Modular Scatter-Gather DMA Core 25

2016.10.28

UG-01085 Subscribe Send Feedback

Core Overview

In a processor subsystem, data transfers between two memory spaces can happen frequently. In order to offload the processor from moving data around a system, a Direct Memory Access (DMA) engine is introduced to perform this function instead. The Modular Scatter-Gather DMA (mSGDMA) is capable of performing data movement operations with preloaded instructions, called descriptors. Multiple descriptors with different transfer sizes, and source and destination addresses have the option to trigger interrupts.

The mSGDMA core has a modular design that facilitates easy integration with the FPGA fabric. The core consists of a dispatcher block with optional read master and write master blocks. The descriptor block receives and decodes the descriptor, and dispatches instructions to the read master and write master blocks for further operation. The block is also configured to transfer additional information to the host. In this context, the read master block reads data through its Avalon-MM master interface, and channels it into the Avalon-ST source interface, based on instruction given by the dispatcher block. Conversely, the write master block receives data from its Avalon-ST sink interface and writes it to the destination address via its Avalon-MM master interface.

Feature Description

The mSGDMA provides three configuration structures for handling data transfers between the Avalon-MM to Avalon-MM, Avalon-MM to Avalon-ST, and Avalon-ST to Avalon-MM modes. The sub-core of the mSGDMA is instantiated automatically according to the structure configured for the mSGDMA use model.

Figure 25-1: mSGDMA Module Configuration with support for Memory-Mapped Reads and Writes

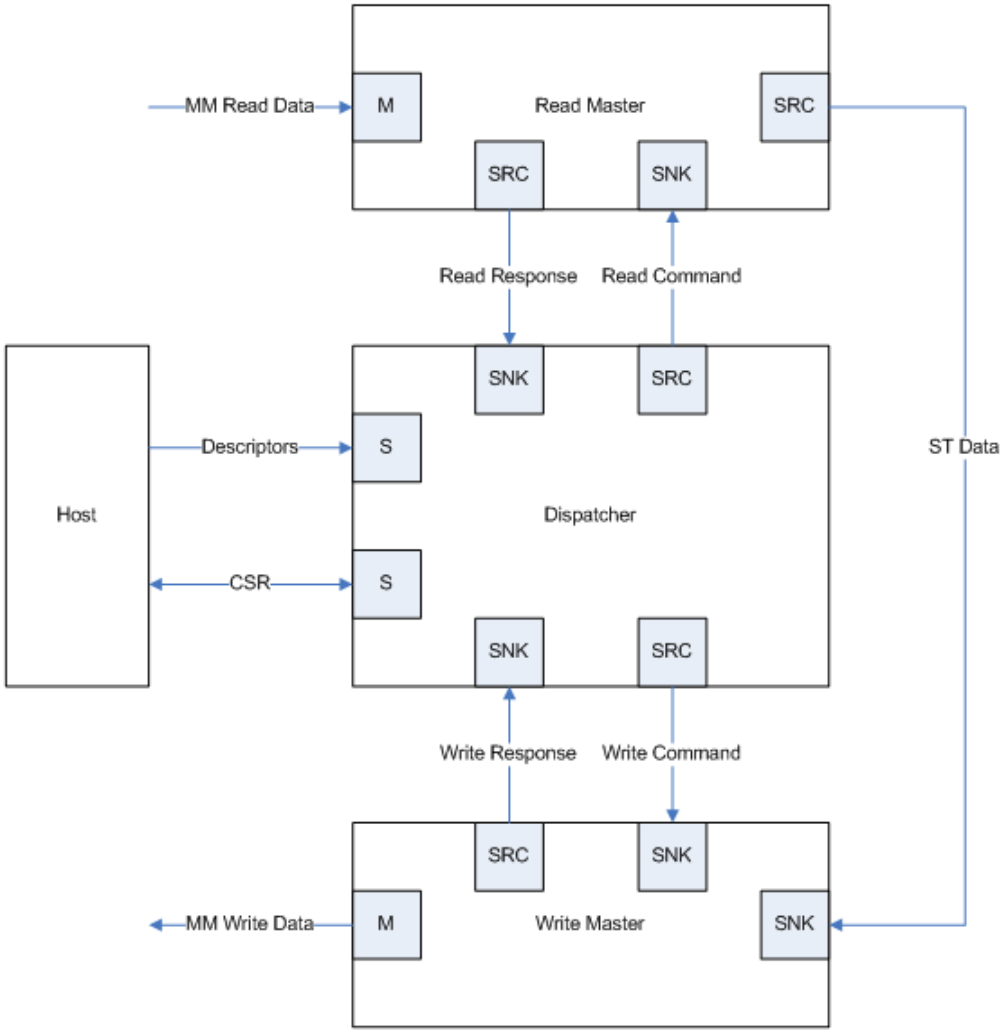


Figure 25-2: mSGDMA Module Configuration with Support for Memory-Mapped Streaming Reads to the Avalon-ST data bus

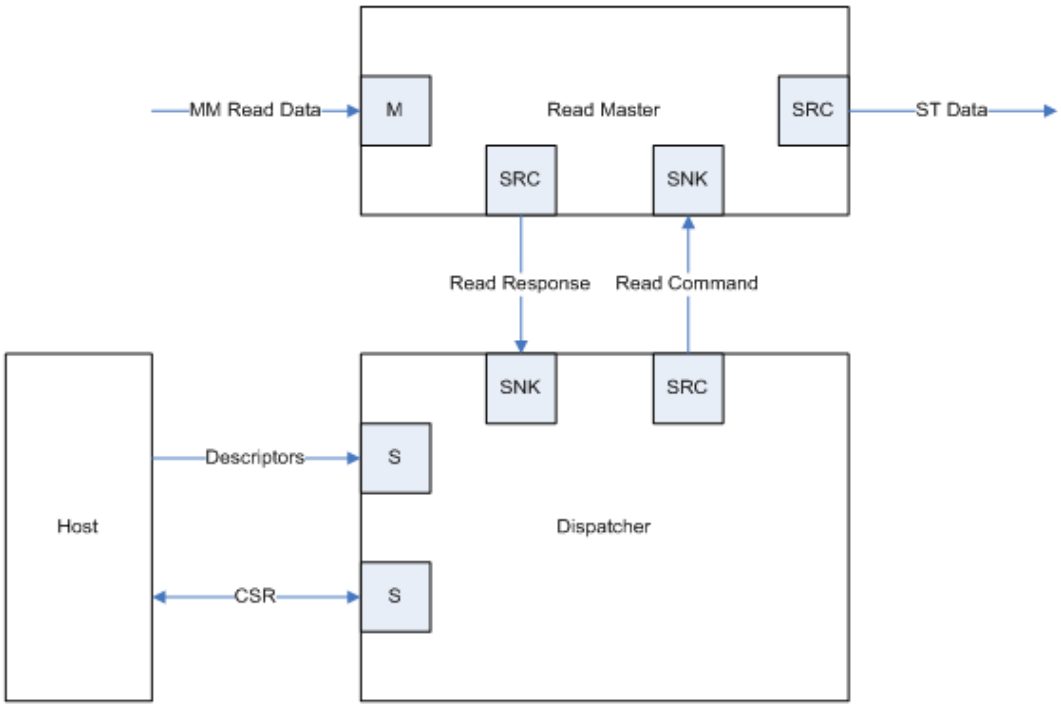
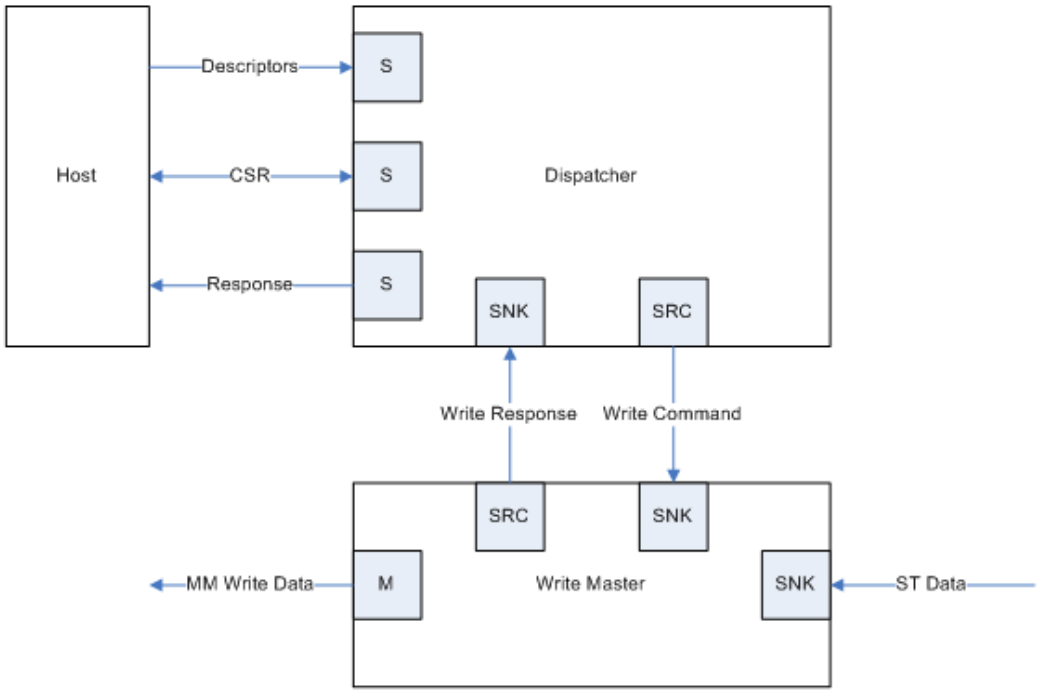


Figure 25-3: mSGDMA Module Configuration with Support for Avalon-ST Data Write Streaming to the Memory-Mapped Bus



The mSGDMA support 32-bit addressing by default. However, the core can support 64-bit addressing when you select **Extended Feature Options** in the parameter editor. It also supports extended features such as dynamic burst count programming, stride addressing, extended discriptor format (64-bit addressing), and unique sequence number identification for executed descriptor.

mSGDMA Interfaces and Parameters

Interface

The mSGDMA consists of the following:

- One Avalon-MM CSR slave port.
- One configurable Avalon-MM Slave or Avalon-ST Source Response port.
- Source and destination data path ports, which can be Avalon-MM or Avalon-ST.

The mSGDMA also provides an active-high-level interrupt output.

Only one clock domain can drive the mSGDMA. The requirement of different clock domains between source and destination data paths are handled by the Qsys fabric.

