Modular SGDMA Dispatcher Core

Author: JCJB
Date: 09/21/2009

Core Overview

The modular scatter-gather direct memory access (SGDMA) dispatcher core is responsible for buffering descriptors and controlling the read and write master cores. You can configure the dispatcher core for the following transfer types:

- Memory to memory
- Data stream to memory
- Memory to data stream

The dispatcher block forms the control path of the modular SGDMA. It connects to the read and write master modules which form the data path of the modular SGDMA. As a result all the interaction with the modular SGDMA involves the dispatcher core.

The dispatcher module contains optional features that allows you to reduce the hardware footprint or increase the overall frequency (Fmax) for the system. The dispatcher core is accompanied by Nios® II processor device drivers allowing you to develop software that uses the modular SGDMA in the Nios II IDE or software build tools.

Block Diagram

Figure 1 shows the significant blocks that make up the dispatcher core.

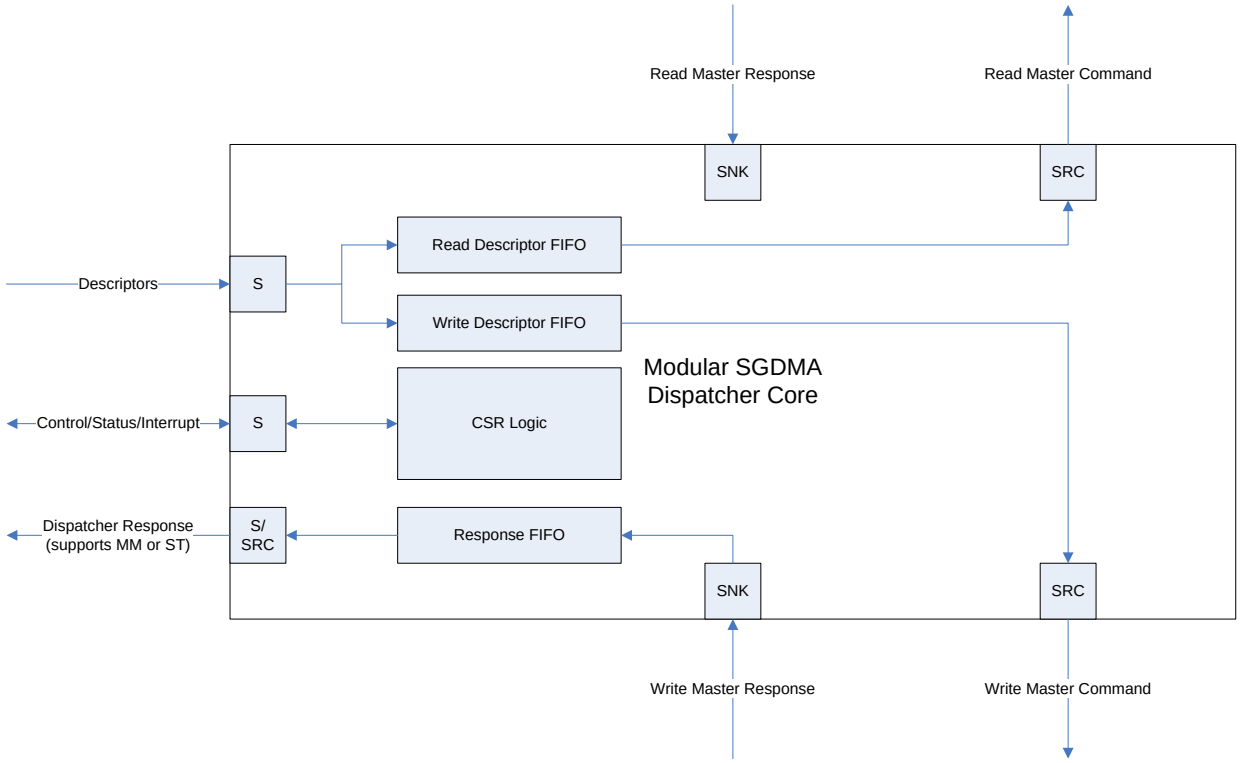

Figure 1. Modular SGDMA Dispatcher Module

Some of the ports are optional, for example the read master command and response ports are not exposed when the modular SGDMA is configured for streaming to memory transfers. When any of the ports are disabled the logic and on-chip memory footprint of the dispatcher core decreases.

The dispatcher response slave port is only used for streaming to memory transfers to communicate information such as errors back to the host. The dispatcher response port is also necessary if a SGDMA descriptor pre-fetching module is added to the modular architecture. In this case the response port should be configured to be a streaming source port.

Common Modular SGDMA Configurations

There are three main modular SGDMA configurations that suit the needs for most system designers as shown below.

