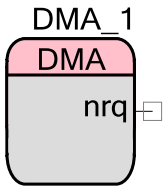
Direct Memory Access (DMA)
1.0



Features

- 24 channels
- Eight priority levels
- 128 Transaction Descriptors (TD)
- 8, 16, and 32-bit data transfers
- Configurable source and destination endianess
- Can generate an interrupt when data transfer is complete

General Description

The DMA component allows data transfers to and from memory, components and registers. The controller supports 8, 16, and 32-bit data transfers, and can be configured to correctly transfer data between a source and destination that have different endianess. Transaction Descriptors can be chained together for complex operations.

DMA has two parts; a component that is placed into the design and an API. The API can be used without placing a component into the design if the firmware is aware of the source and destination addresses.

When to use a DMA component

When you want to unburden the CPU of the task of transferring data or when data needs to be transferred in a predictable way that can be setup before hand. A few basic use cases are:

- Memory to memory
- Memory to peripheral
- Peripheral to memory
- Peripheral to peripheral

Transaction Descriptors can be executed individually or chained together to perform complex transfers.

SRAM Access

The DMA Controller cannot see SRAM from 0x1FFF8000 to 0x1FFFFFFF, but it can see the same memory at 0x20008000 to 0x2000FFFF.

The CPU sees:

```
0x1FFF8000 - 0x1FFFFFFF C-BUS 32KB
0x20000000 - 0x20007FFF S-BUS 32KB
```

The DMA Controller sees:

```
0x20000000 - 0x20007FFF S-BUS 32KB
0x20008000 - 0x2000FFFF C-BUS 32KB
```

Any DMA access to 0x1FFF8000 - 0x1FFFFFFF needs to add 64k to the address to shift the address to the DMA Controller's view. This is true for source or destination.

Input/Output Connections

This section describes the various input and output connections for the DMA. An asterisk (*) in the list of I/O's states that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

nrq – Output

The nrq terminal may be connected to an interrupt, or to a component to notify the component of the completion of the DMA transfer.

drq – Input *

The drq terminal is connected to a component that is capable of requesting a DMA transaction.

The drq input is level-sensitive. The DMA request will continuously occur when drq is HIGH. If drq is driven from a component that is level-sensitive but is needed as a narrow pulse, you must add additional components to accomplish this.

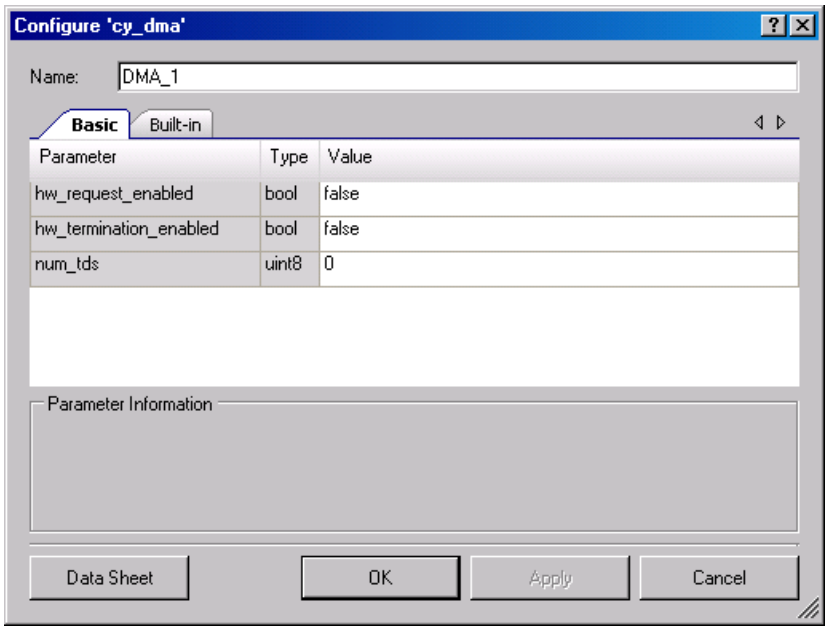
trq – Input *

The trq terminal is connected to a component that is capable of terminating a DMA transaction. A Component may be asked for data from the DMA controller when it knows none is available. It uses this signal to terminate the transaction. This signal is only used when the channel is trying to transfer data. A positive edge on this line at other times is ignored.

Parameters and Setup

Drag a DMA component onto your design and double-click it to open the Configure dialog.

