XMC7000: Using direct memory access (DMA) controllers

About this document

Scope and purpose

This application note describes how to use DMA controllers (DW and DMAC) in XMC7000 family MCUs. DMA controllers can transfer data from a source to a destination without CPU intervention. The application note illustrates how to configure DMA for peripheral-to-memory, memory-to-peripheral, and memory-to-memory data transfers.

Intended audience

This document is intended for anyone who uses the XMC7000 MCU to use the DMA and DMAC functions.

Associated part family

XMC7000 MCU family of XMC™ industrial microcontrollers.

Table of contents

Table of contents

1 Introduction

This application note describes how to use the direct memory access (DMA) controller in XMC7000 family MCUs.

DMA controllers can seamlessly transfer data between memory and on-chip peripherals, or between memories without CPU intervention. This allows the CPU to handle other tasks while the DMA controller transfers data.

Both DW and DMAC have multiple independent DMA channels. Each DMA channel has a separate DMA request input that initiates the transaction and can independently transfer data. See the device datasheet for the number of DMA channels available for each device.

This series supports two types of DMA controllers: Peripheral DMA (DW) and Memory DMA (DMAC). DW is used for peripheral-to-memory and memory-to-peripheral low-latency data transfers for many channels. DMAC is used for memory-to-memory high-memory-bandwidth data transfer for a small number of channels.

These DMA controllers have a descriptor that specifies the transfer operation, and it corresponds flexibly to various applications. Descriptors can be chained; it is possible to have circular lists.

This application note explains the functioning of DMA controllers in the series, initial configuration, and data transfer operations with use cases.

To understand the functionality described and terminology used in this application note, see the “Direct Memory Access” chapter of the architecture technical reference manual (TRM).

1.1 Features

Table 1 compares DW with DMAC, which have similar registers and descriptor structures.

Table 1 DW/DMAC features

Feature	DW	DMAC
Focuses on	Low latency	High memory bandwidth
Useful for	Transfer between peripheral and memory	Transfer between memories
Transfer engine	Shared all channels	Dedicated for each channel
Transfer size	8-bit, 16-bit, 32-bit	8-bit, 16-bit, 32-bit
Channel priority	• Four levels • Preemptable	Four levels
Descriptor type	• Single transfer • 1D/2D transfer • CRC transfer	• Single transfer • 1D/2D transfer • Memory copy • Scatter
Descriptor	• Source and destination address • Transfer size • Descriptor type • Trigger-in type (four types) • Trigger-out type (four types) • Interrupt type (four types) • Descriptor chaining	• Source and destination address • Transfer size • Descriptor type • Trigger-in type (four types) • Trigger-out type (four types) • Interrupt type (four types) • Descriptor chaining

Introduction

Feature	DW	DMAC
Trigger input	• Hardware trigger • Software trigger • Trigger output (tr_out)	• Software trigger • Trigger output (tr_out)

DW can be also used for transfers between memories, but the transfer bandwidth may not be enough when compared with DMAC. DMAC can also be used for transfers between memory and peripherals, but the transfer latency may not be low when compared with DW.

In DW, when preemptable, a higher-priority pending channel can preempt the current channel between single transfers. DMAC does not have the preemptable functionality because it would degrade the overall memory bandwidth.

The descriptor determines the DMA transfer specification. The descriptor type determines the type of DMA transfer operation. Both DMAs support single transfer, 1D transfer, and 2D transfer as descriptor types. In addition, DW supports CRC transfer, while DMAC supports memory copy and scatter. See section 2.3.1 Descriptor type for details of each descriptor type. Descriptors can be chained by storing the pointer of the next descriptor in the current descriptor. A descriptor chain is also referred to as a descriptor list.

Trigger inputs such as hardware trigger, software trigger, and trigger output (tr_out) are input via the trigger multiplexer, which is a peripheral function outside DMA. The trigger multiplexer routes triggers from potential sources to destinations. See the “Trigger Multiplexer” chapter of the architecture TRM for more details.

DW supports hardware trigger, software trigger, and trigger output (tr_out) as trigger inputs, while DMAC supports only software trigger and trigger output (tr_out). See the device datasheet for hardware triggers available. The software trigger is implemented by the trigger multiplexer function. Both DMAs can use the trigger output as their own input trigger. See section 2.3.2 Trigger functionality for each trigger functionality.

1.2 Block diagram

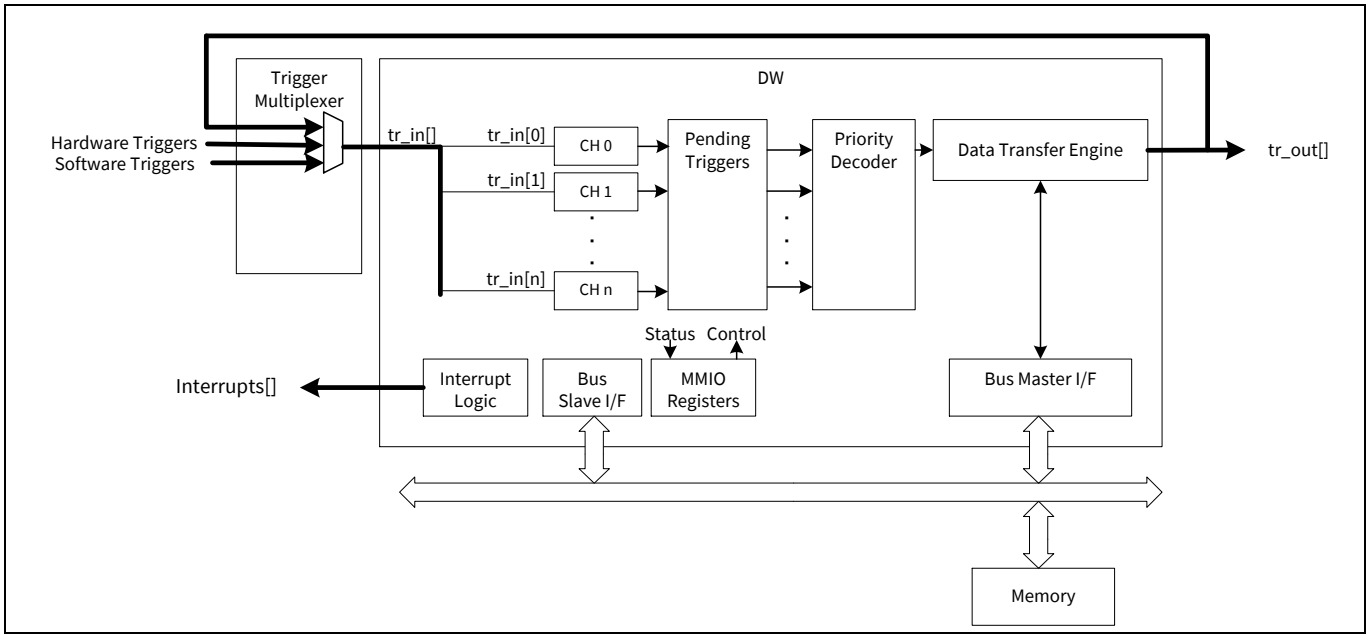


Figure 1 **DW block diagram**

Introduction

DW consists of channels (CH0 – CH*n*), a pending trigger block, priority decoder, data transfer engine, and the interrupt logic. The DW transfer engine is shared by all channels. See the architecture TRM for details of each block.

As mentioned earlier, DW trigger inputs can be a hardware trigger, software trigger, or trigger output (tr_out). These triggers are input via the trigger multiplexer.

The trigger output (tr_out) can be used as its own trigger input, or it can be used as the trigger input to trigger different transfers of other channels.

The memory that is used to store descriptors is outside the DMA block. When the transfer engine activates the next pending channel, the transfer engine reads the descriptor corresponding to the channel from the memory and starts the transfer.

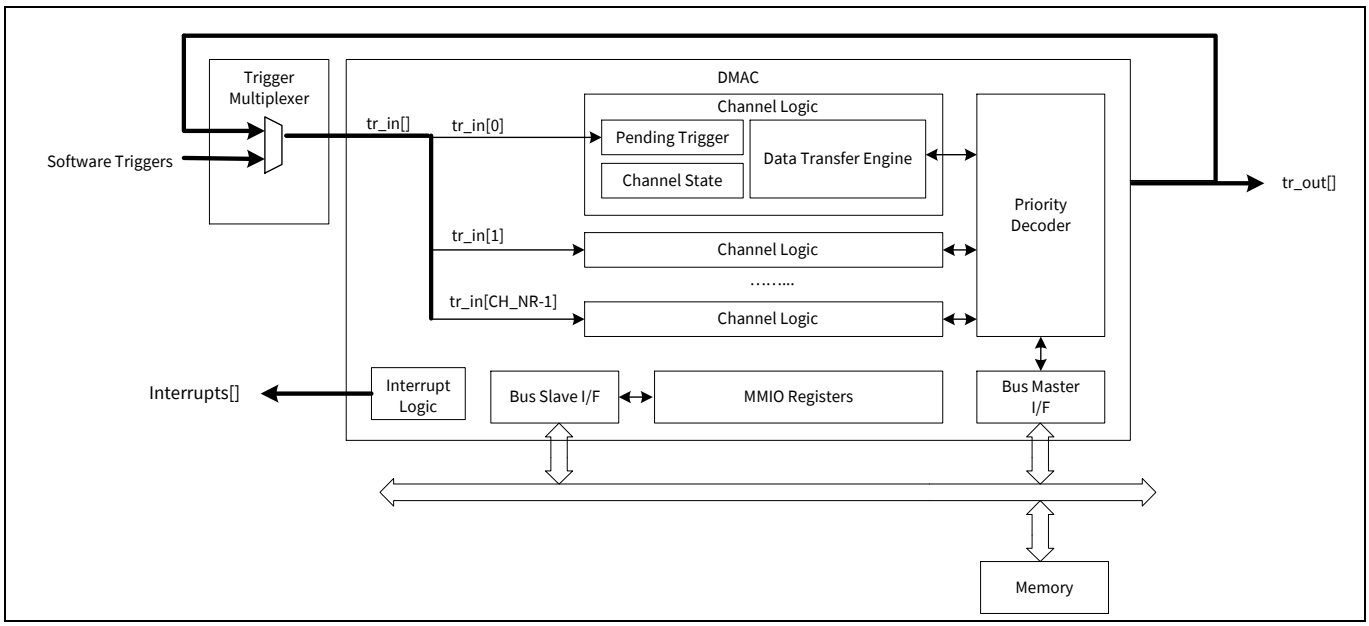


Figure 2 **DMAC block diagram**

The DMAC block consists of the channel logic, priority decoder, and registers. The channel logic itself stores the pending trigger and hosts the current channel state and data transfer engine. DMAC has transfer engines dedicated for each channel. See the architecture TRM for details of each block.

As trigger inputs, DMAC supports software trigger and its own trigger output (tr_out). These trigger inputs are input via the trigger multiplexer. Note that unlike DW, DMAC does not support hardware triggers.

2 Operation overview

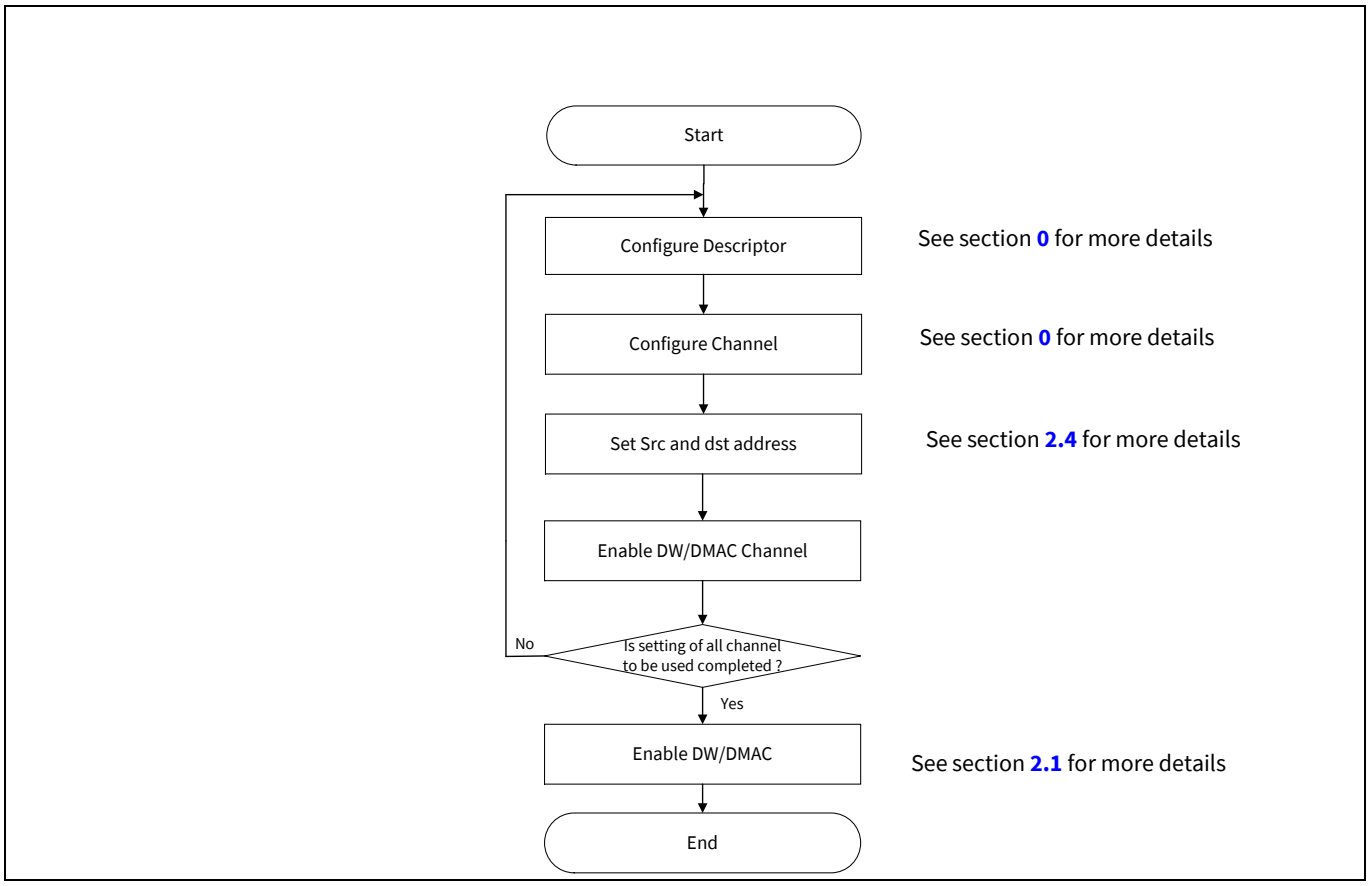


Figure 3 **General configuration of DW/DMAC**

In this example, channel, descriptor, and channel enable are configured for each channel. It is also possible to configure all channels to be used within each step.

A peripheral trigger is required after setting the corresponding DMA channel.

2.1 Disable/enable DW/DMAC

DW and DMAC can be enabled/disabled using the respective bits as shown in Table 2. The default setting after reset is '0' (Disabled).

Table 2 DW/DMAC disable/enable

DMA type	Register (bit)	Description
DW	DW_CTL.ENABLED (bit31)	0: Disable, 1: Enable
DMAC	DMAC_CTL.ENABLED (bit31)	0: Disable, 1: Enable

2.2 Configure channel

In this step, DW/DMAC channel settings such as the channel priority and pointer address of the descriptor corresponding to the channel are configured.

In addition, in DW, the preemptable function and CRC calculation mode for CRC transfer are configured, if necessary.

Table 3 and Table 4 show the registers that are used for configuring a channel in DW and DMAC, respectively. The registers corresponding to the channel number are configured. See the architecture TRM and registers technical reference manual (registers TRM) for more details.

Table 3 Channel configuration for DW

Register (bit)	Description
DW_CH_STRUCT_CH_CURR_PTR	Sets the channel current descriptor pointer. Software needs to initialize this register
DW_CH_STRUCT_CH_CTL.PREEMPTABLE (bit11)	Specifies whether the channel is preemptable
DW_CH_STRUCT_CH_CTL.PRIO (bit9:8)	Sets the channel priority
DW_CH_STRUCT_CH_IDX.X_IDX (bit7:0)	Sets the X indices of the channel into the current descriptor. Software needs to initialize this register.
DW_CH_STRUCT_CH_IDX_Y_IDX (bit15:0)	Sets the Y indices of the channel into the current descriptor. Software needs to initialize this register.
Required only for CRC transfer:	
DW_CRC_CTL.DATA_REVERSE (bit0)	Specifies the bit order (MSb or LSb first) in which a data byte is processed
DW_CRC_CTL.REM_REVERSE (bit8)	Specifies whether the remainder is bit reversed
DW_CRC_DATA_CTL.DATA_XOR (bit7:0)	Sets the byte mask with which each data byte is XORed. You can choose this 8-bit value randomly.
DW_CRC_POL_CTL.POLYNOMIAL	Sets the CRC polynomial
DW_CRC_LFSR_CTL.LFSR32	Sets the seed value for CRC calculation
DW_CRC_REM_CTL.REM_XOR	Sets a mask with which the CRC_LFSR_CTL.LFSR32 register is XORed

Table 4 Channel configuration for DMAC

Register (bit)	Description
DMAC_CH_CURR.PTR	Sets the channel current descriptor pointer. Software needs to initialize this register.
DMAC_CH_CH_CTL.PRIO (bit9:8)	Sets the channel priority

2.3 Configure descriptor

In this step, the descriptor is configured. The descriptor specifies the DMA channel's transfer details. A descriptor is stored in the memory outside DMA and read by the transfer engine. The transfer engine transfers the data according to the descriptor. The descriptor pointer position for each channel is stored in the descriptor pointer register (see section Configure channel).

