STM32L5 – DMA & DMAMUX

Direct memory access controller (DMA)
DMA request multiplexer (DMAMUX)

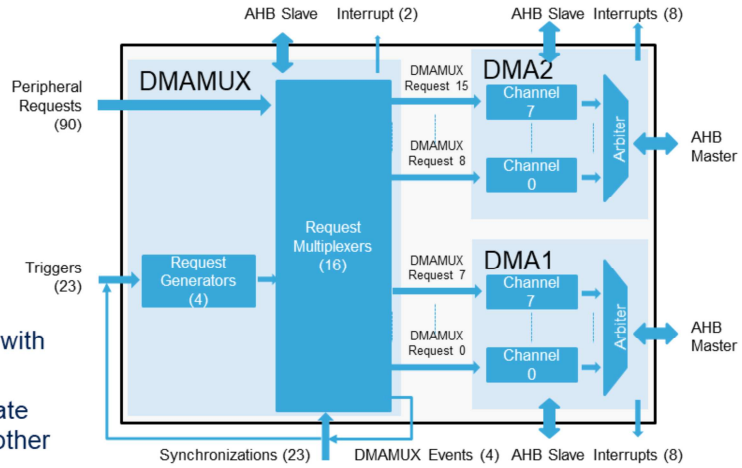
Revision 1.0

life.augmented

Welcome to this presentation of the STM32L5 Direct Memory Access controller (DMA).
It covers the main features of the DMA controller module, enhanced by the new DMA request multiplexer (DMAMUX) module.

Overview

- 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
  - Request chaining: with the request counter to generate an event that is a trigger/synchronization input to another request/channel
- TrustZone®-aware DMA & DMAMUX
  - Resource isolation at the channel level
    - Secure or non-secure channel, if TZEN is enabled
    - Privileged or unprivileged channel



Application benefit

Off-load CPU for TrustZone®-aware 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 trustzone-aware data transfers, from any memory-mapped source towards any memory-mapped destination.

STM32L5 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
        - ❖ Request chaining: with the DMAMUX request counter to generate an event that is an input trigger or synchronization to another request/channel
- TrustZone-aware DMA & DMAMUX
    - ❖ Resource isolation at the channel level
        - ❖ Secure or non-secure channel, if TZEN is enabled
        - ❖ Privileged or unprivileged channel

STM32L5 DMA & DMAMUX instance

DMAMUX features	DMAMUX
Number of peripheral requests	90
Number of request generators	4
Number of triggers	23
Number of synchronizations	23
Number of output DMA requests	16

DMA features	DMA1
Number of channels	8

DMA features	DMA2
Number of channels	8

life.augmented

There are 90 peripheral requests and 4 DMAMUX request generators
There are 23 triggers and synchronization inputs.
There are 16 DMA channels/requests, 8 per DMA controller.

DMA controller (1/3)

- Independently configurable channels over one DMA controller
  - A channel can be assigned to a DMA hardware request from a peripheral, in memory-to-peripheral, peripheral-to-memory or peripheral-to-peripheral 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 separate 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

life.augmented

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, memory-to-peripheral data transfers or peripheral-to-peripheral 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 separate 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.

DMA controller (2/3)

- 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 a fixed address
  - Programmable amount of data to be transferred within a block
    - Up to 256K source data
    - Automatically decremented by hardware
- In Circular Buffer mode (continuous data transfer to/from 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

life.augmented

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 256K source data
  - ❖ Automatically decremented by hardware

In Circular Buffer mode (continuous data transfer to/from 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

DMA controller (3/3)

- Memory-to-memory mode
  - A block transfer starts as soon as the channel is enabled in this mode (no hardware request)
- Peripheral-to-memory, memory-to-peripheral and peripheral-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 has been successfully transferred to the destination
- In any mode
  - Channel arbitration is reassessed between every data transfer

life.augmented

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, memory-to-peripheral and peripheral-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 peripheral-to-peripheral mode, the hardware request from a peripheral is selected to trigger the DMA channel. This peripheral is the DMA initiator and paces the data transfer from/to this peripheral to/from a register belonging to another memory-mapped peripheral, this one being not configured in

DMA mode.

