CYPRESS
PERFORM

AN52705

PSoC® 3 and PSoC 5LP - Getting Started with DMA

Authors: Anu M D, Lakshmi Natarajan
Associated Project: Yes
Associated Part Families: All PSoC® 3 and PSoC 5LP parts
Software Version: PSoC® Creator™ 2.1 SP1 and higher
Related Application Notes: AN61102

Abstract

AN52705 provides an introduction to direct memory access (DMA) in PSoC® 3 and PSoC 5LP. PSoC DMA can transfer data between on-chip peripherals and memory with no CPU intervention. The application note illustrates how to configure the DMA for simple data transfers, including peripheral to memory, memory to peripheral, peripheral to peripheral and memory to memory, using example projects.

Contents

Introduction

The DMA controller (DMAC) in PSoC 3 and PSoC 5LP can transfer data from a source to a destination with no CPU intervention. This allows the CPU to handle other tasks while the DMA does data transfers, thereby achieving a ‘multiprocessing’ environment.

The PSoC DMA Controller (DMAC) is highly flexible – it can seamlessly transfer data between memory and on chip peripherals including ADCs, DACs, Filter, USB, UART, and SPI. There are 24 independent DMA channels.

This application note describes how to configure the DMA for simple data transfers. It includes projects that show several different types of DMA transfers:

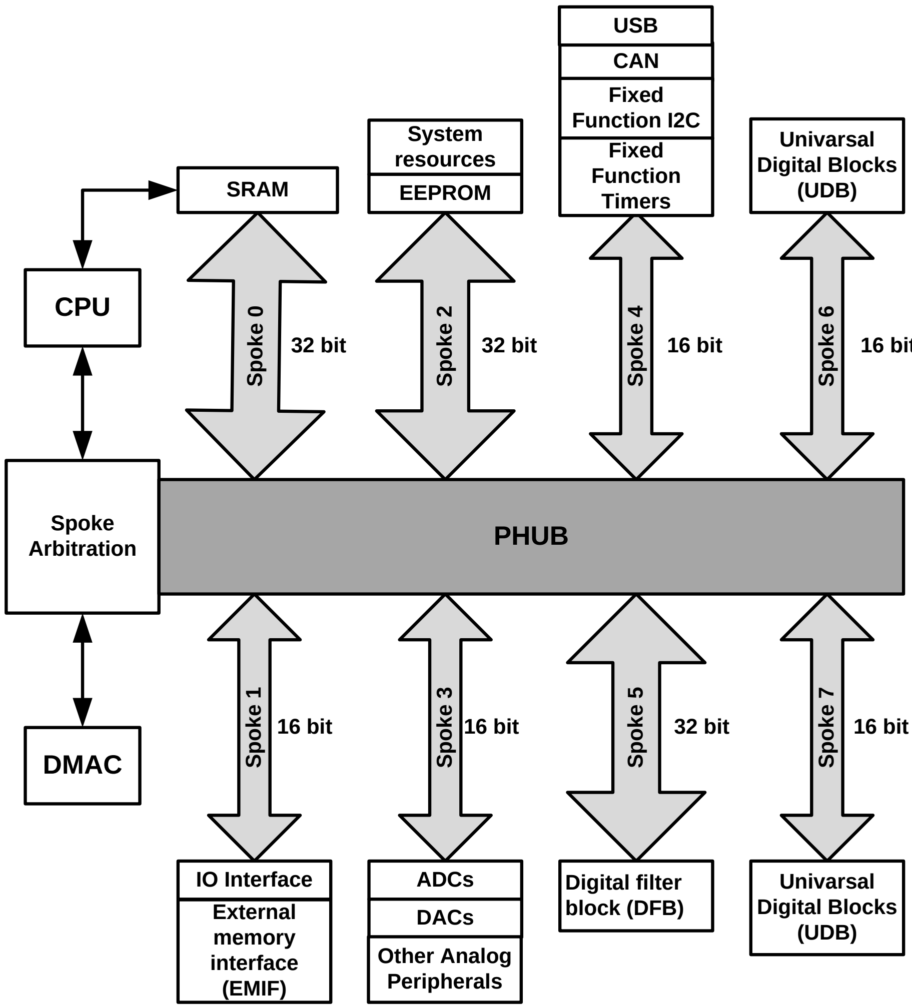
- Peripheral-to-Memory
- Memory-to-Peripheral
- Peripheral-to-Peripheral
- Memory-to-Memory

This application note assumes that you are familiar with developing applications using PSoC Creator for PSoC 3 or PSoC 5LP. If you are new to PSoC 3 or PSoC 5LP, introductions can be found in AN54181, Getting Started with PSoC 3 and AN77759, Getting Started with PSoC 5. If you are new to PSoC Creator, see the PSoC Creator home page.

Basic Concepts of DMA

The DMAC in PSoC 3 and PSoC 5LP is a part of a central hub called the Peripheral HUB (PHUB) that connects on-chip peripherals, as Figure 1 shows. The DMAC is one of the two PHUB bus masters.

Figure 1. Peripheral HUB



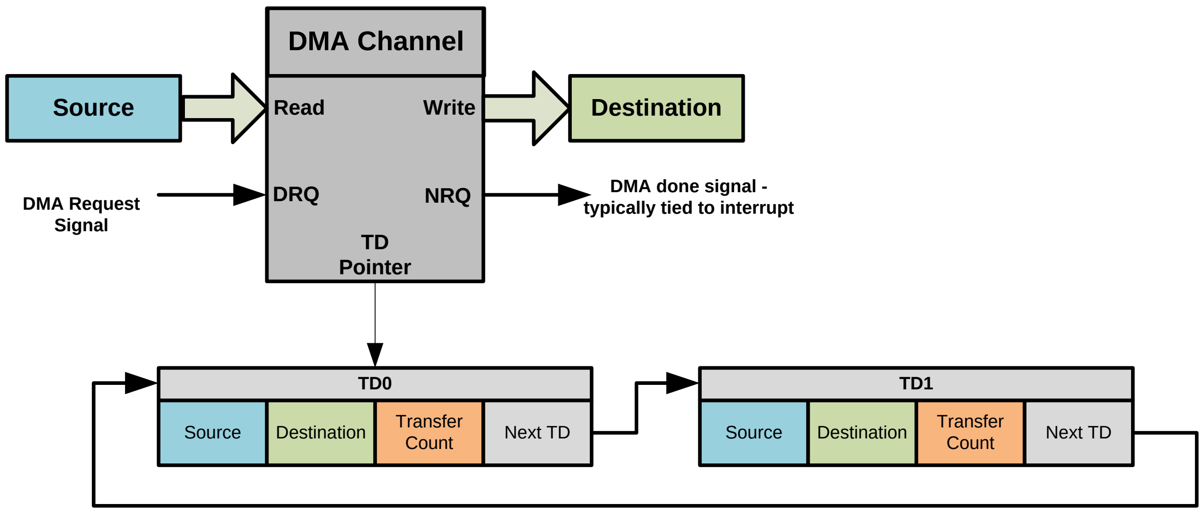
The PHUB has eight data buses that are called spokes. Each spoke connects the CPU and DMAC to one or more peripherals. Spokes can have widths of either 16 bits or 32 bits. Peripherals attached to a spoke can have widths of 8 bits, 16 bits, or 32 bits.

The data width of a peripheral is usually less than or equal to the data width of the spoke to which it is attached. If a peripheral data width is greater than that of the spoke attached to it, the PHUB can transact with the peripheral at the width of the spoke.

The PHUB has two bus masters, the CPU and the DMAC. The CPU and the DMAC can access different PHUB spokes at the same time. If the CPU and DMAC try to access the same spoke at the same time, bus arbitration occurs. See the PSoC 3 and PSoC 5LP Technical Reference Manuals for details.

Each of the 24 DMA channels can independently transfer data. Each channel has a Transaction Descriptor (TD) chain, as Figure 2 shows. The TD contains information such as source address, destination address, transfer count, and the next TD in the chain. There can be as many as 128 TDs. The combination of channel and TD describes the complete DMA transfer.

Figure 2. DMA Channel

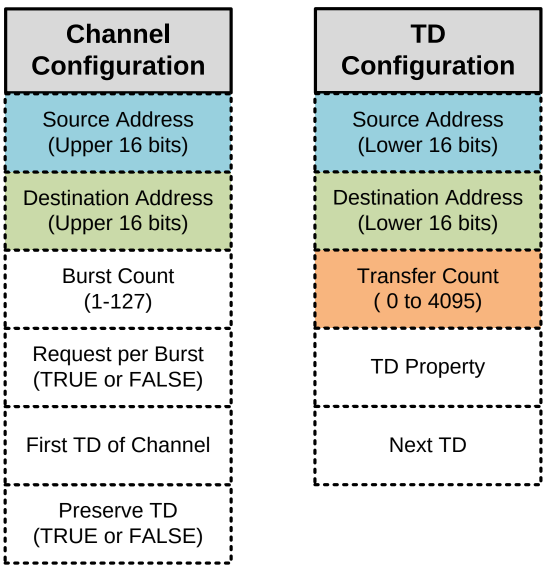


Each DMA channel has a separate DMA request input that initiates a transaction. A DMA request can be initiated by the CPU or by a peripheral. When a DMA request is received, the DMAC accesses the spokes attached to the source and destination and moves data as configured in the channel and the associated TD.

DMA Configuration

A DMA transfer is configured using channel and TD configuration registers. Figure 3 shows the channel and the TD configuration parameters.

Figure 3. DMA Configuration



Channel Configuration

DMA channel configuration parameters are explained as follows:

- **Upper Source Address (16 bits)**

  The upper 16 bits of the 32-bit source address is configured in channel configuration register

- **Upper Destination Address (16 bits)**

  The upper 16 bits of the 32-bit destination address is configured in this channel configuration register

- **Burst Count (1 to 127)**

  Defines the number of bytes the DMA channel must move from the source to destination before it releases the spoke. The DMAC acquires the spoke, transfers the specified number of bytes from the source to the destination and then releases the spoke. For the next burst transfer, it re-acquires the spoke.

  Limit burst count for intra spoke DMA transfers to less than or equal to16.

- **Request Per Burst (0 or 1)**

  When multiple burst transfers are required to finish the DMA data transfer, this bit determines the nature of the bursts.

  0: All subsequent bursts after the first burst are automatically done without a separate DMA request. Only the first burst transfer must have a DMA request.

  1: Every burst transfer requires a separate request.

- **Initial TD**

  Defines the first TD associated with the channel. The pointer to the first TD is stored in channel configuration memory. Subsequent TD pointers are stored in TD configuration memory, similar to a linked list.

- **Preserve TD (0 or 1)**

  Defines whether to save the original TD configuration for re-use, for subsequent DMA transfers. Typically TD configurations are preserved.

  0: Do not preserve the TD configuration.

