MicroBlaze Debug Module (MDM) v3.1

Product Guide

Vivado Design Suite

PG115 April 2, 2014

Table of Contents

Introduction

This document provides the design specification for the MicroBlaze™ Debug Module (MDM) which enables JTAG-based debugging of one or more MicroBlaze processors.

Features

- Support for JTAG-based software debug tools
- Support for debugging up to 32 MicroBlaze processors
- Support for synchronized control of multiple MicroBlaze processors
- Support for a JTAG-based UART with a configurable AXI4-Lite interface
- Based on Boundary Scan (BSCAN) logic in Xilinx® FPGAs
- Direct JTAG-based access to memory with a configurable AXI4 master interface
- Configurable software access to debug functionality through the AXI4-Lite interface
- Support for cross-trigger between connected MicroBlaze cores, Zynq-7000 Processing System and Integrated Logic Analyzer (ILA) cores

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	UltraScale™ Architecture, Zynq®-7000, 7 Series
Supported User Interfaces	AXI4, AXI4-Lite
Resources	See Table 2-1
Provided with Core	
Design Files	RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Not Provided
Simulation Model	VHDL Behavioral
Supported S/W Driver[2]	Standalone and Linux
Tested Design Tools[3]	
Design Entry Tools	Vivado® Design Suite IP Integrator
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide
Synthesis Tools	Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete listing of supported devices, see the Vivado IP catalog.
2. Standalone driver details can be found in the SDK directory (<*install_directory*>/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from //wiki.xilinx.com.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The block diagram of the MicroBlaze™ Debug Module is shown in Figure 1-1.

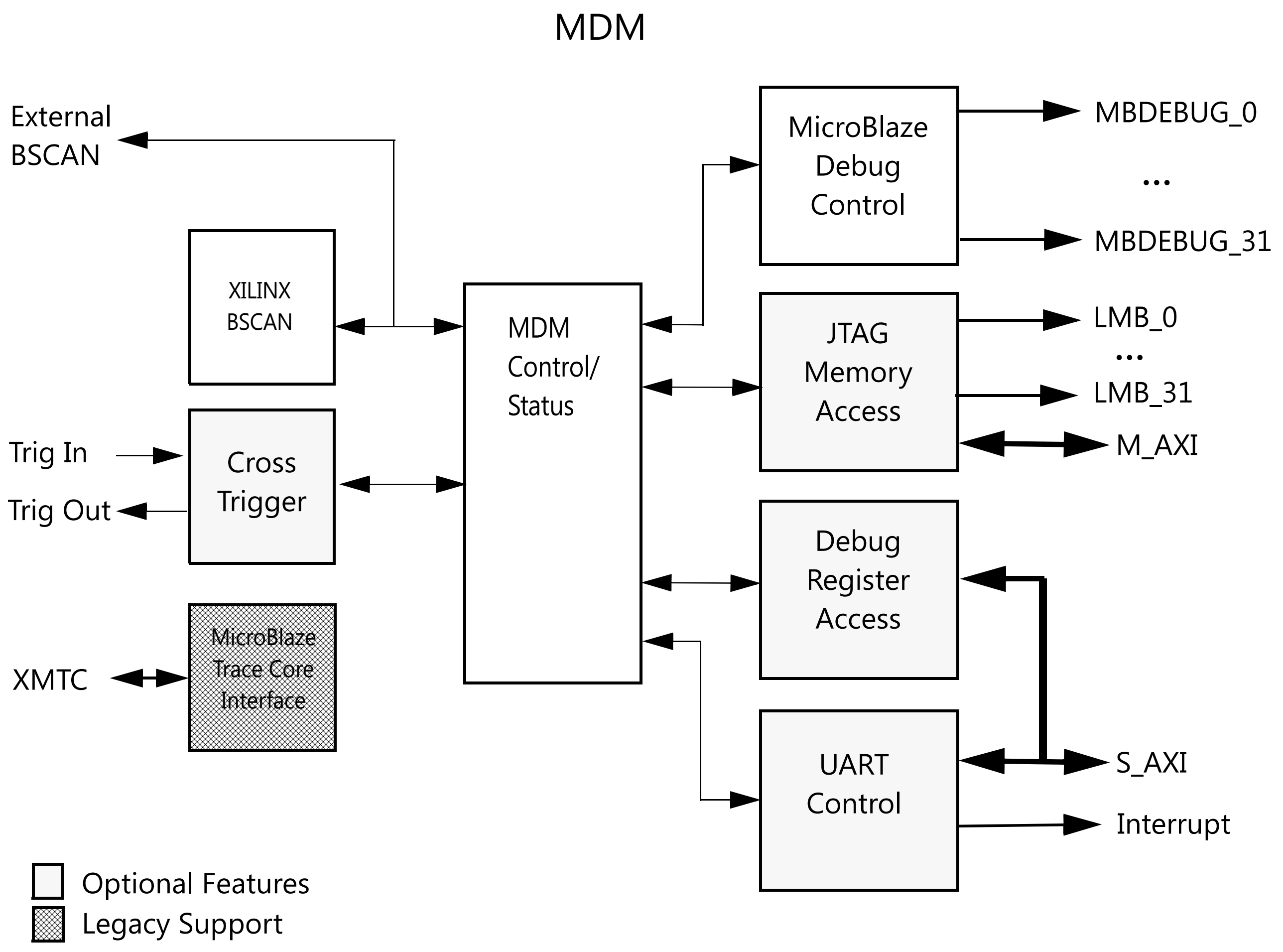


Figure 1-1: **MicroBlaze Debug Module (MDM) Block Diagram**

Feature Summary

The MicroBlaze Debug Module (MDM):

- Enables JTAG-based debugging of one or more MicroBlaze processors.

- Instantiates one BSCAN primitive, or allows an external BSCAN to be used. In devices that contain more than one BSCAN primitive, MDM uses the USER2 BSCAN by default.
- Includes a UART with a configurable slave bus interface which can be configured for an AXI4-Lite interconnect. The UART TX and RX signals are transmitted over the FPGA JTAG port to and from the Xilinx® Microprocessor Debug (XMD) tool. The UART behaves in a manner similar to the LogiCORE™ IP AXI (UART) Lite core.
- Provides a configurable AXI4 master port for direct access to memory from JTAG. This allows fast program download, as well as transparent memory access when the connected MicroBlaze processors are executing.
- Allows software to control debug and observe debug status through the AXI4-Lite slave interface. This is particularly useful for software performance measurements and analysis, using the MicroBlaze extended debug functionality for performance monitoring.
- Includes a cross-trigger capability, which enables routing of trigger events between connected MicroBlaze processors, as well as an external interface compatible with the Zynq-7000 Processing System.

