

DMA Driver  
STM32F407DMA

Designed by Kunal Salvi



<b>1 DMA Driver Documentation</b>	<b>1</b>
1.1 Introduction	1
1.2 Features	1
1.3 Configuration	1
1.4 Functions	2
1.5 Usage	2
1.6 Example	3
1.7 Notes	3
1.8 License	3
<b>2 Topic Index</b>	<b>5</b>
2.1 Topics	5
<b>3 Data Structure Index</b>	<b>7</b>
3.1 Data Structures	7
<b>4 File Index</b>	<b>9</b>
4.1 File List	9
<b>5 Topic Documentation</b>	<b>11</b>
5.1 DMA_Flags	11
5.1.1 Detailed Description	11
<b>6 Data Structure Documentation</b>	<b>13</b>
6.1 DMA_Configuration::Circular_Mode Struct Reference	13
6.1.1 Detailed Description	13
6.1.2 Field Documentation	13
6.1.2.1 Disable	13
6.1.2.2 Enable	13
6.2 DMA_Config Struct Reference	14
6.2.1 Detailed Description	14
6.2.2 Field Documentation	14
6.2.2.1 buffer_length	14
6.2.2.2 circular_mode	14
6.2.2.3 flow_control	14
6.2.2.4 interrupts	15
6.2.2.5 memory_address	15
6.2.2.6 memory_data_size	15
6.2.2.7 memory_pointer_increment	15
6.2.2.8 peripheral_address	15
6.2.2.9 peripheral_data_size	15
6.2.2.10 peripheral_pointer_increment	15
6.2.2.11 priority_level	15
6.2.2.12 Request	16

6.2.2.13 transfer_direction	16
6.3 DMA_Flags_Typedef Struct Reference	16
6.3.1 Detailed Description	16
6.3.2 Field Documentation	16
6.3.2.1 Direct_Mode_Error_Flag	16
6.3.2.2 Fifo_Error_Flag	17
6.3.2.3 Half_Transfer_Complete_Flag	17
6.3.2.4 Transfer_Complete_Flag	17
6.3.2.5 Transfer_Error_Flag	17
6.4 DMA_Configuration::DMA_Interrupts Struct Reference	17
6.4.1 Detailed Description	18
6.4.2 Field Documentation	18
6.4.2.1 Direct_Mode_Error	18
6.4.2.2 Disable	18
6.4.2.3 Fifo_Error	18
6.4.2.4 Half_Transfer_Complete	18
6.4.2.5 Transfer_Complete	18
6.4.2.6 Transfer_Error	18
6.5 DMA_Request Struct Reference	19
6.5.1 Detailed Description	19
6.5.2 Field Documentation	19
6.5.2.1 channel	19
6.5.2.2 Controller	19
6.5.2.3 Stream	19
6.6 DMA_Configuration::Flow_Control Struct Reference	20
6.6.1 Detailed Description	20
6.6.2 Field Documentation	20
6.6.2.1 DMA_Control	20
6.6.2.2 Peripheral_Control	20
6.7 DMA_Configuration::Memory_Data_Size Struct Reference	20
6.7.1 Detailed Description	21
6.7.2 Field Documentation	21
6.7.2.1 byte	21
6.7.2.2 half_word	21
6.7.2.3 word	21
6.8 DMA_Configuration::Memory_Pointer_Increment Struct Reference	21
6.8.1 Detailed Description	21
6.8.2 Field Documentation	22
6.8.2.1 Disable	22
6.8.2.2 Enable	22
6.9 DMA_Configuration::Peripheral_Data_Size Struct Reference	22
6.9.1 Detailed Description	22

6.9.2 Field Documentation	22
6.9.2.1 byte	22
6.9.2.2 half_word	23
6.9.2.3 word	23
6.10 DMA_Configuration::Peripheral_Pointer_Increment Struct Reference	23
6.10.1 Detailed Description	23
6.10.2 Field Documentation	23
6.10.2.1 Disable	23
6.10.2.2 Enable	24
6.11 DMA_Configuration::Priority_Level Struct Reference	24
6.11.1 Detailed Description	24
6.11.2 Field Documentation	24
6.11.2.1 High	24
6.11.2.2 Low	24
6.11.2.3 Medium	25
6.11.2.4 Very_high	25
6.12 DMA_Configuration::Request Struct Reference	25
6.12.1 Detailed Description	27
6.12.2 Field Documentation	27
6.12.2.1 _ADC1	27
6.12.2.2 _ADC2	27
6.12.2.3 _ADC3	27
6.12.2.4 _DAC1	27
6.12.2.5 _DAC2	27
6.12.2.6 _DCMI	27
6.12.2.7 I2C1_RX	28
6.12.2.8 I2C1_TX	28
6.12.2.9 I2C2_RX	28
6.12.2.10 I2C2_TX	28
6.12.2.11 I2C3_RX	28
6.12.2.12 I2C3_TX	28
6.12.2.13 I2S2_RX	28
6.12.2.14 I2S2_TX	28
6.12.2.15 I2S3_RX	29
6.12.2.16 I2S3_TX	29
6.12.2.17 SDIO_RXTX	29
6.12.2.18 SPI1_RX	29
6.12.2.19 SPI1_TX	29
6.12.2.20 SPI2_RX	29
6.12.2.21 SPI2_TX	29
6.12.2.22 SPI3_RX	29
6.12.2.23 SPI3_TX	30

6.12.2.24 TIM1_CH1 . . . . .	30
6.12.2.25 TIM1_CH2 . . . . .	30
6.12.2.26 TIM1_CH3 . . . . .	30
6.12.2.27 TIM1_CH4 . . . . .	30
6.12.2.28 TIM1_COM . . . . .	30
6.12.2.29 TIM1_TRIG . . . . .	30
6.12.2.30 TIM1_UP . . . . .	30
6.12.2.31 TIM2_CH1 . . . . .	31
6.12.2.32 TIM2_CH2 . . . . .	31
6.12.2.33 TIM2_CH3 . . . . .	31
6.12.2.34 TIM2_CH4 . . . . .	31
6.12.2.35 TIM2_UP . . . . .	31
6.12.2.36 TIM3_CH1 . . . . .	31
6.12.2.37 TIM3_CH2 . . . . .	31
6.12.2.38 TIM3_CH3 . . . . .	31
6.12.2.39 TIM3_CH4 . . . . .	32
6.12.2.40 TIM3_TRIG . . . . .	32
6.12.2.41 TIM3_UP . . . . .	32
6.12.2.42 TIM4_CH1 . . . . .	32
6.12.2.43 TIM4_CH2 . . . . .	32
6.12.2.44 TIM4_CH3 . . . . .	32
6.12.2.45 TIM4_UP . . . . .	32
6.12.2.46 TIM5_CH1 . . . . .	32
6.12.2.47 TIM5_CH2 . . . . .	33
6.12.2.48 TIM5_CH3 . . . . .	33
6.12.2.49 TIM5_CH4 . . . . .	33
6.12.2.50 TIM5_TRIG . . . . .	33
6.12.2.51 TIM5_UP . . . . .	33
6.12.2.52 TIM6_UP . . . . .	33
6.12.2.53 TIM7_UP . . . . .	33
6.12.2.54 TIM8_CH1 . . . . .	33
6.12.2.55 TIM8_CH2 . . . . .	34
6.12.2.56 TIM8_CH3 . . . . .	34
6.12.2.57 TIM8_CH4 . . . . .	34
6.12.2.58 TIM8_COM . . . . .	34
6.12.2.59 TIM8_TRIG . . . . .	34
6.12.2.60 TIM8_UP . . . . .	34
6.12.2.61 UART4_RX . . . . .	34
6.12.2.62 UART4_TX . . . . .	34
6.12.2.63 UART5_RX . . . . .	35
6.12.2.64 UART5_TX . . . . .	35
6.12.2.65 UART6_RX . . . . .	35

6.12.2.66 UART6_TX . . . . .	35
6.12.2.67 UART7_RX . . . . .	35
6.12.2.68 UART7_TX . . . . .	35
6.12.2.69 UART8_RX . . . . .	35
6.12.2.70 UART8_TX . . . . .	35
6.12.2.71 USART1_RX . . . . .	36
6.12.2.72 USART1_TX . . . . .	36
6.12.2.73 USART2_RX . . . . .	36
6.12.2.74 USART2_TX . . . . .	36
6.12.2.75 USART3_RX . . . . .	36
6.12.2.76 USART3_TX . . . . .	36
6.13 DMA_Configuration::Transfer_Direction Struct Reference . . . . .	36
6.13.1 Detailed Description . . . . .	37
6.13.2 Field Documentation . . . . .	37
6.13.2.1 Memory_to_memory . . . . .	37
6.13.2.2 Memory_to_peripheral . . . . .	37
6.13.2.3 Peripheral_to_memory . . . . .	37
<b>7 File Documentation</b>	<b>39</b>
7.1 DMA.c File Reference . . . . .	39
7.1.1 Detailed Description . . . . .	41
7.1.2 Function Documentation . . . . .	41
7.1.2.1 DMA1_Stream0_IRQHandler() . . . . .	41
7.1.2.2 DMA1_Stream1_IRQHandler() . . . . .	41
7.1.2.3 DMA1_Stream2_IRQHandler() . . . . .	42
7.1.2.4 DMA1_Stream3_IRQHandler() . . . . .	42
7.1.2.5 DMA1_Stream4_IRQHandler() . . . . .	42
7.1.2.6 DMA1_Stream5_IRQHandler() . . . . .	42
7.1.2.7 DMA1_Stream6_IRQHandler() . . . . .	43
7.1.2.8 DMA1_Stream7_IRQHandler() . . . . .	43
7.1.2.9 DMA2_Stream0_IRQHandler() . . . . .	43
7.1.2.10 DMA2_Stream1_IRQHandler() . . . . .	43
7.1.2.11 DMA2_Stream2_IRQHandler() . . . . .	44
7.1.2.12 DMA2_Stream3_IRQHandler() . . . . .	44
7.1.2.13 DMA2_Stream4_IRQHandler() . . . . .	44
7.1.2.14 DMA2_Stream5_IRQHandler() . . . . .	44
7.1.2.15 DMA2_Stream6_IRQHandler() . . . . .	45
7.1.2.16 DMA2_Stream7_IRQHandler() . . . . .	45
7.1.2.17 DMA_Clock_Disable() . . . . .	45
7.1.2.18 DMA_Clock_Enable() . . . . .	45
7.1.2.19 DMA_Init() . . . . .	46
7.1.2.20 DMA_Memory_To_Memory_Transfer() . . . . .	46

---

7.1.2.21 DMA_Reset()	47
7.1.2.22 DMA_Reset_Flags()	47
7.1.2.23 DMA_Set_Target()	47
7.1.2.24 DMA_Set_Trigger()	47
7.2 DMA.h File Reference	48
7.2.1 Detailed Description	49
7.2.2 Typedef Documentation	49
7.2.2.1 DMA_Config	49
7.2.3 Function Documentation	50
7.2.3.1 DMA_Clock_Disable()	50
7.2.3.2 DMA_Clock_Enable()	51
7.2.3.3 DMA_Init()	51
7.2.3.4 DMA_Memory_To_Memory_Transfer()	52
7.2.3.5 DMA_Reset()	52
7.2.3.6 DMA_Reset_Flags()	53
7.2.3.7 DMA_Set_Target()	53
7.2.3.8 DMA_Set_Trigger()	54
7.3 DMA.h	54
7.4 DMA_Defs.h File Reference	55
7.4.1 Detailed Description	56
7.4.2 Typedef Documentation	56
7.4.2.1 DMA_Flags_Typedef	56
7.4.2.2 DMA_Request	56
7.5 DMA_Defs.h	57
<b>Index</b>	<b>67</b>



# Chapter 1

## DMA Driver Documentation

### 1.1 Introduction

This documentation provides an overview and detailed explanation of the Direct Memory Access (DMA) driver. The DMA driver allows for efficient, high-speed data transfers between peripherals and memory without CPU intervention, making it an essential tool for performance-critical applications.

### 1.2 Features

- **Memory-to-Memory Transfer:** Supports direct data transfers between memory regions.
- **Peripheral-to-Memory / Memory-to-Peripheral Transfers:** Facilitates data exchange between peripherals and memory.
- **Flow Control:** Configurable flow control to use DMA or peripheral as the flow controller.
- **Circular Mode:** Enables continuous data transfers in a circular buffer.
- **Interrupt Handling:** Supports transfer complete, half transfer complete, transfer error, and FIFO error interrupts.
- **Priority Levels:** Configurable priority levels for managing multiple DMA streams.
- **Configurable Data Sizes:** Supports byte, half-word, and word data sizes for both memory and peripherals.

### 1.3 Configuration

The DMA driver is configured using the [DMA\\_Config](#) structure, which contains the following parameters:

- [DMA\\_Request](#) Request: Defines the specific DMA stream and channel to use.
- `uint32_t flow_control`: Specifies whether DMA or peripheral controls the flow.
- `uint32_t transfer_direction`: Defines the direction of data transfer (e.g., memory-to-peripheral).
- `uint32_t priority_level`: Sets the priority of the DMA stream.
- `uint32_t circular_mode`: Enables or disables circular mode for the DMA stream.

- `uint32_t interrupts`: Specifies which DMA interrupts to enable.
- `uint16_t memory_pointer_increment`: Configures memory pointer increment mode.
- `uint16_t peripheral_pointer_increment`: Configures peripheral pointer increment mode.
- `uint32_t peripheral_data_size`: Sets the data size for the peripheral (byte, half-word, or word).
- `uint32_t memory_data_size`: Sets the data size for memory (byte, half-word, or word).
- `uint32_t peripheral_address`: The address of the peripheral.
- `uint32_t memory_address`: The address of the memory.
- `uint16_t buffer_length`: The number of data items to transfer.

## 1.4 Functions

The DMA driver provides the following functions:

- `void DMA_Clock_Enable(DMA_Config *config)`: Enables the clock for the specified DMA controller.
- `void DMA_Clock_Disable(DMA_Config *config)`: Disables the clock for the specified DMA controller.
- `void DMA_Reset(DMA_Config *config)`: Resets the specified DMA controller.
- `void DMA_Reset_Flags(DMA_Flags_Typedef flag)`: Resets all DMA flags.
- `int8_t DMA_Init(DMA_Config *config)`: Initializes the DMA with the specified configuration.
- `void DMA_Set_Target(DMA_Config *config)`: Configures the target memory and peripheral for DMA transfers.
- `void DMA_Set_Trigger(DMA_Config *config)`: Sets up and enables the DMA stream for data transfer.
- `void DMA_Memory_To_Memory_Transfer(uint32_t *source, uint8_t source_data_size, uint8_t destination_data_size)`: Performs a memory-to-memory data transfer using DMA.

