADC_SAR_1 EOC
42	Reserved	GTM_MCS0_IRQ5	DSPI_1 CMD	Reserved	Reserved	Reserved
43	Reserved	GTM_MCS0_IRQ6	DSPI_2 RX	ADC_SD_3 EOC	Reserved	Reserved
44	Reserved	GTM_MCS0_IRQ7	DSPI_2 TX	ADC_SAR_6 EOC	Reserved	Reserved
45	Reserved	LINFlex_15 RX	LINFlex_2 RX	DSPI_2 CMD	Reserved	Reserved
46	Reserved	LINFlex_15 TX	LINFlex_2 TX	DSPI_1 RX	Reserved	Reserved
47	Reserved	DSPI_5 RX	GTM_SPE0	DSPI_1 TX	Reserved	Reserved
48	Reserved	DSPI_5 TX	GTM_SPE1	ADC_SAR_2 EOC	Reserved	Reserved
49	Reserved	DSPI_5 CMD	PSI5-S	ADC_SD_2 EOC	Reserved	Reserved
50	Reserved	DSPI_12 CMD	Reserved	PSI5-S	Reserved	Reserved
51	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
52	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
53	Reserved	DSPI_0 RX	Reserved	Reserved	Reserved	Reserved
54	Reserved	DSPI_0 TX	Reserved	Reserved	Reserved	Reserved
55	Reserved	ADC_SAR_0 EOC	Reserved	Reserved	Reserved	Reserved
56	Reserved	ADC_SAR_3 EOC	Reserved	Reserved	Reserved	Reserved
57	Reserved	ADC_SAR_4 EOC	Reserved	Reserved	Reserved	Reserved
58	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
59	Reserved	Always on	Reserved	Reserved	Reserved	Reserved
60	Reserved	Always on	Reserved	Reserved	Reserved	Reserved
61	Reserved	Always on	Reserved	Reserved	Reserved	Reserved
62	Reserved	Always on	Reserved	Reserved	Reserved	Reserved
63	Always on	Always on	Always on	Always on	Always on	Always on

Example 3 in section 5.3 shows how peripheral eDMA sources are enabled for DMA transfers through the DMAMUX.

2.3 Activation options

The DMAMUX supports three different options for asserting transfer requests to the DMA.

- **Disabled 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 (for example, changing the period of a DMA trigger).
- **Normal mode**: In this mode, a DMA source (compare to Table 2) 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 as well as 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.

- **Periodic Trigger mode**: In this mode, a DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. The period is configured in the registers of the Periodic Interrupt Timer (PIT). Figure 4 shows the relationship between the periodic interrupt, transfer request, and the transfer activation.

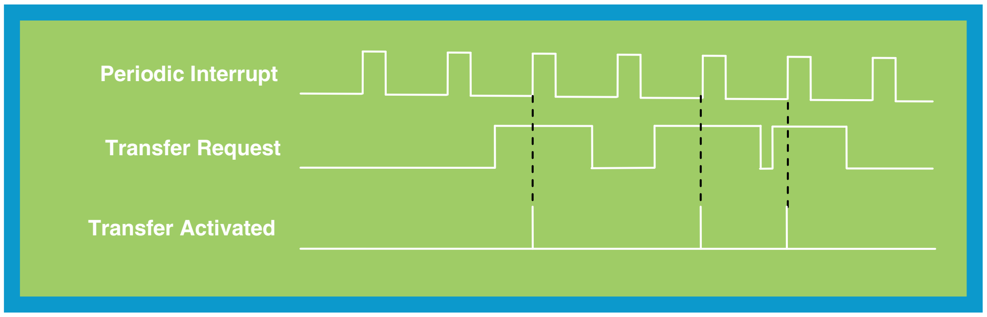

Figure 4. PIT gated transfer activation

The DMAMUX also provides a number of “always enabled” request sources that can be used in periodic trigger mode. These permit transfers to be initiated based only on the PIT. This is shown in Figure 5.

