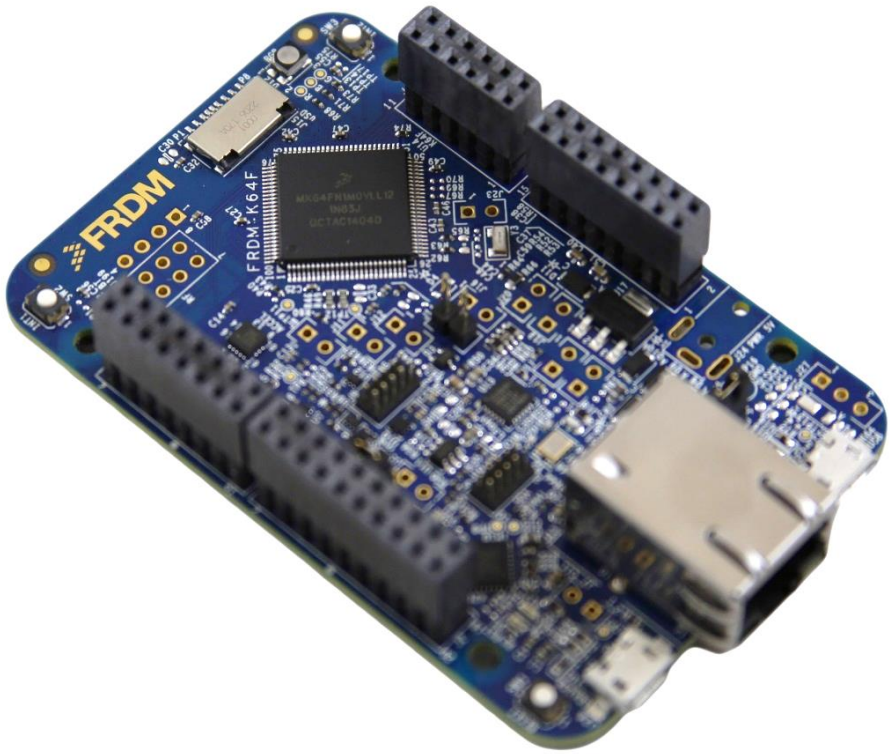
What is and how to configure the eDMA scatter/gather feature

By: Technical Information Center

Introduction

The Enhanced Direct Memory Access (eDMA) is a module capable of performing complex data transfers with minimal intervention from a host processor. This peripheral has a lot of possible configurations and modes of operation that increment the possibilities to automatize the data transfers. This module has a feature called scatter/gather which converts the eDMA in a very customizable peripheral.



Contents

1. About this document

This document explains the scatter/gather feature which is present in the Enhanced Direct Memory Access (eDMA) peripheral. If you need a closer view to this peripheral I widely recommend you to watch the eDMA training video or read the document "Using the DMA module in Kinetis Devices".

Before starting, it is necessary to know what a Transfer Control Descriptor (TCD) is.

2. Transfer Control Descriptor (TCD)

The TCD is a block of 32-bytes of aligned local memory organized to support two-deep (minor and major loops), nested transfer operations. Each channel has its own space in the memory map to store its TCD which is presented as 11 registers of 16 or 32 bits. The TCD is essential because it defines the desired data movement operation and it must be initialized with the appropriate transfer profile prior to activating a channel.

The TCD contains all the information about the data movement (i.e. Source address, source address increment, source transfer size, destination address, destination address increment, destination transfer size, bytes to transfer, number of major loops to execute, etc.).

The following figure shows the TCD structure, if you need a more detailed description about the registers please consult your device's Reference Manual.