## 1.5 Usage

To use the DMA driver in your application, follow these steps:

1. **Configure the DMA settings:** Initialize a `DMA_Config` structure with your desired settings.
2. **Initialize the DMA:** Call `DMA_Init` with the configuration structure.
3. **Set up data transfer:** Use `DMA_Set_Target` and `DMA_Set_Trigger` to configure the transfer settings.
4. **Start the transfer:** If using memory-to-memory transfer, call `DMA_Memory_To_Memory_Transfer`.

## 1.6 Example

```
DMA_Config dma_config;
dma_config.Request = DMA_Configuration.Request.SPI1_RX;
dma_config.flow_control = DMA_Configuration.Flow_Control.DMA_Control;
dma_config.transfer_direction = DMA_Configuration.Transfer_Direction.Peripheral_to_memory;
dma_config.priority_level = DMA_Configuration.Priority_Level.High;
dma_config.circular_mode = DMA_Configuration.Circular_Mode.Disable;
dma_config.interrupts = DMA_Configuration.DMA_Interrupts.Transfer_Complete;
dma_config.memory_pointer_increment = DMA_Configuration.Memory_Pointer_Increment.Enable;
dma_config.peripheral_pointer_increment = DMA_Configuration.Peripheral_Pointer_Increment.Disable;
dma_config.peripheral_data_size = DMA_Configuration.Peripheral_Data_Size.half_word;
dma_config.memory_data_size = DMA_Configuration.Memory_Data_Size.half_word;
dma_config.peripheral_address = (uint32_t)&(SPI1->DR);
dma_config.memory_address = (uint32_t)buffer;
dma_config.buffer_length = BUFFER_SIZE;

DMA_Init(&dma_config);
DMA_Set_Target(&dma_config);
DMA_Set_Trigger(&dma_config);
```

## 1.7 Notes

- Ensure that the appropriate DMA streams and channels are enabled before starting a transfer.
- Pay attention to memory alignment when configuring data sizes.

## 1.8 License

This software is provided "as-is," without any express or implied warranty. In no event shall the authors be held liable for any damages arising from the use of this software.



## Chapter 2

# Topic Index

### 2.1 Topics

Here is a list of all topics with brief descriptions:

DMA_Flags . . . . .	<a href="#">11</a>
---------------------	--------------------



## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">DMA_Configuration::Circular_Mode</a>	
Circular Mode Configuration . . . . .	13
<a href="#">DMA_Config</a>	
DMA configuration structure . . . . .	14
<a href="#">DMA_Flags_Typedef</a>	
DMA Flags Structure . . . . .	16
<a href="#">DMA_Configuration::DMA_Interrupts</a>	
DMA Interrupt Configuration . . . . .	17
<a href="#">DMA_Request</a>	
DMA Request Structure . . . . .	19
<a href="#">DMA_Configuration::Flow_Control</a>	
Flow Control Configuration . . . . .	20
<a href="#">DMA_Configuration::Memory_Data_Size</a>	
Memory Data Size Configuration . . . . .	20
<a href="#">DMA_Configuration::Memory_Pointer_Increment</a>	
Memory Pointer Increment Configuration . . . . .	21
<a href="#">DMA_Configuration::Peripheral_Data_Size</a>	
Peripheral Data Size Configuration . . . . .	22
<a href="#">DMA_Configuration::Peripheral_Pointer_Increment</a>	
Peripheral Pointer Increment Configuration . . . . .	23
<a href="#">DMA_Configuration::Priority_Level</a>	
Priority Level Configuration . . . . .	24
<a href="#">DMA_Configuration::Request</a>	
DMA Request Channels . . . . .	25
<a href="#">DMA_Configuration::Transfer_Direction</a>	
Transfer Direction Configuration . . . . .	36





# Chapter 4

## File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">DMA.c</a>	DMA Driver Implementation for STM32F407VGT6 . . . . .	<a href="#">39</a>
<a href="#">DMA.h</a>	Header file for the DMA driver . . . . .	<a href="#">48</a>
<a href="#">DMA_Defs.h</a>	DMA Configuration Definitions for STM32F407VGT6 . . . . .	<a href="#">55</a>



# Chapter 5

## Topic Documentation

### 5.1 DMA\_Flags

#### Variables

- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream0\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream1\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream2\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream3\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream4\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream5\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream6\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream7\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream0\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream1\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream2\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream3\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream4\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream5\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream6\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream7\_Flag**

#### 5.1.1 Detailed Description



## Chapter 6

# Data Structure Documentation

### 6.1 DMA\_Configuration::Circular\_Mode Struct Reference

Circular Mode Configuration.

```
#include <DMA_Defs.h>
```

#### Data Fields

- uint32\_t [Enable](#)
- uint32\_t [Disable](#)

#### 6.1.1 Detailed Description

Circular Mode Configuration.

This structure defines the settings for circular mode in DMA transfers, allowing the DMA to repeatedly transfer data from a peripheral to memory.

#### 6.1.2 Field Documentation

##### 6.1.2.1 Disable

```
uint32_t Disable
```

Disable circular mode

##### 6.1.2.2 Enable

```
uint32_t Enable
```

Enable circular mode

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.2 DMA\_Config Struct Reference

DMA configuration structure.

```
#include <DMA.h>
```

### Data Fields

- [DMA\\_Request](#) Request
- uint32\_t [flow\\_control](#)
- uint32\_t [transfer\\_direction](#)
- uint32\_t [priority\\_level](#)
- uint32\_t [circular\\_mode](#)
- uint32\_t [interrupts](#)
- uint16\_t [memory\\_pointer\\_increment](#)
- uint16\_t [peripheral\\_pointer\\_increment](#)
- uint32\_t [peripheral\\_data\\_size](#)
- uint32\_t [memory\\_data\\_size](#)
- uint32\_t [peripheral\\_address](#)
- uint32\_t [memory\\_address](#)
- uint16\_t [buffer\\_length](#)

### 6.2.1 Detailed Description

DMA configuration structure.

This structure contains the configuration parameters for the DMA peripheral, including settings for flow control, transfer direction, priority level, circular mode, interrupts, memory and peripheral data sizes, and addresses.

### 6.2.2 Field Documentation

#### 6.2.2.1 `buffer_length`

```
uint16_t buffer_length
```

Number of data items to transfer

#### 6.2.2.2 `circular_mode`

```
uint32_t circular_mode
```

Circular mode enable/disable

#### 6.2.2.3 `flow_control`

```
uint32_t flow_control
```

Flow control mode (DMA or peripheral)

#### 6.2.2.4 interrupts

`uint32_t interrupts`

Interrupts settings

#### 6.2.2.5 memory\_address

`uint32_t memory_address`

Memory base address

#### 6.2.2.6 memory\_data\_size

`uint32_t memory_data_size`

Memory data size (byte, half-word, word)

#### 6.2.2.7 memory\_pointer\_increment

`uint16_t memory_pointer_increment`

Memory pointer increment mode

#### 6.2.2.8 peripheral\_address

`uint32_t peripheral_address`

Peripheral base address

#### 6.2.2.9 peripheral\_data\_size

`uint32_t peripheral_data_size`

Peripheral data size (byte, half-word, word)

#### 6.2.2.10 peripheral\_pointer\_increment

`uint16_t peripheral_pointer_increment`

Peripheral pointer increment mode

#### 6.2.2.11 priority\_level

`uint32_t priority_level`

Priority level (low, medium, high, very high)

### 6.2.2.12 Request

[DMA\\_Request](#) Request

DMA request settings

### 6.2.2.13 transfer\_direction

uint32\_t transfer\_direction

Transfer direction (memory-to-peripheral, peripheral-to-memory, etc.)

The documentation for this struct was generated from the following file:

- [DMA.h](#)

## 6.3 DMA\_Flags\_Typedef Struct Reference

DMA Flags Structure.

```
#include <DMA_Defs.h>
```

### Data Fields

- bool [Transfer\\_Complete\\_Flag](#)
- bool [Half\\_Transfer\\_Complete\\_Flag](#)
- bool [Transfer\\_Error\\_Flag](#)
- bool [Direct\\_Mode\\_Error\\_Flag](#)
- bool [Fifo\\_Error\\_Flag](#)

### 6.3.1 Detailed Description

DMA Flags Structure.

This structure contains flags that indicate the status of a DMA transfer. These flags are used to monitor and handle various states of the DMA transfer process.

### 6.3.2 Field Documentation

#### 6.3.2.1 Direct\_Mode\_Error\_Flag

bool Direct\_Mode\_Error\_Flag

Indicates if there was a direct mode error



### 6.3.2.2 Fifo\_Error\_Flag

```
bool Fifo_Error_Flag
```

Indicates if there was a FIFO error

### 6.3.2.3 Half\_Transfer\_Complete\_Flag

```
bool Half_Transfer_Complete_Flag
```

Indicates if half of the transfer is complete

### 6.3.2.4 Transfer\_Complete\_Flag

```
bool Transfer_Complete_Flag
```

Indicates if the transfer is complete

### 6.3.2.5 Transfer\_Error\_Flag

```
bool Transfer_Error_Flag
```

Indicates if there was a transfer error

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.4 DMA\_Configuration::DMA\_Interrupts Struct Reference

DMA Interrupt Configuration.

```
#include <DMA_Defs.h>
```

### Data Fields

- uint32\_t [Transfer\\_Complete](#)
- uint32\_t [Half\\_Transfer\\_Complete](#)
- uint32\_t [Transfer\\_Error](#)
- uint32\_t [Direct\\_Mode\\_Error](#)
- uint32\_t [Fifo\\_Error](#)
- uint32\_t [Disable](#)

### 6.4.1 Detailed Description

DMA Interrupt Configuration.

This structure defines the interrupt settings for DMA transfers, including transfer complete, half transfer complete, transfer error, direct mode error, FIFO error, and disable options.

### 6.4.2 Field Documentation

#### 6.4.2.1 Direct\_Mode\_Error

```
uint32_t Direct_Mode_Error
```

Interrupt on direct mode error

#### 6.4.2.2 Disable

```
uint32_t Disable
```

Disable all DMA interrupts

#### 6.4.2.3 Fifo\_Error

```
uint32_t Fifo_Error
```

Interrupt on FIFO error

#### 6.4.2.4 Half\_Transfer\_Complete

```
uint32_t Half_Transfer_Complete
```

Interrupt on half transfer complete

#### 6.4.2.5 Transfer\_Complete

```
uint32_t Transfer_Complete
```

Interrupt on transfer complete

#### 6.4.2.6 Transfer\_Error

```
uint32_t Transfer_Error
```

Interrupt on transfer error

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.5 DMA\_Request Struct Reference

DMA Request Structure.

```
#include <DMA_Defs.h>
```

### Data Fields

- DMA\_TypeDef \* [Controller](#)
- DMA\_Stream\_TypeDef \* [Stream](#)
- uint8\_t [channel](#)

### 6.5.1 Detailed Description

DMA Request Structure.

This structure contains the configuration for a specific DMA request, including the DMA controller, stream, and channel associated with the request.

### 6.5.2 Field Documentation

#### 6.5.2.1 channel

```
uint8_t channel
```

DMA channel number

#### 6.5.2.2 Controller

```
DMA_TypeDef* Controller
```

Pointer to the DMA controller

#### 6.5.2.3 Stream

```
DMA_Stream_TypeDef* Stream
```

Pointer to the DMA stream

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.6 DMA\_Configuration::Flow\_Control Struct Reference

Flow Control Configuration.

```
#include <DMA_Defs.h>
```

### Data Fields

- uint32\_t [DMA\\_Control](#)
- uint32\_t [Peripheral\\_Control](#)

### 6.6.1 Detailed Description

Flow Control Configuration.

This structure contains settings for DMA flow control, allowing either the DMA or the peripheral to control the data flow.

### 6.6.2 Field Documentation

#### 6.6.2.1 DMA\_Control

```
uint32_t DMA_Control
```

DMA controls the data flow

#### 6.6.2.2 Peripheral\_Control

```
uint32_t Peripheral_Control
```

Peripheral controls the data flow

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.7 DMA\_Configuration::Memory\_Data\_Size Struct Reference

Memory Data Size Configuration.

```
#include <DMA_Defs.h>
```

### Data Fields

- uint32\_t [byte](#)
- uint32\_t [half\\_word](#)
- uint32\_t [word](#)

### 6.7.1 Detailed Description

Memory Data Size Configuration.

This structure defines the data size for the memory in DMA transfers, including byte, half-word, and word sizes.

### 6.7.2 Field Documentation

#### 6.7.2.1 byte

```
uint32_t byte
```

Memory data size: byte

#### 6.7.2.2 half\_word

```
uint32_t half_word
```

Memory data size: half-word

#### 6.7.2.3 word

```
uint32_t word
```

Memory data size: word

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.8 DMA\_Configuration::Memory\_Pointer\_Increment Struct Reference

Memory Pointer Increment Configuration.

```
#include <DMA_Defs.h>
```

### Data Fields

- uint32\_t [Enable](#)
- uint32\_t [Disable](#)

### 6.8.1 Detailed Description

Memory Pointer Increment Configuration.

This structure defines the settings for memory pointer increment in DMA transfers. It allows the memory address to be incremented after each data transfer.

## 6.8.2 Field Documentation

### 6.8.2.1 Disable

`uint32_t Disable`

Disable memory pointer increment

### 6.8.2.2 Enable

`uint32_t Enable`

Enable memory pointer increment

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.9 DMA\_Configuration::Peripheral\_Data\_Size Struct Reference

Peripheral Data Size Configuration.

```
#include <DMA_Defs.h>
```

### Data Fields

- `uint32_t` [byte](#)
- `uint32_t` [half\\_word](#)
- `uint32_t` [word](#)

### 6.9.1 Detailed Description

Peripheral Data Size Configuration.

This structure defines the data size for the peripheral in DMA transfers, including byte, half-word, and word sizes.

## 6.9.2 Field Documentation

### 6.9.2.1 byte

`uint32_t byte`

Peripheral data size: byte

### 6.9.2.2 half\_word

```
uint32_t half_word
```

Peripheral data size: half-word

### 6.9.2.3 word

```
uint32_t word
```

Peripheral data size: word

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.10 DMA\_Configuration::Peripheral\_Pointer\_Increment Struct Reference

Peripheral Pointer Increment Configuration.

```
#include <DMA_Defs.h>
```

### Data Fields

- uint32\_t [Enable](#)
- uint32\_t [Disable](#)

### 6.10.1 Detailed Description

Peripheral Pointer Increment Configuration.

This structure defines the settings for peripheral pointer increment in DMA transfers. It allows the peripheral address to be incremented after each data transfer.

