AXI DataMover v5.1

LogiCORE IP Product Guide

Vivado Design Suite

PG022 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

IP Facts

Chapter 1: Overview

Feature Summary . . . 7
Applications . . . 8
Unsupported Features . . . 8
Licensing and Ordering Information . . . 9

Chapter 2: Product Specification

Performance . . . 10
Resource Utilization . . . 12
Port Descriptions . . . 14
Design Information . . . 18

Chapter 3: Designing with the Core

Clocking . . . 36
Resets . . . 36

Chapter 4: Design Flow Steps

Customizing and Generating the Core . . . 37
Constraining the Core . . . 46
Simulation . . . 47
Synthesis and Implementation . . . 47

Chapter 5: Example Design

Implementing the Example Design . . . 49
Simulating the Example Design . . . 51
Test Bench for the Example Design . . . 51

Appendix A: Migrating and Upgrading

Migrating to the Vivado Design Suite . . . 53
Upgrading in the Vivado Design Suite . . . 53

Appendix B: Debugging

Finding Help on Xilinx.com 54
Vivado Design Suite Debug Feature 55
Hardware Debug 56

Appendix C: Additional Resources and Legal Notices

Xilinx Resources 57
References 57
Revision History 58
Please Read: Important Legal Notices 59

Introduction

The Xilinx LogiCORE™ IP AXI DataMover core is a soft core that provides the basic AXI4 Read to AXI4-Stream and AXI4-Stream to AXI4 Write data transport and protocol conversion. The function is intended to be a standalone core for custom designs.

Features

- AXI4 Compliant
- Primary AXI4 data width support of 32, 64, 128, 256, 512, and 1,024 bits
- Primary AXI4-Stream data width support of 8, 16, 32, 64, 128, 256, 512 and 1,024 bits
- Parameterized Memory Map Burst Lengths of 2, 4, 8, 16, 32, 64, 128, and 256 data beats
- Optional Unaligned Address access; Up to 64 bit address support.
- Optional General Purpose Store-And-Forward in both Memory Map to Stream (MM2S) and Stream to Memory Map (S2MM)
- Optional Indeterminate Bytes to Transfer (BTT) mode in S2MM
- Supports synchronous/asynchronous clocking for Command/Status interface

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families, UltraScale™ Architecture, Zynq®-7000 All Programmable SoC, 7 Series FPGAs
Supported User Interfaces	AXI4, AXI4-Stream
Resources	See Table 2-4 and Table 2-5.
Provided with Core	
Design Files	VHDL
Example Design	VHDL
Test Bench	VHDL
Constraints File	Delivered during IP generation
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide
Synthesis	
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The AXI DataMover is a key interconnect infrastructure IP that enables high throughput transfer of data between the AXI4 memory-mapped and AXI4-Stream domains. The AXI DataMover provides the MM2S and S2MM AXI4-Stream channels that operate independently in a full-duplex like method. The AXI DataMover IP core is a key building block with 4 KB address boundary protection, automatic burst partitioning, and provides the ability to queue multiple transfer requests using nearly the full bandwidth capabilities of the AXI4-Stream protocol. Furthermore, the AXI DataMover provides byte-level data realignment allowing memory reads and writes to any byte offset location. Based on the requirement of the channels, they can be configured as Basic or Full.

Figure 1-1 and Figure 1-2 show block diagrams of the AXI DataMover core. There are two sub blocks:

- **MM2S**: This block handles transactions from the AXI4 to the AXI4-Stream domain. It has its dedicated AXI4-Stream compliant command and status queues, reset block, and error signals. Based on command inputs, the MM2S block issues a read request on the AXI4 interface. Read data can be optionally stored inside the MM2S block. Datapath interfaces (AXI4-Read and AXI4-Stream Master) can optionally be made asynchronous to command and status interfaces (AXI4-Stream Command and AXI4-Stream Status).
- **S2MM**: This block handles transactions from the AXI4-Stream to AXI4 domain. It has its dedicated AXI4-Stream compliant command and status queues, reset block, and error signals. Based on command inputs and input data from the AXI4-Stream interface, the S2MM block issues a write request on the AXI4 interface. Input stream data can be optionally stored inside a S2MM block. Datapath interfaces (AXI4-Read and AXI4-Stream Master) can optionally be made asynchronous to command and status interfaces (AXI4-Stream Command and AXI4-Stream Status).

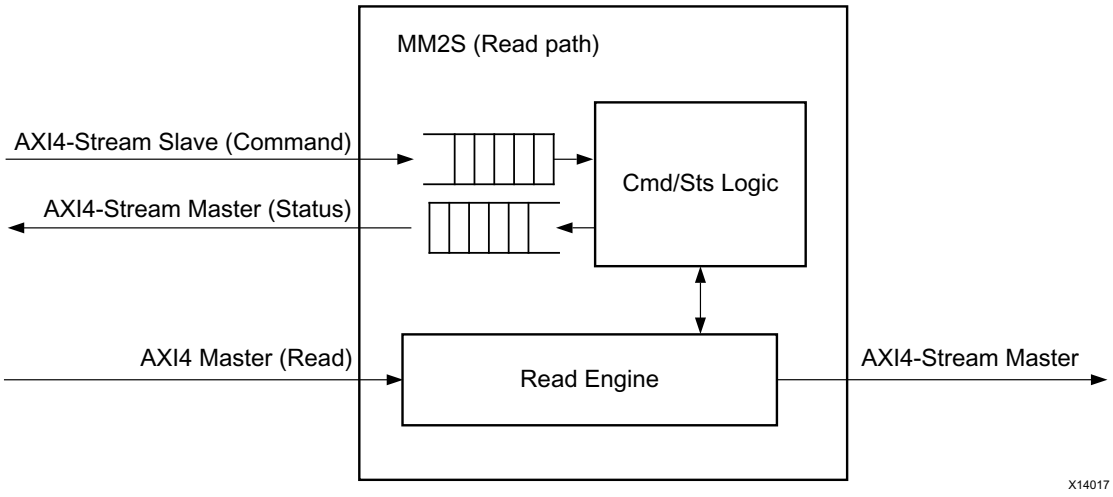

Figure 1-1: **AXI DataMover Read Path**

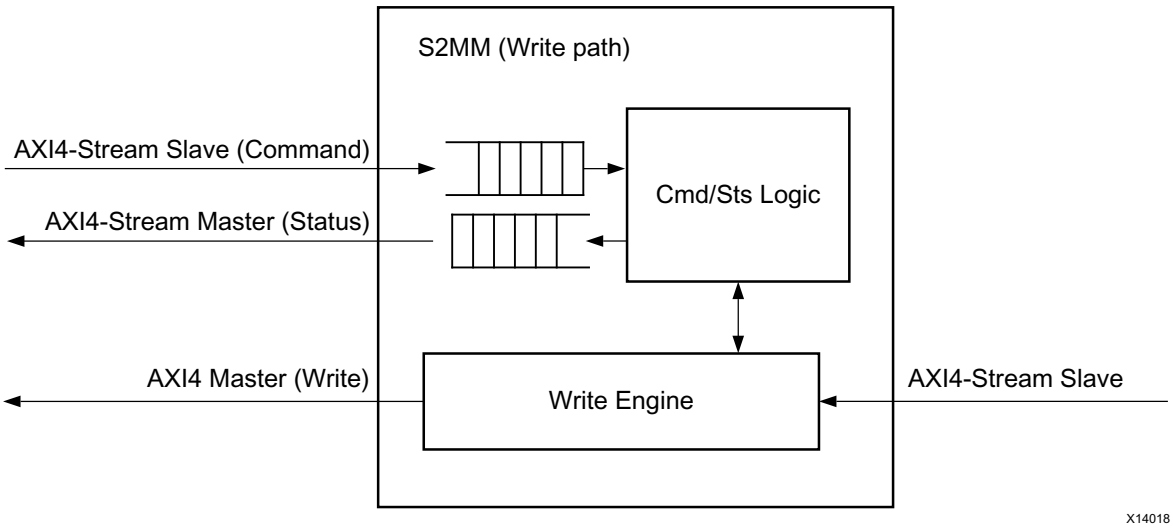

Figure 1-2: **AXI DataMover Write Path**

Feature Summary

AXI4 Compliant

The AXI DataMover core is fully compliant with the AXI4 interface and the AXI4-Stream interface.

AXI4 Data Width

The AXI DataMover core supports the primary AXI4 data bus width of 32, 64, 128, 256, 512, and 1,024 bits.

AXI4-Stream Data Width

The AXI DataMover core supports the primary AXI4-Stream data bus width of 8, 16, 32, 64, 128, 256, 512, and 1,024 bits. The AXI4-Stream data width must be less than or equal to the AXI4 data width for the respective channel.

Maximum Memory Map Burst Length

The AXI DataMover core supports parameterized maximum size of the burst cycles on the AXI MM2S Memory Map interface. In other words, this setting specifies the granularity of burst partitioning. For example, if the burst length is set to 16, the maximum burst on the memory map interface is 16 data beats. Smaller values reduce throughput but result in less impact on the AXI infrastructure. Larger values increase throughput but result in a greater impact on the AXI infrastructure. Valid supported values are 2, 4, 8, 16, 32, 64, 128, and 256 based on the data width that is selected.

Unaligned Transfers

The AXI DataMover core optionally supports the Data Realignment Engine (DRE). When DRE is enabled, data is realigned to the byte (8 bits) level on the Memory Map datapath. DRE support is provided on the AXI4-Stream interface for TDATA widths up to 64 bits.

If the DRE is enabled, data reads can start from any Buffer Address byte offset, and the read data is aligned such that the first byte read is the first valid byte out on the AXI4-Stream. Similarly, when the DRE is enabled, the writes can happen at any byte offset address. What is considered aligned or unaligned is based on the Memory Map data width. For example, if Memory Map Data Width = 32, data is aligned if it is located at address offsets of 0x0, 0x4, 0x8, 0xC, etc. Data is unaligned if it is located at address offsets of 0x1, 0x2, 0x3 and so forth.

Note: Performing unaligned transfers when DRE is disabled will give unpredictable results.

Asynchronous Clocks

The AXI DataMover core supports asynchronous clock domain for Command/Status Stream interface and Memory Map interface.

Store and Forward

The AXI DataMover core supports the optional General Purpose Store-And-Forward feature. Store and Forward buffer is sized based on Memory map data width and Burst size support. It is sized to allow up to six outstanding requests on MM2S channel.

Indeterminate BTT Mode

The AXI DataMover core supports the optional Indeterminate BTT mode for the S2MM channel. This is needed when the number of bytes to be received on the input S2MM Stream Channel is unknown.

Applications

The AXI DataMover provides high-speed data movement between system memory and an AXI4-Stream-based target. This core is intended to be a standalone core for a custom design.

Unsupported Features

The following AXI4 features are not supported by the DataMover design.

- User signals
- Locking transfers
- Caching transfers
- Circular Burst transfers

Licensing and Ordering Information

This Xilinx LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Performance

The AXI DataMover is characterized using methods described in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1]. Table 2-1 shows the results of the characterization runs.

Table 2-1: **Maximum Frequencies**

Family	Speed Grade	FMAX (MHz) All Clocks	
		AXI4	AXI4-Stream
Virtex-7	–1	200	200
Kintex-7		200	200
Artix-7		150	150
Virtex-7	–2	240	240
Kintex-7		240	240
Artix-7		180	180
Virtex-7	–3	280	280
Kintex-7		280	280
Artix-7		200	200

Latency

Table 2-2 describes the latency for the AXI DataMover core. Latency is measured in simulation and indicates AXI DataMover core latency cycles exclusive of system dependent latency or throttling.

Table 2-2: **AXI DataMover Latency**

Description	Clocks
MM2S Channel	
Initial m_axi_mm2s_rvalid to m_axis_mm2s_tvalid (Store and Forward disabled)	1
Initial m_axi_mm2s_rvalid to m_axis_mm2s_tvalid (Store and Forward enabled)	3
AXI4-Stream packet to packet latency (DRE Disabled) m_axis_mm2s_tlast to m_axis_mm2s_tvalid	2
AXI4-Stream packet to packet latency (DRE Enabled) m_axis_mm2s_tlast to m_axis_mm2s_tvalid	3
s_axis_mm2s_cmd_tvalid to m_axi_mm2s_arvalid	8
S2MM Channel	
Initial s_axis_s2mm_tvalid to m_axi_s2mm_awalid (Store and Forward disabled)	2
Initial s_axis_s2mm_tvalid to m_axi_s2mm_awalid (Store and Forward enabled)	20
AXI4-Stream packet to packet latency (DRE Disabled) s_axis_s2mm_tlast to s_axis_s2mm_tready	2
AXI4-Stream packet to packet latency (DRE Enabled) s_axis_s2mm_tlast to s_axis_s2mm_tready	3

Throughput

Table 2-3 describes the latency for the AXI DataMover core. The table provides performance information for a typical configuration. Throughput testing consisted of eight parent commands loaded into the AXI DataMover core with each command having a BTT value as 1 MB and each channel operating simultaneously (full duplex). The core was configured for synchronous operation meaning `m_axis_mm2s_cmdsts_aclk` = `m_axis_s2mm_cmdsts_awclk` = `m_axi_mm2s_aclk` = `m_axi_s2mm_aclk`.