Licensing and Ordering Information

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

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

Product Specification

Standards

The MDM adheres to the AMBA® AXI4 and AXI4-Lite Interface standard (see ARM® *AMBA AXI and ACE Protocol Specification ARM IHI 0022E* [Ref 1]).

Performance

The frequency and latency of the MicroBlaze™ Debug Module are optimized for use together with MicroBlaze. This means that the frequency targets are aligned to MicroBlaze targets.

Resource Utilization

Because the MDM is a module that is used with other design elements in the FPGA, the utilization and timing numbers reported in this section are estimates, and the actual utilization of FPGA resources and timing of the MDM design can vary from the results for a Kintex®-7 device reported in Table 2-1.

Note: UltraScale™ architecture results are expected to be similar to 7 series results.

Table 2-1: **Device Utilization - Kintex-7 FPGAs xc7k325t-2-ffg900**

Parameter Values (other parameters have default values)					Device Resources	
C_USE_CROSS_TRIGGER	C_DBG_MEM_ACESS	C_DBG_REG_ACCESS	C_USE_UART	C_MB_DBG_PORTS	Flip-Flops	LUTs
0	0	0	0	1	105	83
0	0	0	0	2	107	101
0	0	0	0	4	109	106
0	0	0	0	32	161	232
0	0	0	1	1	161	153
1	0	0	0	1	196	201
0	1	0	0	1	370	315
0	0	1	0	1	265	316
1	1	1	1	4	893	1102

Port Descriptions

The I/O signals for the MicroBlaze Debug Module (MDM) are listed and described in Table 2-2.

Table 2-2: **MDM I/O Signals**

Signal Name	Interface	I/O	Initial State	Description
System Signals				
Interrupt		O	0	Interrupt from UART
Debug_SYS_Rst		O	0	Debug system reset
Ext_BRK		O	0	External break
Ext_NM_BRK		O	0	External non-maskable break
AXI4-Lite Slave Interface Signals				
S_AXI_ACLK	S_AXI	I	-	AXI Clock
S_AXI_ARESETN	S_AXI	I	-	AXI Reset, active-Low
S_AXI_AWADDR[C_S_AXI_ADDR_WIDTH-1:0]	S_AXI	I	-	Write Address
S_AXI_AWVALID	S_AXI	I	-	Write Address Valid

Table 2-2: **MDM I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
S_AXI_AWREADY	S_AXI	O	0	Write Address Ready
S_AXI_WDATA[C_S_AXI_DATA_WIDTH-1:0]	S_AXI	I	-	Write Data
S_AXI_WSTB[C_S_AXI_DATA_WIDTH/8-1:0]	S_AXI	I	-	Write Strobes
S_AXI_WVALID	S_AXI	I	-	Write Valid
S_AXI_WREADY	S_AXI	O	0	Write Ready
S_AXI_BRESP[1:0]	S_AXI	O	0x0	Write Response
S_AXI_BVALID	S_AXI	O	0	Write Response Valid
S_AXI_BREADY	S_AXI	I	-	Write Response Ready
S_AXI_ARADDR[C_S_AXI_ADDR_WIDTH-1:0]	S_AXI	I	-	Read Address
S_AXI_ARVALID	S_AXI	I	-	Read Address Valid
S_AXI_ARREADY	S_AXI	O	0	Read Address Ready
S_AXI_RDATA[C_S_AXI_DATA_WIDTH-1:0]	S_AXI	I	-	Read Data
S_AXI_RRESP[1:0]	S_AXI	O	0x0	Read Response
S_AXI_RVALID	S_AXI	O	0	Read Valid
S_AXI_RREADY	S_AXI	I	-	Read Ready
AXI4 Master Interface Signals				
M_AXI_ACLK	M_AXI	I	-	AXI Clock
M_AXI_ARESETN	M_AXI	I	-	AXI Reset, active-Low
M_AXI_AWID[C_M_AXI_THREAD_ID_WIDTH-1:0]	M_AXI	O	0x0	Write Address ID
M_AXI_AWADDR[C_M_AXI_ADDR_WIDTH-1:0]	M_AXI	O	0x0	Write Address
M_AXI_AWLEN[7:0]	M_AXI	O	0x0	Write Address Length
M_AXI_AWSIZE[2:0]	M_AXI	O	0x2	Write Address Size
M_AXI_AWBURST[1:0]	M_AXI	O	0x1	Write Address Burst
M_AXI_AWLOCK	M_AXI	O	0	Write Address Lock
M_AXI_AWCACHE[3:0]	M_AXI	O	0x3	Write Address Cache
M_AXI_AWPROT[2:0]	M_AXI	O	0x2	Write Address Protection
M_AXI_AWQOS[3:0]	M_AXI	O	0x0	Write Address QoS
M_AXI_AWVALID	M_AXI	O	0	Write Address Valid
M_AXI_AWREADY	M_AXI	I	-	Write Address Ready
M_AXI_WDATA[C_M_AXI_DATA_WIDTH-1:0]	M_AXI	O	0x0	Write Data
M_AXI_WSTRB[C_M_AXI_DATA_WIDTH/8-1:0]	M_AXI	O	0x0	Write Strobes
M_AXI_WLAST	M_AXI	O	0	Write Last
M_AXI_WVALID	M_AXI	O	0	Write Valid
M_AXI_WREADY	M_AXI	I	-	Write Ready
M_AXI_BRESP[1:0]	M_AXI	I	-	Write Response

