

Introduction

The On-Chip Peripheral Bus (OPB) Arbiter design described in this document incorporates the features contained in the IBM On-chip Peripheral Bus Arbiter Core manual (version 1.5) for 32-bit implementation, which 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 explained in the **Specification Exceptions** section of this document.

The Xilinx OPB Arbiter design allows the user to tailor the OPB Arbiter to suit a specific 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 Arbiter Design Parameters** section of this document.

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

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	QPro™-R Virtex™-II, QPro Virtex-II, Spartan™-II, Spartan-IIe, Spartan-3, Virtex, Virtex-II, Virtex-E, Virtex-II Pro, Virtex-4	
Version of Core	opb_arbiter	v1.02e
Resources Used		
	Min	Max
I/O	4	904
LUTs	6	252
FFs	4	1477
Block RAMs	0	0
Provided with Core		
Documentation	Product Specification	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	5.1i or later	
Verification	N/A	
Simulation	ModelSim SE/EE 5.6e or later	
Synthesis	XST	
Support		
Support provided by Xilinx, Inc.		

Features (contd)

- Two bus parking modes selectable via Control Register write:
 - park on selected OPB master (specified in Control Register)
 - park on last OPB master granted OPB access
- Watchdog timer 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

Functional Description

The top-level block diagram for the OPB Arbiter is shown in **Figure 1**. 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 modules shown in the block diagram are described in the following sections.

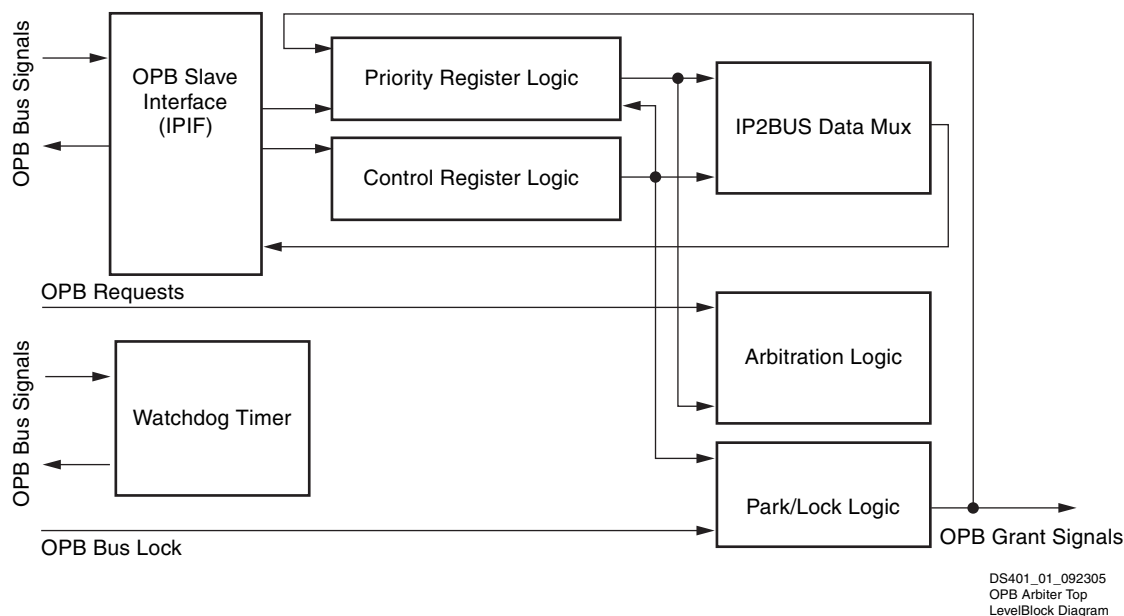


Figure 1: OPB Arbiter Top Level Block Diagram

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 the [OPB Arbiter Control Register](#) section of this document and is present in the design only 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 [OPB Arbiter Register Descriptions](#) section of this document. The Priority Registers are present in the design only if `C_NUM_MASTERS>1`.

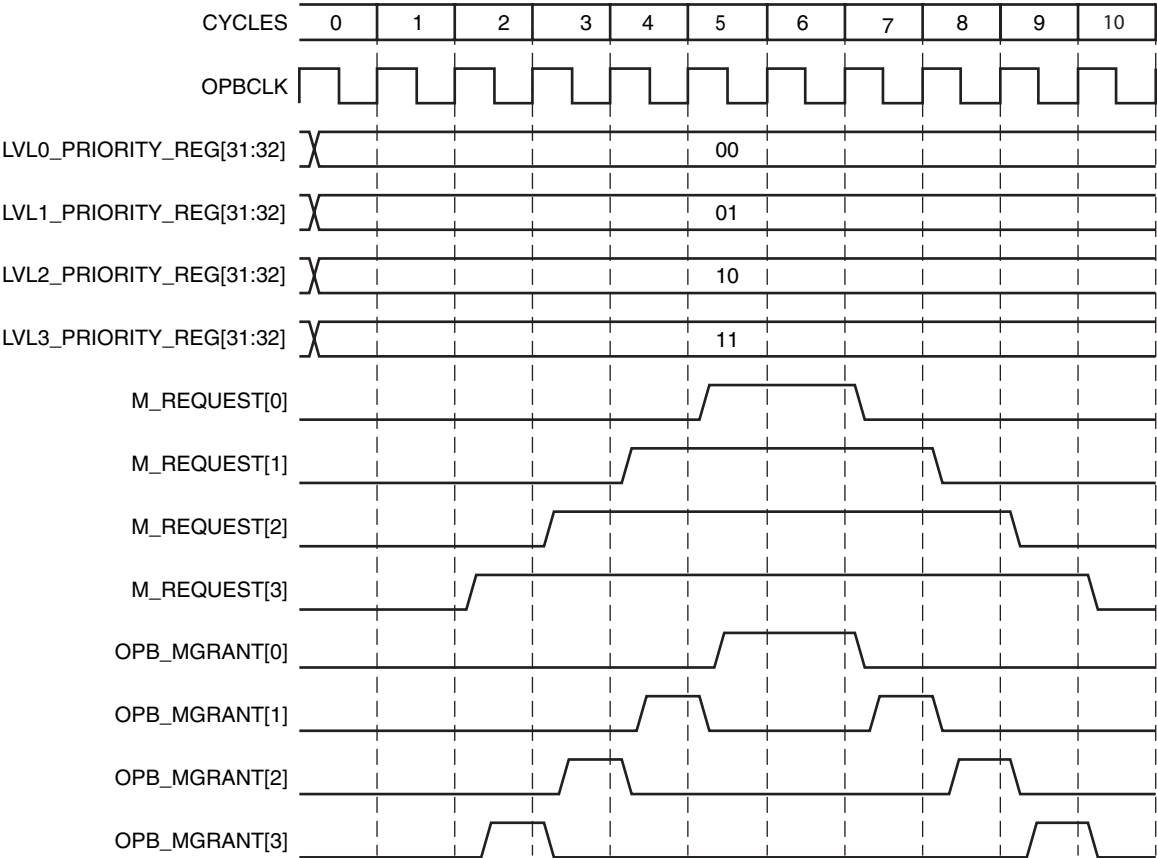
Priority Register Update Logic

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`), therefore, 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.

[Figure 1](#) shows the fixed priority arbitration for a 4 OPB Master system with combinational grant outputs. [Figure 3](#) shows the same system when configured with registered grant outputs with the master requests separated by an additional clock.



