Using the DMA module in Kinetis devices

By: Technical Information Center

1 Introduction

Direct memory access (DMA) allows data transfers between main memory and a peripheral device (such as UARTs) without having each word (byte) handled by the CPU. While the data transfers are being carried out the CPU is free to perform other operations as long as these operations do not require any of the buses used by the DMA controller. Thus DMA provides a way to optimize overall system performance, increase data throughput and offload expensive data operations.

The following example addresses basic DMA controller usage within the K2xx family of microcontrollers. The main goal of this example is to introduce the necessary notions of DMA controlled data transfers, in order to do so a description of the main registers is given as well as two implementation examples.

Contents

2 DMA Examples

Within the provided sample code (Appendix) there are two initializations to choose from, the first one performs a single minor loop with a number of bytes to be transferred of 5 and the second one performs 5 minor loops with a number of bytes to be transferred of 1 per minor loop. From now on the first configuration (single minor loop) will be referred as “Example 1” and the second configuration (five minor loops) will be referred as “Example 2”. The following explanation will assume familiarity with the CodeWarrior debug environment and will rely heavily on it.

Both examples features two toggling LEDs for demonstration’s sake, this task won’t be affected by the DMA transfer as the CPU does not take part in the transfer.
Two buffers are created to illustrate the data transfer, g8bSource represents the data source and g8bDestiny the destination buffer. The arrays are initialized as follows:

```
unsigned char g8bSource[10] = {0,1,2,3,4,5,6,7,8,9};
unsigned char g8bDestiny[10] = {1,0,0,0,0,0,0,0,0,0};
```

These variables are allocated in a space of memory by the Linker, for this example no considerations were taken to place the variables in a special memory allocation. If a specific location is desired this must be specified in the linker file.

The destination array is being initialized so that it will be easier to notice the data movement. Both arrays values as well as their addresses can be seen in the following figures. The figures were taken from CodeWarrior’s variables view.

Name	Value	Location
g8bDestiny	0x1fff800c	0x1fff800c
g8bSource	0x1fff8000	0x1fff8000
[0]	0	0x1fff8000
[1]	1	0x1fff8001
[2]	2	0x1fff8002
[3]	3	0x1fff8003
[4]	4	0x1fff8004
[5]	5	0x1fff8005
[6]	6	0x1fff8006
[7]	7	0x1fff8007
[8]	8	0x1fff8008
[9]	9	0x1fff8009

Figure 1-Source array

Name	Value	Location
g8bDestiny	0x1fff800c	0x1fff800c
[0]	1	0x1fff800c
[1]	0	0x1fff800d
[2]	0	0x1fff800e
[3]	0	0x1fff800f
[4]	0	0x1fff8010
[5]	0	0x1fff8011
[6]	0	0x1fff8012
[7]	0	0x1fff8013
[8]	0	0x1fff8014
[9]	0	0x1fff8015

Figure 2-Destination array

2.1 Example 1

By choosing `DMA_ONE_TRANSACTION` as the initialization parameter in main.h, init_DMA_1Trans will be called and the registers initialization for the single minor loop example will proceed.
The registers initialization can be seen in the following figure.

