Application note

SR5 E1 line - DMA controller and DMA request multiplexer

Introduction

SR5E1x 32-bit Arm® Cortex®-M7 microcontrollers embed direct memory access controllers (DMA) and a DMA request multiplexer (DMAMUX) used to provide high-speed data transfer between peripherals and memory and between memory and memory. This keeps CPU resources free for other operations. Data can be quickly moved by DMA upon a peripheral request or a software trigger without any CPU load.

This application note gives an overview and describes the features of the DMA and DMAMUX in the SR5E1x 32-bit Arm® Cortex®-M7 microcontrollers.

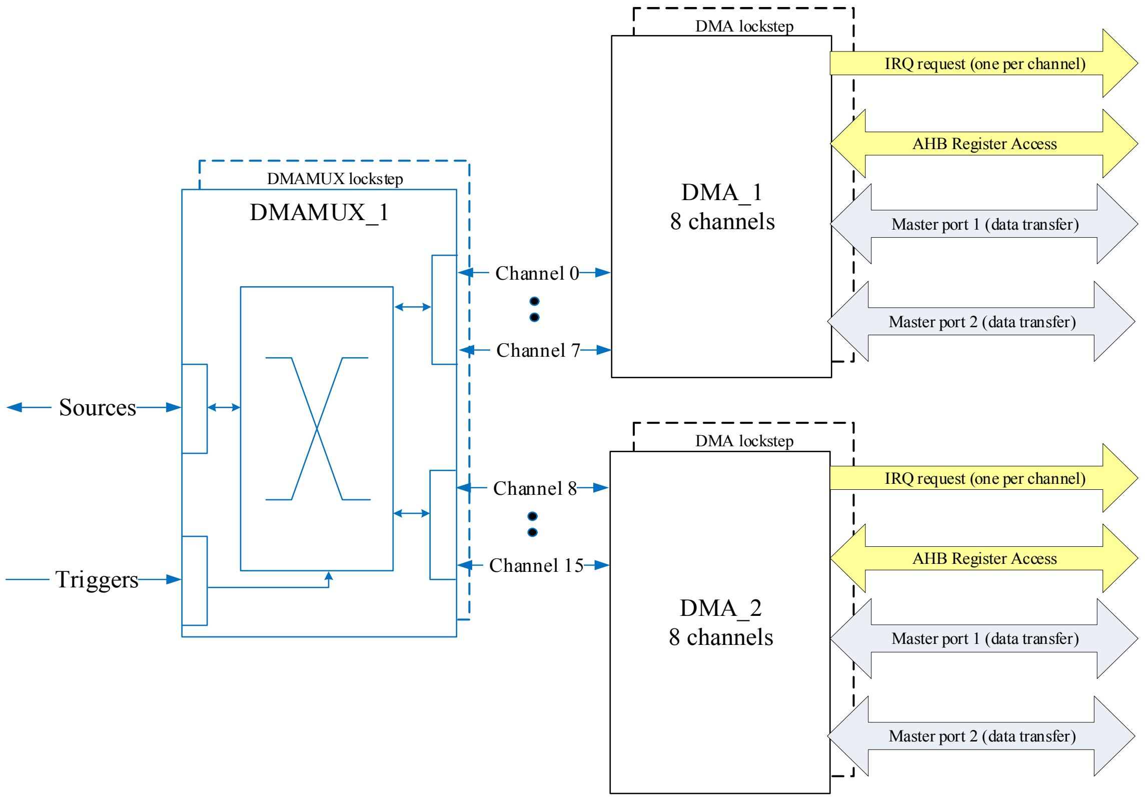
For further information on DMA and DMAMUX in SR5E1x devices, refer to the product reference manuals available on www.st.com.

This document applies to the SR5E1x 32-bit Arm® Cortex®-M7 microcontrollers.

1 DMA and DMAMUX architecture overview

The SR5E1x includes two direct memory access (DMA) instances (DMA_1 and DMA_2) and a DMA request multiplexer (DMAMUX_1). The Figure 1 shows the whole DMA DMAMUX architecture.

Figure 1. DMA and DMAMUX architecture


DMA lockstep
DMA_1
8 channels
IRQ request (one per channel)
AHB Register Access
Master port 1 (data transfer)
Master port 2 (data transfer)
DMAMUX lockstep
DMAMUX_1
Channel 0
Channel 7
Sources
Triggers
Channel 8
Channel 15
DMA lockstep
DMA_2
8 channels
IRQ request (one per channel)
AHB Register Access
Master port 1 (data transfer)
Master port 2 (data transfer)

Each DMA instance has the following features:

- 8 input streams
- Double buffer mode for each stream
- 4-words FIFO per stream

Each DMA instance has a replica that can be enabled in lockstep in order to reach a high level of safety. The enabling of the replica is done at startup via the global system configuration DCF record (LS_DMA).

The unique DMA request multiplexer (DMAMUX_1) has 16 output channels:

- DMAMUX_1 channels 0 to 7 are connected to DMA_1 channels 0 to 7
- DMAMUX_1 channels 8 to 15 are connected to DMA_2 channels 0 to 7

The DMAMUX_1 has a replica that can be enabled in lockstep at startup via the global system configuration DCF record (LS_DMA).

In the following chapters details of the DMAMUX and DMA features and their functionalities are given.

2 DMAMUX description

A peripheral indicates a request for DMA transfer by setting its DMA request signal. The DMA request is pending until it is served by the DMA controller that generates a DMA acknowledge signal, and the corresponding DMA request signal is deasserted.