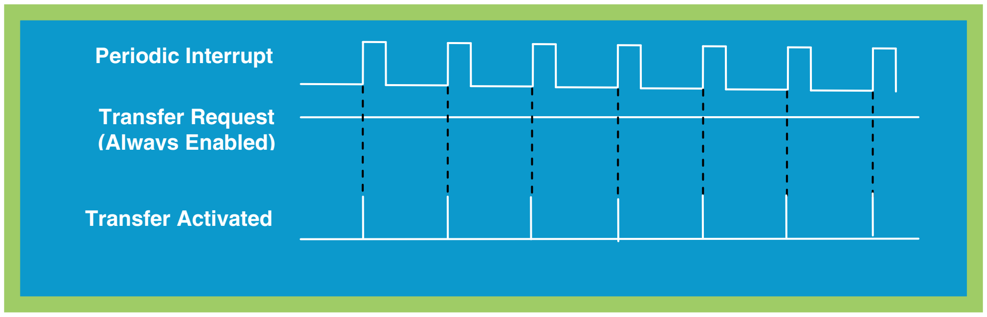

Figure 5. PIT only transfer activation

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, 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, 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 6 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 4 bytes.

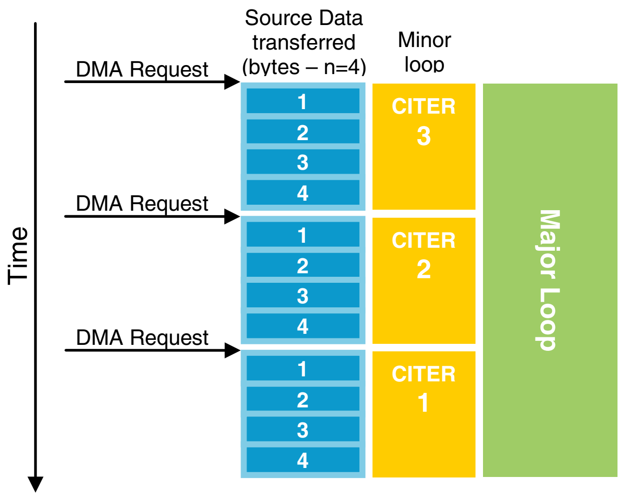

Figure 6. Major and minor transfer loops

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 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 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 source address by adding the current source address to the last source address adjustment: SADDR = SADDR + SLAST
- Updating 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 minor loop, if channel linking is enabled
- Reloading current iteration (CITER) from the beginning major iteration count (BITER) field

4 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 microcontroller's reference manual.

4.1 Configuration steps

To configure the eDMA, perform the following initialization steps:

1. Program the Control Register (eDMA_CR). This step is necessary only if a configuration other than the default is required.
2. Configure Channel Priority Registers (eDMA_DCHPRI*x*) This step is necessary only if a configuration other than the default is required.
3. Enable error interrupts 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 will be 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

4.2 Transfer Control Descriptors (TCD)

All transfer attributes for a channel are defined in the channel's unique TCD. Each TCD is stored in the eDMA controller's local SRAM. 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 will result in unpredictable behavior of the channel. Figure 7 shows the TCD memory map.

