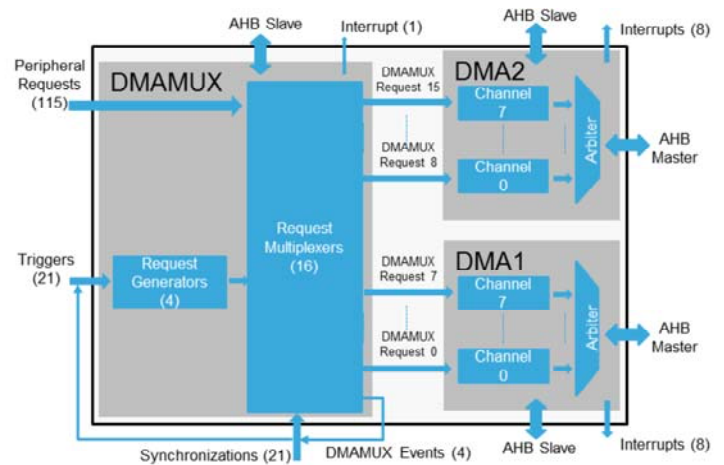


DMA request multiplexer (DMAMUX)



1

- 2x DMA controllers with each one able to perform: :
 - Programmable block transfers with 8 concurrent channels, independently configurable
 - Programmable channel-based priority
 - Data transfers via the AHB master port
- 1x DMA request router (DMAMUX)
 - Programmable request source selection
 - from a peripheral in DMA mode or
 - from a trigger and internally generated
 - Synchronization mode: from a synchronization input with a request counter
 - Requests chaining: with the request counter to generate an event that is an input trigger/synchronization to another request/channel



Application benefit

Off-load CPU for data transfers from a memory-mapped source to a memory-mapped destination

The main application benefit of the DMA is to off-load the CPU for data transfers, from any memory-mapped source towards any memory-mapped destination.

STM32G4 DMA features:

- 2 DMA controllers. For each DMA controller it is possible to do:
 - ❖ Programmable block transfers with 8 concurrent channels, (each of which is independently configurable)
 - ❖ Programmable channel-based priorities
 - ❖ Data transfers via the AHB master port (connected to the bus matrix)
- There is also a DMA request router (DMAMUX), with:
 - ❖ Programmable request source selection : either from a peripheral in DMA mode or from a trigger and then internally generated
 - ❖ Synchronization mode: from a synchronization

input (hardware event) with a DMAMUX request counter

- ❖ Requests chaining: with the DMAMUX request counter to generate an event that is an input trigger or synchronization to another request/channel

STM32G4 DMA & DMAMUX instance

3

DMAMUX features	DMAMUX
Number of peripheral requests	115
Number of request generators	4
Number of triggers	21
Number of synchronizations	21
Number of output DMA requests	16

DMA features	DMA1
Number of channels	8

DMA features	DMA2
Number of channels	8



There are 115 peripheral requests and 4 DMAMUX request generators

There are 21 triggers and synchronization inputs.

There are 16 DMA channels/requests.

- Independently configurable channels over one DMA controller
 - A channel can be assigned to a DMA hardware request from a peripheral, in memory-to-peripheral or peripheral-to-memory transfers
 - Alternatively, a channel is assigned to a software request in memory-to-memory transfers
 - A channel is programmed with a priority level
 - A channel is programmed for a number of data transfers at a block level
- Software controls a channel via separated interrupts and/or flags upon programmable events such as block transfer complete, and/or a half-block transfer complete, and/or a transfer error.
 - A faulty channel is automatically disabled in case of a bus access error.



Let's focus on the DMA controller.

Each channel of the DMA controller is independently configurable:

- A channel can be assigned to a DMA hardware request from a peripheral in peripheral-to-memory, or memory-to-peripheral data transfers.
- Alternatively, a channel is assigned to a software request in memory-to-memory data transfers.
- A channel is programmed with a priority level.
- A channel is programmed for a number of data transfers at a block level.

