LogiCORE IP On-Chip Peripheral Bus V2.0 with OPB Arbiter (v1.00d)

DS402 April 19, 2010 **Product Specification**

Introduction

The LogiCORE IP OPB_V20 On-Chip Peripheral Bus V2.0 with OPB Arbiter (OPB_V20) module is used as the OPB interconnect for Xilinx FPGA based embedded processor systems. The bus interconnect in the OPB V2.0 specification is a distributed multiplexer implemented as an AND function in the master or slave driving the bus, and an OR function to combine the drivers into a single bus.

Features

- Includes parameterized OPB Arbiter
- Includes parameterized I/O signals to support up to 16 masters and any number of slaves. Xilinx recommends a maximum of 16 slaves on the OPB.
- Includes all signals present in the OPB V2.0 specification except for DMA handshake signals
- The OR structure can be implemented using only LUTs or can use a combination of LUTs and fast carry adder to reduce the number of LUTs in the OR interconnect
- Includes a 16-clock Power-on OPB Bus Reset and parameter for high or low external bus reset
- Includes input for reset from Watchdog Timer

The OPB_V20 includes an OPB Arbiter that incorporates the features contained in the IBM On-Chip Peripheral Bus Arbiter core manual (version 1.5) for 32-bit implementation. This manual is referenced throughout this document and is considered the authoritative specification. Any differences between the IBM OPB Arbiter implementation and the Xilinx OPB Arbiter implementation are mentioned in the Specification Exceptions section.

The Xilinx OPB V20 bus core allows the designer to tailor the OPB bus and arbiter to suit the application by setting certain parameters to enable/disable features.

In some cases, setting these parameters may cause the Xilinx OPB Arbiter design to deviate slightly from the IBM OPB Arbiter specification.These parameters are described in the OPB_V20 Design Parameters section.

LogiCORE™ IP Facts	
Core Specifics	
Supported Device Family (1)	Virtex®-5/-5FX, Virtex-4/-4Q/-4QV, Automotive Spartan®-3/-3E/-3A, -3A DSP
Resources Used	See Table 7.
Special Features	None
Provided with Core	
Documentation	Product Specification
Design File Formats	VHDL
Constraints File	N/A
Verification	N/A
Instantiation Template	N/A
Additional Items	None
Design Tool Requirements	
Xilinx Implementation Tools	ISE® 12.1
Verification	Mentor Graphics ModelSim v6.5c and above
Simulation	Mentor Graphics ModelSim v6.5c and above
Synthesis	XST
Support	
Provided by Xilinx, Inc.	

1. For a complete list of supported devices, see the 12.1 release notes for this core.

Functional Description

The diagram of the OPB System using the OPB V20 is shown in Figure 1.

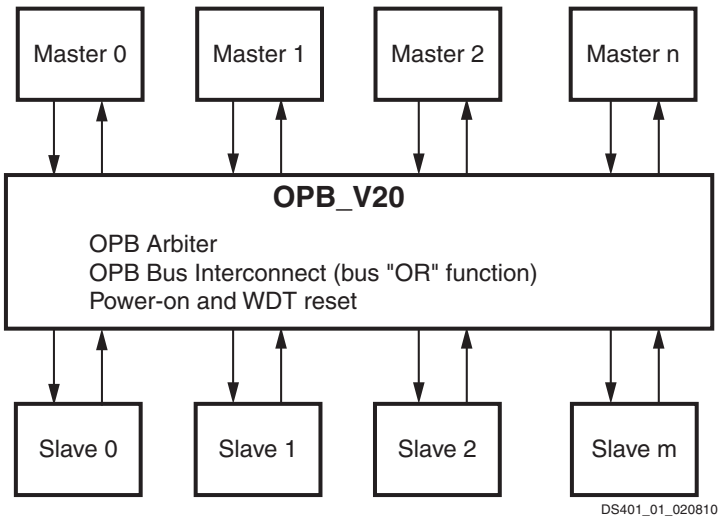

Figure 1: **OPB System Using the OPB V2.0 Device**

Bus Interconnect Overview

OPB Connections

An OPB system is composed of masters, slaves, a bus interconnect, and an arbiter as shown in Figure 1. The OPB_V20 implements the bus interconnect and arbiter function in an OPB system. In Xilinx FPGAs, the OPB is implemented as a simple OR structure.

The OPB bus signals are created by logically OR'ing the signals that drive the bus. OPB devices that are not active during a transaction are required to drive zeros into the OR structure. This structure forms a distributed multiplexer and results in efficient bus implementations in FPGAs. Bus arbitration signals such as M_request and OPB_MGrant are directly connected between the OPB arbiter and each OPB master device.

Parameterization for Multiple Masters and Slaves

The OPB_V20 supports up to 16 masters and an unlimited number of slaves (up to the hardware resources available). The port widths into the OPB_V20 are designed to increase in size as more masters or slaves are added. Xilinx recommends a maximum of 16 slaves on the OPB.

For example, the Master Data bus port is the width of the OPB data bus times the number of masters in the system. For a 32-bit OPB, Master 0 occupies M_DBus(0:31), Master 1 occupies M_DBus(32:63), and so on. Similarly for slaves, Slave 0 occupies Sl_DBus(0:31), Slave 1 occupies Sl_DBus(32:63), and so on. The 32-bit OPB data bus (OPB_DBus) is formed by OR'ing all the master and slave data buses together. As an example, the signals that are OR'ed to form the 32-bit OPB bus for a three master and three slave OPB system are shown in the following table:

Table 1: **OPB Bus OR Structure Example**

OPB Signal	Signals OR'ed Together
OPB_ABus(0:31)	M_ABus(0:31) or M_ABus(32:63) or M_ABus(64:95)
OPB_BE(0:3)	M_BE(0:3) or M_BE(4:7) or M_BE(8:11)
OPB_beXfer	M_beXfer(0) or M_beXfer(1) or M_beXfer(2)
OPB_beAck	Sl_beAck(0) or Sl_beAck(1) or Sl_beAck(2)
OPB_busLock	M_busLock(0) or M_busLock(1) or M_busLock(2)
OPB_rdDBus(0:31)	Sl_DBus(0:31) or Sl_DBus(32:63) or Sl_DBus(64:95)
OPB_wrDBus(0:31)	M_DBus(0:31) or M_DBus(32:63) or M_DBus(64:95)
OPB_DBus(0:31)	OPB_rdDBus(0:31) or OPB_wrDBus(0:31)
OPB_errAck	Sl_errAck(0) or Sl_errAck(1) or Sl_errAck(2)
OPB_dwAck	Sl_dwAck(0) or Sl_dwAck(1) or Sl_dwAck(2)
OPB_dwXfer	M_dwXfer(0) or M_dwXfer(1) or M_dwXfer(2)
OPB_fwAck	Sl_fwAck(0) or Sl_fwAck(1) or Sl_fwAck(2)
OPB_fwXfer	M_fwXfer(0) or M_fwXfer(1) or M_fwXfer(2)
OPB_hwAck	Sl_hwAck(0) or Sl_hwAck(1) or Sl_hwAck(2)
OPB_hwXfer	M_hwXfer(0) or M_hwXfer(1) or M_hwXfer(2)
OPB_retry	Sl_retry(0) or Sl_retry(1) or Sl_retry(2)
OPB_RNW	M_RNW(0) or M_RNW(1) or M_RNW(2)
OPB_select	M_select(0) or M_select(1) or M_select(2)
OPB_seqAddr	M_seqAddr(0) or M_seqAddr(1) or M_seqAddr(2)
OPB_toutSup	Sl_toutSup(0) or Sl_toutSup(1) or Sl_toutSup(2)
OPB_xferAck	Sl_xferAck(0) or Sl_xferAck(1) or Sl_xferAck(2)

For more information on the OPB bus structure, see the IBM document, *On-Chip Peripheral Bus, Architecture Specifications*.

Reset Logic

The OPB_V20 design includes several sources for bus reset. A power-on reset circuit asserts the OPB_Rst for 16 clock cycles anytime the FPGA has completed configuration. External resets that occur during the 16 clock reset time are ignored.

After the 16-clock reset has completed, external resets can be applied to the OPB_V20 reset signals. The external resets are: SYS_Rst (can be configured as high-true or low-true), WDT_Rst, and Debug_SYS_Rst. SYS_Rst is the main user reset for the OPB and can be connected to internal logic or an external signal or switch.

WDT_Rst can be connected to the reset output of a Watchdog Timer to allow for OPB resets in the event of a watchdog time-out. Debug_SYS_Rst can be connected to the reset output of a debug peripheral, such as the JTAG UART, so that the debugger can remotely reset the OPB. SYS_Rst, WDT_Rst, and Debug_SYS_Rst are synchronized to the OPB_Clk in the OPB_V20, but the width of these signals is otherwise unaltered.

OPB Arbiter Overview

Features

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

- Optional OPB slave interface (included in design via a design parameter)
- OPB Arbitration
  - arbitrates between 1-16 OPB Masters (the number of masters is parameterizable)
  - arbitration priorities among masters programmable via register write
  - priority arbitration mode configurable via a design parameter
    - Fixed priority arbitration with processor access to read/write Priority Registers
    - Dynamic priority arbitration implementing a true least recent used (LRU) algorithm
- Two bus parking modes selectable via Control Register write:
  - park on selected OPB master (specified in Control Register)
  - park on last OPB master which was granted OPB access
- Watchdog timer which asserts the OPB time-out signal if a slave response is not detected within 16 clock cycles.
- Registered or combinational Grant outputs configurable via a design parameter.

OPB Arbitration Protocol

OPB Bus arbitration uses the following protocol:

1. An OPB master asserts its bus request signal.
2. The OPB Arbiter receives the request and outputs an individual grant signal to each master according to its priority and the state of the other requests.
3. An OPB master samples its grant signal at the rising edge of the OPB clock. In the following cycle, the OPB master initiates a data transfer between the master and a slave by asserting its select signal.

The OPB Arbiter only issues a bus grant signal during valid arbitration cycles which are defined as either:

- Idle
- The OPB_select and OPB_busLock are de-asserted, indicating that no data transfer is in progress.
- Overlapped arbitration cycle

  The OPB_xferAck is asserted, indicating the final cycle in a data transfer, and OPB_busLock is not asserted. Arbitration in this cycle allows another master to begin a transfer in the following cycle, avoiding the need for a *dead* cycle on the bus.

The Xilinx OPB Arbiter can be configured to have either registered or combinational grant outputs. Registered grant outputs are asserted one clock after each arbitration cycle resulting in one *dead* cycle on the bus. However, registered grant outputs allow the OPB bus to run at higher clock rates.

Figure 2 shows the fixed OPB arbitration protocol with combinational grant outputs, Figure 3 shows the fixed OPB arbitration protocol when the OPB Arbiter has been parameterized to have registered grant outputs.

