

# LogiCORE IP AXI DataMover v3.00a

## *Product Guide*

PG022 October 16, 2012

# Table of Contents

## SECTION I: SUMMARY

### IP Facts

#### Chapter 1: Overview

Operating System Requirements .....	7
Feature Summary .....	8
Applications .....	9
Unsupported Features .....	9
Licensing and Ordering Information .....	10

#### Chapter 2: Product Specification

Performance .....	11
Resource Utilization .....	13
Port Descriptions .....	18

#### Chapter 3: Designing with the Core

General Design Guidelines .....	27
Clocking .....	40
Resets .....	40
Design Parameters .....	45
Allowable Parameter Combinations .....	51

## SECTION II: VIVADO DESIGN SUITE

### Chapter 4: Customizing and Generating the Core

GUI .....	55
Core Implementation.....	61
Output Generation.....	62

### Chapter 5: Constraining the Core

## SECTION III: ISE DESIGN SUITE

### Chapter 6: Customizing and Generating the Core

GUI .....	66
Core Implementation.....	71
Output Generation.....	71

### Chapter 7: Constraining the Core

## SECTION IV: APPENDICES

### Appendix A: Migrating

### Appendix B: Debugging

### Appendix C: Additional Resources

Xilinx Resources .....	78
References .....	78
Technical Support .....	79
Revision History .....	79
Notice of Disclaimer.....	80

# SECTION I: SUMMARY

IP Facts

Overview

Product Specification

Designing with the Core

## Introduction

The Advanced eXtensible Interface (AXI) DataMover is a soft Xilinx LogiCORE™ Intellectual Property (IP) core used as a building block for Scalable Direct Memory Access (DMA) functions. It provides the basic AXI4 Memory Map Read to AXI4-Stream and AXI4-Stream to AXI4 Memory Map Write data transport and protocol conversion. The function is intended to be a standalone core for custom designs or a helper core to higher level DMA type functions.

## Features

- AXI4 Compliant
- Primary AXI4 Memory Map data width support of 32, 64, 128, 256, 512, and 1024 bits
- Primary AXI4-Stream data width support of 8, 16, 32, 64, 128, 256, 512 and 1024 bits (Must be less than or equal to Memory Mapped data width)
- Parameterized Memory Map Burst Lengths of 16, 32, 64, 128, and 256 data beats
- Extended address width support up to 64 bits
- Optional Data Realignment Engine (DRE)
- 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	
<b>Core Specifics</b>	
Supported Device Family <sup>(1)</sup>	Zynq™-7000 <sup>(2)</sup> , Virtex®-7, Kintex™-7, Artix™-7, Virtex-6, Spartan®-6
Supported User Interfaces	AXI4, AXI4-Stream
Resources	See <a href="#">Table 2-4</a> through <a href="#">Table 2-7</a> .
<b>Provided with Core</b>	
Design Files	ISE®: VHDL Vivado™: VHDL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Not Provided
Simulation Model	Not Provided
Supported S/W Driver	N/A
<b>Tested Design Flows<sup>(3)</sup></b>	
Design Entry	ISE Design Suite 14.3 Vivado Design Suite 2012.3 <sup>(4)</sup>
Simulation	QuestaSim-64
Synthesis	Xilinx Synthesis Technology (XST) Vivado Synthesis
<b>Support</b>	
Provided by Xilinx @ <a href="http://www.xilinx.com/support">www.xilinx.com/support</a>	

### Notes:

1. For a complete list of supported derivative devices, see [Embedded Edition Derivative Device Support](#).
2. Supported in ISE Design Suite implementations only.
3. For the supported versions of the tools, see the [Xilinx Design Tools: Release Notes Guide](#).
4. Supports only 7 series devices.

# 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 for the Xilinx AXI DMA core, and enables 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.

Figure 1-1 shows a block diagram of the AXI DataMover core. There are two sub blocks:

- **MM2S Block:** This block handles transactions from the AXI memory map to 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 AXI memory map 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 Block:** This block handles transactions from the AXI4-Stream to AXI memory map 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 AXI memory map 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).

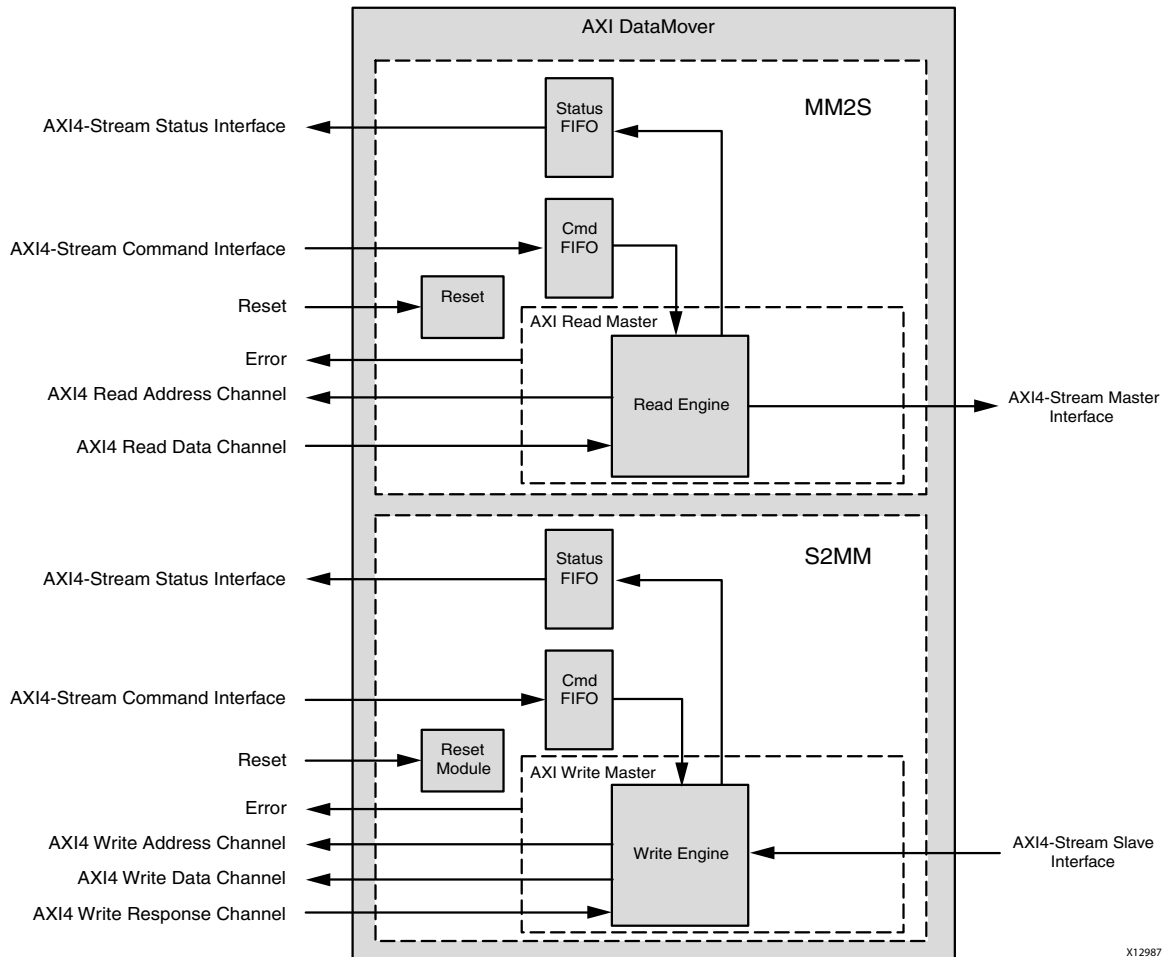


Figure 1-1: AXI DataMover Block Diagram

X12987

## Operating System Requirements

For operating system requirements, see the [Xilinx Design Tools: Release Notes Guide](#).

---

## Feature Summary

### AXI4 Compliant

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

### AXI4 Memory Map Data Width

The AXI DataMover core supports the primary AXI4 Memory Map data bus width of 32, 64, 128, 256, 512, and 1024 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 1024 bits. The AXI4-Stream data width must be less than or equal to the AXI4 Memory Map data width for the respective channel.

### Extended Address Width

The AXI DataMover core supports the extended address width support up to 64 bits.

### 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 16, 32, 64, 128, and 256.

### Unaligned Transfers

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

### 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. When the Store and Forward feature is enabled, a downsizer/upsizer function is automatically inserted on the Stream side if the Stream Channel data width is less than the Memory Mapped data width. When the Store and Forward feature is not enabled, narrow transfers are generated on the AXI4 Memory Map side if the Stream Channel data width is less than the Memory Mapped data width.

## 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 or a helper core to higher-level DMA type functions.

---

## Unsupported Features

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

- User signals
- Locking transfers
- Caching transfers
- Non-incrementing and circular Burst transfers

---

## Licensing and Ordering Information

This Xilinx LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado™ Design Suite and ISE® Design Suite Embedded Edition tools 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

### Maximum Frequencies

The targeted maximum clock frequency for AXI DataMover core are given in [Table 2-1](#).

**Table 2-1: Maximum Clock Frequency**

Family	Speed Grade	F <sub>Max</sub>
Spartan-6	-2	133 MHz
Virtex-6	-1	180 MHz
Virtex-7	-1	180 MHz

### Latency

[Table 2-2](#) describes the latency for the AXI DataMover core. Latency is measured in simulation and indicates AXI DataMover core latency cycles only and does not include system dependent latency or throttling.

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

Description	Clocks
<b>MM2S Channel</b>	
Initial m_axi_mm2s_rvalid to m_axis_mm2s_tvalid (C_MM2S_INCLUDE_SF = 0)	1
Initial m_axi_mm2s_rvalid to m_axis_mm2s_tvalid (C_MM2S_INCLUDE_SF = 1)	3
AXI4-Stream packet to packet latency (C_INCLUDE_MM2S_DRE = 0) m_axis_mm2s_tlast to m_axis_mm2s_tvalid	2
AXI4-Stream packet to packet latency (C_INCLUDE_MM2S_DRE = 1) m_axis_mm2s_tlast to m_axis_mm2s_tvalid	3
s_axis_mm2s_cmd_tvalid to m_axi_mm2s_arvalid	8
<b>S2MM Channel</b>	
Initial s_axis_s2mm_tvalid m_axi_s2mm_avalid (C_S2MM_INCLUDE_SF = 0)	2

Table 2-2: AXI DataMover Latency (Cont'd)

Description	Clocks
Initial s_axis_s2mm_tvalid m_axi_s2mm_avalid (C_S2MM_INCLUDE_SF = 1, C_S2MM_BURST_SIZE = 16)	20
AXI4-Stream packet to packet latency (C_INCLUDE_S2MM_DRE = 0) s_axis_s2mm_tlast to s_axis_s2mm_tready	2
AXI4-Stream packet to packet latency (C_INCLUDE_S2MM_DRE = 1) s_axis_s2mm_tlast to s_axis_s2mm_tready	3

## Throughput

Table 2-3 describes the latency for the AXI DataMover core. The tables provides performance information for a typical configuration. Throughput test consisted of eight parent commands loaded into the AXI DataMover core with each command having 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:

- `C_M_AXI_MM2S_DATA_WIDTH = 32` and `C_M_AXI_S2MM_DATA_WIDTH = 32`
- `C_M_AXIS_MM2S_TDATA_WIDTH = 32` and `C_S_AXIS_MM2S_TDATA_WIDTH = 32`
- `C_MM2S_STSCMD_IS_ASYNC = 0` and `C_S2MM_STSCMD_IS_ASYNC = 0`
- `C_INCLUDE_MM2S_DRE = 0` and `C_INCLUDE_S2MM_DRE = 0`
- `C_MM2S_INCLUDE_SF = 1` and `C_S2MM_INCLUDE_SF = 1`

Table 2-3: AXI DataMover Throughput

AXI DataMover Channel	Primary Clock Frequency	Packet Size	Maximum Total Data Throughput (MB/sec)	Percent of Theoretical
<b>Spartan-6 FPGAs</b>				
MM2S	100	1 MB	391.27	97.75%
S2MM	100	1 MB	391.27	97.75%
<b>Virtex-6 FPGAs</b>				
MM2S	150	1 MB	587.81	97.96%
S2MM	150	1 MB	587.81	97.96%

# Resource Utilization

Resources required for the AXI DataMover core have been estimated for the Virtex®-7 Field Programmable Gate Array (FPGA) in Table 2-4, Kintex™-7 and Artix™-7 FPGAs and Zynq™-7000 device in Table 2-5, Virtex-6 FPGA in Table 2-6, and Spartan®-6 FPGA in Table 2-7. These values were generated using Xilinx CORE Generator™ tools, v14.3. They are derived from post-synthesis reports, and might change during MAP and PAR.

Table 2-4: Virtex-7 FPGA Resource Estimates

C_INCLUDE_MM2S	C_M_AXI_MM2S_ADDR_WIDTHS	C_M_AXI_MM2S_DATA_WIDTH	C_M_AXIS_MM2S_TDATA_WIDTH	C_INCLUDE_MM2S_DRE	C_MM2S_BURST_SIZE	C_INCLUDE_MM2S_SF	C_INCLUDE_S2MM	C_M_AXI_S2MM_ADDR_WIDTHS	C_M_AXI_S2MM_DATA_WIDTH	C_S_AXIS_S2MM_TDATA_WIDTH	C_INCLUDE_S2MM_DRE	C_S2MM_BURST_SIZE	C_S2MM_SUPPORT_INDET_BTT	C_INCLUDE_MM2S_SF	Slices	Slice Reg	Slice LUTs	Block RAM
1	32	32	32	1	16	1	0	32	32	32	1	16	0	1	209	573	623	1
0	32	32	32	1	16	1	1	32	32	32	1	16	0	1	299	804	870	1
1	32	32	32	1	16	1	1	32	32	32	1	16	0	1	667	1377	1448	1
1	32	64	32	1	16	1	1	32	64	32	1	16	1	1	828	1821	1762	4
1	32	128	32	1	16	1	1	32	128	32	1	16	1	1	924	2255	2119	4
1	32	256	32	1	16	1	1	32	256	32	1	16	1	1	1151	3112	2695	8
1	32	512	32	1	16	1	1	32	512	32	1	16	1	1	1521	4809	3924	16
1	32	1024	32	1	16	1	1	32	1024	32	1	16	1	1	2317	8062	5794	34
1	32	32	8	1	16	1	1	32	32	8	1	16	1	1	582	1197	1354	2
1	32	32	16	1	16	1	1	32	32	16	1	16	1	1	509	1366	1496	2
1	32	64	64	1	16	1	1	32	64	64	1	16	1	1	982	2196	2168	2
1	32	128	128	0	16	1	1	32	128	128	0	16	1	1	823	2508	2081	4
1	32	256	256	0	16	1	1	32	256	256	0	16	1	1	1282	4078	2794	8
1	32	512	512	0	16	1	1	32	512	512	0	16	1	1	2090	7205	4388	16
1	32	1024	1024	0	16	1	1	32	1024	1024	0	16	1	1	3359	13323	7986	34
1	32	32	32	1	16	0	1	32	32	32	1	16	0	0	399	1266	1314	0
1	32	32	32	1	32	1	1	32	32	32	1	32	1	1	790	1576	1539	2
1	32	32	32	1	64	1	1	32	32	32	1	64	1	1	793	1596	1587	2
1	32	32	32	1	128	1	1	32	32	32	1	128	1	1	700	1616	1674	2

Table 2-4: Virtex-7 FPGA Resource Estimates (Cont'd)