Figure 4 shows the descriptor structure for DW and DMAC. The DW descriptor consists of six 32-bit words, while the DMAC descriptor consists of eight 32-bit words. However, descriptors of both DMAs have similar functions.

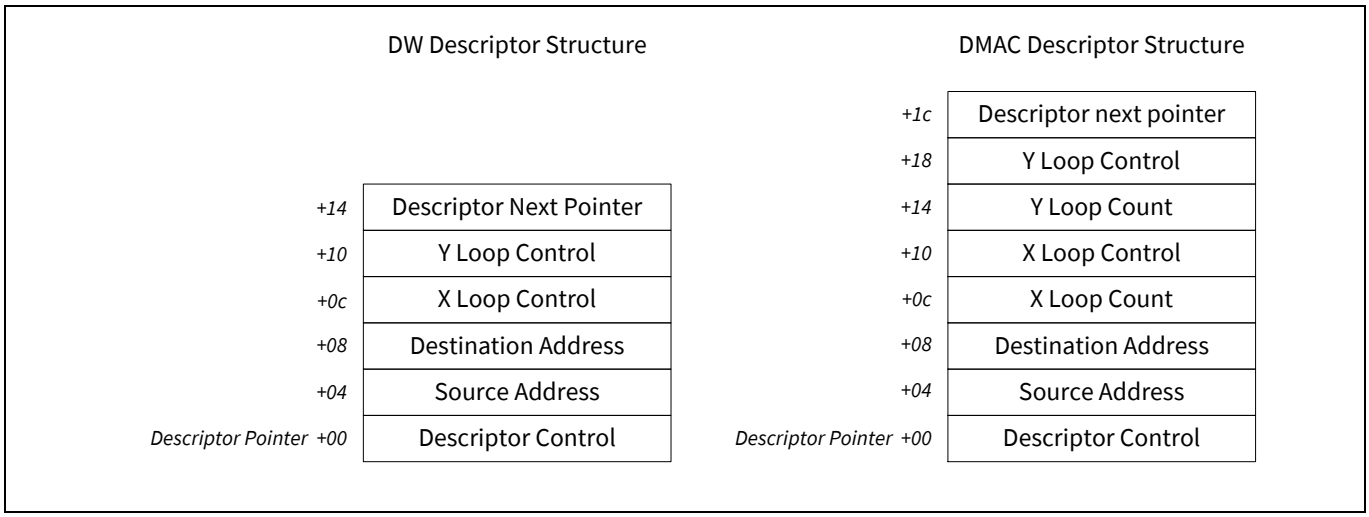


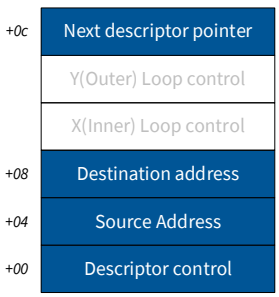
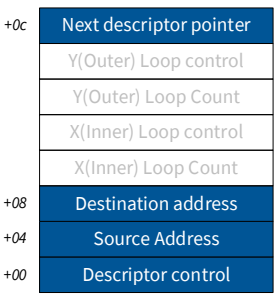
Figure 4 **DW/DMAC descriptor structure**

2.3.1 Descriptor type

This section explains the descriptor type, which determines the type of DMA transfer.

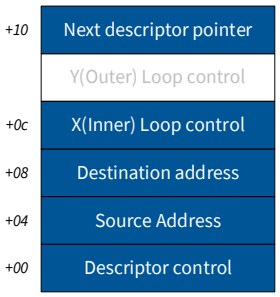
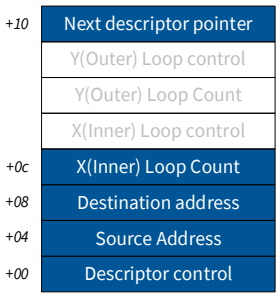
DW has four descriptor types, and DMAC has five descriptor types. The number of descriptor words used varies depending on the descriptor type. Table 5 shows each descriptor type. In Table 5, the transfer example column shows the outline and pseudocode of each descriptor type. The Using Descriptor column shows the descriptor word used by the descriptor types in each DMA. Note that a word address is shifted forward if there is any unused descriptor word.

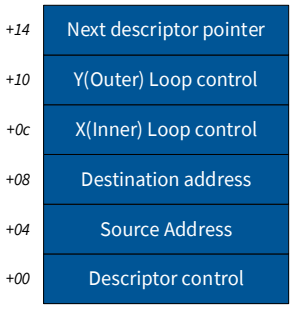
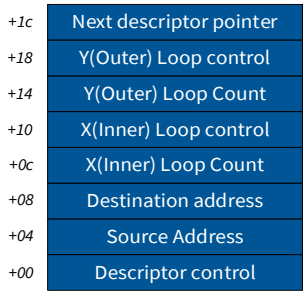
Table 5 DW/DMAC transfer example and using descriptor for each descriptor type

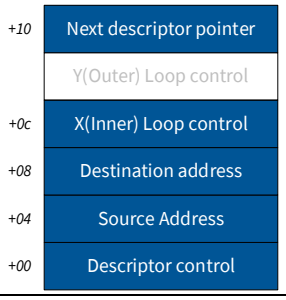
Descriptor type	Transfer example	Using descriptor	
		DW	DMAC
Single transfer	This transfers a single data element: `DST_ADDR = (DATA_SIZE) SRC_ADDR`		

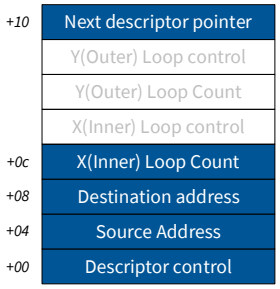
Operation overview

Descriptor type	Transfer example	Using descriptor	
		DW	DMAC

1D transfer	One-dimensional “for” loop transfer:
	`for (X_IDX =0; X_IDX <= COUNT; X_IDX++) {`
	`  DST_ADDR[DST_INCR] = (DATA_SIZE) SRC_ADDR[SRC_INCR]`
	`}`
	`*DST_INCR/SRC_INCR depend on X_INCR DST_ADDR = (DATA_SIZE) SRC_ADDR`
			

2D transfer	Two-dimensional “for” loop transfer:
	`for (Y_IDX =0; Y_IDX <= Y_COUNT; Y_IDX++) {`
	`  for (X_IDX =0; X_IDX <= X_COUNT; X_IDX++) {`
	`DST_ADDR[DST_INCR] = (DATA_SIZE) SRC_ADDR[SRC_INCR]`
	`  }`
	`}`
	`*DST_INCR/SRC_INCR depend on X/Y_INCR`
			

CRC transfer	Calculate CRC of the specified area. Note that for CRC transfer, CRC must be configured with memory mapped I/O (MMIO) registers.		Not supported

Memory copy	One-dimensional “for” loop transfer:
	`for (X_IDX =0; X_IDX <= X_COUNT; X_IDX+) {`
	`  DST_ADDR[IDX] = SRC_ADDR[IDX]`
	`}`
		Not supported	

Scatter	Write a set of 32-bit data elements, whose addresses are “scattered” around one-dimensional “for” loop transfer.	Not supported	

Operation overview

Descriptor type	Transfer example	Using descriptor	
		DW	DMAC

	`for (X_IDX =0; X_IDX <= X_COUNT; X_IDX +=2) {`
	`address = SRC_ADDR[IDX]`
	`data = SRC_ADDR[IDX+1]`
	`*address = data`
	`}`
			+0c Next descriptor pointer
			Y(Outer) Loop control
			Y(Outer) Loop Count
			X(Inner) Loop control
			X(Inner) Loop Count
			+08 Destination address
			+04 Source Address
			+00 Descriptor control

2.3.2 Trigger functionality

This section explains the trigger function. Trigger input, trigger output (tr_out), and interrupt are controlled by the descriptor.

A trigger input activates DMA channel transfer. The trigger output (tr_out) and interrupt are output when the transfer is complete. The trigger operation is specified by TR_IN_TYPE, TR_OUT_TYPE, and INTR_TYPE in the descriptor. Trigger input, trigger output (tr_out), and interrupt can be configured independently for each channel. See the registers TRM for descriptor details.

- There are four types of trigger input operations:
  - Type 0: Trigger results in the execution of one transfer per trigger.
  - Type 1: Trigger results in the execution of one X loop transfer per trigger.
  - Type 2: Trigger results in the execution of an entire descriptor transfer per trigger.
  - Type 3: Trigger results in the execution of entire descriptor chain per trigger.
- There are four types of trigger outputs and interrupt timing:
  - Type 0: Output trigger or interrupt is generated after every element transfer completion.
  - Type 1: Output trigger or interrupt is generated after every X loop transfer completion.
  - Type 2: Output trigger or interrupt is generated after descriptor completion.
  - Type 3: Output trigger or interrupt is generated after completion of entire descriptor chain.

2.3.2.1 Examples for trigger functionality

This section provides examples for different trigger functions. The descriptor list for the following examples is composed of chaining two descriptors (Descriptor 0 and Descriptor 1) in a 2D transfer.

Example 1:

This example describes the operation of a Type 0 trigger.

Figure 5 shows the operation of the trigger input and trigger output or interrupt in Type 0 trigger.

Operation overview

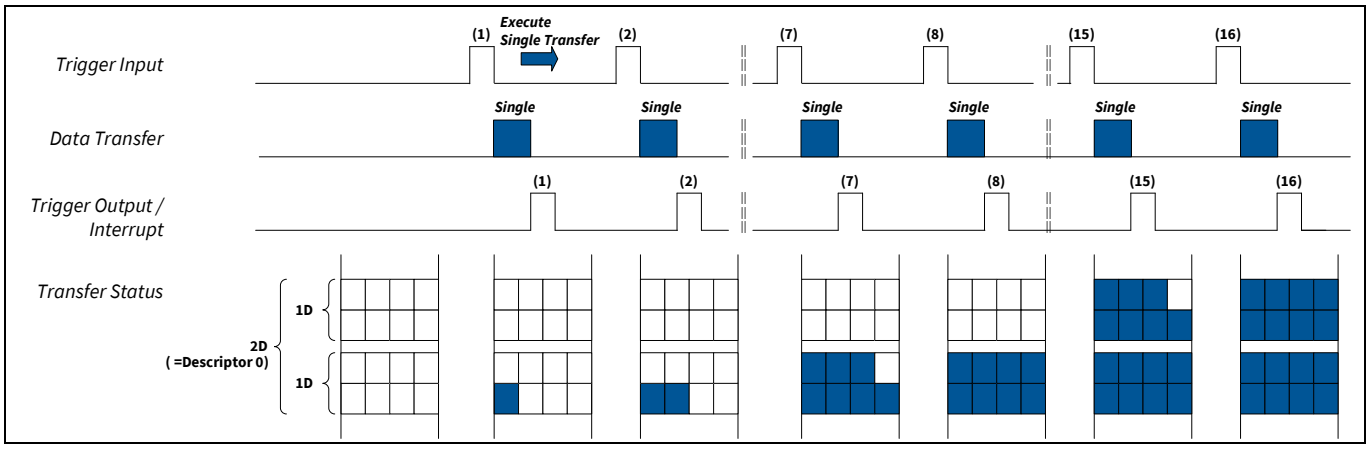


Figure 5 **Trigger operation example 1**

When the trigger input is set in Type 0 trigger, DW/DMAC performs a single transfer with every trigger input. Therefore, 16 trigger inputs are required to complete Descriptor 0.

When trigger outputs and interrupts are set in Type 0 trigger, the trigger output, interrupt, or both are output each time a single transfer is completed. Therefore, trigger output, interrupt, or both are output 16 times with the completion of Descriptor 0.

Example 2:

This example describes the Type 1 trigger operation.

Figure 6 shows the operation of trigger input and trigger output or interrupt in Type 1 trigger.

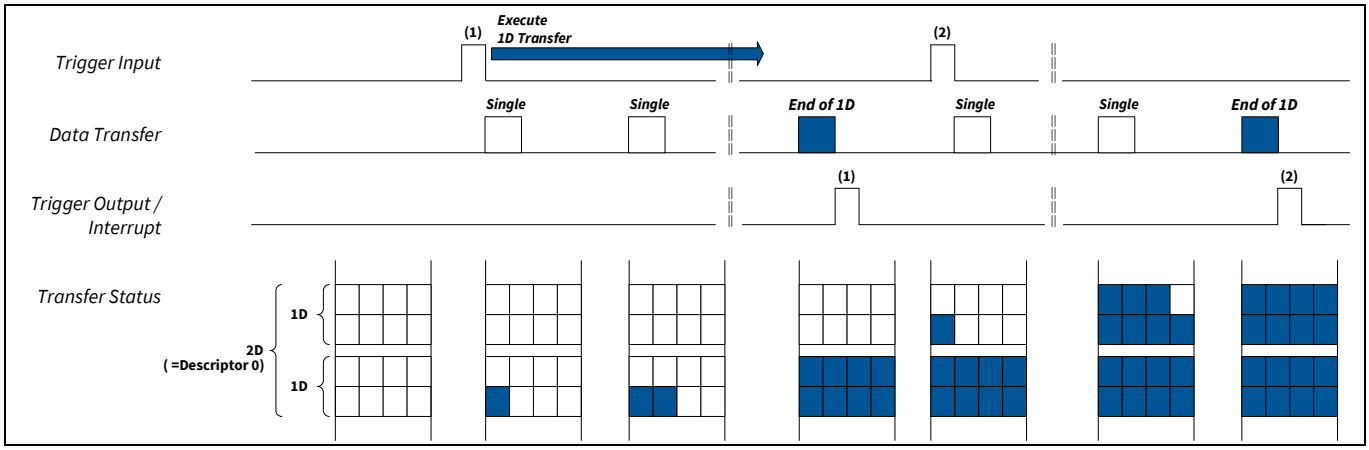


Figure 6 **Trigger operation example 2**

When trigger input is set in Type 1 trigger, DW/DMAC performs a 1D transfer with the trigger input. If the next trigger occurs again, DW/DMAC performs a 1D transfer. Therefore, two trigger inputs are required to complete Descriptor 0.

When trigger outputs and interrupts are set in Type 1 trigger, trigger output, interrupt, or both are output each time a 1D transfer is completed. Therefore, trigger output, interrupt, or both are output twice with the completion of Descriptor 0.

Operation overview

Example 3:

This example describes the operation of Type 2 trigger.

Figure 7 shows the operation of the trigger input and trigger output or interrupt in Type 2 trigger.

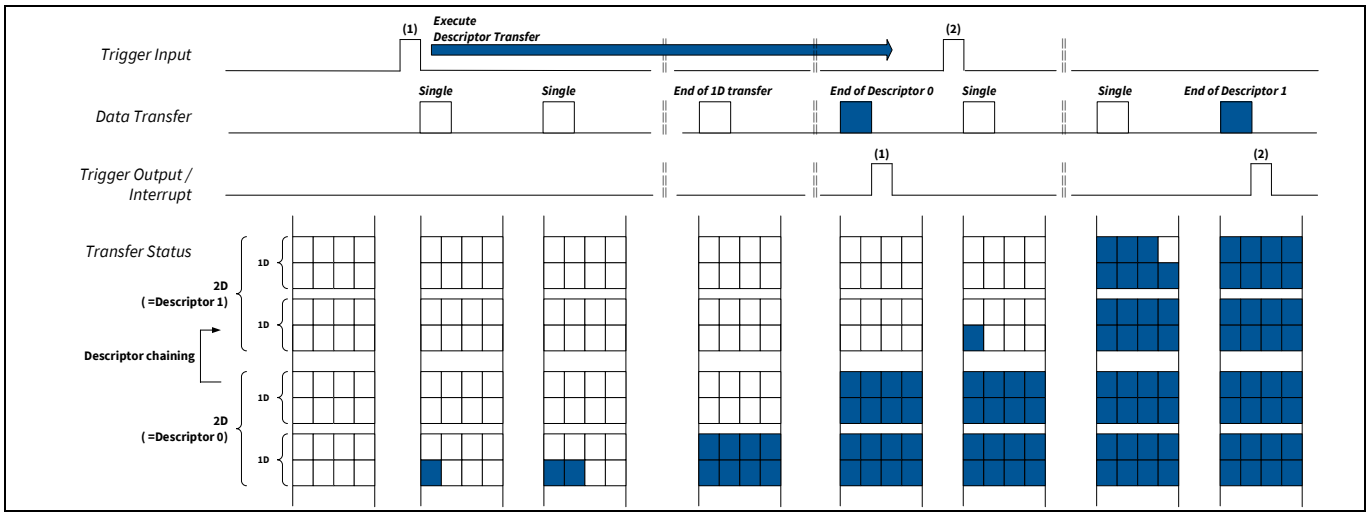


Figure 7 **Trigger operation example 3**

When the trigger input is set in Type 2 trigger, DW/DMAC executes the current descriptor (here: Descriptor 0) with the trigger input. If the next trigger input occurs, DW/DMAC executes the next descriptor (here: Descriptor 1). Therefore, two trigger inputs are required to complete the descriptor list.

