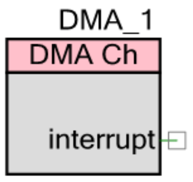
Direct Memory Access (DMA_PDL)

2.0

Features



- Devices support up to two DMA hardware blocks
- Each DMA block supports up to 16 DMA channels
- Supports channel descriptors in SRAM
- 4 Priority Levels for each channel
- Byte, Halfword (2 bytes), and Word (4 bytes) transfers
- Configurable source and destination addresses
- Configurable transfer modes: single transfer, 1D transfer (using X loop) and 2D transfer (using both X and Y loops).
- Configurable input trigger operation:
  - Single Data Element transfer per trigger
  - One X (inner) loop transfer per trigger
  - Entire descriptor per trigger
  - Entire descriptor chain per trigger
- Configurable output trigger
- Configurable interrupt generation

General Description

The DMA Component transfers data to and from memory and registers. These transfers occur independent of the CPU. The DMA transfers can be set up in a byte, halfword (2 bytes), or word (4 bytes) wide. The DMA starts each transaction through an external trigger that can come from a DMA channel (including itself), another DMA channel, a peripheral, or the CPU. The DMA is best used to offload data transfer tasks from the CPU.

When to Use a DMA

The DMA Channel Component can be used in any project that needs to transfer data without CPU intervention based on a hardware trigger signal from another Component.

A common use is transferring data from memory to a peripheral, such as a UART. The DMA can be triggered by the UART FIFO not full signal. The DMA will load data in the UART until the FIFO fills.

The DMA can also be used to take data out of the UART and place it in memory. For example, the DMA can be triggered by the FIFO not empty signal, thus the DMA will transfer data as long as the FIFO is not empty.

Another common use is transferring data from the ADC to memory. The ADC's end of conversion (eoc) signal can be used to trigger the DMA to transfer the ADC result to memory.

The DMA can also be used to move blocks of memory from one memory location to another (RAM to RAM, FLASH to RAM); DMA cannot write to FLASH.

Each DMA channel can be triggered by a hardware signal as described above, or by a firmware register write, or both.

Each DMA channel can be associated to a descriptor. User can create multiple descriptors and have them chained to each other. The user can also reconfigure a descriptor while it is not the active descriptor in the DMA.

Definitions

- **DMA** – Direct Memory Access. A general term that denominates the principle of data transfer without CPU involving.
- **DMA controller** (or block) – HW block in the PSoC 6 MCU; see the section “DMA Controller” of the device *Technical Reference Manual (TRM)* for more information.
- **DMA channel** –Separate single channel of the DMA block.
- **DW** – Data Wire. Another name of the HW DMA block. It is used a few times in the API as reference to the certain DMA block/channel number.
- **DMA driver** – This is a part of the Peripheral Driver Library (PDL) to manage the DMA channel.
- **DMA Component** – Refers to the PSoC Creator DMA_PDL Component which manages a DMA channel using a DMA driver.
- **Descriptor** – a descriptor that sets up the transfer parameters for a DMA channel transfer. The descriptor is initialized in SRAM and reference using a pointer in the DMA channel. Multiple descriptors can be chained.

Quick Start

1. Drag a "DMA [v2.0]" Component from the Component Catalog/System folder onto your schematic (the placed instance takes the name DMA_1).
2. Double-click to open the Configure dialog.
3. Set up the desired DMA settings (input/output, channel/descriptor settings, etc.).
4. Connect the input/output terminals.
5. Build the project in order to verify the correctness of your design. This will add the required PDL modules to the Workspace Explorer and generate the descriptor configuration data and the descriptor declaration(s) for the DMA_1 instance.
6. In *main.c*, initialize the peripheral and start the application:

```
uint32_t src, dst;
cy_stc_dma_channel_config_t channelConfig =
{
    .descriptor  = &DMA_1_Descriptor_1,
    .preemptable = DMA_1_PREEMPTABLE,
    .priority    = DMA_1_PRIORITY,
    .enable      = false
};

(void)Cy_DMA_Descriptor_Init(&DMA_1_Descriptor_1,
&DMA_1_Descriptor_1_config);
(void)Cy_DMA_Channel_Init(DMA_1_HW, DMA_1_DW_CHANNEL, &channelConfig);
Cy_DMA_Descriptor_SetSrcAddress(&DMA_1_Descriptor_1, &src);
Cy_DMA_Descriptor_SetDstAddress(&DMA_1_Descriptor_1, &dst);
Cy_DMA_Channel_Enable(DMA_1_HW, DMA_1_DW_CHANNEL);
Cy_DMA_Enable(DMA_1_HW);
```

7. Build the project and program the device.

Input/Output Connections

This section describes the various input and output connections for the DMA Component. An asterisk (*) in the following list indicates that it may not be shown on the Component symbol for the conditions listed in the description of that I/O.

