USAGE GUIDE FOR SIM3U1XX, SIM3C1XX, AND SIM3L1XX DMA AND DTM MODULES

1. Introduction

The Direct Memory Access (DMA) (SiM3U1xx, SiM3C1xx, and SiM3L1xx) and Data Transfer Manager (DTM) (SiM3L1xx) modules are complex data management modules intended to autonomously transfer data between peripherals and memory. These modules can save system power consumption by allowing the core to enter a low power state or process data in parallel to improve performance. This document discusses usage models for these modules.

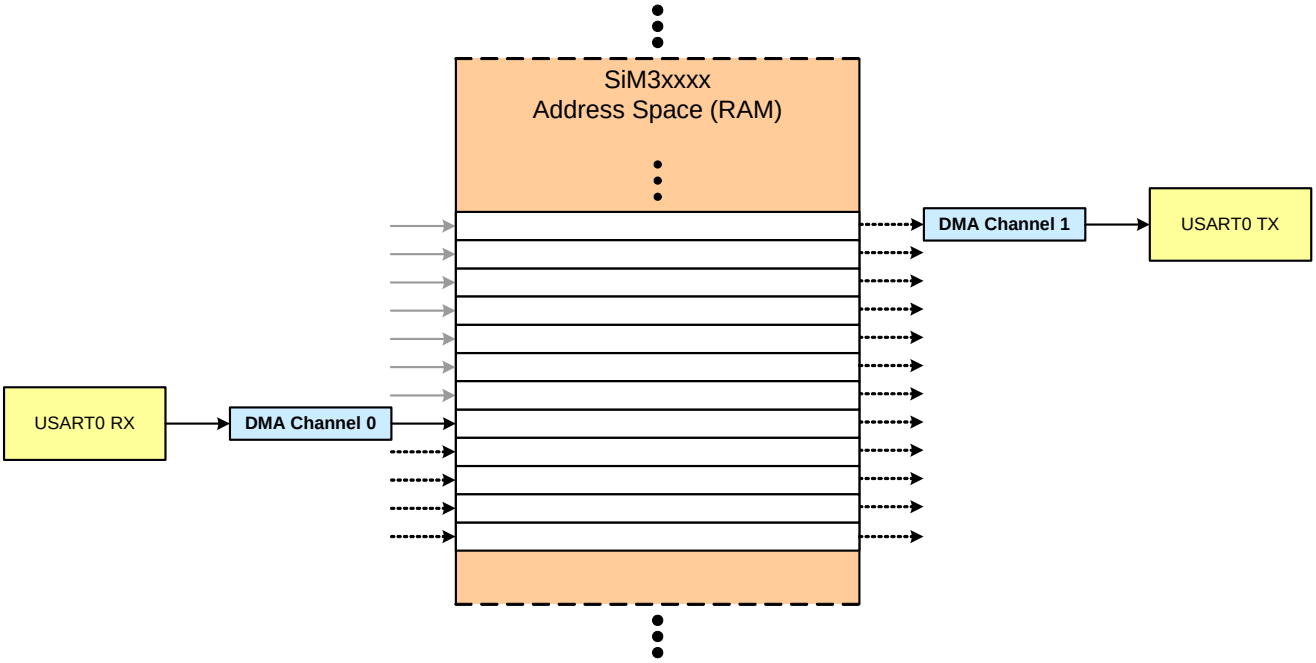


Figure 1. Transferring Data using the DMA and DTM modules

2. Key Points

This key topics of this document are:

- Using the DMA in a basic memory-to-memory data transfer
- Using the DMA from a peripheral-to-memory data transfer
- Using the DMA from memory to a peripheral data transfer
- Complex multi-channel DMA transfers using the Data Transfer Manager (SiM3L1xx devices only)

3. Relevant Documentation

Precision32 Application Notes are listed on the following website: www.silabs.com/32bit-appnotes.

- AN725: Advanced Low Power Techniques for SiM3L1xx Devices
- AN667: Getting Started with the Silicon Labs Precision32 IDE
- AN670: Integrating Silicon Labs SiM3xxxx Devices into the Keil µVision IDE

4. DMA Overview

The DMA consists of two modules: DMA controller (DMACTRL) and DMA peripheral crossbar (DMAXBAR). The controller provides a single access point for all 16 (SiM3U1xx and SiM3C1xx) or 10 (SiM3L1xx) DMA channels and the global DMA controls. The controller is also responsible for handling arbitration between channels. The DMA peripheral crossbar assigns channels to a peripheral. When assigned and properly configured, the peripheral's data request signal will trigger a DMA channel to transfer data. Figure 2 shows a block diagram of the DMA controller and DMA peripheral crossbar.

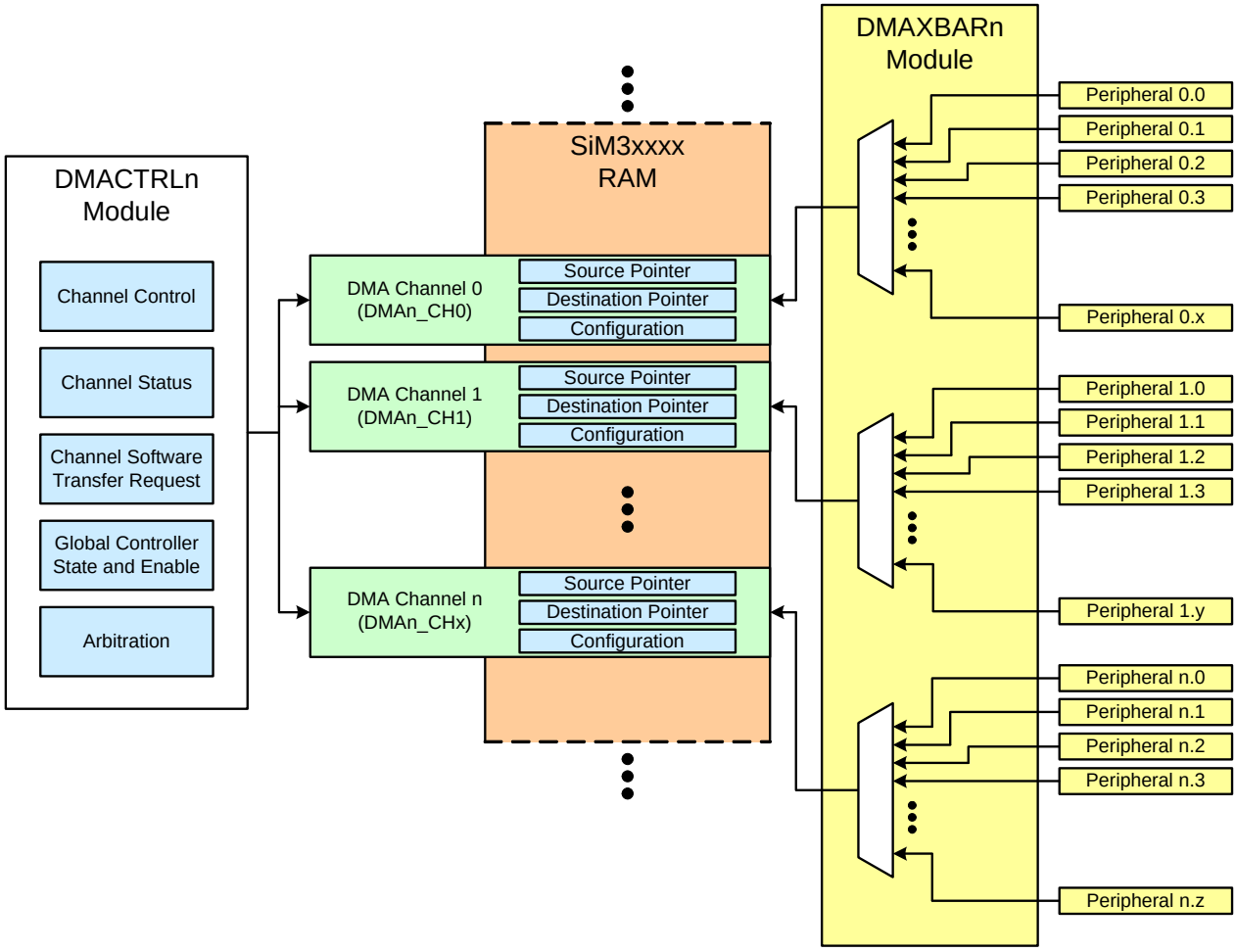


Figure 2. DMACTRL and DMACH Block Diagram

The channels have controls and flags in the DMACTRL registers. In addition, each channel has one or more transfer descriptors stored in memory that describe the data transfer in detail. Each channel can have primary, alternate, or scatter-gather descriptors. The BASEPTR and ABASEPTR registers in the controller point to the starting address of these descriptors in memory. Firmware sets the BASEPTR field, and the controller hardware automatically sets the ABASEPTR field based on the number of channels implemented in the module.

Each channel has separate enables, alternate enables, masks, software requests, programmable priority, and status flags. The channels operate independently, but have a fixed arbitration order.

The STATE field reports the current status of the DMA controller, and the DMAENS bit indicates whether the global DMA enable is set.

4.1. DMA Channel Transfer Descriptors

Each channel has transfer descriptors stored in memory that describe the data transfer in detail. Each descriptor is composed of four 32-bit words in memory organized as follows:

1. **Source End Pointer (word 1)**: The address of the last source data in the transfer.
2. **Destination End Pointer (word 2)**: The last destination address of the transfer.
3. **Channel Configuration (word 3)**: Configuration details for the transfer.
4. **Alignment padding (word 4)**: Not used by the DMA controller. Firmware may use this word for any purpose.

Each channel can have primary, alternate, and scatter-gather descriptors. The primary and alternate descriptors are organized in contiguous blocks in memory for each of the channels. The spacing for these descriptors is fixed, so any unused channels must still be accounted for when placing descriptors in memory. The primary descriptors must be placed at the start of an address block sized for both the primary and alternate descriptors. For SiM3U1xx, SiM3C1xx, and SiM3L1xx devices that implement 16 or 10 DMA channels, the BASEPTR points to the start of the primary descriptors and is 23 bits wide. The valid addresses for the BASEPTR field are multiples of 256 (0x0000_0100), and the required memory for all primary and alternate descriptors for 16 channels is 512 bytes. The scatter-gather descriptors are more flexible and can appear anywhere in memory.

Channel 0's primary descriptor begins at address offset 0x0000, Channel 1's primary descriptor starts at offset 0x0010, and so on. The alternate descriptors begin at the next memory block (256 bytes), regardless of whether or not the primary descriptors for the channels are in use.

Firmware originally sets the channel configuration descriptor; the DMA controller will modify this word as the transfer progresses, so firmware should not write to this descriptor until any active transfers for the channel are complete.

Figure 3 shows the fixed memory configuration for the descriptors.

