Freescale Semiconductor
Application Note

AN2898
Rev. 0.0, 06/2005

Configuring and Using the MAC7100 eDMA Controller

by: Eric Ocasio and David McMenamin
Applications Engineering
Freescale 32-bit Embedded Controller Division

The MAC7100 family of Microcontrollers feature Freescale's enhanced Direct Memory Access (eDMA) controller.

1 Introduction

This application note will provide the reader with a working knowledge of the MAC7100 enhanced DMA controller. Topics covered include: introduction and overview of DMA controllers, features of the MAC7100 eDMA module, interaction between the eDMA and DMA multiplexer and configuration advice for your application. Examples are used throughout this document to demonstrate increasingly complex configurations.

A ZIP file (AN2898SW.zip) containing all of the examples used within this document is available for download from www.freescale.com/mac7100.

1.1 DMA controller Overview

A DMA controller provides the ability to move data from one memory mapped location to another. Once

Table of Contents

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.

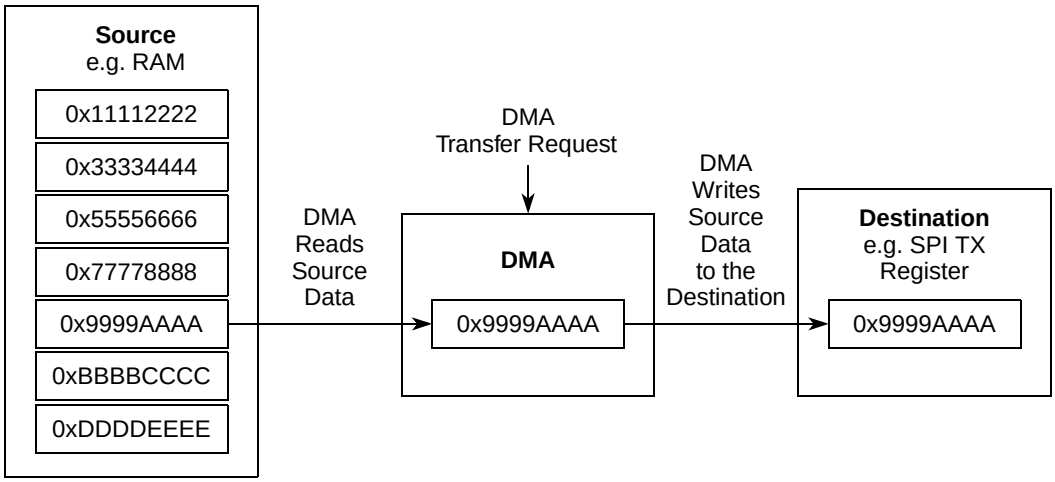

Figure 1. DMA Operational Overview

1.2 MAC7100 eDMA Controller Features

All MAC7100 devices feature a 16-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) memory array.

eDMA transfers can be activated in three ways:

1. Events occurring in peripheral modules and off chip can assert a DMA transfer request.
2. Software activation.
3. Channel to channel linking, where completion of a transfer on one channel activates another.

Each channel can generate an interrupt 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 automatically load a new TCD into its registers. Multiple TCD descriptors can therefore be used with a single channel without extra loading being put on the core.

1.3 eDMA Architectural Integration

To allow the eDMA and CPU to operate simultaneously a multi-master bus architecture is implemented. The MAC7100 multi-master bus has two master and three slave nodes.

The cross-bar switch forms the heart of this multi-master architecture. It links each master to the required slave device. If both masters attempt joint access to the same slave, an arbitration scheme commences, eliminating bus contention. Both fixed priority and round robin arbitration schemes are available.

Arbitration settings for the cross-bar can be configured in the cross-bar switch module registers. Please refer to the *MAC7100 Microcontroller Family Reference Manual* for more details.

The cross-bar switch and interaction between bus masters and slave devices is illustrated in Figure 2. In this example, the eDMA Controller is accessing one of the peripherals on the intelligent peripheral bus while the ARM7TDMI-S™ core is concurrently accessing the SRAM memory. The cross-bar switch has linked the appropriate buses for this situation.

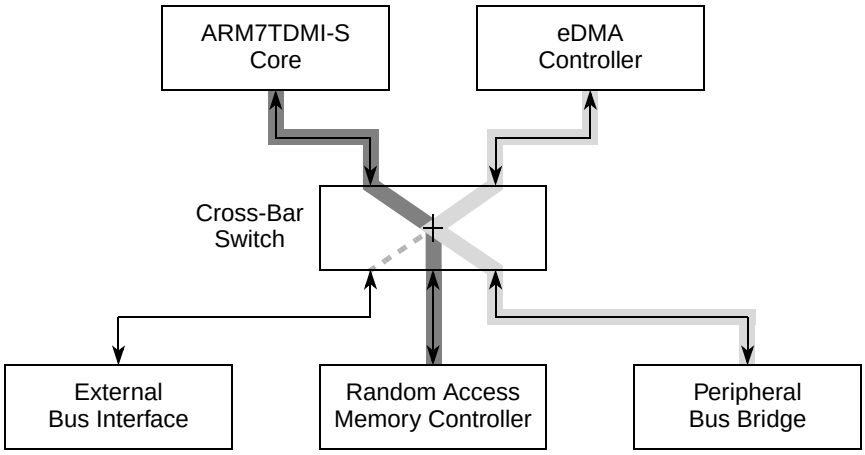

Figure 2. MAC7100 Multi-Master Bus Architecture

2 Activating eDMA Transfers

2.1 Activation Sources

Up to 42 events occurring within other peripheral modules can activate an eDMA transfer. In many modules, event flags can be asserted as either eDMA or Interrupt requests. Table 1 details the eDMA transfer request sources.

Table 1. eDMA Transfer Request Sources

Source	Requests	Comments
eSCI A, eSCI B, eSCI C, eSCI D	8	Each eSCI can generate two requests: one for transmission complete and one for data received.
DSPI A, DSPI B	4	Each DSPI can generate two requests: one for transmission complete and one for data received.
I^2C	2	Two Requests: one for transmission complete and one for data received.
ATD A, ATD B	4	Two requests for each ATD: one for command and one for result available.
eMIOS	16	Comparator match occurred flag for each channel can assert a request.
PIT	8	Any other request can be “gated” along with a PIT request. PIT requests can also be used in standalone to provide periodic data transfers.