  1: Preserve the TD configuration.

  If TDs are preserved, the channel uses a separate TD memory (corresponding to the channel number) to track the ongoing transaction; otherwise the original TD configuration registers are used as working registers to track the ongoing transaction.

TD Configuration

TD configuration parameters are explained below:

- **Lower Source Address (16-bit)**

  The lower 16 bits of the 32-bit source address configured in TD configuration registers

- **Lower Destination Address (16-bit):**

  The lower 16-bits of the 32-bit destination address

- **Transfer Count (0 to 4095 bytes)**

  The total number of bytes to be transferred from the source to the destination.

  The transfer count is used along with the burst count parameter. For example, if you want to move 50 2-byte words of data from a 16-bit peripheral to a memory buffer, the burst count is set to 2 and transfer count is set to 100.

- **Next TD**

  The next TD, similar to a linked list

- **TD Properties**

  Table 1 shows the TD properties defined by the bit fields in the TD property configuration register.

  Table 1. TD Properties

Property	Description
Increment Source Address	If this bit is set, the source address is incremented as the DMA transaction proceeds.
Increment Destination Address	If this bit is set, the destination address is incremented as the DMA transaction proceeds
Swap Enable	If set, DMA swaps the data bytes while it moves data from the source to destination.
Swap Size	Defines the size of the swap performed, if Swap Enable is set. 0: Every 2 bytes are endian swapped during the DMA transfer. 1: Every 4 bytes are endian swapped during the DMA transfer
Auto Execute Next TD	0: The next TD in the chain is executed only after the next DMA request. 1: The next TD in the chain is automatically executed after the current TD transfer is finished.
DMA Completion Event	If set, generates a DMA “done signal” after the data transfer is finished. This is typically used to create an interrupt after the transfer is finished.
Enable TD termination	If set, the ongoing transaction can be terminated using hardware signal

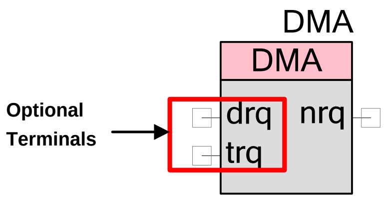
The PSoC 3 Keil Compiler uses big endian format to store 16-bit and 32-bit variables. But the PSoC 3 peripheral registers use little endian format. For this reason, the DMA must be configured to swap bytes when it moves multi-byte data between the peripheral registers and memory in PSoC 3. This is not required for PSoC 5LP as both peripherals and memory uses the same endian format.

Now let us see how to configure the DMA using PSoC Creator.

DMA component overview

Figure 4 shows the DMA channel component in PSoC Creator. This component can be found under the Systems tab in the Component Catalog.

Figure 4. DMA Channel Component

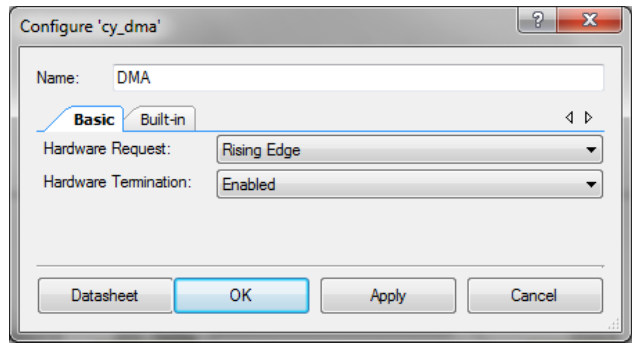


The DMA channel component and an associated API are used to configure the DMA to transfer data.

Hardware Connections of DMA component

You can set the following parameters in the component configuration window as Figure 5 shows.

Figure 5. DMA component Configuration



Hardware Request (drq): This setting defines the type of signal (rising edge/level) used to trigger the DMA channel. Any selection for this parameter except "Disabled" adds a drq terminal to the component. The drq can be connected to any hardware signal, to trigger the DMA channel.

Without the drq terminal, the DMA transaction is triggered only by the CPU.

When this parameter is set to “derived”, the DMA trigger type - edge/level is determined from the source of the DMA trigger. For more information, see the DMA component datasheet.

Hardware Termination (trq)*:* When this option is set to true, another input terminal (trq) is displayed in the component. If TD termination is enabled, a rising edge on this terminal stops an ongoing DMA transaction. Note that trq terminates a TD chain only if there is an ongoing DMA burst transaction. Refer to the component datasheet for more details.

Transfer complete (nrq) : In order to indicate that the DMA transfer is finished, the TD can be configured to create a pulse of width 2 bus clocks at the NRQ terminal of the DMA channel, when the transfer is finished. The nrq terminal can be connected to an interrupt, or to another component for further actions.

Set the TD properties to define whether or not to generate a signal on the nrq terminal, and whether or not to enable TD termination using trq.

Firmware Configuration of DMA

The DMA component generates a source file and corresponding header file for each DMA instance during the project build process. For example, if there is a DMA component instance in your design that has the name DMA_1, then the files - *DMA_1_dma.c* and *DMA_1_dma.h* are created during the build process. These files include the "DmaInitialize" API that is used to initialize the DMA channel. Other channel and TD configuration functions are included in *CyDmac.c* and *CyDmac.h* files in the Generated Source folder.

Following are the firmware configuration steps for DMA:

1. Start the DMA channel
```
Channel_Handle = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST,
                                   HI16(Source Address), HI16(Destination Address))
```
2. Create an instance of a TD
```
TD_Handle = CyDmaTdAllocate();
```
3. Set the TD configuration
```
CyDmaTdSetConfiguration(TD_Handle,Transfer_Count,Next_TD,TD_Property);
```
4. Set the TD address
```
CyDmaTdSetAddress(TD_Handle, LO16(Source Address), LO16(Destination Address))
```
5. Set the channel's initial TD
```
CyDmaChSetInitialTd(Channel_Handle, TD_Handle)
```
6. Enable the DMA channel
```
CyDmaChEnable(Channel_Handle, preserve_TD)
```

The above firmware steps are detailed in Appendix A: DMA Configuration Steps on page 19. A DMA wizard can be used to automatically generate the code to configure the DMA channel; see Appendix B: DMA Wizard Configuration on page 21 for more details.

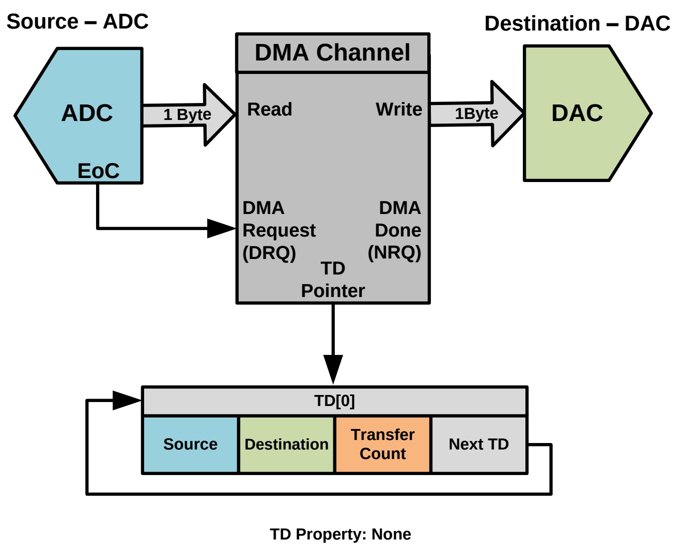
Note that the DMA wizard supports DMA transactions between only a limited set of PSoC peripherals. If the DMA wizard does not support a peripheral, you must manually configure the DMA channel using the functions detailed in Appendix A.

Following are a set of four examples that show in detail how to do DMA transfers between memory and peripherals. A fifth example shows how to build a multiple-TD chain.

Example 1: Peripheral-to-Peripheral Transfer

This example shows how to use DMA to do a simple peripheral-to-peripheral transfer, that is, from an ADC data out register to a DAC data input register as Figure 6.

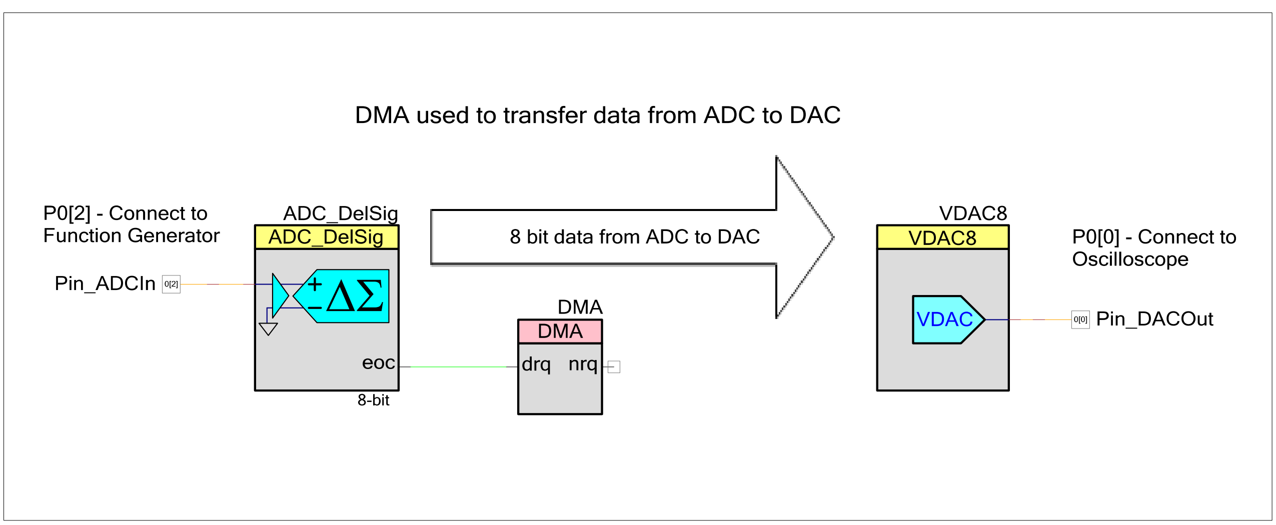
Figure 6. Block Diagram, Peripheral-to-Peripheral Transfer



As Figure 7 shows, the ADC is configured in 8-bit, single-ended mode to match the data format of the VDAC, which is a single-ended 8-bit voltage DAC. The hardware request (DRQ) of the DMA channel is enabled and connected to the ADC EoC signal so that ADC can make a request for data transfers whenever an ADC result is available.

After it receives the request, the DMA channel reads one byte of data from the ADC output register and writes to the DAC data register.

Figure 7. Top Design, Peripheral-to-Peripheral Transfer



Example 1 DMA Configuration

The DMA channel and TD configurations for this project are given in Table 2 and Table 4.