When trigger outputs and interrupts are set in Type 2 trigger, trigger output, interrupt, or both are output each time a current descriptor transfer is completed. Therefore, trigger output, interrupt, or both are output twice with the completion of the descriptor list.

Example 4:

This example describes the operation of Type 3 trigger.

Figure 8 shows the operation of the trigger input and trigger output or interrupt in Type 3 trigger.

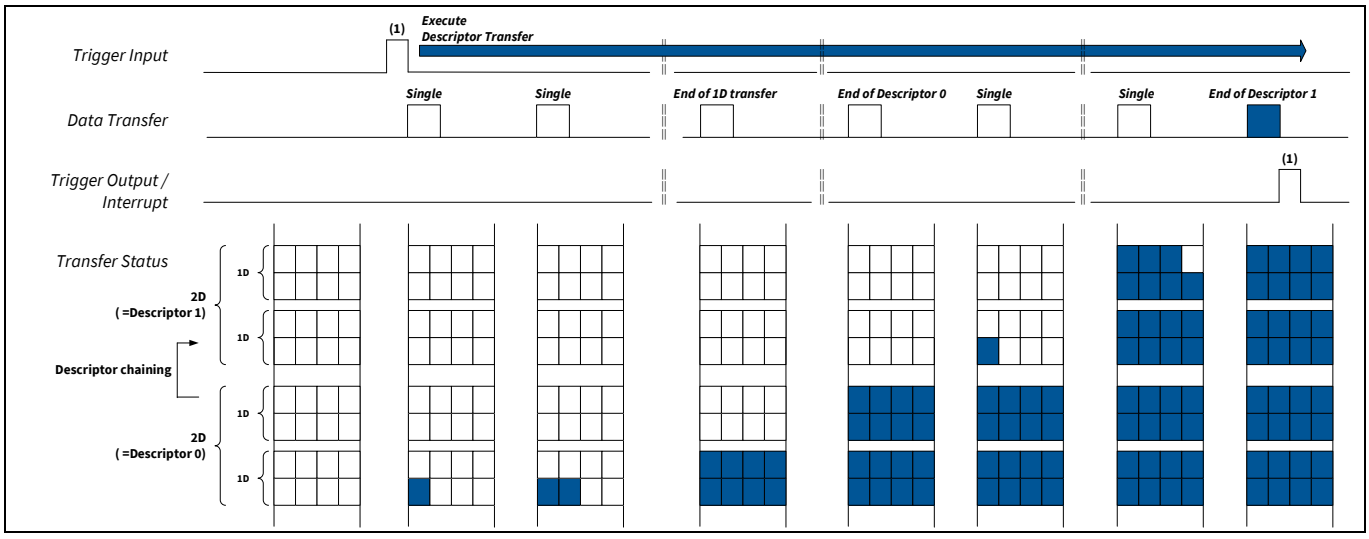


Figure 8 **Trigger operation example 4**

Operation overview

When the trigger input is set in Type 3 trigger, DW/DMAC executes the complete descriptor list with each trigger input. Therefore, one trigger input is required to complete the descriptor list.

When trigger outputs and interrupts are set in Type 3 trigger, the trigger output, interrupt, or both are output when descriptor list transfer is completed. Therefore, the trigger output and/or interrupt are output once with the completion of the descriptor list.

2.4 Disable/enable DW/DMAC channel

DW and DMAC can be configured/programmed to execute multiple independent data transfers. Each data transfer is managed by a channel.

In DMA channel configuration, the DMA channel must be disabled during the configuration, and enabled after configuring the channel. Table 6 shows the registers used for enabling or disabling a DMA channel.

The number of channels varies for different part numbers. See the device datasheet for the number of available channels.

Table 6 Disable/enable DW/DMAC channel

DMA type	Register (bit)	Description
DW	DW_CH_STRUCT_CH_CTL.ENABLED (bit31)	0: Disable, 1: Enable
DMAC	DMAC_CH_CTL.ENABLED (bit31)	0: Disable, 1: Enable

3 DW use cases

This section describes how to use DW using the peripheral driver library (PDL). The code snippets in this application note are part of PDL.

PDL has a configuration part and a driver part. The configuration part mainly configures the parameter values for the desired operation. The driver part configures each register based on the parameter values in the configuration part. You can configure the configuration part according to your system.

In this example, XMC7200 series is used.

3.1 1D transfer (memory-to-peripheral)

3.1.1 Overview

This is an example of transferring the transmit data from the memory to TX-FIFO by DMA when using the SCB UART mode. In this case, the UART uses an 8-bit data frame, and starts the transmission when the data is written to the FIFO. The number of bytes to be transmitted is 1 and the TX FIFO setting is 1 elements-deep with 8-bit data elements. See the “Serial Communications Block (SCB)” chapter of the architecture TRM for more details.

DW starts the transfer with a software trigger from the CPU, and generates an interrupt to the CPU after transferring all data into the transmit FIFO.

Figure 9 shows the data transfer.

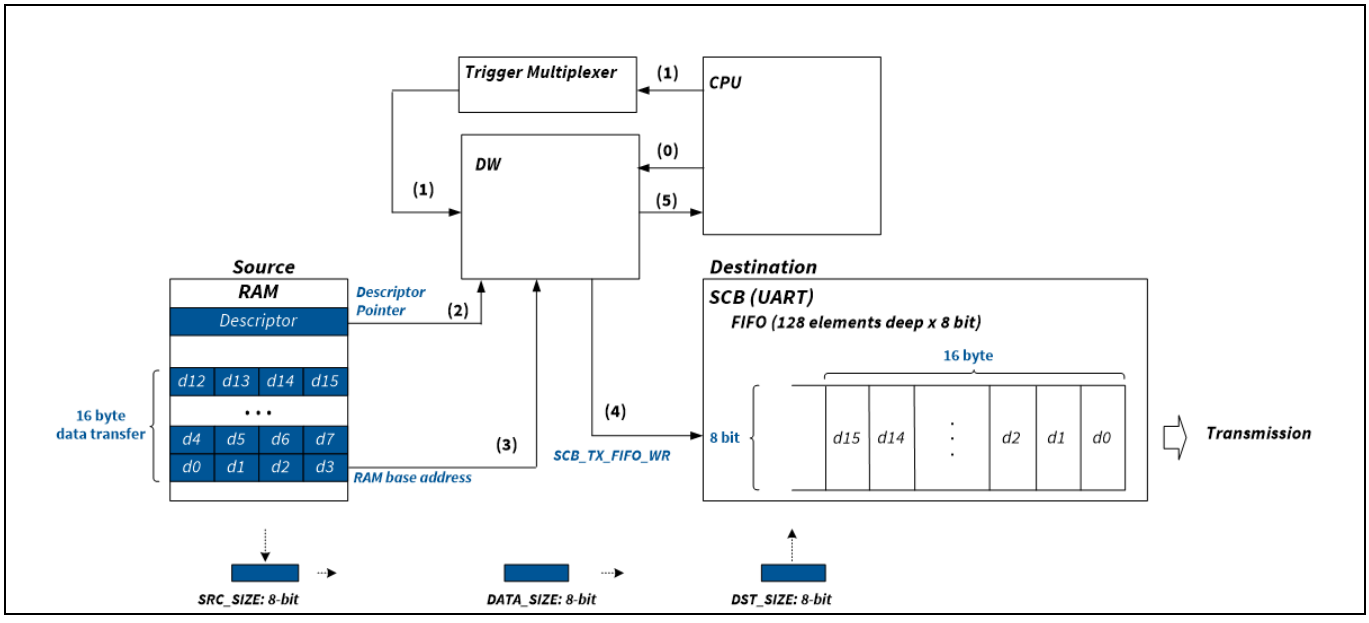


Figure 9 **Use case of a memory-to-peripheral (1D transfer) using UART in SCB**

1. Configure DW according to section 3.1.2 Initial configuration of channel registers, and configure the SCB and trigger multiplexer.
2. CPU notifies a software trigger to DW via the trigger multiplexer.
3. DW reads the descriptor from the specified area (Descriptor Pointer) when activating the next pending channel.

DW use cases

4. DW reads data (d0) from the source address (RAM base address).
5. DW writes the read data (d0) to the destination address (SCB_TX_FIFO_WR). After that, DW increments the source address by 0x01, but there is no increment of the destination address. Then, DW reads the data (d1) from the source address (RAM base address +0x01) and writes it to the destination address (SCB_TX_FIFO_WR) again. DW repeats from (3) to (4) until d15 data is transferred.
6. DW notifies the CPU with an interrupt when the transfer of all data is completed.

Initialize the channel registers and set the descriptor as follows.

3.1.2 Initial configuration of channel registers

This section describes the initialization of the DW channel and descriptor of this use case.

Figure 10 shows the setting procedure for DW.

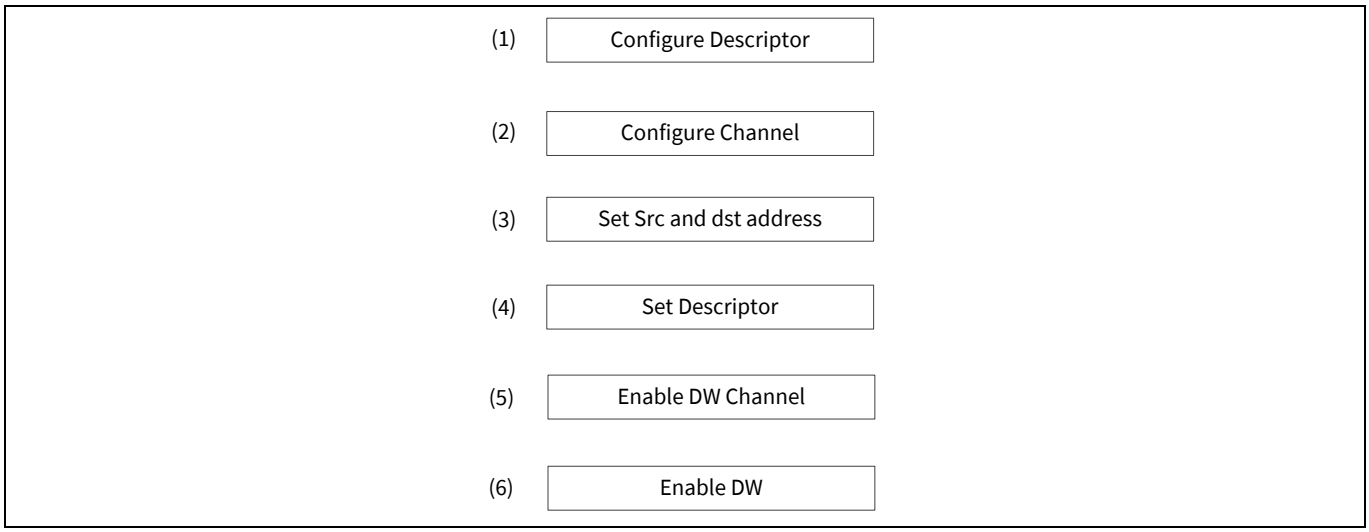


Figure 10 **Setting procedure for DW**

3.1.3 Example code

Code Listing 1 demonstrates an example program to 1D Transfer (memory-to-peripheral). See the architecture TRM and application note for GPIO, UART, and clock configuration.

Figure 11 is a screengrab from the ModusToolbox™ design.modus that shows 1D transfer (memory-to-peripheral) function settings.

DW use cases

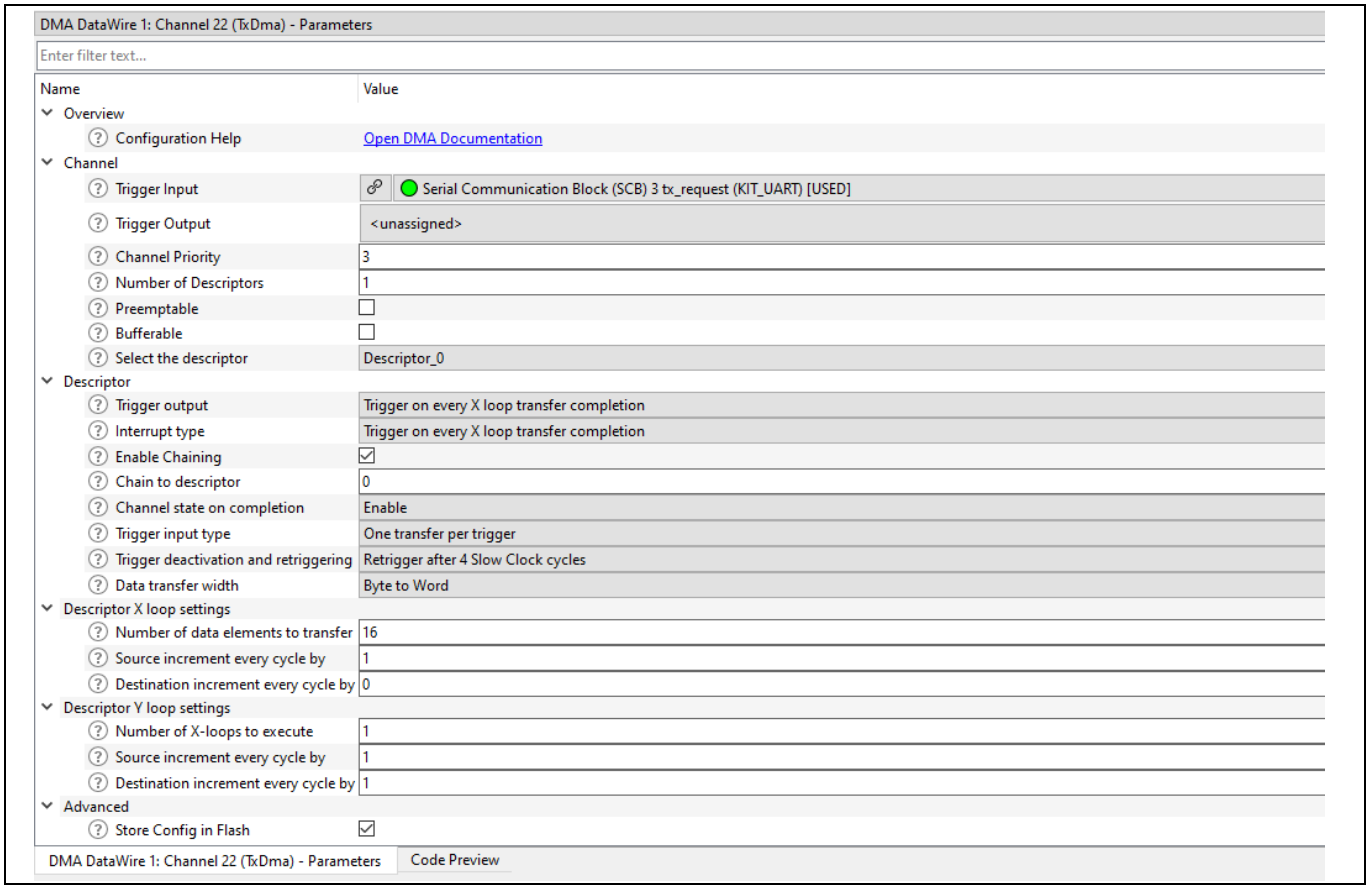


Figure 11 **Operation of 1D transfer (memory-to-peripheral)**

- See *cyip_dw.h mtb_shared/mtb-pdl-cat1/release-v3.3.0/devices/COMPONENT_CAT1C/include/ip* for more information on the union and structure representation of registers.