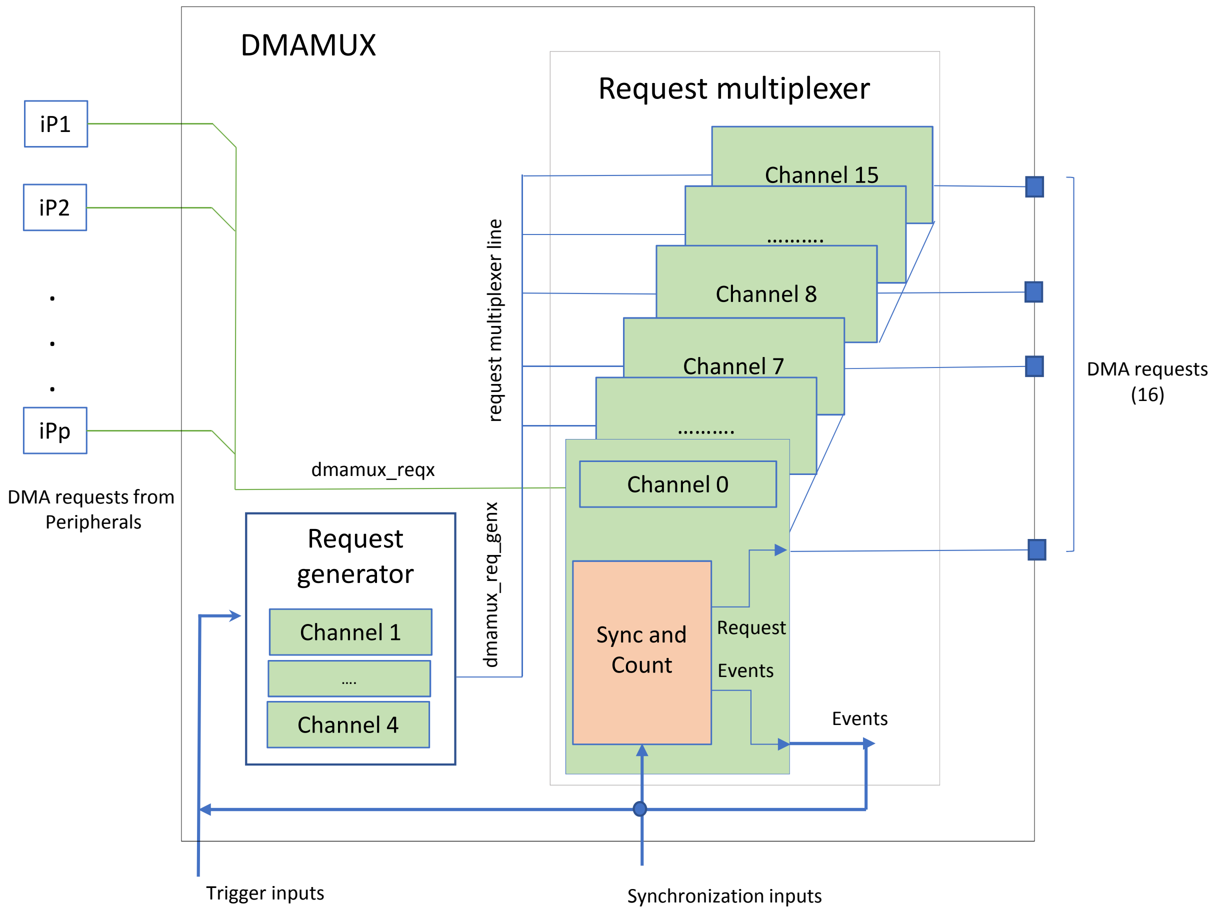
The DMAMUX request multiplexer enables routing a DMA request line between the peripherals and the DMA controllers of the product. The routing function is ensured by a programmable multichannel DMA request line multiplexer. Each channel selects a unique DMA request line, unconditionally or synchronously 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.

As shown in the Figure 2, each DMA request line is connected in parallel to all the channels of the DMAMUX request multiplexer line. A DMA request multiplexer line is sourced either from the peripherals (dmamux_reqx signal) or from the DMAMUX request generator (dmamux_req_genx signal). The DMAMUX request line multiplexer Channel x selects the DMA request line number as configured by the DMAREQ_ID field in the DMAMUX_CxCR register.

The SR5E1x includes one DMA request multiplexer (DMAMUX_1) with 16 output channels:

- DMAMUX_1 channels 0 to 7 are connected to DMA_1 channels 0 to 7
- DMAMUX_1 channels 8 to 15 are connected to DMA_2 channels 0 to 7

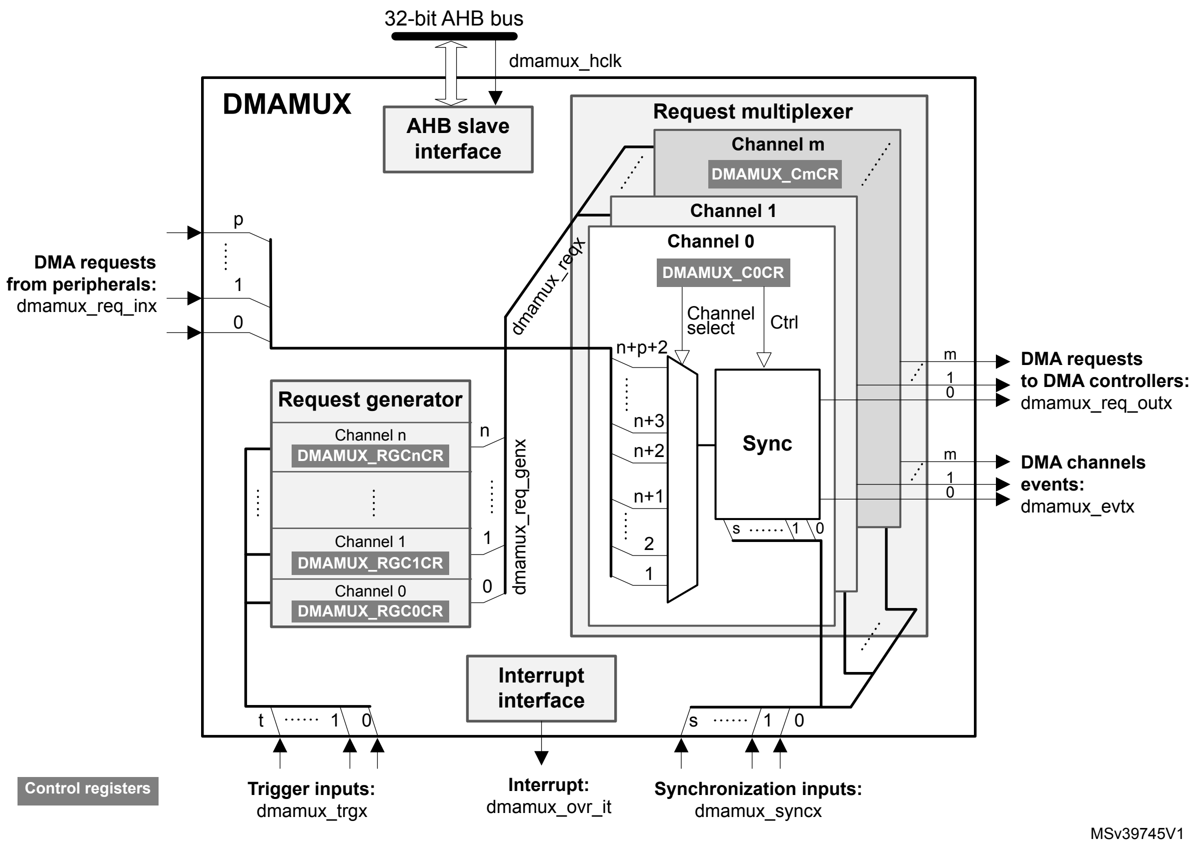
Figure 2. DMAMUX request multiplexer and request generator overview



2.1 DMAMUX features

The Figure 3 below shows a detailed DMAMUX block diagram.

Figure 3. DMAMUX block diagram



The DMAMUX is mainly composed of two components, the request multiplexer and the request generator.