The software can control a channel via the separated interrupts and/or flags upon programmable events such as a block transfer complete, and/or a half-block transfer complete, and/or a transfer error.

A faulty channel is automatically disabled in case of a bus access error.

- Programmable data transfers at a block level
 - Independent source and destination data size: 8-bit, 16-bit, or 32-bit
 - Independent source and destination start address
 - Independent source and destination address increment: contiguously incremented or at fixed address
 - Programmable amount of data to be transferred within a block
 - Up to 65,535 source data
 - Automatically decremented by hardware
- In Circular Buffer mode (continuous data transfer from/to a peripheral), when a block transfer is completed :
 - The following programmed information is automatically reloaded by hardware:
 - Amount of data to be transferred within a block
 - Source and destination start addresses



A channel is programmed for a number of data transfers at a block level with:

- Independent source and destination data size
- Independent source and destination start address
- Independent source and destination address increment (either contiguously incremented or at a fixed address)
- Programmable amount of data to be transferred within a block
 - ❖ Up to 65,535 source data
 - ❖ Automatically decremented by hardware

In Circular Buffer mode (continuous data transfer from/to a peripheral), when a block transfer is completed:

- The programmed amount of data to be transferred within a block is automatically reloaded by hardware
- As well as the source and destination start addresses

- Memory-to-memory mode
 - A block transfer starts as soon as the channel is enabled in this mode (no hardware request)
- Peripheral-to-memory, and memory-to-peripheral modes:
 - A block transfer starts as soon as both 1) the channel is enabled and 2) the peripheral sends a DMA hardware request
 - A DMA hardware request identifies a (single) DMA data transfer
 - Each DMA hardware request is paced and granted by the DMA when each data is successfully transferred to the destination
- In any mode
 - Channel arbitration is reassessed between every data transfer



In Memory-to-memory mode, a block transfer starts as soon as the channel is enabled (there is no hardware request).

Whereas in peripheral-to-memory, and memory-to-peripheral modes:

- A block transfer starts as soon as both 1) the channel is enabled and 2) the peripheral sends a DMA hardware request
- A DMA hardware request identifies a (single) DMA data transfer
- Each DMA hardware request is paced and granted by the DMA when each data is successfully transferred to the destination

In any mode, channel arbitration is reassessed between every data transfer.