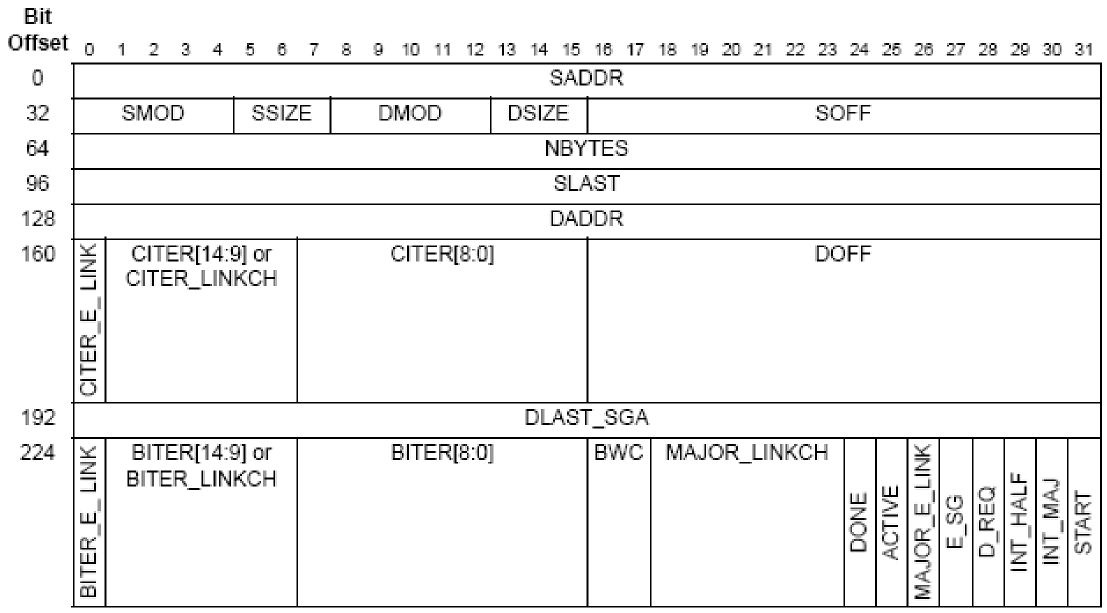

Figure 7. TCD memory map

The following table describes the TCD's elements and their functions.

Table 3. TCD field descriptions

Field	Description
SADDR[31:0]	Source Address. Memory address of the transfer source data. 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[4:0]	Source Address Modulo. 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 Source address modulo feature is disabled xxxxx The number of lower source address bits that are allowed to increment.

Field	Description
SSIZE[2:0]	Source Data Transfer Size. 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 111 Reserved Example: For a transfer of 8 bytes per channel activation and SSIZE = 16 bits, the eDMA will perform four 16-bit reads. If SSIZE were 32 bits, then the eDMA would perform two reads for this transfer.
DMOD[4:0]	Destination Address Modulo. Can be used to implement a circular data destination. As per SMOD but for the destination address.
DSIZE[2:0]	Destination Data Transfer Size. 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[31:0]/ [31:2]/[31/22]	Minor Byte Transfer Count. Number of bytes to be transferred upon each activation of the channel. Length of the field varies depending on enabling/disabling minor offset.
SMLOE[1:0]	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[2:1]	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[21:2]	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[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 of the transfer destination. Allows any area of the memory map to be selected. As the eDMA performs transfers, this field is automatically updated for the next transfer.
CITER_E_LINK	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 will be reported.
CITER_LINKCH[5:0]	Minor Loop Complete Link Channel. As the channel completes a minor loop, it asserts the START bit of the channel defined in CITER_LINKCH[5:0].

Field	Description
CITER[14:0] or CITER[8:0]	Current Iteration Count. Represents the current number of minor loops that are to be executed to complete the major loop. As minor loops are completed, this field is decremented until it is exhausted. When it is exhausted, a major loop is complete. Upon completion of a major loop, the field is reloaded with the value contained in the BITER field. When this field is initially loaded, it must be set to the same value as the BITER field since the eDMA will not copy BITER into CITER until the first major loop has been completed. **NOTE**: If channel linking is disabled, a 15-bit iteration count is used instead of a 6-bit link channel number and 9-bit iteration count.
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.
DLASTSGA[31:0]	Last Destination Address Adjustment or Memory Address for the Next TCD. 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.
BITER_E_LINK	Beginning Enable Channel Linking on Minor Loop Complete. When a major loop is completed, this field is used to reload the CITER_E_LINK field. Hence, when writing the BITER_E_LINK and CITER_E_LINK they must be configured to the same value.
BITER_LINKCH[5:0]	Beginning Minor Loop Complete Link Channel. When a major loop is completed, this field is used to reload the CITER_LINKCH field. Hence, when configuring the BITER_LINKCH and CITER_LINKCH they must be configured to the same value.
BITER[14:0] or BITER[8:0]	Beginning Major Iteration Count. When a major loop is completed, this field is used to reload the CITER field in preparation for the next channel activation. When configuring the BITER and CITER fields, they should be configured to the same value.
BWC[1:0]	Bandwidth Control. 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 4 cycles after each read/write 11 eDMA stalls for 8 cycles after each read/write
MAJORLINKCH[5:0]	Major Loop Complete Link Channel. As the channel completes a major loop—and channel linking on completion of a major loop is enabled (MAJORELINK = 1)—the START bit of the channel defined in MAJORLINKCH[5:0] is asserted.
DONE	Channel Done. This bit is set when the channel completes a major loop. It remains set until the channel is reactivated by a transfer request or it is cleared by software.
ACTIVE	Channel Active. This bit is set if the channel is performing a transfer. It is set when a minor loop transfer is started and it is cleared, by the hardware, when that minor loop is complete.
MAJORELINK	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	Enable Scatter/Gather Processing 0 Scatter/Gather processing is disabled 1 Scatter/Gather processing is enabled