A hardware reset resets the whole system. Software reset resets the registers and FIFOs of the dispatchers of the dispatcher and master modules. For a software reset, read the resetting bit of the status register to determine when a full reset cycle completes.

Descriptor Slave Port

The descriptor slave port is write only and configurable to either 128 or 256 bits wide. The width is dependent on the descriptor format you choose for your system. When writing descriptors to this port, you must set the last bit high so the descriptor can be completely written to the dispatcher module. You can access the byte lanes of this port in any order, as long as the last bit is written to during the last write access.

Control and Status Register Slave Port

The control and status register (CSR) port is read/write accessible and is 32-bits wide. When the dispatcher response port is disabled or set to memory-mapped mode then the CSR port is responsible for sending interrupts to the host.

Response Port

The response port can be set to disabled, memory-mapped, or streaming. In memory-mapped mode the response information is communicated to the host via an Avalon-MM slave port. The response information is wider than the slave port, so the host must perform two read operations to retrieve all the information.

Note: Reading from the last byte of the response slave port performs a destructive read of the response buffer in the dispatcher module. As a result, always make sure that your software reads from the last response address last.

When you configure the response port to an Avalon Streaming source interface, connect it to a module capable of pre-fetching descriptors from memory. The following table shows the ST data bits and their description.

Table 25-1: Response Source Port Bit Fields

ST Data Bits	Description
31 - 0	Actual bytes transferred [31:0]
39 - 32	Error [7:0]
40	Early termination
41	Transfer complete IRQ mask
49 - 42	Error IRQ mask(9)
50	Early termination IRQ mask(9)
51	Descriptor buffer full(10)
255 - 52	Reserved

(9) Interrupt masks are buffered so that the descriptor pre-fetching block can assert the IRQ signal.
(10) Combinational signal to inform the descriptor pre-fetching block that space is available for another descriptor to be committed to the dispatcher descriptor FIFO(s).

Parameters

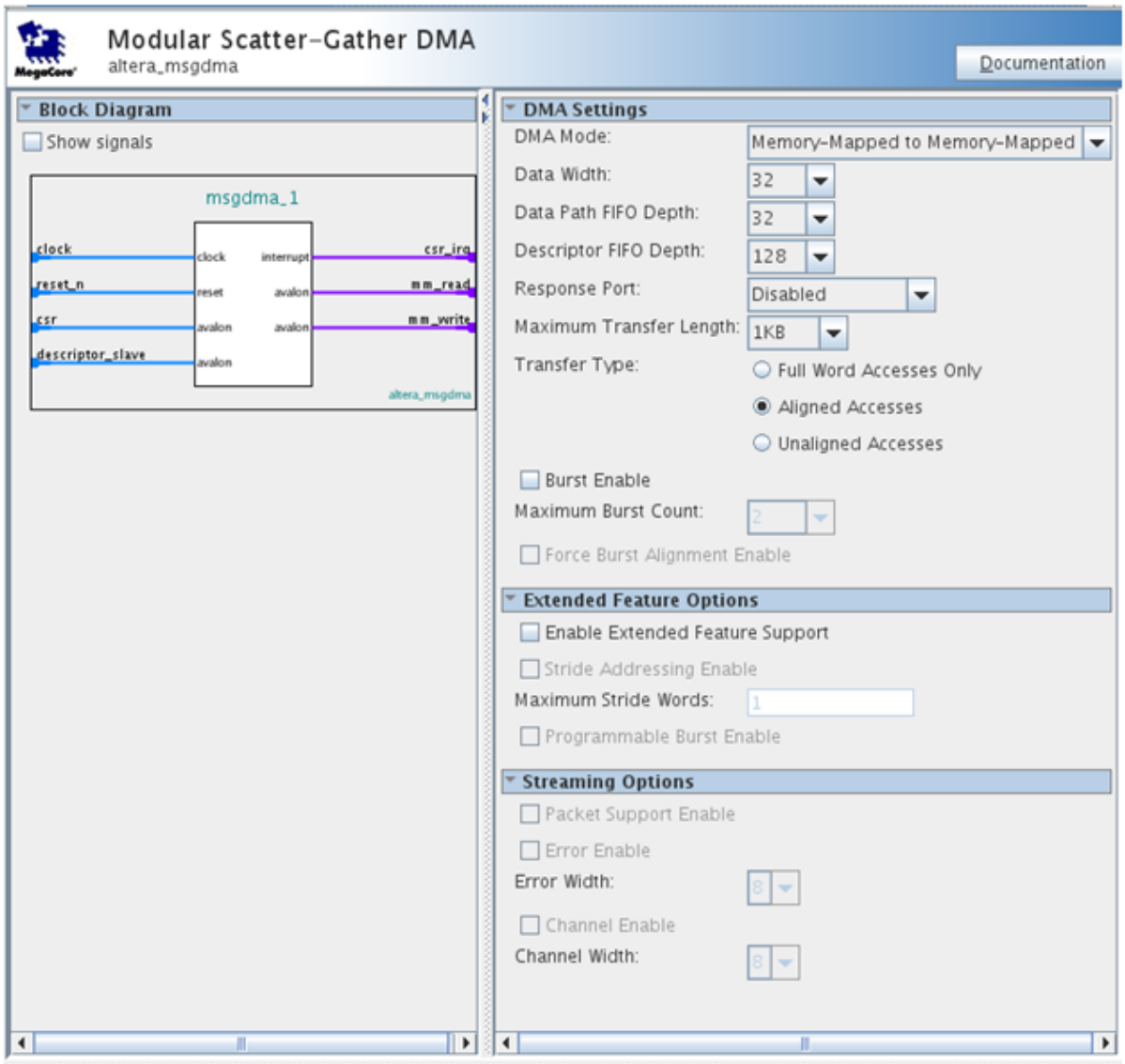
Table 25-2: Component Parameters

Parameter Name	Description	Allowable Range
DMA Mode	Transfer mode of mSGDMA. This parameter determines sub-cores instantiation to construct the mSGDMA structure.	Memory-Mapped to Memory-Mapped, Memory-Mapped to Streaming, Streaming to Memory-Mapped
Data Width	Data path width. This parameter affects both read master and write master data widths.	8, 16, 32, 64, 128, 256, 512, 1024
Use pre-determined master address width	Use pre-determined master address width instead of automatically-determined master address width.	Enable, Disable
Pre-determined master address width	Minimum master address width that is required to address memory slave.	32
Expose mSGDMA read and write master's streaming ports	When enabled, mSGDMA read master's data source port and mSGDMA write master's data sink port will be exposed for connection outside mSGDMA core.	Enable, Disable
Data Path FIFO Depth	Depth of internal data path FIFO.	16, 32, 64, 128, 256, 512, 1024, 2048, 4096
Descriptor FIFO Depth	FIFO size to store descriptor count.	8, 16, 32, 64, 128, 256, 512, 1024
Response Port	Option to enable response port and its port interface type	Memory-Mapped, Streaming, Disabled
Maximum Transfer Legth	Maximum transfer length. With shorter length width being configured, the faster frequency of mSGDMA can operate in FPGA.	1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB, 128KB,256KB, 512KB, 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, 1GB, 2GB
Transfer Type	Supported transaction type	Full Word Accesses Only, Aligned Accesses, Unaligned Accesses
Burst Enable	Enable burst transfer	Enable, Disable
Maximum Burst Count	Maximum burst count	2, 4, 8, 16, 32, 64, 128, 256, 512, 1024
Force Burst Alignment Enable	Disable force burst aligment. Force burst alignment forces the masters to post bursts of length 1 until the address is aligned to a burst boundary.	Enable, Disable

Parameter Name	Description	Allowable Range
Enable Extended Feature Support	Enable extended features. In order to use stride addressing, programmable burst lengths, 64-bit addressing, or descriptor tagging the enhanced features support must be enabled.	Enable, Disable
Stride Addressing Enable	Enable stride addressing. Stride addressing allows the DMA to read or write data that is interleaved in memory. Stride addressing cannot be enabled if the burst transfer option is enabled.	Enable, Disable
Maximum Stride Words	Maximum stride amount (in words)	1 – 2G
Programmable Burst Enable	Enable dynamic burst programming	Enable, Disable
Packet Support Enable	Enable packetized transfer **Note:** When PACKET_ENABLE parameter is disabled and TRANSFER_TYPE is not "Full Word Accesses Only", any unaligned transfer length will cause additional bytes to be written during the last transfer beat of the Avalon streaming data source port of the read master core. Only with this parameter set TRUE, actual bytes transferred is meaningful for the transaction. PACKET_ENABLE only applys for ST-to-MM and MM-to-ST DMA operation mode.	Enable, Disable
Error Enable	Enable error field of ST interface	Enable, Disable
Error Width	Error field width	1, 2, 3, 4, 5, 6, 7, 8
Channel Enable	Enable channel field of ST interface	Enable, Disable
Channel Width	Channel field width	1, 2, 3, 4, 5, 6, 7, 8
Enable Pre-Fetching module	Enables prefetcher modules, a hardware core which fetches descriptors from memory.	Enable, Disable
Enable bursting on descriptor read master	Enable read burst will turn on the bursting capabilities of the prefetcher's read descriptor interface.	Enable, Disable
Data Width of Descriptor read/write master data path	Width of the Avalon-MM descriptor read/write data path.	32, 64, 128, 256
Maximum Burst Count on descriptor read master	Maximum burst count.	Enable, Disable

mSGDMA Parameter Editor

Figure 25-4: Modular Scatter-Gather DMA Parameter Editor



mSGDMA Descriptors

The descriptor slave port is 128-bits for standard descriptors and 256-bits for extended descriptors. The tables below show acceptable standard and extended descriptor formats.

Table 25-3: Standard Descriptor Format

	Byte Lanes			
Offset	**3**	**2**	**1**	**0**
0x0	Read Address[31:0]			
0x4	Write Address[31:0]			
0x8	Length[31:0]			
0xC	Control[31:0]			

Table 25-4: Extended Descriptor Format

	Byte Lanes			
Offset	**3**	**2**	**1**	**0**
0x0	Read Address[31:0]			
0x4	Write Address[31:0]			
0x8	Length[31:0]			
0xC	Write Burst Count[7:0]	Read Burst Count [7:0]	Sequence Number[15:0]	
0x10	Write Stride[15:0]		Read Stride[15:0]	
0x14	Read Address[63:32]			
0x18	Write Address[63:32]			
0x1C	Control[31:0]			

All descriptor fields are aligned on byte boundaries and span multiple bytes when necessary. You can access each byte lane of the descriptor slave port independently of the others, allowing you to populate the descriptor using any access size.