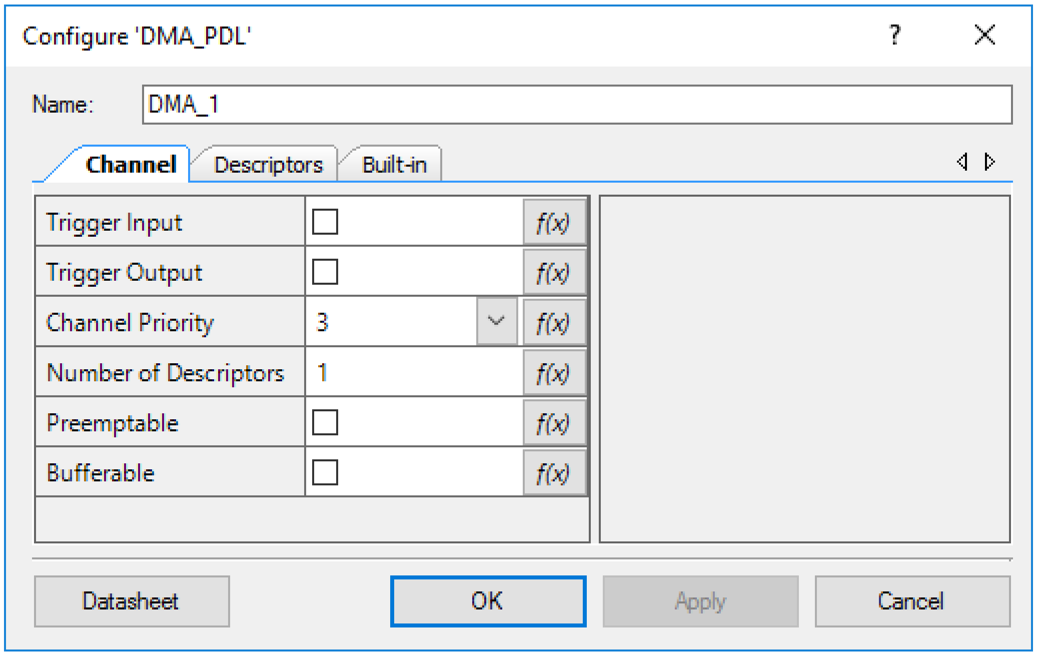
Terminal Name	I/O Type	Description
tr_in*	Digital Input	Visible only if 'Trigger input' is set to True. Input trigger parameter sets up the trigger input signal for the DMA Component.
tr_out*	Digital Output	Visible only if 'Trigger output' is set to True. Output trigger parameter sets up the trigger output signal for the DMA Component.
interrupt	Digital Output	Output terminal that allows to connect the interrupt Component.

Component Parameters

Drag a DMA Component onto your design and double click it to open the Configure dialog. This dialog has the following tabs with different parameters.

Channel Tab

This tab contains DMA channel settings and number of descriptors.