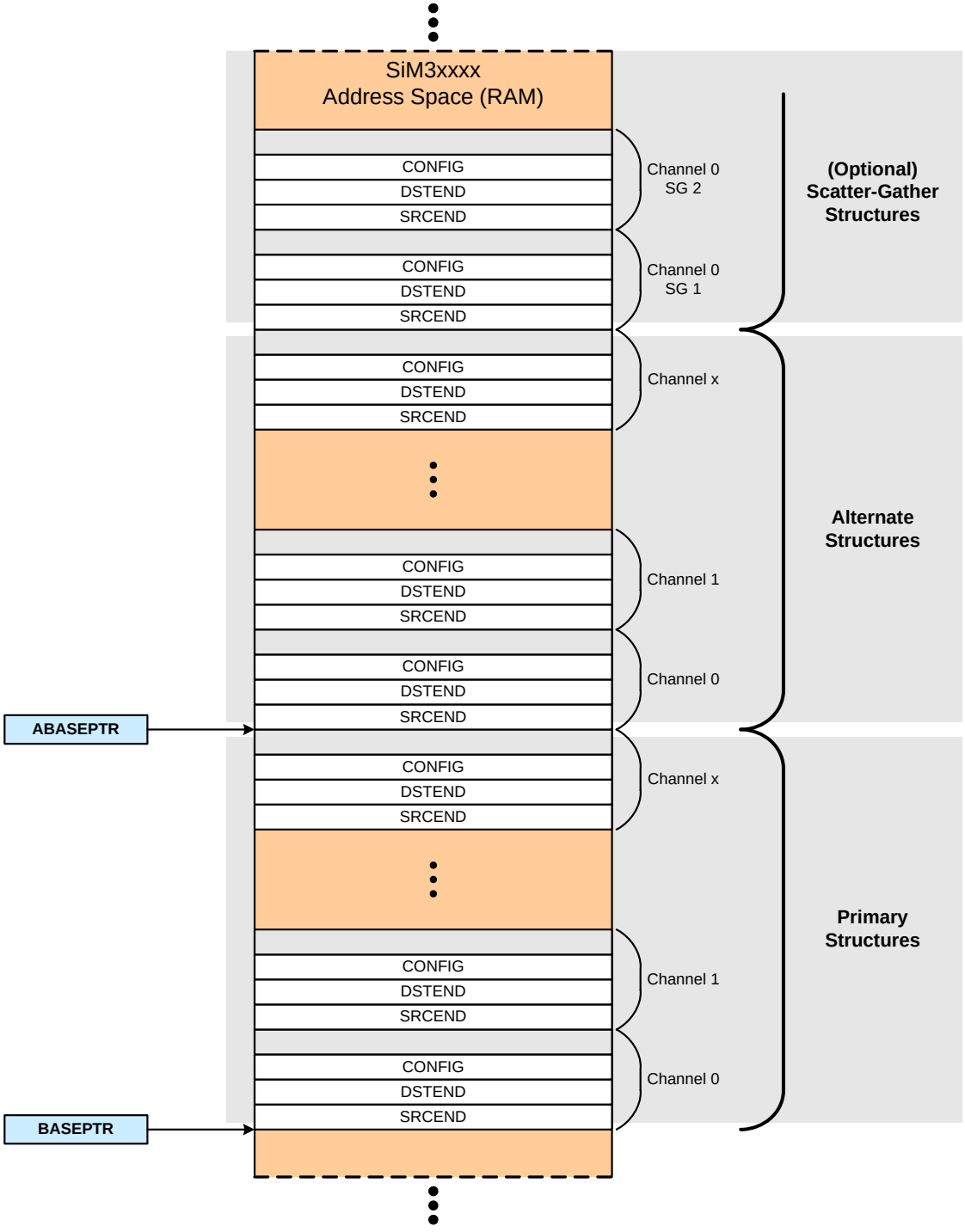


Figure 3. Channel Transfer Descriptor Memory Configuration

4.1.1. Channel Transfer Descriptors

Table 1, Table 2, and Table 3 describe the source end pointer, destination pointer, and configuration descriptors for the primary, alternate, and scatter-gather DMA channel descriptors.

Table 1. DMA0_CHx_SRCEND: Source End Pointer

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	SRCEND[31:16]															
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	SRCEND[15:0]															

Address in Channel Transfer Descriptor: 0x0000

Bit	Name	Function
31:0	SRCEND	**Source End Pointer.** This field is the address of the last source data in the DMA transfer.

Table 2. DMA0_CHx_DSTEND: Destination End Pointer

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	DSTEND[31:16]															
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	DSTEND[15:0]															

Address in Channel Transfer Descriptor: 0x0004

Bit	Name	Function
31:0	DSTEND	**Destination End Pointer.** This field is the last destination address of the DMA transfer.

Table 3. DMA0_CHx_CONFIG: Channel Configuration

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	DSTAIMD		DSTSIZE		SRCAIMD		SRCSIZE		Reserved						RPOWER[3:2]	
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	RPOWER[1:0]		NCOUNT										Reserved	TMD		

Address in Channel Transfer Descriptor: 0x0008

Bit	Name	Function
31:30	DSTAIMD	**Destination Address Increment Mode.** This field must be set to a value that's equal to or greater than the DSTSIZE setting. 00: The destination address increments by one byte after each data transfer. 01: The destination address increments by one half-word after each data transfer. 10: The destination address increments by one word after each data transfer. 11: The destination address does not increment.
29:28	DSTSIZE	**Destination Data Size Select.** The destination size (DSTSIZE) must equal the source size (SRCSIZE). 00: Each DMA destination data transfer writes a byte. 01: Each DMA destination data transfer writes a half-word. 10: Each DMA destination data transfer writes a word. 11: Reserved.
27:26	SRCAIMD	**Source Address Increment Mode.** This field must be set to a value that's equal to or greater than the SRCSIZE setting. 00: The source address increments by one byte after each data transfer. 01: The source address increments by one half-word after each data transfer. 10: The source address increments by one word after each data transfer. 11: The source address does not increment.
25:24	SRCSIZE	**Source Data Size Select.** The destination size (DSTSIZE) must equal the source size (SRCSIZE). 00: Each DMA source data transfer reads a byte. 01: Each DMA source data transfer reads a half-word. 10: Each DMA source data transfer reads a word. 11: Reserved.
23:18	Reserved	Must write 0 to this field.

Bit	Name	Function
17:14	RPOWER	**Transfer Size Select.** This field determines the number of data transfers between each DMA channel re-arbitration. The number of data transfers is given by: $\text{Number of Transfers} = 2^{\text{RPOWER}}$ This field is ignored for peripherals that support single data requests only. A value of 0 for RPOWER should be used for channels interfacing with these types of peripherals.
13:4	NCOUNT	**Transfer Total.** This field is the total number of transfers for the DMA channel. The total number is NCOUNT + 1, so software requiring a total of 4 transfers would set the NCOUNT field to 3. The DMA controller decrements this field as transfers are made.
3	Reserved	Must write 0 to this bit.
2:0	TMD	**Transfer Mode.** 000: Stop the DMA channel. 001: Use the Basic transfer type (single descriptor only). 010: Use the Auto-Request transfer type (single descriptor only). 011: Use the Ping-Pong transfer type (primary and alternate descriptors). 100: Use the Memory Scatter-Gather Primary transfer type (primary, alternate, and scattered descriptors). 101: Use the Memory Scatter-Gather Alternate transfer type (primary, alternate, and scattered descriptors). 110: Use the Peripheral Scatter-Gather Primary transfer type (primary, alternate, and scattered descriptors). 111: Use the Peripheral Scatter-Gather Alternate transfer type (primary, alternate, and scattered descriptors).

4.2. DMA Peripheral Crossbar

Peripherals are assigned to various channels, and the DMA Crossbar can be used to assign a channel to a particular peripheral. These assignments are shown in Table 4.

Table 4. DMA Crossbar Channel Peripheral Assignments for SiM3U1xx Devices

Peripheral	DMA Channel 0	DMA Channel 1	DMA Channel 2	DMA Channel 3	DMA Channel 4	DMA Channel 5	DMA Channel 6	DMA Channel 7	DMA Channel 8	DMA Channel 9	DMA Channel 10	DMA Channel 11	DMA Channel 12	DMA Channel 13	DMA Channel 14	DMA Channel 15
AES0 RX							✓					✓				
AES0 TX						✓					✓					
AES0 XOR								✓					✓			
DMAXT0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DMAXT1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EPCA0 Capture				✓	✓				✓	✓						
EPCA0 Control		✓	✓												✓	✓
I2C0 RX		✓					✓					✓				
I2C0 TX	✓									✓						
I^2S RX					✓	✓					✓	✓				
I^2S TX			✓	✓									✓	✓		
IDAC0				✓			✓								✓	✓
IDAC1		✓	✓										✓	✓		
SARADC0			✓		✓	✓										
SARADC1				✓							✓					✓
SPI0 RX		✓												✓		
SPI0 TX			✓												✓	
SPI1 RX	✓								✓							
SPI1 TX					✓			✓					✓			
TIMER0L Overflow	✓	✓						✓					✓		✓	
TIMER0H Overflow	✓				✓		✓			✓		✓		✓		✓
TIMER1L Overflow	✓	✓						✓					✓		✓	
TIMER1H Overflow	✓	✓		✓				✓			✓		✓			✓

Table 4. DMA Crossbar Channel Peripheral Assignments for SiM3U1xx Devices

Peripheral	DMA Channel 0	DMA Channel 1	DMA Channel 2	DMA Channel 3	DMA Channel 4	DMA Channel 5	DMA Channel 6	DMA Channel 7	DMA Channel 8	DMA Channel 9	DMA Channel 10	DMA Channel 11	DMA Channel 12	DMA Channel 13	DMA Channel 14	DMA Channel 15
USART0 RX	✓						✓		✓			✓		✓		
USART0 TX			✓		✓			✓							✓	
USART1 RX		✓							✓			✓				
USART1 TX						✓				✓			✓			
USB0 EP1 IN								✓								✓
USB0 EP2 IN							✓								✓	
USB0 EP3 IN						✓								✓		
USB0 EP4 IN					✓								✓			
USB0 EP1 OUT				✓								✓				
USB0 EP2 OUT			✓								✓					
USB0 EP3 OUT		✓								✓						
USB0 EP4 OUT	✓								✓							
Software Trigger	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5. DMA Crossbar Channel Peripheral Assignments for SiM3C1xx Devices

Peripheral	DMA Channel 0	DMA Channel 1	DMA Channel 2	DMA Channel 3	DMA Channel 4	DMA Channel 5	DMA Channel 6	DMA Channel 7	DMA Channel 8	DMA Channel 9	DMA Channel 10	DMA Channel 11	DMA Channel 12	DMA Channel 13	DMA Channel 14	DMA Channel 15
AES0 RX							✓					✓				
AES0 TX						✓					✓					
AES0 XOR								✓					✓			
DMAXT0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DMAXT1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EPCA0 Capture				✓	✓				✓	✓						
EPCA0 Control		✓	✓												✓	✓