The core configuration used to generate the throughput data is as follows:

- Memory Map Data Width = 32
- Streaming Data Width = 32
- No unaligned access
- Store and Forward enabled

Table 2-3: **AXI DataMover Throughput**

AXI DataMover Channel	Primary Clock Frequency	Packet Size	Maximum Total Data Throughput (MB/sec)	Percent of Theoretical
MM2S	100	1 MB	391.27	97.75%
S2MM	100	1 MB	391.27	97.75%

Resource Utilization

The AXI DataMover resource utilization for various parameter combinations measured with 7 series, Zynq®-7000 (Table 2-4) and UltraScale™ devices (Table 2-5).

Table 2-4: **7 Series and Zynq-7000 Device Resource Estimates**

MM2S									S2MM									Resources			
Addr Width	Type	MMap Width	Streaming Width	Burst Length	BTT Width	Async Mode	Store and Forward	Unaligned Transfer	Type	MMap Width	Streaming Width	Burst Length	BTT Width	Async Mode	Store and Forward	Unaligned Transfer	Indeterminate BTT	Slice	Luts	Reg	Block RAM
32	Basic	32	32	16	16	NA	NA	NA	Basic	32	32	16	16	NA	NA	NA	NA	207	543	645	0
32	Full	32	32	16	16	FALSE	TRUE	FALSE	Full	32	32	16	16	FALSE	TRUE	FALSE	FALSE	430	1158	1128	2
32	Full	64	64	8	23	FALSE	TRUE	FALSE	Full	64	64	8	23	FALSE	TRUE	FALSE	FALSE	545	1435	1497	3
32	Full	64	32	8	16	FALSE	TRUE	TRUE	Full	64	32	8	16	FALSE	TRUE	TRUE	FALSE	574	1515	1563	3
32	Full	32	32	16	16	TRUE	TRUE	FALSE	Full	32	32	16	16	TRUE	FALSE	FALSE	TRUE	613	1277	1737	6
64	Full	32	32	16	16	FALSE	TRUE	FALSE	Full	32	32	16	16	FALSE	TRUE	FALSE	FALSE	504	1303	1336	2
64	Full	32	32	16	16	TRUE	TRUE	FALSE	Full	32	32	16	16	TRUE	FALSE	FALSE	TRUE	589	1322	1680	8

Table 2-5: **UltraScale Device Resource Estimates**

MM2S									S2MM									Resources			
Addr Width	Type	MMap Width	Streaming Width	Burst Length	BTT Width	Async Mode	Store and Forward	Unaligned Transfer	Type	MMap Width	Streaming Width	Burst Length	BTT Width	Async Mode	Store and Forward	Unaligned Transfer	Indeterminate BTT	Slice/CLB	Luts	Reg	Block RAM
32	Basic	32	32	16	16	NA	NA	NA	Basic	32	32	16	16	NA	NA	NA	NA	134	558	645	
32	Full	32	32	16	16	FALSE	TRUE	FALSE	Full	32	32	16	16	FALSE	TRUE	FALSE	FALSE	265	1193	1130	2
32	Full	64	64	8	23	FALSE	TRUE	FALSE	Full	64	64	8	23	FALSE	TRUE	FALSE	FALSE	347	1481	1499	3
32	Full	64	32	8	16	FALSE	TRUE	TRUE	Full	64	32	8	16	FALSE	TRUE	TRUE	FALSE	352	1569	1564	3
32	Full	32	32	16	16	TRUE	TRUE	FALSE	Full	32	32	16	16	TRUE	FALSE	FALSE	TRUE	313	1300	1740	6
64	Full	32	32	16	16	FALSE	TRUE	FALSE	Full	32	32	16	16	FALSE	TRUE	FALSE	FALSE	316	1357	1338	2
64	Full	32	32	16	16	TRUE	TRUE	FALSE	Full	32	32	16	16	TRUE	FALSE	FALSE	TRUE	353	1451	2000	8

Port Descriptions

The AXI DataMover I/O signals are described in Table 2-6.

Table 2-6: **I/O Signal Description**

Signal Name	Interface	Signal Type	Init Status	Description
Memory Map to Stream Clock and Reset				
m_axi_mm2s_aclk	MM2S	Input	–	Master Clock for MM2S path
m_axi_mm2s_aresetn	MM2S	Input	–	Master Reset for the MM2S logic. Active-Low assertion sensitivity. Must be asserted for three clock periods of the slower of `m_axi_mm2s_aclk` and `m_axis_mm2s_cmdsts_aclk`.
Memory Map to Stream Soft Shutdown Control				
mm2s_halt	MM2S	Input	–	Active-High input signal requesting that the MM2S function perform a soft shutdown and stop. See MM2S Soft Shutdown.
mm2s_halt_cmplt	MM2S	Output	0	Active-High output signal indicating that the MM2S function has completed a soft shutdown and is stopped. See MM2S Soft Shutdown.
Memory Map to Stream Error Detect				
mm2s_err	MM2S	Output	0	MM2S Error Output. This active-High output discrete signal is asserted whenever an Error condition is encountered within the MM2S such as an invalid BTT value of 0. This bit is a “sticky” error indication; after being set it requires an assertion of the `m_axi_mm2s_aresetn` signal to clear it.
Memory Map to Stream Debug Support				
mm2s_dbg_sel(3:0)	MM2S	Input	–	Reserved for internal Xilinx use.
mm2s_dbg_data(31:0)	MM2S	Output	BEEF0000 (if Omit MM2s) BEEF1111 (if Full MM2s) BEEF2222 (if Basic MM2s)	Reserved for internal Xilinx use.

Table 2-6: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
Memory Map to Stream Address Posting Control and Status				
mm2s_allow_addr_req	MM2S	Input	–	Used to control the MM2S in posting an address on the AXI4 Read address channel. A 1 allows posting and a 0 inhibits posting. See Address Posting Control and Status Interface.
mm2s_addr_req_posted	MM2S	Output	0	This output signal is asserted to 1 for one `m_axi_mm2s_aclk` period for each new address posted to the AXI4 Read Address Channel. See Address Posting Control and Status Interface.
mm2s_rd_xfer_cmplt	MM2S	Output	0	This output signal is asserted to 1 for one `m_axi_s2mm_aclk` period for each completed AXI4 read transfer (qualified RLAST data beat) clearing the internal read data controller block.
AXI4 Read Interface Signals				
m_axi*	M_AXI_MM2S	–	–	See the *Vivado AXI Reference Guide* (UG1037)[Ref 2] for the description of AXI4 Signals.
AXI4-Stream Master Interface Signals				
m_axis*	M_AXIS*	–	–	See the *Vivado AXI Reference Guide* (UG1037) [Ref 2] for the description of AXI4 Signals.
Memory Map to Stream Command/Status Channel Asynchronous Clock and Reset				
m_axis_mm2s_cmdsts_aclk	MM2S Command & Status	Input	–	MM2S Command Interface Clock. This clock is only used if asynchronous clocks are enabled in the Vivado® Integrated Design Environment (IDE). The frequency of this clock is expected to be equal or less than the `m_axi_mm2s_aclk`.
m_axis_mm2s_ cmdsts_aresetn	MM2S Command & Status	Input	–	MM2S Command and Status Interface Reset (Active-Low). This reset input is only used if asynchronous clocks are enabled in the Vivado Integrated Design Environment (IDE). Must be asserted for three clock periods of the slower of `m_axis_mm2s_cmdsts_aclk` and `m_axi_mm2s_aclk`.

Table 2-6: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
AXI4 Slave Stream Interface Signals				
s_axis*	S_AXIS*	Input/ Output	Input/ Output	See the *Vivado AXI Reference Guide* (UG1037) [Ref 2] for the description of AXI4 Signals.
Stream to Memory Map Clock and Reset				
m_axi_s2mm_aclk	S2MM	Input	–	Master Clock for S2MM path
m_axi_s2mm_aresetn	S2MM	Input	–	Master Reset for the S2MM logic (Active-Low sensitivity). Must be asserted for three clock periods of the slower of `m_axi_s2mm_aclk` and `m_axis_s2mm_cmdsts_awclk`.
Stream to Memory Map Soft Shutdown Control				
s2mm_halt	S2MM	Input	–	Active-High input signal requesting that the S2MM function perform a soft shutdown and stop. See S2MM Soft Shutdown.
s2mm_halt_cmplt	S2MM	Output	0	Active-High output signal indicating that the S2MM function has completed a soft shutdown and is stopped. See S2MM Soft Shutdown.
Stream to Memory Map Error Detect				
s2mm_err	S2MM	Output	0	S2MM Error Output. This active-High output discrete signal is asserted whenever an Error condition is encountered within the S2MM such as an invalid BTT of 0 or a Stream overrun or underrun when the S2MM Indeterminate BTT is not enabled. This bit is a "sticky" error indication; after being set it requires an assertion of the `m_axi_s2mm_aresetn` signal to clear it.
Stream to Memory Map Debug Support				
s2mm_dbg_sel(3:0)	S2MM	Input	–	Reserved for internal Xilinx use.
s2mm_dbg_data(31:0)	S2MM	Output	CAFE0000 (if Omit S2MM) CAFE1111 (if Full S2MM) CAFE2222 (if Basic S2MM)	Reserved for internal Xilinx use.

Table 2-6: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
Stream to Memory Map Address Posting Control and Status				
s2mm_allow_addr_req	S2MM	Input	–	Used to control the S2MM in posting an address on the AXI4 Write Address Channel. A 1 allows posting and a 0 inhibits posting. See Address Posting Control and Status Interface.
s2mm_addr_req_posted	S2MM	Output	0	This output signal is asserted to 1 for one `m_axi_s2mm_aclk` period for each new address posted to the AXI4 Write Address Channel.
s2mm_wr_xfer_cmplt	S2MM	Output	0	This output signal is asserted to 1 for one `m_axi_s2mm_aclk` period for each completed AXI4 write transfer (qualified WLAST data beat) clearing the internal write data controller block.
s2mm_ld_nxt_len	S2MM	Output	0	This output signal is asserted to 1 for one `m_axi_s2mm_aclk` period for each AXI4 Write Transfer request to be posted to the AXI4 Write Address channel. This reflects internal queue loading so its assertion is prior to it appearing on the Write Address Channel. This signal qualifies the value on the `s2mm_wr_len` output port for use by external logic.
s2mm_wr_len	S2MM	Output	0	This bus reflects the value that is placed on the `m_axi_s2mm_awlen` output (AXI4 Write Address Channel) when it is pulled from the internal queue. The value is only valid when the signal `s2mm_ld_nxt_len` is asserted.
AXI4 Write Interface Signals				
m_axi_s2mm*	M_AXI_S2MM*	–	–	See the *Vivado AXI Reference Guide* (UG1037) [Ref 2] for the description of AXI4 Signals.

Table 2-6: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
Stream to Memory Map Command/Status Channel Asynchronous Clock and Reset				
m_axis_s2mm_cmdsts_awclk	S2MM Command & Status	Input	–	S2MM Command Interface Clock. asynchronous clocks are enabled in the Vivado IDE. The frequency of this clock is expected to be equal or less than the `m_axi_s2mm_aclk`.
m_axis_s2mm_cmdsts_aresetn	S2MM Command & Status	Input	–	S2MM Command Interface Reset (Active-Low). This reset input is only used if asynchronous clocks are enabled in the Vivado IDE. Must be asserted for three clock periods of the slower of `m_axis_s2mm_cmdsts_awclk` and `m_axi_s2mm_aclk`.

Design Information

Command Interface

The DataMover operations are controlled by an AXI slave stream interface that receives transfer commands from the user logic. The MM2S and the S2MM each have a dedicated command interface. A command is loaded with a single data beat on the input command stream interface. The width of the command word is normally 72 bits if 32-bit AXI addressing is being used in the system. However, the command word width is extended (by parameterization) if the system address space grows beyond 32 bits. For example, a 64-bit address system requires the command word to be 104 bits wide to accommodate the wider starting address field. The command interface is an AXI4-Stream interface so the total number of bits should be an integer multiple of 8. For example, if the address space is configured as 33 bits, the address portion in the command should be stuffed to make it 40 bits. This is to ensure compliance with the AXI4-Stream protocol. AXI Datamover will internally ignore the stuffed bits.

The format of the command word is shown in Figure 2-1 and detailed in Table 2-7. It is the same for either the MM2S or S2MM DataMover elements. The command format allows the specification of a single data transfer from 1-byte to 8,388,607 bytes (7FFFFF hex bytes). A command loaded into the command interface is often referred to as the parent command of a transfer. The DataMover automatically breaks up large transfers into intermediate bursts (child transfers) that comply with the AXI4 protocol requirements.