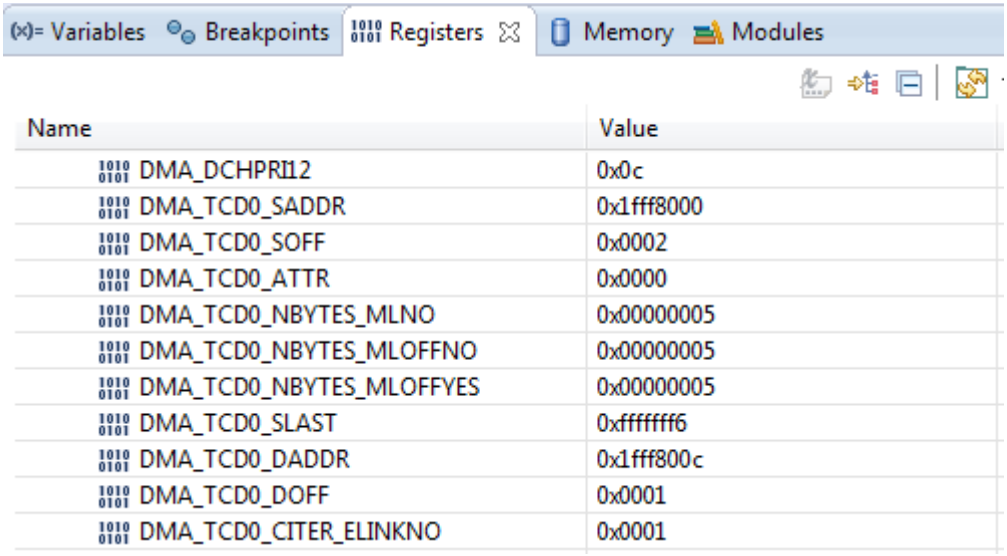

Figure 3-DMA channel 0 registers

As can be seen in the figure the source address (SADRR) and destination address (DADRR) matches those of the source and destination buffers, the number of bytes to be transferred is 5 (NBYTES), there is a source offset address of 2 bits therefore the values that will be read in the array are 0, 2, 4, 6 and 8, this values will be written with an offset of one byte and only one minor loop will be executed per major loop (CITER = 1).

Once the peripheral request has been activated (SW1 in Port C in this case) the transfer process will begin, the source address will be read and its contents loaded to the destination address until NBYTES equals 0, CITER will be decremented and if it equals 0 the major loop will terminate, CITER will be updated with the contents of BITER, address adjustments will be performed (both addresses back to their initial values) and a flag in the

control and status register will be issued to indicate the completion of the major loop. This process is illustrated in the following figures.

g8bDestiny	0x1fff800c	0x1fff800c
[0]	0	0x1fff800c
[1]	2	0x1fff800d
[2]	4	0x1fff800e
[3]	6	0x1fff800f
[4]	8	0x1fff8010
[5]	0	0x1fff8011
[6]	0	0x1fff8012
[7]	0	0x1fff8013
[8]	0	0x1fff8014
[9]	0	0x1fff8015