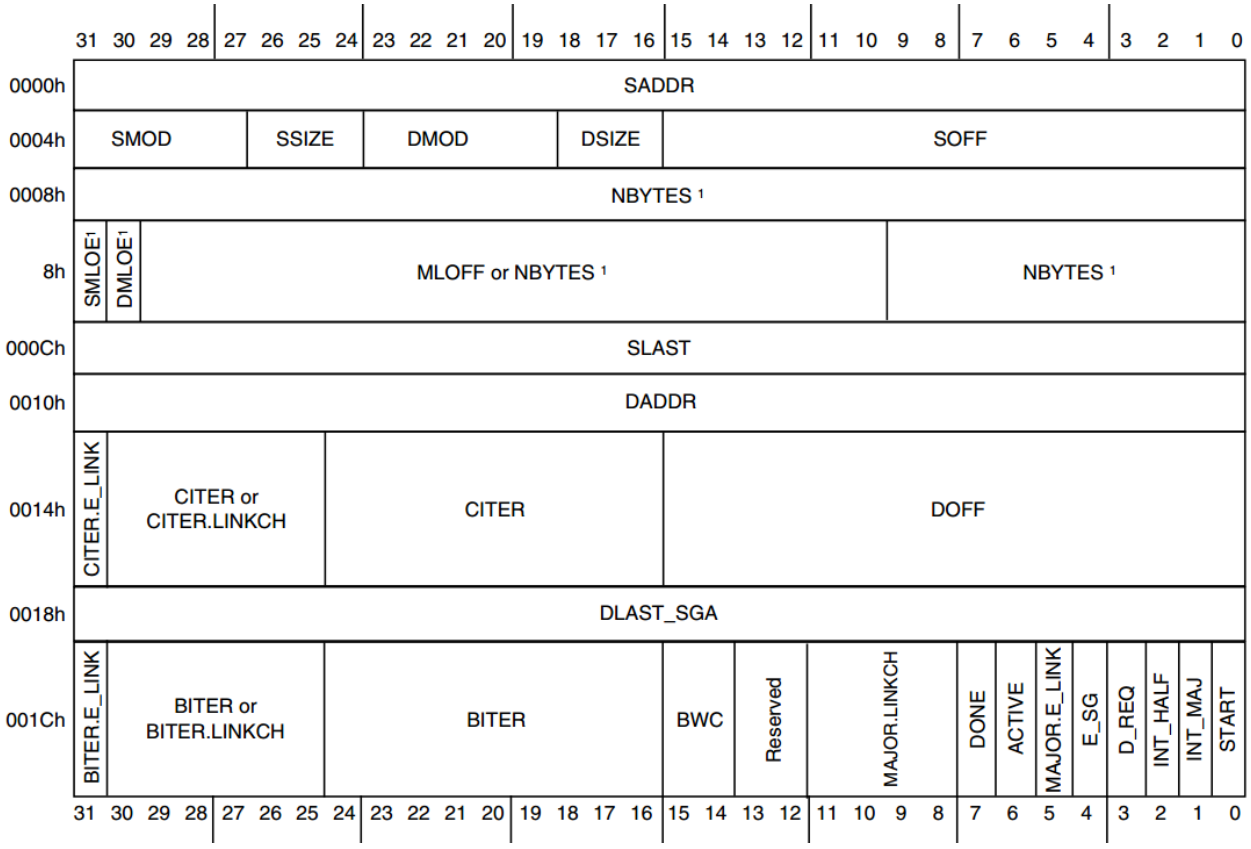


Figure 1. TCD structure.

The table below presents the register details for the TCD.

eDMA Offset	TCDn Register Name	Abbreviation	Width (bits)
0x00	Source Address	TCDn_SADDR	32
0x04	Signed Source Address Offset	TCDn_SOFF	16
0x06	Transfer Attributes	TCDn_ATTR	16
0x08	Minor Byte Count	TCDn_NBYTES	32
0x0C	Last Source Address Adjustment	TCDn_SLAST	32
0x10	Destination Address	TCDn_DADDR	32
0x14	Signed Destination Address Offset	TCDn_DOFF	16
0x16	Current Minor Loop Link, Major Loop Count	TCDn_CITER	16
0x18	Last Destination Address Adjustment / Scatter Gather Address	TDDn_DLAST_SGA	32
0x1C	Control and Status	TCDn_CSR	16
0x1E	Beginning Minor Loop Link, Major Loop Count	TCDn_BITER	16

Table 1. TCD registers details.

TCDs can be processed in a chain so that the eDMA will automatically load the next TCD to be processed, this feature is called scatter/gather.

