Processor Local Bus (PLB) Arbiter Design Specification

2/27/02

Summary

This document will provide the design specification for the Processor Local Bus (PLB) arbiter.

plb_arbiter	

Introduction

The Xilinx 64-bit Processor Local Bus (PLB) arbiter consists of a bus control unit, a watchdog timer, and separate address, write, and read data path units with a three-cycle-only arbitration feature. It contains a DCR slave interface to provide access to its bus error status registers.

The IBM Processor Local Bus (PLB) 64-Bit Architecture Specification and the IBM Processor Local Bus (PLB) 64-Bit Arbiter Core User's Manual will referenced throughout this document. Differences between the IBM PLB Arbiter and the Xilinx PLB Arbiter are highlighted and explained in **Specification Exceptions**, page 65.

PLB Arbiter Overview

Features

The Xilinx PLB Arbiter is a soft IP core designed for Xilinx FPGAs and contains the following features:

- PLB arbitration support for up to 16 masters
  - number of PLB masters is configurable via a design parameter
- PLB address and data steering support for up to 16 masters
- 64-bit and/or 32-bit support for masters and slaves
- PLB address pipelining
- Three cycle arbitration
- Four levels of dynamic master request priority
- PLB watchdog timer
- PLB architecture compliant
- Provides complete PLB bus structure
  - supports up to 16 slaves
    - number of PLB slaves is configurable via a design parameter
  - no external OR gates required for PLB slave input signals

PLB Bus Interconnect

The Xilinx PLB Arbiter consists of a central bus arbiter, the necessary bus control and gating logic, and all necessary bus OR/MUX structures. The Xilinx PLB arbiter provides the entire PLB

bus structure and allows for direct connection for up to 16 masters and 16 slaves. Figure 1 shows an example of the PLB connections for a system with three masters and three slaves.

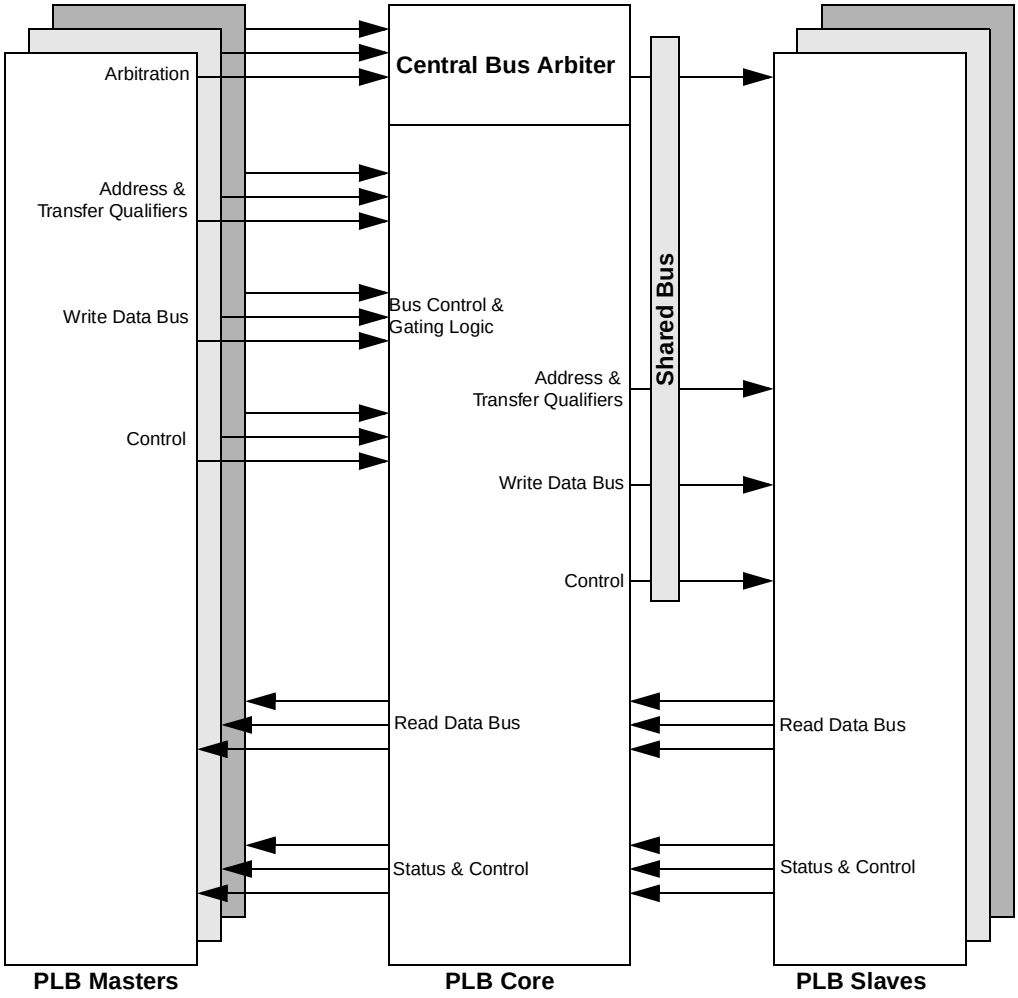


Figure 1: **PLB Interconnect Diagram**

Basic Operation

The Xilinx PLB Arbiter has 3-cycle arbitration during the address phase of the transaction as illustrated in Figure 2.

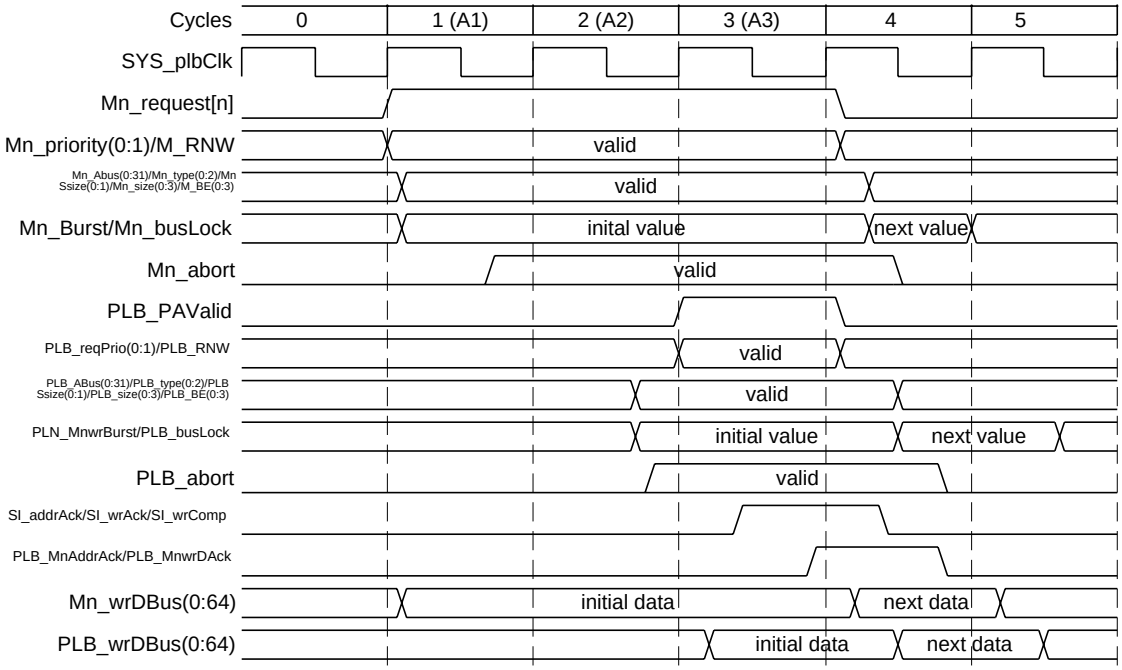


Figure 2: **3-cycle arbitration (Xilinx Implementation)**

During the bus arbitration cycle, the bus arbitration control unit uses the **M_priority[n*2:n*2+1]** signals to determine which master will be granted the bus. Specifically, only the priority inputs of masters with their respective **M_request[n]** signal asserted are used in determining the highest request priority. In addition, the Xilinx PLB Arbiter supports the fixed priority scheme to handle “tie” situations (that is, situations when two or more masters request the bus simultaneously while presenting the same level of request priority). Selection of the priority mode during tie situations is shown in Table 1 where n = C_NUM_MASTERS-1.

Table 1: **Priority Order for Bus Masters**

Highest Priority	Decreasing Priority			Lowest Priority
Master 0	Master 1	...	Master $_{n-1}$	Master $_{n}$

PLB Arbiter Design Parameters

To allow the user to obtain a PLB Arbiter that is uniquely tailored for their system, certain features are parameterizable in the Xilinx PLB Arbiter design. This allows the user to have a design that only utilizes the resources required by their system and runs at the best possible

performance. The features that are parameterizable in the Xilinx PLB Arbiter are shown in Table 2.

Table 2: **PLB Arbiter Design Parameters**

Grouping/Number		Feature/Description	Parameter Name	Allowable Values	Default Value	VHDL Type
PLB Features	G1	Number of PLB Masters	C_NUM_MASTERS	1 - 16	4	integer
	G2	Number of PLB Slaves	C_NUM_SLAVES	1 - 16	8	
	G3	PLB Address Bus Width	C_PLB_AWIDTH	32	32	integer
	G4	PLB Data Bus Width	C_PLB_DWIDTH	32, 64	64	integer
	G5	Include DCR interface	C_DCR_INTFCE	1 = Include DCR slave interface 0 = DCR slave interface not included	1	integer
DCR Interface	G6	DCR Base Address	C_BASEADDR	Valid DCR address(1)	None(2)	std_logic_vector
	G7	DCR High Address	C_HIGHADDR	Valid DCR address(1)	None(2)	std_logic_vector
	G8	DCR Address Bus Width	C_DCR_AWIDTH	10	10	integer
	G9	DCR Data Bus Width	C_DCR_DWIDTH	32	32	integer
Interrupts	G10	Active Interrupt State(3)	C_IRQ_ACTIVE	'0' = interrupt request will be driven as a falling edge '1' = interrupt request will be driven as a rising edge	'1'	std_logic

Notes:
1. The range specified by C_BASEADDR and C_HIGHADDR must comprise a complete, contiguous power of two range such that range = 2^n, and the n least significant bits of C_BASEADDR must be zero. To allow for the 8 DCR registers within the PLB Arbiter, n must be at least 3.
2. No default value will be specified for C_BASEADDR or C_HIGHADDR to insure that the actual value is set, i.e. if the value is not set, a compiler error will be generated.
3. The interrupt request output (Bus_Error_Det, P72) is generated as an edge type interrupt. A specific interrupt acknowledge response is not required.

Allowable Parameter Combinations

The address range specified by C_BASEADDR and C_HIGHADDR must comprise a complete, contiguous power of two range such that range = 2^n, and the n least significant bits of C_BASEADDR must be zero. To allow for the registers in the PLB Arbiter design, this range must be at least 7; therefore n must be at least 3. This means that at a minimum, the three least significant bits of C_BASEADDR must be 0.

The base address and high address parameters determine the number of most significant address bits used to decode the address space. These parameters allow the user to trade-off address space resolution with size and speed of the arbiter.

Note that if the width of the PLB address bus is greater than the width of the DCR data bus, reading the PLB Error Address Register (PEAR_ADDR) will only return that portion of the most significant bits of the offending address that will fit into a DCR data word.

Some parameters can cause other parameters to be irrelevant. See Table 4 for information on the relationship between design parameters.

PLB Arbiter I/O Signals

Table 3 provides a summary of all Xilinx PLB Arbiter input/output (I/O) signals, the interfaces under which they are grouped, and a brief description of the signal.