Figure 4-Destination address after the transfer has been completed

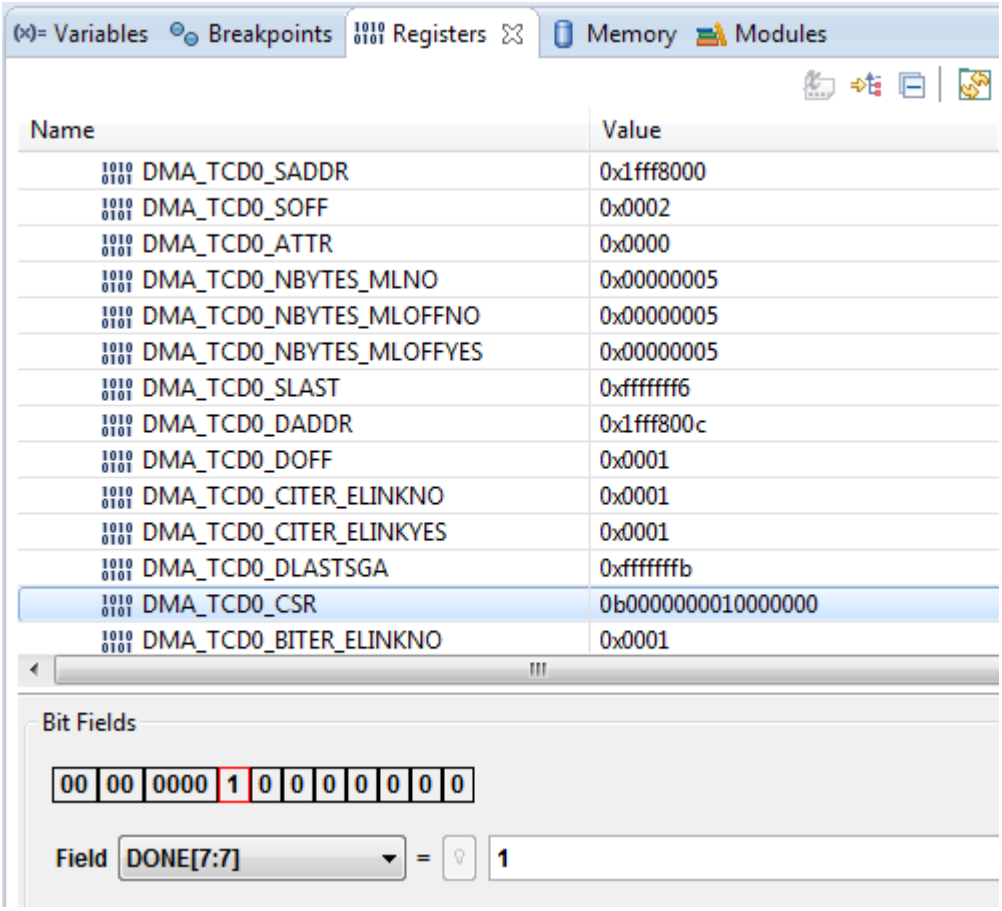

Figure 5-Registers status after the transfer has been completed

By altering certain parameters like address adjustments or offsets the way in which the data will be transferred will change. For instance if the destination address adjustment (DLASTSGA) happened to be zero, once the major loop has finished the destination address will remain unaltered and the next peripheral request will write the following 5 bytes where the previous write left it. This is only for demonstration's sake, because if the process continues it might reach a space in memory that is not allocated for data storage and some data corruptions might occur.

g8bDestiny	0x1fff800c	0x1fff800c
[0]	0	0x1fff800c
[1]	2	0x1fff800d
[2]	4	0x1fff800e
[3]	6	0x1fff800f
[4]	8	0x1fff8010
[5]	0	0x1fff8011
[6]	2	0x1fff8012
[7]	4	0x1fff8013
[8]	6	0x1fff8014
[9]	8	0x1fff8015

Figure 6-Example 1 with DLASTSGA = 0

2.2 Example 2

By choosing `DMA_ONE_ELEMENT` as the initialization parameter in main.h, init_DMA_1Elem will be called and the registers initialization for the single byte 5 minor loops example will proceed.
The registers initialization can be seen in the following figure.

Name	Value
DMA_TCD0_SADDR	0x1fff8000
DMA_TCD0_SOFF	0x0002
DMA_TCD0_ATTR	0x0000
DMA_TCD0_NBYTES_MLNO	0x00000001
DMA_TCD0_NBYTES_MLOFFNO	0x00000001
DMA_TCD0_NBYTES_MLOFFYES	0x00000001
DMA_TCD0_SLAST	0xfffffff6
DMA_TCD0_DADDR	0x1fff800c
DMA_TCD0_DOFF	0x0001
DMA_TCD0_CITER_ELINKNO	0x0005
DMA_TCD0_CITER_ELINKYES	0x0005
DMA_TCD0_DLASTSGA	0xfffffffb
DMA_TCD0_CSR	0b0000000000000000
DMA_TCD0_BITER_ELINKNO	0x0005

Figure 1-DMA channel 0 registers

As can be seen in the figure the source address (SADRR) and destination address (DADRR) matches those of the source and destination arrays, the number of bytes to be transferred per minor loop is 1 (NBYTES), there is a source offset address of 2 bits therefore the values that will be read in the array are 0, 2, 4, 6 and 8, this values will be written with an offset of one bit and only one minor loop will be executed per major loop (CITER = 1).

Once the peripheral request has been activated (SW1 in Port C in this case) the transfer process will begin, the source address will be read and its contents loaded to the destination address until NBYTES equals 0 since NBYTES equals 1 just one minor loop will be executed and CITER will be decreased as can be seen in figure 30 at the end of the first minor loop (transfer) CITER now holds the value 4, both source- and destination- address had been updated with their corresponding offsets.

g8bDestiny	0x1fff800c	0x1fff800c
[0]	0	0x1fff800c
[1]	0	0x1fff800d
[2]	0	0x1fff800e
[3]	0	0x1fff800f
[4]	0	0x1fff8010
[5]	0	0x1fff8011
[6]	0	0x1fff8012
[7]	0	0x1fff8013
[8]	0	0x1fff8014
[9]	0	0x1fff8015