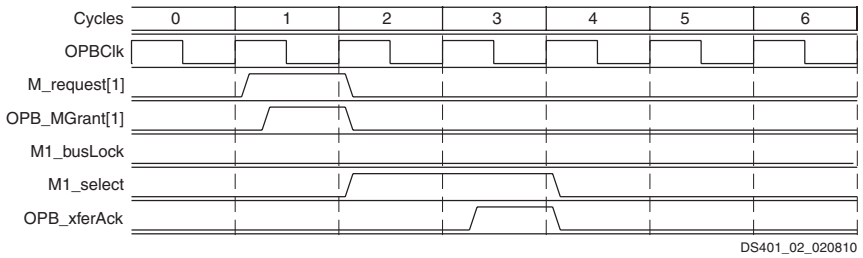

Figure 2: **OPB Fixed Bus Arbitration - Combinational Grant Outputs**

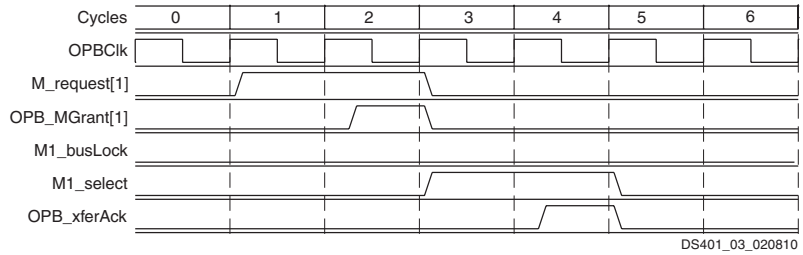

Figure 3: **OPB Fixed Bus Arbitration - Registered Grant Outputs**

An OPB master device need not de-assert its request upon receipt of a bus grant signal if it has multiple bus transfer cycles to perform. Figure 4 shows an OPB arbitration cycle in which an OPB master asserts bus request continuously for four data transfer cycles. The OPB Arbiter has been parameterized for fixed priority arbitration and combinational grant outputs, consequently bus grant is asserted combinationally during valid arbitration cycles.

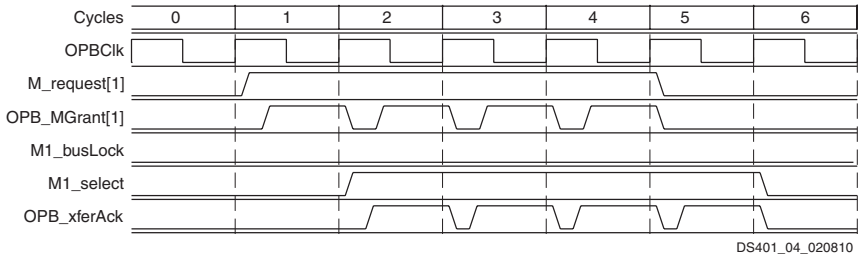

Figure 4: **Continuous Master Bus Request - Fixed priority, Combinational Grant Outputs**

When the OPB Arbiter has been parameterized for registered grant outputs and fixed priority, the bus grants are registered as shown in Figure 5.

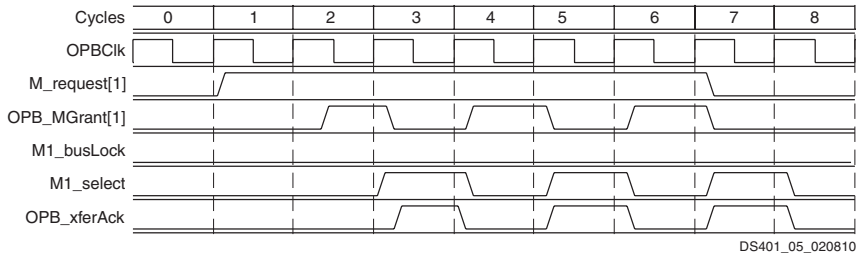

Figure 5: **Continuous Master Bus Request - Fixed priority, Registered Grant Output**

Even if an OPB master asserts request continuously, it does not necessarily receive a valid grant signal. Other OPB masters with higher bus priority may request the OPB and are granted the bus according to OPB arbiter priority.

If an OPB master device needs a non-interruptible sequence of bus cycles, it can use the bus lock signal for this purpose. Bus locking is described later in this document.

The OPB Arbiter supports both dynamic priority arbitration, implementing a Least Recently Used (LRU) algorithm, and fixed priority arbitration, both are described in more detail later in this document.

Figure 6 shows multiple bus request or overlapped bus arbitration when the OPB arbiter is using fixed priority arbitration and combinational grant outputs. Both OPB Master 1 and OPB Master 2 simultaneously request the bus.

Master 1 has a higher priority and is granted the bus. During cycle 2, Master 1 completes its first transaction and Master 2 is granted the bus for cycle 3. Thus, during cycle 2, the arbitration for the bus is overlapped with a data transfer. This overlapped bus arbitration improves the bandwidth of the bus.

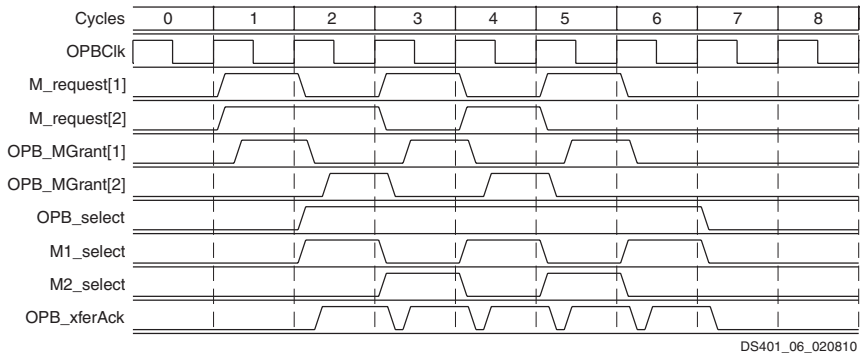

Figure 6: **Multiple Bus Requests - Fixed Priority Arbitration, Combinational Grant Outputs**

With registered grant outputs, there is a cycle between bus grant signals as shown in Figure 7. Using registered grant outputs from the OPB arbiter reduces the number of logic levels between registers and allows the OPB bus to run at a higher clock rate.

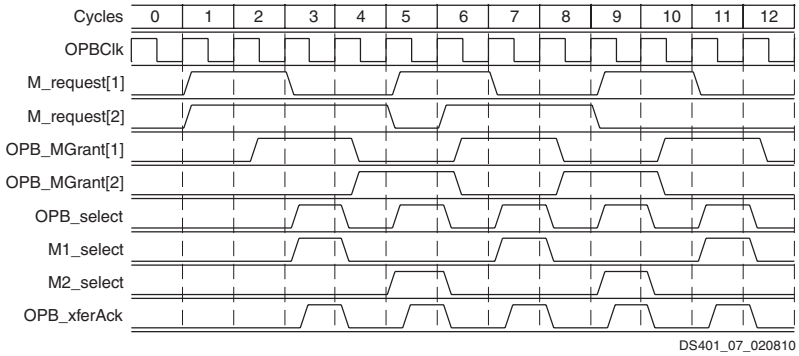

Figure 7: **Multiple Bus Requests - Fixed Priority Arbitration, Registered Grant Outputs**

OPB_V20 Design Parameters

To obtain an OPB bus that is uniquely tailored to the system, certain features in the OPB_V20 design can be parameterized. This allows the designer to have a high performance design that only utilizes the resources required by the system. The features that can be parameterizable in the Xilinx OPB_V20 design are shown in Table 2.

Table 2: **OPB_V20 Design Parameters**

Grouping/Number		Feature/Description	Parameter Name	Allowable Values	Default Value	VHDL Type
OPB Bus Features	G1	Number of OPB Masters	C_NUM_MASTERS	1<SP Superscript>[1] - 16	4	integer
	G2	Number of OPB Slaves	C_NUM_SLAVES	16<SP Superscript>[2]	4	integer
	G3	OPB Data Bus Width	C_OPB_DWIDTH	8,16,32,64,128	32	integer
	G4	OPB Address Bus Width	C_OPB_AWIDTH	16 - 32	32	integer
	G5	Use only LUTs for OR structure (no fast carry)	C_USE_LUT_OR	0 = Use Fast Carry 1 = Use LUT only	1	integer
OPB Bus Features	G6	Level of external reset	C_EXT_RESET_HIGH	0 = Low-true external reset 1 = High-true external reset	1	integer
Arbiter Features	G7	Priority Mode	C_DYNAM_PRIORITY	1 = Dynamic 0 = Fixed	0 = Fixed	integer
	G8	Registered Grant Outputs	C_REG_GRANTS<SP Superscript>[3]	1 = Registered Grant Outputs 0 = Combinational Grant Outputs	1 = Registered Grant Outputs	integer
	G9	Bus Parking	C_PARK	1 = Bus parking supported<SP Superscript>[4] 0 = Bus parking not supported	0 = Bus parking not supported	integer

Table 2: **OPB_V20 Design Parameters *(Cont'd)***

Grouping/Number		Feature/Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Arbiter Slave Interface	G10	OPB Slave Interface	C_PROC_INTRFCE	1 = OPB slave interface supported 0 = OPB slave interface not supported<SP Superscript>[5]	0 = OPB slave interface not supported	integer
	G11	OPB Arbiter Base Address	C_BASEADDR	Valid Address Range<SP Superscript>[6]	None<SP Superscript>[7]	std_logic_vector
	G12	OPB High Address	C_HIGHADDR	Address range must be a power of 2 and >= 0x1FF<SP Superscript>[6]	None<SP Superscript>[7]	integer
	G13	Device Block ID	C_DEV_BLK_ID	0 - 255	0	integer
	G14	Module Identification Register Enable	C_DEV_MIR_ENABLE	0 , 1	0	integer

Notes:

1. When C_NUM_MASTERS = 1, no arbitration is necessary, however, the watchdog timer is included in the arbiter and is needed for the OPB. In this case, all other parameters are meaningless.
2. Maximum value of 16 for C_NUM_SLAVES is not a restriction of the core, but the max value for which this core has been simulated.
3. C_REG_GRANTS should only be set to 0 (indicating that the Grant outputs are combinational) if the desired OPB frequency is less than the f_{MAX} specified for this parameter.
4. When bus parking is supported, the parking mode (park on last master or park on master id) is set in the OPB Arbiter Control Register. If C_PROC_INTRFCE is 0, the parking mode is park on last master.
5. When C_PROC_INTRFCE is 0, none of the OPB Arbiter registers are accessible.
6. The range specified by C_BASEADDR and C_HIGHADDR must comprise a complete, contiguous power of two range such that range = 2m, and the m least significant bits of C_BASEADDR must be zero. This range must be at least 0x1FF.
7. No default value will be specified for C_BASEADDR to insure that the actual value is set, i.e. if the value is not set, a compiler error will be generated.

Allowable Parameter Combinations

The only restriction on parameter combinations in the Xilinx OPB Arbiter is that the address range specified by C_BASEADDR and C_HIGHADDR is a power of 2. To allow for the register offset within the OPB Arbiter design, the range specified by C_BASEADDR and C_HIGHADDR must be at least 0x1FF.

The number of OPB masters can be parameterized to 1 master. Though no arbitration is necessary when there is only one OPB master, the OPB Arbiter contains the watchdog timer for the OPB and is consequently needed in the system.

When there is one OPB master, there are no Control or Priority Registers, and there is no OPB slave interface on the OPB Arbiter.

When C_NUM_MASTERS is set to 1, all other parameters are meaningless. Also, when C_PROC_INTRFCE is set to 0, the OPB Arbiter registers are not accessible.

OPB_V20 I/O Signals

The I/O signals for the OPB_V20 are listed in Table 3. The interfaces referenced in this table are shown in Figure 10 in the OPB Arbiter block diagram.