### 6.10.2 Field Documentation

#### 6.10.2.1 Disable

```
uint32_t Disable
```

Disable peripheral pointer increment

### 6.10.2.2 Enable

uint32\_t Enable

Enable peripheral pointer increment

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.11 DMA\_Configuration::Priority\_Level Struct Reference

Priority Level Configuration.

```
#include <DMA_Defs.h>
```

### Data Fields

- uint32\_t [Low](#)
- uint32\_t [Medium](#)
- uint32\_t [High](#)
- uint32\_t [Very\\_high](#)

### 6.11.1 Detailed Description

Priority Level Configuration.

This structure defines the priority levels for DMA transfers, ranging from low to very high priority.

### 6.11.2 Field Documentation

#### 6.11.2.1 High

uint32\_t High

High priority level

#### 6.11.2.2 Low

uint32\_t Low

Low priority level



### 6.11.2.3 Medium

```
uint32_t Medium
```

Medium priority level

### 6.11.2.4 Very\_high

```
uint32_t Very_high
```

Very high priority level

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.12 DMA\_Configuration::Request Struct Reference

DMA [Request](#) Channels.

```
#include <DMA_Defs.h>
```

### Data Fields

- [DMA\\_Request SPI3\\_RX](#)
- [DMA\\_Request SPI3\\_TX](#)
- [DMA\\_Request SPI2\\_RX](#)
- [DMA\\_Request SPI2\\_TX](#)
- [DMA\\_Request SPI1\\_RX](#)
- [DMA\\_Request SPI1\\_TX](#)
- [DMA\\_Request I2S2\\_RX](#)
- [DMA\\_Request I2S2\\_TX](#)
- [DMA\\_Request I2S3\\_RX](#)
- [DMA\\_Request I2S3\\_TX](#)
- [DMA\\_Request I2C1\\_RX](#)
- [DMA\\_Request I2C1\\_TX](#)
- [DMA\\_Request I2C2\\_RX](#)
- [DMA\\_Request I2C2\\_TX](#)
- [DMA\\_Request I2C3\\_RX](#)
- [DMA\\_Request I2C3\\_TX](#)
- [DMA\\_Request USART1\\_RX](#)
- [DMA\\_Request USART1\\_TX](#)
- [DMA\\_Request USART2\\_RX](#)
- [DMA\\_Request USART2\\_TX](#)
- [DMA\\_Request USART3\\_RX](#)
- [DMA\\_Request USART3\\_TX](#)
- [DMA\\_Request UART4\\_RX](#)
- [DMA\\_Request UART4\\_TX](#)
- [DMA\\_Request UART5\\_RX](#)
- [DMA\\_Request UART5\\_TX](#)

- [DMA\\_Request UART6\\_RX](#)
- [DMA\\_Request UART6\\_TX](#)
- [DMA\\_Request UART7\\_RX](#)
- [DMA\\_Request UART7\\_TX](#)
- [DMA\\_Request UART8\\_RX](#)
- [DMA\\_Request UART8\\_TX](#)
- [DMA\\_Request TIM1\\_UP](#)
- [DMA\\_Request TIM1\\_CH1](#)
- [DMA\\_Request TIM1\\_CH2](#)
- [DMA\\_Request TIM1\\_CH3](#)
- [DMA\\_Request TIM1\\_CH4](#)
- [DMA\\_Request TIM1\\_TRIG](#)
- [DMA\\_Request TIM1\\_COM](#)
- [DMA\\_Request TIM8\\_UP](#)
- [DMA\\_Request TIM8\\_CH1](#)
- [DMA\\_Request TIM8\\_CH2](#)
- [DMA\\_Request TIM8\\_CH3](#)
- [DMA\\_Request TIM8\\_CH4](#)
- [DMA\\_Request TIM8\\_TRIG](#)
- [DMA\\_Request TIM8\\_COM](#)
- [DMA\\_Request TIM2\\_UP](#)
- [DMA\\_Request TIM2\\_CH1](#)
- [DMA\\_Request TIM2\\_CH2](#)
- [DMA\\_Request TIM2\\_CH3](#)
- [DMA\\_Request TIM2\\_CH4](#)
- [DMA\\_Request TIM3\\_CH1](#)
- [DMA\\_Request TIM3\\_CH2](#)
- [DMA\\_Request TIM3\\_CH3](#)
- [DMA\\_Request TIM3\\_CH4](#)
- [DMA\\_Request TIM3\\_UP](#)
- [DMA\\_Request TIM3\\_TRIG](#)
- [DMA\\_Request TIM4\\_CH1](#)
- [DMA\\_Request TIM4\\_CH2](#)
- [DMA\\_Request TIM4\\_CH3](#)
- [DMA\\_Request TIM4\\_UP](#)
- [DMA\\_Request TIM5\\_CH1](#)
- [DMA\\_Request TIM5\\_CH2](#)
- [DMA\\_Request TIM5\\_CH3](#)
- [DMA\\_Request TIM5\\_CH4](#)
- [DMA\\_Request TIM5\\_UP](#)
- [DMA\\_Request TIM5\\_TRIG](#)
- [DMA\\_Request TIM6\\_UP](#)
- [DMA\\_Request TIM7\\_UP](#)
- [DMA\\_Request \\_DAC1](#)
- [DMA\\_Request \\_DAC2](#)
- [DMA\\_Request SDIO\\_RTX](#)
- [DMA\\_Request \\_DCMI](#)
- [DMA\\_Request \\_ADC1](#)
- [DMA\\_Request \\_ADC2](#)
- [DMA\\_Request \\_ADC3](#)

## 6.12.1 Detailed Description

DMA [Request](#) Channels.

This structure contains predefined DMA requests for various peripherals such as SPI, I2C, USART, TIM, DAC, SDIO, and ADC.

## 6.12.2 Field Documentation

### 6.12.2.1 `_ADC1`

[DMA\\_Request](#) `_ADC1`

DMA request for ADC1

### 6.12.2.2 `_ADC2`

[DMA\\_Request](#) `_ADC2`

DMA request for ADC2

### 6.12.2.3 `_ADC3`

[DMA\\_Request](#) `_ADC3`

DMA request for ADC3

### 6.12.2.4 `_DAC1`

[DMA\\_Request](#) `_DAC1`

DMA request for DAC1

### 6.12.2.5 `_DAC2`

[DMA\\_Request](#) `_DAC2`

DMA request for DAC2

### 6.12.2.6 `_DCMI`

[DMA\\_Request](#) `_DCMI`

DMA request for DCMI

#### 6.12.2.7 I2C1\_RX

[DMA\\_Request](#) I2C1\_RX

DMA request for I2C1 RX

#### 6.12.2.8 I2C1\_TX

[DMA\\_Request](#) I2C1\_TX

DMA request for I2C1 TX

#### 6.12.2.9 I2C2\_RX

[DMA\\_Request](#) I2C2\_RX

DMA request for I2C2 RX

#### 6.12.2.10 I2C2\_TX

[DMA\\_Request](#) I2C2\_TX

DMA request for I2C2 TX

#### 6.12.2.11 I2C3\_RX

[DMA\\_Request](#) I2C3\_RX

DMA request for I2C3 RX

#### 6.12.2.12 I2C3\_TX

[DMA\\_Request](#) I2C3\_TX

DMA request for I2C3 TX

#### 6.12.2.13 I2S2\_RX

[DMA\\_Request](#) I2S2\_RX

DMA request for I2S2 RX

#### 6.12.2.14 I2S2\_TX

[DMA\\_Request](#) I2S2\_TX

DMA request for I2S2 TX

### 6.12.2.15 I2S3\_RX

`DMA_Request` I2S3\_RX

DMA request for I2S3 RX

### 6.12.2.16 I2S3\_TX

`DMA_Request` I2S3\_TX

DMA request for I2S3 TX

### 6.12.2.17 SDIO\_RXTX

`DMA_Request` SDIO\_RXTX

DMA request for SDIO RX/TX

### 6.12.2.18 SPI1\_RX

`DMA_Request` SPI1\_RX

DMA request for SPI1 RX

### 6.12.2.19 SPI1\_TX

`DMA_Request` SPI1\_TX

DMA request for SPI1 TX

### 6.12.2.20 SPI2\_RX

`DMA_Request` SPI2\_RX

DMA request for SPI2 RX

### 6.12.2.21 SPI2\_TX

`DMA_Request` SPI2\_TX

DMA request for SPI2 TX

### 6.12.2.22 SPI3\_RX

`DMA_Request` SPI3\_RX

DMA request for SPI3 RX

#### 6.12.2.23 SPI3\_TX

`DMA_Request` SPI3\_TX

DMA request for SPI3 TX

#### 6.12.2.24 TIM1\_CH1

`DMA_Request` TIM1\_CH1

DMA request for TIM1 channel 1

#### 6.12.2.25 TIM1\_CH2

`DMA_Request` TIM1\_CH2

DMA request for TIM1 channel 2

#### 6.12.2.26 TIM1\_CH3

`DMA_Request` TIM1\_CH3

DMA request for TIM1 channel 3

#### 6.12.2.27 TIM1\_CH4

`DMA_Request` TIM1\_CH4

DMA request for TIM1 channel 4

#### 6.12.2.28 TIM1\_COM

`DMA_Request` TIM1\_COM

DMA request for TIM1 commutation

#### 6.12.2.29 TIM1\_TRIG

`DMA_Request` TIM1\_TRIG

DMA request for TIM1 trigger

#### 6.12.2.30 TIM1\_UP

`DMA_Request` TIM1\_UP

DMA request for TIM1 update

### 6.12.2.31 TIM2\_CH1

`DMA_Request` TIM2\_CH1

DMA request for TIM2 channel 1

### 6.12.2.32 TIM2\_CH2

`DMA_Request` TIM2\_CH2

DMA request for TIM2 channel 2

### 6.12.2.33 TIM2\_CH3

`DMA_Request` TIM2\_CH3

DMA request for TIM2 channel 3

### 6.12.2.34 TIM2\_CH4

`DMA_Request` TIM2\_CH4

DMA request for TIM2 channel 4

### 6.12.2.35 TIM2\_UP

`DMA_Request` TIM2\_UP

DMA request for TIM2 update

### 6.12.2.36 TIM3\_CH1

`DMA_Request` TIM3\_CH1

DMA request for TIM3 channel 1

### 6.12.2.37 TIM3\_CH2

`DMA_Request` TIM3\_CH2

DMA request for TIM3 channel 2

### 6.12.2.38 TIM3\_CH3

`DMA_Request` TIM3\_CH3

DMA request for TIM3 channel 3

#### 6.12.2.39 TIM3\_CH4

[DMA\\_Request](#) TIM3\_CH4

DMA request for TIM3 channel 4

#### 6.12.2.40 TIM3\_TRIG

[DMA\\_Request](#) TIM3\_TRIG

DMA request for TIM3 trigger

#### 6.12.2.41 TIM3\_UP

[DMA\\_Request](#) TIM3\_UP

DMA request for TIM3 update

#### 6.12.2.42 TIM4\_CH1

[DMA\\_Request](#) TIM4\_CH1

DMA request for TIM4 channel 1

#### 6.12.2.43 TIM4\_CH2

[DMA\\_Request](#) TIM4\_CH2

DMA request for TIM4 channel 2

#### 6.12.2.44 TIM4\_CH3

[DMA\\_Request](#) TIM4\_CH3

DMA request for TIM4 channel 3

#### 6.12.2.45 TIM4\_UP

[DMA\\_Request](#) TIM4\_UP

DMA request for TIM4 update

#### 6.12.2.46 TIM5\_CH1

[DMA\\_Request](#) TIM5\_CH1

DMA request for TIM5 channel 1



#### 6.12.2.47 TIM5\_CH2

`DMA_Request` TIM5\_CH2

DMA request for TIM5 channel 2

#### 6.12.2.48 TIM5\_CH3

`DMA_Request` TIM5\_CH3

DMA request for TIM5 channel 3

#### 6.12.2.49 TIM5\_CH4

`DMA_Request` TIM5\_CH4

DMA request for TIM5 channel 4

#### 6.12.2.50 TIM5\_TRIG

`DMA_Request` TIM5\_TRIG

DMA request for TIM5 trigger

#### 6.12.2.51 TIM5\_UP

`DMA_Request` TIM5\_UP

DMA request for TIM5 update

#### 6.12.2.52 TIM6\_UP

`DMA_Request` TIM6\_UP

DMA request for TIM6 update

#### 6.12.2.53 TIM7\_UP

`DMA_Request` TIM7\_UP

DMA request for TIM7 update

#### 6.12.2.54 TIM8\_CH1

`DMA_Request` TIM8\_CH1

DMA request for TIM8 channel 1

#### 6.12.2.55 TIM8\_CH2

[DMA\\_Request](#) TIM8\_CH2

DMA request for TIM8 channel 2

#### 6.12.2.56 TIM8\_CH3

[DMA\\_Request](#) TIM8\_CH3

DMA request for TIM8 channel 3

#### 6.12.2.57 TIM8\_CH4

[DMA\\_Request](#) TIM8\_CH4

DMA request for TIM8 channel 4

#### 6.12.2.58 TIM8\_COM

[DMA\\_Request](#) TIM8\_COM

DMA request for TIM8 commutation

#### 6.12.2.59 TIM8\_TRIG

[DMA\\_Request](#) TIM8\_TRIG

DMA request for TIM8 trigger

#### 6.12.2.60 TIM8\_UP

[DMA\\_Request](#) TIM8\_UP

DMA request for TIM8 update

#### 6.12.2.61 UART4\_RX

[DMA\\_Request](#) UART4\_RX

DMA request for UART4 RX

#### 6.12.2.62 UART4\_TX

[DMA\\_Request](#) UART4\_TX

DMA request for UART4 TX

### 6.12.2.63 UART5\_RX

[DMA\\_Request](#) UART5\_RX

DMA request for UART5 RX

### 6.12.2.64 UART5\_TX

[DMA\\_Request](#) UART5\_TX

DMA request for UART5 TX

### 6.12.2.65 UART6\_RX

[DMA\\_Request](#) UART6\_RX

DMA request for UART6 RX

### 6.12.2.66 UART6\_TX

[DMA\\_Request](#) UART6\_TX

DMA request for UART6 TX

### 6.12.2.67 UART7\_RX

[DMA\\_Request](#) UART7\_RX

DMA request for UART7 RX

### 6.12.2.68 UART7\_TX

[DMA\\_Request](#) UART7\_TX

DMA request for UART7 TX

### 6.12.2.69 UART8\_RX

[DMA\\_Request](#) UART8\_RX

DMA request for UART8 RX

### 6.12.2.70 UART8\_TX

[DMA\\_Request](#) UART8\_TX

DMA request for UART8 TX

#### 6.12.2.71 USART1\_RX

[DMA\\_Request](#) USART1\_RX

DMA request for USART1 RX

#### 6.12.2.72 USART1\_TX

[DMA\\_Request](#) USART1\_TX

DMA request for USART1 TX

#### 6.12.2.73 USART2\_RX

[DMA\\_Request](#) USART2\_RX

DMA request for USART2 RX

#### 6.12.2.74 USART2\_TX

[DMA\\_Request](#) USART2\_TX

DMA request for USART2 TX

#### 6.12.2.75 USART3\_RX

[DMA\\_Request](#) USART3\_RX

DMA request for USART3 RX

#### 6.12.2.76 USART3\_TX

[DMA\\_Request](#) USART3\_TX

DMA request for USART3 TX

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)

## 6.13 DMA\_Configuration::Transfer\_Direction Struct Reference

Transfer Direction Configuration.

```
#include <DMA_Defs.h>
```

## Data Fields

- uint32\_t [Peripheral\\_to\\_memory](#)
- uint32\_t [Memory\\_to\\_peripheral](#)
- uint32\_t [Memory\\_to\\_memory](#)

### 6.13.1 Detailed Description

Transfer Direction Configuration.