Table 5. DMA Crossbar Channel Peripheral Assignments for SiM3C1xx Devices (Continued)

Peripheral	DMA Channel 0	DMA Channel 1	DMA Channel 2	DMA Channel 3	DMA Channel 4	DMA Channel 5	DMA Channel 6	DMA Channel 7	DMA Channel 8	DMA Channel 9	DMA Channel 10	DMA Channel 11	DMA Channel 12	DMA Channel 13	DMA Channel 14	DMA Channel 15
I2C0 RX		✓					✓					✓				
I2C0 TX	✓									✓						
I^2S RX					✓	✓					✓	✓				
I^2S TX			✓	✓									✓	✓		
IDAC0				✓			✓								✓	✓
IDAC1		✓	✓										✓	✓		
SARADC0			✓		✓	✓										
SARADC1				✓							✓					✓
SPI0 RX		✓												✓		
SPI0 TX			✓												✓	
SPI1 RX	✓								✓							
SPI1 TX					✓			✓					✓			
TIMER0L Overflow	✓	✓						✓					✓		✓	
TIMER0H Overflow	✓				✓		✓			✓		✓		✓		✓
TIMER1L Overflow	✓	✓						✓					✓		✓	
TIMER1H Overflow	✓	✓		✓				✓			✓		✓			✓
USART0 RX	✓						✓		✓			✓		✓		
USART0 TX			✓		✓			✓							✓	
USART1 RX		✓							✓			✓				
USART1 TX						✓				✓			✓			
Software Trigger	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 6. DMA Crossbar Channel Peripheral Assignments for SiM3L1xx Devices

Peripheral	DMA Channel 0	DMA Channel 1	DMA Channel 2	DMA Channel 3	DMA Channel 4	DMA Channel 5	DMA Channel 6	DMA Channel 7	DMA Channel 8	DMA Channel 9
DTM0 A	✓									
DTM0 B		✓								
DTM0 C			✓							
DTM0 D				✓						
DTM1 A					✓					
DTM1 B						✓				
DTM1 C							✓			
DTM1 D								✓		
DTM2 A			✓				✓			
DTM2 B				✓				✓		
DTM2 C					✓				✓	
DTM2 D						✓				✓
SPI0 TX	✓				✓				✓	
SPI0 RX		✓				✓				✓
ENCDEC0 TX			✓				✓			
ENCDEC0 RX				✓				✓		
AES0 TX	✓				✓					
AES0 RX		✓				✓				
AES0 XOR			✓				✓			
SPI1 TX				✓				✓		
SPI1RX					✓				✓	
USART0 TX		✓		✓			✓		✓	
USART0 RX	✓		✓			✓		✓		
I2C0 RX	✓		✓	✓		✓	✓		✓	✓
I2C0 TX	✓			✓			✓			✓
SARADC0		✓			✓		✓		✓	

Table 6. DMA Crossbar Channel Peripheral Assignments for SiM3L1xx Devices (Continued)

Peripheral	DMA Channel 0	DMA Channel 1	DMA Channel 2	DMA Channel 3	DMA Channel 4	DMA Channel 5	DMA Channel 6	DMA Channel 7	DMA Channel 8	DMA Channel 9
IDAC0			✓			✓		✓		✓
EPCA0 Capture	✓	✓			✓				✓	✓
EPCA0 Control		✓			✓	✓				✓
TIMER0L Overflow	✓		✓		✓		✓		✓	
TIMER0H Overflow	✓		✓		✓		✓		✓	
TIMER1L Overflow		✓		✓		✓		✓		✓
TIMER1H Overflow		✓		✓		✓		✓		✓
DMAXT0	✓		✓		✓		✓		✓	
DMAXT1		✓		✓		✓		✓		✓
Software Trigger	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

4.3. Transfer Types

The DMA channels support five transfer types: basic, auto-request, ping-pong, memory scatter-gather, and peripheral scatter-gather. Table 7 shows the memory requirements for each transfer type.

Table 7. Transfer Memory Requirements

Transfer Type	Transfer Descriptors Required			Maximum Memory (RAM) Required (bytes)		Address Offsets (Primary / Alternate)
	Primary	Alternate	Scatter-Gather	10 Channels Implemented	16 Channels Implemented	10 or 16 Channels Implemented
Basic	✓			160	256	0 / —
Auto-Request	✓			160	256	0 / —
Ping-Pong	✓	✓		320	512	0 / 256
Memory Scatter-Gather	✓	✓	✓	320 + SG	512 + SG	0 / 256 + SG
Peripheral Scatter-Gather	✓	✓	✓	320 + SG	512 + SG	0 / 256 + SG

4.3.1. Basic Transfers

The basic transfer type uses only one descriptor (primary or alternate). In this mode, the channel will make NCOUNT + 1 data moves in 2^{RPOWER} bursts. Each data request moves one 2^{RPOWER} set of data. The number of requests required for a transfer is:

$$\text{Number of Requests} = \frac{NCOUNT + 1}{2^{RPOWER}}$$

Equation 1. Number of Requests for Basic Transfers

Any data remaining can be transferred by firmware or use an extra DMA data request.

After the final data transfer:

1. The DMA channel will write the primary descriptor TMD field with 0.
2. The primary descriptor NCOUNT field will contain 0.
3. The controller automatically disables the channel (the channel bit in CHENSET will read 0).

Figure 4 illustrates the DMA memory descriptors for a basic transfer.

This transfer type is recommended for peripheral-to-memory or memory-to-peripheral transfers.

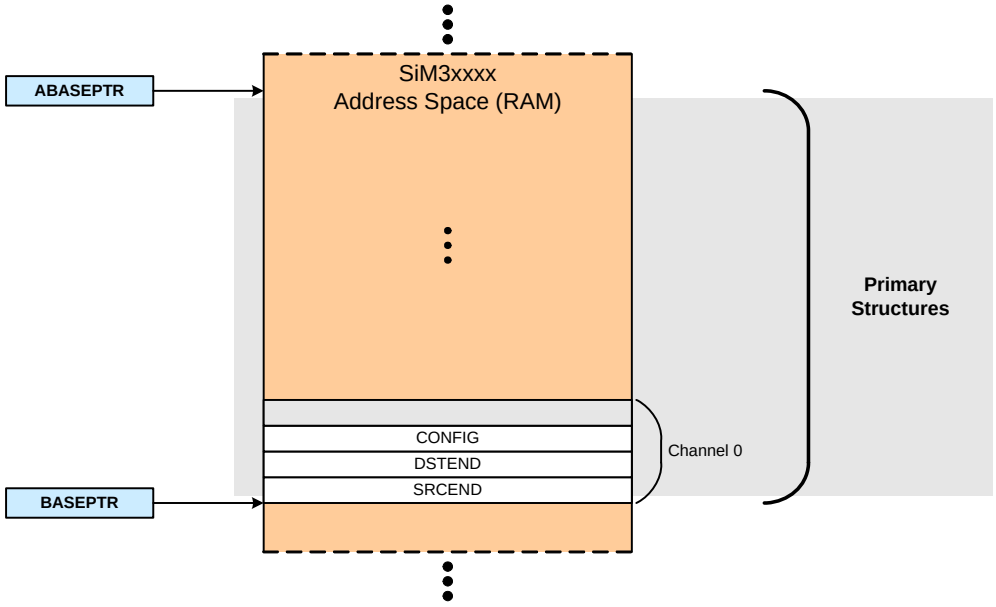


Figure 4. Basic and Auto-Request Transfer Memory Configuration

4.3.2. Auto-Request Transfers

Auto-request transfers use only one descriptor (primary or alternate). This transfer type only requires one data request to transfer all of the data. The controller will arbitrate as normal (every 2^{RPOWER} transfers), and a channel interrupt will occur when the transfer completes. This transfer type is recommended for memory-to-memory transfers.

After the final data transfer:

1. The DMA channel will write the primary descriptor TMD field with 0.
2. The primary descriptor NCOUNT field will contain 0.
3. The controller automatically disables the channel (the channel bit in CHENSET will read 0).

The auto-request memory configuration is identical to the basic transfer shown in Figure 4.

4.3.3. Ping-Pong Transfers

Ping-pong transfers use both the primary and alternate channel descriptors. When the channel completes the transfer described by the first descriptor, it clears the TMD field in the original descriptor to 0 and toggles to point to the other descriptor. A channel interrupt will occur to allow firmware to update the completed transfer's descriptor, as the ping-pong operation will stop without intervention.

As with basic transfers, each 2^{RPOWER} data moves require a new data request. The number of requests is given by Equation 1.

Figure 5 shows an example where a channel's primary descriptor has an RPOWER of 1 with an NCOUNT of 3 and the alternate descriptor has an RPOWER of 0 with an NCOUNT of 4. These descriptors are both configured to move words (DSTSIZE and SRCSIZE set to 2) in ping-pong mode (TMD = 3).

Figure 6 illustrates the ping-pong memory configuration.