Code Listing 1 1D transfer (memory-to-peripheral) example

```
#define KIT_UART_ENABLED 1U
#define KIT_UART_HW SCB3
#define KIT_UART_IRQ scb_3_interrupt_IRQn
#define TxDma_ENABLED 1U
#define TxDma_HW DW1
#define TxDma_CHANNEL 22U
#define TxDma_IRQ cpuss_interrupts_dw1_22_IRQn
#define RxDma_ENABLED 1U
#define RxDma_HW DW1
#define RxDma_CHANNEL 23U
#define RxDma_IRQ cpuss_interrupts_dw1_23_IRQn

void configure_tx_dma(uint8_t* buffer_a, cy_stc_sysint_t* int_config)
{
    cy_en_dma_status_t dma_init_status;

    /* Init descriptor */
    dma_init_status = Cy_DMA_Descriptor_Init(&TxDma_Descriptor_0, &TxDma_Descriptor_0_config);
    if (dma_init_status!=CY_DMA_SUCCESS)
    {
        handle_error();
    }
    dma_init_status = Cy_DMA_Channel_Init(TxDma_HW, TxDma_CHANNEL, &TxDma_channelConfig);
    if (dma_init_status!=CY_DMA_SUCCESS)
    {
        handle_error();
    }

    /* Set source and destination for descriptor 1 */
    Cy_DMA_Descriptor_SetSrcAddress(&TxDma_Descriptor_0, (uint32_t *) buffer_a);
    Cy_DMA_Descriptor_SetDstAddress(&TxDma_Descriptor_0, (uint32_t *) &KIT_UART_HW->TX_FIFO_WR);
```

DW use cases

Code Listing 1 1D transfer (memory-to-peripheral) example

```
    /* Set next descriptor to NULL to stop the chain execution after descriptor 1
    *  is completed.
    */
    Cy_DMA_Descriptor_SetNextDescriptor(Cy_DMA_Channel_GetCurrentDescriptor(TxDma_HW, TxDma_CHANNEL), NULL);

    /* Initialize and enable the interrupt from TxDma */
    Cy_SysInt_Init  (int_config, &tx_dma_complete);
    NVIC_EnableIRQ((IRQn_Type) NvicMux2_IRQn);

    /* Enable DMA interrupt source */
    Cy_DMA_Channel_SetInterruptMask(TxDma_HW, TxDma_CHANNEL, CY_DMA_INTR_MASK);

    /* Enable Data Write block but keep channel disabled to not trigger
    *  descriptor execution because TX FIFO is empty and SCB keeps active level
    *  for DMA.
    */
    Cy_DMA_Enable(TxDma_HW);
}
int main(void)
{
    cy_rslt_t result;
    uint8_t rx_dma_uart_buffer_a[BUFFER_SIZE];
    cy_en_scb_uart_status_t init_status;
    cy_stc_scb_uart_context_t KIT_UART_context;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }
    cy_stc_sysint_t KIT_UART_INT_cfg =
    {
        .intrSrc = ( (NvicMux3_IRQn << 16) | KIT_UART_IRQ ),
        .intrPriority = 7u,
    };

    cy_stc_sysint_t RX_DMA_INT_cfg =
    {
        .intrSrc      = ( (NvicMux3_IRQn << 16) | (IRQn_Type)RxDma_IRQ ),
        .intrPriority = 6u,
    };

    cy_stc_sysint_t TX_DMA_INT_cfg =
    {
        .intrSrc      = ( (NvicMux2_IRQn << 16) | (IRQn_Type)TxDma_IRQ ),
        .intrPriority = 6u,
    };

    /* Configure DMA Rx and Tx channels for operation */
    configure_rx_dma(rx_dma_uart_buffer_a, rx_dma_uart_buffer_b, &RX_DMA_INT_cfg);
    configure_tx_dma(rx_dma_uart_buffer_a, &TX_DMA_INT_cfg);

    /* Initialize and enable interrupt from UART. The UART interrupt sources
    *  are enabled in the Component GUI */
    Cy_SysInt_Init(&KIT_UART_INT_cfg, &Isr_UART);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Start UART operation */
    init_status = Cy_SCB_UART_Init(KIT_UART_HW, &KIT_UART_config, &KIT_UART_context);
    if (init_status!=CY_SCB_UART_SUCCESS)
    {
        handle_error();
    }
    Cy_SCB_UART_Enable(KIT_UART_HW);

    while(1)
    {
        /* Handle RxDma complete */
        if (rx_dma_done==1)
        {
            /* Set source RX Buffer A as source for TxDMA */
            Cy_DMA_Descriptor_SetSrcAddress(&TxDma_Descriptor_0, (uint32_t *) rx_dma_uart_buffer_a);

            Cy_DMA_Channel_SetDescriptor(TxDma_HW, TxDma_CHANNEL, &TxDma_Descriptor_0);
            Cy_DMA_Channel_Enable(TxDma_HW, TxDma_CHANNEL);
         rx_dma_done = 0;
        }
    }

}
```

3.2 1D transfer (peripheral-to-memory)

3.2.1 Overview

This is an example of transferring the transmit data from RX-FIFO to RAM by DMA when using the SCB UART mode. In this case, the UART uses an 8-bit data frame, and starts the transmission when UART receive data. The number of bytes to be transmitted is 32 and the RX FIFO setting is 1 elements-deep with 8-bit data elements. See the “Serial Communications Block (SCB)” chapter of the architecture TRM for more details.

The UART generates a receive interrupt after it receives the FIFO data, and the DW starts transferring the KIT_UART_HW->RX_FIFO_RD FIFO data to RAM. After the data transfer is complete, the DW generates an interrupt.

Figure 12 shows the data transfer.

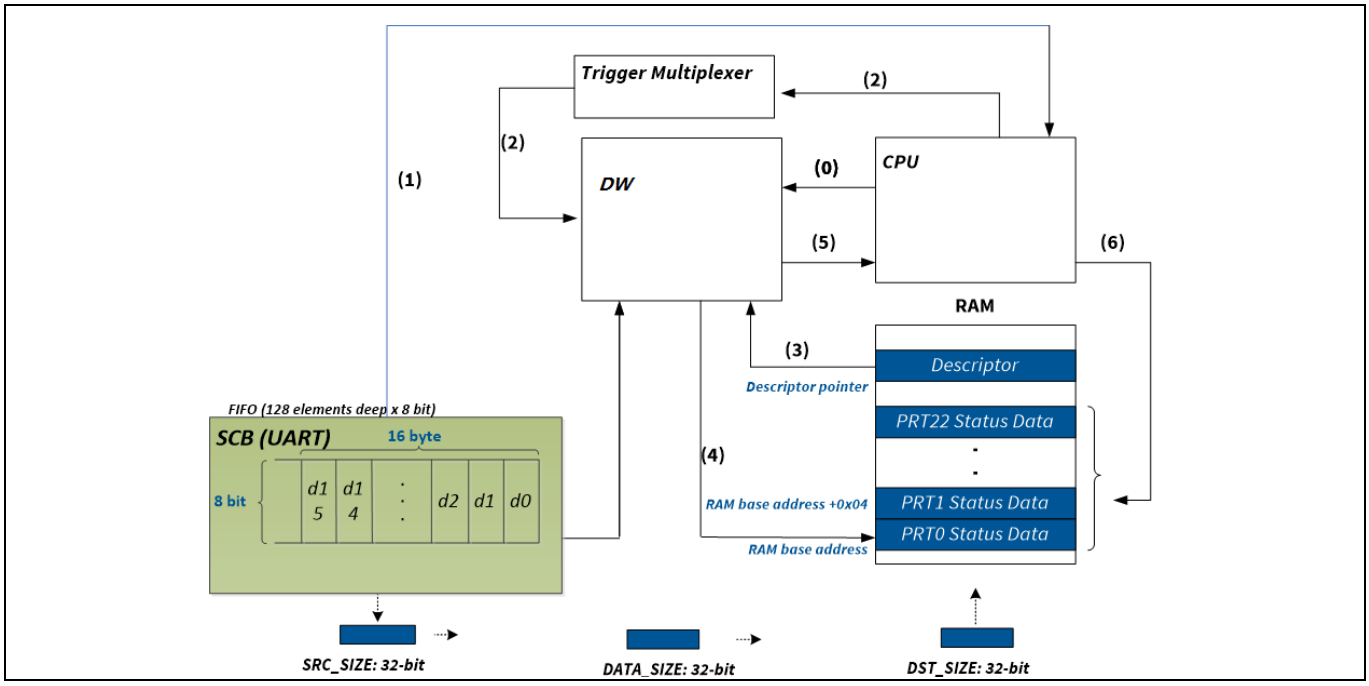


Figure 12 **Operation of 1D transfer (peripheral-to-memory)**

1. Configure DW according to section 3.2.2 Initial configuration for DW. In addition, configure the UART and trigger multiplexer.
2. UART RX interrupt occurs.
3. CPU notifies a software trigger to DW via the trigger multiplexer.
4. DW reads the descriptor from the specified area (Descriptor Pointer) when accepting the transfer.
5. DW reads the data from the source address (SCB_RX_FIFO_RD), and writes the read data to the destination address (RAM base address). After that, DW does not increment the source address and the destination address by 0x04. Then, DW reads the data from the source address (SCB_RX_FIFO_RD) and writes it to the destination address (RAM base address +0x04) again.
6. DW notifies the CPU with an interrupt when all transfers are completed.
7. The CPU accepts the interrupt, reads the port states in the RAM, and sends the received data from UART TX.

When the data is received by UART RX, repeat steps 3 to 6.

3.2.2 Initial configuration for DW

This section describes the initialization of the DW channel and descriptor of this use case.

Figure 13 shows the setting procedure for DW.

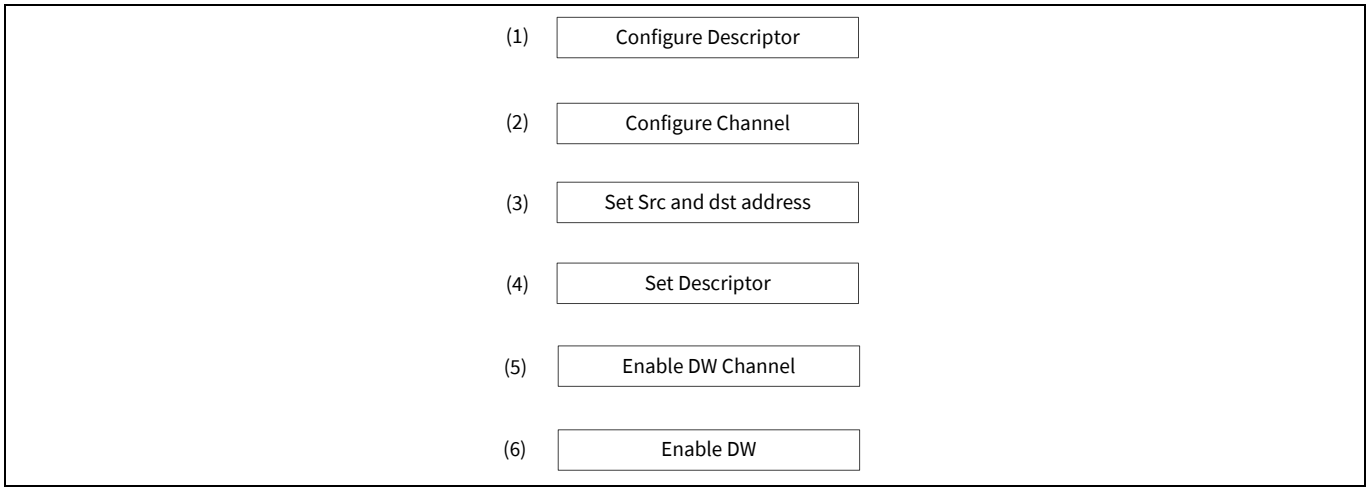


Figure 13 **Setting procedure for DW**

3.2.3 Example code

Code Listing 2. demonstrates an example program to 1D transfer (peripheral-to-memory). See the architecture TRM and application note for Clock configuration.

Figure 14 is a screengrab from ModusToolbox™ design.modus that shows the 1D Transfer (peripheral-to-memory) function settings.

Overview	
Configuration Help	Open DMA Documentation
Channel	
Trigger Input	Serial Communication Block (SCB) 3 rx_request (KIT_UART) [USED]
Trigger Output	<unassigned>
Channel Priority	3
Number of Descriptors	1
Preemptable	☐
Bufferable	☐
Select the descriptor	Descriptor_0
Descriptor	
Trigger output	Trigger on every X loop transfer completion
Interrupt type	Trigger on every X loop transfer completion
Enable Chaining	☑
Chain to descriptor	0
Channel state on completion	Enable
Trigger input type	One transfer per trigger
Trigger deactivation and retriggering	Retrigger after 4 Slow Clock cycles
Data transfer width	Word to Byte
Descriptor X loop settings	
Number of data elements to transfer	16
Source increment every cycle by	0
Destination increment every cycle by	1
Descriptor Y loop settings	
Number of X-loops to execute	1
Source increment every cycle by	1
Destination increment every cycle by	1
Advanced	
Store Config in Flash	☑

Figure 14 **Operation of 1D transfer (peripheral-to-memory)**

DW use cases

See *cyip_dw.h mtb_shared/mtb-pdl-cat1/release-v3.3.0/devices/COMPONENT_CAT1C/include/ip* for more information on the union and structure representation of registers.

Code Listing 2 1D transfer (peripheral -to- memory) example

```
#define KIT_UART_ENABLED 1U
#define KIT_UART_HW SCB3
#define KIT_UART_IRQ scb_3_interrupt_IRQn
#define TxDma_ENABLED 1U
#define TxDma_HW DW1
#define TxDma_CHANNEL 22U
#define TxDma_IRQ cpuss_interrupts_dw1_22_IRQn
#define RxDma_ENABLED 1U
#define RxDma_HW DW1
#define RxDma_CHANNEL 23U
#define RxDma_IRQ cpuss_interrupts_dw1_23_IRQn

void configure_rx_dma(uint8_t* buffer_a, uint8_t* buffer_b, cy_stc_sysint_t* int_config)
{
    cy_en_dma_status_t dma_init_status;

    /* Initialize descriptor 1 */
    dma_init_status = Cy_DMA_Descriptor_Init(&RxDma_Descriptor_0, &RxDma_Descriptor_0_config);
    if (dma_init_status!=CY_DMA_SUCCESS)
    {
        handle_error();
    }


    dma_init_status = Cy_DMA_Channel_Init(RxDma_HW, RxDma_CHANNEL, &RxDma_channelConfig);
    if (dma_init_status!=CY_DMA_SUCCESS)
    {
        handle_error();
    }

    /* Set source and destination address for descriptor 1 */
    Cy_DMA_Descriptor_SetSrcAddress(&RxDma_Descriptor_0, (uint32_t *) &KIT_UART_HW->RX_FIFO_RD);
    Cy_DMA_Descriptor_SetDstAddress(&RxDma_Descriptor_0, (uint32_t *) buffer_a);


    Cy_DMA_Channel_SetDescriptor(RxDma_HW, RxDma_CHANNEL, &RxDma_Descriptor_0);

    /* Initialize and enable interrupt from RxDma */
    Cy_SysInt_Init  (int_config, &rx_dma_complete);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Enable DMA interrupt source. */
    Cy_DMA_Channel_SetInterruptMask(RxDma_HW, RxDma_CHANNEL, CY_DMA_INTR_MASK);

    /* Enable channel and DMA block to start descriptor execution process */
    Cy_DMA_Channel_Enable(RxDma_HW, RxDma_CHANNEL);
    Cy_DMA_Enable(RxDma_HW);
}

int main(void)
{
    cy_rslt_t result;
    uint8_t rx_dma_uart_buffer_a[BUFFER_SIZE];
    cy_en_scb_uart_status_t init_status;
    cy_stc_scb_uart_context_t KIT_UART_context;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }
    cy_stc_sysint_t KIT_UART_INT_cfg =
    {
        .intrSrc = ( (NvicMux3_IRQn << 16) | KIT_UART_IRQ ),
        .intrPriority = 7u,
    };

    cy_stc_sysint_t RX_DMA_INT_cfg =
    {
        .intrSrc      = ( (NvicMux3_IRQn << 16) | (IRQn_Type)RxDma_IRQ ),
        .intrPriority = 6u,
    };

    cy_stc_sysint_t TX_DMA_INT_cfg =
    {
        .intrSrc      = ( (NvicMux2_IRQn << 16) | (IRQn_Type)TxDma_IRQ ),
```

DW use cases

Code Listing 2 1D transfer (peripheral -to- memory) example

```
        .intrPriority = 6u,
    };

    /* Configure DMA Rx and Tx channels for operation */
    configure_rx_dma(rx_dma_uart_buffer_a, rx_dma_uart_buffer_b, &RX_DMA_INT_cfg);
    configure_tx_dma(rx_dma_uart_buffer_a, &TX_DMA_INT_cfg);

    /* Initialize and enable interrupt from UART. The UART interrupt sources
    *  are enabled in the Component GUI */
    Cy_SysInt_Init(&KIT_UART_INT_cfg, &Isr_UART);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Start UART operation */
    init_status = Cy_SCB_UART_Init(KIT_UART_HW, &KIT_UART_config, &KIT_UART_context);
    if (init_status!=CY_SCB_UART_SUCCESS)
    {
        handle_error();
    }
    Cy_SCB_UART_Enable(KIT_UART_HW);

    while(1)
    {
        /* Handle RxDma complete */
        if (rx_dma_done==1)
        {
            /* Set source RX Buffer A as source for TxDMA */
            Cy_DMA_Descriptor_SetSrcAddress(&TxDma_Descriptor_0, (uint32_t *) rx_dma_uart_buffer_a);

            Cy_DMA_Channel_SetDescriptor(TxDma_HW, TxDma_CHANNEL, &TxDma_Descriptor_0);
            Cy_DMA_Channel_Enable(TxDma_HW, TxDma_CHANNEL);
         rx_dma_done = 0;
        }
    }

}
```

3.3 Descriptor chaining

3.3.1 Overview

This is an example of descriptor chaining by storing the pointer of the next descriptor (DESCR_NEXT_PTR) in the current descriptor.

