LogiCORE IP AXI DataMover v3.00.a

Product Guide

PG022 October 19, 2011

XILINX®

Table of Contents

Appendix 7: Additional Resources

LogiCORE IP AXI DataMover (axi_datamover) v3.00.a

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	
Core Specifics	
Supported Device Family(1)	Virtex®-7, Kintex™-7, Virtex-6, Spartan®-6
Supported User Interfaces	AXI4, AXI4-Stream
Resources	See Table 1-4, Table 1-5, Table 1-6, and Table 1-7.
Provided with Core	
Design Files	VHSIC Hardware Description Language (VHDL)
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Not Provided
Simulation Model	Not Provided
Tested Design Tools	
Design Entry Tools	Integrated Software Environment (ISE®) 13.3 software
Simulation(2)	QuestaSim-64
Synthesis Tools(2)	Xilinx Synthesis Technology (XST)
Support	
Provided by Xilinx @ www.xilinx.com/support	

1. For a complete listing of supported devices, see the release notes for this core.
2. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

XILINX®

Chapter 1

Overview

The AXI DataMover is a key interconnect infrastructure IP that enables high throughput transfer of data between the AXI4 memory-mapped domain to the AXI4-Stream domain. The AXI DataMover provides the MM2S and S2MM AXI4-Stream channels that operate independently in a full duplex like method. The AXI DataMover is a key building block for the AXI DMA core and enables 4 kbyte 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. The AXI DataMover is designed to provide high-speed data transfer between the AXI4 Memory Map and the AXI4-Stream transport environments. The block provides two distinct functional elements that support data transport in two directions. The top functional element is the AXI4 Memory Map to Stream (MM2S) block. This block moves data from the Memory Mapped domain to an AXI4-Stream domain. The bottom functional element is a Stream to AXI4 Memory Map (S2MM) block that moves data from the AXI4-Stream domain to the Memory Mapped domain. Each element is controlled via commands loaded into a Command Register/FIFO. Status information regarding the command execution and completion is relayed back to the user logic via a Status Register/FIFO. Both the Command and Status FIFO are accessed via separate AXI4-Stream interfaces. Each of the MM2S and S2MM elements has its own dedicated Command/Status First In First Out (FIFO) pairs. The MM2S and S2MM elements are completely independent from each other. The detailed port references and groupings are provided in Table 2-1.

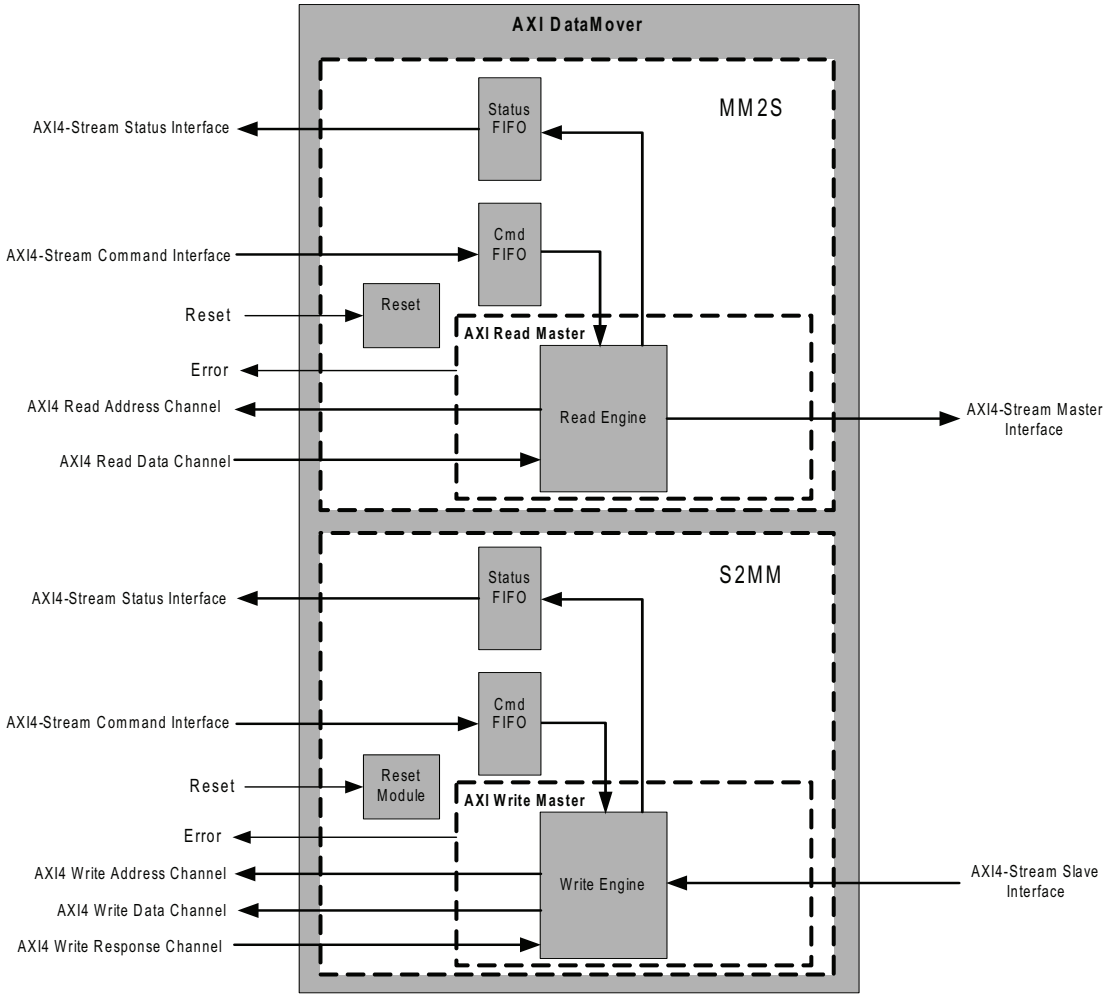

Figure 1-1: **AXI DataMover Block Diagram**

Operating System Requirements

For a list of System Requirements, see the ISE Design Suite 13: 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.

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

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx ISE® Design Suite Embedded Edition software under the terms of the Xilinx End User License. The core is generated using the Xilinx ISE Embedded Development Kit (EDK).

Performance

Maximum Frequencies

The targeted maximum clock frequency for AXI DataMover core are given in Table 1-1.

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

Family	Speed Grade	Fmax
Spartan6	-2	133 MHz
Virtex6	-1	180 MHz

Latency

Table 1-2 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 1-2: **AXI DataMover Latency**

Description	Clocks
MM2S Channel	
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
S2MM Channel	
Initial s_axis_s2mm_tvalid m_axi_s2mm_awalid (C_S2MM_INCLUDE_SF=0)	2
Initial s_axis_s2mm_tvalid m_axi_s2mm_awalid (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 1-3 describes the latency for the AXI DataMover core. The tables provides performance information for a typical configuration. Throughput test consisted of 8 parent command loaded into the AXI DataMover core with each command having BTT value as 1MByte 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 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 1-3: **AXI DataMover Throughput**

AXI DataMover Channel	Primary Clock Frequency	Packet Size	Maximum Total Data Throughput (MBytes/sec)	Percent of Theoretical
Spartan-6 FPGAs				
MM2S	100	1 MByte	391.27	97.75%
S2MM	100	1 MByte	391.27	97.75%
Virtex-6 FPGAs				
MM2S	150	1 MByte	587.81	97.96%
S2MM	150	1 MByte	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 1-4, Kintex™-7 FPGA in Table 1-5, Virtex-6 FPGA in Table 1-6, and Spartan®-6 FPGA in Table 1-7.

Table 1-4: **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
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

Table 1-4: **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
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 1-5: **Kintex-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	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
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

Table 1-5: **Kintex-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	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 1-6: **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

Table 1-6: **Virtex-6 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	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 1-7: **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

XILINX®

Chapter 2

Core Interfaces

This chapter provides detailed descriptions for each interface.

Port Descriptions

The AXI DataMover Input/Output (I/O) signals are described in Table 2-1.

Table 2-1: **I/O Signal Description**

Signal Name	Interface	Signal Type	Init Status	Description
Memory Map to Stream Clock and Reset				
m_axi_mm2s_aclk	MM2S	Input	-	Master Clock for the MM2S synchronization
m_axi_mm2s_aresetn	MM2S	Input	-	Master Reset for the MM2S logic. Active low assertion sensitivity. ***Note:*** Must be asserted for three clock periods of m_axi_mm2s_aclk. See Reset Assertion Timing.
Memory Map to Stream Soft Shutdown Control				
mm2s_halt	MM2S	Input	-	Active high input signal requesting that the MM2S function perform a soft shutdown and stop. All MM2S transfers committed on the MM2S Address channel are completed and the associated data received on the MM2S input Data Channel is discarded. See DataMover Soft Shutdown (Halt) Request Operations.
mm2s_halt_cmplt	MM2S	Output	Zero	Active high output signal indicating that the MM2S function has completed a soft shutdown and is stopped. All MM2S transfers committed on the MM2S Address channel are completed and the associated data received on the MM2S input Data Channel is discarded. See DataMover Soft Shutdown (Halt) Request Operations.

Table 2-1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
Memory Map to Stream Error Detect Discrete				
mm2s_err	MM2S	Output	Zero	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 zero. This bit is a "sticky" error indication; after being set it requires an assertion of the m_axi_mm2s_aresetn signal to clear it.
Memory Map to Stream Debug Support				
mm2s_dbg_sel(3:0)	MM2S	Input	-	Reserved for internal Xilinx use.
mm2s_dbg_data(31:0)	MM2S	Output	BEEF0000 (if Omit MM2s) BEEF1111 (if Full MM2s) BEEF2222 (if Basic MM2s)	Reserved for internal Xilinx use.
Memory Map to Stream Address Posting Controls				
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. This is used in cases where an address qualifier set cannot be posted unless the receiving user logic on the output of the MM2S Stream interface can accept the read data without throttling the Stream interface This can cause throttling on the AXI4 Read Data Channel).
mm2s_addr_req_posted	MM2S	Output	Zero	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.
mm2s_rd_xfer_cmplt	MM2S	Output	Zero	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.

Table 2-1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
Memory Map to Stream Read Address Channel				
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	Zeros	MM2S Read Address
m_axi_mm2s_arlen (7:0)	MM2S	Output	Zeros	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	Zeros	MM2S Read Size Qualifier
m_axi_mm2s_arburst (1:0)	MM2S	Output	Zeros	MM2S Read Burst Type Qualifier. This is always set to Incrementing Burst type ("01").
m_axi_mm2s_arprot (2:0)	MM2S	Constant Output	Zeros	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 zeros.
m_axi_mm2s_arvalid	MM2S	Output	Zero	MM2S Read Address Valid Qualifier
m_axi_mm2s_arready	MM2S	Input	-	MM2S Read Address Ready Status (from Slave)
Memory Map to Stream Read Data Channel				
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	Zero	MM2S Read Ready handshake output
Memory Map to Stream Master Stream Channel				
m_axis_mm2s_tvalid	MM2S Stream	Output	Zero	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	Zeros	MM2S Stream Data
m_axis_mm2s_tkeep ((C_M_AXIS_MM2S_TDATA_WIDTH/8)-1:0)	MM2S Stream	Output	Zeros	MM2S Stream Strobes

Table 2-1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
m_axis_mm2s_tlast	MM2S Stream	Output	Zero	MM2S Stream Last indication
Memory Map to Stream Command/Status Channel Asynchronous Clock and Reset				
m_axis_mm2s_ cmdsts_aclk	MM2S Command & Status	Input	-	MM2S Command Interface Clock. ***Note:*** 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). ***Note:*** 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. ***Note:*** Must be asserted for three clock periods of m_axis_mm2s_cmdsts_aclk. See Reset Assertion Timing.
Memory Map to Stream Command Channel (Slave Stream)				
s_axis_mm2s_cmd_ tvalid	MM2S Command	Input		MM2S Command Valid handshake
s_axis_mm2s_cmd_ tready	MM2S Command	Output	Zero	MM2S Command Ready handshake
s_axis_mm2s_cmd_ tdata((C_M_AXI_MM2S _ADDR_WIDTH+40)-1: 0)	MM2S Command	Input	-	MM2S Command Data
Memory Map to Stream Status Channel (Master Stream)				
m_axis_mm2s_sts_ tvalid	MM2S Status	Output	Zero	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	One	Driven to One

Table 2-1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
m_axis_mm2s_sts_ tlast	MM2S Status	Constant Output	One when MM2S is present, else Zero when omitted	Driven to a constant One when MM2S is present, else Zero when omitted
Stream to Memory Map Clock and Reset				
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). ***Note:*** Must be asserted for three clock periods of m_axi_s2mm_aclk. See Reset Assertion Timing.
Stream to Memory Map Soft Shutdown Control				
s2mm_halt	S2MM	Input	-	Active high input signal requesting that the S2MM function perform a soft shutdown and stop. All S2MM transfers committed on the S2MM Address channel are complete and the associated data transmitted on the S2MM output Data Channel has bogus values. See DataMover Soft Shutdown (Halt) Request Operations.
s2mm_halt_cmplt	S2MM	Output	Zero	Active high output signal indicating that the S2MM function has completed a soft shutdown and is stopped. All S2MM transfers committed on the S2MM Address channel are completed and the associated data transmitted on the S2MM output Data Channel are bogus. See DataMover Soft Shutdown (Halt) Request Operations.
Stream to Memory Map Error Detect Discrete				
s2mm_err	S2MM	Output	Zero	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 zero 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.
Stream to Memory Map Debug Support				
s2mm_dbg_sel(3:0)	S2MM	Input	-	Reserved for internal Xilinx use.
s2mm_dbg_data(31:0)	S2MM	Output	CAFE0000 (if Omit S2MM) CAFE1111 (if Full S2MM) CAFE2222 (if Basic S2MM)	Reserved for internal Xilinx use.

Table 2-1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
Stream to Memory Map Address Posting Controls				
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. This is used in cases where an address qualifier set cannot be posted unless the transmitting user logic on the input of the S2MM Stream interface can provide the write data without throttling the Stream interface. ***Note:*** This in turn can cause throttling on the AXI4 Write Data Channel.)
s2mm_addr_req_posted	S2MM	Output	Zero	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	Zero	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	Zero	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	Zeros	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.
Stream to Memory Map Write Address Channel				
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	Zeros	S2MM Write Address
m_axi_s2mm_awlen (7:0)	S2MM	Output	Zeros	S2MM Write Length Qualifier ***Note:*** 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.

Table 2-1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
m_axi_s2mm_awsize (2:0)	S2MM	Output	Zeros	S2MM Write Qualifier
m_axi_s2mm_awburst (1:0)	S2MM	Output	Zeros	S2MM Write Burst Type Qualifier. This is always set to Incrementing Burst type. ('"01")
m_axi_s2mm_awprot (2:0)	S2MM	Constant Output	Zeros	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 zeros.
m_axi_s2mm_awvalid	S2MM	Output	Zero	S2MM Write Address Valid Qualifier
m_axi_s2mm_awready	S2MM	Input	-	S2MM Write Address Ready Status (from Slave)
Stream to Memory Map Write Data Channel				
m_axi_s2mm_wdata (C_M_AXI_S2MM_ DATA_WIDTH-1:0)	S2MM	Output	Zeros	S2MM Write Data
m_axi_s2mm_wstrb ((C_M_AXI_S2MM_ DATA_WIDTH/8)-1:0)	S2MM	Output	Zeros	S2MM Write Strobes
m_axi_s2mm_wlast	S2MM	Output	Zero	S2MM Write Last indication
m_axi_s2mm_wvalid	S2MM	Output	Zero	S2MM Write Valid handshake output
m_axi_s2mm_wready	S2MM	Input	-	S2MM Write Ready handshake input
Stream to Memory Map Write Response Channel				
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	Zero	S2MM Write Ready handshake output
Stream to Memory Map Slave Stream Channel				
s_axis_s2mm_tvalid	S2MM Stream	Input	-	S2MM Stream Valid handshake in
s_axis_s2mm_tready	S2MM Stream	Output	Zero	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 2-1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
Stream to Memory Map Command/Status Channel Asynchronous Clock and Reset				
m_axis_s2mm_cmdsts_awclk	S2MM Command & Status	Input	-	S2MM Command Interface Clock. ***Note:*** 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). ***Note:*** 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. ***Note:*** Must be asserted for three clock periods of m_axis_s2mm_cmdsts_awclk. See Reset Assertion Timing
Stream to Memory Map Command Channel (Slave Stream)				
s_axis_s2mm_cmd_tvalid	S2MM Command	Input	-	S2MM Command Valid handshake
s_axis_s2mm_cmd_tready	S2MM Command	Output	Zero	S2MM Command Ready handshake
s_axis_s2mm_cmd_tdata((C_M_AXI_S2MM_ADDR_WIDTH+32)-1:0)	S2MM Command	Input	-	S2MM Command Data
Stream to Memory Map Status Channel (Master Stream)				
m_axis_s2mm_sts_tvalid	S2MM Status	Output	Zero	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	Ones	S2MM Status Strobes always driven to ones
m_axis_s2mm_sts_tlast	S2MM Status	Constant Output	One when S2MM is present, else Zero when omitted	Driven to a constant One when S2MM is present, else Zero when omitted

XILINX®

Chapter 3

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the core.