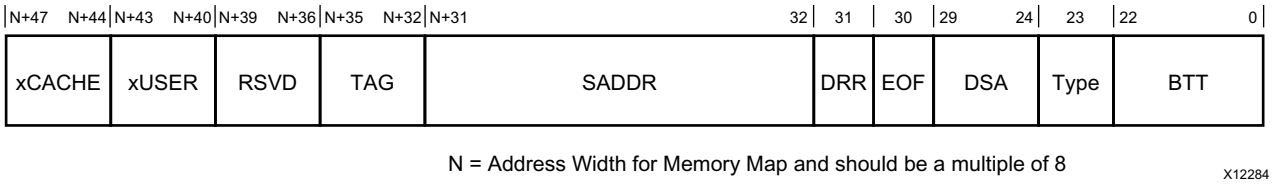

Figure 2-1: **Command Word Layout**

Table 2-7: **Command Word Description**

Bits	Field Name	Description
(N + 47) – (N + 44)[2]	xCACHE	The value written in this field appears on `m_axis_mm2s_arcache` for MM2S block and `m_axis_s2mm_awcache` for S2MM block.
(N + 43) – (N + 40)[2]	xUSER	The value written in this field appears on `m_axis_mm2s_aruser` for MM2S block and `m_axis_s2mm_awuser` for S2MM block.
(N+39) – (N+36)[1]	RSVD	**Reserved**
(N+35) – (N+32)[1]	TAG	**Command TAG** This field is an arbitrary value assigned that is user assigned to the command. The TAG flows through the DataMover execution pipe and gets inserted into the corresponding status word for the command.
(N+31) –32[1]	SADDR	**Start Address** This field indicates the starting address to use for the memory-mapped side of the transfer requested by the command. If DRE is enabled, the lower order address bits of this field indicate the starting alignment to load on the memory mapped side of the DRE.
31	DRR	**DRE ReAlignment Request** This bit is only used if the optional DRE is included by parameterization. The bit indicates that the DRE alignment needs to be re-established prior to the execution of the associated command. The DRE stream side alignment is derived from the DSA field of the command. The Memory Mapped side alignment is derived from the least significant bits of the SADDR field.
30	EOF	**End of Frame** This bit indicates that the command is an End of Frame command. This generally affects the MM2S element (Read Master) because it causes the stream output logic to assert the `TLAST` output on the last data beat of the last transfer needed to complete the command. *If DRE is included*, this also causes the DRE to flush out any intermediate data at the conclusion of the last transfer of the command and submit it to the stream output (in the case of the MM2S Read Master) or to the AXI Write Data Channel (in the case of the S2MM Write Master). This bit should be set for S2MM commands (when the Indeterminate BTT mode is disabled) when `TLAST` is expected to arrive with the number of bytes programmed in the command.

Table 2-7: **Command Word Description *(Cont'd)***

Bits	Field Name	Description
29 – 24	DSA	**DRE Stream Alignment** This field is only used by the MM2S and if the optional MM2S DRE is included by parameterization. The field is only used when the DRR bit of the associated command is also set to 1. This 6-bit field indicates the reference alignment of the MM2S Stream Data Channel for the optional DRE. The value is byte-lane relative. A value of 0 indicates byte lane 0 (least significant byte) is the reference byte lane; a value of 1 indicates byte lane 1, and so on. Valid values are dependent upon the parameterized data width of the stream data channel. For example, a 32-bit wide data channel has only 4-byte lane positions and thus the DSA field can only have values of 0 to 3. ***Note:*** DRE alignment on the associated memory-mapped side is derived from the least significant bits of the SADDR value of the command.
23	Type	This field determines the type of AXI4 access. Setting this to 1 enables INCR. A value of 0 enables FIXED address AXI4 transaction.
22 to 0	BTT	**Bytes to Transfer** This 23-bit field indicates the total number of bytes to transfer for the command. Transfers can be from 1 up to 8,388,607 bytes. A value of 0 is not allowed and causes an internal error from the DataMover. The actual number of BTT bits used by the DataMover is controlled by the parameters 'Width of BTT field' for MM2S and S2MM respectively.

Notes:

1. N is the width of memory-mapped address bus. If the address width is configured as 33, then N should be considered as 40 while generating the command and the upper 7 address bits should be stuffed. The stuffed bits are ignored by the AXI Datamover. This must be done to ensure that the width of command bus is a multiple of 8.
2. These fields are valid only when **Enable xUSERxCACHE** is checked in the Vivado Integrated Design Environment (IDE).

Command FIFO

The command interface of a DataMover element is designed to allow command queuing. The commands are queued in a FIFO.

The command FIFO is by default a synchronous FIFO using the same clock as the memory mapped data and address channels of the associated DataMover element. However, you can specify an asynchronous command interface FIFO. This allows the command interface to be clocked at a different (generally much slower) frequency than the memory-mapped interface.

Command Stream Interface Timing

Loading a command into the command FIFO is mechanized by a single AXI4-Stream data beat. An example of loading five commands into the MM2S Command FIFO is shown in Figure 2-2. In this scenario, the command FIFO is synchronous to the memory-mapped interface. The `tlast` and `tstrb` signals are ignored.

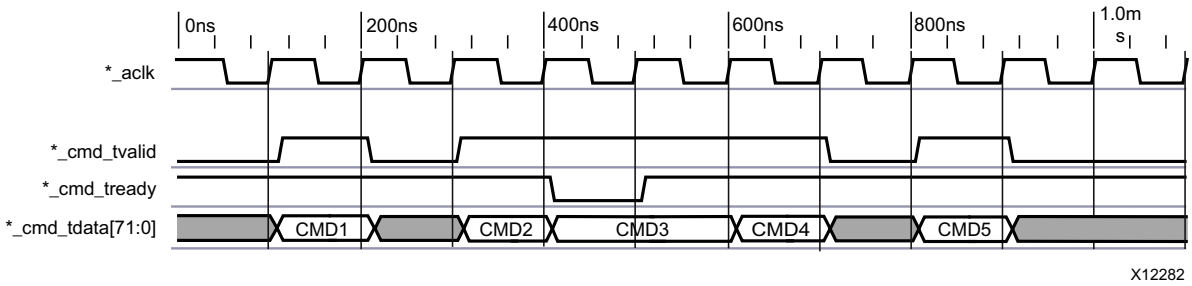

Figure 2-2: **Loading Commands via the Command Interface**

Status Interface

The status of DataMover transfer operations is provided by an AXI Master Stream interface that relays transfer status to the user logic. The MM2S and the S2MM each have a dedicated Status Interface. A status word is read with a single data beat on the Status Stream interface. The width of the status word is fixed at 8 bits except when S2MM is enabled in indeterminate mode (S2MM Status Format in Indeterminate BTT Mode (IBTT).)

The format of the status word is shown in Figure 2-3 and detailed in Table 2-8. It is the same for either the MM2S or S2MM DataMover elements.

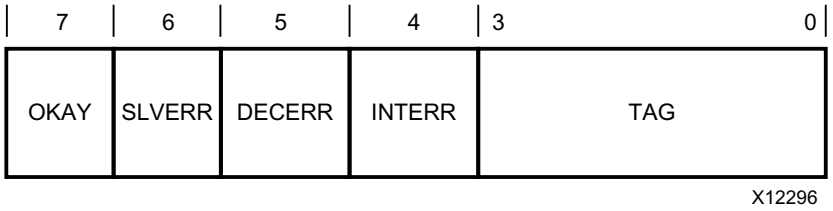

Figure 2-3: **Normal Status Word Layout**

Table 2-8: **Status Word Details**

Bits	Field Name	Description
7	OKAY	**Transfer OKAY** This bit indicates that the associated transfer command has been completed with the OKAY response on all intermediate transfers. 0 = Command had a non-OKAY response during all associated transfers. 1 = Command had a OKAY response during all associated transfers.
6	SLVERR	**Slave Error** Indicates the DataMover encountered a slave reported error condition for the associated command. This is received by the response inputs from the AXI4 interface. 0 = No Error 1 = Slave Asserted Error Condition

Table 2-8: **Status Word Details *(Cont'd)***

Bits	Field Name	Description
5	DECERR	**Decode Error** Indicates the DataMover encountered an address decode error condition for the associated command. This is received by the response inputs from the AXI4 interface and indicates an address decode timeout occurred on an address generated by the DataMover element while executing the corresponding command. 0 = No Error 1 = Address Decode Error Condition
4	INTERR	**Internal Error** Indicates the DataMover encountered an internal error condition for the associated command. A BTT (Bytes to Transfer) value of 0 (zero) in the command word can cause this assertion. This error is also flagged by the S2MM function when the Indeterminate BTT mode is not enabled and the number of bytes received on the AXIS is not the same as what was programmed in the BTT field of the command. In other words, this error is flagged if `TLAST` comes early or late or never. 0 = No Error 1 = Internal Error Condition
3 to 0	TAG	**TAG** This 4-bit field echoes the value of the TAG field of the associated input command whose completion generated the status.

S2MM Status Format in Indeterminate BTT Mode (IBTT)

The DataMover S2MM function can be parameterized to enable support for stream data transfer of an indeterminate number of bytes. This is defined as where the S2MM is commanded (by the BTT command field) to transfer a fixed number of bytes, but it is unknown how many bytes are actually going to be received from the incoming stream interface (at the assertion of `s_axis_s2mm_tlast`). Supporting this operation mode requires additional hardware in the S2MM function, and additional fields in the status word indicating the actual count of the bytes received from the stream interface for the commanded transfer, and whether the `tlast` was received during the transfer.

The format of the S2MM status word with Indeterminate BTT mode enabled is shown in Figure 2-4 and detailed in Table 2-9. This status format does not apply to the MM2S DataMover status interface. See Indeterminate BTT Mode for more information.

Figure 2-4: **S2MM Status Format in IBTT Mode**

Table 2-9: **Special S2MM Status Word Details (Indeterminate BTT Mode Enabled)**

Bits	Field Name	Description
31	EOP	**End of Packet** This bit indicates that the S2MM Stream input received a `tlast` assertion during the execution of the DataMover command associated with the status word. This is not an error condition.
30 to 8	BRCVD	**Bytes Received** This field indicates the actual number of bytes received on the stream interface at the point where `s_axis_s2mm_tlast` was asserted by the Stream Master.
7	OKAY	**Transfer OKAY** This bit indicates that the associated transfer command has been completed with the OKAY response on all intermediate transfers. 0 = Command had a non-OKAY response during all associated transfers. 1 = Command had a OKAY response during all associated transfers.
6	SLVERR	**Slave Error** Indicates the DataMover encountered a slave reported error condition for the associated command. This is received by the response inputs from the AXI4 interface. 0 = No Error 1 = Slave Asserted Error Condition
5	DECERR	**Decode Error** Indicates the DataMover encountered an address decode error condition for the associated command. This is received by the response inputs from the AXI4 interface and indicates an address decode timeout occurred on an address generated by the DataMover element while executing the corresponding command. 0 = No Error 1 = Address Decode Error Condition
4	INTERR	**Internal Error** Indicates the DataMover encountered an internal error condition for the associated command. A BTT (Bytes to Transfer) value of 0 (zero) in the command word can cause this assertion. Additional conditions are to be determined. 0 = No Error 1 = Internal Error Condition
3 to 0	TAG	**TAG** This 4-bit field echoes the value of the TAG field of the associated input command whose completion generated the status.

Status FIFO

The Status Interface of a DataMover is designed to allow for status queuing corresponding to the available command queuing on the Command Interface. The status values are "queued" in a FIFO.

The Status FIFO is by default a synchronous FIFO using the same clock as the memory mapped interface of the associated DataMover element. However, you can specify an asynchronous Command/Status Interface FIFO. This allows the command and status interfaces to be clocked at a different (generally much slower) frequency than the associated memory-mapped interface.

Status Read Interface Timing

Reading a status word from the Status FIFO is mechanized by a single AXI4-Stream data beat. An example of reading five status entries from the MM2S Status FIFO is shown in Figure 2-5. In this scenario, the Status FIFO is synchronous to the memory-mapped address and data channel clock. The example illustrates that a status word is considered read from the Status FIFO only when the `tvalid` and `tready` handshake signals are both asserted at the rising edge of the synchronizing clock.

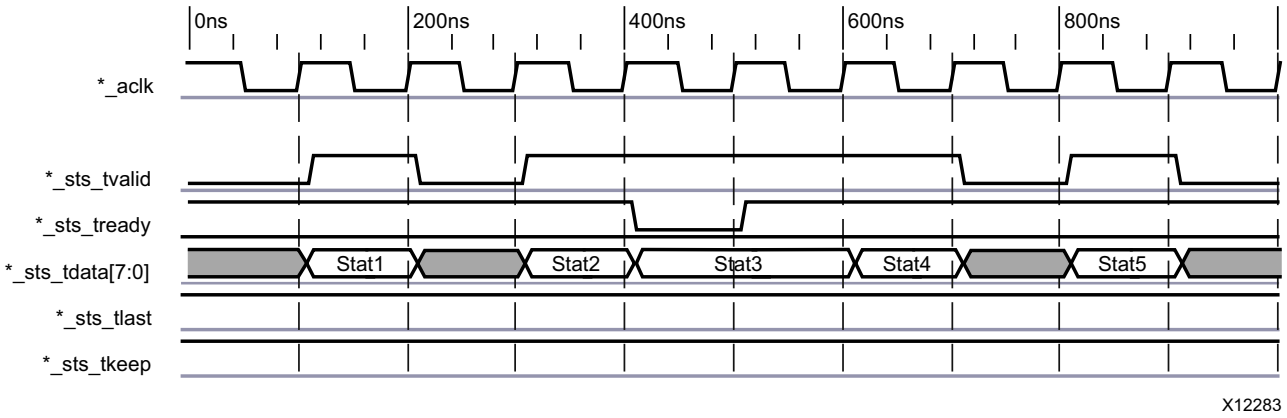

Figure 2-5: **Reading Status over the Status Interface**

MM2S Store and Forward

The MM2S can include an optional Store and Forward block when the **Enable Store Forward** parameter is enabled. Enabling this parameter ensures that child transfers are not posted to the AXI4 Read Address Channel if there is not enough space left in the Store and Forward FIFO for the data.