Table 3: **PLB Arbiter Pin Description**

Grouping		Signal Name	Interface	I/O	Initial State	Description	Page
DCR Signals	P1	DCR_ABus[0:C_DCR_AWIDTH-1]	DCR	I		CPU DCR address bus	
	P2	DCR_Read	DCR	I		CPU read from DCR indicator	
	P3	DCR_Write	DCR	I		CPU write to DCR indicator	
	P4	DCR_DBus[0:C_DCR_DWIDTH-1]	DCR	I		DCR write data bus	
	P5	PLB_dcrAck	DCR	O	0	PLB DCR data transfer acknowledge	
	P6	PLB_dcrDBus[0:C_DCR_DWIDTH-1]	DCR	O	0	PLB DCR read data bus	
PLB Status Signals	P7	PLB_pendPri[0:1]	Master/Slave	O	0	PLB pending request priority	
	P8	PLB_pendReq	Master/Slave	O	0	PLB pending bus request indicator	
	P9	PLB_reqPri[0:1]	Master/Slave	O	0	PLB current request priority	

Table 3: **PLB Arbiter Pin Description *(Continued)***

Grouping		Signal Name	Interface	I/O	Initial State	Description	Page
Master SIgnals	P10	M_abort[0:C_NUM_MASTERS-1]	Master	I		Master abort bus request indicator	
	P11	M_ABus[0:C_NUM_MASTERS*C_PLB_AWIDTH-1]	Master	I		Master address bus	
	P12	M_BE[0:C_NUM_MASTERS*C_PLB_DWIDTH/8-1]	Master	I		Master byte enables	
	P13	M_busLock[0:C_NUM_MASTERS-1]	Master	I		Master bus lock	
	P14	M_compress[0:C_NUM_MASTERS-1]	Master	I		Master compressed data transfer indicator	
	P15	M_guarded[0:C_NUM_MASTERS-1]	Master	I		Master guarded transfer indicator	
	P16	M_lockErr[0:C_NUM_MASTERS-1]	Master	I		Master lock error indicator	
	P17	M_mSize[0:C_NUM_MASTERS*2 -1]	Master	I		Master data bus port width	
	P18	M_ordered[0:C_NUM_MASTERS-1]	Master	I		Master synchronize transfer indicator	
	P19	M_priority[0:C_NUM_MASTERS*2 -1]	Master	I		Master bus request priority	
	P20	M_rdBurst[0:C_NUM_MASTERS-1]	Master	I		Master burst read transfer indicator	
	P21	M_request[0:C_NUM_MASTERS-1]	Master	I		Master bus request	
	P22	M_RNW[0:C_NUM_MASTERS-1]	Master	I		Master read not write	
	P23	M_size[0:C_NUM_MASTERS*4 -1]	Master	I		Master transfer size	
	P24	M_type[0:C_NUM_MASTERS*3-1]	Master	I		Master transfer type	
	P25	M_wrBurst[0:C_NUM_MASTERS-1]	Master	I		Master burst write transfer indicator	
	P26	M_wrDBus[0:C_NUM_MASTERS*C_PLB_DWIDTH -1]	Master	I		Master write data bus	
	P27	PLB_MAddrAck[0:C_NUM_MASTERS-1]	Master	O	0	PLB Master address acknowledge	
	P28	PLB_MBusy[0:C_NUM_MASTERS-1]	Master	O	0	PLB Master slave busy indicator	
	P29	PLB_MErr[0:C_NUM_MASTERS-1]	Master	O	0	PLB Master slave error indicator	
	P30	PLB_MRdBTerm[0:C_NUM_MASTERS-1]	Master	O	0	PLB Master terminate read burst indicator	
	P31	PLB_MRdDAck[0:C_NUM_MASTERS-1]	Master	O	0	PLB Master read data acknowledge	
	P32	PLB_MRdDBus[0:C_NUM_MASTERS*C_PLB_DWIDTH -1]]	Master	O	0	PLB Master read data bus	
	P33	PLB_MRdWdAddr[0:C_NUM_MASTERS*4 -1]	Master	O	0	PLB Master read word address	
	P34	PLB_MRearbitrate[0:C_NUM_MASTERS-1]	Master	O	0	PLB Master bus rearbitrate indicator	
	P35	PLB_MSSize[0:C_NUM_MASTERS*2 -1]	Master	O	0	PLB Master slave data bus port width	
	P36	PLB_MWrBTerm[0:C_NUM_MASTERS-1]	Master	O	0	PLB Master terminate write burst indicator	
	P37	PLB_MWrDAck[0:C_NUM_MASTERS-1]	Master	O	0	PLB Master write data acknowledge	
Slave Signals	P38	PLB_abort	Slave	O	0	PLB abort bus request indicator	
	P39	PLB_ABus[0:C_PLB_AWIDTH-1]	Slave	O	0	PLB address bus	
	P40	PLB_BE[0:C_PLB_DWIDTH/8-1]	Slave	O	0	PLB byte enables	
	P41	PLB_busLock	Slave	O	0	PLB bus lock	
	P42	PLB_compress	Slave	O	0	PLB compressed data transfer indicator	
	P43	PLB_guarded	Slave	O	0	PLB guarded transfer indicator	
	P44	PLB_lockErr	Slave	O	0	PLB lock error indicator	

Table 3: **PLB Arbiter Pin Description *(Continued)***

Grouping		Signal Name	Interface	I/O	Initial State	Description	Page
	P45	PLB_masterID[0:log2(C_NUM_MASTERS)-1]	Slave	O	0	PLB current master identifier	
	P46	PLB_MSize[0:1]	Slave	O	0	PLB data bus port width indicator	
	P47	PLB_ordered	Slave	O	0	PLB synchronize transfer indicator	
	P48	PLB_PAValid	Slave	O	0	PLB primary address valid indicator for up to 66 MHz	
	P49	PLB_rdBurst	Slave	O	0	PLB burst read transfer indicator	
	P50	PLB_rdPrim	Slave	O	0	PLB secondary to primary read request indicator	
	P51	PLB_RNW	Slave	O	0	PLB read not write	
	P52	PLB_SAValid	Slave	O	0	PLB secondary address valid for up to 66 MHz	
	P53	PLB_size[0:3]	Slave	O	0	PLB transfer size	
	P54	PLB_type[0:2]	Slave	O	0	PLB transfer type	
	P55	PLB_wrBurst	Slave	O	0	PLB burst write transfer indicator	
	P56	PLB_wrDBus[0:C_PLB_DWIDTH-1]	Slave	O	0	PLB write data bus	
	P57	PLB_wrPrim	Slave	O	0	PLB secondary to primary write request indicator	
	P58	Sl_addrAck[0:C_NUM_SLAVES-1]	Slave	I		Slave address acknowledge	
	P59	Sl_MErr[0:C_NUM_SLAVES*C_NUM_MASTERS-1]	Slave	I		Slave error indicator	
	P60	Sl_MBusy[0:C_NUM_SLAVES*C_NUM_MASTERS-1]	Slave	I		Slave busy indicator	
	P61	Sl_rdBTerm[0:C_NUM_SLAVES-1]	Slave	I		Slave terminate read burst transfer	
	P62	Sl_rdComp[0:C_NUM_SLAVES-1]	Slave	I		Slave read transfer complete indicator	
	P63	Sl_rdDAck[0:C_NUM_SLAVES-1]	Slave	I		Slave read data acknowledge	
	P64	Sl_rdDBus[0:C_NUM_SLAVES*C_PLB_DWIDTH-1]	Slave	I		Slave read data bus	
	P65	Sl_rdWdAddr[0:C_NUM_SLAVES*4-1]	Slave	I		Slave read word address	
	P66	Sl_rearbitrate[0:C_NUM_SLAVES-1]	Slave	I		Slave rearbitrate bus indicator	
	P67	Sl_SSize[0:C_NUM_SLAVES*2-1]	Slave	I		Slave data bus port size indicator	
	P68	Sl_wait[0:C_NUM_SLAVES-1]	Slave	I		Slave wait indicator	
	P69	Sl_wrBTerm[0:C_NUM_SLAVES-1]	Slave	I		Slave terminate write burst transfer	
	P70	Sl_wrComp[0:C_NUM_SLAVES-1]	Slave	I		Slave write transfer complete indicator	
	P71	Sl_wrDAck[0:C_NUM_SLAVES-1]	Slave	I		Slave write data acknowledge	
Interrupt	P72	Bus_Error_Det	System	O	0	Bus Error Interrupt	
System Signals	P73	Clk	System	I		System clock	
	P74	Rst	System	I		Reset	
	P75	ArbReset	System	O	0	Registered reset output from arbiter	
	P76	ArbAddrVldReg	System	O	0	Valid address	

Table 3: **PLB Arbiter Pin Description *(Continued)***

Grouping		Signal Name	Interface	I/O	Initial State	Description	Page
IBM Toolkit Support(1)	P77	PLB_SaddrAck	Simulation	O	0	Output of slave Sl_addrAck OR gate	
	P78	PLB_Swait	Simulation	O	0	Output of slave Sl_wait OR gate	
	P79	PLB_Srearbitrate	Simulation	O	0	Output of slave Sl_rearbitrate OR gate	
	P80	PLB_SwrDAck	Simulation	O	0	Output of slave Sl_wrDAck OR gate	
	P81	PLB_SwrComp	Simulation	O	0	Output of slave Sl_wrComp OR gate	
	P82	PLB_SwrBTerm	Simulation	O	0	Output of slave Sl_wrBTerm OR gate	
	P83	PLB_SrdDBus[0:C_PLB_DWIDTH-1]	Simulation	O	0	Output of slave Sl_rdDBus OR gate	
	P84	PLB_SrdWdAddr[0:3]	Simulation	O	0	Output of slave Sl_rdWdAddr OR gate	
	P85	PLB_SrdDAck	Simulation	O	0	Output of slave Sl_rdDAck OR gate	
	P86	PLB_SrdComp	Simulation	O	0	Output of slave Sl_rdComp OR gate	
	P87	PLB_SrdBTerm	Simulation	O	0	Output of slave Sl_rdBTerm OR gate	
	P88	PLB_SMBusy[0:C_NUM_MASTERS-1]	Simulation	O	0	Output of slave Sl_MBusy OR gate	
	P89	PLB_SMErr[0:C_NUM_MASTERS-1]	Simulation	O	0	Output of slave Sl_MErr OR gate	
	P90	PLB_Sssize[0:1]	Simulation	O	0	Output of slave Sl_SSize OR gate	

Notes:
1. The outputs in this section are required to connect with the PLB Monitor Bus Functional Model (BFM) supplied with the IBM PLB Toolkit. These outputs are not needed otherwise, but can be used as debug signals if desired.

Parameter/Port Dependencies

The width of many of the PLB Arbiter signals depends on the number of PLB masters and number of PLB slaves in the design. In addition, when certain features are parameterized away, the related input signals are unconnected and the related output signals are set to a constant values. The dependencies between the PLB Arbiter design parameters and I/O signals are shown in Table 4.

Table 4: **Parameter-Port Dependencies**

		Name	Affects	Depends	Relationship Description
Design Parameters	G1	C_NUM_MASTERS	P10-P37, P45, P59, P60		The width of many buses is set by the number of PLB masters in the design.
	G2	C_NUM_SLAVES	P58 - P71		The width of many buses is set by the number of PLB slaves in the design.
	G3	C_PLB_AWIDTH	P11, P39		
	G4	C_PLB_DWIDTH	P12, P26, P32, P40, P56, P64		
	G5	C_DCR_INTFCE	G6 - G9, P1- P6		
	G6	C_BASEADDR		G5	Unconnected if C_DCR_INTFCE=0.
	G7	C_HIGHADDR		G5	Unconnected if C_DCR_INTFCE=0.
	G8	C_DCR_AWIDTH	P1	G5	Unconnected if C_DCR_INTFCE=0.
	G9	C_DCR_DWIDTH	P4, P6	G5	Unconnected if C_DCR_INTFCE=0.
	G10	C_IRQ_ACTIVE	P72		

Table 4: **Parameter-Port Dependencies *(Continued)***

		Name	Affects	Depends	Relationship Description
I/O Signals	P1	DCR_ABus[0:C_DCR_AWIDTH-1]		G5, G8	Width varies with the size of the DCR address bus. This input is unconnected if C_DCR_INTFCE=0.
	P2	DCR_Read		G5	This input is unconnected if C_DCR_INTFCE=0.
	P3	DCR_Write		G5	This input is unconnected if C_DCR_INTFCE=0.
	P4	DCR_DBus[0:C_DCR_DWIDTH-1]		G5, G9	Width varies with the size of the DCR data bus. This input is unconnected if C_DCR_INTFCE=0.
	P5	PLB_dcrAck		G5	This output is grounded if C_DCR_INTFCE=0.
	P6	PLB_dcrDBus[0:C_DCR_DWIDTH-1]		G5, G9	Width varies with the size of the DCR data bus. This output is grounded if C_DCR_INTFCE=0.
	P7	PLB_pendPri[0:1]			
	P8	PLB_pendReq			
	P9	PLB_reqPri[0:1]			
	P10	M_abort[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P11	M_ABus[0:C_NUM_MASTERS*C_PLB_AWIDTH-1]		G1, G3	Width varies with the size of the PLB address bus and the number of PLB masters.
	P12	M_BE[0:C_NUM_MASTERS*C_PLB_DWIDTH/8-1]		G1,G4	Width varies with the size of the PLB data bus and the number of PLB masters.
	P13	M_busLock[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P14	M_compress[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P15	M_guarded[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P16	M_lockErr[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P17	M_mSize[0:C_NUM_MASTERS*2 -1]		G1	Width varies with the number of PLB masters.
	P18	M_ordered[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P19	M_priority[0:C_NUM_MASTERS*2 -1]		G1	Width varies with the number of PLB masters.
	P20	M_rdBurst[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P21	M_request[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P22	M_RNW[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P23	M_size[0:C_NUM_MASTERS*4 -1]		G1	Width varies with the number of PLB masters.
	P24	M_type[0:C_NUM_MASTERS*3-1]		G1	Width varies with the number of PLB masters.
	P25	M_wrBurst[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P26	M_wrDBus[0:C_NUM_MASTERS*C_PLB_DWIDTH -1]		G1, G4	Width varies with the size of the PLB data bus and the number of PLB masters.
	P27	PLB_MAddrAck[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P28	PLB_MBusy[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P29	PLB_MErr[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P30	PLB_MRdBTerm[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P31	PLB_MRdDAck[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P32	PLB_MRdDBus[0:C_NUM_MASTERS*C_PLB_DWIDTH -1]]		G1, G4	Width varies with the size of the PLB data bus and the number of PLB masters.
	P33	PLB_MRdWdAddr[0:C_NUM_MASTERS*4 -1]		G1	Width varies with the number of PLB masters.
	P34	PLB_MRearbitrate[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
	P35	PLB_MSSize[0:C_NUM_MASTERS*2 -1]		G1	Width varies with the number of PLB masters.