- Interrupt events for each channel

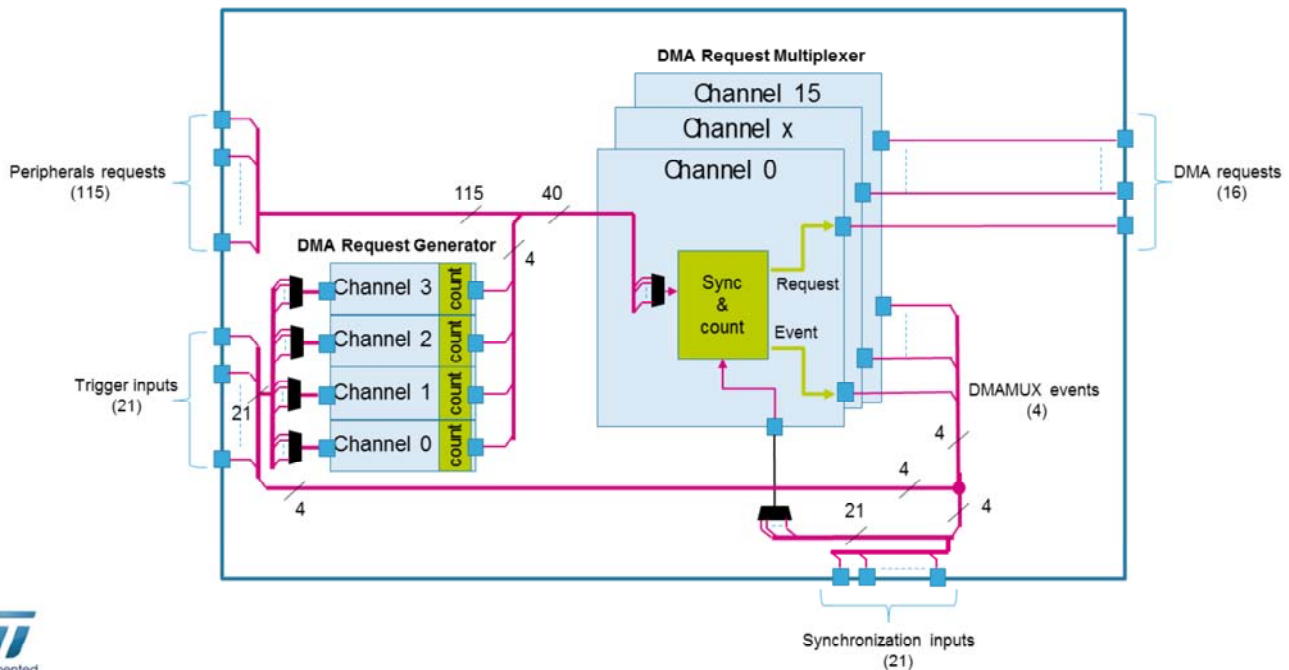
Interrupt event	Description
Half transfer	Set when half of the block of data has been transferred
Transfer complete	Set when the block transfer is completed
Transfer error	Set when an error occurs during a data transfer
Global interrupt	Set whenever a half transfer, a transfer complete or a transfer error event occurs

Each DMA channel can notify software with an interrupt triggered by any of 4 possible events:

- Half-block transfer completion
- Block transfer completion
- Transfer error
- Any of the 3 above events (i.e. global)

DMAMUX Block Diagram

8



The DMAMUX has two main sub-blocks: the request multiplexer and the request generator.

The DMAMUX request multiplexer enables routing a DMA request from the peripherals to the DMA controllers.

The routing function is ensured by the programmable multi-channel DMA request multiplexer.

Each channel selects a unique DMA request, unconditionally or synchronously with events, from its DMAMUX synchronization inputs.

The DMAMUX may also be used as a DMA request generator from programmable events on its input trigger signals.

A DMA request multiplexer channel generates both a request to the DMA controller and an event that can be used as a synchronization input as well as a trigger input.

Do not confuse DMA request generator channels (0 to 3) with DMA request multiplexer channels (0 to 15).

DMAMUX Request Multiplexer (1/6)

9

- For each multiplexer channel
 - Configuration register: DMAMUX_CxCR
 - Programmable input request selection (via DMAREQ_ID field)
 - For each request from a peripheral working in DMA mode, a DMAREQ_ID is assigned
 - DMAREQ_ID = 0 corresponds to when no DMA request is selected
 - After having configured this channel and the DMA controller channel to which it is routed, the DMA channel can be enabled
 - It is **NOT** allowed to configure two different channels with a same DMA request input



For each multiplexer channel, there is a configuration register: DMAMUX_CxCR with

- Programmable input request selection (via DMAREQ_ID field).
 - ❖ For each request from a peripheral working in DMA mode, a DMAREQ_ID is assigned.
 - ❖ DMAREQ_ID = 0x00 corresponds to no DMA request selected.
- After having configured this channel and the DMA controller channel to which it is routed, the DMA channel can be enabled. It is NOT allowed to configure two different channels with the same DMA request input.