Figure 7-First minor loop

Name	Value
DMA_TCD0_SADDR	0x1fff8002
DMA_TCD0_SOFF	0x0002
DMA_TCD0_ATTR	0x0000
DMA_TCD0_NBYTES_MLNO	0x00000001
DMA_TCD0_NBYTES_MLOFFNO	0x00000001
DMA_TCD0_NBYTES_MLOFFYES	0x00000001
DMA_TCD0_SLAST	0xfffffff6
DMA_TCD0_DADDR	0x1fff800d
DMA_TCD0_DOFF	0x0001
DMA_TCD0_CITER_ELINKNO	0x0004
DMA_TCD0_CITER_ELINKYES	0x0004
DMA_TCD0_DLASTSGA	0xfffffffb
DMA_TCD0_CSR	0b0000000000000000
DMA_TCD0_BITER_ELINKNO	0x0005

Figure 2-Register status after first minor loop

The process continues for the next 3 elements before CITER reaches 0, as can be seen in the following figures.

g8bDestiny	0x1fff800c	0x1fff800c
[0]	0	0x1fff800c
[1]	2	0x1fff800d
[2]	4	0x1fff800e
[3]	6	0x1fff800f
[4]	0	0x1fff8010
[5]	0	0x1fff8011
[6]	0	0x1fff8012
[7]	0	0x1fff8013
[8]	0	0x1fff8014
[9]	0	0x1fff8015

Figure 8-Fourth minor loop

Name	Value
DMA_TCD0_SADDR	0x1fff8008
DMA_TCD0_SOFF	0x0002
DMA_TCD0_ATTR	0x0000
DMA_TCD0_NBYTES_MLNO	0x00000001
DMA_TCD0_NBYTES_MLOFFNO	0x00000001
DMA_TCD0_NBYTES_MLOFFYES	0x00000001
DMA_TCD0_SLAST	0xfffffff6
DMA_TCD0_DADDR	0x1fff8010
DMA_TCD0_DOFF	0x0001
DMA_TCD0_CITER_ELINKNO	0x0001
DMA_TCD0_CITER_ELINKYES	0x0001
DMA_TCD0_DLASTSGA	0xfffffffb
DMA_TCD0_CSR	0b0000000000000000
DMA_TCD0_BITER_ELINKNO	0x0005

Figure 9-Register status after the fourth minor loop

For the last minor loop CITER finally reaches zero, both source- and destination-address have been adjusted for the following major loop, CITER has been updated with the value in BITER and the done flag in the control and status register has been activated. This process can be verified in the following figures.

g8bDestiny	0x1fff800c	0x1fff800c
[0]	0	0x1fff800c
[1]	2	0x1fff800d
[2]	4	0x1fff800e
[3]	6	0x1fff800f
[4]	8	0x1fff8010
[5]	0	0x1fff8011
[6]	0	0x1fff8012
[7]	0	0x1fff8013
[8]	0	0x1fff8014
[9]	0	0x1fff8015

Figure 10-Last minor loop

Name	Value
DMA_TCD0_SADDR	0x1fff8000
DMA_TCD0_SOFF	0x0002
DMA_TCD0_ATTR	0x0000
DMA_TCD0_NBYTES_MLNO	0x00000001
DMA_TCD0_NBYTES_MLOFFNO	0x00000001
DMA_TCD0_NBYTES_MLOFFYES	0x00000001
DMA_TCD0_SLAST	0xfffffff6
DMA_TCD0_DADDR	0x1fff800c
DMA_TCD0_DOFF	0x0001
DMA_TCD0_CITER_ELINKNO	0x0005
DMA_TCD0_CITER_ELINKYES	0x0005
DMA_TCD0_DLASTSGA	0xfffffffb
DMA_TCD0_CSR	0b0000000010000000
DMA_TCD0_BITER_ELINKNO	0x0005

Figure 11- Register status after last minor loop

If the data transfer is desired to be in a different order, for instance 8, 6, 4, 2 and 0 instead of 0, 2, 4, 6 and 8, this could be achieved only by modifying the source address in order to start at the address of the desired byte (8 in this case) and modifying the source address offset sign, so that it diminishes the source address by 2 bits per basic transfer.