The AXI DataMover can be found in **AXI Infrastructure** in the Xilinx® CORE Generator™ software graphical user interface (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.

CORE Generator Software Parameter Screen

The AXI DataMover GUI contains one screen (Figure 3-1) that provides information about the core, allows for configuration of the core, and provides the ability to generate the core.

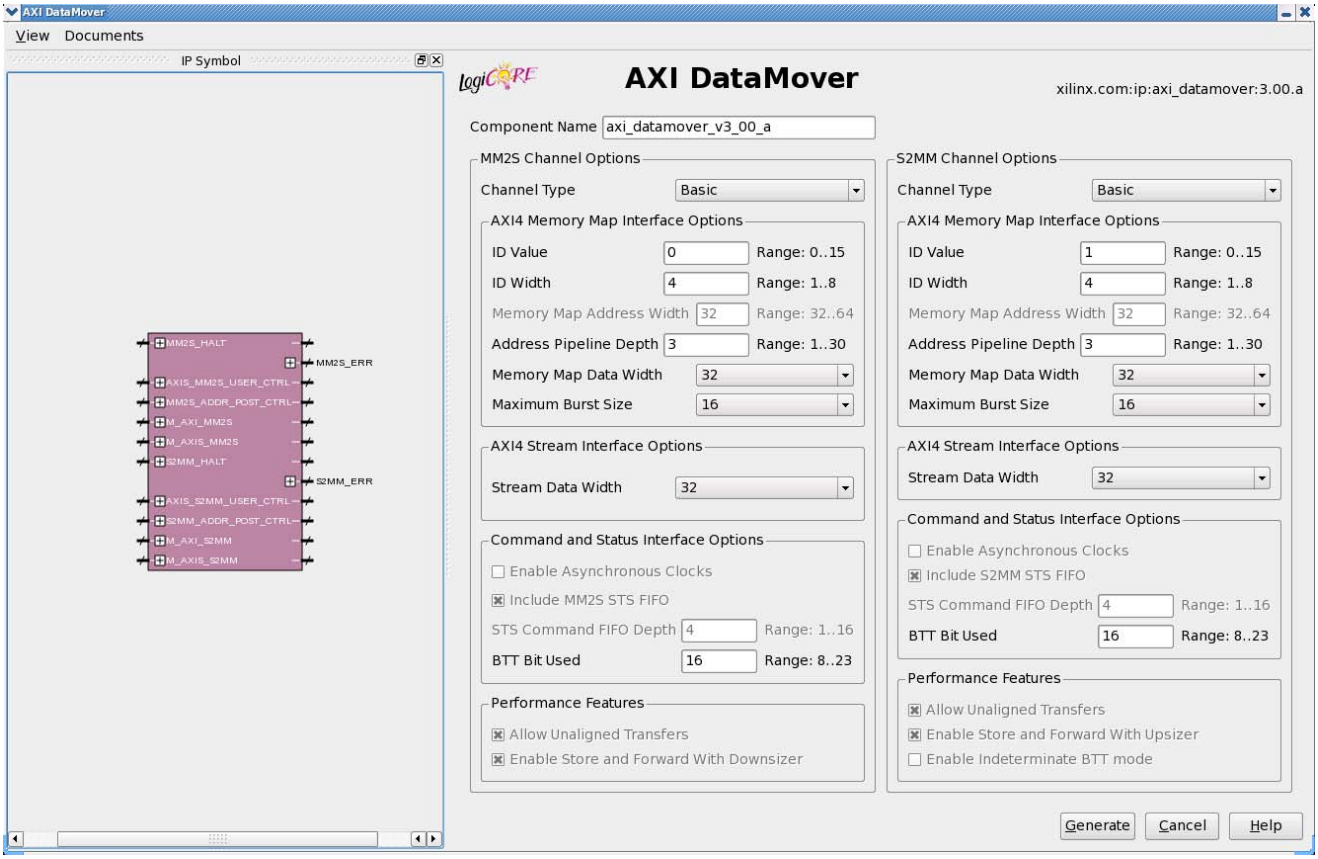

Figure 3-1: **AXI DataMover GUI**

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 log2(Stream Data Width/8 X 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 in 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 log2(Stream Data Width/8 X 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.

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 software are placed in the project directory. The file output list may include some or all of the following files.

Directory Hierarchy

<project directory>

Top-level project directory for the CORE Generator software

<project directory>/<axi_datamover_component name>

Contains the AXI DataMover doc and source files.

<axi_datamover_component name>/doc

Contains the AXI DataMover solution PDF documentation.

<axi_datamover_component name>/hdl/src/vhdl

Contains the source files for AXI DataMover core.

File Details

<project directory>

This is the top-level file. It contains templates for instantiation the core and the xco file.

Table 3-1: **Project Directory**

Name	Description
<project_dir>	
<axi_datamover_component_name>.xco	Log file from CORE Generator software describing which options were used to generate the AXI DataMover core. An XCO file is generated by the CORE Generator software 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 software.
<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 software.
<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>

This directory contains the doc and hdl folder.

<axi_datamover_component name>/doc

This directory contains the appropriate product guide.

Table 3-2: **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	The AXI DataMover Product Guide.

Back to Top

<axi_datamover_component name>/hdl/src/vhdl

This directory contains AXI DataMover source files including the proc_common library helper files.

XILINX®

Chapter 4

Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

General Design Guidelines

Command Interface

The DataMover operations are controlled via 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 (via 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 4-1 and detailed in Table 4-1. 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 (7FFFFF hex bytes). A command loaded into the command interface is often referred to as the parent command of a transfer. The DataMover automatically breaks up large transfers into intermediate bursts (child transfers) that comply with the AXI4 Memory Mapped protocol requirements.

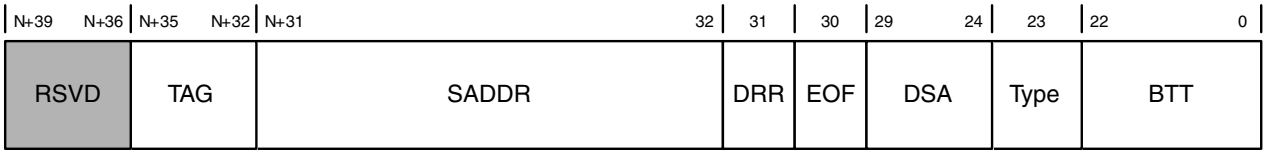

Figure 4-1: **Command Word Layout**

Table 4-1: **Command Word Description**

Bits	Field Name	Description
(N+39) - (N+36) (1)	RSVD	**Reserved** This field is reserved to pad the command width to an even multiple of 8 bits (required for AXI4-Stream interfaces).
(N+35) - (N+32) (1)	TAG	**Command TAG** 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.
(N+31) -32 (1)	SADDR	**Start Address** This field indicates the starting address to use for the Memory Mapped side of the transfer requested by the command. If DRE is enabled, the lower order address bits of this field indicate the starting alignment to load on the Memory Mapped side of the DRE.
31	DRR	**DRE ReAlignment Request** This bit is only used if the optional DRE is included via 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 LS bits of the SADDR field.
30	EOF	**End of Frame** This bit indicates that the command is an End of Frame command. This generally affects the MM2S element (Read Master) because it causes the Stream output logic to assert the TLAST output on the last data beat of the last transfer needed to complete the command. *If DRE is included*, this also causes the DRE to Flush out any intermediate data at the conclusion of the last transfer of the command and submit it to the Stream output (in the case of the MM2S Read Master) or to the AXI Write Data Channel (in the case of the S2MM Write Master).
29-24	DSA	**DRE Stream Alignment** This field is only used by the MM2S and if the optional MM2S DRE is included via 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 (the LS Byte) is the reference byte lane; a value of 1 indicates byte lane 1, and so on. Valid values are dependent upon the parameterized data width of the Stream data Channel. For example, a 32-bit wide data channel has only 4-byte lane positions and thus the DSA field can only have values of 0 to 3. ***Note:*** DRE alignment on the associated Memory Mapped side is derived from the LS bits of the SADDR value of the command.
23	Type	**Reserved** This 1-bit field is currently reserved and ignored by the DataMover.

Table 4-1: **Command Word Description *(Cont'd)***

Bits	Field Name	Description
22-0	BTT	**Bytes to Transfer** 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 zero 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.

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. See the parameters C_MM2S_STSCMD_FIFO_DEPTH and C_S2MM_STSCMD_FIFO_DEPTH. If the parameter value is set to 1, the FIFO is reduced to a simple register implementation. The user interface for the Command FIFO is an AXI Slave Stream interface. The Command Interface only uses the basic stream signals TREADY, TVALID, TDATA. The TLAST and TSTRB signals are ignored.

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, the user 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's clocking frequency. The selection of synchronous or asynchronous is made via the C_MM2S_STSCMD_IS_ASYNC and C_S2MM_STSCMD_IS_ASYNC parameters assignments (0 = synchronous, 1 = asynchronous). In this mode, the Command Interface derives its synchronization from the associated Memory Mapped Interface clock. The reset is also derived from the associated Memory Mapped reset. The separate Command/Status Interface clock and reset inputs are ignored in synchronous mode.

When the Asynchronous Command FIFO is selected, the Command FIFO is implemented as an async FIFO with two clocks and an async reset/init port. The input side clocking is derived from the Command Interface input clock. The read side of the FIFO is clocked by the associated Memory Mapped Interface clock. The FIFO is reset/initialized by either an assertion of the Command Interface reset or the Memory Mapped Interface reset.

Command Load Timing via the Command Stream Interface

Loading a command into the Command FIFO is mechanized via 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 4-2. 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. If either TVALID or TREADY are deasserted, then throttling is occurring and the Command data must be held on the TDATA bus until both TVALID and TREADY are asserted at a rising clock edge (see the CMD3 load cycle timing in Figure 4-2).

The Asynchronous FIFO case is exactly the same except that the synchronization clock is not the Memory Mapped Address and Data Channel clock but the Command Interface input clock.

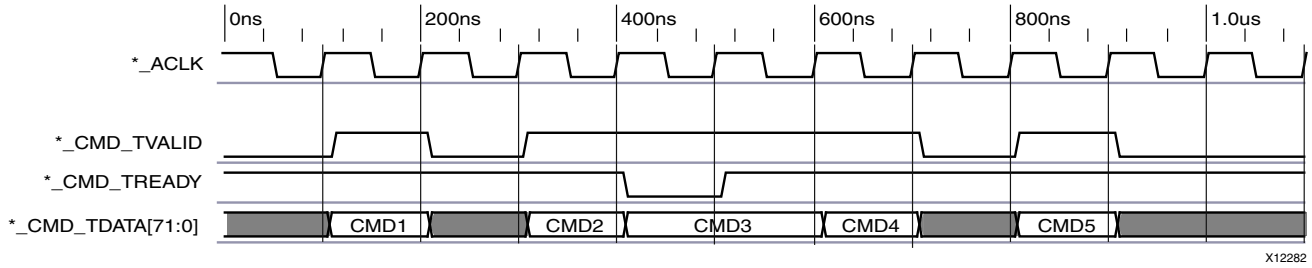

Figure 4-2: **Loading Commands via the Command Interface**

Status Interface

The status of the DataMover's transfer operations are provided via 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). The format of the Status word is shown in Figure 4-3 and detailed in Table 4-2. It is the same for either the MM2S or S2MM DataMover elements.

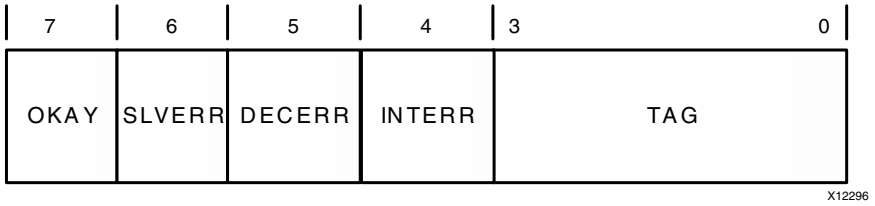

Figure 4-3: **Normal Status Word Layout**

Table 4-2: **Status Word Details**

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

Table 4-2: **Status Word Details *(Cont'd)***

Bits	Field Name	Description
5	DECERR	**Decode Error** Indicates the DataMover Element encountered an address decode error condition for the associated command. This is received via 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	**Internal Error** 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. '0' = No Error '1' = Internal Error Condition
3-0	TAG	**TAG** This 4-bit field echoes the value of the TAG field of the associated input Command whose completion generated the Status.

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 the case where the S2MM is commanded (via 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 4-4 and detailed in Table 4-3. This status format does not apply to the MM2S DataMover status interface.

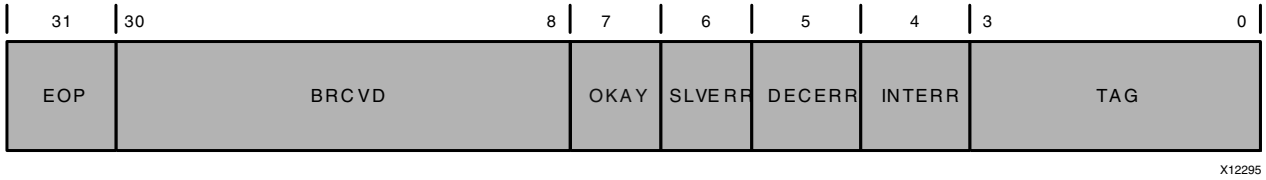

Figure 4-4: **Special S2MM Status Word Layout (IBTT Mode Enabled)**

Table 4-3: **Special S2MM Status Word Details (Indeterminate BTT Mode Enabled)**

Bits	Field Name	Description
31	EOP	**End of Packet** This bit indicates that the S2MM Stream input received a TLAST assertion during the execution of the DataMover command associated with the Status Word. This is not an error condition. It is needed by certain Users (that is, Scatter Gather Engines) to identify the actual End of Packet for the input Stream vs the theoretical maximum that could occur.
30 - 8	BRCVD	**Bytes Received** This field indicates the actual number of bytes received on the Stream interface at the point where s_axis_s2mm_tlast was asserted by the Stream Master.
7	OKAY	**Transfer OKAY** This bit indicates that the associated transfer command has been completed with the OKAY response on all intermediate transfers. '0' = Command had a non-OKAY response during all associated transfers '1' = Command had a OKAY response during all associated transfers
6	SLVERR	**Slave Error** Indicates the DataMover Element encountered a Slave reported error condition for the associated command. This is received via the Response inputs from the AXI4 Memory Mapped interface. '0' = No Error '1' = Slave Asserted Error Condition
5	DECERR	**Decode Error** Indicates the DataMover Element encountered an address decode error condition for the associated command. This is received via 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	**Internal Error** 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 TBD at this point. '0' = No Error '1' = Internal Error Condition
3-0	TAG	**TAG** This 4-bit field echoes the value of the TAG field of the associated input Command whose completion generated the Status.

Status FIFO

The Status Interface of a DataMover 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. See the parameters C_MM2S_STSCMD_FIFO_DEPTH and C_S2MM_STSCMD_FIFO_DEPTH. Status values have a one-to-one correlation to loaded commands via the Command Interface. If the FIFO depth parameter value is set to 1, the Status FIFO is reduced to a simple register implementation. The user interface for the Status FIFO is an AXI Slave Stream interface. The Status Interface only uses the basic stream signals TREADY, TVALID, TDATA. The TLAST and TSTRB signals are driven to constant logic high (ones).

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, the user 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's clocking frequency.

The selection of synchronous or asynchronous mode is made via the C_MM2S_STSCMD_IS_ASYNC and C_S2MM_STSCMD_IS_ASYNC parameters assignments (0 = synchronous, 1 = asynchronous). In this mode, the Status Interface derives its synchronization from the associated Memory Mapped Interface clock. The reset is also derived from the associated Memory Mapped reset. The separate Command/Status Interface clock and reset inputs are ignored in synchronous mode.

When the asynchronous Status Interface is selected, the Status FIFO is implemented as an asynchronous FIFO with independent clocks for reading and writing and an asynchronous reset/init port. The input side (write side) clocking is derived from the Command/Status Interface input clock. The read side of the FIFO is clocked by the associated Memory Mapped Interface clock. The FIFO is reset/initialized by either an assertion of the separate Command/Status Interface reset or the Memory Mapped Interface reset.