Table 3: **OPB_V20 I/O Signals**

Grouping		Signal Name	MSB:LSB	I/O	Description
System Signals	P1	SYS_Rst		I	External System Reset
	P2	WDT_Rst		I	Watchdog Timer Reset
	P3	Debug_SYS_Rst		I	Debug System Reset
Master Signals	P4	M_ABus	0:C_OPB_AWIDTH*C_NUM_MASTERS-1	I	Master Address Bus
	P5	M_BE	0:(C_OPB_DWIDTH)/8*C_NUM_MASTERS-1	I	Master Byte Enables
	P6	M_beXfer	0:C_NUM_MASTERS-1	I	Master Byte Enable Transfer
	P7	M_busLock	0:C_NUM_MASTERS-1	I	Master Buslock
	P8	M_DBus	0:C_OPB_DWIDTH*C_NUM_MASTERS-1	I	Master Databus
	P9	M_DBusEn	0:C_NUM_MASTERS-1	I	Master Databus Enable
	P10	M_DBusEn32_63	0:C_NUM_MASTERS-1	I	Master Databus Enable (bits 32:63)
	P11	M_dwXfer	0:C_NUM_MASTERS-1	I	Master Doubleword Transfer
	P12	M_fwXfer	0:C_NUM_MASTERS-1	I	Master Fullword Transfer
	P13	M_hwXfer	0:C_NUM_MASTERS-1	I	Master Halfword Transfer
	P14	M_request	0:C_NUM_MASTERS-1	I	Master Request
	P15	M_RNW	0:C_NUM_MASTERS-1	I	Master Read/ Not Write
	P16	M_select	0:C_NUM_MASTERS-1	I	Master Select
	P17	M_seqAddr	0:C_NUM_MASTERS-1	I	Master Sequential Address
Slave Signals	P18	Sl_beAck	0:C_NUM_SLAVES-1	I	Slave Byte Enable Acknowledge
	P19	Sl_DBus	0:C_OPB_DWIDTH*C_NUM_SLAVES-1	I	Slave Data Bus
	P20	Sl_DBusEn	0:C_NUM_SLAVES-1	I	Slave Data Bus Enable
	P21	Sl_DBusEn32_63	0:C_NUM_SLAVES-1	I	Slave Data Bus Enable (bits 32:63)
	P22	Sl_errAck	0:C_NUM_SLAVES-1	I	Slave Error Acknowledge
	P23	Sl_dwAck	0:C_NUM_SLAVES-1	I	Slave Doubleword Acknowledge
	P24	Sl_fwAck	0:C_NUM_SLAVES-1	I	Slave Fullword Acknowledge
	P25	Sl_hwAck	0:C_NUM_SLAVES-1	I	Slave Halfword Acknowledge
	P26	Sl_retry	0:C_NUM_SLAVES-1	I	Slave Retry
	P27	Sl_toutSup	0:C_NUM_SLAVES-1	I	Slave Timeout Suppress
	P28	Sl_xferAck	0:C_NUM_SLAVES-1	I	Slave Transfer Acknowledge

Table 3: **OPB_V20 I/O Signals *(Cont'd)***

Grouping		Signal Name	MSB:LSB	I/O	Description
OPB Signals	P29	OPB_Clk		I	OPB Clock
	P30	OPB_Rst		O	OPB Reset
	P31	OPB_MRequest	0:C_NUM_MASTERS-1	O	OPB Master Request
	P32	OPB_ABus	0:C_OPB_AWIDTH-1	O	OPB Address Bus
	P33	OPB_BE	0:(C_OPB_DWIDTH)/8-1	O	OPB Byte Enables
	P34	OPB_beXfer		O	OPB Byte Enable Transfer
	P35	OPB_beAck		O	OPB Byte Enable Acknowledge
	P36	OPB_busLock		O	OPB Buslock
	P37	OPB_rdDBus	0:C_OPB_DWIDTH-1	O	OPB Read Data Bus
	P38	OPB_wrDBus	0:C_OPB_DWIDTH-1	O	OPB Write Data Bus
	P39	OPB_DBus	0:C_OPB_DWIDTH-1	O	OPB Data Bus
	P40	OPB_errAck		O	OPB Error Acknowledge
	P41	OPB_dwAck		O	OPB Doubleword Acknowledge
	P42	OPB_dwXfer		O	OPB Doubleword Transfer
	P43	OPB_fwAck		O	OPB Fullword Acknowledge
	P44	OPB_fwXfer		O	OPB Fullword Transfer
	P45	OPB_hwAck		O	OPB Halfword Acknowledge
	P46	OPB_hwXfer		O	OPB Halfword Transfer
	P47	OPB_MGrant	0:C_NUM_MASTERS-1	O	OPB Master Grant
	P48	OPB_pendReq	0:C_NUM_MASTERS-1	O	OPB Pending Request
	P49	OPB_retry		O	OPB Retry
	P50	OPB_RNW		O	OPB Read/ Not Write
	P51	OPB_select		O	OPB Select
	P52	OPB_seqAddr		O	OPB Sequential Address
	P53	OPB_timeout		O	OPB Timeout
	P54	OPB_toutSup		O	OPB Timeout Suppress
	P55	OPB_xferAck		O	OPB Transfer Acknowledge

Notes:

1. Master names have been modified slightly from that in the IBM OPB V2.0 specification to support parameterization of the number of masters.

Parameter - Port Dependencies

The width of many of the OPB_V20 signals depends on the number of OPB masters 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 OPB Arbiter parameters and I/O signals are shown in Table 4.

OPB_V20 Arbiter Register Descriptions

The OPB Arbiter contains addressable registers for read/write operations as shown in Table 4 if the design has been parameterized to support a processor interface (C_PROC_INTRFCE = 1). The base address for these registers is set in the parameter C_BASEADDR. The registers are located at an offset of 0x00000100 from C_BASEADDR. Each register is addressable on a 32-bit boundary.

Each priority level has a unique Priority Register which contains the master id for the master at that priority level. The Priority Registers are readable and writable by the processor. The number of priority levels and consequently the number of Priority Registers varies with the parameter C_NUM_MASTERS.

Table 4 shows all of the OPB Arbiter registers and their addresses when the maximum number of masters has been selected. In this case, 17 registers are required.

Table 4: **OPB Arbiter Registers**

Register Name	OPB Address (offset from C_BASEADDR)	Access
OPB Arbiter Control Register	0x100	Read/Write
LVL0 Priority Register	0x104	Read/Write
LVL1 Priority Register	0x108	Read/Write
LVL2 Priority Register	0x10C	Read/Write
LVL3 Priority Register	0x110	Read/Write
LVL4 Priority Register	0x114	Read/Write
LVL5 Priority Register	0x118	Read/Write
LVL6 Priority Register	0x11C	Read/Write
LVL7 Priority Register	0x120	Read/Write
LVL8 Priority Register	0x124	Read/Write
LVL9 Priority Register	0x128	Read/Write
LVl10 Priority Register	0x12C	Read/Write
LVL11 Priority Register	0x130	Read/Write
LVL12 Priority Register	0x134	Read/Write
LVL13 Priority Register	0x138	Read/Write
LVL14 Priority Register	0x13C	Read/Write
LVL15 Priority Register	0x140	Read/Write

The register definitions and address locations of the Xilinx OPB Arbiter deviate from the IBM OPB Arbiter specification. This deviation is necessary to support parameterization of the number of OPB Masters. See section "Specification Exceptions" for more information.

OPB Arbiter Control Register

The OPB Arbiter Control Register is shown in Figure 8 for the case where the OPB Arbiter has been parameterized for fixed priority arbitration without bus parking with a data width of 32 bits. The reset values for the DPE, DPERW, PEN, and PENRW bits vary based on the parameterization of the arbitration and bus parking scheme.

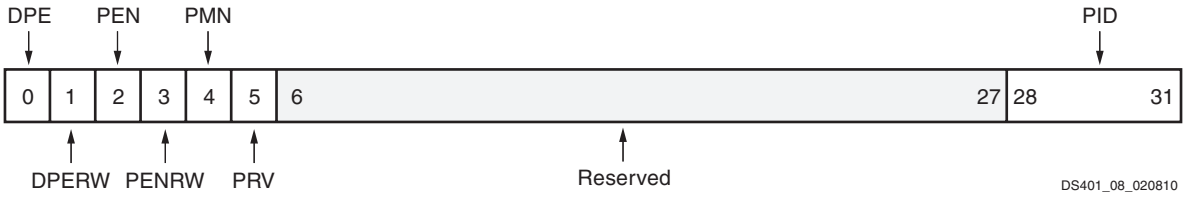

Figure 8: **OPB Arbiter Control Register**

Table 5 shows the Control Register bit definitions. The Control Register definition deviates from the IBM OPB Arbiter specification. This deviation is necessary because the Park Master ID (PID) field varies in width based on the number of masters supported in the design.

This field has been shifted so that it is LSB aligned and can be read easily by the software as an integer value. Also, the DPERW and PENRW bits have been added to provide information to the processor about the parameters chosen for the design and the accessibility of the DPE and PEN bits.

Because the processor requires more than one OPB bus cycle to update the master's priority levels, the bit Priority Registers Valid (PRV), has been added to indicate that the Priority Registers are being modified and are not valid.

Table 5: **OPB Arbiter Control Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0	DPE	Read(1) Read/Write(2)	0<SP Superscript>[1] 1<SP Superscript>[2]	**Dynamic Priority Enable.** Enables dynamic priority arbitration algorithm and update of the Priority Register. 0 - dynamic priority arbitration disabled 1 - dynamic priority arbitration enabled
1	DPERW	Read	0<SP Superscript>[1] 1<SP Superscript>[2]	**Dynamic Priority Enable Bit Read/Write.** This bit informs the software as to the access of the DPE bit. If the OPB Arbiter is parameterized to only support fixed priority arbitration, the DPE bit is always set to '0' to reflect that dynamic priority arbitration is not available. 0 - DPE bit is read only 1 - DPE bit is read/write
2	PEN	Read/Write	0<SP Superscript>[3] 1<SP Superscript>[4]	**Park Enable.** Enables parking on a master when no other masters have requests asserted. 0 - parking disabled 1 - parking enabled
3	PENRW	Read	0<SP Superscript>[3] 1<SP Superscript>[4]	**Park Enable Bit Read/Write.** This bit informs the software as to the access of the PEN bit. If the OPB Arbiter is parameterized to not support bus parking, the PEN bit is always set to '0' to reflect that bus parking is not available. 0 - PEN bit is read only 1 - PEN bit is read/write

Table 5: **OPB Arbiter Control Register Bit Definitions *(Cont'd)***

Bit(s)	Name	Core Access	Reset Value	Description
4	PMN	Read/Write	0	**Park On Master Not Last.** When parking is enabled, this bit determines if the arbiter parks on the master who was last granted the bus or on the master specified by the Parked Master ID bits. 0 - park on the master who had just been granted the bus 1 - park on the master specified by the Parked Master ID bits
5	PRV	Read/Write	1	**Priority Registers Valid.** This bit indicates that the Priority Registers all contain unique master IDs. This bit is negated by the processor before the processor modifies the Priority Registers and is asserted by the processor when the modifications are complete. Whenever this bit is negated, the OPB Arbiter uses the masters' IDs as their priority levels to perform bus arbitration.
6:C_OPB_DWIDTH - log_2(C_NUM_MASTERS) - 1	Reserved			
C_OPB_DWIDTH - log_2(C_NUM_MASTERS): C_OPB_DWIDTH - 1	PID	Read/Write	0000<SP Superscript >[5]	**Parked Master ID.** These bits contain the ID of the master to park on if parking is enabled and the Park On Master Not Last bit is set.