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

DMA Interrupts

- Interrupt events for each channel

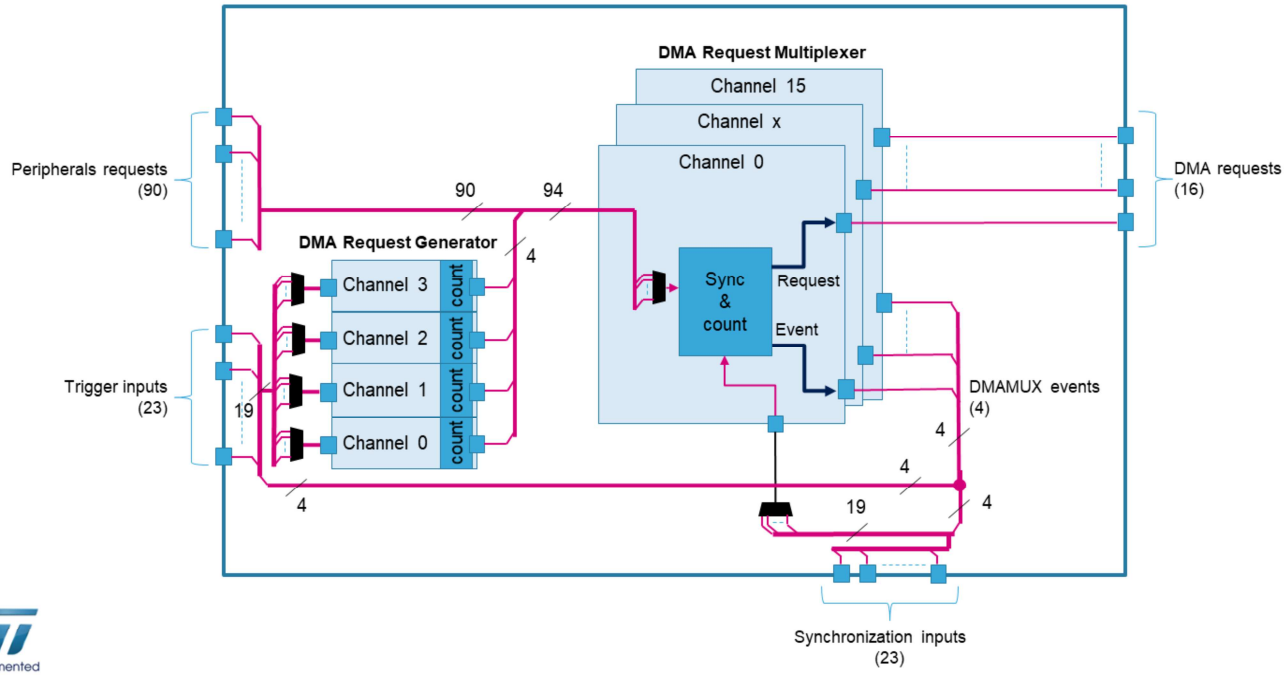
Interrupt event	Description
Half transfer	Set when half of the block of data has been transferred
Transfer complete	Set when the block transfer is completed
Transfer error	Set when an error occurs during a data transfer
Global interrupt	Set whenever a half transfer, a transfer complete or a transfer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

life.augmented

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

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

DMAMUX Block Diagram



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)

- 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 configuring this channel and the DMA controller channel to which it is routed, the DMA channel can be enabled
    - Two different channels are not allowed to be configured with the same DMA request input

life.augmented

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 configuring this channel and the DMA controller channel to which it is routed, the DMA channel can be enabled. Two different channels are not allowed to be configured with the same non-null DMA request input.

DMAMUX Request Multiplexer (2/6)

- 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: the DMA request counter is automatically reloaded with the value programmed in the 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 a different DMA channel, via synchronization and/or trigger

life.augmented

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 value programmed in the 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 a different DMA channel, via synchronization and/or trigger.