Table 2-2: **MDM I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
M_AXI_BID[C_M_AXI_THREAD_ID_WIDTH-1:0]	M_AXI	I	-	Write Response ID
M_AXI_BVALID	M_AXI	I	-	Write Response Valid
M_AXI_BREADY	M_AXI	O	0	Write Response Ready
M_AXI_ARID[C_M_AXI_THREAD_ID_WIDTH-1:0]	M_AXI	O	0x0	Read Address ID
M_AXI_ARADDR[C_M_AXI_ADDR_WIDTH-1:0]	M_AXI	O	0x0	Read Address
M_AXI_ARLEN[7:0]	M_AXI	O	0x0	Read Address Length
M_AXI_ARSIZE[2:0]	M_AXI	O	0x0	Read Address Size
M_AXI_ARBURST[1:0]	M_AXI	O	0x1	Read Address Burst
M_AXI_ARLOCK	M_AXI	O	0	Read Address Lock
M_AXI_ARCACHE[3:0]	M_AXI	O	0x3	Read Address Cache
M_AXI_ARPROT[2:0]	M_AXI	O	0x2	Read Address Protection
M_AXI_ARQOS[3:0]	M_AXI	O	0x0	Read Address QoS
M_AXI_ARVALID	M_AXI	O	0	Read Address Valid
M_AXI_ARREADY	M_AXI	I	-	Read Address Ready
M_AXI_RID[C_M_AXI_THREAD_ID_WIDTH-1:0]	M_AXI	I	-	Read ID
M_AXI_RDATA[C_M_AXI_DATA_WIDTH-1:0]	M_AXI	I	-	Read Data
M_AXI_RRESP[1:0]	M_AXI	I	-	Read Response
M_AXI_RLAST	M_AXI	I	-	Read Last
M_AXI_RVALID	M_AXI	I	-	Read Valid
M_AXI_RREADY	M_AXI	O	0	Read Ready
LMB Master Interface Signals (n = 0 - 31)				
LMB_Data_Addr_n[0:C_DATA_SIZE-1]	LMB_n	O	0x0	Data Address
LMB_Data_Read_n[0:C_DATA_SIZE-1]	LMB_n	I	-	Data Read Bus
LMB_Data_Write_n[0:C_DATA_SIZE-1]	LMB_n	O	0x0	Data Write Bus
LMB_Addr_Strobe_n	LMB_n	O	0	Address Strobe
LMB_Read_Strobe_n	LMB_n	O	0	Read Strobe
LMB_Write_Strobe_n	LMB_n	O	0	Write Strobe
LMB_Ready_n	LMB_n	I	-	Ready
LMB_Wait_n	LMB_n	I	-	Wait
LMB_CE_n	LMB_n	I	-	Correctable Error
LMB_UE_n	LMB_n	I	-	Uncorrectable Error
LMB_Byte_Enable_n[0:(C_DATA_SIZE-1)/8]	LMB_n	O	0x0	Byte Enable
MicroBlaze Debug Interface Signals (n = 0 - 31)				
Dbg_Clk_n	MBDEBUG_n	O	0	MicroBlaze Debug Clock
Dbg_TDI_n	MBDEBUG_n	O	0	MicroBlaze Debug TDI

Table 2-2: **MDM I/O Signals** ***(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
Dbg_TDO_n	MBDEBUG_n	I	-	MicroBlaze debug TDO
Dbg_Reg_En_n	MBDEBUG_n	O	0	MicroBlaze debug register enable
Dbg_Capture_n	MBDEBUG_n	O	0	MicroBlaze debug capture
Dbg_Shift_n	MBDEBUG_n	O	0	MicroBlaze debug shift
Dbg_Update_n	MBDEBUG_n	O	0	MicroBlaze debug update
Dbg_Rst_n	MBDEBUG_n	O	0	MicroBlaze debug reset
External Cross Trigger Signals (n = 0,1,2,3)				
Trig_In_n		I	-	Cross trigger inputs
Trig_Ack_In_n		O	0	Cross trigger input acknowledge
Trig_Out_n		O	0	Cross trigger outputs
Trig_Ack_Out_n		I	-	Cross trigger output acknowledge
MicroBlaze Trace Core Interface Signals				
Ext_JTAG_DRCK	XMTC	O	0	Connection to MicroBlaze trace core
Ext_JTAG_RESET	XMTC	O	0	
Ext_JTAG_SEL	XMTC	O	0	
Ext_JTAG_CAPTURE	XMTC	O	0	
Ext_JTAG_SHIFT	XMTC	O	0	
Ext_JTAG_UPDATE	XMTC	O	0	
Ext_JTAG_TDI	XMTC	O	0	
Ext_JTAG_TDO	XMTC	I	-	
External BSCAN Interface Signals				
bscan_tdi	BSCAN	I	-	Connection to external BSCAN
bscan_reset	BSCAN	I	-	
bscan_shift	BSCAN	I	-	
bscan_update	BSCAN	I	-	
bscan_capture	BSCAN	I	-	
bscan_sel	BSCAN	I	-	
bscan_drck	BSCAN	I	-	
bscan_tdo	BSCAN	O	0	

Register Spaces

AXI Slave Register Space

Table 2-3 describes the MDM registers accessible through the AXI4-Lite slave interface.

Table 2-3: **MDM AXI4-Lite Slave Registers**

Register Name	Size (bits)	Address Offset	R/W	Description
JTAG UART Registers (C_USE_UART = 1)				
UART_RX_FIFO	8	0x00	R	JTAG UART receive data
UART_TX_FIFO	8	0x04	W	JTAG UART transmit data
UART_STATUS	8	0x08	R	JTAG UART status
UART_CTRL	8	0x0C	W	JTAG UART control
Debug Register Access (C_DBG_REG_ACCESS = 1)				
DBG_STATUS	1	0x10	R	Debug register access status bit 0 - Access lock acquired
DBG_CTRL	20	0x10	W	Debug register access control
DBG_DATA	32	0x14	R/W	Debug register access data Read data or Write data
DBG_LOCK	16	0x18	W	Debug register access lock

The JTAG UART registers are identical to the AXI UART Lite registers (see the *LogiCORE IP AXI UART Lite Product Guide* (PG142) [Ref 2]), except that the Status register bits 5 - 7 (Overrun Error, Frame Error, Parity Error) are never set, and the Control register bit 2 is not reserved.

UART Receive FIFO Register (UART_RX_FIFO)

This 16-entry-deep FIFO contains data received by the UART from JTAG. The FIFO bit definitions are shown in Table 2-5. Reading this register results in reading the data word from the top of the FIFO. When a read request is issued to an empty FIFO, a bus error (SLVERR) is generated and the result is undefined. The register is a read-only register. Issuing a write request to the receive data FIFO does nothing but generates a successful write acknowledgement. Table 2-4 shows the location for data on the AXI slave interface. The register is only implemented if C_USE_UART is set to 1.

Table 2-4: **UART Receive FIFO Register (UART_RX_FIFO)**

Reserved	UART_RX
31 — 8	7 — 0

Table 2-5: **UART Receive FIFO Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31:8	-	R	0	Reserved
7:0	UART_RX	R	0	UART Receive Data

UART Transmit FIFO Register (UART_TX_FIFO)

This 16-entry-deep FIFO contains data to be output by the UART using JTAG. The FIFO bit definitions are shown in Table 2-7. Data to be transmitted is written into this register. When a write request is issued while the FIFO is full, a bus error (SLVERR) is generated and the data is not written into the FIFO. This is a write-only location. Issuing a read request to the transmit data FIFO generates the read acknowledgement with zero data. Table 2-6 shows the location for data on the AXI interface. The register is only implemented if C_USE_UART is set to 1.