Memory-Mapped to Streaming

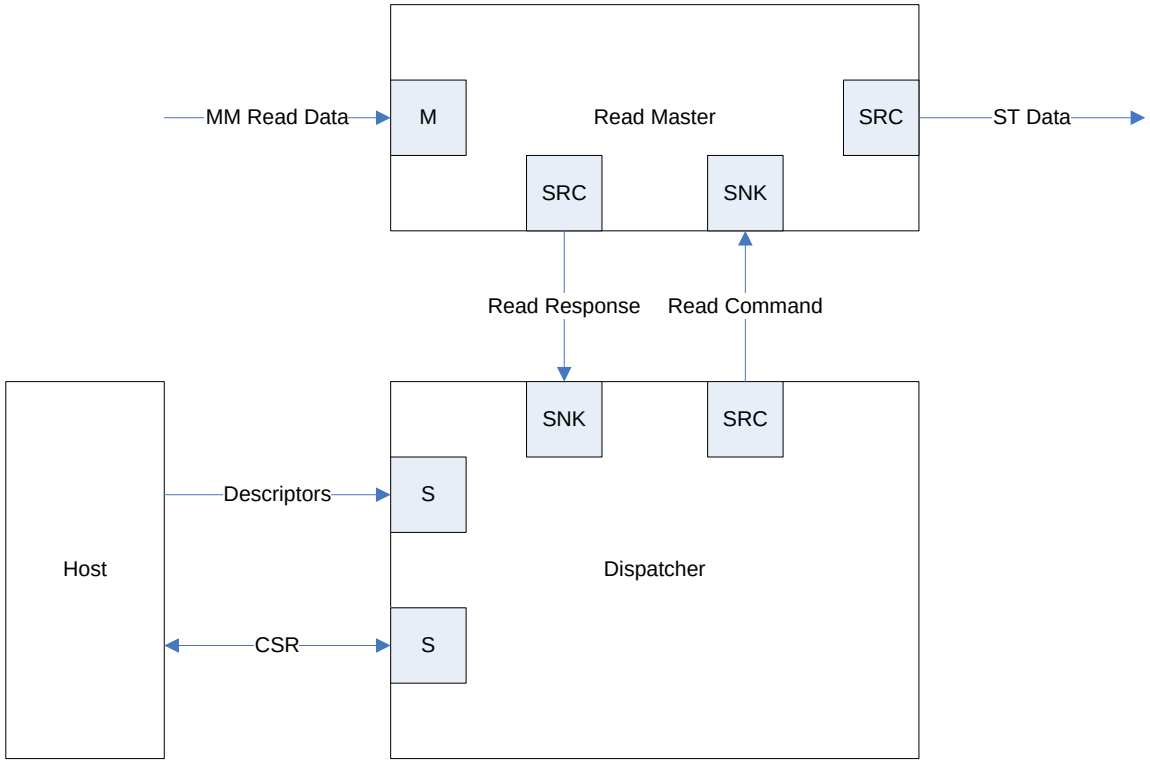

Figure 2. Memory-Mapped to Streaming Modular SGDMA Architecture

Streaming to Memory-Mapped

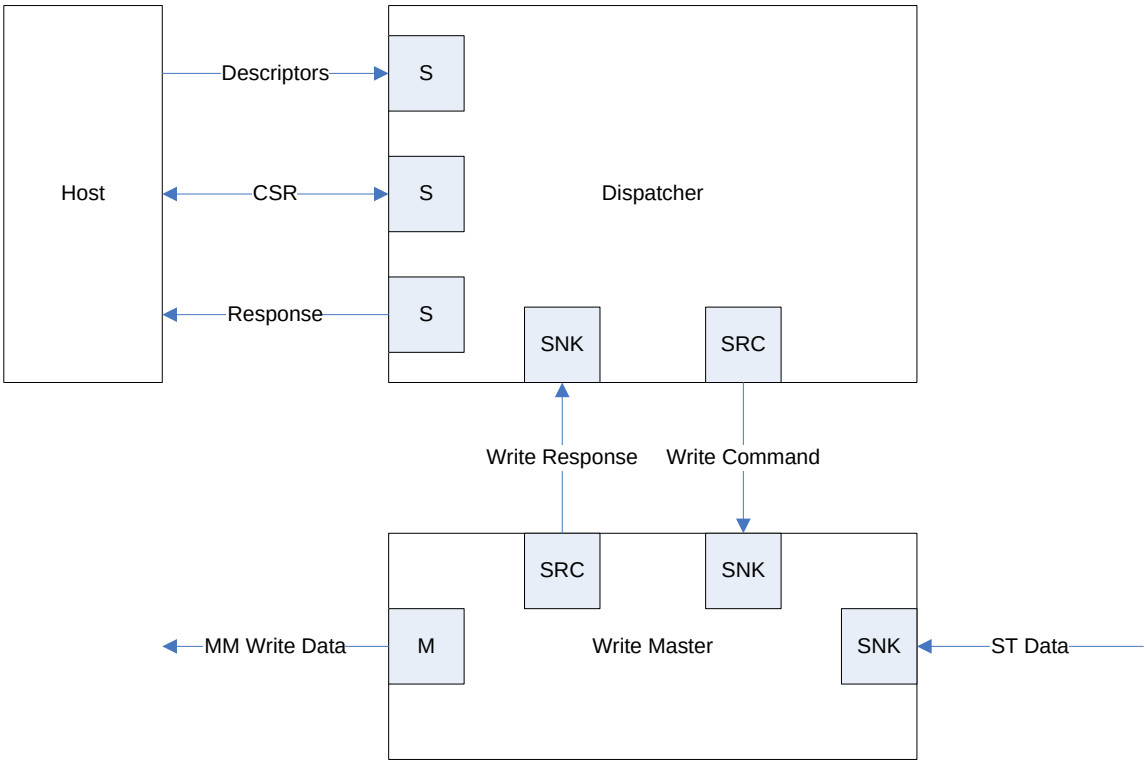

Figure 3. Streaming to Memory-Mapped Modular SGDMA Architecture

Memory-Mapped to Memory-Mapped

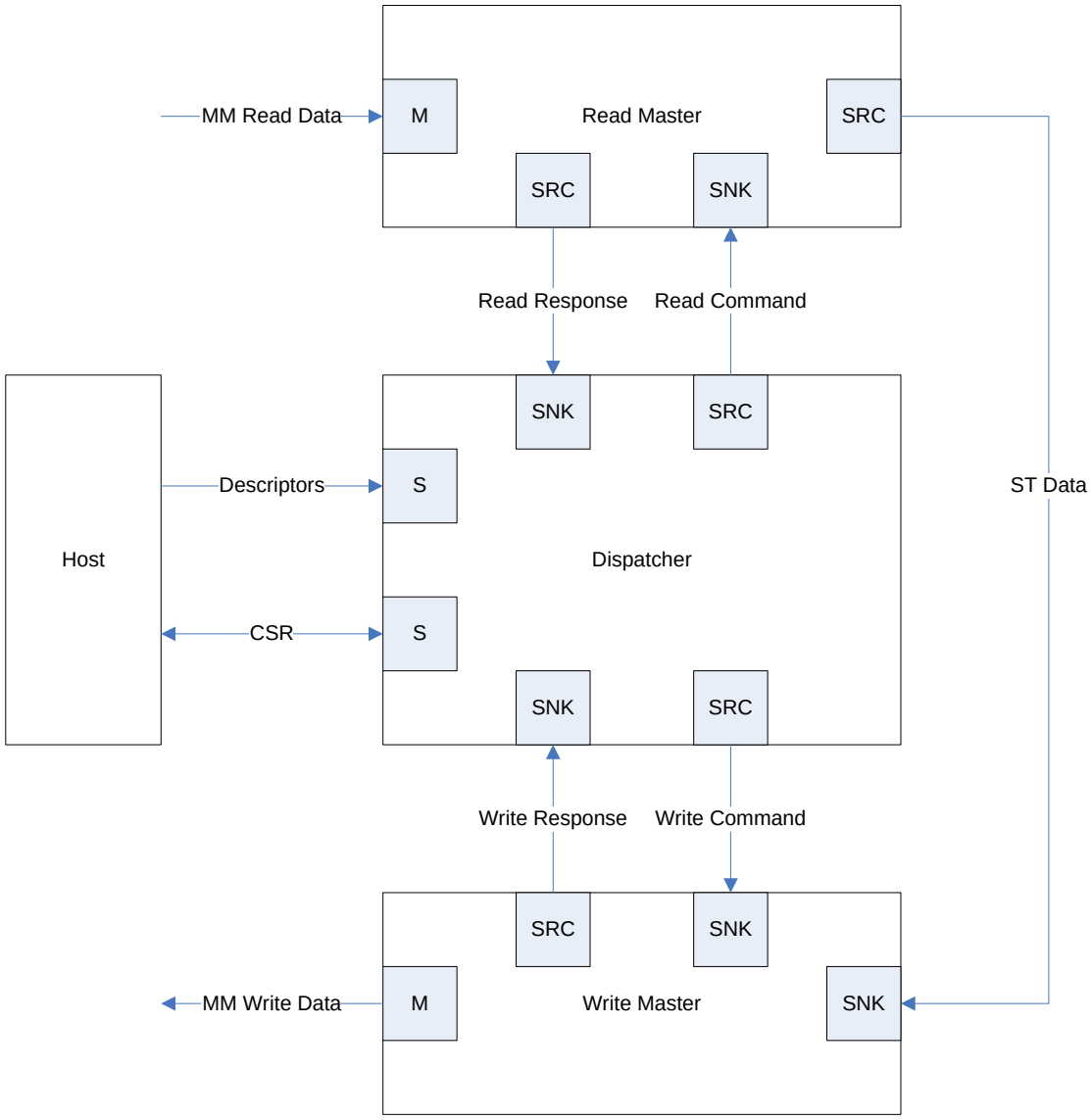

Figure 4. Memory-Mapped to Memory-Mapped Modular SGDMA Architecture

Dispatcher Port Listing

This section will detail the various Avalon-MM and ST ports that are exposed by the modular SGDMA dispatcher block. Some of the signals and port are optionally removed depending on the settings you create in the component GUI. Also some of the signals outlined below are not used by the dispatcher block and are only documented for completion.

Read Commands Source Port

Bits	**Signal Information**
31-0	Read Address [31:0]
63-32	Length [31:0]
71-64	Transmit Channel [7:0]
72	Generate SOP
73	Generate EOP
74	Stop[1]
75	Reset[1]
83-76	Read Burst Count [7:0]
99-84	Read Stride [15:0]
107-100	Transmit Error [7:0]
108	Early Done Enable
140-109	Read Address [63:32]
255-141	<reserved>[2]

Table 1. Read Commands Source Port Bitfields

[1] Combinational signals that don't obey flow control
[2] Reserved bits driven to ground

Read Response Sink Port

Bits	**Signal Information**
0	Reset Delayed[1]
1	Stop State[1]
2	Early Done Strobe[2]
3	Done Strobe[3]
255-4	<reserved>

Table 2. Read Response Sink Port Bitfields

[1] Combinational signals that don't obey flow control
[2] Asserted when last read has been posted
[3] Asserted when all data has been read

Write Commands Source Port

Bits	Signal Information
31-0	Write Address [31:0]
63-32	Length [31:0]