The request multiplexer includes a synchronization unit per channel, with inputs/outputs as follows:

- Inputs:
  - dmamux_reqx: DMA request from a peripheral (dmamux_req_inx) and from the request generator (dmamux_req_genx)
  - dmamux_syncx: optional synchronization event
- Outputs:
  - dmamux_req_outx: DMA request dmamux_reqx forwarded from the input to the output (to DMA controller)
  - dmamux_evtx: optional generated event that may be used to trigger/synchronize other DMAMUX channels

The request generator allows DMA request generation on interrupt signals or events, with input/output as follows:

- Input: dmamux_trgx, trigger event inputs to the request generator subblock
- Output: dmamux_req_genx, DMA request from the request generator subblock to the DMAMUX request multiplexer channels

Thanks to the request generator block, user software can trigger DMA transfers based on signals from peripherals that do not implement the DMA requests.

2.2 Request routing and synchronization

2.2.1 Unconditional request forwarding

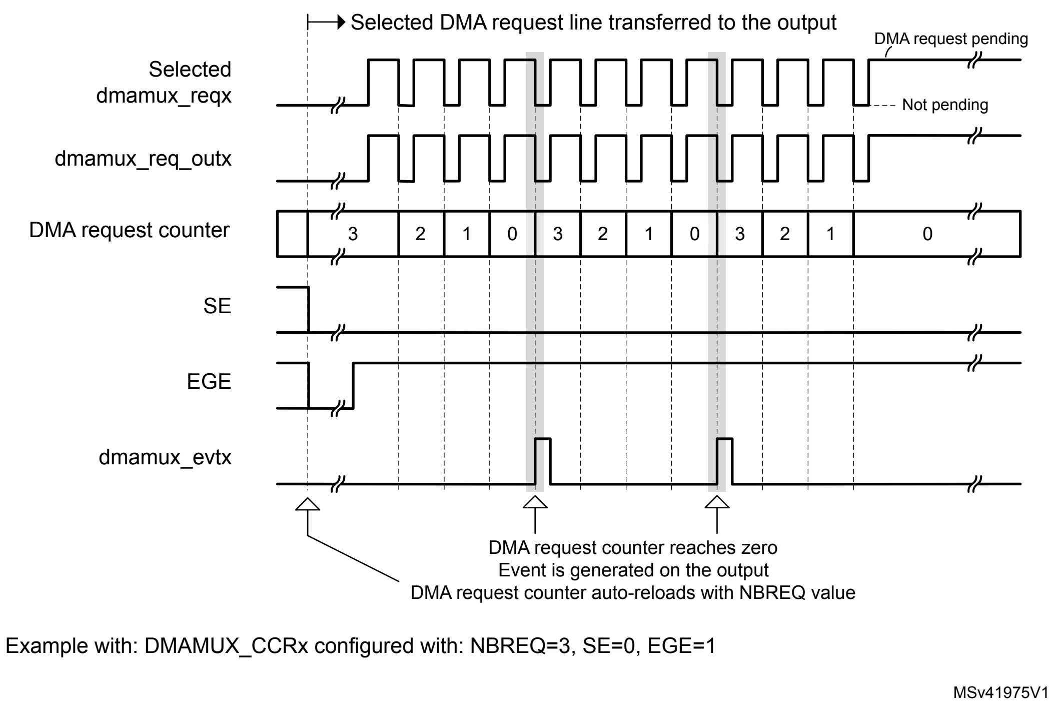
In order to perform peripheral-to-memory or memory-to-peripheral transfers, the DMA controller channel requires each time a peripheral DMA request line. Each time a request occurs, the DMA channel transfers data from/to the peripheral. The DMAMUX request multiplexer channel allows the selection/routing of the peripheral DMA request line to the DMA channel.

When the multiplexer is set, it ensures the actual routing of the DMA request line. The connection of the peripheral DMA request to the multiplexer channel output is selected through the programmed ID in the DMAREQ_ID bits of the channel control register (DMAMUX_CxCR).

After the configuration of a DMAMUX channel, the corresponding DMA controller channel can be configured on its turn. Two different DMAMUX channels cannot be configured to select the same peripheral DMA request line as source unless the application ensures that these channels are not requested to be served at the same time.

In the unconditional forwarding, the selected dmamux_reqx signal is sent to the dmamux_req_outx output as soon as the dmamux_reqx is available.

Figure 4. **DMA request line multiplexer channel–unconditional forwarding and event generation**


Selected DMA request line transferred to the output
DMA request pending
Not pending
Selected dmamux_reqx
dmamux_req_outx
DMA request counter
3
2
1
0
3
2
1
0
3
2
1
0
SE
EGE
dmamux_evtx
DMA request counter reaches zero
Event is generated on the output
DMA request counter auto-reloads with NBREQ value
Example with: DMAMUX_CCRx configured with: NBREQ=3, SE=0, EGE=1
MSv41975V1

2.2.2 Synchronized request forwarding

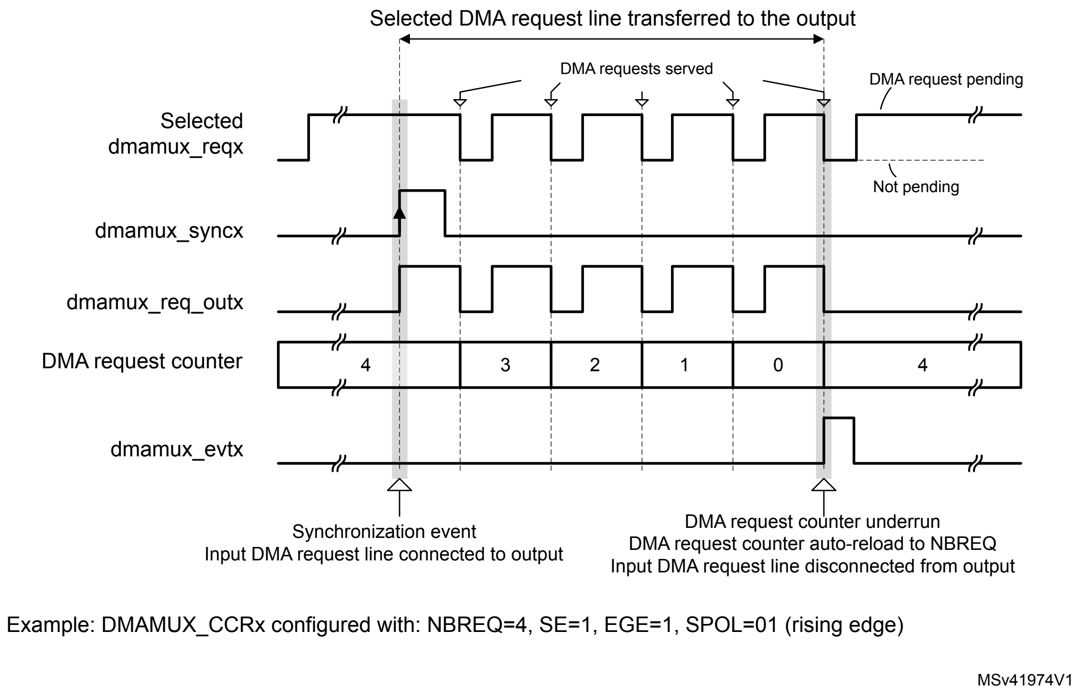
In addition to unconditional request forwarding, the synchronization unit allows the software to implement conditional request forwarding. The routing is effectively done only when a defined condition is detected. The DMA transfers can be synchronized with internal or external signals.