Table 2-6: **UART Transmit FIFO Register (UART_TX_FIFO)**

Reserved		UART_TX	
31	8	7	0

Table 2-7: **UART Transmit FIFO Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31:8	-	R	0	Reserved
7:0	UART_TX	R	0	UART Transmit Data

UART Status Register (UART_STATUS)

The status register contains the status of the receive and transmit data FIFOs, and when interrupt is enabled. This is a read only register. If a write request is issued to the status register it will do nothing but generate write acknowledgement. Bit assignment in the register is shown in Table 2-8 and described in Table 2-9. The register is only implemented if C_USE_UART is set to 1.

Table 2-8: **UART Status Register (UART_STATUS)**

Reserved		UART_STATUS	
31	5	4	0

Table 2-9: **UART Status Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-5	Reserved	N/A	0	Reserved
4	Interrupt Enabled	R	0	Indicates that interrupt is enabled. 0 = Interrupt is disabled 1 = Interrupt is enabled
3	TX FIFO Full	R	0	Indicates if the transmit FIFO is full. 0 = Transmit FIFO is not full 1 = Transmit FIFO is full
2	TX FIFO Empty	R	1	Indicates if the transmit FIFO is empty. 0 = Transmit FIFO is not empty 1 = Transmit FIFO is empty
1	RX FIFO Full	R	0	Indicates if the receive FIFO is full. 0 = Receive FIFO is not full 1 = Receive FIFO is full
0	RX FIFO Valid Data	R	0	Indicates if the receive FIFO has valid data. 0 = Receive FIFO is empty 1 = Receive FIFO has valid data

UART Control Register (UART_CTRL)

The control register contains the enable interrupt bit and reset for the receive and transmit data FIFO. This is write only register. Issuing a read request to the control register generates the read acknowledgement with zero data. Bit assignment in the register is shown in Table 2-10 and described in Table 2-11. The register is only implemented if C_USE_UART is set to 1.

Table 2-10: **UART Control Register (UART_CTRL)**

Reserved		UART_CTRL	
31	5	4	0

Table 2-11: **UART Control Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-5	Reserved	N/A	0	Reserved
4	Interrupt Enabled	W	0	Enable interrupt for the MDM JTAG UART 0 = Disable interrupt signal 1 = Enable interrupt signal
3	Reserved	N/A	0	Reserved
2	Clear EXT_BRK signal	W	0	Clear the EXT_BRK signal set by JTAG 0 = Do nothing 1 = Clear the signal

Table 2-11: **UART Control Register Bit Definitions *(Cont'd)***

Bits	Name	Access	Reset Value	Description
1	Reset RX FIFO	W	1	Reset/clear the receive FIFO Writing a 1 to this bit position clears the receive FIFO 0 = Do nothing 1 = Clear the receive FIFO
0	Reset TX FIFO	W	1	Reset/clear the transmit FIFO Writing a 1 to this bit position clears the transmit FIFO 0 = Do nothing 1 = Clear the transmit FIFO

Debug Register Access Status Register (DBG_STATUS)

The status register contains the status of the access lock. This is a read only register. Bit assignment in the register is shown in Table 2-12 and described in Table 2-13. The register is only implemented if C_DBG_REG_ACCESS is set to 1.

Table 2-12: **Debug Register Access Status Register (DBG_STATUS)**

Reserved	LOCK
31 ... 1	0

Table 2-13: **Debug Register Access Status Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-1	Reserved	N/A	0	Reserved
0	Access Lock	R	0	Indicates the access lock status. 0 = The lock is not acquired 1 = The lock has been acquired by the JTAG interface

Debug Register Access Control Register (DBG_CTRL)

The control register contains the enable interrupt bit and reset for the receive and transmit data FIFO. This is a write-only register. Issuing a read request to the control register generates the read acknowledgement with zero data. Writing to the register has no effect until unlocked using the DBG_LOCK register. Bit assignment in the register is shown in Table 2-14 and described in Table 2-15. The register is only implemented if C_DBG_REG_ACCESS is set to 1.

Table 2-14: **Debug Register Access Control Register (DBG_CTRL)**

Reserved	DBG_CTRL
31 ... 20	19 ... 0

Table 2-15: **Debug Register Access Control Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-20	Reserved	N/A	0	Reserved
19-18	Access Lock Type	W	00	Access lock type write: 00 = Release access lock to abort atomic sequence 01 = Lock before first data access and unlock after last 10 = Lock before first data access, otherwise keep lock 11 = Force lock acquisition, even if acquired by JTAG
17	Access MDM	W	0	Access MDM or MicroBlaze Debug register: 0 = MicroBlaze debug register access 1 = MDM debug register access
16-9	MDM Command	W	0	MDM command, see Table 2-18
8-0	Bit Size	W	0	Number of bits in the accessed debug register-1

Debug Register Access Data Register (DBG_DATA)

The data register is used to read data from or write data to the debug register indicated by the DBG_CTRL register. Accessing the register has no effect until unlocked using the DBG_LOCK register. Bit assignment in the register is shown in Table 2-16. The register is only implemented if C_DBG_REG_ACCESS is set to 1.

Table 2-16: **Debug Register Access Data Register (DBG_DATA)**

DBG_DATA	
31	0

Debug Register Access Locking Register (DBG_LOCK)

The lock register is used to unlock access to the AXI Debug register access registers DBG_CTRL and DBG_DATA by writing the magic value 0xebab. Writing any other value locks access. This is a write only register. Bit assignment in the register is shown in Table 2-17. The register is only implemented if C_DBG_REG_ACCESS is set to 1.

Table 2-17: **Debug Register Access Locking Register (DBG_LOCK)**

Reserved		DBG_LOCK	
31	16	15	0

MDM Interrupts

If the interrupt enable register bit in the JTAG UART control register is set, the UART raises the interrupt signal in the cycle when the TX FIFO goes empty, or in every cycle where the RX FIFO has data available.

Debug Register Space

The user-accessible MDM Debug registers are listed and described in Table 2-18. These registers are accessible using the Debug register access functionality. The values to write to the DBG_CTRL MDM register when accessing the MDM Debug registers are also listed in the table.

Additional MDM commands are used to access MicroBlaze extended debug features, as defined in the *MicroBlaze Processor Reference Guide* (UG984) [Ref 3].

All other MDM commands are reserved for Xilinx internal use, to handle the JTAG debug protocol, including the JTAG UART and AXI Memory Access From Debug.

