Freescale Semiconductor
Application Note

Document Number:AN4522
Rev. 1, 6/2012

Examples of Setting the DMA Controller on the Power Architecture® MPC5675K Family of Microcontrollers

by: Vladimir Skorocky
Automotive and Industrial Solutions Group

Contents

1 Introduction

Direct memory access (DMA) is a technique for the transfer of data between main memory and a peripheral device without passing it through the CPU. It is possible to achieve higher transmission speed by parallel work with CPU, because the DMA controller is used only for data transfer. Transfer speed of the DMA controller is limited with the simultaneous access to the internal buses which could be shared between bus masters and bus slaves. It is possible to increase the speed by appropriate setting of the priority level in the X-BAR modules, the DMA controller, and the DMA multiplexer. The second factor that limits transfer speed is the response time of the connected peripheral device.

This application note provides information about using DMA controller on the microcontrollers from the MPC5xxx family. The main purpose of this application note is to show how the DMA controller can be used to transfer, link, and split relatively complex data blocks. The description of the main control registers is given with respect to the presented examples. For more detailed description, please see MPC5675KRM: MPC5675K Microcontroller—Reference Manual , available on http://www.freescale.com.

1.1 Features of the data transfer over DMA

The following modules are used for the data transfer over DMA:

- CPU: Starts the data transfer over DMA and solves the DMA interrupts and errors.
- X-BAR: This module is the arbiter that decides which master module gets the access to a slave by the simultaneous request from two or more masters. The priority of the individual masters is set by the X-BAR control registers. For example, the masters for for the microcontroller MPC5675K are:
  - Core0 and Core1 instruction ports
  - Core0 and Core1 load store port
  - DMA0
  - FlexRay

  Similarly, the slaves for the same microcontroller are:
  - Instruction port
  - Load store port
  - Peripheral bridge

  The system can connect only one master and slave at a time. In case several masters need one slave, X-BAR assigns the slave according to the priority set in the X-BAR control registers.
- DMA Multiplexer: Connects the peripheral requests to the DMA. All requests from the peripheral devices can be connected to any DMA channel. Some of the devices such as GPIOs and memory, have a quick response time, and do not need a request. Such devices are known as 'always requestors.'
- DMA: Includes independent channels that can connect peripheral device with memory, or memory with memory for data transfers. The channels can work independently with simultaneous access to memory and periphery, or can be set for sequential transfer. The technique for sequential transfer of the data blocks is known as link. The main purpose of this application note is to describe the possibilities of the link on the examples of the different levels.

Figure 1 shows a parallel data flow over DMA M2 master port and M1 Core0 load/store port by simultaneous access to the RAM memory. DMA channel 31 transfers data from SRAM0 to peripheral bridge and at the same time, the CPU requests for access to SRAM0.

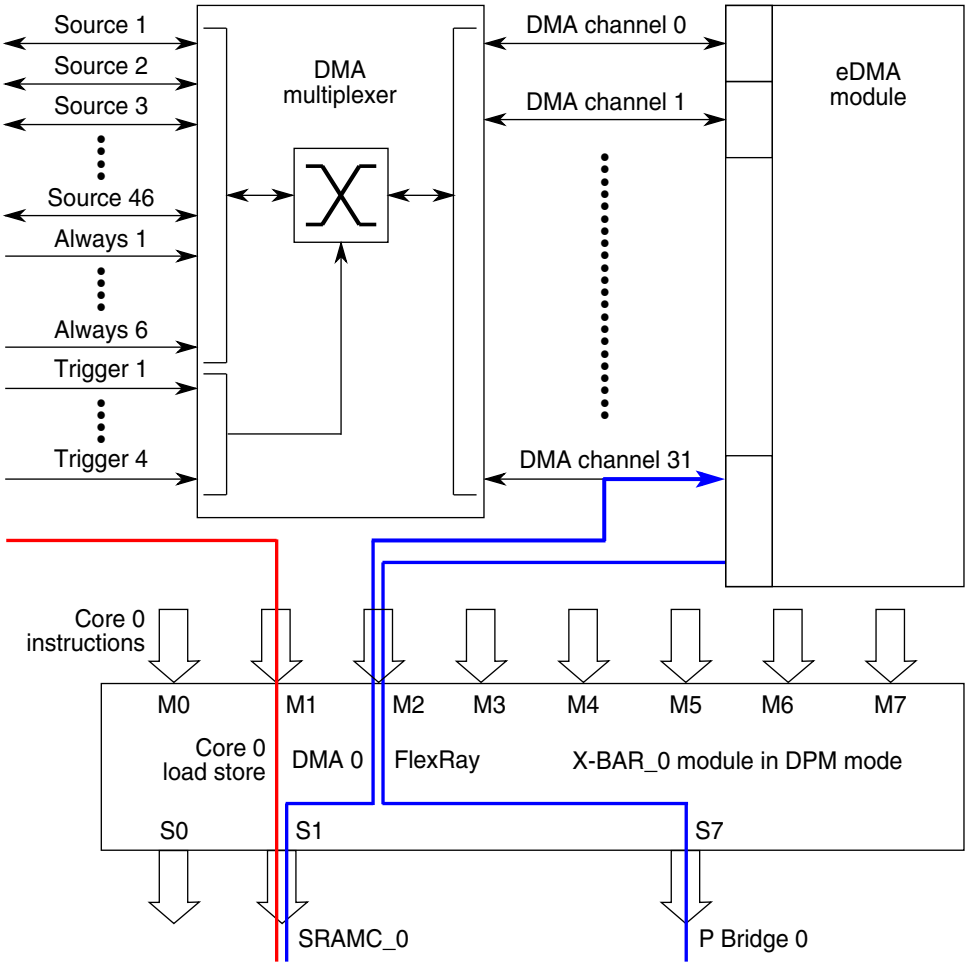


Figure 1. Transfer between SRAM and the peripheral bridge.

1.2 Brief description of the DMA module on the MPC5675x

There are 32 DMA channels on the MPC5675x MCUs for data transfer from memory to memory, or from memory to a peripheral device. These channels can work concurrently or sequentially. Other microcontrollers from this family have the same or similar registers. For more detailed information, see MPC5675KRM: MPC5675K Microcontroller—Reference Manual, available at http://www.freescale.com. The main difference between this microcontroller and others is in the number of channels.

The following table presents a short description of the main control registers of the MPC5675K MCU:

Table 1. MPC5675K MCU control registers

Register	Description
DMACR	DMA Control Register Defines the basic operating configuration of the eDMA module.

Table continues on the next page...

Table 1. MPC5675K MCU control registers (continued)

Register	Description
DMAES	DMA Error Status register Provides information concerning the last recorded channel error.
DMAERQL	DMA Enable Request Low register Provides a bit map for the 32 implemented channels to enable the request signal for each channel.
DMAEEIL	DMA Enable Error Interrupt Low register Provides a bit map for the 32 implemented channels to enable the error interrupt signal for each channel
DMAINTL	DMA Interrupt Request Low register Provides a bit map for the 32 implemented channels signaling the presence of an interrupt request for each channel.
DMAERRL	DMA Error Low register Provides a bit map for the 32 implemented channels signaling the presence of an error for each channel.
DMAGPOR	DMA General-Purpose Output register Indicates to the Cache Coherency module that writes from this DMA are marked as global.
DMA Channel n Priority registers	When the fixed-priority channel arbitration mode is enabled (DMACR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within a group.
TCD	Transfer Control Descriptor Information on TCD is described in Description of the eDMA basic registers

1.3 DMA channel multiplexer (DMACHMUX)

DMA multiplexer is the second most important module in DMA transfer configuration. The schematic drawing is shown in Figure 2.

Following are the main features:

- 52 peripheral sources (named peripheral slots). The first 46 sources are dedicated for peripheral requests.
- It is possible to trigger the first four slots through dedicated peripheral or external trigger.
- Last six of them are 'always requestors,' which suggests that transfer is done as soon as possible without the peripheral request.
- 32 independently selectable DMA channels.

NOTE

Typical 'always requestors' are memory and GPIO.

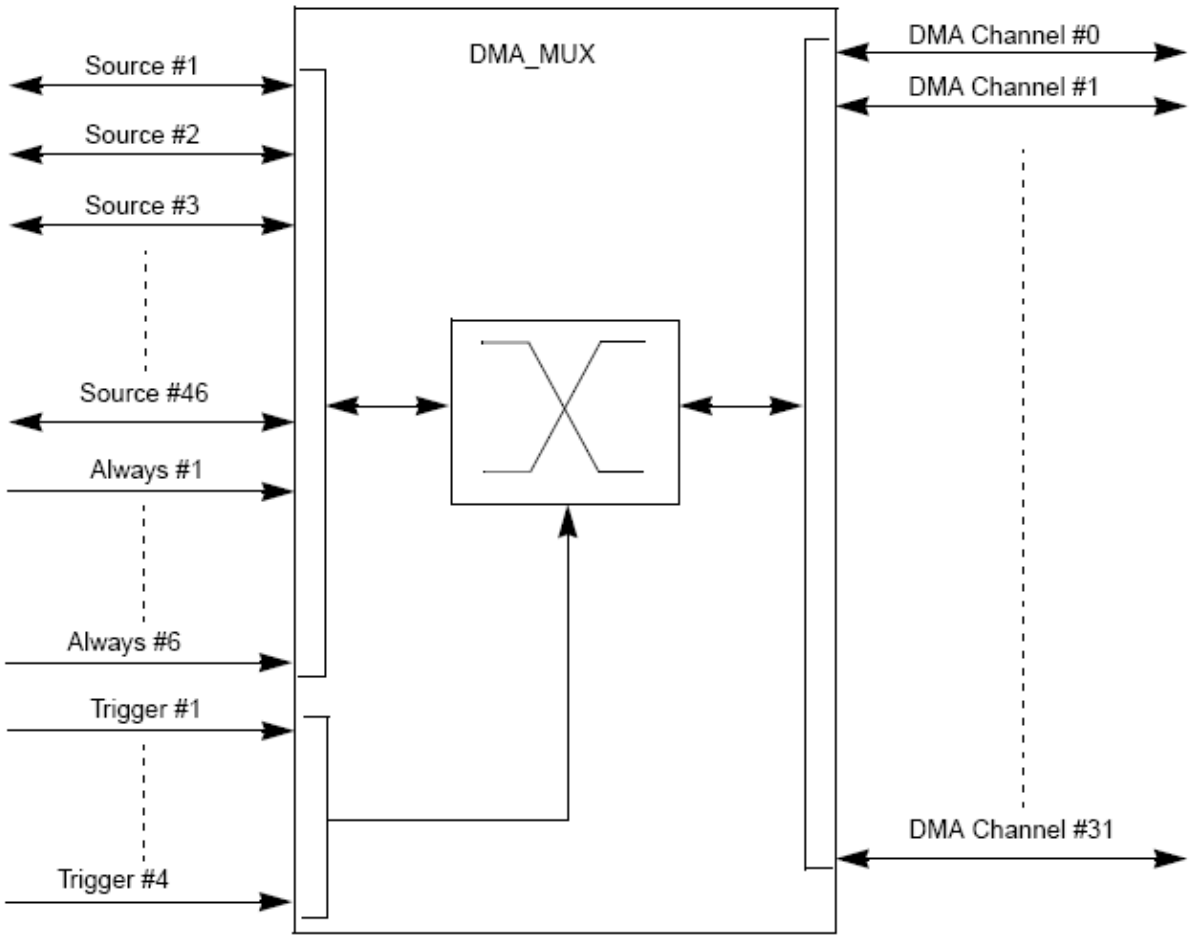
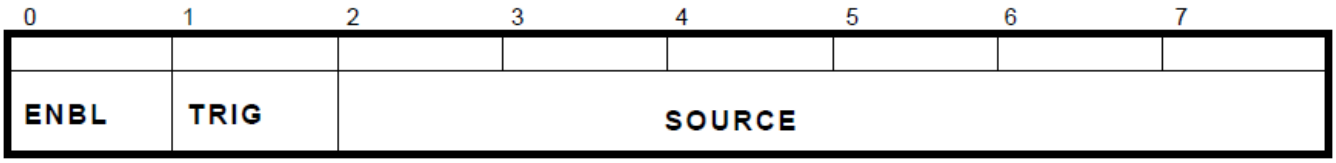


Figure 2. DMA multiplexer

Channel Configuration (CHCONFIGx) register: There are 32 CHCONFIGx registers mapped into memory on the address 0xFFFD_C000 to 0xFFFD_C01F. Following is the field description of the CHCONFIGx register:

Table 2. CHCONFIGx field description

Field Name	Description
ENBL	DMA Channel Enable bit Has to be configured for this register for all the active channels.
TRIG	DMA Channel Trigger Enable Enables the periodic trigger capability for the DMA Channel. The trigger bit is valid only for first four slots.
SOURCE	DMA Channel Source field Specifies the DMA source for relevant channel.