S2MM Store and Forward

The S2MM can include an optional Store and Forward block when the **Enable Store Forward** parameter is enabled. Enabling this parameter ensures that transfers are not posted to the AXI4 Write Address Channel until all of the data needed for the requested transfer is present in the Store and Forward FIFO.

Indeterminate BTT Mode

The AXI DataMover S2MM function has a special operating mode to support the case when the amount of data being received in the Stream Channel is unknown (or indeterminate).

An additional feature of the Indeterminate BTT mode is the absorption of overflow data from the input stream channel. Overflow is defined as the stream data that is received that exceeds the BTT value for the corresponding parent transfer command and the EOF bit is also set in that command. The data absorption occurs from the point of the BTT value being reached to the next `TLAST` data beat.

Address Posting Control and Status Interface

The AXI Data Mover provides additional Address Posting Control and its Status signals that allows you to exercise control over initiating transactions on the memory map side even when corresponding commands already were issued on the command interface. These signals can be controlled by user logic to minimize the need for the AXI DataMover to throttle the AXI4 Interface.

These ports are:

- `mm2s_allow_addr_req` (input to DataMover)
- `mm2s_addr_req_posted` (output from DataMover)
- `mm2s_rd_xfer_cmplt` (output from DataMover)
- `s2mm_allow_addr_req` (input to DataMover))
- `s2mm_addr_req_posted` (output from DataMover)
- `s2mm_wr_xfer_cmplt` (output from DataMover)
- `s2mm_ld_nxt_len` (output from DataMover)
- `s2mm_wr_len` (output from DataMover)

Note: These signals are irrelevant for most of the use cases and are classified for advance use. By default the access to these signals is disabled. You can gain access to these signals by enabling them in the customization Vivado IDE.

Example Design

The connection of these ports to an external user logic is shown in Figure 2-6. This is representative of the loopback connection on the AXI4-Stream side with external storage facility. The user logic should have the ability to control the AXI DataMover Address posting to the AXI4 bus through the `mm2s_allow_addr_req` and `s2mm_allow_addr_req` signals. When asserted High, the associated DataMover Address Controller is allowed to post a transfer address to the AXI4 bus and thus commit to a transfer. The `mm2s_allow_addr_req` controls the MM2S Address Controller and the `s2mm_allow_addr_req` controls the S2MM Address Controller. When asserted Low, the associated Address Controller is inhibited from posting transfer address to the AXI4 bus.

The AXI DataMover also provides status back to the user logic indicating when an address set has been committed to the AXI4 bus through the `mm2s_addr_req_posted` and `s2mm_addr_req_posted` signals. In addition, the MM2S and S2MM also provide a status bit indicating when a scheduled Read or Write Data Channel transfer has completed through the `mm2s_rd_xfer_cmplt` and `s2mm_wr_xfer_cmplt` signals.

The S2MM function provides two additional outputs (`s2mm_wr_len` and `s2mm_ld_nxt_len`) that are used to provide some lookahead to the user monitoring logic by indicating the needed data beats for each of the upcoming Write Transfers that are being queued in the S2MM Write Data Controller. By monitoring the input stream from the MM2S, the user monitoring logic can count the incoming data and notify the write side monitor logic when the exact amount of data has been received to satisfy a queued write transfer.

This control and status mechanism allows the AXI DataMover to pipeline Read requests to the AXI4 without over-committing the Data FIFO capacity (filling it up and throttling the AXI4 Read data Channel). It also can keep the DataMover from issuing Write transfers until the write data is actually present in the Data FIFO. See the timing diagrams in Figure 2-8 and Figure 2-9 for a better understanding of usage of these signals.

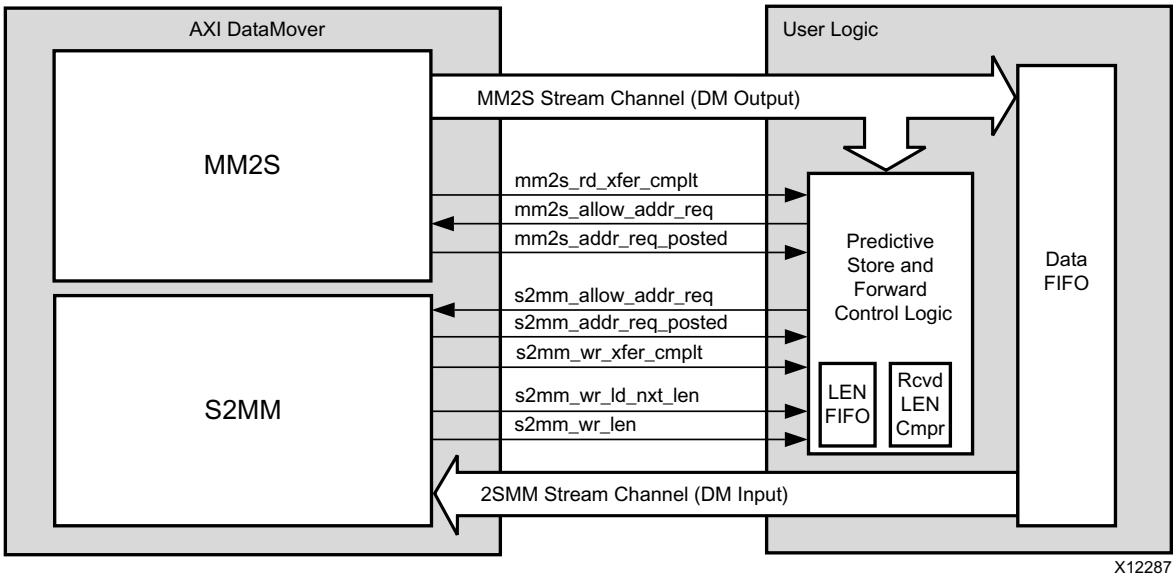

Figure 2-6: **AXI DataMover Address Posting Control and Status Interface Use Case**

Request Spawning

One important aspect of the DataMover operation is the ability to spawn multiple child AXI requests when executing a single command from the corresponding Command FIFO. Request spawning occurs when the requested Bytes to Transfer (BTT) specified by the command exceeds a parameterized burst data beat limit (default is 16 but can also be set to 32, 64, 128, or 256).

MM2S Soft Shutdown

The DataMover MM2S soft shutdown is initiated by the active-High assertion of the input signal `mm2s_halt`. During a soft shutdown, DataMover gracefully completes the existing transactions on the AXI4 side. You will find data on the streaming side sometime while its exiting gracefully. The `m_axis_mm2s_tdata` data output is then driven with invalid data values. The MM2S completes all committed memory map requests presented on the MM2S Memory Map Address Channel. Input data from the Memory Map Data Channel received during the cleanup operations are discarded.

When the soft halt operations are completed, the MM2S asserts the `mm2s_halt_cmplt` output. This output remains asserted until the MM2S is reset through the hard reset input `m_axi_mm2s_aresetn` (or `m_axis_mm2s_cmdsts_aresetn` if asynchronous command interface is in use).

S2MM Soft Shutdown

The S2MM soft shutdown is initiated by the active-High assertion of the input signal `s2mm_halt`. During a soft shutdown, the S2MM function asserts the S2MM Stream `s_axis_s2mm_tready` output signal and ignores the remaining S2MM Stream inputs. The S2MM completes all committed memory map requests presented on the S2MM Memory Map Address Channel. Output data to the Memory Map Data Channel transmitted during the cleanup operations are invalid data values.

When the soft halt operations are completed, the S2MM asserts the `s2mm_halt_cmplt` output. This output remains asserted until the S2MM is reset by the hard reset input `m_axi_s2mm_aresetn` (or `m_axis_s2mm_cmdsts_aresetn` if the asynchronous command interface is in use).

DataMover Basic

Some applications of the DataMover do not need the high-performance features it provides. In these applications, resource utilization is more important than performance. The DataMover provides the ability to select a reduced function option through the Vivado IDE.

The following feature simplifications characterize the Basic version:

- 32-bit and 64-bit Memory Mapped Data Width and 8, 16, 32, and 64-bit Stream width (parameterized). Starting transfer address must be aligned to address boundaries that are multiples of the Stream Data width (in bytes).
- Maximum AXI4 Burst Length support of 2, 4, 8, 16, 32, and 64 data beats (parameterized).
- No DRE support.

- One-Deep Command and Status Queuing (Parent command). The Command and Status FIFOs are replaced with a FIFO register for each.
- Commanded transfer lengths (Bytes to Transfer) are limited to the Max AXI4 Burst Length multiplied by the Stream data width (in bytes).

  Example: Maximum burst length = 32, Stream Data Width = 4 bytes (32 bits), the maximum commanded transfer length (BTT) is 128 bytes.
- No breakup of transfers into smaller bursts.
- 4K byte boundaries are not monitored.
- Automatic transfer splitting at an AXI 4K address boundary is not supported.
- No Store and Forward support.
- When the streaming data width is different than the memory-mapped data width, the axi datamover exhibits narrow burst on the memory-mapped side. For example, if the memory-mapped data width is 32 and the streaming data width is 8, the arsize/awsize output is `3'b000`.

DataMover Basic Command Interface

The format of the Basic command word is shown in Figure 2-7 and detailed in Table 2-10.

Note: * = S2MM or MM2S

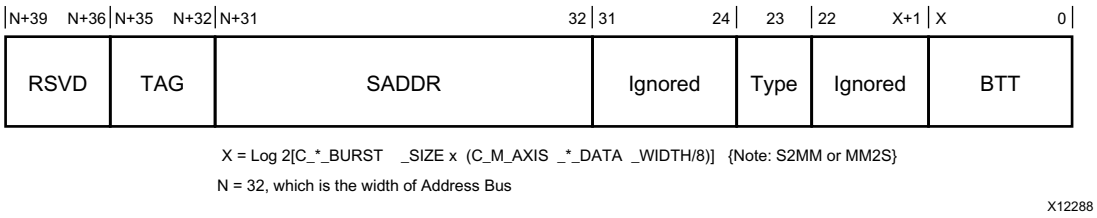

Figure 2-7: **DataMover Basic Command Word Layout**

Table 2-10: **DataMover Basic Command Word Details**

Bits	Field Name	Description
(N+39) – (N+36)(1)	RSVD	**Reserved** This field is reserved to pad the command width to an even multiple of 8 bits. (required for AXI4-Stream interfaces)
(N+35) – (N+32)(1)	TAG	**Command TAG** You assign this field an arbitrary value to the command. The TAG flows through the DataMover execution pipe and gets inserted into the corresponding status word for the command.
(N+31) –32(1)	SADDR	**Start Address** This field indicates the starting address to use for the memory mapped side of the transfer requested by the command.
31 – 4	Ignored	This field is ignored by the DataMover Basic. Can be any value but zeroes are recommended.
23	TYPE	**TYPE** Specifies the type of AXI4 access. A '1' means INCR access, while '0' means a FIXED address access.
22 –(X+1)	Ignored	This field is ignored by the DataMover Basic. Can be any value but zeroes are recommended.
X – 0	BTT	**Bytes to Transfer** This field indicates the total number of bytes to transfer for the command. The maximum allowed value is set by the following formula: Max Burst Length x (Streaming Data width)/8

Notes:

1. N is the width of the Address Memory Mapped Address bus. If the address space selected is 33 bits, N should be considered as 40 and the upper 7 bits of the address should be stuffed. This has to be done to make the data width a multiple of 8.

DataMover Basic Status Interface

The format of the status word is the same as the full version and is shown in Figure 2-3, and detailed in Table 2-8.

Example DataMover Read(MM2S) Timing

Figure 2-8 illustrates example timing on read (MM2S) path in synchronous mode.