64	End on EOP
65	<reserved>
66	Stop[1]
67	Reset[1]
75-68	Write Burst Count [7:0]
91-76	Write Stride [15:0]
123-92	Write Address [63:32]
255-124	<reserved>

Figure 3. Write Commands Source Port Bitfields

[1] Combinational signals that don't obey flow control
[2] Reserved bits driven to ground

Write Response Sink Port

Bits	Signal Information
31-0	Actual Bytes Transferred [31:0]
32	Reset Delayed[1]
33	Stop State[1]
41-34	Error [7:0]
42	Early Termination
43	Done Strobe
255-44	<reserved>

Figure 4. Write Response Sink Port Bitfields

[1] Combinational signals that don't obey flow control

Response Source Port

The response port can be configured as an Avalon-MM slave port or ST source port. When configured as a source port you should connect it to a module capable of pre-fetching descriptors from memory. At the time of this writing the pre-fetching module is not available.

Bits	Signal Information
31-0	Actual bytes transferred [31:0]
39-32	Error[7:0]
40	Early termination
41	Transfer complete IRQ mask[1]
49-42	Error IRQ mask[1]
50	Early termination IRQ mask[1]
51	Descriptor buffer full[2]
255-52	<reserved>

Figure 5. Response Source Port Bitfields

[1] Interrupt masks are buffered so that descriptor pre-fetching block can assert the IRQ signal
[2] 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)

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 to use in your system. The layout of the descriptor information will be outlined in the software programming model section of this document. It is important to note that when writing descriptors to this port you must set the last bit high for the descriptor to 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.