The following table shows the numbering of all sources in MPC567xx:

Table 3. MPC567xx DMA sources

Number of source	Name of the source (slot)
1	DSPI_0 DSPI_TFFF
2	DSPI_0 DSPI_RFDF
3	DSPI_1 DSPI_TFFF
4	DSPI_1 DSPI_RFDF
5	DSPI_2 DSPI_TFFF
6	DSPI_2 DSPI_RFDF
7	CTU CTU
8	CTU FIFO1
9	CTU FIFO2
10	CTU FIFO3
11	CTU FIFO4
12	FlexPWM_0 comp_val
13	FlexPWM_0 capt
14	eTimer_0 DREQ0
15	eTimer_0 DREQ1
16	eTimer_1 DREQ0
17	eTimer_1 DREQ1
18	eTimer_2 DREQ0
19	eTimer_2 DREQ1
20	ADC_0 DMA
21	ADC_1 DMA
22	LINFlexD_0 Transmit
23	LINFlexD_0 Receive
24	LINFlexD_1 Transmit
25	LINFlexD_1 Receive
26	FlexPWM_1 comp_val
27	FlexPWM_1 capt
28	CTU_1 CTU
29	CTU_1 FIFO1
30	CTU_1 FIFO2
31	CTU_1 FIFO3
32	CTU_1 FIFO4
33	ADC_2 DMA

Table continues on the next page...

Table 3. MPC567xx DMA sources (continued)

Number of source	Name of the source (slot)
34	ADC_3 DMA
35	LINFlexD_2 Transmit
36	LINFlexD_2 Receive
37	LINFlexD_3 Transmit
38	LINFlexD_3 Receive
39	FLEXPWM_2 comp_val
40	FLEXPWM_2 capt
41	IIC_0 Transmit
42	IIC_0 Receive
43	IIC_1 Transmit
44	IIC_1 Receive
45	IIC_2 Transmit
46	IIC_2 Receive
47	Not used
48	Always Requestor
49	Always Requestor
50	Always Requestor
51	Always Requestor
52	Always Requestor
53	Always Requestor

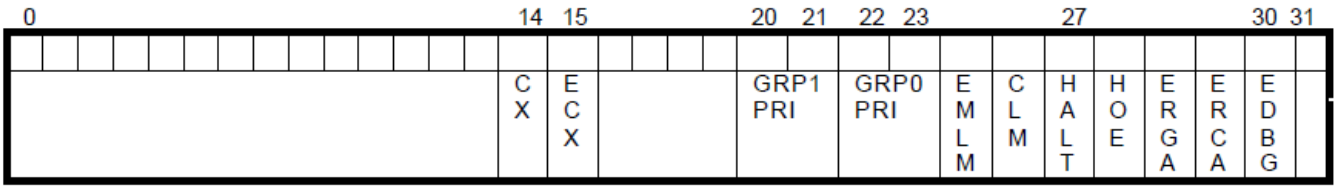
Example: To trigger the transfer of DSPI_0_DSPI_RFDF over DMA channel 4, set CHCONFIG4 = 0xC2 where 4 is the number of the DMA channel, 0xC enables the channel and sets it in the triggered mode, and 2 is the number of the source.

2 Description of the eDMA basic registers

This section describes how to set important control registers with respect to more complex transfers that are shown in Examples of link to the DMA channels. These examples are based on the memory-to-memory transfer. This type of transfer is easy to use because it does not need peripheral requests. For explanation of the examples, it is sufficient to describe only the DMA Control Register (DMACR) and the Transfer Control Descriptors (TCD). For more information about other registers, see MPC5675KRM: MPC5675K Microcontroller- Reference Manual, available on http://www.freescale.com.

2.1 DMA Control Register (DMACR)

This is a memory-mapped register at the address 0xFFF4_4000 and is common for all the DMA channels. This register sets mode of the arbitration, modifies the TCD fields, and can stop the DMA transfer.



The following table provides the field description of the DMACR.

Table 4. DMACR field description

Field name	Number	Description
CX	14	Cancel Transfer 0 Normal operation 1 Cancel the remaining data transfer
ECX	15	Enable Cancel Transfer
GRP1PRI	20-21	Channel Group 1 Priority
GRP0PRI	22-23	Channel Group 0 Priority
EMLM	24	Enable Minor Loop Mapping This bit is described in Transfer Control Descriptor (TCD)
CLM	25	Continuous Link Mode 0 A minor loop channel link made to itself goes through channel arbitration 1 A minor loop channel link made to itself does not go through channel arbitration before being activated again.
HALT	26	Halt DMA Operations
HOE	27	Halt On Error
ERGA	28	Enable Round-Robin Group Arbitration
ERCA	29	Enable Round-Robin Channel Arbitration
EDBG	30	Enable Debug

2.2 Transfer Control Descriptor (TCD)

This section describes the TCD data structure in detail. In the TCDs, the basic parameters of data transfer such as source and destination address, number of major and minor loops, and address correction after each data transfer, are set. Data transfer starts when the start bit of the appropriate TCD is set to 1. For the MPC5675K MCU, the DMACR block is mapped on the address 0xFFFD_C000. The first TCD is shifted up by 0x1000 bytes, that is, on the address 0xFFFD_D000. The description of particular components is presented in the following sections.

NOTE

The description of the registers and its positions are specified only for the MPC5675K MCU.

The following table shows the mapping of the basic parameters of the transfer in TCD field:

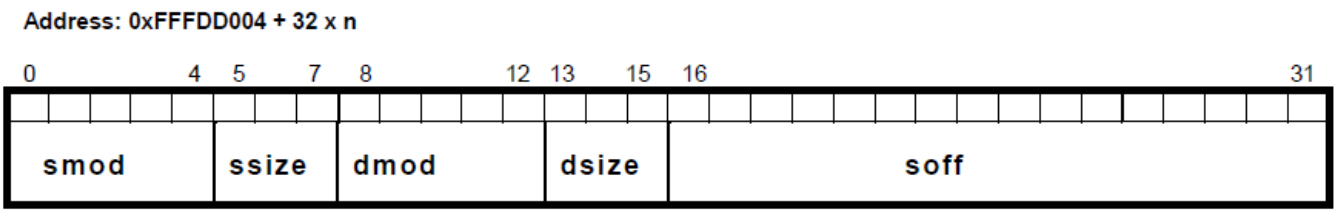
Table 5. Basic transfer parameters

DMA Offset	TCDn field	
0x1000 + (32 × n) + 0x00	Source Address (saddr)	
0x1000 + (32 × n) + 0x04	Transfer Attributes (smod, ssize, dmod, dsize)	Signed Source Address Offset (soff)
0x1000 + (32 × n) + 0x08	Signed Minor Loop Offset (smloe, dmloe, mloff)	Inner “Minor” Byte Count (nbytes)
0x1000 + (32 × n)+ 0x0C	Last Source Address Adjustment (slast)	
0x1000 + (32 × n) + 0x10	Destination Address (daddr)	
0x1000 + (32 × n) + 0x14	Current “Major” Iteration Count (citer)	Signed Destination Address Offset (doff)
0x1000 + (32 × n) + 0x18	Last Destination Address Adjustment/Scatter Gather Address (dlast_sga)	
0x1000 + (32 × n)+ 0x1C	Beginning “Major” Iteration Count (biter)	Channel Control/Status (bwc, major.linkch, done, active, major.e_link, e_sg, d_req, int_half, int_maj, start)

2.2.1 Source address

The source address specifies the memory address pointing to the source data.

2.2.2 Source modulo (smod), source size (ssize), destination modulo (dmod),destination size (dsize)



The following table presents the field descriptions.

Table 6. Field descriptions

Field name	Description
smod	Source Modulo This 5-bit field specifies the size of the mask for the source address. It is useful if there is a need to repeat the data blocks on the minor loop level. For this purpose, the beginning of the transferred block must be aligned on the proper address. For instance, when smod is 7, it is necessary to align the address of the data block on address 0bxxxx_xxxx_x000_0000 having seven zeroes. When the address reaches the value 0bxxxx_xxxx_x111_1111, the address pointer will be set on the value 0bxxxx_xxxx_x000_0000 again. Disadvantage of this solution is that only the data blocks with length 2^N can be transferred. When the value of smod or dmod is set to 0, the modulo feature is disabled.
ssize	Source Size Determines the size of the source data 3b000 = 8 bits 3b001 = 16 bits 3b010 = 32 bits 3b011 = 32 bits 3b100 = 16 byte burst 3b101= 32 byte burst
dmod	Destination Modulo This field has the same meaning as smod. The only difference is that this value controls the destination address.
dsize	Destination Size Specifies the size of the destination data. See the ssize definition.
soff	Source Offset (signed value) As source read is completed, this value is added to the source address as the next address for reading. This value is added to the source address after read is completed.

NOTE

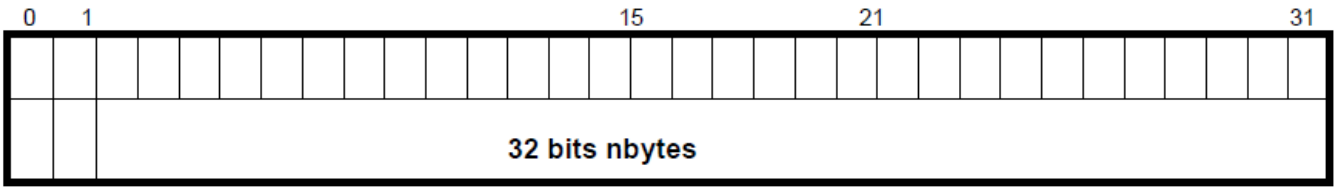
The destination offset (doff) field is placed at the word 5 of the TCD.

2.2.3 Nbytes

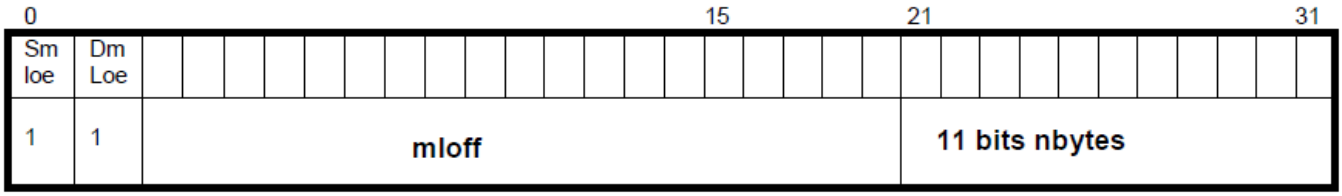
Nbytes is the number of bytes to be transferred in the minor loop. This number must be divisible by the ssize and dsize values, otherwise it may cause configuration error. This word could be set as follows:

- When the EMLM bit in DMACR is 0, that is, when DMACR[EMLM] = 0; the whole word contains the number of the transferred bytes in the Minor loop.

Address: 0xFFFDD008 + 32 x n



- When DMACR[EMLM] =1, that is, when Minor loop mapping is enabled, and smloe or dmloe fields (or both) are set, the second word looks as follows:



The bits 2–21 constitute the Minor Loop Offset (mloff) field. This signed value is added to the source or destination address when the Minor loop ends.

- If smloe is set, the offset is added to the source address after end of the Minor loop.
- If dmloe is set, the offset is added to the destination address. The nbytes field is in bits 21–31 of this word.

2.2.4 Slast

Slast specifies the last source address adjustment value. The adjustment value is added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to refer the next data structure.

2.2.5 Daddr

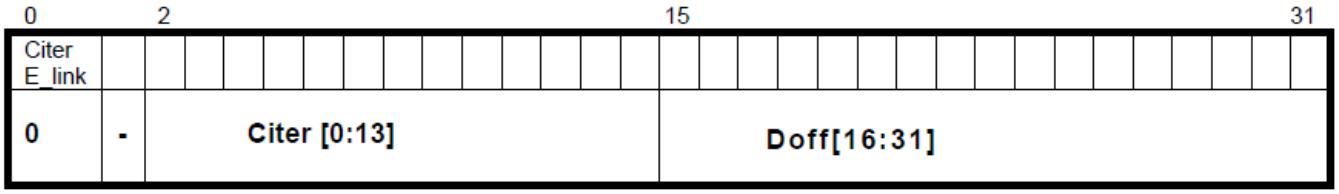
Daddr specifies the destination address, which is the memory address pointing to the destination data.