Notes:

1. OPB Arbiter parameterized to support fixed priority arbitration
2. OPB Arbiter parameterized to support dynamic priority arbitration
3. OPB Arbiter parameterized to not support bus parking
4. OPB Arbiter parameterized to support bus parking
5. The number of bits required by the PID field will vary with the number of masters supported by the OPB Arbiter. A field width of 4 is shown here for the default value.

The OPB Arbiter Control Register can be accessed from the OPB bus only if the OPB Arbiter is parameterized to support a processor interface (C_PROC_INTRFCE=1).

If the OPB Arbiter has been parameterized to only support fixed priority arbitration, the DPE bit is set to '0', indicating that the DPE bit is read-only by the processor core. The DPERW bit is then set to '0'.

If the OPB Arbiter has been parameterized to support dynamic priority arbitration, the DPE bit can be read from and written to by the processor. The DPERW bit is set to '1' and reflects the fact that the priority mode of the arbiter can be controlled by the DPE bit.

Although the OPB Arbiter has been parameterized to support dynamic priority arbitration, the OPB Arbiter can be put in a fixed priority arbitration mode by negating the DPE bit. As a result, both priority modes are available and supported.

If the OPB Arbiter has been parameterized to not support bus parking, the PEN bit is set to '0'. The PENRW bit is then set to '0' and reflects the fact that the PEN bit is read-only by the processor core.

If the OPB Arbiter has been parameterized to support bus parking, the PEN bit can be read from and written to by the processor. The PENRW bit is set to '1' and reflects the fact that the bus parking mode of the arbiter can be controlled by the PEN, PMN, and PID bits of the Control Register.

Upon reset, the parking mode is set to park on the last master which is the default value in the Control Register of the PMN bit. To park on the master whose ID is contained in the Control Register, the OPB Arbiter must be parameterized to support a processor interface so that the Control Register can be written with the appropriate value.

Because the processor requires multiple bus cycles to update the masters' priority levels in the Priority Registers, there is a period of time in which the Priority Registers do not contain unique master IDs. Though the arbiter functions properly in this circumstance, a particular master does not have a priority level associated with it and consequently never receives a grant if it issues a request. This could cause the OPB to stop. Consequently, whenever the processor begins to modify the priority levels of the masters, it first negates the PRV bit in the Control Register, indicating to the arbitration logic that the Priority Register values should not be used in bus arbitration. In this case, the arbitration logic uses the masters' IDs as their priority level until the PRV bit has been asserted.

OPB Arbiter Priority Registers

Each Priority Register holds the master ID of the OPB master at that priority level as shown in Table 6. Each master's relative priority is determined by its ID's location within these registers.

The LVL0 Priority Register holds the ID of the master with highest priority (level 0), the LVL1 Priority Register holds the ID of the master with the next highest priority (level 1), and so on. The LVL[C_NUM_MASTERS-1] Priority Register holds the ID of the master with lowest priority.

These registers are reset to values which assign level 0 (highest) priority to Master 0, level 1 (next highest) priority to Master 1, level 2 priority to Master 2, etc.

The number of bits required by the master ID within each Priority Register varies based on the number of masters support by the OPB Arbiter. The master ID is always aligned to the LSB position within the register so that the master ID appears as an integer when accessed by software.

The IBM OPB Arbiter refers to the priority levels as High, Medium High, Medium Low, and Low because it only implemented arbitration among 4 masters. Because the Xilinx implementation of the OPB arbiter supports parameterization of the number of OPB masters in the system, numbers are used to represent priority levels instead of text descriptors. Level 0 always remains the highest priority level regardless of the number of masters implemented. The higher the level number, the lower the priority.

Table 6: **OPB Arbiter LVLn Priority Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0 : C_OPB_DWIDTH - log2(C_NUM_MASTERS) -1	Reserved			
C_OPB_DWIDTH - log2(C_NUM_MASTERS) : C_OPB_DWIDTH-1	LnPM	Read/Write	mmmm<SP Superscript>[1]	**Level n Priority Master ID.** This field contains the ID of the master at level n priority. <SP Superscript>[2]

Notes:

1. "mmmm" represents the bit encoding of the Master ID at this priority level
2. n = 0 - C_NUM_MASTERS - 1

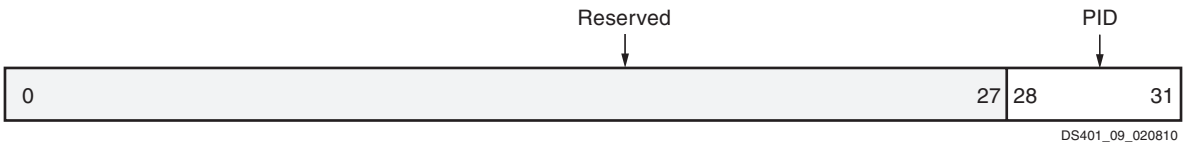

Figure 9: **OPB Arbiter OPB Arbiter LVLn Priority Register**

Each master ID must be contained in one of the Priority Registers, otherwise that master's request is ignored by the arbiter (because it has no priority value) and a grant to that master is never asserted.

This could cause the bus to stall because there is no mechanism in the OPB specification for a master to timeout while waiting for a grant. Consequently, each Priority Register must contain a unique master ID and all master IDs must be contained in one of the Priority Registers.

Because the processor can not update the Priority Registers in a single OPB transaction, there may be several clock cycles in which a particular master ID is not contained within a Priority Register.

Consequently, the Priority Registers Valid (PRV) bit in the Control Register is used to indicate that the values of the Priority Registers are being modified and should not be used in OPB arbitration.

The processor negates this bit before modifying the Priority Registers and then asserts this bit when the modification is complete. The processor insures that whenever the PRV bit is asserted, all master IDs have been assigned a priority.

When the PRV bit is negated, the OPB arbiter assigns priority based on the master ID, so that Master 0 has level 0 priority (highest), Master 1 has level 1 priority, and Master n has level n priority. Once the PRV bit has been asserted, the values in the Priority Registers will again be used to determine OPB ownership.

The Priority Register can be accessed from the OPB for read and write operations. Note that regardless of the priority mode selected for the OPB Arbiter (even if the OPB Arbiter has been parameterized to fixed priority arbitration), the processor core can set the desired priority levels of the OPB masters by writing to these registers.

OPB Arbiter Block Diagram

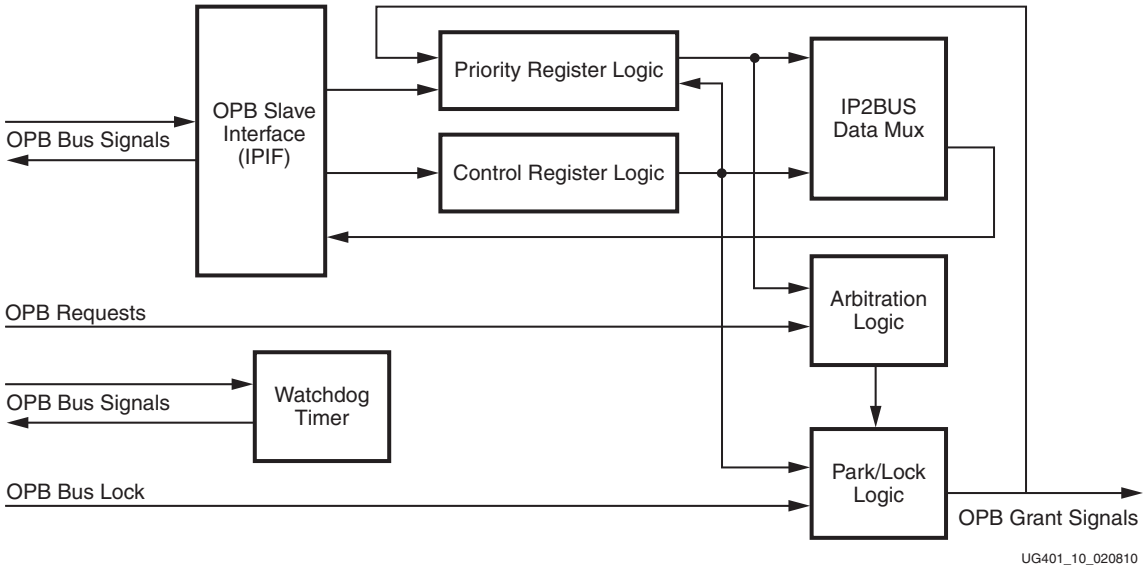

Figure 10: **OPB Arbiter Top-level Block Diagram**

The top-level block diagram for the OPB Arbiter is shown in Figure 10. The IPIF block is the OPB bus interface block that handles the OPB bus protocol for reading and writing the Priority Registers and the Control Register within the OPB Arbiter.

The ARB2BUS data mux contains the data multiplexor required to output data to the OPB bus during a read cycle. The Arbitration Logic block determines which incoming request has the highest priority and the Park /Lock Logic block determines which master should be granted the bus based on this priority as well as whether the bus is locked or if bus parking is enabled.

The Watchdog Timer asserts the OPB timeout signal if a slave response (OPB_xferAck, OPB_retry, or OPB_toutSup) has not been received within 16 clock cycles of the master taking control of the bus.

When C_NUM_MASTERS=1, the only logic in the OPB Arbiter is the Watchdog Timer. The OPB Grant signal is set to VCC and all OPB Bus output signals are set to GND.

The following sections describe each module in the block diagram.

OPB Slave Interface (IPIF)

The IPIF block implements a slave interface to the OPB and is only present in the design if C_PROC_INTRFCE=1 and C_NUM_MASTERS>1.

Its address in the OPB memory map is determined by setting the parameter C_BASEADDR. All registers are addressed by an offset to C_BASEADDR as shown in Table 4.

The IPIF block outputs a register write clock enable and a register read clock enable for the register which was addressed depending on the type of data transfer specified by the master. When the data transfer is complete, the IPIF block generates the transfer acknowledge.

Control Register Logic

The Control Register Logic block simply contains the OPB Arbiter Control Register described in section "OPB Arbiter Control Register" and is only present in the design if C_NUM_MASTERS > 1.

Priority Register Logic

The Priority Register logic contains the Priority Registers and the logic to update the priority of the OPB masters. Descriptions of the OPB Arbiter Priority Registers are found in the "Parameter - Port Dependencies" section. The Priority Registers are only present in the design if C_NUM_MASTERS>1.

Priority Register Update Logic

Fixed Priority Parameterization (C_DYNAM_PRIORITY=0)

When the OPB Arbiter is parameterized to support only fixed arbitration, the dynamic priority enable bit in the Control Register is permanently disabled. The Priority Registers are loaded at reset with the value of the Master ID which matches the priority level of the register.

For example, the LVL0 Priority Register is loaded with '0' to represent the ID of Master 0. Likewise, the LVLn Priority Register is loaded with the bit encoding of n to represent the ID of Master n as shown in Table 6.

The Priority Registers can be loaded with different Master IDs by writing to the Priority Registers (if C_PROC_INTRFCE=1), consequently, the priorities of the OPB Masters can be changed as desired by software.

The Priority Registers Valid (PRV) bit in the Control Register is negated whenever the processor modifies the Priority Registers and is asserted whenever the modification is complete.

The ID of the masters are used to determine OPB ownership whenever the PRV bit is negated. The relative priorities of the OPB Masters are then determined by the connection of master devices to the request/grant signals. The values of the Priority Registers are used in OPB arbitration whenever the PRV bit is asserted.

Fixed priority arbitration for a 4 OPB Master system with combinational grant outputs is shown in Figure 11. Figure 12 shows the same system when configured with registered grant outputs with the master requests separated by an additional clock.