Table 2-18: **MDM User-Accessible Debug Registers**

Register Name	Size (bits)	MDM Command	DBG_CTRL Value	R/W	Description
General					
MDM Configuration	32	00001100	0x6181F	R	The MDM configuration.
MDM Extended Configuration	35	00001100	0x61822	R	The MDM extended configuration.
Which MicroBlaze	8-32	00001101	8: 0x61A07 9: 0x61A08 ... 32: 0x61A1F	W	Which MicroBlaze processors to access. This register has variable bit size.
Cross Trigger Debug Registers (C_USE_CROSS_TRIGGER = 1)					
External Cross Trigger Control	10	01000000	0x68009	W	External cross trigger control register, used to set the trigger output source, and trigger input mask for each of the four triggers.
Cross Trigger Control	16	01000110	0x68C0F	W	Cross trigger control register, used to set the trigger output source, trigger input mask and logic function to combine inputs for each of the eight triggers.
Cross Trigger Status	24	01000010	0x68417	R	Cross trigger status register to read the current values of all trigger inputs and outputs.

MDM Configuration Debug Register

This register contains the current MDM configuration. This register is a read-only register. Issuing a write request to the register does nothing.

Table 2-19: **MDM Configuration Debug Register**

Magic	Res	Ext	UART	Width	Ports	Version
31 24	23	22	21	20 16	15 8	7 0

Table 2-20: **MDM Configuration Debug Register Bit Definitions**

Bits	Name	Access	Value	Description
31 - 24	Magic	R	0x42	Magic value 0x42
23	Reserved	R	1	Reserved
22	Extended	R	0, 1	Set to 1 if extended configuration register is available
21	UART	R	0, 1	Parameter C_USE_UART
20 - 16	Width	R	00111	JTAG UART character width - 1, 8 bit characters
15 - 8	Ports	R	1-32	Parameter C_MB_DBG_PORTS
7 - 0	Version	R	0x67	Version of MDM and JTAG UART protocol

MDM Extended Configuration Debug Register

This register contains the current extended MDM configuration. This register is a read-only register. Issuing a write request to the register does nothing. The register is only available if the MDM Configuration Debug register bit 22 is set.

Table 2-21: **MDM Extended Configuration Debug Register**

CT	RA	MA	Magic	Res	Ext	UART	Width	Ports	Version
34	33	32	31 24	23	22	21	20 16	15 8	7 0

Table 2-22: **MDM Extended Configuration Debug Register Bit Definitions**

Bits	Name	Access	Value	Description
34	Cross Trigger	R	0, 1	Parameter C_USE_CROSS_TRIGGER
33	Register Access	R	0, 1	Parameter C_DBG_REG_ACCESS
32	Memory Access	R	0, 1	Parameter C_DBG_MEM_ACCESS
31 - 0	N/A	R	N/A	See Table 2-20

Which MicroBlaze Debug Register

This register defines which MicroBlaze processor or processors are accessed. This register is a write-only register. Issuing a read request has no effect, and undefined data is read.

It is possible to write to the debug registers of more than one processor simultaneously, by setting more than one bit in this register. When reading from a debug register, only one bit must be set in this register, otherwise read data is undefined.

The register bit size is variable, and depends on the C_MB_DBG_PORTS parameter:

- When C_MB_DBG_PORTS set to 1, the register is not used, and the only connected processor is always accessed.
- When C_MB_DBG_PORTS ranges from 2 to 8, the register has 8 bits.
- When C_MB_DBG_PORTS is greater than 8, the register has the same number of bits as the parameter.

The processors are defined from the left in the register according to Table 2-24.

Table 2-23: **Which MicroBlaze Debug Register**

31	8	7	0

Table 2-24: **Which MicroBlaze Debug Register Bit Definitions**

C_MB_DBG_PORTS	Bits	Access	Reset Value	Description
2	7 - 6	W	0	Bit 7 = Access processor number 0 Bit 6 = Access processor number 1
3	7 - 5	W	0	Bit 7 = Access processor number 0 Bit 6 = Access processor number 1 Bit 5 = Access processor number 2
...	...	...	...	...
8	7 - 0	W	0	Bit 7 = Access processor number 0 ... Bit 0 = Access processor number 7
8 - 32	(n-1) - 0	W	0	Bit n-1 = Access processor number 0 ... Bit 0 = Access processor number n-1

External Cross Trigger Control Debug Register

This register defines the trigger source for each of the four external trigger outputs, and the mask for each of the four external trigger inputs. This register is a write-only register. Issuing a read request has no effect, and undefined data is read.

Table 2-25: **External Cross Trigger Control Debug Register**

Reserved		External Trigger	
31	10	9	0

Table 2-26: **External Cross Trigger Control Debug Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31 - 10	Reserved	N/A	0	Reserved
9 - 6	Output Select	W	See Notes(1)	Trigger output source for the selected external trigger output, according to Table 2-29.
5 - 2	Input Mask	W	1111	Mask bits for each external trigger input for the selected trigger output. Bits with value 0 do not affect the output. Bit 5 = External trigger input 0 ... Bit 2 = External trigger input 3
1 - 0	Index	W	00	Selected external trigger output to change: 00 = External Trigger 0 01 = External Trigger 1 10 = External Trigger 2 11 = External Trigger 3

Notes:

1. The reset value is 1 for External Trigger 0, 2 for External Trigger 1, 3 for External Trigger 2 and 4 for External Trigger 3.

Cross Trigger Control Debug Register

This register defines trigger source for each of the eight trigger outputs, and the mask for each of the eight trigger inputs. This register is a write-only register. Issuing a read request has no effect, and undefined data is read.

Writing the register affects the cross trigger outputs of the MicroBlaze processor or processors selected by the Which MicroBlaze Debug register.

See the *MicroBlaze Processor Reference Guide* (UG984) [Ref 3], for the definition of the eight MicroBlaze trigger inputs and outputs.