Field	Description
DREQ	Disable Request. If set when the channel completes a major loop, the eDMA clears the corresponding DMAERQ, disabling the transfer request. 0 The channel's DMAERQ bit is not affected 1 The channel's DMAERQ bit is cleared upon completion of a major loop
INTHALF	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	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	Channel Start. Writing this bit as a 1 explicitly activates the channel and a minor loop transfer is performed.

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

5 Example eDMA Configurations

This section details three example eDMA configurations, starting with a simple configuration and progressing to the more advanced features and functions of the eDMA at an application level:

1. Basic transfer (including channel linking)
2. Scatter/gather
3. Peripheral DMA request (using the SDADC EOC DMA request)

5.1 Example configuration 1: basic transfer

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

5.1.1 Requirements

Two data arrays are created in internal SRAM. The first, data_array0[] is of 64-byte size while data_array1[] is 16-byte to demonstrate different setups for the TCDs.

These arrays will be placed at the beginning of the dedicated data SRAM at address 0x40001000 and are to be moved to 0x4001F000 (data_array0) and 0x4001FF00 (data_array1).

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 by means of channel linking and 64 bits are moved by the second channel.

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

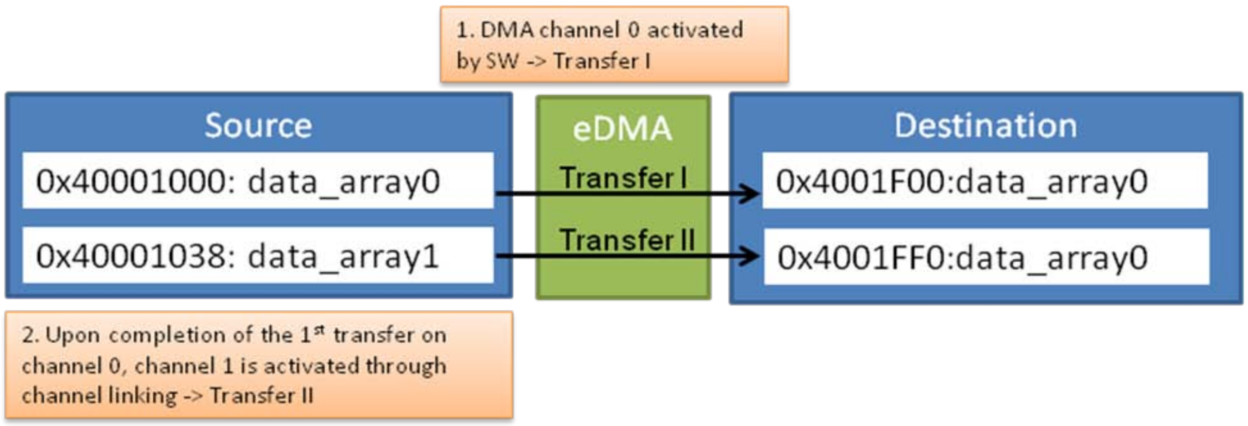
1. DMA channel 0 activated by SW -> Transfer I
Source
0x40001000: data_array0
0x40001038: data_array1
eDMA
Transfer I
Transfer II
Destination
0x4001F00:data_array0
0x4001FF0:data_array0
2. Upon completion of the 1st transfer on channel 0, channel 1 is activated through channel linking -> Transfer II