CSR Slave Port

The control and status register 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 as well. The register map will be outlined in the software programming model section of this document.

Response Slave 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 a slave port. The response information is wider than the slave port so the host must perform two read operations to retrieve all the information. 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. The response slave port memory map will be shown in the software programming model section of this document.

Software Programming Model

The modular SGDMA is architected to be feature rich as well as consume a minimal hardware footprint whenever possible. As a result there are two descriptor formats recognized by the modular SGDMA called 'standard' and 'extended'. The standard format contains the minimal set of features needed for any transfer. The extended format contains all the features contained in the standard format as well as other additional features. In order to use the extended descriptor format you must select 'enhanced features' in the dispatcher module graphical user interface. You must also enable the appropriate enhanced features in the master modules.

Software Files

The modular SGDMA dispatcher module provides the following device driver files:

- **descriptor_regs.h –** defines the register map of the descriptor slave port, providing symbolic constants to access the low-level hardware.
- **csr_regs.h –** defines the register map of the csr slave port, providing symbolic constants to access the low-level hardware.
- **response_regs.h –** defines the register map of the optional response slave port, providing symbolic constants to access the low-level hardware.
- **sgdma_dispatcher.h –** defines the descriptor formats used by the dispatcher core, also provides prototypes for descriptor construction and accessing the descriptor, csr, and response slave ports.
- **sgdma_dispatcher.c –** provides descriptor construction and accessor functions for the descriptor, csr, and response slave ports.

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 6. Standard Descriptor Format

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]			

Table 7. Extended Descriptor Format

Descriptor Fields

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

Important to note is that control field of the descriptors is located at a different offset depending on the format 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 always located at the end of a descriptor to allow the host 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 the read or write address is will not need to be provided. When performing memory-mapped to streaming transfers the write address should not be written as 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 should not be written as the data source is a streaming port.

If the read or write address field is written to in a configuration where that address is not needed the modular SGDMA will simply ignore the unnecessary address. Writing address bits that are not accessible to the modular SGDMA will be lost and can cause the hardware to alias into 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 is specified in bytes. The length field is also used for streaming to memory-mapped packet transfers to limit 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 ST→MM program the length field with the largest possible value of 0xFFFFFFFF. This allows you to specify a maximum packet size for each descriptor that packet support has been enabled.

Sequence Number Field

The sequence number field is available only when the dispatcher enhanced features are enabled. The sequence number is an arbitrary value that you assign to a descriptor so that you can 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 simply program the descriptors to have unique sequence numbers then access the dispatcher CSR slave port to determine which descriptor is being 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. Since the programmable burst count is an eight bit field for each master you can at most only program burst counts of 1-128. Programming a value of zero or anything larger than 128 beats will be converted to the maximum burst count specified for each master automatically.

Programmable burst counts are only recommended for modular SGDMA transfers between slave ports that specify different maximum burst counts. For example if the modular SGDMA is used to transfer data to slave ports that specify maximum burst counts of 2, 64, and 128 you would set the maximum burst count to 128 and program the burst counts of 2, 64, and 128 for the transfers to each slave port. Since burst transfers lock the arbitration posting large bursts to slave ports that can't handle large bursts may not be beneficial which the programmable burst count serves to address. To learn more about bursting and the arbitration logic created by SOPC Builder refer to the Avalon-MM Design Optimizations Guide.

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 will 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 1. A stride of 0 instructs the master to continuously access the same address. A stride of 2 instructs the master to skip every other word in a sequential transfer. You can use this feature to perform interleaved data accesses or 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 2 and the write master for a stride distance of 1. To complete the decimation operation you could also insert a filter between the two masters as well.

Control Field

The control field which is available for both the standard and extended descriptor formats. You use this field to program characteristics of the transfer like parked descriptors, error handling, and interrupt masks. The interrupt masks are programmed into the descriptor so that interrupt enables can be unique for each transfer.

Bit Offset	Name
7-0	Transmit Channel
8	Generate SOP
9	Generate EOP
10	Park Reads
11	Park Writes
12	End on EOP
13	<reserved>
14	Transfer Complete IRQ Mask
15	Early Termination IRQ Mask
23-16	Transmit Error/Error IRQ Mask
24	Early done enable
30-25	<reserved>
31	Go[1]

Table 8. Descriptor Control Register Bitfield

[1] *Writing '1' to the 'go' bit commits the entire descriptor into the descriptor FIFO(s)*

Transmit Channel – Used to emit a channel number during MM→ST transfers

Generate SOP – Used to emit a start of packet on the first beat of a MM→ST transfer

Generate EOP – Used to emit an end of packet on last beat of a MM → ST transfer

Park Reads – When set the dispatcher will continue to reissue the same descriptor to the read master when no other descriptors are buffered. This is commonly used for video frame buffering.

Park Writes – When set the dispatcher will continue to reissue the same descriptor to the write master when no other descriptors are buffered.

End on EOP – End on end of packet will allow the write master to continuously transfer data during ST→MM transfers without knowing how much data is arriving ahead of time. This is commonly used for packet based traffic such as Ethernet.

End on EOP or Length – End on end of packet or length performs the same operation as 'End on EOP' except for ST→MM transfers when enabled the write master will complete based on a limiting length value. For example if you set the length field to 1MB for a ST→MM transfer and 1MB of data has been transmitted without the EOP arriving at the slave port the write master will complete the transfer early which will be referred to later as early termination.

Transfer Complete IRQ Mask – Used to signal an interrupt to the host when a transfer completes. In the case of MM→ST transfers this interrupt will be based on the read master completing a transfer. In the case of ST→MM or MM→MM transfers this interrupt will be based on the write master completing a transfer.

Early Termination IRQ Mask – Used to signal an interrupt to the host when a ST→MM transfer completes early. For example if you set this bit and set the length field to 1MB for ST→MM transfers this interrupt will fire when more than 1MB of data arrives to the write master without the end of packet being seen.