NOTE

Table 1 shows all the request sources; on some derivatives, the request will not be present if the peripheral module is not implemented on the device.

Channels can also be activated by software and by channel linking. Each channel TCD provides a START bit, which activates the channel when asserted. This makes it possible to activate each channel in software. The START bit also provides a useful tool for test and debug, making it possible to assess if the channel operates as expected 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

As there are 42 peripheral request sources and 16 channels, a multiplexer is required to route the required request signal to the appropriate channel. The DMA Multiplexer (DMAMux) performs this task. It also provides the ability to gate a transfer request with the Periodic Interrupt Controller (PIT) on channels 0–7; this is discussed further in Section 2.3. The logical structure of the DMAMux is illustrated in Figure 3.

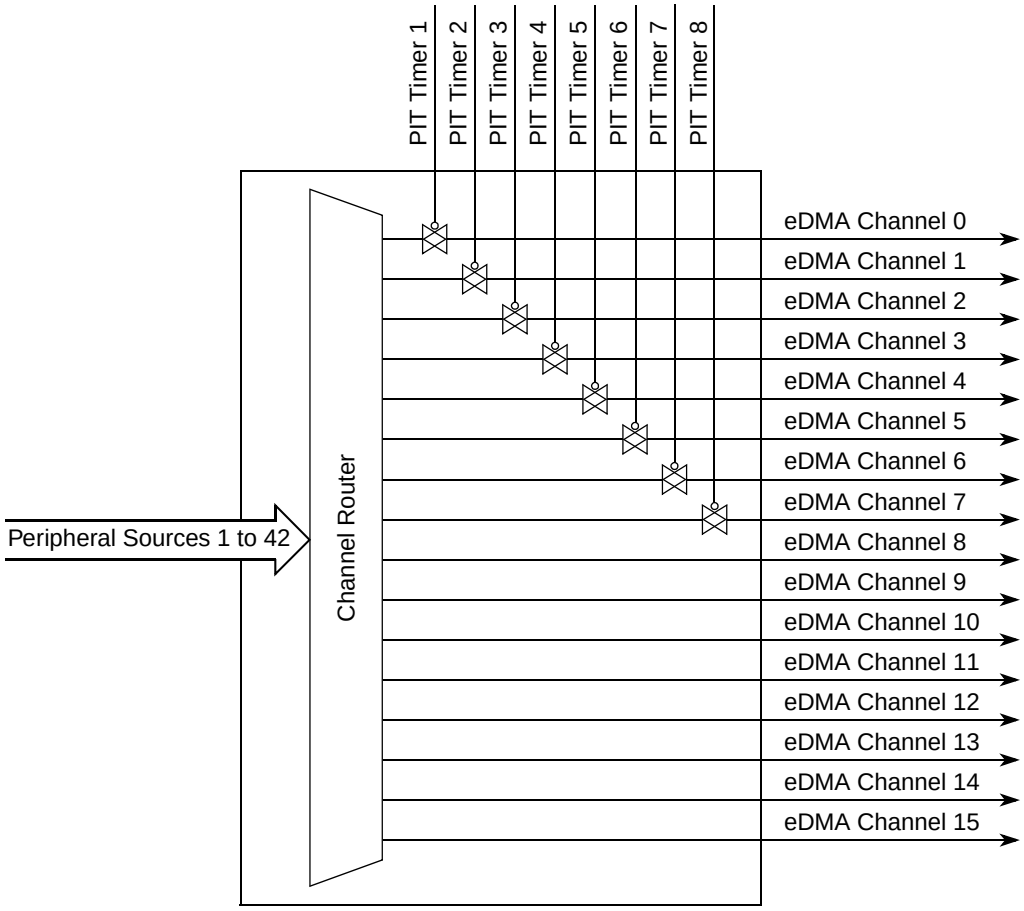

Figure 3. DMAMux Block Diagram

2.3 Activation Options

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

1. Disabled Mode: No request signal is routed to the eDMA channel and the channel is disabled. This is the reset state of a channel in the DMAMux. Disabled mode can also be used to suspend an eDMA channel while it is reconfigured or not required.
2. Normal Mode: A DMA request (for example, eSCI A transmission complete) is routed directly to the specified eDMA channel.
3. Periodic Trigger Mode: A PIT is used in conjunction with the DMA request source. For the request to be routed to the channel, both the DMA request source and the period interrupt must be active. This provides a means to “gate” or “throttle” transfer requests using the PIT. Figure 4 shows the relationship between the periodic interrupt, transfer request and the transfer activation.

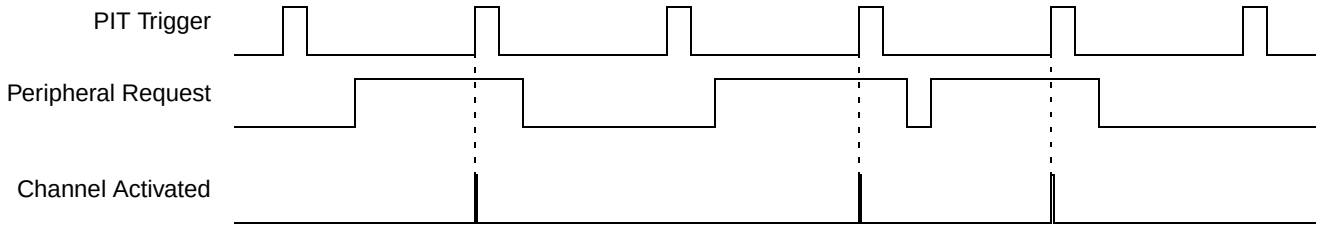

Figure 4. PIT Gated Transfer Activation

The hardware 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. See Table 2 for a full listing of request sources and their encodings.