Figure 1 Configure DMA Dialog



The DMA provides the following parameters.

hw_request_enabled

If enabled, adds the terminal, drq, that allows a DMA request to be made from hardware.

hw_termination_enabled

If enabled, adds the terminal, trq, that allows a DMA request to be terminated from hardware.

num_tds

This parameter specifies the number of Transaction Descriptors needed to support the data transfer.

priority

This parameter specifies the priority level of the DMA channel. Each DMA channel is assigned one of eight priorities, 0 to 7 with priority 0 the highest. This parameter is configured in the DMA Editor in Design-Wide Resources (in the project's cydwr file).

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "DMA_1" to the first instance of a component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "DMA."

Function	Description
APIs For This DMA Instance	
DMA_DmaInitialize	Allocates and initializes a DMAC channel to be used by the caller.
DMA_DmaRelease	Frees and disables the DMA channel associated with this instance of the component.
DMA Library APIs (APIs Shared by All Instances of DMA)	
CyDmacConfigure	Sets the DMAC Configuration register with the default values.
CyDmacError	Gets the error bits from the DMAC.
CyDmacClearError	Clears the error bits in the error register of the DMAC.
CyDmacErrorAddress	Get the address where the last DMAC error occurred.
CyDmaChAlloc	Allocates a channel of the DMAC to be used by the caller.
CyDmaChFree	Frees a channel allocated by CyDmaChAlloc.
CyDmaChEnable	Enables the DMA channel for execution.
CyDmaChDisable	Disables the DMA channel.
CyDmaChPriority	Sets the priority of a DMA channel.
CyDmaChSetExtendedAddress	Sets the high 16 bits of the source and destination addresses.
CyDmaChSetInitialTd	Set the initial TD for the channel.
CyDmaChSetRequest	Request to terminate a chain of TD's, one TD or start the DMA.
CyDmaChGetRequest	Checks if the CyDmaChSetRequest request was satisfied.
CyDmaChStatus	Determines the status of the current Transaction descriptor.
CyDmaChSetConfiguration	Sets Configuration information for the channel.
CyDmaTdAllocate	Allocates a Transaction Descriptor from the free list for use.
CyDmaTdFree	Returns a Transaction Descriptor back to the free list.

Function	Description
CyDmaTdFreeCount	Gets the number of free Transaction Descriptors available.
CyDmaTdSetConfiguration	Configures the Transaction Descriptor.
CyDmaTdGetConfiguration	Gets the configuration for the Transaction Descriptor.
CyDmaTdSetAddress	Sets the lower 16 bits of the source and destination addresses.
CyDmaTdGetAddress	Gets the lower 16 bits of the source and destination addresses.

uint8 DMA_DmaInitialize(uint8 burstCount, uint8 requestPerBurst, uint16 upperSrcAddress, uint16 upperDestAddress)

Description: Allocates and initializes a DMAC channel to be used by the caller.

Parameters: (uint8) burstCount. Specifies the size of bursts (1 to 127) this TD should be divided into. If this value is zero then the whole transfer is done is one burst.

(uint8) requestPerBurst. The whole data can be split into multiple burst. If multiple bursts is required to complete:

Value	Action
0	All subsequent bursts after the first burst will be automatically requested and carried out
1	All subsequent bursts after the first burst must also be individually requested.

(uint16) upperSrcAddress. Upper 16 bits of the source address.

(uint16) upperDestAddress. Upper 16 bits of the destination address.

Return Value: (uint8) The channel that can be used by the caller for DMA activity. DMA_INVALID_CHANNEL (0xFF) if there are no channels left.

Side Effects: None

void DMA_DmaRelease(void)

Description: Frees the channel associated with this instance of the component. The channel cannot be used again unless DMA_DmaInitialize is called again.

Parameters: None

Return Value: None

Side Effects: None

void CyDmacConfigure(void)

Description: Creates a linked list of all the TD's to be allocated. This function is called by the startup code and does not normally need to be called by the user. This function could be called by the user if all the Dma channels are inactive.

Parameters: None

Return Value: None

Side Effects: None

uint8 CyDmacError(void)

Description: Return the value of the DMA_ERROR type, which contains the error types for the last failed DMA transaction.

Parameters: None

Return Value: Returns the error data (4 bits) from the DMA_ERROR type.