Status Read Timing via the Status Stream Interface

Reading a status word from the Status FIFO is mechanized via a single AXI4-Stream data beat. An example of reading five status entries from the MM2S Status FIFO is shown in Figure 4-5. In this scenario, the Status FIFO is synchronous to the Memory Mapped Address and Data Channel clock. The example illustrates that a Status word is considered read from the Status FIFO only when both the TVALID and TREADY handshake signals are both asserted at the rising edge of the synchronizing clock. If either TVALID or TREADY are deasserted, throttling is occurring and the Status data is held on the TDATA output bus until both TVALID and TREADY are asserted at a rising clock edge. See the Stat3 read cycle timing in Figure 4-5.

The Asynchronous FIFO case is exactly the same except that the synchronization clock is not the Memory Mapped Address and Data Channel clock but the Command/Status Interface input clock.

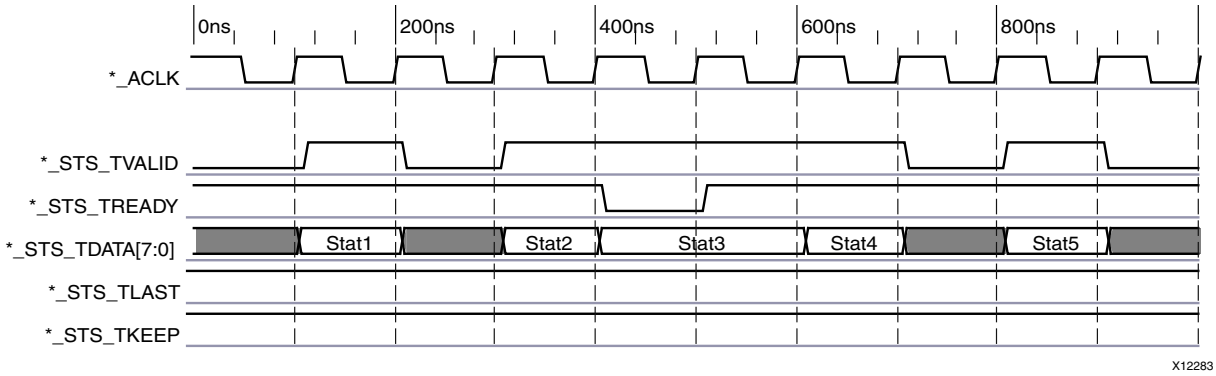

Figure 4-5: **Reading Status via the Status Interface**

AXI Read Master

The AXI Read Master block diagram is shown in Figure 4-6. The Read Master fetches the loaded transfer commands from the MM2S Command FIFO. It calculates the required Address and transfer qualifiers for reading data from the Memory Mapped side, posts the requests on the Read Address Channel, and merges the incoming data from the Read Data Channel to the corresponding output Stream interface.

The Read Master is designed to support AXI overlapped transfers. This is the scenario where address and qualifiers are presented to the Address Channel outputs in parallel with ongoing Data Channel operations. This allows for seamless transitions (no bubbles) in the Data Channel operation from one read request to the next. Overlapped transfers are required to achieve the optimum performance and utilization of the AXI interconnect and AXI Masters and Slaves.

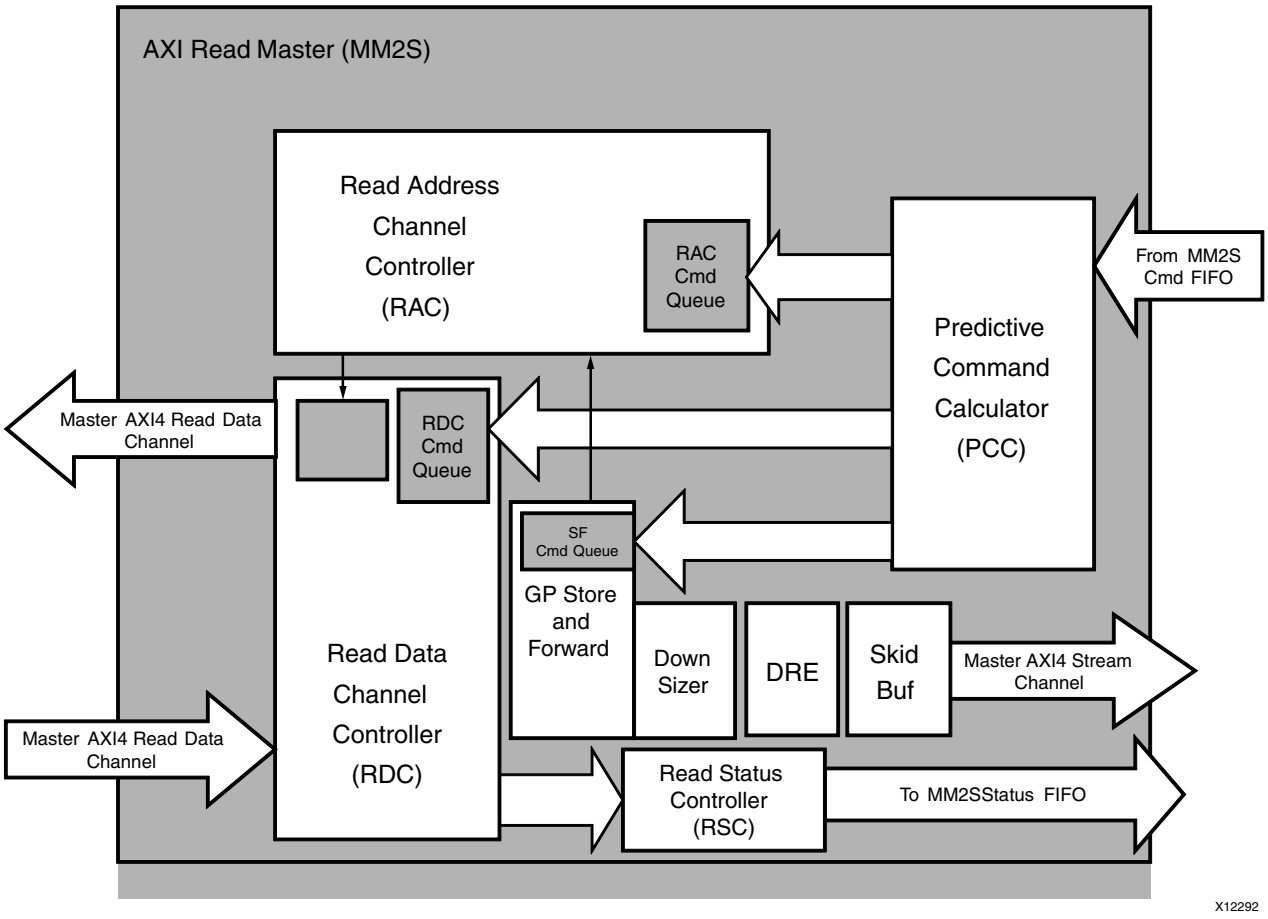

Figure 4-6: **AXI Read Master Block Diagram**

The primary block within the Read Master is the Predictive Command Calculator (PCC). It is responsible for fetching a command from the Command FIFO, decoding it, and then calculating the ensuing AXI transfer(s) needed to complete the command. The PCC uses predictive calculations to generate address/qualifiers in a sequential manner for the AXI4 Memory Mapped Address Channel ahead of when the associated data transfer on Data Channel occurs. The calculations include the starting address for the transfer, the number of data beats (burst length) to use for the transfer, and transfer width. The DataMover only produces Incrementing burst type (does not support fixed or wrapping). In addition, the PCC has to account for any potential 4K byte address boundary roll-overs and, if needed, split transfers to avoid a 4K rollover. When a transfer calculation has completed, the PCC loads the required control information for the Address Channel into the PCC Address Control FIFO and the PCC Data Control FIFO.

When an Address/Qualifier data set has been loaded into the Read Address Channel Controller's Command Queue, the Read Address Channel Controller pops the information from the FIFO and posts it on the AXI4 Address Channel as a read transfer request to the AXI Interconnect. As such, the Address Controller is responsible for the mechanization of the required AXI4 addressing protocol for the Read Master.

When a specific Read request has been posted on the AXI4 Address Channel and accepted by the Interconnect, the Address Controller will fetch and post the next available address and qualifier data set from the Read Address Channel Controller Command Queue if available.

The Read Data Channel Controller works in parallel with the Read Address Controller. It pops the data channel control set from the Read Data Channel Controller Command Queue. This data set is used to set up the Data Channel in preparation for the ensuing read data coming in from the Interconnect (as a result of an Address Channel address cycle completion). The Read Data Channel Controller is also responsible for controlling the optional DRE block and merging the incoming Read Data from the Memory Mapped Read data Channel to the Read Master's output Stream interface.

An optional Data Re-alignment Engine (DRE) can be inserted in the datapath to compensate for a mis-match in address/byte-lane alignment between the Memory Mapped side and the desired byte-lane alignment on the Stream side. The DRE alignment mapping is set up whenever a Read Command is processed with the DRR bit set to '1'. The DRE then retains the sequential alignment information internally (as data is streamed through it). The Read Master DRE is commanded to flush out intermediate data to the Stream Interface when the last data beat of the last AXI4 Read operation occurs that was calculated from a Command with the EOF bit set.

The Read Master incorporates a Skid Buffer to provide output registering of the Stream interface signals and still maintains the TREADY/TVALID throttling capability. The Skid Buffer is discussed in more detail in a later section. See Skid Buffer.

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 (this is the default). The MM2S Store and Forward block is coupled to the Read Address Controller so 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 associated data. The Store and Forward is also fitted with a downsizer feature that is automatically enabled if the AXI4 Memory Mapped data width is wider than the Stream Channel data width. The depth of the MM2S Store and Forward data FIFO is set by the following calculation:

((C_MM2S_ADDR_PIPE_DEPTH+2) * C_MM2S_BURST_SIZE) rounded up to the next power of 2.

AXI Write Master

The AXI Write Master block diagram is shown in Figure 4-7 for normal operation mode with general purpose store-and-forward enabled. The Write Master fetches the loaded transfer commands from the S2MM Command FIFO. It calculates the required Address and transfer qualifiers for writing data to the Memory Mapped side, posts the requests on the Write Address Channel, and merges the incoming data from the Write Stream Channel to the corresponding Write Data Channel interface.

Just like the Read Master, the Write Master is designed to support AXI4 overlapped transfers. This is the scenario where address and qualifiers are presented to the Address Channel outputs in parallel with ongoing Data Channel operations.

This allows for seamless transitions (no bubbles) in the Data Channel operation from one write request to the next. Overlapped transfers are required to achieve the optimum performance and utilization of the AXI interconnect and AXI4 Masters and Slaves.

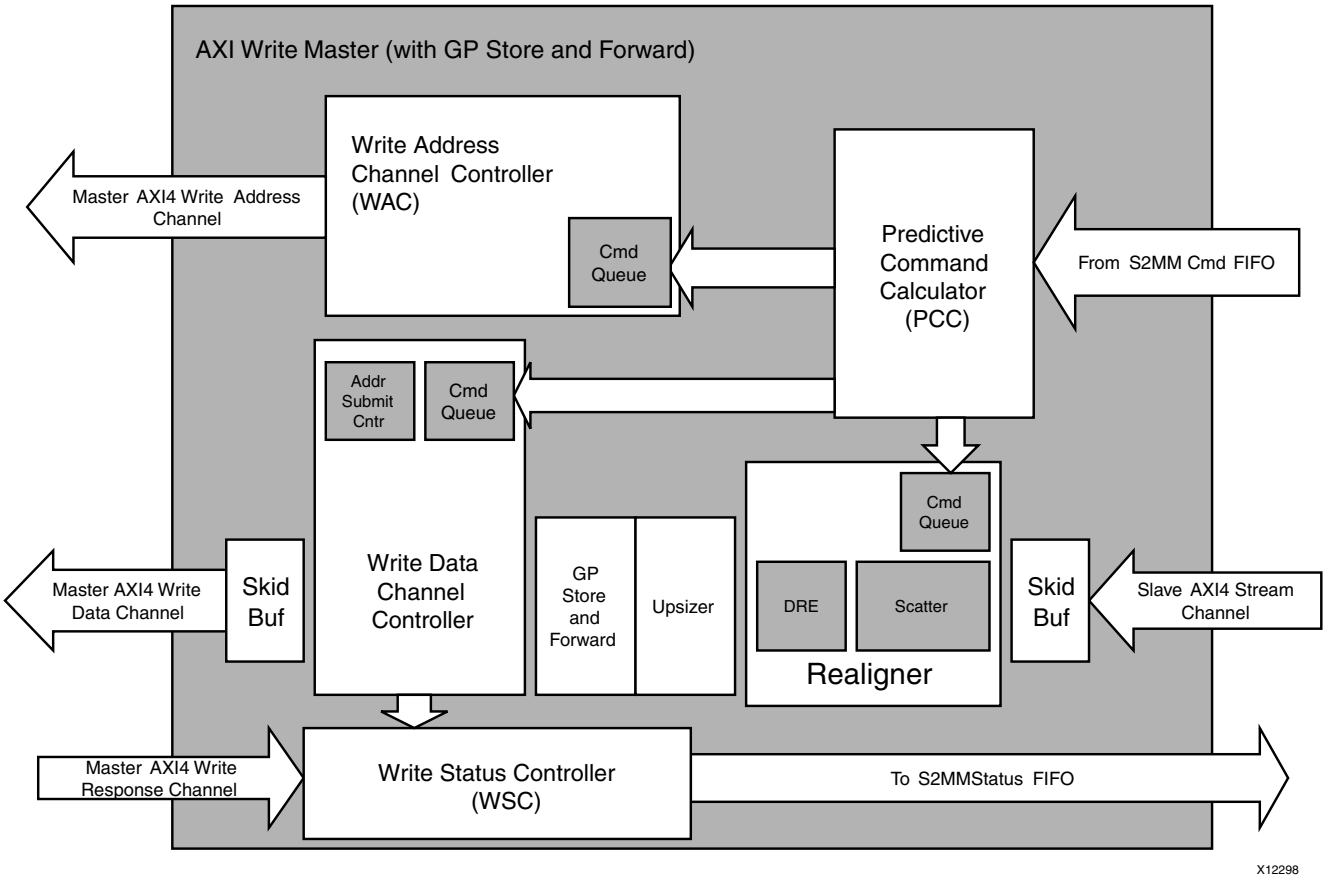

Figure 4-7: **AXI S2MM (Write) Master Block Diagram (Normal Mode)**

The primary block within the Write Master is the Predictive Command Calculator (PCC). This is the same design as that used for the Read Master. It is responsible for fetching a command from the Write Command FIFO, decoding it, and then calculating the ensuing AXI4 write transfer(s) needed to complete the command. The PCC uses predictive calculations to generate address/qualifiers in a sequential manner for the AXI4 Memory Mapped Address Channel ahead of when the associated data transfer occurs on the Write Data Channel. The first transfer command is needed by the Realigner block which is the first block in the datapath flow. The Realigner command includes any DRE alignment control as well as the data byte count to pass and if a TLAST should be expected from the Stream input.

The command is loaded into the Realigner command queue. Next, the PCC calculates the commands for the AXI4 Memory Mapped transfer qualifiers and control. The calculations include the starting address for the transfer, the number of data beats (burst length) to use for the transfer and transfer width. The PCC only produces incrementing burst type commands (it does not support fixed or wrapping).

In addition, the PCC has to account for any potential 4K byte address boundary roll-overs and, if needed, it will split transfers to avoid a 4K rollover. When a transfer calculation has completed, the PCC loads the required control information for the Write Address Channel into the Write Address Controller command queue and the commands for the Write Data Channel into the Write Data Controller command queue.

When an Address/Qualifier data set has been loaded into the Address Controller command queue, the Write Address Channel Controller pops the information from the queue and (when allowed by the Store and Forward block) posts it on the AXI4 Write Address Channel as a write transfer request to the AXI Interconnect. As such, the Address Controller is responsible for the mechanization of the required AXI4 addressing protocol for the Write Master. When a specific write request has been posted on the AXI4 Address Channel and accepted by the Interconnect, the Address Controller will fetch and post the next transfer address/qualifier data set from the command queue if available.

The Write Data Controller works in parallel with the Write Address Controller. It pops the data channel control set from its command queue and processes it. This command is used to set up the Data Channel in preparation for the ensuing write data coming in from the Store and Forward block. The input data from the Store and Forward is routed to the AXI4 Write Data Channel per the PCC command set up.

An optional Data Realignment Engine (DRE) can be inserted in the datapath to compensate for a mismatch in address/byte-lane alignment between the Memory Mapped side and the byte-lane alignment on the input Stream side. The S2MM DRE is included within the Realigner block and is controlled via the Realigner's command queue (loaded by the PCC). The DRE alignment mapping is set up whenever a Write Command is processed with the DRR bit set to '1'. The DRE then retains the sequential alignment information internally (as data is streamed through it). The Write Master DRE is commanded to flush out intermediate data to the Data Channel Interface when the last data beat of the last AXI Write operation occurs that was calculated from a Write Command with the EOF bit set.