For example, the user software can use the synchronization unit to initiate or adjust data transmission throughput. DMA request can be forwarded in one of the following ways:

- Each time an edge is detected on a GPIO pin (EXTI)
- In response to a periodic event from a timer
- In response to an asynchronous event from a peripheral
- In response to an event from another request router (request chaining)

On top of DMA request conditioning, the synchronization unit allows the generation of events that may be used by other DMAMUX subblocks (such as the request generator or another DMAMUX request multiplexer channel).

Figure 5. DMA request line multiplexer channel–synchronous mode and event generation


Selected DMA request line transferred to the output
DMA requests served
DMA request pending
Selected dmamux_reqx
Not pending
dmamux_syncx
dmamux_req_outx
DMA request counter
4
3
2
1
0
4
dmamux_evtx
Synchronization event
Input DMA request line connected to output
DMA request counter underrun
DMA request counter auto-reload to NBREQ
Input DMA request line disconnected from output
Example: DMAMUX_CCRx configured with: NBREQ=4, SE=1, EGE=1, SPOL=01 (rising edge)
MSv41974V1

When the DMAMUX channel is configured in synchronous mode its behavior is as follows:

1. The request multiplexer input (DMA request from the peripheral) can become active but it is not forwarded on the DMAMUX request multiplexer output until the synchronization signal is received.
2. When the sync event is received the request multiplexer connects its input and output and the pending peripheral request, if any, is forwarded.
3. Each forwarded DMA request decrements the request multiplexer counter (user programmed value). When the counter reaches zero and the last forwarded request is acknowledged by the DMA controller, the connection between the DMA controller and the peripheral is disabled (not forwarded) waiting for a new synchronization event.
4. If a new synchronization event occurs before the request counter underrun, the synchronization overrun flag bit SOFx is set in the DMAMUX_CSR status register.

For each underrun of the counter, the request multiplexer line can generate an optional event to synchronize with a second DMAMUX line. The same event can be used in some low-power scenarios to switch the system back to stop mode without any CPU intervention.

Synchronization mode can be used to automatically synchronize data transfers with a timer, for example, or to trigger the transfers on a peripheral event.

The synchronization signal (SYNC_ID), the synchronization signal polarity (SPOL) and the number of requests to forward (NBREQ+1) are configured in the request line multiplexer channel configuration register (DMAMUX_CxCR).

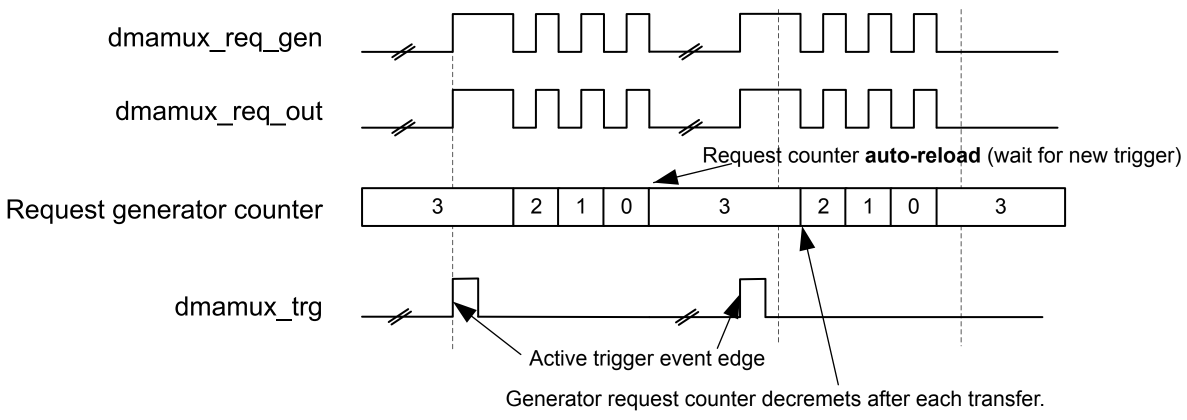
2.3 Request generation

The request generator can be considered as an intermediary between a peripheral and the DMA controllers. It allows peripherals without DMA capability (such as RTC alarm or comparators) to generate a programmable number of DMA requests on an event. The trigger signal (SIG_ID), the trigger polarity (GPOL) and the number of requests minus 1 to generate (GNBREQ) are configured in the request generator configuration register (DMAMUX_RGxCR).

Upon the trigger event reception, the corresponding generator channel starts generating DMA requests on its output. Each time the DMAMUX generated request is served by the connected DMA controller, a built-in DMA request counter (one counter per request generator channel) is decremented.

At its underrun, the request generator channel stops generating DMA requests and the DMA request counter is automatically reloaded to its programmed value upon the next trigger event.

Figure 6. **DMA request generation**


dmamux_req_gen
dmamux_req_out
Request counter **auto-reload** (wait for new trigger)
Request generator counter
3
2
1
0
3
2
1
0
3
dmamux_trg
Active trigger event edge
Generator request counter decremets after each transfer.

If a new trigger event is received while the generator is managing the previous triggered DMA request sequence, then the request trigger event overrun flag bit OFx is asserted by the hardware in the status DMAMUX_RGSR register.

2.4 Request generation and synchronization

In order to implement autonomous transfer and control scenarios, the DMAMUX offers the possibility to combine request generation and request synchronization feature within the same configuration.