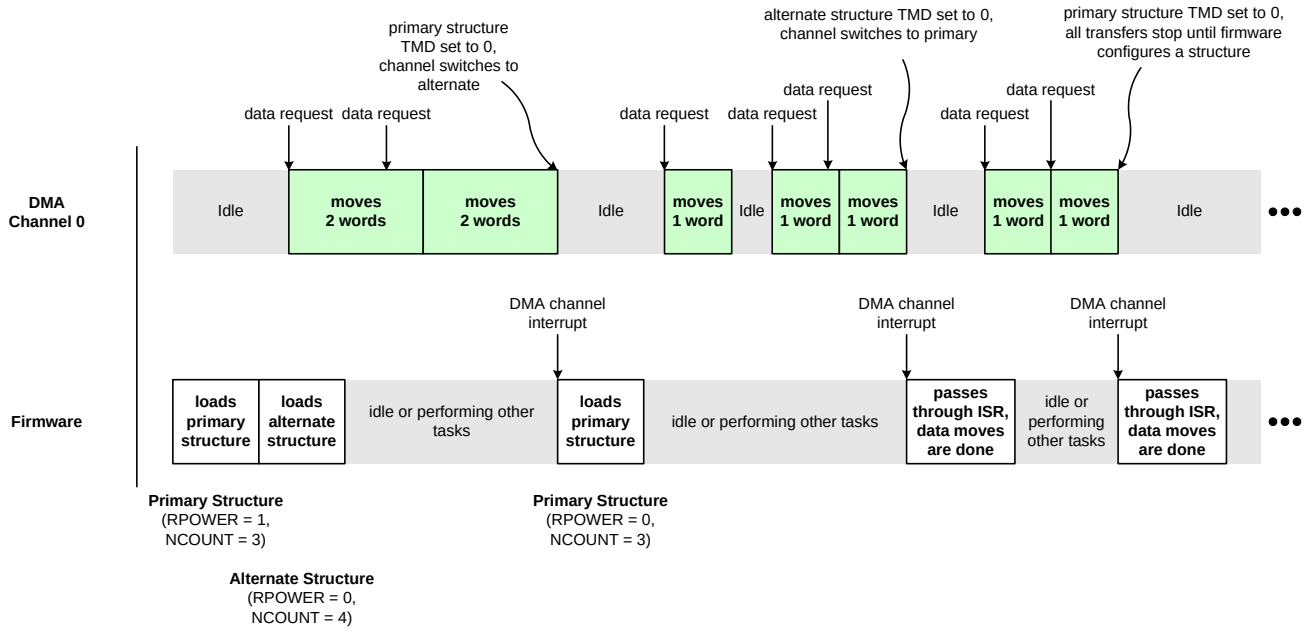


Figure 5. Ping-Pong Transfer Example

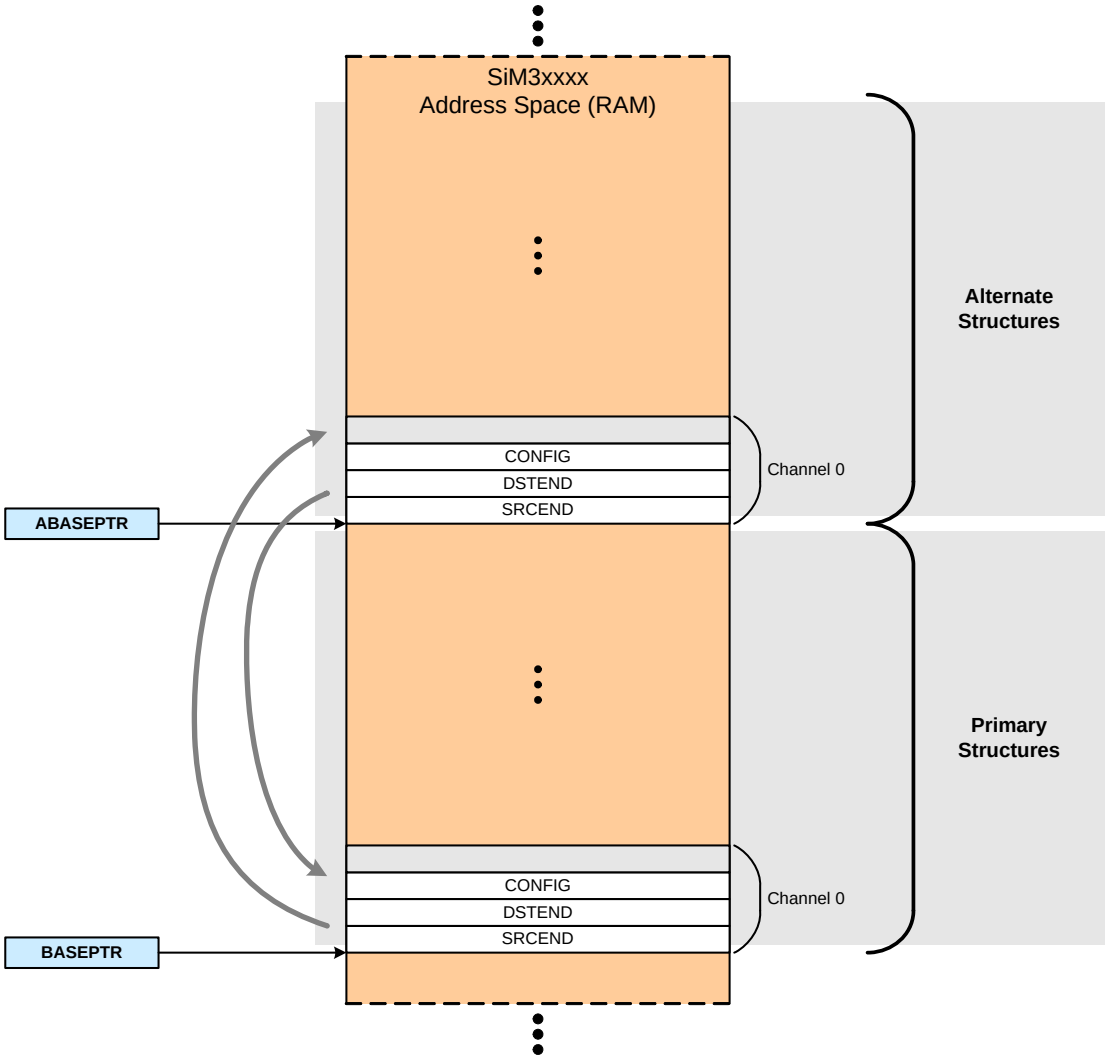


Figure 6. Ping-Pong Transfer Memory Configuration

4.3.4. Memory Scatter-Gather Transfers

The memory scatter-gather transfer uses primary, alternate, and scatter-gather descriptors. This transfer type allows a DMA channel to be set for multiple transfers at once without core intervention at the price of extra memory for the scatter-gather descriptors.

The primary descriptor in this mode contains the number and location of the scatter-gather descriptors. The primary descriptor should be programmed as follows:

1. Memory scatter-gather primary mode (TMD = 4).
2. RPOWER = 2.
3. NCOUNT set to the value specified by Equation 2.
4. SRCEND is set to the location of the last word of all the scatter-gather descriptors.
5. DSTEND is set to the location of the last word in the single alternate descriptor.

$$NCOUNT = (\text{Number of SG Structures} \times 4) - 1$$

Equation 2. NCOUNT Value for Scatter-Gather Transfers

The scatter-gather descriptors must be stacked contiguously in memory. The channel will copy the scatter-gather descriptors into the alternate descriptor location and execute them one by one. The scatter-gather descriptors should be programmed to memory scatter-gather alternate mode (TMD = 5), except for the last descriptor, which should use the auto-request transfer type (TMD = 2).

Once started, the DMA channel execution process is as follows:

1. Copy scatter-gather 1 (SG1) to the alternate descriptor.
2. Jump to the alternate descriptor and execute.
3. Jump back to the primary descriptor.
4. Copy scatter-gather 2 (SG2) to the alternate descriptor.
5. Jump to the alternate descriptor and execute.
6. Jump back to the primary descriptor.

The channel will continue in this pattern until the channel encounters a scatter-gather descriptor set to a basic or auto-request transfer.

Only one data request is required to execute all of the scattered transactions. The channel interrupt will occur once the last scatter-gather descriptor (programmed to a basic transfer) executes, if enabled. Arbitration occurs every 2^{RPOWER} of the scatter-gather descriptors.

Figure 7 shows the memory scatter-gather memory configuration.

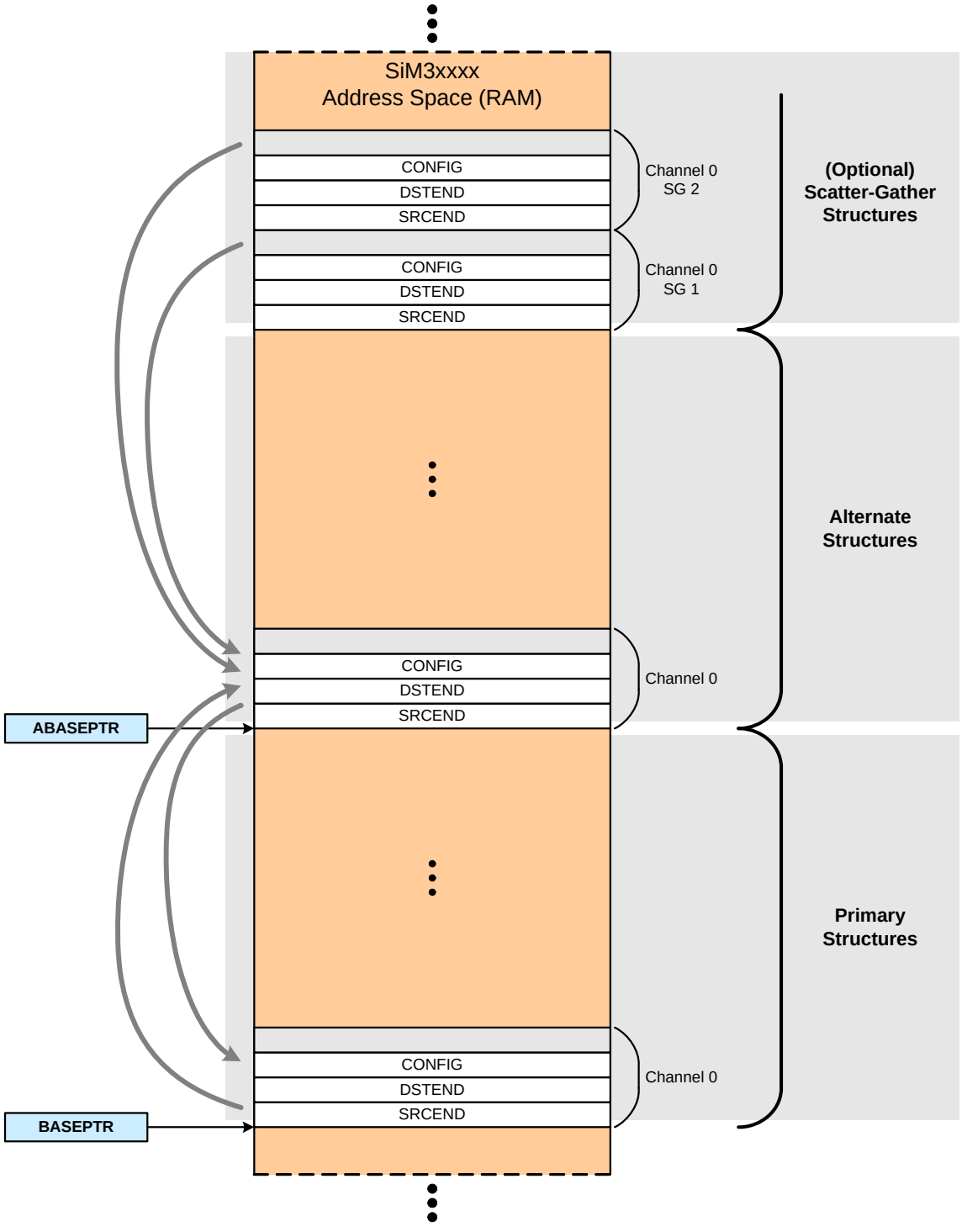


Figure 7. Memory and Peripheral Scatter-Gather Transfer Memory Configuration

4.3.5. Peripheral Scatter-Gather Transfers

The peripheral scatter-gather transfer is very similar to the memory scatter-gather transfer and uses primary, alternate, and scatter-gather descriptors. This transfer type allows a DMA channel to be set for multiple transfers at once without core intervention at the price of extra memory for the scatter-gather descriptors. A data request is required for each 2^{RPOWER} data move of the scatter-gather descriptor tasks. The RPOWER value can be different for each scatter-gather task. Equation 1 describes the total number of data requests required to complete a transfer.