DS401_02_092305
Fixed Priority Arbitration,
Combinational Grant Outputs
for 4 OPB Masters

Figure 2: Fixed Priority Arbitration, Combination Grant Outputs for 4 OPB Masters

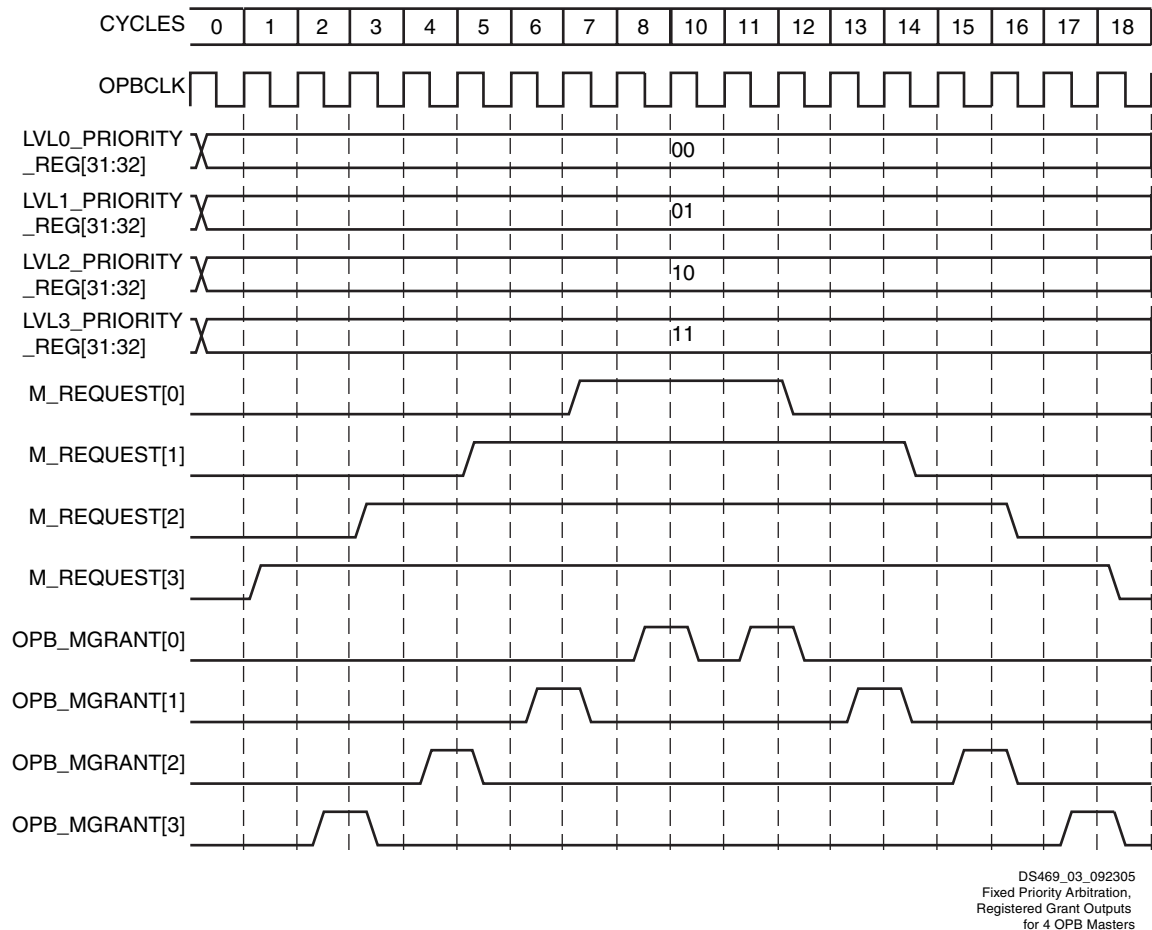


Figure 3: Fixed Priority Arbitration, Registered Grant Outputs for 4 OPB Masters

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. Therefore, 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. **Figure 4** shows the dynamic priority arbitration for a 4 OPB master system with combinational grant outputs. **Figure 5** shows dynamic priority arbitration for a 4 master OPB system with registered grant outputs.

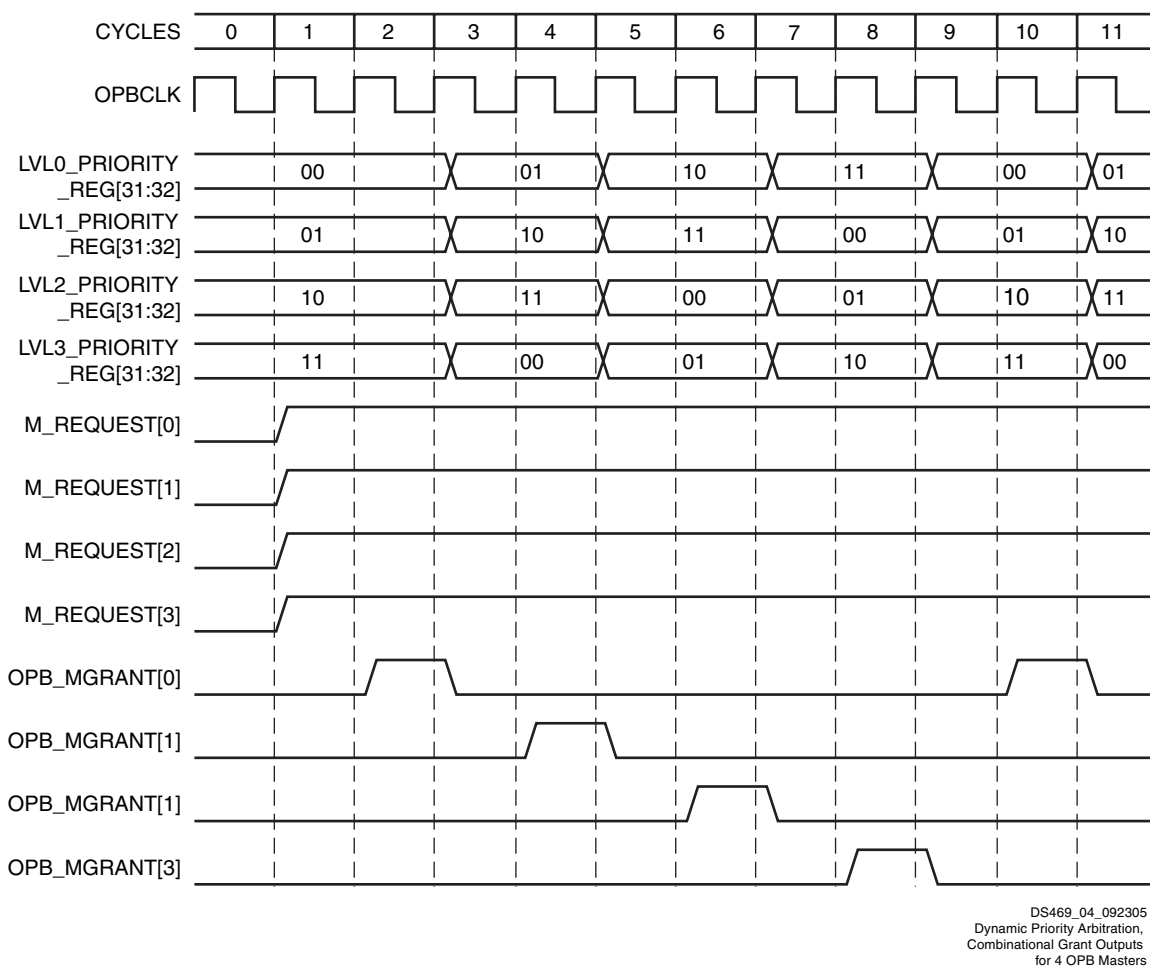


Figure 4: Dynamic Priority Arbitration, Combinational Grant Outputs- 4 OPB Masters