DMAMUX Request Multiplexer (2/6)

10

- For each multiplexer channel
 - A built-in DMA request counter
 - Programmable (via NBREQ field)
 - A served DMA request decrements the programmed DMA request counter. At its underrun:
 - DMA request counter is automatically reloaded with the programmed value in NBREQ field
 - At its underrun, can be generated a DMAMUX event
 - If enabled (via EGE field)
 - 4 DMAMUX events (from Channels 0 to 3) are looped back and connected to the DMAMUX as trigger inputs and synchronization inputs.
 - This allows request chaining for another different DMA channel, via synchronization and/or trigger



For each multiplexer channel:

- A built-in DMA request counter is programmable (via the NBREQ field).
- A served DMA request decrements the programmed DMA request counter. At its underrun, the DMA request counter is automatically reloaded with the programmed value in NBREQ field.
- At its underrun, a DMAMUX event can be generated if enabled (via EGE field).
- 4 DMAMUX events (from Channels 0 to 3) are looped back and connected to the DMAMUX as trigger inputs and synchronization inputs. This allows request chaining for another different DMA channel, via synchronization and/or trigger.

DMAMUX Request Multiplexer (3/6)

11

- For each multiplexer channel
 - 2 operating modes (as programmed via the SE field)
 - Unconditional mode: Input request is output as is
 - Synchronized mode: A number of requests is grouped and delayed/synchronized



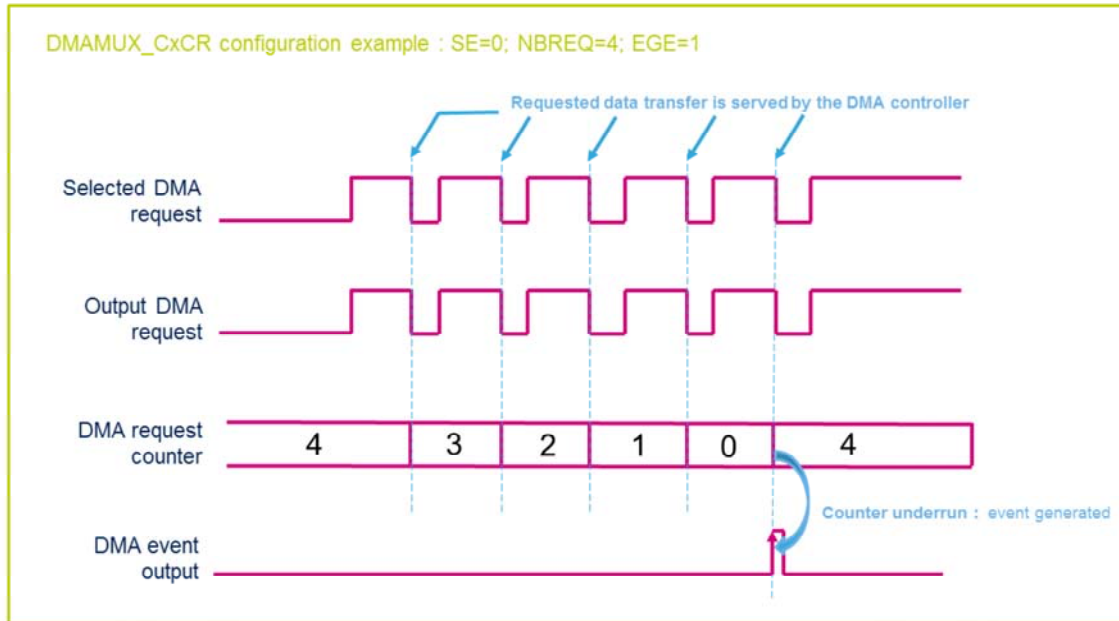
For each multiplexer channel, there are 2 operating modes (as programmed via the SE field):

- Unconditional mode: Input request is output as is.
- Synchronized mode: A number of requests is grouped and delayed/synchronized.

DMAMUX Request Multiplexer (4/6)

Unconditional mode

12



When the request multiplexer channel is configured unconditionally (SE=0), the DMA request is transmitted as is and as paced by the DMA controller.

When the DMA controller has served a data transfer, the DMA request is de-asserted and the built-in DMA request counter is decremented.