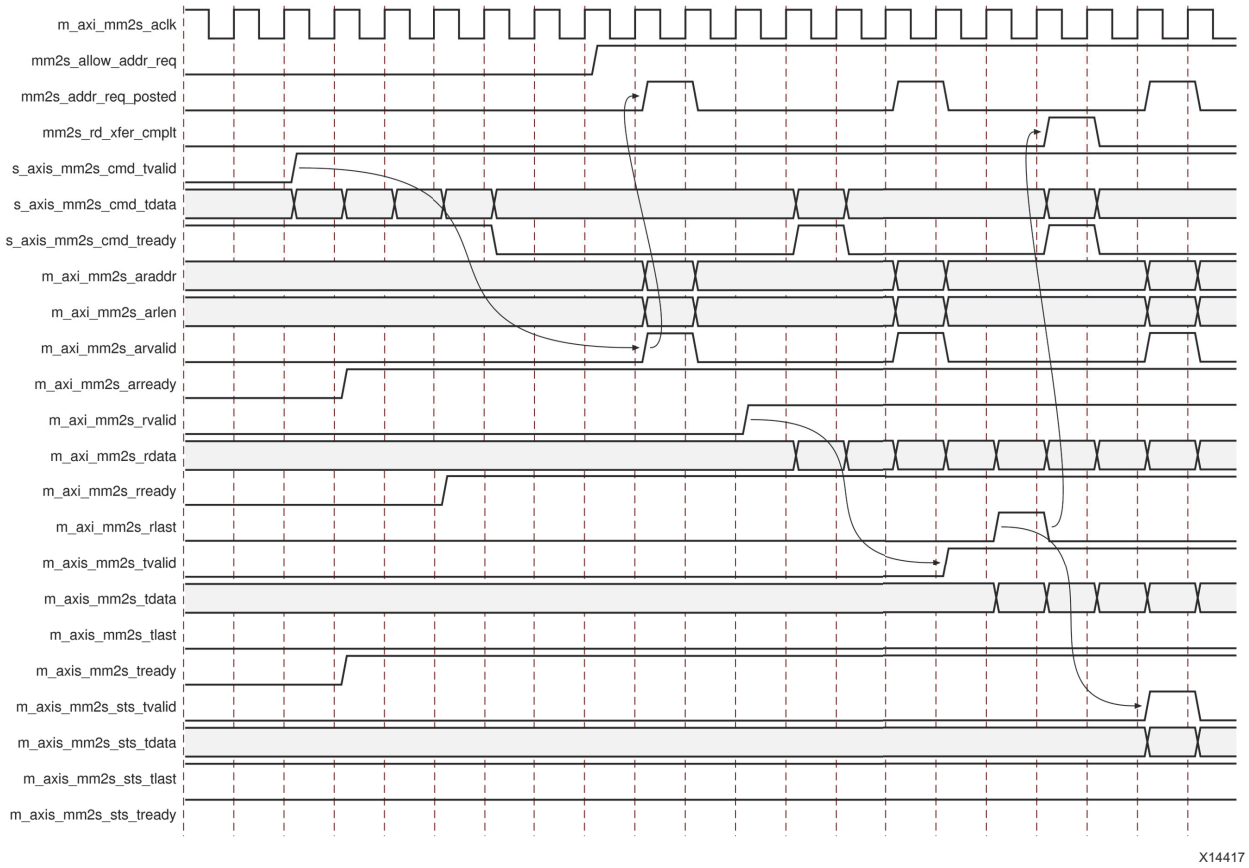

Figure 2-8: **Example Timing on Read (MM2S) Path in Synchronous Mode**

Dataflow:

1. After receiving commands on the AXI4-Stream command interface (`s_axis_mm2s_cmd_tvalid`) and if `mm2s_allow_addr_req` is High, AXI DataMover initiates a read cycle on the AXI4 interface by asserting `m_axi_mm2s_arvalid` and other address bus signals.
2. It also asserts `mm2s_addr_req_posted` indicating address is posted on the MMap interface.
3. Read data is stored in internal FIFO if enabled.
4. AXI DataMover starts sending out data on the streaming interface by asserting `m_axis_mm2s_tvalid` and other associated signals.
5. AXI DataMover asserts `mm2s_rd_xfer_cmplt` indicating data is completely read on the MMap interface.
6. AXI4-Stream Status interface signals `m_axis_mm2s_sts_tvalid` and other associated signals are asserted indicating the status for a particular command that was posted on command interface

IMPORTANT: *A single parent command can generate multiple child commands on the AXI4 Interface. Status signals are asserted when all child commands are processed.*

Example DataMover Write(S2MM) Timing

Figure 2-9 illustrates example timing on write (S2MM) path in synchronous mode.

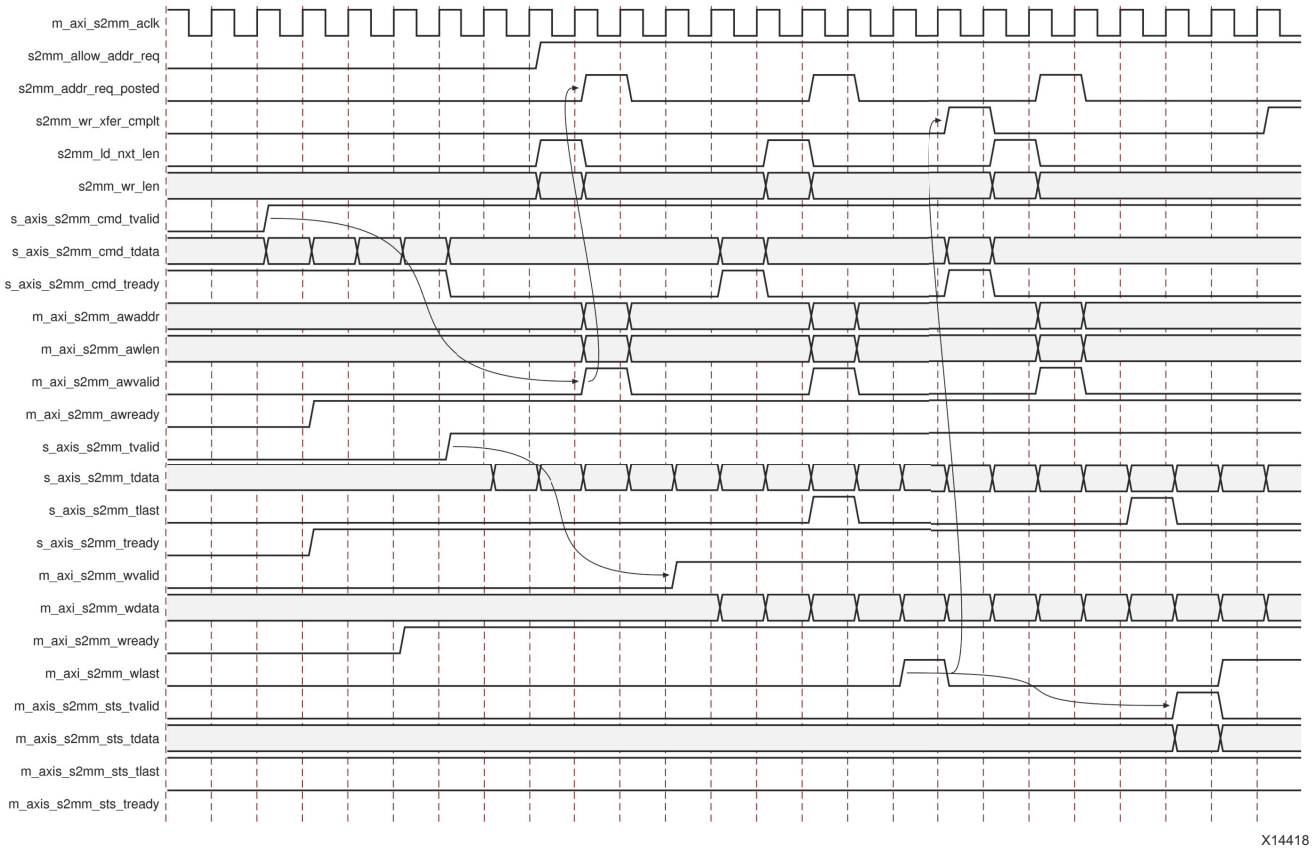

Figure 2-9: **Example Timing on Write (S2MM) Path in Synchronous Mode**

Dataflow:

1. After receiving commands on the AXI4-Stream command interface (`s_axis_s2mm_cmd_tvalid`) and if `s2mm_allow_addr_req` is High, AXI DataMover initiates write cycles on the AXI4 interface by asserting `m_axi_s2mm_awvalid` and other address bus signals.
2. AXI DataMover also asserts `mm2s_addr_req_posted` indicating address is posted on MMap interface.
3. AXI DataMover accepts data on the streaming interface by asserting `s_axis_s2mm_tready`.
4. Incoming data is stored in FIFO if enabled.
5. AXI DataMover starts sending out data on MMap interface by asserting `m_axi_s2mm_wvalid` and other associated signals.

6. AXI DataMover asserts `s2mm_wr_xfer_cmplt` indicating data is completely written on the MMap interface.
7. AXI4-Stream Status interface signals `m_axis_s2mm_sts_tvalid` and other associated signals are asserted indicating the status for a particular command that was posted on command interface.
8. AXI DataMover also asserts additional signals `s2mm_ld_nxt_len` along with `s2mm_wr_len` indicating the burst length of the write transfer to be posted on the AXI4 interface.

IMPORTANT: *A single parent command can generate multiple child commands on the AXI4 Interface. Status signals are asserted when all child commands are processed.*

Note: In the absence of any S2MM command, AXI DataMover will pull the `s_axis_s2mm_tready` signal to Low after taking in four beats of streaming data. This will throttle the input data stream. To have a minimum amount of throttling, ensure that a valid command is issued to the S2MM interface much before the actual data arrives.

AXI DataMover does not support null bytes. (`TKEEP` completely deasserted). A start of a streaming packet is identified by `TVALID` while its end is determined by `TLAST`. AXI DataMover does not support sparse/null `TKEEP` between the packet boundaries. `TKEEP` can be sparse only at `TLAST` beat. `TKEEP` can also be sparse at the start of a packet when DRE (unaligned transfers) is enabled.

Designing with the Core

Figure 3-1 and Figure 3-2 show typical use cases of AXI DataMover. Figure 3-1 shows a multichannel application of DataMover in the MM2S path.

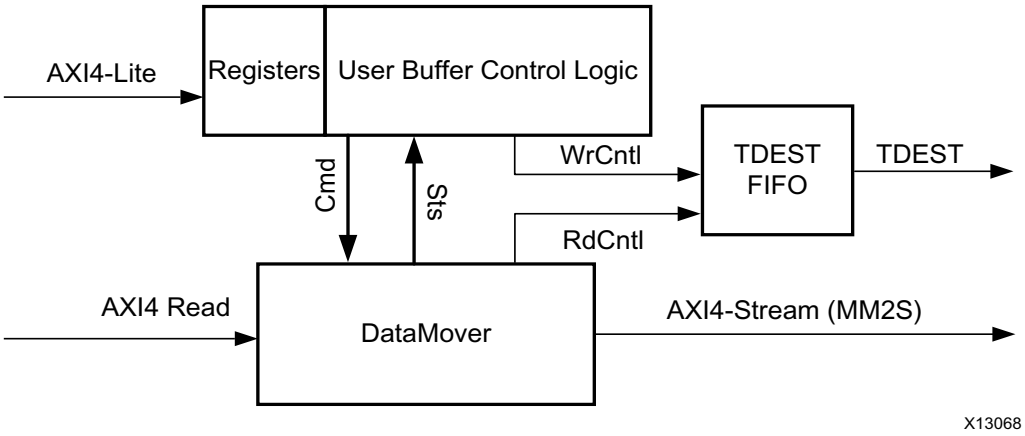

Figure 3-1: **Typical Application of MM2S DataMover**

Figure 3-2 shows a multichannel application of DataMover in the S2MM path. Incoming TDEST information can be used to pick the corresponding destination address on the AXI4 side and the same TDEST value can be stored in the register space.

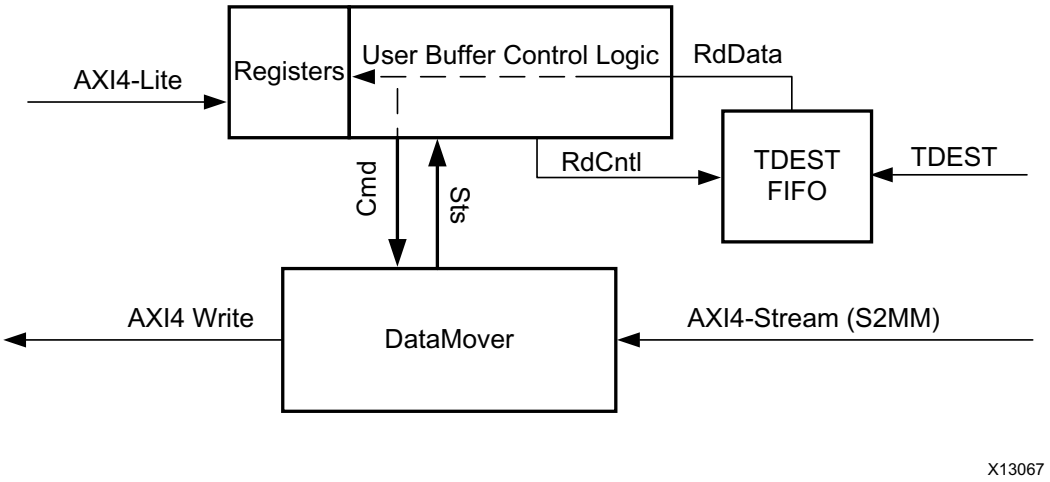

Figure 3-2: **Typical Application of S2MM DataMover**

Clocking

The AXI DataMover has two clock inputs for each of the MM2S and S2MM blocks for a total of four clock inputs. The `m_axi_mm2s_aclk` is the main synchronizing clock for the MM2S block. This clock synchronizes both the associated AXI4 interface and Stream interface. The second clock for the MM2S element is `m_axis_mm2s_cmdsts_aclk`. This clock is used only when MM2S is configured in asynchronous mode. When used, it synchronizes the user sides of the Command and Status interfaces.

The S2MM block has identical clocking schemes as the MM2S block but with two different clocks, the `m_axi_axi_s2mm_aclk` and `m_axis_s2mm_cmdsts_awclk`.

Resets

The AXI DataMover has two reset inputs for each of the MM2S and S2MM blocks for a total of four reset inputs.

AXI DataMover requires that the input reset assertion must be a minimum of three clock periods of the synchronizing clock. Table 3-1 shows the clock and reset signals and its associated interface.