C_INCLUDE_MM2S	C_M_AXI_MM2S_ADDR_WIDTHS	C_M_AXI_MM2S_DATA_WIDTH	C_M_AXIS_MM2S_TDATA_WIDTH	C_INCLUDE_MM2S_DRE	C_MM2S_BURST_SIZE	C_INCLUDE_MM2S_SF	C_INCLUDE_S2MM	C_M_AXI_S2MM_ADDR_WIDTHS	C_M_AXI_S2MM_DATA_WIDTH	C_S_AXIS_S2MM_TDATA_WIDTH	C_INCLUDE_S2MM_DRE	C_S2MM_BURST_SIZE	C_S2MM_SUPPORT_INDET_BTT	C_INCLUDE_MM2S_SF	Slices	Slice Reg	Slice LUTs	Block RAM
1	32	32	32	1	256	1	1	32	32	32	1	256	1	1	747	1634	1667	4
1	32	32	32	0	16	1	1	32	32	32	0	16	1	1	606	1313	1346	2
1	64	32	32	1	16	1	1	64	32	32	1	16	1	1	620	1816	1841	2
1	64	1024	1024	1	256	1	1	64	1024	1024	1	256	1	1	3541	13605	7956	34
2	32	32	32	0	16	0	1	32	32	32	0	16	0	0	192	730	561	0
2	32	64	32	0	32	0	1	32	64	32	0	32	0	0	188	745	593	0
2	32	64	64	0	64	0	1	32	64	64	0	64	0	0	500	976	606	0

Table 2-5: Kintex-7 FPGA and Zynq-7000 Device Resource Estimates

C_INCLUDE_MM2S	C_M_AXI_MM2S_ADDR_WIDTHS	C_M_AXI_MM2S_DATA_WIDTH	C_M_AXIS_MM2S_TDATA_WIDTH	C_INCLUDE_MM2S_DRE	C_MM2S_BURST_SIZE	C_INCLUDE_MM2S_SF	C_INCLUDE_S2MM	C_M_AXI_S2MM_ADDR_WIDTHS	C_M_AXI_S2MM_DATA_WIDTH	C_S_AXIS_S2MM_TDATA_WIDTH	C_INCLUDE_S2MM_DRE	C_S2MM_BURST_SIZE	C_S2MM_SUPPORT_INDET_BTT	C_INCLUDE_MM2S_SF	Slices	Slice Reg	Slice LUTs	Block RAM
1	32	32	32	1	16	1	0	32	32	32	1	16	0	1	267	573	618	1
0	32	32	32	1	16	1	1	32	32	32	1	16	0	1	365	804	867	1
1	32	32	32	1	16	1	1	32	32	32	1	16	0	1	698	1377	1446	1
1	32	64	32	1	16	1	1	32	64	32	1	16	1	1	803	1821	1780	4
1	32	128	32	1	16	1	1	32	128	32	1	16	1	1	965	2255	2077	4
1	32	256	32	1	16	1	1	32	256	32	1	16	1	1	1161	3112	2677	8
1	32	512	32	1	16	1	1	32	512	32	1	16	1	1	1551	4809	3931	16
1	32	1024	32	1	16	1	1	32	1024	32	1	16	1	1	2265	8062	5888	34

Table 2-5: Kintex-7 FPGA and Zynq-7000 Device Resource Estimates (Cont'd)

C_INCLUDE_MM2S	C_M_AXI_MM2S_ADDR_WIDTHS	C_M_AXI_MM2S_DATA_WIDTH	C_M_AXIS_MM2S_TDATA_WIDTH	C_INCLUDE_MM2S_DRE	C_MM2S_BURST_SIZE	C_INCLUDE_MM2S_SF	C_INCLUDE_S2MM	C_M_AXI_S2MM_ADDR_WIDTHS	C_M_AXI_S2MM_DATA_WIDTH	C_S_AXIS_S2MM_TDATA_WIDTH	C_INCLUDE_S2MM_DRE	C_S2MM_BURST_SIZE	C_S2MM_SUPPORT_INDET_BTT	C_INCLUDE_MM2S_SF	Slices	Slice Reg	Slice LUTs	Block RAM
1	32	32	8	1	16	1	1	32	32	8	1	16	1	1	602	1197	1349	2
1	32	32	16	1	16	1	1	32	32	16	1	16	1	1	635	1366	1462	2
1	32	64	64	1	16	1	1	32	64	64	1	16	1	1	938	2196	2187	2
1	32	128	128	0	16	1	1	32	128	128	0	16	1	1	908	2508	2017	4
1	32	256	256	0	16	1	1	32	256	256	0	16	1	1	1293	4078	2739	8
1	32	512	512	0	16	1	1	32	512	512	0	16	1	1	1925	7205	4597	16
1	32	1024	1024	0	16	1	1	32	1024	1024	0	16	1	1	3537	13323	7750	34
1	32	32	32	1	16	0	1	32	32	32	1	16	0	0	408	1266	1339	0
1	32	32	32	1	32	1	1	32	32	32	1	32	1	1	743	1576	1563	2
1	32	32	32	1	64	1	1	32	32	32	1	64	1	1	742	1596	1600	2
1	32	32	32	1	128	1	1	32	32	32	1	128	1	1	688	1616	1699	2
1	32	32	32	1	256	1	1	32	32	32	1	256	1	1	615	1634	1713	4
1	32	32	32	0	16	1	1	32	32	32	0	16	1	1	652	1313	1347	2
1	64	32	32	1	16	1	1	64	32	32	1	16	1	1	857	1816	1741	2
1	64	1024	1024	1	256	1	1	64	1024	1024	1	256	1	1	3390	13605	8061	34
2	32	32	32	0	16	0	1	32	32	32	0	16	0	0	372	730	555	0
2	32	64	32	0	32	0	1	32	64	32	0	32	0	0	184	745	593	0
2	32	64	64	0	64	0	1	32	64	64	0	64	0	0	539	976	597	0

Table 2-6: Virtex-6 FPGA Resource Estimates

C_INCLUDE_MM2S	C_M_AXI_MM2S_ADDR_WIDTHS	C_M_AXI_MM2S_DATA_WIDTH	C_M_AXIS_MM2S_TDATA_WIDTH	C_INCLUDE_MM2S_DRE	C_MM2S_BURST_SIZE	C_INCLUDE_MM2S_SF	C_INCLUDE_S2MM	C_M_AXI_S2MM_ADDR_WIDTHS	C_M_AXI_S2MM_DATA_WIDTH	C_S_AXIS_S2MM_TDATA_WIDTH	C_INCLUDE_S2MM_DRE	C_S2MM_BURST_SIZE	C_S2MM_SUPPORT_INDET_BTT	C_INCLUDE_MM2S_SF	Slices	Slice Reg	Slice LUTs	Block RAM
1	32	32	32	1	16	1	0	32	32	32	1	16	0	1	290	572	600	1
0	32	32	32	1	16	1	1	32	32	32	1	16	0	1	300	804	878	1
1	32	32	32	1	16	1	1	32	32	32	1	16	0	1	595	1377	1466	1
1	32	64	32	1	16	1	1	32	64	32	1	16	1	1	787	1821	1795	4
1	32	128	32	1	16	1	1	32	128	32	1	16	1	1	959	2255	2068	4
1	32	256	32	1	16	1	1	32	256	32	1	16	1	1	1248	3111	2648	8
1	32	512	32	1	16	1	1	32	512	32	1	16	1	1	1558	4809	3876	16
1	32	1024	32	1	16	1	1	32	1024	32	1	16	1	1	2167	8075	5789	34
1	32	32	8	1	16	1	1	32	32	8	1	16	1	1	617	1197	1368	2
1	32	32	16	1	16	1	1	32	32	16	1	16	1	1	550	1366	1488	2
1	32	64	64	1	16	1	1	32	64	64	1	16	1	1	1065	2196	2110	2
1	32	128	128	0	16	1	1	32	128	128	0	16	1	1	1111	2508	1922	4
1	32	256	256	0	16	1	1	32	256	256	0	16	1	1	1334	4078	2735	8
1	32	512	512	0	16	1	1	32	512	512	0	16	1	1	1921	7205	4547	16
1	32	1024	1024	0	16	1	1	32	1024	1024	0	16	1	1	3264	13337	8046	34
1	32	32	32	1	16	0	1	32	32	32	1	16	0	0	444	1266	1312	0
1	32	32	32	1	32	1	1	32	32	32	1	32	1	1	645	1576	1596	2
1	32	32	32	1	64	1	1	32	32	32	1	64	1	1	519	1596	1682	2
1	32	32	32	1	128	1	1	32	32	32	1	128	1	1	822	1616	1652	2
1	32	32	32	1	256	1	1	32	32	32	1	256	1	1	798	1633	1644	4
1	32	32	32	0	16	1	1	32	32	32	0	16	1	1	558	1313	1357	2
1	64	32	32	1	16	1	1	64	32	32	1	16	1	1	559	1816	1851	2
1	64	1024	1024	1	256	1	1	64	1024	1024	1	256	1	1	3384	13605	7944	34
2	32	32	32	0	16	0	1	32	32	32	0	16	0	0	246	729	575	0
2	32	64	32	0	32	0	1	32	64	32	0	32	0	0	318	744	572	0
2	32	64	64	0	64	0	1	32	64	64	0	64	0	0	410	975	607	0



Table 2-7: Spartan-6 FPGA Resource Estimates

C_INCLUDE_MM2S	C_M_AXI_MM2S_ADDR_WIDTHS	C_M_AXI_MM2S_DATA_WIDTH	C_M_AXIS_MM2S_TDATA_WIDTH	C_INCLUDE_MM2S_DRE	C_MM2S_BURST_SIZE	C_INCLUDE_MM2S_SF	C_INCLUDE_S2MM	C_M_AXI_MM2S_ADDR_WIDTHS	C_M_AXI_S2MM_DATA_WIDTH	C_S_AXIS_S2MM_TDATA_WIDTH	C_INCLUDE_S2MM_DRE	C_S2MM_BURST_SIZE	C_S2MM_SUPPORT_INDET_BTT	C_INCLUDE_MM2S_SF	Slices	Slice Reg	Slice LUTs	Block RAM
1	32	32	32	1	16	1	0	32	32	32	1	16	0	1	227	573	569	2
0	32	32	32	1	16	1	1	32	32	32	1	16	0	1	256	804	781	1
1	32	32	32	1	16	1	1	32	32	32	1	16	0	1	570	1376	1318	3
1	32	64	32	1	16	1	1	32	64	32	1	16	1	1	754	1821	1652	6
1	32	128	32	1	16	1	1	32	128	32	1	16	1	1	845	2262	1966	10
1	32	256	32	1	16	1	1	32	256	32	1	16	1	1	1103	3118	2497	18
1	32	512	32	1	16	1	1	32	512	32	1	16	1	1	1478	4827	3675	34
1	32	1024	32	1	16	1	1	32	1024	32	1	16	1	1	2154	8119	5415	70
1	32	32	8	1	16	1	1	32	32	8	1	16	1	1	512	1196	1187	4
1	32	32	16	1	16	1	1	32	32	16	1	16	1	1	428	1366	1336	4
1	32	64	64	1	16	1	1	32	64	64	1	16	1	1	939	2197	2037	6
1	32	128	128	0	16	1	1	32	128	128	0	16	1	1	916	2513	1879	10
1	32	256	256	0	16	1	1	32	256	256	0	16	1	1	1215	4087	2604	18
1	32	512	512	0	16	1	1	32	512	512	0	16	1	1	1918	7230	4129	34
1	32	1024	1024	0	16	1	1	32	1024	1024	0	16	1	1	3321	13390	7434	70
1	32	32	32	1	16	0	1	32	32	32	1	16	0	0	552	1269	1232	0
1	32	32	32	1	32	1	1	32	32	32	1	32	1	1	672	1575	1457	4
1	32	32	32	1	64	1	1	32	32	32	1	64	1	1	690	1596	1479	4
1	32	32	32	1	128	1	1	32	32	32	1	128	1	1	510	1615	1523	6
1	32	32	32	1	256	1	1	32	32	32	1	256	1	1	598	1634	1592	10
1	32	32	32	0	16	1	1	32	32	32	0	16	1	1	503	1313	1221	4
1	64	32	32	1	16	1	1	64	32	32	1	16	1	1	771	1817	1549	4
1	64	1024	1024	1	256	1	1	64	1024	1024	1	256	1	1	3452	13670	7498	70
2	32	32	32	0	16	0	1	32	32	32	0	16	0	0	234	730	529	0
2	32	64	32	0	32	0	1	32	64	32	0	32	0	0	218	745	585	0
2	32	64	64	0	64	0	1	32	64	64	0	64	0	0	255	977	687	0

# Port Descriptions

The AXI DataMover I/O signals are described in [Table 2-8](#).

Table 2-8: I/O Signal Description

Signal Name	Interface	Signal Type	Init Status	Description
<b>Memory Map to Stream Clock and Reset</b>				
m_axi_mm2s_aclk	MM2S	Input	–	Master Clock for the MM2S synchronization
m_axi_mm2s_aresetn	MM2S	Input	–	Master Reset for the MM2S logic. Active-Low assertion sensitivity. Must be asserted for three clock periods of m_axi_mm2s_aclk. See <a href="#">Reset Assertion Timing in Chapter 3</a> .
<b>Memory Map to Stream Soft Shutdown Control</b>				
mm2s_halt	MM2S	Input	–	Active-High input signal requesting that the MM2S function perform a soft shutdown and stop. See <a href="#">DataMover Soft Shutdown (Halt) Request Operations in Chapter 3</a> .
mm2s_halt_cmplt	MM2S	Output	0	Active-High output signal indicating that the MM2S function has completed a soft shutdown and is stopped. See <a href="#">DataMover Soft Shutdown (Halt) Request Operations in Chapter 3</a> .
<b>Memory Map to Stream Error Detect Discrete</b>				
mm2s_err	MM2S	Output	0	Detected Error output discrete. 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.
<b>Memory Map to Stream Debug Support</b>				
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 2-8: I/O Signal Description (Cont'd)

Signal Name	Interface	Signal Type	Init Status	Description
<b>Memory Map to Stream Address Posting Controls</b>				
mm2s_allow_addr_req	MM2S	Input	–	This input is used to control when the MM2S is allowed to post an address qualifier set on the AXI4 Read address channel. A "1" allows posting and a "0" inhibits posting. See <a href="#">DataMover External Store and Forward Support in Chapter 3</a> .
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 qualifier set posted to the AXI4 Read Address Channel. The assertion is not dependent on the address qualifier set being accepted by the AXI4. See <a href="#">DataMover External Store and Forward Support in Chapter 3</a> .
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.
<b>Memory Map to Stream Read Address Channel</b>				
m_axi_mm2s_arid (C_M_AXI_MM2S_ID_WIDTH-1:0)	MM2S	Constant Output	C_M_AXI_MM2S_ARID	MM2S Read ID Qualifier. This is always driven with a constant output set by the value assigned to the C_M_AXI_MM2S_ARID parameter.
m_axi_mm2s_araddr (C_M_AXI_MM2S_ADDR_WIDTH-1:0)	MM2S	Output	0	MM2S Read Address
m_axi_mm2s_arlen (7:0)	MM2S	Output	0	MM2S Read Length Qualifier. AXI4 proposal expands this from 4 bits to 8 bits to support Burst lengths of up to 256.
m_axi_mm2s_arsize (2:0)	MM2S	Output	0	MM2S Read Size Qualifier
m_axi_mm2s_arburst (1:0)	MM2S	Output	0	MM2S Read Burst Type Qualifier. This is always set to Incrementing Burst type ("01").
m_axi_mm2s_arprot (2:0)	MM2S	Constant Output	0	MM2S Read Protection Qualifier. This is always driven with a constant output of "000."
m_axi_mm2s_arcache (3:0)	MM2S	Constant Output	0011	MM2S Cache Qualifier. This is always driven with a constant output of "0011" unless the MM2S function is omitted; then it is driven with zeroes.
m_axi_mm2s_arvalid	MM2S	Output	0	MM2S Read Address Valid Qualifier

Table 2-8: I/O Signal Description (Cont'd)

Signal Name	Interface	Signal Type	Init Status	Description
m_axi_mm2s_arready	MM2S	Input	–	MM2S Read Address Ready Status (from Slave)
<b>Memory Map to Stream Read Data Channel</b>				
m_axi_mm2s_rdata (C_M_AXI_MM2S_DATA_WIDTH-1:0)	MM2S	Input	–	MM2S Read Data
m_axi_mm2s_rresp (1:0)	MM2S	Input	–	MM2S Read Response
m_axi_mm2s_rlast	MM2S	Input	–	MM2S Read Last Indication
m_axi_mm2s_rvalid	MM2S	Input	–	MM2S Read Valid Handshake Input
m_axi_mm2s_rready	MM2S	Output	0	MM2S Read Ready Handshake Output
<b>Memory Map to Stream Master Stream Channel</b>				
m_axis_mm2s_tvalid	MM2S Stream	Output	0	MM2S Stream Valid Handshake
m_axis_mm2s_tready	MM2S Stream	Input	–	MM2S Stream Ready Handshake
m_axis_mm2s_tdata (C_M_AXIS_MM2S_TDATA_WIDTH-1:0)	MM2S Stream	Output	0	MM2S Stream Data
m_axis_mm2s_tkeep ((C_M_AXIS_MM2S_TDATA_WIDTH/8)-1:0)	MM2S Stream	Output	0	MM2S Stream Strobes
m_axis_mm2s_tlast	MM2S Stream	Output	0	MM2S Stream Last Indication
<b>Memory Map to Stream Command/Status Channel Asynchronous Clock and Reset</b>				
m_axis_mm2s_cmdsts_aclk	MM2S Command & Status	Input	–	MM2S Command Interface Clock. This clock is only used if the MM2S Command and Status Interfaces are specified to be asynchronous to the MM2S Memory Mapped Data Channel Clock. 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 the MM2S Command and Status Interfaces are specified to be asynchronous to the MM2S Memory Mapped Data Channel Clock. Must be asserted for three clock periods of m_axis_mm2s_cmdsts_aclk. See <a href="#">Reset Assertion Timing in Chapter 3</a> .