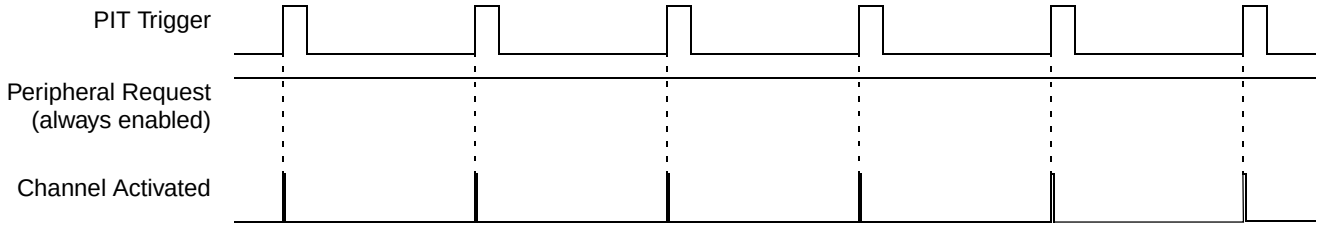

Figure 5. PIT Only Transfer Activation

Table 2. Transfer Request Sources and CHCONFIG*n* Encodings

SOURCE[5:0]	Request Source
0x00	Unassigned (disabled)
0x01	I^2C Transmit
0x02	I^2C Receive
0x03	DSPI_A Transmit
0x04	DSPI_A Receive
0x05	DSPI_B Transmit
0x06	DSPI_B Receive
0x07	eSCI_A Transmit
0x08	eSCI_A Receive
0x09	eSCI_B Transmit
0x0A	eSCI_B Receive
0x0B	eSCI_C Transmit
0x0C	eSCI_C Receive
0x0D	eSCI_D Transmit
0x0E	eSCI_D Receive

SOURCE[5:0]	Request Source
0x0F	eMIOS Channel 0
0x10	eMIOS Channel 1
0x11	eMIOS Channel 2
0x12	eMIOS Channel 3
0x13	eMIOS Channel 4
0x14	eMIOS Channel 5
0x15	eMIOS Channel 6
0x16	eMIOS Channel 7
0x17	eMIOS Channel 8
0x18	eMIOS Channel 9
0x19	eMIOS Channel 10
0x1A	eMIOS Channel 11
0x1B	eMIOS Channel 12
0x1C	eMIOS Channel 13
0x1D	eMIOS Channel 14

SOURCE[5:0]	Request Source
0x1E	eMIOS Channel 15
0x1F	ATD_A Result
0x20	ATD_A Command
0x21	ATD_B Result
0x22	ATD_B Command
0x23	Always Enabled 0
0x24	Always Enabled 1
0x25	Always Enabled 2
0x26	Always Enabled 3
0x27	Always Enabled 4
0x28	Always Enabled 5
0x29	Always Enabled 6
0x2A	Always Enabled 7

2.4 Handling Multiple Transfer Requests

Only one channel can actively perform a transfer. Therefore, to handle multiple pending transfer requests the eDMA controller offers channel prioritisation. Fixed priority or round robin prioritisation 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. The lower priority channel will halt its transfer on completion of the current read/write operation and allow the channel of higher priority to carry out its transfer. The lower priority channel will resume its transfer once 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, from the highest to the lowest, checking for a pending request. When a channel with a pending request is reached, it is allowed to perform its transfer. Once the transfer has been completed, the eDMA continues to cycle through the channels looking for the next pending request.

3 Transfer Process

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

3.1 Major and Minor Transfer Loops

Each time a channel is activated and executes, “n” bytes 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.

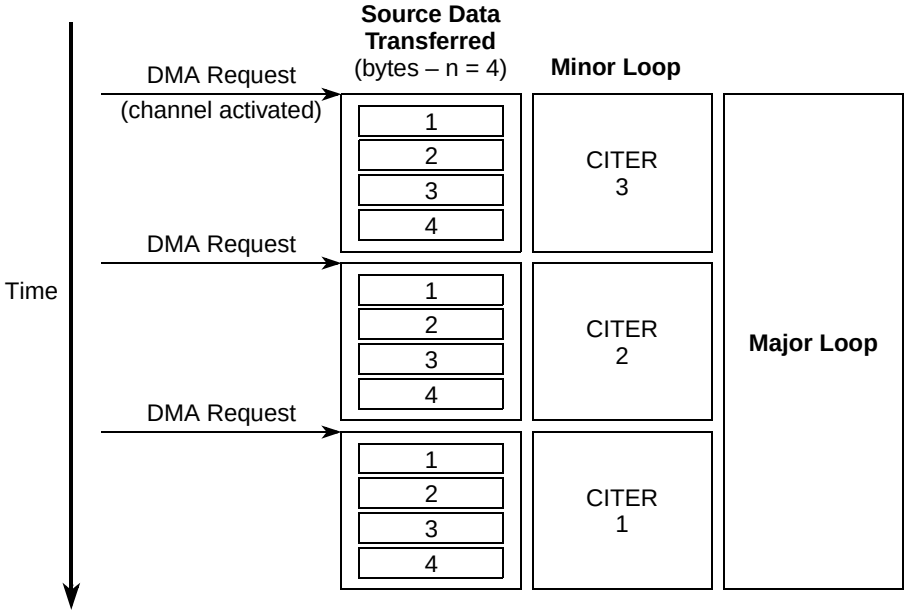

Figure 6. Major and Minor Transfer Loops

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

3.2 Completing a Minor Transfer Loop

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

- Decrements the current iteration counter
- Updates the source address by adding the current source address to the signed source offset. SADDR = SADDR + SOFF. The source address is updated automatically as transfers are performed. On completion of the minor loop the source address will contain the source address for the last piece of data that was read in the minor loop. The offset is added to this value.
- Updates the destination address by adding the current destination address to the signed destination offset. DADDR = DADDR + DOFF.
- Updates the channel status bits.

- If channel linking is enabled upon completion of the minor loop, the start bit of the linked channel is asserted.
- If the major loop is half complete and the major loop half complete interrupt is enabled, an interrupt request is asserted.

3.3 Completing a Major Transfer Loop

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

- Updates the source address by adding the current source address to the last source address adjustment. The last source address adjustment contains the address offset that should be added to the present source address in order to calculate the address of the new source data. SADDR = SADDR + SLAST.
- Updates the destination address by adding the current destination address to the last destination address adjustment. The last destination address adjustment contains the address offset that should be added to the present destination address in order to calculate the address of the new destination data. DADDR = DADDR + DLAST.
- Updates the channel status bits.
- If channel linking is enabled upon completion of the major loop the start bit of the linked channel is asserted.
- If the major loop complete interrupt is enabled, assert an interrupt request.
- The current iteration (CITER) field is reloaded from the beginning 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 please consult the *MAC7100 Microcontroller Family Reference Manual* (MAC7100RM).

4.1 Configuration Steps

To configure the eDMA the following initialisation steps should be followed:

1. Write the eDMA control register (only necessary if configuration other than default is required),
2. Configure channel priority registers in DCHPRI*x* (only necessary if configuration other than default is required),
3. Enable error Interrupts using either the DMAEEI or DMASEEI register (only necessary if configuration other than default is required),
4. Write the transfer control descriptors for channels that will be utilised, and
5. Configure the appropriate peripheral module and configure the eDMA to route the activation signal to the appropriate channel.