Note: The location of the control field is dependent on the descriptor format you used. The last bit of the control field commits the descriptor to the dispatcher buffer when it is asserted. As a result, the control field is located at the end of a descriptor. This allows you to write the descriptor sequentially to the dispatcher block.

Read and Write Address Fields

The read and write address fields correspond to the source and destination address for each buffer transfer. Depending on the transfer type, you do not need to provide the read or write address. When performing memory-mapped to transfers, the write address must not be written as there is no destination address. There is no destination address since the data is being transfer to a streaming port. Likewise, when performing streaming to memory-mapped transfers, the read address must not be written as the data source is a streaming port.

If a read or write address descriptor is written in a configuration that does not require it, the mSGDMA ignores the unnecessary address. If a standard descriptor is used and an attempt to write a 64-bit address is made, the upper 32 bits are lost and can cause the hardware to alias a lower address space. 64-bit addressing requires the use of the extended descriptor format.

Length Field

The length field is used to specify the number of bytes to transfer per descriptor. The length field is also used for streaming to memory-mapped packet transfers. This limits the number of bytes that can be transferred before the end-of-packet (EOP) arrives. As a result, you must always program the length field. If you do not wish to limit packet based transfers in the case of Avalon-ST to Avalon-MM transfers, program the length field with the largest possible value of 0xFFFFFFFF. This method allows you to specify a maximum packet size for each descriptor that has packet support enabled.

Sequence Number Field

The sequence number field is available only when using extended descriptors.The sequence number is an arbitrary value that you assign to a descriptor, so that you can determine which descriptor is being operated on by the read and write masters. When performing memory-mapped to memory-mapped transfers, this value is tracked independently for masters since each can be operating on a different descriptor. To use this functionality, program the descriptors with unique sequence numbers. Then, access the dispatcher CSR slave port to determine which descriptor is operated on.

Read and Write Burst Count Fields

The programmable read and write burst counts are only available when using the extended descriptor format. The programmable burst count is optional and can be disabled in the read and write masters. Because the programmable burst count is an eight bit field for each master, the maximum that you can program burst counts of 1 to 128. Programming a value of zero or anything larger than 128 beats will be converted to the maximum burst count specified for each master automatically.

The maximum programmable burst count is 128 but when you instantiate the DMA, you can have different selections up to 1024. Refer to the MAX_BURST_COUNT parameter in the parameter table. You will still have a burst count of 128 if you program for greater than 128. Programing to 0, gets the maximum burst count selected during instantiation time.

Related Information

Parameters on page 25-6
For more information, refer to the MAX_BURST_COUNT parameter.

Read and Write Stride Fields

The read and write stride fields are optional and only available when using the extended descriptor format. The stride value determines how the read and write masters increment the address when accessing memory. The stride value is in terms of words, so the address incrementing is dependent on the master data width.

When stride is enabled, the master defaults to sequential accesses, which is the equivalent to a stride distance of one. A stride of zero instructs the master to continuously access the same address. A stride of two instructs the master to skip every other word in a sequential transfer. You can use this feature to perform interleaved data accesses, or to perform a frame buffer row and column transpose. The read and write stride distances are stored independently allowing, you to use different address incrementing for read and write accesses in memory-to-memory transfers. For example, to perform a 2:1 data decimation transfer, you would simply configure the read master for a stride distance of two and the write master for a stride distance of one. To complete the decimation operation you could also insert a filter between the two masters.

Control Field

The control field is available for both the standard and extended descriptor formats. This field can be programmed to configure parked descriptors, error handling, and interrupt masks. The interrupt masks are programmed into the descriptor so that interrupt enables are unique for each transfer.

Table 25-5: Descriptor Control Field Bit Definition

Bit	Sub-Field Name	Definition
31	Go	Commits all the descriptor information into the descriptor FIFO. As the host writes different fields in the descriptor, FIFO byte enables are asserted to transfer the write data to appropriate byte locations in the FIFO. However, the data written is not committed until the go bit has been written. As a result, ensure that the go bit is the last bit written for each descriptor. Writing '1' to the go bit commits the entire descriptor into the descriptor FIFO(s).
30:25	<reserved>	
24	Early done enable	Hides the latency between read descriptors. When the read master is set, it does not wait for pending reads to return before requesting another descriptor. Typically this bit is set for all descriptors except the last one. This bit is most effective for hiding high read latency. For example, it reads from SDRAM, PCIe, and SRIO.
23:16	Transmit Error / Error IRQ Enable	For for Avalon-MM to Avalon-ST transfers, this field is used to specify a transmit error. This field is commonly used for transmitting error information downstream to streaming components, such as an Ethernet MAC. In this mode, these control bits control the error bits on the streaming output of the read master. For Avalon-ST to Avalon-MM transfers, this field is used as an error interrupt mask. As errors arrive at the write master streaming sink port, they are held persistently. When the transfer completes, if any error bits were set at any time during the transfer and the error interrupt mask bits are set, then the host receives an interrupt. In this mode, these control bits are used as an error encountered interrupt enable.

Bit	Sub-Field Name	Definition
15	Early Termination IRQ Enable	Signals an interrupt to the host when a Avalon-ST to Avalon-MM transfer completes early. For example, if you set this bit and set the length field to 1MB for Avalon-ST to Avalon-MM transfers, this interrupt asserts when more than 1MB of data arrives to the write master without the end of packet being seen.
14	Transfer Complete IRQ Enable	Signals an interrupt to the host when a transfer completes. In the case of Avalon-MM to Avalon-ST transfers, this interrupt is based on the read master completing a transfer. In the case of Avalon-ST to Avalon-MM or Avalon-MM to Avalon-MM transfers, this interrupt is based on the write master completing a transfer.
13	<reserved>	
12	End on EOP	End on end of packet allows the write master to continuously transfer data during Avalon-ST to Avalon-MM transfers without knowing how much data is arriving ahead of time. This bit is commonly set for packet-based traffic such as Ethernet.
11	Park Writes	When set, the dispatcher continues to reissue the same descriptor to the write master when no other descriptors are buffered.
10	Park Reads	When set, the dispatcher continues to reissue the same descriptor to the read master when no other descriptors are buffered. This is commonly used for video frame buffering.
9	Generate EOP	Emits an end of packet on last beat of a Avalon-MM to Avlaon-ST transfer
8	Generate SOP	Emits a start of packet on the first beat of a Avalon-MM to Avalon-ST transfer
7:0	Transmit Channel	Emits a channel number during Avalon-MM to Avalon-ST transfers

Programming Model

Stop DMA Operation

The stop DMA operation is also referring to stop dispatcher. Once the “stop DMA” bit is set in the Control Register, no further new read or write transaction is issued. However, existing transactions pending completion are allowed to complete. The command buffer in both the read master and write master must

be clear before the DMA resumes operation via a reset request. Proceed with the follwing steps for the stop DMA operation:

1. Set the "stop DMA" bit of the Control Register.
2. Recursively check if "Stopped" bit of Status Register is asserted.
3. When the "Stopped" bit of the Status Register is asserted, reset the DMA by setting the "Reset Dispatcher" bit of the Control Register.
4. Check if the "Resetting" bit of the Status Register is deasserted. If it is, DMA is now back in normal operation.

Stop Descriptor Operation

The Stop Descriptor temporary stops the dispatcher core from continuing to issue commands to the read master and write master. The dispatcher core operates in the sense that it can accept a descriptor sent by the host up to its descriptor FIFO limit. Proceed with the follwing steps for the stop descriptor operation:

1. Set "Stop Descriptor" bit of Control Register.
2. Check if "Stopped" bit of Status Register is asserted.

To resume DMA from its previously stop descriptor operation, do the following:

1. Unset the "Stop Descriptor" bit of Control Register.
2. Check if "Stopped" bit of Status Register is deasserted.

Recovery from Stopped on Error and Stopped on Early Termination

When stopped on error or stopped on early termination occurs, mSGDMA requires a software reset to continue operation.

1. When the "Stopped" bit of the Status register is asserted, reset the DMA by setting the "Reset Dispatcher" bit of Control register.
2. Check if the "Resetting" bit of Status register is deasserted. If it is, DMA is now back in normal operation.

Register Map of mSGDMA

The following table illustrates the Altera mSGDMA register map under observation by host processor from its Avalon-MM CSR interfaces.

Table 25-6: CSR Registers Map

		Byte Lanes			
Offset	**Attribute**	**3**	**2**	**1**	**0**
0x0	Read/Clear	Status			
0x4	Read/Write	Control			
0x8	Read	Write Fill Level[15:0]		Read Fill Level[15:0]	
0xC	Read	<reserved>(11)		Response Fill Level[15:0]	

(11) Writing to reserved bits will have no impact on the hardware, reading will return unknown data.

Byte Lanes			
0x10	Read	Write Sequence Number[15:0](12)	Read Sequence Number[15:0][2]
0x14	N/A	<reserved>[1]	
0x18	N/A	<reserved>[1]	
0x1C	N/A	<reserved>[1]	

Status Register

Table 25-7: Status Register Bit Definition

Bit	Name	Description
31:10	<reserved>	N/A
9	IRQ	Set when an interrupt condition occurs.
8	Stopped on Early Termination	When the dispatcher is programmed to stop on early termination, this bit is set. Also set, when the write master is performing a packet transfer and does not receive EOP before the pre-determined amount of bytes are transferred, which is set in the descriptor length field. If you do not wish to use early termination you should set the transfer length of the descriptor to 0xFFFFFFFF ,which gives you the maximum packet based transfer possible (early termination is always enabled for packet transfers).
7	Stopped on Error	When the dispatcher is programmed to stop errors and when an error beat enters the write master the bit is set.
6	Resetting	Set when you write to the software reset register and the SGDMA is in the middle of a reset cycle. This reset cycle is necessary to make sure there are no incoming transfers on the fabric. When this bit de-asserts you may start using the SGDMA again.
5	Stopped	Set when you either manually stop the SGDMA, or you setup the dispatcher to stop on errors or early termination and one of those conditions occurred. If you manually stop the SGDMA this bit is asserted after the master completes any read or write operations that were already in progress.
4	Response Buffer Full	Set when the response buffer is full.
3	Response Buffer Empty	Set when the response buffer is empty.
2	Descriptor Buffer Full	Set when either the read or write command buffers are full.
1	Descriptor Buffer Empty	Set when both the read and write command buffers are empty.
0	Busy	Set when the dispatcher still has commands buffered, or one of the masters is still transferring data.