Table 4: **Parameter-Port Dependencies *(Continued)***

	Name	Affects	Depends	Relationship Description
P36	PLB_MWrBTerm[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
P37	PLB_MWrDAck[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
P38	PLB_abort			
P39	PLB_ABus[0:C_PLB_AWIDTH-1]		G3	Width varies with the size of the PLB address bus.
P40	PLB_BE[0:C_PLB_DWIDTH/8-1]		G4	Width varies with the size of the PLB data bus.
P41	PLB_busLock			
P42	PLB_compress			
P43	PLB_guarded			
P44	PLB_lockErr			
P45	PLB_masterID[0:log2(C_NUM_MASTERS)-1]		G1	Width varies with the number of PLB masters.
P46	PLB_MSize[0:1]			
P47	PLB_ordered			
P48	PLB_PAValid			
P49	PLB_rdBurst			
P50	PLB_rdPrim			
P51	PLB_RNW			
P52	PLB_SAValid			
P53	PLB_size[0:3]			
P54	PLB_type[0:2]			
P55	PLB_wrBurst			
P56	PLB_wrDBus[0:C_PLB_DWIDTH-1]		G4	Width varies with the size of the PLB data bus.
P57	PLB_wrPrim			
P58	Sl_addrAck[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.
P59	Sl_MErr[0:C_NUM_SLAVES*C_NUM_MASTERS-1]		G1	Width varies with the number of PLB slaves and the number of PLB masters.
P60	Sl_MBusy[0:C_NUM_SLAVES*C_NUM_MASTERS-1]		G1	Width varies with the number of PLB slaves and the number of PLB masters.
P61	Sl_rdBTerm[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.
P62	Sl_rdComp[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.
P63	Sl_rdDAck[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.
P64	Sl_rdDBus[0:C_NUM_SLAVES*C_PLB_DWIDTH-1]		G2,G3	Width varies with the number of PLB slaves and the size of the PLB data bus.
P65	Sl_rdWdAddr[0:C_NUM_SLAVES*4-1]		G2	Width varies with the number of PLB slaves.
P66	Sl_rearbitrate[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.
P67	Sl_SSize[0:C_NUM_SLAVES*2-1]		G2	Width varies with the number of PLB slaves.
P68	Sl_wait[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.
P69	Sl_wrBTerm[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.
P70	Sl_wrComp[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.
P71	Sl_wrDAck[0:C_NUM_SLAVES-1]		G2	Width varies with the number of PLB slaves.

Table 4: **Parameter-Port Dependencies *(Continued)***

	Name	Affects	Depends	Relationship Description
P72	Bus_Error_Det		G5,G10	C_IRQ_ACTIVE determines the active state of the interrupt. If C_DCR_INTFCE=0, then interrupts are always enabled, otherwise, interrupts are enabled by writing to the PLB Arbiter Control Register.
P73	Clk			
P74	Rst			
P75	ArbReset			
P76	ArbAddrVldReg			
P77	PLB_SaddrAck			
P78	PLB_Swait			
P79	PLB_Srearbitrate			
P80	PLB_SwrDAck			
P81	PLB_SwrComp			
P82	PLB_SwrBTerm			
P83	PLB_SrdDBus[0:C_PLB_DWIDTH-1]		G4	Width varies with the size of the PLB data bus.
P84	PLB_SrdWdAddr[0:3]			
P85	PLB_SrdDAck			
P86	PLB_SrdComp			
P87	PLB_SrdBTerm			
P88	PLB_SMBusy[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
P89	PLB_SMErr[0:C_NUM_MASTERS-1]		G1	Width varies with the number of PLB masters.
P90	PLB_Sssize[0:1]		G1	Width varies with the number of PLB masters.

PLB Arbiter Registers

The PLB Arbiter contains seven DCR-accessible registers to provide error address and status information if the design has been parameterized to contain a DCR interface (C_DCR_INTFCE = 1) as shown in Table 5. The base address for these registers is set in the parameter C_BASEADDR.

Table 5: **PLB Arbiter DCR Registers**

Register Name	Description	DCR Address	Access
PESR_MERR_DETECT	Master Error Detect Bits	C_BASEADDR + 0x00	Read/Write
PESR_MDRIVE_PEAR	Master Driving PEAR	C_BASEADDR + 0x01	Read
PESR_RNW_ERR	Read/Write Error	C_BASEADDR + 0x02	Read
PESR_LCK_ERR	Lock Error Bit	C_BASEADDR + 0x03	Read
PEAR_ADDR	PLB Error Address	C_BASEADDR + 0x04	Read
PEAR_BYTE_EN	PLB Error Byte Enables	C_BASEADDR + 0x05	Read
PACR	PLB Arbiter Control Register	C_BASEADDR + 0x06	Read/Write

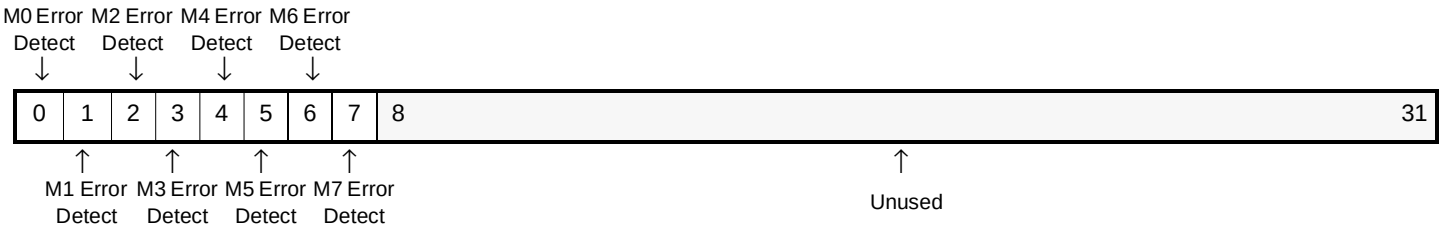
PLB Error Status Registers

There are four PESR registers that provide error information - was an error detected, which Master's address and byte enables are in the PEARs, was the error due to a read or write transaction, and did the master lock the error condition. If a read of the PESR_MERR_DETECT register returns all zeros, then no masters detected any errors and no further reads are necessary.

PESR_MERR_DETECT: Master Error Detect Bits

This register contains the error detect bit for each master. The bit location corresponds to the PLB Master. For example, if PLB Master 0 has detected an error, then bit 0 will be set. Writing a '1' to a bit in this register clears this bit and the corresponding bit in the other PESRs (PESR_MDRIVE_PEAR, PESR_RNW_ERR, and PESR_LCK_ERR). If a particular master detected an error and had locked the PEARs, writing a '1' to the corresponding bit in this register would clear and unlock the master's error fields and unlock the PEARs. The bits in this register are reset when a '1' has been written to the register, Rst has been asserted, or a '1' has been written to the Software Reset bit in the PACR. Table 6 shows the bit definitions of this register when the number of PLB masters is 8 and the width of the DCR data bus is 32.

Table 6: **PESR_MERR_DETECT (C_NUM_MASTERS=8, C_DCR_DWIDTH=32)**



The bit definitions for PESR_MERR_DETECT are shown in Table 7. The bits is this register are reset by writing a '1' to the bit.

Table 7: **PLB Arbiter PESR_MERR_DETECT Bit Definitions**

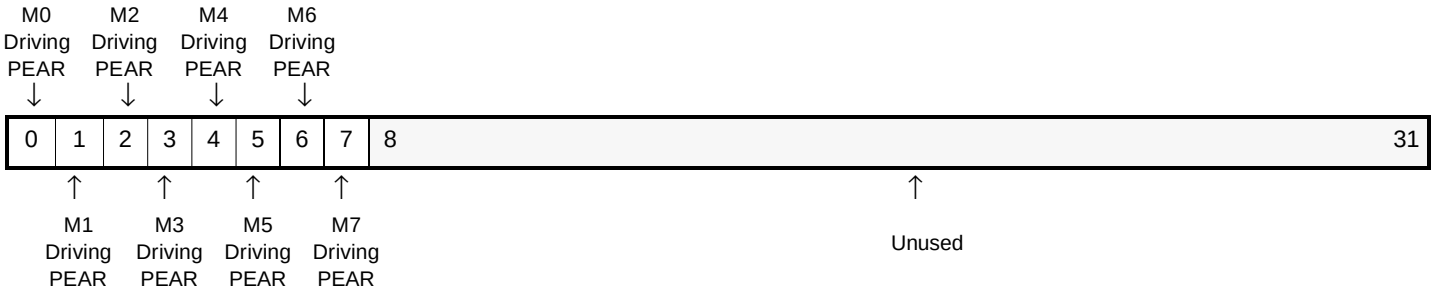
Bit(s)	Name	Core Access	Reset Value	Description
0 to C_NUM_MASTERS-1	Master Error Detect	Read/Write	'0'	**Master Error Detect.** Read: Error detect bit for PLB Masters 0 to C_NUM_MASTERS-1 respectively. • '1' - error detected • '0' - no error detected Write: Clear error bit for PLB Masters 0 to C_NUM_MASTERS -1 respectively. • '1' - clear and unlock corresponding master's error fields and PEARs • '0' - do not clear error
C_NUM_MASTERS to C_DCR_DWIDTH-1	Unused			

PESR_MDRIVE_PEAR: Master Driving PEAR

This register indicates which PLB Master is driving the PEARs. Each bit location in this register corresponds to a PLB Master. For example, if PLB Master 0 is driving the PEARs, then bit 0 will be set. Only one master can drive the PEARs, therefore, only one bit will be set in this register. Writing to this register has no effect. The bits in this register are reset when a '1' is written to the corresponding bits in PESR_MERR_DETECT, Rst has been asserted, or a '1' has been written

to the Software Reset bit in the PACR. Table 8 shows the bit definitions of this register when the number of PLB masters is 8 and the width of the DCR data bus is 32.

Table 8: **PESR_MDRIVE_PEAR (C_NUM_MASTERS=8, C_DCR_DWIDTH=32)**



The bit definitions for PESR_MDRIVE_PEAR are shown in Table 9.

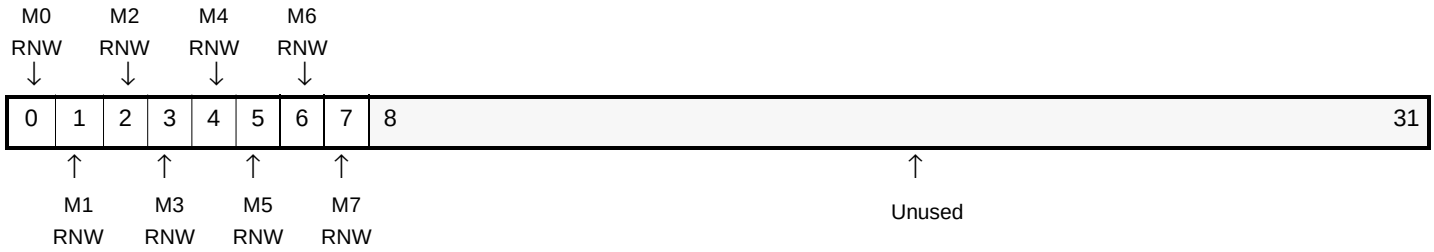
Table 9: **PLB Arbiter PESR_MDRIVE_PEAR Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0 to C_NUM_MASTERS-1	Master Driving PEAR	Read	'0'	**Master Driving PEAR.** Read: PEAR bit for PLB Masters 0 to C_NUM_MASTERS-1 respectively. • '1' - master is driving PEAR • '0' - master is not driving PEAR Write: No effect.
C_NUM_MASTERS to C_DCR_DWIDTH-1	Unused			

PESR_RNW_ERR: Master Read/Write Bits

This register indicates the read/write condition that caused the error for each PLB Master. Each bit location in this register corresponds to a PLB Master. For example, if PLB Master 0 detected an error during a read operation, bit 0 would be set. If PLB Master 1 detected an error during a write operation, bit 1 would be reset. Writing to this register has no effect. The bits in this register are reset when a '1' is written to the corresponding bits in PESR_MERR_DETECT, Rst has been asserted, or a '1' has been written to the Software Reset bit in the PACR. Table 10 shows the bit definitions of this register when the number of PLB masters is 8 and the width of the DCR data bus is 32.

Table 10: **PESR_RNW_ERR (C_NUM_MASTERS=8, C_DCR_DWIDTH=32)**



The bit definitions for PESR_RNW_ERR are shown in Table 11.

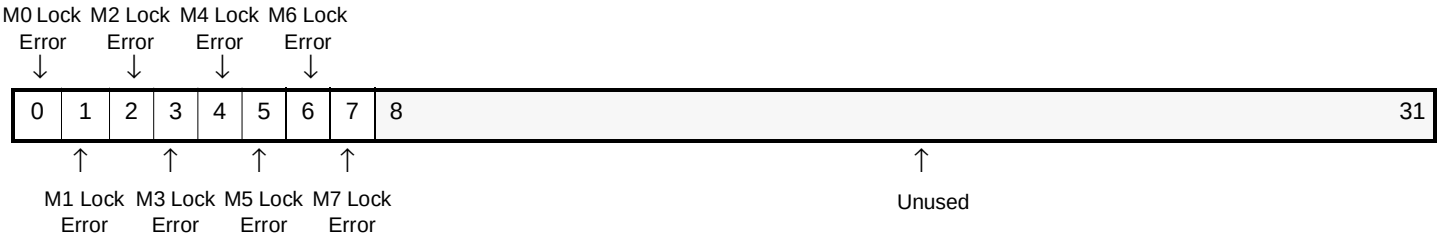
Table 11: **PLB Arbiter PESR_RNW_ERR Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0 to C_NUM_MASTERS-1	Master Read Not Write	Read	'0'	**Master Read Not Write.** Read: RNW status for each master • '1' - error was in response to a read • '0' - error was in response to a write Write: No effect.
C_NUM_MASTERS to C_DCR_DWIDTH-1	Unused			

PESR_LCK_ERR: Master Lock Error Bits

This register indicates whether each PLB Master has locked their error bits. Each bit location in this register corresponds to a PLB Master. Setting the Master's lock error bit means that the master's error fields are locked, i.e., subsequent errors cannot overwrite master's error fields until error is cleared. If the Master's lock error bit is reset, the master's error fields are not locked and subsequent errors will overwrite the master's error fields. Writing to this register has no effect. The bits in this register are reset when a '1' is written to the corresponding bits in PESR_MERR_DETECT, Rst has been asserted, or a '1' has been written to the Software Reset bit in the PACR. Table 12 shows the bit definitions of this register when the number of PLB masters is 8 and the width of the DCR data bus is 32.

