DMA Driver

STM32F407DMA

Designed by Kunal Salvi

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`
: 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:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

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_RXTX
- 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	*config*	Pointer to the `DMA_Config` 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	*config*	Pointer to the DMA_Config 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	*config*	Pointer to the DMA_Config 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	*source*	Pointer to the source memory location.
in	*source_data_size*	Size of the data at the source (8, 16, or 32 bits).
in	*dest_data_size*	Size of the data at the destination (8, 16, or 32 bits).
in	*destination*	Pointer to the destination memory location.
in	*source_increment*	If true, the source address will be incremented after each transfer.
in	*destination_increment*	If true, the destination address will be incremented after each transfer.
in	*length*	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`	*config*	Pointer to the `DMA_Config` 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

flag	The `DMA_Flags_Typedef` 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`	*config*	Pointer to the `DMA_Config` 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 `DMA_Config` 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	*config*	Pointer to the DMA_Config 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	*config*	Pointer to the `DMA_Config` 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	*config*	Pointer to the DMA_Config 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	*config*	Pointer to the `DMA_Config` 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	*config*	Pointer to the DMA_Config 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	*config*	Pointer to the DMA_Config 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	*source*	Pointer to the source memory location.
in	*source_data_size*	Size of the data at the source (8, 16, or 32 bits).
in	*dest_data_size*	Size of the data at the destination (8, 16, or 32 bits).
in	*destination*	Pointer to the destination memory location.
in	*source_increment*	If true, the source address will be incremented after each transfer.
in	*destination_increment*	If true, the destination address will be incremented after each transfer.
in	*length*	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	*source*	Pointer to the source memory location.
in	*source_data_size*	Size of the data at the source (8, 16, or 32 bits).
in	*dest_data_size*	Size of the data at the destination (8, 16, or 32 bits).
in	*destination*	Pointer to the destination memory location.
in	*source_increment*	If true, the source address will be incremented after each transfer.
in	*destination_increment*	If true, the destination address will be incremented after each transfer.
in	*length*	Number of data items to transfer.

7.2.3.5 DMA_Reset()

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

Resets the specified DMA controller.

Parameters

in	*config*	Pointer to the DMA_Config 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

in	*config*	Pointer to the `DMA_Config` 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

flag	The DMA_Flags_Typedef 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

flag	The `DMA_Flags_Typedef` 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

in	*config*	Pointer to the DMA_Config 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	*config*	Pointer to the DMA_Config 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	*config*	Pointer to the DMA_Config 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	*config*	Pointer to the DMA_Config structure that contains the configuration settings.

7.3 DMA.h

Go to the documentation of this file.

```

#ifndef DMA_H_
#define DMA_H_

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

extern DMA_Flags_Typedef DMA1_Stream0_Flag;
extern DMA_Flags_Typedef DMA1_Stream1_Flag;
extern DMA_Flags_Typedef DMA1_Stream2_Flag;
extern DMA_Flags_Typedef DMA1_Stream3_Flag;
extern DMA_Flags_Typedef DMA1_Stream4_Flag;
extern DMA_Flags_Typedef DMA1_Stream5_Flag;
extern DMA_Flags_Typedef DMA1_Stream6_Flag;
extern DMA_Flags_Typedef DMA1_Stream7_Flag;

extern DMA_Flags_Typedef DMA2_Stream0_Flag;
extern DMA_Flags_Typedef DMA2_Stream1_Flag;
extern DMA_Flags_Typedef DMA2_Stream2_Flag;
extern DMA_Flags_Typedef DMA2_Stream3_Flag;
extern DMA_Flags_Typedef DMA2_Stream4_Flag;
extern DMA_Flags_Typedef DMA2_Stream5_Flag;
extern DMA_Flags_Typedef DMA2_Stream6_Flag;
extern DMA_Flags_Typedef DMA2_Stream7_Flag;
typedef struct DMA_Config
{
    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;
} DMA_Config;

void DMA_Clock_Enable(DMA_Config *config);

void DMA_Clock_Disable(DMA_Config *config);

void DMA_Reset(DMA_Config *config);

void DMA_Reset_Flags(DMA_Flags_Typedef flag);

int8_t DMA_Init(DMA_Config *config);

void DMA_Set_Target(DMA_Config *config);

void DMA_Set_Trigger(DMA_Config *config);

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);

#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.