The Write Master incorporates a Skid Buffer to provide output registering of the Write data Channel interface signals and still maintain the WREADY/WVALID throttling capability. The Skid Buffer is discussed in more detail in a later section (see Skid Buffer).

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. In this mode, PCC is replaced with a special Indeterminate Bytes to Transfer Command Calculator (IBTTCC) for the Indeterminate mode (see Figure 4-8). This command calculator has a special interface with the S2MM Store and Forward block that allows the Store and Forward to relay to the IBTTCC the actual amount of data that has been received from the Realigner. The IBTTCC then can calculate the Address Controller command and the Write Data Controller command based on that actual value, not a predicted value.

An additional feature of the Indeterminate BTT mode is the absorption of overflow data from the input Stream channel by the Realigner block. 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.

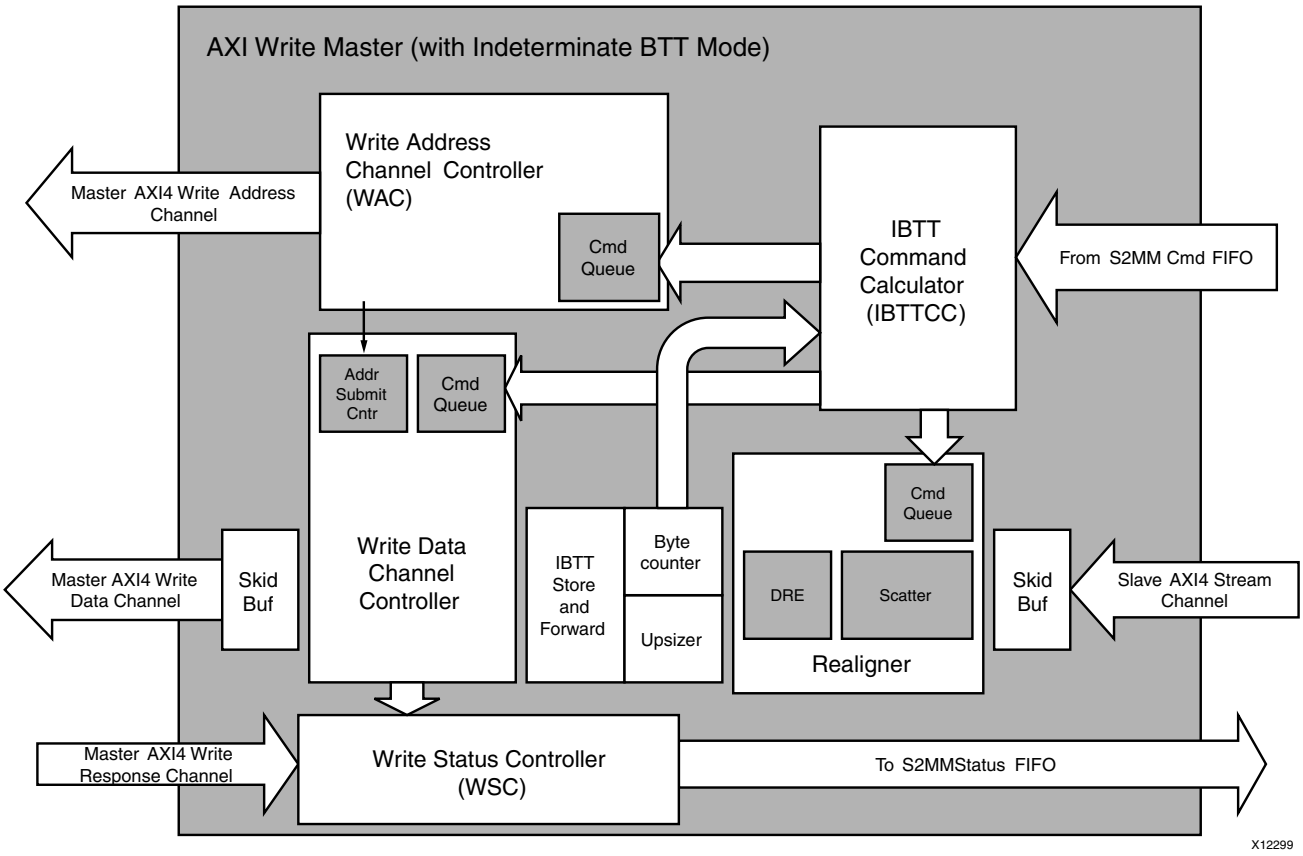

Figure 4-8: **AXI S2MM (Write) Master Block Diagram (IBTT Mode)**

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. The Store and Forward block is coupled to the Write Address Controller so 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 Store and Forward is also fitted with an upsizer feature that is automatically enabled if the AXI4 Memory Mapped data width is wider than the Stream Channel data width. 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) * C_S2MM_BURST_SIZE) rounded up to the next power of 2.

Indeterminate BTT Mode Store and Forward

When the S2MM is set to support Indeterminate BTT mode, the IBTT Store and Forward block is automatically enabled; the user has no control of this. The IBTT Store and Forward is also fitted with an upsizer for narrow stream support as well as a byte counter that tracks the number of received bytes from the Stream Channel. A special interface with the command calculator module is included that relays the received byte count to the IBTTCC. Further information can be found in S2MM Indeterminate BTT (Internal) Operation Mode.

Skid Buffer

The high Fmax requirements of the DataMover require I/O interfaces be registered. One problematic area is the Memory Mapped Data Channel to/from Stream flow. The protocol of the two interfaces (full master/slave throttling) does not allow for a simple registering of the interface signals. To handle the need to register the interfaces to break timing paths and to maintain full Master/Slave throttling, the use of a Skid Buffer is employed inside the DataMover. A Skid Buffer is used in the Data Channel/Stream path in each of the two DataMover elements. The Skid Buffer adds a clock of latency to the associated datapath.

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 4-5.

DataMover Basic Feature Reduction from the Full Version

The Basic implementation trades off features and performance for simplicity and reduced resource utilization. The current use case is the AXI Scatter Gather helper core (axi_sg) where Transfer Descriptor records are being shuttled to and from Memory. These transfers are bursts of 16 data beats or less and are always aligned to 32-bit addressing in Memory. 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)
- Max 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: Max Burst length = 32, Stream Data Width = 4 bytes (32-bits), the maximum commanded transfer length (BTT) is 128 bytes
  - Precludes the need for complex Predictive Command Calculator logic
  - 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
  - User must ensure that 4K address boundary crossing scenario do not occur as a result of any Parent Command loaded into DataMover
- No Store and Forward support
- Overlapped transfers are limited to one deep
  The PCC Address and PCC Data Control FIFOs are replaced with a register

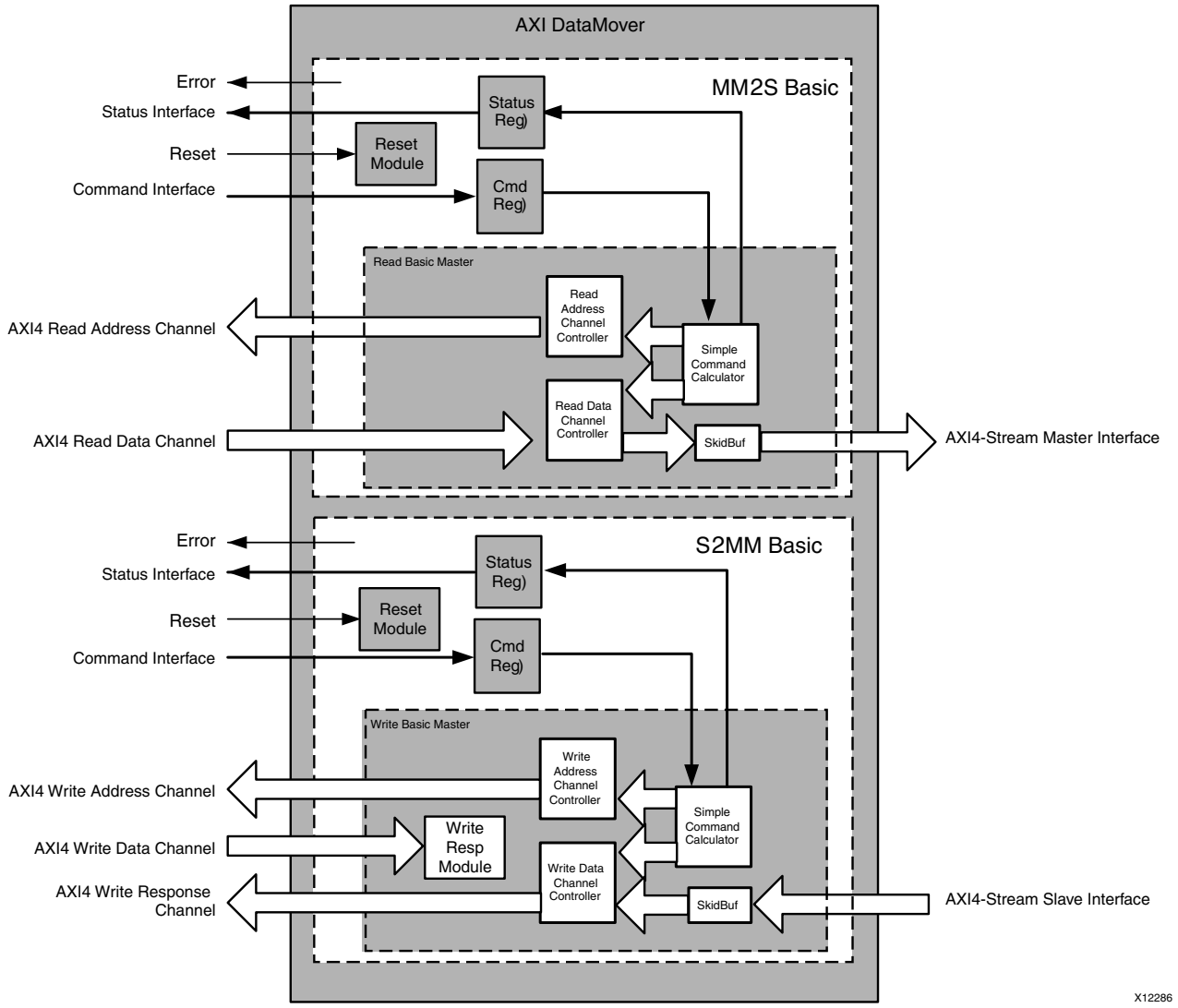

Figure 4-9: **DataMover Basic Block Diagram**

DataMover Basic Command Interface

The DataMover Basic operations are controlled via the AXI Slave Stream Command interface exactly as the full-featured version is controlled. However, there is not a Command FIFO but a Command Register. This register is implemented to operate like a one-entry deep FIFO. In operation, this allows the DataMover Read or Write functions to have up to two commands queued, one being executed internally and another loaded into the Command Register waiting for execution.

The format of the Basic command word is shown in Figure 4-10 and detailed in Table 4-4. The format is the same as the full version command but without packetizing, DRE, and MSBs of the BTT ignored by the Basic version. The command format allows the specification of a data transfer from 1 byte to a maximum set by the formula (C_*_BURST_SIZE x (C_M_AXIS_*_TDATA_WIDTH/8).

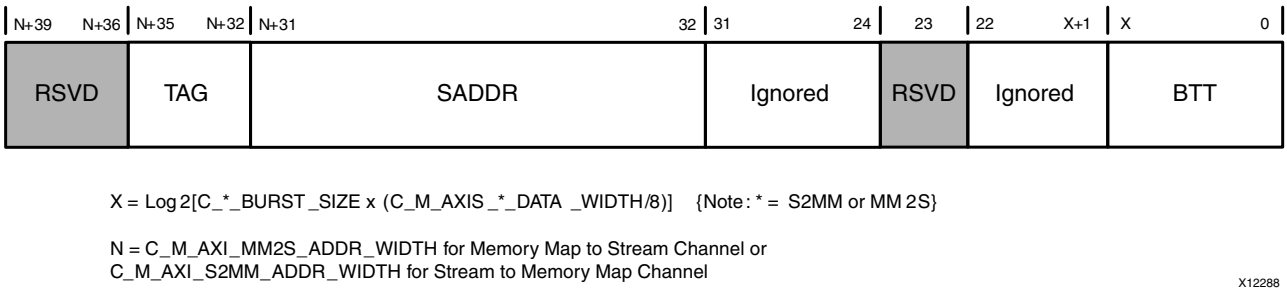

Figure 4-10: **DataMover Basic Command Word Layout**

Table 4-4: **DataMover Basic Command Word Details**

Bits	Field Name	Description
(N+39) - (N+36)[1]	RSVD	**Reserved** This field is reserved to pad the command width to an even multiple of 8 bits. (required for AXI4-Stream interfaces)
(N+35) - (N+32)[1]	TAG	**Command TAG** 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[1]	SADDR	**Start Address** This field indicates the starting address to use for the Memory Mapped side of the transfer requested by the command.
31-24	Ignored	This field is ignored by the DataMover Basic. Can be any value but zeroes are recommended.
23	RSVD	**Reserved** 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.

Table 4-4: **DataMover Basic Command Word Details *(Cont'd)***

Bits	Field Name	Description
X-0	BTT	**Bytes to Transfer** This field indicates the total number of bytes to transfer for the command. The maximum allowed value is set by the following formula: C_*_BURST_SIZE x (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 status of the DataMover Basic's transfer operations is provided via an AXI Master Stream interface that relays transfer status to the user logic. In the Basic version, the Status FIFO is replaced with a register that behaves as a one-deep FIFO. As with the Full version, the 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. The format of the Status word is the same as the full version and is shown in Figure 4-3 and detailed in Table 4-2.

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 clock also synchronizes the internal Command/Status 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_S2MM_STSCMD_IS_ASYNC parameter.

The Asynchronous Command and Status interface mode is shown in Figure 4-11 and the Synchronous Command and Status interface mode is shown in Figure 4-12.

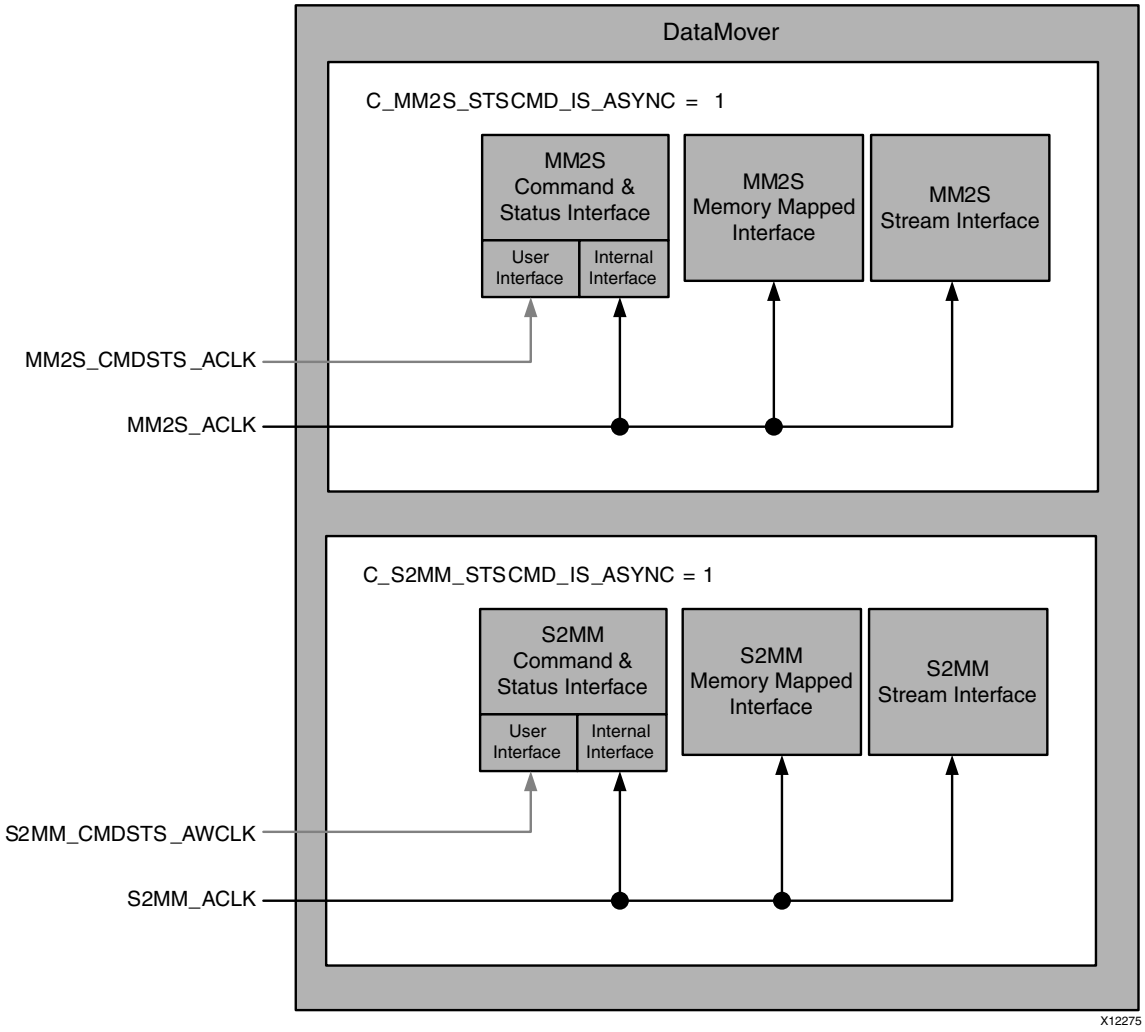

Figure 4-11: **DataMover Clocking Structure (Asynchronous Command/Status Interfaces)**