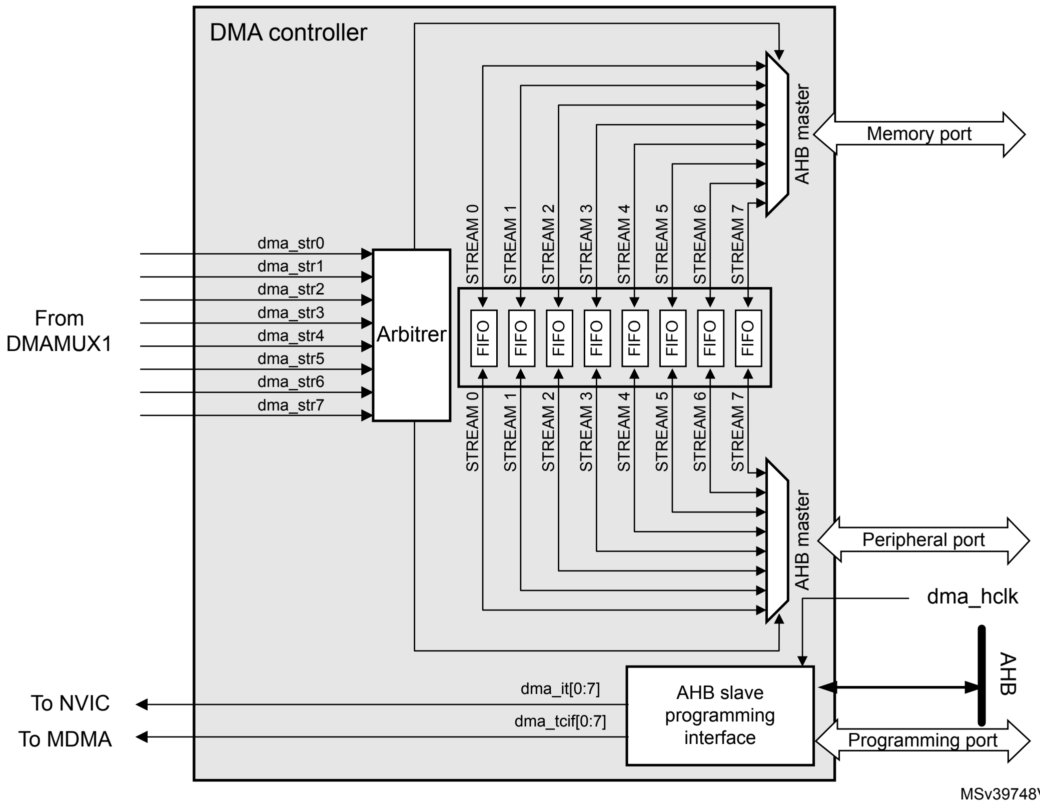
3 DMA description

Direct memory access (DMA) is used in order to provide high-speed data transfer between peripherals and memory and between memory and memory. Data can be quickly moved by DMA without any CPU action.

SR5E1x devices embed two DMA controllers, 8 input streams each, dedicated to managing memory access requests from one or more peripherals. Each DMA has two ports, one peripheral port and one memory port, which can work simultaneously.

Each DMA stream is driven by one DMAMUX1 output channel (request). DMAMUX1 output request can be individually programmed in order to select the DMA request source signal, from any of the 151 available request input signals. Each DMA controller has an arbiter for handling the priority between DMA requests. The Figure 7 shows the DMA block diagram.

Figure 7. DMA block diagram



3.1 DMA transfer properties

A DMA transfer is characterized by the following properties:

- DMA stream/channel
- Stream priority
- Source/destination addresses
- Transfer mode
- Transfer size
- Source/destination address incrementing or nonincrementing
- Source and destination data width
- Transfer type
- FIFO mode
- Source/destination burst size

- Double-buffer mode
- Flow control

The following subsections provide a detailed description of each DMA transfer property.

3.1.1 DMA streams/channels

SR5E1x devices embed two DMA controllers, offering up to 16 streams in total (eight per controller), each dedicated to managing memory access requests from one or more peripherals.

Each DMA stream is driven by one DMAMUX1 output channel (request).

Each stream can be configured to perform:

- Regular type transactions: memory-to-peripherals, peripherals-to-memory, or memory-to-memory transfers
- Double-buffer type transactions: double buffer transfers using two memory pointers for the memory (while the DMA is reading/writing from/to a buffer, the application can write/read to/from the other buffer).

The DMA source request for a stream is made by means of the field DMAREQ_ID in the DMAMUX_CxCR register. The Table 1 shows the assignment of input multiplexer to the possible resources to be used in the field DMAREQ_ID.

Table 1. DMAMUX: assignment of multiplexer inputs to resources

DMA request MUX input	Resource	DMA request MUX input	Resourcer	DMA request MUX input	Resource
1	DMAMUX_Req GO	55	TIM8_COM	109	Reserved
2	DMAMUX_Req G1	56	TIM2_CH1	110	Reserved
3	DMAMUX_Req G2	57	TIM2_CH2	111	Reserved
4	DMAMUX_Req G3	58	TIM2_CH3	112	Cordic_read
5	ADC1	59	TIM2_CH4	113	Cordic_write
6	B-DAC1_CH1	60	TIM2_UP	114	Reserved
7	B-DAC1_CH2	61	TIM3_CH1	115	Reserved
8	TIM6_UP	62	TIM3_CH2	116	HRTIM2_MASTER (hrtim_dma1)
9	TIM?_UP	63	TIM3_CH3	117	HRTIM2_TIMA (hrtim_dma2)
10	SPI1_RX	64	TIM3_CH4	118	HRTIM2_TIMB (hrtim_dma3)
11	SPI1_TX	65	TIM3_UP	119	HRTIM2_TIMC (hrtim_dma4)
12	SPI2_RX	66	TIM3_TRIG	120	HRTIM2_TIMD (hrtim_dma5)
13	SPI2_TX	67	TIM4_CH1	121	HRTIM2_TIME (hrtim_dma6)
14	SPI3 RX	68	TIM4 CH2	122	HRTIM2_TIMF (hrtim_dma7)
15	SPI3_TX	69	TIM4_CH3	123	Reserved
16	I2C1_RX	70	TIM4_CH4	124	Reserved
17	I2C1_TX	71	TIM4_UP	125	Reserved
18	I2C2_RX	72	TIM5_CH1	126	Reserved
20	Reserved	74	TIM5 CH3	128	Reserved
21	Reserved	75	TIM5_CH4	129	Reserved
22	Reserved	76	TIM5_UP	130	CAN_SUB_1_DMU_1_TX
23	Reserved	77	TIM5_TRIG	131	CAN_SUB_1_DMU_1_TXE
24	UART1_RX	78	TIM15_CH1	132	CAN_SUB_1_DMU_1_RX0
25	UART1_TX	79	TIM15_UP	1	CAN_SUB_1_DMU_1_RX1
26	UART2_RX	80	TIM15_TRIG	34	CAN_SUB_1_DMU_2_TX
27	UART2_TX	81	TIM15_COM	35	CAN_SUB_1_DMU_2_TXE