g8bSource	0x1fff8000	0x1fff8000
[0]	0	0x1fff8000
[1]	1	0x1fff8001
[2]	2	0x1fff8002
[3]	3	0x1fff8003
[4]	4	0x1fff8004
[5]	5	0x1fff8005
[6]	6	0x1fff8006
[7]	7	0x1fff8007
[8]	8	0x1fff8008
[9]	9	0x1fff8009

Figure 12-Source array

To achieve this the source-address must match the address of number 8 in the array this can be done by adding 8 to the source address that is being currently used. The source offset (SOFF) register must be set to -2 bits and the source-address adjustment is set to 10 bits in order to reestablish the initial value of the source address. This can be verified in the following figure.

Name	Value
DMA_TCD0_SADDR	0x1fff8008
DMA_TCD0_SOFF	0xfffe
DMA_TCD0_ATTR	0x0000
DMA_TCD0_NBYTES_MLNO	0x00000001
DMA_TCD0_NBYTES_MLOFFNO	0x00000001
DMA_TCD0_NBYTES_MLOFFYES	0x00000001
DMA_TCD0_SLAST	0x0000000a
DMA_TCD0_DADDR	0x1fff800c
DMA_TCD0_DOFF	0x0001
DMA_TCD0_CITER_ELINKNO	0x0005
DMA_TCD0_CITER_ELINKYES	0x0005
DMA_TCD0_DLASTSGA	0xfffffffb
DMA_TCD0_CSR	0b0000000000000000
DMA_TCD0_BITER_ELINKNO	0x0005

Figure 13-Register status for Example 2 modification

The results of this modification are being shown in the following figure.

Name	Value	Location
i	13814	0x1ffffff4
g8bDestiny	0x1fff800c	0x1fff800c
[0]	8	0x1fff800c
[1]	6	0x1fff800d
[2]	4	0x1fff800e
[3]	2	0x1fff800f
[4]	0	0x1fff8010
[5]	0	0x1fff8011
[6]	0	0x1fff8012
[7]	0	0x1fff8013
[8]	0	0x1fff8014
[9]	0	0x1fff8015

Figure 14-Descending data movement

3 Appendix

The following appendix contains the code that was used for the implementation example:

main.c

```
#include "derivative.h" /* include peripheral declarations */
#include "dma.h"
#include "main.h"


int main(void)
{
	int i; // Delay variable

	// Enable clock and MUX for LEDs
	SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK;
	PORTC_PCR9 = PORT_PCR_MUX(1);
	PORTC_PCR10 = PORT_PCR_MUX(1);

	// Set LEDs as outputs
	GPIOC_PDDR |= (1 << 10) | (1 << 9);
	GPIOC_PCOR |= (1 << 9);  // Turn off Green LED D9
	GPIOC_PSOR |= (1 << 10); // Turn on Blue LED D10

	// Set SW1 as peripheral for DMA request on falling edge
	PORTC_PCR1 |=PORT_PCR_MUX(0x01) |
PORT_PCR_IRQC(0x02)|PORT_PCR_PE_MASK|PORT_PCR_PS_MASK;

	// Select size of data transfer
	#if (TEST)
		init_DMA_1Elem();
	#else
		init_DMA_1Trans();
	#endif

	while(1)
	{
		// Delay
		for(i = 0; i < 500000; i++)
			asm("nop");

		// Toggle Green LED D9 and Blue LED D10
		GPIOC_PTOR = (1 << 9) | (1 << 10);
	}
}
```

main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#define DMA_ONE_TRANSACTION 0    // 5 bytes per DMA request
#define DMA_ONE_ELEMENT     1    // 1 element per DMA request

// Select option
#define TEST  DMA_ONE_ELEMENT


#endif /* MAIN_H_ */
```

dma.c

```
#include "dma.h"
#include "derivative.h"

unsigned char g8bSource[10] = {0,1,2,3,4,5,6,7,8,9};
unsigned char g8bDestiny[10] = {1,0,0,0,0,0,0,0,0,0};