```
#ifndef DMA_DEFS_H_
#define DMA_DEFS_H_

#include "main.h"

typedef struct DMA_Flags_Typedef
{
    bool Transfer_Complete_Flag;
    bool Half_Transfer_Complete_Flag;
    bool Transfer_Error_Flag;
    bool Direct_Mode_Error_Flag;
    bool Fifo_Error_Flag;
}DMA_Flags_Typedef;

typedef struct DMA_Request {
    DMA_TypeDef *Controller;
    DMA_Stream_TypeDef *Stream;
    uint8_t channel;
} DMA_Request;


static const struct DMA_Configuration {

    struct Request{

    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_RXTX;
    DMA_Request _DCMI;
    DMA_Request _ADC1;
    DMA_Request _ADC2;
    DMA_Request _ADC3;
    }Request;

    struct Flow_Control
    {
        uint32_t DMA_Control;
        uint32_t Peripheral_Control;
    }Flow_Control;

    struct Transfer_Direction {
        uint32_t Peripheral_to_memory;
        uint32_t Memory_to_peripheral;
        uint32_t Memory_to_memory;
    } Transfer_Direction;

    struct Priority_Level {
        uint32_t Low;
        uint32_t Medium;
        uint32_t High;
        uint32_t Very_high;
    } Priority_Level;

    struct Peripheral_Data_Size
    {
        uint32_t byte;
        uint32_t half_word;
        uint32_t word;
    }Peripheral_Data_Size;

    struct Memory_Data_Size
    {
        uint32_t byte;
        uint32_t half_word;
        uint32_t word;
    }Memory_Data_Size;

    struct Circular_Mode
    {
        uint32_t Enable;
        uint32_t Disable;
    }Circular_Mode;

    struct DMA_Interrupts
    {
        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;
    }DMA_Interrupts;

    struct Memory_Pointer_Increment
    {
        uint32_t Enable;
        uint32_t Disable;
    }Memory_Pointer_Increment;

    struct Peripheral_Pointer_Increment {
        uint32_t Enable;
        uint32_t Disable;
    } Peripheral_Pointer_Increment;


} DMA_Configuration = {

        .Circular_Mode = {

                .Enable = 1 « DMA_SxCR_CIRC_Pos,
                .Disable = 0 « DMA_SxCR_CIRC_Pos,
            },

        .Memory_Pointer_Increment = {
                .Enable = 1 « 10,
                .Disable = 0 « 10,
        },

        .Peripheral_Pointer_Increment = {
                .Enable = 1 « 9,
                .Disable = 0 « 9,
        },

        .DMA_Interrupts = {
                .Transfer_Complete = 1 « 4,
                .Half_Transfer_Complete = 1 « 3,
                .Transfer_Error = 1 « 2,
                .Direct_Mode_Error = 1 « 1,
                .Fifo_Error = 1 « 7,
                .Disable = 0 « 1,
        },

        .Flow_Control = {
                .DMA_Control = 0 « 5,
                .Peripheral_Control = 1 « 5,
        },

        .Transfer_Direction =
        {
            .Peripheral_to_memory = 0 « 6,
            .Memory_to_peripheral = 1 « 6,
            .Memory_to_memory = 2 « 6,
        },

        .Priority_Level =
        {
            .Low = 0 « 16,
            .Medium = 1 « 16,
            .High = 2 « 16,
            .Very_high = 3 « 16,
        },

        .Memory_Data_Size = {

                .byte = 0 « 13,
                .half_word = 1 « 13,
                .word = 2 « 13,

        },

        .Peripheral_Data_Size = {
                .byte = 0 « 11,
                .half_word = 1 « 11,
                .word = 2 « 11,
        },

        .Request = {

                .SPI3_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream0, // DMA1_Stream2
                        .channel = 0,
                },

                .SPI3_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream5, // DMA1_Stream7
                        .channel = 0,
                },

                .SPI2_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream3,
                        .channel = 0,
                },

                .SPI2_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream4,
                        .channel = 0,
                },

                .SPI1_RX = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream0, // DMA2_Stream2
                        .channel = 3,
                },

                .SPI1_TX = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream3, // DMA2_Stream5
                        .channel = 3,
                },

                .I2S2_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream3,
                        .channel = 3,
                },

                .I2S2_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream4,
                        .channel = 3,
                },

                .I2S3_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream2,
                        .channel = 2,
                },

                .I2S3_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream7,
                        .channel = 0,
                },

                .I2C1_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream0, // DMA1_Stream5
                        .channel = 1,
                },

                .I2C1_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6, // DMA1_Stream7
                        .channel = 1,
                },

                .I2C2_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream2,
                        .channel = 7,
                },

                .I2C2_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream7,
                        .channel = 7,
                },

                .I2C3_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream2,
                        .channel = 3,
                },

                .I2C3_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream4,
                        .channel = 3,
                },

                .USART1_RX = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream2,
                        .channel = 4,
                },

                .USART1_TX = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream7,
                        .channel = 4,
                },

                .USART2_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream5,
                        .channel = 4,
                },

                .USART2_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 4,
                },

                .USART3_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream1,
                        .channel = 4,
                },

                .USART3_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream3,
                        .channel = 4,
                },

                .UART4_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream2,
                        .channel = 4,
                },

                .UART4_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream4,
                        .channel = 4,
                },

                .UART5_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream0,
                        .channel = 4,
                },

                .UART5_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream7,
                        .channel = 4,
                },

                .UART6_RX = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream1,
                        .channel = 5,
                },

                .UART6_TX = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream6,
                        .channel = 5,
                },