This example uses two descriptors: Descriptor 0 and Descriptor 1. Both descriptors are for a 1D transfer that transfers UART RX data to the memory (RAM) with UART RX interrupt. However, Descriptor 0 has RAM base address 0 as the destination address, and Descriptor 1 has RAM base address 1 as the destination address.

DW transfers UART RX data to the memory (RAM) with UART RX interrupt. DW notifies the interrupt to the CPU when the transfer is completed. The pointer to the next descriptor (DESCR_NEXT_PTR) in the descriptor (Descriptor 0) of this transfer is set to a pointer to another descriptor (Descriptor 1). As a result, DW can start the Descriptor 1 transfer when Descriptor 0 transfer completes. Descriptor 0 and Descriptor 1 have different destination addresses.

DW allows data of the same port address to be transferred to different RAM addresses. In other words, it can have a double buffer. In this case, Descriptor 0 and Descriptor 1 are chained with each other, and Descriptor 0 and Descriptor 1 have a circular list.

Figure 15 shows 1D transfer with descriptor chaining.

DW use cases

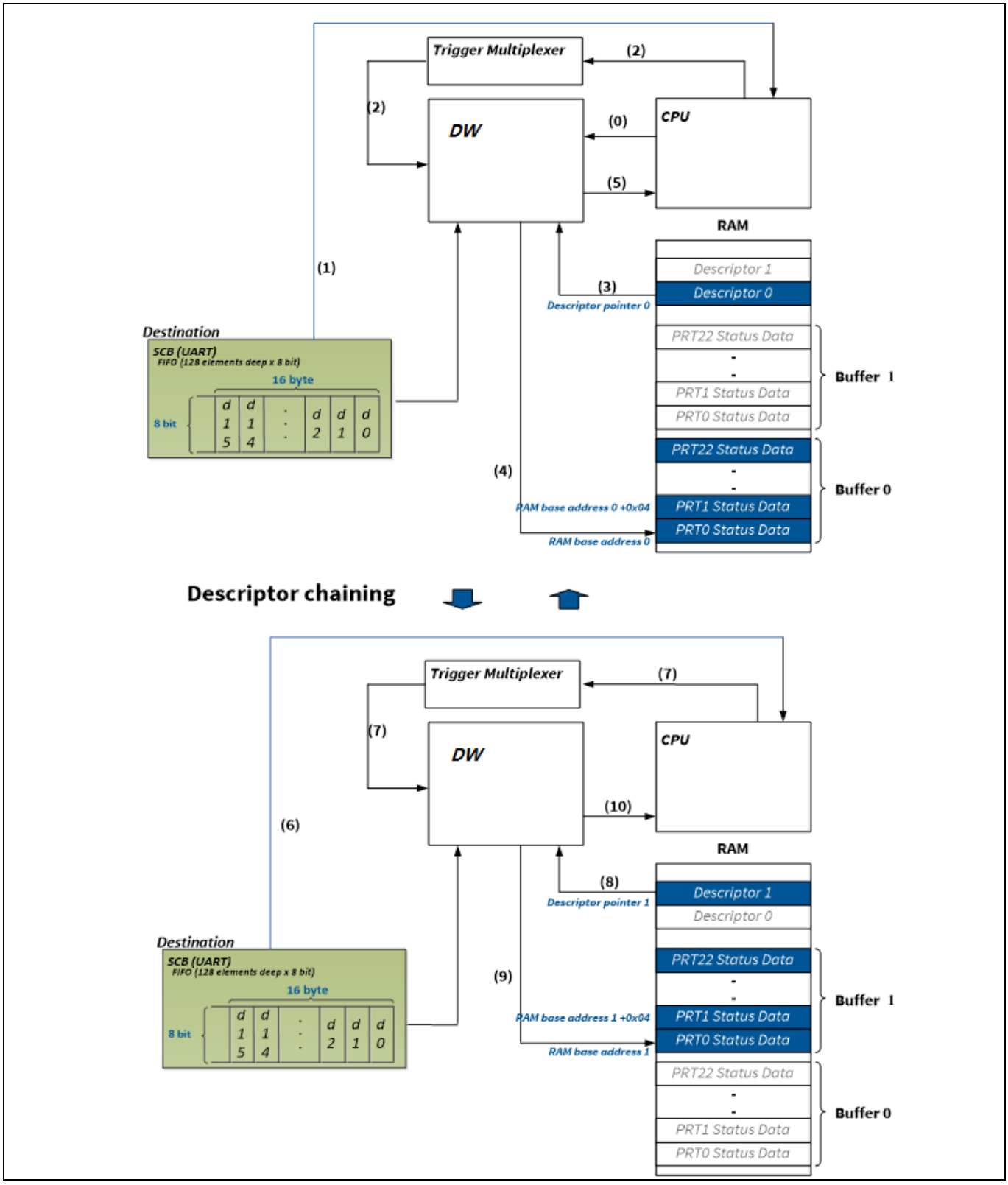


Figure 15 **1D transfer with descriptor chaining**

1. Configure DW according to the usage example. In addition, configure UART and trigger multiplexer.
2. UART RX Interrupt occurs.
3. CPU notifies a software trigger to DW via the trigger multiplexer.

DW use cases

4. DW reads the descriptor from the specified area (Descriptor Pointer 0) when activating the next pending transfer.
5. DW reads the data from the source address (SCB_RX_FIFO_RD), and writes the read data to the destination address (RAM base address 0). After that, DW does not increment the source address and the destination address by 0x04. Then, DW reads the data from the source address (SCB_RX_FIFO_RD) and writes it to the destination address (RAM base address 0 +0x04) again.
6. DW notifies the CPU with an interrupt when all transfers are completed.
7. UART RX Interrupt occurs again.
8. CPU notifies a software trigger to DW via the trigger multiplexer again.
9. DW reads the descriptor from the specified area (Descriptor Pointer 1) when activating the next pending transfer.
10. DW reads the data from the source address (SCB_RX_FIFO_RD), and writes the read data to the destination address (RAM base address 0). After that, DW does not increment the source address and the destination address by 0x04. Then, DW reads the data from the source address (SCB_RX_FIFO_RD) and writes it to the destination address (RAM base address 0 +0x04) again.
11. DW notifies the CPU through an interrupt when all transfers are completed. Descriptor 1 chains to Descriptor 0. Therefore, when the UART RX Interrupt again, steps from (3) are repeated.

3.3.2 Initial configuration

This section describes the initialization of the DW channel and descriptor of this use case.

Figure 16 shows the setting procedure for DW.

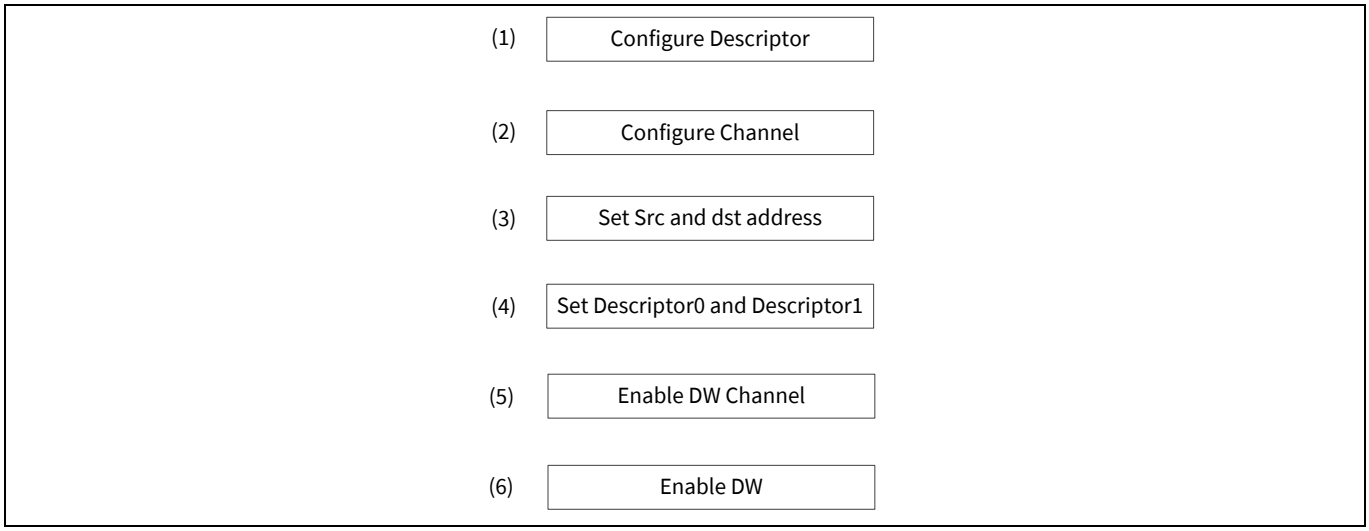


Figure 16 **Setting procedure for DW**

3.3.3 Example code

Figure 17 from ModusToolbox™ design.modus, shows the UART RX DMA for DMA Descriptor chaining function. You can switch the descriptor.

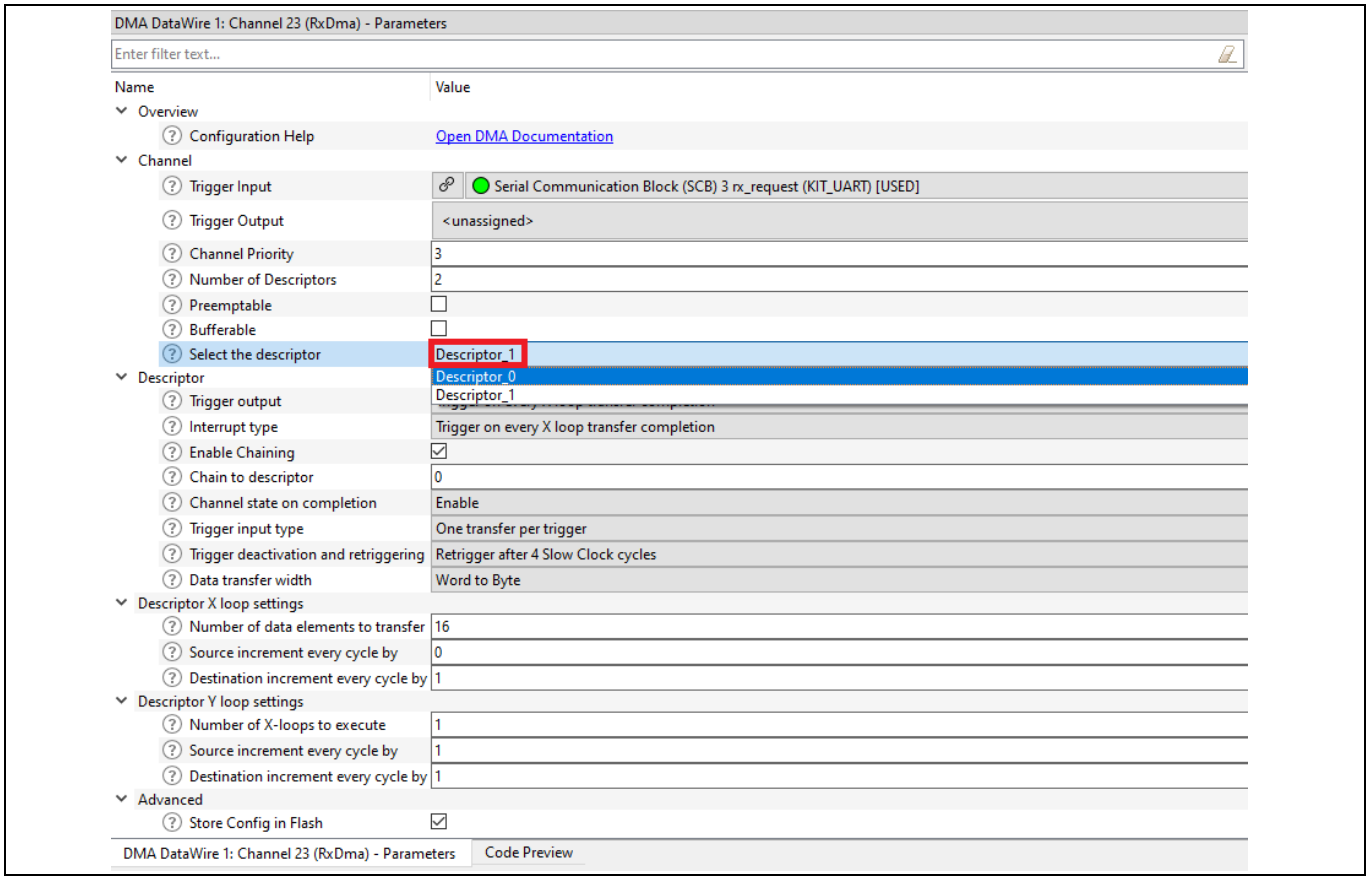


Figure 17 **Operation of descriptor chaining**

Code Listing 3 demonstrates an example program to Descriptor Chaining. See the architecture TRM and application note for UART configuration.

Code Listing 3 Descriptor chaining example

```
#define KIT_UART_ENABLED 1U
#define KIT_UART_HW SCB3
#define KIT_UART_IRQ scb_3_interrupt_IRQn
#define TxDma_ENABLED 1U
#define TxDma_HW DW1
#define TxDma_CHANNEL 22U
#define TxDma_IRQ cpuss_interrupts_dw1_22_IRQn
#define RxDma_ENABLED 1U
#define RxDma_HW DW1
#define RxDma_CHANNEL 23U
#define RxDma_IRQ cpuss_interrupts_dw1_23_IRQn


void configure_rx_dma(uint8_t* buffer_a, uint8_t* buffer_b, cy_stc_sysint_t* int_config)
{
    cy_en_dma_status_t dma_init_status;

    /* Initialize descriptor 1 */
    dma_init_status = Cy_DMA_Descriptor_Init(&RxDma_Descriptor_0, &RxDma_Descriptor_0_config);
    if (dma_init_status!=CY_DMA_SUCCESS)
    {
        handle_error();
    }
```

DW use cases

Code Listing 3 Descriptor chaining example

```
    dma_init_status = Cy_DMA_Channel_Init(RxDma_HW, RxDma_CHANNEL, &RxDma_channelConfig);
    if (dma_init_status!=CY_DMA_SUCCESS)
    {
        handle_error();
    }

    /* Set source and destination address for descriptor 1 */
    Cy_DMA_Descriptor_SetSrcAddress(&RxDma_Descriptor_0, (uint32_t *) &KIT_UART_HW->RX_FIFO_RD);
    Cy_DMA_Descriptor_SetDstAddress(&RxDma_Descriptor_0, (uint32_t *) buffer_a);


    Cy_DMA_Channel_SetDescriptor(RxDma_HW, RxDma_CHANNEL, &RxDma_Descriptor_0);

    /* Initialize and enable interrupt from RxDma */
    Cy_SysInt_Init  (int_config, &rx_dma_complete);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Enable DMA interrupt source. */
    Cy_DMA_Channel_SetInterruptMask(RxDma_HW, RxDma_CHANNEL, CY_DMA_INTR_MASK);

    /* Enable channel and DMA block to start descriptor execution process */
    Cy_DMA_Channel_Enable(RxDma_HW, RxDma_CHANNEL);
    Cy_DMA_Enable(RxDma_HW);
}

int main(void)
{
    cy_rslt_t result;
    uint8_t rx_dma_uart_buffer_a[BUFFER_SIZE];
    uint8_t rx_dma_uart_buffer_b[BUFFER_SIZE];
    cy_en_scb_uart_status_t init_status;
    cy_stc_scb_uart_context_t KIT_UART_context;
    uint32_t active_descr = DMA_DESCR0; /* flag to control which descriptor to use */

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    cy_stc_sysint_t KIT_UART_INT_cfg =
    {
        .intrSrc = ( (NvicMux3_IRQn << 16) | KIT_UART_IRQ ),
        .intrPriority = 7u,
    };

    cy_stc_sysint_t RX_DMA_INT_cfg =
    {
        .intrSrc      = ( (NvicMux3_IRQn << 16) | (IRQn_Type)RxDma_IRQ ),
        .intrPriority = 6u,
    };

    cy_stc_sysint_t TX_DMA_INT_cfg =
    {
        .intrSrc      = ( (NvicMux2_IRQn << 16) | (IRQn_Type)TxDma_IRQ ),
        .intrPriority = 6u,
    };

    /* Configure DMA Rx and Tx channels for operation */
    configure_rx_dma(rx_dma_uart_buffer_a, rx_dma_uart_buffer_b, &RX_DMA_INT_cfg);
    configure_tx_dma(rx_dma_uart_buffer_a, &TX_DMA_INT_cfg);

    /* Initialize and enable interrupt from UART. The UART interrupt sources
    *  are enabled in the Component GUI */
    Cy_SysInt_Init(&KIT_UART_INT_cfg, &Isr_UART);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Start UART operation */
    init_status = Cy_SCB_UART_Init(KIT_UART_HW, &KIT_UART_config, &KIT_UART_context);
    if (init_status!=CY_SCB_UART_SUCCESS)
    {
        handle_error();
    }
    Cy_SCB_UART_Enable(KIT_UART_HW);


    /* Initialize flags */
    rx_dma_error = 0;
    tx_dma_error = 0;
    uart_error = 0;
```