Table 2-8: I/O Signal Description (Cont'd)

Signal Name	Interface	Signal Type	Init Status	Description
<b>Memory Map to Stream Command Channel (Slave Stream)</b>				
s_axis_mm2s_cmd_tvalid	MM2S Command	Input	–	MM2S Command Valid Handshake
s_axis_mm2s_cmd_tready	MM2S Command	Output	0	MM2S Command Ready Handshake
s_axis_mm2s_cmd_tdata((C_M_AXI_MM2S_A DDR_WIDTH+40)-1:0)	MM2S Command	Input	–	MM2S Command Data
<b>Memory Map to Stream Status Channel (Master Stream)</b>				
m_axis_mm2s_sts_tvalid	MM2S Status	Output	0	MM2S Status Valid Handshake
m_axis_mm2s_sts_tready	MM2S Status	Input	–	MM2S Status Ready Handshake
m_axis_mm2s_sts_tdata(7:0)	MM2S Status	Output	Undefined until m_axis_mm2s_sts_tvalid is asserted	MM2S Status Data
m_axis_mm2s_sts_tkeep(0:0)	MM2S Status	Constant Output	1	Driven to 1
m_axis_mm2s_sts_tlast	MM2S Status	Constant Output	1 when MM2S is present, else 0 when omitted	Driven to a constant 1 when MM2S is present, else 0 when omitted
<b>Stream to Memory Map Clock and Reset</b>				
m_axi_s2mm_aclk	S2MM	Input	–	Master Clock for the MM2S Synchronization
m_axi_s2mm_aresetn	S2MM	Input	–	Master Reset for the MM2S logic (Active-Low sensitivity). Must be asserted for three clock periods of m_axi_s2mm_aclk. See <a href="#">Reset Assertion Timing in Chapter 3</a> .

Table 2-8: I/O Signal Description (Cont'd)

Signal Name	Interface	Signal Type	Init Status	Description
<b>Stream to Memory Map Soft Shutdown Control</b>				
s2mm_halt	S2MM	Input	–	Active-High input signal requesting that the S2MM function perform a soft shutdown and stop. See <a href="#">DataMover Soft Shutdown (Halt) Request Operations</a> in Chapter 3.
s2mm_halt_cmplt	S2MM	Output	0	Active-High output signal indicating that the S2MM function has completed a soft shutdown and is stopped. See <a href="#">DataMover Soft Shutdown (Halt) Request Operations</a> in Chapter 3.
<b>Stream to Memory Map Error Detect Discrete</b>				
s2mm_err	S2MM	Output	0	Detected Error output discrete. 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 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.
<b>Stream to Memory Map Debug Support</b>				
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 2-8: I/O Signal Description (Cont'd)

Signal Name	Interface	Signal Type	Init Status	Description
<b>Stream to Memory Map Address Posting Controls</b>				
s2mm_allow_addr_req	S2MM	Input	–	This input is used to control when the S2MM is allowed to post an address qualifier set on the AXI4 Write Address Channel. A “1” allows posting and a “0” inhibits posting. See <a href="#">DataMover External Store and Forward Support in Chapter 3</a> .
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 qualifier set posted to the AXI4 Write Address Channel. The assertion is not dependent on the address qualifier set being accepted by the AXI4.
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 is used to qualify 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.
<b>Stream to Memory Map Write Address Channel</b>				
m_axi_s2mm_awid (C_M_AXI_S2MM_ID_WIDTH-1:0)	S2MM	Constant Output	C_M_AXI_S2MM_AWID	S2MM Write Address ID Qualifier. This is always driven with a constant output set by the value assigned to the C_M_AXI_S2MM_AWID parameter.
m_axi_s2mm_awaddr (C_M_AXI_S2MM_ADDR_WIDTH-1:0)	S2MM	Output	0	S2MM Write Address
m_axi_s2mm_awlen (7:0)	S2MM	Output	0	S2MM Write Length Qualifier This qualifier has been expanded in the AXI4 proposal from 4 bits to 8 bits to support Burst lengths of up to 256 data beats.
m_axi_s2mm_awsz (2:0)	S2MM	Output	0	S2MM Write Qualifier

Table 2-8: I/O Signal Description (Cont'd)

Signal Name	Interface	Signal Type	Init Status	Description
m_axi_s2mm_awburst (1:0)	S2MM	Output	0	S2MM Write Burst Type Qualifier. This is always set to Incrementing Burst type. ("01")
m_axi_s2mm_awprot (2:0)	S2MM	Constant Output	0	S2MM Write Protection Qualifier. This is always driven with a constant output of "000."
m_axi_s2mm_awcache (3:0)	S2MM	Constant Output	0011	S2MM Cache Qualifier. This is always driven with a constant output of "0011" unless the S2MM function is omitted; then it is driven with zeroes.
m_axi_s2mm_awvalid	S2MM	Output	0	S2MM Write Address Valid Qualifier
m_axi_s2mm_awready	S2MM	Input	–	S2MM Write Address Ready Status (from Slave)
<b>Stream to Memory Map Write Data Channel</b>				
m_axi_s2mm_wdata (C_M_AXI_S2MM_DATA_WIDTH-1:0)	S2MM	Output	0	S2MM Write Data
m_axi_s2mm_wstrb ((C_M_AXI_S2MM_DATA_WIDTH/8)-1:0)	S2MM	Output	0	S2MM Write Strobes
m_axi_s2mm_wlast	S2MM	Output	0	S2MM Write Last Indication
m_axi_s2mm_wvalid	S2MM	Output	0	S2MM Write Valid Handshake Output
m_axi_s2mm_wready	S2MM	Input	–	S2MM Write Ready Handshake Input
<b>Stream to Memory Map Write Response Channel</b>				
m_axi_s2mm_bresp (1:0)	S2MM	Input	–	S2MM Write ID. This is passed to the Read Stream output.
m_axi_s2mm_bvalid	S2MM	Input	–	S2MM Write Valid Handshake Input
m_axi_s2mm_bready	S2MM	Output	0	S2MM Write Ready Handshake Output
<b>Stream to Memory Map Slave Stream Channel</b>				
s_axis_s2mm_tvalid	S2MM Stream	Input	–	S2MM Stream Valid Handshake In
s_axis_s2mm_tready	S2MM Stream	Output	0	S2MM Stream Ready Handshake Out
s_axis_s2mm_tdata (C_S_AXIS_S2MM_TDATA_WIDTH-1:0)	S2MM Stream	Input	–	S2MM Stream Data In
s_axis_s2mm_tkeep ((C_S_AXIS_S2MM_TDATA_WIDTH/8)-1:0)	S2MM Stream	Input	–	S2MM Stream Strobes In
s_axis_s2mm_tlast	S2MM Stream	Input	–	S2MM Stream Last Indication



Table 2-8: I/O Signal Description (Cont'd)

Signal Name	Interface	Signal Type	Init Status	Description
<b>Stream to Memory Map Command/Status Channel Asynchronous Clock and Reset</b>				
m_axis_s2mm_cmdsts_awclk	S2MM Command & Status	Input	–	S2MM Command Interface Clock. This clock is only used if the S2MM Command and Status Interfaces are specified to be asynchronous to the S2MM Memory Mapped Data Channel Clock. The frequency of this clock is expected to be equal or less than the m_axi_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 the S2MM Command and Status Interfaces are specified to be asynchronous to the S2MM Memory Mapped Data Channel Clock. Must be asserted for three clock periods of m_axis_s2mm_cmdsts_awclk. See <a href="#">Reset Assertion Timing in Chapter 3</a>
<b>Stream to Memory Map Command Channel (Slave Stream)</b>				
s_axis_s2mm_cmd_tvalid	S2MM Command	Input	–	S2MM Command Valid Handshake
s_axis_s2mm_cmd_tready	S2MM Command	Output	0	S2MM Command Ready Handshake
s_axis_s2mm_cmd_tdata((C_M_AXI_S2MM_A_DDR_WIDTH+32)-1:0)	S2MM Command	Input	–	S2MM Command Data
<b>Stream to Memory Map Status Channel (Master Stream)</b>				
m_axis_s2mm_sts_tvalid	S2MM Status	Output	0	S2MM Status Valid Handshake
m_axis_s2mm_sts_tready	S2MM Status	Input	–	S2MM Status Ready Handshake
m_axis_s2mm_sts_tdata(7:0)	S2MM Status	Output	Undefined until m_axis_s2mm_sts_tvalid is asserted	S2MM Status Data
m_axis_s2mm_sts_tkeep	S2MM Status	Constant Output	1	S2MM Status Strobes always driven to 1's
m_axis_s2mm_sts_tlast	S2MM Status	Constant Output	1 when S2MM is present, else 0 when omitted	Driven to a constant "1" when S2MM is present, else "0" when omitted

# 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.



**TIP:** You can optionally use TDEST FIFO to store TDEST information while queuing commands in DataMover.

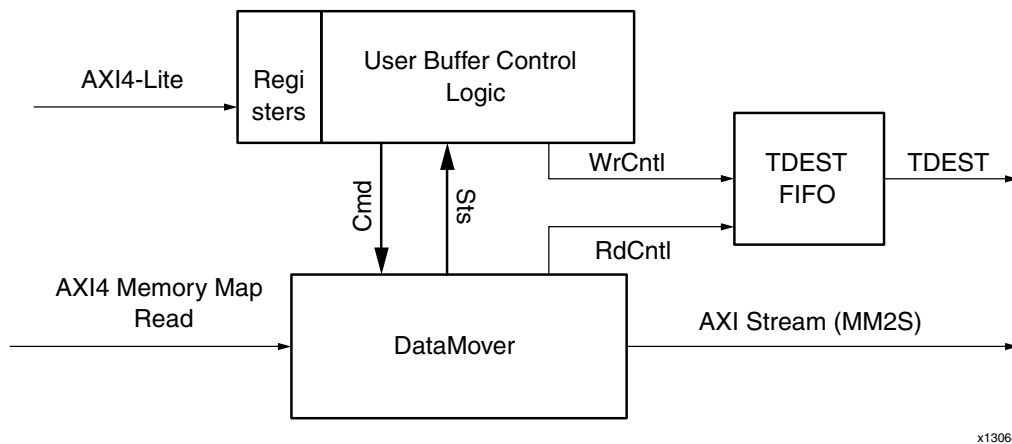


Figure 3-1: Typical Application of MM2S DataMover

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

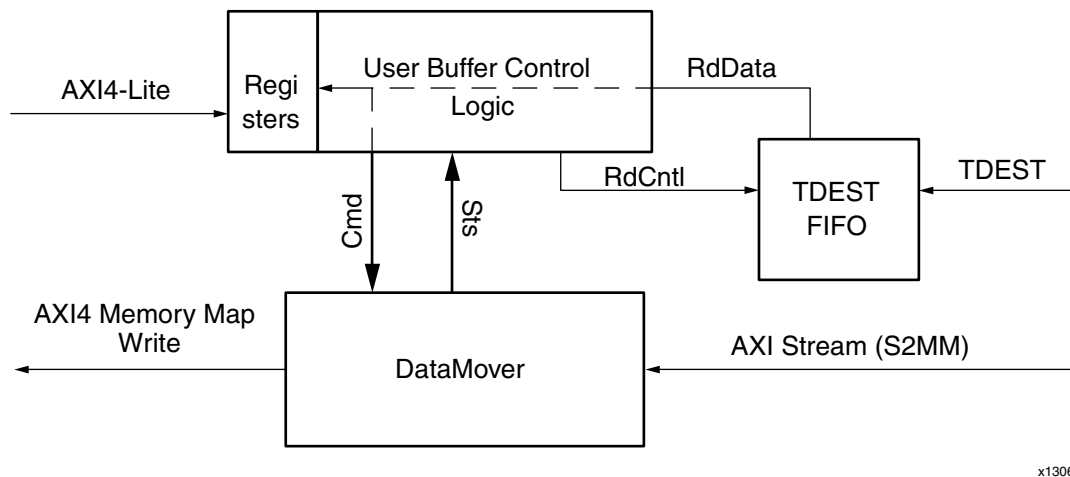


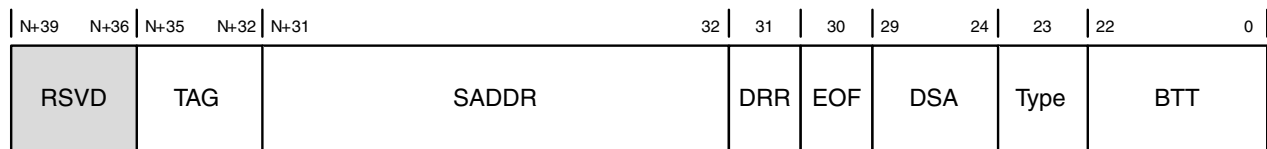
Figure 3-2: Typical Application of S2MM DataMover

# General Design Guidelines

## 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 must grow (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 format of the command word is shown in Figure 3-3 and detailed in Table 3-1. 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 (7FFFFFF 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 Memory Mapped protocol requirements.



N = C\_M\_AXI\_MM2S\_ADDR\_WIDTH for Memory Map to Stream Channel  
 or C\_M\_AXI\_S2MM\_ADDR\_WIDTH for Stream to Memory Map Channel

X12284

Figure 3-3: Command Word Layout

Table 3-1: Command Word Description

Bits	Field Name	Description
(N+39) - (N+36) <sup>(1)</sup>	RSVD	<b>Reserved</b> 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) <sup>(1)</sup>	TAG	<b>Command TAG</b> This field is an arbitrary value assigned by the user to the Command. The TAG flows through the DataMover execution pipe and gets inserted into the corresponding status word for the Command.