The primary descriptor in this mode contains the number and location of the scatter-gather descriptors. The primary descriptor should be programmed as follows:

1. Peripheral scatter-gather primary mode (TMD = 6).
2. RPOWER = 2.
3. NCOUNT set to the value specified by Equation 2.
4. SRCEND is set to the location of the last word of all the scatter-gather descriptors.
5. DSTEND is set to the location of the last word in the single alternate descriptor.

The scatter-gather descriptors must be stacked contiguously in memory. The channel will copy the scatter-gather descriptors into the alternate descriptor location and execute them one by one. The scatter-gather descriptors should be programmed to peripheral scatter-gather alternate mode (TMD = 7), except for the last descriptor, which should use the basic transfer type (TMD = 1).

Once started, the DMA channel execution process is as follows:

1. Copy scatter-gather 1 (SG1) to the alternate descriptor.
2. Jump to the alternate descriptor and execute.
3. Jump back to the primary descriptor.
4. Copy scatter-gather 2 (SG2) to the alternate descriptor.
5. Jump to the alternate descriptor and execute.
6. Jump back to the primary descriptor.

The channel will continue in this pattern until the channel encounters a scatter-gather descriptor set to a basic or auto-request transfer.

The channel interrupt will occur once the last scatter-gather descriptor (programmed to a basic transfer) executes, if enabled.

Figure 7 shows the peripheral scatter-gather memory configuration.

4.4. Data Requests

Each DMA channel has two data requests: single and burst. Peripherals can support single requests, burst requests, or both. If configured to use a DMA channel, peripherals request data as needed using the appropriate request type. Table 8 and Table 9 lists the supported requests for the supported triggers and peripherals.

The RPOWER field is only valid for peripherals that support burst requests. For peripherals that only support single requests, the RPOWER field is ignored and re-arbitration occurs after every single data move.

Table 8. Supported Trigger or Peripheral Data Requests for SiM3U1xx/SiM3C1xx Devices

Peripheral Module	Supported Request Types	Number of Data Transfers Per Data Request	RPOWER Setting	Data Size
AESn	burst only	4	2	word
EPCAn	burst only	1, 2, 4, or 8	0, 1, 2, or 3	word
I2Cn	single only	1	unused	word
I2Sn	burst only	1 or 4	0 or 2	word
IDACn	burst only	1	0	word
SARADCn	burst only	4	2	word
SPIn	burst only	1, 2, 4, or 8	0, 1, 2, or 3	byte
TIMERn overflow	burst only	any	any	byte, half-word, or word
USARTn	single only	1	unused	byte, half-word, or word
USBn	both	1, 2, 4, or 8	0, 1, 2, or 3	word
External Trigger	burst only	any	any	byte, half-word, or word
Software Trigger	burst only	any	any	byte, half-word, or word

Table 9. Supported Trigger or Peripheral Data Requests for SiM3L1xx Devices

Peripheral Module	Supported Request Types	Number of Data Transfers Per Data Request	RPOWER Setting	Data Size
AESn	burst only	4	2	word
EPCAn	burst only	1, 2, 4, or 8	0, 1, 2, or 3	word
I2Cn	single only	1	unused	word
ENCDECn	single only	1	unused	byte, half-word, or word, depending on the mode
DTMn	burst only	varies based on the peripheral	varies based on the peripheral	varies based on the peripheral
IDACn	single only	1	unused	word
SARADCn	burst only	4	2	word
SPIn	burst only	1, 2, 4, or 8	0, 1, 2, or 3	byte
TIMERn overflow	burst only	any	any	byte, half-word, or word
USARTn	single only	1	unused	word
External Trigger	burst only	any	any	byte, half-word, or word
Software Trigger	burst only	any	any	byte, half-word, or word

In addition to peripheral-initiated transfers, all of the supported DMA channels can select the rising or falling edges of one of the DMA external transfer start signals to initiate data transfers. When the selected edge occurs on the external signal, the DMA channels with the DMA0T0/1 signals selected in the DMAXBARx.CHANnSEL field will start the corresponding channel's data transfer as defined by the DMA channel data descriptor in memory. The DMA module external trigger sources are routed to peripheral pins using the crossbar.

4.5. Masking Channels

DMA channels can be temporarily disabled by setting the channel bit in CHREQMSET. Setting this bit to 1 causes the DMA channel to no longer respond to data requests from peripherals. The channel will always respond to software-initiated transfer requests, even if CHREQMSET is set for the channel. Firmware can write a 1 to the CHREQMCLR register to clear the mask for a channel.

It is recommended that firmware set the channel request mask (CHREQMSET) for channels using software-initiated transfers to avoid any peripherals connected to the channel from requesting DMA transfers.

4.6. Errors

The ERROR bit in the BERRCLR register indicates when a DMA bus error occurs. If enabled, this bit will generate an interrupt.

4.7. Arbitration

The DMA controller is a master on the AHB bus. This allows the module to control data transfers without any interaction with the core.

The channels are in a fixed priority order. Channel 0 has the highest priority, and the last implemented channel has the lowest priority. This fixed order can be superceded by using the programmable high priority setting (CHHPSET). At each re-arbitration period, the controller gives control of the bus to the highest priority channel with a pending data request.

The RPOWER field in the channel transfer descriptors determines when the re-arbitration periods occur. The channel in control of the bus will make 2^{RPOWER} data moves before the controller re-arbitrates. If the channel still has the highest priority, it can transfer again until the next re-arbitration period. The RPOWER field is only valid for peripherals that support burst requests. For peripherals that only support single requests, re-arbitration will occur after each single data move.

Figure 8 shows an example controller arbitration with two channels active. Channel 0 has an RPOWER of 1 (2 data moves), and channel 1 has an RPOWER of 0 (1 data move). Both channels are set to move words (DSTSIZE and SRCSIZE set to 2).

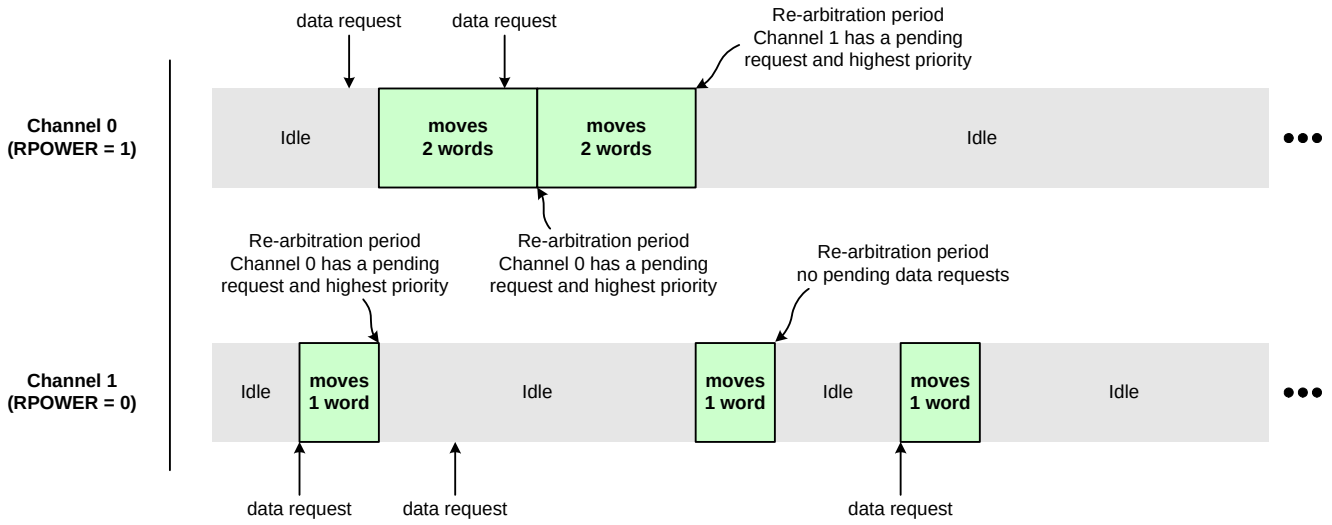


Figure 8. DMA Arbitration Example

4.8. Fast Mode

The SCONFIG module contains a bit (FDMAEN) that enables faster DMA transfers when set to 1. It is recommended that all applications using the DMA set this bit to 1.

5. Using the DMA for a Memory-to-Memory Transfer

The memory-to-memory transfer is the most basic DMA operation since it doesn't require the interaction with a peripheral. A memory-to-memory transfer can use the Auto-Request DMA transfer type.

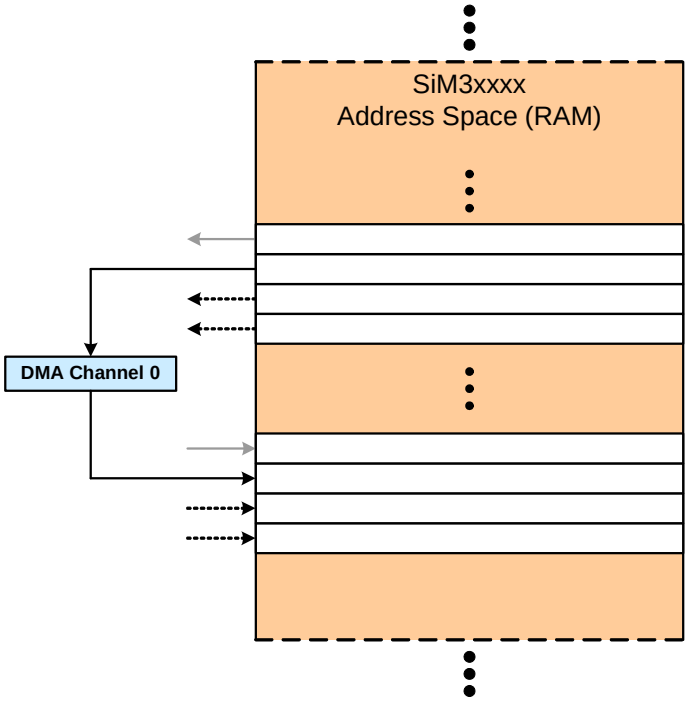


Figure 9. Memory-to-Memory DMA Transfer

To configure a DMA channel for a memory-to-memory data transfer:

1. Enable the AHB and APB clocks to the DMA controller.
2. Enable the DMA module (DMAEN = 1).
3. Set the address location of the channel transfer descriptors (BASEPTR) according to the restrictions in section "4.1. DMA Channel Transfer Descriptors".
4. Use the CHALTCLR register to set the channel to use the primary descriptor.
5. Create the primary descriptor in memory for the desired transfer:
   a. Set the SRCEND field to the last address of the source data.
   b. Set the DSTEND field to the last address of the destination memory.
   c. Set the destination and source address increment modes (DSTAIMD and SRCAIMD). In most cases, these values should be the same.
   d. Set the destination and source data size (DSTSIZE and SRCSIZE) to the same value.
   e. Set the RPOWER to the desired number of data transfers between rearbitration. In most cases, this value can be 0.
   f. Set the NCOUNT field to the total number of transfers minus 1.
   g. Set the transfer mode to the auto-request type (TMD = 2).
6. Disable data requests for the channel using the CHREQMSET register.
7. Set the DMA to fast mode using the FDMAEN bit in the SCONFIG module.
8. Enable the DMA channel using the CHENSET register.
9. (Optional) Enable the DMA channel interrupt.
10. Submit a request to start the transfer.