This structure defines the direction of data transfer for DMA, including Peripheral-to-Memory, Memory-to-Peripheral, and Memory-to-Memory transfers.

### 6.13.2 Field Documentation

#### 6.13.2.1 Memory\_to\_memory

```
uint32_t Memory_to_memory
```

Transfer data from memory to memory

#### 6.13.2.2 Memory\_to\_peripheral

```
uint32_t Memory_to_peripheral
```

Transfer data from memory to peripheral

#### 6.13.2.3 Peripheral\_to\_memory

```
uint32_t Peripheral_to_memory
```

Transfer data from peripheral to memory

The documentation for this struct was generated from the following file:

- [DMA\\_Defs.h](#)



# Chapter 7

## File Documentation

### 7.1 DMA.c File Reference

DMA Driver Implementation for STM32F407VGT6.

```
#include "DMA.h"
```

#### Functions

- void [DMA1\\_Stream0\\_IRQHandler](#) (void)  
*DMA1 Stream 0 Interrupt Handler.*
- void [DMA1\\_Stream1\\_IRQHandler](#) (void)  
*DMA1 Stream 1 Interrupt Handler.*
- void [DMA1\\_Stream2\\_IRQHandler](#) (void)  
*DMA1 Stream 2 Interrupt Handler.*
- void [DMA1\\_Stream3\\_IRQHandler](#) (void)  
*DMA1 Stream 3 Interrupt Handler.*
- void [DMA1\\_Stream4\\_IRQHandler](#) (void)  
*DMA1 Stream 4 Interrupt Handler.*
- void [DMA1\\_Stream5\\_IRQHandler](#) (void)  
*DMA1 Stream 5 Interrupt Handler.*
- void [DMA1\\_Stream6\\_IRQHandler](#) (void)  
*DMA1 Stream 6 Interrupt Handler.*
- void [DMA1\\_Stream7\\_IRQHandler](#) (void)  
*DMA1 Stream 7 Interrupt Handler.*
- void [DMA2\\_Stream0\\_IRQHandler](#) (void)  
*DMA2 Stream 0 Interrupt Handler.*
- void [DMA2\\_Stream1\\_IRQHandler](#) (void)  
*DMA2 Stream 1 Interrupt Handler.*
- void [DMA2\\_Stream2\\_IRQHandler](#) (void)  
*DMA2 Stream 2 Interrupt Handler.*
- void [DMA2\\_Stream3\\_IRQHandler](#) (void)  
*DMA2 Stream 3 Interrupt Handler.*
- void [DMA2\\_Stream4\\_IRQHandler](#) (void)

- DMA2 Stream 4 Interrupt Handler.*

  - void [DMA2\\_Stream5\\_IRQHandler](#) (void)
- DMA2 Stream 5 Interrupt Handler.*

  - void [DMA2\\_Stream6\\_IRQHandler](#) (void)
- DMA2 Stream 6 Interrupt Handler.*

  - void [DMA2\\_Stream7\\_IRQHandler](#) (void)
- DMA2 Stream 7 Interrupt Handler.*

  - void [DMA\\_Reset\\_Flags](#) ([DMA\\_Flags\\_Typedef](#) flag)

*Resets all DMA flags in the provided [DMA\\_Flags\\_Typedef](#) structure.*
- void [DMA\\_Clock\\_Enable](#) ([DMA\\_Config](#) \*config)

*Enables the clock for the specified DMA controller.*
- void [DMA\\_Clock\\_Disable](#) ([DMA\\_Config](#) \*config)

*Disables the clock for the specified DMA controller.*
- void [DMA\\_Reset](#) ([DMA\\_Config](#) \*config)

*Resets the specified DMA controller.*
- int8\_t [DMA\\_Init](#) ([DMA\\_Config](#) \*config)

*Initializes the DMA with the specified configuration.*
- void [DMA\\_Set\\_Target](#) ([DMA\\_Config](#) \*config)

*Configures the target memory and peripheral for DMA transfers.*
- void [DMA\\_Set\\_Trigger](#) ([DMA\\_Config](#) \*config)

*Sets up and enables the DMA stream for data transfer.*
- void [DMA\\_Memory\\_To\\_Memory\\_Transfer](#) (uint32\_t \*source, uint8\_t source\_data\_size, uint8\_t dest\_data\_size, uint32\_t \*destination, bool source\_increment, bool destination\_increment, uint16\_t length)

*Performs a memory-to-memory data transfer using DMA.*

## Variables

- [DMA\\_Flags\\_Typedef](#) [DMA1\\_Stream0\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA1\\_Stream1\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA1\\_Stream2\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA1\\_Stream3\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA1\\_Stream4\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA1\\_Stream5\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA1\\_Stream6\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA1\\_Stream7\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA2\\_Stream0\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA2\\_Stream1\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA2\\_Stream2\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA2\\_Stream3\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA2\\_Stream4\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA2\\_Stream5\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA2\\_Stream6\\_Flag](#)
- [DMA\\_Flags\\_Typedef](#) [DMA2\\_Stream7\\_Flag](#)



### 7.1.1 Detailed Description

DMA Driver Implementation for STM32F407VGT6.

This file provides the implementation of the DMA driver for the STM32F407VGT6 microcontroller. It includes functions to initialize DMA, handle DMA interrupts, configure DMA streams, and manage memory-to-memory transfers.

#### Version

1.0

#### Date

2024-08-21

#### Author

Your Name

#### Copyright

Copyright (c) 2024

### 7.1.2 Function Documentation

#### 7.1.2.1 DMA1\_Stream0\_IRQHandler()

```
void DMA1_Stream0_IRQHandler (  
    void )
```

DMA1 Stream 0 Interrupt Handler.

This function handles the interrupt for DMA1 Stream 0. It checks the status flags for FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete, and clears the respective interrupt flag after handling it.

#### 7.1.2.2 DMA1\_Stream1\_IRQHandler()

```
void DMA1_Stream1_IRQHandler (  
    void )
```

DMA1 Stream 1 Interrupt Handler.

This function handles the interrupt for DMA1 Stream 1. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA1_Stream1_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

### 7.1.2.3 DMA1\_Stream2\_IRQHandler()

```
void DMA1_Stream2_IRQHandler (
    void )
```

DMA1 Stream 2 Interrupt Handler.

This function handles the interrupt for DMA1 Stream 2. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA1_Stream2_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

### 7.1.2.4 DMA1\_Stream3\_IRQHandler()

```
void DMA1_Stream3_IRQHandler (
    void )
```

DMA1 Stream 3 Interrupt Handler.

This function handles the interrupt for DMA1 Stream 3. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA1_Stream3_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

### 7.1.2.5 DMA1\_Stream4\_IRQHandler()

```
void DMA1_Stream4_IRQHandler (
    void )
```

DMA1 Stream 4 Interrupt Handler.

This function handles the interrupt for DMA1 Stream 4. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA1_Stream4_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

### 7.1.2.6 DMA1\_Stream5\_IRQHandler()

```
void DMA1_Stream5_IRQHandler (
    void )
```

DMA1 Stream 5 Interrupt Handler.

This function handles the interrupt for DMA1 Stream 5. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA1_Stream5_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.7 DMA1\_Stream6\_IRQHandler()

```
void DMA1_Stream6_IRQHandler (
    void )
```

DMA1 Stream 6 Interrupt Handler.

This function handles the interrupt for DMA1 Stream 6. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA1_Stream6_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.8 DMA1\_Stream7\_IRQHandler()

```
void DMA1_Stream7_IRQHandler (
    void )
```

DMA1 Stream 7 Interrupt Handler.

This function handles the interrupt for DMA1 Stream 7. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA1_Stream7_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.9 DMA2\_Stream0\_IRQHandler()

```
void DMA2_Stream0_IRQHandler (
    void )
```

DMA2 Stream 0 Interrupt Handler.

This function handles the interrupt for DMA2 Stream 0. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA2_Stream0_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.10 DMA2\_Stream1\_IRQHandler()

```
void DMA2_Stream1_IRQHandler (
    void )
```

DMA2 Stream 1 Interrupt Handler.

This function handles the interrupt for DMA2 Stream 1. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA2_Stream1_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.11 DMA2\_Stream2\_IRQHandler()

```
void DMA2_Stream2_IRQHandler (  
    void )
```

DMA2 Stream 2 Interrupt Handler.

This function handles the interrupt for DMA2 Stream 2. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA2_Stream2_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.12 DMA2\_Stream3\_IRQHandler()

```
void DMA2_Stream3_IRQHandler (  
    void )
```

DMA2 Stream 3 Interrupt Handler.

This function handles the interrupt for DMA2 Stream 3. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA2_Stream3_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.13 DMA2\_Stream4\_IRQHandler()

```
void DMA2_Stream4_IRQHandler (  
    void )
```

DMA2 Stream 4 Interrupt Handler.

This function handles the interrupt for DMA2 Stream 4. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA2_Stream4_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.14 DMA2\_Stream5\_IRQHandler()

```
void DMA2_Stream5_IRQHandler (  
    void )
```

DMA2 Stream 5 Interrupt Handler.

This function handles the interrupt for DMA2 Stream 5. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA2_Stream5_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.15 DMA2\_Stream6\_IRQHandler()

```
void DMA2_Stream6_IRQHandler (
    void )
```

DMA2 Stream 6 Interrupt Handler.

This function handles the interrupt for DMA2 Stream 6. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA2_Stream6_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.16 DMA2\_Stream7\_IRQHandler()

```
void DMA2_Stream7_IRQHandler (
    void )
```

DMA2 Stream 7 Interrupt Handler.

This function handles the interrupt for DMA2 Stream 7. It checks the status flags for different events such as FIFO error, direct mode error, transfer error, half transfer complete, and transfer complete. For each event, it sets the corresponding flag in the `DMA2_Stream7_Flag` structure and clears the respective interrupt flag in the DMA interrupt flag clear register.

#### 7.1.2.17 DMA\_Clock\_Disable()

```
void DMA_Clock_Disable (
    DMA_Config * config)
```

Disables the clock for the specified DMA controller.

This function disables the clock for the DMA controller specified in the `DMA_Config` structure. It checks whether the controller is `DMA1` or `DMA2` and then disables the corresponding clock by clearing the appropriate bit in the RCC AHB1 peripheral clock enable register.

##### Parameters

in	<i>config</i>	Pointer to the <code>DMA_Config</code> structure that contains the DMA controller configuration.
----	---------------	--

#### 7.1.2.18 DMA\_Clock\_Enable()

```
void DMA_Clock_Enable (
    DMA_Config * config)
```

Enables the clock for the specified DMA controller.

This function enables the clock for the DMA controller specified in the `DMA_Config` structure. It checks whether the controller is `DMA1` or `DMA2` and then enables the corresponding clock by setting the appropriate bit in the RCC AHB1 peripheral clock enable register.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure that contains the DMA controller configuration.
----	---------------	---

**7.1.2.19 DMA\_Init()**

```
int8_t DMA_Init (
    DMA\_Config * config)
```

Initializes the DMA with the specified configuration.

This function configures the DMA stream with the settings provided in the [DMA\\_Config](#) structure. It enables the clock for the specified DMA controller, sets up the DMA channel, circular mode, flow control, priority level, data sizes, transfer direction, and interrupts. If interrupts are enabled, it also configures the NVIC for the corresponding DMA stream.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the configuration parameters.
----	---------------	--

## Returns

int8\_t Returns 1 on successful initialization, or -1 if an error occurs.

**7.1.2.20 DMA\_Memory\_To\_Memory\_Transfer()**

```
void DMA_Memory_To_Memory_Transfer (
    uint32_t * source,
    uint8_t source_data_size,
    uint8_t dest_data_size,
    uint32_t * destination,
    bool source_increment,
    bool destination_increment,
    uint16_t length)
```

Performs a memory-to-memory data transfer using DMA.

This function configures and initiates a DMA transfer from a source memory location to a destination memory location. It sets up the data size, increment modes, and the length of the transfer. The function enables the DMA stream, waits for the transfer to complete, and then disables the stream.

## Parameters

in	<i>source</i>	Pointer to the source memory location.
in	<i>source_data_size</i>	Size of the data at the source (8, 16, or 32 bits).
in	<i>dest_data_size</i>	Size of the data at the destination (8, 16, or 32 bits).
in	<i>destination</i>	Pointer to the destination memory location.
in	<i>source_increment</i>	If true, the source address will be incremented after each transfer.
in	<i>destination_increment</i>	If true, the destination address will be incremented after each transfer.
in	<i>length</i>	Number of data items to transfer.

### 7.1.2.21 DMA\_Reset()

```
void DMA_Reset (
    DMA_Config * config)
```

Resets the specified DMA controller.

This function resets the DMA controller specified in the [DMA\\_Config](#) structure. It checks whether the controller is DMA1 or DMA2 and then triggers a reset by setting the appropriate bit in the RCC AHB1 peripheral reset register.

#### Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure that contains the DMA controller configuration.
----	---------------	---

### 7.1.2.22 DMA\_Reset\_Flags()

```
void DMA_Reset_Flags (
    DMA_Flags_Typedef flag)
```

Resets all DMA flags in the provided [DMA\\_Flags\\_Typedef](#) structure.

This function sets all the flags in the provided [DMA\\_Flags\\_Typedef](#) structure to `false`, effectively resetting the state of the flags that monitor DMA events such as direct mode error, FIFO error, half transfer complete, transfer complete, and transfer error.

#### Parameters

<i>flag</i>	The <a href="#">DMA_Flags_Typedef</a> structure whose flags are to be reset.
-------------	--

### 7.1.2.23 DMA\_Set\_Target()

```
void DMA_Set_Target (
    DMA_Config * config)
```

Configures the target memory and peripheral for DMA transfers.