Table 3-1: Command Word Description (Cont'd)

Bits	Field Name	Description
(N+31) -32 <sup>(1)</sup>	SADDR	<p><b>Start Address</b></p> <p>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.</p>
31	DRR	<p><b>DRE ReAlignment Request</b></p> <p>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.</p>
30	EOF	<p><b>End of Frame</b></p> <p>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. <i>If DRE is included</i>, 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).</p>
29-24	DSA	<p><b>DRE Stream Alignment</b></p> <p>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.</p> <p><b>Note:</b> DRE alignment on the associated Memory Mapped side is derived from the least significant bits of the SADDR value of the command.</p>
23	Type	<p><b>Reserved</b></p> <p>This 1-bit field is currently reserved and ignored by the DataMover.</p>
22 to 0	BTT	<p><b>Bytes to Transfer</b></p> <p>This 23-bit field indicates the total number of bytes to transfer for the command. A transfer of 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 C_MM2S_BTT_USED and C_S2MM_BTT_USED.</p>

**Notes:**

1. N is equal to the value assigned to the parameter C\_M\_AXI\_MM2S\_ADDR\_WIDTH or C\_M\_AXI\_S2MM\_ADDR\_WIDTH depending on the applicable DataMover command interface.

## Command FIFO

The Command interface of a DataMover element is designed to allow command queuing. The commands are “queued” in a FIFO that has a parameterized depth. For more information, see the parameters [C\\_MM2S\\_STSCMD\\_FIFO\\_DEPTH](#), page 45 and [C\\_S2MM\\_STSCMD\\_FIFO\\_DEPTH](#), page 48.

The Command FIFO is by default a synchronous FIFO clocked by the same clock that is clocking 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) clock frequency than the Memory Mapped Data and Address channel clocking frequency.

The selection of synchronous or asynchronous is made through the `C_MM2S_STSCMD_IS_ASYNC` and `C_S2MM_STSCMD_IS_ASYNC` parameters assignments (0 = synchronous, 1 = asynchronous).

## Command Load Timing by the Command Stream Interface

Loading a command into the Command FIFO is mechanized by a single AXI4-Stream data beat. Because the DataMover Command is a wide-parallel word format, only one stream data beat is required to load one command. An example of loading five commands into the MM2S Command FIFO is shown in [Figure 3-4](#). In this scenario, the Command FIFO is synchronous to the Memory Mapped Address and Data Channel clock. The example illustrates that a command is considered loaded into the Command FIFO only when both the `TVALID` and `TREADY` handshake signals are both asserted at the rising edge of the clock. `TLAST` and `TSTRB` signals are ignored.

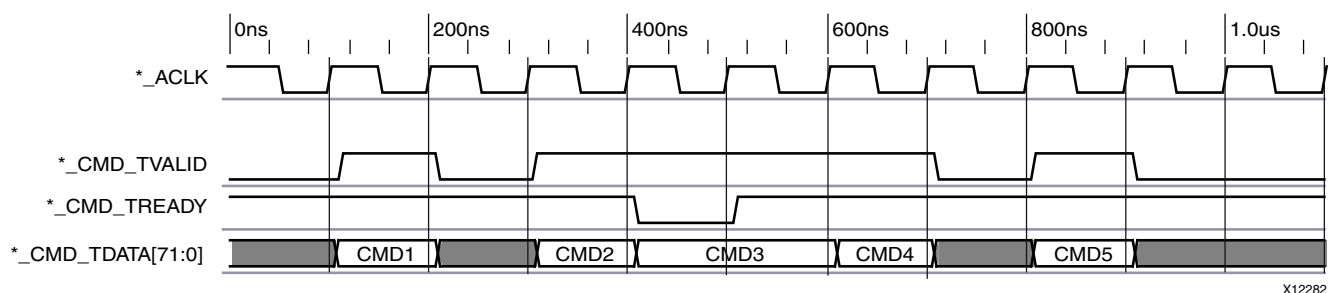


Figure 3-4: Loading Commands via the Command Interface

## Status Interface

The status of DataMover transfer operations are 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. One exception is the S2MM side when Indeterminate BTT mode is enabled ([Special S2MM Status Format when Indeterminate BTT Support is Enabled, page 31](#)). The format of the status word is shown in [Figure 3-5](#) and detailed in [Table 3-2](#). It is the same for either the MM2S or S2MM DataMover elements.

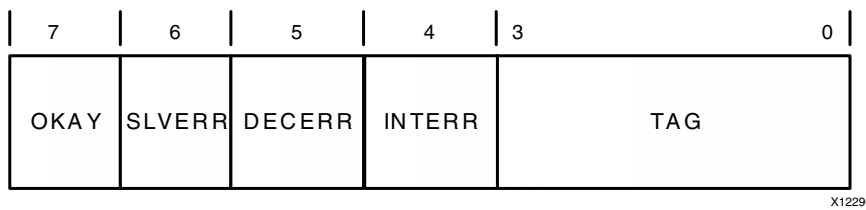


Figure 3-5: Normal Status Word Layout

Table 3-2: Status Word Details

Bits	Field Name	Description
7	OKAY	<p><b>Transfer OKAY</b></p> <p>This bit indicates that the associated transfer command has been completed with the OKAY response on all intermediate transfers.</p> <p>0 = Command had a non-OKAY response during all associated transfers</p> <p>1 = Command had a OKAY response during all associated transfers</p>
6	SLVERR	<p><b>Slave Error</b></p> <p>Indicates the DataMover Element encountered a Slave reported error condition for the associated command. This is received by the Response inputs from the AXI4 Memory Mapped interface.</p> <p>0 = No Error</p> <p>1 = Slave Asserted Error Condition</p>
5	DECERR	<p><b>Decode Error</b></p> <p>Indicates the DataMover Element encountered an address decode error condition for the associated command. This is received by the Response inputs from the AXI4 Memory Mapped interface and indicates an address decode timeout occurred on an address generated by the DataMover Element while executing the corresponding Command.</p> <p>0 = No Error</p> <p>1 = Address Decode Error Condition</p>

Table 3-2: Status Word Details (Cont'd)

Bits	Field Name	Description
4	INTERR	<p><b>Internal Error</b></p> <p>Indicates the DataMover Element 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. The S2MM function can also assert this if the input stream TLAST assert occurs prematurely (relative to the commanded BTT for the transfer) and the Indeterminate BTT mode is not enabled.</p> <p>0 = No Error 1 = Internal Error Condition</p>
3 to 0	TAG	<p><b>TAG</b></p> <p>This 4-bit field echoes the value of the TAG field of the associated input Command whose completion generated the Status.</p>

### Special S2MM Status Format when Indeterminate BTT Support is Enabled

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 3-6 and detailed in Table 3-3. This status format does not apply to the MM2S DataMover status interface.



X12295

Figure 3-6: Special S2MM Status Word Layout (IBTT Mode Enabled)

Table 3-3: Special S2MM Status Word Details (Indeterminate BTT Mode Enabled)

Bits	Field Name	Description
31	EOP	<b>End of Packet</b> 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. It is needed by certain Users (that is, Scatter Gather Engines) to identify the actual End of Packet for the input Stream versus the theoretical maximum that could occur.
30 to 8	BRCVD	<b>Bytes Received</b> 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	<b>Transfer OKAY</b> 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	<b>Slave Error</b> Indicates the DataMover Element encountered a Slave reported error condition for the associated command. This is received by the Response inputs from the AXI4 Memory Mapped interface. 0 = No Error 1 = Slave Asserted Error Condition
5	DECERR	<b>Decode Error</b> Indicates the DataMover Element encountered an address decode error condition for the associated command. This is received by the Response inputs from the AXI4 Memory Mapped 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	<b>Internal Error</b> Indicates the DataMover Element 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. The S2MM function can also assert this if the input stream TLAST assert occurs prematurely (relative to the commanded BTT for the transfer) and the Indeterminate BTT mode is not enabled. Additional conditions are To Be Determined. 0 = No Error 1 = Internal Error Condition
3 to 0	TAG	<b>TAG</b> 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 element 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 that has a parameterizable depth. For more information, see the parameters [C\\_MM2S\\_STSCMD\\_FIFO\\_DEPTH, page 45](#) and [C\\_S2MM\\_STSCMD\\_FIFO\\_DEPTH, page 48](#). Status values have a one-to-one correlation to loaded commands by the Command Interface.

The Status FIFO is by default a synchronous FIFO clocked by the same clock that is clocking the Memory Mapped Data and Address channels 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) clock frequency than the associated Memory Mapped Data and Address channel clocking frequency.

The selection of synchronous or asynchronous mode is made by the `C_MM2S_STSCMD_IS_ASYNC` and `C_S2MM_STSCMD_IS_ASYNC` parameters assignments (0 = synchronous, 1 = asynchronous).

## Status Read Timing by the Status Stream Interface

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 3-7](#). 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 both the `TVALID` and `TREADY` handshake signals are both asserted at the rising edge of the synchronizing clock.

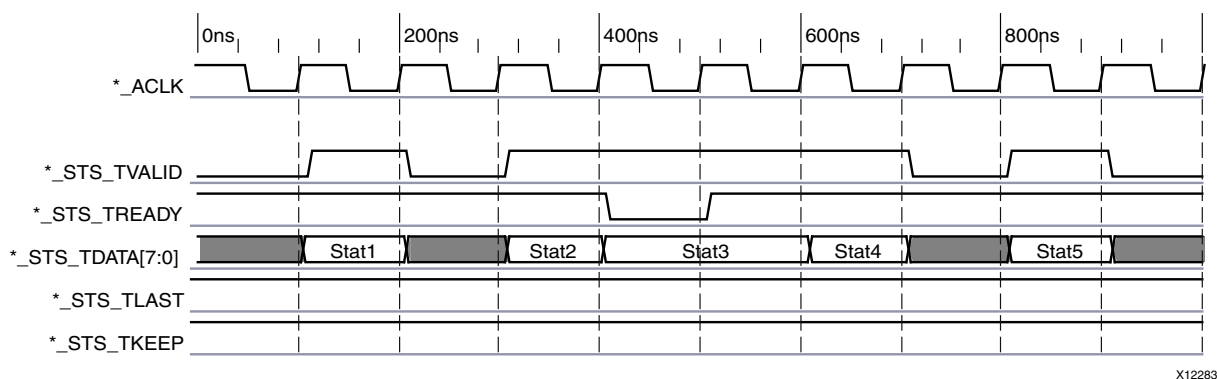


Figure 3-7: Reading Status over the Status Interface

X12283

## General Purpose MM2S Store and Forward

The MM2S can include an optional Store and Forward block when the parameter `C_MM2S_INCLUDE_SF` is assigned a value of 1 (the default). 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. The depth of the MM2S Store and Forward data FIFO is set by the following calculation:

$((C\_MM2S\_ADDR\_PIPE\_DEPTH+2) \times C\_MM2S\_BURST\_SIZE)$  rounded up to the next power of 2.

## Indeterminate BTT Mode

The 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). This mode is enabled by the top level parameter `C_S2MM_SUPPORT_INDET_BTT` being set to a value of 1.

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.

The corresponding status output by the S2MM block for the associated transfer command does not have the EOP bit set and the BRCVD field in the status word only reflects the commanded BTT value, not the actual number of bytes received for the input overflow packet. Only the data up to the BTT value is written to the Memory Mapped space by the S2MM AXI4 Write Data Channel.

## General Purpose S2MM Store and Forward

The S2MM can include an optional General Purpose Store and Forward block when the parameter `C_S2MM_INCLUDE_SF` is assigned a value of 1. This is the default. 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. The depth of the S2MM General Purpose (GP) Store and Forward data FIFO is set by the following calculation:

$((C\_S2MM\_ADDR\_PIPE\_DEPTH+2) \times C\_S2MM\_BURST\_SIZE)$  rounded up to the next power of 2.

## 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 implementation. The parameters C\_INCLUDE\_S2MM and C\_INCLUDE\_MM2S are used to include/omit the S2MM and MM2S elements independently. In addition, the same parameters are used to select between a Full version and the Basic version of the DataMover. See [Table 3-5](#).

### DataMover Basic Feature Reduction from the Full Version

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 Memory Map Burst Length support of 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 Memory Map Burst Length multiplied by the Stream data width (in bytes)

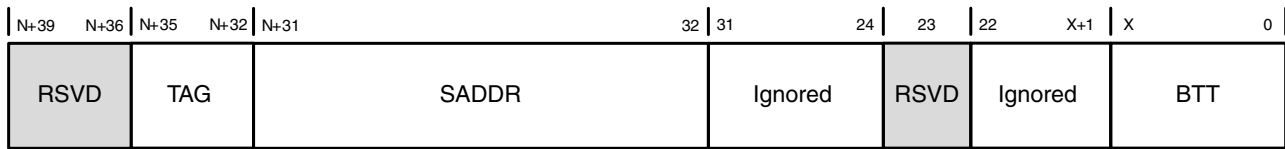
Example: Maximum burst length = 32, Stream Data Width = 4 bytes (32 bits), the maximum commanded transfer length (BTT) is 128 bytes

- 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

### DataMover Basic Command Interface

The format of the Basic command word is shown in [Figure 3-8](#) and detailed in [Table 3-4](#).

**Note:** \* = S2MM or MM2S



$X = \text{Log}_2[C\_*\_BURST\_SIZE \times (C\_M\_AXIS\_*\_DATA\_WIDTH/8)]$  {Note: \* = S2MM or MM2S}

N = C\_M\_AXI\_MM2S\_ADDR\_WIDTH for Memory Map to Stream Channel or  
C\_M\_AXI\_S2MM\_ADDR\_WIDTH for Stream to Memory Map Channel

X12288

Figure 3-8: DataMover Basic Command Word Layout

Table 3-4: DataMover Basic Command Word Details

Bits	Field Name	Description
(N+39) - (N+36) <sup>(1)</sup>	RSVD	<b>Reserved</b> 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) <sup>(1)</sup>	TAG	<b>Command TAG</b> The user assigns 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 <sup>(1)</sup>	SADDR	<b>Start Address</b> This field indicates the starting address to use for the Memory Mapped side of the transfer requested by the command.
31-24	Ignored	This field is ignored by the DataMover Basic. Can be any value but zeroes are recommended.
23	RSVD	<b>Reserved</b> This 1-bit field is reserved and ignored by the DataMover.
22-(X+1)	Ignored	This field is ignored by the DataMover Basic. Can be any value but zeroes are recommended.
X-0	BTT	<b>Bytes to Transfer</b> This field indicates the total number of bytes to transfer for the command. The maximum allowed value is set by the following formula: $C\_*\_BURST\_SIZE \times (C\_M\_AXIS\_*\_DATA\_WIDTH/8)$ {Note: * = S2MM or MM2S}

**Notes:**

1. N is equal to the value assigned to the parameter C\_M\_AXI\_MM2S\_ADDR\_WIDTH or C\_M\_AXI\_S2MM\_ADDR\_WIDTH depending on the applicable DataMover command interface.

## DataMover Basic Status Interface

The format of the status word is the same as the full version and is shown in [Figure 3-5](#), and detailed in [Table 3-2](#).

### Example DataMover Read(MM2S) Timing

[Figure 3-9](#) illustrates example timing on read (MM2S) path in synchronous mode.