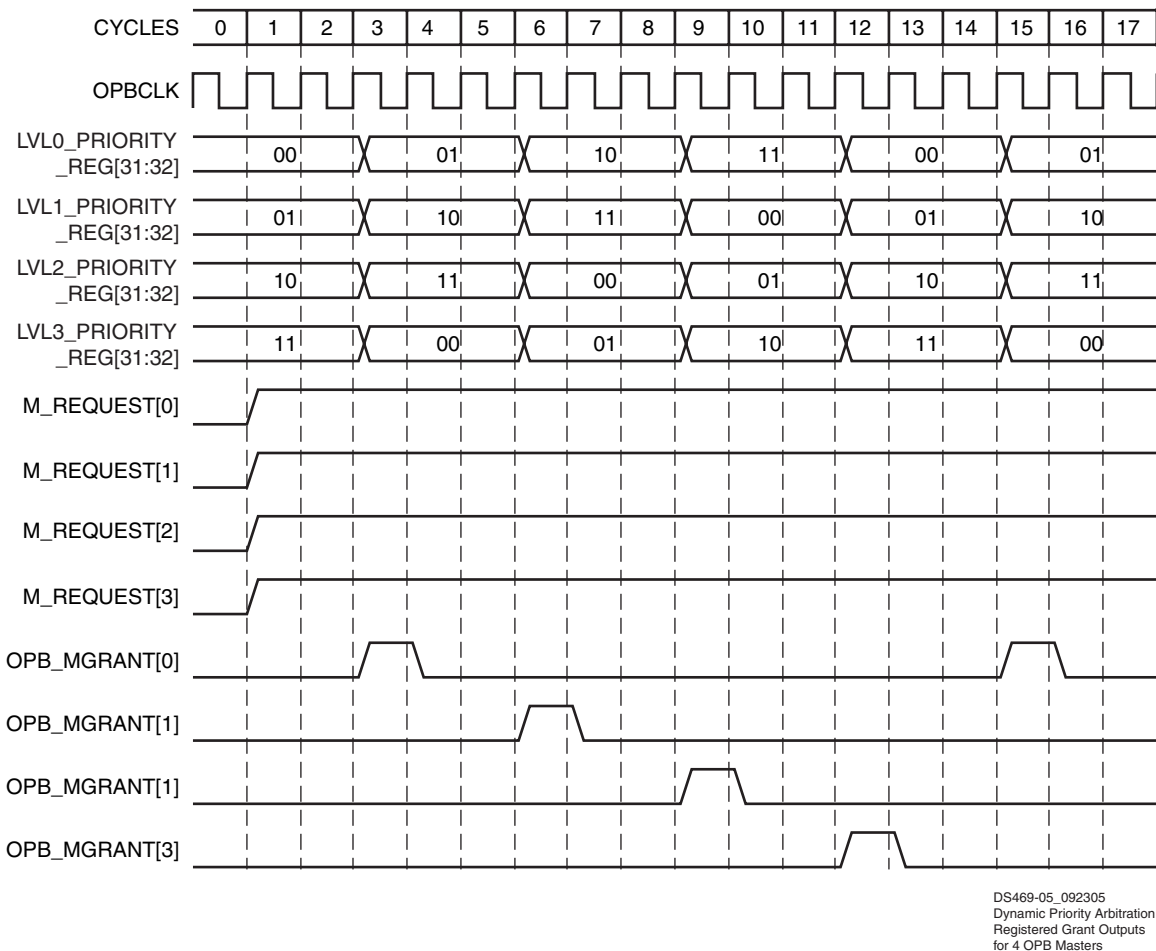
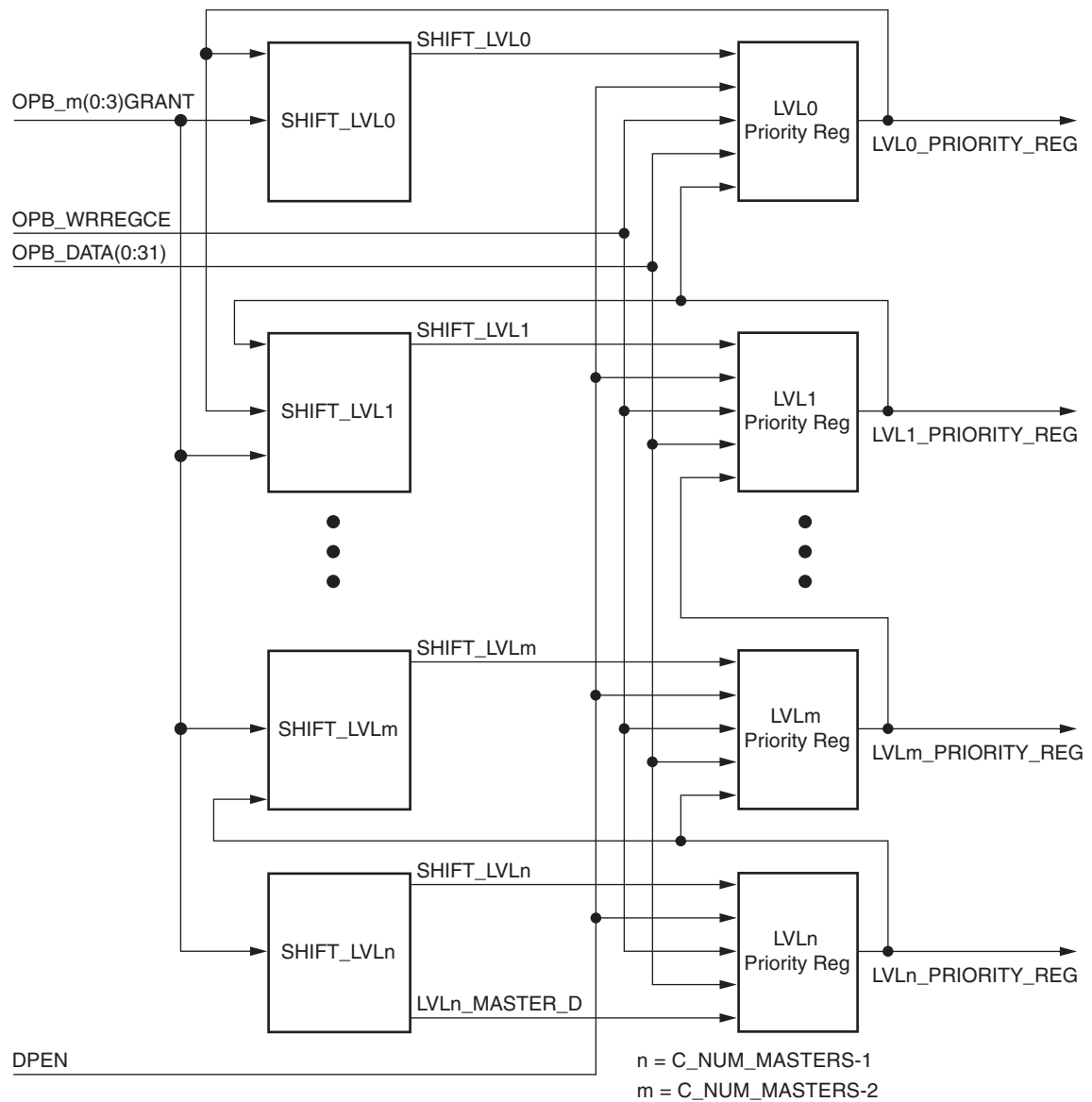


Figure 5: Dynamic Priority Arbitration, Registered Grant Outputs- 4 OPB Masters

Priority Registers and Register Update Logic Block Diagram

The block diagram for the Priority registers and the register update logic is shown in Figure 6. 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.