Table 2. Channel Configuration Settings

Parameter	Project Setting
Upper Source Address	HI16(CYDEV_PERIPH_BASE)
Upper Destination Address	HI16(CYDEV_PERIPH_BASE)
Burst Count	1 (One byte)
Request Per Burst	1 (True)
Initial TD	TD[0]
Preserve TD	1 (True)

The channel configuration has the upper 16 bits of the 32-bit address for both the source and destination addresses.

CYDEV_PERIPH_BASE, defined in the PSoC Creator auto-generated file *cydevice.h*, defines the base address of all PSoC peripherals including the ADC and the DAC.

HI16 is a PSoC Creator macro that returns the upper 16 bits of a 32-bit value. This macro is used to get the upper 16 bits of the source and destination address.

As an alternative, you can assign the upper source and destination addresses relative to the component registers, as Table 3 shows. The address definitions can be found in the component files ADC_DelSig.h, and VDAC8.h, respectively.

Table 3. Alternative Upper Addresses

Parameter	Project Setting
Upper Source Address	HI16(ADC_Delsig_DEC_OUTSAMP_PTR)
Upper Destination Address	HI16(VDAC8_DATA_PTR)

The TDs can be viewed as an array of chained TDs; in this case we need only a one-element array TD[0].

Table 4. TD[0] Configuration Settings

Parameter	Project Setting
Lower Source Address	LO16(ADC_Delsig_DEC_OUTSAMP_PTR)
Lower Destination Address	LO16(VDAC8_DATA_PTR)
Transfer Count	1 (One byte)
TD property	None(0)
Next TD	TD[0] (Loop back to the same TD)

The LO16 macro returns the lower 16-bits of a 32-bit value.

The DMA channel must move one byte for each DMA request, so the burst count is set to 1 byte and the request per burst is set to True.

The next TD to be executed is set to the same TD (looped), so the same transaction is repeated on each DMA request. The Preserve TD parameter is set to True.

Example 1 Project Files

The project Eg1_ADC_DMA_DAC in the AN52705.zip file attached to this application note demonstrates this example. See Appendix D: Example Projects – Test Setup on page 24 for details on how to test this project

The DMA configuration code for this example is given below. See Appendix B: DMA Wizard Configuration on page 21 for details on how to generate this configuration code using the DMA wizard.

Example 1 DMA Configuration Code

```
/* Define for DMA Configuration */
#define  DMA_BYTES_PER_BURST    1
#define  DMA_REQUEST_PER_BURST  1
#define  DMA_SRC_BASE  (CYDEV_PERIPH_BASE)
#define  DMA_DST_BASE  (CYDEV_PERIPH_BASE)

/* Variable declarations for the DMA channel.
 * DMA_Chan is used to store the DMA channel */
uint8 DMA_Chan;
/* DMA_TD array is used to store all of the TDs associated with the channel
 * Since there is only one TD in this example, DMA_TD array contains only one element */
uint8 DMA_TD[1];

/* DMA Configuration steps */

/* Step 1 */
/* DMA Initializations done for both the DMA Channels
 * Burst count = 1, (8 bit data transferred to VDAC one at a time)
 * Request per burst = 1 (transfer burst only on new request)
 * High byte of source address = Upper 16 bits of ADC data register
 * High byte of destination address = Upper bytes of the VDAC8 data register
 * DMA_Chan holds the channel handle returned by the 'DmaInitialize' function. This is
 * used for all further references of the channel */
DMA_Chan = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST,
                             HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE));
/* Step 2 */
/* Allocate TD for DMA Channel
 * DMA_TD[0] is a variable that holds the TD handle returned by the TD allocate function.
 * This is used for all further references of the TD */
DMA_TD[0] = CyDmaTdAllocate();

/* Step 3 */
/* Configure TD[0]
 * Transfer count = 1 (total number of bytes to transfer from the ADC to DAC)
 * Next Td = DMA_TD[0]. The same td has to repeat itself for every ADC EoC.
 * Configuration = No special TD configurations required */
CyDmaTdSetConfiguration(DMA_TD[0], 1, DMA_TD[0], 0);

/* Step 4 */
/* Configure the td addres
 * Source address = Lower 16 bits of ADC data register
 * Destination address = Lower 16 bits of VDAC8 data register */
CyDmaTdSetAddress(DMA_TD[0], LO16((uint32)ADC_DelSig_DEC_SAMP_PTR),
                             LO16((uint32)VDAC8_Data_PTR));
/* Step 5 */
/* Map the TD to the DMA Channel */
CyDmaChSetInitialTd(DMA_Chan, DMA_TD[0]);

/* Step 6 */
/* Enable the channel
 * The Channel is enabled with Preserve TD parameter set to 1. This preserves the
 * original TD configuration and reload it after the transfer is complete so that the TD
 * can be repeated */
CyDmaChEnable(DMA_Chan, 1);
```

Example 2: Peripheral-to-Memory Transfer

This example shows how to do a peripheral-to-memory transfer from an ADC data out register to a 16-bit memory array, as Figure 8 shows.

Figure 8. Block Diagram, Peripheral-to-Memory Transfer

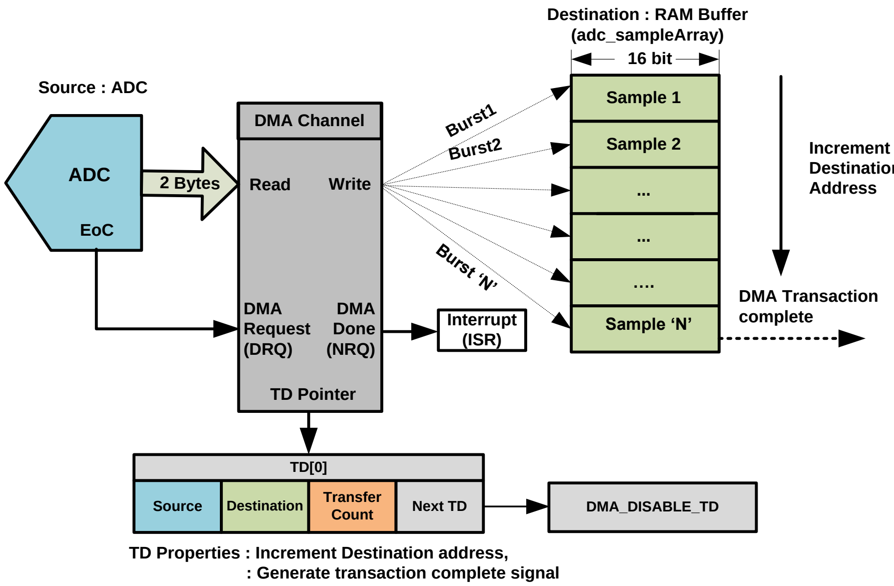
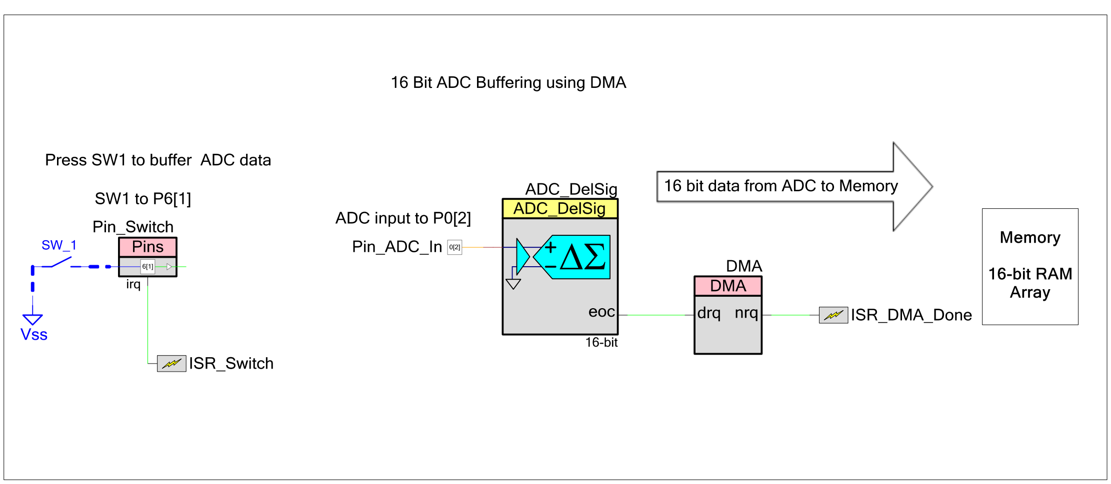


Figure 9 shows the top design of the project. Each time the Pin_Switch is pressed; ISR_Switch is triggered, and a flag is set in the isr to enable the DMA channel. Once the DMA channel is enabled, the EoC signal from ADC activates the DMA channel request.

On each DMA request, the DMA fetches 2 bytes from the source – the ADC output register - writes them to the destination RAM buffer, and increments the destination address by 2. The transfer count is decremented by 2 after each burst transfer. This repeats until the transfer count is 0, which generates a transaction complete signal at the NRQ terminal of the DMA component, which activates the ISR_DMA_Done interrupt.

In the interrupt service routine a flag is set to indicate that the transaction is complete .The DMA channel is disabled when the transaction is completed, and re-enabled when the switch is pressed again.

Figure 9. Top Design, Peripheral-to-Memory Transfer



Example 2 DMA Configuration

The DMA channel and TD configurations for the project are given in Table 5 and Table 6.

Table 5 Channel Configuration Setting

Parameter	Project Setting
Upper Source Address	HI16(CYDEV_PERIPH_BASE)
Upper Destination Address	HI16(CYDEV_SRAM_BASE)
Burst Count	2 (Two bytes)
Request Per Burst	1 (True)
Initial TD	TD[0]
Preserve TD	1 (True)

The channel configuration has the upper 16 bits of the 32-bit address for both the source and destination addresses. The source address is same as in Example1.

CYDEV_SRAM_BASE, defined in the PSoC Creator auto-generated file *cydevice.h*, defines the base address of SRAM. This is used with HI16 macro to specify the upper 16 bits of destination address.

As an alternative, the RAM array pointer can be used with HI16 macro to specify upper 16bits of source address for PSoC 5LP but not for PSoC 3. This is because the upper 16 bits of the address of RAM variables is zero for PSoC 3, but the Keil compiler stores Keil-specific information in the upper 16 bits of the variable address. For this reason, HI16 (&adc_sampleArray) returns an incorrect address when used with PSoC 3 – Keil compiler.

In this example, a 2-byte ADC result must be moved from ADC to RAM array on each DMA request and therefore the burst count is set to 2 and the request per burst is set to true.

The Preserve TD is set to 1 (TRUE) so that the original TD setting, i.e. source address, destination address and transfer count, are preserved and the transactions can be repeated.

The lower 16 bits of source and destination address are specified in the transaction descriptor (TD[0]) configuration as given in Table 5.