DMA request MUX input	Resource	DMA request MUX input	Resourcer	DMA request MUX input	Resource
28	UART3_RX	82	TIM16_CH1	36	CAN_SUB_1_DMU_2_RX0
29	UART3_TX	83	TIM_16_UP	137	CAN_SUB_1_DMU_2_RX1
30	Reserved	84	TIM_TS_UP	138	CAN_SUB_1_DMU_3_TX
31	Reserved	85	Reserved	139	CAN_SUB_1_DMU_3_TXE
32	Reserved	86	Reserved	140	CAN_SUB_1_DMU_3_RX0
33	Reserved	87	Reserved	141	CAN_SUB_1_DMU_3_RX1
34	Reserved	88	Reserved	142	CAN_SUB_1_DMU_4_TX
35	Reserved	89	Reserved	143	CAN_SUB_1_DMU_4_TXE
36	ADC2	90	Reserved	144	CAN_SUB_1_DMU_4_RX0
37	ADC3	91	DAC1_CH1	145	CAN_SUB_1_DMU_4_RX1
38	ADC4	92	DAC1_CH2	146	CAN_SUB_1_M_CAN_1
39	ADC5	93	DAC2_CH1	147	CAN_SUB_1_M_CAN_2
40	Reserved	94	DAC2_CH2	148	Reserved
41	Reserved	95	HRTIM1_MASTER (hrtim_dma1)	149	Reserved
42	TIM1 CH1	96	HRTIM1 TIMA (hrtim_dma2)	150	SD ADC1
43	TIM1_CH2	97	HRTIM1_TIMB (hrtim_dma3)	151	SD_ADC2
44	TIM1_CH3	98	HRTIM1_TIMC (hrtim_dma4)	152	Reserved
45	TIM1 CH4	99	HRTIM1 TIMD (hrtim_dma5)	153	Reserved
46	TIM1_UP	100	HRTIM1_TIME (hrtim_dma6)	154	Reserved
47	TIM1_TRIG	101	HRTIM1 TIMF (hrtim_dma7)	155	Reserved
48	TIM1 COM	102	DAC3 CH1	156	Reserved
49	TIM8 CH1	103	DAC3 CH2	157	Reserved
50	TIM8_CH2	104	DAC4_CH1	158	Reserved
51	TIM8_CH3	105	DAC4_CH2	159	Reserved
52	TIM8_CH4	106	SPI4_RX	160	Reserved
53	TIM8_UP	107	SPI4_TX		
54	TIM8_TRIG	108	Reserved		

Since the DMAMUX may also be used as a DMA request generator from programmable events on its input trigger signals, the Table 2 shows the resources that can trigger a DMA requests generation by means of DMAMUX trigger inputs. For such purpose the SIG_ID field in the register DMAMUX_RGxCR has to be configured.

Table 2. **DMAMUX: assignment of trigger inputs to resources**

Trigger input	Resource	Trigger input	Resource
0	EXTI LINE0	16	DMAMUX1_ch0_event
1	EXTI LINE1	17	DMAMUX1_ch1_event
2	EXTI LINE2	18	DMAMUX1_ch2_event
3	EXTI LINE3	19	DMAMUX1_ch3_event
4	EXTI LINE4	20	Reserved
5	EXTI LINE5	21	Reserved
6	EXTI LINE6	22	Reserved
7	EXTI LINE7	23	Reserved
8	EXTI LINE8	24	Reserved
9	EXTI LINE9	25	Reserved
10	EXTILINE10	26	Reserved
11	EXTI LINE11	27	Reserved
12	EXTILINE12	28	Reserved
13	EXTILINE13	29	Reserved
14	EXTILINE14	30	Reserved
15	EXTILINE15	31	Reserved

The DMA request line can be selected by the DMAMUX channels synchronously with events from its DMAMUX synchronization inputs, the Table 3 shows the assignment of DMA synchronization input to resources.

Table 3. **DMAMUX: assignment of synchronization inputs to resources**

Sync input	Resource	Sync input	Resource
0	EXTI LINE0	16	DMAMUX1_ch0_event
1	EXTI LINE1	17	DMAMUX1_ch1_event
2	EXTI LINE2	18	DMAMUX1_ch2_event
3	EXTI LINE3	19	DMAMUX1_ch3_event
4	EXTI LINE4	20	Reserved
5	EXTI LINE5	21	Reserved
6	EXTI LINE6	22	Reserved

3.1.2 Stream priority

Each DMA port has an arbiter for handling the priority between other DMA streams. Stream priority is software-configurable (there are four software levels) by means of the DMA_SxCR register. If two or more DMA streams have the same software priority level, the hardware priority is used (stream 0 has priority over stream 1, etc.).

3.1.3 Source, destination, and transfer mode

A DMA transfer is defined by a source address and a destination address. Both source and destination transfers can address peripherals and memories in the entire 4-Gbyte area, at addresses comprised between 0x0000 0000 and 0xFFFF FFFF.

DMA is capable of performing three different transfer modes: memory-to-peripheral, peripheral-to-memory, or memory-to-memory transfers.

The direction is configured using the DIR[1:0] bits in the DMA_SxCR register.

Table 4. **Source and destination address**

Bits DIR[1:0] of the DMA_SxCR register	Direction	Source address	Destination address
00	Peripheral-to-memory	DMA_SxPAR	DMA_SxM0AR
01	Memory-to-peripheral	DMA_SxM0AR	DMA_SxPAR
10	Memory-to-memory	DMA_SxPAR	DMA_SxM0AR
11	reserved		

When the data width (programmed in the PSIZE or MSIZE bits in the DMA_SxCR register) is a half-word or a word, respectively, the peripheral or memory address written into the DMA_SxPAR or DMA_SxM0AR/M1AR registers has to be aligned on a word or half-word address boundary, respectively.

3.1.4 Transfer mode

DMA can perform three different transfer modes:

- Peripheral to memory
- Memory to peripheral
- Memory to memory

3.1.5 Transfer size

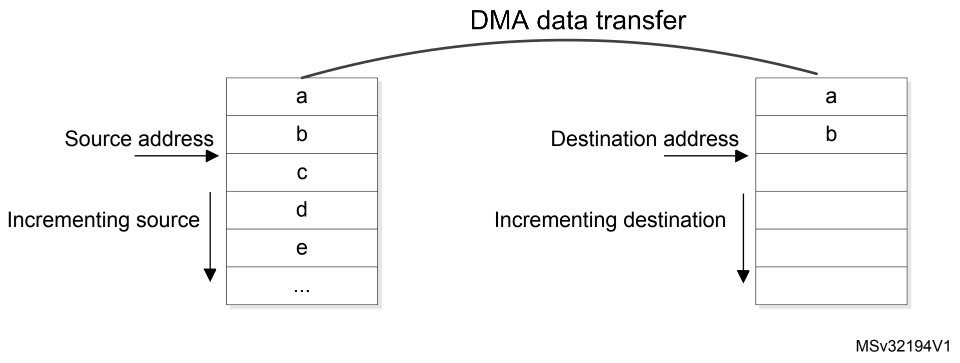
The transfer size defines the volume of data to be transferred from source to destination.

The transfer size is defined by the DMA_SxNDTR register value and by the peripheral side data width. Depending on the received request (burst or single), the transfer size value is decreased by the amount of the transferred data.

3.1.6 Incrementing source/destination address

After each data transfer, the DMA can be configured to increment automatically the source and/or destination address.

Figure 8. **DMA source address and destination address incrementing**



3.1.7 Source and destination data width

Data width for source and destination can be defined as:

- Byte (8 bits)
- Half-word (16 bits)
- Word (32 bits)

3.1.8 Transfer types

There are two transfer types:

- Circular mode: the circular mode is available to handle circular buffers and continuous data flows (the DMA_SxNDTR register is then reloaded automatically with the previously programmed value).
- Normal mode: once the DMA_SxNDTR register reaches zero, the stream is disabled (the EN bit in the DMA_SxCR register is then equal to 0).