4.2 Transfer Control Descriptors

All transfer attributes for a channel are defined in the unique TCD for the channel. Each TCD is stored in the eDMA controller module SRAM. Only the DONE, ACTIVE and STATUS fields are initialised at reset. All other TCD fields are undefined at reset and must be written by software before the channel is activated. Failure to do this will result in unpredictable behaviour of the channel. Figure 7 shows the TCD memory map.

Bit Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	SADDR																															
32	SMOD					SSIZE			DMOD					DSIZE			SOFF															
64	NBYTES																															
96	SLAST																															
128	DADDR																															
160	CITER_E_LINK	CITER[14:9] or CITER_LINKCH							CITER[8:0]								DOFF															
192	DLAST_SGA																															
224	BITER_E_LINK	BITER[14:9] or BITER_LINKCH							BITER[8:0]								BWC	MAJOR_LINKCH							DONE	ACTIVE	MAJOR_E_LINK	E_SG	D_REQ	INT_HALF	INT_MAJ	START

Figure 7. TCD Memory Map

Table 4-3. Transfer Control Descriptor Field Descriptions

Bits	Name	Description
0–31	SADDR[31:0]	Source address. Memory address of the transfer source data. This 32-bit field allows any area of the MAC7100 memory map to be selected. As the eDMA performs transfers this field is automatically updated for the next transfer.
32–36	SMOD[4:0]	Source address modulo. Provides a simple implementation of a circular data queue. 00000 Source address modulo feature is disabled. non-0 The number of lower source address bits that are allowed to increment. A circular buffer is created as the lower address fields wrap to their original value while the upper fields remain fixed.
37–39	SSIZE[2:0]	Source data transfer size. This field defines the read data size for the eDMA engine. It does not define the amount of data transferred per channel activation. 000 8-bit. 100 16-byte burst. 001 16-bit. 101 Reserved. 010 32-bit. 110 Reserved. 011 Reserved. 111 Reserved. For example, for a transfer of 8 bytes per channel activation and SSIZE = 16-bits the eDMA will perform 4 16-bit reads. If SSIZE was 32-bits the eDMA would perform 2 reads for this transfer.
40–44	DMOD[4:0]	Destination address modulo. Can be utilised to implement a circular destination queues. As per SMOD but for the destination address.

Table 4-3. Transfer Control Descriptor Field Descriptions (continued)

Bits	Name	Description
45–47	DSIZE[2:0]	Destination data transfer size. Defines the data write size for the eDMA engine. As per SSIZE.
48–63	SOFF[15:0]	Source address signed offset. This signed offset is added to the current source address, upon completion of a minor loop, to calculate the new source address.
64–95	NBYTES[31:0]	Minor byte transfer count. Number of bytes to be transferred on each activation of the channel.
96–127	SLAST[31:0]	Last source address adjustment. This signed offset is added to the source address on 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.
128–159	DADDR[31:0]	Destination address. Memory address of the transfer destination. This 32-bit field allows any area of the MAC7100 memory map to be selected. As the eDMA performs transfers this field is automatically updated for the next transfer.
160 [1]	CITER_ E_LINK [1]	Enable channel linking on minor loop complete. As the channel completes a minor loop it asserts the START bit of the channel defined in CITER_LINKCH[5:0]. 0 Channel-to-channel linking is disabled. 1 Channel-to-channel linking is enabled. **Note:** This bit must be equal to the BITER_E_LINK bit, otherwise a configuration error will be reported.
161–166 [1]	CITER_ LINKCH[5:0] [1]	Minor loop complete link channel. If CITER_E_LINK is set, this 6-bit field specifies the channel that will be started following completion of the minor loop. 00xxxx Linked channel number xxxx. 01xxxx Reserved. 10xxxx Reserved. 11xxxx Reserved.
161–175 [1] or 167–175 [1]	CITER[14:0] [1] or CITER[8:0] [1]	Current major iteration count. This value 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 as 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.
176–191	DOFF[15:0]	Destination address signed offset. This signed offset is added to the current destination address upon completion of a minor loop, to calculate the next destination address.
192–223 [2]	DLAST_SGA [31:0] [2]	Last destination address adjustment or the memory address for the next TCD to be loaded into this channel. If scatter/gather is disabled (E_SG = 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 (E_SG = 1) this field is used as a pointer to a 0-modulo-32 region that contains the next TCD for this channel.
224 [3]	BITER_ E_LINK [3]	Beginning enable channel linking on minor loop complete. When a major loop is completed this field is used to re-load the CITER_E_LINK field. Hence when writing the BITER_E_LINK and CITER_E_LINK they must be configured to the same value.
225–230 [3]	BITER_ LINKCH[5:0] [3]	Beginning minor loop complete link channel. [3] When a major loop is completed, this field is used to re-load the CITER_LINKCH field. Hence when configuring the BITER_LINKCH and CITER_LINKCH they must be configured to the same value.
225–239 [3] or 231–239 [3]	BITER[14:0] [3] or BITER[8:0] [3]	Beginning major iteration count. [3] When a major loop is completed, this field is used to re-load 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.

Table 4-3. Transfer Control Descriptor Field Descriptions (continued)