Table 6. TD[0] Configuration Settings

Parameter	Project Setting
Lower Source Address	LO16(ADC_Delsig_DEC_OUTSAMP_PTR)
Lower Destination Address	LO16(&adc_sampleArray)
Transfer Count	200 x 2 (No. of samples × Bytes per sample)
TD properties	Increment Destination Address, Generate DMA done event, Swap Enable required only for PSoC 3. (TD_INC_DST_PTR | DMA__TD_TERMOUT_EN | `TD_SWAP_EN)`
Next TD	`DMA_DISABLE_TD`

The transfer count is set to '400' which is equal to the 'Number of samples to be buffered × Bytes per Sample'.

The TD property is set to increment the destination address after each burst transfer and generate a transaction complete signal once the specified number of samples is buffered. In PSoC 3 project, the TD is also configured to swap the bytes while moving data from ADC to memory as explained in TD property section. The bits corresponding to each of the TD property is defined in the PSoC Creator auto-generated file CyDmac.h. The required property bit fields are OR-ed together to set the TD property.

In order to stop the DMA transfers after buffering the specified number of samples, TD[0] is chained to '`DMA_DISABLE_TD`' which disables the DMA channel.

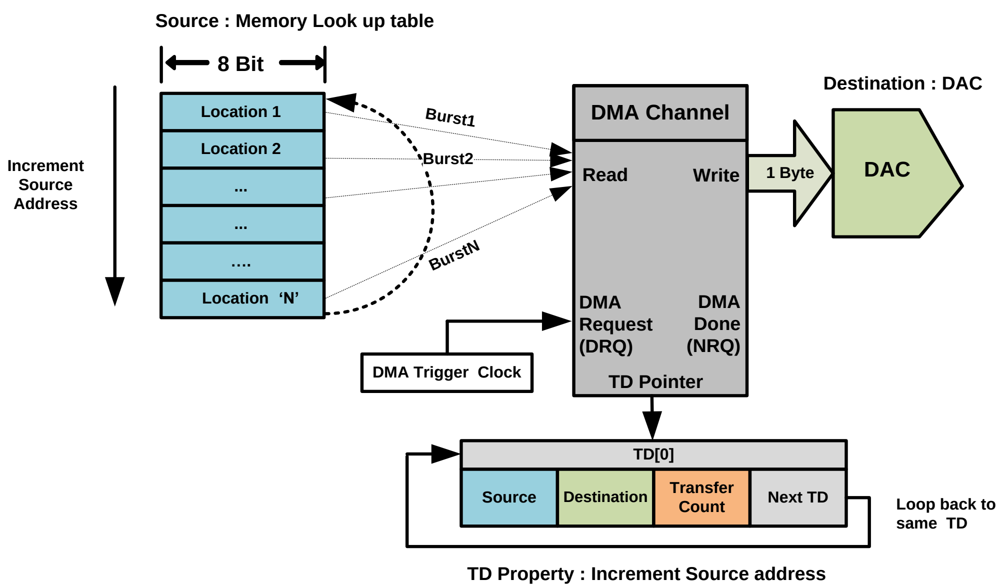
Example 2 Project Files

The project Eg2_ADC_DMA_Mem in the AN52705.zip file attached to this application note demonstrates this example. The DMA configuration code for the example is similar to Example 1. The arguments passed to the functions are given in the channel and TD configuration tables above. See Appendix D: Example Projects – Test Setup on page 24 for details on how to test this project.

Example 3: Memory-to-Peripheral Transfer

This example shows how to use the DMA for a memory-to-peripheral data transfer. The example demonstrates wave generation using a DAC, as Figure 10 shows.

Figure 10. Block Diagram, Memory-to-Peripheral Transfer

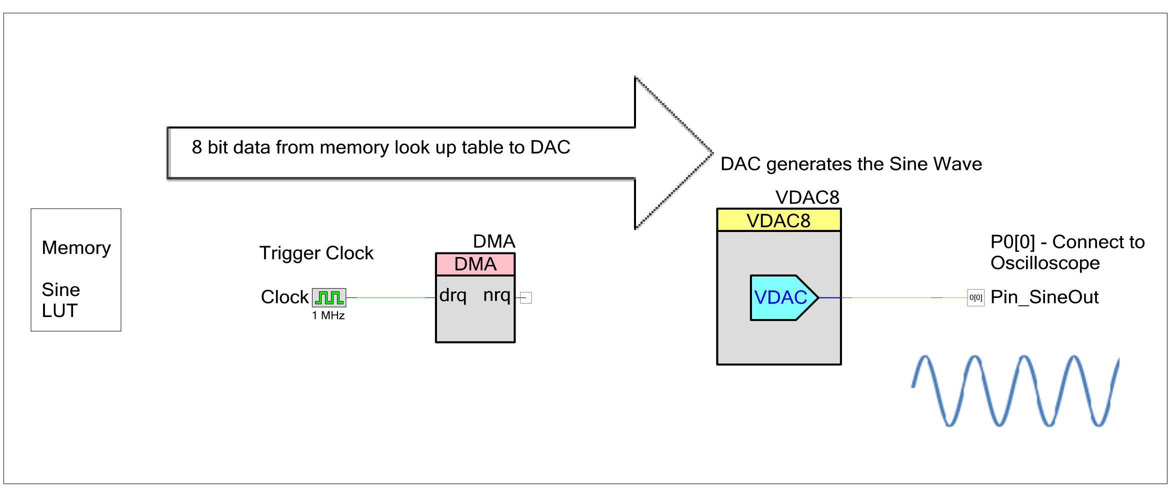


A sine lookup table with 128 points is stored in flash memory. These values are sequentially sent to a DAC, using DMA, to create a sine wave. Figure 11 shows the top design for the project.

A clock component is used to periodically generate DMA requests (drq). When the request is received, the DMA channel fetches one byte of data from the lookup table and writes it to the DAC data register. The source address is incremented by one and the transfer count is decremented by one after each burst transfer. This continues until all table values are sent to DAC.

The TD configuration is preserved and reloaded at the end of the transfer so as to generate a continuous sine wave. The frequency of the sine wave is equal to the DMA trigger clock frequency divided by number of points in the lookup table.

Figure 11. Top Design, Memory-to-Peripheral Transfer



Example 3 DMA Configuration

The DMA channel and TD configurations for the project are given in Table 7 and Table 8.

Table 7. Channel Configuration

Parameter	Project Setting
Upper Source Address	HI16(CYDEV_FLS_BASE) for PSoC 3 HI16 (&sineTable) for PSoC 5LP
Upper Destination Address	HI16(CYDEV_PERIPH_BASE)
Burst Count	1 (One byte)
Request Per Burst	1 (True)
Initial TD	TD[0]
Preserve TD	1 (True)

The source for DMA transfer is the sineTable array that is kept in flash memory. The *HI16(&sineTable)* sets the upper 16 bits of the source address for PSoC 5LP whereas *HI16(CYDEV_FLS_BASE)* is used to identify the upper 16 bits of source address for PSoC 3 for the reasons mentioned in the previous examples.

The DMA channel must move one byte from look up table array to DAC for each DMA request. So, the burst count is set to 1 byte and the request per burst is set to true.

The original TD configurations are preserved so that it can be re-used.

The lower 16 bits of source and destination addresses are set using the LO16 macro as given in Table 8.

Table 8. TD[0] Configuration

Parameter	Project Setting
Lower Source Address	LO16 (&sineTable)
Lower Destination Address	LO16(VDAC8_DATA_PTR)
Transfer Count	128 (No. of bytes in the sine look up table)
TD property	Increment source address (TD_INC_SRC_ADR)
Next TD	TD[0] - Loop back to the same TD again

The transfer count is set to the total number of bytes in the sine look up table.

The TD is configured to increment the source address, i.e. look up table pointer, after each burst transfer.

At the end of the transfer, one complete cycle of sine wave is generated at the DAC output. The TD is preserved and looped back to itself so as to generate a continuous wave.

Example 3 Project Files

The project Eg3_Mem_DMA_DAC in the AN52705.zip file attached to this application note demonstrates this example. The DMA configuration code for the example is similar to Example 1. The arguments passed to the functions are given in the channel and TD configuration tables above. See Appendix D: Example Projects – Test Setup on page 24 for details on how to test this project

Example 4: Memory-to-Memory Transfer

This example shows how to use DMA to do a memory-to-memory transfer. It also demonstrates how to trigger a DMA channel using the CPU. In this example, an 8-byte flash array is copied to an 8-byte RAM array on a CPU request, as Figure 12 shows.

Figure 12. Block Diagram, Memory-to-Memory Transfer

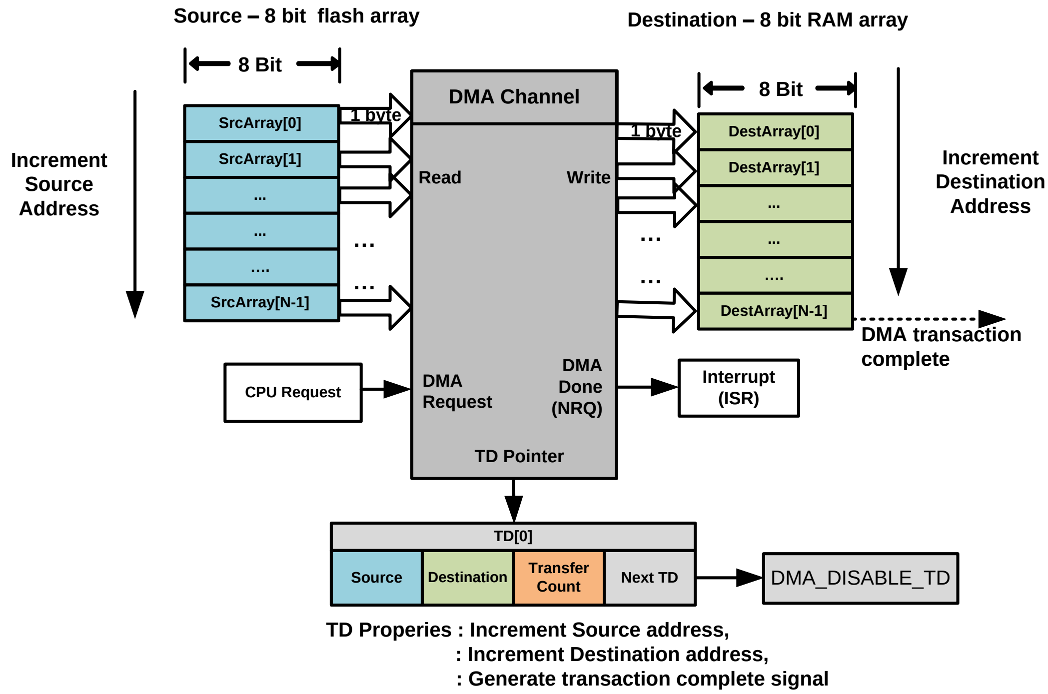
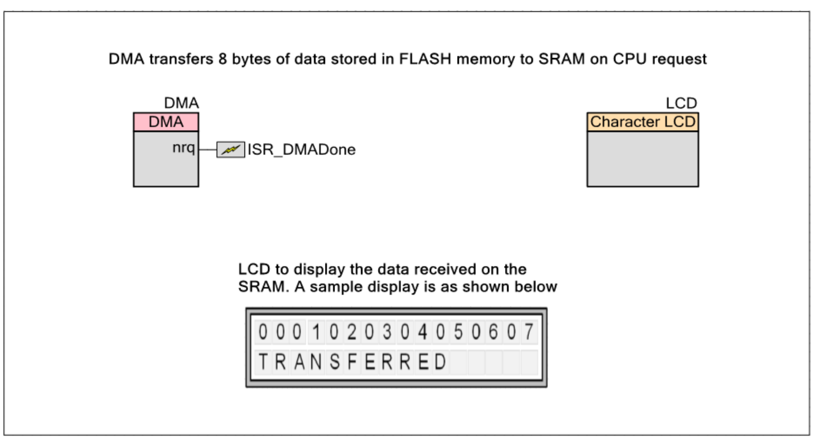


Figure 13 shows the top design of the project. The CyDmaChSetRequest function is used to activate the DMA transfer approximately one second after device power up.