(12) Sequence numbers will only be present when dispatcher enhanced features are enabled.

Control Register

Table 25-8: Control Register Bit Definition

Bit	Name	Description
31:10	<reserved>	N/A
5	Stop Descriptors	Setting this bit stops the SGDMA dispatcher from issuing more descriptors to the read or write masters. Read the stopped status register to determine when the dispatcher stopped issuing commands and the read and write masters are idle.
4	Global Interrupt Enable Mask	Setting this bit allows interrupts to propagate to the interrupt sender port. This mask occurs after the register logic so that interrupts are not missed when the mask is disabled.
3	Stop on Early Termination	Setting this bit stops the SGDMA from issuing more read/write commands to the master modules if the write master attempts to write more data than the user specifies in the length field for packet transactions. The length field is used to limit how much data can be sent and is always enabled for packet based writes.
2	Stop on Error	Setting this bit stops the SGDMA from issuing more read/write commands to the master modules if an error enters the write master module sink port.
1	Reset Dispatcher	Setting this bit resets the registers and FIFOs of the dispatcher and master modules. Since resets can take multiple clock cycles to complete due to transfers being in flight on the fabric, you should read the resetting status register to determine when a full reset cycle has completed.
0	Stop Dispatcher	Setting this bit stops the SGDMA in the middle of a transaction. If a read or write operation is occurring, then the access is allowed to complete. Read the stopped status register to determine when the SGDMA has stopped. After reset, the dispatcher core defaults to a start mode of operation.

The response slave port of mSGDMA contains registers providing information of the executed transaction. This register map is only applicable when the response mode is enabled and set to memory mapped. Also when the response port is enabled, it needs to have responses read because it buffers responses. When setup as a memory-mapped slave port, reading byte offset 0x7 outputs the response. If the response FIFO becomes full the dispatcher stops issuing transfer commands to the read and write masters. The following describes the registers definition.

Table 25-9: Response Registers Map

Byte Lanes					
Offset	Access	3	2	1	0
0x0	Read	Actual Bytes Transferred[31:0]			

Byte Lanes					
Offset	Access	3	2	1	0
0x4	Read	<reserved>[13]	<reserved>	Early Termination[14]	Error[7:0]

The following list explains each of the fields:

- **Actual bytes transferred** determines how many bytes transferred when the mSGDMA is configured in Avalon-ST to Avlaon-MM mode with packet support enabled. Since packet transfers are terminated by the IP providing the data, this field counts the number of bytes between the start-of-packet (SOP) and end-of-packet (EOP) received by the write master. If the early termination bit of the response is set, then the actual bytes transferred is an underestimate if the transfer is unaligned.
- **Error** Determines if any errors were issued when the mSGDMA is configured in Avalon-ST to Avlaon-MM mode with error support enabled. Each error bit is persistent so that errors can accumulate throughout the transfer.
- **Early Termination** determines if a transfer terminates because the transfer length is exceeded when the SGDMA is configured in Avalon-ST to Avalon-MM mode with packet support enabled. This bit is set when the number of bytes transferred exceeds the transfer length set in the descriptor before the end-of-packet is received by the write master.

(13) Reading from byte 7 outputs the response FIFO.
(14) Early Termination is a single bit located at bit 8 of offset 0x4.

Modular Scatter-Gather DMA Prefetcher Core

The mSGDMA Prefetcher core is an additional micro core to the existing mSGDMA core. The Prefetcher core provides extra functionality through the Avalon-MM and dispatcher core. The Avalon-MM fetches a series of descriptors from memory that describes the required data transfers before passing them to dispatcher core for data transfer execution. The series of descriptors in memory can be linked together to form a descriptor list. This allows the DMA core to execute multiple descriptors in single run, thus enabling transfer to a non-contiguous memory space and improves system performance.

Feature Description

Supported Features

- Descriptor linked list
- Data transfer to non-contiguous memory space
- Descriptor write back
- Hardware descriptor polling
- 64-bit address spaces

Functional Description

Architecture Overview

The Prefetcher core supports all the three existing Modular SGDMA configurations:

- Memory-Mapped to Memory-Mapped
- Memory-Mapped to Streaming
- Streaming to Memory-Mapped

On interfaces facing host and external peripherals, it has dedicated Avalon-MM read and write master interfaces to fetch series of descriptors from memory as well as performing a descriptor write back. It has one Avalon Memory-Mapped CSR slave interface for the host processor to access the configuration register in the Prefetcher core.

On interfaces facing the internal dispatcher core, it has an Avalon-MM descriptor write master interface to write a descriptor to the dispatcher core. It has Avalon-ST response sink interface to receive response information from the dispatcher core upon completion of each descriptor execution.

Figure 25-5: Memory-Mapped to Memory-Mapped Configuration with Prefetcher Enabled

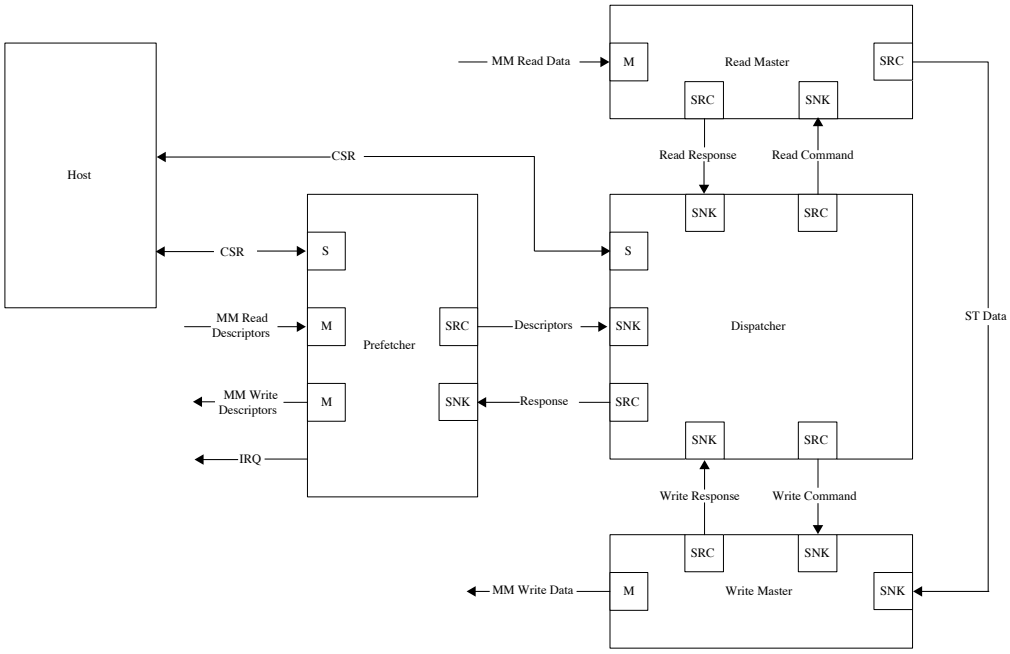


Figure 25-6: Memory-Mapped to Streaming Configuration with Prefetcher Enabled

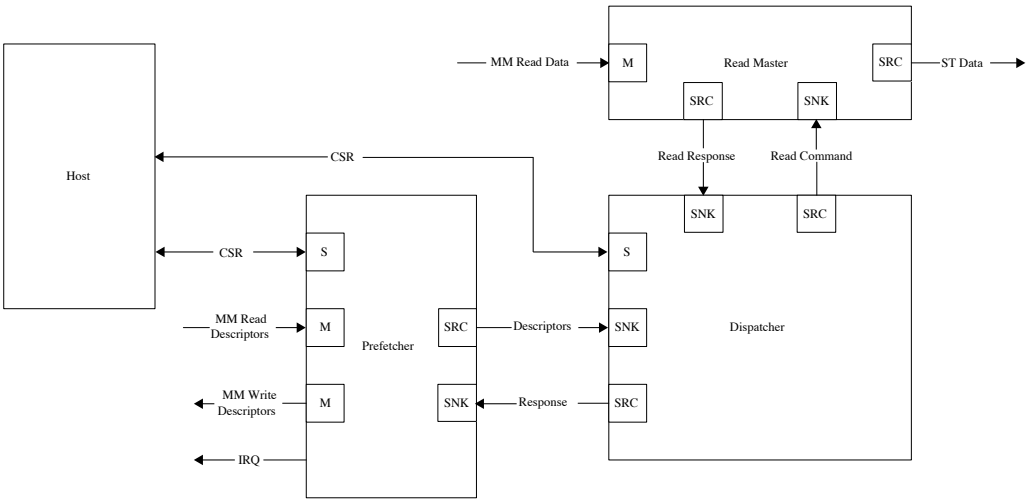
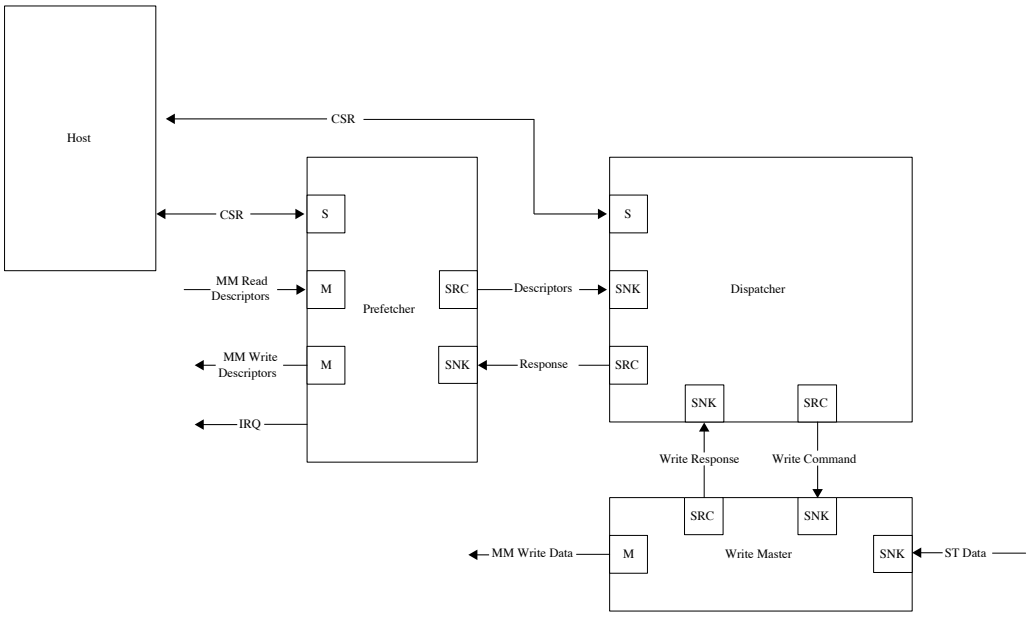


Figure 25-7: Streaming to Memory-Mapped Configuration with Prefetcher Enabled



Descriptor Format

The mSGDMA without the Prefetcher core defines two types of descriptor formats. Standard descriptor format which consists of 128 bits and extended descriptor format which consists of 256 bits. With the Prefetcher core enabled, the existing descriptor format is expanded to 256 bits and 512 bits respectively in order to accommodate additional control information for the prefetcher operation.