This function sets up the DMA stream to transfer data between the specified memory and peripheral addresses. It configures the memory and peripheral data sizes, the number of data items to transfer, and the memory pointer increment mode. The function also clears the previous configurations for data size and memory increment before applying the new settings.

#### Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the target configuration.
----	---------------	--

### 7.1.2.24 DMA\_Set\_Trigger()

```
void DMA_Set_Trigger (
    DMA_Config * config)
```

Sets up and enables the DMA stream for data transfer.

This function configures the DMA stream trigger by clearing any pending interrupt flags for the specified stream and then enables the stream for data transfer. The function determines which register to modify (LIFCR or HIFCR) based on the stream number and the DMA controller (DMA1 or DMA2).

## Parameters

in	config	Pointer to the <a href="#">DMA_Config</a> structure that contains the configuration settings.
----	--------	---

## 7.2 DMA.h File Reference

Header file for the DMA driver.

```
#include "main.h"
#include "DMA_Defs.h"
```

### Data Structures

- struct [DMA\\_Config](#)  
*DMA configuration structure.*

### Typedefs

- typedef struct DMA\_Config [DMA\\_Config](#)  
*DMA configuration structure.*

### Functions

- void [DMA\\_Clock\\_Enable](#) ([DMA\\_Config](#) \*config)  
*Enables the clock for the specified DMA controller.*
- void [DMA\\_Clock\\_Disable](#) ([DMA\\_Config](#) \*config)  
*Disables the clock for the specified DMA controller.*
- void [DMA\\_Reset](#) ([DMA\\_Config](#) \*config)  
*Resets the specified DMA controller.*
- void [DMA\\_Reset\\_Flags](#) ([DMA\\_Flags\\_Typedef](#) flag)  
*Resets all DMA flags in the provided [DMA\\_Flags\\_Typedef](#) structure.*
- int8\_t [DMA\\_Init](#) ([DMA\\_Config](#) \*config)  
*Initializes the DMA with the specified configuration.*
- void [DMA\\_Set\\_Target](#) ([DMA\\_Config](#) \*config)  
*Configures the target memory and peripheral for DMA transfers.*
- void [DMA\\_Set\\_Trigger](#) ([DMA\\_Config](#) \*config)  
*Sets up and enables the DMA stream for data transfer.*
- void [DMA\\_Memory\\_To\\_Memory\\_Transfer](#) (uint32\_t \*source, uint8\_t source\_data\_size, uint8\_t dest\_data\_size, uint32\_t \*destination, bool source\_increment, bool destination\_increment, uint16\_t length)  
*Performs a memory-to-memory data transfer using DMA.*



## Variables

- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream0\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream1\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream2\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream3\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream4\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream5\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream6\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA1\_Stream7\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream0\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream1\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream2\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream3\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream4\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream5\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream6\_Flag**
- [DMA\\_Flags\\_Typedef](#) **DMA2\_Stream7\_Flag**

## 7.2.1 Detailed Description

Header file for the DMA driver.

### Author

Kunal Salvi

This file contains the function prototypes and data structures for configuring and using the DMA peripheral. It supports functions for initializing the DMA, setting up memory-to-memory transfers, and managing DMA flags and interrupts.

### Version

1.0

### Date

2024-08-22

### Copyright

Copyright (c) 2024

## 7.2.2 Typedef Documentation

### 7.2.2.1 DMA\_Config

```
typedef struct DMA_Config DMA_Config
```

DMA configuration structure.

This structure contains the configuration parameters for the DMA peripheral, including settings for flow control, transfer direction, priority level, circular mode, interrupts, memory and peripheral data sizes, and addresses.

## 7.2.3 Function Documentation

### 7.2.3.1 DMA\_Clock\_Disable()

```
void DMA_Clock_Disable (  
    DMA_Config * config)
```

Disables the clock for the specified DMA controller.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the DMA controller settings.
----	---------------	---

This function disables the clock for the DMA controller specified in the [DMA\\_Config](#) structure. It checks whether the controller is `DMA1` or `DMA2` and then disables the corresponding clock by clearing the appropriate bit in the RCC AHB1 peripheral clock enable register.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure that contains the DMA controller configuration.
----	---------------	---

### 7.2.3.2 DMA\_Clock\_Enable()

```
void DMA_Clock_Enable (  
    DMA\_Config * config)
```

Enables the clock for the specified DMA controller.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the DMA controller settings.
----	---------------	---

This function enables the clock for the DMA controller specified in the [DMA\\_Config](#) structure. It checks whether the controller is `DMA1` or `DMA2` and then enables the corresponding clock by setting the appropriate bit in the RCC AHB1 peripheral clock enable register.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure that contains the DMA controller configuration.
----	---------------	---

### 7.2.3.3 DMA\_Init()

```
int8_t DMA_Init (  
    DMA\_Config * config)
```

Initializes the DMA with the specified configuration.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the configuration parameters.
----	---------------	--

## Returns

`int8_t` Returns 1 on successful initialization, or -1 if an error occurs.

This function configures the DMA stream with the settings provided in the [DMA\\_Config](#) structure. It enables the clock for the specified DMA controller, sets up the DMA channel, circular mode, flow control, priority level, data sizes, transfer direction, and interrupts. If interrupts are enabled, it also configures the NVIC for the corresponding DMA stream.

**Parameters**

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the configuration parameters.
----	---------------	--

**Returns**

int8\_t Returns 1 on successful initialization, or -1 if an error occurs.

**7.2.3.4 DMA\_Memory\_To\_Memory\_Transfer()**

```
void DMA_Memory_To_Memory_Transfer (
    uint32_t * source,
    uint8_t source_data_size,
    uint8_t dest_data_size,
    uint32_t * destination,
    bool source_increment,
    bool destination_increment,
    uint16_t length)
```

Performs a memory-to-memory data transfer using DMA.

**Parameters**

in	<i>source</i>	Pointer to the source memory location.
in	<i>source_data_size</i>	Size of the data at the source (8, 16, or 32 bits).
in	<i>dest_data_size</i>	Size of the data at the destination (8, 16, or 32 bits).
in	<i>destination</i>	Pointer to the destination memory location.
in	<i>source_increment</i>	If true, the source address will be incremented after each transfer.
in	<i>destination_increment</i>	If true, the destination address will be incremented after each transfer.
in	<i>length</i>	Number of data items to transfer.

This function configures and initiates a DMA transfer from a source memory location to a destination memory location. It sets up the data size, increment modes, and the length of the transfer. The function enables the DMA stream, waits for the transfer to complete, and then disables the stream.

**Parameters**

in	<i>source</i>	Pointer to the source memory location.
in	<i>source_data_size</i>	Size of the data at the source (8, 16, or 32 bits).
in	<i>dest_data_size</i>	Size of the data at the destination (8, 16, or 32 bits).
in	<i>destination</i>	Pointer to the destination memory location.
in	<i>source_increment</i>	If true, the source address will be incremented after each transfer.
in	<i>destination_increment</i>	If true, the destination address will be incremented after each transfer.
in	<i>length</i>	Number of data items to transfer.

**7.2.3.5 DMA\_Reset()**

```
void DMA_Reset (
    DMA_Config * config)
```

Resets the specified DMA controller.

## Parameters

<i>in</i>	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the DMA controller settings.
-----------	---------------	---

This function resets the DMA controller specified in the [DMA\\_Config](#) structure. It checks whether the controller is DMA1 or DMA2 and then triggers a reset by setting the appropriate bit in the RCC AHB1 peripheral reset register.

## Parameters

<i>in</i>	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure that contains the DMA controller configuration.
-----------	---------------	---

## 7.2.3.6 DMA\_Reset\_Flags()

```
void DMA_Reset_Flags (
    DMA\_Flags\_Typedef flag)
```

Resets all DMA flags in the provided [DMA\\_Flags\\_Typedef](#) structure.

## Parameters

<i>flag</i>	The <a href="#">DMA_Flags_Typedef</a> structure whose flags are to be reset.
-------------	--

This function sets all the flags in the provided [DMA\\_Flags\\_Typedef](#) structure to `false`, effectively resetting the state of the flags that monitor DMA events such as direct mode error, FIFO error, half transfer complete, transfer complete, and transfer error.

## Parameters

<i>flag</i>	The <a href="#">DMA_Flags_Typedef</a> structure whose flags are to be reset.
-------------	--

## 7.2.3.7 DMA\_Set\_Target()

```
void DMA_Set_Target (
    DMA\_Config * config)
```

Configures the target memory and peripheral for DMA transfers.

## Parameters

<i>in</i>	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the target configuration.
-----------	---------------	--

This function sets up the DMA stream to transfer data between the specified memory and peripheral addresses. It configures the memory and peripheral data sizes, the number of data items to transfer, and the memory pointer increment mode. The function also clears the previous configurations for data size and memory increment before applying the new settings.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the target configuration.
----	---------------	--

**7.2.3.8 DMA\_Set\_Trigger()**

```
void DMA_Set_Trigger (
    DMA_Config * config)
```

Sets up and enables the DMA stream for data transfer.

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure containing the configuration settings.
----	---------------	--

This function configures the DMA stream trigger by clearing any pending interrupt flags for the specified stream and then enables the stream for data transfer. The function determines which register to modify (LIFCR or HIFCR) based on the stream number and the DMA controller (DMA1 or DMA2).

## Parameters

in	<i>config</i>	Pointer to the <a href="#">DMA_Config</a> structure that contains the configuration settings.
----	---------------	---

**7.3 DMA.h**

[Go to the documentation of this file.](#)

```
00001
00110 #ifndef DMA_H_
00111 #define DMA_H_
00112
00113 #include "main.h"
00114 #include "DMA_Defs.h"
00115
00119 extern DMA_Flags_Typedef DMA1_Stream0_Flag;
00120 extern DMA_Flags_Typedef DMA1_Stream1_Flag;
00121 extern DMA_Flags_Typedef DMA1_Stream2_Flag;
00122 extern DMA_Flags_Typedef DMA1_Stream3_Flag;
00123 extern DMA_Flags_Typedef DMA1_Stream4_Flag;
00124 extern DMA_Flags_Typedef DMA1_Stream5_Flag;
00125 extern DMA_Flags_Typedef DMA1_Stream6_Flag;
00126 extern DMA_Flags_Typedef DMA1_Stream7_Flag;
00127
00128 extern DMA_Flags_Typedef DMA2_Stream0_Flag;
00129 extern DMA_Flags_Typedef DMA2_Stream1_Flag;
00130 extern DMA_Flags_Typedef DMA2_Stream2_Flag;
00131 extern DMA_Flags_Typedef DMA2_Stream3_Flag;
00132 extern DMA_Flags_Typedef DMA2_Stream4_Flag;
00133 extern DMA_Flags_Typedef DMA2_Stream5_Flag;
00134 extern DMA_Flags_Typedef DMA2_Stream6_Flag;
00135 extern DMA_Flags_Typedef DMA2_Stream7_Flag;
00145 typedef struct DMA_Config
00146 {
00147     DMA_Request Request;
00148     uint32_t flow_control;
00149     uint32_t transfer_direction;
00150     uint32_t priority_level;
00151     uint32_t circular_mode;
00152     uint32_t interrupts;
00153     uint16_t memory_pointer_increment;
00154     uint16_t peripheral_pointer_increment;
00155     uint32_t peripheral_data_size;
00156     uint32_t memory_data_size;
```

```

00157     uint32_t peripheral_address;
00158     uint32_t memory_address;
00159     uint16_t buffer_length;
00160 } DMA_Config;
00161
00167 void DMA_Clock_Enable(DMA_Config *config);
00168
00174 void DMA_Clock_Disable(DMA_Config *config);
00175
00181 void DMA_Reset(DMA_Config *config);
00182
00188 void DMA_Reset_Flags(DMA_Flags_Typedef flag);
00189
00197 int8_t DMA_Init(DMA_Config *config);
00198
00204 void DMA_Set_Target(DMA_Config *config);
00205
00211 void DMA_Set_Trigger(DMA_Config *config);
00212
00224 void DMA_Memory_To_Memory_Transfer(uint32_t *source,
00225                                     uint8_t source_data_size, uint8_t dest_data_size,
00226                                     uint32_t *destination, bool source_increment,
00227                                     bool destination_increment, uint16_t length);
00228
00229 #endif /* DMA_H_ */

```

## 7.4 DMA\_Defs.h File Reference

DMA Configuration Definitions for STM32F407VGT6.

```
#include "main.h"
```

### Data Structures

- struct [DMA\\_Flags\\_Typedef](#)  
*DMA Flags Structure.*
- struct [DMA\\_Request](#)  
*DMA Request Structure.*
- struct **DMA\_Configuration**  
*DMA Configuration Structure.*
- struct [DMA\\_Configuration::Request](#)  
*DMA Request Channels.*
- struct [DMA\\_Configuration::Flow\\_Control](#)  
*Flow Control Configuration.*
- struct [DMA\\_Configuration::Transfer\\_Direction](#)  
*Transfer Direction Configuration.*
- struct [DMA\\_Configuration::Priority\\_Level](#)  
*Priority Level Configuration.*
- struct [DMA\\_Configuration::Peripheral\\_Data\\_Size](#)  
*Peripheral Data Size Configuration.*
- struct [DMA\\_Configuration::Memory\\_Data\\_Size](#)  
*Memory Data Size Configuration.*
- struct [DMA\\_Configuration::Circular\\_Mode](#)  
*Circular Mode Configuration.*
- struct [DMA\\_Configuration::DMA\\_Interrupts](#)  
*DMA Interrupt Configuration.*
- struct [DMA\\_Configuration::Memory\\_Pointer\\_Increment](#)  
*Memory Pointer Increment Configuration.*
- struct [DMA\\_Configuration::Peripheral\\_Pointer\\_Increment](#)  
*Peripheral Pointer Increment Configuration.*

## Typedefs

- typedef struct DMA\_Flags\_Typedef [DMA\\_Flags\\_Typedef](#)  
*DMA Flags Structure.*
- typedef struct DMA\_Request [DMA\\_Request](#)  
*DMA Request Structure.*

### 7.4.1 Detailed Description

DMA Configuration Definitions for STM32F407VGT6.

This file contains the definitions and structures required for configuring the DMA (Direct Memory Access) controller on the STM32F407VGT6 microcontroller. It includes various configurations such as request channels, flow control, transfer direction, priority levels, and interrupt settings.

#### Version

1.0

#### Date

2024-08-21

#### Author

Your Name

#### Copyright

Copyright (c) 2024

### 7.4.2 Typedef Documentation

#### 7.4.2.1 DMA\_Flags\_Typedef

```
typedef struct DMA_Flags_Typedef DMA_Flags_Typedef
```

DMA Flags Structure.