When it receives a request from the CPU, the DMA transfers 8 bytes from the flash array to the RAM array as configured in the channel and TD configuration registers.

Figure 13. Top Design: Memory-to-Memory Transfer



The TD source and destination addresses are incremented as the transfer proceeds.

When the transfer is complete, a pulse is generated at the nrq signal terminal of the DMA. This activates the ISR_DMADone interrupt which sets the flag to indicate that the transfer is complete. The new RAM contents are then displayed on the LCD.

Example 4 DMA Configuration

The DMA channel and TD configurations for the project are given in Table 9 and Table 10.

Table 9. Channel Configuration

Parameter	Project Setting
Upper Source Address	HI16(CYDEV_FLS_BASE), for PSoC 3 HI16(&sourceArray), for PSoC 5LP
Upper Destination Address	HI16(CYDEV_SRAM_BASE)
Burst Count	1 (One byte)
Request Per Burst	0 (False)
Initial TD	TD[0]
Preserve TD	0 (False)

The source for DMA transfer is the 'sourceArrray' defined in the flash memory. The destination is 'destinationArray' in RAM. The upper 16 bits of the source address in flash are set to HI16(&sourceArray) in PSoC 5LP and HI16(CYDEV_FLS_BASE) in PSoC 3, as explained in previous examples. Similarly, the upper 16 bits of the destination address in SRAM are set using the macro HI16(CYDEV_SRAM_BASE).

The burst count is set to 1 byte so that the DMA reads byte by byte from flash and writes it to the RAM array. You can set the burst count to 8 bytes for faster data transfers. However, you should also generally set the burst count to a low value so as to allow the spoke to be shared by other DMA channels.

The request per burst parameter is set to false so that separate requests are not required for each burst transfer.

Table 10. TD[0] Configuration

Parameter	Project Setting
Source Address	LO16(&sourceArray)
Destination Address	LO16(&destinationArray)
Transfer Count	8 (bytes)
TD property	Increment source address Increment destination address Generate DMA done signal (TD_INC_SRC_ADR | TD_INC_DST_ADR | DMA__TD_TERMOUT_EN)
Next TD	DMA_DISABLE_TD (0xFE)

The lower 16 bits of the source and the destination addresses for the TD configuration are identified by the LO16 macro.

The transfer count is set to 8 so that a total of 8 bytes are transferred from the source to the destination.

The TD is configured to increment the source address i.e., the flash array pointer, and the destination address i.e., the RAM array pointer, after each burst transfer. The TD is also configured to send a termout pulse on the nrq line after all the 8 bytes are moved from the flash to the RAM array. This pulse is used to trigger an ISR to indicate that the transfer is complete. The next TD is set to DMA_DISABLE_TD (0xFE) to disable the DMA channel after the transfer is finished.

Since the transaction has to happen only one time, the TD configuration does not need to be preserved.

Example 4 Project Files

The project Eg4_Mem_DMA_Mem in the AN52705.zip file attached to this application note demonstrates this example. The DMA configuration code for the example is similar to Example 1. The arguments passed to the functions are given in the channel and TD configuration tables above. See Appendix D: Example Projects – Test Setup on page 24 for details on how to test this project.

Example 5: TD Chaining

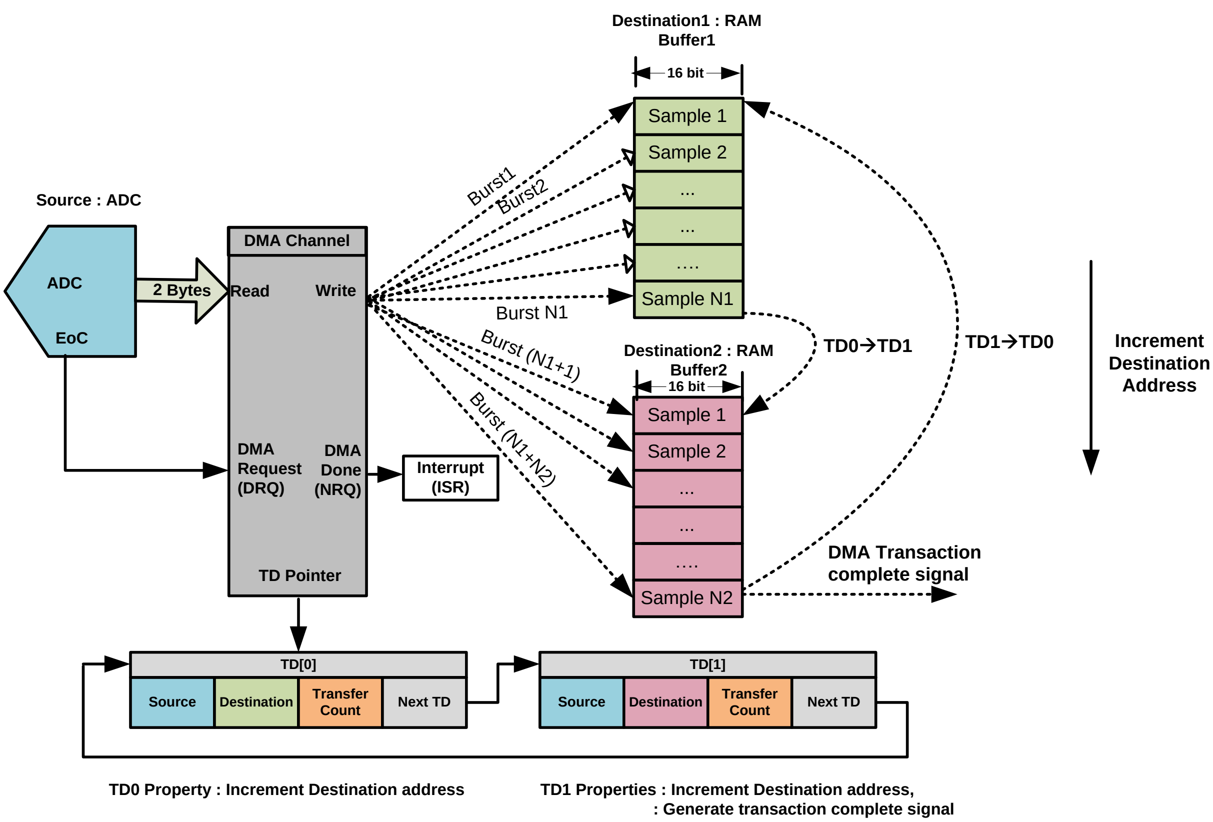
This example project shows how to use multiple TDs with a single channel and chain them to one another. In this example the ADC data is sent to two separate RAM buffers, one after the other, using a single DMA channel and two TDs.

The DMA channel is configured to do two transactions:

- Transaction 1: ADC to RAM buffer1
- Transaction 2: ADC to RAM buffer2

These two transactions are configured using two separate transaction descriptors - TD[0] and TD[1] - and are chained to one another using the TD chaining feature of the DMA, as Figure 14 shows.

Figure 14. Block Diagram, TD Chaining



This form of TD configuration can also be used to overcome the maximum transfer count limit of a single TD, which is 4096 bytes for a single DMA channel. Note that the upper 16 bits of the source and the destination addresses must be the same for all of the TDs in a chain.

The top design for the project is the same as in Example 2.

Example 5 DMA Configuration

The channel and TD configurations for this project are given in Table 11, Table 12 and Table 13.

Table 11. Channel Configuration

Parameter	Setting
Upper Source Address	HI16(CYDEV_PERIPH_BASE)
Upper Destination Address	HI16(CYDEV_SRAM_BASE)
Burst Count	2 (Two Bytes)
Request Per Burst	1 (True)
Initial TD	TD[0]
Preserve TD	1 (True)

The channel and TD configurations are similar to Example 2. The **Next TD** parameter of TD[0] is set to TD[1], and vice versa, to chain the transactions.

Table 12. TD[0] Configuration

Parameter	Project Setting
Lower Source Address	LO16(ADC_Delsig_DEC_OUTSAMP_PTR)
Lower Destination Address	LO16(adc_sampleArray1)
Transfer Count	N1×2 (No. of samples × Bytes per sample)
TD properties	Increment Destination Address : TD_INC_DST_ADR Generate DMA done event : DMA__TD_TERMOUT_EN Swap Enable required for PSoC 3 : TD_SWAP_EN
Next TD	TD[1]

Table 13. TD[1] Configuration

Parameter	Project Setting
Lower Source Address	LO16(ADC_Delsig_DEC_OUTSAMP_PTR)
Lower Destination Address	LO16(adc_sampleArray2)
Transfer Count	N2×2 (No. of samples × Bytes per sample)
TD properties	Increment Destination Address : TD_INC_DST_ADR Generate DMA done event : DMA__TD_TERMOUT_EN Swap Enable required for PSoC 3 : TD_SWAP_EN
Next TD	TD[0]

Example 5 Project Files

Eg5_TD_Chaining in the AN52705.zip file that is attached to this application note demonstrates the TD chaining example. The DMA configuration code for the example is similar to Example 1. The arguments passed to the functions are given in the channel and TD configuration tables above. See Appendix D: Example Projects – Test Setup on page 24 for details on how to test this project.

Summary

This application note has described the DMA controller in PSoC 3 and PSoC 5LP. Using simple PSoC Creator example projects, the application note has also shown how to configure the DMA for different types of data transfers. For more advanced information, see the PSoC 3 and PSoC 5LP Technical Reference Manuals and the PSoC Creator DMA component datasheet.