Table 25-10: Standard Descriptor Format when Prefetcher is Enabled

	Byte Lanes			
Offset	3	2	1	0
0x0	Read Address [31-0]			
0x4	Write Address [31-0]			
0x8	Length [31-0]			
0xC	Next Desc Ptr [31-0]			
0x10	Actual Bytes Trasferred [31-0]			
0x14	Reserved [15-0]		Status [15-0]	
0x18	Reserved [31-0]			

0x1C	Control [31, 30, 29..0]

Table 25-11: Extended Descriptor Format when Prefetcher is Enabled

	Byte Lanes			
Offset	3	2	1	0
0x0	Read Address [31-0]			
0x4	Write Address [31-0]			
0x8	Length [31-0]			
0xC	Next Desc Ptr [31-0]			
0x10	Actual Bytes Trasferred [31-0]			
0x14	Reserved [15-0]		Status [15-0]	
0x18	Reserved [31-0]			
0x1C	Write Burst Count [7-0]	Read Burst Count [7-0]	Sequence Number [15-0]	
0x20	Write Stride [15-0]		Read Stride [15-0]	
0x24	Read Address [63-32]			
0x28	Write Address [63-32]			
0x2C	Next Desc Ptr [63-32]			
0x30	Reserved [31-0]			
0x34	Reserved [31-0]			
0x38	Reserved [31-0]			
0x3C	Control [31, 30, 29..0]			

Descriptor Fields Definition

Next Descriptor Pointer

The next descriptor pointer field specifies the address of the next descriptor in the linked list.

Actual Bytes Transferred

Specifies the actual number of bytes that has been transferred. This field is not applicable if Modular SGDMA is configured as Memory-Mapped to Streaming transfer.

Table 25-12: Status

Bits	Fields	Description
15:9	Reserved	Reserved fields

Bits	Fields	Description
8	Early Termination	Indicates early termination condition where write master is performing a packet transfer and does not receive EOP before pre-determined amount of bytes are transferred. This status bit is similar to status register bit 8 of the dispatcher core. For more details refer to dispatcher core CSR definition. This field is not applicable if Modular SGDMA is configured as Memory-Mapped to Streaming transfer.
7:0	Error	Indicates an error has arrived at the write master streaming sink port. This field is not applicable if Modular SGDMA is configured as Memory-Mapped to Streaming transfer.

Table 25-13: Control

Bits	Fields	Description
30	Owned by Hardware	This field determines whether hardware or software has write access to the current descriptor. When this field is set to 1, the Modular SGDMA can update the descriptor and software should not access the descriptor due to the possibility of race conditions. Otherwise, it is safe for software to update the descriptor.

For bit 31 and 29:0, refer to descriptor control field bit 31 and 29:0 defined in dispatcher core. **Table 25-5**

Descriptor Processing

The DMA descriptors specify data transfers to be performed. With the Prefetcher core, a descriptor is stored in memory and accessed by the Prefetcher core through its descriptor write and descriptor read Avalon-MM master. The mSGDMA has an internal FIFO to store descriptors read from memory. This FIFO is located in the dispatcher's core. The descriptors must be initialized and aligned on a descriptor read/write data width boundary. The Prefetcher core relies on a cleared Owned By Hardware bit to stop processing. When the Owedn by Hardware bit is 1, the Prefetcher core goes ahead to process the descriptor. When the Owned by Hardware bit is 0, the Prefetcher core does not process the current descriptor and assumes the linked list has ended or the next descriptor linked list is not yet ready.

Each time a descriptor has been processed, the core updates the Actual Byte Transferred, Status and Control fields of the descriptor in memory (descriptor write back). The Owned by Hardware bit in the descriptor control field is cleared by the core during descriptor write back. Refer to software programming model section to know more about recommended way to set up the Prefetcher core, building and updating the descriptor list.

In order for the Prefetcher to know which memory addresses to perform descriptor write back, the next descriptor pointer information will need to be buffered in Prefetcher core. This buffer depth will be similar to descriptor FIFO depth in dispatcher core. This information is taken out from buffer each time a response is received from dispatcher.

Registers

Register Map

Table 25-14: Register Map

Name	Address Offset	Description
Control	0x0	Specifies the Prefetcher core behavior such as when to start the core.
Next Descriptor Pointer Low	0x1	Contains the address [31:0] of the next descriptor to process. Software sets this register to the address of the first descriptor as part of the system initialization sequence. If descriptor polling is enabled, this register is also updated by hardware to store the latest next descriptor address. The latest next descriptor address is used by the Prefetcher core to perform descriptor polling.
Next Descriptor Pointer High	0x2	Contains the address [63:32] of the next descriptor to process. Software set this register to the address of the first descriptor as part of the system initialization sequence. This field is used only when higher than 32-bit addressing is used when mSGDMA's extended feature is enabled. If descriptor polling is enabled, this register is also updated by hardware to store the latest next descriptor address. The latest next descriptor address is used by the Prefetcher core to perform descriptor polling.
Descriptor Polling Frequency	0x3	Descriptor Polling Frequency
Status	0x4	Status Register

Control Register

The address offset for the Control Register table is 0x0.

Table 25-15: Control Register

Bit	Fields	Access	Default Value	Description
31:5	Reserved	R	0x0	Reserved fields

Bit	Fields	Access	Default Value	Description
4	Park Mode	R/W	0x0	This bit enables the mSGDMA to repeatedly execute the same linked list over and over again. In order for this to work, software need to setup the last descriptor to point back to the first descriptor. 1: Park mode is enabled. Pefetcher will not clear the owned by hardware field during descriptor write back 0: Park mode is disabled. Prefetcher will clear the owned by hardware field during descriptor write back. Software can terminate the park mode operation by clearing this field. Since this field is in CSR and not in descriptor field itself, this termination event is asynchronous to current descriptor in progress (user can't deterministically choose which descriptor in the linked list to stop). Park mode feature is not intended to be used on the fly. User must not enable this bit when the Prefetcher is already in operation. This bit shall be set during initialization/configuration phase of the control register.
3	Global Interrupt Enable Mask	R/W	0x0	Setting this bit will allow interrupts to propagate to the interrupt sender port. This mask occurs after the register logic so that interrupts are not missed when the mask is disabled. **Note:** There is an equivalent global interrupt enable mask bit in dispatcher core CSR. When the Prefetcher is enabled, software shall use this bit. When the Prefetcher is disabled, software shall use equivalent global interrupt enable mask bit in dispatcher core CSR.

Bit	Fields	Access	Default Value	Description
2	Reset_Prefetcher	R/W1S(15)	0x0	This bit is used when software intends to stop the Prefetcher core when it is in the middle of data transfer. When this bit is 1, the Prefetcher core begin its reset sequence. This bit is automatically cleared by hardware when the reset sequence has completed. Therefore, software need to poll for this bit to be cleared by hardware to ensure the reset sequence has finished. This function is intended to be used along with reset dispatcher function in dispatcher core. Once the reset sequence in the Prefetcher core has completed, software is expected to reset the dispatcher core, polls for dispatcher's reset sequence to be completed by reading dispatcher core status register.

Bit	Fields	Access	Default Value	Description
1	Desc_Poll_En	R/W	0x0	Descriptor polling enable bit. 1: When the last descriptor in current linked list has been processed, the Prefetcher core polls the Owned By Hardware bit of next descriptor to be set and automatically resumes data transfer without the need for software to set the Run bit. The polling frequency is specified in Desc_Poll_Freq register. 0: When the last descriptor in current linked list has been processed, the Prefetcher stops operation and clears the run bit. In order to restart the DMA engine, software need to set the Run bit back to 1. In case software intends to disable polling operation in the middle of transfer, software can write this field to 0. In this case, the whole mSGDMA operation is stopped when the Prefetcher core encounter owned by hardware bit = 0. **Note:** This bit should be set during initialization or configuration of the control register.

Bit	Fields	Access	Default Value	Description
0	Run	R/W1S	0x0	Software sets this bit to 1 to start the descriptor fetching operation which subsequently initiates the DMA transaction. When descriptor polling is disabled, this bit is automatically cleared by hardware when the last descriptor in the descriptor list has been processed or when the Prefetcher core read owned by hardware bit = 0. When descriptor polling is enabled, mSGDMA operation is continuously run. Thus the run bit stays 1. This field is also cleared by hardware when reset sequence process triggered by Reset_Prefetcher bit completes.

Descriptor Polling Frequency

Table 25-16: Desc_Poll_Freq

Bit	Fields	Access	Default	Description
31:16	Reserved	R	0x0	Reserved fields
15:0	Poll_Freq	R/W	0x0	Specifies the frequency of descriptor polling operation. The polling frequency is in term of number of clock cycles. The poll period is counted from the point where read data is received by the Prefetcher core.

Status

Table 25-17: Status

Bit	Fields	Access	Default Value	Description
31:1	Reserved	R	0x0	Reserved fields

(15) W1S register attribute means, software can write 1 to set the field. Software write 0 to this field has no effect.

Bit	Fields	Access	Default Value	Description
0	IRQ	R/W1C[16]	0x0	Set by hardware when an interrupt condition occurs. Software must perform a write 1 to this field in order to clear it. There is an equivalent IRQ status bit in the dispatcher core CSR. When the Prefetcher is enabled, software uses this bit as an IRQ status indication. When the Prefetcher is disabled, software uses equivalent IRQ status bit in dispatcher core CSR.

Interfaces

Avalon-MM Read Descriptor

This interface is used to fetch descriptors in memory. It supports non-burst or burst mode which configurable during generation time.

Table 25-18: Avalon-MM Read Descriptor

Signal Role	Width	Description
Address	32 to 64-bit	Avalon-MM read address. 32-bits if extended feture is disabled. 32- to 64-bits if extended feature is enabled.
Read	1	Avalon-MM read control
Read data	32, 64, 128, 256, 512	Avalon-MM read data bus. Data width is configurable during IP generation.
Wait request	1	Avalon-MM wait request for backpressure control.
Read data valid	1	Avalon-MM read data valid indication.