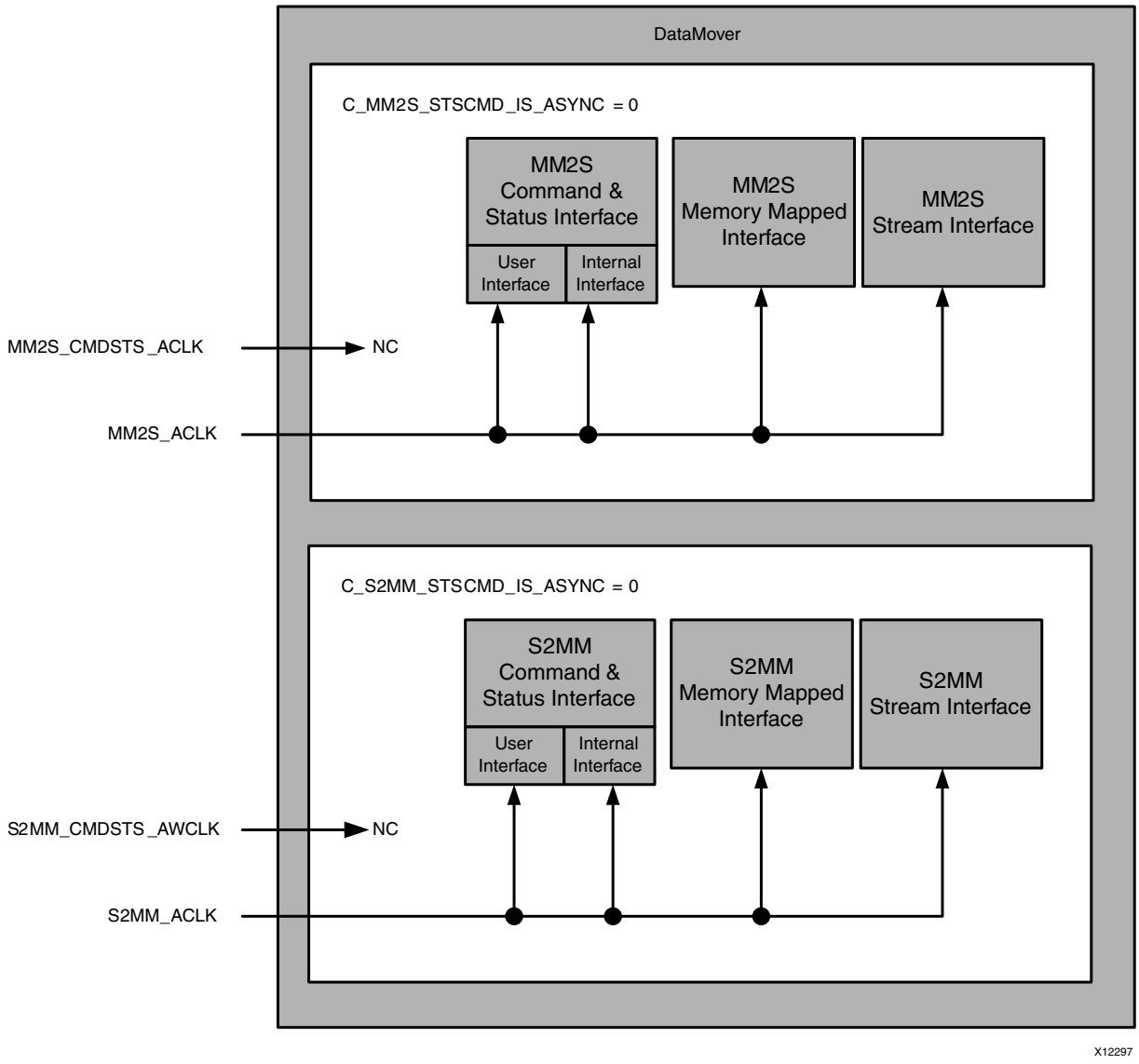

Figure 4-12: **DataMover Clocking Structure (Synchronous Command/Status Interfaces)**

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.

The `m_axi_mm2s_aclk` is the main synchronizing clock for the MM2S block. This clock synchronizes both the associated MM2S Memory Mapped and MM2S Stream interfaces as well as all of the MM2S internal logic. The `m_axi_s2mm_aclk` is the main synchronizing clock for the S2MM block. This clock synchronizes both the associated S2MM Memory Mapped and S2MM Stream interfaces as well as the all of the S2MM internal logic.

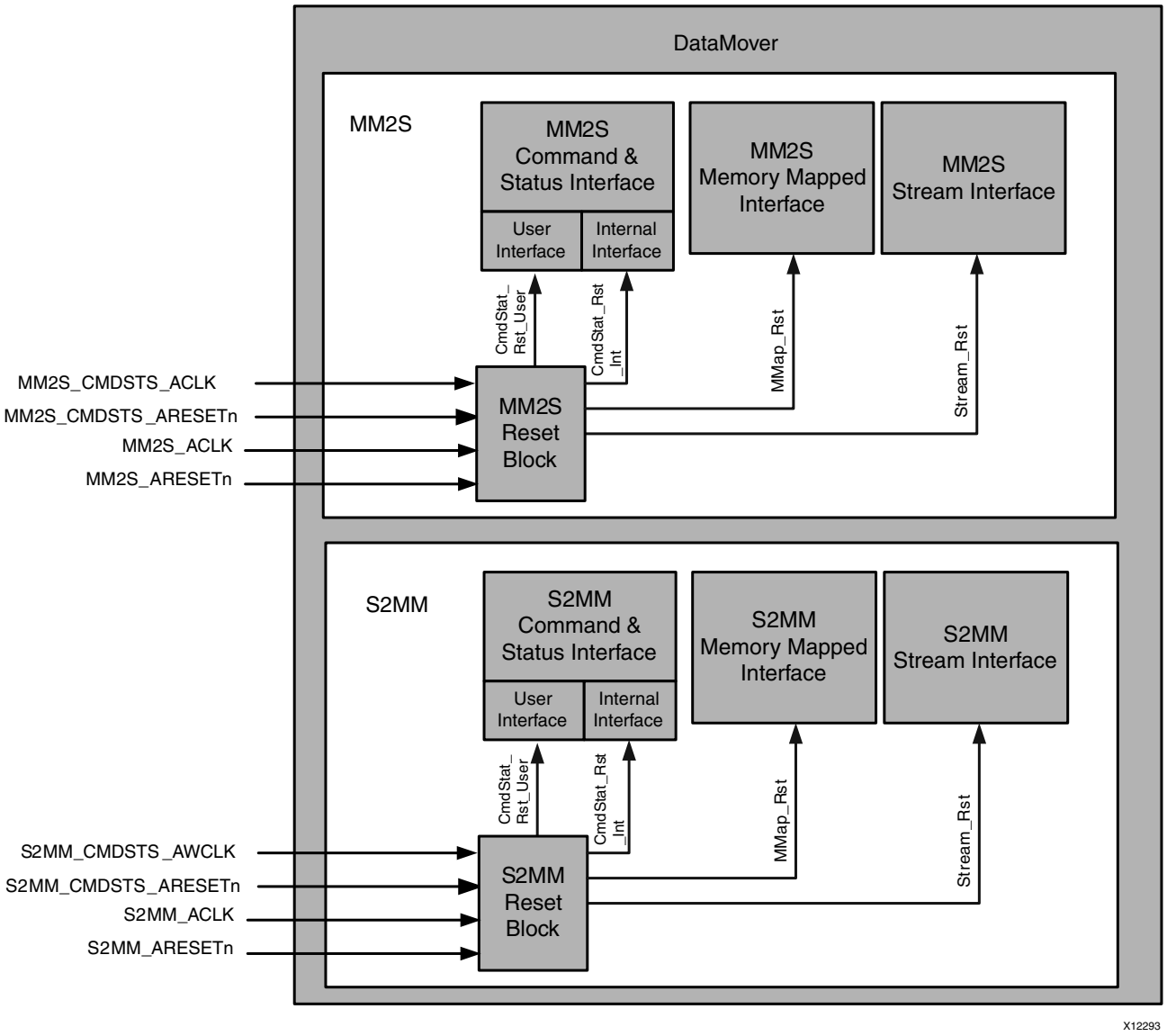

Figure 4-13: **DataMover Reset Scheme**

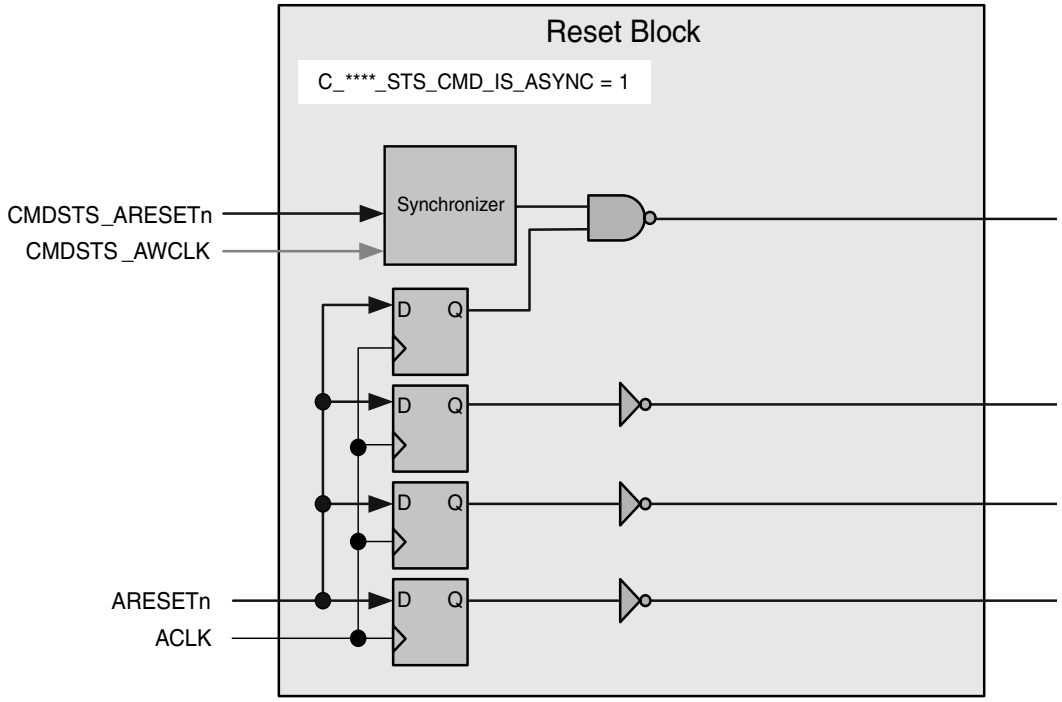

Figure 4-14: **Reset Block Diagram (Asynchronous Command/Status Interface)**

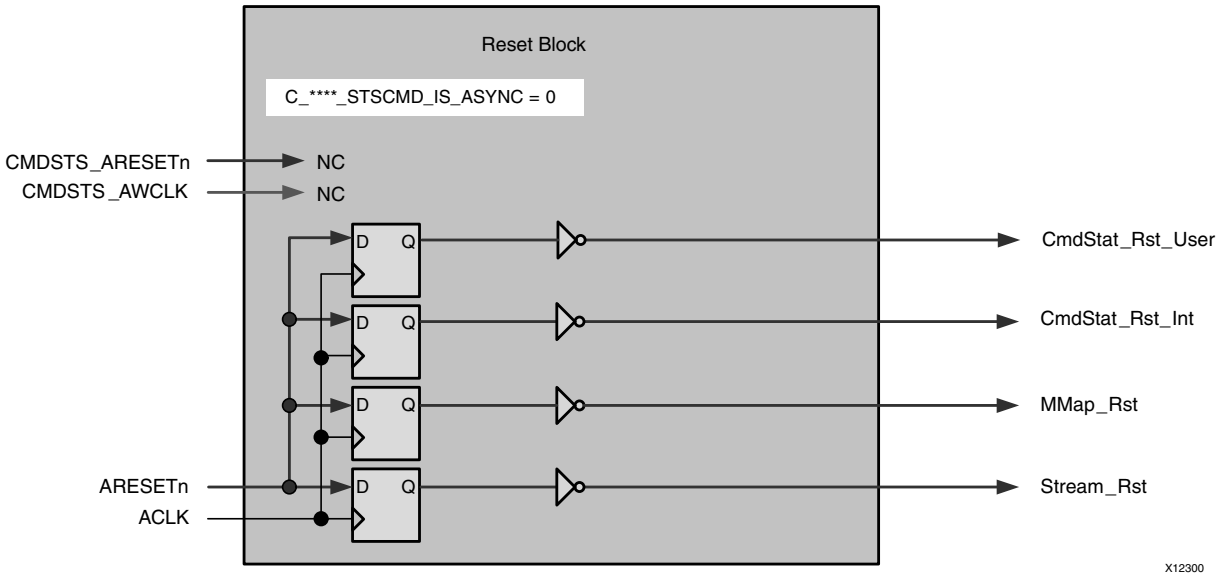

Figure 4-15: **Reset Block Diagram (Synchronous Command/Status Interface)**

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 4-16.

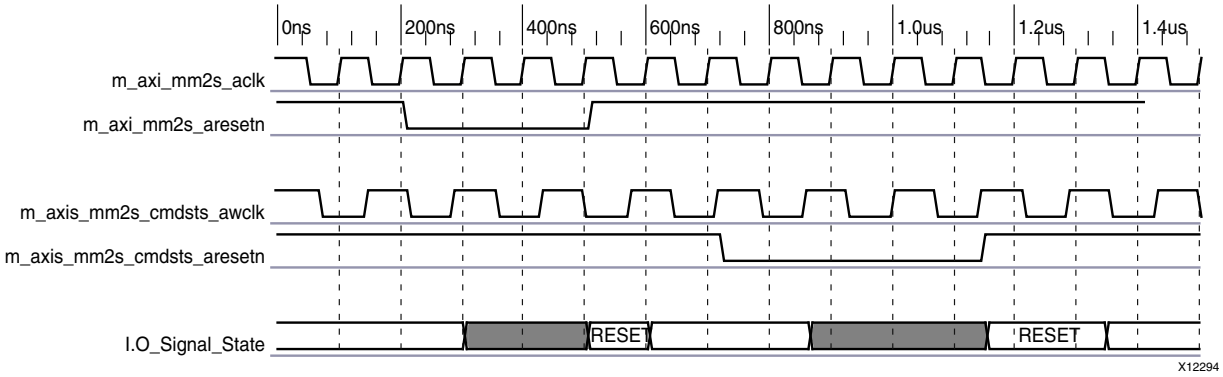

Figure 4-16: **Reset Assertion Timing**

AXI DataMover Operation

The following sections provide information on DataMover operation and special considerations.

S2MM Indeterminate BTT (Internal) Operation Mode

The S2MM transfer function has an additional operational exposure when compared to the MM2S. This situation arises when the Controller (external to DataMover) generating transfer commands does not have a prior knowledge of the amount of data that is going to be received on the S2MM input Stream interface. In this case, it is desirable for the DataMover to accept commands that have a BTT value that really represents a "do not exceed" value, not the actual bytes to transfer value.

The DataMover S2MM function must keep track of incoming Data from the input Stream and only generate the necessary transfers to write that amount of data to the associated Memory Map Write Channel. The user indicates the end of the input transfer with the assertion of the `s_axis_s2mm_tlast` signal on the last data beat of the Stream input. This mode of operation requires some additional hardware resources to provide the S2MM Command Calculator insight into the incoming data. The hardware assist consists of a data buffer FIFO, a counter to track the number of writes in the input side of the Buffer, an additional FIFO that is loaded with a data beat count and control flags that correspond to a single burst transfer on the AXI4 Memory Map Write Channel. In addition, a bytes transferred counter keeps track of the actual number of bytes received from the Stream interface. This Bytes Received count is then appended to the status word for the associated command and supplied back to the external controller via the S2MM Status interface.