Transmit Error/Error IRQ Mask – For MM→ST transfers this field is used to specify a transmit error. This is commonly used for transmitting error information downstream to streaming components such as an Ethernet MAC. For ST→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 and 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 will receive an interrupt.

Early Done – Used to hide the latency between read descriptors. When set the read master will not wait for pending reads to return before requesting another descriptor. Typically this bit will be set for all descriptors except the last one. This bit is most effective for hiding high read latency (SDRAM, PCIe, SRIO, etc…)

Go – Used to commit all the descriptor information into the descriptor FIFO. As the host writes the other fields to the descriptor FIFO byte enables are being used to write to each byte lane of the FIFO; however, the data written is not committed until the go bit has been written. As a result ensure the go bit is the last bit written for each descriptor.

Dispatcher Control and Status Registers (CSR)

		Byte Lanes			
Offset	**Access**	**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>[1]		Response Fill Level[15..0]	
0x10	Read	Write Sequence Number[15..0][2]		Read Sequence Number[15..0][2]	
0x14	N/A	<reserved>[1]			
0x18	N/A	<reserved>[1]			
0x1C	N/A	<reserved>[1]			

Table 9. Dispatcher Control and Status Register Map

[1] Writing to reserved bits will have no impact on the hardware, reading will return unknown data
[2] Sequence numbers will only be present when dispatcher enhanced features are enabled

CSR Status Register

Bit	Name	Description
0	Busy	Set when the dispatcher still has commands buffered or one of the masters is still transferring data
1	Descriptor Buffer Empty	Set when both the read **and** write command buffers are empty
2	Descriptor Buffer Full	Set when either the read **or** write command buffers are full
3	Response Buffer Empty	Set when the response buffer is empty
4	Response Buffer Full	Set when the response buffer is full
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 will be asserted after the master completes any read or write operations that were already commencing.
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 transfers in flight on the fabric. When this bit de-asserts you may start using the SGDMA again.
7	Stopped on Error	Set when the dispatcher is programmed to stop errors and an error beat enters the write master.
8	Stopped on Early Termination	Set when the dispatcher is programmed to stop on early termination and 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 will give you the maximum packet based transfer possible (early termination is always enabled for packet transfers).
9	IRQ	Set when an interrupt condition occurs.
31-10	<reserved>	N/A

Table 10. CSR Status Register Bitfields

CSR Control Register

Bit	Name	Description
0	Stop Dispatcher	Setting this bit will stop the SGDMA in the middle of a transaction. If a read or write operation is occurring then the access will be 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.
1	Reset Dispatcher	Setting this bit will reset 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.
2	Stop on Error	Setting this bit will stop the SGDMA from issuing more read/write commands to the master modules if an error enters the write master module sink port.
3	Stop on Early Termination	Setting this bit will stop the SGDMA from issuing out 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.
4	Global Interrupt Enable Mask	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.
5	Stop Descriptors	Setting this bit will stop the SGDMA dispatcher from issuing more descriptors to the read or write masters. Read the **stopped** status register to determine when the dispatcher has stopped issuing commands and the read and write masters are idle.
31-6	<reserved>	N/A

Table 11. CSR Control Register Bitfields

Response Registers

		Byte Lanes			
Offset	**Access**	**3**	**2**	**1**	**0**
0x0	Read	Actual Bytes Transferred[31:0]			
0x4	Read	<reserved>[2]	<reserved>	Early Termination[1]	Error[7:0]

Table 12. Response Register Map

[1] Early Termination is a single bit located at bit 8 of offset 0x4
[2] Reading from byte 7 pops the response FIFO

Programming with the Modular SGDMA Controller

To use the modular SGDMA you must encapsulate the descriptor fields into either a standard or extended descriptor data structure. Response information from streaming to memory-mapped packet transfers is also encapsulated into a data structure. An application programming interface (API) has been provided to map all the data structure fields to the underlining hardware registers so you are not responsible for accessing the hardware directly from your application.

Standard Descriptor Data Structure

```
typedef struct {
  alt_u32 *read_address;
  alt_u32 *write_address;
  alt_u32 transfer_length;
  alt_u32 control;
} sgdma_standard_descriptor_packed sgdma_standard_descriptor;
```

Extended Descriptor Data Structure

```
typedef struct {
  alt_u32 *read_address_low;
  alt_u32 *write_address_low;
  alt_u32 transfer_length;
  alt_u16 sequence_number;
  alt_u8  read_burst_count;
  alt_u8  write_burst_count;
  alt_u16 read_stride;
  alt_u16 write_stride;
  alt_u32 *read_address_high;
  alt_u32 *write_address_high;
  alt_u32 control;
} sgdma_extended_descriptor_packed sgdma_extended_descriptor;
```

Response Data Structure

```
typedef struct {
  alt_u32 actual_bytes_transferred;
  alt_u8 error;
  alt_u8 early_termination;
} sgdma_response_packed sgdma_response;
```