Table 2-27: **Cross Trigger Control Debug Register**

Reserved		Trigger	
31	16	15	0

Table 2-28: **Cross Trigger Control Debug Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31 - 16	Reserved	N/A	0	Reserved
15 - 12	Output Select	W	See Notes[1]	Trigger output source for the selected MicroBlaze trigger output, according to Table 2-29.
11 - 4	Input Mask	W	11111111	Mask bits for each MicroBlaze trigger input for the selected trigger output. Bits with value 0 do not affect the output. Bit 11 = MicroBlaze Trigger input 0 ... Bit 4 = MicroBlaze Trigger input 7
3	And/Or	W	0	Logic function to combine inputs: 0 = Logic or, triggers if any input is set 1 = Logic and, triggers only if all inputs are set
2 - 0	Index	W	000	Selected MicroBlaze trigger output to change: 000 = MicroBlaze Trigger output 0 ... 111 = MicroBlaze Trigger output 7

Notes:

1. The reset value is 9 for Trigger 0, 10 for Trigger 1, 11 for Trigger 2, 12 for Trigger 3, and 13 for Trigger 4-7.

Table 2-29: **Cross Trigger Output Select Field Definition**

Field Value	Name	Description
0000	Static One	Static one, no input selected
0001 - 1000	Trigger Input	0001 = Select MicroBlaze Trigger Input 0 ... 1000 = Select MicroBlaze Trigger Input 7
1001 - 1100	External Trigger Input	1001 = Select External Trigger Input 0 ... 1100 = Select External Trigger Input 3
1101	Static Zero	Static zero, no input selected
1110 - 1111	Reserved	Reserved

Cross Trigger Status Debug Register

This register contains the current trigger input and output values. This register is a read-only register. Issuing a write request to the register does nothing.

The MicroBlaze cross trigger status (bits 0 to 15) represents the MicroBlaze processor selected by the Which MicroBlaze Debug register. If more than one processor is selected, the result is undefined.

Table 2-30: **Cross Trigger Status Debug Register**

Reserved		Trigger Status	
31	24	23	0

Table 2-31: **Cross Trigger Status Debug Register Bit Definitions**

Bits	Name	Access	Description
31 - 24	Reserved	N/A	Reserved
23 - 16	Trigger Outputs	R	Current selected MicroBlaze processor trigger output values. Bit 23 = Trigger 0 ... Bit 16 = Trigger 7
15 - 8	Trigger Inputs	R	Current selected MicroBlaze processor trigger input values. Bit 15 = Trigger 0 ... Bit 8 = Trigger 7
7 - 4	External Trigger Outputs	R	Current external trigger input values. Bit 7 = External Trigger Output 0 ... Bit 4 = External Trigger Output 3
3 - 0	External Trigger Inputs	R	Current external trigger input values. Bit 3 = External Trigger Input 0 ... Bit 0 = External Trigger Input 3

Designing with the Core

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

General Design Guidelines

The MicroBlaze™ Debug Module supports multiple MicroBlaze cores, making it possible to use one MDM for multiprocessor systems with up to 32 processors.

The MDM outputs, `Ext_BRK` and `Ext_NM_BRK`, are not currently used, and need not be connected to MicroBlaze.

When using the JTAG-based UART, the MDM Interrupt output can be connected to an interrupt controller to provide interrupt-driven serial output. If this signal is not connected, only polled-mode serial output is available.

When using the JTAG memory access, the AXI4 Master should be connected to access the system memory accessible by the connected MicroBlaze cores, usually through the processor cache interfaces M_AXI_DC and M_AXI_IC. Depending on the AXI interconnect topology, this might exclude peripheral I/O accessed through the processor peripheral interface M_AXI_DP. The LMB Master ports should be connected to access the LMB local memory of the corresponding processor. This can be done by adding an additional LMB slave input to the data-side LMB BRAM Interface Controller (see the *LogiCORE IP LMB BRAM Interface Controller Product Guide* (PG112) [Ref 4]).

Clocking

The `S_AXI_ACLK` input is only used when the JTAG-based UART or Debug register access is enabled, and AXI4-Lite slave interconnect is used. Then it should normally be set to the same clock as the interconnect.

The M_AXI_ACLK input is only used when JTAG Memory Access is enabled, and AXI4 master interconnect and/or LMB master interface is used. Then it must be set to the same clock as the interconnect and LMB interface. Different clocks for AXI4 and LMB is not supported in this case.

Apart from the JTAG-based UART, Debug register access and JTAG memory access, the MDM is clocked from the BSCAN, with a clock frequency determined by the JTAG connection.

When programming a System ACE™ device, the MDM clock must be at least twice as fast as the System ACE tool controller clock for the ELF file to load correctly.

Resets

The `Debug_SYS_Rst` output can be used to reset the entire embedded system on the FPGA, including all processors and peripherals. Normally it is connected to a `proc_sys_reset` IP core. The Xilinx® Microprocessor Debugger (XMD) command `rst` can be used to activate the signal.

The Debug bus connecting each individual MicroBlaze processor handled by the MDM, has the `Dbg_Rst` reset signal. This signal can be used to just reset an individual processor. The XMD command `rst -processor` can be used to activate the signal for the selected target processor.

The `S_AXI_ARESETN` input is only used when the JTAG-based UART or Debug register access is enabled, and AXI4-Lite slave interconnect is used. Then it should normally be set to the same reset as the interconnect.

The `M_AXI_ARESETN` input is only used when JTAG Memory Access is enabled, and AXI4 master interconnect and/or LMB master interface is used. Then it must use the same reset as the interconnect. A corresponding reset for LMB must also be used.

Debug Register Access Sequence

Before starting to use debug register access functionality, the DBG_LOCK register must be written with the magic value (see Table 2-17) to enable access to the DBG_CTRL and DBG_DATA registers. By writing any other value to the DBG_LOCK register, access is again prohibited.

To access a single debug data register in a MicroBlaze processor or the MDM the following steps should be used:

1. Set the DBG_CTRL register value (see Table 2-15) to define the debug register size, the 8-bit MDM command, whether the register is an MDM or MicroBlaze debug register, and the Access Lock type 01 (lock before first data access, and unlock after last data access). The DBG_CTRL values for all user accessible debug registers are defined in Table 2-18.

2. Read back the DBG_STAT register (see Table 2-13), and make sure the lock has been acquired. If not, go back to step 1.
3. Read or write data using the DBG_DATA register. The number of 32-bit accesses must correspond to the register bit size. If the bit size is 32 bits or less, one access is required, if it is 64 bits or less, two accesses are required, and so on.

To perform a sequence of atomic accesses to debug data registers in a MicroBlaze processor or the MDM, for example, to write a command followed by reading the corresponding data, the following steps should be used:

1. Set the DBG_CTRL register value (see Table 2-15) to define the debug register size, the 8-bit MDM command, whether the register is an MDM or MicroBlaze debug register, and the Access Lock type 10 (lock before first data access, and keep lock).
2. Read back the DBG_STAT register (see Table 2-13), and make sure lock is acquired. If not, go to step 1.
3. Read or write data using the DBG_DATA register. The number of 32-bit accesses must correspond to the register bit size.
4. Repeat from step 1, until the last register to be accessed is reached.
5. Set the DBG_CTRL register value to define the debug register size, the 8-bit MDM command, whether the register is an MDM or MicroBlaze debug register, and the Access Lock type 01 (lock before first data access, and unlock after last data access) for the last register.
6. Read back the DBG_STAT register, and ensure that lock is still acquired.
7. Read or write data using the DBG_DATA register.