Figure 8. Example 1 requirements

5.1.2 Module configuration

This example uses software channel activation for the first TCD and activation by channel linking for the second 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 and 1 is given below (includes enabling channel linking on channel 1):

```
/* Configure CH0 */
DMA_0.TCD[0].SADDR = (int)&data_array0[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 = 64;  /* 16x32-bits */
DMA_0.TCD[0].SLAST = -64;
DMA_0.TCD[0].DADDR = 0x4001F000;
DMA_0.TCD[0].CITER_ELINK = 0;
//DMA_0.TCD[0].CITER_LINKCH = 0;
DMA_0.TCD[0].CITER = 1;
DMA_0.TCD[0].DOFF = 0x4;
DMA_0.TCD[0].DLAST_SGA = -64;
DMA_0.TCD[0].BITER = 1;
DMA_0.TCD[0].BITER_ELINK = 0;
//DMA_0.TCD[0].BITER_LINKCH = 0;
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 = 0;
DMA_0.TCD[0].INTHALF = 0;
DMA_0.TCD[0].INTMAJ = 0;
```

```
/* Configure CH1 */
DMA_0.TCD[1].SADDR = (int)&data_array1[0];
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;
DMA_0.TCD[1].NBYTES = 16;  /* 4x32-bits */
DMA_0.TCD[1].SLAST = -16;
DMA_0.TCD[1].DADDR = 0x4001FF00;
DMA_0.TCD[1].CITER_ELINK = 0;
//DMA_0.TCD[1].CITER_LINKCH = 0;
DMA_0.TCD[1].CITER = 1;
DMA_0.TCD[1].DOFF = 0x4;
DMA_0.TCD[1].DLAST_SGA = -16;
DMA_0.TCD[1].BITER = 1;
DMA_0.TCD[1].BWC = 0;
//DMA_0.TCD[1].MAJORLINKCH = 2;   /* Link to channel 2 */
DMA_0.TCD[1].MAJORELINK = 0;      /* Enable channel linking */
DMA_0.TCD[1].ESG = 0;
DMA_0.TCD[1].DREQ = 0;
DMA_0.TCD[1].INTHALF = 0;
DMA_0.TCD[1].INTMAJ = 0;
```

NOTE

Bit fields that are commented out are shown so that all of the 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 and it is linked to channel 1 by setting MAJORLINKCH = 1.