3.1.9 DMA FIFO mode

Each stream has an independent 4-word (4 * 32 bits) FIFO and the threshold level is software-configurable between 1/4, 1/2, 3/4 or full. The FIFO is used to temporarily store data coming from the source before transmitting them to the destination.

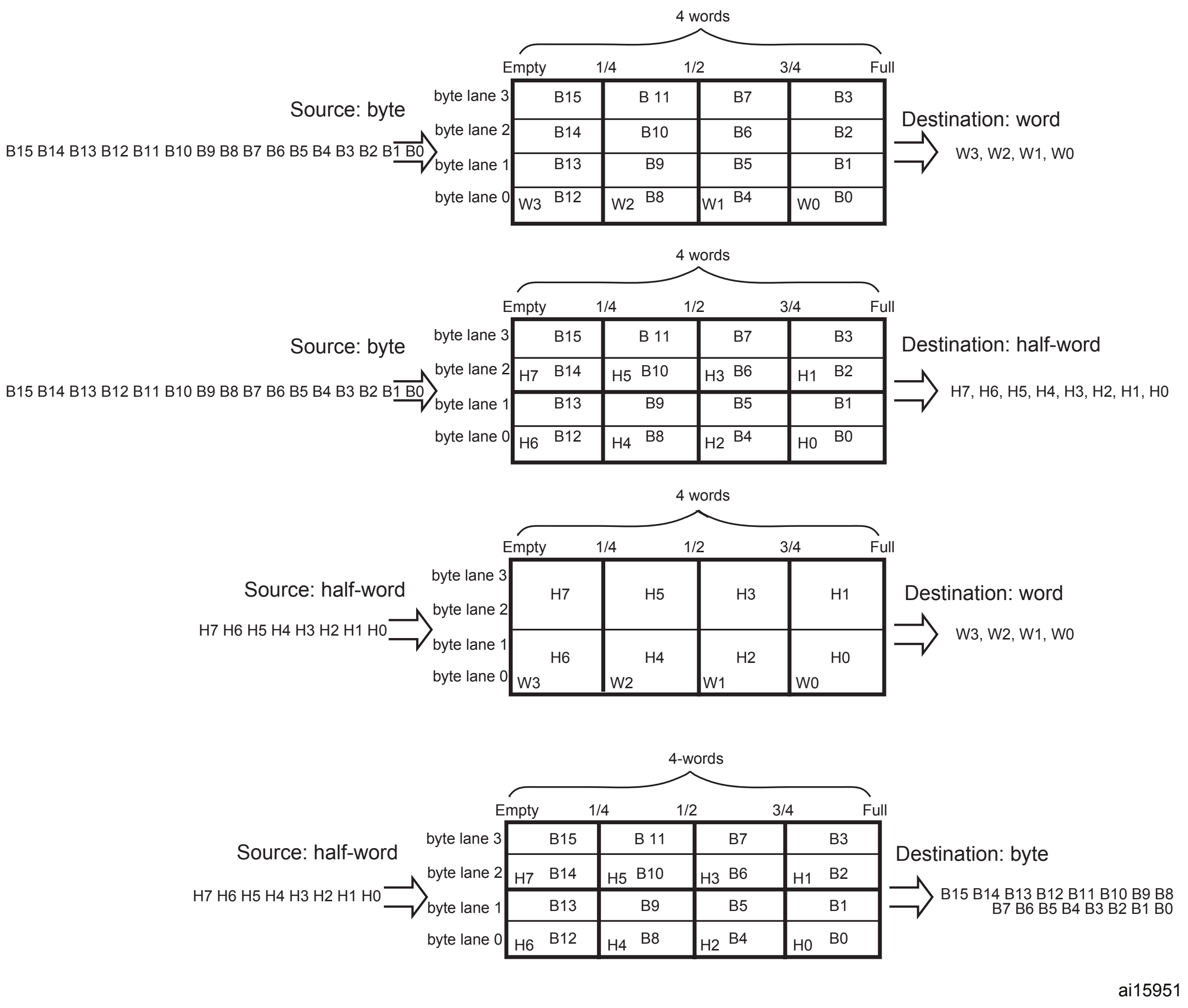
DMA FIFO can be enabled or disabled by software; when disabled, the direct mode is used. If DMA FIFO is enabled, data packing/unpacking and/or burst mode can be used. The configured DMA FIFO threshold defines the DMA memory port request time.

The DMA FIFOs implemented on SR5E1x devices help to:

- Reduce SRAM access and so give more time for the other masters to access the bus matrix without additional concurrency,
- Allow software to do burst transactions that optimize the transfer bandwidth,
- Allow packing/unpacking data to adapt source and destination data width with no extra DMA access.

The structure of the FIFO differs depending on the source and destination data widths, and is described in the Figure 9.

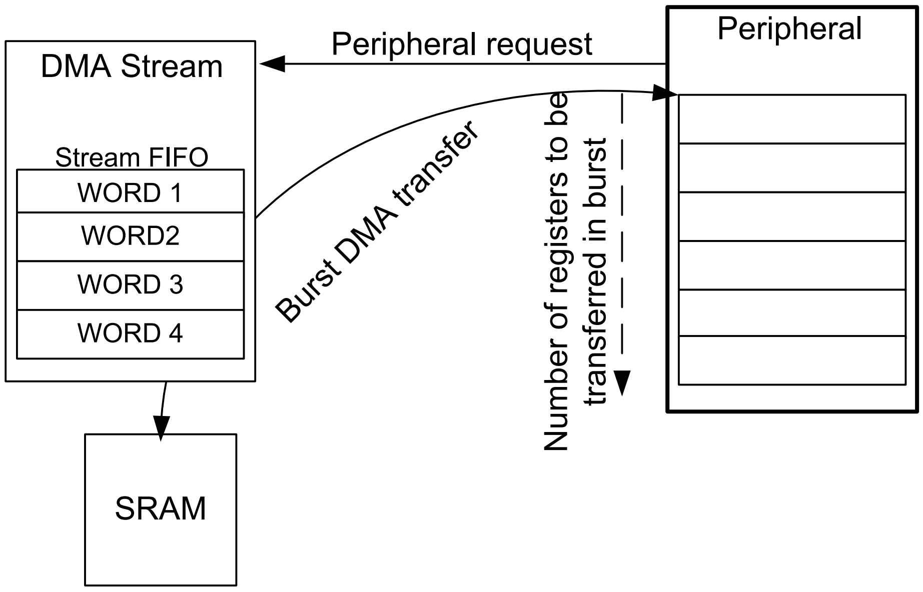
Figure 9. FIFO structure



3.1.10 Source and destination burst size

The implemented DMA FIFOs guarantees the burst transfers.