DS469_06_092305
Priority Register Logic

Figure 6: 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 7 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 multiplexors 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 therefore 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 deasserted 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.

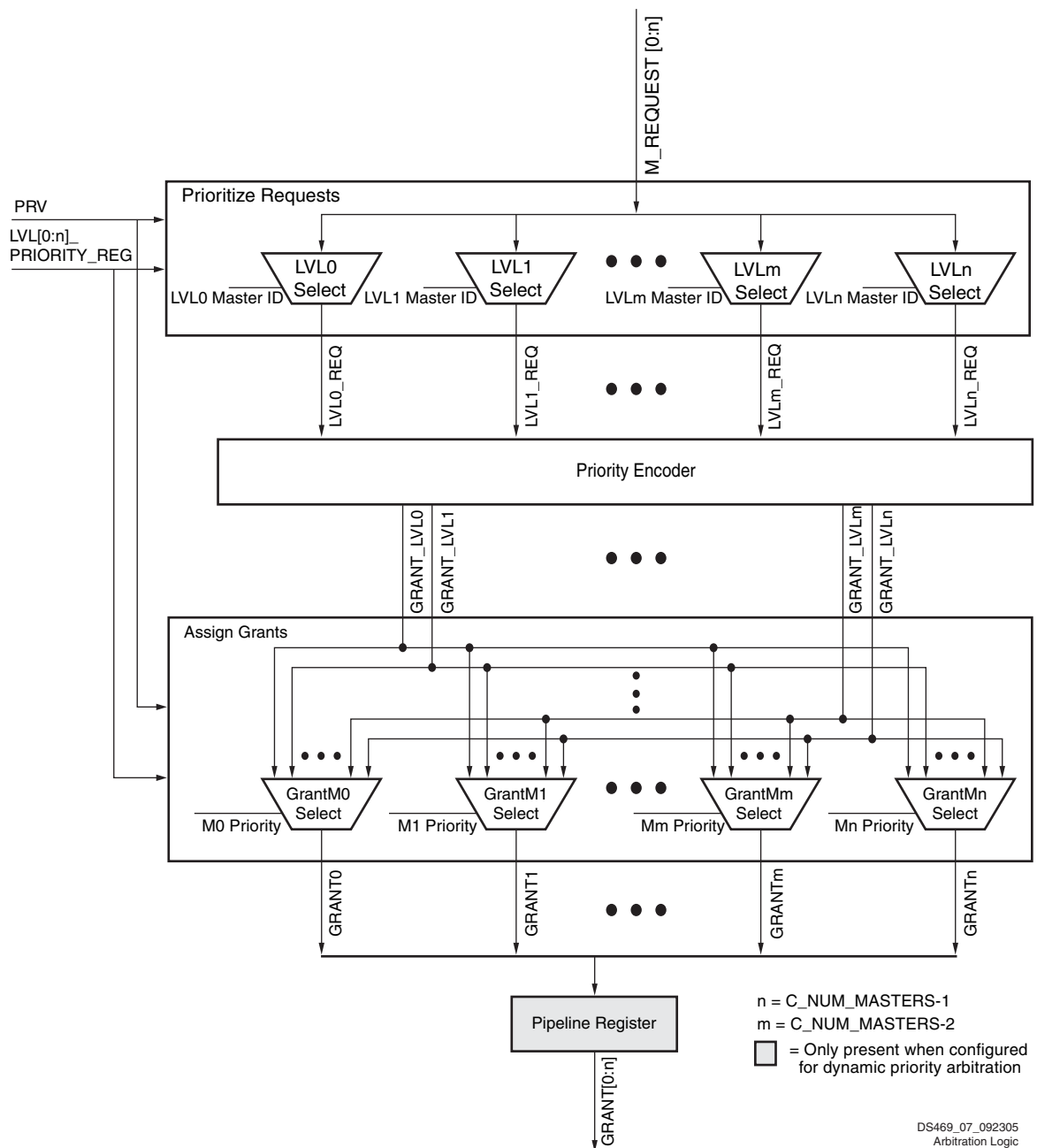
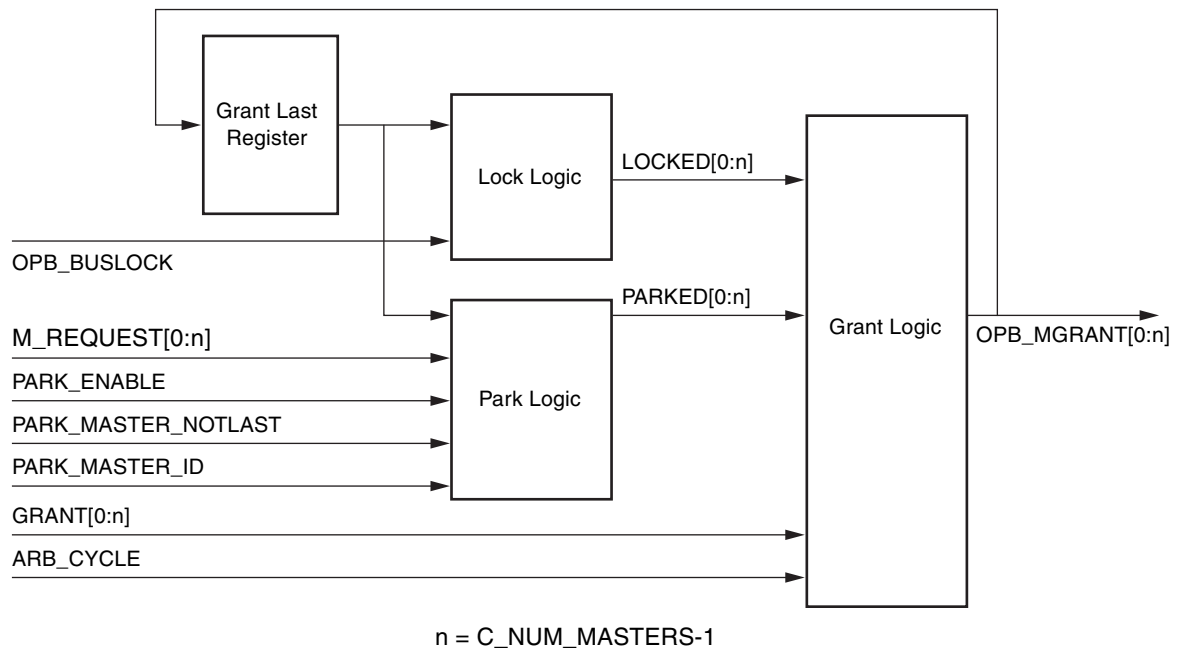


Figure 7: Arbitration Logic

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 8. The OPB Arbiter can be parameterized to register the master grant signals by setting the parameter C_REG_GRANTS to 1.



DS469_08_092305
Park/Lock Logic

Figure 8: 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 ORed 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 deasserted 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 9** shows the OPB bus lock operation when the OPB Arbiter is configured for fixed priority arbitration and combinational grant outputs. **Figure 10** 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.