Bit	Define	Description
Bit 3	DMAC_PERIPH_ERR	Set to 1 when a peripheral responds to a bus transaction with an error response. Cleared by writing a 1.
Bit 2	DMAC_UNPOP_ACC	Set to 1 when an access is attempted to an invalid address. Cleared by writing a 1.
Bit 1	DMAC_BUS_TIMEOUT	Set to 1 when a bus timeout occurs. Cleared by writing a 1. Timeout values are determined by the BUS_TIMEOUT field in the PHUBCFG register.

Side Effects: None

void CyDmacClearError(uint8 error)

Description: Clears the error bits in the error register of the DMAC.

Parameters: uint8 error. Bitmask of the error bits to clear in the DMA_ERROR type.

Bit	Define	Description
Bit 3	DMAC_PERIPH_ERR	Set to 1 when a peripheral responds to a bus transaction with an error response. Cleared by writing a 1.
Bit 2	DMAC_UNPOP_ACC	Set to 1 when an access is attempted to an invalid address. Cleared by writing a 1.
Bit 1	DMAC_BUS_TIMEOUT	Set to 1 when a bus timeout occurs. Cleared by writing a 1. Timeout values are determined by the BUS_TIMEOUT field in the PHUBCFG register.

Return Value: None

Side Effects: None

uint32 CyDmacErrorAddress(void)

Description: When a BUS_TIMEOUT, UNPOP_ACC and PERIPH_ERR occurs the address of the error is written to the error address register and can be read with this function. If there are multiple errors, only the address of the first error is saved.

Parameters: None

Return Value: The address that caused the error.

Side Effects: None

uint8 CyDmaChAlloc(void)

Description: Allocates a channel from the DMAC to be used in all functions that require a channel handle.

Parameters: None

Return Value: The allocated channel number. Zero is a valid channel number. DMA_INVALID_CHANNEL is returned if there are no channels available.

Side Effects: None

cystatus CyDmaChFree(uint8 chHandle)

Description: Frees a channel handle allocated by CyDmaChAlloc.

Parameters: uint8 chHandle. The handle previously returned by CyDmaChAlloc or Dma_DmaInitalize.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

cystatus CyDmaChEnable(uint8 chHandle, uint8 preserveTds)

Description: Enables the DMA channel. A software or hardware request still needs to happen before the channel will be executed.

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitalize.

Parameters: uint8 preserveTds. Preserve the TDs transfer count, source and destination address's after the TD transfer is complete.

Value	Action
0	DMA controller should set Values of TDs to reflect current state of TD execution
1	DMA controller should restore the original configuration values of the TD.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

cystatus CyDmaChDisable(uint8 chHandle)

Description: Disables the DMA channel. Once this function is called. CyDmaChStatus may be called to determine when the channel is disabled and determine which TDs were being executed.

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitalize.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

cystatus CyDmaChPriority(uint8 chHandle, uint8 priority)

Description: Sets the priority of a DMA channel.

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitalize.

uint8 priority. The priority to set the channel to, 0 - 7.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

cystatus CyDmaChSetExtendedAddress(uint8 chHandle, uint16 source, uint16 destination)

Description: Sets the high 16 bits of the source and destination addresses for the DMA channel (all TD's in the chain).

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitalize.

uint16 source. 16 bit address of the DMA transfer source.

uint16 destination. 16 bit address of the DMA transfer destination.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

cystatus CyDmaChSetInitialTd(uint8 chHandle, uint8 startTd)

Description: Set the initial TD to be executed for the channel when the CyDmaChEnable function is called.

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitialize.

uint8 startTd. Index of TD to set as the first TD associated with the channel. Zero is a valid TD Index.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

cystatus CyDmaChSetRequest(uint8 chHandle, uint8 request)

Description: Allows the caller to terminate a chain of TDs, terminate one TD, or create a direct request to start the DMA channel.

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitalize.

uint8 request. One of the following constants. Each of the constants is a 3-bit value.

Request Values	Description
CPU_REQ	Create a direct request to start the DMA channel
CPU_TERM_TD	Terminate one TD
CPU_TERM_CHAIN	Terminate a chain of TDs

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

cystatus CyDmaChGetRequest(uint8 chHandle)

Description: This function allows the caller of CyDmaChSetRequest to determine if the request was completed.

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitalize.

Return Value: Returns a 3-bit field corresponding to the 3 bits of the request that describes the state of the previously posted request. If the value is zero, the request was completed.

DMA_INVALID_CHANNEL if the handle is invalid.