3. Scatter/gather feature

As it is stated in the Reference Manual, scatter/gather is the process of automatically loading a new TCD into a channel, so it allows an eDMA channel to use multiple TCDs. The name is derived from the case to enable an eDMA channel to **scatter** the eDMA data to multiple destinations or **gather** it from multiple sources.

When scatter/gather is enabled and the channel has finished its major loop, a new TCD is fetched from system memory and loaded into that channel's descriptor location in eDMA programmer's model, thus replacing the current descriptor.

Because the programmer is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the programmer attempts to execute a dynamic scatter/gather operation by enabling the DMA_TCDn_CSR[ESG] (Enable Scatter/Gather) bit at the same time the eDMA engine is retiring the channel. The DMA_TCDn_CSR[ESG] would be set in the programmer's model, but it would be unclear whether the actual scatter/gather request was honored before the channel retired.

The user must clear the DMA_TCDn_CSR[DONE] bit before writing the DMA_TCDn_CSR[MAJORELINK] or DMA_TCDn_CSR[ESG] bits. The DMA_TCDn_CSR[DONE] bit is cleared automatically by the eDMA engine after a channel begins its execution.

4. Use case

It is known that the eDMA peripheral supports circular data queues through the modulo feature (see DMA_TCDn_ATTR[SMOD] and DMA_TCDn_ATTR[DMOD] registers) but it has a huge limitation, the size of the queue must be a power of 2.

What if I need a circular queue with a size different than a power of 2? The scatter/gather feature is the answer to this question (hint: analyze the implementation for the function EDMA_DRV_ConfigLoopTransfer in the Kinetis SDK).

The solution is as simple as having two or more TCDs with the scatter/gather feature enabled. The last TCD should be configured to load the first one when it finishes its execution. It could be useful to have more than two TCDs if you need to be interrupted more frequently (i.e. for monitoring the buffer's occupancy).

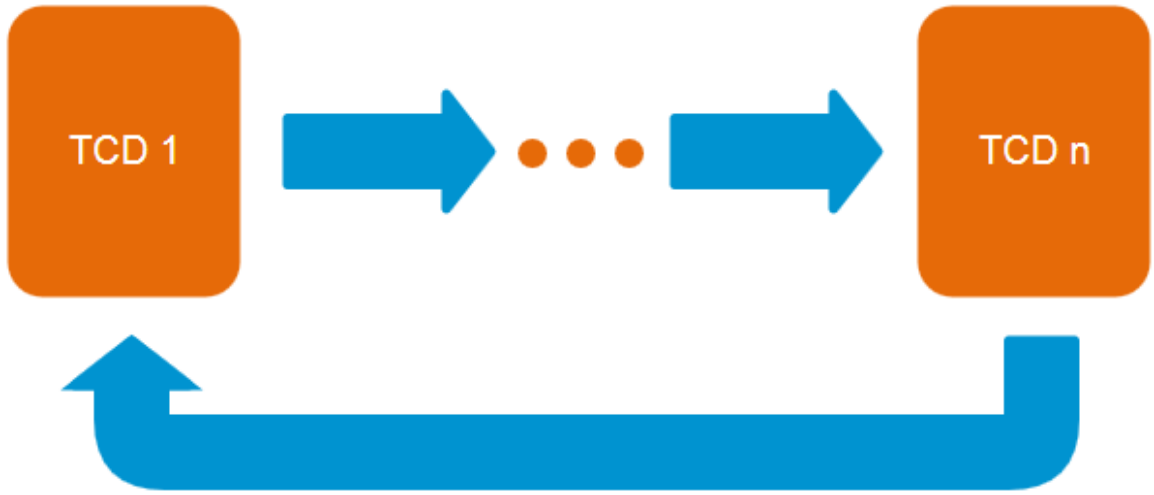


Figure 2. TCDs execution diagram.

5. Software configuration

The configuration to enable the mentioned use case is described in this section. For this example, the FRDM-K64F board is used where the switch SW2 triggers a single movement.