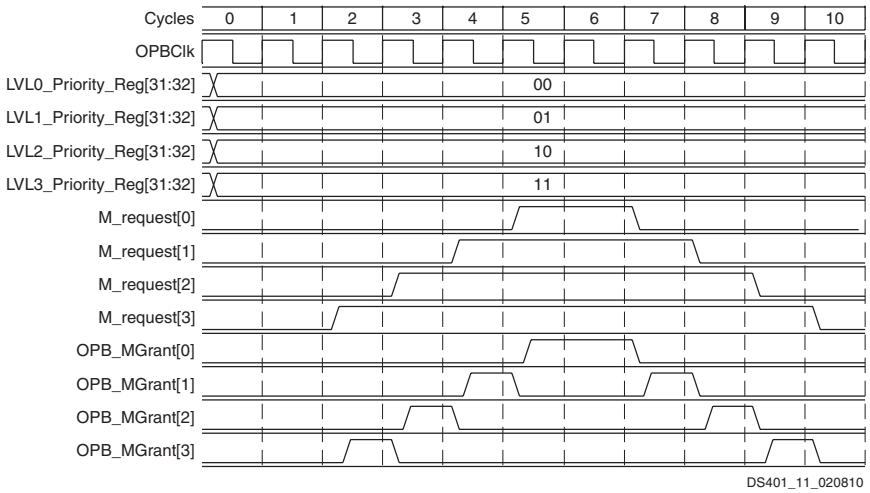

Figure 11: **Fixed Priority Arbitration, Combination Grant Outputs for 4 OPB Masters**

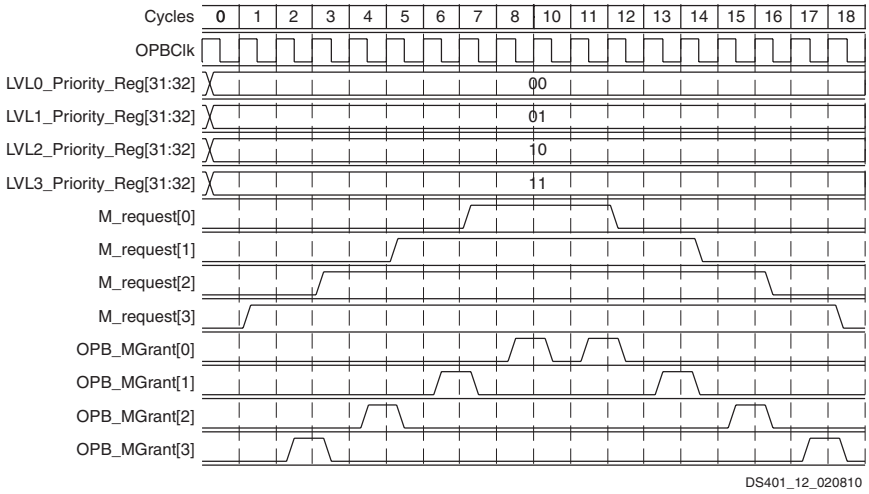

Figure 12: **Fixed Priority Arbitration, Registered Grant Outputs for 4 OPB Masters**

Dynamic Priority Parameterization (C_DYNAM_PRIORITY=1)

When the OPB Arbiter is parameterized to support dynamic priority arbitration, dynamic priority arbitration mode is enabled at reset or at any other time by writing a "1" to the DPE bit of the Control Register.

Disabling dynamic priority arbitration mode by setting the DPE bit to "0" puts the OPB Arbiter into a fixed priority arbitration mode. This effectively freezes the values of the Priority registers (unless updated by an OPB write to the registers) and thus its ordering of arbitration priorities among the attached master devices.

Setting the master priorities by software and not allowing them to update results in a static assignment of priority among the OPB masters.

Upon reset, the Priority registers contains the reset values as described in Table 6 and the dynamic priority bit is enabled. When dynamic priority arbitration mode is enabled, the contents of the Priority Registers are reordered after every request-grant cycle, moving the ID of the most recently granted master to the lowest Priority register and moving all other master IDs up one level of priority.

The dynamic priority arbitration mode operation results in an implementation of the least recently used (LRU) algorithm. The lowest priority master ID will be the one which was granted the bus most recently, and the highest priority master ID will be the one which was granted the bus the least in the recent past.

The values to be loaded into the Priority registers are either the values written to the register from the OPB or a shift of the master ID from the next lowest Priority register. In the case of the low Priority register, the ID of the master last granted the bus is loaded into this register.

The master ID of the master granted the bus is loaded into the lowest Priority register and the IDs in all other Priority registers up to the priority of the one just granted move up one position in priority. The Priority registers above the one just granted hold their master ID values.

A pipeline register exists between the arbitration logic and Priority Register update logic which delays the master grant signals by an additional clock. Consequently, if the OPB Arbiter is configured for dynamic priority arbitration and registered grant outputs, the master grant signals will be output 2 clocks after a valid arbitration cycle.

Dynamic priority arbitration for a 4 OPB master system with combinational grant outputs is shown in Figure 13. Figure 14 shows dynamic priority arbitration for a 4 master OPB system with registered grant outputs.

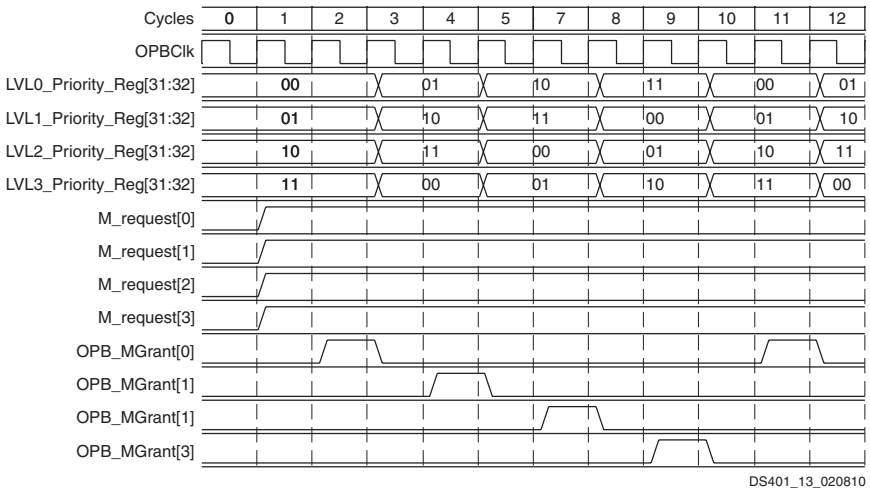

Figure 13: **Dynamic Priority Arbitration, Combinational Grant Outputs- 4 OPB Masters**

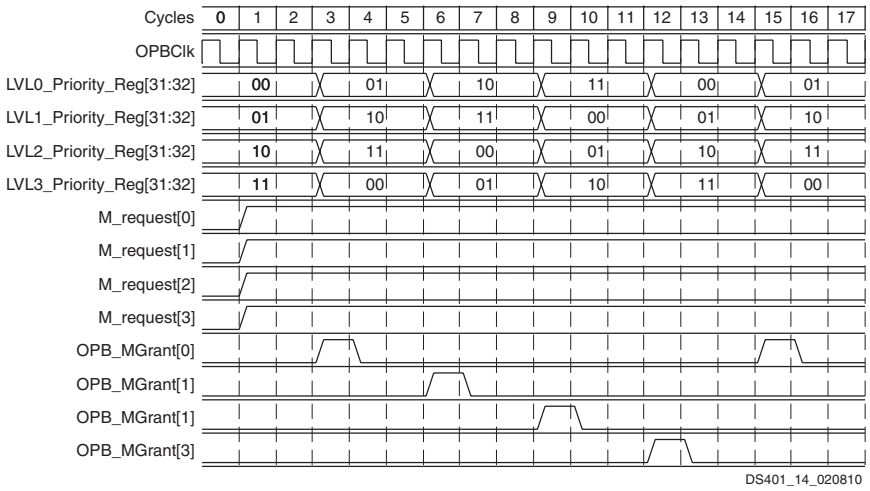

Figure 14: **Dynamic Priority Arbitration, Registered Grant Outputs- 4 OPB Masters**

Block Diagram

The block diagram for the Priority registers and the register update logic is shown in Figure 15. The gray shaded blocks represent the Priority Register update logic which is only present when the OPB Arbiter is parameterized to support Dynamic Priority arbitration.

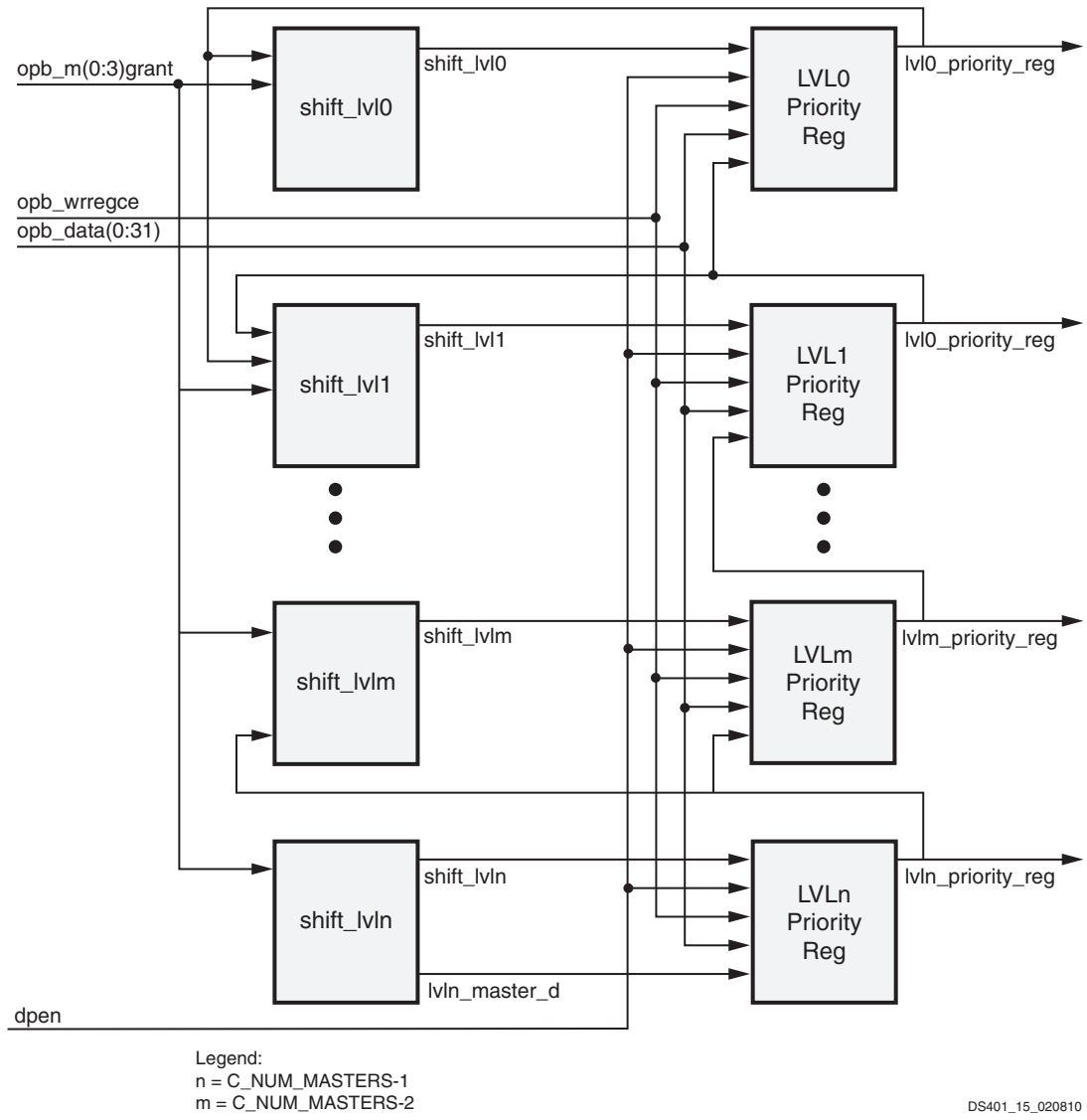

Figure 15: **Priority Register Logic**

When the OPB Arbiter has been parameterized to support a processor interface (C_PROC_INTRFCE=1), the Priority registers can still be loaded by the processor, allowing the processor to change the priorities of the OPB Masters.