DW use cases

Code Listing 3 Descriptor chaining example

```
    rx_dma_done = 0;

    __enable_irq();

    while (1)
    {
        /* Indicate status if RxDma error or TxDma error or UART error occurs */
        if ((rx_dma_error==1) | (tx_dma_error==1) | (uart_error==1))
        {
            handle_error();
        }

        /* Handle RxDma complete */
        if (rx_dma_done==1)
        {
            /* Ping Pong between rx_dma_uart_buffer_a and rx_dma_uart_buffer_b */
            /* Ping Pong buffers give firmware time to pull the data out of one or the other buffer */
            if (DMA_DESCR0 == active_descr)
            {
                /* Set source RX Buffer A as source for TxDMA */
                Cy_DMA_Descriptor_SetSrcAddress(&TxDma_Descriptor_0, (uint32_t *) rx_dma_uart_buffer_a);
                active_descr = DMA_DESCR1;
            }
            else
            {
                /* Set source RX Buffer B as source for TxDMA */
                Cy_DMA_Descriptor_SetSrcAddress(&TxDma_Descriptor_0, (uint32_t *) rx_dma_uart_buffer_b);
                active_descr = DMA_DESCR0;
            }

            Cy_DMA_Channel_SetDescriptor(TxDma_HW, TxDma_CHANNEL, &TxDma_Descriptor_0);
            Cy_DMA_Channel_Enable(TxDma_HW, TxDma_CHANNEL);
            rx_dma_done = 0;
        }
    }
}
```

3.4 2D transfer (peripheral-to-memory)

3.4.1 Overview

This is an example of transferring the transmit data from RX-FIFO to RAM by DMA with 2D transfer when using the SCB UART mode. In this case, the UART uses an 8-bit data frame and starts the transmission when UART receives data. The number of bytes to be transmitted is 32, and the RX FIFO setting is 1 element deep with 8-bit data elements. See the “Serial Communications Block (SCB)” chapter of the architecture TRM for more details.

The UART generates a receive interrupt after it receives the FIFO data, and the DW starts transferring the KIT_UART_HW->RX_FIFO_RD FIFO data to RAM. After the data transfer is complete, the DW generates an interrupt.

Figure 18 shows the data transfer

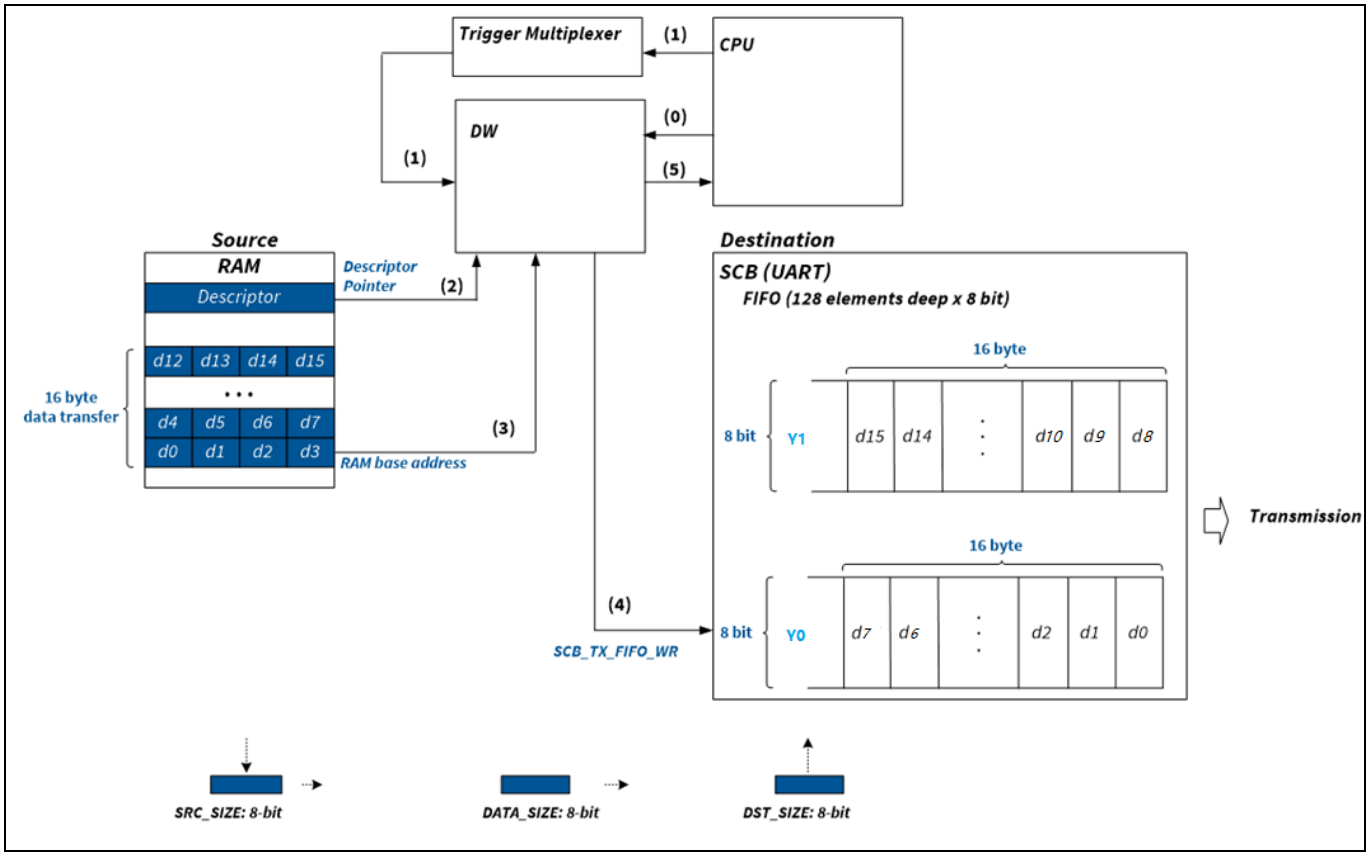


Figure 18 **2D transfer (peripheral-to-memory)**

1. Configure DW according to section 3.4.2. In addition, configure the UART and trigger multiplexer.
2. UART RX interrupt occurs.
3. CPU notifies a software trigger to DW via the trigger multiplexer.
4. DW reads the descriptor from the specified area (Descriptor Pointer) when accepting the transfer.
5. DW reads the data from the source address (SCB_RX_FIFO_RD), and writes the read data to the destination address (RAM base address). After that, DW does not increment the source address and the destination address by 0x04.

6. DW reads the data from the source address (SCB_RX_FIFO_RD) and writes it to the destination address (RAM base address +0x04) again. After receiving low 8 bytes data, the Y count will increment, and the system will transfer high 8 bytes data to destination address.
7. DW notifies the CPU with an interrupt when all transfers are completed.
8. The CPU accepts the interrupt and reads the port states in the RAM, and sends the received data from UART TX.
9. When the device receives data from UART RX again, steps 3 to 6 are repeated.

3.4.2 Initial configuration

This section describes the initialization of the DW channel and descriptor of this use case.

Figure 19 shows the setting procedure for DW.

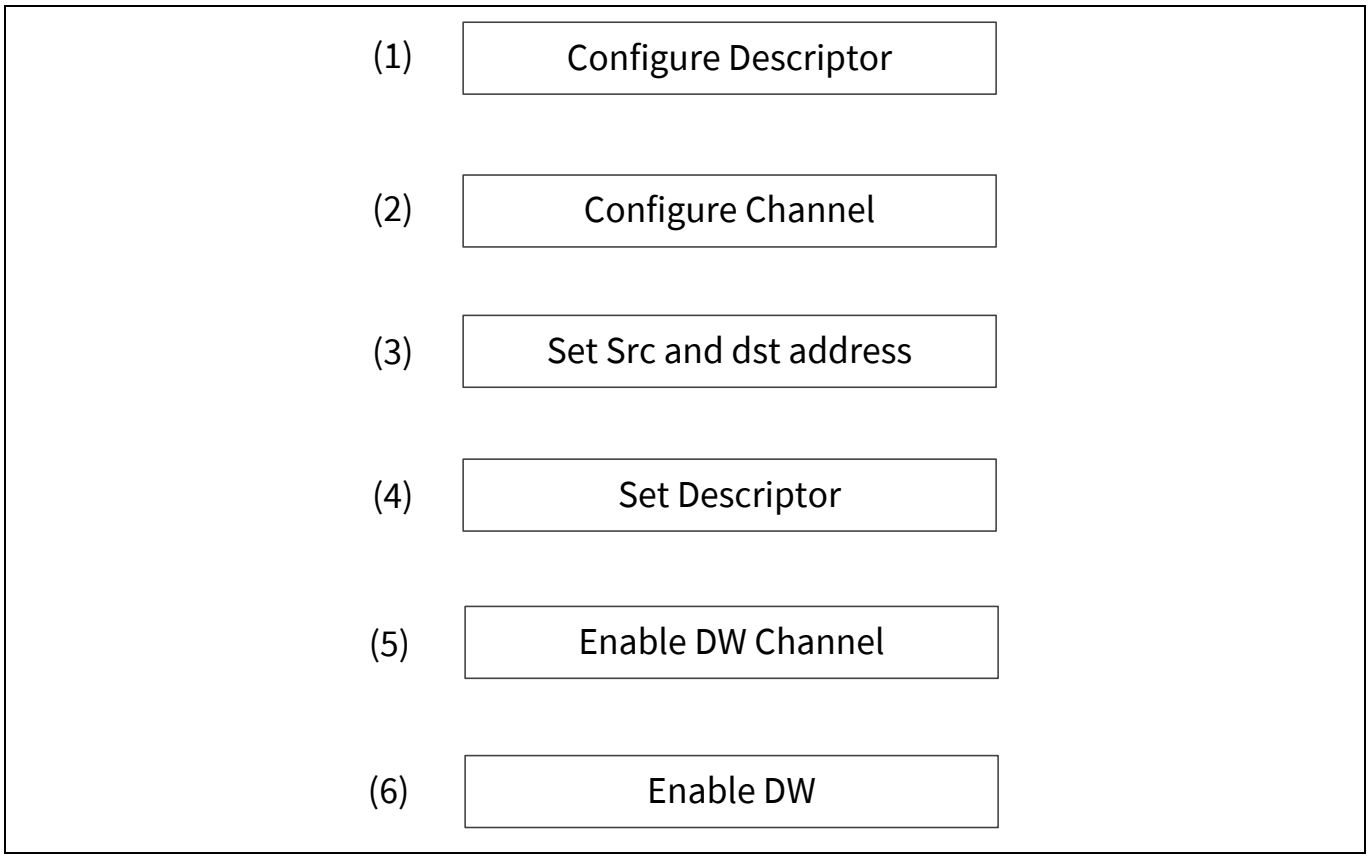


Figure 19 **Setting procedure for DW**

3.4.3 Example code

Figure 20 shows the UART RX DMA for DMA 2D transfer.

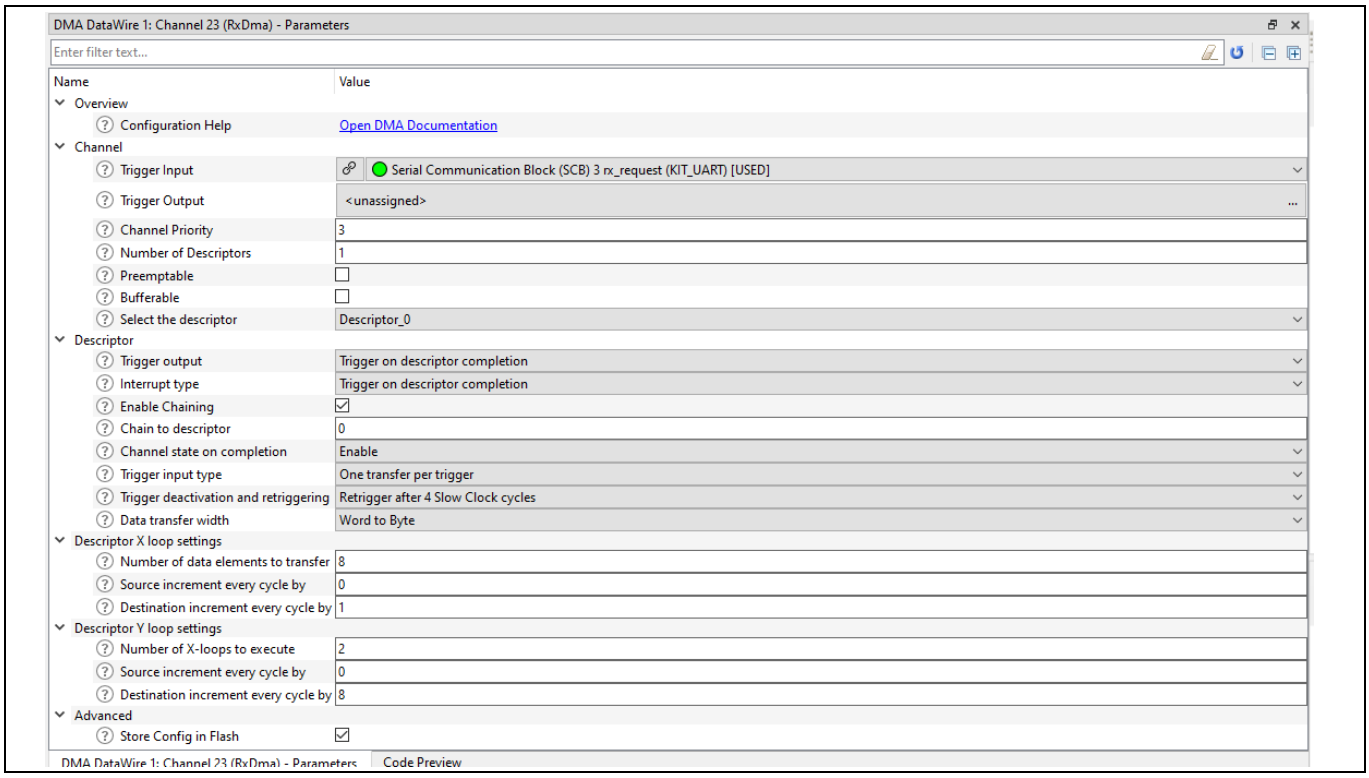


Figure 20 **Operation of 2D transfer**

Code Listing 4 demonstrates an example program to 2D transfer (peripheral-to-memory). See the architecture TRM and application note for UART and Clock configuration.