Dispatcher API

Name	Description
`construct_standard_st_to_mm_descriptor()`	Creates a standard format ST to MM descriptor.
`construct_standard_mm_to_st_descriptor()`	Creates a standard format MM to ST descriptor.
`construct_standard_mm_to_mm_descriptor()`	Creates a standard format MM to MM descriptor.
`construct_extended_st_to_mm_descriptor()`	Creates an extended format ST to MM descriptor.
`construct_extended_mm_to_st_descriptor()`	Creates an extended format MM to ST descriptor.
`construct_extended_mm_to_mm_descriptor()`	Creates an extended format MM to MM descriptor.
`construct_extended_st_to_mm_descriptor_64()`	Creates an extended format ST to MM descriptor with 64-bit addressing.
`construct_extended_mm_to_st_descriptor_64()`	Creates an extended format MM to ST descriptor with 64-bit addressing.
`construct_extended_mm_to_mm_descriptor_64()`	Creates an extended format MM to MM descriptor with 64-bit addressing.
`write_standard_descriptor()`	Writes a standard format descriptor to the descriptor slave port.
`write_extended_descriptor()`	Writes an extended format descriptor to the descriptor slave port. Supports both 32-bit and 64-bit addressing descriptors.
`read_mm_response()`	Reads a transfer response from the response slave port.
`read_csr_status()`	Reads the CSR slave port status register.
`read_csr_control()`	Reads the CSR slave port control register.
`read_csr_read_descriptor_buffer_fill_level()`	Reads the CSR slave port read descriptor buffer fill level register.
`read_csr_write_descriptor_buffer_fill_level()`	Reads the CSR slave port write descriptor buffer fill level register.
`read_csr_response_buffer_fill_level()`	Reads the CSR slave port response buffer fill level register.
`read_csr_read_sequence_number()`	Reads the CSR slave port read sequence number register.
`read_csr_write_sequence_number()`	Reads the CSR slave port write sequence register.
`read_busy()`	Reads the dispatcher busy status.
`read_descriptor_buffer_empty()`	Reads the descriptor buffer empty status.
`read_descriptor_buffer_full()`	Reads the descriptor buffer full status.
`read_response_buffer_empty()`	Reads the response buffer empty status.
`read_response_buffer_full()`	Reads the response buffer full status.
`read_stopped()`	Reads the dispatcher stopped status. You should use this function to determine if the master or dispatcher modules have stopped.
`read_resetting()`	Reads the dispatcher reset pending status. You should use this function to determine if the dispatcher and master modules have completed a reset cycle.
`read_stopped_on_error()`	Reads the dispatcher stopped on error status.

`read_stopped_on_early_termination()`	Reads the dispatcher stopped on early termination status.
`read_irq()`	Reads the dispatcher interrupt bit.
`clear_irq()`	Clears the dispatcher interrupt bit.
`stop_dispatcher()`	Stops the dispatcher from issuing anymore commands to the masters and stops the read and write masters in the middle of any transfers.
`start_dispatcher()`	Starts the dispatcher block allowing it to issue transfer commands.
`reset_dispatcher()`	Resets the dispatcher and master modules.
`enable_stop_on_error()`	Enables the dispatcher stop on error logic. Use this to stop the dispatcher from issuing more commands if any errors occur during a ST to MM transfer.
`disable_stop_on_error()`	Disables the dispatcher stop on error logic.
`enable_stop_on_early_termination()`	Enables the dispatcher stop on early termination logic. Use this to stop the dispatcher from issuing more commands if any errors occur during a ST to MM transfers.
`disable_stop_on_early_termination()`	Disables the dispatcher stop on early termination logic.
`enable_global_interrupt_mask()`	Enables the dispatcher global interrupt mask.
`disable_global_interrupt_mask()`	Disables the dispatcher global interrupt mask.
`stop_descriptors()`	Stops the dispatcher from issuing any more read or write master commands. The read and write masters are allowed to finish any transfers already started.
`start_descriptors()`	Starts the dispatcher back up issuing any descriptor commands that are already buffered.

Figure 12. Dispatcher Driver Function List

construct_standard_st_to_mm_descriptor()

Prototype: int construct_standard_st_to_mm_descriptor(
sgdma_standard_descriptor *descriptor,
alt_u32 *write_address,
alt_u32 length,
alt_u32 control)

Thread-safe: Yes

Available from ISR: Yes

Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**

Parameters: *descriptor – a pointer to a standard descriptor structure.
*write_address – a pointer to the base address of the destination memory.
length – transfer length.
control – control field.

Returns: Always returns 0.

Description: This function creates a standard ST to MM descriptor structure. Use the write_standard_descriptor() function to write the descriptor to the dispatcher module. The 'go' bit will be automatically set so you do not need to add it to the control field.

construct_standard_mm_to_st_descriptor()

Prototype: int construct_standard_mm_to_st_descriptor(
sgdma_standard_descriptor *descriptor,
alt_u32 *read_address,
alt_u32 length,
alt_u32 control)

Thread-safe: Yes

Available from ISR: Yes

Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**

Parameters: *descriptor – a pointer to a standard descriptor structure.
*read_address – a pointer to the base address of the source memory.
length – transfer length.
control – control field.

Returns: Always returns 0.

Description: This function creates a standard MM to ST descriptor structure. Use the write_standard_descriptor() function to write the descriptor to the dispatcher module. The 'go' bit will be automatically set so you do not need to add it to the control field.

construct_standard_mm_to_mm_descriptor()

Prototype: int construct_standard_mm_to_mm_descriptor(
sgdma_standard_descriptor *descriptor,
alt_u32 *read_address,
alt_u32 *write_address,
alt_u32 length,
alt_u32 control)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**
Parameters: *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 – transfer length.
control – control field.
Returns: Always returns 0.
Description: This function creates a standard MM to MM descriptor structure. Use the write_standard_descriptor() function to write the descriptor to the dispatcher module. The 'go' bit will be automatically set so you do not need to add it to the control field.

construct_extended_st_to_mm_descriptor()

Prototype: int construct_extended_st_to_mm_descriptor(
sgdma_standard_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)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**
Parameters: *descriptor – a pointer to an extended descriptor structure.
*write_address – a pointer to the base address of the destination memory.
length – transfer length.
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…

Returns: Always returns 0.

Description: This function creates an extended ST to MM descriptor structure. Use the write_extended_descriptor() function to write the descriptor to the dispatcher module. The 'go' bit will be automatically set so you do not need to add it to the control field.

construct_extended_st_to_mm_descriptor_64()

Prototype:
```
int construct_extended_st_to_mm_descriptor(
        sgdma_standard_descriptor *descriptor,
        alt_u32 *write_address_low,
        alt_u32 *write_address_high,
        alt_u32 length,
        alt_u32 control,
        alt_u16 sequence_number,
        alt_u8 write_burst_count,
        alt_u16 write_stride)
```

Thread-safe: Yes

Available from ISR: Yes

Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**

Parameters:

*descriptor – a pointer to an extended descriptor structure.