Bits	Name	Description
240–241	BWC[1:0]	Bandwidth control. This provides a means of controlling the amount of bus bandwidth the eDMA uses. 00 No eDMA engine stalls (consume 100% bandwidth) 01 Reserved. 10 eDMA engine stalls for 4 cycles after each read/write 11 eDMA engine stalls for 8 cycles after each read/write
242–247 [3]	MAJOR_ LINKCH[5:0] [3]	Major loop complete link channel. 00xxxx Linked channel number. 11xxxx Reserved.
248	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.
249	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 is cleared, by the hardware, when that minor loop is complete.
250 [3]	MAJOR_ E_LINK [3]	Enable channel-to-channel linking on major loop complete. As the channel completes a major loop it asserts the START bit of the channel defined in MAJOR_LINKCH[5:0]. 0 Channel linking on completion of a major loop is disabled. 1 Channel linking on completion of a major loop is enabled.
251 [3]	E_SG [3]	Enable scatter/gather processing. If scatter/gather is enabled the channel loads a new TCD on completion of the major loop. DLAST_SGA provides the memory pointer to the new TCD structure that is to be loaded. 0 Scatter/gather processing is disabled. 1 Scatter/gather processing is enabled.
252	D_REQ	Disable request. If this bit is set when the channel completes a major loop, the eDMA clears the corresponding DMAERQ, disabling the transfer request. 0 The channel DMAERQ bit is not affected. 1 The channel DMAERQ bit is cleared when the major loop is complete.
253	INT_HALF	Enable an interrupt when major counter is half complete. When CITER = BITER ÷ 2 the eDMA asserts an interrupt request in the DMAINT register. 0 The major loop half-point interrupt is disabled. 1 The major loop half-point interrupt is enabled.
254	INT_MAJ	Enable an interrupt when major iteration count completes. When CITER = 0 the eDMA asserts an interrupt request in the DMAINT register. 0 The end-of-major-loop interrupt is disabled. 1 The end-of-major-loop interrupt is enabled.
255	START	Channel start. Writing this bit as 0b1 explicitly activates the channel and a minor loop transfer is performed. If a channel TCD is configured with an illegal value or an illegal combination of values, a channel error will be reported in the DMAERR register. 0 The channel is not explicitly started. 1 The channel is explicitly activated when this bit is written as a one.

NOTES:
1. Mask set L49P devices do not implement channel linking, and CITER_E_LINK must be written as zero. A 15-bit CITER count is always used.
2. Mask set L49P devices do not implement scatter/gather, thus this field is always used for DLAST.
3. Mask set L49P devices do not implement channel linking or scatter/gather; thus BITER_E_LINK, MAJOR_LINKCH[5:0], MAJOR_E_LINK and E_SG must be written as zero. A 15-bit BITER count is always used.

5 Example eDMA Configurations

This section walks through four example eDMA configurations, starting with a simple configuration and building on this to introduce the more advanced features and functions of the eDMA at an application level. All the example code is written using the Freescale MAC7100 header file, which is included in AN2898SW.zip.

5.1 Example 1: A Basic Transfer

This example configures the eDMA for a basic software triggered eDMA transfer. It assumes that the MAC7100 evaluation board is being used with the device operating in expanded mode and the external SRAM mapped to location 0.

5.1.1 Requirements

Three 32-bit data values, located in external SRAM memory, are to be relocated into internal SRAM. The data is located at address 0x500 and is to be moved to address 0x40002000. 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 the second activation, the second 32-bit value is transferred and on the third, the third piece of data. Figure 8 shows the requirements of this example.

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

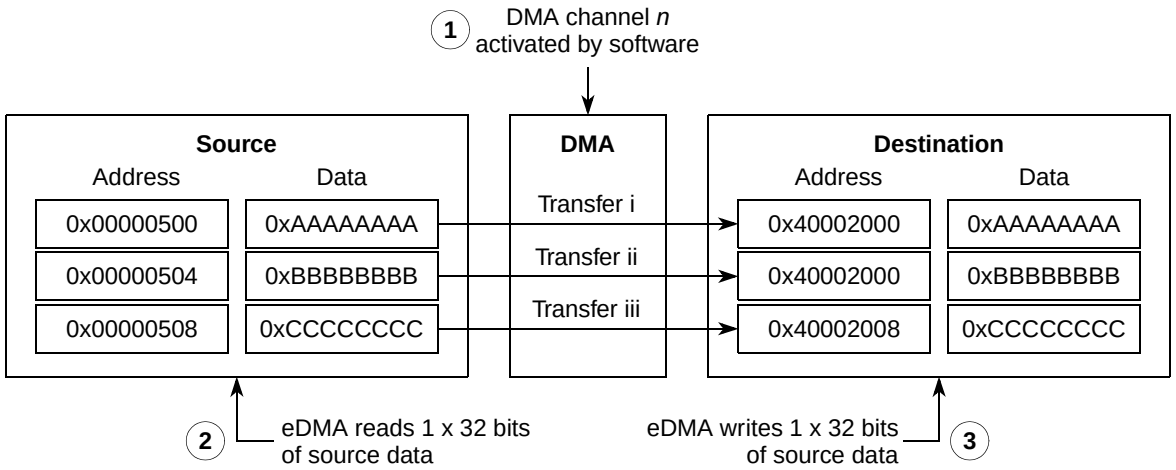

Figure 8. Example 1 Requirements

5.1.2 Module configuration

This example uses only software channel activation and the default eDMA wide configurations, therefore it is not necessary to configure the DMAMux or the eDMA module registers. 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 is given below:

```
/* Load data for DMA to Move*/
(*(unsigned long volatile *)(0x00000500)) = 0xAAAAAAAA;
(*(unsigned long volatile *)(0x00000504)) = 0xBBBBBBBB;
(*(unsigned long volatile *)(0x00000508)) = 0xCCCCCCCC;
/* Configure DMA Channel 0 TCD */
EDMAC_TCD0_W0 = EDMAC_SADDR(0x500);  /* Source Address = 0x500, Ext RAM */
EDMAC_TCD0_W1 = (0
       | EDMAC_SMOD(0x0)             /* Source Modulo, feature disabled */
       | EDMAC_SSIZE(0x2)            /* Source Size = 0x2 -> 32-bit transfers */
       | EDMAC_DMOD(0x0)             /* Destination Modulo, feature disabled */
       | EDMAC_DSIZE(0x2)            /* Destination Size = 0x2 -> 32-bit transfers */
       | EDMAC_SOFF(0x4));           /* Source address offset = 0x4 = 32-bit */
EDMAC_TCD0_W2 = EDMAC_NBYTES(0x4);   /* Transfer 4 bytes(32-bits) per ch activation*/
EDMAC_TCD0_W3 = EDMAC_SLAST(-12);    /* Restore Source address by -12 -> 3x32-bits */
EDMAC_TCD0_W4 = EDMAC_DADDR(0x40002000);/* Destination Address = 0x40002000, Int SRAM */
EDMAC_TCD0_W5 = (0
       /*| EDMAC_CITER_E_LINK        /* Do not set ELINK bit, no channel linking */
       | EDMAC_CITER(0x3)            /* Current Iteration Count -> 3x “NBYTES” xfer */
       | EDMAC_DOFF(0x4));           /* Destination address offset = 0x4 */
EDMAC_TCD0_W6 = EDMAC_DLAST(-12);    /* Restore Dest address by -12 -> 3x32-bits */
EDMAC_TCD0_W7 = (0
       /*| EDMAC_BITER_E_LINK        /* Do not set ELINK bit, no channel linking */
       | EDMAC_BITER(0x3)            /* Beginning Iteration Count = 3 = CITER */
       | EDMAC_BWC(0x0)              /* Bandwidth control = 0 -> No eDMA stalls */
       | EDMAC_MAJOR_LINKCH(0x0));   /* Ignored, no channel linking */
       /*| EDMAC_DONE                /* Done, status flag */
       /*| EDMAC_ACTIVE              /* Active, status flag */
       /*| EDMAC_MAJOR_E_LINK        /* Do not set ELINK bit, no channel linking */
       /*| EDMAC_E_SG                /* Do not set E_SG, no scatter-gather */
       /*| EDMAC_D_REQ               /* D_REQ = 0 -> DMAERQ bit not affected */
       /*| EDMAC_INT_HALF            /* No interrupt on half of major loop */
       /*| EDMAC_INT_MAJ             /* No interrupt on major loop complete */
       /*| EDMAC_START);             /* Do not explicitly start channel */
EDMAC_TCD0_W7 |= EDMAC_START         /* Activate Channel -> Perform Transfer i */
EDMAC_TCD0_W7 |= EDMAC_START         /* Activate Channel -> Perform Transfer ii */
EDMAC_TCD0_W7 |= EDMAC_START         /* Activate Channel -> Perform Transfer iii */
```