Code Listing 4 2D transfer (peripheral-to-memory) example

```
#define KIT_UART_ENABLED 1U
#define KIT_UART_HW SCB3
#define KIT_UART_IRQ scb_3_interrupt_IRQn
#define TxDma_ENABLED 1U
#define TxDma_HW DW1
#define TxDma_CHANNEL 22U
#define TxDma_IRQ cpuss_interrupts_dw1_22_IRQn
#define RxDma_ENABLED 1U
#define RxDma_HW DW1
#define RxDma_CHANNEL 23U
#define RxDma_IRQ cpuss_interrupts_dw1_23_IRQn


void configure_rx_dma(uint8_t* buffer_a, uint8_t* buffer_b, cy_stc_sysint_t* int_config)
{
    cy_en_dma_status_t dma_init_status;

    /* Initialize descriptor 1 */
    dma_init_status = Cy_DMA_Descriptor_Init(&RxDma_Descriptor_0, &RxDma_Descriptor_0_config);
    if (dma_init_status!=CY_DMA_SUCCESS)
    {
        handle_error();
    }


    dma_init_status = Cy_DMA_Channel_Init(RxDma_HW, RxDma_CHANNEL, &RxDma_channelConfig);
    if (dma_init_status!=CY_DMA_SUCCESS)
    {
        handle_error();
    }
```

DW use cases

Code Listing 4 2D transfer (peripheral-to-memory) example

```
    /* Set source and destination address for descriptor 1 */
    Cy_DMA_Descriptor_SetSrcAddress(&RxDma_Descriptor_0, (uint32_t *) &KIT_UART_HW->RX_FIFO_RD);
    Cy_DMA_Descriptor_SetDstAddress(&RxDma_Descriptor_0, (uint32_t *) buffer_a);


    Cy_DMA_Channel_SetDescriptor(RxDma_HW, RxDma_CHANNEL, &RxDma_Descriptor_0);

    /* Initialize and enable interrupt from RxDma */
    Cy_SysInt_Init  (int_config, &rx_dma_complete);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Enable DMA interrupt source. */
    Cy_DMA_Channel_SetInterruptMask(RxDma_HW, RxDma_CHANNEL, CY_DMA_INTR_MASK);

    /* Enable channel and DMA block to start descriptor execution process */
    Cy_DMA_Channel_Enable(RxDma_HW, RxDma_CHANNEL);
    Cy_DMA_Enable(RxDma_HW);
}

int main(void)
{
    cy_rslt_t result;
    uint8_t rx_dma_uart_buffer_a[BUFFER_SIZE];
    cy_en_scb_uart_status_t init_status;
    cy_stc_scb_uart_context_t KIT_UART_context;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }
    cy_stc_sysint_t KIT_UART_INT_cfg =
    {
        .intrSrc = ( (NvicMux3_IRQn << 16) | KIT_UART_IRQ ),
        .intrPriority = 7u,
    };

    cy_stc_sysint_t RX_DMA_INT_cfg =
    {
        .intrSrc      = ( (NvicMux3_IRQn << 16) | (IRQn_Type)RxDma_IRQ ),
        .intrPriority = 6u,
    };

    cy_stc_sysint_t TX_DMA_INT_cfg =
    {
        .intrSrc      = ( (NvicMux2_IRQn << 16) | (IRQn_Type)TxDma_IRQ ),
        .intrPriority = 6u,
    };

    /* Configure DMA Rx and Tx channels for operation */
    configure_rx_dma(rx_dma_uart_buffer_a, rx_dma_uart_buffer_b, &RX_DMA_INT_cfg);
    configure_tx_dma(rx_dma_uart_buffer_a, &TX_DMA_INT_cfg);

    /* Initialize and enable interrupt from UART. The UART interrupt sources
    *  are enabled in the Component GUI */
    Cy_SysInt_Init(&KIT_UART_INT_cfg, &Isr_UART);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Start UART operation */
    init_status = Cy_SCB_UART_Init(KIT_UART_HW, &KIT_UART_config, &KIT_UART_context);
    if (init_status!=CY_SCB_UART_SUCCESS)
    {
        handle_error();
    }
    Cy_SCB_UART_Enable(KIT_UART_HW);

    while(1)
    {
        /* Handle RxDma complete */
        if (rx_dma_done==1)
        {
            /* Set source RX Buffer A as source for TxDMA */
            Cy_DMA_Descriptor_SetSrcAddress(&TxDma_Descriptor_0, (uint32_t *) rx_dma_uart_buffer_a);

            Cy_DMA_Channel_SetDescriptor(TxDma_HW, TxDma_CHANNEL, &TxDma_Descriptor_0);
            Cy_DMA_Channel_Enable(TxDma_HW, TxDma_CHANNEL);
         rx_dma_done = 0;
        }
    }

}
```

3.5 CRC transfer

3.5.1 Overview

This section describes an example of CRC transfer. CRC transfer is a DW-specific descriptor type. CRC transfer calculates the CRC in the area that is specified by the source address and size, and transfers the result to the destination address.

This is an example of program code validation in the flash with CRC transfer. The CPU calculates the CRC of the program code area with DW before program execution. CRC calculation is performed by using CRC32. When the result of CRC calculation matches the expected value, the CPU starts the execution of the program. In this example, note that it is necessary to prepare for the expected value of the program code.

See the architecture TRM for details of CRC parameters that can be set by DW.

Figure 21 shows a use case of a CRC transfer.

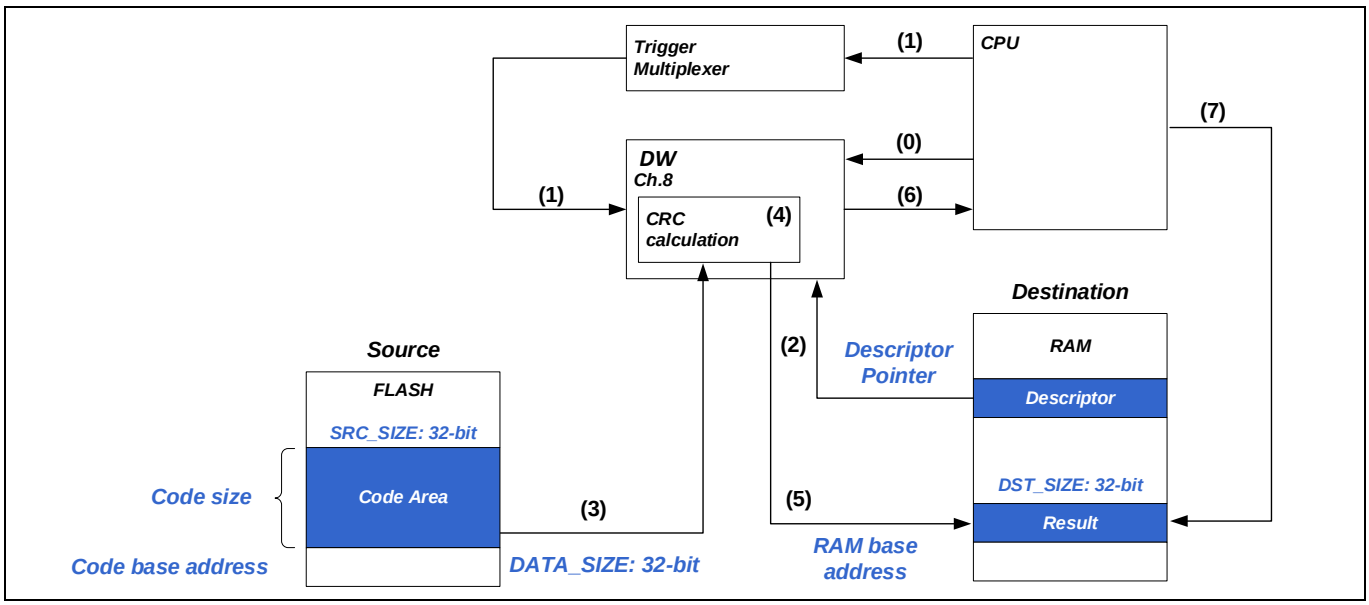


Figure 21 **Use case of CRC transfer**

1. Configure DW according to the usage example.
2. CPU notifies a software trigger to DW via the trigger multiplexer.
3. DW reads the descriptor from the specified area (Descriptor Pointer) when accepting the transfer.
4. DW reads the data from the source address (code base address).
5. DW inputs the read data to the CRC calculator, and reads the data again after incrementing the address. DW repeats (4) until it reaches the area specified by the transfer size (code size).
6. When CRC calculation of the specified area is completed, the result is transferred to the destination address (RAM base address).
7. DW notifies an interrupt to the CPU.

When the CPU accepts an interrupt, it compares the result with the expected value. When it matches, CPU starts program execution. If it does not match, CPU transfers to safe operation mode.

3.5.2 Initial configuration

This section describes the initialization of the DW channel and descriptor of this use case.

Figure 22 shows the setting procedure for DW.

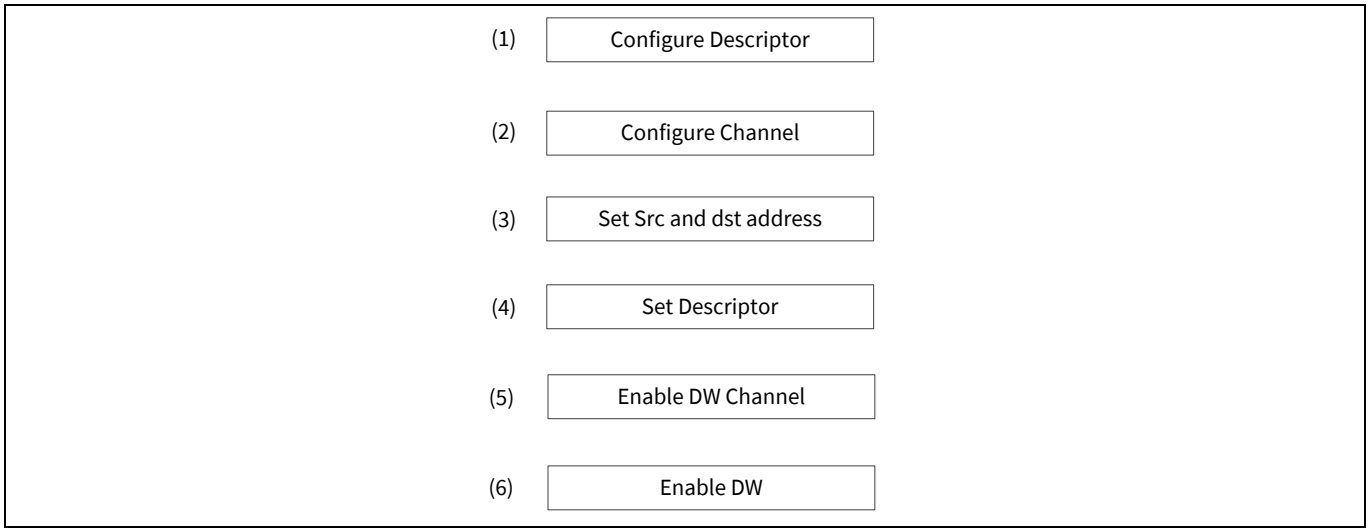


Figure 22 **Setting procedure for DW**

3.5.3 Example code

Code Listing 5 demonstrates an example program to DMA CRC..

Code Listing 5 CRC transfer example

```
#define BUFFER_SIZE (5ul)
#define DW_CHANNEL  (7)
#define DW_CH_INTRRUPT  (cpuss_interrupts_dw1_7_IRQn)
#define DW_SW_TRIG TRIG_OUT_MUX_2_PDMA1_TR_IN7

#define DMA_IRQ_NUM          NvicMux3_IRQn
#define DMA_INTR_NUM         ((DMA_IRQ_NUM << 16u) | (IRQn_Type)DW_CH_INTRRUPT)
#define DMA_INTR_PRIORITY    (6u)

#define TEST_PATTERN_NUM (sizeof(stcCrcConfig)/sizeof(cy_stc_dma_crc_config_t))

static bool isComplete = false;
static uint32_t CRCEmulate32bit(stc_crc32_emu_config_t * crc_params);
uint32_t generatedCrcResult, emulatedCrcResult;

void DW1_Ch_IntHandler(void);
static cy_stc_dma_descriptor_t stcDescr =
{
    .ctl = 0UL,
    .src = 0UL,
    .dst = 0UL,
    .xCtl = 0UL,
    .yCtl = 0UL,
    .nextPtr = 0UL,
};
const  uint8_t   au8SrcBuffer[] = {0x12u, 0x34u, 0x56u, 0x78u, 0x9au};
static uint32_t crcLfsrResulttDst = 0ul;

const cy_stc_dma_crc_config_t stcCrcConfig[] =
{
//  { data_reverse, data_xor, rem_reverse, reminderXor,   polynomial,  lfsrInitVal }
    {          1ul,         0u,         1ul, 0xFFFFFFFFul, 0x04c11db7ul, 0xFFFFFFFFul },
    {          0ul,         0u,         1ul, 0xFFFFFFFFul, 0x04c11db7ul, 0xFFFFFFFFul },
    {          1ul,         0u,         0ul, 0xFFFFFFFFul, 0x04c11db7ul, 0xFFFFFFFFul },
    {          1ul,       0x5Au,        1ul, 0xFFFFFFFFul, 0x04c11db7ul, 0xFFFFFFFFul },
    {          1ul,         0u,         1ul, 0xFFFFFFFFul, 0xABCDEF01ul, 0xFFFFFFFFul },
```

DW use cases

Code Listing 5 CRC transfer example

```
    {          1ul,         0u,       1ul, 0xFFFFFFFFul, 0x04c11db7ul, 0x12345678ul },
    {          1ul,         0u,       1ul, 0x5A5A1234ul, 0x04c11db7ul, 0xFFFFFFFFul },

};

const cy_stc_dma_channel_config_t chnlConfig =
{
    .descriptor = &stcDescr,
    .preemptable = false,
    .priority = 3,
    .enable = true,
    .bufferable = false,
};

static  cy_stc_dma_descriptor_config_t stcDmaDescrConfig =
{
   .retrigger = CY_DMA_RETRIG_4CYC,
   .interruptType = CY_DMA_X_LOOP,
   .triggerOutType = CY_DMA_X_LOOP,
   .channelState = CY_DMA_CHANNEL_ENABLED,
   .triggerInType = CY_DMA_DESCR,
   .dataSize = CY_DMA_BYTE,
   .srcTransferSize = CY_DMA_TRANSFER_SIZE_DATA,
   .dstTransferSize = CY_DMA_TRANSFER_SIZE_WORD,
   .descriptorType = CY_DMA_CRC_TRANSFER,
   .srcAddress = (void*)au8SrcBuffer,
   .dstAddress = NULL,
   .srcXincrement = 1,
   .dstXincrement = 1,
   .xCount = 5,
   .srcYincrement = 0,
   .dstYincrement = 0,
   .yCount = 0,
   .nextDescriptor = &stcDescr,
};

const cy_stc_sysint_t stc_sysint_irq_cfg =
{
    .intrSrc  = DMA_INTR_NUM,
    .intrPriority  = DMA_INTR_PRIORITY,
};
int main(void)
{
   cy_rslt_t result;

    SCB_DisableDCache(); // Disables D cache because DMA also reads descriptor in the SRAM.

    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /*Initialize the User LED*/
    Cy_GPIO_Pin_Init(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN, &CYBSP_USER_LED1_config );

    Cy_DMA_Descriptor_Init(&stcDescr,&stcDmaDescrConfig);
    Cy_DMA_Channel_SetDescriptor(DW1, DW_CHANNEL, &stcDescr);
    Cy_DMA_Channel_Init(DW1, DW_CHANNEL, &chnlConfig);
    Cy_DMA_Enable(DW1);

    /* Interrupt Initialization */
    Cy_SysInt_Init(&stc_sysint_irq_cfg, &DW1_Ch_IntHandler);
    NVIC_ClearPendingIRQ(DMA_IRQ_NUM);
    NVIC_EnableIRQ((IRQn_Type)DMA_IRQ_NUM);

    Cy_DMA_Channel_SetInterruptMask(DW1, DW_CHANNEL, CY_DMA_INTR_MASK);

    __enable_irq();

    for(uint32_t i = 1; i < TEST_PATTERN_NUM; i++)
    {
         Cy_DMA_Crc_Init(DW1, &stcCrcConfig[i]);

        stc_crc32_emu_config_t crcParams = { 0ul };
        crcParams.indata       = au8SrcBuffer;
        crcParams.size         = (uint32_t)sizeof(au8SrcBuffer);
        crcParams.poly.u32     = (uint32_t)stcCrcConfig[i].polynomial;
        crcParams.data_xor     = (uint8_t)stcCrcConfig[i].dataXor;
        crcParams.rem_xor.u32  = (uint32_t)stcCrcConfig[i].reminderXor;
        crcParams.option.ROBIT = (bool)(stcCrcConfig[i].reminderReverse == 0) ? CRC_MSB_FIRST: CRC_LSB_FIRST;
        crcParams.option.RIBIT = (bool)(stcCrcConfig[i].dataReverse == 0) ? CRC_MSB_FIRST: CRC_LSB_FIRST;
         SetCrcSeed(stcCrcConfig[i].lfsrInitVal);
```

DW use cases

Code Listing 5 CRC transfer example

```
        for(uint32_t j = 0; j < 5; j++) // try 5 times with same parameter
        {
            /*Trigger DMA by SW */
            isComplete = false;
            Cy_DMA_Channel_Enable(DW1, DW_CHANNEL);
            Cy_TrigMux_SwTrigger(DW_SW_TRIG, CY_TRIGGER_TWO_CYCLES);
            // wait for DMA completion
            while(isComplete == false);

            /* Check CRC result */
            //uint32_t generatedCrcResult = Cy_DMA_GetCrcRemainderResult(DW1);
            generatedCrcResult = DW1->CRC_REM_RESULT;

            /* Emulate generating CRC */
            emulatedCrcResult = CRCEmulate32bit(&crcParams);

            if(generatedCrcResult != emulatedCrcResult)
            {
                 while(1)
                 {
                         //If data is incorrect, infinite loop
                 }
            }
        }
    }

    for(;;)
    {
         Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN);
         Cy_SysLib_Delay(1000u);
    }
}

void DW1_Ch_IntHandler(void)
{
    uint32_t masked;

    masked = Cy_DMA_Channel_GetStatus(DW1, DW_CHANNEL);
    if ((masked & CY_DMA_INTR_CAUSE_COMPLETION) != 0ul)
    {
        /* Clear Complete DMA interrupt flag */
         Cy_DMA_Channel_ClearInterrupt(DW1, DW_CHANNEL);

        /* Mark the transmission as complete */
        isComplete = true;
    }
    else
    {
        CY_ASSERT(false);
    }
}
```

4 DMAC use case

This section describes how to use DMAC using the peripheral driver library (PDL). The code snippets in this application note are part of PDL.