2.2.6 Current major iteration count (citer), destination offset (doff)

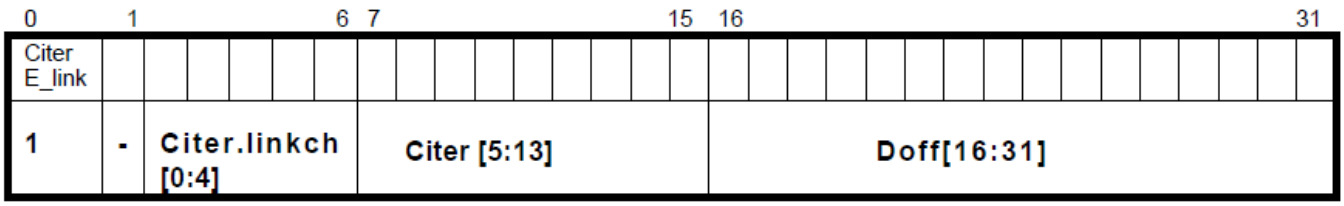
It is possible to set citer into two modifications as follows:

- When citer e_link=0: This means that this DMA channel is not linked with another one on the minor loop level. Then the citer field can be set on the bits 2–13. The doff, or the Destination Signed Offset field value is added to the destination address after the destination write is completed.

Address: 0xFFFDD014 + 32 x n



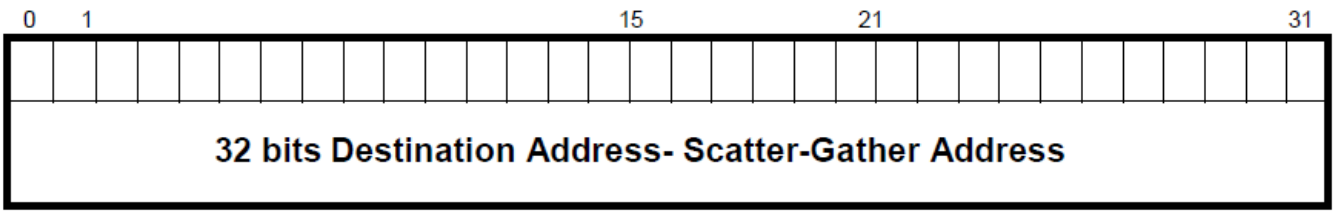
- When citer e_link=1: In this case, the DMA channel is linked with another channel. The number of channels is then written in citer.linkch[0:4] fields. After minor loop of this channel ends, it starts working the channel with the number that is written in this field.



2.2.7 Destination address—Scatter-Gather address

Scatter-Gather (SG) is the last destination address adjustment or the memory address for the next TCD to be loaded into this channel.

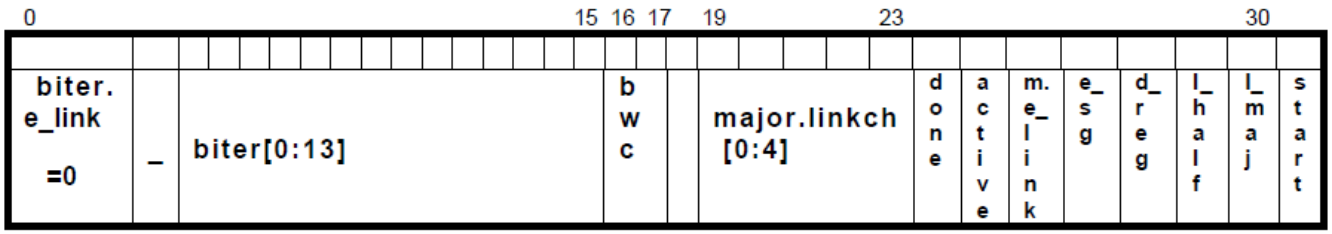
- If scatter-gather processing for the channel is disabled, or e_sg = 0, then adjustment value is added to the destination address at the completion of the outer major iteration count. This value can be applied to restore the destination address to the initial value, or adjust the address to refer the next data structure.
- If scatter-gather processing for the channel is enabled, or e_sg = 1, this address points to the structure in memory that corresponds the TCD data structure. When the first transfer ends, this structure is loaded into TCD and the transfer continues with new parameters. The scatter-gather address must be aligned on the 32 bytes boundary, otherwise a configuration error is reported.



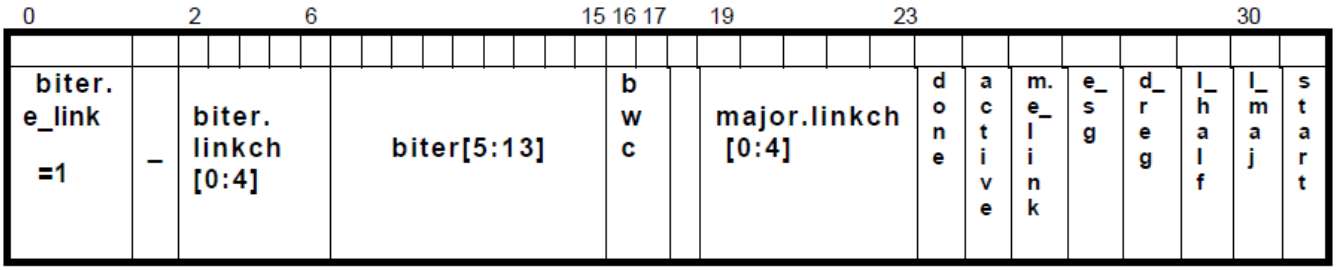
2.2.8 Biter and status/control bits

This word could be set as follows:

- If the biter.e_link field is cleared , then the content of bits 2–15 is biter (beginning 'major' iteration count). The biter field sets the initial number of the major loop counter (citer). Bits 19–23 contain the DMA channel number for major link, when bit m.e_link is asserted.



- If the bit 0 is asserted, then the bits 7–15 constitute the biter field. Bits 2–6 contain the DMA channel number for minor link.



The following table contains a brief description of particular fields.

Table 7. Field descriptions

Field	Description
biter.e_link	Enables channel-to-channel linking on the minor loop level. This is the initial value copied to citer.e_link field when the major loop is completed. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
biter[0:4] or biter.linkch[0:4]	If biter.e_link = 0, then no channel-to-channel linking is performed after the inner Minor loop is exhausted. If biter.e_link = 1, then after Minor loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by biter.linkch[5:0] by setting the Start bit of that channel.
biter[5:13]	Beginning “major” Iteration Count This is the initial value copied into the citer field or citer.linkch field when the Major loop is completed. The citer fields control the iteration count and linking during channel execution.
bwc[0:1]	This field forces the eDMA module to stall after the completion of each read/write access to control the bus request bandwidth seen by the platform’s cross-bar arbitration switch.

Table continues on the next page...

Table 7. Field descriptions (continued)

Field	Description
major.linkch[0:4]	Link Channel Number If major.e_link = 0, then no channel-to-channel linking is performed after the outer “major” loop counter is exhausted. If major.e_link = 1, then after the “major” loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by major.linkch[5:0] by setting the Start bit of that channel.
done	Channel Done This flag indicates the eDMA module has completed the outer major loop. It is set by the eDMA engine as the citer count reaches 0; it is cleared by software, or the hardware when the channel is activated. This bit must be cleared in order to write the major.e_link or e_sg bits.
active	Channel Active This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the eDMA engine as the inner Minor loop completes or if any error condition is detected.
major.e_link	Enables channel-to-channel linking once the major loop is complete. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by major.linkch[5:0]. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
e_sg	Enable Scatter/Gather Processing As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses dlast_sga as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. 0 The current channel’s TCD is in “normal” format. 1 The channel-to-channel linking is enabled. The dlast_sga field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes the execution.
d_req	Disable Request If this flag is set, the eDMA hardware automatically clears the corresponding DMAERQ bit when the current major iteration count reaches 0. 0 The DMAERQ bit of the channel is not affected. 1 The DMAERQ bit of the channel is cleared when the outer major loop is complete.

Table continues on the next page...

Table 7. Field descriptions (continued)

Field	Description
inf_half	Enables an interrupt when major counter is half complete. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
int_maj	Enables an interrupt when major iteration count completes. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
start	Channel Start If this flag is set, the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

2.3 Functional description of the DMA module

It is possible to split the control of the DMA module into three levels:

- Basic transfer
- Minor loop
- Major loop

If the Start signal in the the TCD is asserted in the corresponding channel, then the first Major loop starts. The number of the Major loops is set in the Biter field in the word 7 of the TCD. Initially, the values of the fields biter and citer are equal.

Biter specifies the number of the minor loops to be performed in one data transfer. If this loop needs peripheral request, the DMA channel is waiting for it. After peripheral request is asserted, the Minor loop transfers nbytes of data. The data is transferred over one or more basic transfers. When nbytes is transferred, the citer counter is decreased. To start next Major loop, it is necessary to assert the Start bit again. The transfer is finished when the citer = 0. At this moment, the done bit is asserted in the word 7 and the citer counter is asserted on the value of biter.

The basic function of eDMA is summarized as follows:

1. The DMA channel with right setting of all parameters, waits for the Start control bit to get asserted.
2. The Start bit can be asserted in the program or by other channel (in the case of link). When the Start bit is asserted, the eDMA engine starts the initial Major loop.
3. The Major loop starts the first Minor loop. The Minor loop waits for the peripheral request unless the always requestor is connected.
4. Peripheral request starts the Minor loop which in turn, starts the Basic transfer.
5. Basic transfer reads and writes ssize or dsize bytes from a source address to a destination address.
6. The Minor loop decreases the parameter nbytes. If nbytes = 0, the minor loop ends, else the next Basic transfer starts.
7. Minor loop that is finished decreases the citer counter.
8. If the citer counter = 0, then the Major loop is finished, otherwise the eDMA waits for asserting the next Start bit.

In the following figure, the first column signed w is word in TCD and the first row signed b is the number of bit.

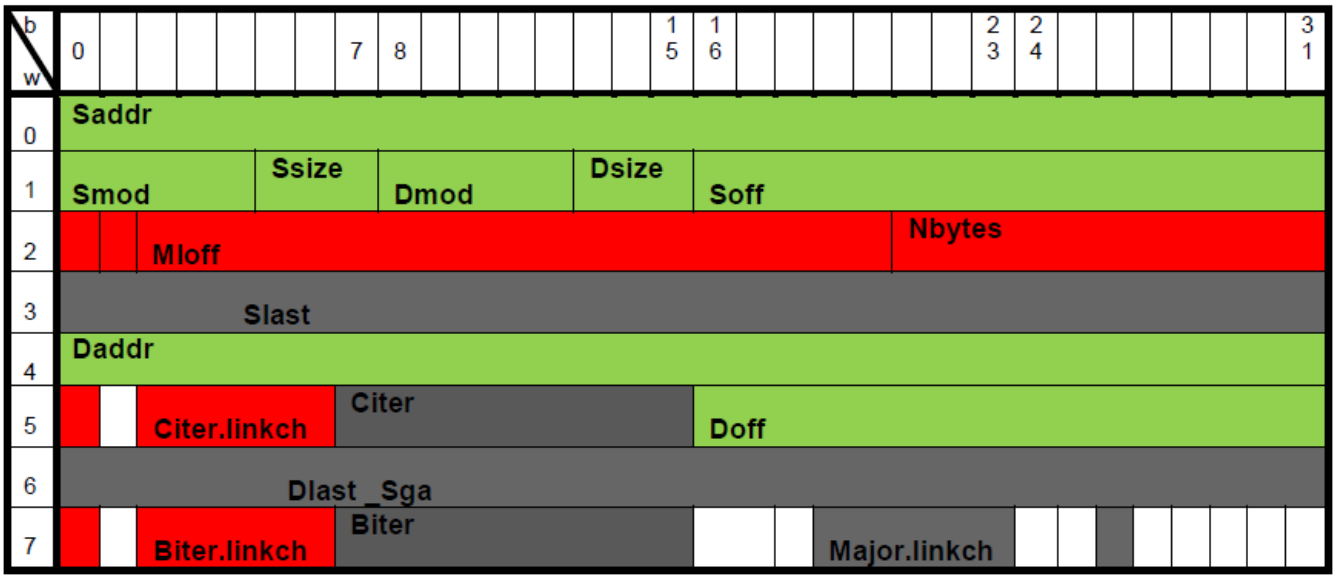


Figure 3. TCD block for functional description

Figure 3 shows the TCD parameters for different levels of the transfer. In Figure 3, the parameters for the basic transfer are marked in green, red color is used for the minor loop and gray for the major loop. The following sections describe the levels of the DMA transfer in detail.