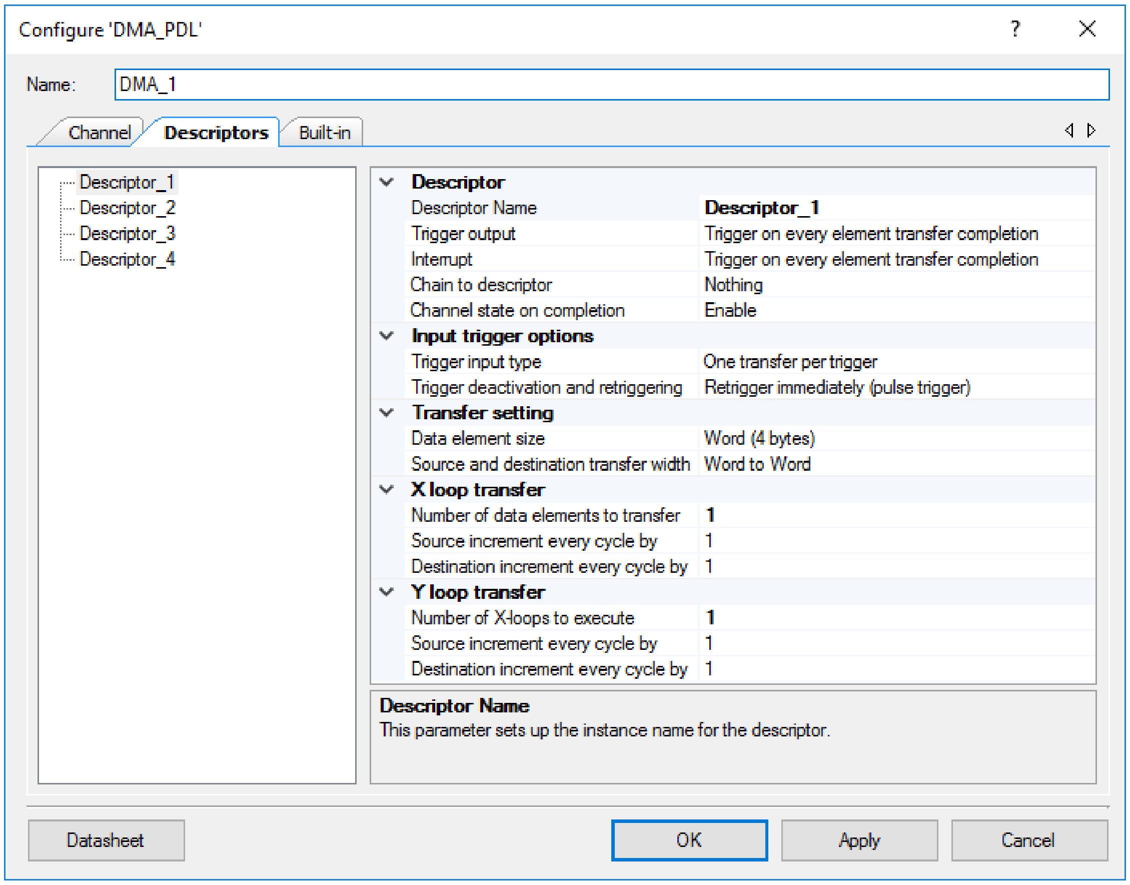
Name: DMA_1

Trigger Input	☐
Trigger Output	☐
Channel Priority	3
Number of Descriptors	1
Preemptable	☐
Bufferable	☐

Parameter Name	Description
Trigger input	Input trigger parameter sets up the trigger input signal for the DMA Component.
Trigger Output	Output trigger parameter sets up the trigger output signal for the DMA Component.
Channel Priority	Priority of the channel in the DMA block.
Number of Descriptors	The number of descriptors sets up rest of the customizer. Based on the number set in this parameter as many instances of Descriptors are created in the customizer.
Preemptable	If preemptable is enabled, and there is a higher priority pending channel, then that higher priority channel can preempt the current channel between single transfers.
Bufferable	This parameter specifies whether a write transaction can complete without waiting for the destination to accept the write transaction data.

Descriptors Tab

This tab contains the Descriptors configuration settings. Same set of parameters are created for each of descriptors.



Descriptor_1
Descriptor_2
Descriptor_3
Descriptor_4

Descriptor	
Descriptor Name	**Descriptor_1**
Trigger output	Trigger on every element transfer completion
Interrupt	Trigger on every element transfer completion
Chain to descriptor	Nothing
Channel state on completion	Enable
Input trigger options	
Trigger input type	One transfer per trigger
Trigger deactivation and retriggering	Retrigger immediately (pulse trigger)
Transfer setting	
Data element size	Word (4 bytes)
Source and destination transfer width	Word to Word
X loop transfer	
Number of data elements to transfer	**1**
Source increment every cycle by	1
Destination increment every cycle by	1
Y loop transfer	
Number of X-loops to execute	**1**
Source increment every cycle by	1
Destination increment every cycle by	1

Descriptor Name
This parameter sets up the instance name for the descriptor.

The **Descriptors** tab includes a copy/paste descriptor settings feature. Right-click and select **Copy** (in the context menu) on any descriptor (in the descriptor list on the left). This copies its settings into an internal buffer. Then, right-click and select **Paste** on any descriptor to replace its settings (all except 'Descriptor Name' and 'Chain to descriptor') with previously copied values from the internal buffer. Ctrl+C and Ctrl+V shortcuts work as well.