DMAMUX Request Multiplexer (3/6)

- 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

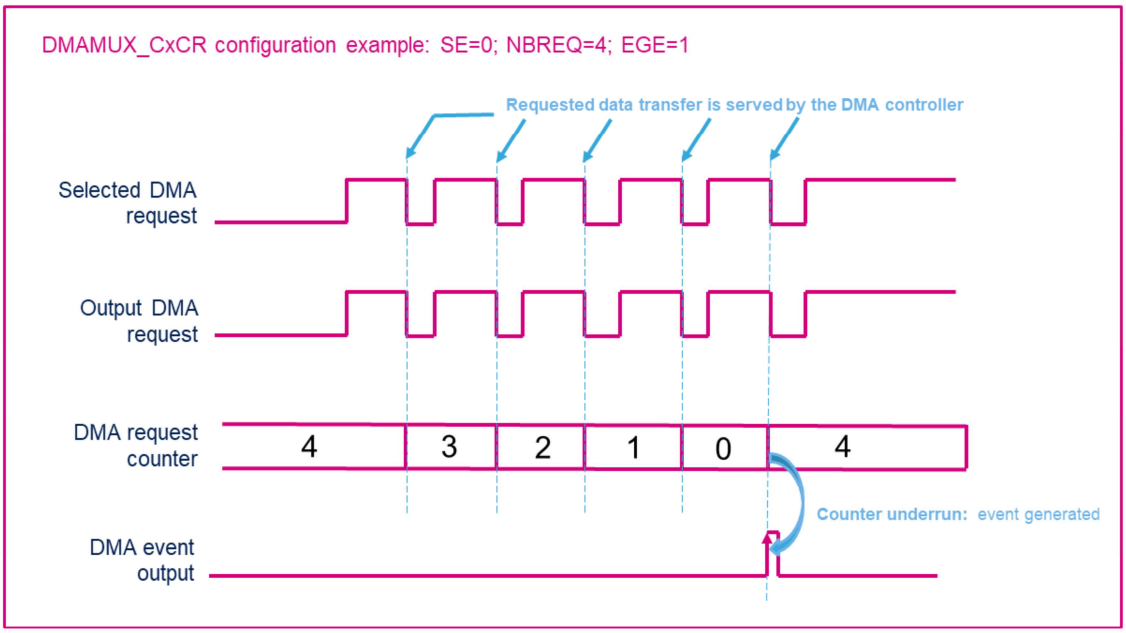
life.augmented

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



When the request multiplexer channel is configured unconditionally (SE=0), the DMA request is transmitted immediately.
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

- Additionally, in Synchronous mode
  - 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, the output DMA request is connected to the pending input request
  - At the counter underrun, the 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)

life.augmented

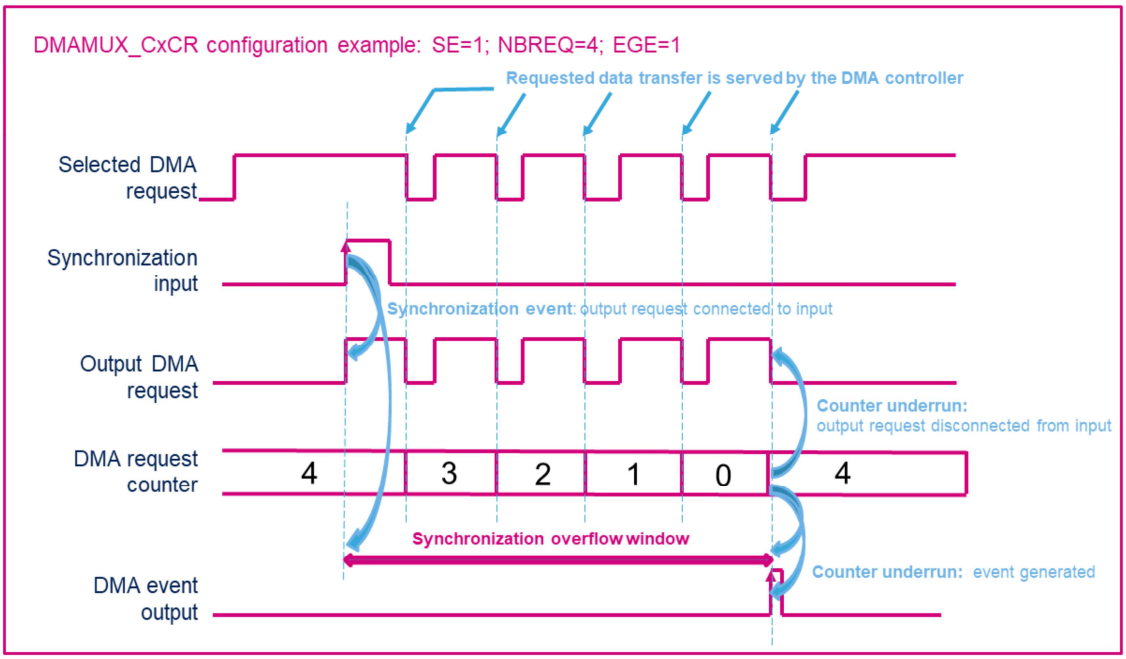
Additionally, in Synchronous mode:

- 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



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.

DMAMUX Request Generator

- 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 the GE field), with
    - Programmable trigger input selection (via the SIG_ID field)
    - Programmable trigger event: none/rising/falling/either edge (via the GPOL field)
    - A built-in request counter (via the GNBREQ field)
  - A served DMA request decrements the programmed request counter
  - At its underrun:
    - Request counter is automatically reloaded with the value programmed in the 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 the OIE field)

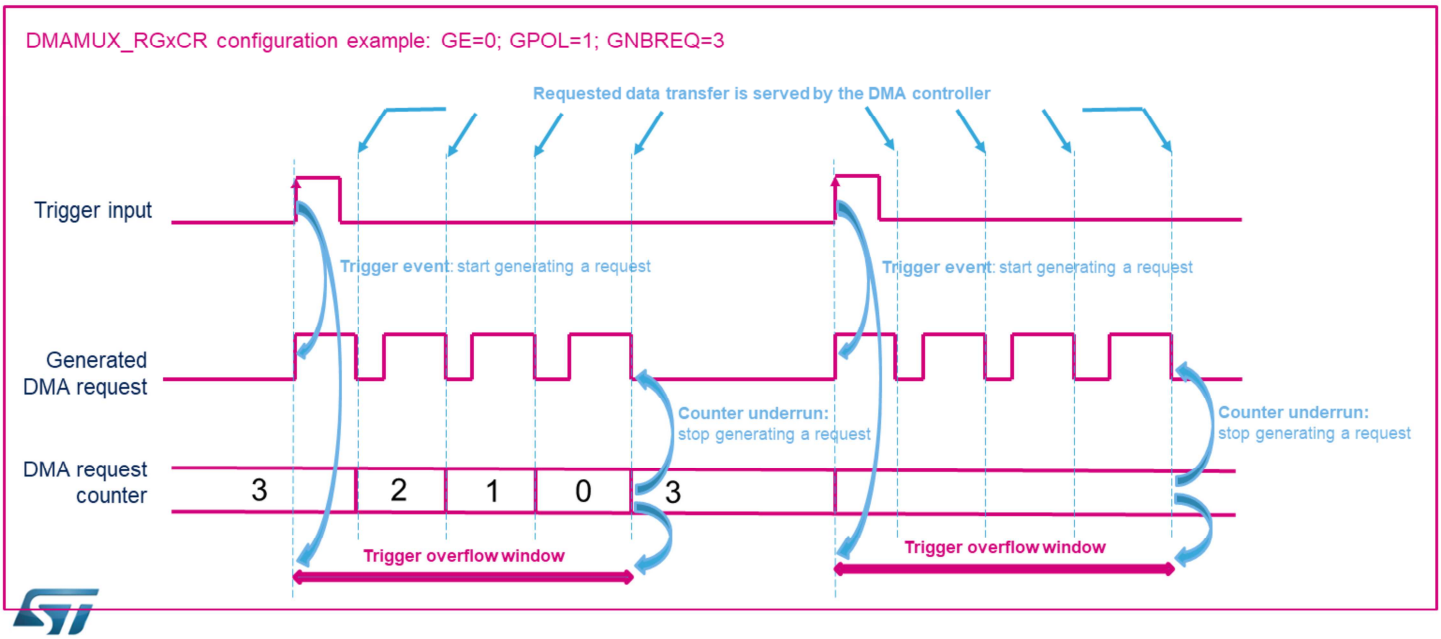
life.augmented

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 the GE field), with:
  - ❖ Programmable trigger input selection (via the SIG_ID field)
  - ❖ Programmable trigger event: none/rising/falling/either edge (via the GPOL field)
  - ❖ A built-in request counter (via the GNBREQ field)
- A served DMA request decrements the programmed request counter
- At its underrun:
  - ❖ Request counter is automatically reloaded with the value programmed in the GNBREQ field
  - ❖ Request generator stops generating a request

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

DMAMUX Request Generator



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

DMAREQ_ID	Resource
1	dmamux_req_gen0
2	dmamux_req_gen1
3	dmamux_req_gen2
4	dmamux_req_gen3
5	ADC1