Table 3-1: **Clock, Reset and its Associated Interface**

		Asynchronous Mode	
Blocks	**Interface**	**Disabled**	**Enabled**
MM2S	Memory map and Streaming Interface	m_axi_mm2s_aclk and m_axi_mm2s_aresetn	m_axi_mm2s_aclk and m_axi_mm2s_aresetn
	Command and Status Interface	m_axi_mm2s_aclk and m_axi_mm2s_aresetn	m_axis_mm2s_cmdsts_aclk and m_axis_mm2s_cmdsts_aresetn
		Asynchronous Mode	
Blocks	**Interface**	**Disabled**	**Enabled**
S2MM	Memory map and Streaming Interface	m_axi_s2mm_aclk and m_axi_s2mm_aresetn	m_axi_s2mm_aclk and m_axi_s2mm_aresetn
	Command and Status Interface	m_axi_s2mm_aclk and m_axi_s2mm_aresetn	m_axis_s2mm_cmdsts_awclk and m_axis_s2mm_cmdsts_aresetn

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 3]
- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 4]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite.

The AXI DataMover can be found in **\AXI_Infrastructure** and also in **Embedded_Processing\AXI_Infrastructure** in the IP catalog.

To access the AXI DataMover, perform the following:

1. Open a project by selecting **File** then **Open Project** or create a new project by selecting **File** then **New Project** in the Vivado design tools.
2. Open the IP catalog and navigate to any of the taxonomies.
3. Double-click **AXI DataMover** to bring up the **AXI DataMover** Vivado Integrated Design Environment (IDE).

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 3].

The AXI DataMover Vivado IDE contains one screen with two tabs (Figure 4-1 and Figure 4-2) that provide information about the core, allow configuration of the core, and provides the ability to generate the core.

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 4] for detailed information. Vivado IDE might auto-compute certain configuration values when validating or generating the design, as noted in this section. You can view the parameter value after successful completion of the `validate_bd_design` command.