*write_address_low – a pointer to the lower 32-bit base address of the destination memory.

*write_address_high – a pointer to the upper 32-bit base address of the destination memory.

length – transfer length.

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…

Returns: Always returns 0.

Description: This function creates an extended ST to MM descriptor structure. Use the write_extended_descriptor() function to write the descriptor to the

dispatcher module. The 'go' bit will be automatically set so you do not need to add it to the control field.

construct_extended_mm_to_st_descriptor()

Prototype: int construct_extended_mm_to_st_descriptor(
sgdma_standard_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)

Thread-safe: Yes

Available from ISR: Yes

Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**

Parameters: *descriptor – a pointer to an extended descriptor structure.
*read_address – a pointer to the base address of the source memory.
length – transfer length.
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: Always returns 0.

Description: This function creates an extended ST to MM descriptor structure. Use the write_extended_descriptor() function to write the descriptor to the dispatcher module. The 'go' bit will be automatically set so you do not need to add it to the control field.

construct_extended_mm_to_st_descriptor_64()

Prototype: int construct_extended_mm_to_st_descriptor(
sgdma_standard_descriptor *descriptor,
alt_u32 *read_address_low,
alt_u32 *read_address_high,
alt_u32 length,
alt_u32 control,
alt_u16 sequence_number,
alt_u8 read_burst_count,
alt_u16 read_stride)

Thread-safe: Yes

Available from ISR: Yes

Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**

Parameters: *descriptor – a pointer to an extended descriptor structure.
*read_address_low – a pointer to the lower 32-bit base address of the source memory.
*read_address_high – a pointer to the upper 32-bit base address of the source memory.
length – transfer length.
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: Always returns 0.

Description: This function creates an extended ST to MM descriptor structure. Use the write_extended_descriptor() function to write the descriptor to the dispatcher module. The ‘go’ bit will be automatically set so you do not need to add it to the control field.

construct_extended_mm_to_mm_descriptor()

Prototype: int construct_extended_mm_to_st_descriptor(
sgdma_standard_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)

Thread-safe: Yes

Available from ISR: Yes

Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**

Parameters:
*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 – transfer length.
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: Always returns 0.

Description: This function creates an extended MM to MM descriptor structure. Use the write_extended_descriptor() function to write the descriptor to the dispatcher module. The ‘go’ bit will be automatically set so you do not need to add it to the control field.

construct_extended_mm_to_mm_descriptor_64()

Prototype: int construct_extended_mm_to_st_descriptor(
sgdma_standard_descriptor *descriptor,
alt_u32 *read_address_low,
alt_u32 *read_address_high,
alt_u32 *write_address_low,
alt_u32 *write_address_high,
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)

Thread-safe: Yes

Available from ISR: Yes

Include: **<sgdma_dispatcher.h>, <descriptor_regs.h>**

Parameters: *descriptor – a pointer to an extended descriptor structure.
*read_address_low – a pointer to the lower 32-bit base address of the source memory.
*read_address_high – a pointer to the upper 32-bit base address of the source memory.
*write_address_low – a pointer to the lower 32-bit base address of the destination memory.
*write_address_high – a pointer to the upper 32-bit base address of the destination memory.
length – transfer length.
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: Always returns 0.

Description: This function creates an extended MM to MM descriptor structure. Use the write_extended_descriptor() function to write the descriptor to the dispatcher module. The 'go' bit will be automatically set so you do not need to add it to the control field.

write_standard_descriptor()

Prototype: int write_standard_descriptor(
alt_u32 csr_base,
alt_u32 descriptor_base,
sgdma_standard_descriptor *descriptor)
Thread-safe: No
Available from ISR: No
Include: **<sgdma_dispatcher.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 **errno.h**.
Description: Sends a fully formed standard 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.

write_extended_descriptor()

Prototype: int write_extended_descriptor(
alt_u32 csr_base,
alt_u32 descriptor_base,
sgdma_extended_descriptor *descriptor)
Thread-safe: No
Available from ISR: No
Include: **<sgdma_dispatcher.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 **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.

read_mm_response()

Prototype:	int read_mm_response(alt_u32 csr_base, alt_u32 response_base, sgdma_response *response)
Thread-safe:	No
Available from ISR:	No
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port. descriptor_base – base address of the dispatcher descriptor slave port. *response – a pointer to a response structure.
Returns:	Returns 0 upon success. Other return codes are defined in **errno.h**.
Description:	Populates a response structure with response data from the write master module. Response data is only applicable to ST to MM transfers where packet support has been enabled in the write master module. This function is not reentrant since it must complete reading the entire response structure from the dispatcher module and cannot be pre-empted.

read_csr_status()

Prototype:	alt_u32 read_csr_status(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Returns the contents of the dispatcher status register
Description:	This function returns the dispatcher status register. Use the macros defined in **csr_regs.h** to mask and shift the individual status bits.

read_csr_control()

Prototype:	alt_u32 read_csr_control(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Returns the contents of the dispatcher control register.
Description:	This function returns the dispatcher control register. Use the macros defined in **csr_regs.h** to mask and shift the individual control bits.

read_csr_read_descriptor_buffer_fill_level()

Prototype: alt_u16 read_csr_read_descriptor_buffer_fill_level(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the read descriptor buffer fill level.
Description: This function returns the dispatcher read descriptor buffer fill level.

read_csr_write_descriptor_buffer_fill_level()

Prototype: alt_u16 read_csr_write_descriptor_buffer_fill_level(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the write descriptor buffer fill level.
Description: This function returns the dispatcher write descriptor buffer fill level.

read_csr_response_buffer_fill_level()

Prototype: alt_u16 read_csr_response_buffer_fill_level(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the response buffer fill level.
Description: This function returns the dispatcher response buffer fill level.

read_csr_read_sequence_number()

Prototype: alt_u16 read_csr_read_sequence_number(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the read sequence number.
Description: This function returns the dispatcher read sequence number.