6	ADC2
7	DAC1
8	DAC2
9	TIM6_UP
10	TIM7_UP
11	SPI1_RX
12	SPI1_TX
13	SPI2_RX
14	SPI2_TX
15	SPI3_RX
16	SPI3_TX
17	I2C1_RX
18	I2C1_TX
19	I2C2_RX
20	I2C2_TX
21	I2C3_RX
22	I2C3_TX
23	I2C4_RX

DMAREQ_ID	Resource
24	I2C4_TX
25	USART1_RX
26	USART1_TX
27	USART2_RX
28	USART2_TX
29	USART3_RX
30	USART3_TX
31	UART4_RX
32	UART4_TX
33	UART5_RX
34	UART5_TX
35	LPUART1_RX
36	LPUART1_TX
37	SAI1_A
38	SAI1_B
39	SAI2_A
40	SAI2_B
41	OCTOSPI1
42	TIM1_CH1
43	TIM1_CH2
44	TIM1_CH3
45	TIM1_CH4
46	TIM1_UP

DMAREQ_ID	Resource
47	TIM1_TRIG
48	TIM1_COM
49	TIM8_CH1
50	TIM8_CH2
51	TIM8_CH3
52	TIM8_CH4
53	TIM8_UP
54	TIM8_TRIG
55	TIM8_COM
56	TIM2_CH1
57	TIM2_CH2
58	TIM2_CH3
59	TIM2_CH4
60	TIM2_UP
61	TIM3_CH1
62	TIM3_CH2
63	TIM3_CH3
64	TIM3_CH4
65	TIM3_UP
66	TIM3_TRIG
67	TIM4_CH1
68	TIM4_CH2
69	TIM4_CH3

DMAREQ_ID	Resource
70	TIM4_CH4
71	TIM4_UP
72	TIM5_CH1
73	TIM5_CH2
74	TIM5_CH3
75	TIM5_CH4
76	TIM5_UP
77	TIM5_TRIG
78	TIM15_CH1
79	TIM15_UP
80	TIM15_TRIG
81	TIM15_COM
82	TIM16_CH1
83	TIM16_UP
84	TIM17_CH1
85	TIM17_UP
86	DFSDM1_FLT0
87	DFSDM1_FLT1
88	DFSDM1_FLT2
89	DFSDM1_FLT3
90	AES_IN
91	AES_OUT
92	HASH_IN

DMAREQ_ID	Resource
93	USBPD_TX
94	USBPD_RX

life.augmented

This table shows the STM32L5 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

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	20	LPTIM1_OUT
9	EXTI LINE9	21	LPTIM2_OUT
10	EXTI LINE10	22	LPTIM3_OUT
11	EXTI LINE11		

life.augmented

This table shows the STM32L5 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).