The Indeterminate BTT support is enabled when the C_S2MM_SUPPORT_INDET_BTT parameter is assigned a value of integer 1. If Indeterminate BTT support is not enabled and the input stream interface has a TLAST assertion (`s_axis_s2mm_tlast`) prior to the expected byte count from the BTT field in the Command, the S2MM logs an internal error status in the corresponding Status word and halt operation.

The S2MM Indeterminate BTT mode of operation also includes an additional feature that comes into play when a Stream Packet overflow occurs. If the S2MM is executing a parent command where the EOF bit is asserted in the command word, the S2MM absorbs and discards overflow packet data from the point that the bytes to transfer (BTT) for the command has been reached to the TLAST data beat of the input stream packet. This feature is needed for video applications where incoming line data can exceed expectations but the DataMover needs to complete and be ready for the next packet transfer without throttling the Stream input or erring out and needing a reset.

Memory Mapped and Stream Data Relationship without Store and Forward Enabled

In general, it is recommended that the data width of the Stream interface match the data width of the corresponding Memory Mapped interface. AXI4 Memory Mapped interfaces can have data widths that are 32 to 1024 bits in width. When the Stream data width and the Memory Mapped data width match, the DataMover is able to "connect" the Stream datapath to the Memory Mapped Data channel without any special considerations. An example of a 32-bit Stream and 32-bit Memory Mapped data flow is shown in Figure 4-17. In this case the Stream has a data width that is 4-bytes wide and matches the Memory Mapped width of 4 bytes.

All transfers on the Memory Mapped side can move 4 bytes per data beat and the Stream side can move 4 bytes per data beat. DRE insertion and usage is straight forward because there is a one-to-one correlation between byte lanes on the Memory Mapped Side and virtual byte lanes on the stream side.

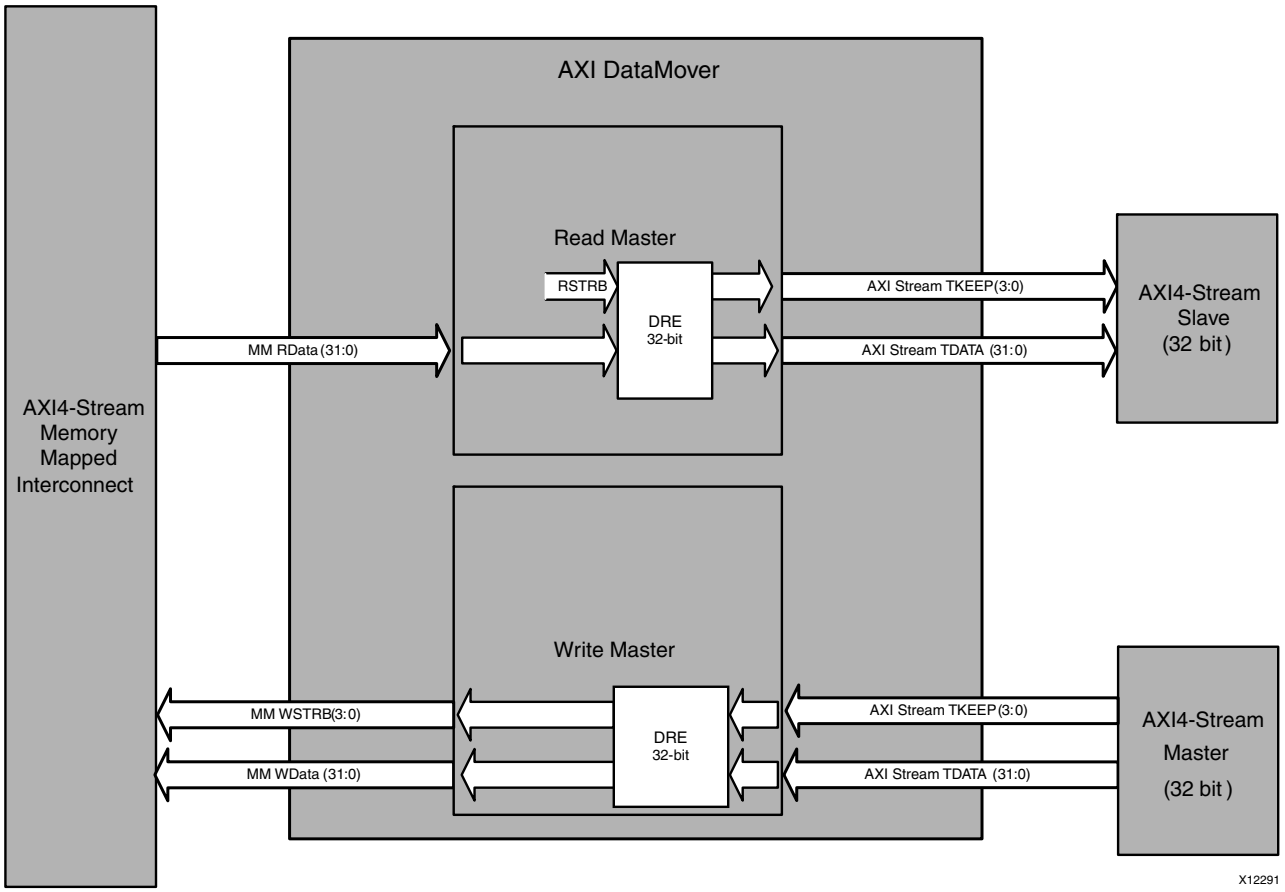

Figure 4-17: **Memory Mapped to Stream Support Block Diagram (32-bit Stream example)**

Narrow Stream Support Without Internal Store and Forward

For enhanced usability, AXI DataMover design also supports "Narrow Streams" when the internal Store and Forward functions with upsizer/downsizer are not enabled due to resource limitations. This situation is defined as the data width of the Stream interface associated with a Memory Mapped interface that is allowed to be less than the data width of the Memory Mapped interface. However, you need to consider certain restrictions that apply. It is required that parameterized Stream data widths be powers of 2 starting with 8 bits (that is, 8, 16, 32, 64, 128, 256, 512 and 1024 bits). This is due to the requirement that the Memory Mapped interface data widths be restricted to 32 to 1024 bits.

The DataMover is designed such that the posted transfer size (AWSIZE(2:0) and ARSIZE(2:0)) for Memory Mapped requests is being set according to the data width of the corresponding Stream Interface data width. The DataMover does not perform byte packing or unpacking operations. Because the Read and Write elements of the DataMover are independent, it is possible that the Stream width of the Read Master Stream is different than that of the Write Master Stream and each behaves according to the associated Stream/Memory Mapped data width relationship.

Some example cases are discussed in the next three sections to illustrate the data flow required by the DataMover for Narrow Stream support.

32-Bit Memory Mapped Data Width with 8-Bit Stream Data Width

The first scenario for consideration is where the Memory Mapped data width is 32-bits and the Stream data width is 8 bits. In this case, the DataMover is configured for a data channel width of 32 bits on the Memory Mapped interfaces and an 8-bit data width on the data stream interfaces. DRE is not an option in this case as an 8-bit wide DRE is a pipe. This configuration is depicted in Figure 4-18. The posted transfer size (AWSIZE(2:0) and ARSIZE(2:0)) for Memory Mapped requests is set to be "000" for 1 byte per data beat transfers. The Write Master mirrors the 8-bit data across all byte lanes of the Memory Mapped Data Channel. This ensures the write data byte is on the applicable byte lane for each data beat of the transfer. However, the Read Master incorporates a 4=>1 Mux that selects the read data byte from the appropriate byte lane of the Memory Mapped Read Data Channel. The Mux is controlled by the LS 2 bits of the internal tracking address of each data beat of the read transfer. The output of the Mux is then routed to the 8-bit output Stream. The anticipated I/O timing of a Read and a Write transfer in this configuration is shown in Figure 4-19 and Figure 4-20 respectively.

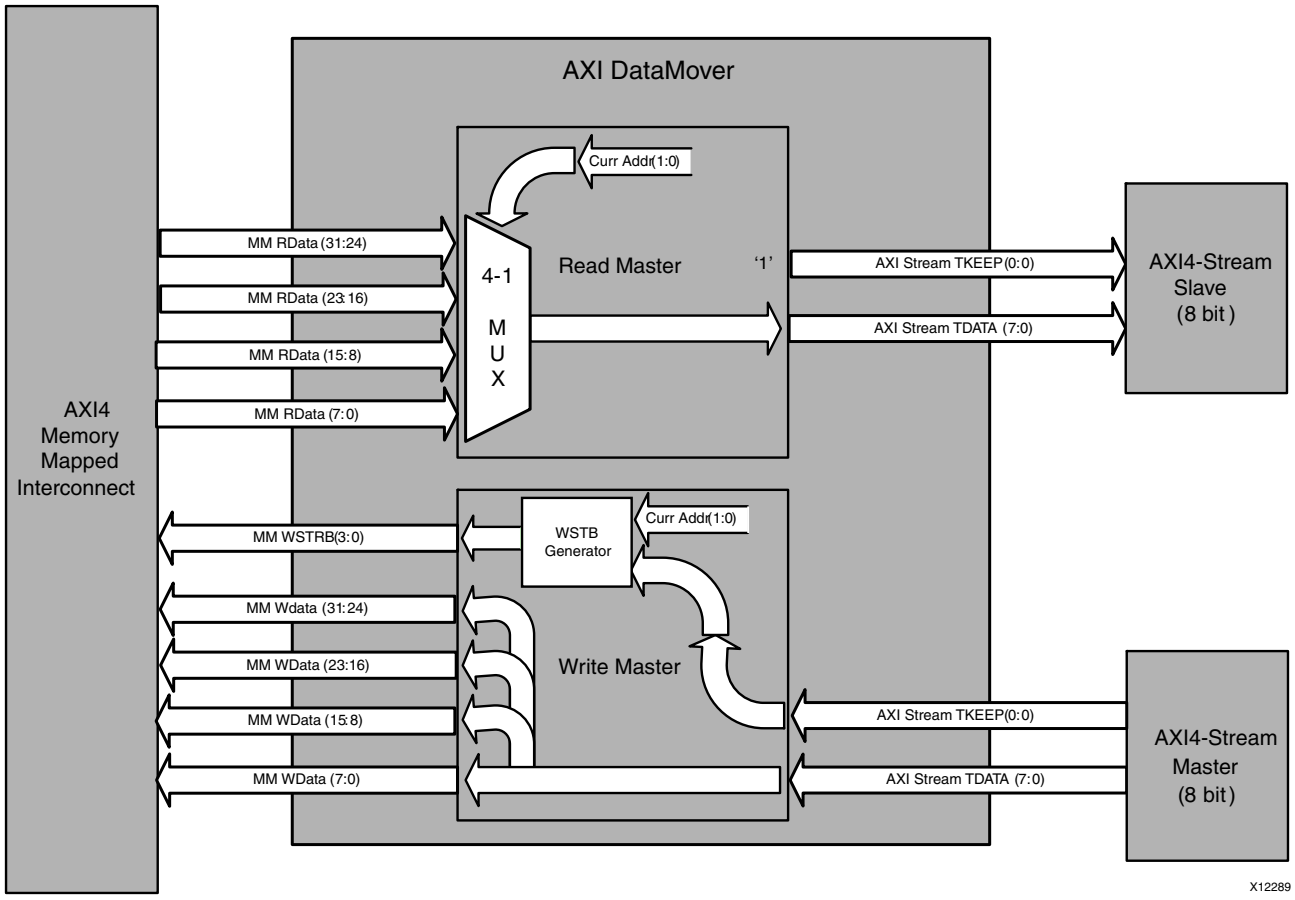

Figure 4-18: **Memory Mapped to Stream Support Block Diagram (32-bit Stream Example)**

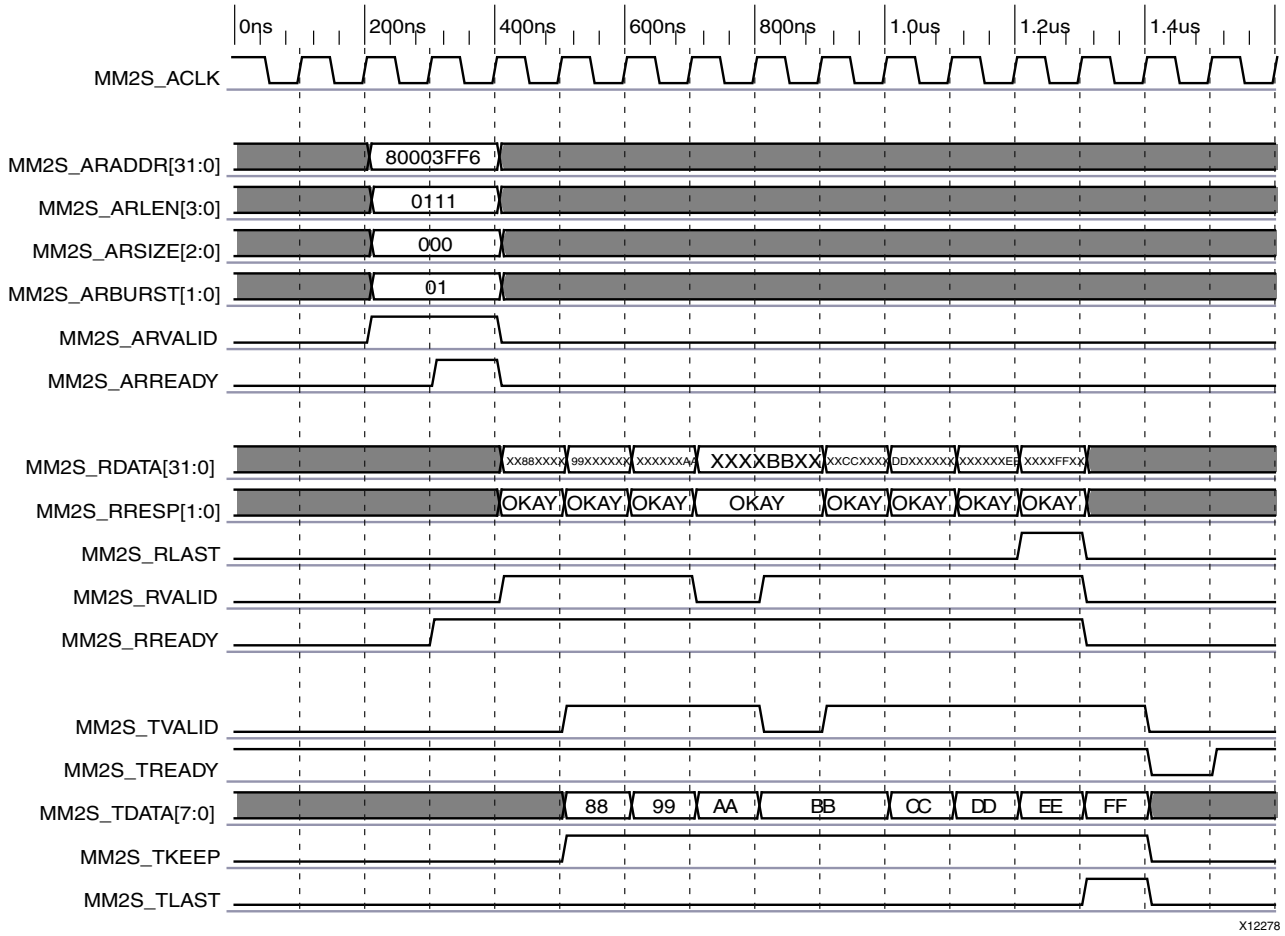

Figure 4-19: **32-Bit Memory Mapped Read to 8-bit Stream Transfer (No DRE, No Store and Forward)**