Table 12: **PESR_LCK_ERR (C_NUM_MASTERS=8, C_DCR_DWIDTH=32)**



The bit definitions for PESR_LCK_ERR are shown in Table 13.

Table 13: **PLB Arbiter PESR_LCK_ERR Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0 to C_NUM_MASTERS-1	Master Lock Error	Read	'0'	**Master Lock Error.** Read: Lock error bit for each master • '1' -error fields are locked (subsequent errors cannot overwrite master's error fields until error is cleared) • '0' - error fields are not locked (subsequent errors can overwrite master's error fields) Write: No effect.
C_NUM_MASTERS to C_DCR_DWIDTH-1	Unused			

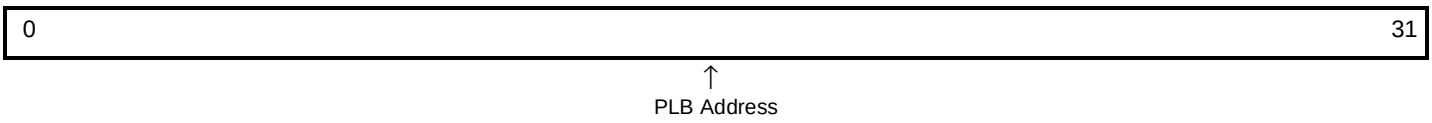
Bus Error Address Registers

There are two PEAR registers - one contains the PLB address of the transaction that caused the error and the other contains the value of the PLB byte enables during the transaction that caused the error.

PEAR_ADDR: PLB Error Address

This register contains the PLB address of the transaction that caused the error as shown in Table 14. Note that the width of the PLB address bus must be <= the width of the DCR data bus for the entire PLB address to be stored. Otherwise, only the most significant bits of the PLB address are stored. This register is cleared when Rst is asserted.

Table 14: **PEAR_ADDR (C_PLB_AWIDTH=32, C_DCR_DWIDTH=32)**



The bit definitions for PEAR_ADDR are shown in Table 13.

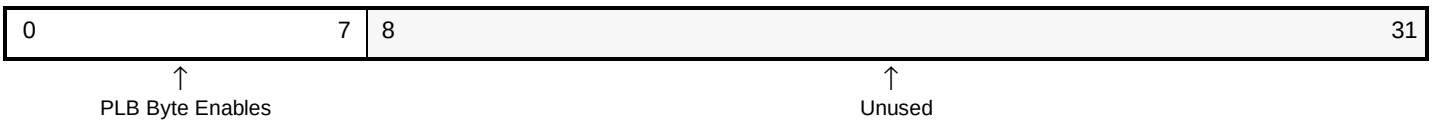
Table 15: **PLB Arbiter PEAR_ADDR Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0 to C_PLB_AWIDTH-1	Bus Error Address	Read	'0'	**Bus Error Address.** Read: PLB address where error occurred Write: No effect.
C_PLB_AWIDTH to C_DCR_DWIDTH-1	Unused			

PEAR_BYTE_EN: PLB Error Byte Enables

This register contains the values of the PLB byte enables during the transaction that caused the error as shown in Table 16. The width of the PLB byte enable bus is the width of the PLB data bus divided by 8. Therefore, if the PLB data bus is 64 bits wide, there are 8 byte enables. This register is cleared when Rst is asserted.

Table 16: **PEAR_BYTE_EN (C_PLB_DWIDTH=64, C_DCR_DWIDTH=32)**



The bit definitions for PEAR_BYTE_EN are shown in Table 17.

Table 17: **PLB Arbiter PEAR_BYTE_EN Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0 to C_PLB_DWIDTH/8-1	Bus Error Address	Read	'0'	**Bus Error Address.** Read: PLB byte enable value when error occurred Write: No effect.
C_PLB_DWIDTH/8 to C_DCR_DWIDTH-1	Unused			

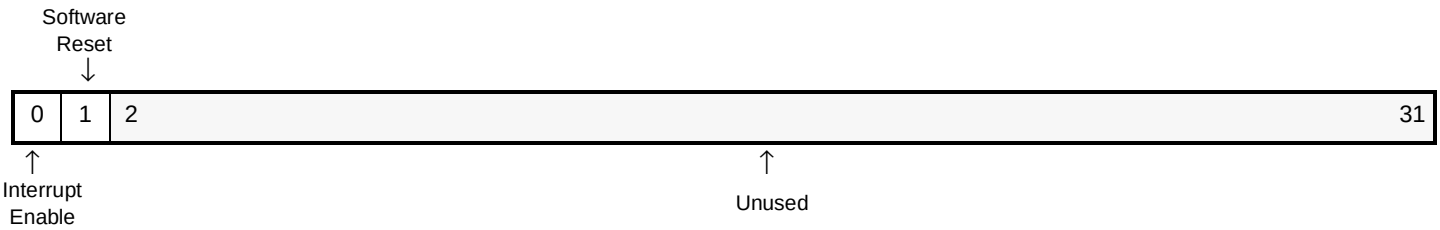
PLB Arbiter Control Register

There is one PLB Arbiter Control register that enables or disables the interrupt request output from the PLB Arbiter and provides a software reset.

PACR: PLB Arbiter Control Register

This register contains one bit that enables or disables the interrupt request and another bit used to reset the PLB Arbiter as shown in Table 18. Note that the default state of the control register is to have interrupts enabled, therefore if the PLB Arbiter is parameterized to not have a DCR interface (C_DCR_INTFCE = 0) then interrupts are still enabled. Also note that when the Software reset bit is asserted, ALL registers and flip-flops within the PLB Arbiter including all DCR registers are reset. This register is reset to the default state whenever Rst is asserted or a '1' is written to the Software Reset bit.

Table 18: **PACR (C_DCR_DWIDTH=32)**



The bit definitions for PACR are shown in Table 19.

Table 19: **PLB Arbiter PACR Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0	Interrupt Enable	Read/Write	'1'	**Interrupt Enable.** Read: PLB Arbiter Interrupt Enable Write: • '1' - enable interrupts • '0' - disable interrupts
1	Software Reset	Read/Write	'0'	**Software Reset.** Read: This bit will always read '0' since it is reset whenever a '1' is written to it. Write: • '1' - reset the PLB • '0' - resume normal PLB operation
2 to C_DCR_DWIDTH-1	Unused			

PLB Arbiter Interrupt Description

The PLB Arbiter has one interrupt request output called Bus_Error_Det. This interrupt is an edge type interrupt and is automatically reset to the inactive state on the next clock cycle, therefore an explicit interrupt acknowledge is not required. The active level of the Bus_Error_Det interrupt is determined by the design parameter, C_IRQ_ACTIVE.

Note that if interrupts are enabled, then an interrupt request from the PLB Arbiter will be generated whenever any bus error is detected regardless of whether masters have locked their error fields or not. Also note that if the parameter C_DCR_INTFCE is 0 indicating that there is no DCR interface, then interrupts will be still be enabled as the default state of the Interrupt Enable bit in the PACR is asserted.

PLB Arbiter Block Diagram

Figure 3 provides a comprehensive block diagram of the PLB arbiter.

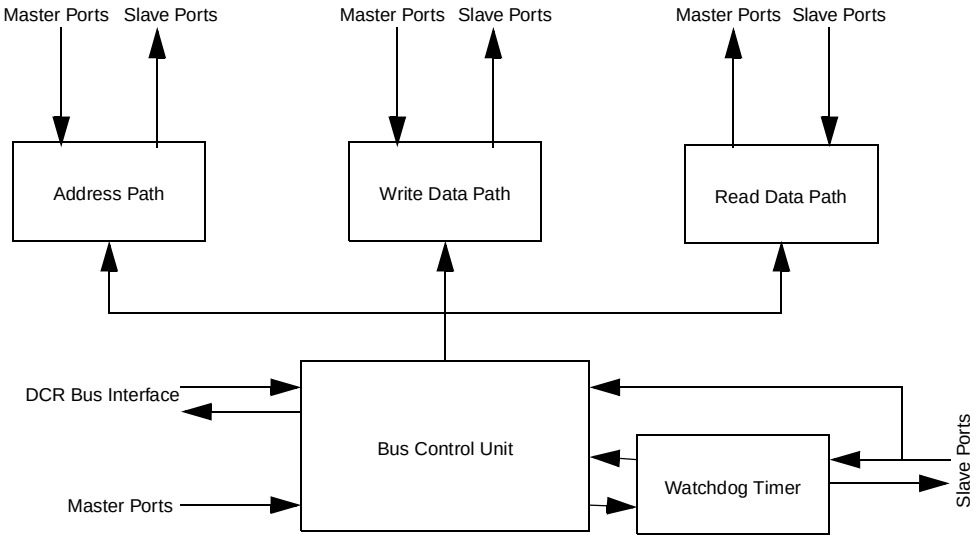


Figure 3: **PLB Arbiter Block Diagram**

Bus Control Unit

The PLB arbiter bus control unit consists of a bus arbitration control unit that manages the address and data flow through the PLB and DCRs. The bus arbitration control unit supports arbitration for 16 PLB masters. The address and data flow control logic provides address pipelining and address and data steering support for 16 PLB masters and 16 PLB slaves.

The PLB arbiter device control registers are used to control and report status from the bus arbitration control unit and address flow control logic. The registers are accessed by using the *move from device* control register (*mfdcr*) and *move to device* control register (*mtdcr*) instructions, which move data between the device control registers and the processor's general purpose registers.

Refer to **PLB Arbiter Registers**, page 45 for additional information.

Address Path Unit

The PLB arbiter address path unit contains the necessary muxing to select the master address which will be driven to the slave devices on the PLB address output.

Read Data Path Unit

The PLB arbiter read data path unit contains the necessary steering logic for the master and slave read data buses.

Write Data Path Unit

The PLB arbiter write data path unit contains the necessary steering logic for the master and slave write data buses.

Watchdog Timer

The PLB arbiter watchdog timer provides the necessary handshake to complete the transfer in the event that a master's request times out on the PLB. When handshaking the master, the **M_type[n*3:n*3+2]** signals are ignored and all transfers are terminated as if they were normal memory transfers (**M_type[n*3:n*3+2]** = 0b000). During burst transfers, the **PLB_MRdBTerm[n]** and **PLB_MWrBTerm[n]** signals are driven, when appropriate, in order to minimize the impact on bus latency associated with this type of transfer.

Refer to **PLB Arbiter Operations**, page 55 to see how the PLB arbiter handles the timed out PLB transfers.

Master[n] Interface

Figure 4 shows all master[n] interface I/O signals (where *n* is the number of a master 0 to C_NUM_MASTERS-1). Refer to the *IBM 64-bit Processor Local Bus Architecture Specifications* for detailed functional descriptions of these signals. Note that C_PLB_DWIDTH =64 and C_PLB_AWIDTH=32 in this diagram.

Slave Interface[m]

Figure 5 demonstrates all slave[m] interface I/O signals (where m = 0 to C_NUM_SLAVES-1). Refer to the *IBM 64-Bit Processor Local Bus Architecture Specifications* for detailed functional descriptions of these signals. Note that C_NUM_MASTERS=8, C_PLB_DWIDTH=64, and C_PLB_AWIDTH=32 in this diagram.

DCR Interface

The device control register (DCR) interface allows the CPU core in the system to read and write the DCRs in the PLB arbiter. For additional information on the DCR bus, see the *IBM 32-Bit Device Control Register Bus Architecture Specifications*.

Figure 6 demonstrates all DCR interface input/output signals when C_DCR_DWIDTH = 32 and C_DCR_AWIDTH=10.