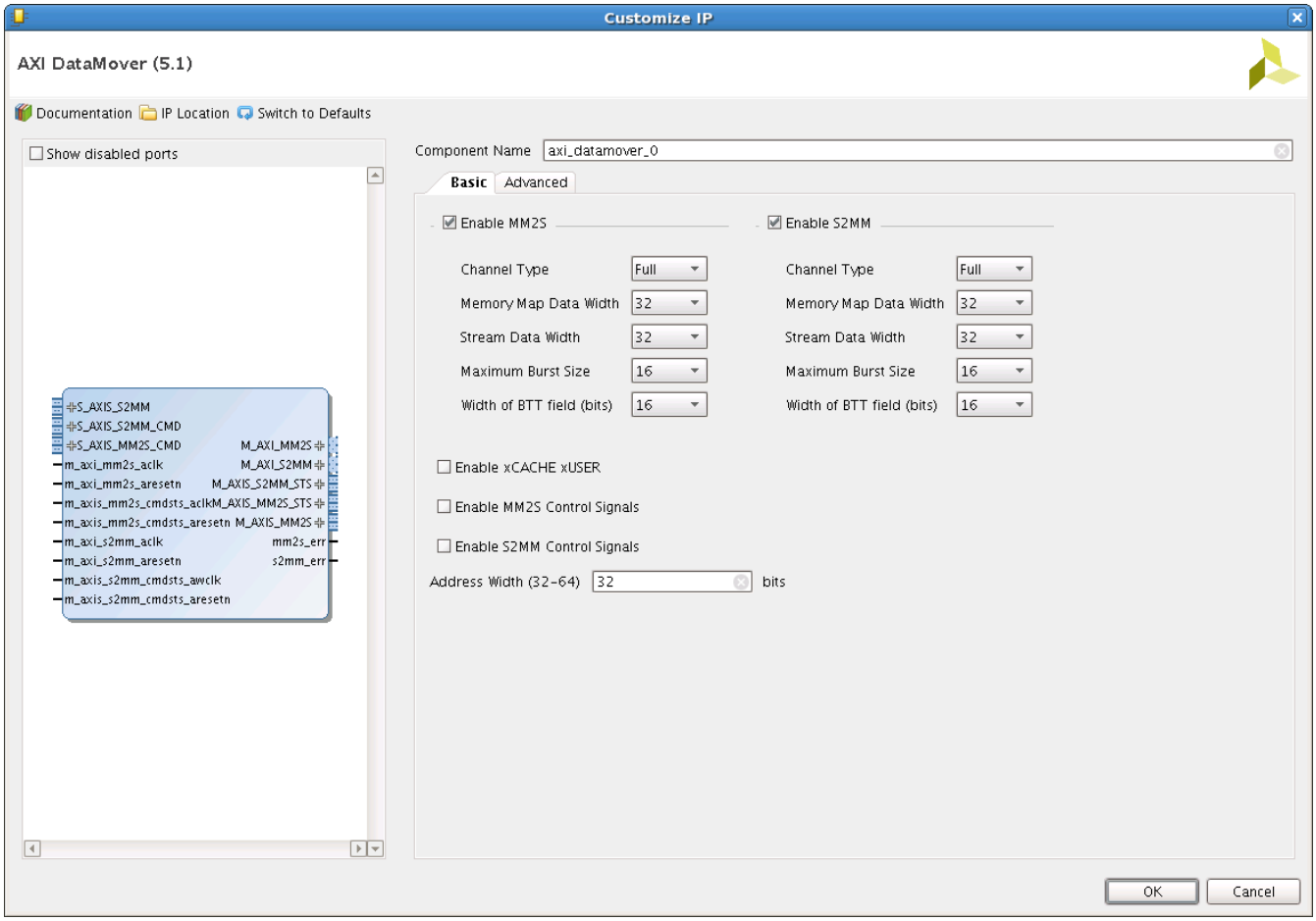

Figure 4-1: **IP Catalog — Basic Tab**

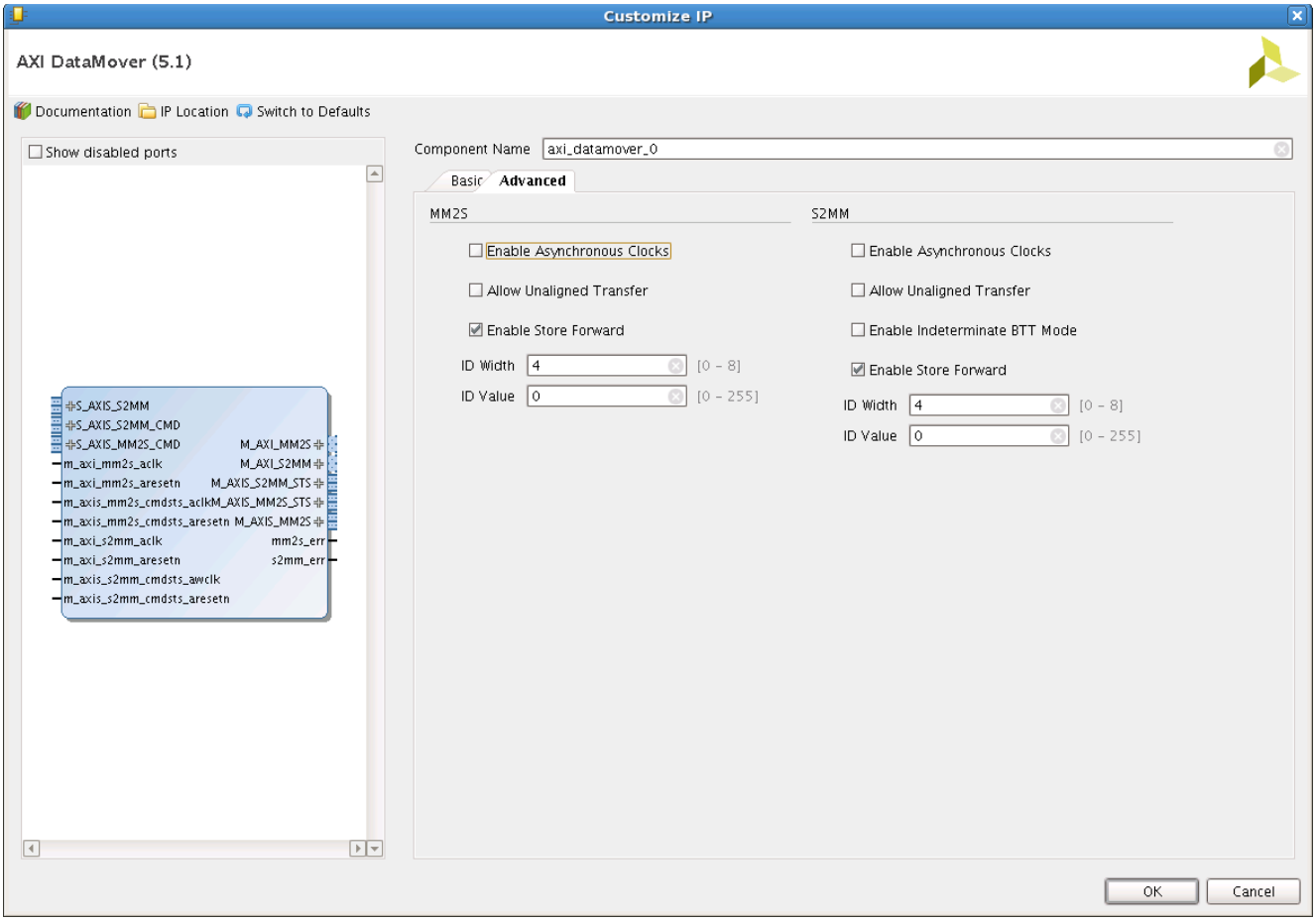

Figure 4-2: **IP Catalog — Advanced Tab**

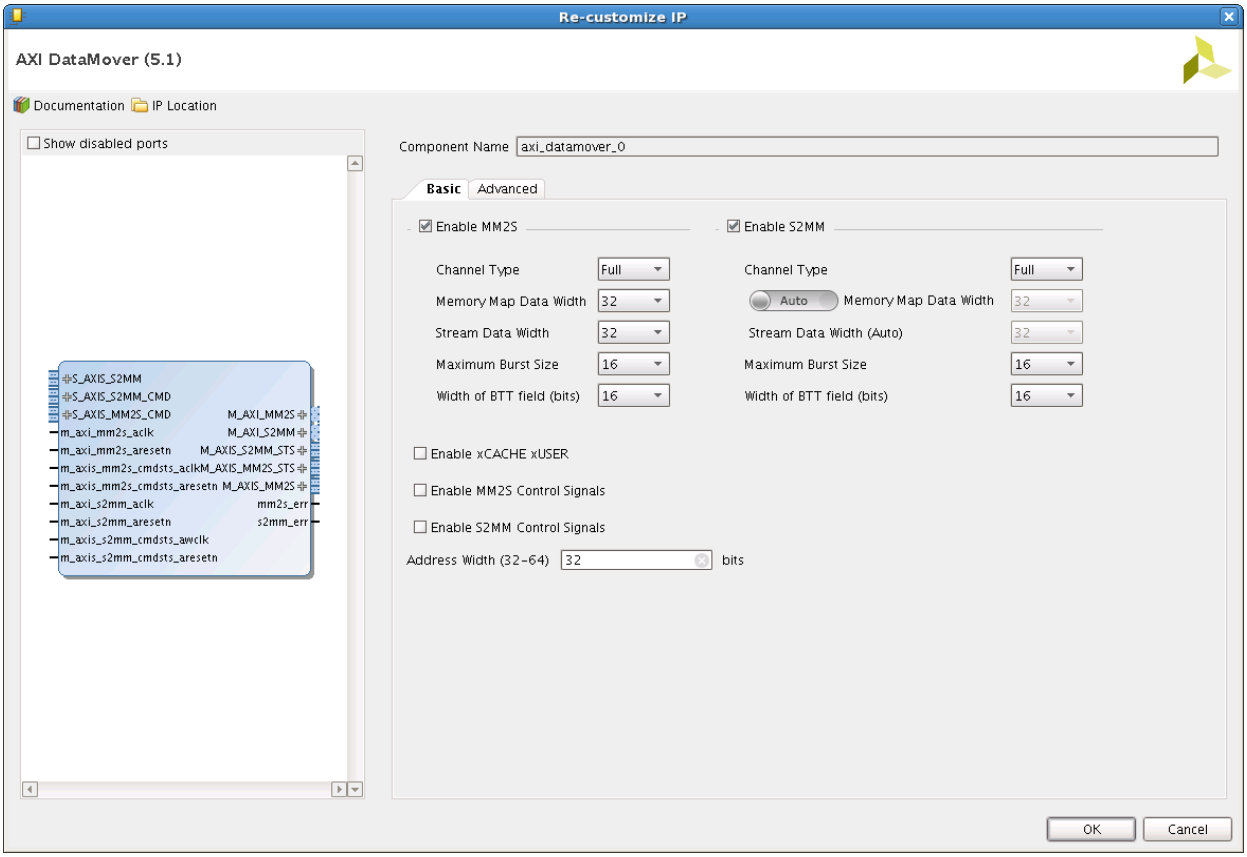

Figure 4-3: **IP Integrator — Basic tab**

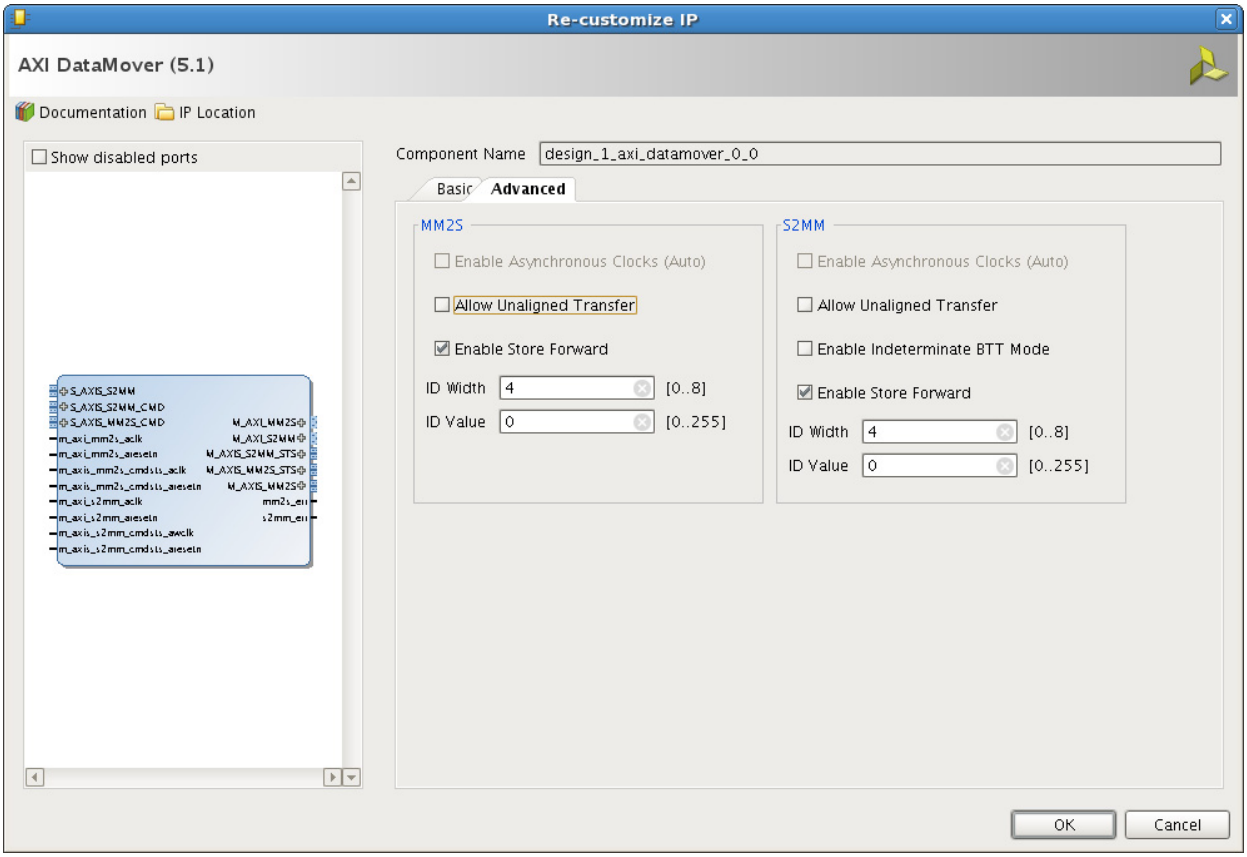

Figure 4-4: **IP Integrator — Advanced Tab**

Component Name – The base name of the output files generated for the core. Names must begin with a letter and can be composed of any of the following characters: a to z, 0 to 9, and "_".

Basic Options

The following describes the fundamental options that affect the MM2S and S2MM channels of the AXI DataMover core.

- **MM2S Channel Options** – This box allows you to configure the MM2S channel options.
  - **Channel Type** – Choose Full or Basic. Selecting Full allows the MM2S channel to be configured for all possible combination and advance features. The Basic mode restricts some of features and allows the MM2S to be used only for 32 or 64-bit wide data.
  - **Memory Mapped Data Width** – Specifies the data width in bits of the AXI4 Read bus. Valid values are 32, 64, 128, 256, 512, and 1,024. Depending on the Channel Type, these options vary.

- **Stream Data Width** – Specifies the data width in bits of the MM2S Stream bus. Valid values are 8, 16, 32, 64, 128, 256, 512, and 1,024. This value cannot be more than the Memory Mapped Data Width.
- **Maximum Burst Size** – This option specifies the maximum size of the burst cycles on the AXI MM2S Memory Map Read interface. In other words, this setting specifies the granularity of burst partitioning. For example, if the burst length is set to 16, the maximum burst on the memory map interface is 16 data beats. Smaller values reduce throughput but result in less impact on the AXI infrastructure. Larger values increase throughput but result in a greater impact on the AXI infrastructure. Valid values are 2, 4, 8, 16, 32, 64, 128, and 256 based on the data width that is selected.
- **Bytes To Transfer (BTT) Bit Used** – Specifies the valid number of bits in the number of BTT field of the MM2S command. Valid options are 8 to 23.

- **S2MM Channel Options** – This box allows you to configure the S2MM channel options.
  - **Channel Type** – You can choose a Full, Basic, or Omit. Selecting Full allows the S2MM channel to be configured for all possible combination and advance features. The Basic mode restricts some of features and allows the S2MM to be used only for 32 or 64-bit wide data. The Omit mode completely disables the channel.
  - **Memory Mapped Data Width** – Specifies the data width in bits of the AXI4 Write bus. Valid values are 32, 64, 128, 256, 512, and 1,024. The choices vary depending on the Channel Type chosen.

    Note: When the IP is used in the IP integrator, the Memory Mapped Data Width parameter is auto set based on the data width of the Streaming interface. You can change this value by switching it to "Manual" mode.
  - **Stream Data Width** – Specifies the data width in bits of the S2MM Stream bus. Valid values are 8, 16, 32, 64, 128, 256, 512, and 1,024. This value cannot be more than the Memory Mapped Data Width.

    Note: When the IP is used in the Vivado IP integrator, the Stream Data Width parameter is automatically set based on the connection made.
  - **Maximum Burst Size** – This option specifies the maximum size of the burst cycles on the AXI S2MM Memory Map Read interface. In other words, this setting specifies the granularity of burst partitioning. For example, if the burst length is set to 16, the maximum burst on the memory map interface is 16 data beats. Smaller values reduce throughput but result in less impact on the AXI infrastructure. Larger values increase throughput but result in a greater impact on the AXI infrastructure. Valid values are 2, 4, 8, 16, 32, 64, 128, and 256 based on the data width that is selected.
  - **Bytes To Transfer (BTT) Bit Used** – Specifies the valid number of bits in the number of the BTT field of the S2mm command. Valid options are 8 to 23.
- **Enable xCache and xUser** – Select this option if you wish to change the *cache and *user signals of the AXI4 interface.

- **Enable MM2S Control Signals, Enable S2MM Control Signals** - Enabling this exposes all the control and status signals of the MM2S, S2MM interface. When enabled, you need to connect the signals carefully. By default, these are not exposed and tied to default states.
- **Address Width (32 -64)** Specify the width of the address space. It can be any value between 32 and 64.

Advanced Options

The following describes the advanced options of the MM2S and S2MM channels of the AXI DataMover core.

- **MM2S Channel Options** – This box allows you to configure the advance options of the MM2S channel options.

The following options are only available when the channel is configured in Full mode.

- **Enable Asynchronous Clocks** – This setting allows you to operate the MM2S Command and Status Stream interface asynchronously with the MM2S Memory Map interface.

  Note: The Enable Asynchronous Clocks parameter is automatically set when the IP is used in the IP integrator.
- **Allow Unaligned Transfers** – This setting enables or disables the MM2S Data Realignment Engine (DRE). When checked, the DRE is enabled and allows data realignment to the byte (8 bits) level on the MM2S Memory Map datapath. For the MM2S channel, data is read from the memory. If the DRE is enabled, data reads can start from any Buffer Address offset, and the read data is aligned such that the first byte read is the first valid byte out on the AXI4-Stream. What is considered aligned or unaligned is based on the Memory Map data width**.**
- **Enable Store and Forward** – This setting provides the inclusion/omission of the MM2S Store and Forward function. This option is available in Full mode only. Further, this option is always enabled when the memory-map data width is different than the streaming data width.
- **ID width** – This value sets the width of the ID ports. Setting this to 0 disables the ID ports.
- **ID Value** – This is the value that is put on the ID ports.

The following is a list of S2MM Channel parameters.

- **S2MM Channel Options** – This box allows you to configure the advance options of the S2MM channel options.

The following options are only available when the channel is configured in Full mode.

- **Enable Asynchronous Clocks** – This setting allows you to operate the S2MM Command and Status Stream interface asynchronously with S2MM Memory Map interface.

  Note: The Enable Asynchronous Clock parameter is automatically set when the IP is used in the IP integrator.

- **Allow Unaligned Transfers** – This option enables or disables the S2MM Data Realignment Engine (DRE). When checked, the DRE is enabled and allows data realignment to the byte (8 bits) level on the S2MM Memory Map datapath. For the S2MM channel, data is written to the memory. If the DRE is enabled, data writes can start from any Buffer Address offset, and the read data is aligned such that the first byte read is the first valid byte out on the AXI4-Stream. What is considered aligned or unaligned is based on the Memory Map data width**.**
- **Enable Indeterminate BTT Mode** – This setting provides the Indeterminate BTT mode. This is needed when the number of bytes to be received on the input S2MM Stream Channel is unknown at the time the transfer command is posted to the DataMover S2MM command input. When enabled, the Store and Forward option is not available for use.
- **Enable Store and Forward** – This setting provides the inclusion/omission of the S2MM Store and Forward function. This option is available in Full mode only. Indeterminate-BTT mode takes care of the Store and Forward, as such the Store and Forward option is not available when I-BTT feature is enabled.
- **ID width** – This value sets the width of the ID ports. Setting this to 0 disables the ID ports.
- **ID Value** – This is the value that is put on the ID ports.

User Parameters

Table 4-1 shows the relationship between the Vivado IDE fields in the Vivado Design Suite (described in Customizing and Generating the Core) and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter	User Parameter	Default Value
Enable MM2S	c_enable_mm2s	1
Channel type	c_include_mm2s	Full
Memory Map Data Width	c_m_axi_mm2s_data_width	32
Stream data Width	c_m_axis_mm2s_tdata_width	32
Max burst Size	c_mm2s_burst_size	16
Width of BTT field	c_mm2s_btt_used	16
Enable S2MM	c_enable_s2mm	1
Channel type	c_include_s2mm	Full
Memory Map Data Width	c_m_axi_s2mm_data_width	32
Stream data Width	c_s_axis_s2mm_tdata_width	32
Max burst Size	c_s2mm_burst_size	16
Width of BTT field	c_s2mm_btt_used	16
Enable xCACHE xUSER	c_enable_cache_user	FALSE
Enable MM2S Control Signal	c_enable_mm2s_adv_sig	0
Enable S2MM Control Signal	c_enable_s2mm_adv_sig	0
Enable Async Clocks	c_mm2s_stscmd_is_async	FALSE
Allow Unaligned Transfer	c_include_mm2s_dre	FALSE
Enable Store and Forward	c_mm2s_include_sf	TRUE
ID Width	c_m_axi_mm2s_id_width	4
ID Value	c_m_axi_mm2s_arid	0
Enable Async Clocks	c_s2mm_stscmd_is_async	FALSE
Allow Unaligned Transfer	c_include_s2mm_dre	FALSE
Enable Store and Forward	c_s2mm_include_sf	TRUE
Enable Indeterminate BTT Mode	c_s2mm_support_indet_btt	FALSE
ID Width	c_m_axi_s2mm_id_width	4
ID Value	c_m_axi_s2mm_awid	0
Address Width (32-64)	c_addr_width	32

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1].

Constraining the Core

Required Constraints

The necessary XDC constraints are delivered when the IP is generated.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

The `m_axis_mm2s_cmdsts_aclk` should be less than or equal to `m_axi_mm2s_aclk`. Similarly, `m_axis_s2mm_cmdsts_awclk` should be less than or equal to `m_axi_s2mm_aclk`.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5].

IMPORTANT: *For cores targeting 7 series or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about core synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 1].

For information about implementing the example design, see Implementing the Example Design in Chapter 5.

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite.

The top module instantiates all components of the core and example design that are needed to implement the design in hardware, as shown in Figure 5-1. This includes clock generator, register configuration, data generator, and data checker modules.