At the counter underrun, if enabled via the EGE field, an event can be generated.

DMAMUX Request Multiplexer (5/6)

Synchronous mode

13

- In Synchronous mode, additionally
 - The request is conditioned with
 - Programmable synchronization input selection (via SYNC_ID field)
 - Programmable synchronization event : none/rising/falling/either edge (via SPOL field)
 - The single built-in request counter (via NBREQ field) that may be also used for event generation
 - After the synchronization event
 - Output DMA request is connected to the pending input request
 - At the counter underrun
 - DMA output request is disconnected from the multiplexer channel input
 - A synchronization overrun flag (SOFx in DMAMUX_CSR) is reported
 - If a new synchronization event occurs before the counter underrun
 - An interrupt is then generated if enabled (via SOIE field)



In Synchronous mode, additionally:

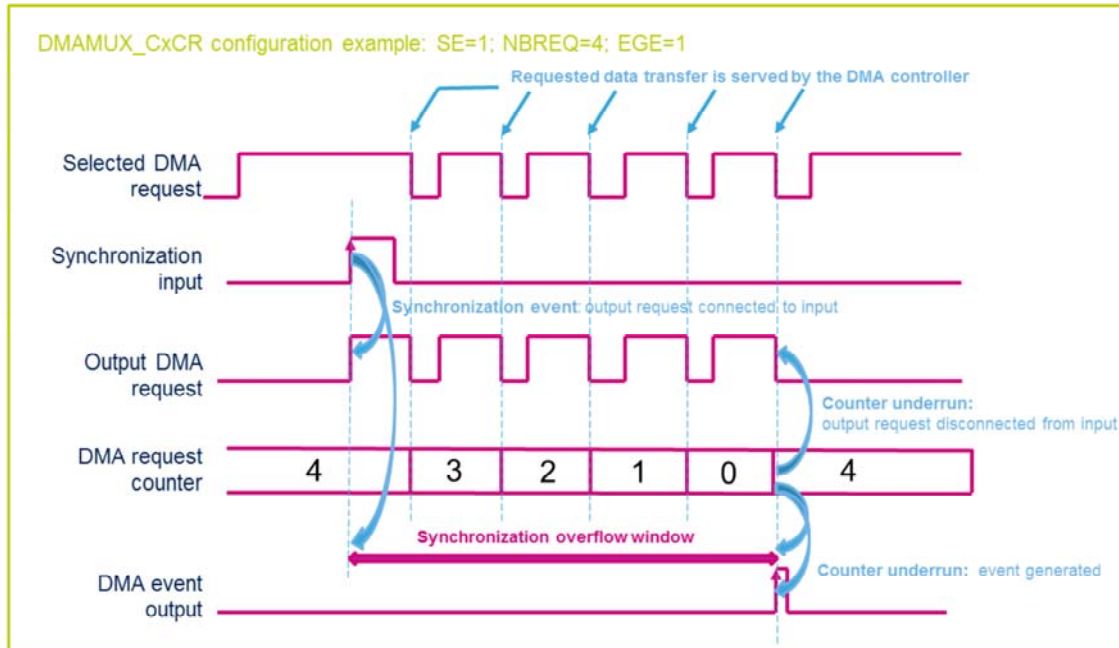
- The request is conditioned with:
 - ❖ A Programmable synchronization input selection (via SYNC_ID field)
 - ❖ A Programmable synchronization event: none/rising/falling/either edge (via SPOL field)
 - ❖ The single built-in request counter (via NBREQ field) that may be also used for event generation
- After the synchronization event:
 - ❖ Output DMA request is connected to the pending input request
- At the counter underrun :
 - ❖ DMA output request is disconnected from the multiplexer channel input
- Finally, a synchronization overrun flag (SOFx in DMAMUX_CSR) is reported:
 - ❖ If a new synchronization event occurs before the counter underrun

- ❖ An interrupt is then generated if enabled (via SOIE field)

DMAMUX Request Multiplexer (6/6)

Synchronous mode

14



When the DMAMUX channel is configured in Synchronous mode, its behavior is as follows.