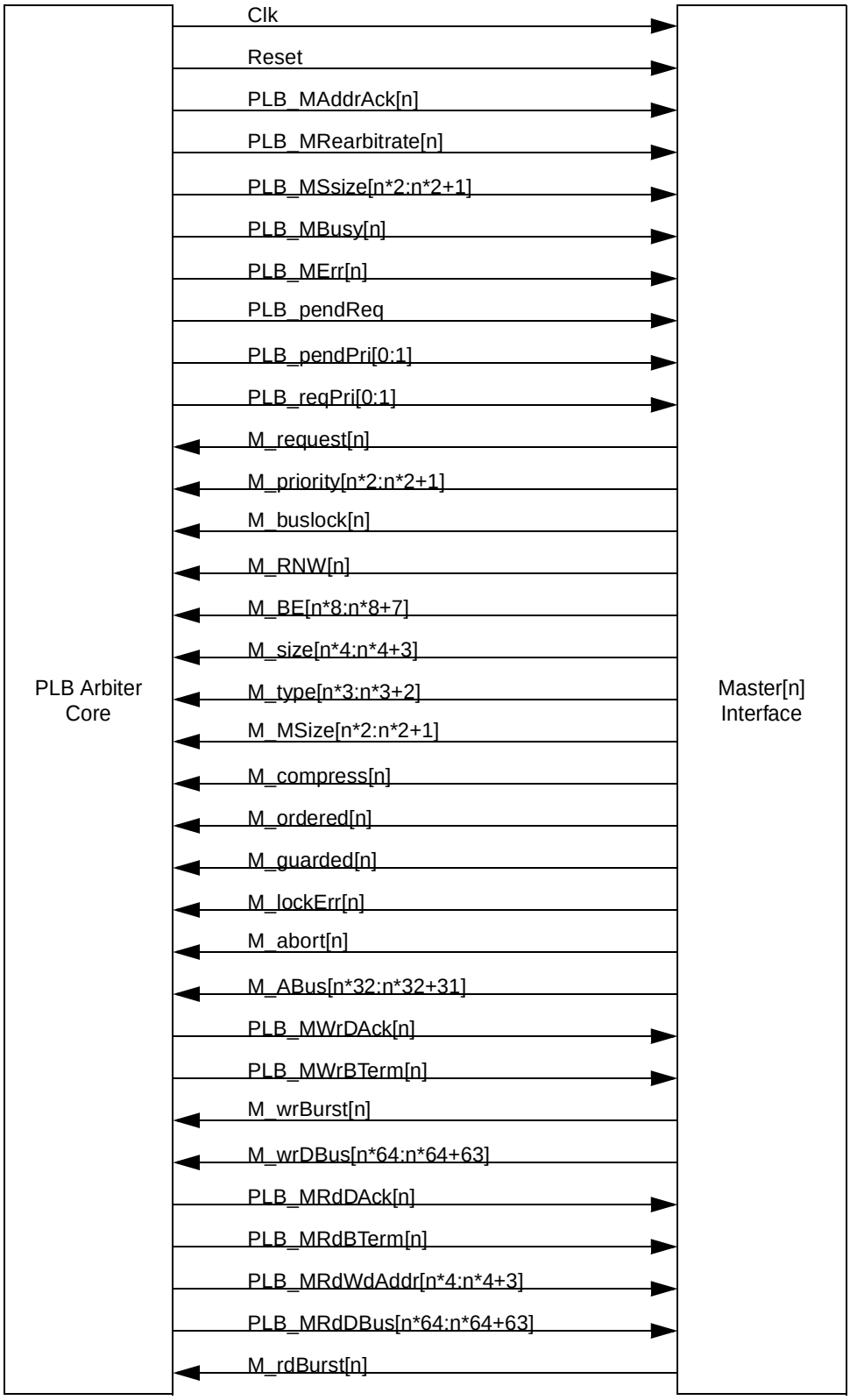


Figure 4: **Master[n] Interface**

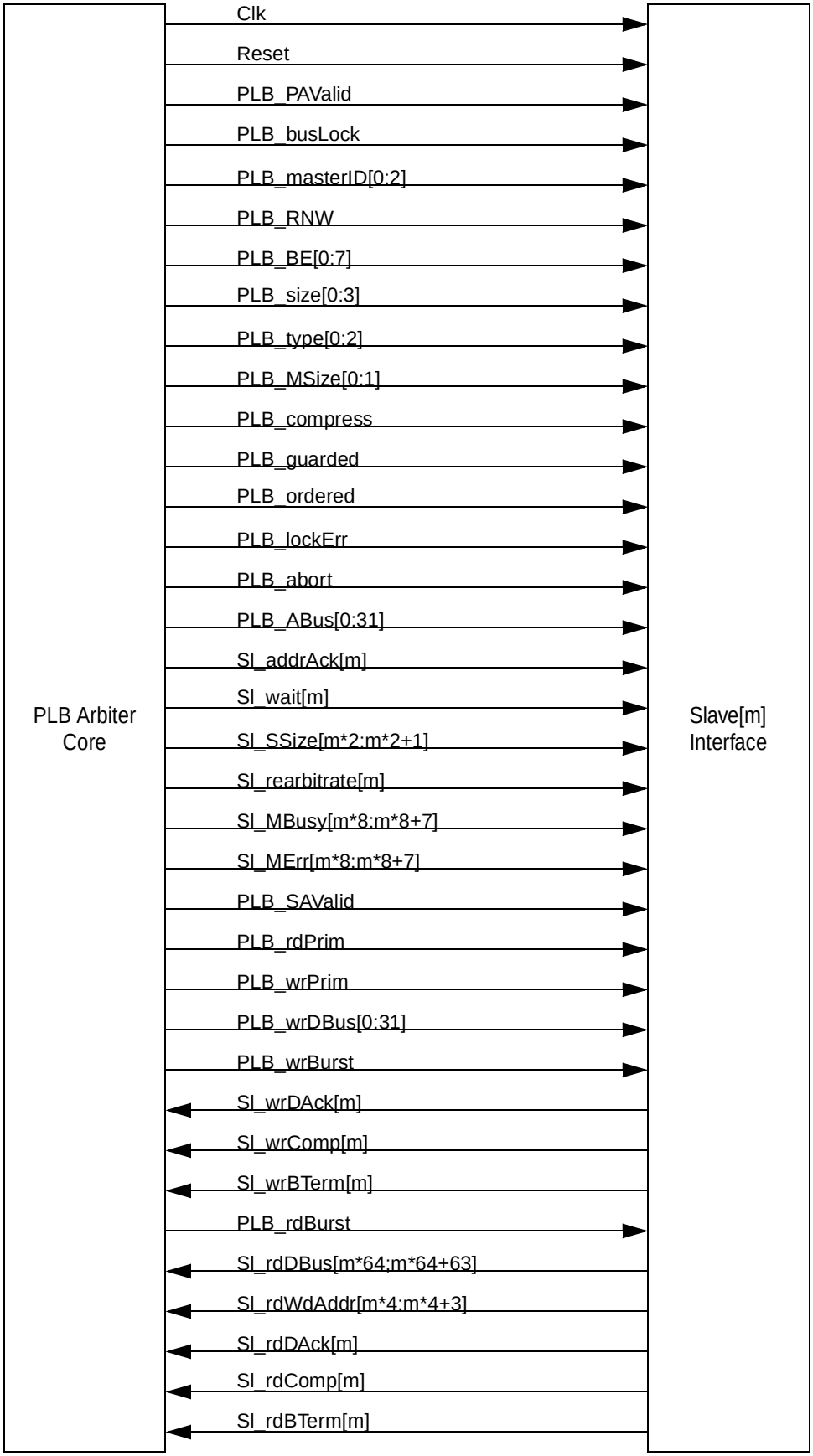


Figure 5: **Slave[m] Interface**

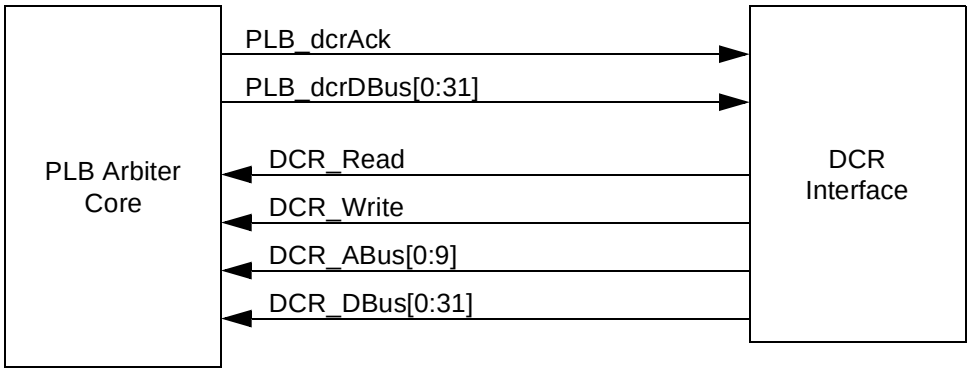


Figure 6: **DCR Interface**

PLB Arbiter Operations

The *IBM 64-bit Processor Local Bus Architecture Specifications* document provides a comprehensive discussion on the various PLB arbiter operations and transfers that are similar to the PLB. This section on PLB arbiter operations provides detailed information on operations that are specific to the PLB arbiter, namely:

- Single Read Transfer Bus Time-Out
- Single Write Transfer Bus Time-Out
- Line Read Transfer Bus Time-Out
- Line Write Transfer Bus Time-Out
- Burst Read Transfer Bus Time-Out
- Burst Write Transfer Bus Time-Out
- Read Transfer
- Pipelined Read Transfer
- Pipelined Write Transfer

Note that in all of the following timing diagrams, n = 0 to C_NUM_MASTERS-1 and m = 0 to C_NUM_SLAVES-1.

Single Read Transfer Bus Time-Out

Figure 7 shows a bus time-out for a single read transfer on the PLB. The PLB arbiter samples the **SI_wait[m]** and **SI_rearbitrate[m]** signals 15 cycles after the initial assertion of the **PLB_PAValid** signal, and if both are negated, it asserts the **PLB_MAddrAck[n]** signal. During the cycle in which **PLB_MAddrAck[n]** is asserted, the PLB watchdog timer samples the **SI_addrAck[m]** signal, and if negated, it completes the handshaking to the master by asserting the **PLB_MRdDAck[n]** and **PLB_MErr[n]** signals two cycles later.

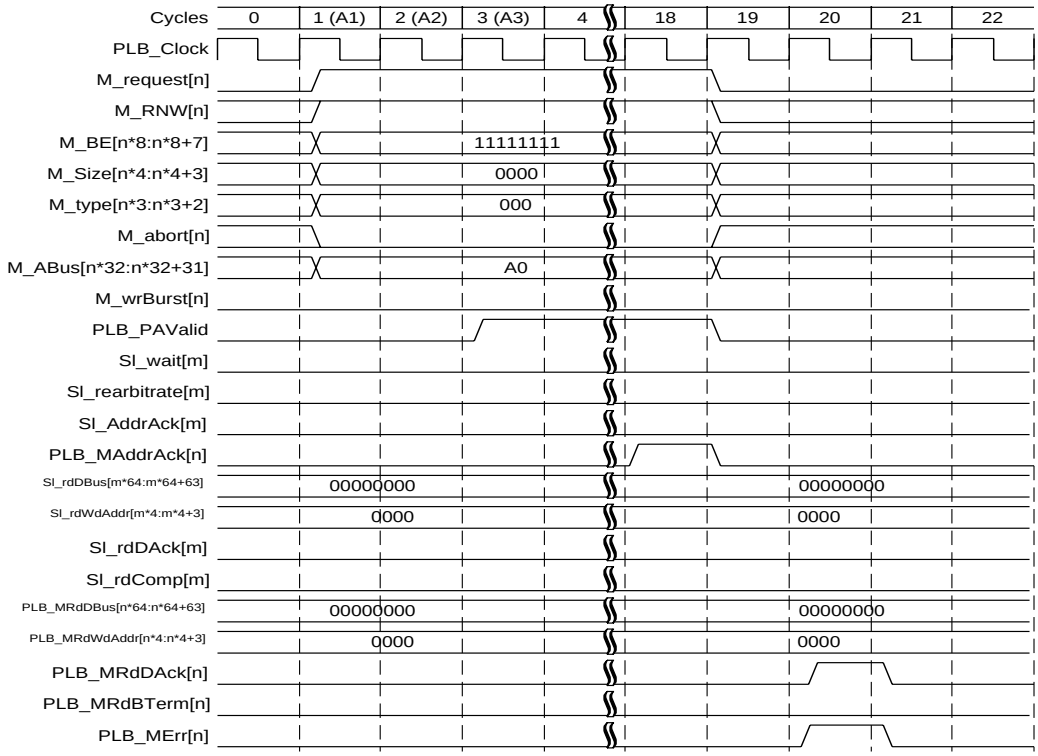


Figure 7: **Single Read Transfer Bus Time-Out**

Single Write Transfer Bus Time-Out

Figure 8 shows a bus time-out for a single write transfer on the PLB. The PLB arbiter samples the **SI_wait[m]** and **SI_rearbitrate[m]** signals 15 cycles after the initial assertion of the **PLB_PAValid** signal, and if both are negated, it asserts the **PLB_MAddrAck[n]** signal. During the cycle in which **PLB_MAddrAck[n]** is asserted, the PLB watchdog timer samples the **SI_addrAck[m]** signal, and if negated, it completes the handshaking to the master by asserting the **PLB_MWrDAck[n]** and **PLB_MErr[n]** signals two cycles later.