Care must be taken to always release the access lock, because JTAG is not able to access the debug registers while it is set. If a sequence of atomic accesses needs to be aborted, the access lock can be released by writing access lock type 00 to the DBG_CTRL register.

Cross Trigger Programming

The MDM provides a programmable cross trigger crossbar that allows the eight trigger inputs of each connected processor and the four external trigger inputs to affect each of the eight trigger outputs of each connected processor and four external trigger outputs.

IMPORTANT: *In order to use the MDM cross trigger functionality, all connected MicroBlaze processors must be configured to use Extended Debug (C_DEBUG_ENABLED = 2).*

In the simplest use case no programming is necessary, because the initial crossbar configuration is set up to:

- Connect the first four trigger inputs (Trigger Input 0 - 3) of the MicroBlaze processors to the external trigger outputs (External Trigger Output 0 - 3).
- Connect the external trigger inputs (External Trigger Input 0 - 3) to the first four trigger outputs (Trigger Output 0 - 3) of the MicroBlaze processors.

See the *MicroBlaze Processor Reference Guide* (UG984) [Ref 3], for the definition of the MicroBlaze trigger inputs and outputs.

The crossbar is programmed per output and connected processor. The following sequence of debug register accesses can be used to select and program a trigger output of a specific MicroBlaze processor:

1. If there are more than one MicroBlaze processor, use the Which MicroBlaze Debug register (see Table 2-24) to select which processor to affect. If the trigger outputs of more than one processor in the crossbar should be connected the same way, all those processors can be selected for simultaneous write.
2. For each of the 8 trigger outputs to define, write the Cross Trigger Control register (see Table 2-28) with the appropriate value, setting the register Index field to the trigger output number.
3. For each of the 4 external trigger outputs to define, write the External Cross Trigger Control register (see Table 2-26) with the appropriate value, setting the register Index field to the external trigger output number.

To force a trigger output, a Cross Trigger Control register can be written twice, first to set the Output Select field to Static One, and then to restore the previous value.

Protocol Description

The MDM Debug protocol is Xilinx internal, and not described in this document. All the details of the protocol are handled transparently to the user by the Xilinx Microprocessor Debugger (XMD) or by the Software Development Kit (SDK) System Debugger.

Design Flow Steps

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

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 7]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 8]

Customizing and Generating the Core

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

If you are customizing and generating the core in the Vivado IP Integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5] for detailed information. IP Integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

Vivado Integrated Design Environment

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

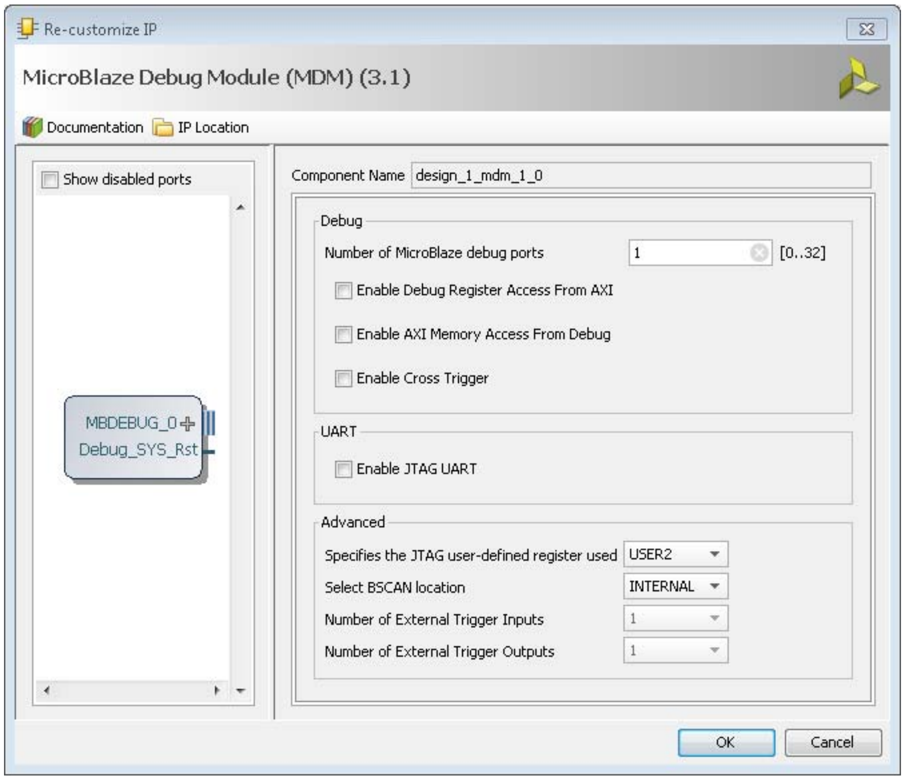
1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 7].

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

The MDM parameters are divided into three categories: Debug, UART and Advanced. When using the Vivado IP Integrator, the address for the JTAG-based UART is auto-generated.

The Customize IP dialog box is shown in Figure 4-1.



Component Name design_1_mdm_1_0

Debug

Number of MicroBlaze debug ports 1 [0..32]

Enable Debug Register Access From AXI

Enable AXI Memory Access From Debug

Enable Cross Trigger

UART

Enable JTAG UART

Advanced

Specifies the JTAG user-defined register used USER2

Select BSCAN location INTERNAL

Number of External Trigger Inputs 1

Number of External Trigger Outputs 1

Figure 4-1: **Customize IP Dialog Box**

- **Number of MicroBlaze debug ports** - Sets the number of ports available to connect to MicroBlaze processors.
- **Enable Debug Register Access From AXI** - Enables the functionality to access JTAG Debug Registers from AXI and the AXI4-Lite slave interface.
- **Enable AXI Memory Access From Debug** - Enables the functionality to access memory directly from JTAG, using the AXI4 master interface and the LMB master interfaces.
- **Enable Cross Trigger** - Enables the cross trigger functionality, and the external cross-trigger interfaces.
- **Enable JTAG UART** - Enables the JTAG UART and the AXI4-Lite slave interface to access the UART registers.