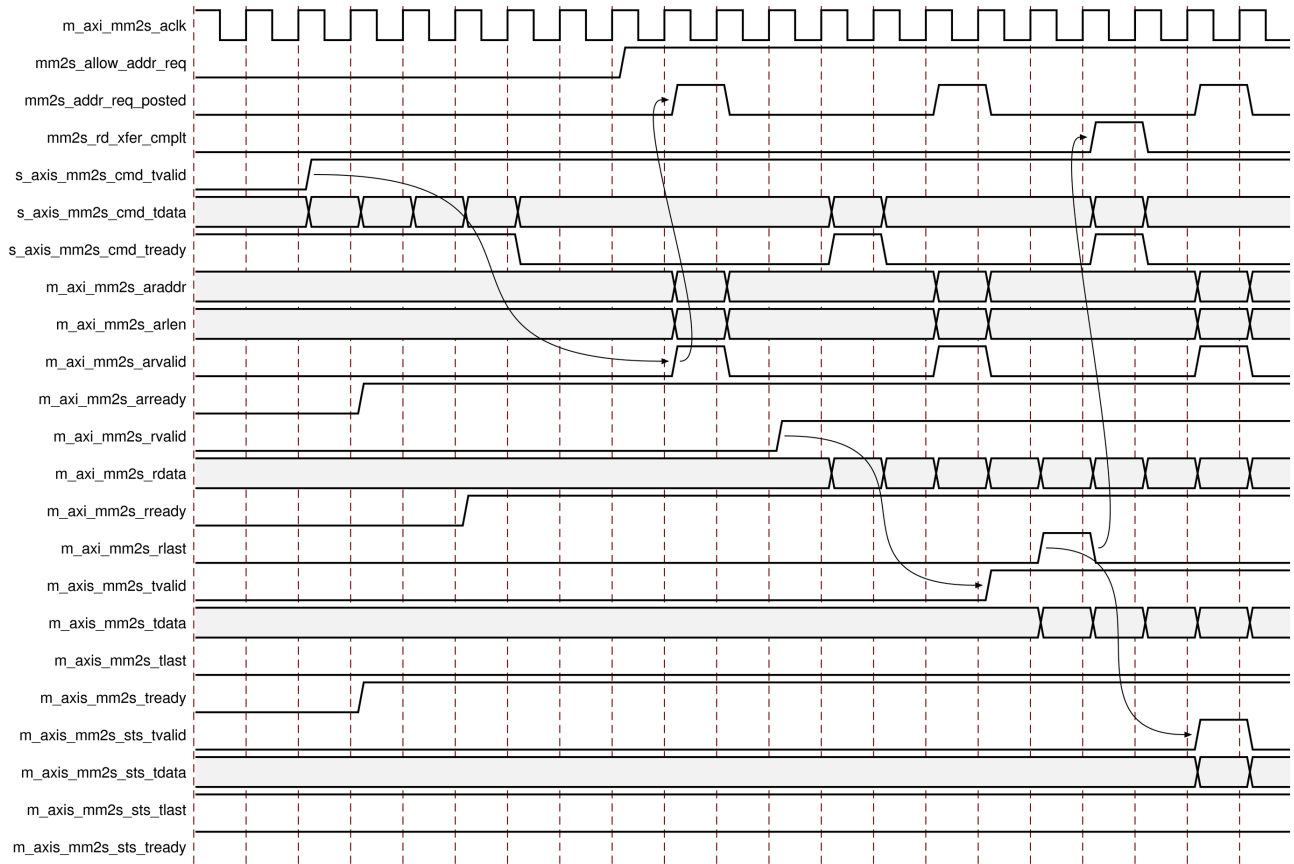


Figure 3-9: Example Timing on Read (MM2S) Path in Synchronous Mode

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 read cycle on the AXI MMap 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_cplt` 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 AXI MMap Interface. Status signals are asserted when all child commands are processed.*

---

## Example DataMover Write(S2MM) Timing

Figure 3-10 illustrates example timing on write (S2MM) path in synchronous mode.

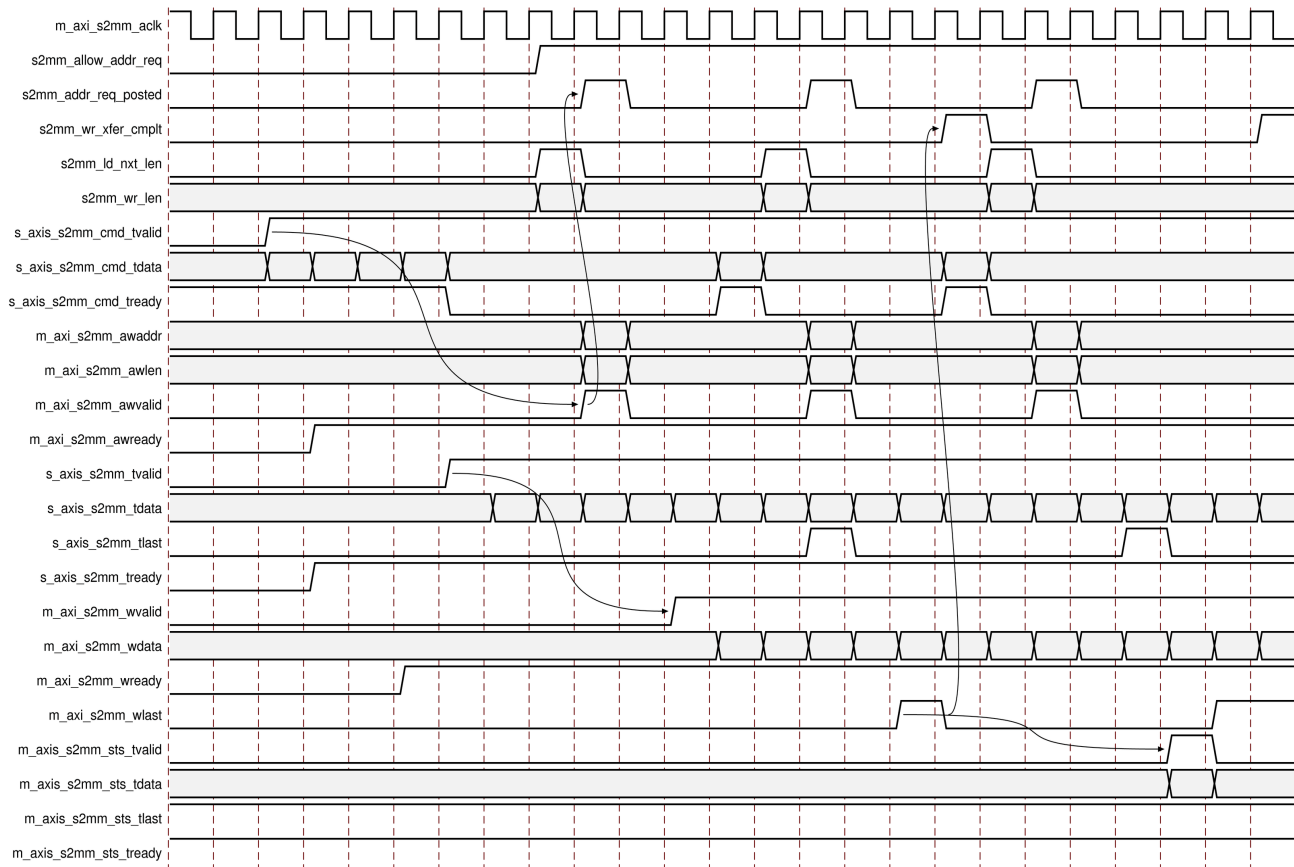


Figure 3-10: Example Timing on Write (S2MM) Path in Synchronous Mode

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 AXI MMap 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_cpltd` 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 AXI MMap interface.



---

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

---

## Clocking

The 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 Memory Mapped interface and Stream interface. The second clock for the MM2S element is the `m_axis_mm2s_cmdsts_awclk`. This clock is used only when the parameter `C_MM2S_STSCMD_IS_ASYNC` is assigned a value of 1. When used, it synchronizes the User sides of the Command and Status interfaces. If the parameter `C_MM2S_STSCMD_IS_ASYNC` is assigned a value of 0, the `m_axis_mm2s_cmdsts_awclk` is not used and the User sides of the Command and Status interfaces are synchronized with the `m_axi_mm2s_aclk`.

The S2MM block has identical clocking schemes as the MM2S block but with two different clocks, the `m_axis_s2mm_cmdsts_awclk` and `m_axis_s2mm_cmdsts_awclk`. Synchronous or Asynchronous Command/Status mode is controlled by the `C_MM2S_STSCMD_IS_ASYNC` parameter.

## Resets

The DataMover has two reset inputs for each of the MM2S and S2MM blocks for a total of four reset inputs. The DataMover is designed such that the Command and Status interfaces can optionally be clocked and reset with a secondary clock and reset input for each of the MM2S and S2MM functions. This is intended to allow these interfaces to operate at a slower clock frequency than the main data payload paths.



## Reset Assertion Timing

Because the DataMover internally synchronizes the input resets by registering, internal logic reset and port I/O reaction are delayed by up to two clocks after the first rising edge of the synchronizing clock. This synchronization period requires that any reset assertion to the DataMover must be a minimum of three clock periods of the synchronizing clock. The reset period and I/O relationship is shown in [Figure 3-11](#).

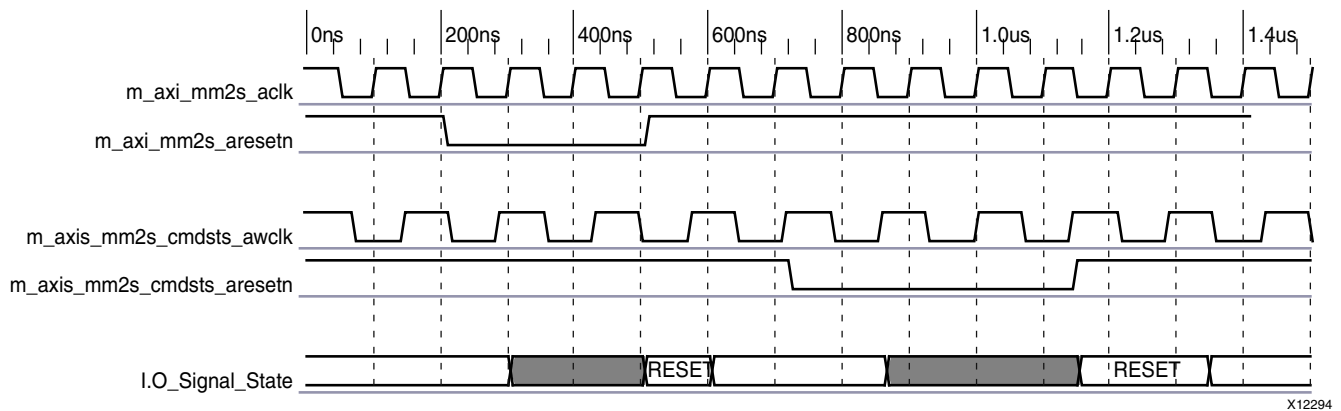


Figure 3-11: Reset Assertion Timing

## DataMover External Store and Forward Support

The AXI DataMover has additional features to allow for support of an external Store and Forward function should the internal ones not provide the needed function. An example use case is AXI Central Direct Memory Access (CDMA), where a single data FIFO solution is optimum but with both Read and Write Address posting control by the Store and Forward Controller. The purpose of Store and Forward is to eliminate or minimize the need for the DataMover to throttle the AXI4 Read and Write Data Channels. This implies that the DataMover MM2S function does not post a Read Request to the AXI4 unless the associated read data transfer can complete without the DataMover throttling by deasserting the `m_axi_mm2s_rready` signal.

In addition, the DataMover S2MM function does not post a Write Request to the AXI4 unless the associated write data transfer can complete without the DataMover throttling by deasserting the `m_axi_s2mm_wvalid` signal. These two requirements imply a data storage FIFO is needed and special monitoring logic must be employed to track the data input and output levels. The monitoring logic is then allowed to control the DataMover address/qualifier posting to AXI4 based on the available space and available data levels.

## DataMover External Store and Forward Ports

The DataMover external Store and Forward interface consists of eight ports. 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)

## Usage

The connection of these ports to an external Store and Forward module is shown in [Figure 3-12](#). This is representative of the AXI CDMA use case. The external Store and Forward module has the ability to control the DataMover Address/Qualifier 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 transfer address/qualifiers 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/qualifiers to the AXI4 bus.

The DataMover also provides status back to the Store and Forward block indicating when an address/qualifier 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 also provides two more outputs (`s2mm_wr_len` and `s2mm_ld_nxt_len`) that are used to provide some lookahead to the 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 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 DataMover to pipeline Read requests to the AXI4 without over-committing the Store and Forward Data FIFO capacity (filling it up and throttling the AXI4 Read data Channel). It also can keep the DataMover from pipelining Write transfers until the write data is actually present in the data fifo and ready to be written. This keeps the DataMover from pipelining write requests to the AXI4 when the write data is not yet available (causing the DataMover to throttle the AXI4 Write Data Channel).

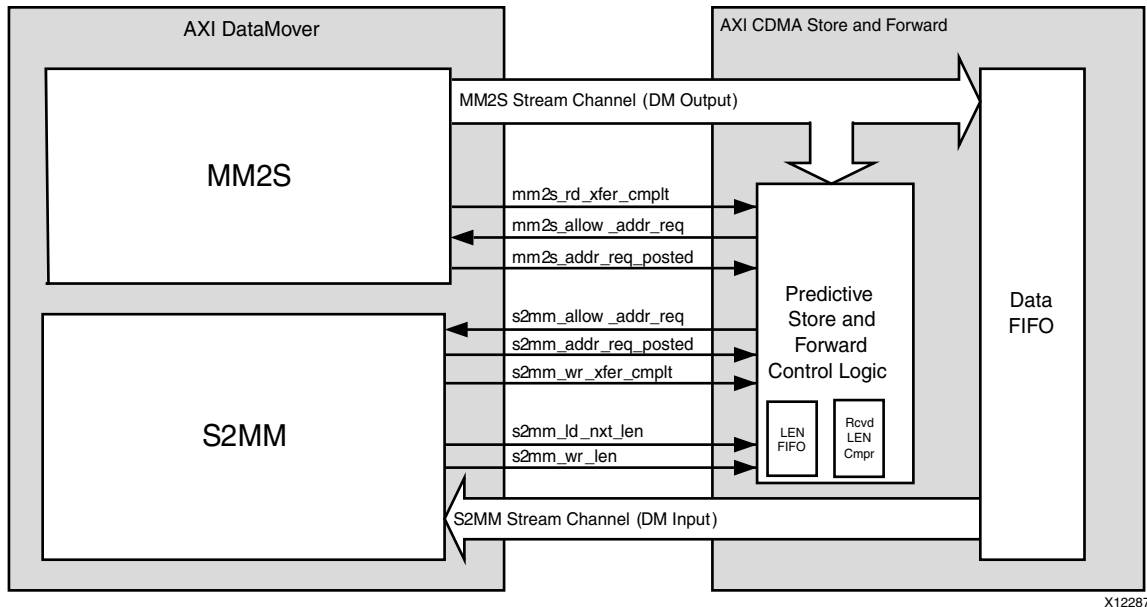


Figure 3-12: CDMA Store and Forward Connection Example with Address Pipeline Control

### 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. This 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). The parameters are [C\\_MM2S\\_BURST\\_SIZE, page 46](#) and [C\\_S2MM\\_BURST\\_SIZE, page 49](#).

### DataMover Soft Shutdown (Halt) Request Operations

The overall command queue design of DataMover allows it to fully exploit the AXI4 address pipeline capability. This is sometimes detrimental when it is required for the DataMover to be stopped quickly to support a soft reset in the Host core. The AXI4 Memory Mapped Bus requires that transfers committed on the Address Channel by a Master must also have the associated Data Channel transfers completed. This requirement causes the DataMover to implement a soft shutdown scheme that completes all committed Memory Map transfers before it can stop operation. It is desirable for the DataMover to only complete what is committed to the Memory Map bus and not allow any other queued commands to advance in the pipeline to the point of being committed to the Memory Map Bus.

## MM2S Soft Shutdown

The DataMover MM2S soft shutdown is initiated by the active-High assertion of the input signal `mm2s_halt`. When the soft halt operations are completed, the MM2S asserts the `mm2s_halt_cpltd` outputs. 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).