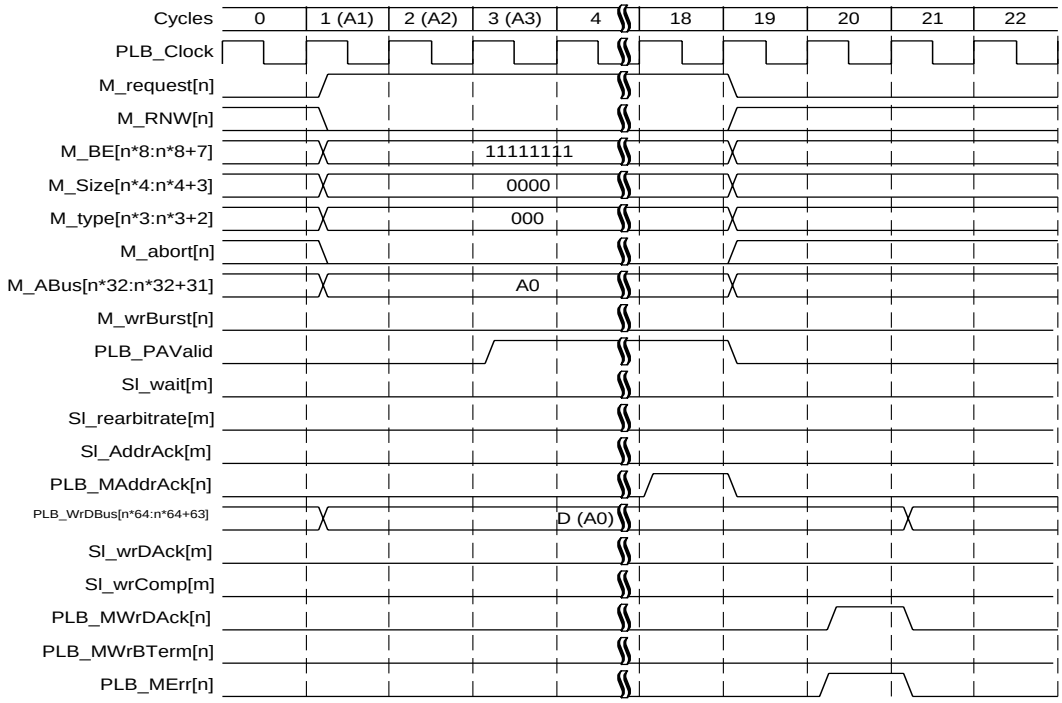


Figure 8: **Single Write Transfer Bus Time-Out**

Line Read Transfer Bus Time-Out

Figure 9 shows a bus time-out for a 4-word line read transfer on the PLB. The PLB arbiter samples the **SI_wait[m]** and **SI_rearbitrate[m]** signals 15 cycles after the initial assertion of the **PLB_PAValid** signal, and if both are negated, it asserts the **PLB_MAddrAck[n]** signal. During the cycle in which **PLB_MAddrAck[n]** is asserted, the PLB watchdog timer samples the **SI_addrAck[m]** signal, and if negated, it completes the handshaking to the master by asserting the **PLB_MRdDAck[n]** and **PLB_MErr[n]** signals two cycles later.

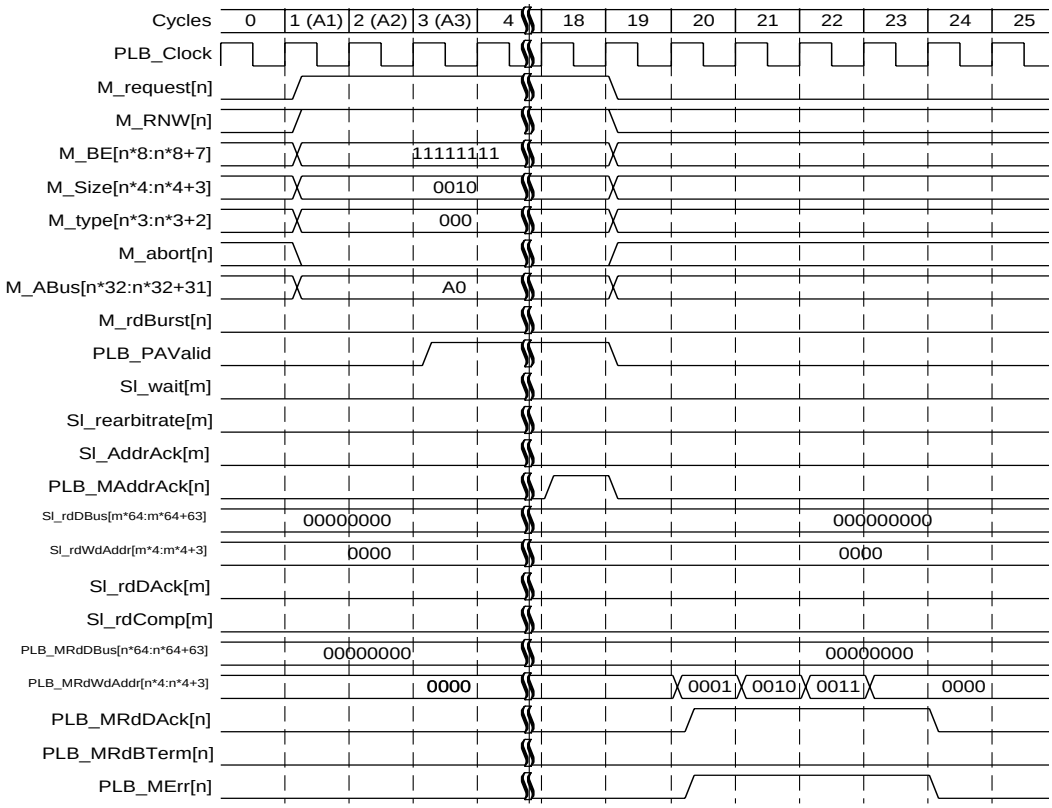


Figure 9: **Line Read Transfer Bus Time-Out**

Line Write Transfer Bus Time-Out

Figure 10 shows a bus time-out for a line write transfer on the PLB. The PLB arbiter samples the **SI_wait[m]** and **SI_rearbitrate[m]** signals 15 cycles after the initial assertion of the **PLB_PAValid** signal, and if both are negated, it asserts the **PLB_MAddrAck[n]** signal. During the cycle in which **PLB_MAddrAck[n]** is asserted, the PLB watchdog timer samples the **SI_addrAck[m]** signal, and if negated, it completes the handshaking to the master by asserting the **PLB_MWrDAck[n]** and **PLB_MErr[n]** signals two cycles later.

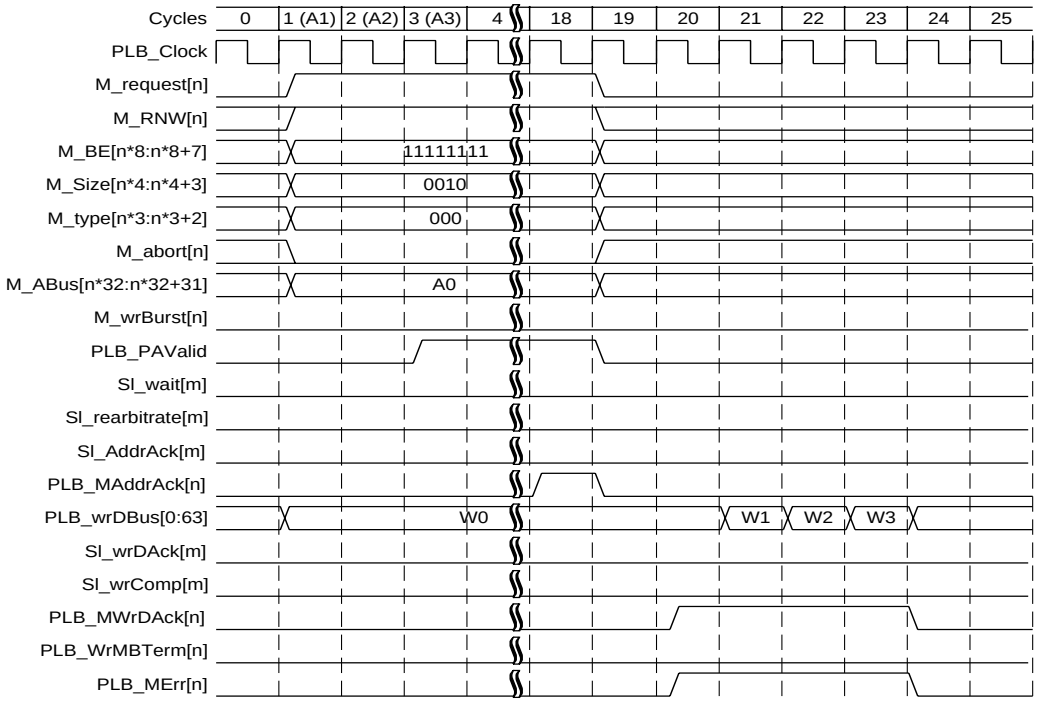


Figure 10: **Line Write Transfer Bus Time-Out**

Burst Read Transfer Bus Time-Out

Figure 11 shows a bus time-out for a burst read transfer on the PLB. The PLB arbiter samples the **Sl_wait[m]** and **Sl_rearbitrate[m]** signals 15 cycles after the initial assertion of the **PLB_PAValid** signal, and if both are negated, it asserts the **PLB_MAddrAck[n]** signal. During the cycle in which **PLB_MAddrAck[n]** is asserted, the PLB watchdog timer samples the **Sl_addrAck[m]** signal, and if negated, it completes the handshaking to the master by asserting the **PLB_MRdDAck[n]** and **PLB_MErr[n]** signals two cycles later. Also, in the cycle following the assertion of **PLB_MAddrAck[n]**, the PLB watchdog timer samples the **PLB_rdBurst** signal, and if asserted, it asserts the **PLB_MRdBTerm[n]** signal in the same cycle.

Note: The master is expected to negate its **M_rdBurst[n]** signal in response to the assertion of **PLB_MRdBTerm[n]**.

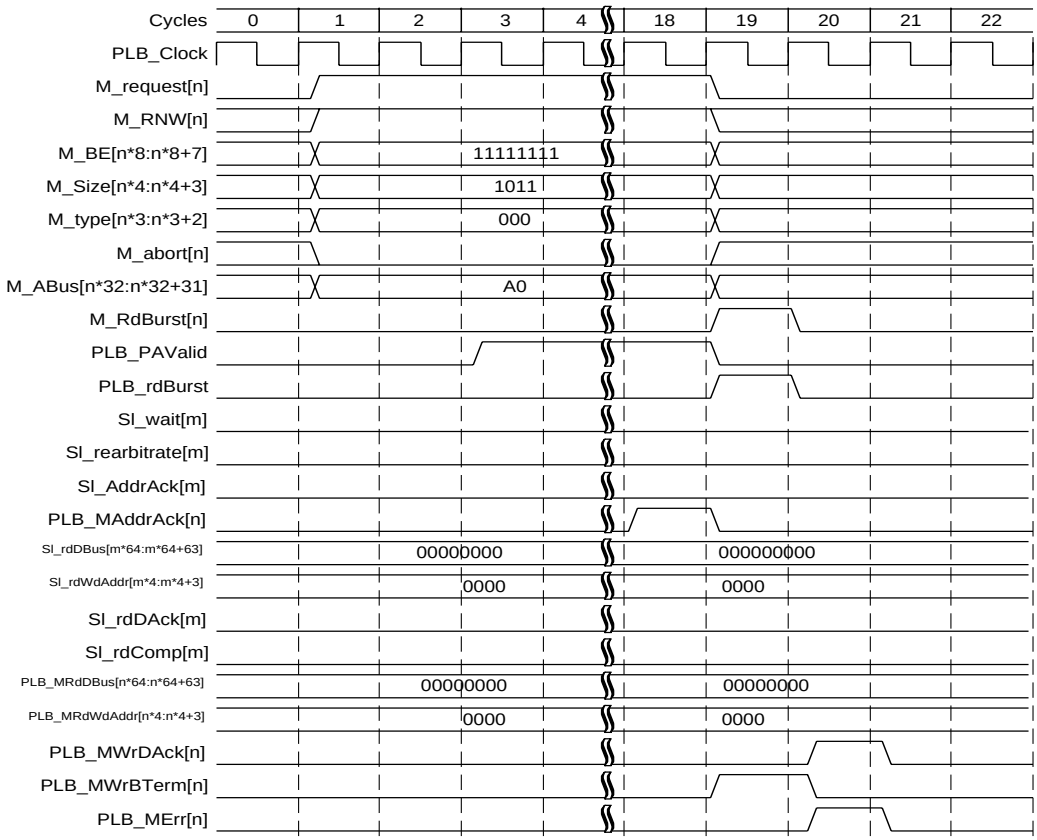


Figure 11: **Burst Read Transfer Bus Time-Out**

Burst Write Transfer Bus Time-Out

Figure 12 shows a bus time-out for a burst write transfer on the PLB. The PLB arbiter samples the **SI_wait[m]** and **SI_rearbitrate[m]** signals 15 cycles after the initial assertion of the **PLB_PAValid** signal, and if both are negated, it asserts the **PLB_MAddrAck[n]** signal. During the cycle in which **PLB_MAddrAck[n]** is asserted, the PLB watchdog timer samples the **SI_addrAck[m]** signal, and if negated, it completes the handshaking to the master by asserting the **PLB_MWrDAck[n]** and **PLB_MErr[n]** signals two cycles later. Also, in the cycle following the assertion of **PLB_MAddrAck[n]**, the PLB watchdog timer samples the **PLB_wrBurst** signal, and if asserted, it asserts the **PLB_MWrBTerm[n]** signal in the same cycle.

Note: The master is expected to negate its **M_wrBurst[n]** signal in response to the assertion of **PLB_MWrBTerm[n]**.