For memory-to-memory transfers that do not rely on a peripheral, the easiest way to initiate these transfers is to use the software request in the CHSWRCN register. It is recommended that firmware set the channel request mask (CHREQMSET) for channels using software-initiated transfers to avoid any peripherals connected to the channel from requesting DMA transfers.

Alternate start-of-transfer triggers could be the external triggers (DMAXT0 or DMAXT1) or a timer overflow trigger.

The **DMA_Memory_to_Memory.c** code example included in the software package demonstrates this type of DMA transfer.

6. Using the DMA for a Peripheral-to-Memory or Memory-to-Peripheral Transfers

A peripheral-to-memory or memory-to-peripheral transfer can use the Basic DMA transfer type.

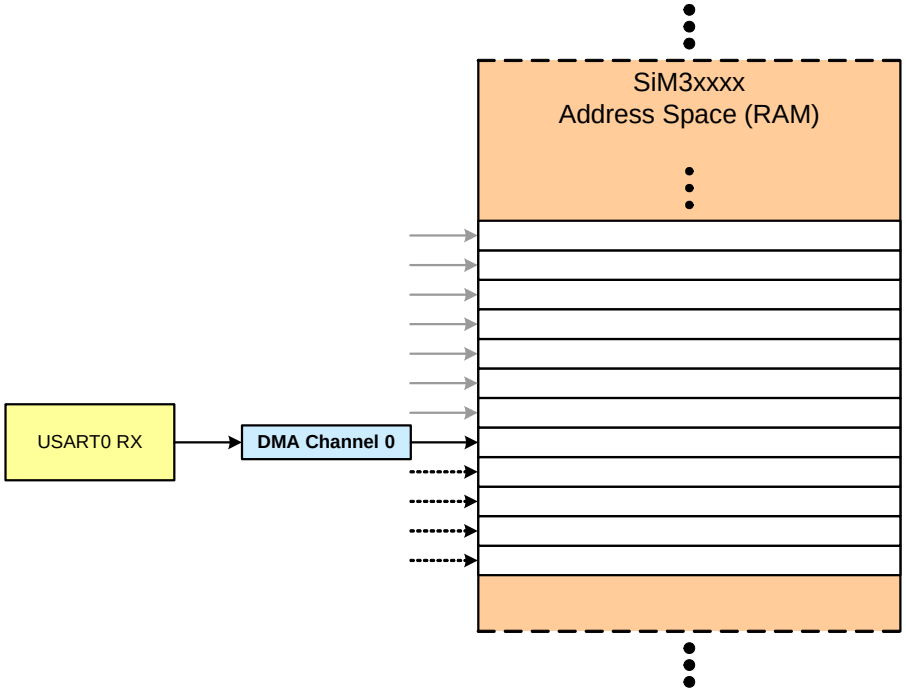


Figure 10. Peripheral-to-Memory DMA Transfer

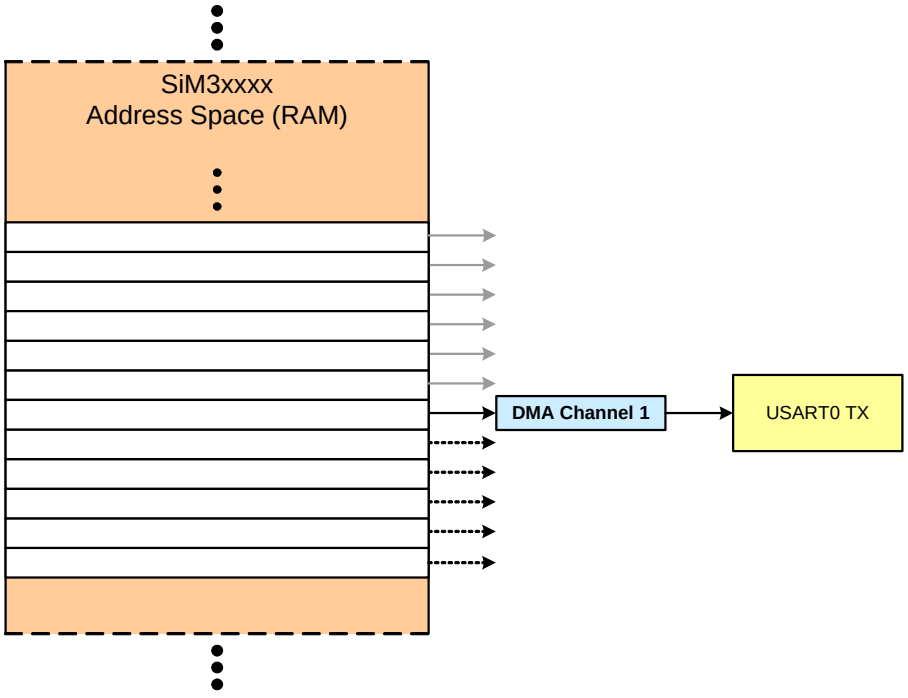


Figure 11. Memory-to-Peripheral DMA Transfer

To configure a DMA channel for a peripheral-to-memory (receive) or memory-to-peripheral (transmit) data transfer:

1. Enable the AHB and APB clocks to the DMA controller.
2. Enable the DMA module (DMAEN = 1).
3. Set the address location of the channel transfer descriptors (BASEPTR).
4. Route the DMA signals from the peripheral function to a DMA channel.
5. Use the CHALTCLR register to set the channel to use the primary descriptor.
6. Create the primary descriptor in memory for the desired transfer:
    a. Set the SRCEND field to the last address of the source data.
    b. Set the DSTEND field to the peripheral FIFO register.
    c. Set the destination and source address increment modes (DSTAIMD and SRCAIMD). For peripheral-to-memory transfers, the source should be in non-incrementing mode. For peripheral-to-memory transfers, the destination should be in non-incrementing mode.
    d. Set the destination and source data size (DSTSIZE and SRCSIZE) to the same value.
    e. Set the RPOWER to the desired number of data transfers between rearbitration. See the appropriate RPOWER value for the peripheral (Table 4, Table 5, and Table 6).
    f. Set the NCOUNT field to the total number of transfers minus 1.
    g. Set the transfer mode to the basic type (TMD = 1).
7. Enable data requests for the channel (CHREQMCLR).
8. Set the DMA to fast mode using the FDMAEN bit in the SCONFIG module.
9. Enable the DMA channel using the CHENSET register.
10. (Optional) Enable the DMA channel interrupt.
11. Enable the peripheral to start the transfer.

The CHALTSET register can set a DMA channel to use the alternate descriptor instead of the primary descriptor. Firmware can use the CHALTCLR register to set the channel back to the primary descriptor. The controller automatically updates the CHALTSET fields to indicate which descriptor is in use during transfers that use the alternate descriptor (ping-pong and scatter-gather).

The **DMA_Peripheral_to_Memory.c** code example included in the software package demonstrates a peripheral-to-memory DMA transfer by using the USART0 peripheral (receive only) to receive ASCII characters from the CP210x USB-to-UART bridge on the MCU Card and store them in memory. The characters can be entered using a Terminal program on the PC.

The **DMA_Memory_to_Peripheral.c** code example included in the software package demonstrates a memory-to-peripheral DMA transfer by using the USART0 peripheral (transmit only) to transmit an ASCII table stored in memory to the CP210x USB-to-UART bridge on the MCU Card. The table is displayed in a Terminal program on the PC.

7. Using the DMA for a Delayed Peripheral-to-Memory-to-Peripheral Transfer

The ping-pong data transfer allows a single DMA channel to perform multiple actions. After the first transfer described by the primary descriptor completes, the DMA channel will automatically transition to the transfer in the alternate descriptor. To stop the DMA, load the last descriptor with a basic transfer type (TMD = 1). The DMA will interrupt after each ping-pong transfer completes.

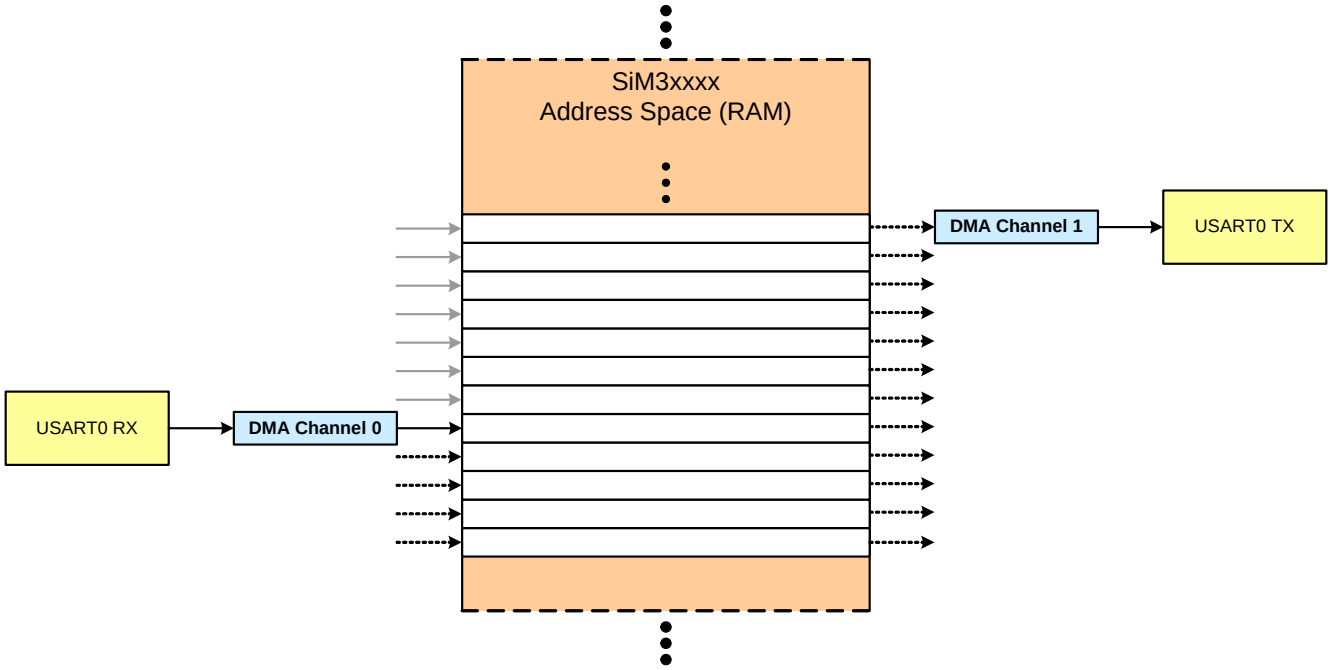


Figure 12. Peripheral-to-Memory-to-Peripheral DMA Transfer

To configure a DMA channel for ping-pong data transfer:

1. Enable the AHB and APB clocks to the DMA controller.
2. Enable the DMA module (DMAEN = 1).
3. Set the address location of the channel transfer descriptors (BASEPTR).
4. Route the DMA signals from the peripheral function to a DMA channel.
5. Use the CHALTCLR register to set the channel to use the primary descriptor.
6. Create the primary descriptor in memory for the desired transfer:
   a. Set the SRCEND field to the last address of the source data.
   b. Set the DSTEND field to the peripheral FIFO register.
   c. Set the destination and source address increment modes (DSTAIMD and SRCAIMD). For peripheral-to-memory transfers, the source should be in non-incrementing mode. For peripheral-to-memory transfers, the destination should be in non-incrementing mode.
   d. Set the destination and source data size (DSTSIZE and SRCSIZE) to the same value.
   e. Set the RPOWER to the desired number of data transfers between rearbitration. See the appropriate RPOWER value for the peripheral.
   f. Set the NCOUNT field to the total number of transfers minus 1.
   g. If this is the last action for the DMA channel, set the transfer mode to the basic type (TMD = 1). Otherwise, set the transfer mode to the ping-pong type (TMD = 3).

7. Create the alternate descriptor in memory for the desired transfer:
   a. Set the SRCEND field to the last address of the source data.
   b. Set the DSTEND field to the peripheral FIFO register.
   c. Set the destination and source address increment modes (DSTAIMD and SRCAIMD). For peripheral-to-memory transfers, the source should be in non-incrementing mode. For peripheral-to-memory transfers, the destination should be in non-incrementing mode.
   d. Set the destination and source data size (DSTSIZE and SRCSIZE) to the same value.
   e. Set the RPOWER to the desired number of data transfers between rearbitration. See the appropriate RPOWER value for the peripheral.
   f. Set the NCOUNT field to the total number of transfers minus 1.
   g. If this is the last action for the DMA channel, set the transfer mode to the basic type (TMD = 1). Otherwise, set the transfer mode to the ping-pong type (TMD = 3).
8. Enable data requests for the channel.
9. Set the DMA to fast mode using the FDMAEN bit in the SCONFIG module.
10. Enable the DMA channel using the CHENSET register.
11. (Optional) Enable the DMA channel interrupt.
12. Submit a request to start the transfer.

The **DMA_Peri_to_Mem_to_Peri.c** code example included in the software package demonstrates a ping-pong DMA transfer by using the USART0 peripheral (transmit and receive) to receive 2 10-byte character sets from a Terminal program on the PC using the CP210x USB-to-UART bridge on the MCU Card. When the second set is received, a second DMA channel displays the received characters in the Terminal program on the PC. This code example uses two different DMA channels due to the DMA peripheral crossbar mapping and since both the USART0 receive and USART0 transmit features are used.

8. Using the DMA for a Simultaneous Peripheral-to-Memory-to-Peripheral Transfer

A single DMA channel is unable to service more than one peripheral simultaneously. In addition, DMA channels cannot automatically start or pause a transfer in another channel without core intervention.

The easiest way to perform a peripheral-to-memory-to-peripheral transfer (i.e., I2C-to-memory-to-UART) is to set up several buffers in memory. Once the first peripheral transfers data to the first buffer, the DMA channel done interrupt will occur, if enabled. Inside this interrupt service routine, firmware can start the first DMA channel to transfer data from the source peripheral to a second buffer and set up a second DMA channel to transfer data from the first buffer to the end peripheral.

The DTM module on SiM3L1xx devices allows these types of transfer to occur without core intervention.

9. DTM Overview (SiM3L1xx Devices Only)

The DTM module collects DMA request signals from various peripherals and generates a series of master DMA requests based on a state-driven configuration. This master request drives a set of DMA channels to perform functions such as assembling and transferring communication packets to external radio peripherals. This capability saves power by allowing the MCU to remain in low power modes such as PM2 during complex transfer operations. A combination of simple and peripheral-scatter-gather DMA configurations can be used to perform complex operations while limiting the memory requirements (for example, by implementing direct peripherals-to-peripheral transfers).

Each DTM block supports up to 15 user-configurable states. Each state can be set up to run a certain number of DMA operations from one peripheral (the source) to another (the destination), including memory areas such as flash and RAM. Each state also has the ability to define two options for what the next state in the sequence will be, dependent on the condition of the counters and other parameters in the DTM block.

Each DTM block is capable of driving up to four DMA channels (A, B, C and D), and each DTM state can be configured to drive a particular request line to the DMA. This allows basic DMA operations to replace a long sequence of peripheral-scatter-gather tasks in most applications, saving memory.

9.1. Counters

The DTM modules contain three different counters: a master counter, a state counter, and a timeout counter. The 16-bit master counter, represented in the MSTCOUNT register, can be initialized by firmware to track the number of DMA requests that have occurred. MSTCOUNT is decremented on each DMA operation unless the active state configuration specifies otherwise.

The 8-bit State counter, represented by the STCOUNT field in the CONTROL register, also decrements each time a DMA request is generated. This is used to track the number of requests since the active state was last entered (from 1 to 256). The STCOUNT field is automatically loaded with the value of STRELOAD in the state description when a state is entered.

A 16-bit timeout counter is represented by the TOCOUNT field in the TIMEOUT register. An internal 8-bit prescaler divides the APB clock frequency and TOCOUNT is decremented every 256 APB clock cycles. Each state can selectively reload TOCOUNT and enable or disable the timeout counter while the state is active. If a TOCOUNT reload is requested, the timeout counter will be reloaded from the TORELOAD field in the TIMEOUT register, and the internal prescaler will reset. If TOCOUNT reaches 0 and the internal prescaler overflows, a timeout error is declared and the DTM transitions to its DONE state. The TOERRI flag in the CONTROL register will be set and an interrupt will be generated if enabled. When it is used, the length of the timeout is equal to 256 x (TRELOAD + 1) APB clock cycles.

9.1.1. State Machine Control

Each of the 15 available states in a DTM block has configuration information which defines the state operation when it is active. States are set up by firmware in the RAM or flash region of the device, and when a state becomes active, its information is read into the DTM block's STATE register.

9.1.2. Source, Destination, and DTM Channel

The SRCMOD and DSTMOD fields define the source trigger and the destination trigger for the transfers that will occur in the active state. The available sources and destinations are detailed in Table 10 and Table 11. If the required DMA transfer is going to or from a memory location, the value 1111b (0xF) should be used in the corresponding field.

Table 10. DTM Source Module Options

SRCMOD	Source	SRCMOD	Source
0000	SPI0 Receive	1000	EPCA0 Capture
0001	SPI1 Receive	1001	ENCDEC0 Output
0010	AES0 Output	1010	Reserved
0011	Reserved	1011	Reserved
0100	USART0 Receive	1100	Reserved
0101	Reserved	1101	DMA0T0
0110	I2C0 Receive	1110	DMA0T1
0111	SARADC0 Output	1111	Memory Transfer (No Source)

Table 11. DTM Destination Module Options

DSTMOD	Destination	DSTMOD	Destination
0000	SPI0 Transmit	1000	EPCA0 Capture
0001	SPI1 Transmit	1001	ENCDEC0 Input
0010	AES0 Data In	1010	Reserved
0011	AES0 XOR In	1011	Reserved
0100	USART0 Transmit	1100	Reserved
0101	Reserved	1101	DMA0T0
0110	I2C0 Transmit	1110	DMA0T1
0111	IDAC0 Input	1111	Memory Transfer (No Destination)

When a given state is active, the DTM waits until both its source and destination peripherals have asserted their DMA request signals, indicating that both are ready to transmit/receive DMA traffic. At this time, the DTM asserts its master DMA request signal for the channel specified in the state's DTMCHSEL field, causing the DMA engine to perform the next task in that channel's sequence of operation. This DMA task satisfies the source and destination peripheral requests by moving data from the source to the destination. In general, the source and destination peripherals will not be assigned to a DMA channel in the DMA crossbar, and all related DMA traffic will be requested by the DTM.

9.1.3. State Transitions

Each state is associated with a number, 0 through 14. The number 15 is reserved for a DONE state, which terminates DTM operations. The states define two possible paths for the next state, defined in the PRIST (primary state) and SECST (secondary state) fields of the state structure. These two fields may be loaded with any valid state value, including 15 (the DONE state). A simple representation of the DTM state transitions is shown in Figure 13.

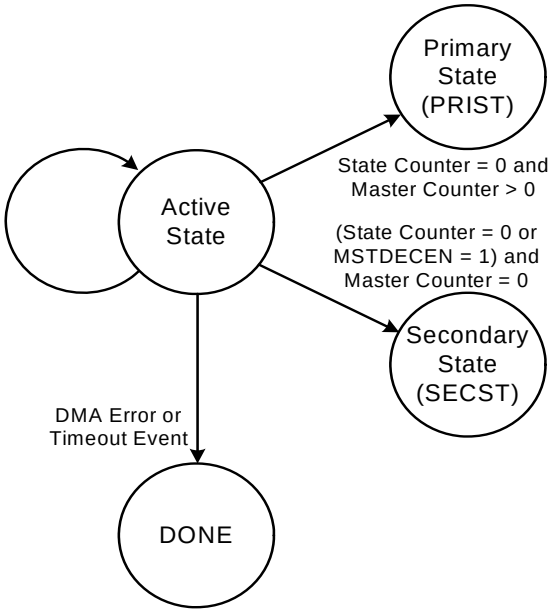


Figure 13. State Transition Diagram

When a state is entered, it becomes the active state. Its information is loaded from memory into the STATE register, and its state number will be reported in the ST field of the CONTROL register. At the same time, the state counter (STCOUNT) will be loaded with the value in the state's STRELOAD field. While a state is active, the DTM will manage the data transfer between the selected source and destination peripherals, using the selected DTM channel to request DMA operations. The operation will last as long as the DMA is still actively transferring the data. After the transfer is complete, the state counter is decremented. If the MSTDECEN bit in the state structure is set to 1, the master counter will also be decremented.