[16] W1C register attribute means, software write 1 to clear the field. Software write 0 to this field has no effect.

Signal Role	Width	Description
Burstcount	1/2/3/4/5	Avalon-MM burst count. The maximum burst count is configurable during IP generation. This signal role is applicable only when the Enable Bursting on the descriptor read master is turned on.

Avalon-MM Write Descriptor

This interface is used to access the Prefetcher CSR registers. It has fixed write and read wait time of 0 cycles and read latency of 1 cycle.

Table 25-19: Avalon-MM Write Descriptor

Signal Role	Width	Description
Address	32 to 64	Avalon-MM write address
Write	1	Avalon-MM read control
Wait request	1	Avalon-MM waitrequest for backpressure control
Write data	32, 64, 128, 256, 512	Avalon-MM write data bus
Byte enable	4, 8, 16, 32, 64	Avalon-MM write byte enable control. Its width is automatically derived from selected data width

Avalon-MM CSR

This interface is used to access the Prefetcher CSR registers. It has fixed write and read wait time of 0 cycles and read latency of 1 cycle.

Table 25-20: Avalon-MM CSR

Signal Role	Width	Description
Address	3	Avalon-MM write address
Write	1	Avalon-MM read control
Read	1	Avalon-MM write control
Write data	32	Avalon-MM write data bus
Read data	32	Avalon-MM read data bus

Avalon-ST Descriptor Source

This interface is used by the Prefetcher to write descriptor information into the dispatcher core.

Table 25-21: Avalon-ST Descriptor Source

Signal Role	Width	Description
Valid	1	Avalon-ST valid control
Ready	1	Avalon-ST ready control with ready latency of 0. Refer to dispatcher's descriptor format for wrtie data definition.
Data	128/256	Avalon-ST data bus

Avalon-ST Response

This interface is used by the Prefetcher core to retrieve response information from dispatcher's core upon each transfer completion.

Table 25-22: Avalon-ST Response

Signal Role	Width	Description
Valid	1	Avalon-ST valid control. Prefetcher core expects valid signal to remain high while the bus is being back pressured.
Ready	1	Avalon-ST ready control. Used by the Prefetcher core to back pressure the external ST response source.
Data	256	Avalon-ST data bus. Refer to dispatcher's response source format for ST data definition. Prefetcher core expects data signals to remain constant while the bus is being back pressured.

Streaming interface (ST) data bus format and definition are similar to the dispatcher's response source format:

Table 25-23: Avalon-ST Response Data Format and Definition

Bits	Signal Information
[31:0]	Acutal bytes transferred [31:0]
[39:32]	Error [7:0]
40	Early termination
41	Transfer complete IRQ mask
[49:42]	Error IRQ mask
50	Early termination IRQ mask

Bits	Signal Information
51	Descriptor buffer full
[255:52]	Reserved

IRQ Interface

When the Prefetcher is enabled, IRQ generation no longer outputs from the dispatcher's core. It will be outputted from the Prefetcher core. The sources of the interrupt remain the same which are transfer completion, early termination, and error detection. Masking bits for each of the interrupt sources are programmed in the descriptor. This information will be passed to the Prefetcher core through the ST response interface. An equivalent global interrupt enable mask and IRQ status bit which are defined in dispatcher core are now defined in the Prefetcher core as well. These two bits need to be defined in the Prefetcher core since the actual IRQ register is now located in the Prefetcher core.

Software Programming Model

Setting up Descriptor and mSGDMA Configuration Flow

The following is the recommended software flow to setup the descriptor and configuring the mSGDMA.

1. Build the descriptor list and terminate the list with a non-hardware owned descriptor (Owned By Hardware = 0).
2. Configure mSGDMA by accessing dispatcher core control register (for example: to configure Stop on Error, Stop on Early Termination, etc…)
3. Configure mSGDMA by accessing the Prefetcher core configuration register (for example: to write the address of the first descriptor in the first list to the next descriptor pointer register and set the Run bit to 1 to initiate transfers).
4. While the core is processing the first list, your software may build a second list of descriptors.
5. An IRQ can be generated each time a descriptor transfer is completed (depends whether transfer complete IRQ mask is set for that particular descriptor). If you only need an IRQ to be generated when mSGDMA finishes processing the first list, you only need to set transfer complete IRQ mask for the last descriptor in the first list.
6. When the last descriptor in the first linked list has been processed, an IRQ will be generated if the descriptor polling is disabled. Following this, your software needs to update the next descriptor pointer register with the address of the first descriptor in the second linked list before setting the run bit back to 1 to resume transfers. If descriptor polling is enabled, software does not need to update the next descriptor pointer register (for second descriptor linked list onwards) and set the run bit back to 1. These 2 steps are automatically done by hardware. The address of the new list is indicated by next descriptor pointer fields of the previous list. The Prefetcher core polls for the Owned by Hardware bit to be 1 in order to resume transfers. Software only needs to flip the Owned by Hardware bit of the first descriptor in second linked list to 1 to indicate to the Prefetcher core that the second linked list is ready.
7. If there are new descriptors to add, always add them to the list which the core is not processing (indicated by Owned By Hardware = 0). For example, if the core is processing the first list, add new descriptors to the second list and so forth. This method ensures that the descriptors are not updated when the core is processing them. Your software can read the descriptor in the memory to know the status of the transfer (for example; to know the actual bytes being transferred, any error in the transfer).

Resetting Prefetcher Core Flow

The following is the recommended flow for software to stop the mSGDMA when it is in the middle of operation.

1. Write 1 to the Prefetcher control register bit 2 (Reset_Prefetcher bit set to 1).
2. Poll for control register bit 2 to be 0 (Reset_Prefetcher bit cleared by hardware).
3. Trigger software reset condition in the dispatcher core.
4. Poll for software reset condition in the dispatcher core to be completed by reading the dispatcher core status register.
5. The whole reset flow has completed, software can reconfigure the mSGDMA.

Parameters

Table 25-24: Prefetcher Parameters

Name	Legal Value	Description
Enable Pre-fetching Module	1 or 0	1: Pre-fetching is enabled 0: Pre-fetching is disabled
Enable bursting on descriptor read master	1 or 0	1: Pre-fetching module uses Avalon-MM bursting when fetching descriptors.
Data Width (Avalon-MM Read/Write Descriptor)	32, 64, 128, 256, 512	Specifies the read and write data width of Avalon-MM read and write descriptor master.
Maximum Burst Count (Avalon-MM Read Descriptor)	1, 2, 4, 8, 16	Specifies the maximum read burst count of Avalon-MM read descriptor master.
Enable Extended Feature Support	1 or 0	This is a derived parameter from the mSGDMA top level composed. This is needed by this core to determine descriptor length (different length for standard/ extended descriptor).
FIFO Depth	8, 16, 32, 64, 128, 256, 512, 1024	This is a derived parameter from the mSGDMA top level composed. This is needed by this core to determine its buffer depth to store next descriptor pointer information for descriptor write back.

Driver Implementation

Following section contains the APIs for the mSGDMA HAL Driver. An open mSGDMA API will instantiate an mSGDMA device with optional register interrupt service routine (ISR). You must define your own specific handling mechanism in the callback function when using an ISR. A callback function will be called by the ISR on error, early termination, and on transfer complete.

alt_msgdma_standard_descriptor_async_transfer

Table 25-25: alt_msgdma_standard_descriptor_async_transfer

Prototype:	int alt_msgdma_standard_descriptor_async_transfer(alt_msgdma_dev *dev, alt_msgdma_standard_descriptor *desc)
Include:	< modular_sgdma_dispatcher.h >, < altera_msgdma_csr_regs.h>, <altera_msgdma_descriptor_regs.h>, <sys/alt_errno.h>, <sys/alt_irq.h>, <io.h>
Parameters:	*dev — a pointer to msgdma instance. *desc — a pointer to a standard descriptor structure
Returns:	"0" for success, –ENOSPC indicates FIFO buffer is full, –EPERM indicates operation not permitted due to descriptor type conflict, -ETIME indicates Time out and skipping the looping after 5 msec.
Description:	A descriptor needs to be constructed and passing as a parameter pointer to *desc when calling this function. This function will call the helper function "alt_msgdma_descriptor_async_transfer" to start a non-blocking transfer of one standard descriptor at a time. If the FIFO buffer for a read/write is full at the time of this call, the routine will immediately return –ENOSPC, the application can then decide how to proceed without being blocked. -ETIME will be returned if the time spending for writing the descriptor to the dispatcher takes longer than 5 msec. You are advised to refer to the helper function for details. If a callback routine has been previously registered with this particular mSGDMA controller, the transfer will be set up to enable interrupt generation.

alt_msgdma_extended_descriptor_async_transfer

Table 25-26: alt_msgdma_extended_descriptor_async_transfer

Prototype:	int alt_msgdma_extended_descriptor_async_transfer(alt_msgdma_dev *dev, alt_msgdma_extended_descriptor *desc)
Include:	< modular_sgdma_dispatcher.h >, < altera_msgdma_csr_regs.h>, <altera_msgdma_descriptor_regs.h>, <sys/alt_errno.h>, <sys/alt_irq.h>, <io.h>
Parameters:	*dev — a pointer to mSGDMA instance. *desc — a pointer to an extended descriptor structure
Returns:	"0" for success, –ENOSPC indicates FIFO buffer is full, –EPERM indicates operation not permitted due to descriptor type conflict, -ETIME indicates time out and skipping the looping after 5 msec.
Description:	A descriptor needs to be constructed and passing as a parameter pointer to the *desc when calling this function. This function will call the helper function "alt_msgdma_descriptor_async_transfer" to start a non-blocking transfer of one standard descriptor at a time. If the FIFO buffer for a read/write is full at the time of this call, the routine will immediately return –ENOSPC, the application can then decide how to proceed without being blocked.-ETIME will be returned if the time spending for writing descriptor to the dispatcher takes longer than 5 msec. You are advised to refer the helper function for details. If a callback routine has been previously registered with this particular mSGDMA controller, the transfer will be set up to enable interrupt generation.