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

```
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 address 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 will 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 completed, the major loop is complete.

5.2 Example configuration 2: scatter/gather

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

5.2.1 Requirements

Two data arrays and one TCD array are created in internal SRAM. The first, data_array0[] is of a 56-byte size while data_array1[] is 8 bytes to demonstrate different set-ups for the TCDs.

These arrays will be placed at the beginning of the dedicated data SRAM at address 0x40001000 and are to be moved to 0x4001F000 (data_array0) and 0x4001FF00 (data_array1).

The TCD_SG 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 TCD_SG is loaded into the channel's TCD. 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.

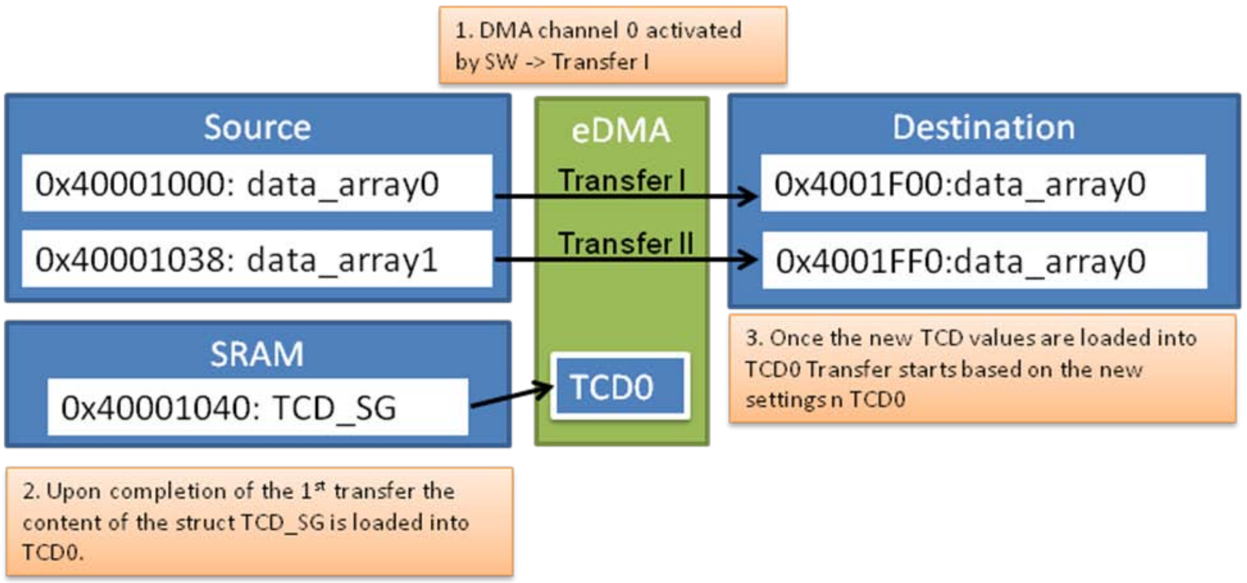
1. DMA channel 0 activated by SW -> Transfer I
Source
0x40001000: data_array0
0x40001038: data_array1
eDMA
Transfer I
Transfer II
Destination
0x4001F00:data_array0
0x4001FF0:data_array0
SRAM
0x40001040: TCD_SG
TCD0
3. Once the new TCD values are loaded into TCD0 Transfer starts based on the new settings n TCD0
2. Upon completion of the 1st transfer the content of the struct TCD_SG is loaded into TCD0.

Figure 9. Example 2 requirements

5.2.2 Module configuration

This example uses software channel activation for the first transfer. The second transfer is started automatically as soon as 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)&data_array0[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 = 56;  /* 14x32-bits */
DMA_0.TCD[0].SLAST = -56;
DMA_0.TCD[0].DADDR = 0x4001F000;
```

```
DMA_0.TCD[0].CITER_ELINK = 0;
//DMA_0.TCD[0].CITER_LINKCH = 0;
DMA_0.TCD[0].CITER = 1;
DMA_0.TCD[0].DOFF = 0x4;
DMA_0.TCD[0].DLAST_SGA = (int)&TCD_SG[0];
DMA_0.TCD[0].BITER = 1;
DMA_0.TCD[0].BITER_ELINK = 0;
//DMA_0.TCD[0].BITER_LINKCH = 0;
DMA_0.TCD[0].BWC = 0;
//DMA_0.TCD[0].MAJORLINKCH = 1;  /* Link to channel 1 */
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 = 0;
```

NOTE

Bit fields that are commented out are shown so that all of 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) is given below:

```
vuint32_t TCD_SG[] = {
      (int)&data_array1[0],0x02020004,
      0x00000008,0xfffffff8,
      0x4001ff00,0x00010004,
      0xfffffff8,0x00010001,
};
```

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 TCD_SG structure is configured according to the TCD memory map (compare Figure 7). Note that the first element of the array contains the source address (SADDR) and hence 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 as soon as the new TCD is loaded.

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

5.3 Example configuration 3: peripheral DMA transfer

This example configures the eDMA for a peripheral DMA transfer using the Sigma-Delta Analog-to-Digital Converter (SDADC) as the peripheral DMA source, demonstrating the use of the DMA multiplexer (DMAMUX).

NOTE

This example was developed on MPC5746M, using this microcontroller's SDADC. This module may not be part of other products in the MPC57xx family. Although the SDADC may not be present on all devices, it serves as a good example for transferring data to any peripheral module.

5.3.1 Requirements

SDADC_0 is configured for continuous conversion and FIFO full (end-of-conversion/EOC) DMA requests. The SDADC is continuously converting inputs and storing the results in the FIFO. When the FIFO is full, a DMA transfer requests to transfer the data to SRAM where a table of 10 × 16 results is built up. Upon completion of this table, an interrupt is generated.