read_csr_write_sequence_number()

Prototype: alt_u16 read_csr_write_sequence_number(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the write sequence number.
Description: This function returns the dispatcher write sequence number.

read_busy()

Prototype: alt_u32 read_busy(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the busy bit.
Description: This function returns the busy bit of the CSR status register. When the SGDMA is busy this bit will be set to a logical one.

read_descriptor_buffer_empty()

Prototype: alt_u32 read_descriptor_buffer_empty(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the descriptor buffer empty bit.
Description: This function returns the descriptor buffer empty bit of the CSR status register. When the read and write descriptor buffers are empty this bit will be set to a logical one.

read_descriptor_buffer_full()

Prototype: alt_u32 read_descriptor_buffer_full(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the descriptor buffer full bit.
Description: This function returns the descriptor buffer full bit of the CSR status register. When the read or write descriptor buffers are full this bit will be set to a logical one.

read_response_buffer_empty()

Prototype: alt_u32 read_response_buffer_empty(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the response buffer empty bit.
Description: This function returns the response buffer empty bit of the CSR status register. When the response buffer is empty this bit will be set to a logical one.

read_response_buffer_full()

Prototype: alt_u32 read_response_buffer_full(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the response buffer full bit.
Description: This function returns the response buffer full bit of the CSR status register. When the response buffer is full this bit will be set to a logical one.

read_stopped()

Prototype: alt_u32 read_stopped(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the stopped bit.
Description: This function returns the stopped bit of the CSR status register. When the dispatcher and master modules have stopped this bit will be set to a logical one.

read_resetting()

Prototype: alt_u32 read_resetting(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the resetting bit.
Description: This function returns the stopped bit of the CSR status register. When the dispatcher and master modules are still resetting after a call to **reset_dispatcher()** this bit will be set to a logical one.

read_stopped_on_error()

Prototype: alt_u32 read_stopped_on_error(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Returns the stopped on error bit.
Description: This function returns the stopped on error bit of the CSR status register. This bit will only be set if you have configured the CSR control bit for stop on error and an error enters the write master module during a transfer.

read_stopped_on_early_termination()

Prototype:	alt_u32 read_stopped_on_early_termination(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Returns the stopped on early termination bit.
Description:	This function returns the stopped on early termination bit of the CSR status register. This bit will only be set if you have configured the CSR control bit for stop on early termination and the write master module performs a packet transfer that exceeds the length you have set in the descriptor.

read_irq()

Prototype:	alt_u32 read_irq(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Returns the irq bit.
Description:	This function returns the irq bit of the CSR status register. This bit will only be set if the global interrupt register has been enabled and an interrupt condition occurs during a transfer.

clear_irq()

Prototype:	void clear_irq(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Does not return anything.
Description:	This function clears the irq bit of the CSR status register by performing a write access. The write access only clears the irq bit and leaves the rest of the status register unmodified.

stop_dispatcher()

Prototype: void stop_dispatcher(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Does not return anything.
Description: This function writes to the stop bit of the CSR control register. Since one of the master modules could be in the middle of a read or write access you must call **read_stopped()** to determine if the SGDMA has come to a complete stop.

reset_dispatcher()

Prototype: void reset_dispatcher(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Does not return anything.
Description: This function writes to the reset bit of the CSR control register. This reset will cause the dispatcher and master modules to enter the reset state so all registers and FIFOs will be cleared. A reset may be prolonged if a read or write access is occurring so you must call **read_resetting()** to determine if the SGDMA has completed a reset cycle before configuring it again.

enable_stop_on_error()

Prototype: void enable_stop_on_error(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Does not return anything.
Description: This function writes to the stop on error bit of the CSR control register. This bit is used to prevent any more commands to be sent to the write master module if an error occurred during a packet transfer.

disable_stop_on_error()

Prototype: void disable_stop_on_error(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Does not return anything.
Description: This function writes to the stop on error bit of the CSR control register. This bit is used to allow more commands to be sent to the write master module even if an error occurred during a packet transfer.

enable_stop_on_early_termination()

Prototype: void enable_stop_on_early_termination(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Does not return anything.
Description: This function writes to the stop on early termination bit of the CSR control register. This bit is used to prevent any more commands to be sent to the write master module if a packet transfer lasts longer than the number of bytes programmed into the descriptor length field.

disable_stop_on_early_termination()

Prototype: void disable_stop_on_early_termination(alt_u32 csr_base)
Thread-safe: Yes
Available from ISR: Yes
Include: **<sgdma_dispatcher.h>**
Parameters: csr_base – base address of the dispatcher CSR slave port.
Returns: Does not return anything.
Description: This function writes to the stop on early termination bit of the CSR control register. This bit is used to allow more commands to be sent to the write master module even if a packet transfer lasts longer than the number of bytes programmed into the descriptor length field.

enable_global_interrupt_mask()

Prototype:	void enable_global_interrupt_mask(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Does not return anything.
Description:	This function writes to the enable global interrupt mask bit of the CSR control register. This bit is used to enable the global interrupt mask.

disable_global_interrupt_mask()

Prototype:	void disable_global_interrupt_mask(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Does not return anything.
Description:	This function writes to the enable global interrupt mask bit of the CSR control register. This bit is used to disable the global interrupt mask.

stop_descriptors()

Prototype:	void stop_descriptors(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Does not return anything.
Description:	This function disables the descriptor buffer logic from issuing read and write commands to the master blocks. Any descriptor that has already been issued is allowed to complete.

start_descriptors()

Prototype:	void start_descriptors(alt_u32 csr_base)
Thread-safe:	Yes
Available from ISR:	Yes
Include:	**<sgdma_dispatcher.h>**
Parameters:	csr_base – base address of the dispatcher CSR slave port.
Returns:	Does not return anything.
Description:	This function re-enabled the descriptor buffer logic to continue issuing read and write commands to the master blocks.