PDL has a configuration part and a driver part. The configuration part mainly configures the parameter values for the desired operation. The driver part configures each register based on the parameter values in the configuration part. You can configure the configuration part according to your system.

4.1 Memory-to-memory (memory copy)

This section describes an example of memory copy. Memory copy is an DMAC-specific descriptor type. This is a special 1D transfer. Memory copy transfer data from the area specified by the source address and size to the destination address.

This is an example of data transfer from flash to RAM. This descriptor type is useful for copying the program code for RAM execution and copying the vector table. In the memory copy example, consecutive flash memory areas are transferred to RAM.

Figure 23 shows a use case of memory-to-memory transfer using memory copy.

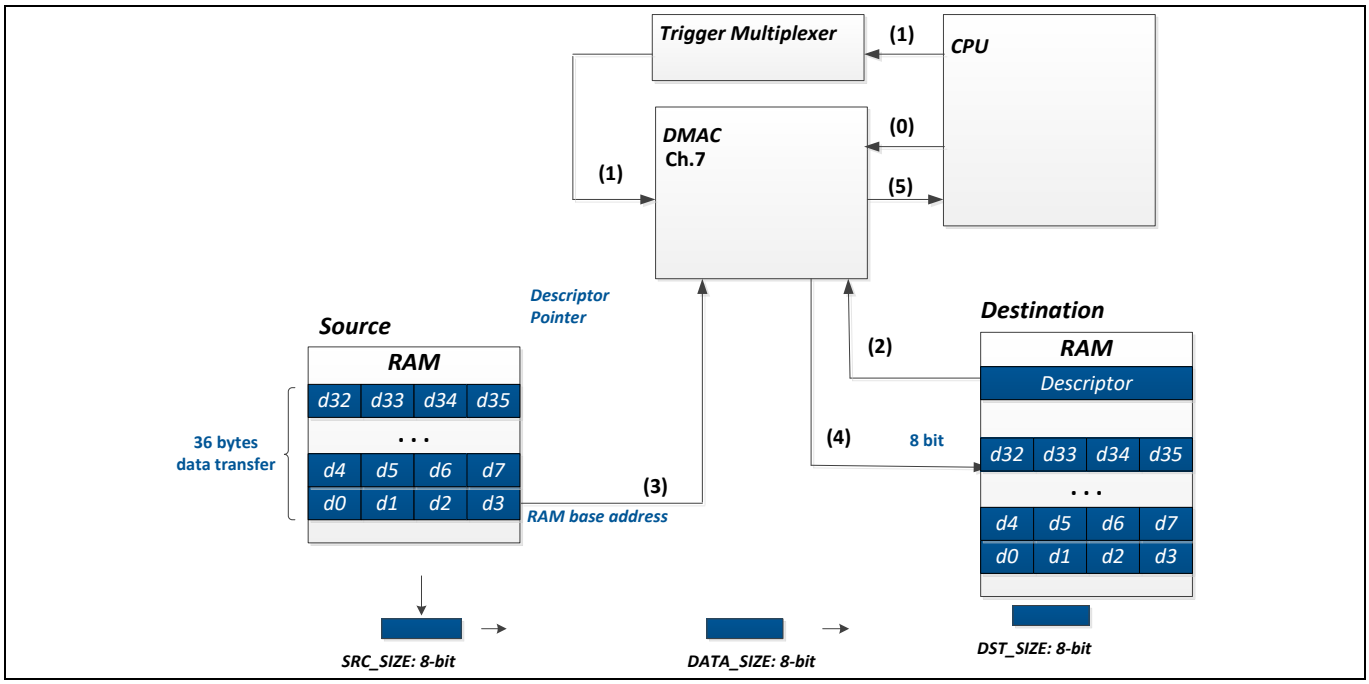


Figure 23 **Use case of memory-to-memory using memory copy**

1. Set DMAC according to the usage example setting.
2. CPU notifies a software trigger to DMAC via the trigger multiplexer.
3. DMAC reads the descriptor from the specified area (Descriptor Pointer) when accepting the transfer.
4. DMAC reads the data from the source address (code base address).
5. DMAC writes the read data to the destination address (RAM base address). After that, increment the source address and destination address. DMAC repeats (3) (4) until it reaches the area specified by the transfer size (code size).
6. When memory copy of the specified area is completed, DMAC notifies an interrupt to CPU.

DMAC use case

4.1.1 Initial configuration

This section describes the initialization of the DMAC channel and descriptor of this use case.

Figure 24 shows the setting procedure for DMAC.

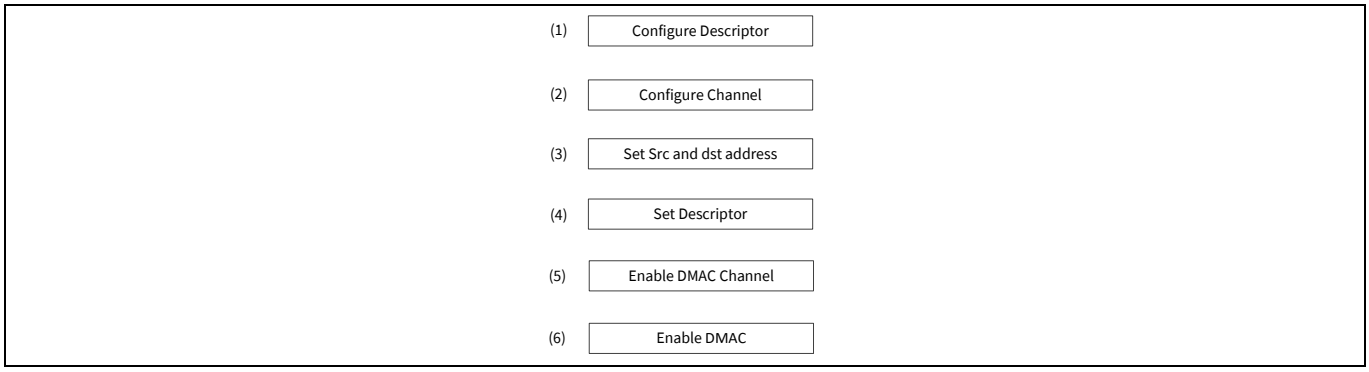


Figure 24 **Setting procedure for DMAC**

4.1.2 Example code

Code Listing 6 demonstrates an example program to DMAC. See the architecture TRM and application note for GPIO and clock configuration.

Figure 25 is a screengrab from ModusToolbox™ design.modus that shows the DMAC Channel parameters.

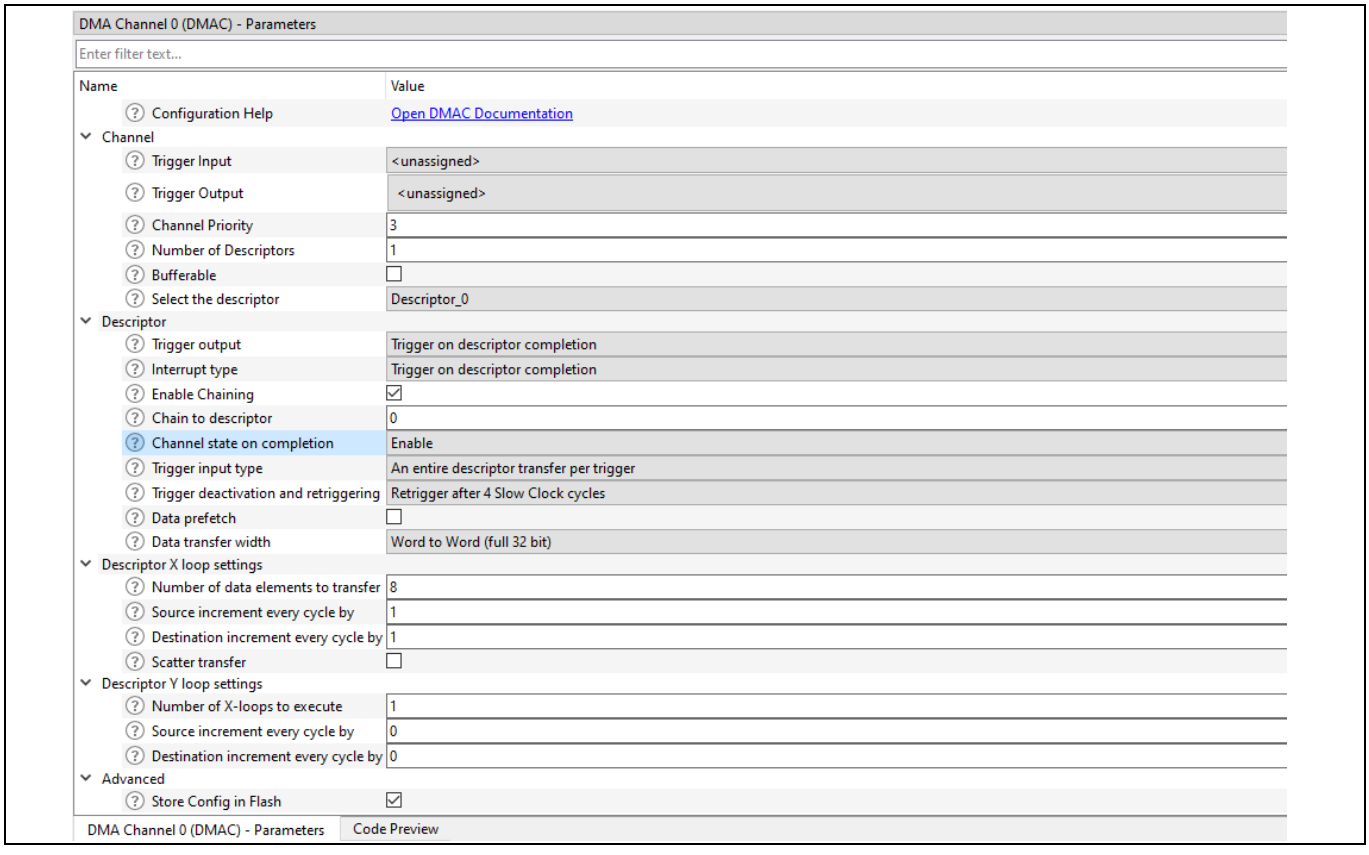


Figure 25 **Operation of DMAC**

DMAC use case

Code Listing 6 DMAC transfer example

```
#define DMAC_HW DMAC
#define DMAC_CHANNEL 0U
#define DMAC_IRQ cpuss_interrupts_dmac_0_IRQn
#define DATACNT 8

uint8_t dstData[DATACNT];
uint8_t srcData[DATACNT];

int main(void)
{
    cy_rslt_t result;
    uint32_t i;
#if defined (CY_DEVICE_SECURE)
    cyhal_wdt_t wdt_obj;

    /* Clear watchdog timer so that it doesn't trigger a reset */
    result = cyhal_wdt_init(&wdt_obj, cyhal_wdt_get_max_timeout_ms());
    CY_ASSERT(CY_RSLT_SUCCESS == result);
    cyhal_wdt_free(&wdt_obj);
#endif

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /*Initialize the User LED*/
    Cy_GPIO_Pin_Init(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN, &CYBSP_USER_LED1_config );
    Cy_GPIO_Pin_Init(CYBSP_USER_LED2_PORT, CYBSP_USER_LED2_PIN, &CYBSP_USER_LED2_config );

    /* Enable global interrupts */
    __enable_irq();

          for(i = 0; i < DATACNT; i++)
        {
                dstData[i] = 0;
                srcData[i] = i;
        }
        Cy_DMAC_Descriptor_Init(&DMAC_Descriptor_0, &DMAC_Descriptor_0_config);
        Cy_DMAC_Descriptor_SetSrcAddress(&DMAC_Descriptor_0, srcData);
        Cy_DMAC_Descriptor_SetDstAddress(&DMAC_Descriptor_0, dstData);
        Cy_DMAC_Channel_Init(DMAC_HW, 0U, &DMAC_channelConfig);
        Cy_DMAC_Enable(DMAC);
#if (CY_CPU_CORTEX_M7) && defined (ENABLE_CM7_DATA_CACHE)
        SCB_CleanDCache_by_Addr((uint32_t*)srcData, DATACNT);
        SCB_CleanDCache_by_Addr((uint32_t*)&DMAC_Descriptor_0, sizeof(Descriptor_0));
#endif
        Cy_DMAC_Channel_Enable(DMAC_HW, 0U);
        Cy_TrigMux_SwTrigger(TRIG_OUT_MUX_3_MDMA_TR_IN0, CY_TRIGGER_TWO_CYCLES);

        while(!Cy_DMAC_Channel_GetInterruptStatus(DMAC_HW,0U))
        {
        }
#if (CY_CPU_CORTEX_M7) && defined (ENABLE_CM7_DATA_CACHE)
        SCB_InvalidateDCache_by_Addr ((uint32_t*)dstData, DATACNT);
#endif
        int32_t cmpRes = memcmp(srcData, dstData, DATACNT);
        if (cmpRes !=0)
        {
                Cy_GPIO_Write(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN, 0);
                /* Insert error handling */
        }
        else
        {
                Cy_GPIO_Write(CYBSP_USER_LED2_PORT, CYBSP_USER_LED2_PIN, 0);
                /* working case */
        }

    for (;;)
    {
    }
}
```

5 Glossary

Table 7 Glossary

Terms	Description
DMA controller	Direct memory access controller
DW	Peripheral DMA
DMAC	Memory DMA
Single transfer	This transfers a single data element (8-bit, 16-bit, or 32-bit). See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
1D transfer	This performs a one-dimensional "for” loop (described in C). See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
2D transfer	This performs a two-dimensional "for” loop (described in C). See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
CRC transfer	This performs a one-dimensional “for” loop similar to the 1D transfer. However, the source data is not transferred to a destination. A CRC is calculated over the source data. Only DW is supported. See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
Memory copy	This is a special case of 1D transfer. Only DMAC is supported. See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
Scatter	This descriptor type is intended to write a set of 32-bit data elements, whose addresses are “scattered” around the address space. Only DMAC is supported. See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
Descriptor	A descriptor specifies the details of data transfer of DMA channels. See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
Descriptor chain	A DMA channel executes the next descriptor specified in the current descriptor when it completes executing the descriptor. See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
Descriptor list	Same as descriptor chain.
Descriptor pointer	The start address of the memory where the descriptor is stored. See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
Descriptor type	The transfer operation type performed by DMA. See the *Descriptors* section in the “Direct Memory Access” chapter of the architecture TRM for details.
Descriptor word	The composition element of the descriptor. There are descriptor control, source/destination address, X/Y loop control, and descriptor next pointer. See the *DW Descriptor Structure* and *DMAC Descriptor Structure* sections in the “Direct Memory Access” chapter of the architecture TRM for details.

Glossary

Preemptable	DW'-specific functions. See the *Channels* section in the “Direct Memory Access” chapter of the architecture TRM for details.
MMIO	Memory-mapped I/O
ADC	Analog-to-digital converter. See the “SAR ADC” chapter of the architecture TRM for details.
SCB	Serial communications block. See the “Serial Communications Block (SCB)” chapter of the architecture TRM for details.
Trigger multiplexer	A trigger multiplexer routes triggers from a source peripheral to a destination. See the “Trigger Multiplexer” chapter of the architecture TRM for details.

References

[1] Contact Technical support to obtain more XMC7000 family references documents.

[2] Device datasheets:

- XMC7100 series 32-bit Arm® Cortex®-M7 microcontroller datasheet
- XMC7200 series 32-bit Arm® Cortex®-M7 microcontroller datasheet

[3] Technical reference manuals:

- XMC7000 MCU architecture technical reference manual
- XMC7100 MCU registers Technical reference manual
- XMC7200 MCU registers Technical reference manual

[4] Application notes:

- AN234118 - GPIO usage setup in XMC7000 family
- AN234127 - How to use serial communications block (SCB) in XMC7000 family

Revision history

Document revision	Date	Description of changes
**	2021-12-06	New application note.
*A	2022-06-27	Replaced P-DMA with DW in all instances across the document. Replaced M-DMA with DMAC in all instances across the document. Updated P-DMA use cases. Updated 1D transfer (memory-to-peripheral). Updated Configuration and example code. Updated almost entire section. Updated 1D transfer (peripheral-to-memory). Updated Configuration and example code. Updated almost entire section. Updated Descriptor chaining. Updated Configuration and example code. Updated almost entire section. Updated 2D transfer (peripheral-to-memory). Updated Configuration and example code. Updated almost entire section. Updated CRC transfer. Updated Configuration and example code. Updated almost entire section. Updated DMAC use case. Updated Memory-to-memory (memory copy). Updated Configuration and example code. Updated almost entire section.
*B	2023-07-19	Updated code for PDL version 3.3.0. Changed DW 2D function from ADC mode to UART mode. Added necessary ModusToolbox™ figures. Removed redundant descriptions tables and figures in each codes. Updated reference documents links.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-07-19

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-34225 Rev. *B

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.