2.3.1 Flowchart of the DMA transfer

The basic function of the DMA module is shown in Figure 4. In the following figure, the Basic transfer is inside block 5, the Minor loop is in blocks 5 and 6 and Major loop goes over the blocks 2–10 and then back to block 2. Parts of this flowchart are used in the next flowcharts for linking with the same numbers of blocks.

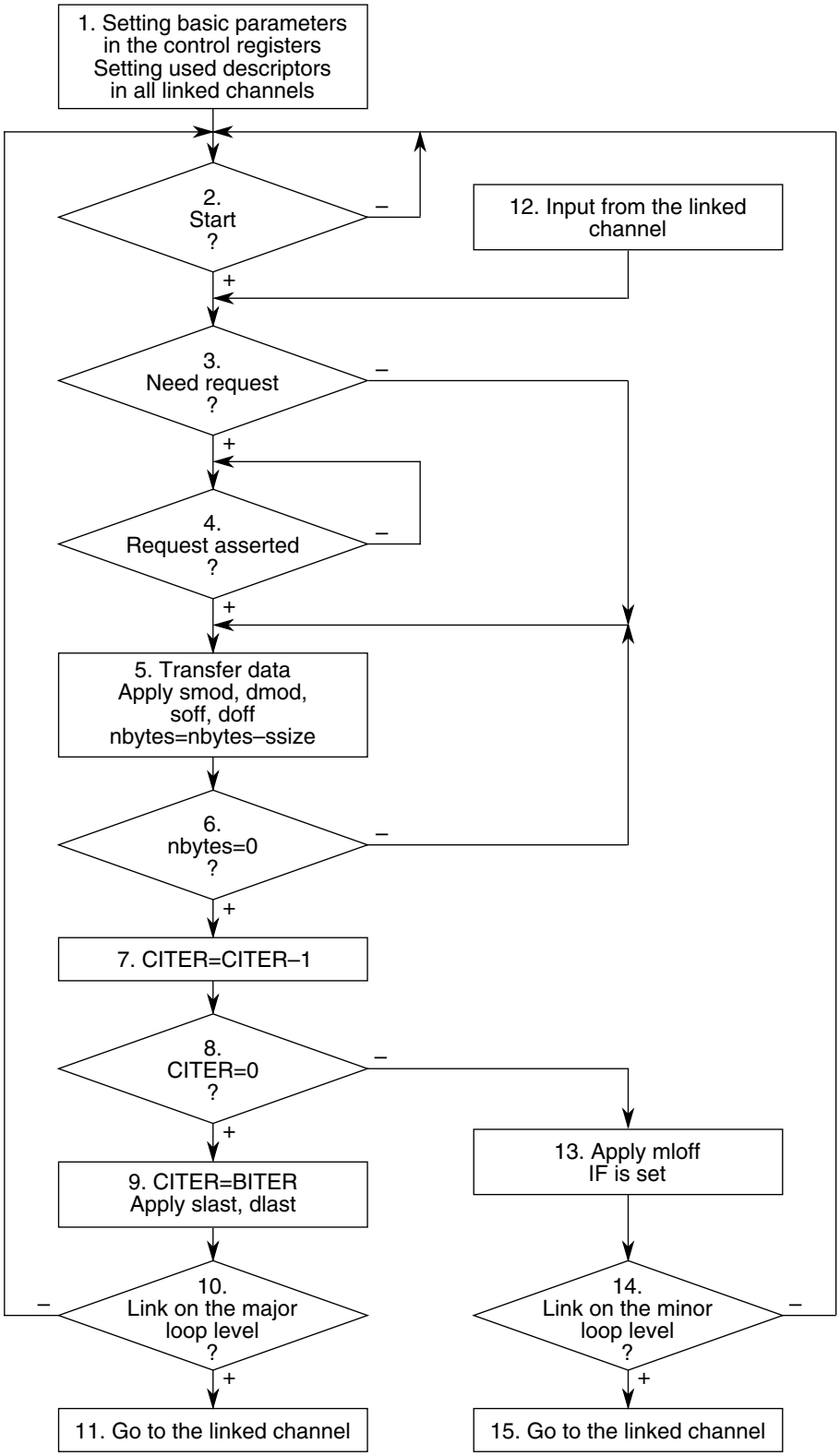


Figure 4. Basic diagram of the DMA control and data flow

2.3.2 The Basic transfer

The Basic transfer is on the lowest level of the DMA transfer. The basic transfer is placed inside the block 5 on the functional diagram shown in Figure 3 and the parameters for Basic transfer in the TCD are marked green. The data is transferred from the source address to the actual destination address. The number of the transmitted bytes is determined by higher value of ssize and dsize. After each read or write, either the source or the destination address is changed. The signed value of soff or doff is added to the actual address. The number of the transferred bytes is then subtracted from parameter nbytes and basic transfer continues until nbytes = 0. Nbytes is a basic parameter of the higher Minor loop level. The fields smod and dmod determine the number of last bits in address that is possible to change on the Basic transfer level. For this purpose, it is necessary to align address to appropriate number of zeros in the lower bits of the address. It is also necessary to set the value of nbytes divisible by ssize and dsize, otherwise the configuration error occurs.

Following are the basic parameters for this transfer:

Table 8. Basic transfer parameters

ssize,dsize	Source/Destination Data Transfer Size: 000 8-bit 001 16-bit 010 32-bit 011 16-byte (32-bit, 4-beat, WRAP4 burst) 101 32-byte (32-bit, 8 beat, WRAP8 burst)
smod / dmod	Source /Destination Address Modulo It is possible to change the number of the lower address bits. 0 means mod is not in function.
soff/doff	Value that is added after read/write to the source/destination address.

Example:

nbytes = 32;

ssize = 1; /*2 bytes*/

dsize = 2; /*4 bytes*/

smod = 3;

dmod = 4;

soff = 2;

doff = 4;

1. After the Start bit is set, the actual transfer address is set with value from saddr.
2. Transfer 2 bytes from the actual source address and again 2 bytes from this source address to 4 bytes on the actual destination address.
3. So, both the actual source and destination addresses increase. Twice the value of soff is added to the actual source address because two reads are required to transfer 4 bytes. The actual destination address is increased by 4.
4. nbytes = nbytes – 4.
5. Repeat steps 2–4 once. Then, the actual source address is limited by smod.
6. Repeat steps 2–6 once. Then, the actual destination address is limited by dmod.
7. Repeat steps 2–8 once.
8. The basic transfer ends when nbytes = 0.

NOTE

It is necessary to align address on the appropriate number of last bits to avoid unexpected results, using smod or dmod.

The following figure demonstrates the above-mentioned example.

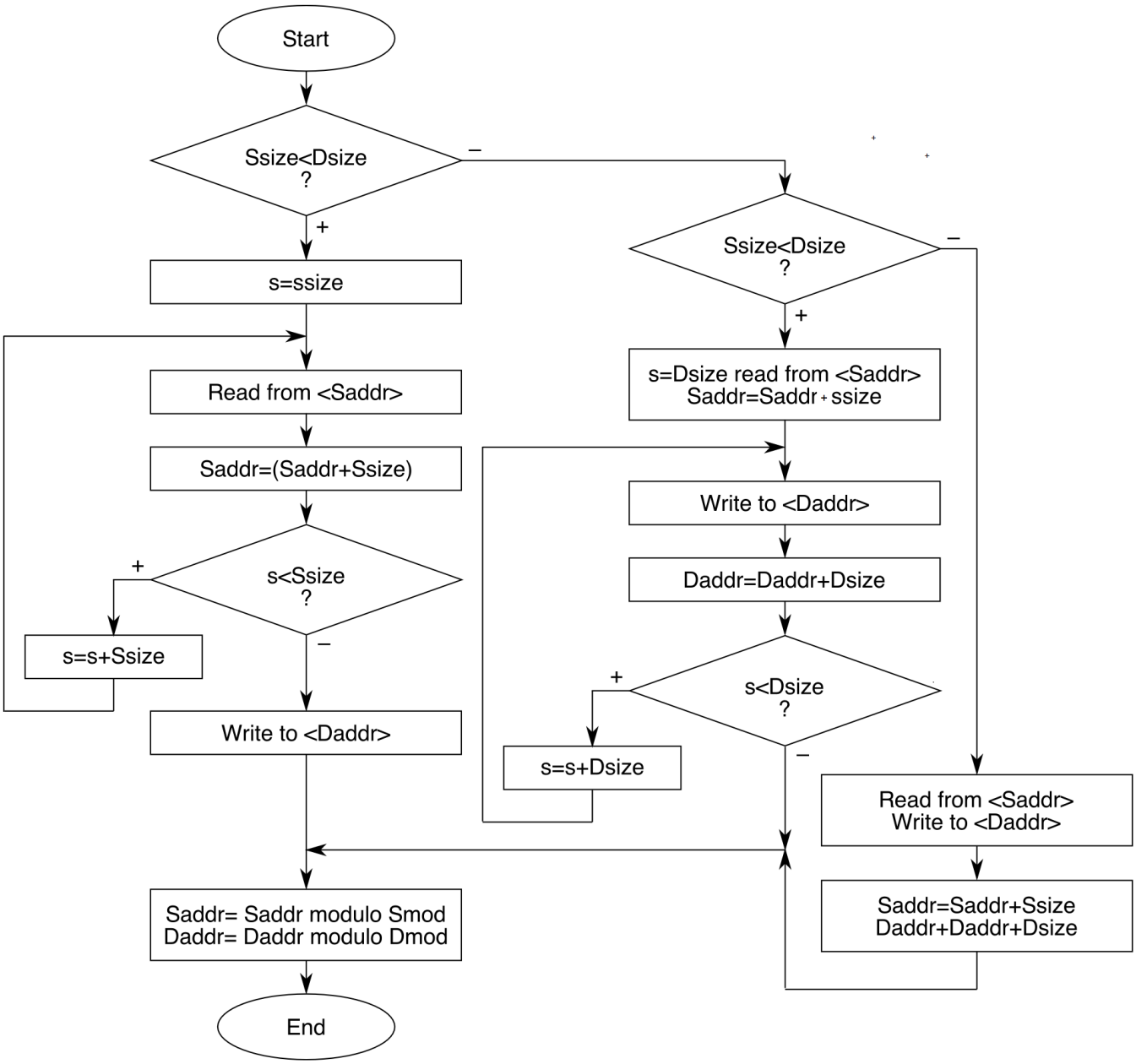


Figure 5. Flowchart of the Basic transfer

Figure 5 corresponds to the block 5 in Figure 4.

2.3.3 The Minor loop

The following table describes the parameters for the minor loop.

Table 9. Minor loop parameters

Field name	Description
smloe (Bit 0 in TCD Word 2)	If this bit is asserted, the value of mloff is added to actual source address at the end of the minor loop.
dmloe (Bit 1 in TCD Word 2)	If this bit is asserted, the value of mloff is added to actual destination address at the end of the minor loop
mloff	Minor loop offset that is added to actual address after minor loop ended
citer e_link (Bit 0 in TCD Word 5)	Enable link on the minor loop level.
citer_linkch	Linked channel
biter e_linkch (Bit 0 in TCD Word 7)	Enable link on the minor loop level.
biter_linkch	Linked channel
nbytes	Number of bytes transmitted in one minor loop.

1. The minor loop starts after the peripheral request is asserted. The 'always requestors' don't need a peripheral request. It must be noted that some of the peripheral requests can be limited by the trigger.
2. After the peripheral request is asserted, the Minor loop starts the basic transfer.
3. At the end of this basic transfer the number of transferred bytes is subtracted from nbytes. The value of nbytes must be divisible by ssize and dsize.
4. When the value of nbytes reaches 0, then the citer counter is decreased. The citer counter is already controlled by the Major loop.