If the master counter is non-zero and the state counter is equal to zero, the state machine will transition to the primary state defined by PRIST. If the master counter reaches zero and either the state counter is zero or MSTDECEN = 1, the state machine will transition to the secondary state defined by SECST. Finally, if a timeout error occurs (TOCOUNT reaches zero) when timeouts are enabled, or if a DMA error occurs for the selected channel, the state machine will transition to the DONE state and generate the appropriate flags. Upon exit from a state, the value of that state is loaded into the LASTST field in the CONTROL register.

In some scenarios, a state will need to remain active until MSTCOUNT reaches zero, even if there are more than 256 requests generated. In such cases, this is accomplished by setting the value of PRIST to the active state number.

It is also possible to instruct a state to hold off any further transfer requests until an external pin input (specified by the INHSEL field in the CONTROL register) is asserted. The DTMINH and INHSPOL fields in the state structure configure this capability for the selected inhibit pin.

9.1.4. Interrupts

Within a state structure, the user can selectively enable timeout interrupts and state transition interrupts. The timeout counter and its associated interrupt are enabled using the TOERRIEN flag. If TIOERRIEN is set, TOCOUNT is loaded with the value of TORELOAD on entry into the state. If the TOCOUNT field reaches zero, the TIOERRI interrupt flag will be set, and the state machine transitions to DONE.

The PRISTIEN and SECSTIEN flags enable interrupts upon transition to the primary and secondary states, respectively. When either of these interrupts occurs, the DTMI interrupt flag will be set by hardware.

10. Using the DTM Module on SiM3L1xx Devices for Peripheral-to-Memory-to-Peripheral Transfers

The DTM allows DMA channels to automatically chain without core intervention. When creating DTM code, it is recommended to first start with the basic peripheral configuration and verify the peripheral is configured correctly before adding DMA support. Once the DMA operation is verified, add the final DTM piece to create the full system.

To set up a DTM system:

1. Enable AHB and APB clocks to the DTM module.
2. Start with DTM/DMA channels disabled.
3. Configure the DMA channel or channels. Ensure the DMA crossbar maps the DMA channels to DTM channels. Wait to enable the channels until after the DTM module is completely initialized.
4. Initialize the state fields for the DTM operation. For each state:
   a. Set the DTM Channel for the state (A, B, C, or D).
   b. Set the source field (SRCMOD).
   c. Set the destination field (DSTMOD).
   d. Select the next state for the primary state transition (PRIST).
   e. Select the next state for the secondary state transition (SECST).
   f. Set the number of DMA transfers for the state using the STRELOAD field.
   g. Set any active interrupts or other options for the state.
5. Initialize the DTM module.
6. Clear all interrupts in the DTM module.
7. (Optional) Enable DTM module interrupts.
8. Enable the DTM module and DMA channels.

The **DTM_Peri_to_Mem_to_Peri.c** code example included in the software package demonstrates two basic DMA transfers using the USART0 peripheral (transmit and receive) to receive and transmit a 20-byte character set from and to a Terminal program on the PC using the CP210x USB-to-UART bridge on the MCU Card. The DTM automatically starts the transmit DMA transfer after the receive DMA transfer, displaying the characters in the Terminal program on the PC. This code example uses two different DMA and DTM0 channels for the two states S0 (receive) and S1 (transmit).

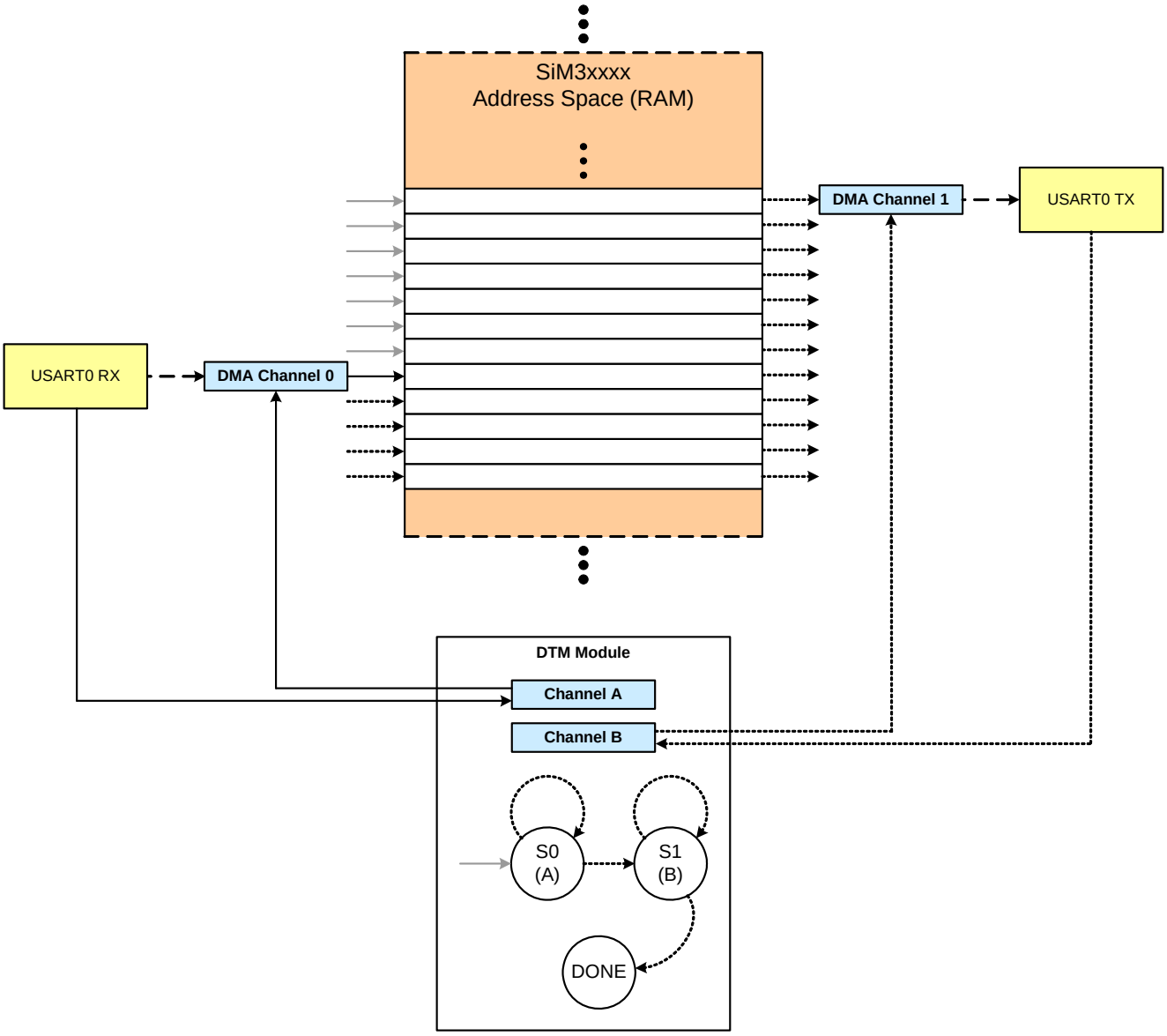
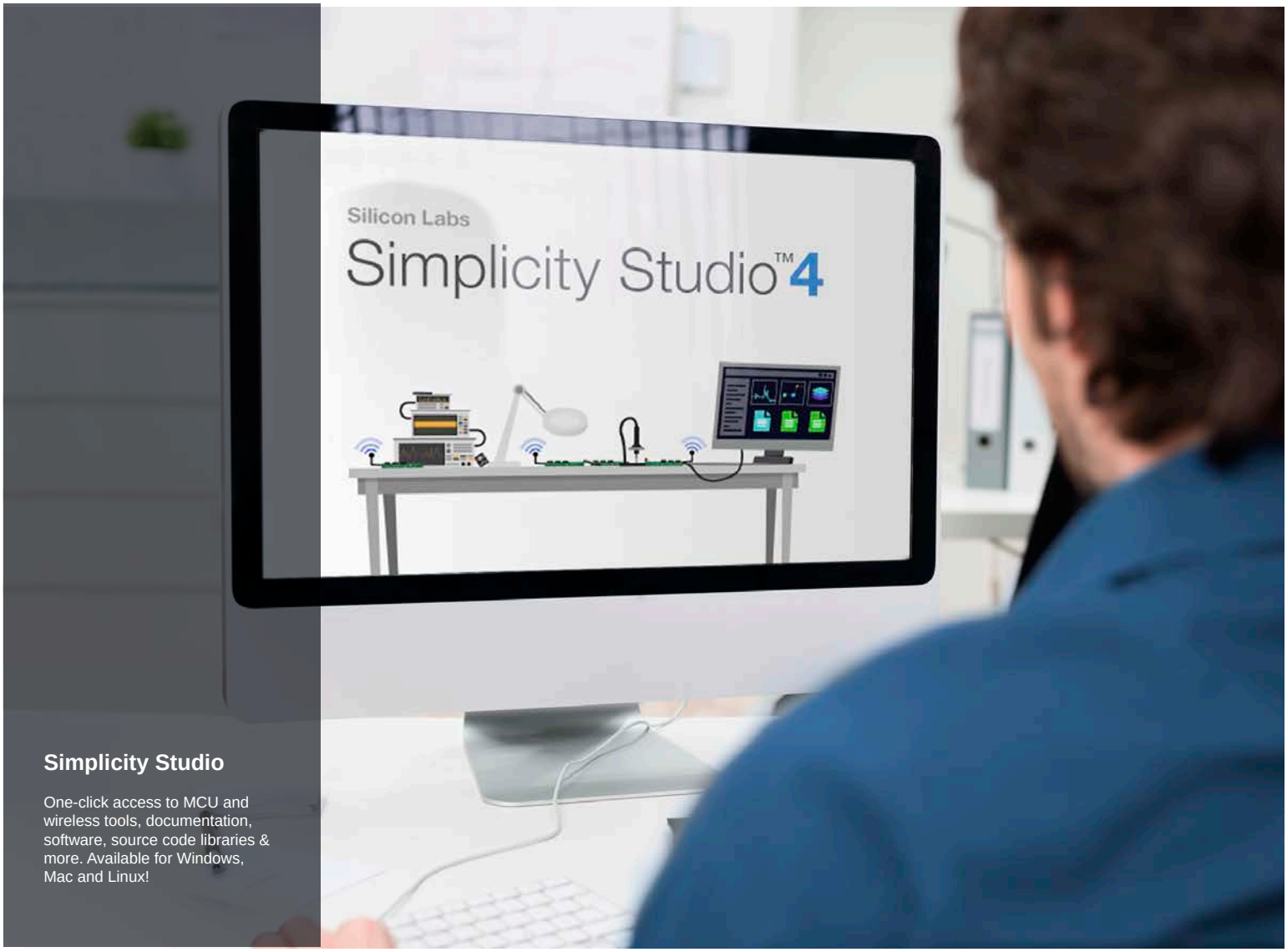


Figure 14. DTM Peripheral-to-Memory-to-Peripheral Example



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com