Also, the arbiter can be set to operate in a fixed priority mode by the processor writing to the Control Register and negating the DPE bit. However, the pipeline registers between the arbitration logic and the register update logic are still present, thereby delaying the grant outputs by an additional clock cycle.

ARB2BUS Data Mux

When a read of the OPB Arbiter Priority Registers or the OPB Arbiter Control Register is requested on the OPB, the ARB2BUS Data Mux outputs the requested data to the IPIF which sends this data to the OPB with the required protocol. This logic is only present in the design if C_PROC_INTRFCE=1 and C_NUM_MASTERS>1.

Arbitration Logic

Figure 16 depicts the functional block diagram of arbitration logic for the OPB arbiter. This logic is only present in the design if C_NUM_MASTERS>1. The pipeline registers are only present in the arbitration logic if C_DYNAM_PRIORITY = 1.

All master request signals are input to the Prioritize Request block which consists of multiplexers which prioritize the master's requests into the signals lvl0_req, lvl1_req, lvlm_req, and lvln_req based on the requesting masters' priorities if PRV =1 or the master IDs if PRV = 0. (n = C_NUM_MASTERS-1, m= C_NUM_MASTERS-2)

The prioritized request signals are then input into the priority encoder which determines which priority grant is asserted, i.e., grant_lvl0, grant_lvl1, grant_lvlm, and grant_lvln.

The prioritized grant signals are then input to the Assign Grants block to determine which master's grant signal is asserted based on the priority of that master, again determined by examination of the master IDs in the Priority Registers if PRV = 1, or the master IDs if PRV = 0.

The master's priority code selects the appropriate prioritized grant signal to be output to that master. These intermediate grant signals are then registered if the OPB Arbiter is configured to support dynamic priority arbitration to reduce the number of logic levels between the arbitration logic and the Priority Register update logic.

Because the Priority Register update logic is not present when the OPB Arbiter is not configured to support dynamic priority arbitration, these pipeline registers are not necessary and consequently are not present in the design.

The arbitration logic also contains the logic for detecting valid arbitration cycles which is input to the Park/Lock logic. Valid arbitration cycles are defined as when either the OPB_select signal is de-asserted indicating no data transfer is in progress or when OPB_XferAck is asserted, indicating the final cycle in a data transfer and OPB_busLock is not asserted.

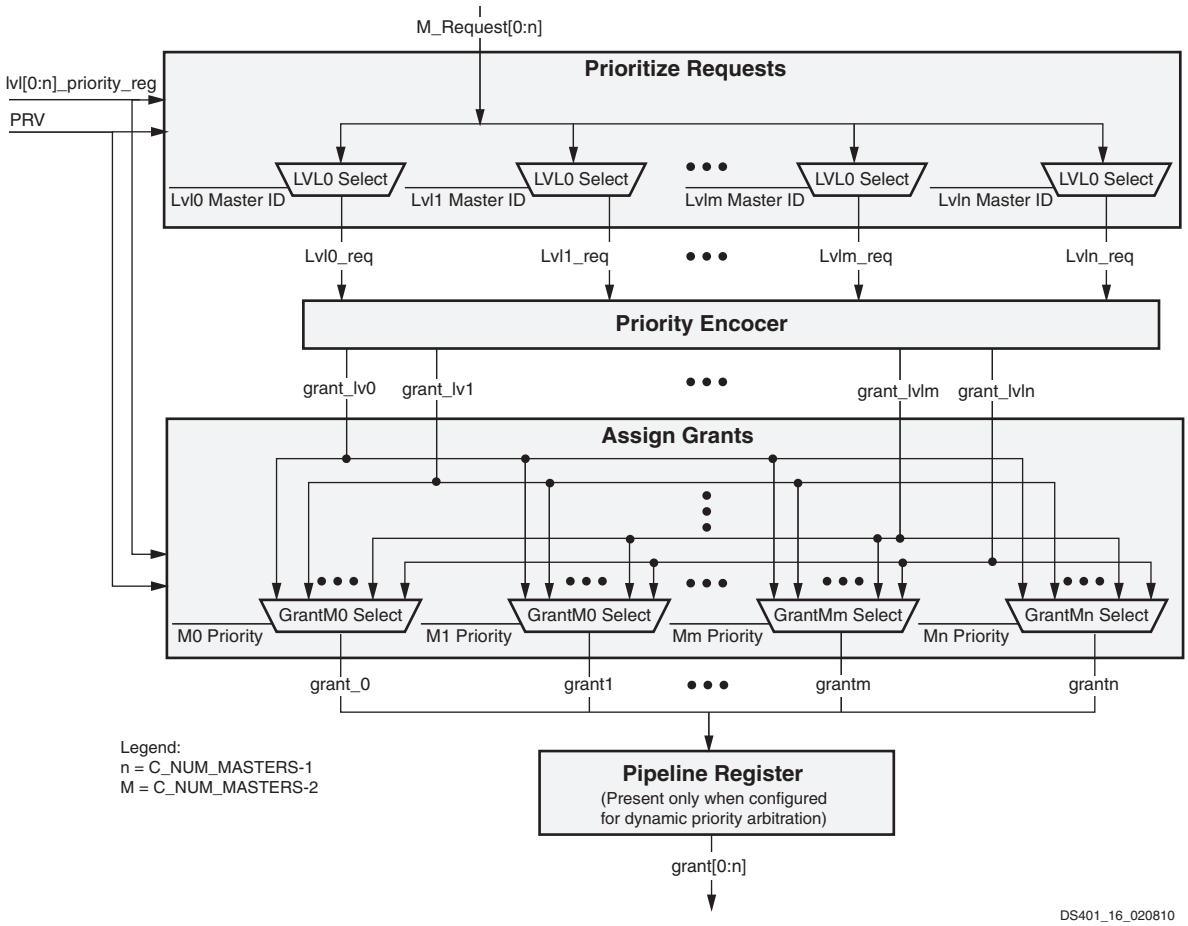

Figure 16: **Arbitration Logic**

Park/Lock Logic

The Park/Lock logic processes the raw or registered grant outputs from the Arbitration Logic and is only present in the design if C_NUM_MASTERS>1.

It provides for the parking (if C_PARK=1) and locking functionality of the OPB Arbiter and generates the final grant signals which are sent to the OPB master devices during valid arbitration cycles as shown in Figure 17. The OPB Arbiter can be parameterized to register the master grant signals by setting the parameter C_REG_GRANTS to 1.

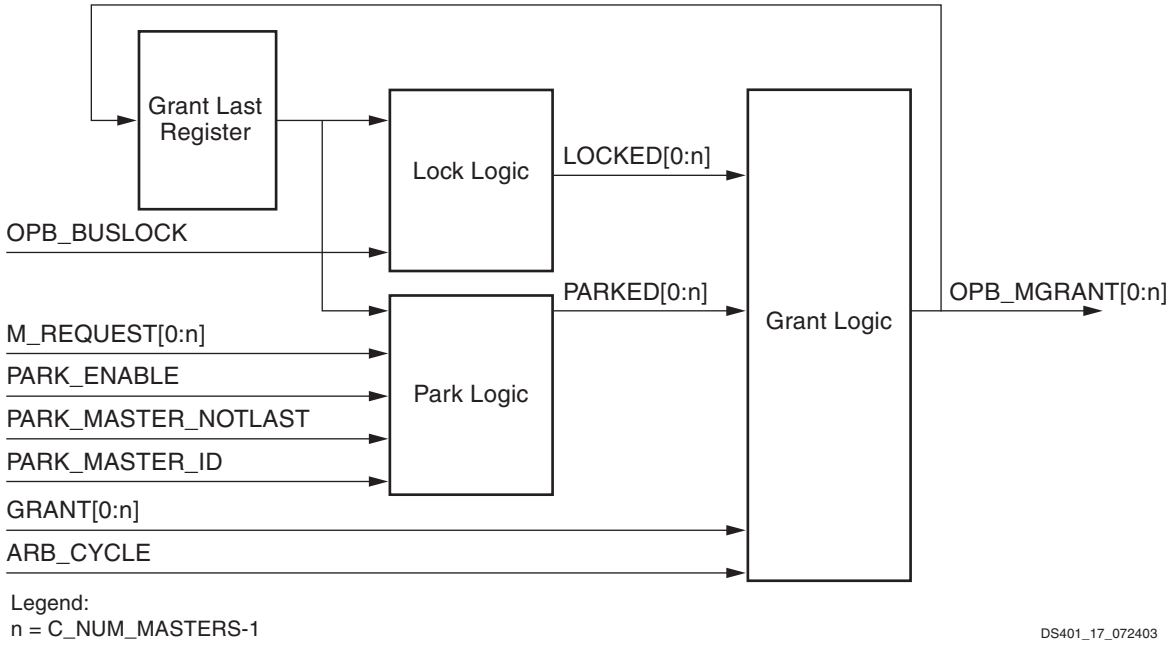

Figure 17: **Park/Lock Logic**

Grant Last Register

The Grant Last Register holds the state of the grant outputs from the last request/grant arbitration cycle. This information is used to determine which master currently has control of the bus to implement bus locking and park on last master parking.

Lock Logic

If an OPB master asserts the bus lock signal upon assuming control of the bus, the OPB arbiter will continue to grant the OPB to the master which locked the bus. Bus lock signals from all attached masters are OR'ed together to form OPB_busLock, which is an input to the OPB Arbiter.

When OPB_busLock is asserted, bus arbitration is locked to the last granted master (as indicated by the Grant Last register). All other master Grant outputs are gated off and will not be asserted, regardless of the state of the Request inputs or the programmed priorities.

When the OPB bus is locked, bus request and grant signals have no effect on bus arbitration. The OPB master may proceed with data transfer cycles while asserting bus lock without engaging in bus arbitration and without regard to the state of the request and grant signals.

Grant signals will be generated if the master asserts its request signal. The locked master's grant signal will be asserted in response to its request signal during valid arbitration cycles. However, the locked master need not assert its request or receive an asserted grant signal to control the bus.

The master owns the bus by virtue of asserting its bus lock signal after being granted the bus and before another valid arbitration cycle.The master which asserted bus lock will retain control of the bus until bus lock is de-asserted for at least one complete cycle.

The OPB arbiter will detect the bus lock signal and will continue to grant the bus to the current master, regardless of other (higher priority) requests. Figure 18 shows the OPB bus lock operation when the OPB Arbiter is configured for fixed priority arbitration and combinational grant outputs. Figure 19 shows the OPB bus lock operation when the OPB Arbiter is configured for fixed priority arbitration and registered grant outputs.