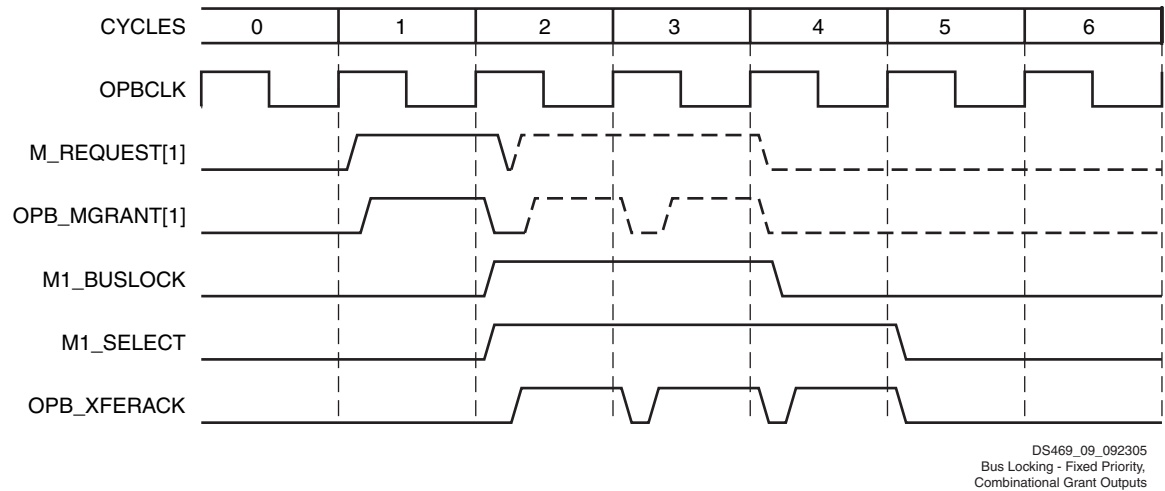


Figure 9: **Bus Locking - Fixed Priority, Combinational Grant Outputs**

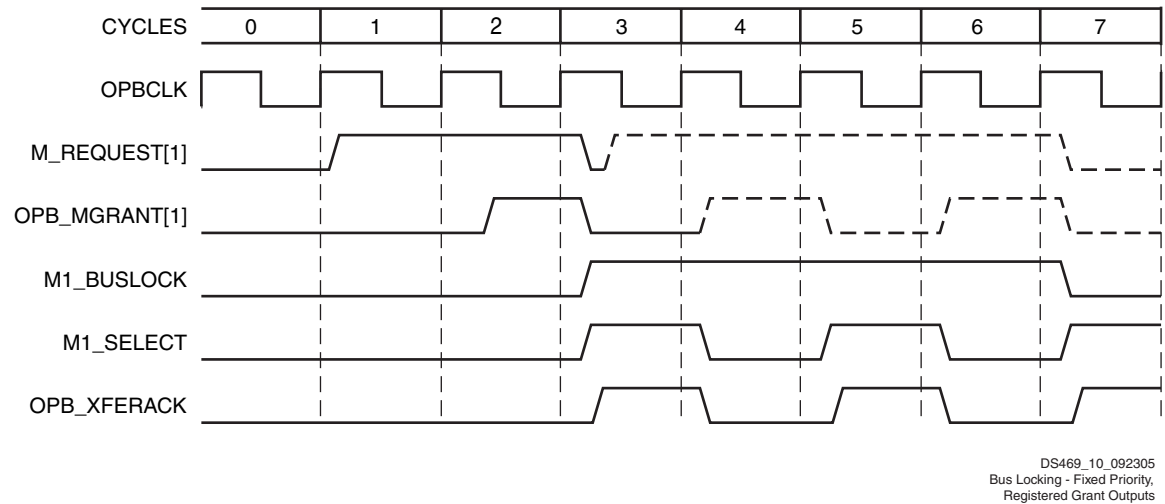


Figure 10: **Bus Locking - Fixed Priority, Registered Grant Outputs**

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 11** illustrates this behavior when the OPB Arbiter is configured to support fixed priority arbitration and combinational grant outputs. **Figure 12** 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.