About the Author

Name: Anu M D

Title: Sr. Applications Engineer

Background: BE in Electronics and Communication from Model Engineering College, Cochin.

Contact: anmd@cypress.com

Appendix A: DMA Configuration Steps

Step 1: DMA Channel Initialization

```
Channel_Handle = DMA_DmaInitialize(
             DMA_BYTES_PER_BURST,
             DMA_REQUEST_PER_BURST,
             HI16(Source Address),
             HI16(Destination Address))
```

The API function DmaInitialize() configures several DMA channel parameters as follows:

- ***DMA_BYTES_PER_BURST:*** the number of bytes to be read and written by the DMA channel in one burst

  For example, if you want to define DMA to collect 8-bit ADC data, set this parameter to 1 because the DMA channel must move 1 byte from source to destination on each request. Or, if you want to collect 16-bit ADC data, set this parameter to 2.

- ***DMA_REQUEST_PER_BURST:*** whether each burst must have a separate request.

  If set to 1, each burst transfer must be individually requested. If set to 0, all subsequent bursts after the first burst are automatically carried out without separate request. (Only the first burst transfer must have a DMA request.)

- ***HI16(Source Address):*** the upper 16 bits of the source address. HI16 is a macro created by PSoC Creator to specify the upper 16 bits of a 32-bit value or address.

- ***HI16(DestinationAddress):*** the upper 16 bits of the destination address. Use macros provided in the previous table to identify the upper 16 bits of source and destination addresses in PSoC 3.

  The PSoC 3 Keil compiler stores Keil-specific information in the upper 16 bits of the variable addressess. For this reason, use the following constants shown in Table 14. They are defined in *CyDevice.h* along with HI16 macro to configure the upper 16 bits of source and destination address for PSoC 3 especially when the source or destination for the DMA transfer is RAM or flash memory.

Table 14. Upper 16-bit Address Macros

Source	DMA_SRC_BASE
Peripheral	CYDEV_PERIPH_BASE
RAM	CYDEV_SRAM_BASE
Flash	CYDEV_FLS_BASE

Step2: TD allocation

```
TD_Handle = CyDmaTdAllocate();
```

The API function CyDmaTdAllocate() creates an instance of a TD and returns the handle to that TD. The TD handle is used by other APIs to configure the TD. To create multiple TDs, call the function multiple times.

Step 3: TD configuration

```
CyDmaTdSetConfiguration(TD_Handle,
                        Transfer_Count,
                        Next_TD,
                        TD_Property);
```

The API function CyDmaTdSetConfiguration() configures a TD, using the following parameters:

- ***TD_Handle:*** a handle previously returned by the CyDmaTdAllocate() function
- ***Transfer_Count:*** the total number of bytes to be moved from source to destination.
- ***Next_TD:*** the index of the next TD in the TD chain. If this TD is to be the last in the chain, use the macro DMA_DISABLE_TD (0xFE) as to disable the DMA channel after the TD transfer is complete.
- ***TD_Property:*** use the TD Configuration register flags shown in Table 15 on page 20 to set the properties of the DMA transaction. OR the flags together to configure the TD property. For example, to configure the TD to swap 4 bytes during the data transfer, use:

  ```
  (TD_SWAP_EN | TD_SWAP_SIZE4)
  ```

Table 15. TD Properties

Configuration Flag	Function
TD_SWAP_EN	Perform endian swap; swap bytes while moving data from source to destination.
TD_SWAP_SIZE4	Set swap size = 4 bytes. Default swap size is 2 bytes.
TD_AUTO_EXEC_NEXT	The next TD in the chain is activated automatically when the current TD finishes.
TD_TERMIN_EN	End this TD if a positive edge on the trq input line occurs. The positive edge must occur during a burst. That is the only time the DMAC listens for it.
DMA__TD_TERMOUT_EN	If this flag is used, a pulse is generated on the nrq line when the TD transfer is complete. This flag is specific to a DMA component instance and is defined in the component instance header file. For example, if the DMA component instance name is DMA_1 in the top design, the termout macro for the instance is '*DMA_1__TD_TERMOUT_EN*' which is included in *DMA_1_dma.h.*
TD_INC_DST_ADR	Increments destination address according to the size of each data burst transaction.
TD_INC_SRC_ADR	Increments source address according to the size of each data burst transaction.

Step 4: Configuring TD source and destination

```
CyDmaTdSetAddress(TD_Handle,
                  LO16(source),
                  LO16(destination))
```

The API function CyDmaTdSetAddress() sets the source and destination addresses of a TD, using the following parameters:

- ***TD_Handle:*** a handle previously returned by the CyDmaTdAllocate() function
- ***LO16(source):*** the lower 16 bits of the source address
- ***LO16(destination):*** the lower 16 bits of the destination address

PSoC is highly programmable - many components are created from the programmable digital and analog blocks, and the physical location of a peripheral may change based on the design. Therefore, a conventional register map listing all the source and destination addresses is not possible

Instead, the registers for each component are defined in the component API header files generated by PSoC Creator during the build process. You should review these header files to identify the component's register addresses.

Step 5: Attach the TD to the channel

```
CyDmaChSetInitialTd(Channel_Handle,
                    TD_Handle)
```

The API function CyDmaChSetInitialTD() sets the first TD of a DMA channel:

- ***Channel_Handle:*** the handle of the DMA instance returned by the DMA_DmaInitialize() function
- ***TD_Handle:*** a handle previously returned by the CyDmaTdAllocate() function

Step 6: Enable DMA channel

```
CyDmaChEnable(Channel_Handle,
              Preserve_TD)
```

The API function CyDmaChEnable() enables the DMA channel:

- ***Channel_Handle:*** the handle of the DMA instance returned by the DMA_DmaInitialize() function
- ***Preserve_TD***: if TRUE, the DMA channel retains the TD configurations (source, destination and transfer count) so that the TD can be repeated

Other Important DMA API Functions

To activate a DMA channel from a CPU request, use this function:

```
CyDmaChSetRequest(Channel_Handle, CPU_REQ);
```

To disable a DMA channel, use this function:

```
CyDmaChDisable(Channel_Handle);
```

Appendix B: DMA Wizard Configuration

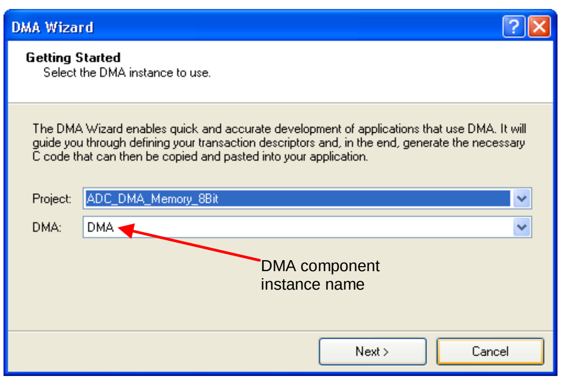
As an alternative to the steps described in Appendix A, the DMA Wizard can make it easy to define the firmware configuration of a DMA channel and TD. However, the wizard supports only a few peripherals as DMA source or destination. If a peripheral is not supported, follow the configuration steps described in Appendix A.

To start the DMA wizard, go to **PSoC Creator > Tools > DMA Wizard.**

Step 1: Select a DMA channel (DMA component instance)

Select the DMA channel to be configured, as Figure 15 shows:

Figure 15. Select DMA Channel



Select the dialog parameters as follows:

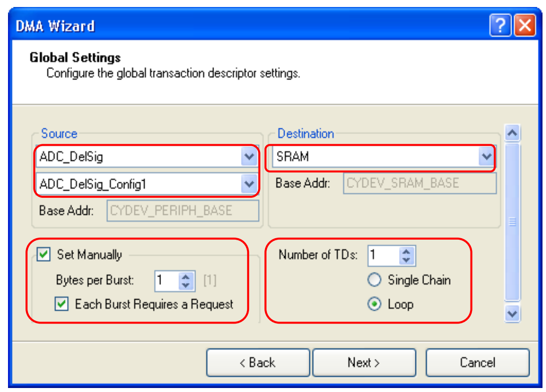
- ***Project:*** name of the PSoC Creator project
- ***DMA:*** the name DMA component instance in your project

Click **Next** when done.

Step 2: Select global settings

Select the DMA transfer global settings, as Figure 16 shows:

Figure 16. Global Settings



Use this dialog to select the DMA channel configuration parameters:

Source and Destination: the upper 16 bits of the source and destination addresses

Bytes per Burst: the number of bytes to be moved in a single burst

Each Burst Requires a Request: whether each burst requires a separate request

Number of TDs: the number of transaction descriptors to be associated with the DMA channel (1 to 128).

Single Chain or Loop: this defines what '**Next TD**' for the last TD in the chain. If single chain, the next TD is DMA_DISABLE_TD (0xFE). If loop, it is the first TD.

Click **Next** when done.

Step 3: Define the transaction descriptors for the channel

Select the DMA transfer global settings, as Figure 17 shows. Table 16 describes each TD configuration parameter.