Parameter Name	Description
Descriptor	
Descriptor Name	This parameter sets up the instance name for the Descriptor. The default is 'Descriptor_N', where N is a sequence number of descriptor.

Parameter Name	Description
Trigger output	This is the selection for what event would trigger the DMA output: - Trigger on every element transfer completion (default); - Trigger on every X loop transfer completion; - Trigger on descriptor completion; - Trigger on completion of entire descriptor chain.
Interrupt	This is the selection for what event would trigger the DMA interrupt: - Trigger on every element transfer completion (default) - Trigger on every X loop transfer completion - Trigger on descriptor completion - Trigger on completion of entire descriptor chain
Channel state on completion	This is the state (enable/disable) of the channel when the descriptor is completed. In case of “disable” the channel needs to be reenabled (by Cy_DMA_Channel_Enable) after the current descriptor completion for further functioning. The default is ‘enable’.
Chain to descriptor	This parameter allows to configure what will be executed after current descriptor. The dropdown shows a list of all the channel’s descriptors. The default is ‘Nothing’ which is an equivalent of ‘NULL’ pointer in C code.
Input trigger options	
Trigger input type	Trigger input type will set up the character of each trigger: - One transfer per trigger (default) - One X loop transfer per trigger - An entire descriptor transfer per trigger - Entire descriptor chain per descriptor
Trigger deactivation and retriggering	This parameter sets up the trigger deactivation options for the descriptor: - Retrigger immediately (pulse trigger) (default) - Retrigger after 4 Clk_Slow cycles - Retrigger after 16 Clk_Slow cycles - Wait for trigger reactivation For the ‘Retrigger’ options, the start time is at the completion of current transaction, e.g., the trigger input is insensitive to the trigger signal level during 4/16 Clk_Slow cycles after the completion of current transaction – it is for slow trigger sources to let the signal enough time to go low to don’t cause undesirable triggering.

Parameter Name	Description
Transfer settings	
Data element size	This parameter sets up the data element size parameter in the descriptor: - Byte - Halfword (2 bytes) - Word (4 bytes) (default)
Source and destination transfer width	This sets up the source and destination transfer size parameters: - Byte to Byte - Halfword to Halfword - Word to Byt - Word to Halfword - Byte to Word - Halfword to Word - Word to Word (default)
X loop transfer	
Number of data elements to transfer	This parameter configures how many transfers are effected in the X loops. If this number equals 1 – the descriptor is determined as “single transfer”, otherwise as “1D” or “2D” – dependent on “Number of X-loops to execute” parameter. Range is 1…256. Default is 1.
Source increment every cycle by	This integer determines the source address increment after each data transfer. When this value is set to zero, it is equivalent to disabling the source increment feature. In terms of the absolute address, the unit of address increment is defined by the “Source and destination transfer width” setting. Range is -2048...2047. Default is 1.
Destination increment every cycle by	This integer determines the destination address increment after each transfer. When this value is set to zero, it is equivalent to disabling the destination increment feature. In terms of the absolute address, the unit of address increment is defined by the “Source and destination transfer width” setting. Range is -2048...2047. Default is 1.
Y loop transfer	
Number of X-loops to execute	This parameter configures how many X-loops are executed in each Y loop. If this number equals 1 – the descriptor is determined as “1D”, otherwise - “2D”. Range is 1…256. Default is 1.
Source increment every cycle by	This integer determines the source address increment after each X loop execution. When this value is set to zero, it is equivalent to disabling the source increment feature in Y loop. In terms of the absolute address, the unit of address increment is defined by the “Source and destination transfer width” setting. Range is -2048...2047. Default is 1.
Destination increment every cycle by	This integer determines the destination address increment after each transfer. When this value is set to zero, it is equivalent to disabling the destination increment feature in Y loop. In terms of the absolute address, the unit of address increment is defined by the “Source and destination transfer width” setting. Range is -2048...2047. Default is 1.