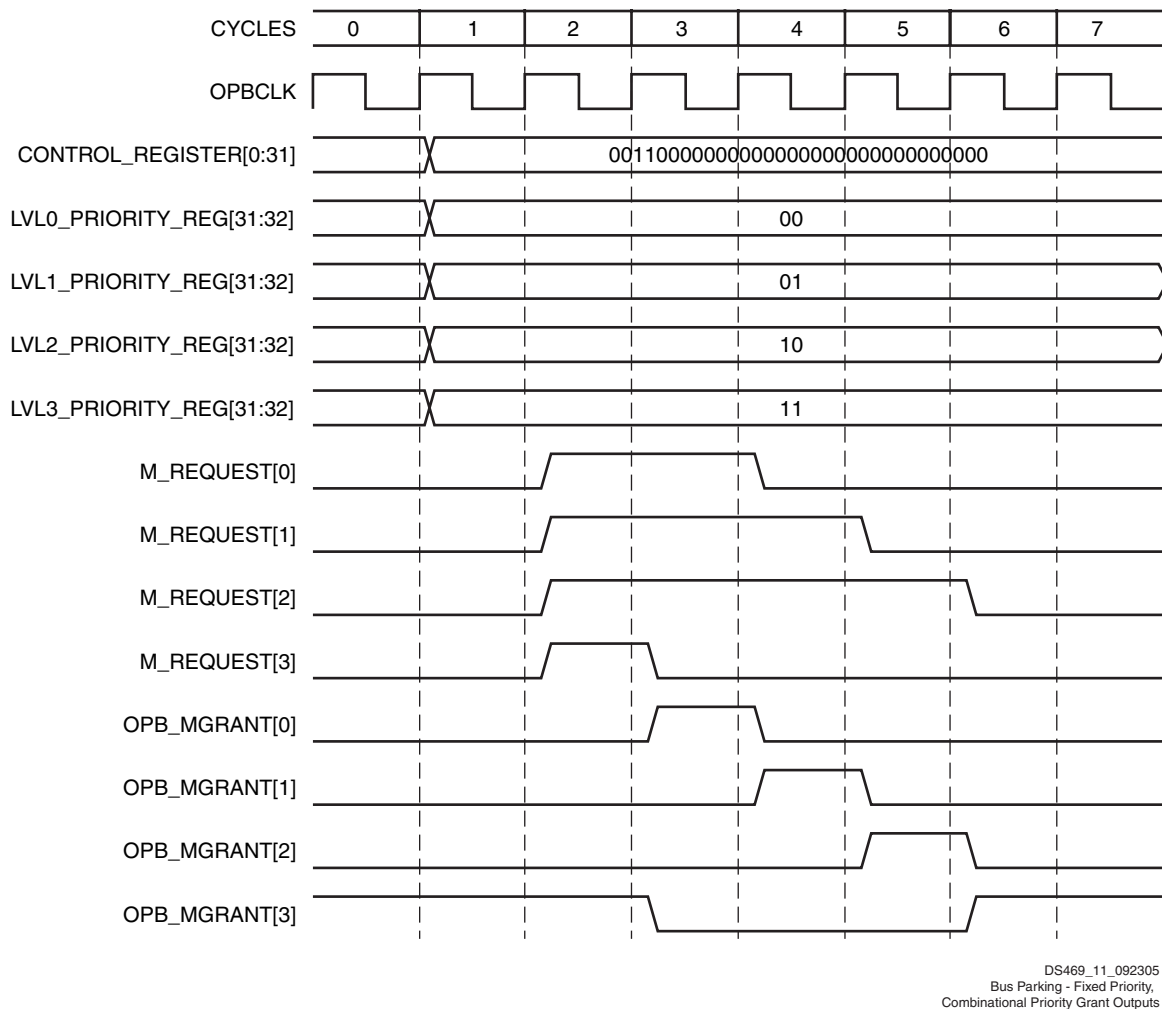


Figure 11: Bus Parking - Fixed Priority Arbitration, Combinational Grant Outputs

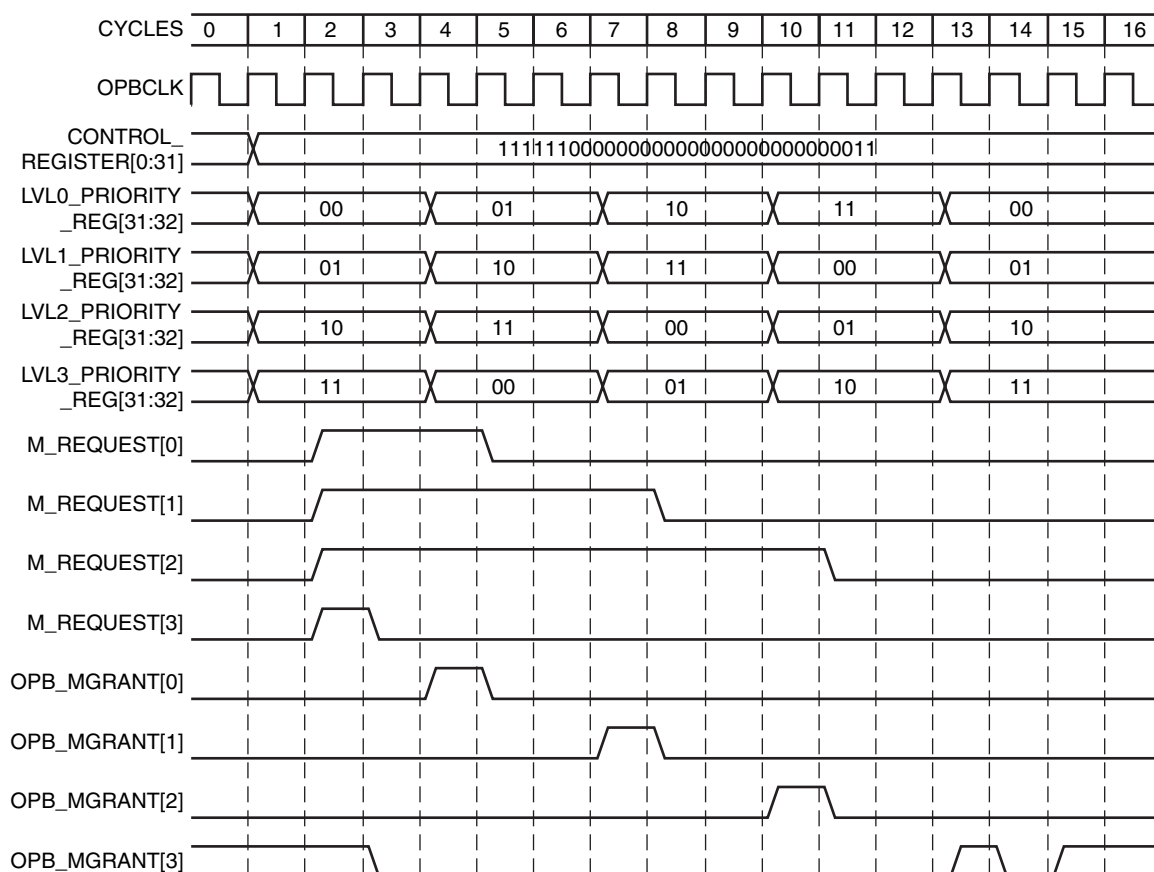
When the grant outputs are registered, the parked master's grant will assert once, negate, and then stay asserted. This allows the internal arbitration state machine a clock cycle to check if OPB_select is asserted before parking. This case occurs when a request from a master is aborted, but the grant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

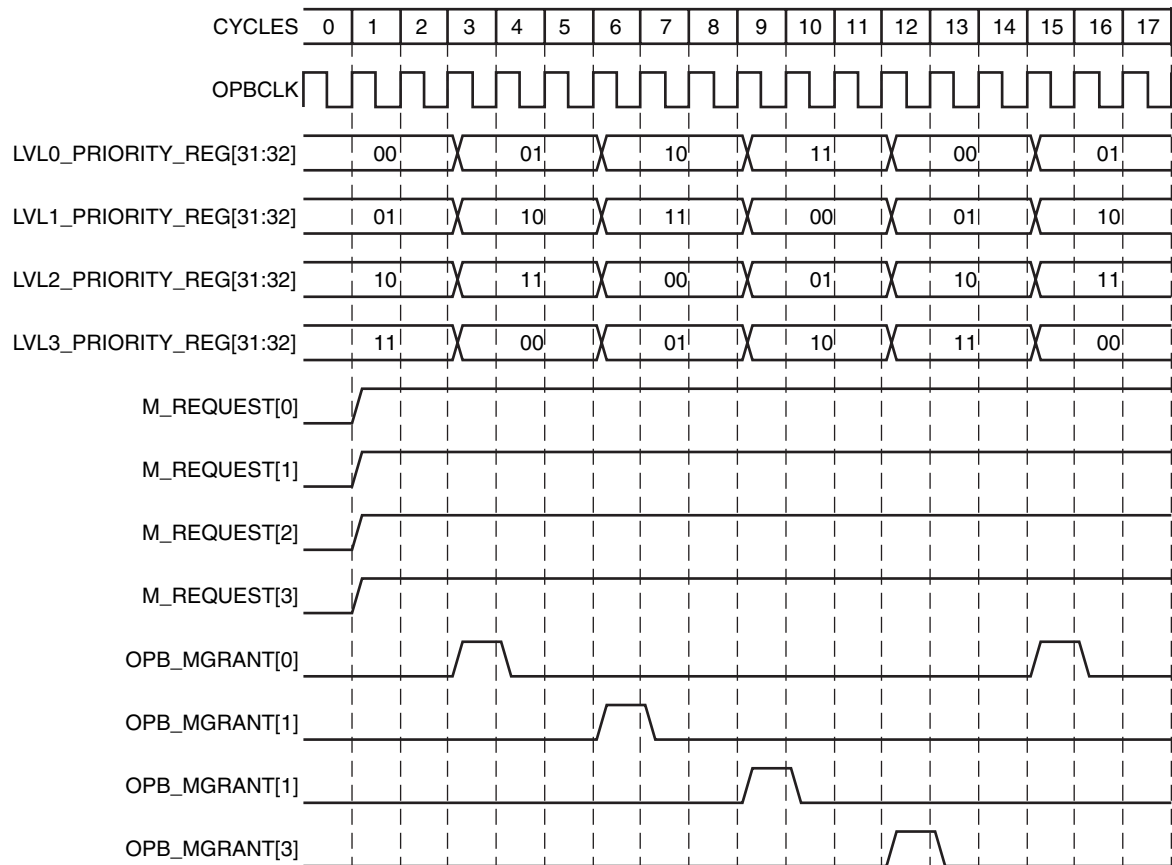
Park on Master Not Last

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 13** 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.