NOTE

Bit fields which 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.

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 the channel 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. Once all three transfers have been completed, the major loop is complete. Configuring n-bytes to be 12, the number of bytes that are moved in the example, would result in all three 32-bit pieces of data being moved to the destination in a single channel activation.

5.2 Example 2: PIT-Gated DMA Requests

In this example, the eDMA is used to supply the analog-to-digital converter (ATD) with a command word and move the result of ATD conversion to a location in external RAM. The ATD command word stores all

of the information that the ATD module requires for a conversion, so by using the DMA to provide the command words, the module can be instructed to perform conversions without any CPU intervention. Once the result is transferred by the eDMA to external RAM it is displayed pictorially using the 8 LEDs on the MAC7100 EVB. The application makes use of the potentiometer on the EVB to supply the ATD with a variable input voltage. As the POT is turned, the ATD input voltage will change, causing an illuminated LED to move up or down the LED bank.

5.2.1 Requirements

The input to the ATD should be sampled every 1 ms. To achieve this a 32-bit ATD command word must be supplied to the ATD command word register every 1 ms, when the module is able to accept the command. The ATD command word register is located at address 0xFC0E0010. This example will only require a single command word to be provided to the ATD; it is stored in a variable labelled “command.”

Once the ATD has completed the conversion the result is moved from the ATD result register, located at address 0xFC0E0014 to address 0x500 in external RAM. Figure 9 illustrates the functionality of this example.

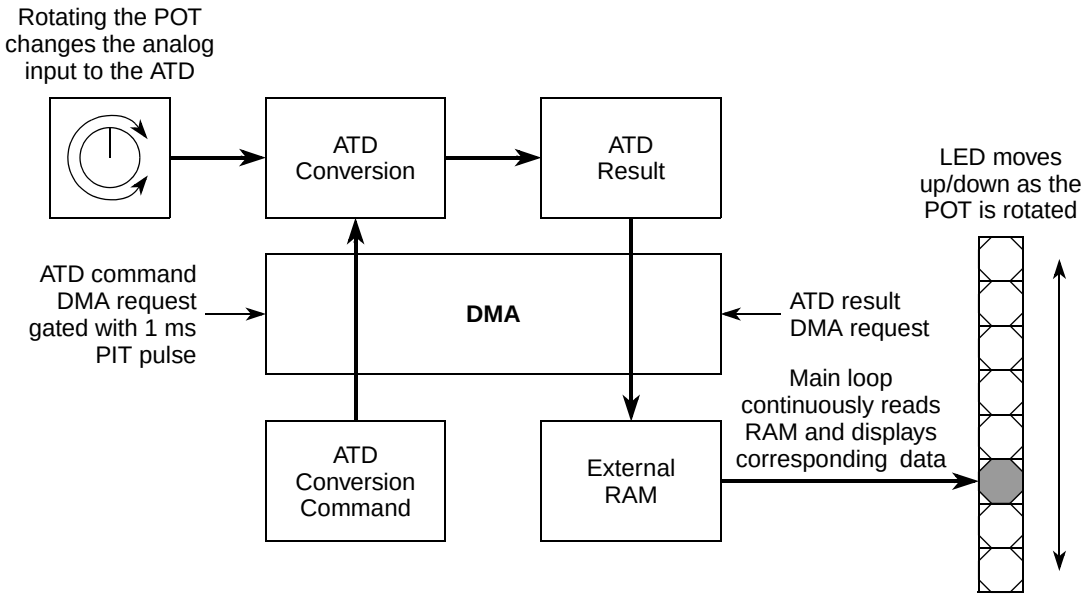

Figure 9. Example 2 Overview

5.2.2 Module Configuration

To implement this example two eDMA channels are required: one to transfer the command word and one to transfer the result. The command word transfer request requires both a 1 ms PIT trigger and the ATD command request flag to be asserted, ensuring that the module is able to perform the transfer and that the commands are supplied at a regular interval. The DMAMux must be configured for PIT gated channel activation. Channel 1 will be configured to perform this transfer.

Channel 0 will be used to transfer the ATD result to RAM. This transfer will be activated when the ATD result ready flag is asserted. The default channel arbitration will give channel 1 priority over channel 0.

This configuration ensures that the ATD receives a command word every 1 ms. It could however cause results to be overwritten in the result register before they have been moved by the eDMA, as the channel reading the results does not have priority. An ATD interrupt can be generated if a result is lost. The set up could be changed to ensure every result is captured to give the channel reading the results higher priority.

The resultant DMAMux configuration for channels 0 and 1 is:

```
/* Configure DMAMux for Channel 0 */
DMAMUX_CHCONFIG0 = (0
       | DMAMUX_ENABLE               /* Enable routing of DMA request */
       /*| DMAMUX_TRIG               /* Trigger Mode: Normal */
       | DMAMUX_SOURCE(0x1F));       /* Channel Activation Source: ATD_A Result */

/* Configure DMAMux for Channel 1 */
DMAMUX_CHCONFIG1 = (0
       | DMAMUX_ENABLE               /* Enable routing of DMA request */
       | DMAMUX_TRIG                 /* Trigger Mode: Periodic */
       | DMAMUX_SOURCE(0x20));       /* Channel Activation Source: ATD_A Command */
```

Channel 1 is configured to use a periodic trigger; the PIT 2 module must be enabled and configured for the desired time interval. For details on the PIT configuration, please refer to the source code for this example.