Side Effects: None

cystatus CyDmaChStatus(uint8 chHandle, uint8 * currentTd, uint8 * state)

Description: Determines the status of the DMA Channel.

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitalize.

uint8 * currentTd. Address to store the Index of the current Transaction Descriptor. Can be NULL if the value is not needed.

uint8 * state. Address to store the State of the Channel. Can be NULL if the value is not needed.

Bit 1	STATUS_CHAIN_ACTIVE	0: channel is not currently being serviced by DMAC
		1: channel is currently being serviced by DMAC
Bit 0	STATUS_TD_ACTIVE	0: TD chain is inactive; either no DMA requests have triggered a new chain or the previous chain has completed.
		1: TD chain has been triggered by a DMA request

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

cystatus CyDmaChSetConfiguration(uint8 chHandle, uint8 burstCount, uint8 requestPerBurst, uint8 tdDone0, uint8 tdDone1, uint8 tdStop)

Description: Sets Configuration information for the channel.

Parameters: uint8 chHandle. A handle previously returned by CyDmaChAlloc or Dma_DmaInitialize.

uint8 burstCount. Specifies the size of small bursts (1 to 127) this TD should be divided into. If this value is zero then the whole transfer is done is one burst.

(uint8) requestPerBurst. The whole data can be split into multiple burst. If multiple bursts is required to complete:

Value	Action
0	All subsequent bursts after the first burst will be automatically requested and carried out
1	All subsequent bursts after the first burst must also be individually requested.

uint8 tdDone0. Selects one of the TERMOUT0 interrupt lines to signal completion. The line connected to the nrq terminal will determine the TERMOUT0_SEL definition and should be used as supplied by cyfitter.h

uint8 tdDone1. Selects one of the TERMOUT1 interrupt lines to signal completion. The line connected to the nrq terminal will determine the TERMOUT1_SEL definition and should be used as supplied by cyfitter.h

uint8 tdStop. Selects one of the TERMIN interrupt lines to signal to the DMAC that the TD should terminate.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if chHandle is invalid.

Side Effects: None

uint8 CyDmaTdAllocate(void)

Description: Allocates a Transaction Descriptor for use with an allocated DMA channel.

Parameters: None

Return Value: Zero based index of the Transaction Descriptor to be used by the caller. Zero is a valid TD index.

DMA_INVALID_TD if there are no free TDs available.

Side Effects: None

void CyDmaTdFree(uint8 tdHandle)

Description: Returns a Transaction Descriptor to the free list.

Parameters: uint8 tdHandle. Zero based index of the Transaction Descriptor to free.

Return Value: None

Side Effects: None

uint8 CyDmaTdFreeCount(void)

Description: Returns the number of free Transaction Descriptors available to be allocated.

Parameters: None

Return Value: The number of free Transaction Descriptors.

Side Effects: None

cystatus CyDmaTdSetConfiguration(uint8 tdHandle, uint16 transferCount, uint8 nextTd, uint8 configuration)

Description: Configures a Transaction Descriptor.

Parameters: uint8 tdHandle. A handle previously returned by CyDmaTdAlloc.

uint16 transferCount. Size of the data transfer (in bytes) for this Transaction Descriptor. Transfer count is limited to 0x0FFF. A larger value will cause the function to return CYRET_BADPARAM.

uint8 nextTd. Zero based index of the next Transaction Descriptor in the TD chain. Zero is a valid index to the next TD, DMA_INVALID_TD (0xFF) is end of chain.

uint8 configuration. Configuration bit field corresponding to bits 24 – 31 in the PHUB_TDMEMX_ORIG_TD0 register.

Configuration Options	Description
TD_SWAP_EN	Perform endian swap
TD_SWAP_SIZE4	Swap size = 4 bytes
TD_AUTO_EXEC_NEXT	The next TD in the chain will trigger automatically when the current TD completes
TD_TERMIN_EN	Terminate this TD if a positive edge on the "trq" input line occurs. The positive edge has to occur during a burst. That is the only time the DMAC will listen for it.
TD_TERMOUT1_EN	When this TD completes the TERMOUT1 signal selected by TERMOUT1_SEL will toggle if this bit is set.
TD_TERMOUT0_EN	When this TD completes the TERMOUT0 signal selected by TERMOUT0_SEL will toggle if this bit is set.
TD_INC_DST_ADR	Increment DST_ADR according to the burstCount set for the channel.
TD_INC_SRC_ADR	Increment SRC_ADR according to the burstCount set for the channel.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if tdHandle is invalid.