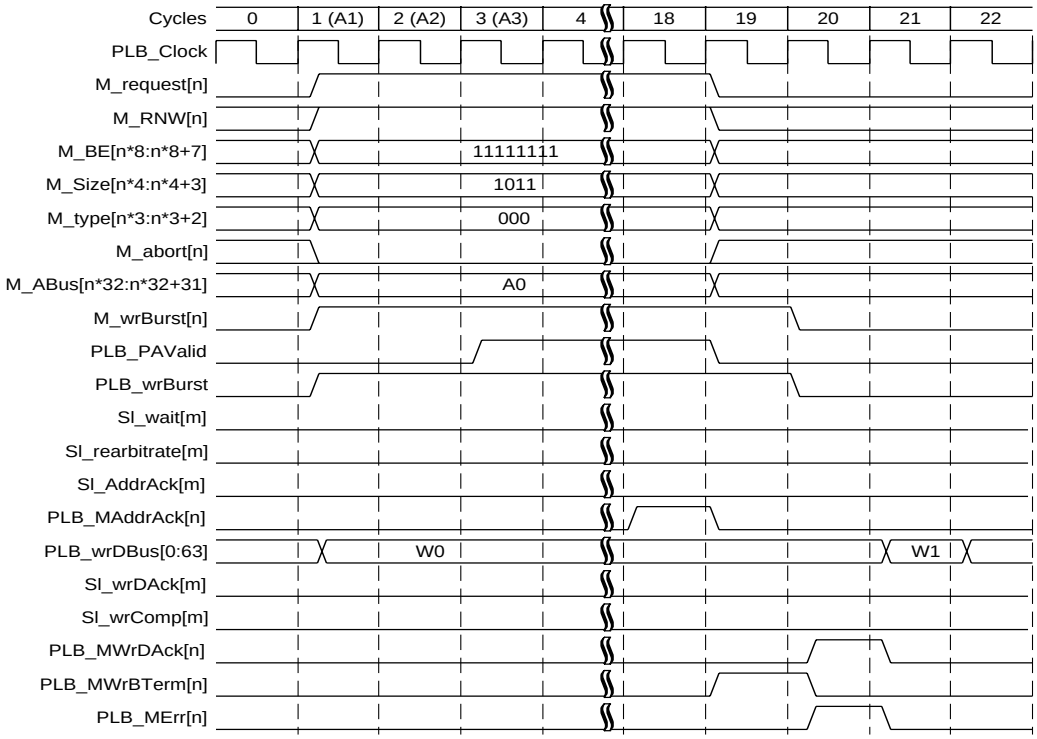


Figure 12: **Burst Write Transfer Bus Time-Out**

Read Transfer

Figure 13 shows the operation of a single read data transfer on the PLB bus with one cycle delay before **PLB_PAValid** is driven out. Note that the **Master [n]** transfer qualifiers are driven to the PLB arbiter in cycle 1, and then they are driven out onto the PLB bus in cycle number 2

after this master wins the grant from the PLB arbiter. Thus these qualifiers are available at the beginning of the address valid cycle (cycle #3) for the slave to use.

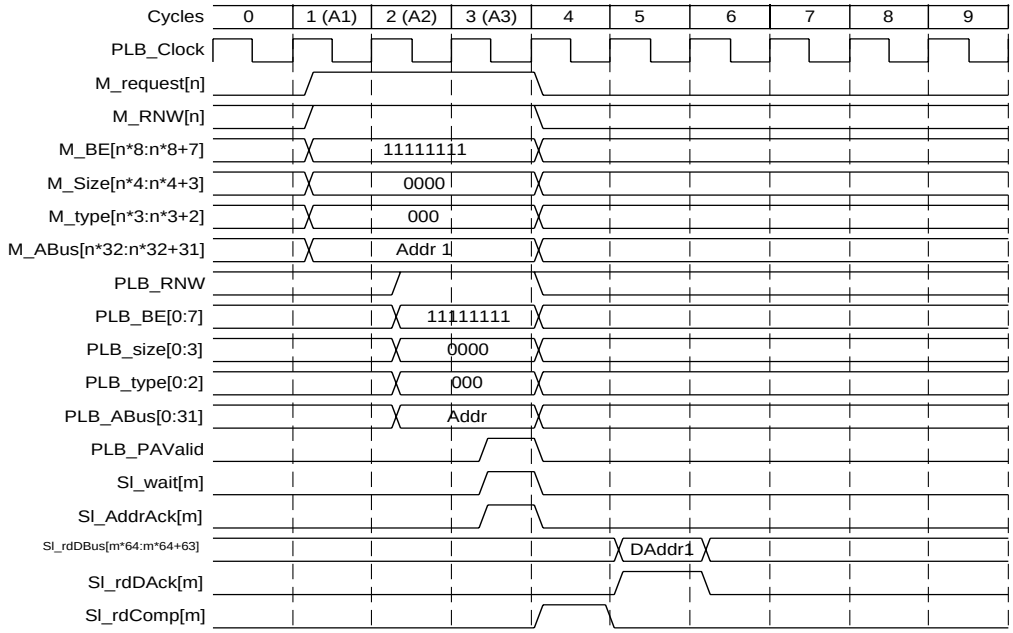


Figure 13: **Read Transfer**

Pipelined Read Transfer

In Figure 14 a master is requesting multiple reads. The first read is a primary read that is acknowledged by the slave in cycle 3. The master has another request pending, keeping its request on, which becomes a secondary read request on the PLB Bus in clock 6. This is acknowledged by the slave device in the next clock cycle as **Sl_rdComp** is given. Note that **PLB_rdPrim** is given with **Sl_rdComp**.

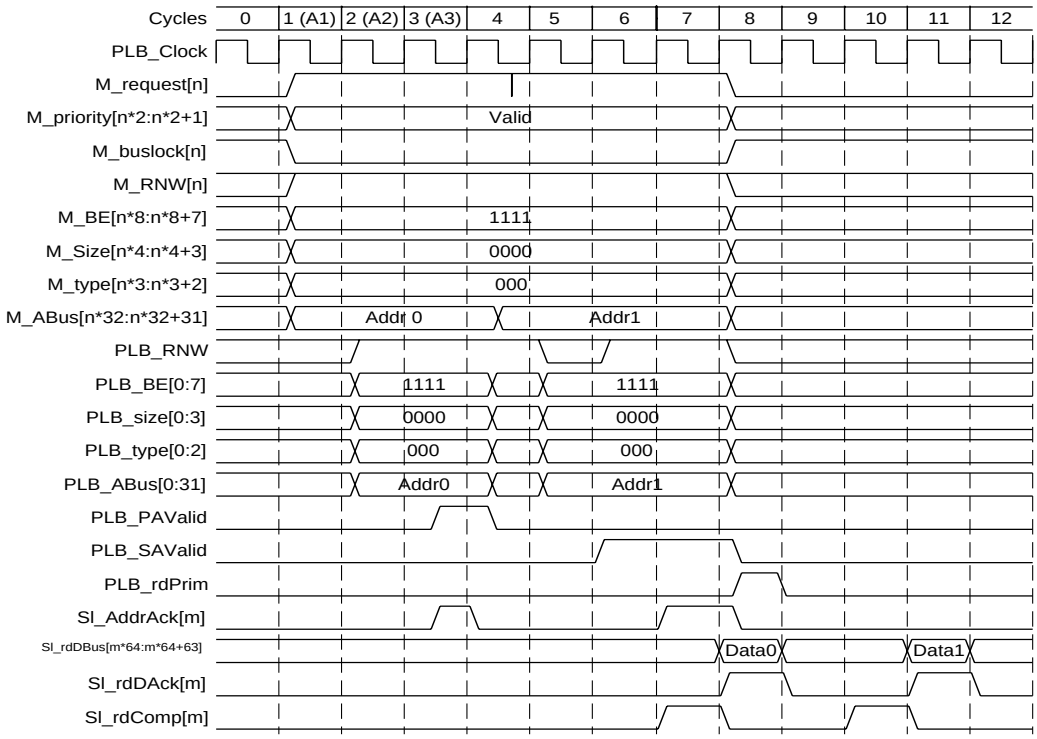


Figure 14: **Back-to-Back Reads**

Pipelined Write Transfer

In Figure 15, a master is requesting multiple writes. The first write is a primary write and is the next request becomes a secondary write request which is acknowledged by the slave device. **PLB_wrPrim** is driven in the cycle that the first write completes (slave asserts **Sl_wrComp**).

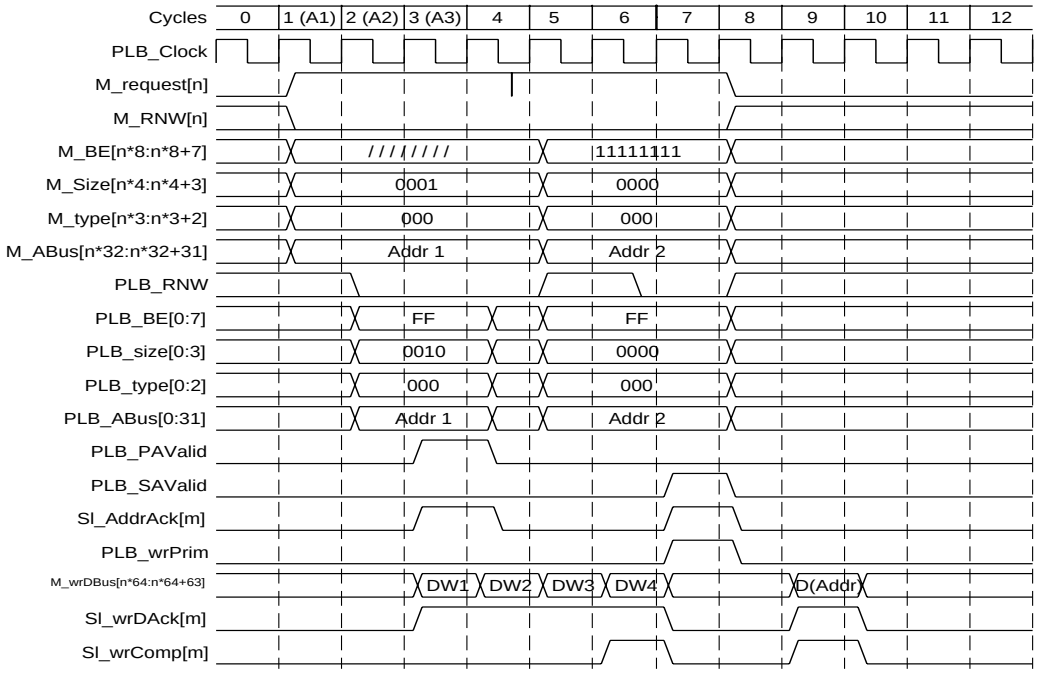


Figure 15: **Pipelined Write Transfer**

Design Implementation

Device Utilization and Performance Benchmarks

Since the PLB Arbiter is a module that will be used with other design pieces in the FPGA, the utilization and timing numbers reported in this section are just estimates. As the PLB Arbiter is combined with other pieces of the FPGA design, the utilization of FPGA resources and timing of the PLB Arbiter design will vary from the results reported here.

In order to analyze the PLB Arbiter's timing within the FPGA, a design was created that instantiated the PLB Arbiter with registers on all of the PLB Arbiter inputs and outputs. This allowed a constraint to be placed on the clock net for the PLB Arbiter to yield more realistic timing results. The f_{MAX} parameter shown in Table 20 was calculated with registers on the PLB Arbiter inputs and outputs. Note however, that the resource utilizations reported in Table 20 do not include the registers on the PLB Arbiter inputs and outputs.

The PLB Arbiter benchmarks are shown in Table 20 for a Virtex-II -5 FPGA using multi-pass place and route.

Table 20: **PLB Arbiter FPGA Performance and Resource Utilization Benchmarks (Virtex-II -5)**

Parameter Values			Device Resources			f_{MAX} (MHz)
C_NUM_MASTERS	C_NUM_SLAVES	C_DCR_INTFCE	Slices	Slice Flip-Flops	4-input LUTs	f_{MAX}
1						
2						
2						
4						

Table 20: **PLB Arbiter FPGA Performance and Resource Utilization Benchmarks (Virtex-II -5) *(Continued)***

4						
4						
4						
4						
4						
8						
8						

Notes:

1. These benchmark designs contain only the PLB Arbiter with registered inputs/outputs without any additional logic. Benchmark numbers approach the performance ceiling rather than representing performance under typical user conditions.
2. Device resource numbers do not include the registers for the PLB Arbiter I/O.
3. Max frequency calculated with registers on the PLB Arbiter I/O.

Specification Exceptions

The Xilinx PLB Arbiter is completely Core-connect compliant and has successfully passed the IBM-supplied PLB Compliance test suite. This section outlines the minor implementation differences between the Xilinx PLB Arbiter and the IBM PLB Arbiter. These implementation differences are due to optimization of the Xilinx PLB Arbiter to better utilize the features that exist in Xilinx FPGAs.

PLB Bus Structure

The Xilinx PLB Arbiter provides the full PLB bus structure. No external OR gates are required for the slave input data. Each PLB slave connects directly to the Xilinx PLB Arbiter as shown in Figure 1.

PLB Arbiter Registers

The PLB Arbiter contains seven DCR-accessible registers to provide error address and status information if the design has been parameterized to contain a DCR interface (C_DCR_INTFCE = 1). There are four PESR registers that provide error information - was an error detected, which Master's address and byte enables are in the PEARs, was the error due to a read or write transaction, and did the master lock the error condition. If a read of the PESR_MERR_DETECT register returns all zeros, then no masters detected any errors and no further reads are necessary. There are two PEAR registers - one contains the PLB address of the transaction that caused the error and the other contains the value of the PLB byte enables during the transaction that caused the error. There is one PLB Arbiter Control Register (PACR) that enables or disables the interrupt request output from the PLB Arbiter and provides a PLB Arbiter reset. Note that since the Xilinx PLB Arbiter does not support High Bus Utilization, the PLB Arbiter Control Register (PACR) does not contain a bit for controlling this feature.