The Minor loop corresponds to the blocks 5 and 6 in Figure 6 and the Minor loop parameters are marked with red color in Figure 3.

2.3.4 The Major loop

1. The Major loop starts by asserting the Start bit, or from any linked channel.
2. After the Start bit is asserted, the Major loop starts the first Minor loop (See The Minor loop).
3. When the Minor loop ends, the DMA decreases the citer counter.
4. When citer = 0, the whole transfer ends or starts the channel that is linked on the major loop level.

The following table depicts the main parameters for the Major loop.

Table 10. Major loop parameters

Field	Description
citer	Actual minor loop count
biter	Basic minor loop count
Slast	The offset to be added to the saddr after the Major loop ends
dlast_Sga	The offset to be added to the daddr after the Major loop ends or the address of the data structure for the Scatter-Gather transfer.
major.e_link	Enable link on the Major loop level
major-linkch	Number of the linked channel

2.3.5 The Minor loop link

The Minor loop link is a technique to transfer relative complex data structures such as Ethernet frames, DSPI FIFO words, and so on. The link suggests that DMA needs more than one channel to transfer one data structure because it is not possible to accomplish data transfer with one set of parameters. Link on the Minor loop is enabled by setting citere_link and bitere_link bits and the number of linked channels can be determined by setting biter_linkch and citer_linkch bits. Both pairs of these parameters must have the same value otherwise it may cause a configuration error. When the bit pairs citere_link and bitere_link, and biter_linkch and citer_linkch are set, and nbytes has expired, the control of the data transfer is given to the second channel. On the Major loop level, it is possible to use link with the same channel to avoid the need to assert the Start signal for each Major loop.

NOTE

Both the channels must have the same values on the biter counters at the start of the transition.

The end of the Major loop finishes the Minor loop link. In case the two channels are linked on the Minor loop level and the citer counter 1 has expired, the Major loop ends the Minor loop link. However on the channel 2, one Minor loop rests. It is possible to avoid this problem by linking Channel 1 with Channel 2 on the Major loop level. (See the flowcharts in Figure 6 and Figure 7). When the data transfer is in progress, it is not possible to make any intervention to stop it until the channel ends the whole transfer.

When the transfer is running, it is not possible to make any correct intervention to stop it until the channel ends. It is possible to set DMACR[CXFR] that forces the Minor loop to stop transfer but in this case, the number of data bytes transferred can't be calculated.

The Minor loop link for two channels is shown in Figure 6. The numbers of the blocks correspond to the general data flowchart.

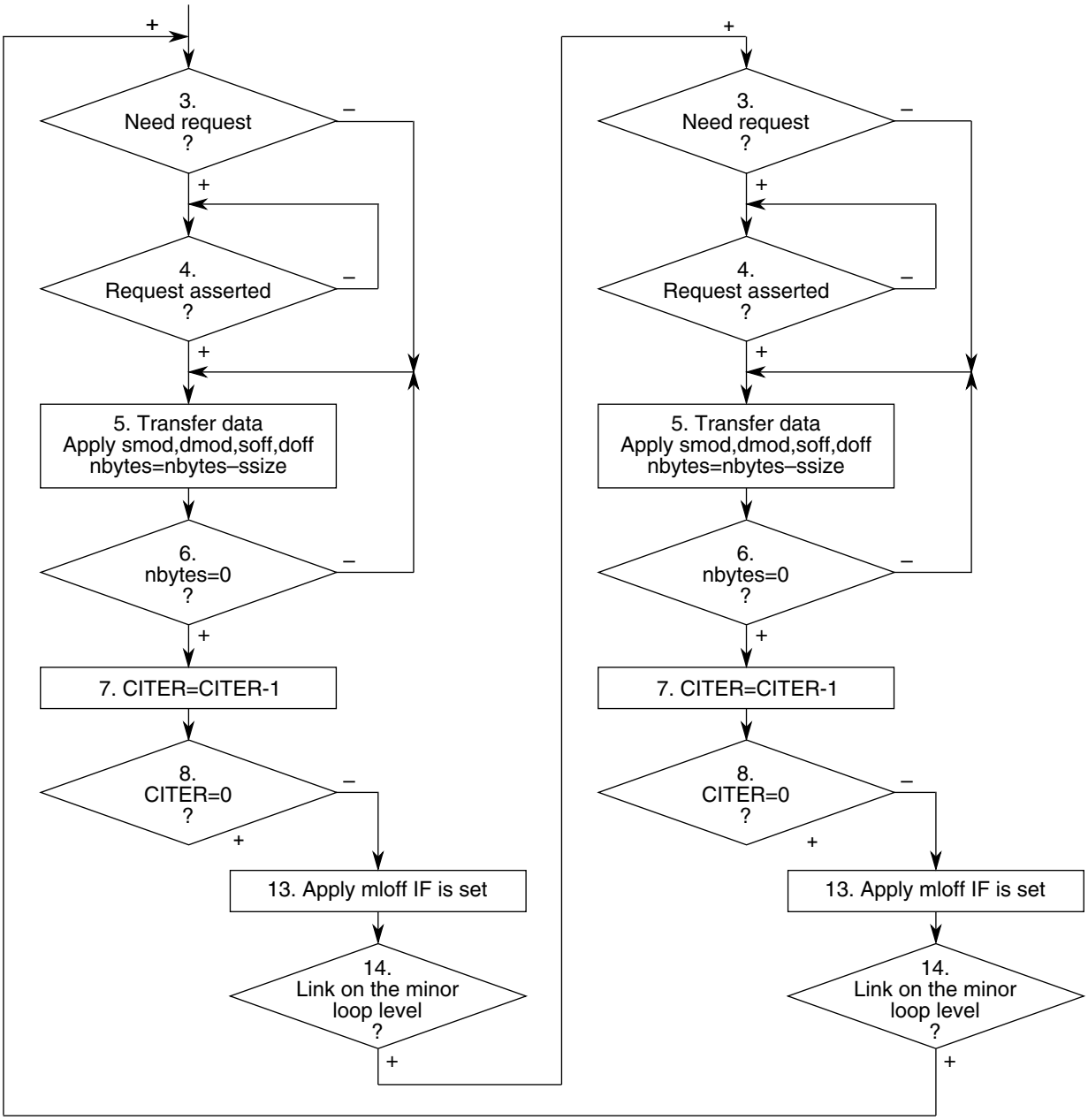


Figure 6. Link two channels on the Minor loop level

2.3.6 Major loop link

The Major loop link is enabled by setting majore_link and major_linkch. All Major loops must be initialized by Start bit in TCD with the exception when a channel is started from other linked channel on this level. The following flowchart shows link of two channels. The numbers of blocks in the following figure correspond to the numbers on the basic DMA transfer in Figure 5.

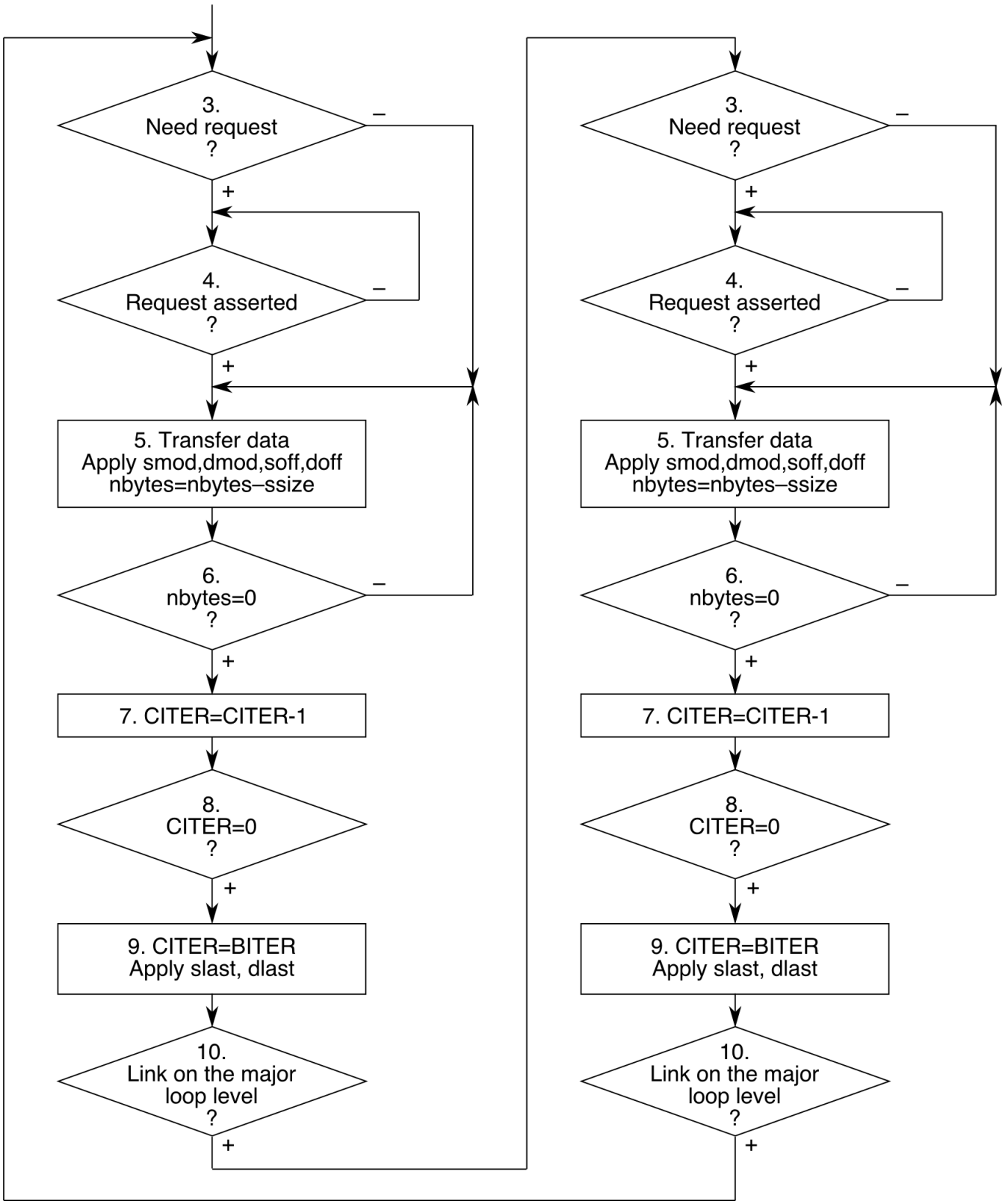


Figure 7. Basic diagram of the DMA control and data flow for link on the Major loop level

2.3.7 Scatter-Gather

It is the most complex method to control a DMA data flow. When e_sg is set, the Word 6 in TCD is set as the Scatter-Gather address (SGA). This address must be aligned on 32-bytes boundary. This address must have a structure that corresponds exactly with TCD. When the Major loop ends on the channel, the TCD is overloaded with the structure saved on the address saved in the dlast_sga field and the DMA engine starts data transfer controlled with the parameters loaded from it. The drawback of this method is that additional time is needed for transfer of the structure of new parameters.

2.4 Conclusion

Following is a summary of important points from the previous sections of this application note.

- The 'always requestors' are GPIO and data memory.
- nbytes must be divisible by ssize and dsize.
- citere_link and bitere_link must have the same value.
- citer_linkch and biter_ linkch in one TCD must have the same value if citere_link and bitere_link are set.
- The last Minor loop by the Minor loop link for two channels needs to be performed with the help of Major loop method.
- It is possible to stop the Minor loop transfer by setting DMACR[CXFR] but it is not possible to determine how many bytes were transferred.
- It is possible to enable Minor loop offset (mloff) only if DMACR[EMLM] is set and at least one out of smloe or dmloe is also set.
- The saddr and daddr bits are "floating“ which means that the values of all offsets and the last address must be set carefully otherwise it causes exception.
- DMACR[HALT] stops the transfer after a channel ends the Major loop transfer.
- Set citer_linkch and biter_linkch to 0 if citere_link and bitere_link are 0, otherwise the citer and biter bits may be set on the wrong value.
- The Major loop can be started by the Start bit in TCD or, from other channel by Major loop link.
- The Scatter-Gather address must be aligned on the 32-byte boundary.
- The transfer data structure to TCD in Scatter-Gather mode causes small delay in data transfer.
- For sending data blocks over infinite loop without intervention of the CPU, use combination of Major and Minor loops link. In this case, use only one Major loop in one channel. See Figure 7. The numbers of blocks in this figure correspond to the numbers on the basic DMA transfer in Figure 5.