Note The X loop increment is effective only inside the X loop; that is, each new X loop (in case of 2D descriptor) starts with a base srs/dst address + Y loop increment (accumulative during Y loop execution). The following code shows the X and Y loops and their addresses incrementing:

```
cy_stc_dma_descriptor_config_t descriptor; /* example of 2D descriptor instance */
uint32_t * xAddress; /* internal storage of the accumulative X incremented address */
uint32_t * yAddress; /* internal storage of the accumulative Y incremented address */
uint32_t yIdx, xIdx; /* internal X and Y loop counters */

yAddress = descriptor.srcAddress;

for(yIdx = 0; yIdx < descriptor.yCount; yIdx++)
{
    xAddress = yAddress;

    for(xIdx = 0; xIdx < descriptor.xCount; xIdx++)
    {
        (void) *xAddress; /* data access/transfer */

        xAddress += descriptor.srcXincrement * descriptor.srcTransferSize;
    }

    yAddress += descriptor.srcYincrement * descriptor.srcTransferSize;
}
```

where the srcTransferSize is a number of bytes in single data transfer in this example.

PSoC Creator DMA Editor

There is a dedicated DMA Editor in the PSoC Creator Design-Wide Resources (DWR) file (visible in the Workspace Explorer). This editor shows the DMA HW block and channel numbers for each DMA Component, which are set by PSoC Creator during project generation. For more informationation, refer to the PSoC Creator Help “DMA Editor” topic.

DMA Placement

You can force the DMA Component placement into a certain DMA block/channel using a control file. Refer to the PSoC Creator Help “Control File” topic for an explanation of how to create and use a control file. Refer also to the “Drectives” topic for the placement_force directive description for the DMA Component.

The example control file line for DMA is:

```
attribute placement_force of \DMA_1:DW\ : label is "DMA(0,21)";
```

It places the DMA_1 Component into the DMA block 1, channel 5.

Application Programming Interface

The Application Programming Interface (API) is provided by the **DMA** and the **Trigger Multiplexer** driver modules from the PDL. The DMA driver is copied into the "pdl\drivers\peripheral\dma\" directory of the application project after a successful build. The Trigger Multiplexer is a design-wide driver and permanently exists in the "pdl\drivers\peripheral\trigmux\" directory (independently on the DMA Component existence in the project).

Refer to the PDL documentation for a detailed description of the complete API. To access this document, right-click on the Component symbol on the schematic and choose the "**Open PDL Documentation…**" option in the drop-down menu.

The Component generates the configuration structures and base address described in the Global Variables and Preprocessor Macros sections. Pass the generated data structure and/or the base address to the associated DMA driver function in the application initialization code to configure the DMA channel. Once the channel is initialized, the application code can perform run-time changes by referencing the provided base/descriptor address in the driver API functions.

By default, PSoC Creator assigns the instance name **DMA_1** to the first instance of a Component in a given design. You can rename it 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 section is **DMA_1** and the first descriptor name is **Descriptor_1**.

Global Variables

The DMA Component populates the following peripheral initialization data structures. The generated code is placed in C source and header files that are named after the instance of the Component (e.g. DMA_1.c). Each variable is also prefixed with the instance name of the Component.

cy_stc_dma_descriptor_config_t DMA_1_Descriptor_1_config

A configuration structure for the Descriptor_1. This should be used in the associated Cy_DMA_Descriptor_Init() function. The similar structure is generated for each descriptor.

cy_stc_dma_descriptor_t DMA_1_Descriptor_1

A descriptor itself declaration. The similar is generated for each descriptor.

Preprocessor Macros

The DMA Component generates the following preprocessor macros. Note that each macro is prefixed with the instance name of the Component (e.g. “DMA_1”).

#define DMA_1_DW_BLOCK

The number of the DW/DMA block that owns the channel.

#define DMA_1_DW_CHANNEL

The number of the DW/DMA channel.

#define DMA_1_HW

The pointer to the base address of the HW DW/DMA block instance registers.

#define DMA_1_INTR_MASK

The interrupt mask to be used with DMA_1_SetInterruptMask() API function.

#define DMA_1_PRIORITY

The channel priority setting from the GUI.

#define DMA_1_DESCRIPTOR_NUM

The channel number of descriptors setting from the GUI.

#define DMA_1_PREEMPTABLE

The channel Preemptable setting from the GUI.

#define DMA_1_BUFFERABLE

The channel Bufferable setting from the GUI.

Component Functions

The DMA_PDL Component provides the following functions. Note that each function is prefixed with the instance name of the Component (e.g., “DMA_1”).

void DMA_1_Start (void const * srcAddress, void const * dstAddress)

Description:	At the first call (after the system reset) this function runs the DMA_1_Init(). At the each call it sets the source/destination addresses into the first descriptor (associated with this DMA channel) and then enables the channel using Cy_DMA_Channel_Enable().

void DMA_1_Init (void)

Description: Based on the settings for the descriptor in the customizer, this function runs Cy_DMA_Descriptor_Init () for each descriptor, then initializes the channel and enables the DMA block using Cy_DMA_Channel_Init() and Cy_DMA_Enable() correspondingly.

void DMA_1_Trigger (uint32_t cycles)

Description: Invokes the Cy_TrigMux_SwTrigger() PDL driver function.