                .UART7_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream3,
                        .channel = 5,
                },

                .UART7_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream1,
                        .channel = 5,
                },

                .UART8_RX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 5,
                },

                .UART8_TX = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream0,
                        .channel = 5,
                },

                .TIM1_UP = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream5,
                        .channel = 6,
                },

                .TIM2_UP = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 3,
		},

		.TIM3_CH1 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream4,
			.channel = 5,
		},

		.TIM3_CH2 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream5,
			.channel = 5,
		},

		.TIM3_CH3 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream6,
			.channel = 5,
		},

		.TIM3_CH4 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream2,
			.channel = 5,
		},

		.TIM4_CH1 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream0,
			.channel = 2,
		},

		.TIM4_CH2 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream3,
			.channel = 2,
		},

		.TIM4_CH3 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream7,
			.channel = 2,
		},

		.TIM4_UP = {
			.Controller = DMA1,
			.Stream = DMA1_Stream6,
			.channel = 2,
		},

		.TIM5_CH1 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream2,
			.channel = 6,
		},

		.TIM5_CH2 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream4,
			.channel = 6,
		},

		.TIM5_CH3 = {
			.Controller = DMA1,
			.Stream = DMA1_Stream0,
			.channel = 6,
		},

		.TIM5_UP = {
			.Controller = DMA1,
			.Stream = DMA1_Stream6,
			.channel = 6,
		},

		.TIM6_UP = {
			.Controller = DMA1,
			.Stream = DMA1_Stream1,
			.channel = 7,
		},

		.TIM7_UP = {
			.Controller = DMA1,
			.Stream = DMA1_Stream2,
			.channel = 1,
		},

		._DAC2 = {
```

```
                        .Controller = DMA1,
                        .Stream = DMA1_Stream5,
                        .channel = 7,
                },

                ._DAC2 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 7,
                },

                .SDIO_RXTX = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream3,
                        .channel = 4,
                },

                ._DCMI = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream1,
                        .channel = 1,
                },

                ._ADC1 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream0,
                        .channel = 0,
                },

                ._ADC2 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream2,
                        .channel = 1,
                },

                ._ADC3 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream1,
                        .channel = 2,
                },

                .TIM1_UP = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream5,
                        .channel = 6,
                },

                .TIM1_CH1 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream1,
                        .channel = 6,
                },

                .TIM1_CH2 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream6,
                        .channel = 6,
                },

                .TIM1_CH3 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream6,
                        .channel = 6,
                },

                .TIM1_CH4 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream4,
                        .channel = 6,
                },

                .TIM1_TRIG = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream0,
                        .channel = 6,
                },

                .TIM1_COM = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream0,
                        .channel = 6,
                },

                .TIM8_UP = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream1,
                        .channel = 7,
                },

                .TIM8_CH1 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream2,
                        .channel = 7,
                },

                .TIM8_CH2 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream4,
                        .channel = 7,
                },

                .TIM8_CH3 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream6,
                        .channel = 7,
                },

                .TIM8_CH4 = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream7,
                        .channel = 7,
                },

                .TIM8_TRIG = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream1,
                        .channel = 7,
                },

                .TIM8_COM = {
                        .Controller = DMA2,
                        .Stream = DMA2_Stream1,
                        .channel = 7,
                },

                .TIM2_UP = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 3,
                },

                .TIM2_CH1 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream5,
                        .channel = 3,
                },

                .TIM2_CH2 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream7,
                        .channel = 3,
                },

                .TIM2_CH3 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream1,
                        .channel = 3,
                },

                .TIM2_CH4 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream7,
                        .channel = 3,
                },

                .TIM3_CH1 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream4,
                        .channel = 5,
                },

                .TIM3_CH2 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream5,
                        .channel = 5,
                },

                .TIM3_CH3 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 5,
                },

                .TIM3_CH4 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream2,
                        .channel = 5,
                },

                .TIM3_UP = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream2,
                        .channel = 5,
                },

                .TIM3_TRIG = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream5,
                        .channel = 5,
                },

                .TIM4_CH1 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream0,
                        .channel = 2,
                },

                .TIM4_CH2 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream3,
                        .channel = 2,
                },

                .TIM4_CH3 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream7,
                        .channel = 2,
                },

                .TIM4_UP = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 2,
                },

                .TIM5_CH1 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream2,
                        .channel = 6,
                },

                .TIM5_CH2 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream4,
                        .channel = 6,
                },

                .TIM5_CH3 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream0,
                        .channel = 6,
                },

                .TIM5_CH4 = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 6,
                },

                .TIM5_UP = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream6,
                        .channel = 6,
                },

                .TIM5_TRIG = {
                        .Controller = DMA1,
                        .Stream = DMA1_Stream3,
                        .channel = 6,
                },


        },


};


#endif /* DMA_DEFS_H_ */
```

Index