DMA and DMAMUX Channel Isolation (1/4)

19

- DMA and DMAMUX are TrustZone®- aware at channel level
- A DMA channel x can be programmed by the software and isolated as a:
  - Privileged or unprivileged resource
    - Depending on the PRIV bit of the DMA_CCRx register, as first written by a privileged software
  - Secure or non-secure resource, if TZEN is active at the device level
    - Depending on the SECM bit of the DMA_CCRx register, as first written by a secure software
  - Via the DMA TrustZone® -aware and privileged-aware AHB slave port

life.augmented

The DMA and DMAMUX modules present in the STM32L5 microcontroller support resource isolation at channel level. This is a new feature vs STM32L4 series.
DMA and DMAMUX are TrustZone®- aware at a channel level
A DMA channel x can be programmed by the software and isolated as a:

- Privileged or unprivileged resource, depending on the PRIV bit of the DMA_CCRx register, as first written by a privileged software
- Secure or non-secure resource, if TZEN is active at the device level, depending on the SECM bit of the DMA_CCRx register, as first written by a secure software
- Via the DMA TrustZone® -aware and privileged-aware AHB slave port

DMA and DMAMUX Channel Isolation (2/4)

- The secure state and the privileged state of a DMA channel x (i.e. the PRIV and SECM bits of the DMA_CCRx control register) rule out:
  - The read/write access rights to the other configuration/status registers and the other register fields of channel x
  - The read/write access rights to the registers and register fields belonging to the DMAMUX output channel y which is connected to the DMA channel x
    - Via the DMAMUX TrustZone® -aware and privileged-aware AHB slave port
- Thus, a DMAMUX request multiplexer/generator channel is also isolated as a secure or non-secure resource, and as a privileged or unprivileged resource

life.augmented

The secure state and the privileged state of a DMA channel x (i.e. the PRIV and SECM bits of the DMA_CCRx control register) rule out:
- the read/write access rights to the other configuration/status registers and the other register fields of channel x
- the read/write access rights to the registers and register fields belonging to the DMAMUX output channel y which is connected to the DMA channel x via the DMAMUX TrustZone® -aware and privileged-aware AHB slave port

Thus, a DMAMUX request multiplexer/generator channel is also isolated as a secure or non-secure resource, and as a privileged or unprivileged resource

DMA and DMAMUX Channel Isolation (3/4)

- Once a channel x is set as secure/non-secure and as privileged/unprivileged:
  - An unprivileged channel performs unprivileged DMA transfers
  - A privileged channel performs privileged DMA transfers
    - Via a privileged-aware DMA AHB master port
  - A non-secure channel performs non-secure DMA transfers
  - A secure channel can perform secure or non-secure DMA transfers, with
    - Secure or non-secure data read from the source address
      - Depending on the securely written SSEC bit of the DMA_CCRx register
    - Secure or non-secure data write to the destination address
      - Depending on the securely written DSEC bit of the DMA_CCRx register
    - Via a TrustZone®-aware DMA AHB master port
- A source/destination peripheral or memory address is secure or non-secure and privileged or unprivileged, depending on the globally configured MPU, SAU and GTZC units at the system level (See STM32L5 system security chapter).