There are two buffers, one for the source and another for the destination. The source buffer content is modified on each iteration to probe that the data is being transferred successfully.

```
uint16_t sourceBuffer[TOTAL_BUFFER_ELEMENTS] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
uint8_t destinationBuffer[TOTAL_BUFFER_ELEMENTS];
```

A structure containing the TCD fields is defined which makes easier the configuration for a single TCD.

```
typedef struct
{
	uint32_t SADDR;
	uint16_t SOFF;
	uint16_t ATTR;
	union
	{
		uint32_t NBYTES_MLNO;
		uint32_t NBYTES_MLOFFNO;
		uint32_t NBYTES_MLOFFYES;
	};
	uint32_t SLAST;
	uint32_t DADDR;
	uint16_t DOFF;
	union
	{
		uint16_t CITER_ELINKNO;
		uint16_t CITER_ELINKYES;
	};
	uint32_t DLASTSGA;
	uint16_t CSR;
	union
	{
		uint16_t BITER_ELINKNO;
		uint16_t BITER_ELINKYES;
	};

}TCD_t;
```

Two TCDs are declared, each one is replaced with the other when its major loop is completed which results in a circular buffer. When a major loop is completed the blue LED is toggled in the eDMA interrupt handler.

```
/***** Configure the eDMA peripheral to work with the Scatter/Gather feature enabled,
working with two TCDs to build a circular buffer, each TCD transfers 5 bytes
(complete buffer is 10 bytes length). *****/
/* Declare the two TCDs that will be used, these must be memory aligned. */
TCD_t tcd[TCDs_AMOUNT] __attribute__ ((aligned (32))); // TCDs_AMOUNT is the amount
of TCDs used in this application.

/* Enable the clock for the eDMA and the DMAMUX. */
SIM_SCGC7 |= SIM_SCGC7_DMA_MASK;
SIM_SCGC6 |= SIM_SCGC6_DMAMUX_MASK;

/* Enable the eDMA channel 0 and set the PORTC as the DMA request source. */
DMAMUX_CHCFG0 = DMAMUX_CHCFG_ENBL_MASK | DMAMUX_CHCFG_SOURCE(51);

/* Enable the interrupts for the channel 0. */
/* Clear all the pending events. */
NVIC_ClearPendingIRQ(DMA0_IRQn);
/* Enable the DMA interrupts. */
NVIC_EnableIRQ(DMA0_IRQn);

/*** Configure the first TCD. ***/
/* Set memory address for source and destination. */
/* TOTAL_BUFFER_ELEMENTS: amount of bytes in the circular buffer, in this case 10. */
/* TCDs_AMOUNT: amount of TCDs to “divide” the circular buffer, in this case 2. */
/* TCD0 is a constant with a value of 0 while TCD1 is a 1. */
tcd[TCD0].SADDR=(uint32_t)(sourceBuffer + TCD0 * TOTAL_BUFFER_ELEMENTS/TCDs_AMOUNT);
tcd[TCD0].DADDR=(uint32_t)(destinationBuffer+TCD0*TOTAL_BUFFER_ELEMENTS/TCDs_AMOUNT);

/* Set an offset for source and destination address. */
tcd[TCD0].SOFF = 0x02; // Since the source buffer is uint16_t, the source address
offset is 2 bytes per transaction.
tcd[TCD0].DOFF = 0x01; // Since the destination buffer is uint8_t, the destination
address offset is 1 byte per transaction.

/* Set source and destination data transfer size is 1 byte. */
tcd[TCD0].ATTR = DMA_ATTR_SSIZE(0) | DMA_ATTR_DSIZE(0);
/* Amount of bytes to be transferred on each channel’s service request (minor loop)*/
tcd[TCD0].NBYTES_MLNO = 0x01;

/* Current major iteration count (5 iteration of 1 byte each one). */
tcd[TCD0].CITER_ELINKNO = DMA_CITER_ELINKNO_CITER(TOTAL_BUFFER_ELEMENTS/TCDs_AMOUNT);
tcd[TCD0].BITER_ELINKNO = DMA_BITER_ELINKNO_BITER(TOTAL_BUFFER_ELEMENTS/TCDs_AMOUNT);

/* Address for the next TCD to be loaded in the scatter/gather mode. */
tcd[TCD0].SLAST = 0;                    // Source address adjustment not used.
tcd[TCD0].DLASTSGA= (uint32_t)&tcd[TCD1];//The tcd[TCD1] is the next TCD to be loaded
```

```
/* Setup control and status register. */
tcd[TCD0].CSR = DMA_CSR_ESG_MASK | DMA_CSR_INTMAJOR_MASK;  //Enable the
scatter/gather feature.

/*** Configure the second TCD. ***/
/* Set memory address for source and destination. */
tcd[TCD1].SADDR=(uint32_t)(sourceBuffer + TCD1 * TOTAL_BUFFER_ELEMENTS/TCDs_AMOUNT);
tcd[TCD1].DADDR=(uint32_t)(destinationBuffer+TCD1*TOTAL_BUFFER_ELEMENTS/TCDs_AMOUNT);

/* Set an offset for source and destination address. */
tcd[TCD1].SOFF = 0x02; // Since the source buffer is uint16_t, the source address
offset is 2 bytes per transaction.
tcd[TCD1].DOFF = 0x01; // Since the destination buffer is uint8_t, the destination
address offset is 1 byte per transaction.

/* Set source and destination data transfer size is 1 byte. */
tcd[TCD1].ATTR = DMA_ATTR_SSIZE(0) | DMA_ATTR_DSIZE(0);
/* Amount of bytes to be transferred on each channel's service request (minor loop)*/
tcd[TCD1].NBYTES_MLNO = 0x01;

/* Current major iteration count (5 iteration of 1 byte each one). */
tcd[TCD1].CITER_ELINKNO = DMA_CITER_ELINKNO_CITER(TOTAL_BUFFER_ELEMENTS/TCDs_AMOUNT);
tcd[TCD1].BITER_ELINKNO = DMA_BITER_ELINKNO_BITER(TOTAL_BUFFER_ELEMENTS/TCDs_AMOUNT);

/* Address for the next TCD to be loaded in the scatter/gather mode. */
tcd[TCD1].SLAST = 0;                    // Source address adjustment not used.
tcd[TCD1].DLASTSGA= (uint32_t)&tcd[TCD0];//The tcd[TCD0] is the next TCD to be loaded

/* Setup control and status register. */
tcd[TCD1].CSR = DMA_CSR_ESG_MASK | DMA_CSR_INTMAJOR_MASK;  //Enable the
scatter/gather feature and the end-of-major loop interrupt.
```