Figure 17. Add Transaction Descriptors

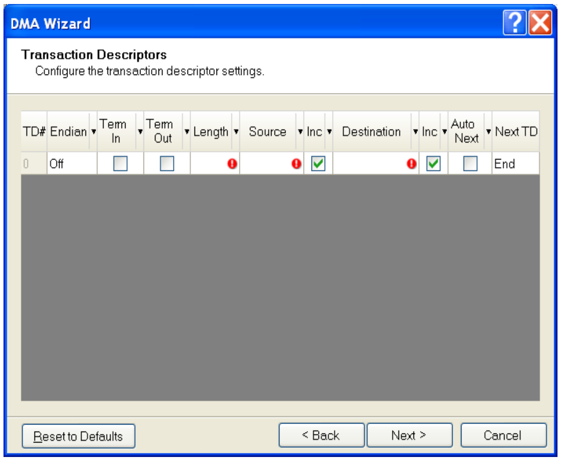


Table 16. TD Configuration Details

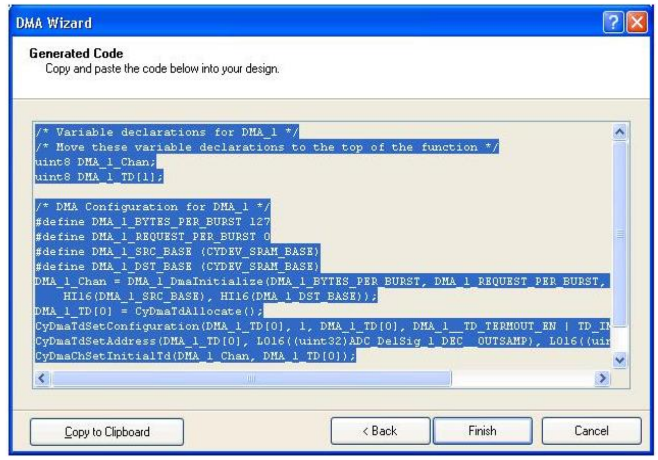
Field	Description
TD#	Displays the logical number for the Transaction Descriptor.
Endian	Enables 2- or 4-byte endian byte swapping. This enables swapping the byte while the data moves from source to destination. The **Bytes per Burst** setting must be set as a multiple of the endian selection. This is usually used for DMA transfers between PSoC 3 memory and peripherals because of the difference in endianess.
Term In	Enables ending the TD transaction on a rising edge of the TERMIN (trq) signal.
Term Out	Enables the creation of the TERMOUT (nrq) signal when the TD finishes.
Length	This specifies the transfer count for the TD in bytes (0 to 4095). This is the total number of bytes that the DMA should transfer to complete the transaction.
Source	The lower 16 bits of the source address for the DMA transfer. A drop-down list of addresses for the source is given by the DMA wizard if the source selected is a component (not memory). You can also edit or enter the source address manually.
Inc (Source)	Enables incrementing of the source address as the DMA does the transaction. If this is enabled, every time the DMA reads the data from source, the source address is incremented by the number of bytes that the DMA has read. The DMA increments the source address until the entire transaction (transfer count) is finished.
Destination	The lower 16 bits of the source address for the DMA transfer. A drop-down list of addresses for the destination is given by the DMA wizard if the destination selected is a component (not memory). You can also edit or enter the destination address manually.
Inc(Destination)	Enables incrementing of the destination address as the DMA does the transaction. The DMA increases the destination address until the entire transaction (transfer count) is finished.
Auto Next	Automatically execute the next TD without another DMA request.
Next TD	The next logical TD in the chain of TDs. Set to END if this TD chain is finished with this TD.

Click **Next** when done. The required code is then generated.

Step 4: Copy the code created by the DMA Wizard

After the DMA channels and TD configuration are finished, the wizard creates code for the DMA channel. This code includes the configuration for the DMA channel and the TDs The code is generated in a window in the DMA Wizard dialog, as Figure 18 shows. To use the code, select all in the window, copy it, and paste it in your *main*.c.

Figure 18. Generated Code

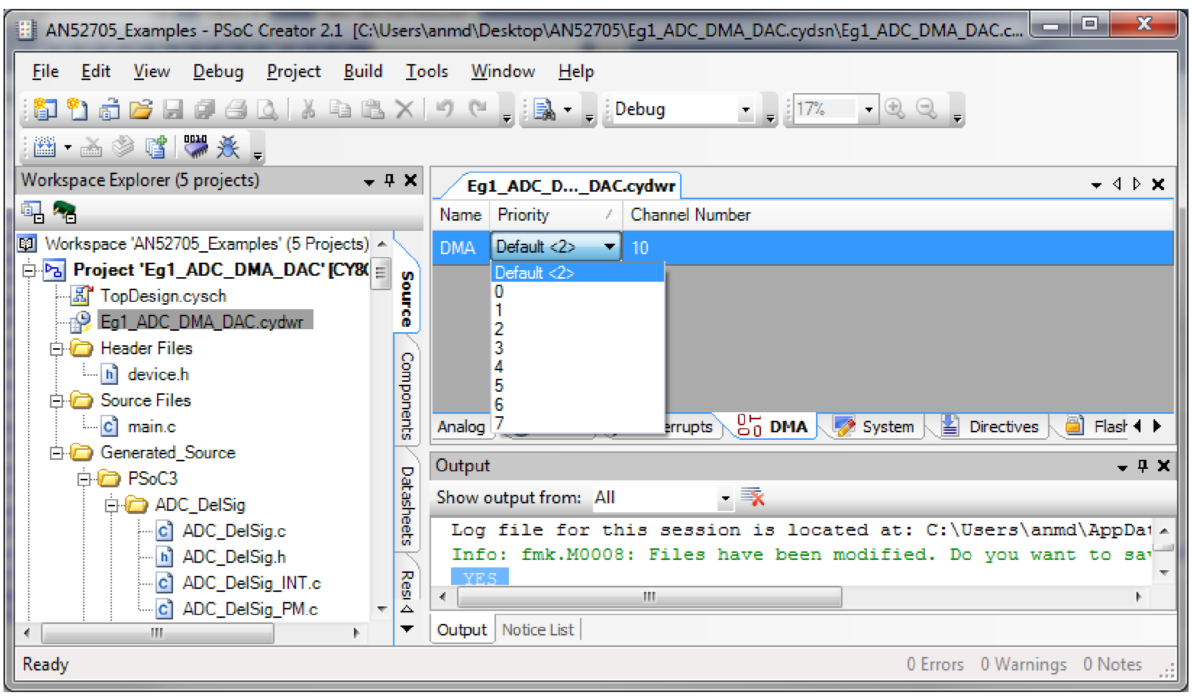


For more information on the wizard, see the PSoC Creator Help file.

Appendix C: Setting DMA Channel Priority

When multiple DMA channel requests are active, the DMA channels are processed by DMAC based on channel priority settings. Each DMA channel can be given one of the eight different priorities. DMA channel priority is set in PSoC Creator in **Design Wide Resources (*.cydwr) > DMA**, as Figure 19 shows.

Figure 19. Setting DMA Channel Priority



When both the CPU and DMAC request access to the same spoke on PHUB at the same time, the CPU has priority by default. The PHUB manages arbitration between DMA and CPU, and among the DMA channels. For more information, see PSoC® 3, PSoC® 5LP Architecture TRM.

Appendix D: Example Projects – Test Setup

Example 1: Peripheral-to-Peripheral Transfer – Eg1_ADC_DMA_DAC

In this example project the ADC sampling frequency (fs) is 384 kHz. The output is reconstructed best if the input frequency is less than or equal to ~84kHz because the delta sigma ADCs have a low pass nature with a -3dB drop at 0.22 fs. The test setup is as follows:

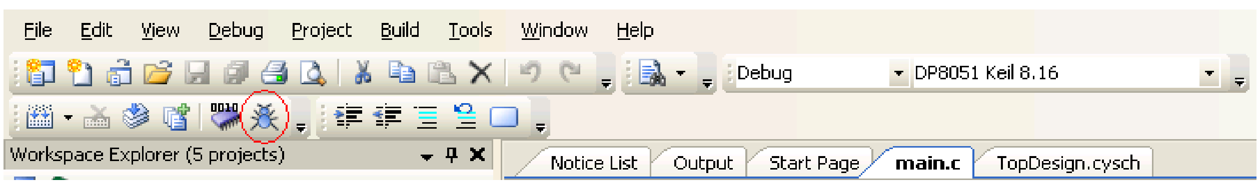
1. Connect the function generator to pin P0[2], the input to the ADC.
2. Set the function generator to make a sine wave of 100 Hz.
3. Connect the oscilloscope probe to pin P0[0], the VDAC output.
4. Build the project and program the device.
5. Look at the output from pin P0[0] on the oscilloscope. It should be a sine wave of frequency 100 Hz, the same as the input.

Example 2: Peripheral-to-Memory Transfer – Eg2_ADC_DMA_Mem

The test setup is as follows:

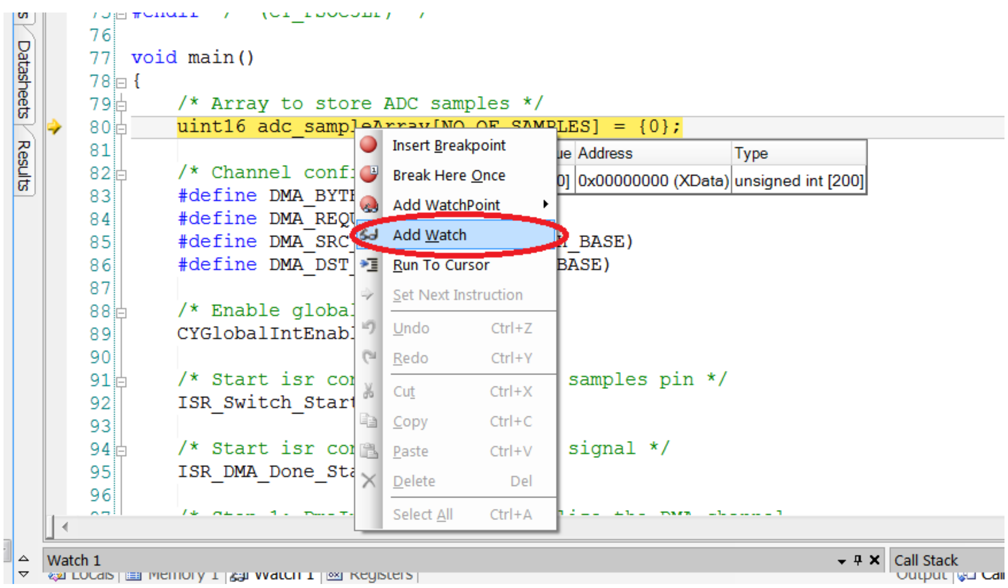
1. Connect the input signal to pin P0[2], the input to the ADC. Make sure that the input is within the ADC range V_{SSA} to 2.048 V.
2. Connect P6[1] to switch(SW1) on the DVK.
3. Build the project.
4. Press F5 or click the debug icon, as Figure 20 shows, to download the program and start debugging.

   Figure 20. Debug Button



5. Add adc_sampleArray as a watch variable, as Figure 21 shows:

   Figure 21. Watch Variable



6. Put a breakpoint inside the `if(DMADone_flag)` loop, as Figure 22 shows:

Figure 22. Add a Breakpoint

```
        }/* If statement ends here */

        /* DMADone_flag is set inside ISR_DMA_Done after the
         * of ADC samples are buffered */
        if(DMADone_flag)
        {
            /* Put a break point here to view the data in the
            DMADone_flag = 0;

        }/* If statement ends here */

    }/* for loop ends here */

    /* Place your application code here. */
}
```

7. Press F5 to run the program. Press the switch (SW1) connected to P6[1] to enable the DMA to start ADC sample buffering. The execution stops at the breakpoint after the DMA has transferred the specified number of samples from ADC to memory. The result can be verified by monitoring the **adc_sampleArray** in the watch window, as Figure 23 shows:

Figure 23. ADC Samples in Watch Window

Watch 1

Name	Value	Address	Type	Radix
adc_sampleArray	[200]	0x00000000 (XData)	unsigned int [200]	Default
0	0x7FFE	0x00000000 (XData)	unsigned int	Default
1	0x7FFD	0x00000002 (XData)	unsigned int	Default
2	0x7FFE	0x00000004 (XData)	unsigned int	Default
3	0x7FFF	0x00000006 (XData)	unsigned int	Default
4	0x7FFC	0x00000008 (XData)	unsigned int	Default
5	0x7FFD	0x0000000A (XData)	unsigned int	Default
6	0x7FFE	0x0000000C (XData)	unsigned int	Default

Example 3: Memory-to-Peripheral Transfer – Eg3_Mem_DMA_DAC

The test setup is as follows:

1. Connect the oscilloscope probe to pin P0[0], the VDAC output.
2. Build the project and program the device.
3. Observe a sine wave of frequency 7.8 kHz on the oscilloscope.