DS469_12_092305
Bus Parking - Dynamic Priority
Arbitration, Registered Grant Outputs

Figure 12: Bus Parking - Dynamic Priority Arbitration, Registered Grant Outputs

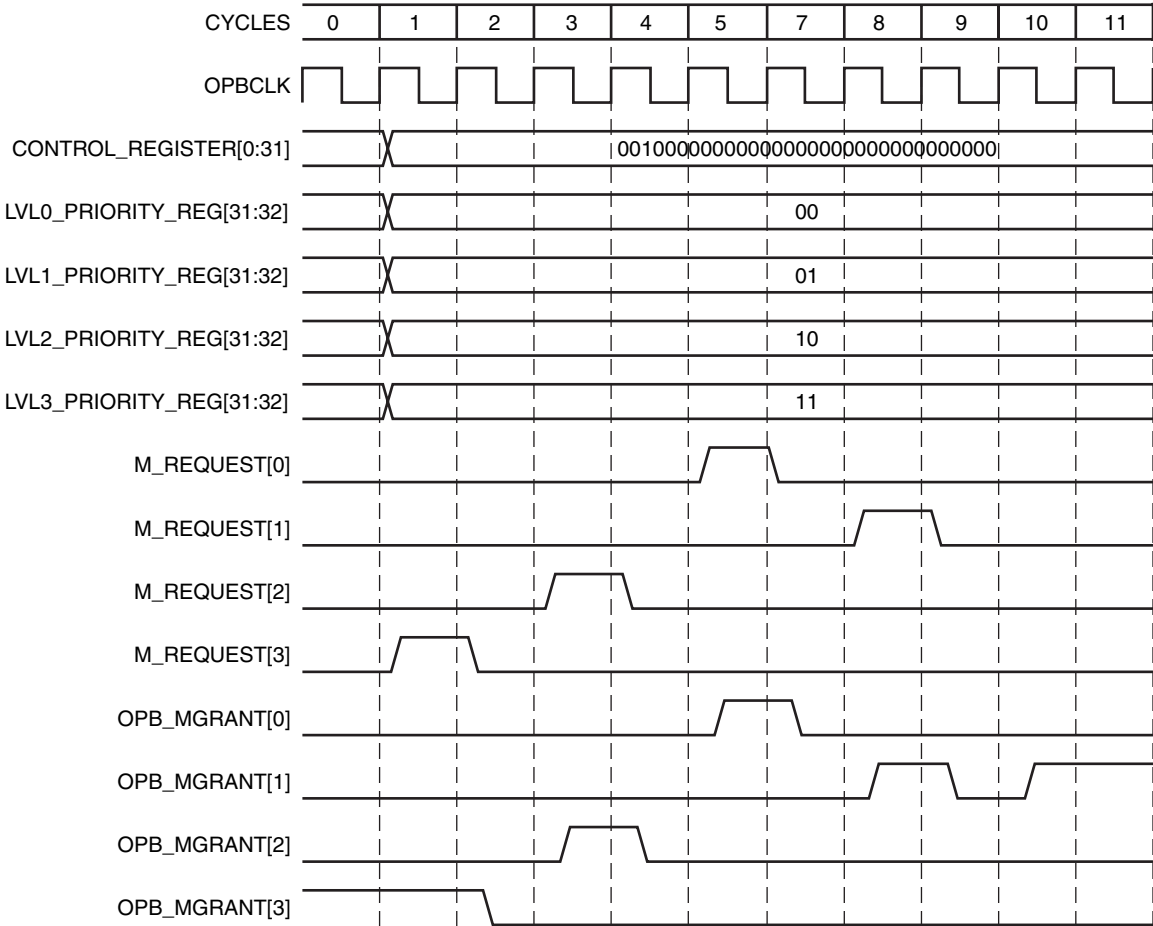


DS469_13_092305
Bus Parking on Master Not Last -
Fixed Priority Arbitration,
Combinational Grant Outputs

Figure 13: 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 14 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.



DS469_14_092305
Bus Parking on Last Master -
Fixed Priority Arbitration,
Combinational Grant Outputs

Figure 14: 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 or 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 18.

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. 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 deasserting Mn_select in the cycle following the assertion of OPB_timeout as shown in **Figure 15**. If OPB_busLock is not asserted, the OPB Arbiter will perform a bus arbitration in the cycle in which OPB_select is deasserted. If OPB_busLock is asserted, the requesting master retains control of the OPB, but must still deassert 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.

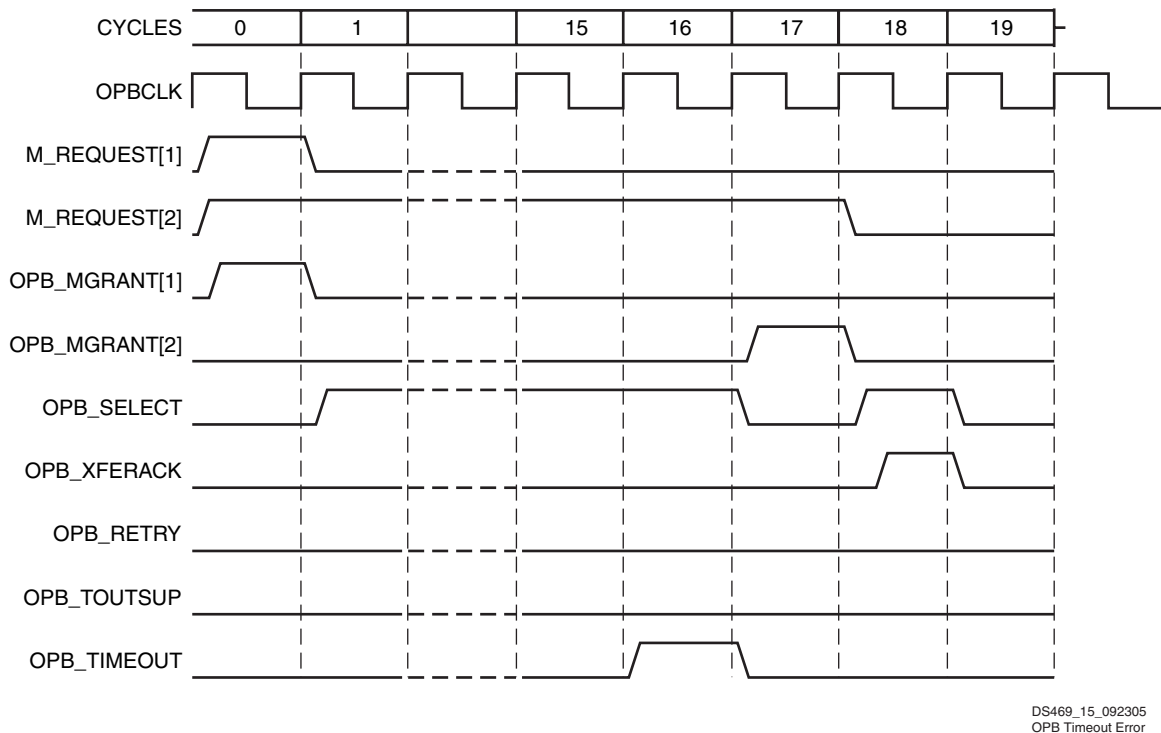


Figure 15: OPB Timeout Error

To prevent a bus timeout, an OPB slave must assert OPB_toutSup (OR of all slave's Sln_ToutSup) within 16 cycles from 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 16**.