Note that the bus grant signal is asserted one clock later.

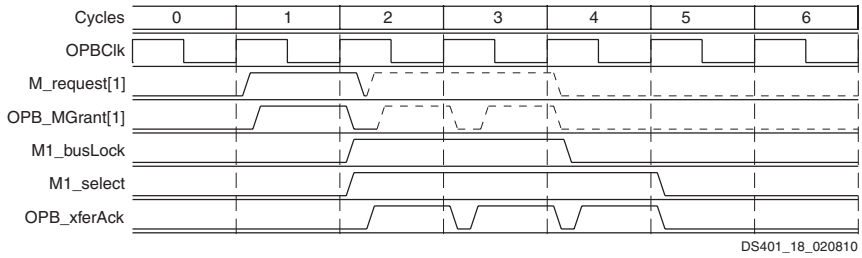

Figure 18: **Bus Locking - Fixed Priority, Combinational Grant Outputs**

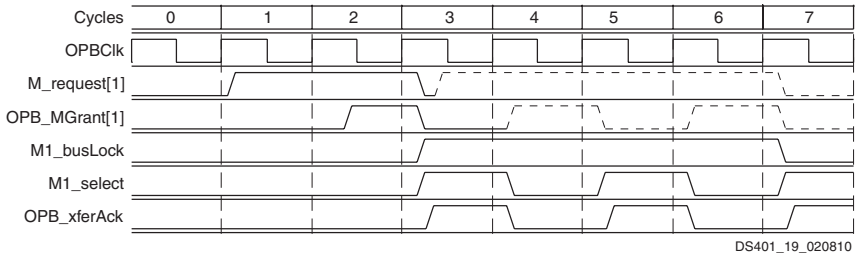

Figure 19: **Bus Locking - Fixed Priority, Registered Grant Outputs**

Park Logic

When C_PARK=1, the OPB Arbiter supports bus parking. Bus parking is when a master's grant signal is asserted during valid arbitration cycles when no other master devices are requesting. This reduces latency for the parked master eliminating the need for a request/grant cycle when initiating a new OPB transfer.

Asserting a grant signal for parking is considered an arbitration, because it determines which device controls the bus. If dynamic priority mode is enabled, the ID of the parked master will be shifted to the lowest priority slot of the Priority Register.

The master will remain parked (its grant signal asserted) so long as no other master asserts a request signal. If the parked master and another master assert request at the same time, the parked master will control the bus because the bus was parked on this master even though this master is at a lowest priority.

Figure 20 illustrates this behavior when the OPB Arbiter is configured to support fixed priority arbitration and combinational grant outputs.

Figure 21 illustrates this behavior when the OPB Arbiter is configured to support dynamic priority arbitration when the OPB Arbiter is configured to support dynamic priority arbitration and registered grant outputs.

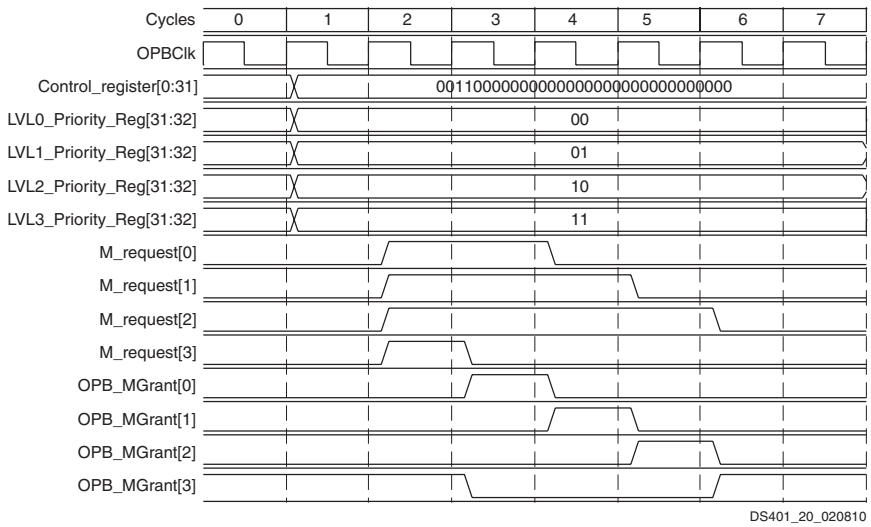

Figure 20: **Bus Parking - Fixed Priority Arbitration, Combinational Grant Outputs**

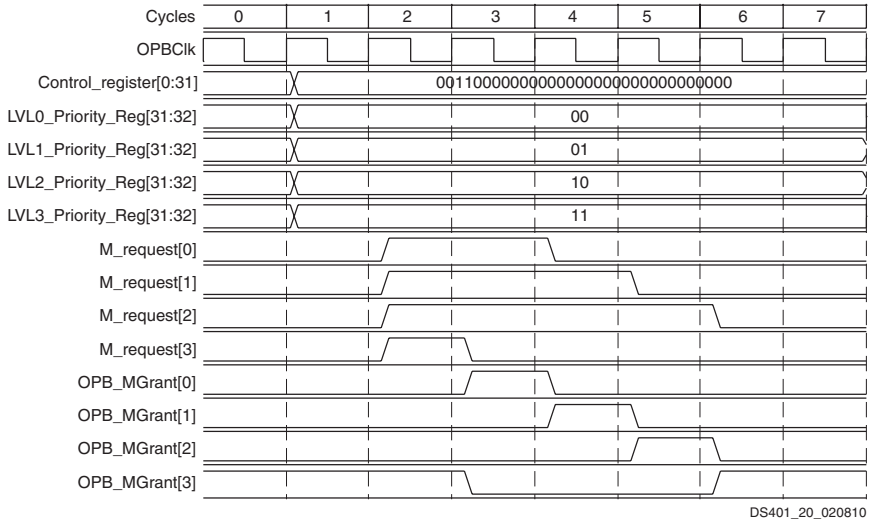

Figure 21: **Bus Parking - Fixed Priority Arbitration, Combinational Grant Outputs**

When the grant outputs are registered, the parked master's grant will assert once, negate, and then stay asserted. This allows the internal arbitration state machine a clock cycle to check if OPB_select is asserted before parking. This case occurs when a request from a master is aborted, but the grant is in the internal pipeline.

When C_PARK = 1, bus parking is enabled or disabled by the value of the Park Enable control bit in the OPB Arbiter Control Register (see Table 5). The bus can either be parked on the master who was last granted the bus, or on a specified master as indicated by the Park Master ID bits in the OPB Arbiter Control Register. If bus parking is not desired, the park logic can be eliminated by setting C_PARK=0.

Park on Master Not Last

When bus parking is enabled (bit PEN = 1 in the OPB Arbiter Control Register and C_PARK=1) and Park on Master Not Last is selected (bit PMNL = 1 in the OPB Arbiter Control Register), the bus will be parked on the master whose ID is contained in the Park Master ID (PMID) field of the OPB Arbiter Control Register.

This master's grant signal will be asserted during valid arbitration cycles when no other master's request signal is asserted. Figure 22 shows bus parking on the master specified in the Control Register for a 4 OPB master system when the OPB Arbiter is parameterized for fixed priority arbitration and combinational grant outputs.

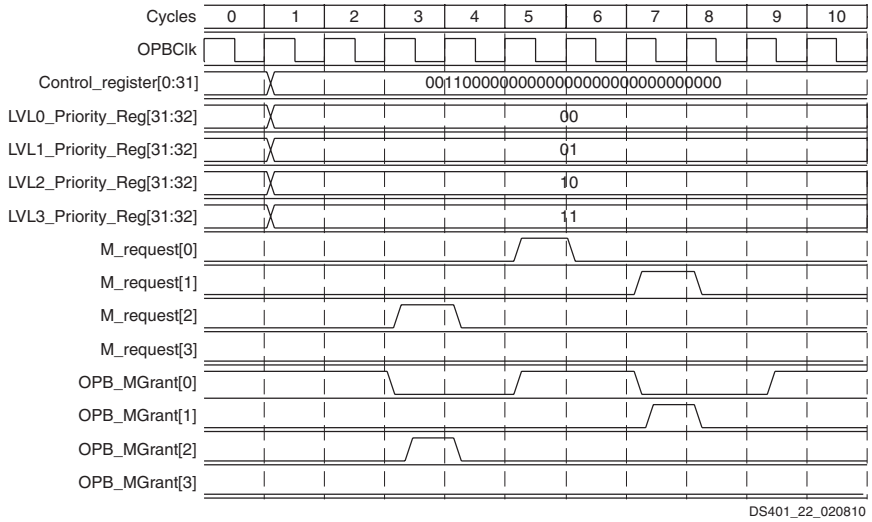

Figure 22: **Bus Parking on Master Not Last - Fixed Priority Arbitration, Combinational Grant Outputs**

Park on Last Master

When bus parking is enabled (bit PEN = 1 in the OPB Arbiter Control Register and C_PARK=1) and Park on Master Not Last is negated (bit PMNL = 0 in the OPB Arbiter Control Register), the bus will be parked on the master which was most recently granted the bus as indicated by the Grant Last register.

This master's grant signal will be asserted during valid arbitration cycles when no other master's request signal is asserted. Figure 23 shows bus parking on the last master with the OPB Arbiter parameterized for fixed priority arbitration and combinational grant outputs for a 4 OPB Master system.

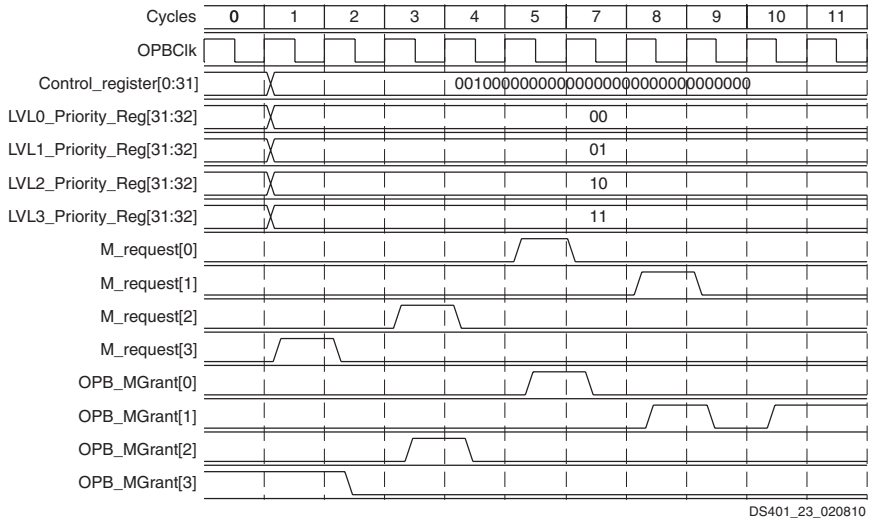

Figure 23: **Bus Parking on Last Master - Fixed Priority Arbitration, Combinational Grant Outputs**

Grant Logic

The Grant Logic block determines the final grant signals to the OPB Masters based on the intermediate grant signals from the arbitration logic and the results of the park and lock logic.

The OPB Arbiter can be parameterized so that the grant signals are either registered outputs are combinational outputs. Registering the grant signals allows for higher OPB clock frequencies at the cost of a 1-cycle arbitration latency.

Combinational grant outputs allow the grant signals to be asserted within the same clock cycle as the Master request signals, however, the overall clock frequency of the OPB will be affected. Basic OPB arbitration using registered grant outputs is shown in Figure 3.