/*     init_DMA_1Trans
 *
 *     Initializes channel 0 DMA registers in order to perform 1 data transfer of 5 bytes
 *     and sets Port C as DMA request source
 *
 * */
void init_DMA_1Trans(void)
{
       // Enable clock for DMAMUX and DMA
       SIM_SCGC6 |= SIM_SCGC6_DMAMUX_MASK;
       SIM_SCGC7 |= SIM_SCGC7_DMA_MASK;

       // Enable Channel 0 and set Port C as DMA request source
       DMAMUX_CHCFG0 |= DMAMUX_CHCFG_ENBL_MASK | DMAMUX_CHCFG_SOURCE(51);

       // Enable request signal for channel 0
       DMA_ERQ = DMA_ERQ_ERQ0_MASK;

       // Set memory address for source and destination
       DMA_TCD0_SADDR = (uint32_t)&g8bSource;
       DMA_TCD0_DADDR = (uint32_t)&g8bDestiny;

       // Set an offset for source and destination address
       DMA_TCD0_SOFF = 0x02; // Source address offset of 2 bits per transaction
       DMA_TCD0_DOFF = 0x01; // Destination address offset of 1 bit per transaction

       // Set source and destination data transfer size
       DMA_TCD0_ATTR = DMA_ATTR_SSIZE(0) | DMA_ATTR_DSIZE(0);

       // Number of bytes to be transfered in each service request of the channel
       DMA_TCD0_NBYTES_MLNO = 0x05;
```

```
	// Current major iteration count (a single iteration of 5 bytes)
	DMA_TCD0_CITER_ELINKNO = DMA_CITER_ELINKNO_CITER(1);
	DMA_TCD0_BITER_ELINKNO = DMA_BITER_ELINKNO_BITER(1);

	// Adjustment value used to restore the source and destiny address to the initial
value
	DMA_TCD0_SLAST = -0x0A;          // Source address adjustment
	DMA_TCD0_DLASTSGA = -0x05;// Destination address adjustment

	// Setup control and status register
	DMA_TCD0_CSR = 0;
}


/*     init_DMA_1Elem
 *
 *     Initializes channel 0 DMA registers in order to perform 5 data transfers of 1 byte
each
 *     and sets Port C as DMA request source
 *
 * */
void init_DMA_1Elem(void)
{

	// Enable clock for DMAMUX and DMA
	SIM_SCGC6 |= SIM_SCGC6_DMAMUX_MASK;
	SIM_SCGC7 |= SIM_SCGC7_DMA_MASK;

	// Enable Channel 0 and set Port C as DMA request source
	DMAMUX_CHCFG0 |= DMAMUX_CHCFG_ENBL_MASK | DMAMUX_CHCFG_SOURCE(51);

	// Enable request signal for channel 0
	DMA_ERQ = DMA_ERQ_ERQ0_MASK;

	// Set memory address for source and destination
	DMA_TCD0_SADDR = (uint32_t)&g8bSource;
	DMA_TCD0_DADDR = (uint32_t)&g8bDestiny;

	// Set an offset for source and destination address
	DMA_TCD0_SOFF = 0x02; // Source address offset of 2 bits per transaction
	DMA_TCD0_DOFF = 0x01; // Destination address offset of 1 bit per transaction

	// Set source and destination data transfer size
	DMA_TCD0_ATTR = DMA_ATTR_SSIZE(0) | DMA_ATTR_DSIZE(0);

	// Number of bytes to be transfered in each service request of the channel
	DMA_TCD0_NBYTES_MLNO = 0x01;

	// Current major iteration count (5 iteration of 1 byte each)
	DMA_TCD0_CITER_ELINKNO = DMA_CITER_ELINKNO_CITER(5);
	DMA_TCD0_BITER_ELINKNO = DMA_BITER_ELINKNO_BITER(5);
```

DMA transfer introduction example

```
	// Adjustment value used to restore the source and destiny address to the initial
value
	DMA_TCD0_SLAST = -0x0A;          // Source address adjustment
	DMA_TCD0_DLASTSGA = -0x05;// Destination address adjustment

	// Setup control and status register
	DMA_TCD0_CSR = 0;
}
```

dma.h

```
#ifndef DMA_H_
#define DMA_H_

/* Prototypes */
void init_DMA_1Trans(void);
void init_DMA_1Elem(void);

#endif /* DMA_H_ */
```