This structure contains flags that indicate the status of a DMA transfer. These flags are used to monitor and handle various states of the DMA transfer process.

#### 7.4.2.2 DMA\_Request

```
typedef struct DMA_Request DMA_Request
```

DMA Request Structure.

This structure contains the configuration for a specific DMA request, including the DMA controller, stream, and channel associated with the request.



## 7.5 DMA\_Defs.h

[Go to the documentation of this file.](#)

```

00001
00017 #ifndef DMA_DEFS_H_
00018 #define DMA_DEFS_H_
00019
00020 #include "main.h"
00021
00028 typedef struct DMA_Flags_Typedef
00029 {
00030     bool Transfer_Complete_Flag;
00031     bool Half_Transfer_Complete_Flag;
00032     bool Transfer_Error_Flag;
00033     bool Direct_Mode_Error_Flag;
00034     bool Fifo_Error_Flag;
00036 }DMA_Flags_Typedef;
00037
00044 typedef struct DMA_Request {
00045     DMA_TypeDef *Controller;
00046     DMA_Stream_TypeDef *Stream;
00047     uint8_t channel;
00048 } DMA_Request;
00049
00050
00058 static const struct DMA_Configuration {
00059
00066     struct Request{
00067
00068         DMA_Request SPI3_RX;
00069         DMA_Request SPI3_TX;
00070         DMA_Request SPI2_RX;
00071         DMA_Request SPI2_TX;
00072         DMA_Request SPI1_RX;
00073         DMA_Request SPI1_TX;
00074         DMA_Request I2S2_RX;
00075         DMA_Request I2S2_TX;
00076         DMA_Request I2S3_RX;
00077         DMA_Request I2S3_TX;
00078         DMA_Request I2C1_RX;
00079         DMA_Request I2C1_TX;
00080         DMA_Request I2C2_RX;
00081         DMA_Request I2C2_TX;
00082         DMA_Request I2C3_RX;
00083         DMA_Request I2C3_TX;
00084         DMA_Request USART1_RX;
00085         DMA_Request USART1_TX;
00086         DMA_Request USART2_RX;
00087         DMA_Request USART2_TX;
00088         DMA_Request USART3_RX;
00089         DMA_Request USART3_TX;
00090         DMA_Request UART4_RX;
00091         DMA_Request UART4_TX;
00092         DMA_Request UART5_RX;
00093         DMA_Request UART5_TX;
00094         DMA_Request UART6_RX;
00095         DMA_Request UART6_TX;
00096         DMA_Request UART7_RX;
00097         DMA_Request UART7_TX;
00098         DMA_Request UART8_RX;
00099         DMA_Request UART8_TX;
00100         DMA_Request TIM1_UP;
00101         DMA_Request TIM1_CH1;
00102         DMA_Request TIM1_CH2;
00103         DMA_Request TIM1_CH3;
00104         DMA_Request TIM1_CH4;
00105         DMA_Request TIM1_TRIG;
00106         DMA_Request TIM1_COM;
00107         DMA_Request TIM8_UP;
00108         DMA_Request TIM8_CH1;
00109         DMA_Request TIM8_CH2;
00110         DMA_Request TIM8_CH3;
00111         DMA_Request TIM8_CH4;
00112         DMA_Request TIM8_TRIG;
00113         DMA_Request TIM8_COM;
00114         DMA_Request TIM2_UP;
00115         DMA_Request TIM2_CH1;
00116         DMA_Request TIM2_CH2;
00117         DMA_Request TIM2_CH3;
00118         DMA_Request TIM2_CH4;
00119         DMA_Request TIM3_CH1;
00120         DMA_Request TIM3_CH2;
00121         DMA_Request TIM3_CH3;
00122         DMA_Request TIM3_CH4;
00123         DMA_Request TIM3_UP;

```

```

00124     DMA_Request TIM3_TRIG;
00125     DMA_Request TIM4_CH1;
00126     DMA_Request TIM4_CH2;
00127     DMA_Request TIM4_CH3;
00128     DMA_Request TIM4_UP;
00129     DMA_Request TIM5_CH1;
00130     DMA_Request TIM5_CH2;
00131     DMA_Request TIM5_CH3;
00132     DMA_Request TIM5_CH4;
00133     DMA_Request TIM5_UP;
00134     DMA_Request TIM5_TRIG;
00135     DMA_Request TIM6_UP;
00136     DMA_Request TIM7_UP;
00137     DMA_Request _DAC1;
00138     DMA_Request _DAC2;
00139     DMA_Request SDIO_RXTX;
00140     DMA_Request _DCMI;
00141     DMA_Request _ADC1;
00142     DMA_Request _ADC2;
00143     DMA_Request _ADC3;
00144     }Request;
00145
00152     struct Flow_Control
00153     {
00154         uint32_t DMA_Control;
00155         uint32_t Peripheral_Control;
00156     }Flow_Control;
00157
00164     struct Transfer_Direction {
00165         uint32_t Peripheral_to_memory;
00166         uint32_t Memory_to_peripheral;
00167         uint32_t Memory_to_memory;
00168     } Transfer_Direction;
00169
00176     struct Priority_Level {
00177         uint32_t Low;
00178         uint32_t Medium;
00179         uint32_t High;
00180         uint32_t Very_high;
00181     } Priority_Level;
00182
00189     struct Peripheral_Data_Size
00190     {
00191         uint32_t byte;
00192         uint32_t half_word;
00193         uint32_t word;
00194     }Peripheral_Data_Size;
00195
00202     struct Memory_Data_Size
00203     {
00204         uint32_t byte;
00205         uint32_t half_word;
00206         uint32_t word;
00207     }Memory_Data_Size;
00208
00215     struct Circular_Mode
00216     {
00217         uint32_t Enable;
00218         uint32_t Disable;
00219     }Circular_Mode;
00220
00228     struct DMA_Interrupts
00229     {
00230         uint32_t Transfer_Complete;
00231         uint32_t Half_Transfer_Complete;
00232         uint32_t Transfer_Error;
00233         uint32_t Direct_Mode_Error;
00234         uint32_t Fifo_Error;
00235         uint32_t Disable;
00236     }DMA_Interrupts;
00237
00244     struct Memory_Pointer_Increment
00245     {
00246         uint32_t Enable;
00247         uint32_t Disable;
00248     }Memory_Pointer_Increment;
00249
00256     struct Peripheral_Pointer_Increment {
00257         uint32_t Enable;
00258         uint32_t Disable;
00259     } Peripheral_Pointer_Increment;
00260
00261
00262 } DMA_Configuration = {
00263
00264     .Circular_Mode = {
00265

```

```

00266         .Enable = 1 « DMA_SxCR_CIRC_Pos,
00267         .Disable = 0 « DMA_SxCR_CIRC_Pos,
00268     },
00269
00270     .Memory_Pointer_Increment = {
00271         .Enable = 1 « 10,
00272         .Disable = 0 « 10,
00273     },
00274
00275     .Peripheral_Pointer_Increment = {
00276         .Enable = 1 « 9,
00277         .Disable = 0 « 9,
00278     },
00279
00280     .DMA_Interrupts = {
00281         .Transfer_Complete = 1 « 4,
00282         .Half_Transfer_Complete = 1 « 3,
00283         .Transfer_Error = 1 « 2,
00284         .Direct_Mode_Error = 1 « 1,
00285         .Fifo_Error = 1 « 7,
00286         .Disable = 0 « 1,
00287     },
00288
00289     .Flow_Control = {
00290         .DMA_Control = 0 « 5,
00291         .Peripheral_Control = 1 « 5,
00292     },
00293
00294     .Transfer_Direction =
00295     {
00296         .Peripheral_to_memory = 0 « 6,
00297         .Memory_to_peripheral = 1 « 6,
00298         .Memory_to_memory = 2 « 6,
00299     },
00300
00301     .Priority_Level =
00302     {
00303         .Low = 0 « 16,
00304         .Medium = 1 « 16,
00305         .High = 2 « 16,
00306         .Very_high = 3 « 16,
00307     },
00308
00309     .Memory_Data_Size = {
00310
00311         .byte = 0 « 13,
00312         .half_word = 1 « 13,
00313         .word = 2 « 13,
00314
00315     },
00316
00317     .Peripheral_Data_Size = {
00318         .byte = 0 « 11,
00319         .half_word = 1 « 11,
00320         .word = 2 « 11,
00321     },
00322
00323     .Request = {
00324
00325         .SPI3_RX = {
00326             .Controller = DMA1,
00327             .Stream = DMA1_Stream0, // DMA1_Stream2
00328             .channel = 0,
00329         },
00330
00331         .SPI3_TX = {
00332             .Controller = DMA1,
00333             .Stream = DMA1_Stream5, // DMA1_Stream7
00334             .channel = 0,
00335         },
00336
00337         .SPI2_RX = {
00338             .Controller = DMA1,
00339             .Stream = DMA1_Stream3,
00340             .channel = 0,
00341         },
00342
00343         .SPI2_TX = {
00344             .Controller = DMA1,
00345             .Stream = DMA1_Stream4,
00346             .channel = 0,
00347         },
00348
00349         .SPI1_RX = {
00350             .Controller = DMA2,
00351             .Stream = DMA2_Stream0, // DMA2_Stream2
00352             .channel = 3,