Figure 10. **DMA burst transfer**



In response to a burst request from the peripheral, DMA reads/writes the number of data units (data unit can be a word, a half-word, or a byte) programmed by the burst size (4x, 8x, or 16x data unit). The burst size on the DMA peripheral port must be set according to the peripheral needs/capabilities.

Caution is required when choosing the FIFO threshold (bits FTH[1:0] of the DMA_SxFCR register) and the size of the memory burst (MBURST[1:0] of the DMA_SxCR register): The content pointed by the FIFO threshold must exactly match an integer number of memory burst transfers. If this is not in the case, a FIFO error (flag FEIFx of the DMA_HISR or DMA_LISR register) is generated when the stream is enabled, then the stream is automatically disabled. The Table 5 shows the possible combinations of memory burst size, FIFO threshold configuration and data size.

Table 5. **Possible burst configurations**

MSIZE	FIFO level	MBURST = INCR4	MBURST = INCR8	MBURST = INCR16
Byte	1/4	1 burst of 4 bytes	Forbidden	Forbidden
	1/2	2 bursts of 4 bytes	1 burst of 8 bytes	
	3/4	3 bursts of 4 bytes	Forbidden	
	Full	4 bursts of 4 bytes	2 bursts of 8 bytes	1 burst of 16 bytes
Half-word	1/4	Forbidden	Forbidden	Forbidden
	1/2	1 burst of 4 half-words		
	3/4	Forbidden		
	Full	2 bursts of 4 half-words	1 burst of 8 half-word	
Word	1/4	Forbidden	Forbidden	
	1/2			
	3/4			
	Full	1 burst of 4 words		

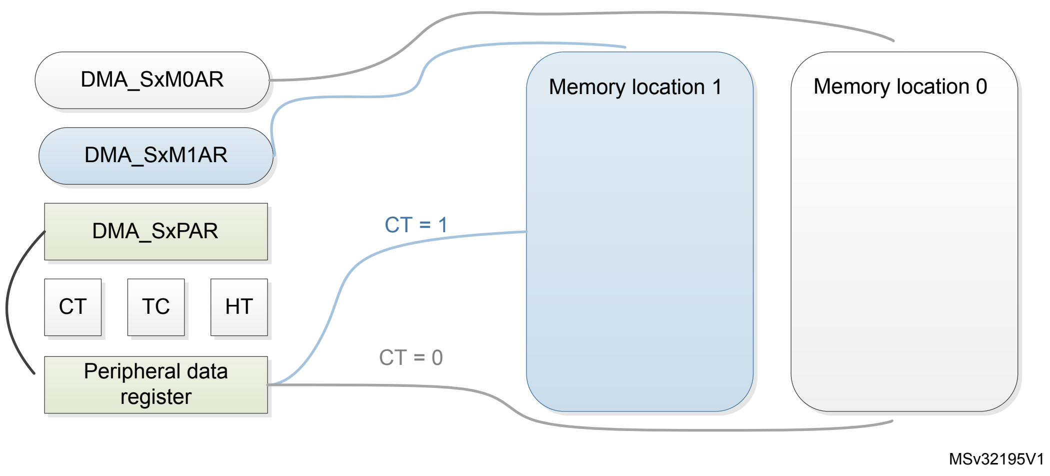
To ensure data coherence, each group of transfers that form a burst is indivisible: AHB transfers are locked and the arbiter of the AHB bus matrix does not remove the DMA master's access rights during the burst transfer sequence.

3.1.11 Double-buffer mode

A double-buffer stream works as a regular (single-buffer) stream, with the difference that it has two memory pointers. When the double-buffer mode is enabled, the circular mode is automatically enabled and at each end of the transaction (DMA_SxNDTR register reach 0), the memory pointers are swapped.

This allows the software to process one memory area while the second memory area is being filled/used by the DMA transfer.

Figure 11. Double-buffer mode



In double-buffer mode, it is possible to update the base address for the AHB memory port on-the-fly (DMA_SxM0AR or DMA_SxM1AR) when the stream is enabled:

- When the CT (current Target) bit in the DMA_SxCR register is equal to 0, the current DMA memory target is memory location 0, and so the base address memory location 1 (DMA_SxM1AR) can be updated.
- When the CT bit in the DMA_SxCR register is equal to 1, the current DMA memory target is memory location 1, and so the base address memory location 0 (DMA_SxM0AR) can be updated.

3.1.12 Flow control

The flow controller is the unit that controls the data transfer length and is responsible for stopping the DMA transfer.

The flow controller can be either the DMA or the peripheral:

- With DMA as flow controller:
  In this case, it is necessary to define the transfer size value in the DMA_SxNDTR register before enabling the associated DMA stream. When a DMA request is served, the transfer size value decreases by the amount of transferred data (depending on the type of request: burst or single).
  When the transfer size value reaches 0, the DMA transfer is finished and the DMA stream is disabled.
- With the peripheral as flow controller:
  SR5E1 has no peripheral to use the peripheral flow controller mode.

3.2 Setting up a DMA transfer

To configure DMA stream x (where x is the stream number), the following procedure should be applied:

1. If the stream is enabled, disable it by resetting the EN bit in the DMA_SxCR register, then read this bit in order to confirm that there is no ongoing stream operation. Writing this bit to zero is not immediately effective since it is actually written to 0 once all the current transfers are finished. When the EN bit is read as 0, this means that the stream is ready to be configured. It is therefore necessary to wait for the EN bit to be cleared before starting any stream configuration. All the stream dedicated bits set in the status register (DMA_LISR and DMA_HISR) from the previous data block DMA transfer must be cleared before the stream can be reenabled.
2. Set the peripheral port register address in the DMA_SxPAR register. The data is moved from/to this address to/from the peripheral port after the peripheral event.
3. Set the memory address in the DMA_SxMA0R register (and in the DMA_SxMA1R register in the case of a double-buffer mode). The data is written to or read from this memory after the peripheral event.
4. Configure the total number of data items to be transferred in the DMA_SxNDTR register. After each peripheral event or each beat of the burst, this value is decremented.
5. Use DMAMUX1 to route a DMA request line to the DMA channel.
6. If the peripheral is intended to be the flow controller and if it supports this feature, set the PFCTRL bit in the DMA_SxCR register.
7. Configure the stream priority using the PL[1:0] bits in the DMA_SxCR register.
8. Configure the FIFO usage (enable or disable, threshold in transmission and reception).
9. Configure the data transfer direction, peripheral and memory incremented/fixed mode, single or burst transactions, peripheral and memory data widths, circular mode, double-buffer mode, and interrupts after half and/or full transfer and/or errors in the DMA_SxCR register.
10. Activate the stream by setting the EN bit in the DMA_SxCR register. As soon as the stream is enabled, it can serve any DMA request from the peripheral connected to the stream.

Revision history

Table 6. Document revision history

Date	Revision	Changes
15-Jun-2023	1	Initial release.

Contents

List of tables

List of figures