- **Specify the JTAG user-defined register used** - Select JTAG user-defined register. Can be set to *USER1*, *USER2*, *USER3* or *USER4*. Should never need to be changed from *USER2*, unless there is a conflict with another IP core in the system.
- **Select BSCAN location** - Selects whether internal or external BSCAN is used. Should never need to be changed from *INTERNAL* in an embedded system.
- **Number of External Trigger Inputs** - Defines the number of external trigger input interfaces to use when cross trigger is enabled.
- **Number of External Trigger Outputs** - Defines the number of external trigger output interfaces to use when cross trigger is enabled.

Parameter Values

To allow the user to obtain an MDM that is uniquely tailored a specific system, certain features can be parameterized in the MDM design. This allows the user to configure a design that only utilizes the resources required by the system. The features that can be parameterized in MDM designs are shown in Table 4-1.

Table 4-1: **MDM Design Parameters**

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
System Parameters				
Target FPGA family	C_FAMILY	See LogiCORE™ IP Facts Table	virtex7	string
Debug Parameters				
Number of MicroBlaze debug ports	C_MB_DBG_PORTS	0-32	1	integer
Enable Debug register access from AXI	C_DBG_REG_ACCESS	0,1	0	integer
Enable AXI memory access from debug	C_DBG_MEM_ACCESS	0,1	0	integer
Enable cross trigger	C_USE_CROSS_TRIGGER	0,1	0	integer
UART Parameters				
Enable JTAG UART	C_USE_UART	0,1	1	integer
Advanced Parameters				
Specifies the JTAG user-defined register used	C_JTAG_CHAIN	1 = USER1 2 = USER2 3 = USER3 4 = USER4	2	integer
Select BSCAN location	C_USE_BSCAN	0 = INTERNAL 2 = EXTERNAL	0	integer

Table 4-1: **MDM Design Parameters *(Cont'd)***

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Number of external trigger inputs	C_TRIG_IN_PORTS	0,1,2,3,4	1	integer
Number of external trigger outputs	C_TRIG_OUT_PORTS	0,1,2,3,4	1	integer

In addition to the parameters listed in this table, there are also parameters that are inferred for each AXI interface in the tools. Through the design, these inferred parameters control the behavior of the AXI Interconnect. For a complete list of the interconnect settings related to the AXI interface, see the *LogiCORE AXI Interconnect IP Product Guide* (PG059) [Ref 9].

Allowable Parameter Combinations

There are no restrictions on parameter combinations for this core.

Output Generation

The following files are generated by the IP in Vivado IP Integrator:

- Verilog/VHDL Template
- VHDL source files
- VHDL wrapper file in the library work

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

The MDM Debug logic is fully synchronous with the BSCAN module. Internally BUFG primitives are instantiated to buffer the DRCK clock from the BSCAN module.

The MDM JTAG-based UART and Debug register access is fully synchronous to the AXI4-Lite slave bus interface clock, and isolated from the BSCAN clock region with FIFOs and synchronization flip-flops.

The JTAG Memory Access AXI4 master interface and LMB master interfaces are fully synchronous to the AXI4 master clock, and isolated from the BSCAN clock region with FIFOs and synchronization flip-flops.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Migrating and Upgrading

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

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 10].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Debugging

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

Finding Help on Xilinx.com

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

Documentation

This product guide is the main document associated with the MDM core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx® Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Answer Records

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

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

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

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

Master Answer Record for the MDM Core

AR: 54413

Contacting 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.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Additional Resources.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Note: Access to WebCase is not available in all cases. Log in to the WebCase tool to see your specific support options.

Debug Tools

The main tools available to address MDM design issues are the Vivado lab tools.

Vivado Lab Tools

Vivado® lab tools insert logic analyzer and virtual I/O cores directly into your design. Vivado lab tools also allow you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

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

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

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 11].

Reference Boards

All 7 series and UltraScale™ architecture Xilinx development boards support the MDM. These boards can be used to prototype designs and establish that the core can communicate with the system.

Simulation Debug

The simulation debug flow for Questa SIM is described below. A similar approach can be used with other simulators.

- Check for the latest supported versions of Questa SIM in the Xilinx Design Tools: Release Notes Guide. Is this version being used? If not, update to this version.
- If using Verilog, do you have a mixed mode simulation license? If not, obtain a mixed-mode license.
- Ensure that the proper libraries are compiled and mapped. In the Vivado Design Suite **Flow > Simulation Settings** can be used to define the libraries.
- Have you associated the intended software program for the MicroBlaze™ processor with the simulation? Use the command **Tools > Associate ELF Files** in Vivado Design Suite.
- When observing the traffic on the interfaces connected to the MDM, see the timing in the relevant specification:
    - For AXI4 and AXI4-Lite, see the *AMBA® AXI and ACE Protocol Specification* [Ref 1].

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado Lab Tools are valuable resources to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado Lab Tools for debugging the specific problems.

Many of these common issues can also be applied to debugging design simulations.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.

Interface Debug

AXI4-Lite Slave Interface

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `S_AXI_ARREADY` asserts when the read address is valid, and output `S_AXI_RVALID` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `S_AXI_ACLK` input is connected and toggling.
- The interface is not being held in reset, and `S_AXI_ARESETN` is an active-Low reset.
- If the simulation has been run, verify in simulation and/or a Vivado Lab Tools debugging tool capture that the waveform is correct for accessing the AXI4-Lite interface.

Application Software Development

Device Drivers

The MDM JTAG-based UART interface is supported by the UART Lite driver, included with Xilinx Software Development Kit.

Additional Resources and Legal Notices

Xilinx Resources

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

For a glossary of technical terms used in Xilinx documentation, see the Xilinx Glossary.

References

These documents provide supplemental material useful with this user guide:

1. *AMBA AXI and ACE Protocol Specification* (ARM IHI 0022E)
2. *LogiCORE IP AXI UART Lite Product Guide* (PG142)
3. *MicroBlaze Processor Reference Guide* (UG984)
4. *LogiCORE IP LMB BRAM Interface Controller Product Guide* (PG112)
5. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
6. *Vivado Design Suite User Guide, Designing with IP* (UG896)
7. *Vivado Design Suite User Guide: Getting Started* (UG910)
8. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
9. *LogiCORE AXI Interconnect IP Product Guide* (PG059)
10. *ISE® to Vivado® Design Suite Migration Methodology Guide* (UG911)
11. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/02/2014	3.1	Document version number advanced to match the core version number. Updated due to core revision. Added description of debug enhancements.
03/20/2013	1.0	This Product Guide replaces PG062. There are no documentation changes for this release.

Please Read: Important Legal Notices