Example 4: Memory-to-Memory Transfer – Eg4_Mem_DMA_Mem

The test setup is as follows:

1. Connect a character LCD module to header P18 (LCD Module - Port 2) of the CY8CKIT-001 PSoC Development Kit.
2. Make sure jumper J12 is in the ON position to power the LCD.
3. Build the project and program the device.
4. Look at the LCD display. The first row displays the contents of the destination array. Initially all values are zero. After a delay of one second the first row displays 00 to 07, showing that the DMA has successfully transferred the data from flash to RAM. The second row displays the message TRANSFERRED. Figure 24 shows an example of the LCD display:

Figure 24. LCD Display of DMA Transfer

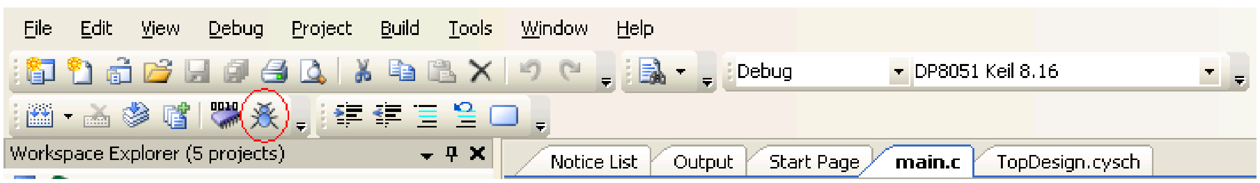
```
0001020304050607
TRANSFERRED
```

Example 5: TD Chaining– Eg5_TD_Chaining

The test setup for this example is same as that of Example 2. The test setup is as follows:

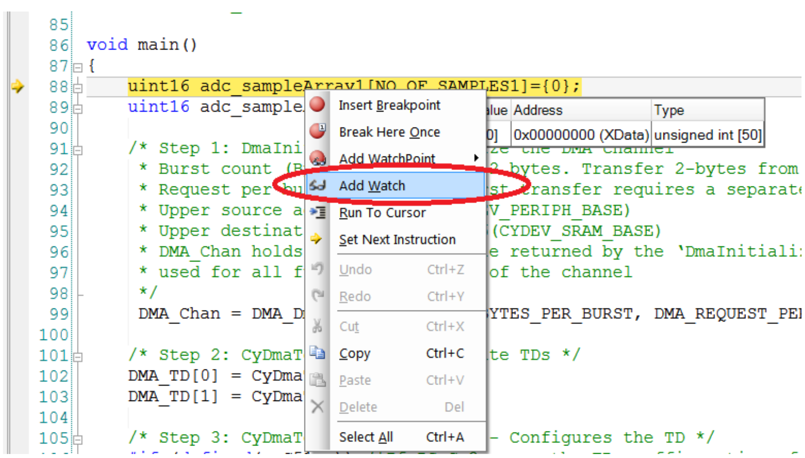
1. Connect the input signal to pin P0[2], the input to the ADC. Make sure that the input is within the ADC range V_{SSA} to 2.048 V.
2. Connect P6[1] to switch(SW1) on the DVK.
3. Build the project.
4. Press F5 or click the debug icon, as Figure 25 shows, to download the program and start debugging.

   Figure 25. Debug Button



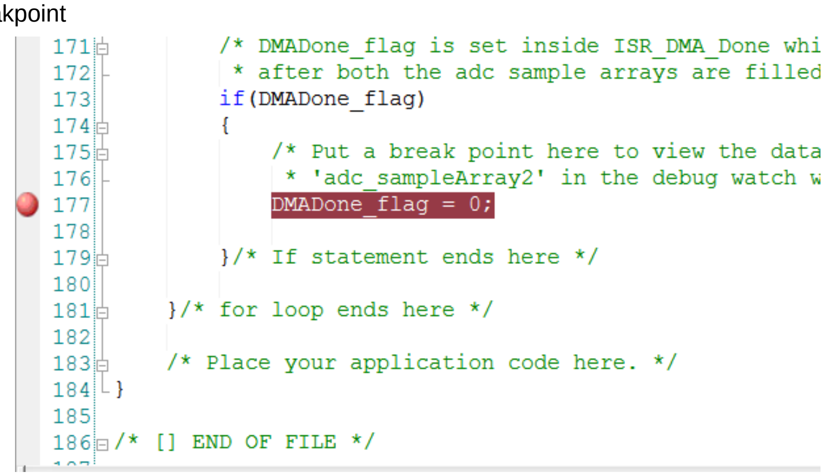
5. Add adc_samplearray1 and adc_samplearray2 as watch variables as Figure 26 shows.

   Figure 26. Watch Variables



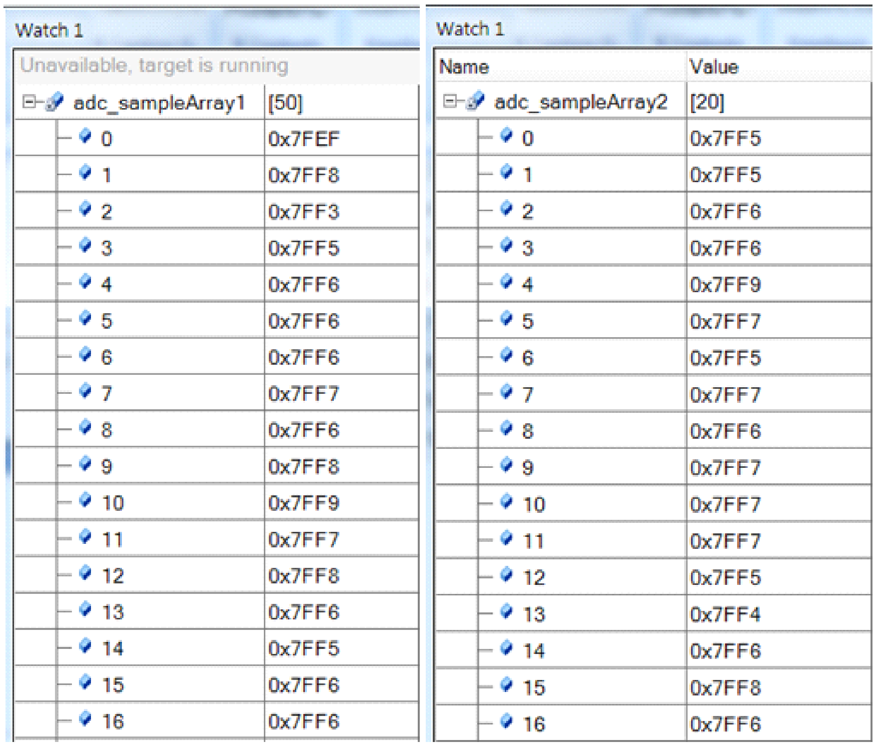
6. Put a breakpoint inside the `if(DMADone_flag)` loop, as Figure 27 shows.

   Figure 27. Add a Breakpoint



7. Press F5 to run the program. Press the switch (SW1) connected to P6[1] to enable the DMA to start ADC sample buffering. The execution stops at the breakpoint after the DMA has transferred the specified number of samples from ADC to memory. To verify the result, monitor adc_sampleArray1 and adc_sampleAv rray2 in the watch window, as Figure 28 shows.

Figure 28. ADC Samples in Watch Window



Watch 1

Unavailable, target is running

adc_sampleArray1	[50]
0	0x7FEF
1	0x7FF8
2	0x7FF3
3	0x7FF5
4	0x7FF6
5	0x7FF6
6	0x7FF6
7	0x7FF7
8	0x7FF6
9	0x7FF8
10	0x7FF9
11	0x7FF7
12	0x7FF8
13	0x7FF6
14	0x7FF5
15	0x7FF6
16	0x7FF6

Watch 1

Name	Value
adc_sampleArray2	[20]
0	0x7FF5
1	0x7FF5
2	0x7FF6
3	0x7FF6
4	0x7FF9
5	0x7FF7
6	0x7FF5
7	0x7FF7
8	0x7FF6
9	0x7FF7
10	0x7FF7
11	0x7FF7
12	0x7FF5
13	0x7FF4
14	0x7FF6
15	0x7FF8
16	0x7FF6

Appendix E: Frequently Asked Questions:

1. How can you buffer more than 4095 bytes using DMA?

   The maximum transfer count of a TD is limited to 4095 bytes. If you need to transfer more than 4095 bytes using a single DMA channel, use multiple TDs and chain them as shown in Example 5.

2. How do you find the source and destination addresses of the peripherals for DMA data transfer?

   PSoC is highly programmable - many components are created from the programmable digital and analog blocks, and the physical location of a peripheral may change based on the design. Therefore, a conventional register map listing all the source and destination addresses is not possible

   Instead, the registers for each component are defined in the component API header files generated by PSoC Creator during the build process. You should review these header files to identify the component's register addresses.

3. How do you use DMA with communication protocols such as UART, SPI etc.?

   When using communication protocols such as UART and SPI with DMA, set the buffer size to 4 or less so that internal interrupts are not triggered for data transfers . Use hardware FIFO pointers as read and write data addresses for the DMA and trigger the DMA using FIFO level status configured as interrupts. Make the Hardware Request of the DMA channel as level triggered in order to use it with FIFO levels.

4. Timing Details of DMA transfer?

   The timing details of DMA transfer can be found in the PSoC 3, PSoC 5LP Technical Reference Manual. A detailed discussion on DMA timing is beyond the scope of this application note.

Document History

Document Title: PSoC® 3 and PSoC 5LP - Getting Started with DMA – AN52705

Document Number: 001-52705

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2710860	LNAT	05/25/09	New Application Note.
*A	2768731	LNAT	09/24/09	Updated the projects for PSoC Creator Beta 3 version. Added information about configuring the Termout signals
*B	2951774	LNAT	06/14/10	Updated the projects for PSoC Creator Beta 4.1 Added more information regarding the DMA configuration
*C	2966485	LNAT	08/26/10	Updated the projects for PSoC Creator Beta 5. Used DMA Wizard in the projects.
*D	3269575	LRDK	06/06/11	Rewritten in Simplified English.
*E	3355465	ANUP	08/26/2011	Updated Introduction Updated TD0 Configuration section Updated channel configuration table Updated Figure 8 Updated operation section.
*F	3444066	ANMD	11/22/2011	Project updates for PSoC Creator 2.0. Updated template.
*G	3822782	ANCY	11/27/2012	Updated for PSoC 5LP.
*H	3844498	ANMD	12/18/2012	Re-written to improve clarity

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com