Each channel in this example is transferring data to or from the static-address, 32-bit wide command or result register, respectively. Therefore, it is necessary to restore the address pointers in the TCD when the major or minor transfer loop is complete. This example has no table of data to transfer, making only a single minor loop necessary to complete a major loop. The source and destination addresses are therefore restored on completion of the major loop.

The TCD configuration for channels 0 and 1 is:

```
/* Configure DMA Channel 0 TCD */
EDMAC_TCD0_W0 = EDMAC_SADDR(0xFC0E0014);/* Source Address = ATD Result Register
EDMAC_TCD0_W1 = (0
       | EDMAC_SMOD(0x0)             /* Source Modulo, feature disabled */
       | EDMAC_SSIZE(0x2)            /* Source Size = 0x2 -> 32-bit transfers */
       | EDMAC_DMOD(0x0)             /* Destination Modulo, feature disabled */
       | EDMAC_DSIZE(0x2)            /* Destination Size = 0x2 -> 32-bit transfers */
       | EDMAC_SOFF(0x0));           /* Source addr offset = 0x0, do not increment */
EDMAC_TCD0_W2 = EDMAC_NBYTES(0x4);   /* Transfer 4 bytes per channel activation */
EDMAC_TCD0_W3 = EDMAC_SLAST(0x0);    /* Do not adjust SADDR upon channel completion */
EDMAC_TCD0_W4 = EDMAC_DADDR(0x500);  /* Destination Address = 0x500, Ext RAM */
EDMAC_TCD0_W5 = (0
       /*| EDMAC_CITER_E_LINK        /* Do not set ELINK bit, no channel linking */
       | EDMAC_CITER(0x1)            /* Current Iter Count -> 1 "NBYTES" transfer */
       | EDMAC_DOFF(0x0));           /* Destination addr offset = 0x0, no increment */
EDMAC_TCD0_W6 = EDMAC_DLAST(0x0);    /* Do not adjust DADDR upon channel completion */
EDMAC_TCD0_W7 = (0
       /*| EDMAC_BITER_E_LINK        /* Do not set ELINK bit, no channel linking */
       | EDMAC_BITER(0x1)            /* Beginning Iteration Count = 1 = CITER */
       | EDMAC_BWC(0x0)              /* Bandwidth control = 0 -> No eDMA stalls */
       | EDMAC_MAJOR_LINKCH(0x0));   /* Ignored, no channel linking */
       /*| EDMAC_DONE                /* Done, status flag */
       /*| EDMAC_ACTIVE              /* Active, status flag */
       /*| EDMAC_MAJOR_E_LINK        /* Do not set ELINK bit, no channel linking */
       /*| EDMAC_E_SG                /* Do not set E_SG, no scatter-gather */
       /*| EDMAC_D_REQ               /* D_REQ = 0 -> DMAERQ bit not affected */
       /*| EDMAC_INT_HALF            /* No interrupt on half of major loop */
       /*| EDMAC_INT_MAJ             /* No interrupt on major loop complete */
```

```
        /*| EDMAC_START);             /* Do not explicitly start channel */
/* Configure DMA Channel 1 TCD */
EDMAC_TCD1_W0 = EDMAC_SADDR((uint32)&command);/* Source Addr = address of command var */
EDMAC_TCD1_W1 = (0
        | EDMAC_SMOD(0x0)             /* Source Modulo, feature disabled */
        | EDMAC_SSIZE(0x2)            /* Source Size = 0x2 -> 32-bit transfers */
        | EDMAC_DMOD(0x0)             /* Destination Modulo, feature disabled */
        | EDMAC_DSIZE(0x2)            /* Destination Size = 0x2 -> 32-bit transfers */
        | EDMAC_SOFF(0x0));           /* Source addr offset = 0x0, do not increment */
EDMAC_TCD1_W2 = EDMAC_NBYTES(0x4);    /* Transfer 4 bytes per channel activation */
EDMAC_TCD1_W3 = EDMAC_SLAST(0x0);     /* Do not adjust SADDR upon channel completion */
EDMAC_TCD1_W4 = EDMAC_DADDR(0xFC0E0010);/* Dest Addr = ATD Command Word Register */
EDMAC_TCD1_W5 = (0
        /*| EDMAC_CITER_E_LINK        /* Do not set ELINK bit, no channel linking */
        | EDMAC_CITER(0x1)            /* Current Iter Count -> 1 "NBYTES" transfer */
        | EDMAC_DOFF(0x0));           /* Destination addr offset = 0x0, no increment */
EDMAC_TCD1_W6 = EDMAC_DLAST(0x0);     /* Do not adjust DADDR upon channel completion */
EDMAC_TCD1_W7 = (0
        /*| EDMAC_BITER_E_LINK        /* Do not set ELINK bit, no channel linking */
        | EDMAC_BITER(0x1)            /* Beginning Iteration Count = 1 = CITER */
        | EDMAC_BWC(0x0)              /* Bandwidth control = 0 -> No eDMA stalls */
        | EDMAC_MAJOR_LINKCH(0x0));   /* Ignored, no channel linking */
        /*| EDMAC_DONE                /* Done, status flag */
        /*| EDMAC_ACTIVE              /* Active, status flag */
        /*| EDMAC_MAJOR_E_LINK        /* Do not set ELINK bit, no channel linking */
        /*| EDMAC_E_SG                /* Do not set E_SG, no scatter-gather */
        /*| EDMAC_D_REQ               /* D_REQ = 0 -> DMAERQ bit not affected */
        /*| EDMAC_INT_HALF            /* No interrupt on half of major loop */
        /*| EDMAC_INT_MAJ             /* No interrupt on major loop complete */
        /*| EDMAC_START);             /* Do not explicitly start channel */
```

Using these configurations will produce the required eDMA functionality for this example. Please refer to the full source code for this example in the ZIP file.

5.3 Example 3: Circular Buffers

This example highlights the functionally of the address Modulo feature which simplifies the implementation of circular data buffers.