life.augmented

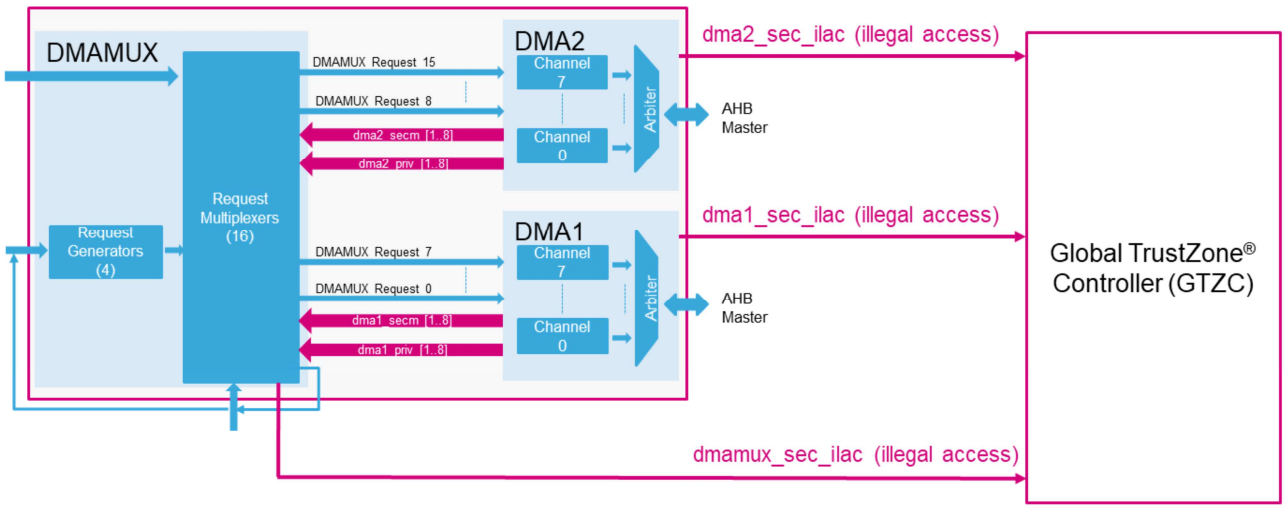
Once a channel x is configured as secure/non-secure and as privileged/unprivileged:

- An unprivileged channel performs unprivileged DMA transfers
- A privileged channel performs privileged DMA transfers via a privileged-aware DMA AHB master port
- A non-secure channel performs non-secure DMA transfers
- A secure channel can perform secure or non-secure DMA transfers,
  - ❖ with Secure or non-secure data read from the source address depending on the securely written SSEC bit of the DMA_CCRx register
  - ❖ Secure or non-secure data write to the destination address depending on the securely written DSEC bit of the DMA_CCRx register
  - ❖ Via a TrustZone®-aware DMA AHB master port

A source/destination peripheral or memory address is secure

or non-secure and privileged or unprivileged, depending on the globally configured MPU, SAU and GTZC units at the system level (See STM32L5 system security chapter).

DMA and DMAMUX Channel Isolation (4/4)



The DMA controller generates a secure bus, dma_secm[7:0], reflecting the SECM bit of the DMA_CCRx register, in order to keep the DMAMUX informed of the secure / non-secure state of each DMA channel x.
The DMA controller also generates a security illegal access pulse event, dma_sec_ilac, on an illegal non-secure software access to a secure DMA register or register field. This event is routed to the TrustZone interrupt controller.
The DMA controller generates a privileged bus, dma_priv[7:0], reflecting the PRIV bit of the DMA_CCRx register, in order to keep the DMAMUX, informed of the privileged / unprivileged state of each DMA channel x.
The DMAMUX also generates a security illegal access pulse event, dmamux_sec_ilac, on an illegal non-secure software access to a secure DMAMUX register or register field. This event is routed to the TrustZone interrupt controller.

DMAMUX Interrupt

- Interrupt event for each request generator channel

Interrupt event	Description
Non-secure request generator trigger overrun	Set when a trigger input overrun is detected on a non-secure channel
Secure request generator trigger overrun	Set when a trigger input overrun is detected on a secure channel

- Interrupt event for each request multiplexer channel

Interrupt event	Description
Non secure request multiplexer synchronization overrun	Set when a synchronization input overrun is detected on a non-secure channel
Secure request multiplexer synchronization overrun	Set when a synchronization input overrun is detected on a secure channel

life.augmented

Each DMAMUX request generator channel can notify software of a trigger overrun through two interrupt requests, one per security level.
A request generator overrun is signaled when a new DMA request trigger event occurs before the request generator counter is equal to zero.
Each DMAMUX request multiplexer channel can notify software of a synchronization overrun through two interrupt requests, one per security level.
A synchronization overrun is signaled when a new synchronization occurs before the request counter is equal to zero.

DMA/DMAMUX in low-power modes

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/Stop 2	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

life.augmented

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.