Triggers correspondent Trigger Multiplexer output, independently on whether the Component’s tr_in terminal is connected to any trigger signal or not.

The Component also includes a set of Component-specific wrapper functions that provide simplified access to the basic DMA operation. These functions are generated during the build process and are all prefixed with the name of the Component instance.

Interrupt Service Routine

Each instance of a Component can have an interrupt. To enable and configure it:

1. Drag the Interrupt Component to TopDesign and connect it to the DMA Component terminal called Interrupt.
2. Use Cy_DMA_Channel_SetInterruptMask() function call to enable interrupt from the Component.
3. Optionally, you can use the Cy_DMA_Channel_GetStatus () function in the interrupt handler to get the interrupt cause.

API Memory Usage

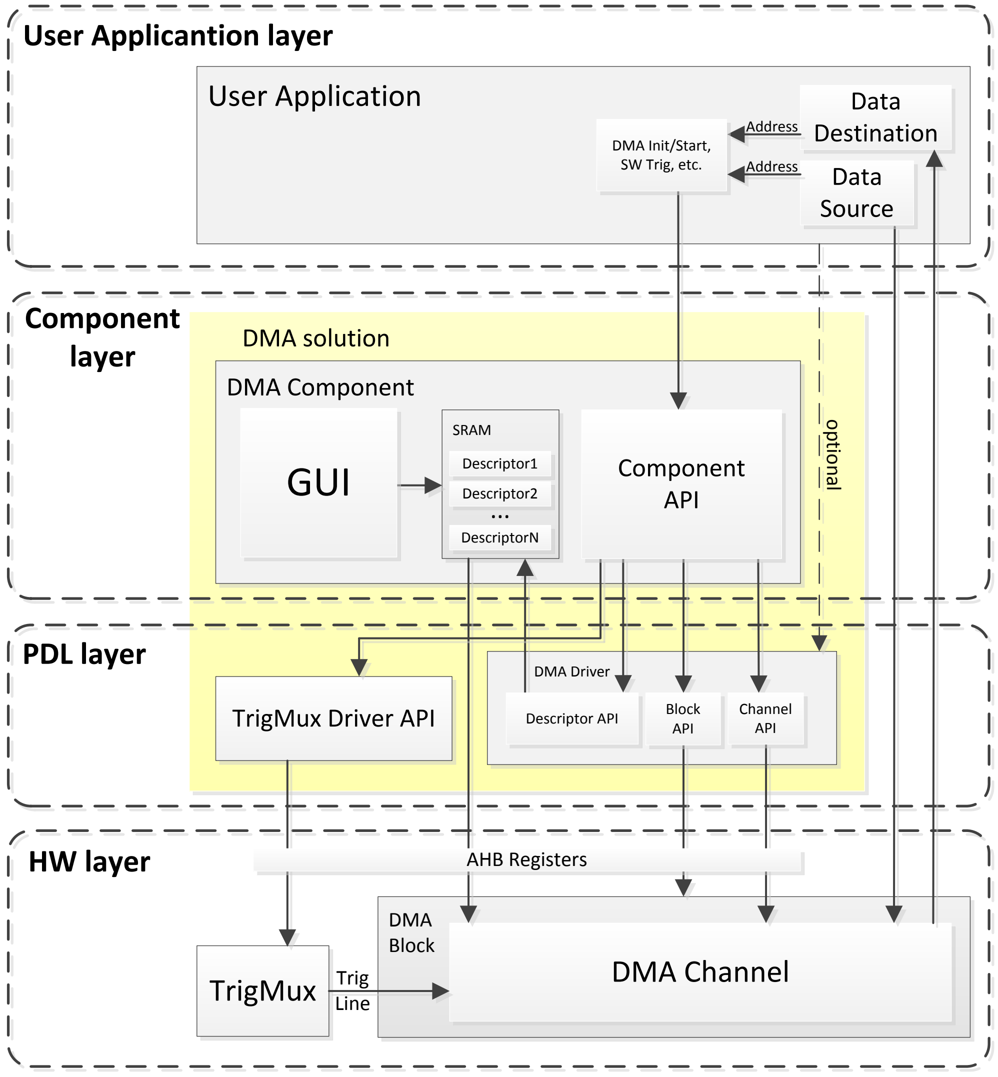
The Component is designed to use API from the dma Peripheral Driver Library (PDL) module. That is why the Component itself only consumes resources necessary to allocate structures for driver operation and start the Component.