```

```
00353     },
00354
00355     .SPI1_TX = {
00356         .Controller = DMA2,
00357         .Stream = DMA2_Stream3, // DMA2_Stream5
00358         .channel = 3,
00359     },
00360
00361     .I2S2_RX = {
00362         .Controller = DMA1,
00363         .Stream = DMA1_Stream3,
00364         .channel = 3,
00365     },
00366
00367     .I2S2_TX = {
00368         .Controller = DMA1,
00369         .Stream = DMA1_Stream4,
00370         .channel = 3,
00371     },
00372
00373     .I2S3_RX = {
00374         .Controller = DMA1,
00375         .Stream = DMA1_Stream2,
00376         .channel = 2,
00377     },
00378
00379     .I2S3_TX = {
00380         .Controller = DMA1,
00381         .Stream = DMA1_Stream7,
00382         .channel = 0,
00383     },
00384
00385     .I2C1_RX = {
00386         .Controller = DMA1,
00387         .Stream = DMA1_Stream0, // DMA1_Stream5
00388         .channel = 1,
00389     },
00390
00391     .I2C1_TX = {
00392         .Controller = DMA1,
00393         .Stream = DMA1_Stream6, // DMA1_Stream7
00394         .channel = 1,
00395     },
00396
00397     .I2C2_RX = {
00398         .Controller = DMA1,
00399         .Stream = DMA1_Stream2,
00400         .channel = 7,
00401     },
00402
00403     .I2C2_TX = {
00404         .Controller = DMA1,
00405         .Stream = DMA1_Stream7,
00406         .channel = 7,
00407     },
00408
00409     .I2C3_RX = {
00410         .Controller = DMA1,
00411         .Stream = DMA1_Stream2,
00412         .channel = 3,
00413     },
00414
00415     .I2C3_TX = {
00416         .Controller = DMA1,
00417         .Stream = DMA1_Stream4,
00418         .channel = 3,
00419     },
00420
00421     .USART1_RX = {
00422         .Controller = DMA2,
00423         .Stream = DMA2_Stream2,
00424         .channel = 4,
00425     },
00426
00427     .USART1_TX = {
00428         .Controller = DMA2,
00429         .Stream = DMA2_Stream7,
00430         .channel = 4,
00431     },
00432
00433     .USART2_RX = {
00434         .Controller = DMA1,
00435         .Stream = DMA1_Stream5,
00436         .channel = 4,
00437     },
00438
00439     .USART2_TX = {
```

```
00440         .Controller = DMA1,
00441         .Stream = DMA1_Stream6,
00442         .channel = 4,
00443     },
00444
00445     .USART3_RX = {
00446         .Controller = DMA1,
00447         .Stream = DMA1_Stream1,
00448         .channel = 4,
00449     },
00450
00451     .USART3_TX = {
00452         .Controller = DMA1,
00453         .Stream = DMA1_Stream3,
00454         .channel = 4,
00455     },
00456
00457     .UART4_RX = {
00458         .Controller = DMA1,
00459         .Stream = DMA1_Stream2,
00460         .channel = 4,
00461     },
00462
00463     .UART4_TX = {
00464         .Controller = DMA1,
00465         .Stream = DMA1_Stream4,
00466         .channel = 4,
00467     },
00468
00469     .UART5_RX = {
00470         .Controller = DMA1,
00471         .Stream = DMA1_Stream0,
00472         .channel = 4,
00473     },
00474
00475     .UART5_TX = {
00476         .Controller = DMA1,
00477         .Stream = DMA1_Stream7,
00478         .channel = 4,
00479     },
00480
00481     .UART6_RX = {
00482         .Controller = DMA2,
00483         .Stream = DMA2_Stream1,
00484         .channel = 5,
00485     },
00486
00487     .UART6_TX = {
00488         .Controller = DMA2,
00489         .Stream = DMA2_Stream6,
00490         .channel = 5,
00491     },
00492
00493     .UART7_RX = {
00494         .Controller = DMA1,
00495         .Stream = DMA1_Stream3,
00496         .channel = 5,
00497     },
00498
00499     .UART7_TX = {
00500         .Controller = DMA1,
00501         .Stream = DMA1_Stream1,
00502         .channel = 5,
00503     },
00504
00505     .UART8_RX = {
00506         .Controller = DMA1,
00507         .Stream = DMA1_Stream6,
00508         .channel = 5,
00509     },
00510
00511     .UART8_TX = {
00512         .Controller = DMA1,
00513         .Stream = DMA1_Stream0,
00514         .channel = 5,
00515     },
00516
00517     .TIM1_UP = {
00518         .Controller = DMA2,
00519         .Stream = DMA2_Stream5,
00520         .channel = 6,
00521     },
00522
00523     .TIM2_UP = {
00524         .Controller = DMA1,
00525         .Stream = DMA1_Stream6,
00526         .channel = 3,
```

```
00527     },
00528
00529     .TIM3_CH1 = {
00530         .Controller = DMA1,
00531         .Stream = DMA1_Stream4,
00532         .channel = 5,
00533     },
00534
00535     .TIM3_CH2 = {
00536         .Controller = DMA1,
00537         .Stream = DMA1_Stream5,
00538         .channel = 5,
00539     },
00540
00541     .TIM3_CH3 = {
00542         .Controller = DMA1,
00543         .Stream = DMA1_Stream6,
00544         .channel = 5,
00545     },
00546
00547     .TIM3_CH4 = {
00548         .Controller = DMA1,
00549         .Stream = DMA1_Stream2,
00550         .channel = 5,
00551     },
00552
00553     .TIM4_CH1 = {
00554         .Controller = DMA1,
00555         .Stream = DMA1_Stream0,
00556         .channel = 2,
00557     },
00558
00559     .TIM4_CH2 = {
00560         .Controller = DMA1,
00561         .Stream = DMA1_Stream3,
00562         .channel = 2,
00563     },
00564
00565     .TIM4_CH3 = {
00566         .Controller = DMA1,
00567         .Stream = DMA1_Stream7,
00568         .channel = 2,
00569     },
00570
00571     .TIM4_UP = {
00572         .Controller = DMA1,
00573         .Stream = DMA1_Stream6,
00574         .channel = 2,
00575     },
00576
00577     .TIM5_CH1 = {
00578         .Controller = DMA1,
00579         .Stream = DMA1_Stream2,
00580         .channel = 6,
00581     },
00582
00583     .TIM5_CH2 = {
00584         .Controller = DMA1,
00585         .Stream = DMA1_Stream4,
00586         .channel = 6,
00587     },
00588
00589     .TIM5_CH3 = {
00590         .Controller = DMA1,
00591         .Stream = DMA1_Stream0,
00592         .channel = 6,
00593     },
00594
00595     .TIM5_UP = {
00596         .Controller = DMA1,
00597         .Stream = DMA1_Stream6,
00598         .channel = 6,
00599     },
00600
00601     .TIM6_UP = {
00602         .Controller = DMA1,
00603         .Stream = DMA1_Stream1,
00604         .channel = 7,
00605     },
00606
00607     .TIM7_UP = {
00608         .Controller = DMA1,
00609         .Stream = DMA1_Stream2,
00610         .channel = 1,
00611     },
00612
00613     ._DAC2 = {
```

```
00614         .Controller = DMA1,
00615         .Stream = DMA1_Stream5,
00616         .channel = 7,
00617     },
00618
00619     ._DAC2 = {
00620         .Controller = DMA1,
00621         .Stream = DMA1_Stream6,
00622         .channel = 7,
00623     },
00624
00625     .SDIO_RXTX = {
00626         .Controller = DMA2,
00627         .Stream = DMA2_Stream3,
00628         .channel = 4,
00629     },
00630
00631     ._DCMI = {
00632         .Controller = DMA2,
00633         .Stream = DMA2_Stream1,
00634         .channel = 1,
00635     },
00636
00637     ._ADC1 = {
00638         .Controller = DMA2,
00639         .Stream = DMA2_Stream0,
00640         .channel = 0,
00641     },
00642
00643     ._ADC2 = {
00644         .Controller = DMA2,
00645         .Stream = DMA2_Stream2,
00646         .channel = 1,
00647     },
00648
00649     ._ADC3 = {
00650         .Controller = DMA2,
00651         .Stream = DMA2_Stream1,
00652         .channel = 2,
00653     },
00654
00655     .TIM1_UP = {
00656         .Controller = DMA2,
00657         .Stream = DMA2_Stream5,
00658         .channel = 6,
00659     },
00660
00661     .TIM1_CH1 = {
00662         .Controller = DMA2,
00663         .Stream = DMA2_Stream1,
00664         .channel = 6,
00665     },
00666
00667     .TIM1_CH2 = {
00668         .Controller = DMA2,
00669         .Stream = DMA2_Stream6,
00670         .channel = 6,
00671     },
00672
00673     .TIM1_CH3 = {
00674         .Controller = DMA2,
00675         .Stream = DMA2_Stream6,
00676         .channel = 6,
00677     },
00678
00679     .TIM1_CH4 = {
00680         .Controller = DMA2,
00681         .Stream = DMA2_Stream4,
00682         .channel = 6,
00683     },
00684
00685     .TIM1_TRIG = {
00686         .Controller = DMA2,
00687         .Stream = DMA2_Stream0,
00688         .channel = 6,
00689     },
00690
00691     .TIM1_COM = {
00692         .Controller = DMA2,
00693         .Stream = DMA2_Stream0,
00694         .channel = 6,
00695     },
00696
00697     .TIM8_UP = {
00698         .Controller = DMA2,
00699         .Stream = DMA2_Stream1,
00700         .channel = 7,
```

```
00701     },
00702
00703     .TIM8_CH1 = {
00704         .Controller = DMA2,
00705         .Stream = DMA2_Stream2,
00706         .channel = 7,
00707     },
00708
00709     .TIM8_CH2 = {
00710         .Controller = DMA2,
00711         .Stream = DMA2_Stream4,
00712         .channel = 7,
00713     },
00714
00715     .TIM8_CH3 = {
00716         .Controller = DMA2,
00717         .Stream = DMA2_Stream6,
00718         .channel = 7,
00719     },
00720
00721     .TIM8_CH4 = {
00722         .Controller = DMA2,
00723         .Stream = DMA2_Stream7,
00724         .channel = 7,
00725     },
00726
00727     .TIM8_TRIG = {
00728         .Controller = DMA2,
00729         .Stream = DMA2_Stream1,
00730         .channel = 7,
00731     },
00732
00733     .TIM8_COM = {
00734         .Controller = DMA2,
00735         .Stream = DMA2_Stream1,
00736         .channel = 7,
00737     },
00738
00739     .TIM2_UP = {
00740         .Controller = DMA1,
00741         .Stream = DMA1_Stream6,
00742         .channel = 3,
00743     },
00744
00745     .TIM2_CH1 = {
00746         .Controller = DMA1,
00747         .Stream = DMA1_Stream5,
00748         .channel = 3,
00749     },
00750
00751     .TIM2_CH2 = {
00752         .Controller = DMA1,
00753         .Stream = DMA1_Stream7,
00754         .channel = 3,
00755     },
00756
00757     .TIM2_CH3 = {
00758         .Controller = DMA1,
00759         .Stream = DMA1_Stream1,
00760         .channel = 3,
00761     },
00762
00763     .TIM2_CH4 = {
00764         .Controller = DMA1,
00765         .Stream = DMA1_Stream7,
00766         .channel = 3,
00767     },
00768
00769     .TIM3_CH1 = {
00770         .Controller = DMA1,
00771         .Stream = DMA1_Stream4,
00772         .channel = 5,
00773     },
00774
00775     .TIM3_CH2 = {
00776         .Controller = DMA1,
00777         .Stream = DMA1_Stream5,
00778         .channel = 5,
00779     },
00780
00781     .TIM3_CH3 = {
00782         .Controller = DMA1,
00783         .Stream = DMA1_Stream6,
00784         .channel = 5,
00785     },
00786
00787     .TIM3_CH4 = {
```



```
00788         .Controller = DMA1,
00789         .Stream = DMA1_Stream2,
00790         .channel = 5,
00791     },
00792
00793     .TIM3_UP = {
00794         .Controller = DMA1,
00795         .Stream = DMA1_Stream2,
00796         .channel = 5,
00797     },
00798
00799     .TIM3_TRIG = {
00800         .Controller = DMA1,
00801         .Stream = DMA1_Stream5,
00802         .channel = 5,
00803     },
00804
00805     .TIM4_CH1 = {
00806         .Controller = DMA1,
00807         .Stream = DMA1_Stream0,
00808         .channel = 2,
00809     },
00810
00811     .TIM4_CH2 = {
00812         .Controller = DMA1,
00813         .Stream = DMA1_Stream3,
00814         .channel = 2,
00815     },
00816
00817     .TIM4_CH3 = {
00818         .Controller = DMA1,
00819         .Stream = DMA1_Stream7,
00820         .channel = 2,
00821     },
00822
00823     .TIM4_UP = {
00824         .Controller = DMA1,
00825         .Stream = DMA1_Stream6,
00826         .channel = 2,
00827     },
00828
00829     .TIM5_CH1 = {
00830         .Controller = DMA1,
00831         .Stream = DMA1_Stream2,
00832         .channel = 6,
00833     },
00834
00835     .TIM5_CH2 = {
00836         .Controller = DMA1,
00837         .Stream = DMA1_Stream4,
00838         .channel = 6,
00839     },
00840
00841     .TIM5_CH3 = {
00842         .Controller = DMA1,
00843         .Stream = DMA1_Stream0,
00844         .channel = 6,
00845     },
00846
00847     .TIM5_CH4 = {
00848         .Controller = DMA1,
00849         .Stream = DMA1_Stream6,
00850         .channel = 6,
00851     },
00852
00853     .TIM5_UP = {
00854         .Controller = DMA1,
00855         .Stream = DMA1_Stream6,
00856         .channel = 6,
00857     },
00858
00859     .TIM5_TRIG = {
00860         .Controller = DMA1,
00861         .Stream = DMA1_Stream3,
00862         .channel = 6,
00863     },
00864
00865 },
00866 };
00867
00868
00869 };
00870
00871
00872 #endif /* DMA_DEFS_H_ */
```



# Index

- [\\_ADC1](#)
    - [DMA\\_Configuration::Request, 27](#)
  - [\\_ADC2](#)
    - [DMA\\_Configuration::Request, 27](#)
  - [\\_ADC3](#)
    - [DMA\\_Configuration::Request, 27](#)
  - [\\_DAC1](#)
    - [DMA\\_Configuration::Request, 27](#)
  - [\\_DAC2](#)
    - [DMA\\_Configuration::Request, 27](#)
  - [\\_DCMI](#)
    - [DMA\\_Configuration::Request, 27](#)
- [buffer\\_length](#)
  - [DMA\\_Config, 14](#)
- [byte](#)
  - [DMA\\_Configuration::Memory\\_Data\\_Size, 21](#)
  - [DMA\\_Configuration::Peripheral\\_Data\\_Size, 22](#)
- [channel](#)
  - [DMA\\_Request, 19](#)
- [circular\\_mode](#)
  - [DMA\\_Config, 14](#)
- [Controller](#)
  - [DMA\\_Request, 19](#)
- [Direct\\_Mode\\_Error](#)
  - [DMA\\_Configuration::DMA\\_Interrupts, 18](#)
- [Direct\\_Mode\\_Error\\_Flag](#)
  - [DMA\\_Flags\\_Typedef, 16](#)
- [Disable](#)
  - [DMA\\_Configuration::Circular\\_Mode, 13](#)
  - [DMA\\_Configuration::DMA\\_Interrupts, 18](#)
  - [DMA\\_Configuration::Memory\\_Pointer\\_Increment, 22](#)
  - [DMA\\_Configuration::Peripheral\\_Pointer\\_Increment, 23](#)
- [DMA Driver Documentation, 1](#)
- [DMA.c, 39](#)
  - [DMA1\\_Stream0\\_IRQHandler, 41](#)
  - [DMA1\\_Stream1\\_IRQHandler, 41](#)
  - [DMA1\\_Stream2\\_IRQHandler, 41](#)
  - [DMA1\\_Stream3\\_IRQHandler, 42](#)
  - [DMA1\\_Stream4\\_IRQHandler, 42](#)
  - [DMA1\\_Stream5\\_IRQHandler, 42](#)
  - [DMA1\\_Stream6\\_IRQHandler, 42](#)
  - [DMA1\\_Stream7\\_IRQHandler, 43](#)
  - [DMA2\\_Stream0\\_IRQHandler, 43](#)
  - [DMA2\\_Stream1\\_IRQHandler, 43](#)
  - [DMA2\\_Stream2\\_IRQHandler, 43](#)
  - [DMA2\\_Stream3\\_IRQHandler, 44](#)
  - [DMA2\\_Stream4\\_IRQHandler, 44](#)
  - [DMA2\\_Stream5\\_IRQHandler, 44](#)
  - [DMA2\\_Stream6\\_IRQHandler, 44](#)
  - [DMA2\\_Stream7\\_IRQHandler, 45](#)
  - [DMA\\_Clock\\_Disable, 45](#)
  - [DMA\\_Clock\\_Enable, 45](#)
  - [DMA\\_Init, 46](#)
  - [DMA\\_Memory\\_To\\_Memory\\_Transfer, 46](#)
  - [DMA\\_Reset, 46](#)
  - [DMA\\_Reset\\_Flags, 47](#)
  - [DMA\\_Set\\_Target, 47](#)
  - [DMA\\_Set\\_Trigger, 47](#)
- [DMA.h, 48](#)
  - [DMA\\_Clock\\_Disable, 50](#)
  - [DMA\\_Clock\\_Enable, 51](#)
  - [DMA\\_Config, 49](#)
  - [DMA\\_Init, 51](#)
  - [DMA\\_Memory\\_To\\_Memory\\_Transfer, 52](#)
  - [DMA\\_Reset, 52](#)
  - [DMA\\_Reset\\_Flags, 53](#)
  - [DMA\\_Set\\_Target, 53](#)
  - [DMA\\_Set\\_Trigger, 54](#)
- [DMA1\\_Stream0\\_IRQHandler](#)
  - [DMA.c, 41](#)
- [DMA1\\_Stream1\\_IRQHandler](#)
  - [DMA.c, 41](#)
- [DMA1\\_Stream2\\_IRQHandler](#)
  - [DMA.c, 41](#)
- [DMA1\\_Stream3\\_IRQHandler](#)
  - [DMA.c, 42](#)
- [DMA1\\_Stream4\\_IRQHandler](#)
  - [DMA.c, 42](#)
- [DMA1\\_Stream5\\_IRQHandler](#)
  - [DMA.c, 42](#)
- [DMA1\\_Stream6\\_IRQHandler](#)
  - [DMA.c, 42](#)
- [DMA1\\_Stream7\\_IRQHandler](#)
  - [DMA.c, 43](#)
- [DMA2\\_Stream0\\_IRQHandler](#)
  - [DMA.c, 43](#)
- [DMA2\\_Stream1\\_IRQHandler](#)
  - [DMA.c, 43](#)
- [DMA2\\_Stream2\\_IRQHandler](#)
  - [DMA.c, 43](#)
- [DMA2\\_Stream3\\_IRQHandler](#)
  - [DMA.c, 44](#)
- [DMA2\\_Stream4\\_IRQHandler](#)
  - [DMA.c, 44](#)