During a soft shutdown, DataMover gracefully completes the existing transactions on the AXI MM 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.

## S2MM Soft Shutdown

The S2MM soft shutdown is initiated by the active-High assertion of the input signal `s2mm_halt`. When the soft halt operations are completed, the S2MM asserts the `s2mm_halt_cpltd` 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).

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.

# Design Parameters

The AXI DataMover Design Parameters are listed and described in [Table 3-5](#).

Table 3-5: Design Parameters

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
<b>Memory Map to Stream Parameters</b>				
C_INCLUDE_MM2S	0, 1, 2	2	Integer	0 = Exclude Memory Map to Stream channel (Omit Mode) 1 = Include Memory Map to Stream channel (Full Mode) 2 = Include Memory Map to Stream channel but with limited functionality for a reduction in resource utilization (Basic Mode) When C_INCLUDE_MM2S = 0, all inputs to the MM2S element are ignored and all outputs from the MM2s element are driven to zeroes.
C_M_AXI_MM2S_ARID	0 to 255	0	Integer	The value to drive onto the MM2S Address Channel ARID output
C_M_AXI_MM2S_ID_WIDTH	1 to 8	4	Integer	The bit width of any MM2S ID buses
C_M_AXI_MM2S_ADDR_WIDTH	32 to 64	32	Integer	Address width on the Memory Map to Stream channel
C_M_AXI_MM2S_DATA_WIDTH	32, 64, 128, 256, 512, 1024	32	Integer	Data width on the Memory Map to Stream channel's Memory Map interface
C_M_AXIS_MM2S_TDATA_WIDTH	8, 16, 32, 64, 128, 256, 512, 1024	32	Integer	Data width on the Memory Map to Stream channel's Stream interface. Must be equal to or less than C_M_AXI_MM2S_DATA_WIDTH
C_INCLUDE_MM2S_STSFIFO	0, 1	1	Integer	0 = Exclude Memory Map to Stream Status FIFO 1 = Include Memory Map to Stream Status FIFO
C_MM2S_STSCMD_FIFO_DEPTH	1, 4, 8, 16	4	Integer	Depth of Memory Map to Stream Status and Command FIFO. A specified depth of 1 indicates a single register implementation.  <b>Note:</b> If C_MM2S_STSCMD_IS_ASYNC is set to a value of 1 (asynchronous mode), the Command and Status FIFO depth is automatically set to a depth of 16 and this parameter is ignored.

Table 3-5: Design Parameters (Cont'd)

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
C_MM2S_STSCMD_IS_ASYNC	0, 1	0	Integer	0 = MM2S Command and Status Stream interfaces are synchronous to the MM2S Memory Mapped interface (uses same clock) 1 = MM2S Command and Status Stream interfaces are asynchronous to the MM2S Memory Mapped interface (uses different clock)
C_INCLUDE_MM2S_DRE	0, 1	1	Integer	DRE support is only available for AXI4-Stream data widths of 16, 32, and 64 bits. 0 = Exclude Memory Map to Stream Data Realignment engine 1 = Include Memory Map to Stream Data Realignment engine
C_MM2S_BURST_SIZE	16, 32, 64, 128, 256	16	Integer	Memory Map to Stream channel maximum allowed burst length (in data beats).  <b>Note:</b> When the parameter C_M_AXIS_MM2S_TDATA_WIDTH is set to 1024-bit, the MM2S burst length is internally limited to 32 data beats so that the AXI 4K address boundary restriction is not violated. When C_S_AXIS_MM2S_TDATA_WIDTH is set to 512 or 256, the MM2S burst length is internally limited to 64 or 128 respectively.
C_MM2S_BTT_USED	8 to 23	16	Integer	Actual number of bits to be used from the MM2S Command BTT field. The BTT used bits are extracted from the field starting from (C_MM2S_BTT_USED-1) down to bit 0.  <b>Note:</b> Value assigned to this parameter must be greater than or equal to the value $\log_2(C\_M\_AXIS\_MM2S\_TDATA\_WIDTH/8 \times C\_MM2S\_BURST\_SIZE) + 1$ .

Table 3-5: Design Parameters (Cont'd)

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
C_MM2S_ADDR_PIPE_DEPTH	1 to 30	3	Integer	This parameter specifies the internal MM2S queuing depth used for child command address pipelining. The value controls how many address qualifier sets can be committed (pipelined) to the AXI4 Read Address Channel before the associated read data starts flowing into the MM2S function on the AXI4 Read Data Channel. The effective pipelining that is observed on the AXI4 Read Address Channel is the value assigned to this parameter plus 2.
C_MM2S_INCLUDE_SF	0, 1	1	Integer	This parameter specifies the inclusion/omission of the MM2S (Read) Store and Forward function. If the MM2S Stream Channel data width is less than the Memory Map Data Width, a downsizer function is automatically inserted on the Stream side of the Store and Forward FIFO. 0 = Omit MM2S GP Store and Forward 1 = Include MM2S GP Store and Forward
<b>Stream to Memory Map Parameters</b>				
C_INCLUDE_S2MM	0, 1, 2	2	Integer	0 = Exclude Stream to Memory Map channel (Omit Mode) 1 = Include Stream to Memory Map channel (Full Mode) 2 = Include Stream to Memory Map channel but with limited functionality for a reduction in resource utilization (Basic Mode) When C_INCLUDE_S2MM = 0, all inputs to the S2MM element are ignored and all outputs from the S2MM element are driven to Zeroes.
C_M_AXI_S2MM_AWID	0 to 255	1	Integer	The value to drive onto the S2MM Address Channel AWID output and the Data Channel WID output.
C_M_AXI_S2MM_ID_WIDTH	1 to 8	4	Integer	The bit width of any S2MM ID buses
C_M_AXI_S2MM_ADDR_WIDTH	32 to 64	32	Integer	Address width on the Stream to Memory Map channel
C_M_AXI_S2MM_DATA_WIDTH	32, 64, 128, 256, 512, 1024	32	Integer	Data width on the Stream to Memory Map channel's Memory Map interface

Table 3-5: Design Parameters (Cont'd)

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
C_S_AXIS_S2MM_TDATA_WIDTH	8, 16, 32, 64, 128, 256, 512, 1024	32	Integer	Data width on the Stream to Memory Map channel's Stream interface. Must be equal to or less than C_M_AXI_S2MM_DATA_WIDTH.
C_INCLUDE_S2MM_STSFIFO	0, 1	1	Integer	0 = Exclude Stream to Memory Map Status FIFO 1 = Include Stream to Memory Map Status FIFO
C_S2MM_STSCMD_FIFO_DEPTH	1, 4, 8, 16	4	Integer	Depth of Stream to Memory Map Status and Command FIFO. A specified depth of 1 indicates a single register implementation.  <b>Note:</b> If C_S2MM_STSCMD_IS_ASYNC is set to a value of 1 (asynchronous mode), the Command and Status FIFO depth is automatically set to a depth of 16 and this parameter is ignored.
C_S2MM_STSCMD_IS_ASYNC	0,1	0	Integer	0 = S2MM Command and Status Stream interfaces are synchronous to the S2MM Memory Mapped interface (uses same clock) 1 = S2MM Command and Status Stream interfaces are asynchronous to the S2MM Memory Mapped interface (uses different clock)
C_INCLUDE_S2MM_DRE	0, 1	1	Integer	DRE support is only available for AXI4-Stream data widths of 16, 32, and 64 bits. 0 = Exclude Stream to Memory Map Data Realignment engine 1 = Include Stream to Memory Map Data Realignment engine



Table 3-5: Design Parameters (Cont'd)

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
C_S2MM_BURST_SIZE	16, 32, 64, 128, 256	16	Integer	<p>Stream to Memory Map channel maximum allowed burst length (in data beats).</p> <p><b>Note:</b> When the parameter C_S_AXIS_S2MM_TDATA_WIDTH is set to 1024-bit, the S2MM burst length is internally limited to 32 data beats so that the AXI 4K address boundary restriction is not violated. When C_S_AXIS_S2MM_TDATA_WIDTH is set to 512 or 256, the S2MM burst length is internally limited to 64 or 128 respectively.</p>
C_S2MM_BTT_USED	8 to 23	16	Integer	<p>Actual number of bits to be used from the S2MM Command BTT field. The BTT used bits are extracted from the field starting from (C_S2MM_BTT_USED-1) down to bit 0.</p> <p><b>Note:</b> Value assigned to this parameter must be greater than or equal to the value <math>\log_2(C\_S\_AXIS\_S2MM\_TDATA\_WIDTH/8 \times C\_S2MM\_BURST\_SIZE) + 1</math>.</p>
C_S2MM_SUPPORT_INDET_BTT	0, 1	0	Integer	<p>This parameter enables 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's S2MM command input.</p> <p>0 = S2MM indeterminate BTT mode is disabled. 1 = S2MM Indeterminate BTT mode is enabled.</p> <p><b>Note:</b> The features enabled by the parameters C_S2MM_SUPPORT_INDET_BTT and C_S2MM_INCLUDE_SF are mutually exclusive. They cannot both be set simultaneously.</p>

Table 3-5: Design Parameters (Cont'd)

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
C_S2MM_ADDR_PIPE_DEPTH	1 to 30	3	Integer	This parameter specifies the internal S2MM queuing depth used for child command address pipelining. The value controls how many address qualifier sets can be committed (pipelined) to the AXI4 Write Address Channel before the associated write data starts flowing into the S2MM function on the AXI4 Write Data Channel. The effective pipelining that is observed on the AXI4 Write Address Channel is the value assigned to this parameter plus 2.
C_S2MM_INCLUDE_SF	0, 1	1	Integer	<p>This parameter specifies the inclusion/omission of the S2MM (Write) Store and Forward function. If the S2MM Stream Channel data width is less than the Memory Map Data Width, an upsizer function is automatically inserted on the Steam side of the Store and Forward fifo.</p> <p>0 = Omit S2MM GP Store and Forward 1 = Include S2MM GP Store and Forward</p> <p><b>Note:</b> The features enabled by the parameters C_S2MM_SUPPORT_INDET_BTT and C_S2MM_INCLUDE_SF are mutually exclusive. They cannot both be set simultaneously.</p>
<b>FPGA Family Type</b>				
C_FAMILY	virtex6, spartan6, virtex7, kintex7	Virtex-6	string	Specifies the FPGA Family the implementation targets

# Allowable Parameter Combinations

Table 3-6: Allowable Parameter Combination

Parameter Name	Affects Parameter	Relationship Description
<b>MM2S Design Parameters</b>		
C_INCLUDE_MM2S	C_M_AXI_MM2S_ARID, C_M_AXI_MM2S_ID_WIDTH, C_M_AXI_MM2S_ADDR_WIDTH, C_M_AXI_MM2S_DATA_WIDTH, C_M_AXIS_MM2S_TDATA_WIDTH, C_INCLUDE_MM2S_STSFIFO, C_INCLUDE_MM2S_DRE, C_MM2S_STSCMD_FIFO_DEPTH, C_MM2S_STSCMD_IS_ASYNC, C_MM2S_BURST_SIZE, C_MM2S_BTT_USED, C_MM2S_INCLUDE_SF	A value of 0 assigned to the affecting parameter removes the MM2S DataMover block from the implementation. The affected parameters are ignored internally as a result.  Some of affected parameters specify port widths. These parameters should be left at default values to maintain default port width even though they are not used for internal logic implementation.
C_INCLUDE_MM2S	C_M_AXI_MM2S_DATA_WIDTH	A value of 2 assigned to the affecting parameter causes the Basic version of the MM2S DataMover block to be implemented. The affected parameter is limited to a value of 32 or 64 as a result.
C_INCLUDE_MM2S	C_M_AXIS_MM2S_TDATA_WIDTH	A value of 2 assigned to the affecting parameter causes the Basic version of the MM2S DataMover block to be implemented. The affected parameter is limited to a value of 8, 16, 32, or 64 as a result.
C_INCLUDE_MM2S	C_INCLUDE_MM2S_STSFIFO, C_INCLUDE_MM2S_DRE, C_MM2S_STSCMD_FIFO_DEPTH, C_MM2S_STSCMD_IS_ASYNC, C_MM2S_BURST_SIZE, C_MM2S_INCLUDE_SF	A value of 2 assigned to the affecting parameter causes the Basic version of the MM2S DataMover block to be implemented. The affected parameters are ignored internally as a result.
C_M_AXI_MM2S_DATA_WIDTH	C_M_AXIS_MM2S_TDATA_WIDTH	The affected parameter's assigned value cannot exceed the affecting parameter's assigned value.
C_INCLUDE_MM2S_STSFIFO	C_MM2S_STSCMD_FIFO_DEPTH	A value assignment of 0 to the affecting parameter causes the affected parameter to be ignored.

Table 3-6: Allowable Parameter Combination (Cont'd)

Parameter Name	Affects Parameter	Relationship Description
C_MM2S_BURST_SIZE	C_MM2S_BTT_USED	The value assigned to this parameter requires the affected parameter assigned value to be greater than or equal to the value $\log_2(C\_M\_AXIS\_MM2S\_TDATA\_WIDTH / 8 \times C\_MM2S\_BURST\_SIZE)$ .
C_M_AXIS_MM2S_TDATA_WIDTH	C_MM2S_BTT_USED	The value assigned to this parameter requires the affected parameter assigned value to be greater than or equal to the value $\log_2(C\_M\_AXIS\_MM2S\_TDATA\_WIDTH / 8 \times C\_MM2S\_BURST\_SIZE) + 1$ .
<b>S2MM Design Parameters</b>		
C_INCLUDE_S2MM	C_S2MM_ARID, C_M_AXI_S2MM_ID_WIDTH, C_M_AXI_S2MM_ADDR_WIDTH, C_M_AXI_S2MM_DATA_WIDTH, C_S_AXIS_S2MM_TDATA_WIDTH, C_INCLUDE_S2MM_STSFIFO, C_INCLUDE_S2MM_DRE, C_S2MM_STSCMD_FIFO_DEPTH, C_S2MM_STSCMD_IS_ASYNC, C_S2MM_BURST_SIZE, C_S2MM_SUPPORT_INDET_BTT C_S2MM_INCLUDE_SF	A value of 0 assigned to the affecting parameter removes the S2MM DataMover block from the implementation. The affected parameters are ignored internally as a result.  <b>Note:</b> Some affected parameters specify port widths for the S2MM block. These parameters should be left at default values to maintain the default port width even though they are not used for internal logic implementation.
C_INCLUDE_S2MM	C_M_AXI_S2MM_DATA_WIDTH	A value of 2 assigned to the affecting parameter causes the Basic version of the S2MM DataMover block to be implemented. The affected parameter is limited to a value of 32 or 64 as a result.
C_INCLUDE_S2MM	C_S_AXIS_S2MM_TDATA_WIDTH	A value of 2 assigned to the affecting parameter causes the Basic version of the S2MM DataMover block to be implemented. The affected parameter is limited to a value of 8, 16, 32, or 64 as a result.
C_INCLUDE_S2MM	C_INCLUDE_S2MM_STSFIFO, C_INCLUDE_S2MM_DRE, C_S2MM_STSCMD_FIFO_DEPTH, C_S2MM_STSCMD_IS_ASYNC, C_S2MM_BURST_SIZE, C_S2MM_INCLUDE_SF, C_S2MM_SUPPORT_INDET_BTT	A value of 2 assigned to the affecting parameter causes the Basic version of the S2MM DataMover block to be implemented. The affected parameters are ignored internally as a result.