The following sections demonstrate the examples of the links on the memory-to-memory transfer that is most instructive and changing of address is not limited.

3 Examples of link to the DMA channels

3.1 Introduction

The following examples show possibilities of using the DMA modules in data transfer between two areas of RAM memory. The following code is used to set the basic parameters.

```
struct tcd_t1
vuint32_t SADDR; /* source address */
vuint16_t SMOD:5; /* source address modulo */
vuint16_t SSIZE:3; /* source transfer size */
vuint16_t DMOD:5; /* destination address modulo */
vuint16_t DSIZE:3; /* destination transfer size */
vint16_t SOFF; /* signed source address offset */
vuint32_t SMLOE:1; /* inner (“minor”) byte count */
```

```
vuint32_t DMLOE:1;
vuint32_t MLOFF:20;
vuint32_t NBYTES:10;
vint32_t SLAST; /* last destination address adjustment, or scatter/gather address (if e_sg =
1) */
vuint32_t DADDR; /* destination address */
vuint16_t CITERE_LINK:1;
vuint16_t CITER_LINKCH:6;
vuint16_t CITER:9;
vint16_t DOFF; /* signed destination address offset */
vint32_t DLAST_SGA;
vuint16_t BITERE_LINK:1; /* beginning ("major") iteration count */
vuint16_t BITER_LINKCH:6;
vuint16_t BITER:9;
vuint16_t BWC:2; /* bandwidth control */
vuint16_t MAJORLINKCH:6; /* enable channel-to-channel link */
vuint16_t DONE:1; /* channel done */
vuint16_t ACTIVE:1; /* channel active */
vuint16_t MAJORE_LINK:1; /* enable channel-to-channel link */
vuint16_t E_SG:1; /* enable scatter/gather descriptor */
vuint16_t D_REQ:1; /* disable ipd_req when done */
vuint16_t INT_HALF:1; /* interrupt on citer = (biter >> 1) */
vuint16_t INT_MAJ:1; /* interrupt on major loop completion */
vuint16_t START:1; /* explicit channel start */
} D1[32]; /* transfer_control_descriptor */
```

For the demonstration of data transfer, two source data array is used.

```
vuint16_t source_data1[80]=
{0x0100,0x0101,0x0102,0x0103,0x0104,0x0105,0x0106,0x0107,
0x0108,0x0109,0x010a,0x010b,0x010c,0x010d,0x010e,0x010f,
0x0110,0x0111,0x0112,0x0113,0x0114,0x0115,0x0116,0x0117,
0x0118,0x0119,0x011a,0x011b,0x011c,0x011d,0x011e,0x011f,
0x0120,0x0121,0x0122,0x0123,0x0124,0x0125,0x0126,0x0127,
0x0128,0x0129,0x012a,0x012b,0x012c,0x012d,0x012e,0x012f,
0x0130,0x0131,0x0132,0x0133,0x0134,0x0135,0x0136,0x0137,
0x0138,0x0139,0x013a,0x013b,0x013c,0x013d,0x013e,0x013f,
0x0140,0x0141,0x0142,0x0143,0x0144,0x0145,0x0146,0x0147,
0x0148,0x0149,0x014a,0x014b,0x014c,0x014d,0x014e,0x014f
};
vuint32_t source_data2[40]=
{0xf100f101,0xf102f103,0xf104f105,0xf106f107,
0xf108f109,0xf10af10b,0xf10cf10d,0xf10ef10f,
0xf110f111,0xf112f113,0xf114f115,0xf116f117,
0xf118f119,0xf11af11b,0xf11cf11d,0xf11ef11f,
0xf120f121,0xf122f123,0xf124f125,0xf126f127,
0xf128f129,0xf12af12b,0xf12cf12d,0xf12ef12f,
0xf130f131,0xf132f133,0xf134f135,0xf136f137,
0xf138f139,0xf13af13b,0xf13cf13d,0xf13ef13f,
0xf140f141,0xf142f143,0xf144f145,0xf146f147,
0xf148f149,0xf14af14b,0xf14cf14d,0xf14ef14f
};
```

and one destination data array with initial values= 0x0000:

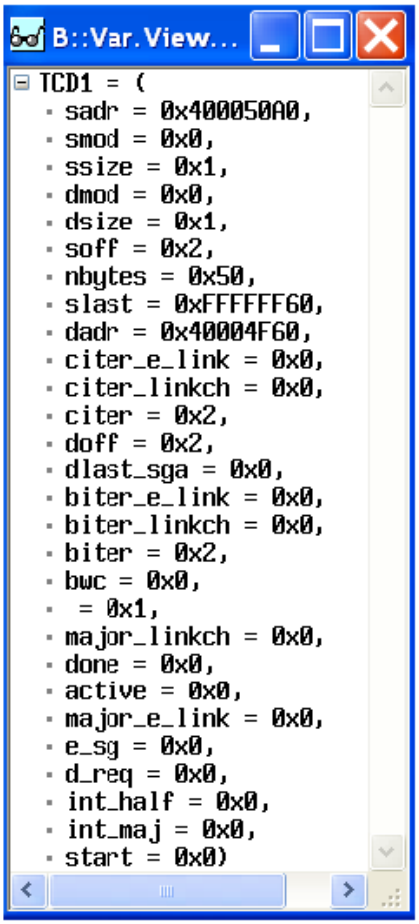
```
__attribute__(( aligned(32) ))

 vuint16_t dest_buffer[160]=
{0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,……


    };
```

3.2 Example 1

This basic example is very simple. DMA translated 0x50 bytes of data from array source_data1 to destination address in two major loops.



Address of source_data1

Ssize 0x1 = 2 bytes

Dsize 2 bytes
Offset increase source address +2 after each reading
Nbytes = 80 number bytes in minor loop
Number to be added to actual address after end of the transfer=160
Address of the dest_buffer

Number of the main loops
Each write increase destination address + 2 bytes
Dlast=0 next transfer will write at destination address + 160

Figure 8. Setting of the TCD1

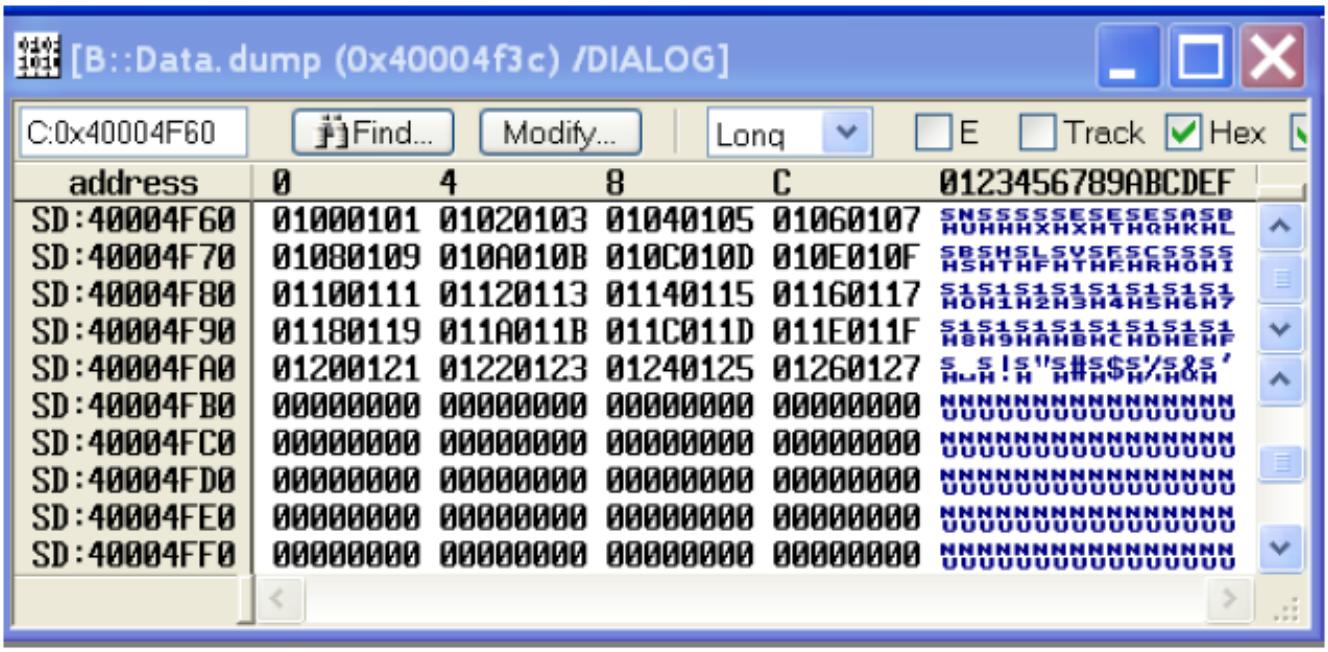
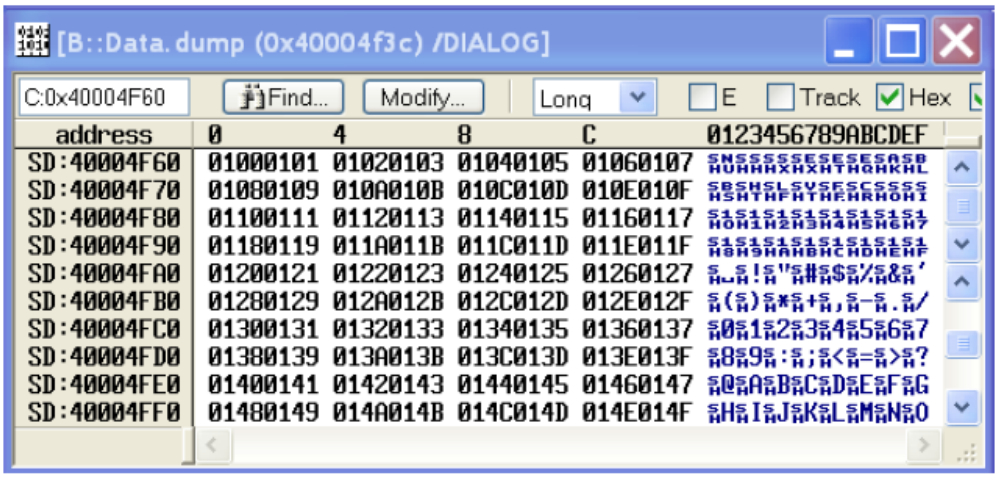


Figure 9. Dest_buffer after first Major loop

After first Minor loop, it is necessary to assert the Start bit in TCD again.

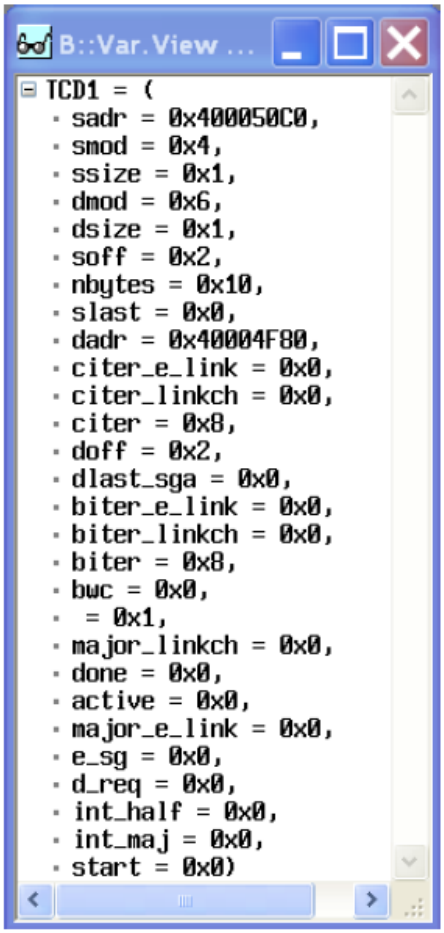


Dest_buffer after second major loop

Figure 10. Dest_buffer after second Major loop

3.3 Example 2: Using the smod and dmod parameters

This example is focused on the use of smod and dmod parameters.