DMA2\_Stream5\_IRQHandler  
     DMA.c, [44](#)  
 DMA2\_Stream6\_IRQHandler  
     DMA.c, [44](#)  
 DMA2\_Stream7\_IRQHandler  
     DMA.c, [45](#)  
 DMA\_Clock\_Disable  
     DMA.c, [45](#)  
     DMA.h, [50](#)  
 DMA\_Clock\_Enable  
     DMA.c, [45](#)  
     DMA.h, [51](#)  
 DMA\_Config, [14](#)  
     buffer\_length, [14](#)  
     circular\_mode, [14](#)  
     DMA.h, [49](#)  
     flow\_control, [14](#)  
     interrupts, [14](#)  
     memory\_address, [15](#)  
     memory\_data\_size, [15](#)  
     memory\_pointer\_increment, [15](#)  
     peripheral\_address, [15](#)  
     peripheral\_data\_size, [15](#)  
     peripheral\_pointer\_increment, [15](#)  
     priority\_level, [15](#)  
     Request, [15](#)  
     transfer\_direction, [16](#)  
 DMA\_Configuration::Circular\_Mode, [13](#)  
     Disable, [13](#)  
     Enable, [13](#)  
 DMA\_Configuration::DMA\_Interrupts, [17](#)  
     Direct\_Mode\_Error, [18](#)  
     Disable, [18](#)  
     Fifo\_Error, [18](#)  
     Half\_Transfer\_Complete, [18](#)  
     Transfer\_Complete, [18](#)  
     Transfer\_Error, [18](#)  
 DMA\_Configuration::Flow\_Control, [20](#)  
     DMA\_Control, [20](#)  
     Peripheral\_Control, [20](#)  
 DMA\_Configuration::Memory\_Data\_Size, [20](#)  
     byte, [21](#)  
     half\_word, [21](#)  
     word, [21](#)  
 DMA\_Configuration::Memory\_Pointer\_Increment, [21](#)  
     Disable, [22](#)  
     Enable, [22](#)  
 DMA\_Configuration::Peripheral\_Data\_Size, [22](#)  
     byte, [22](#)  
     half\_word, [22](#)  
     word, [23](#)  
 DMA\_Configuration::Peripheral\_Pointer\_Increment, [23](#)  
     Disable, [23](#)  
     Enable, [23](#)  
 DMA\_Configuration::Priority\_Level, [24](#)  
     High, [24](#)  
     Low, [24](#)  
     Medium, [24](#)  
     Very\_high, [25](#)  
 DMA\_Configuration::Request, [25](#)  
     \_ADC1, [27](#)  
     \_ADC2, [27](#)  
     \_ADC3, [27](#)  
     \_DAC1, [27](#)  
     \_DAC2, [27](#)  
     \_DCMI, [27](#)  
     I2C1\_RX, [27](#)  
     I2C1\_TX, [28](#)  
     I2C2\_RX, [28](#)  
     I2C2\_TX, [28](#)  
     I2C3\_RX, [28](#)  
     I2C3\_TX, [28](#)  
     I2S2\_RX, [28](#)  
     I2S2\_TX, [28](#)  
     I2S3\_RX, [28](#)  
     I2S3\_TX, [29](#)  
     SDIO\_RXTX, [29](#)  
     SPI1\_RX, [29](#)  
     SPI1\_TX, [29](#)  
     SPI2\_RX, [29](#)  
     SPI2\_TX, [29](#)  
     SPI3\_RX, [29](#)  
     SPI3\_TX, [29](#)  
     TIM1\_CH1, [30](#)  
     TIM1\_CH2, [30](#)  
     TIM1\_CH3, [30](#)  
     TIM1\_CH4, [30](#)  
     TIM1\_COM, [30](#)  
     TIM1\_TRIG, [30](#)  
     TIM1\_UP, [30](#)  
     TIM2\_CH1, [30](#)  
     TIM2\_CH2, [31](#)  
     TIM2\_CH3, [31](#)  
     TIM2\_CH4, [31](#)  
     TIM2\_UP, [31](#)  
     TIM3\_CH1, [31](#)  
     TIM3\_CH2, [31](#)  
     TIM3\_CH3, [31](#)  
     TIM3\_CH4, [31](#)  
     TIM3\_TRIG, [32](#)  
     TIM3\_UP, [32](#)  
     TIM4\_CH1, [32](#)  
     TIM4\_CH2, [32](#)  
     TIM4\_CH3, [32](#)  
     TIM4\_UP, [32](#)  
     TIM5\_CH1, [32](#)  
     TIM5\_CH2, [32](#)  
     TIM5\_CH3, [33](#)  
     TIM5\_CH4, [33](#)  
     TIM5\_TRIG, [33](#)  
     TIM5\_UP, [33](#)  
     TIM6\_UP, [33](#)  
     TIM7\_UP, [33](#)  
     TIM8\_CH1, [33](#)  
     TIM8\_CH2, [33](#)  
     TIM8\_CH3, [34](#)

- TIM8\_CH4, [34](#)
- TIM8\_COM, [34](#)
- TIM8\_TRIG, [34](#)
- TIM8\_UP, [34](#)
- UART4\_RX, [34](#)
- UART4\_TX, [34](#)
- UART5\_RX, [34](#)
- UART5\_TX, [35](#)
- UART6\_RX, [35](#)
- UART6\_TX, [35](#)
- UART7\_RX, [35](#)
- UART7\_TX, [35](#)
- UART8\_RX, [35](#)
- UART8\_TX, [35](#)
- USART1\_RX, [35](#)
- USART1\_TX, [36](#)
- USART2\_RX, [36](#)
- USART2\_TX, [36](#)
- USART3\_RX, [36](#)
- USART3\_TX, [36](#)
- DMA\_Configuration::Transfer\_Direction, [36](#)
  - Memory\_to\_memory, [37](#)
  - Memory\_to\_peripheral, [37](#)
  - Peripheral\_to\_memory, [37](#)
- DMA\_Control
  - DMA\_Configuration::Flow\_Control, [20](#)
- DMA\_Defs.h, [55](#)
  - DMA\_Flags\_Typedef, [56](#)
  - DMA\_Request, [56](#)
- DMA\_Flags, [11](#)
- DMA\_Flags\_Typedef, [16](#)
  - Direct\_Mode\_Error\_Flag, [16](#)
  - DMA\_Defs.h, [56](#)
  - Fifo\_Error\_Flag, [16](#)
  - Half\_Transfer\_Complete\_Flag, [17](#)
  - Transfer\_Complete\_Flag, [17](#)
  - Transfer\_Error\_Flag, [17](#)
- DMA\_Init
  - DMA.c, [46](#)
  - DMA.h, [51](#)
- DMA\_Memory\_To\_Memory\_Transfer
  - DMA.c, [46](#)
  - DMA.h, [52](#)
- DMA\_Request, [19](#)
  - channel, [19](#)
  - Controller, [19](#)
  - DMA\_Defs.h, [56](#)
  - Stream, [19](#)
- DMA\_Reset
  - DMA.c, [46](#)
  - DMA.h, [52](#)
- DMA\_Reset\_Flags
  - DMA.c, [47](#)
  - DMA.h, [53](#)
- DMA\_Set\_Target
  - DMA.c, [47](#)
  - DMA.h, [53](#)
- DMA\_Set\_Trigger
  - DMA.c, [47](#)
  - DMA.h, [54](#)
- Enable
  - DMA\_Configuration::Circular\_Mode, [13](#)
  - DMA\_Configuration::Memory\_Pointer\_Increment, [22](#)
  - DMA\_Configuration::Peripheral\_Pointer\_Increment, [23](#)
- Fifo\_Error
  - DMA\_Configuration::DMA\_Interrupts, [18](#)
- Fifo\_Error\_Flag
  - DMA\_Flags\_Typedef, [16](#)
- flow\_control
  - DMA\_Config, [14](#)
- Half\_Transfer\_Complete
  - DMA\_Configuration::DMA\_Interrupts, [18](#)
- Half\_Transfer\_Complete\_Flag
  - DMA\_Flags\_Typedef, [17](#)
- half\_word
  - DMA\_Configuration::Memory\_Data\_Size, [21](#)
  - DMA\_Configuration::Peripheral\_Data\_Size, [22](#)
- High
  - DMA\_Configuration::Priority\_Level, [24](#)
- I2C1\_RX
  - DMA\_Configuration::Request, [27](#)
- I2C1\_TX
  - DMA\_Configuration::Request, [28](#)
- I2C2\_RX
  - DMA\_Configuration::Request, [28](#)
- I2C2\_TX
  - DMA\_Configuration::Request, [28](#)
- I2C3\_RX
  - DMA\_Configuration::Request, [28](#)
- I2C3\_TX
  - DMA\_Configuration::Request, [28](#)
- I2S2\_RX
  - DMA\_Configuration::Request, [28](#)
- I2S2\_TX
  - DMA\_Configuration::Request, [28](#)
- I2S3\_RX
  - DMA\_Configuration::Request, [28](#)
- I2S3\_TX
  - DMA\_Configuration::Request, [29](#)
- interrupts
  - DMA\_Config, [14](#)
- Low
  - DMA\_Configuration::Priority\_Level, [24](#)
- Medium
  - DMA\_Configuration::Priority\_Level, [24](#)
- memory\_address
  - DMA\_Config, [15](#)
- memory\_data\_size
  - DMA\_Config, [15](#)
- memory\_pointer\_increment

- DMA\_Config, [15](#)
- Memory\_to\_memory
  - DMA\_Configuration::Transfer\_Direction, [37](#)
- Memory\_to\_peripheral
  - DMA\_Configuration::Transfer\_Direction, [37](#)
- peripheral\_address
  - DMA\_Config, [15](#)
- Peripheral\_Control
  - DMA\_Configuration::Flow\_Control, [20](#)
- peripheral\_data\_size
  - DMA\_Config, [15](#)
- peripheral\_pointer\_increment
  - DMA\_Config, [15](#)
- Peripheral\_to\_memory
  - DMA\_Configuration::Transfer\_Direction, [37](#)
- priority\_level
  - DMA\_Config, [15](#)
- Request
  - DMA\_Config, [15](#)
- SDIO\_RXTX
  - DMA\_Configuration::Request, [29](#)
- SPI1\_RX
  - DMA\_Configuration::Request, [29](#)
- SPI1\_TX
  - DMA\_Configuration::Request, [29](#)
- SPI2\_RX
  - DMA\_Configuration::Request, [29](#)
- SPI2\_TX
  - DMA\_Configuration::Request, [29](#)
- SPI3\_RX
  - DMA\_Configuration::Request, [29](#)
- SPI3\_TX
  - DMA\_Configuration::Request, [29](#)
- Stream
  - DMA\_Request, [19](#)
- TIM1\_CH1
  - DMA\_Configuration::Request, [30](#)
- TIM1\_CH2
  - DMA\_Configuration::Request, [30](#)
- TIM1\_CH3
  - DMA\_Configuration::Request, [30](#)
- TIM1\_CH4
  - DMA\_Configuration::Request, [30](#)
- TIM1\_COM
  - DMA\_Configuration::Request, [30](#)
- TIM1\_TRIG
  - DMA\_Configuration::Request, [30](#)
- TIM1\_UP
  - DMA\_Configuration::Request, [30](#)
- TIM2\_CH1
  - DMA\_Configuration::Request, [30](#)
- TIM2\_CH2
  - DMA\_Configuration::Request, [31](#)
- TIM2\_CH3
  - DMA\_Configuration::Request, [31](#)
- TIM2\_CH4
  - DMA\_Configuration::Request, [31](#)
- TIM2\_UP
  - DMA\_Configuration::Request, [31](#)
- TIM3\_CH1
  - DMA\_Configuration::Request, [31](#)
- TIM3\_CH2
  - DMA\_Configuration::Request, [31](#)
- TIM3\_CH3
  - DMA\_Configuration::Request, [31](#)
- TIM3\_CH4
  - DMA\_Configuration::Request, [31](#)
- TIM3\_TRIG
  - DMA\_Configuration::Request, [32](#)
- TIM3\_UP
  - DMA\_Configuration::Request, [32](#)
- TIM4\_CH1
  - DMA\_Configuration::Request, [32](#)
- TIM4\_CH2
  - DMA\_Configuration::Request, [32](#)
- TIM4\_CH3
  - DMA\_Configuration::Request, [32](#)
- TIM4\_UP
  - DMA\_Configuration::Request, [32](#)
- TIM5\_CH1
  - DMA\_Configuration::Request, [32](#)
- TIM5\_CH2
  - DMA\_Configuration::Request, [32](#)
- TIM5\_CH3
  - DMA\_Configuration::Request, [33](#)
- TIM5\_CH4
  - DMA\_Configuration::Request, [33](#)
- TIM5\_TRIG
  - DMA\_Configuration::Request, [33](#)
- TIM5\_UP
  - DMA\_Configuration::Request, [33](#)
- TIM6\_UP
  - DMA\_Configuration::Request, [33](#)
- TIM7\_UP
  - DMA\_Configuration::Request, [33](#)
- TIM8\_CH1
  - DMA\_Configuration::Request, [33](#)
- TIM8\_CH2
  - DMA\_Configuration::Request, [33](#)
- TIM8\_CH3
  - DMA\_Configuration::Request, [34](#)
- TIM8\_CH4
  - DMA\_Configuration::Request, [34](#)
- TIM8\_COM
  - DMA\_Configuration::Request, [34](#)
- TIM8\_TRIG
  - DMA\_Configuration::Request, [34](#)
- TIM8\_UP
  - DMA\_Configuration::Request, [34](#)
- Transfer\_Complete
  - DMA\_Configuration::DMA\_Interrupts, [18](#)
- Transfer\_Complete\_Flag
  - DMA\_Flags\_Typedef, [17](#)

- transfer\_direction
  - DMA\_Config, [16](#)
- Transfer\_Error
  - DMA\_Configuration::DMA\_Interrupts, [18](#)
- Transfer\_Error\_Flag
  - DMA\_Flags\_Typedef, [17](#)
- UART4\_RX
  - DMA\_Configuration::Request, [34](#)
- UART4\_TX
  - DMA\_Configuration::Request, [34](#)
- UART5\_RX
  - DMA\_Configuration::Request, [34](#)
- UART5\_TX
  - DMA\_Configuration::Request, [35](#)
- UART6\_RX
  - DMA\_Configuration::Request, [35](#)
- UART6\_TX
  - DMA\_Configuration::Request, [35](#)
- UART7\_RX
  - DMA\_Configuration::Request, [35](#)
- UART7\_TX
  - DMA\_Configuration::Request, [35](#)
- UART8\_RX
  - DMA\_Configuration::Request, [35](#)
- UART8\_TX
  - DMA\_Configuration::Request, [35](#)
- USART1\_RX
  - DMA\_Configuration::Request, [35](#)
- USART1\_TX
  - DMA\_Configuration::Request, [36](#)
- USART2\_RX
  - DMA\_Configuration::Request, [36](#)
- USART2\_TX
  - DMA\_Configuration::Request, [36](#)
- USART3\_RX
  - DMA\_Configuration::Request, [36](#)
- USART3\_TX
  - DMA\_Configuration::Request, [36](#)
- Very\_high
  - DMA\_Configuration::Priority\_Level, [25](#)
- word
  - DMA\_Configuration::Memory\_Data\_Size, [21](#)
  - DMA\_Configuration::Peripheral\_Data\_Size, [23](#)