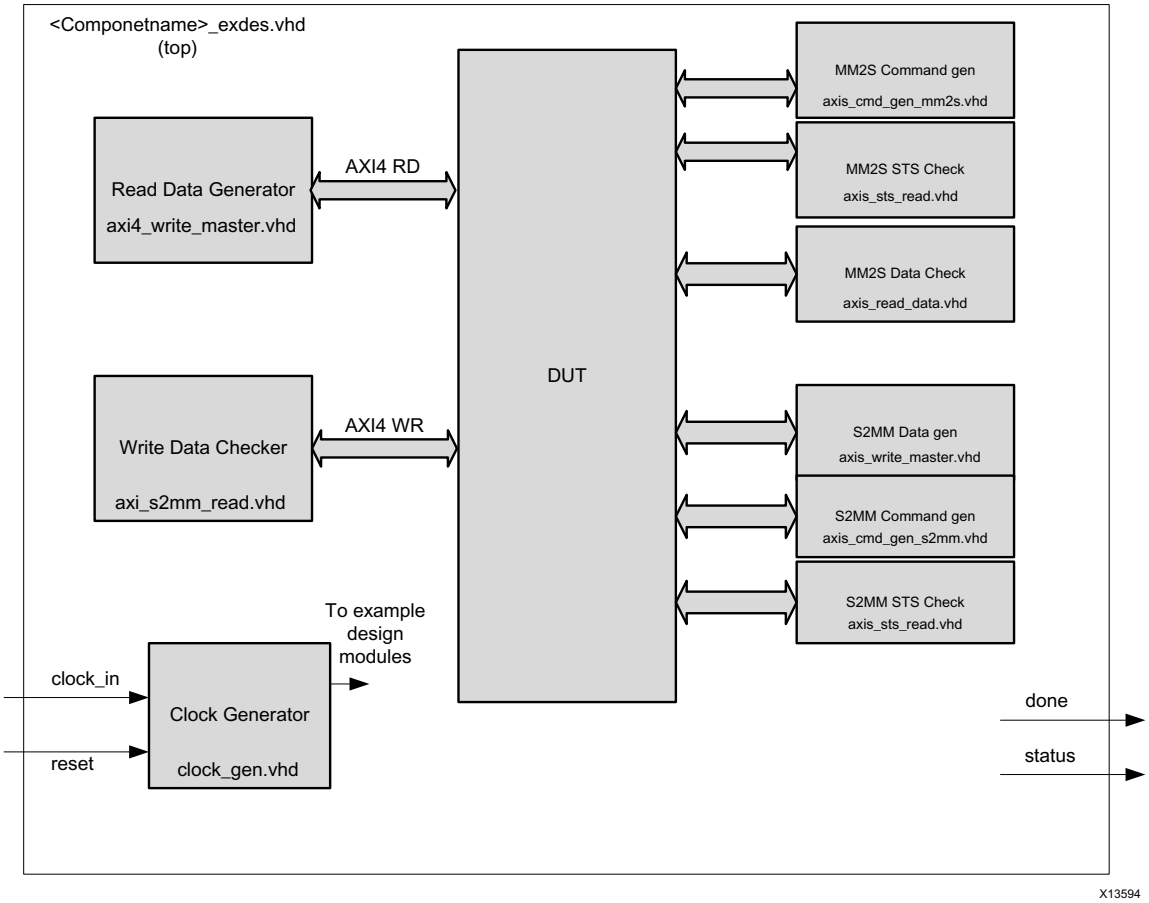

Figure 5-1: **Block Diagram of Example Design**

This example design demonstrates transactions on the AXI4 and AXI4-Stream interfaces of the DUT.

Clock generator: The Clocking Wizard is used to generate the clocks for the example design. When DUT is in synchronous mode, The Clocking Wizard generates a 100 MHz clock for all the AXI interfaces in the example design. When DUT is in asynchronous mode, a 50 MHz clock for the command interface and a 100 MHz clock for the AXI4 and AXI4-Stream interface are generated by the Clocking Wizard. DUT and other modules of the example design are kept under reset until MMCME2 is locked.

Read Data generator: This uses an AXI block RAM which is filled (with a fixed amount of transfers) after MMCM is locked. MM2S channel reads this AXI block RAM and transfers data to the AXI4-Stream interface.

MM2S Command Generator: This module generates one MM2S command.

MM2S STS Checker: This module checks the status that is received from the MM2S channel.

Read Data checker: This module checks the data transferred on the MM2S AXI4-Stream interface.

Write path generator: When the Write (S2MM) channel is configured, this module drives the transactions (with a fixed amount of transfers) on the S2MM AXI4-Stream interface.

S2MM Command Generator: This module generates one S2MM command.

S2MM STS Checker: This module checks the status that is received from the S2MM channel.

Write path checker: This module checks the data received on the AXI4 interface. Data received on the AXI4 interface is also written into another AXI block RAM.

The test starts soon after the MMCM is locked. The 'Done' pin is asserted after the test is completed. If the data transfer is successful then the 'Status' pin is asserted. These two pins can be connected to an LED to know the status of the test.

Implementing the Example Design

After following the steps described in Chapter 4, Design Flow Steps, implement the example design as follows:

1. Right-click the core in the Hierarchy window, and select **Open IP Example Design**.
2. A new window pops up, asking you to specify a directory for the example design. Select a new directory or keep the default directory.

A new project is automatically created in the selected directory and it is opened in a new Vivado Integrated Design Environment (IDE) window.

3. In the Flow Navigator (left-side pane), click **Run Implementation** and follow the directions.

Example Design Directory Structure

In the current project directory, a new project with the name `<component_name>_example` is created and the files are delivered in the `<component_name>_example/<component_name>_example.srcs/` directory. This directory and its subdirectories contain all the source files that are required to create the AXI DataMover controller example design.

Table 5-1 shows the files delivered as part of example design.

Table 5-1: Example Design Directory

Name	Description
<component_name>_exdes.vhd	Top-level HDL file for the example design.
clock_gen.vhd	Clock generation module for example design.
axi4_write_master.vhd	Read path data generator module for example design.
axis_data_read.vhd	Read path data checker module for example design.
axis_write_master.vhd	Write path data generator module for example design.
axi_s2mm_read.vhd	Write path data checker module for example design.
axis_cmd_gen_mm2s.vhd	MM2S Command generator
axis_cmd_gen_s2mm.vhd	S2MM Command generator
axis_sts_read.vhd	STS data checker

Table 5-2 shows the files delivered as part of the test bench.

Table 5-2: **Simulation Directory**

Name	Description
<component_name>_exdes_tb.vhd	Test Bench for the example design

Table 5-3 shows the XDC file delivered as part of example design.

Table 5-3: **Constraints Directory**

Name	Description
<component_name>_exdes.xdc	Top level constraints file for the example design.

The XDC delivered with the example design has I/O constraints configured for the KC705 board. These constraints are commented by default. Uncomment the constraints before implementing the example design on KC705 board.

Simulating the Example Design

Using the AXI DataMover example design (delivered as part of the AXI DataMover), you can quickly simulate and observe the behavior of the AXI DataMover.

Simulation Results

The simulation script compiles the AXI DataMover example design and supporting simulation files. It then runs the simulation and checks to ensure that it completed successfully.

If the test fails, the following message displays: `Test Failed!!!`

If the test passes, the following message displays: `Test Completed Successfully`

If the test hangs, the following message displays: `Test Failed!! Test Timed Out`

Test Bench for the Example Design

This section contains information about the provided test bench in the Vivado Design Suite.

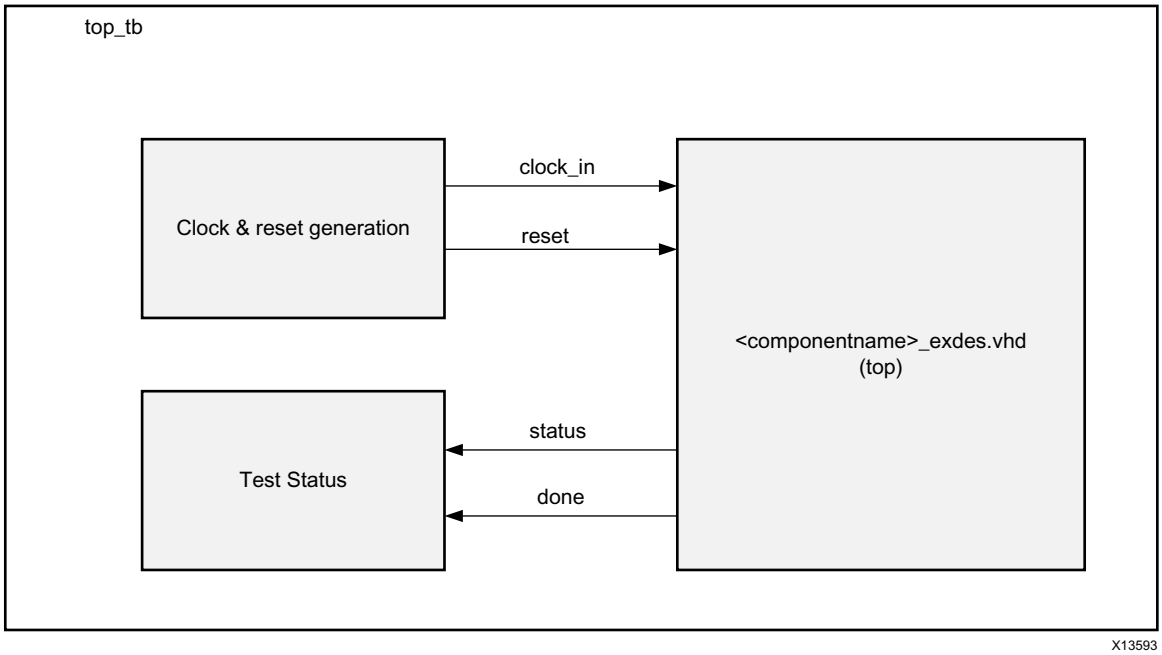

Figure 5-2: **AXI DataMover Example Design Test Bench**

Figure 5-2 shows test bench for the AXI DataMover example design. The top-level test bench generates a 200 MHz differential clock and drives an initial reset to the example design.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information on migrating from ISE tools to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 6].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite. There are no parameter or port changes for this core.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the AXI DataMover core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the AXI DataMover core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the **AXI DataMover Core**

AR: 47651.

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Vivado Design Suite Debug Feature

There are many tools available to address AXI DataMover core design issues. It is important to know which tools are useful for debugging various situations.

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado Integrated Design Environment (IDE) is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 7].

Hardware Debug

- What value should be driven on `mm2s_allow_addr_req` and `s2mm_allow_addr_req` if you do not want to use these signals?

  Answer: `mm2s_allow_addr_req` and `s2mm_allow_addr_req` should be tied to '1' if you do not want to control it. There are example timing diagrams (Figure 2-8 and Figure 2-9) to provide more clarity on usage of these signals. By default, these pins are not exposed and are tied to '1'.

- When `mm2s_halt` is asserted, `m_axis_mm2s_tvalid` is asserted sometimes.

  Answer: When AXI DataMover goes through soft shutdown, it flushes internal FIFOs, thus you will find some residual data appearing on streaming side.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

To search for Xilinx documentation, go to www.xilinx.com/support

Unless otherwise noted, IP references are for the product documentation page.

These documents provide supplemental material useful with this product guide:

1. *Vivado Design Suite User Guide: Designing with IP* (UG896)
2. *Vivado Design Suite: Vivado AXI Reference Guide* (UG1037)
3. *Vivado Design Suite User Guide: Getting Started* (UG910)
4. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide - Logic Simulation* (UG900)
6. *ISE to Vivado Design Suite Migration Guide* (UG911)
7. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
8. *Synthesis and Simulation Design Guide* (UG626)
9. *AXI Interconnect LogiCORE IP Product Guide* (PG059)
10. *AMBA AXI4-Stream Protocol Specification* (ARM IHI 0051A)
11. *ARM AXI4 Memory Mapped Specification*
12. *ARM AXI4-Stream Interface Specification* (ARM DUI 0534B)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	5.1	Added support for UltraScale+ families.
04/01/2015	5.1	Added support for 64-bit addressing.
11/19/2014	5.1	Revised the description of TKEEP on page 33.
10/01/2014	5.1	• Remove non-incrementing Burst transfers from unsupported features. • Minor changes to descriptions in Table 2-5. • Added GUI Parameter to User Parameter Relationship table
04/02/2014	5.1	• Updated the Clock Frequencies section. • Many other minor updates.
12/18/2013	5.1	Added UltraScale™ architecture support.
10/02/2013	5.1	• Modified demonstration test bench path in Output Generation. • Added example design. • Updated screen displays. • Updated performance numbers in Table 2-1. • Updated Figure 8-2. • Added IP integrator information to Chapter 4.
03/20/2013	5.0	• Revision number advanced to 5.0 to align with core version number. • Updated for Vivado design tools and core version 5.0. • Removed all ISE, Virtex®-6, and Spartan®-6 material. • Removed Design Parameters section in Chapter 3. • Added Type field and Maximum Burst Size option. • Updated screen captures in Chapter 4. • Updated many of the I/O signals.
12/18/2012	2.1	• Updated for Vivado 2012.4 and ISE v14.4 design tools. • Updated Debugging appendix. Updated core version. • Replaced Figure 1-1 with two new figures. • Updated max frequency numbers and devices. • Removed many rows from resource utilization tables. • Removed Allowable Parameter Combinations section. • Updated screen captures. • Updated output hierarchies.
10/16/2012	2.0.1	• Updated for Vivado 2012.3 and ISE v14.3. • Added MM2S and S2MM block Information • Added two figures showing typical use cases for DataMover • Removed AXI Read Master, AXI Write Master sections, AXI DataMover Operation, and Parameter -- I/O Signal Dependencies sections • Added two new sections to Chapter 3: Example DataMover Read(MM2S) Timing Example DataMover Write(S2MM) Timing

Date	Version	Revision
07/25/2012	2.0	Updated for Vivado 2012.2, Zynq® features, and ISE v14.2 Added Vivado content in Customizing and Generating the Core
07/11/2012	1.1	Template update.
10/19/2011	1.0	Initial Xilinx release.

Please Read: Important Legal Notices