Table 3-6: Allowable Parameter Combination (Cont'd)

Parameter Name	Affects Parameter	Relationship Description
C_M_AXI_S2MM_DATA_WIDTH	C_S_AXIS_S2MM_TDATA_WIDTH	The affected parameter's assigned value cannot exceed the affecting parameter's assigned value.
C_INCLUDE_S2MM_STSFIFO	C_S2MM_STSCMD_FIFO_DEPTH	A value assignment of 0 to the affecting parameter causes the affected parameter to be ignored.
C_S2MM_BURST_SIZE	C_S2MM_BTT_USED	The value assigned to this parameter requires the affected parameter assigned value to be greater than or equal to the value $\log_2(C\_S\_AXIS\_S2MM\_TDATA\_WIDTH/8 \times C\_S2MM\_BURST\_SIZE)$ .
C_S_AXIS_S2MM_TDATA_WIDTH	C_S2MM_BTT_USED	The value assigned to this parameter requires the affected parameter assigned value to be greater than or equal to the value $\log_2(C\_S\_AXIS\_S2MM\_TDATA\_WIDTH/8 \times C\_S2MM\_BURST\_SIZE) + 1$ .

# SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

# Customizing and Generating the Core

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

---

## GUI

The AXI DataMover can be found in **\AXI\_Infrastructure** and also in **Embedded\_Processing\AXI\_Infrastructure\DMA** in the Vivado 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 on **AXI DataMover** to bring up the **AXI DataMover** GUI.

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

In most cases the DataMover can be configured with the option specified on the "Basic Options" tab. All the options available in this GUI are essentially the same as the CORE Generator™ GUI. The GUI has been split into two tabs for sake of simplicity and enhanced for ease-of-use.

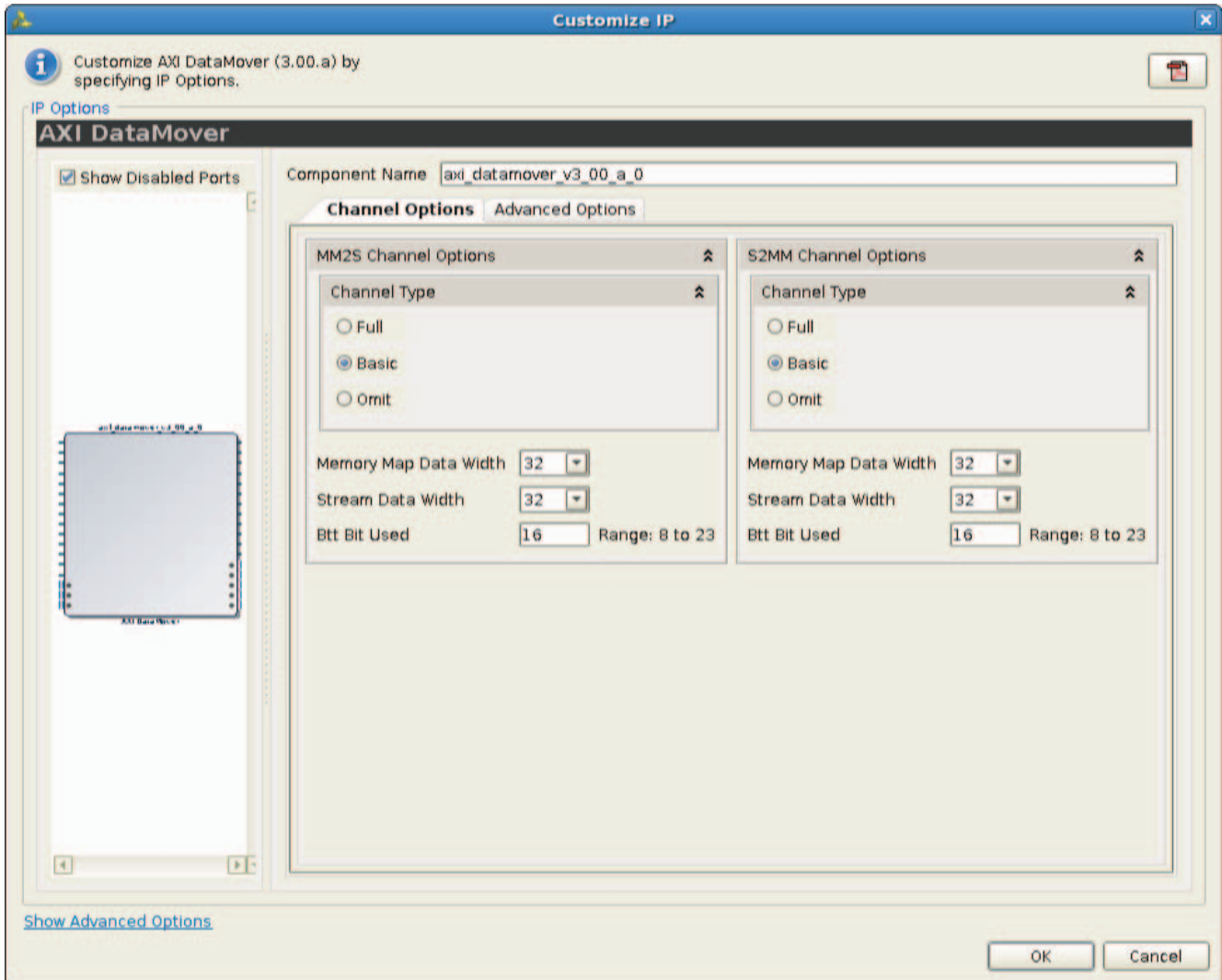


Figure 4-1: AXI DataMover GUI – Channel Options Tab



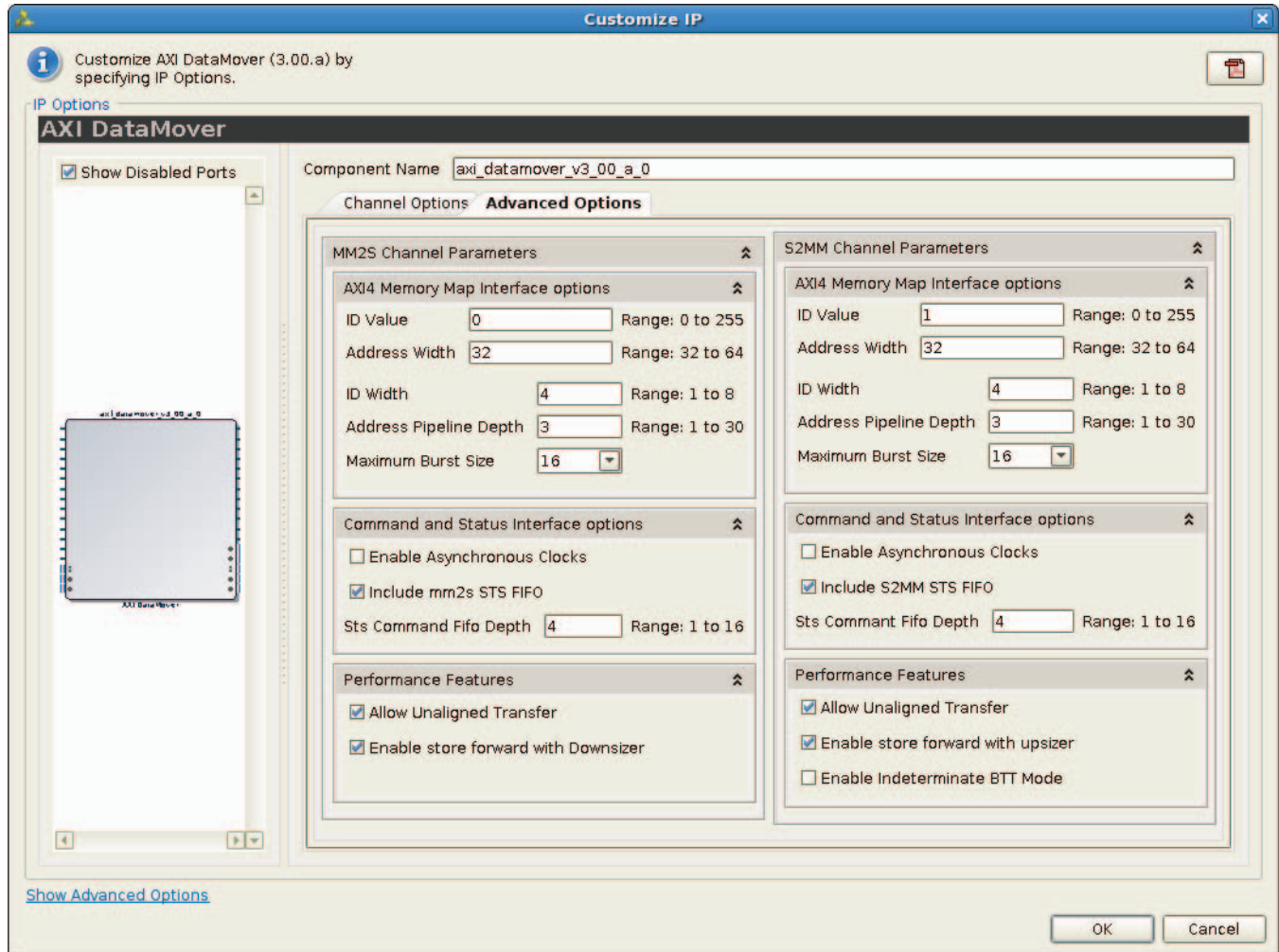


Figure 4-2: AXI DataMover GUI – Advanced Options Tab

**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** – You can choose a Full, Basic, or Omit modes. 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. The Omit mode completely disables the channel.

- **Memory Mapped Data Width** – Specifies the data width in bits of the AXI Memory Mapped Read bus. Valid values are 32, 64, 128, 256, 512, and 1024. 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 1024. This value cannot be more than the Memory Mapped Data Width.
- **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** – The user 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 AXI Memory Mapped Write bus. Valid values are 32, 64, 128, 256, 512, and 1024. The choices vary depending on the Channel Type chosen.
  - **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 1024. This value cannot be more than the Memory Mapped Data Width.
  - **Bytes To Transfer (BTT) Bit Used** – Specifies the valid number of bits in the number of BTT field of the S2MM command. Valid options are 8 to 23.

## 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.
  - **ID Value** – This setting is the ID value of MM2S ID bus and drive onto the MM2S Address Channel ARID output.
  - **Memory Mapped Address Width** – This setting is the address width of the MM2S Channel. AXI DataMover core supports Memory Map address width from 32 bits to 64 bits.
  - **ID Width** – This setting is the bit width of MM2S ID bus.
  - **Address Pipeline Depth** – This setting provides internal MM2S queuing depth used for child command address pipelining. The value controls how many address qualifier sets can be committed (pipelined) to the AXI4 Read Data Channel.

- **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 16, 32, 64, 128, and 256.

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 MM2S Memory Map interface.
- **Include MM2S STS FIFO** – This setting allows you to include or exclude the MM2S Status (STS) FIFO. If the Status FIFO is included, its depth is defined by `C_MM2S_STSCMD_FIFO_DEPTH`.
- **STS Command FIFO Depth** – This setting is the depth of the MM2S Status and Command FIFO. A specified value of 1 indicates a single register implementation. Valid values are 1, 4, 8, and 16. If asynchronous clocks are enabled, the Command and Status FIFO depth is automatically set to a depth of 16 and this option setting is ignored.
- **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 stream data width `C_M_AXIS_MM2S_TDATA_WIDTH`.
- **Enable Store and Forward** – This setting provides the inclusion/omission of the MM2S Store and Forward function. If the MM2S Stream Channel data width is less than the Memory Mapped data width, a downsizer function is automatically inserted on the Stream side of the Store and Forward module.

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.
  - **ID Value** – This setting is the ID value of S2MM ID bus and drive onto the S2MM Address Channel ARID output.
  - **Memory Mapped Address Width** – This setting is the address width of the S2MM Channel. AXI DataMover core supports Memory Map address width from 32 bits to 64 bits.
  - **ID Width** – This setting is the bit width of S2MM ID bus.

- **Address Pipeline Depth** – This setting provides internal S2MM queuing depth used for child command address pipelining. The value controls how many address qualifier sets can be committed (pipelined) to the AXI4 Read Data Channel.
- **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 16, 32, 64, 128, and 256.

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.
- **Include S2MM STS FIFO** – This setting allows you to include or exclude the S2MM Status (STS) FIFO. If the Status FIFO is included, its depth is defined by `C_S2MM_STSCMD_FIFO_DEPTH`.
- **STS Command FIFO Depth** – This setting is the depth of the S2MM Status and Command FIFO. A specified value of 1 indicates a single register implementation. Valid values are 1, 4, 8, and 16. If asynchronous clocks are enabled, the Command and Status FIFO depth is automatically set to a depth of 16 and this option setting is ignored.
- **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 stream data width `C_S_AXIS_S2MM_TDATA_WIDTH`.
- **Enable Store and Forward** – This setting provides the inclusion/omission of the S2MM Store and Forward function. If the S2MM Stream Channel data width is less than the Memory Mapped data width, an upsizer function is automatically inserted on the Stream side of the Store and Forward module.
- **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's S2MM command input.

---

## Core Implementation

### Functional Simulation

VHDL and Verilog source files for `axi_datamover_v3_00_a` are provided un-encrypted for use in behavioral simulation within a simulation environment. Neither a test bench nor test fixture is provided with the AXI DataMover core.

### Synthesis

Synthesis of the AXI DataMover can be performed with Vivado synthesis.

### Xilinx Tools

See the [LogiCORE IP Facts Table](#).

### Static Timing Analysis

Static timing analysis can be performed using trace, following `ngdbuild`, `map`, and `par`.







# Output Generation

The output files generated from the Xilinx Vivado IP catalog are placed in the project directory. The file output list can include some or all of the following files.

The component name of the IP generated is `axi_datamover_v3_00_a_0`.

 **<project directory>/ip**

Top-level project directory; name is user-defined

-  **<project directory>/ip/axi\_datamover\_v3\_00\_a\_0**  
AXI DataMover folders and files
-  **proc\_common\_v3\_00\_a**  
Helper cores
-  **sim**  
Simulation wrapper
-  **synth**  
Synthesis wrapper
-  **axi\_datamover.veo/.vho**  
Instantiation template files
-  **<axi\_datamover\_v3\_00\_a\_0>axi\_datamover\_v3\_00\_a/hdl/src/vhdl**  
AXI DMA RTL files

## <project directory>

The `project` directory contains the `axi_datamover` core RTL files as well as all its helper cores. The `proc_common_v3_00_a` directory contains the helper cores used by the `axi_datamover`.

## <project directory>/ip/axi\_datamover\_v3\_00\_a\_0

The `axi_datamover_v3_00_a_0` name directory contains the following folders and files.

Table 4-1: `axi_datamover_v3_00_a_0` Directory

Name	Description
<code>&lt;project_dir&gt;/ip/axi_datamover_v3_00_a_0</code>	
<code>proc_common_v3_00_a</code>	Helper cores folder
<code>sim</code>	Simulation wrapper folder
<code>synth</code>	Synthesis wrapper folder
<code>axi_datamover.veo/.vho</code>	Instantiation template files

[Back to Top](#)

**<axi\_datamover\_v3\_00\_a\_0>axi\_datamover\_v3\_00\_a/hdl/src/  
vhd1**

The `axi_dma` RTL files are delivered under `/ip/axi_datamover_v3_00_a/hdl/src/vhd1` directory.

# Constraining the Core

There are no applicable constraints for this core in the Vivado™ Design Suite environment.



## SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

# Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the ISE® Design Suite environment.

---

## GUI

The AXI DataMover can be found in AXI Infrastructure in the Xilinx CORE Generator™ GUI View by Function pane.

To access the AXI DataMover, do the following:

1. Open a project by selecting **File** then **Open Project** or create a new project by selecting **File** then **New Project**.
2. With an open project, choose **AXI Infrastructure** in the View by Function pane.
3. Double-click on **AXI DataMover**; this brings up the AXI DataMover GUI.

The AXI DataMover GUI contains one screen ([Figure 6-1](#)) that provides information about the core, allows for configuration of the core, and provides the ability to generate the core.