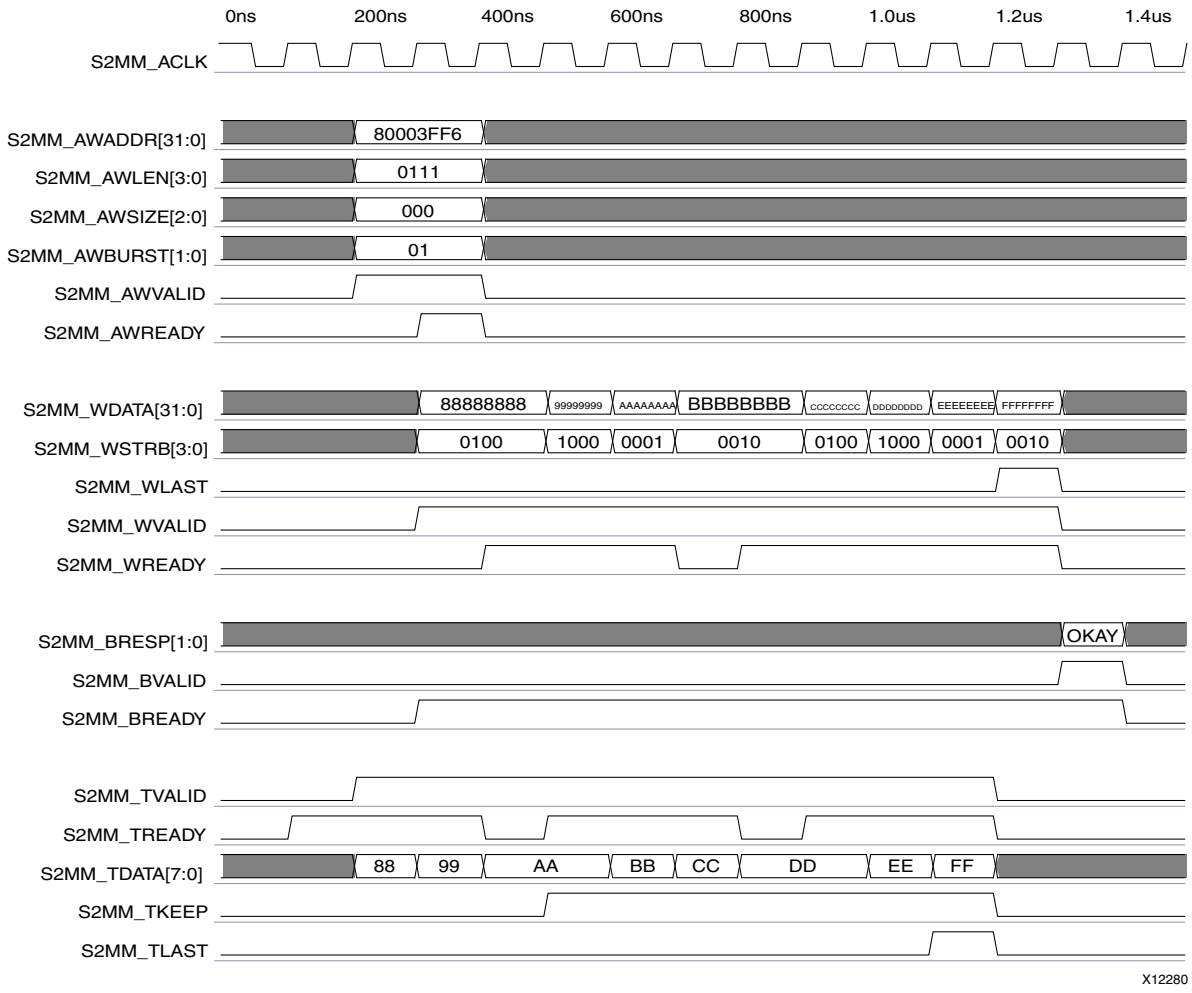

Figure 4-20: **32-Bit Memory Mapped Write from 8-bit Stream Transfer (No DRE, No Store and Forward or IBTT Mode)**

32-Bit Memory Mapped Data Width with 16-Bit Stream Data Width

The second scenario for consideration occurs when the Memory Mapped data width is 32-bits and the Stream data width is 16 bits. In this case, the DataMover is configured for a data channel width of 32 bits on the Memory Mapped interfaces and a 16-bit data width on the data Stream interfaces. This configuration is depicted in Figure 4-21. The posted transfer size (AWSIZE(2:0) and ARSIZE(2:0)) for Memory Mapped requests is set to be "001" for 2 byte per data beat transfers. The Write Master mirrors the 16-bit data across byte lanes 0/1 and 2/3 of the Memory Mapped Write Data Channel. This ensures the write data byte is on the applicable byte lanes for each data beat of the transfer. The Read Master incorporates a 2=>1 Mux that selects the read data slice (2-bytes) from the appropriate byte lanes of the Memory Mapped Read Data Channel. The Mux is controlled by bit 1 of the internal tracking address of each data beat of the read transfer.

The output of the Mux is then routed to the 16-bit output Stream. The anticipated I/O timing of a Read and a Write transfer in this configuration is shown in Figure 4-21 and Figure 4-22 respectively.

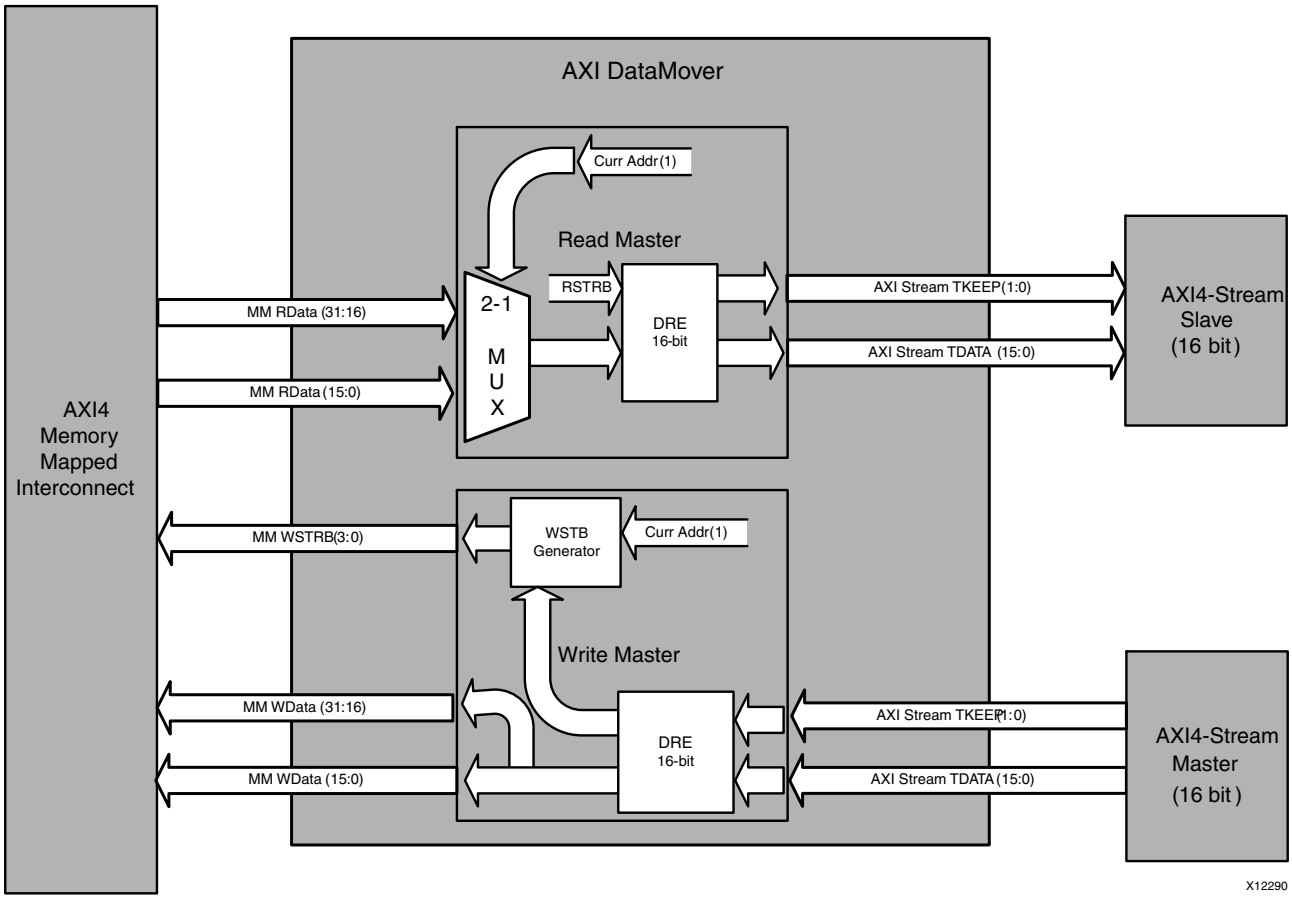

Figure 4-21: **Narrow Stream Support Block Diagram (16-bit Stream Example)**

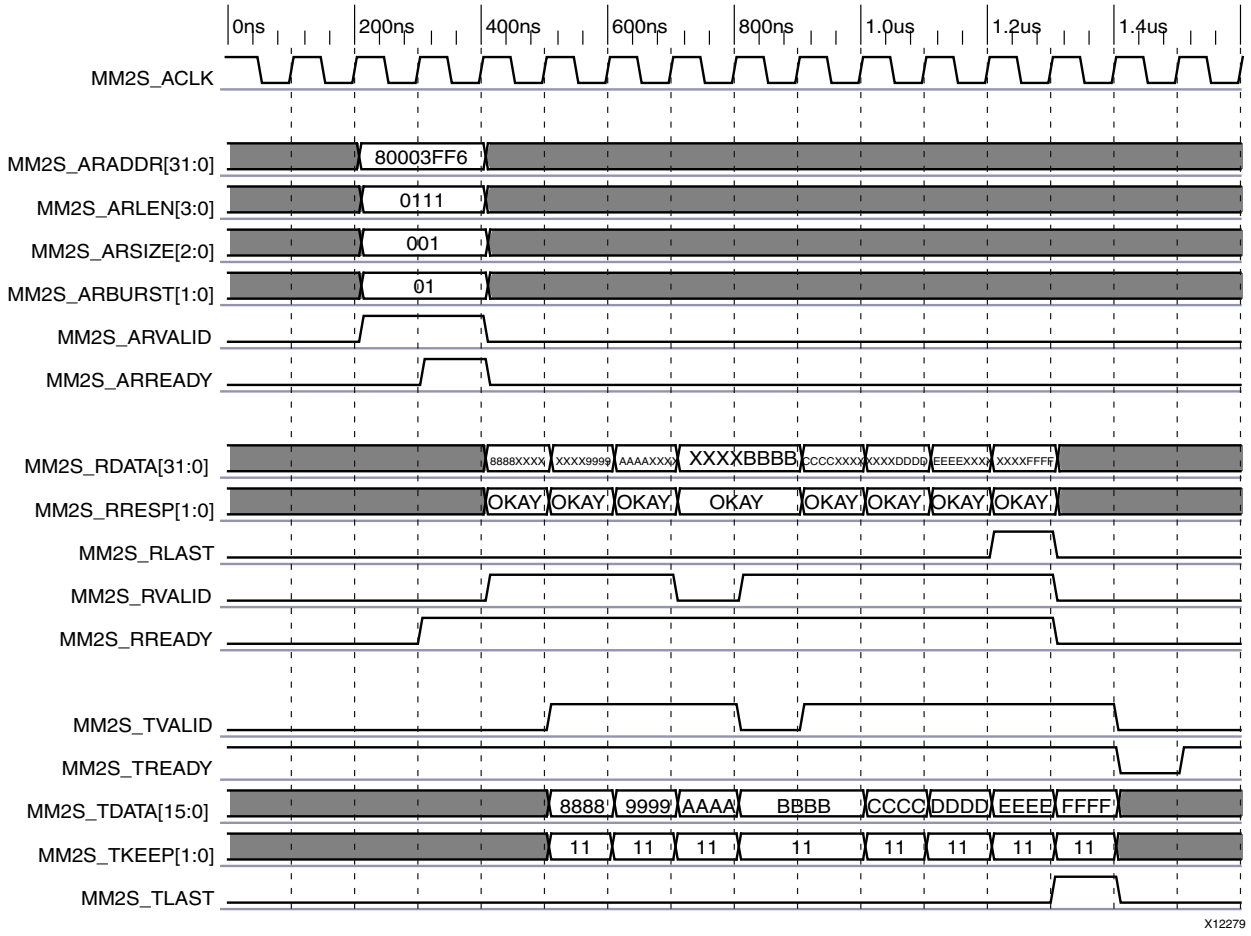

Figure 4-22: **32-Bit Memory Mapped Read to 16-bit Stream Transfer (No DRE, No Store and Forward)**

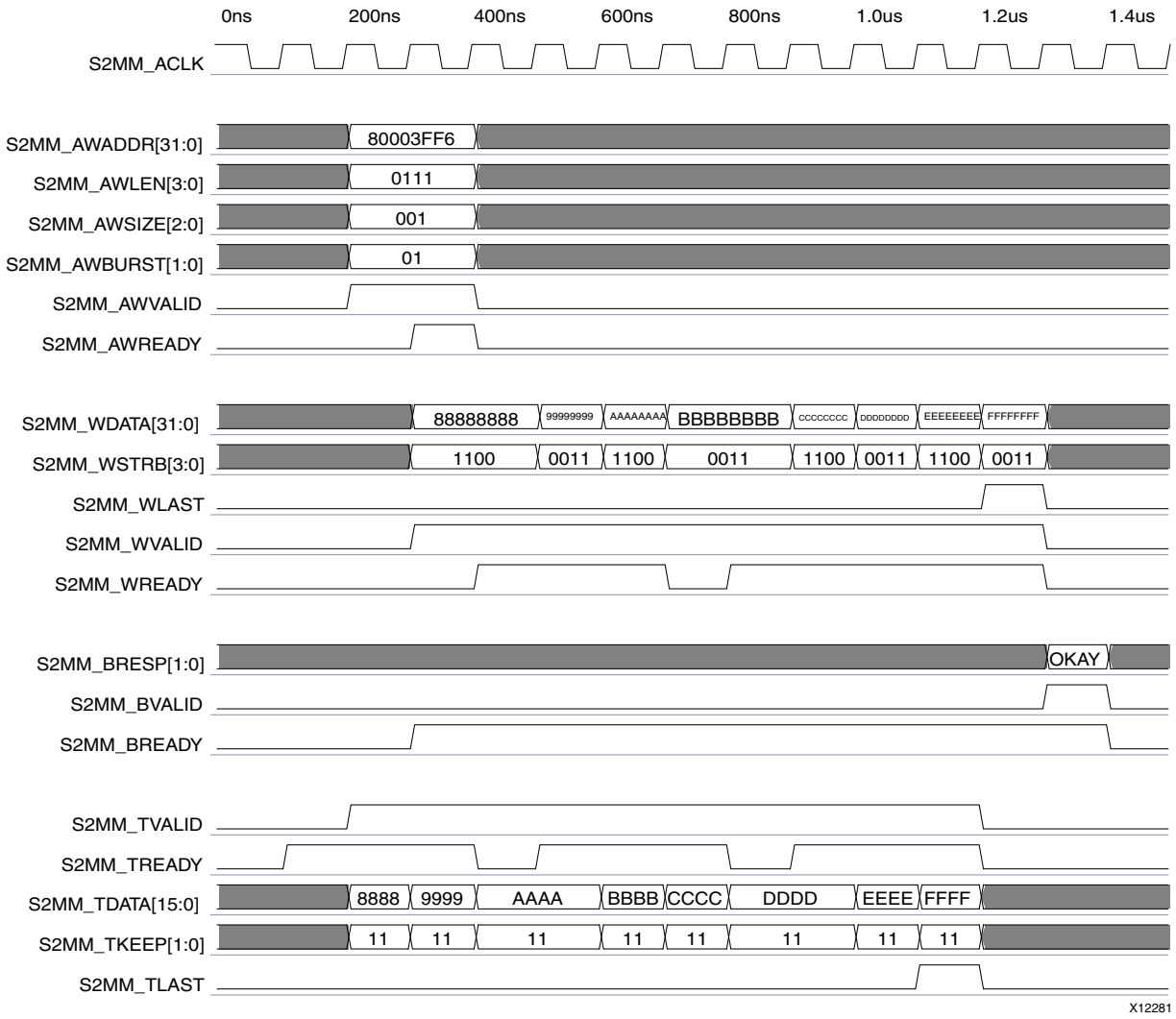

Figure 4-23: **32-Bit Memory Mapped Write from 16-bit Stream Transfer (No DRE, No Store and Forward or IBTT Mode)**

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) V3_00_a 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. It has been demonstrated in actual embedded systems that have Masters that pipeline transfers requests without being able to complete the associated data channel transfers can cause system lockups. 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's address/qualifier posting to AXI4 based on the available space and available data levels.

DataMover External Store and Forward Ports

The DataMover's external Store and Forward interface consists of 8 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 4-24. This is representative of the AXI CDMA use case. The external Store and Forward module has the ability to control the DataMover's Address/Qualifier posting to the AXI4 bus via 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 a address/qualifier set has been committed to the AXI4 bus via 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 via 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).