Watchdog Timer

The watchdog timer generates the OPB_timeout signal within the 16th cycle following the assertion of OPB_select if there is no response from a slave (OPB_xferAck or OPB_retry) and if toutSup is not asserted by an addressed slave device to suppress the timeout. A slave must assert Sl_xferAck or Sl_retry on or before the rising edge of the 16th clock cycle after the assertion of OPB_select if toutSup is not asserted. This logic is always present in the OPB Arbiter design.

Upon assertion of OPB_timeout, the master device which initiated the transfer cycle must terminate the transfer by de-asserting Mn_select in the cycle following the assertion of OPB_timeout as shown in Figure 24.

If OPB_busLock is not asserted, the OPB Arbiter will perform a bus arbitration in the cycle in which OPB_select is de-asserted. If OPB_busLock is asserted, the requesting master retains control of the OPB, but must still de-assert Mn_select following the assertion of OPB_timeout for at least one cycle.

If OPB_xferAck or OPB_retry are asserted in the 16th cycle following the assertion of Mn_select coincident to the assertion of OPB_timeout, the master device should ignore OPB_timeout, and respond to the slave's OPB_xferAck or OPB_retry signal.

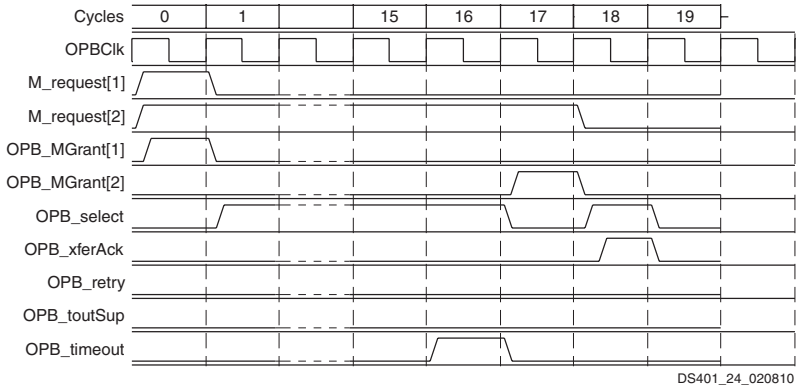

Figure 24: **OPB Timeout Error**

To prevent a bus timeout, an OPB slave must assert OPB_toutSup (OR of all slave's Sln_ToutSup) on or before the rising edge of the 15th clock cycle after the assertion of OPB_select. OPB_toutSup will be used by the OPB Arbiter to suppress the assertion of OPB_timeout and to suspend the timeout counter. When OPB_toutSup is asserted, the timeout counter holds its current value. When OPB_toutSup is negated, the timeout counter resumes counting. OPB timeout error suppression is shown in Figure 25.

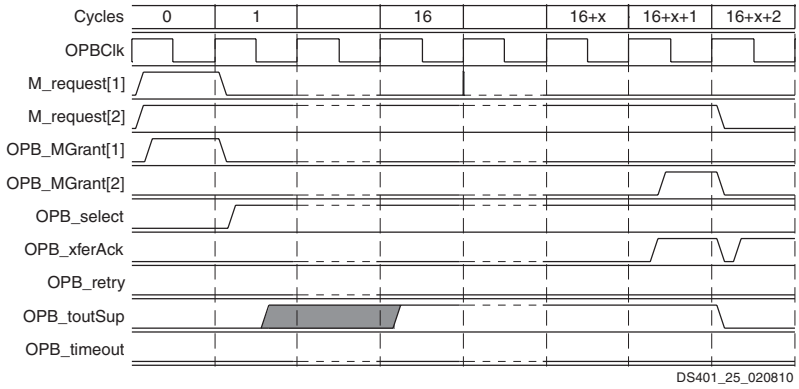

Figure 25: **OPB Timeout Error Suppression**

Design Implementation

Target Technology

The intended target technology is a Virtex-II FPGA.

Device Utilization and Performance Benchmarks

Because the OPB V20 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 OPB V20 is combined with other pieces of the FPGA design, the utilization of FPGA resources and timing of the OPB V20 design will vary from the results reported here.

In order to analyze the OPB V20 timing within the FPGA, a design was created that instantiated the OPB V20 with registers on all of the inputs and outputs. This allowed a constraint to be placed on the clock net to yield more realistic timing results. The f_{MAX} parameter shown in Table 7 was calculated with registers on the OPB V20 inputs and outputs. However, the resource utilizations reported in Table 7 do not include the registers on the OPB V20 inputs and outputs.

The OPB V20 benchmarks are shown in Table 7 for a Virtex-II Pro -6 FPGA.

Table 7: **OPB V20 FPGA Performance and Resource Utilization Benchmarks (Virtex-II Pro -6)**

Parameter Values							Device Resources			f_{MAX} (MHz)
C_NUM_MASTERS	C_NUM_SLAVES	C_USE_LUT_OR	C_DYNAM_PRIORITY	C_PARK	C_PROC_INTRFCE	C_REG_GRANTS	Slices	Slice Flip-Flops	4-input LUTs	f_{MAX}
1	2	1	N/A	N/A	N/A	N/A	46	5	80	181
1	4	1	N/A	N/A	N/A	N/A	47	5	80	127
1	8	1	N/A	N/A	N/A	N/A	119	5	160	200
1	16	1	N/A	N/A	N/A	N/A	167	5	240	234

Table 7: **OPB V20 FPGA Performance and Resource Utilization Benchmarks (Virtex-II Pro -6)**

2	2	0	0	0	0	0	127	7	165	222
2	4	0	0	0	0	0	128	7	164	130
2	8	0	0	0	0	0	127	7	203	229
2	16	0	0	0	0	0	167	7	283	243
2	4	0	1	1	1	1	173	51	230	152
4	2	1	0	0	0	0	136	9	176	204
4	4	1	0	0	0	0	137	9	176	156
4	8	1	0	0	0	0	209	9	256	216
4	4	1	1	0	0	0	161	22	206	150
4	4	1	0	1	0	0	145	13	185	198
4	4	1	0	0	1	0	184	40	250	124
4	4	1	0	0	0	1	140	14	178	217
4	4	1	1	1	1	1	230	74	307	109
8	2	0	0	0	0	0	169	13	274	215
8	4	0	0	0	0	0	170	13	274	163
8	4	0	1	1	1	1	410	121	597	103

Notes:

1. These benchmark designs contain only the OPB V20 with registered inputs/outputs without any additional logic. Benchmark numbers approach the performance ceiling rather than representing performance under typical user conditions.
2. 32-bit OPB data widths and 32-bit OPB address widths are verified as part of the IPIF verification.
3. Device resource numbers do not include the registers for the OPB V20 I/O.
4. Max frequency calculated with registers on the OPB V20 I/O.

Specification Exceptions

Register Definitions and Addressing

To support parameterization of the number of masters, the Xilinx OPB Arbiter uses a separate Priority Register for each priority level. Because the number of Priority Registers can vary, the OPB Arbiter Control Register is placed at the base address of the OPB Arbiter so that its location does not vary.

Because the number of bits required for the master IDs will vary with the number of masters, the fields in the Control Register and the Priority Registers that contain master IDs are LSB aligned. The bit ordering of these registers is then different than that specified in the IBM OPB Arbiter specification.

The Control Register also contains additional bits (DPERW, PENRW, PRV) not in the IBM OPB Arbiter specification. The DPERW bit indicates whether the DPE bit can be modified, the PENRW bit indicates whether the PEN bit can be modified, and the PRV bit indicates whether the Priority Registers contain valid data, i.e, all master IDs are contained in a Priority Register.

I/O Signals

The signal ARB_DbusEn is no longer an I/O signal. The gating of the OPB Arbiter's data bus with the enable signal is done internally within the IPIF Module.

The master request signals and master grant 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 supported by the Xilinx OPB Arbiter. Table 8 summarizes the I/O signal name modifications and variations.

Table 8: Xilinx OPB Arbiter I/O Signal Variations

IBM OPB Arbiter Signal Name	Xilinx OPB Arbiter Signal Name
ARB_DBusEn	no longer an I/O signal
ARB_XferAck	Sl_XferAck
M0_request, M1_request, M2_request, M3_request	M_request[0:C_NUM_MASTERS-1]
OPB_M0Grant, OPB_M1Grant, OPB_M2Grant, OPB_M3Grant	OPB_MGrant[0:C_NUM_MASTERS-1]

Priority Level Nomenclature

The IBM OPB Arbiter refers to the priority levels as High, Medium High, Medium Low, and Low because it only implemented arbitration among 4 OPB masters. Because the Xilinx implementation of the OPB arbiter supports parameterization of the number of OPB masters in the system, it was decided that numbers should be used to represent priority levels instead of text descriptors.

Level 0 will always remain the highest priority level regardless of the number of masters implemented. The higher the level number, the lower the priority.

Grant Outputs

The IBM OPB Arbiter only outputs combinational bus grant signals for both fixed and dynamic priority arbitration. The Xilinx OPB Arbiter can be configured to output combinational or registered bus grants.

Also, when the Xilinx OPB Arbiter is configured to support dynamic priority arbitration, the bus grants will be output one clock after the arbitration cycle due to a pipeline register between the arbitration logic and the Priority Register update logic.

Bus Parking

When utilizing dynamic priority arbitration and the bus is parked on a particular bus master, this bus master is moved to lowest bus priority. In the IBM OPB Arbiter, if another master requests the bus at the same time as the parked bus master, the higher priority master will gain control of the bus. In the Xilinx OPB Arbiter, the parked bus master will gain control of the bus, even though this master is at a lower priority.

OPB Timeout

To obtain better timing in the FPGA implementation of the OPB, the OPB_timeout signal is registered. This means that all slaves must assert Sl_xferAck or Sl_retry on or before the rising edge of the 16th clock cycle after the assertion of OPB_select. If an OPB slave wishes to assert Sl_toutSup, Sl_toutSup must be asserted on or before the rising edge of the 15th clock after the assertion of OPB_select.

Clock and Power Management

The IBM OPB 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 OPB 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 OPB Arbiter, consequently the following I/O signal is not used:

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

Scan Test Chains

The IBM OPB Arbiter contains an internal scan chain for testing and verification purposes. Xilinx FPGAs support boundary scan testing but the internal flip-flops within the architecture do not provide for an internal scan chain. Consequently, the internal scan chain implemented in the IBM OPB Arbiter is not supported in the Xilinx implementation and the following I/O signals are not used:

- LSSD_AClk - FPGA implementation does not support scan
- LSSD_BClk - FPGA implementation does not support scan
- LSSD_CClk - FPGA implementation does not support scan
- LSSD_scanGate - FPGA implementation does not support scan
- LSSD_scanIn - FPGA implementation does not support scan
- LSSD_scanOut - FPGA implementation does not support scan

Reference Documents

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

1. IBM 64-Bit On-Chip Peripheral Bus Architectural Specification (v2.0)
2. IBM On-Chip Peripheral Bus Arbiter Core User's Manual (v1.5), 32-Bit Implementation

Revision History

Date	Version	Revision
11/15/04	1.0	Initial Xilinx release.
4/7/05	1.1	Updated for EDK 7.1.1 SP1; updated trademarks and supported device listing.
6/7/05	1.2	Incorporated CR209288 (recommends a max of 16 slaves on the opb).
12/2/09	1.3	Created v1.10d for EDK_L 11.4 release.
4/19/10	1.4	Updated images and tables; converted to current data sheet template.

Notice of Disclaimer