alt_msgdma_descriptor_async_transfer

Table 25-27: alt_msgdma_descriptor_async_transfer

Prototype:	static int alt_msgdma_descriptor_async_transfer(alt_msgdma_dev *dev, alt_msgdma_standard_descriptor *standard_desc, alt_msgdma_extended_descriptor *extended_desc)
Include:	< modular_sgdma_dispatcher.h >, < altera_msgdma_csr_regs.h>, <altera_msgdma_descriptor_regs.h>, <sys/alt_errno.h>, <sys/alt_irq.h>, <io.h>
Parameters:	*dev — a pointer to mSGDMA instance. *standard_desc — Pointer to single standard descriptor. *extended_desc — Pointer to single extended descriptor.
Returns:	“0” for success, –ENOSPC indicates FIFO buffer is full, –EPERM indicates operation not permitted due to descriptor type conflict, -ETIME indicates Time out and skipping the looping after 5 msec.
Description:	Helper functions for both “alt_msgdma_standard_descriptor_async_transfer” and “alt_msgdma_extended_descriptor_async_transfer”. **Note:** Either one of both *standard_desc and *extended_desc must be assigned with NULL, another with proper pointer value. Failing to do so can cause the function return with "-EPERM ". If a callback routine has been previously registered with this particular mSGDMA controller, the transfer will be set up to enable interrupt generation. It is the responsibility of the application developer to check source interruption, status completion and creating suitable interrupt handling. **Note:** "stop on error" of the CSR control register is always masking within this function. The CSR control can be set by user through calling "alt_register_callback" with user defined control setting.

alt_msgdma_standard_descriptor_sync_transfer

Table 25-28: alt_msgdma_standard_descriptor_sync_transfer

Prototype:	int alt_msgdma_standard_descriptor_sync_transfer(alt_msgdma_dev *dev, alt_msgdma_standard_descriptor *desc)
Include:	< modular_sgdma_dispatcher.h >, < altera_msgdma_csr_regs.h>, <altera_msgdma_descriptor_regs.h>, <sys/alt_errno.h>, <sys/alt_irq.h>, <io.h>
Parameters:	*dev — a pointer to mSGDMA instance. *desc — a pointer to a standard descriptor structure
Returns:	"0" for success, "error" indicates errors or conditions causing msgdma stop issuing commands to masters, suggest checking the bit set in the error with CSR status register."-EPERM" indicates operation not permitted due to descriptor type conflict. "-ETIME" indicates Time out and skipping the looping after 5 msec.
Description:	A standard descriptor needs to be constructed and passing as a parameter pointer to *desc when calling this function. This function will call helper function "alt_msgdma_descriptor_sync_transfer" to start a blocking transfer of one standard descriptor at a time. If the FIFO buffer for a read or write is full at the time of this call, the routine will wait until a free FIFO buffer is available to continue processing or a 5 msec time out. The function will return "error" if errors or conditions causing the dispatcher to stop issuing the commands to both the read and write masters before both the read and write command buffers are empty. It is the responsibility of the application developer to check errors and completion status.

alt_msgdma_extended_descriptor_sync_transfer

Table 25-29: alt_msgdma_extended_descriptor_sync_transfer

Prototype:	int alt_msgdma_extended_descriptor_sync_transfer(alt_msgdma_dev *dev, alt_msgdma_extended_descriptor *desc)
Include:	< modular_sgdma_dispatcher.h >, < altera_msgdma_csr_regs.h>, <altera_msgdma_descriptor_regs.h>, <sys/alt_errno.h>, <sys/alt_irq.h>, <io.h>
Parameters:	*dev — a pointer to msgdma instance. *desc — a pointer to an extended descriptor structure
Returns:	"0" for success, "error" indicates errors or conditions causing msgdma stop issuing commands to masters, suggest checking the bit set in the error with CSR status register."-EPERM" indicates operation not permitted due to descriptor type conflict. "-ETIME" indicates Time out and skipping the looping after 5 msec.
Description:	An extended descriptor needs to be constructed and passing as a parameter pointer to *desc when calling this function. This function will call helper function "alt_msgdma_descriptor_sync_transfer" to startcommencing a blocking transfer of one extended descriptor at a time. If the FIFO buffer for one of read or write is full at the time of this call, the routine will wait until free FIFO buffer available for continue processing or 5 msec time out. The function will return "error" if errors or conditions causing the dispatcher stop issuing the commands to both read and write masters before both read and write command buffers are empty. It is the responsibility of the application developer to check errors and completion status. -ETIME will be returned if the time spending for waiting the FIFO buffer, writing descriptor to the dispatcher and any pending transfer to complete take longer than 5msec.

alt_msgdma_descriptor_sync_transfer

Table 25-30: alt_msgdma_descriptor_sync_transfer

Prototype:	int alt_msgdma_descriptor_sync_transfer(alt_msgdma_dev *dev, alt_msgdma_standard_descriptor *standard_desc, alt_msgdma_extended_descriptor *extended_desc)
Include:	< modular_sgdma_dispatcher.h >, < altera_msgdma_csr_regs.h>, <altera_msgdma_descriptor_regs.h>, <sys/alt_errno.h>, <sys/alt_irq.h>, <io.h>
Parameters:	*dev — a pointer to msgdma instance. *standard_desc — Pointer to single standard descriptor. *extended_desc — Pointer to single extended descriptor.
Returns:	“0” for success, “error” indicates errors or conditions causing msgdma stop issuing commands to masters, suggest checking the bit set in the error with CSR status register.”-EPERM” indicates operation not permitted due to descriptor type conflict. “-ETIME” indicates Time out and skipping the looping after 5 msec.
Description:	Helper functions for both “alt_msgdma_standard_descriptor_sync_transfer” and “alt_msgdma_extended_descriptor_sync_transfer”. **Note:** Either one of both *standard_desc and *extended_desc must be assigned with NULL, another with proper pointer value. Failing to do so can cause the function return with "-EPERM . **Note:** "stop on error" of CSR control register is always being masked and the interrupt is always disabled within this function. The CSR control can be set by user through calling "alt_register_callback" with user defined control setting.

alt_msgdma_construct_standard_st_to_mm_descriptor

Table 25-31: alt_msgdma_construct_standard_st_to_mm_descriptor

Prototype:	int alt_msgdma_construct_standard_st_to_mm_descriptor (alt_msgdma_dev *dev, alt_msgdma_standard_descriptor *descriptor, alt_u32 *write_address, alt_u32 length, alt_u32 control)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev-a pointer to msgdma instance. *descriptor – a pointer to a standard descriptor structure. *write_address – a pointer to the base address of the destination memory. length - is used to specify the number of bytes to transfer per descriptor. The largest possible value can be filled in is "0Xffffffff". control – control field.
Returns:	"0" for success, -EINVAL for invalid argument, could be due to argument which has larger value than hardware setting value.
Description:	Function will call helper function "alt_msgdma_construct_standard_descriptor" for constructing st_to_mm standard descriptors. Unnecessary elements are set to 0 for completeness and will be ignored by the hardware.

alt_msgdma_construct_standard_mm_to_st_descriptor

Table 25-32: alt_msgdma_construct_standard_mm_to_st_descriptor

Prototype:	int alt_msgdma_construct_standard_mm_to_st_descriptor (alt_msgdma_dev *dev, alt_msgdma_standard_descriptor *descriptor, alt_u32 *read_address, alt_u32 length, alt_u32 control)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev-a pointer to msgdma instance. *descriptor – a pointer to a standard descriptor structure. *read_address – a pointer to the base address of the source memory. length – is used to specify the number of bytes to transfer per descriptor. The largest possible value can be filled in is "0Xffffffff". control – control field.
Returns:	"0" for success, -EINVAL for invalid argument, could be due to argument which has larger value than hardware setting value.
Description:	Function will call helper function "alt_msgdma_construct_standard_descriptor" for constructing mm_to_st standard descriptors. Unnecessary elements are set to 0 for completeness and will be ignored by the hardware.

alt_msgdma_construct_standard_mm_to_mm_descriptor

Table 25-33: alt_msgdma_construct_standard_mm_to_mm_descriptor

Prototype:	int alt_msgdma_construct_standard_mm_to_mm_descriptor (alt_msgdma_dev *dev, alt_msgdma_standard_descriptor *descriptor, alt_u32 *read_address, alt_u32 *write_address, alt_u32 length, alt_u32 control)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev-a pointer to msgdma instance. *descriptor – a pointer to a standard descriptor structure. *read_address – a pointer to the base address of the source memory. *write_address – a pointer to the base address of the destination memory. length – is used to specify the number of bytes to transfer per descriptor. The largest possible value can be filled in is "0Xffffffff". control – control field.
Returns:	"0" for success, -EINVAL for invalid argument, could be due to argument which has larger value than hardware setting value.
Description:	Function will call helper function "alt_msgdma_construct_standard_descriptor" for constructing mm_to_mm standard descriptors. Unnecessary elements are set to 0 for completeness and will be ignored by the hardware.

alt_msgdma_construct_standard_descriptor

Table 25-34: alt_msgdma_construct_standard_descriptor

Prototype:	static int alt_msgdma_construct_standard_descriptor (alt_msgdma_dev *dev, alt_msgdma_standard_descriptor *descriptor, alt_u32 *read_address, alt_u32 *write_address, alt_u32 length, alt_u32 control)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev-a pointer to msgdma instance. *descriptor – a pointer to a standard descriptor structure. *read_address – a pointer to the base address of the source memory. *write_address – a pointer to the base address of the destination memory. length - is used to specify the number of bytes to transfer per descriptor. The largest possible value can be filled in is "0Xffffffff". control – control field.
Returns:	"0" for success, -EINVAL for invalid argument, could be due to argument which has larger value than hardware setting value.
Description:	Helper functions for constructing mm_to_st, st_to_mm, mm_to_mm standard descriptors. Unnecessary elements are set to 0 for completeness and will be ignored by the hardware.

alt_msgdma_construct_extended_st_to_mm_descriptor

Table 25-35: alt_msgdma_construct_extended_st_to_mm_descriptor

Prototype:	int alt_msgdma_construct_extended_st_to_mm_descriptor (alt_msgdma_dev *dev, alt_msgdma_extended_descriptor *descriptor, alt_u32 *write_address, alt_u32 length, alt_u32 control, alt_u16 sequence_number, alt_u8 write_burst_count, alt_u16 write_stride)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev-a pointer to msgdma instance. *descriptor – a pointer to an extended descriptor structure. *write_address – a pointer to the base address of the destination memory. length – is used to specify the number of bytes to transfer per descriptor. The largest possible value can be filled in is "0Xffffffff". control – control field. sequence number – programmable sequence number to identify which descriptor has been sent to the master block. write_burst_count – programmable burst count between 1 and 128 and a power of 2. Setting to 0 will cause the master to use the maximum burst count instead. write_stride – programmable transfer stride. The stride value determines by how many words the master will increment the address. For fixed addresses the stride value is 0, sequential it is 1, every other word it is 2, etc…power of 2. Setting to 0 will cause the master to use the maximum burst count instead. write_stride – programmable transfer stride. The stride value determines by how many words the master will increment the address. For fixed addresses the stride value is 0, sequential it is 1, every other word it is 2, etc…
Returns:	"0" for success, -EINVAL for invalid argument, could be due to argument which has larger value than hardware setting value.
Description:	Function will call helper function "alt_msgdma_construct_extended_descriptor" for constructing st_to_mm extended descriptors. Unnecessary elements are set to 0 for completeness and will be ignored by the hardware.