All information in the IBM PEAR and PESR is implemented in the Xilinx PLB Arbiter register set, but the register organization and addresses are different than in the IBM implementation. Refer to **PLB Arbiter Registers**, page 45 for more information.

Address Phase Arbitration

The Xilinx PLB Arbiter has 3-cycle arbitration during the address phase of the transaction as shown in Figure 2. This means that PAValid is asserted 1-clock later than in the IBM implementation.

I/O Signals

The master interface signals and many of the PLB signals have been combined into a bus with an index that varies with the number of masters. This modification more easily supports the parameterization of the number of masters and the number of slaves supported by the Xilinx PLB Arbiter. Table 23 summarizes the I/O signal name modifications and variations.

Table 23: **Xilinx PLB Arbiter/IBM PLB Arbiter Signal Cross Reference Table**

IBM Signal Name	Xilinx Signal Name	Description
DCR Signals		
CPU_dcrAddr(0:9)	DCR_ABus[0:C_DCR_AWIDTH-1]	CPU DCR address bus
CPU_exeMfDcr	DCR_Read	CPU read from DCR indicator
CPU_exeMtDcr	DCR_Write	CPU write to DCR indicator
XXX_dcrData(0:31)	DCR_DBus[0:C_DCR_DWIDTH-1]	DCR write data bus
PGM_dcrAddr(0:5)	implemented as C_BASEADDR generic	Program DCR address space locator
PLB_dcrAck	same	PLB DCR data transfer acknowledge
PLB_dcrData(0:31)	PLB_dcrDBus[0:C_DCR_DWIDTH-1]	PLB DCR read data bus
LSSD Signals(1)		
LSSD_AClk	not implemented(1)	LSSD scan A clock
LSSD_BClk	not implemented(1)	LSSD scan B clock
LSSD_CClk	not implemented(1)	LSSD scan C clock
LSSD_scanGate	not implemented(1)	LSSD scan gate
LSSD_scanIn	not implemented(1)	LSSD scan input
LSSD_scanOut	not implemented(1)	LSSD scan output
Master Signals		
Mn_abort	M_abort[0:C_NUM_MASTERS-1]	Master n abort bus request indicator
Mn_ABus(0:31)	M_ABus[0:C_NUM_MASTERS*C_PLB_AWIDTH-1]	Master n address bus
Mn_BE(0:7)	M_BE[0:C_NUM_MASTERS*C_PLB_DWIDTH/8-1]	Master n byte enables
Mn_busLock	M_busLock[0:C_NUM_MASTERS-1]	Master n bus lock
Mn_compress	M_compress[0:C_NUM_MASTERS-1]	Master n compressed data transfer indicator
Mn_guarded	M_guarded[0:C_NUM_MASTERS-1]	Master n guarded transfer indicator
Mn_lockErr	M_lockErr[0:C_NUM_MASTERS-1]	Master n lock error indicator
Mn_mSize(0:1)	M_mSize[0:C_NUM_MASTERS*2 -1]	Master n data bus port width
Mn_ordered	M_ordered[0:C_NUM_MASTERS-1]	Master n synchronize transfer indicator
Mn_priority(0:1)	M_priority[0:C_NUM_MASTERS*2 -1]	Master n bus request priority
Mn_rdBurst	M_rdBurst[0:C_NUM_MASTERS-1]	Master n burst read transfer indicator

Table 23: **Xilinx PLB Arbiter/IBM PLB Arbiter Signal Cross Reference Table**

IBM Signal Name	Xilinx Signal Name	Description
Mn_request	M_request[0:C_NUM_MASTERS-1]	Master n bus request
Mn_RNW	M_RNW[0:C_NUM_MASTERS-1]	Master n read not write
Mn_size(0:3)	M_size[0:C_NUM_MASTERS*4 -1]	Master n transfer size
Mn_type(0:2)	M_type[0:C_NUM_MASTERS*3-1]	Master n transfer type
Mn_wrBurst	M_wrBurst[0:C_NUM_MASTERS-1]	Master n burst write transfer indicator
Mn_wrDBus(0:63)	M_wrDBus[0:C_NUM_MASTERS*C_PLB_DWIDTH -1]	Master n write data bus
PGM Signals		
PGM_DlyAValid	not implemented(2)	Set address valid response to either 1 or 2 clocks
PLB Signals		
PLB_abort	same	PLB abort bus request indicator
PLB_ABus(0:31)	PLB_ABus[0:C_PLB_AWIDTH-1]	PLB address bus
PLB_BE(0:7)	PLB_BE[0:C_PLB_DWIDTH/8-1]	PLB byte enables
PLB_busLock	same	PLB bus lock
PLB_compress	same	PLB compressed data transfer indicator
PLB_guarded	same	PLB guarded transfer indicator
PLB_lockErr	same	PLB lock error indicator
PLB_masterID(0:2)	PLB_masterID[0:log2(C_NUM_MASTERS)-1]	PLB current master identifier
PLB_MnAddrAck	PLB_MAddrAck[0:C_NUM_MASTERS-1]	PLB master n address acknowledge
PLB_MnBusy	PLB_MBusy[0:C_NUM_MASTERS-1]	PLB master n slave busy indicator
PLB_MnErr	PLB_MErr[0:C_NUM_MASTERS-1]	PLB master n slave error indicator
PLB_MnRdBTerm	PLB_MRdBTerm[0:C_NUM_MASTERS-1]	PLB master n terminate read burst indicator
PLB_MnRdDAck	PLB_MRdDAck[0:C_NUM_MASTERS-1]	PLB master n read data acknowledge
PLB_MnRdDBus(0:63)	PLB_MRdDBus[0:C_NUM_MASTERS*C_PLB_DWIDTH -1]]	PLB master n read data bus
PLB_MnRdWdAddr(0:3)	PLB_MRdWdAddr[0:C_NUM_MASTERS*4 -1]	PLB master n read word address
PLB_MnRearbitrate	PLB_MRearbitrate[0:C_NUM_MASTERS-1]	PLB master n bus rearbitrate indicator
PLB_MnsSize(0:1)	PLB_MSSize[0:C_NUM_MASTERS*2 -1]	PLB master n slave data bus port width
PLB_MnWrBTerm	PLB_MWrBTerm[0:C_NUM_MASTERS-1]	PLB master n terminate write burst indicator
PLB_MnWrDAck	PLB_MWrDAck[0:C_NUM_MASTERS-1]	PLB master n write data acknowledge
PLB_mSize(0:1)	PLB_MSize[0:1]	PLB data bus port width indicator
PLB_ordered	same	PLB synchronize transfer indicator
PLB_PAValid	same	PLB primary address valid indicator for up to 66MHz

Table 23: **Xilinx PLB Arbiter/IBM PLB Arbiter Signal Cross Reference Table**

IBM Signal Name	Xilinx Signal Name	Description
PLB_PAValid2	not implemented(3)	PLB primary address valid indicator for over 66MHz
PLB_pendPri(0:1)	same	PLB pending request priority
PLB_pendReq	same	PLB pending bus request indicator
PLB_rdBurst	same	PLB burst read transfer indicator
PLB_rdPrim	same	PLB secondary to primary read request indicator
PLB_rdWdAddrWDT(0:3)	same name - implemented as internal signal	PLB read word address for line reads that time out
PLB_reqPri(0:1)	same	PLB current request priority
PLB_RNW	same	PLB read not write
PLB_SAValid	same	PLB secondary address valid for up to 66MHz
PLB_SAValid2	not implemented(3)	PLB secondary address valid for over 66MHz
PLB_size(0:3)	same	PLB transfer size
PLB_sleepReq	not implemented(4)	PLB core sleep request
PLB_type(0:2)	same	PLB transfer type
PLB_wrBurst	same	PLB burst write transfer indicator
PLB_wrDBus(0:63)	PLB_wrDBus[0:C_PLB_DWIDTH-1]	PLB write data bus
PLB_wrPrim	same	PLB secondary to primary write request indicator
Slave signals		
Sl_addrAck	Sl_addrAck[0:C_NUM_SLAVES-1]	Slave address acknowledge
Sl_err(0:7)	Sl_MErr[0:C_NUM_SLAVES*C_NUM_MASTERS-1]	Slave error indicator
Sl_MBusy(0:7)	Sl_MBusy[0:C_NUM_SLAVES*C_NUM_MASTERS-1]	Slave busy indicator
Sl_rdBTerm	Sl_rdBTerm[0:C_NUM_SLAVES-1]	Slave terminate read burst transfer
Sl_rdComp	Sl_rdComp[0:C_NUM_SLAVES-1]	Slave read transfer complete indicator
Sl_rdDAck	Sl_rdDAck[0:C_NUM_SLAVES-1]	Slave read data acknowledge
Sl_rdDBus(0:63)	Sl_rdDBus[0:C_NUM_SLAVES*C_PLB_DWIDTH-1]	Slave read data bus
Sl_rdWdAddr(0:3)	Sl_rdWdAddr[0:C_NUM_SLAVES*4-1]	Slave read word address
Sl_rearbitrate	Sl_rearbitrate[0:C_NUM_SLAVES-1]	Slave rearbitrate bus indicator
Sl_sSize(0:1)	Sl_SSize[0:C_NUM_SLAVES*2-1]	Slave data bus port size indicator
Sl_wait	Sl_wait[0:C_NUM_SLAVES-1]	Slave wait indicator
Sl_wrBTerm	Sl_wrBTerm[0:C_NUM_SLAVES-1]	Slave terminate write burst transfer
Sl_wrComp	Sl_wrComp[0:C_NUM_SLAVES-1]	Slave write transfer complete indicator
Sl_wrDAck	Sl_wrDAck[0:C_NUM_SLAVES-1]	Slave write data acknowledge
System Signals		

Table 23: **Xilinx PLB Arbiter/IBM PLB Arbiter Signal Cross Reference Table**

IBM Signal Name	Xilinx Signal Name	Description
Reset	same	Reset
SysClk	Clk	System clock
Xilinx PLB Arbiter Unique Signals		
N/A	ArbReset	Registered reset from arbiter
N/A	ArbAddrVldReg	Indicates either PAValid or SAValid is asserted
N/A	Bus_Error_Det	Bus error interrupt
IBM Toolkit Support Signals(5)		
N/A	PLB_SaddrAck	Output of slave Sl_addrAck OR gate
N/A	PLB_Swait	Output of slave Sl_wait OR gate
N/A	PLB_Srearbitrate	Output of slave Sl_rearbitrate OR gate
N/A	PLB_SwrDAck	Output of slave Sl_wrDAck OR gate
N/A	PLB_SwrComp	Output of slave Sl_wrComp OR gate
N/A	PLB_SwrBTerm	Output of slave Sl_wrBTerm OR gate
N/A	PLB_SrdDBus[0:C_PLB_DWIDTH-1]	Output of slave Sl_rdDBus OR gate
N/A	PLB_SrdWdAddr[0:3]	Output of slave Sl_rdWdAddr OR gate
N/A	PLB_SrdDAck	Output of slave Sl_rdDAck OR gate
N/A	PLB_SrdComp	Output of slave Sl_rdComp OR gate
N/A	PLB_SrdBTerm	Output of slave Sl_rdBTerm OR gate
N/A	PLB_SMBusy[0:C_NUM_MASTERS-1]	Output of slave Sl_MBusy OR gate
N/A	PLB_SMErr[0:C_NUM_MASTERS-1]	Output of slave Sl_MErr OR gate
N/A	PLB_Sssize[0:1]	Output of slave Sl_SSize OR gate

Notes:
1. LSSD is not implemented in the Xilinx PLB Arbiter
2. The Xilinx PLB arbiter implements 3-cycle arbitration in the address phase, therefore, this signal is not needed.
3. The Xilinx PLB arbiter PAValid and SAValid signals are valid across all clock frequencies. Signals PAValid2 and SAValid2 are not needed.
4. A power-down or sleep mode is not implemented in the Xilinx PLB Arbiter.
5. The Xilinx arbiter contains the slave OR gates. The outputs of the slave OR gates are supplied to connect to the PLB Monitor BFM.

Clock and Power Management

The IBM PLB Arbiter Core supports clock and power management by gating clocks to all internal registers and providing a sleep request signal to a central clock and power management unit in the system. This sleep request signal is asserted by the IBM PLB Arbiter to indicate when it is permissible to shut off clocks to the arbiter. These functions are not supported in the Xilinx implementation of the PLB Arbiter, therefore the following I/O signal is not used:

- ARB_sleepReq -the FPGA implementation of the PLB bus will not support sleep modes

System Reset

The IBM PLB Arbiter has specific requirements about the duration of the reset signal. The Xilinx PLB Arbiter does not have these requirements. The PLB Arbiter reset is filtered and synchronized in the Processor System Reset Module.

Arbitration Priority

The Xilinx PLB Arbiter implements fixed priority when two or more masters have the same priority inputs. Priority order in this case is Master 0, Master 1, Master 2,... Master N.

High Bus Utilization

The Xilinx PLB Arbiter does not support the high bus utilization feature.

LSSD

The Xilinx PLB Arbiter does not support LSSD.

Reference Documents

The following documents contain reference information important to understanding the Xilinx PLB Arbiter design:

- IBM 64-Bit Processor Local Bus Architectural Specification
- IBM 64-Bit Processor Local Bus Arbiter Core User's Manual
- IBM 32-Bit Device Control Register Bus Architecture Specifications.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
	v1.00a	Initial Xilinx release.
1/31/02	v1.01a	Release to incorporate Slave OR gates
2/27/02		