The request multiplexer input (DMA request from the peripheral) can be pending, and it will not be forwarded on the DMAMUX request multiplexer output until the synchronization event is received.

Then, the request multiplexer connects its input and output and all the peripheral requests will be forwarded.

Each forwarded and granted DMA request decrements the request multiplexer counter (set at a defined programmed value).

When the counter reaches zero, the connection between the DMA controller and the peripheral is cut, waiting for a new synchronization event.

For each underrun of the counter, a request multiplexer can generate an optional event to synchronize and/or trigger a second DMAMUX request multiplexer channel.

The same event can be used in some low-power scenarios

to switch the system back to Stop mode without CPU intervention.

Synchronization mode can be used to automatically synchronize data transfers with a timer for example, or to condition transfers from any peripheral event that is mapped as a synchronization input.

Additionally, a synchronization overflow can notify the software if a programmed number of DMA requests has not been completed between two synchronization events.

- For each channel
 - A DMA request can be generated, following a trigger event. And selected as input of a DMAMUX request multiplexer channel (via DMAREQ_ID field of the DMAMUX_CxCR)
 - The request is generated, via the configuration register DMAMUX_RGxCR, if enabled (by GE field), with
 - Programmable trigger input selection (via SIG_ID field)
 - Programmable trigger event : none/rising/falling/either edge (via GPOL field)
 - A built-in request counter (via GNBREQ field)
 - A served DMA request decrements the programmed request counter
 - At its underrun:
 - Request counter is automatically reloaded with the programmed value in GNBREQ field
 - Request generator stops generating a request
 - A trigger overrun flag (OFx in DMAMUX_RGSR) is reported
 - If a new trigger event occurs before the counter underrun
 - An interrupt is then generated if enabled (via OIE field)



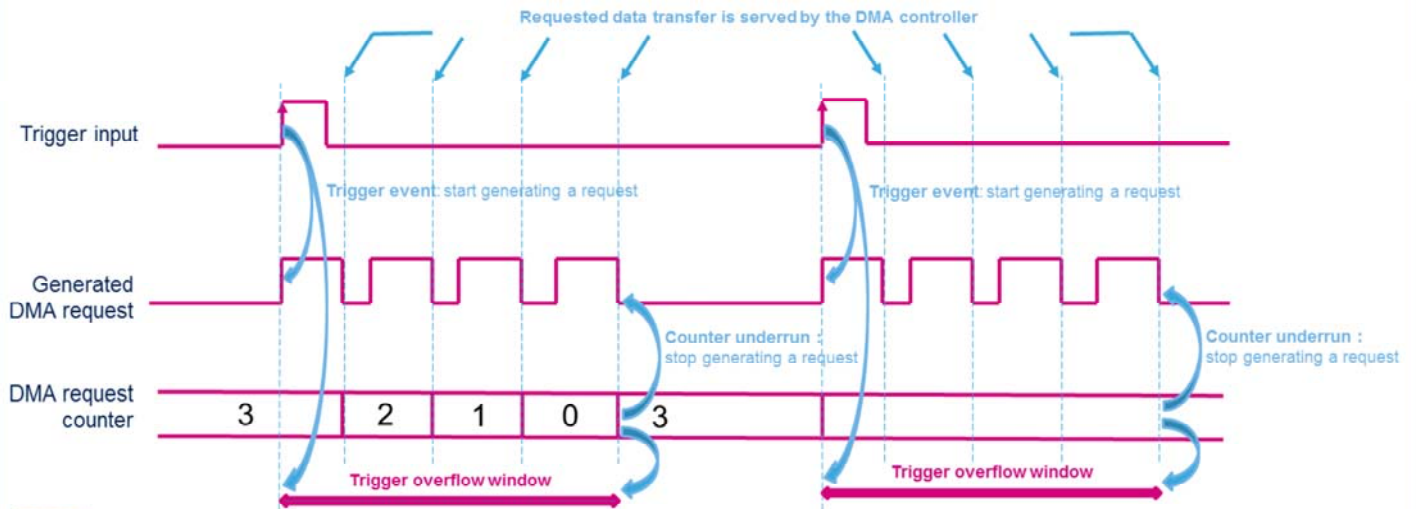
For each request generator channel:

- A DMA request can be generated, following a trigger event. And selected as input of a DMAMUX request multiplexer channel (via DMAREQ_ID field of the DMAMUX_CxCR)
- The request is generated, via the configuration register DMAMUX_RGxCR, if enabled (by GE field), with:
 - ❖ Programmable trigger input selection (via SIG_ID field)
 - ❖ Programmable trigger event : none/rising/falling/either edge (via GPOL field)
 - ❖ A built-in request counter (via GNBREQ field)
- A served DMA request decrements the programmed request counter
- At its underrun:
 - ❖ Request counter is automatically reloaded with the programmed value in GNBREQ field
 - ❖ Request generator stops generating a request

- A trigger overrun flag (OFx in DMAMUX_RGSR) is reported:
 - ❖ If a new trigger event occurs before the counter underrun
 - ❖ An interrupt is then generated if enabled (via OIE field)

DMAMUX Request Generator 16

DMAMUX_RGxCR configuration example : GE=0; GPOL=1; GNBREQ=3



On a trigger event, a programmed number of DMA requests (GNBREQ+1) is generated.

There may be a trigger overflow if two trigger events occur before the GNBREQ+1 requests and data transfers are completed.

DMAMUX request multiplexer inputs

17

DMAREQ_ID	Resource	DMAREQ_ID	Resource	DMAREQ_ID	Resource
1	dmamux_req_gen0	44	TIM1_CH3	87	TIM20_CH2
2	dmamux_req_gen1	45	TIM1_CH4	88	TIM20_CH3
3	dmamux_req_gen2	46	TIM1_UP	89	TIM20_CH4
4	dmamux_req_gen3	47	TIM1_TRIG	90	TIM20_UP
5	ADC1	48	TIM1_COM	91	AES_IN
6	DAC1_CH1	49	TIM8_CH1	92	AES_OUT
7	DAC1_CH2	50	TIM8_CH2	93	TIM20_TRIG
8	TIM6_UP	51	TIM8_CH3	94	TIM20_COM
9	TIM7_UP	52	TIM8_CH4	95	HRTIM_MASTER
10	SPI1_RX	53	TIM8_UP	96	HRTIM_TIMA
11	SPI1_TX	54	TIM8_TRIG	97	HRTIM_TIMB
12	SPI2_RX	55	TIM8_COM	98	HRTIM_TIMC
13	SPI2_TX	56	TIM2_CH1	99	HRTIM_TIMD
14	SPI3_RX	57	TIM2_CH2	100	HRTIM_TIME
15	SPI3_TX	58	TIM2_CH3	101	HRTIM_TIMF
16	I2C1_RX	59	TIM2_CH4	102	DAC3_CH1
17	I2C1_TX	60	TIM2_UP	103	DAC3_CH2
18	I2C2_RX	61	TIM3_CH1	104	DAC4_CH1
19	I2C2_TX	62	TIM3_CH2	105	DAC4_CH2
20	I2C3_RX	63	TIM3_CH3	106	SPI4_RX
21	I2C3_TX	64	TIM3_CH4	107	SPI4_TX
22	I2C4_RX	65	TIM3_UP	108	SAI1_A
23	I2C4_TX	66	TIM3_TRIG	109	SAI1_B
24	USART1_RX	67	TIM4_CH1	110	FMAC_Read
25	USART1_TX	68	TIM4_CH2	111	FMAC_Write
26	USART2_RX	69	TIM4_CH3	112	Cordic_Read
27	USART2_TX	70	TIM4_CH4	113	Cordic_Write
28	USART3_RX	71	TIM4_UP	114	USBPD_RX
29	USART3_TX	72	TIM5_CH1	115	USBPD_TX
30	UART4_RX	73	TIM5_CH2		
31	UART4_TX	74	TIM5_CH3		
32	UART5_RX	75	TIM5_CH4		
33	UART5_TX	76	TIM5_UP		
34	LPUART1_RX	77	TIM5_TRIG		
35	LPUART1_TX	78	TIM15_CH1		
36	ADC2	79	TIM15_UP		
37	ADC3	80	TIM15_TRIG		
38	ADC4	81	TIM15_COM		
39	ADC5	82	TIM16_CH1		
40	QUADSPI	83	TIM16_UP		
41	DAC2_CH1	84	TIM17_CH1		
42	TIM1_CH1	85	TIM17_UP		
43	TIM1_CH2	86	TIM20_CH1		