alt_msgdma_construct_extended_mm_to_st_descriptor

Table 25-36: alt_msgdma_construct_extended_mm_to_st_descriptor

Prototype:	int alt_msgdma_construct_extended_mm_to_st_descriptor (alt_msgdma_dev *dev, alt_msgdma_extended_descriptor *descriptor, alt_u32 *read_address, alt_u32 length, alt_u32 control, alt_u16 sequence_number, alt_u8 read_burst_count, alt_u16 read_stride)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev-a pointer to msgdma instance. *descriptor – a pointer to an extended descriptor structure. *read_address – a pointer to the base address of the source memory. length – is used to specify the number of bytes to transfer per descriptor. The largest possible value can be filled in is "0Xffffffff". control – control field. sequence_number – programmable sequence number to identify which descriptor has been sent to the master block. read_burst_count – programmable burst count between 1 and 128 and a power of 2. Setting to 0 will cause the master to use the maximum burst count instead. read_stride – programmable transfer stride. The stride value determines by how many words the master will increment the address. For fixed addresses the stride value is 0, sequential it is 1, every other word it is 2, etc…
Returns:	"0" for success, -EINVAL for invalid argument, could be due to argument which has larger value than hardware setting value.
Description:	Function call helper function "alt_msgdma_construct_extended_descriptor" for constructing mm_to_st extended descriptors. Unnecessary elements are set to 0 for completeness and will be ignored by the hardware.

alt_msgdma_construct_extended_mm_to_mm_descriptor

Table 25-37: alt_msgdma_construct_extended_mm_to_mm_descriptor

Prototype:	int alt_msgdma_construct_extended_mm_to_mm_descriptor (alt_msgdma_dev *dev, alt_msgdma_extended_descriptor *descriptor, alt_u32 *read_address, alt_u32 *write_address, alt_u32 length, alt_u32 control, alt_u16 sequence_number, alt_u8 read_burst_count, alt_u8 write_burst_count, alt_u16 read_stride, alt_u16 write_stride)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev-a pointer to msgdma instance. *descriptor – a pointer to an extended descriptor structure. *read_address – a pointer to the base address of the source memory. *write_address – a pointer to the base address of the destination memory. length – is used to specify the number of bytes to transfer per descriptor. The largest possible value can be filled in is "0Xffffffff". control – control field. sequence_number – programmable sequence number to identify which descriptor has been sent to the master block. read_burst_count – programmable burst count between 1 and 128 and a power of 2. Setting to 0 will cause the master to use the maximum burst count instead. write_burst_count – programmable burst count between 1 and 128 and a power of 2. Setting to 0 will cause the master to use the maximum burst count instead. read_stride – programmable transfer stride. The stride value determines by how many words the master will increment the address. For fixed addresses the stride value is 0, sequential it is 1, ever other word it is 2, etc… write_stride – programmable transfer stride. The stride value determines by how many words the master will increment the address. For fixed addresses the stride value is 0, sequential it is 1, every other word it is 2, etc…
Returns:	"0" for success, -EINVAL for invalid argument, could be due to argument which has larger value than hardware setting value.
Description:	Function call helper function "alt_msgdma_construct_extended_descriptor" for constructing mm_to_mm extended descriptors. Unnecessary elements are set to 0 for completeness and will be ignored by the hardware.

alt_msgdma_construct_extended_descriptor

Table 25-38: alt_msgdma_construct_extended_descriptor

Prototype:	static int alt_msgdma_construct_descriptor (alt_msgdma_dev *dev, alt_msgdma_extended_descriptor *descriptor, alt_u32 *read_address, alt_u32 *write_address, alt_u32 length, alt_u32 control, alt_u16 sequence_number, alt_u8 read_burst_count, alt_u8 write_burst_count, alt_u16 read_stride, alt_u16 write_stride)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev-a pointer to msgdma instance. *descriptor – a pointer to an extended descriptor structure. *read_address – a pointer to the base address of the source memory. *write_address – a pointer to the base address of the destination memory. length – is used to specify the number of bytes to transfer per descriptor. The largest possible value can be filled in is "0Xffffffff". control – control field. sequence_number – programmable sequence number to identify which descriptor has been sent to the master block. read_burst_count – programmable burst count between 1 and 128 and a power of 2. Setting to 0 will cause the master to use the maximum burst count instead. write_burst_count – programmable burst count between 1 and 128 and a power of 2. Setting to 0 will cause the master to use the maximum burst count instead. read_stride – programmable transfer stride. The stride value determines by how many words the master will increment the address. For fixed addresses the stride value is 0, sequential it is 1, ever other word it is 2, etc… write_stride – programmable transfer stride. The stride value determines by how many words the master will increment the address. For fixed addresses the stride value is 0, sequential it is 1, every other word it is 2, etc…
Returns:	"0" for success, -EINVAL for invalid argument, could be due to argument which has larger value than hardware setting value.
Description:	Helper functions for constructing mm_to_st, st_to_mm, mm_to_mm extended descriptors. Unnecessary elements are set to 0 for completeness and will be ignored by the hardware.

alt_msgdma_register_callback

Table 25-39: alt_msgdma_register_callback

Prototype:	void alt_msgdma_register_callback(alt_msgdma_dev *dev, alt_msgdma_callback callback, alt_u32 control, void *context)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*dev — a pointer to msgdma instance. callback — Pointer to callback routine to execute at interrupt level control — Setting control register and OR with other control bits in the non_ blocking and blocking transfer function. *context — pointer to user define context
Returns:	N/A
Description:	Associate a user-specific routine with the mSGDMA interrupt handler. If a callback is registered, all non-blocking mSGDMA transfers will enable interrupts that will cause the callback to be executed. The callback runs as part of the interrupt service routine, and great care must be taken to follow the guidelines for acceptable interrupt service routine behavior as described in the Nios II Software Developer's Handbook. However, user can change some of the CSR control setting in blocking transfer by calling this function.

alt_msgdma_open

Table 25-40: alt_msgdma_open

Prototype:	alt_msgdma_dev* alt_msgdma_open (const char* name)
Include:	< modular_sgdma_dispatcher.h >
Parameters:	*name — Character pointer to name of msgdma peripheral as registered with the HAL. For example, an mSGDMA in Qsys would be opened by asking for “MSGDMA_0_DISPATCHER_INTERNAL".
Returns:	Pointer to msgdma device instance struct, or null if the device. * could not be opened.
Description:	Retrieves a pointer to the mSGDMA instance.

alt_msgdma_write_standard_descriptor

Table 25-41: alt_msgdma_write_standard_descriptor

Prototype:	int alt_msgdma_write_standard_descriptor (alt_u32 csr_base, alt_u32 descriptor_base, alt_msgdma_standard_descriptor *descriptor)
Include:	< modular_sgdma_dispatcher.h >, <altera_msgdma_descriptor_regs.h>
Parameters:	csr_base – base address of the dispatcher CSR slave port. descriptor_base – base address of the dispatcher descriptor slave port. *descriptor – a pointer to a standard descriptor structure.
Returns:	Returns 0 upon success. Other return codes are defined in "alt_errno.h".
Description:	Sends a fully formed standard descriptor to the dispatcher module. If the dispatcher descriptor buffer is full, “-ENOSPC” is returned. This function is not reentrant since it must complete writing the entire descriptor to the dispatcher module and cannot be pre-empted.

alt_msgdma_write_extended_descriptor

Table 25-42: alt_msgdma_write_extended_descriptor

Prototype:	int alt_msgdma_write_extended_descriptor (alt_u32 csr_base, alt_u32 descriptor_base, alt_msgdma_extended_descriptor *descriptor)
Include:	< modular_sgdma_dispatcher.h >, <altera_msgdma_descriptor_regs.h>
Parameters:	csr_base – base address of the dispatcher CSR slave port. descriptor_base – base address of the dispatcher descriptor slave port. *descriptor – a pointer to an extended descriptor structure.
Returns:	Returns 0 upon success. Other return codes are defined in "alt_errno.h".
Description:	Sends a fully formed extended descriptor to the dispatcher module. If the dispatcher descriptor buffer is full an error is returned. This function is not reentrant since it must complete writing the entire descriptor to the dispatcher module and cannot be pre-empted.

alt_avalon_msgdma_init

Table 25-43: alt_avalon_msgdma_init

Prototype:	void alt_msgdma_init (alt_msgdma_dev *dev, alt_u32 ic_id, alt_u32 irq)
Include:	< modular_sgdma_dispatcher.h >, <altera_msgdma_descriptor_regs.h>, <altera_msgdma_csr_regs.h>
Parameters:	*dev – a pointer to mSGDMA instance. ic_id – id of irq interrupt controller irq – irq number that belonged to mSGDMA instance
Returns:	N/A
Description:	Initializes the mSGDMA controller. This routine is called from the ALTERA_AVALON_MSGDMA_INIT macro and is called automatically by "alt_sys_init.c".

alt_msgdma_irq

Table 25-44: alt_msgdma_irq

Prototype:	void alt_msgdma_irq(void *context)
Include:	< modular_sgdma_dispatcher.h >, <sys/alt_irq.h>, <altera_msgdma_csr_regs.h>
Parameters:	*context – a pointer to mSGDMA instance.
Returns:	N/A
Description:	Interrupt handler for mSGDMA. This function will call the user defined interrupt handler if user registers their own interrupt handler with calling “alt_register_callback”.

Document Revision History

Table 25-45: Document Revision History

Date	Version	Changes
May 2016	2016.05.03	Updated tables: • Table 25-2
December 2015	2015.12.16	Added "alt_msgdma_irq" section.

November 2015	2015.11.06	Updated sections:
		• Response Port
		• Component Parameters
		Sections added:
		• Programming Model
		  • Stop DMA Operation
		  • Stop Descriptor Operation
		  • Recovery from Stopped on Error and Stopped on Early Termination
		• Modular Scatter-Gather DMA Prefetcher Core
		• Driver Implementation
		Section removed:
		• Unsupported Feature

July 2014	2014.07.24	Initial release