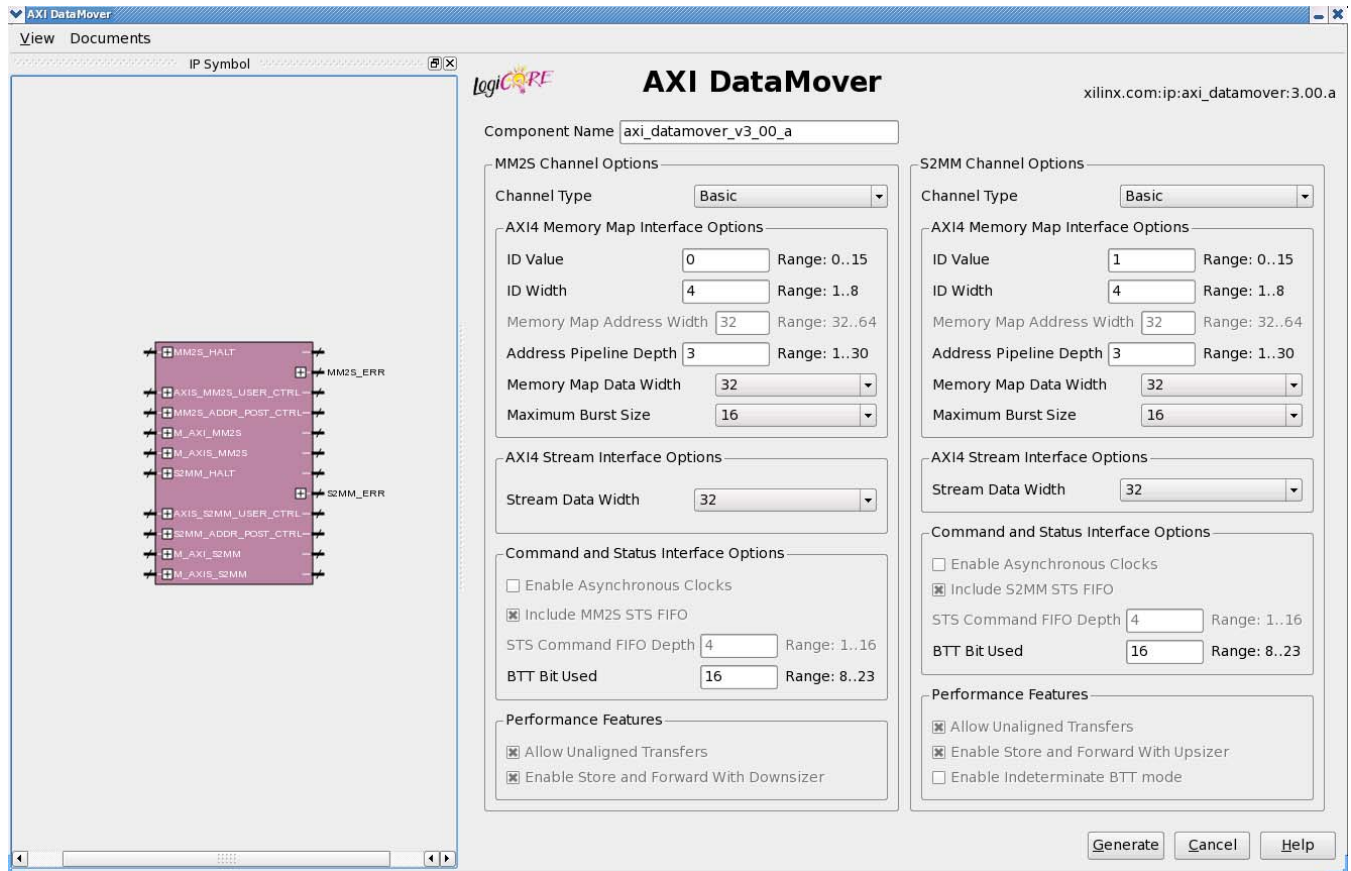


Figure 6-1: AXI DataMover GUI

- **Component Name** – This field contains 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 “\_”.
- **MM2S Channel Options** – The following subsections describe options that affect only the MM2S channel of the AXI DataMover core.
  - **Channel Type** – This option allows you to configure the MM2S channel in Basic, Full mode or to disable the channel. Setting the channel type to Basic disables the channel. Setting channel type to Full includes the Full version of DataMover. Setting the channel type to Omit includes the Basic version of DataMover.
  - **ID Value** – This setting is the ID value of MM2S ID bus and drive onto the MM2S Address Channel ARID output.
  - **ID Width** – This setting is the bit width of MM2S ID bus.
  - **Memory Map Address Width** – This setting is the address width of the MM2S Channel. AXI DataMover core supports Memory Map address width from 32 bits to 64 bits.

- **Address Pipeline Depth** – This setting provides internal MM2S queuing depth used for child command address pipelining. The value controls how many address qualifier sets can be committed (pipelined) to the AXI4 Read Data Channel.
- **Memory Map Data Width** – This setting is the data width in bits of the AXI MM2S Memory Map Read data bus. Valid values are 32, 64, 128, 256, 512, and 1024.
- **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 16, 32, 64, 128, and 256.
- **Stream Data Width** – This setting is the data width in bits of the AXI MM2S Stream data bus. Valid values are 8, 16, 32, 64, 128, 256, 512, and 1024. This value must be less than or equal to the AXI MM2S Memory Map data width.
- **Enable Asynchronous Clocks** – This setting allows you to operate the MM2S Command and Status Stream interface asynchronously with MM2S Memory Map interface.
- **Include MM2S STS FIFO** – This setting allows you to include or exclude the MM2S Status (STS) FIFO. If the Status FIFO is included, its depth is defined by `C_MM2S_STSCMD_FIFO_DEPTH`.
- **STS Command FIFO Depth** – This setting is the depth of the MM2S Status and Command FIFO. A specified value of 1 indicates a single register implementation. Valid values are 1, 4, 8, and 16. If asynchronous clocks are enabled, the Command and Status FIFO depth is automatically set to a depth of 16 and this option setting is ignored.
- **BTT Bit Used** – This setting defines the actual number of BTT bits used by the AXI DataMover core out of the 23-bit BTT field in the DataMover Command. The value assigned to this field must be greater than or equal to the value  $\log_2(\text{Stream Data Width}/8 \times \text{Maximum Burst Size})$ .
- **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 stream data width `C_M_AXIS_MM2S_TDATA_WIDTH`. For example, if `C_M_AXIS_MM2S_TDATA_WIDTH = 32`, data is aligned if it is located at word offsets (32-bit offset), that is, 0x0, 0x4, 0x8, 0xC, and so forth. If `C_M_AXIS_MM2S_TDATA_WIDTH = 64`, data is aligned if it is located at the double-word offset (64-bit offsets), that is, 0x0, 0x8, 0x10, 0x18, and so forth.

For use cases where all transfers are `C_M_AXIS_MM2S_TDATA_WIDTH` aligned, DRE can be disabled to save FPGA resources.

**Note:** Having an unaligned address with the DRE disabled produces undefined results. DRE support is only available for the AXI4-Stream data width setting of 64-bits and less.

- **Enable Store and Forward** – This setting provides the inclusion/omission of the MM2S Store and Forward function. If the MM2S Stream Channel data width is less than the Memory Mapped data width, a downsizer function is automatically inserted on the Stream side of the Store and Forward module.
- **S2MM Channel Options** – The following subsections describe options that affect only the S2MM channel of the AXI DataMover core.
  - **Channel Type** – This option allows you to configure the S2MM channel in Basic, Full mode or to disable the channel. Setting the channel type to Basic disables the channel. Setting channel type to Full includes the Full version of DataMover core. Setting the channel type to Omit includes the Basic version of DataMover core.
  - **ID Value** – This setting is the ID value of S2MM ID bus and drive onto S2MM Address Channel AWID output.
  - **ID Width** – This options sets the bit width of the S2MM ID bus.
  - **Memory Map Address Width** – Address width of the S2MM Channel. The AXI DataMover core supports Memory Map address width from 32 bits to 64 bits.
  - **Address Pipeline Depth** – This setting provides internal S2MM queuing depth used for child command address pipelining. The value controls how many address qualifier set can be committed (pipelined) to the AXI4 Write Data Channel.
  - **Memory Map Data Width** – This option sets the data width in bits of the AXI S2MM Memory Map Write data bus. Valid values are 32, 64, 128, 256, 512, and 1024.
  - **Stream Data Width** – This option sets the data width in bits of the AXI S2MM Stream data bus. Valid values are 8, 16, 32, 64, 128, 256, 512, and 1024. This value must be less than or equal to the AXI S2MM Memory Map data width.
  - **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 16, 32, 64, 128, and 256.
  - **Enable Asynchronous Clocks** – This setting allows you to operate S2MM Command and Status Stream interface asynchronously with S2MM Memory Map interface.

- **Include S2MM STS FIFO** – This setting allows you to include or exclude the S2MM Status FIFO. If the Status FIFO is included, its width is defined by `C_S2MM_STSCMD_FIFO_DEPTH`.
- **STS Command FIFO Depth** – This options sets the depth of the S2MM Status and Command FIFO. A specified value of 1 indicates a single register implementation. Valid values are 1, 4, 8, and 16. If asynchronous clocks are enabled, the Command and Status FIFO depth is automatically set to a depth of 16 and this option setting is ignored.
- **BTT Bit Used** – This setting defines the actual number of BTT bits used by the DataMover core out of the 23-bit BTT field in the DataMover Command. The value assigned to this field must be greater than or equal to the value  $\log_2(\text{Stream Data Width}/8 \times \text{Maximum Burst Size})$ .
- **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 stream data width `C_S_AXIS_S2MM_TDATA_WIDTH`. For example, if `C_S_AXIS_S2MM_TDATA_WIDTH = 32`, data is aligned if it is located at word offsets (32-bit offset), that is, 0x0, 0x4, 0x8, 0xC, and so forth. If `C_S_AXIS_S2MM_TDATA_WIDTH = 64`, data is aligned if it is located at the double-word offset (64-bit offsets), that is, 0x0, 0x8, 0x10, 0x18, and so forth. For use cases where all transfers are `C_S_AXIS_S2MM_TDATA_WIDTH` aligned, DRE can be disabled to save FPGA resources.  
  
**Note:** Having unaligned address with DRE disabled produces undefined results. The DRE support is only available for AXI4-Stream data width setting of 64-bits and less.
- **Enable Store and Forward** – This setting provides the inclusion/omission of the S2MM Store and Forward function. If the S2MM Stream Channel data width is less than the Memory Mapped data width, an upsizer function is automatically inserted on the Stream side of the Store and Forward module.
- **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's S2MM command input.

---

# Core Implementation

## Functional Simulation

VHDL and Verilog source files for `axi_datamover_v3_00_a` are provided un-encrypted for use in behavioral simulation within a simulation environment. Neither a test bench nor test fixture is provided with the AXI DataMover core.

## Synthesis

Synthesis of the AXI DataMover can be performed with XST.

## Xilinx Tools

See the [LogiCORE IP Facts Table](#) table.





## Static Timing Analysis

Static timing analysis can be performed using trace, following `ngdbuild`, `map`, and `par`.

---

# Output Generation

The output files generated from the Xilinx CORE Generator tool are placed in the project directory. The file output list can include some or all of the following files.

-  [<project directory>](#)  
Top-level project directory; name is user-defined
  -  [<project directory>/<axi\\_datamover\\_component name>](#)  
AXI DataMover doc and source files
    -  [<axi\\_datamover\\_component name>/doc](#)  
AXI DataMover solution PDF documentation
    -  [<axi\\_datamover\\_component name>/hdl/src/vhdl](#)  
Source files for AXI DataMover core

## <project directory>

The `project` directory contains templates for instantiation of the core and the xco file.

Table 6-1: Project Directory

Name	Description
<project_dir>	
<axi_datamover_component_name>.xco	Log file from CORE Generator tool describing which options were used to generate the AXI DataMover core. An XCO file is generated by the CORE Generator tool for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator tool.
<axi_datamover_component_name>_flist.txt	A text file listing all of the output files produced when the customized AXI DataMover core was generated in the CORE Generator tool.
<axi_datamover_component_name>.vho	The HDL template for instantiating the AXI DataMover core.
<axi_datamover_component_name_synth>.vhd	The HDL synthesis wrapper file with the modified parameter configuration of AXI DataMover core.
<axi_datamover_component_name_sim>.vhd	The structural simulation model for the AXI DataMover core. It is used for functionally simulating the core.

[Back to Top](#)

## <project directory>/<axi\_datamover\_component name>

The `axi_datamover_component` name directory contains the doc and hdl folder.

## <axi\_datamover\_component name>/doc

The `doc` directory contains the appropriate product guide.

Table 6-2: Doc Directory

Name	Description
<project_dir>/<axi_datamover_component_name>/doc	
axi_datamover_v3_00_a_readme.txt axi_datamover_v3_00_a_readme_vinfo.html	The AXI DataMover release notes and core information file in text and html format.
pg022_axi_datamover.pdf	<i>AXI DataMover Product Guide</i>

[Back to Top](#)



**<axi\_datamover\_component name>/hdl/src/vhdl**

The `hdl/src/vhdl` contains AXI DataMover source files including the `proc_common` library helper files.

# Constraining the Core

There are no applicable constraints for this core in the ISE® Design Suite environment.

## SECTION IV: APPENDICES

Migrating

Debugging

Additional Resources

# Migrating

For information on migrating to the Vivado™ Design Suite, see *Vivado Design Suite Migration Methodology Guide* (UG911).

# Debugging

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

# Additional Resources

---

## Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

[www.xilinx.com/support](http://www.xilinx.com/support).

For a glossary of technical terms used in Xilinx documentation, see:

[www.xilinx.com/company/terms.htm](http://www.xilinx.com/company/terms.htm).

---

## References

To search for Xilinx documentation, go to [www.xilinx.com/support](http://www.xilinx.com/support)

These documents provide supplemental material useful with this product guide:

1. *LogiCORE IP AXI Interconnect Product Guide* (PG059)
2. *Vivado Design Suite Migration Methodology Guide* (UG911)
3. Vivado™ Design Suite documentation:  
[www.xilinx.com/cgi-bin/docs/rdoc?v=2012.3;t=vivado+userguides](http://www.xilinx.com/cgi-bin/docs/rdoc?v=2012.3;t=vivado+userguides)
4. *AMBA® AXI4-Stream Protocol Specification*
5. *ARM® AXI4 Memory Mapped Specification*
6. *ARM AXI4-Stream Interface Specification*

## Technical Support

Xilinx provides technical support at [www.xilinx.com/support](http://www.xilinx.com/support) for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the IDS Embedded Edition Derivative Device Support web page ([www.xilinx.com/ise/embedded/ddsupport.htm](http://www.xilinx.com/ise/embedded/ddsupport.htm)) for a complete list of supported derivative devices for this core.

See the IP Release Notes Guide ([XTP025](#)) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/11	1.0	Initial Xilinx release.
07/11/12	1.1	Template update.
07/25/12	2.0	Updated for Vivado 2012.2, Zynq features, and ISE v14.2 Added Vivado content in Customizing and Generating the Core
10/16/12	2.0.1	<ul style="list-style-type: none"> <li>• Updated for Vivado 2012.3 and ISE v14.3.</li> <li>• Added MM2S and S2MM block Information</li> <li>• Added two figures showing typical use cases for DataMover</li> <li>• Removed AXI Read Master, AXI Write Master sections, AXI DataMover Operation, and Parameter -- I/O Signal Dependencies sections</li> <li>• Added two new sections to Chapter 3: Example DataMover Read(MM2S) Timing Example DataMover Write(S2MM) Timing</li> </ul>

---

## Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA and ARM are registered trademarks of ARM in the EU and other countries. All other trademarks are the property of their respective owners.