Address of source_data1
Smod=4 it is possible to change only 16 address
Ssize 0x1 = 2 bytes
Dmod=6 it is possible to change only 64 address
Dsize 2 bytes
Offset increase source address +2 after each reading
Nbytes = 16 number bytes in minor loop
Number to be add. to actual source address after end of the transfer=0
Address of the dest_buffer

Number of the major loops=8
Each write increase destination address + 2 bytes

Dlast=0 next transfer will write at destination address + 0

Figure 11. Setting of the TCD1

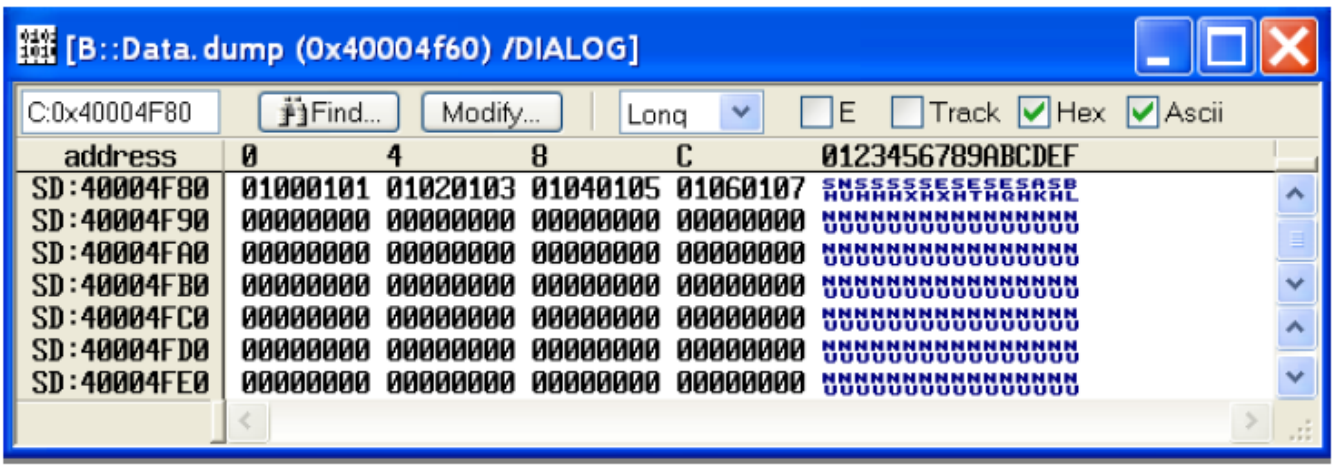


Figure 12. Dest_buffer after first Major loop

The saddr parameter is limited with number 4 which means that it is possible to change only the lower 4 bits of saddr.

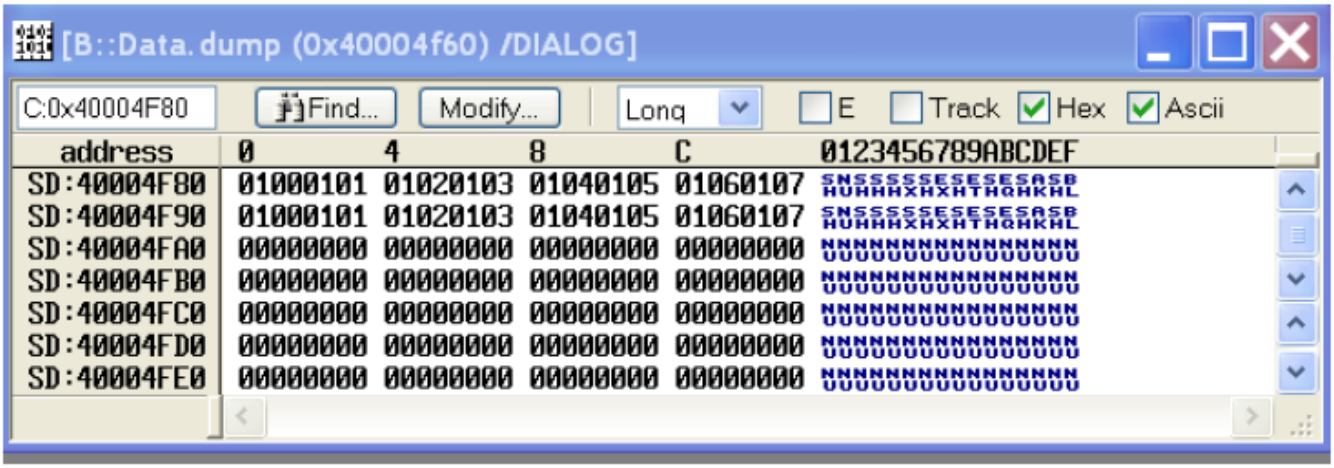


Figure 13. Dest_buffer after second Major loop

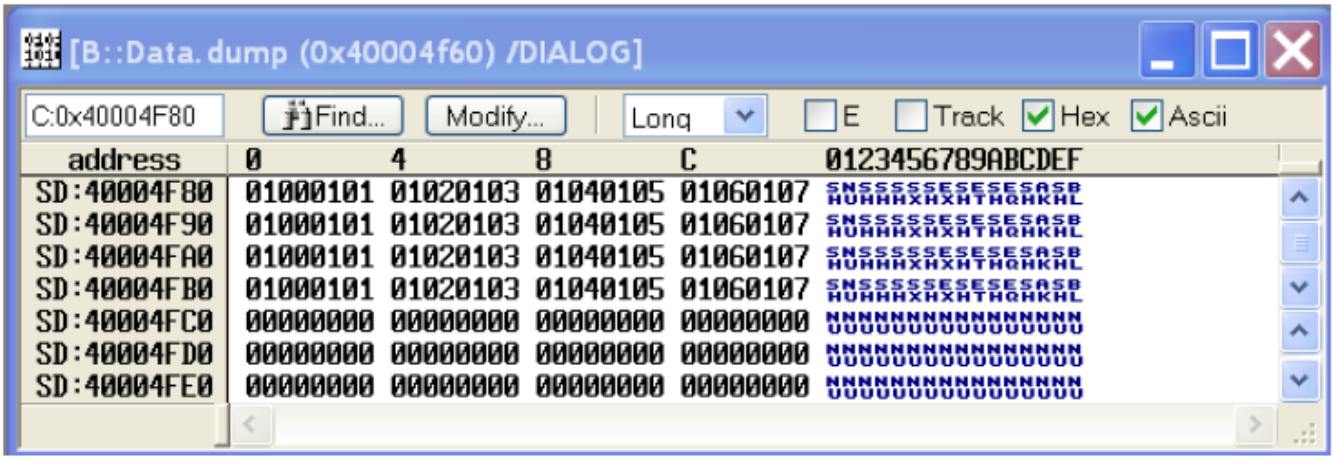


Figure 14. Dest_buffer after fourth Major loop till end of the transfer

After the fourth Major loop, the Dmod parameter comes into function because Daddr is increased by 64. It suggests that the fifth Minor loop starts to write on the same place as the first one, that is, on the address 0x40004F80. The next data transfer from the sixth to eighth Minor loop will be written on the next address. Then the final Dest_buffer will be the same as after the fourth Major loop.

3.4 Example 3: Minor loop link 1

This example links arrays source_data1 and source_data2 on the minor loop level.

NOTE

When the number of major loops on TCD1 is exhausted, then one minor loop rests on TCD2. Therefore, it is necessary to use TCD1 major loop link with TCD2 and then the last minor loop can be finished on TCD2.

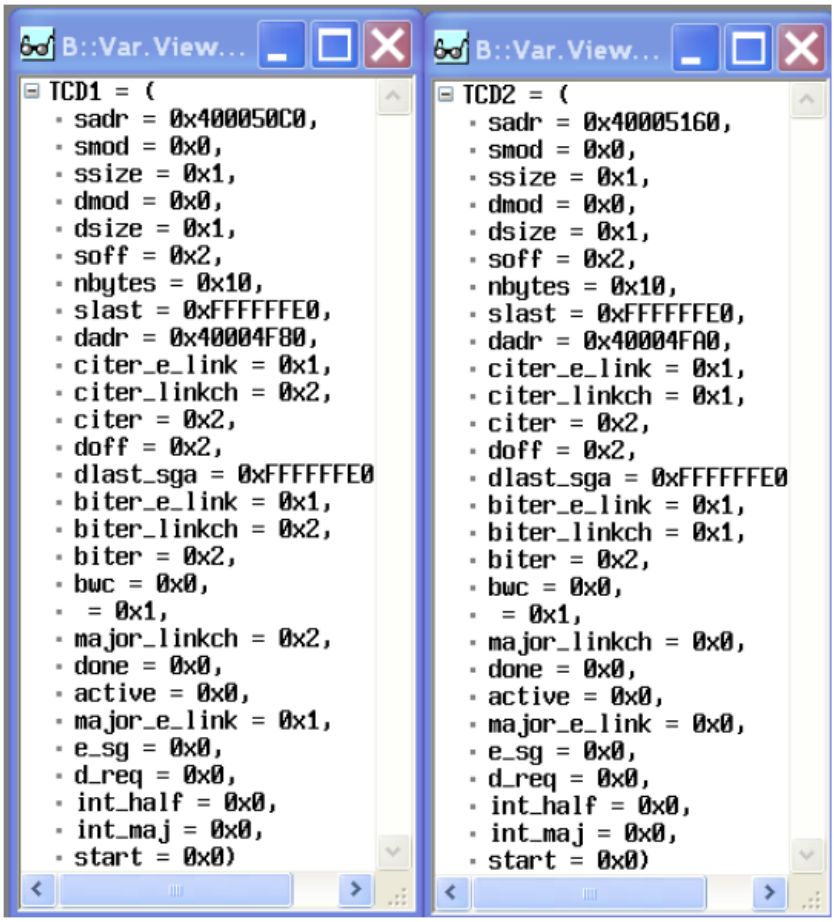


Figure 15. Setting of the TCD1 and TCD2 for Minor loop link 1

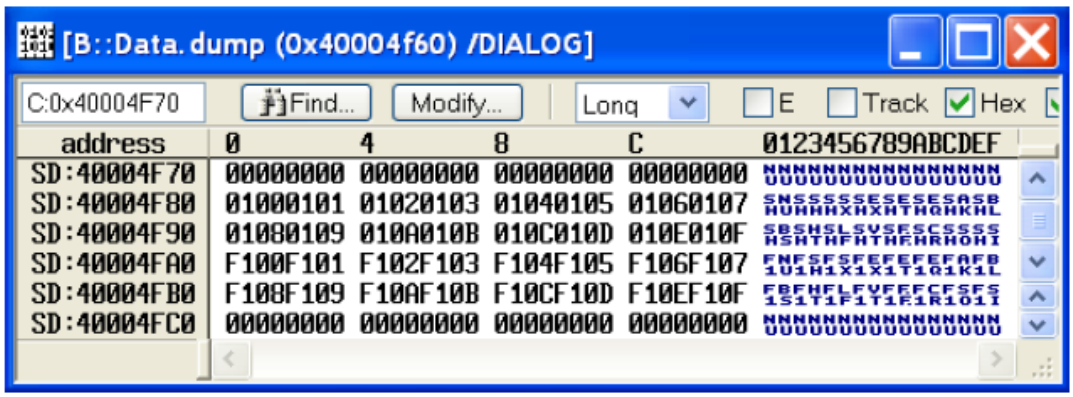
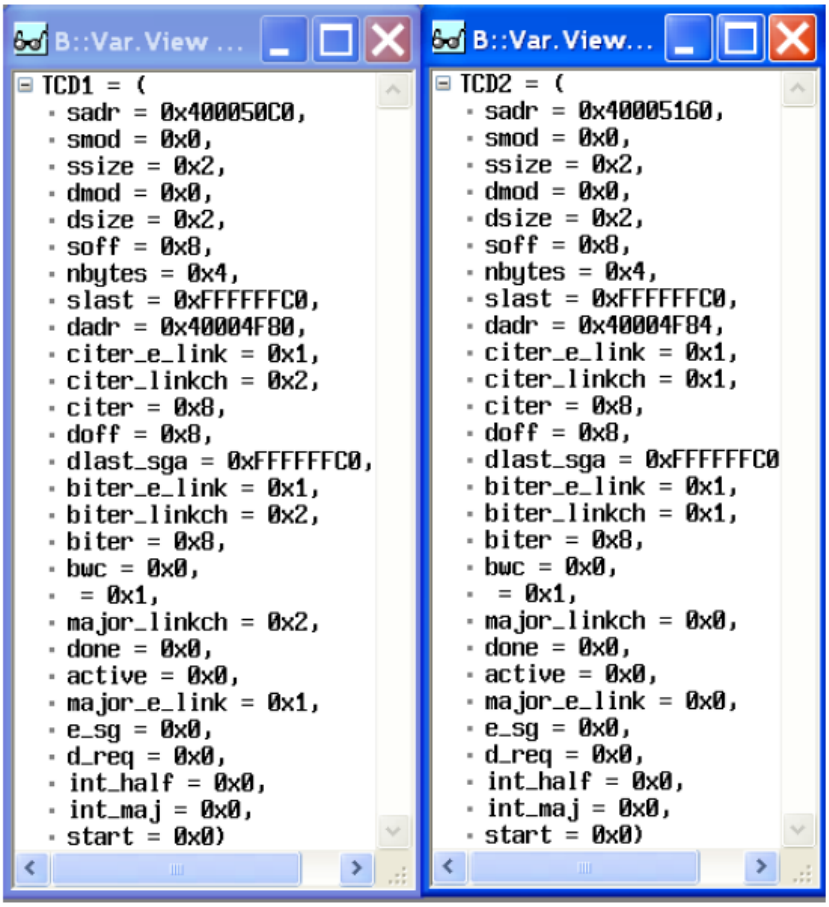


Figure 16. Dest_buffer after end of the transfer

3.5 Example 4: Minor loop link 2

This example is similar to the example given in Example 3: Minor loop link 1, with different offsets and nbytes. It is possible to alternate words from source_data1 and source_data2 in dest_buffer.