5.3.1 Requirements

A 16-byte source buffer is located at address 0x40002000. It is required that multiple copies of the source data be used to fill two 64-byte buffers located at address 0x40004000 and 0x40005000. Figure 10 shows this requirement. The first buffer should be filled on the first channel activation and the second on the second activation.

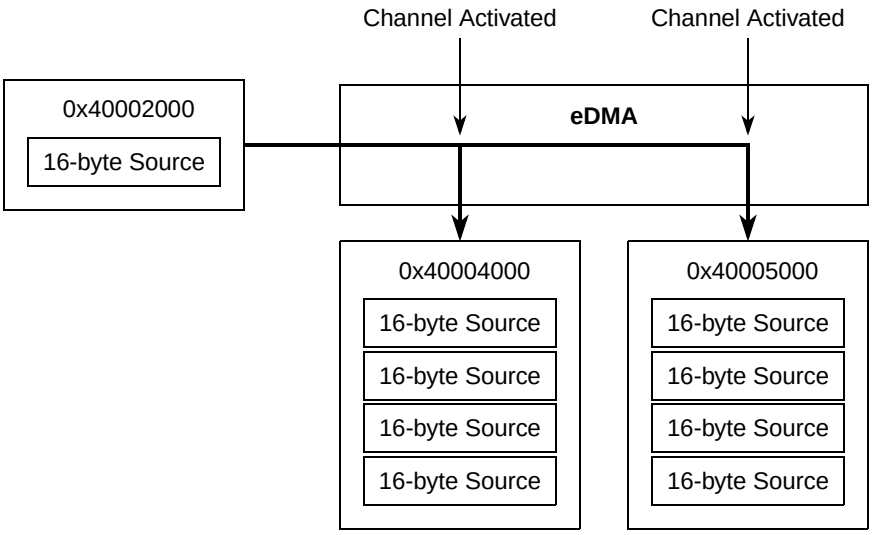

Figure 10. Example 3 Overview

5.3.2 Module Configuration

Each time the channel is activated the 16-bytes of source data must be transferred to the destination address four times. 64-bytes must therefore be transferred for each channel activation. To create a circular data queue the SMOD field is configured to equal four. This fixes all the address bits in the source address except for the lowest four bits. For example:

Transfer	Source Address	Transfer	Source Address
1	0x40002000	17	0x40002000
2	0x40002001	18	0x40002001
3	0x40002002	19	0x40002002
4	0x40002003	20	0x40002003
5	0x40002004	21	0x40002004
6	0x40002005	22	0x40002005
7	0x40002006	23	0x40002006
8	0x40002007	24	0x40002007
9	0x40002008	25	0x40002008
10	0x40002009	26	0x40002009
11	0x4000200A	27	0x4000200A
12	0x4000200B	28	0x4000200B
13	0x4000200C	29	0x4000200C
14	0x4000200D	30	0x4000200D
15	0x4000200E	31	0x4000200E
16	0x4000200F	32	0x4000200F

If the SMOD field were set to zero, after the 16th transfer the source address would continue to increment. When only the lower four bits are allowed to increment the address wraps to its original value once the maximum value has been reached in the lower four bits.

The TCD configuration for this example is:

```
/* Configure DMA Channel 0 TCD */
EDMAC_TCD0_W0 = EDMAC_SADDR(0x40002000);/* Source Address = 0x40002000
EDMAC_TCD0_W1 = (0
       | EDMAC_SMOD(0x4)             /* Source Modulo = 4 -> 16-byte circular queue */
       | EDMAC_SSIZE(0x0)            /* Source Size = 0x0 -> 8-bit transfers */
       | EDMAC_DMOD(0x0)             /* Destination Modulo, feature disabled */
       | EDMAC_DSIZE(0x0)            /* Destination Size = 0x0 -> 8-bit transfers */
       | EDMAC_SOFF(0x1));           /* Source address offset = 0x1, increment by 1 */
EDMAC_TCD0_W2 = EDMAC_NBYTES(0x40);  /* Transfer 64 bytes per channel activation */
EDMAC_TCD0_W3 = EDMAC_SLAST(0x0);    /* Do not adjust SADDR upon channel completion */
EDMAC_TCD0_W4 = EDMAC_DADDR(0x40004000);/* Destination Address = 0x40004000 */
EDMAC_TCD0_W5 = (0
       /*| EDMAC_CITER_E_LINK        /* Do not set ELINK bit, no channel linking */
       | EDMAC_CITER(0x1)            /* Current Iteration Count -> 1 “NBYTES” transfer
*/
       | EDMAC_DOFF(0x1));           /* Destination address offset = 0x0, no increment
*/
EDMAC_TCD0_W6 = EDMAC_DLAST(0xFC0);  /* Add 0xFC0 bytes to DADDR on channel complete */
EDMAC_TCD0_W7 = (0
       /*| EDMAC_BITER_E_LINK        /* Do not set ELINK bit, no channel linking */
       | EDMAC_BITER(0x1)            /* Beginning Iteration Count = 1 = CITER */
       | EDMAC_BWC(0x0)              /* Bandwidth control = 0 -> No eDMA stalls */
       | EDMAC_MAJOR_LINKCH(0x0));   /* Ignored, no channel linking */
       /*| EDMAC_DONE                /* Done, status flag */
       /*| EDMAC_ACTIVE              /* Active, status flag */
       /*| EDMAC_MAJOR_E_LINK        /* Do not set ELINK bit, no channel linking */
       /*| EDMAC_E_SG                /* Do not set E_SG, no scatter-gather */
       /*| EDMAC_D_REQ               /* D_REQ = 0 -> DMAERQ bit not affected */
       /*| EDMAC_INT_HALF            /* No interrupt on half of major loop */
       /*| EDMAC_INT_MAJ             /* No interrupt on major loop complete */
       /*| EDMAC_START);             /* Do not explicitly start channel */
```

By activating the channel twice the two 64-byte data buffers will be created at the appropriate destination address. Notice the TCD is configured so that after a buffer is filled a major loop is complete. The start address of the next buffer is calculated at this point by adding the appropriate offset to the current source address.

6 Conclusion

This application note should have provided you with a good understanding of the MAC7100 eDMA controller.

You should now be able to create eDMA configurations suitable for your application. The source code provide along with this application note can be used as a basis for your configurations.

For more information on the Freescale MAC7100 family, please see www.freescale.com/mac7100

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2898
Rev. 0.0
06/2005