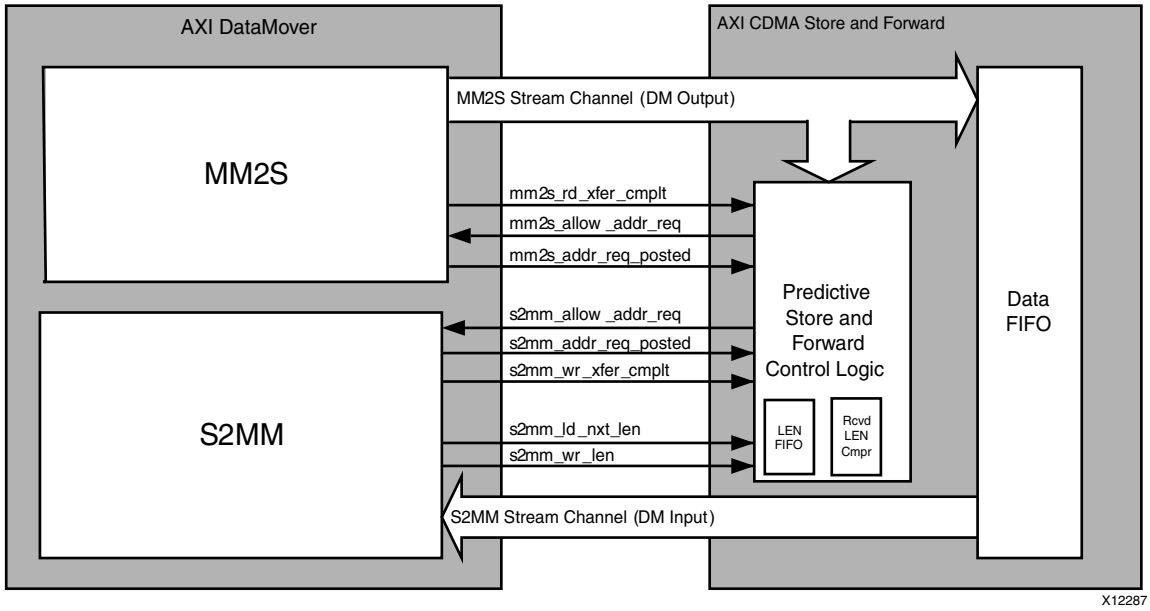

Figure 4-24: **CDMA Store and Forward Connection Example with Address Pipeline Control**

Request Spawning

One important aspect of the DataMover operation is the ability of the DataMover 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 mentioned parameters are C_MM2S_BURST_SIZE and C_S2MM_BURST_SIZE. The Native Data width used by DataMover is dependent upon the inclusion/omission of the Store and Forward feature (with upsizer or downsizer depending upon the direction of transfer). If the Store and Forward feature is enabled, then native data width is equal to the Memory Map side data width. If the Store and Forward feature is not enabled, then native data width is equal to the Stream side data width. For example, a 16-bit Stream interface connection sets the DataMover native data width to 16-bits. The DataMover will command transfers (AWSIZE) of 2-bytes per data beat with burst lengths equal to 16, assuming the default setting of the Burst Length limit parameter is defaulted at 16. If the DataMover's Stream interface is 8 bits, then the DataMover will command transfers (AWSIZE) of 1 byte per data beat with burst lengths equal to 16 (also assuming the default setting of the Burst Length limit parameter is defaulted at 16).

MM2S Gather Operations

When the MM2S DRE function is included, DataMover supports Gather operations. A Gather operation is defined as collecting (reading) data from various Memory Mapped addresses and concatenating the data into a single contiguous Stream packet. Supporting Gather requires the inclusion of the MM2S DRE and the use of multiple commands per packet. Each segment to be transferred requires its own parent command to be loaded in the DataMover with the EOF bit of each segment command suppressed until the last segment command.

S2MM Scatter Operations

The S2MM part of the DataMover supports Scatter operations when the S2MM DRE is included. A Scatter operation is the opposite of a Gather. Scatter is used to deconstruct a single contiguous Stream packet into constituent parts where the data segment for each of the parts is written to a different Memory Mapped destination address. Each segment to be transferred requires its own parent command to be loaded in the DataMover with the EOF bit of each segment command suppressed until the last segment command.

DataMover Soft Shutdown (Halt) Request Operations

The overall command queue design of DataMover allows it to fully exploit theAXI4 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 operations. It is also desirable that the DataMover 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 via the active high assertion of the input signal `mm2s_halt`. When the soft halt operations are completed, the MM2S asserts the `mm2s_halt_cmplt` outputs. This output remains asserted until the MM2S is reset via 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, the MM2S function asserts the `m_axis_mm2s_tvalid`, `m_axis_mm2s_tlast`, and `m_axis_mm2s_tstrb` output MM2S Stream signals and ignores the `m_axis_mm2s_tready` input signal. The `m_axis_mm2s_tdata` data output is then driven with bogus 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 via the active high assertion of the input signal `s2mm_halt`. When the soft halt operations are completed, the S2MM asserts the `s2mm_halt_cmplt` output. This output remains asserted until the S2MM is reset via 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 bogus data values.

Design Parameters

The AXI DataMover Design Parameters are listed and described in Table 4-5.

Table 4-5: **Design Parameters**

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
Memory Map to Stream Parameters				
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 Zeros.
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	1 = Include Memory Map to Stream Status FIFO 0 = Exclude 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. ***Note:*** 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 4-5: **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. 1 = Include Memory Map to Stream Data Realignment engine 0 = Exclude 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). ***Note:*** When the parameter C_M_AXIS_MM2S_TDATA_WIDTH is set to1024 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. ***Note:*** Value assigned to this parameter must be greater than or equal to the value log2(C_M_AXIS_MM2S_TDATA_WIDTH/8 x C_MM2S_BURST_SIZE) + 1.
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.

Table 4-5: **Design Parameters *(Cont'd)***

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
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
Stream to Memory Map Parameters				
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 Zeros.
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
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	1 = Include Stream to Memory Map Status FIFO 0 = Exclude Stream to Memory Map Status FIFO

Table 4-5: **Design Parameters *(Cont'd)***

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
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. ***Note:*** 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. 1 = Include Stream to Memory Map Data Realignment engine 0 = Exclude Stream to Memory Map Data Realignment engine
C_S2MM_BURST_SIZE	16, 32, 64, 128, 256	16	Integer	Stream to Memory Map channel maximum allowed burst length (in data beats). ***Note:*** 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.
C_S2MM_BTT_USED	8 to 23	16	Integer	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. ***Note:*** Value assigned to this parameter must be greater than or equal to the value log2(C_S_AXIS_S2MM_TDATA_WIDTH/8 x C_S2MM_BURST_SIZE) + 1.

Table 4-5: **Design Parameters *(Cont'd)***

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
C_S2MM_SUPPORT_INDET_BTT	0,1	0	Integer	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. 0 = The S2MM indeterminate BTT mode is disabled. 1 = The S2MM Indeterminate BTT mode is enabled. ***Note:*** 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.
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	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. 0 = Omit S2MM GP Store and Forward 1= Include S2MM GP Store and Forward ***Note:*** 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.

Table 4-5: **Design Parameters *(Cont'd)***

Parameter Name	Allowable Values	Default Values	VHDL Type	Feature/Description
FPGA Family Type				
C_FAMILY	virtex6, spartan6, virtex7, kintex7	virtex6	string	Specifies the FPGA Family the implementation targets

Allowable Parameter Combinations

Table 4-6: **Allowable Parameter Combination**

Parameter Name	Affects Parameter	Relationship Description
MM2S Design Parameters		
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.

Table 4-6: **Allowable Parameter Combination *(Cont'd)***

Parameter Name	Affects Parameter	Relationship Description
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.
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 log2(C_M_AXIS_MM2S_TDATA_WIDTH/8 x 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 log2(C_M_AXIS_MM2S_TDATA_WIDTH/8 x C_MM2S_BURST_SIZE) + 1.
S2MM Design Parameters		
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. ***Note:*** 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.

Table 4-6: **Allowable Parameter Combination *(Cont'd)***

Parameter Name	Affects Parameter	Relationship Description
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.
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 log2(C_S_AXIS_S2MM_TDATA_WIDTH/8 x 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 log2(C_S_AXIS_S2MM_TDATA_WIDTH/8 x C_S2MM_BURST_SIZE) + 1.

Parameter - I/O Signal Dependencies

Table 4-7: **Parameter - I/O Signal Dependencies**

Parameter Name	Affects Signal	Relationship Description
MM2S Parameter-Port Dependency		
C_INCLUDE_MM2S	m_axi_mm2s_arid, m_axi_mm2s_araddr, m_axi_mm2s_arlen, m_axi_mm2s_arsize, m_axi_mm2s_arburst, m_axi_mm2s_arcache, m_axi_mm2s_arvalid, m_axi_mm2s_rready, m_axis_mm2s_tvalid, m_axis_mm2s_tdata, m_axis_mm2s_tkeep, m_axis_mm2s_tlast, s_axis_mm2s_cmd_tready, m_axis_mm2s_sts_tvalid, m_axis_mm2s_sts_tdata, m_axis_mm2s_sts_tkeep, m_axis_mm2s_sts_tlast	A value of 0 assigned to the affecting parameter removes the MM2S DataMover block from the implementation. The affected output ports are driven with constant logic zero(s) by the DataMover.
C_INCLUDE_MM2S	m_axi_mm2s_aclk, m_axi_mm2s_aresetn, m_axi_mm2s_arready, m_axi_mm2s_rdata, m_axi_mm2s_rresp, m_axi_mm2s_rlast, m_axi_mm2s_rvalid, m_axis_mm2s_tready, m_axis_mm2s_cmdsts_awclk, m_axis_mm2s_cmdsts_aresetn, s_axis_mm2s_cmd_tvalid, s_axis_mm2s_cmd_tdata, m_axis_mm2s_sts_tready	A value of 0 assigned to the affecting parameter removes the MM2S DataMover block from the implementation. The affected input ports are ignored by the DataMover.
C_M_AXI_MM2S_ARID	m_axi_mm2s_arid	The affecting parameter's assigned value is driven out on the affected port as a constant value.
C_M_AXI_MM2S_ID_WIDTH	m_axi_mm2s_arid	The affecting parameter's assigned value directly sets the width of the affected port.
C_M_AXI_MM2S_ADDR_WIDTH	m_axi_mm2s_araddr	The affecting parameter's assigned value directly sets the width of the effected port.

Table 4-7: **Parameter - I/O Signal Dependencies *(Cont'd)***

Parameter Name	Affects Signal	Relationship Description
C_M_AXI_MM2S_ADDR_WIDTH	s_axis_mm2s_cmd_wdata	The affecting parameter's assigned value expands the width of the affected port by the amount exceeding the value of 32.
C_M_AXI_MM2S_DATA_WIDTH	m_axi_mm2s_rdata	The affecting parameter's assigned value directly sets the width of the affected port.
C_M_AXIS_MM2S_TDATA_WIDTH	m_axis_mm2s_tdata	The affecting parameter's assigned value directly sets the width of the affected port.
C_M_AXIS_MM2S_TDATA_WIDTH	m_axis_mm2s_tdata	The affected port's width is set by the affecting parameter's value divided by 8.
C_MM2S_STSCMD_IS_ASYNC	m_axis_mm2s_cmdsts_awclk, m_axis_mm2s_cmdsts_aresetn	If the affecting parameter is assigned a value of 0, the affected ports are ignored by MM2S block. If the affecting parameter is assigned a value of 1, the affected ports are used to synchronize and reset the associated MM2S Command and Status AXI4-Stream interfaces.

Table 4-7: **Parameter - I/O Signal Dependencies *(Cont'd)***

Parameter Name	Affects Signal	Relationship Description
S2MM Parameter-Port Dependency		
C_INCLUDE_S2MM	m_axi_s2mm_awid, m_axi_s2mm_awaddr, m_axi_s2mm_awlen, m_axi_s2mm_awsize, m_axi_s2mm_awburst, m_axi_s2mm_awcache, m_axi_s2mm_awvalid, m_axi_s2mm_wdata, m_axi_s2mm_wstrb, m_axi_s2mm_wlast, m_axi_s2mm_bready, m_axis_s2mm_sts_tvalid, m_axis_s2mm_sts_tdata, m_axis_s2mm_sts_tstrb, m_axis_s2mm_sts_tlast	A value of 0 assigned to the affecting parameter removes the S2MM DataMover block from the implementation. The affected output ports are driven with constant logic zero(s) by the DataMover.
C_INCLUDE_S2MM	m_axi_s2mm_aclk, m_axi_s2mm_aresetn, m_axi_s2mm_awready, m_axi_s2mm_bresp, m_axi_s2mm_bvalid, s_axis_s2mm_tvalid, s_axis_s2mm_tdata, s_axis_s2mm_tlast, m_axis_s2mm_cmdsts_awclk, m_axis_s2mm_cmdsts_aresetn, s_axis_s2mm_cmd_tvalid, s_axis_s2mm_cmd_tdata, m_axis_s2mm_sts_tready	A value of 0 assigned to the affecting parameter removes the S2MM DataMover block from the implementation. The affected input ports are ignored by the DataMover.
C_S2MM_AWID	m_axi_s2mm_awid	The affecting parameter's assigned value is driven out on the affected port as a constant value.
C_M_AXI_S2MM_ID_WIDTH	m_axi_s2mm_awid	The affecting parameter's assigned value directly sets the width of the affected port.
C_M_AXI_S2MM_ADDR_WIDTH	m_axi_s2mm_awaddr	The affecting parameter's assigned value directly sets the width of the affected port.
C_M_AXI_S2MM_ADDR_WIDTH	s_axis_s2mm_cmd_tdata	The affecting parameter's assigned value expands the width of the affected port by the amount exceeding the value of 32.
C_M_AXI_S2MM_DATA_WIDTH	m_axi_s2mm_wdata	The affecting parameter's assigned value directly sets the width of the affected port.

Table 4-7: **Parameter - I/O Signal Dependencies *(Cont'd)***

Parameter Name	Affects Signal	Relationship Description
C_M_AXI_S2MM_DATA_WIDTH	m_axi_s2mm_wstrb	The affected port's width is set by the affecting parameter's value divided by 8.
C_S_AXIS_S2MM_TDATA_WIDTH	s_axis_s2mm_tdata	The affecting parameter's assigned value directly sets the width of the affected port.
C_S_AXIS_S2MM_TDATA_WIDTH	s_axis_s2mm_tkeep	The affected port's width is set by the affecting parameter's value divided by 8.
C_S2MM_STSCMD_IS_ASYNC	m_axis_s2mm_cmdsts_awclk, m_axis_s2mm_cmdsts_aresetn	If the affecting parameter is assigned a value of 0, the affected ports are ignored by S2MM block. If the affecting parameter is assigned a value of 1, the affected ports are used to synchronize and reset the associated S2MM Command and Status AXI4-Stream interfaces.

XILINX®

Chapter 5

Detailed Example Design

There is no example design for this core.

XILINX®

Chapter 6

Constraining the Core

There are no applicable constraints for this core.

XILINX®

Appendix 7

Additional Resources

Xilinx Resources

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

http://www.xilinx.com/support.

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

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

List of Acronyms

Acronym	Spelled Out
AXI	Advanced eXtensible Interface
BTT	Bytes to Transfer
CDMA	Central Direct Memory Access
CPU	Central Processing Unit
DDR	Double Data Rate
DMA	Direct Memory Access
DRE	Data Realignment Engine
DSP	Digital Signal Processing
EDK	Embedded Development Kit
EOF	End of Frame
EOP	End of Packet
FF	Flip-Flop
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GP	General Purpose
GUI	Graphical User Interface
I/O	Input/Output
IBTTCC	Indeterminate Bytes to Transfer Command Calculator
IP	Intellectual Property

Acronym	Spelled Out
ISE®	Integrated Software Environment
LUT	Lookup Table
MM2S	Memory Map to Stream
PCC	Predictive Command Controller
RAC	Read Address Channel Controller
RDC	Read Data Channel Controller
R/WC	Read/Write to Clear
RAM	Random Access Memory
RO	Read Only
RW	Read/Write
S2MM	Stream to Memory Map
SG	Scatter Gather
STS	Status
VDMA	Video Direct Memory Access
VHDL	VHSIC Hardware Description Language (VHSIC an acronym for Very High-Speed Integrated Circuits)
WAC	Write Address Channel Controller
WDC	Write Data Channel Controller
WSC	Write Status Controller
XPS	Xilinx® Platform Studio (part of the EDK software)
XST	Xilinx Synthesis Technology

Solution Centers

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.

References

These documents provide supplemental material useful with this user guide:

- *AMBA® AXI4-Stream Protocol Specification*
- ARM® AXI4 Memory Mapped Specification
- ARM AXI4-Stream Interface Specification
- AXI Interconnect IP Data Sheet (DS768)

Go to www.xilinx.com/support to search for Xilinx documentation.

Technical Support

Xilinx provides technical support at 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 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

Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx ISE® Design Suite Embedded Edition software under the terms of the Xilinx End User License. The core is generated using the Xilinx ISE Embedded Development Kit (EDK).

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 software, contact your sales representative.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/11	1.0	Initial Xilinx release.

Notice of Disclaimer