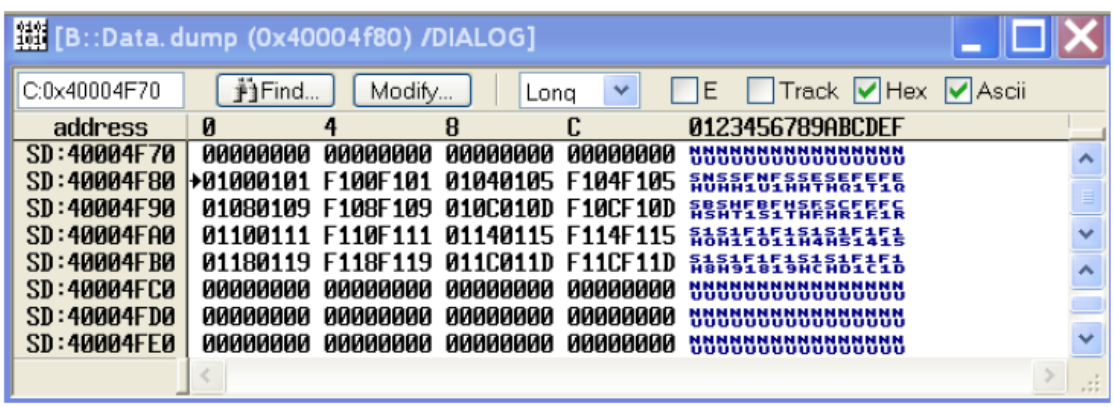
Ssize 0x1 = 2 bytes

Dsize 2 bytes
Offset increase source address +8
Nbytes = 4 number bytes in minor loop
Number to be add. to actual source address after end of the transfer=-64
Address of the dest_buffer

Number of the main loops=8
Each write increase destination addr. + 2
Dlast=-64 next transfer will write at original destination address
Link on the minor lopp allowed
Number of linked channel
Number of major loops=8

TCD1 is linked with TCD2 on the major loop level

Figure 17. Setting of the TCD1 and TCD2 for Minor loop link 2



Final dest_buffer with alternate words

Figure 18. Dest_buffer after end of the transfer

NOTE

For the performance of the last Minor loop, it is necessary to use the Major loop link in TCD1. (major e_link =1 and major_linkch=2)

3.6 Example 5: Major loop link

This basic example shows the use of Major loop link. Minor loop has 16 bytes. Major loop executes four minor loops and then continues TCD2 with similar parameters.

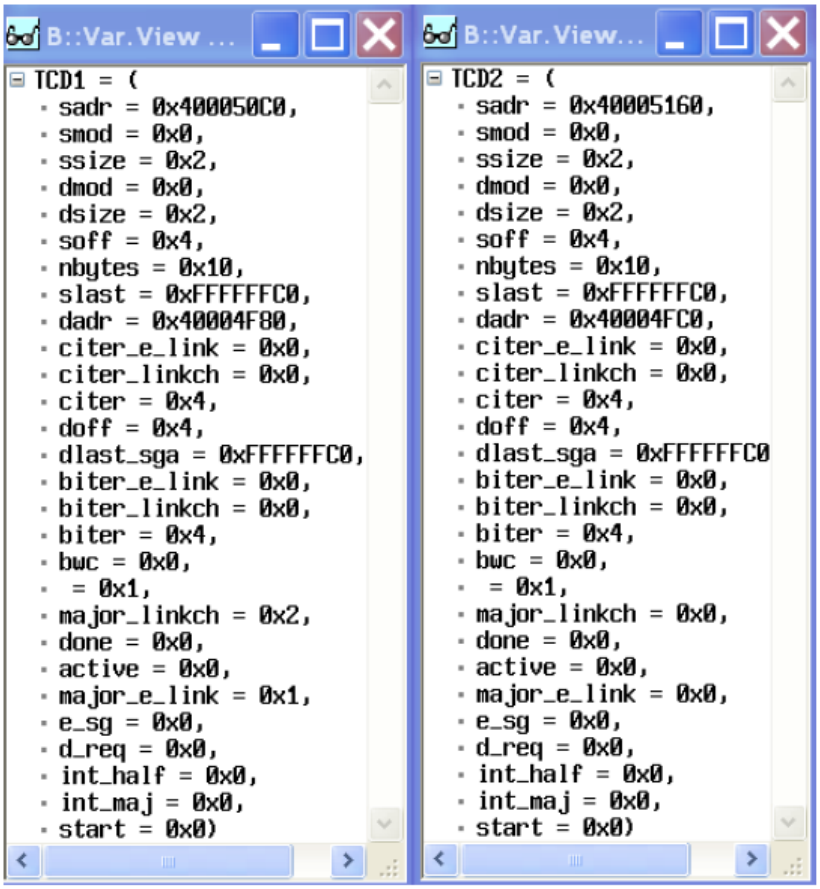


Figure 19. Setting of the TCD1 and TCD2 for Major loop link 2

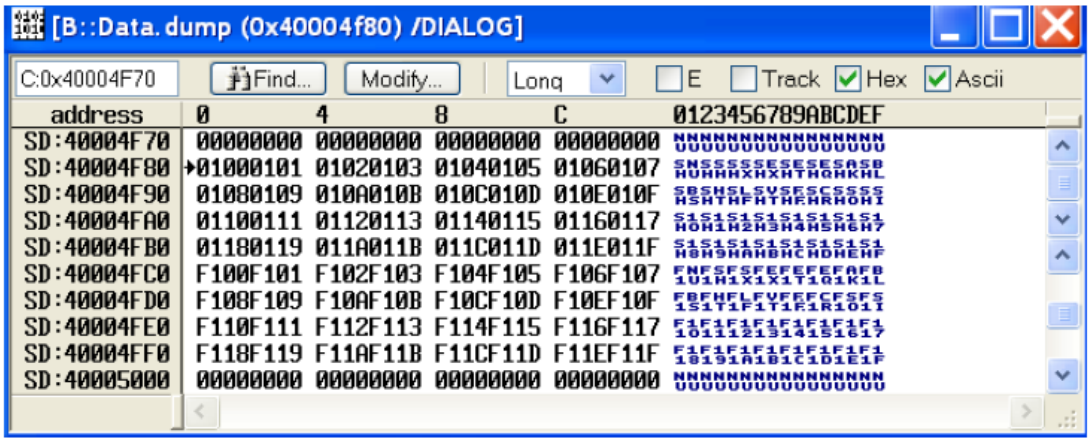
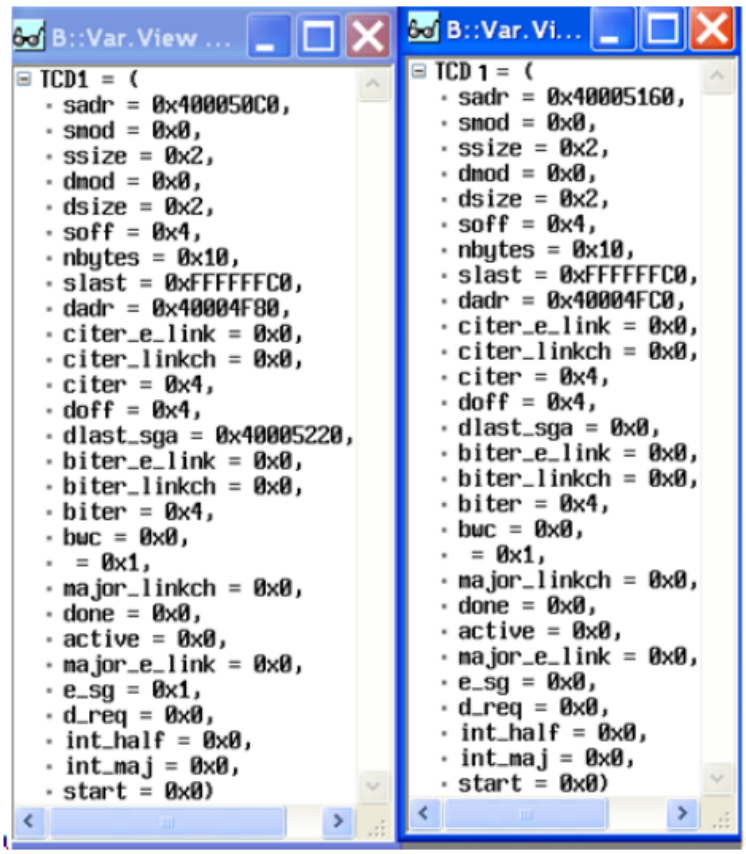


Figure 20. Dest_buffer after the end of transfer

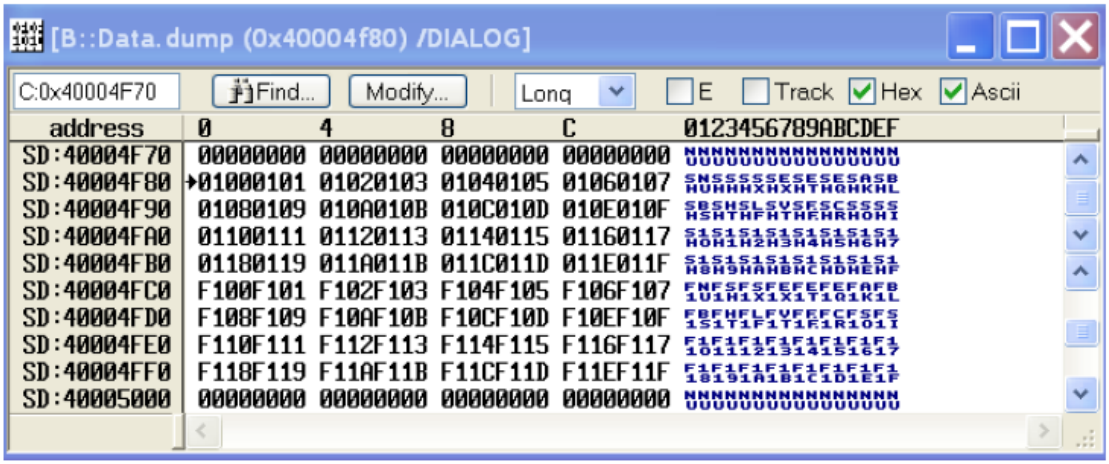
3.7 Example 6: Scatter-Gather



Dadr now points to area in memory where is saved second data structure that has the same structure as TCD but different parameters

E_sg in original TCD is attempted = after transfer will loaded be in the TCD1 the second data structure

Figure 21. Setting of the TCD1 and TCD2 for Major loop link 2



The final structure is the same as by example5, but the loading of the second parameters performed delay

Figure 22. Dest_buffer after end of the scatter / gather transfer

4 Summary

This application note provides specific information for implementing a DMA controller on the Power Architecture MPC567xx processors family. For the other families of microcontrollers, there could be small differences in the performance of the TCD block. Please see the appropriate user or reference manual on http://www.freescale.com

5 Reference

The following reference manual can be found on http://www.freescale.com

- MPC5675KRM : MPC5675K Microcontroller-Reference Manual

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4522
Rev. 1, 6/2012