This table shows the STM32G4 mapping of the DMAMUX request multiplexer inputs, for any channel.

Assigning a request input is programmed by the DMAREQ_ID for any DMAMUX request multiplexer channel x (DMAMUX_CxCR register).

The same request input must not be mapped to two different channels.

DMAMUX trigger & synchronization inputs

18

SIG_ID SYNC_ID	Resource	SIG_ID SYNC_ID	Resource
0	EXTI LINE0	12	EXTI LINE12
1	EXTI LINE1	13	EXTI LINE13
2	EXTI LINE2	14	EXTI LINE14
3	EXTI LINE3	15	EXTI LINE15
4	EXTI LINE4	16	dmamux_evt0
5	EXTI LINE5	17	dmamux_evt1
6	EXTI LINE6	18	dmamux_evt2
7	EXTI LINE7	19	dmamux_evt3
8	EXTI LINE8		
9	EXTI LINE9		
10	EXTI LINE10		
11	EXTI LINE11		



This table shows the STM32G4 mapping of the trigger inputs and the synchronization inputs for any channel.

Assigning a trigger input is programmed by the SIG_ID field of any DMAMUX request generator x (DMAMUX_RGxCR register).

Assigning a synchronization input is programmed by the SYNC_ID field of any DMAMUX request multiplexer channel x (DMAMUX_CxCR register).

- Interrupt event for each request generator channel

Interrupt event	Description
Request generator trigger overrun	Set when a trigger input overrun is detected (before that, a programmed number of DMA requests created by the DMAMUX request generator has been completed)

- Interrupt event for each request multiplexer channel

Interrupt event	Description
Request multiplexer synchronization overrun	Set when a synchronization input overrun is detected (before that, a programmed number of transmitted DMA requests or generated DMAMUX events has been completed)



Each DMAMUX request generator channel can notify software of a trigger overrun.
Each DMAMUX request multiplexer channel can notify software of a synchronization overrun.

DMA/DMAMUX in low-power modes

20

Mode	Description
Run	Active.
Low-power run	Active.
Sleep	Active. DMA/DMAMUX interrupts can wake up the CPU.
Low-power sleep	Active. DMA/DMAMUX interrupts can wake up the CPU.
Stop 0/Stop 1	Clocked-off & frozen. DMA/DMAMUX registers retention.
Standby	Powered-down. DMA/DMAMUX must be reinitialized after exiting Standby mode.
Shutdown	Powered-down. DMA/DMAMUX must be reinitialized after exiting Shutdown mode.



This table indicates the state of the DMA controller and DMAMUX according to the power mode.
In Sleep and Low-power sleep modes, the DMA controller and the DMAMUX remain active and can be used for example to transfer UART or I2C received characters to memory, and afterwards to wake up the CPU.