Side Effects: None

cystatus CyDmaTdGetConfiguration(uint8 tdHandle, uint16 * transferCount, uint8 * nextTd, uint8 * configuration)

Description: Retrieves the configuration of the Transaction Descriptor. If a NULL pointer is passed as a parameter, that parameter will be skipped. The user may request only the values they are interested in.

Parameters: uint8 tdHandle. A handle previously returned by CyDmaTdAlloc.

uint16 * transferCount. Address to store the size of the data transfer (in bytes) for this Transaction Descriptor (TD).. A size of zero will cause the transfer to go indefinitely.

uint8 * nextTd. Address to store the Zero based index of the next TD in the TD chain.

uint8 * configuration. Address to store the Bit field of configuration bits. See CyDmaTdSetConfiguration.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if tdHandle is invalid.

Side Effects: If a TD has a transfer count of N and is executed so the transfer count is 0 and then gets re-executed the Transfer count of zero will be interpreted as do forever. Be careful when requesting a td with a transfer count of zero.

cystatus CyDmaTdSetAddress(uint8 tdHandle, uint16 source, uint16 destination)

Description: Sets the lower 16 bits of the source and destination addresses for this TD only.

Parameters: uint8 tdHandle. A handle previously returned by CyDmaTdAlloc.

uint16 source. Lower 16 address bits of the Source of the data transfer.

uint16 destination. Lower 16 address bits of the Destination of the data transfer.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if tdHandle is invalid.

Side Effects: None

cystatus CyDmaTdGetAddress(uint8 tdHandle, uint16 * source, uint16 * destination)

Description: Retrieves the lower 16 bits of the source and/or destination addresses for this TD only. if NULL is passed for a pointer parameter, that value will be skipped. The user may request only the values of interest.

Parameters: uint8 tdHandle. A handle previously returned by CyDmaTdAlloc.

uint16 * source. Address to store the lower 16 address bits of the Source of the data transfer.

uint16 * destination. Address to store the lower 16 address bits of the Destination of the data transfer.

Return Value: CYRET_SUCCESS if successful.

CYRET_BAD_PARAM if tdHandle is invalid.

Side Effects: None

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the DMA component. This example assumes the component has been placed in the schematic and renamed to DMA_1.

An interrupt component named "isr_1" has also been placed on the schematic and connected to the nrq terminal of the DMA_1 component.

```
#include <device.h>

uint8 Finished = 0;

CY_ISR(DmaDone)
{
    Finished = 1;
}

void main(void)
{
    uint8 MyTD;
    uint8 MyChannel;

    /* Perform dma in one burst.*/
    MyChannel = DMA_1_DmaInitialize(0,
                                    0,   /* Automatically request carry out bursts.*/
                                    0,   /* upper address bits are zero. */
                                    0);  /* upper address bits are zero. */

    /* Get a Transaction Descriptor. */
    MyTD = CyDmaTdAllocate();
    if(MyTD == DMA_INVALID_TD)
    {
```

```
        /* Error Condition. */
    }

    /* Setup a TD. */
    /* Set TD to transfer 100 bytes with no next TD, */
    CyDmaTdSetConfiguration(MyTD,
                            100,
                            DMA_INVALID_TD,
                            TD_INC_DST_ADR | TD_INC_SRC_ADR | TD_TERMOUT0_EN);

    /* Copy from 0x2000 to 0x3000. */
    CyDmaTdSetAddress(MyTD, 0x2000, 0x3000);

    /* Associate the TD with the channel. */
    CyDmaChSetInitialTd(MyChannel, MyTD);

    /* Setup the Interrupt connected to the nrq terminal. */
    isr_1_SetVector(DmaDone);
    isr_1_SetPriority(7);
    isr_1_Enable();

    /* Enable the channel. */
    CyDmaChEnable(MyChannel, 0);

    /* Request DMA action. */
    CyDmaChSetRequest(MyChannel, CPU_REQ);

    /* Wait for the interrupt to signal completion. */
    while(!Finished)
        ;

    /* We are done with the DMA and Interrupt components. */
    isr_1_Disable();
    DMA_1_DmaRelease();
    while(1);
}
```

References

Not applicable

DC and AC Electrical Characteristics

Not applicable

PRELIMINARY