The initial configuration must be the first TCD, the second TCD will be automatically loaded when the first TCD's major loop is finished.

```
/*** Push the first TCD into memory. ***/
DMA_TCD0_SADDR = tcd[TCD0].SADDR;
DMA_TCD0_SOFF = tcd[TCD0].SOFF;
DMA_TCD0_ATTR = tcd[TCD0].ATTR;
DMA_TCD0_NBYTES_MLNO = tcd[TCD0].NBYTES_MLNO;
DMA_TCD0_SLAST = tcd[TCD0].SLAST;
DMA_TCD0_DADDR = tcd[TCD0].DADDR;
DMA_TCD0_DOFF = tcd[TCD0].DOFF;
DMA_TCD0_CITER_ELINKNO = tcd[TCD0].CITER_ELINKNO;
DMA_TCD0_DLASTSGA = tcd[TCD0].DLASTSGA;
DMA_TCD0_CSR = tcd[TCD0].CSR;
DMA_TCD0_BITER_ELINKNO = tcd[TCD0].BITER_ELINKNO;

/* Enable request signal for channel 0. */
DMA_ERQ = DMA_ERQ_ERQ0_MASK;
```

The result is a circular buffer with a length different than a power of two. This example project was written to work with the Kinetis Design Studio (KDS) 3.0.0, the full scatter/gather example project can be downloaded in this link.

6. Conclusion

This document has demonstrated how easy is to configure the scatter/gather feature and how powerful it can be because it enables the possibility of use a single channel with totally different configurations (i.e. different functionalities) without the CPU intervention. This could be useful if the amount of DMA channels is not enough or if a single DMA configuration doesn't satisfy the application's needs (i.e. a huge circular buffer is needed).