Functional Description

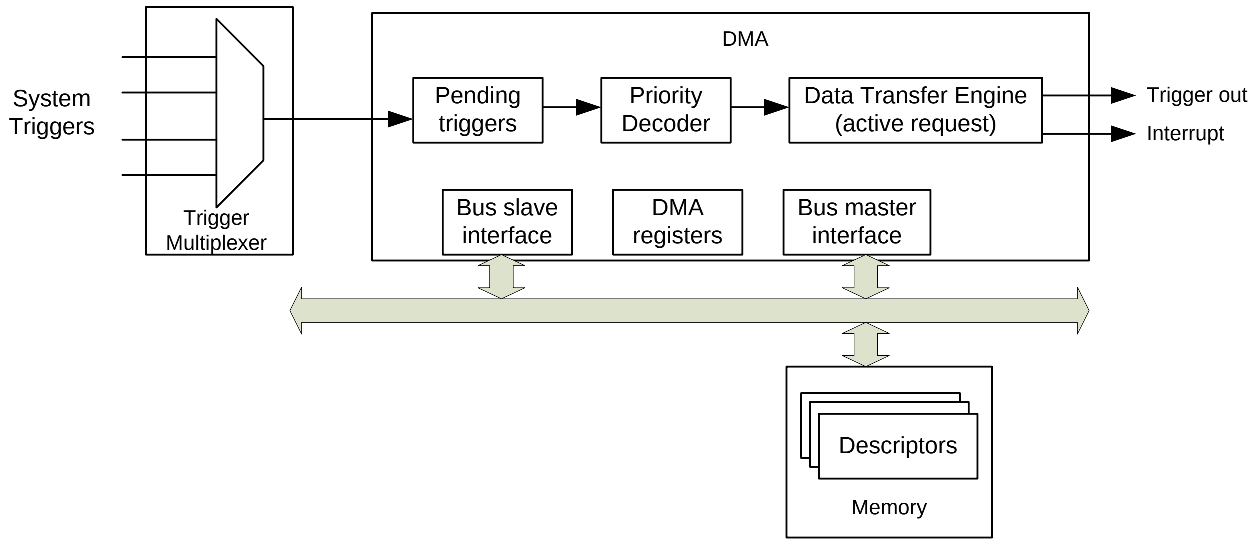
The DMA Component represents a DMA channel that can have up to 4096 descriptors that are pre-configured in the customizer. Each descriptor can be chained to another descriptor. That means that on descriptor completion DMA channel will automatically be switched to the next descriptor. Depending on Trigger input type configured in customizer DMA Component can wait for the next trigger or execute the next descriptor.

Block Diagram and Configuration

The following is a structure of the Component-driver DMA solution:



The following is a simplified diagram of the DMA hardware:



DMA controller supports multiple independent data transfers that are managed by a channel. Each channel connects to a specific system trigger through a trigger multiplexer that is outside the DMA controller.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- specific deviations – deviations that are applicable only for this Component

Refer to **PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6)** for information on MISRA compliance and deviations of the files generated by PSoC Creator.

The DMA Component doesn't have any specific deviations.

Registers

See the device *Technical Reference Manual (TRM)* for more information about the registers.

Resources

The Component uses one DMA channel.

DC and AC Electrical Characteristics

Refer to Digital Peripherals in the Electrical Specifications section of the Device Family Datasheet.

Component Changes

This section lists the major changes in the Component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
2.0.c	Minor datasheet edits.	
2.0.b	Edited the datasheet.	Fixed a typo.
2.0.a	Edited the datasheet.	Clarify technical details.
2.0	Key changes include: • The GUI can generate “single transfer”, “1D” and “2D” descriptor types. • The Start API function gets source and destination addresses as parameters, and enables DMA block. • There are Set/Get API functions for each descriptor setting. Most of the other API changes make the interface more consistent within itself and with the rest of PSoC Creator/PDL content.	PSoC 6 public release.
1.0.a	Updated datasheet	Changed several API descriptions.
1.0	First Component version	