In this application-typical scenario the DMA is transferring the data from the SDADC's buffer to a location in SRAM in ten minor loops to create a result table with a total of 160 conversion results. After ten minor loops, the major loop is completed. Further DMA requests are disabled and an interrupt request is issued to the core.

The DMA_Mux is configured to route the peripheral DMA request to an eDMA channel and this channel's TCD is programmed to transfer the data upon receiving the request.

When the channel performing the transfer is activated by the peripheral request, the first 16-bit piece of data in the sequence is moved from the source to the destination. After fifteen more transfers, the minor loop is complete and the DMA waits for the next transfer request. After a total of 10 minor loops, the major loop is completed, further DMA requests are disabled, and an interrupt is issued to the core.

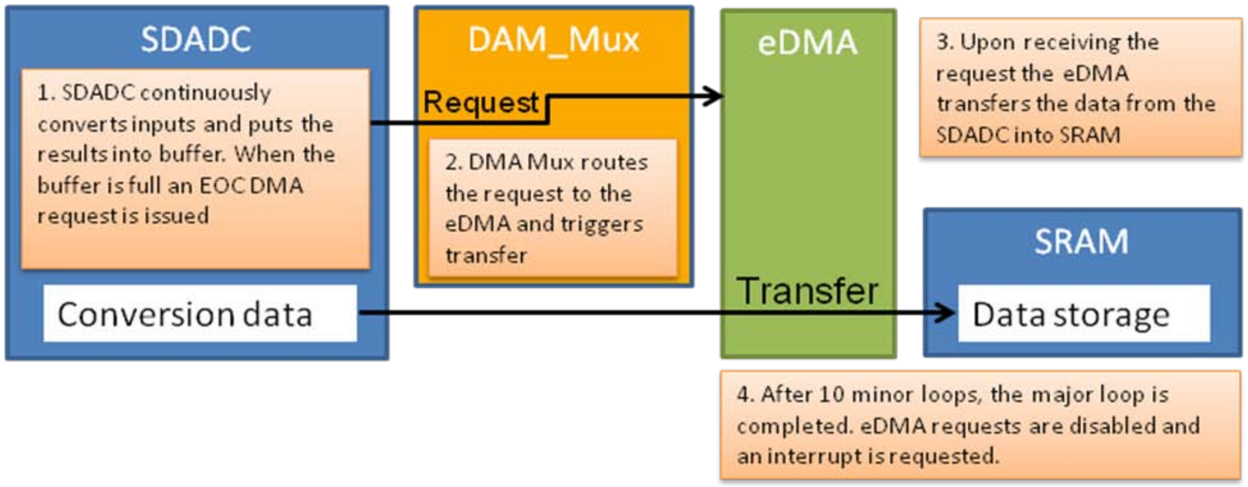
SDADC
1. SDADC continuously converts inputs and puts the results into buffer. When the buffer is full an EOC DMA request is issued
Conversion data
DAM_Mux
Request
2. DMA Mux routes the request to the eDMA and triggers transfer
eDMA
Transfer
3. Upon receiving the request the eDMA transfers the data from the SDADC into SRAM
SRAM
Data storage
4. After 10 minor loops, the major loop is completed. eDMA requests are disabled and an interrupt is requested.

Figure 10. Example 3 requirements

5.3.2 Module configuration

According to the peripheral DMA request table (Table 2), DMAMUX_0 is configured to route source 4, the SDADC_0 EOC DMA request, to channel 7 of the eDMA.

The eDMA's ERQL register has to be configured to enable SDADC DMA requests on channel 7.

Configuring 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 eDMA and DMA_Mux to perform the transfer on channel 7 is given below:

```
/* Configure CH7 */
DMA_0.TCD[7].SADDR = (vuint32_t)&SDADC_0.CDR.R + 2; /* Lower half-word from CDR
DMA_0.TCD[7].SMOD = 0;
DMA_0.TCD[7].SSIZE = 0x1;         /* 16-bit */
DMA_0.TCD[7].DMOD = 0;
DMA_0.TCD[7].DSIZE = 0x1;         /* 16-bit */
DMA_0.TCD[7].SOFF = 0;
DMA_0.TCD[7].NBYTES = 32;         /* 16x16-bits -> 16 results from the FIFO */
DMA_0.TCD[7].SLAST = 0;
DMA_0.TCD[7].DADDR = 0x4001E000;
DMA_0.TCD[7].CITER_ELINK = 0;
//DMA_0.TCD[7].CITER_LINKCH = 0;
DMA_0.TCD[7].CITER = 10;          /* 10 Minor loops to build up the result table */
DMA_0.TCD[7].DOFF = 0x2;
DMA_0.TCD[7].DLAST_SGA = -320;
DMA_0.TCD[7].BITER = 10;          /* 10 Minor loops to build up the result table */
DMA_0.TCD[7].BITER_ELINK = 0;
//DMA_0.TCD[7].BITER_LINKCH = 0;
DMA_0.TCD[7].BWC = 0;
DMA_0.TCD[7].MAJORLINKCH = 0;     /* No Linking */
DMA_0.TCD[7].MAJORELINK = 0;
DMA_0.TCD[7].ESG = 0;
DMA_0.TCD[7].DREQ = 1;            /* Clear channel's DMA request bit upon completion of
DMA_0.TCD[7].INTHALF = 0;
DMA_0.TCD[7].INTMAJ = 1;          /* The end-of-major loop interrupt is enabled */

/* Configure SDADC Mux channels */
DMACHMUX_0.CHCONFIG[7].R= 0x84;   /* DMA MUX 0 to set SDADC0 EOC on DMA channel 7 */

/* Enable channels */
DMA_0.ERQL.R = 0x00000080;        /* Enable channels 7 for SDADC eDMA requests */
```

NOTE

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

With this configuration, sixteen 16-bit values are transferred from the SDADC’s Converted Data Register (CDR) into SRAM per minor loop. There are ten minor loops: Citer/Biter = 10. The source address (SADDR) is configured to start at the lower half of the 32-bit register because the result is only 16-bit. The source offset (SOFF) is set to 0 because the FIFO automatically puts the new result into the CDR when the previous one has been read, until all 16 results have of the full FIFO have been read. The destination address offset (DOFF) is 2 bytes, to store the sixteen results in a total of 32 bytes of SRAM. DREQ is set to 1 to disable further DMA requests from the DMA_Mux. INTMAJ is set to 1 to request an interrupt at completion of a major loop. The code is configured in a way that it will get stuck in an infinite loop in the interrupt service routine to show that an interrupt occurred and had been serviced.

6 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 Register (ES), which can be a powerful tool for diagnosing problems with DMA transfers. Some hints for using the ES to debug errors when developing code are given below.

The structure of the ES is shown in Figure 11.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	UCE	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	CPE	ERRCHN						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11. Error Status Register

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

Table 4. Error Status Register field descriptions

Field	Description
VLD	ERRH and ERRL status bits, indicating an error in one of the channels. The exact channel number can be determined by checking the ERRH and ERRL registers.
UCE	Indicates that the last error was an uncorrectable ECC error in the TCD RAM. This error may be resolved by initializing the TCD RAM.
ECX	Transfer cancelled via the error cancel transfer bit DMA_CR[ECX].
CPE	Channel priority error. This error occurs when two or more channels have the same priority. Channel priorities have to be unique.
ERRCHN	Error channel number.
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 has to be programmed to be the same value as TCD_BITER.
SGE	Sgatter/gather error, this indicates a configuration error in the TCD_DLASTSGA field which has to be on a 32-byte boundary when sgatter/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.

7 Conclusion

This application note provides a good understanding and working knowledge of the MPC57xx eDMA controller, enabling the user to create eDMA configurations suitable for applications. The source code provided along with this application note can be used as a basis for configurations.

For more information on the Freescale MPC57xx family, visit: freescale.com

Document Number AN4765
Rev 1, 08/2013