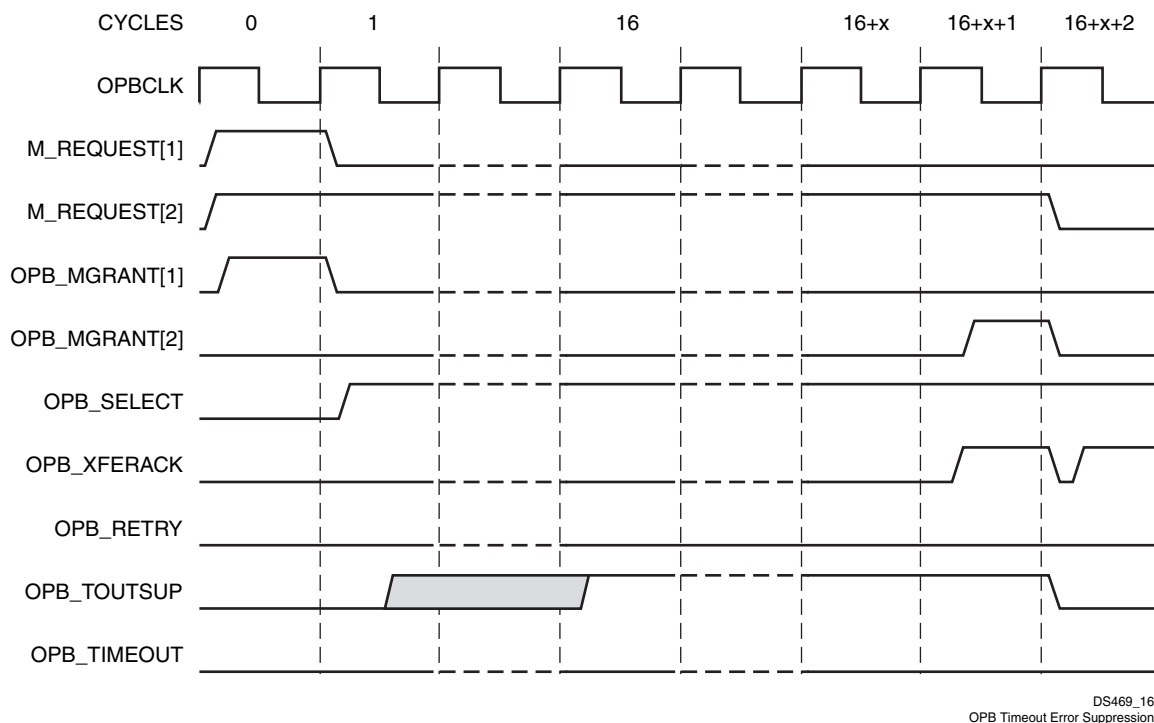


Figure 16: OPB Timeout Error Suppression

OPB Arbiter I/O Signals

The I/O signals for the OPB Arbiter are listed in [Table 1](#). The interfaces referenced in this table are shown in [Figure 1](#) in the OPB Arbiter block diagram.

Table 1: OPB Arbiter I/O Signals

Port	Signal Name	Interface	I/O	Initial State	Description
OPB Slave Signals					
P1	ARB_DBus(0:C_OPB_DWIDTH-1)	IPIF	O	0	Arbiter output data bus
P2	ARB_xferAck	IPIF	O	0	Arbiter transfer acknowledge
P3	ARB_Retry	IPIF	O	0	Arbiter retry
P4	ARB_ToutSup	IPIF	O	0	Arbiter timeout suppress
P5	ARB_ErrAck	IPIF	O	0	Arbiter error acknowledge
P6	OPB_ABus(0:C_OPB_AWIDTH-1)	IPIF	I		OPB address bus
P7	OPB_BE(0:C_OPB_DWIDTH/8-1)	IPIF	I		OPB byte enables
P8	OPB_DBus(0:C_OPB_DWIDTH-1)	IPIF	I		OPB data bus
P9	OPB_RNW	IPIF	I		Read not Write (OR of all master RNW signals)
P10	OPB_seqAddr	IPIF	I		OPB sequential address

Table 1: OPB Arbiter I/O Signals (Contd)

Port	Signal Name	Interface	I/O	Initial State	Description
Arbitration Signals					
P11	M_request[0:C_NUM_MASTERS-1] ⁽¹⁾	Arbitration Logic	I		Request from OPB Masters
P12	OPB_xferAck	Arbitration Logic	I		Transfer Acknowledge indicating end of data transfer cycle (OR of all slave xferAcks)
P13	OPB_select	Arbitration Logic Watchdog Timer	I		Master has taken control of the bus (OR of all master selects)
P14	OPB_retry	Watchdog Timer	I		Bus cycle retry (OR of all slave retrys)
P15	OPB_toutSup	Watchdog Timer	I		Suppress timeout (OR of all slave toutSups)
P16	OPB_timeout	Watchdog Timer	O	0	Timeout signal for OPB
P17	OPB_busLock	Park/Lock Logic	I		Bus lock (OR of all master buslocks)
P18	OPB_MGrant[0:C_NUM_MASTERS-1] ⁽¹⁾	Park/Lock Logic	O	0	Grant to OPB Masters
System					
P19	OPB_Clk	System	I		System clock
P20	OPB_Rst	System	I		System Reset (active high)
Notes:					
1. Name has been modified slightly from that in the IBM OPB Arbiter specification to support parameterization of the number of masters					

The signal, ARB_DBusEn, is not an output of the Xilinx OPB Arbiter as the IPIF module internally gates the OPB Arbiter data bus with the enable signal.

The following signals listed in the IBM OPB Arbiter core are not supported:

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

OPB Arbiter Design Parameters

To obtain an OPB Arbiter that is uniquely tailored to a specific system, certain features in the OPB Arbiter design can be parameterized. This allows the user to have a high performance design that utilizes only the resources required by the system. **Table 2** shows the parameterizable features in the Xilinx OPB Arbiter design.

Table 2: OPB Arbiter Design Parameters

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Arbiter Features					
G1	Number of OPB Masters	C_NUM_MASTERS	1 ⁽¹⁾ - 16	4	integer
G2	Priority Mode	C_DYNAM_PRIORITY	1 = Dynamic 0 = Fixed	0 = Fixed	integer
G3	Registered Grant Outputs	C_REG_GRANTS	1 = Registered Grant Outputs 0 = Combinational Grant Outputs ⁽²⁾	1 = Registered Grant Outputs	integer
G4	Bus Parking	C_PARK	1 = Bus parking supported ⁽³⁾ 0 = Bus parking not supported	0 = Bus parking not supported	integer
OPB Interface					
G5	OPB Slave Interface	C_PROC_INTRFCE	1 = OPB slave interface supported 0 = OPB slave interface not supported ⁽⁴⁾	0 = OPB slave interface not supported	integer
G6	OPB Arbiter Base Address	C_BASEADDR	Valid Address Range ⁽⁶⁾	None ⁽⁵⁾	std_logic_vector
G7	OPB Data Bus Width	C_OPB_DWIDTH	32	32	integer
G8	OPB Address Bus Width	C_OPB_AWIDTH	32	32	integer
G9	Device Block ID ⁽⁵⁾	C_DEV_BLK_ID	See note 6.	0	integer

Table 2: OPB Arbiter Design Parameters

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
G10	Module Identification Register Enable ⁽⁵⁾	C_DEV_MIR_ENABLE	See note 6.	0	integer
G11	OPB High Address	C_HIGHADDR	Address range must be a power of 2 and $\geq 0x1FF$ ⁽⁷⁾	None ⁽⁵⁾	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. 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.
3. 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.
4. When C_PROC_INTRFCE is 0, none of the OPB Arbiter registers are accessible.
5. 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.
6. Address range specified by C_BASEADDR and C_HIGHADDR must be at least 0x1FF and must be a power of 2. C_BASEADDR must be a multiple of the range, where the range is C_HIGHADDR - C_BASEADDR + 1.

Allowable Parameter Combinations

The only restriction on parameter combinations in the Xilinx OPB Arbiter design 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 clock frequency of the OPB specified in the Platform Builder tool must be low enough to allow for combinational Grant outputs. Therefore, the parameter C_REG_GRANTS can only be set to 0 if the value of the OPB frequency is less than the f_{MAX} specified for this parameter.

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 therefore needed in the system. When there is only 1 OPB master, there will be no Control Register or Priority Registers, therefore, there will be no OPB slave interface on the OPB Arbiter. Because there will not be an OPB slave interface, the OPB Arbiter, when parameterized for 1 OPB master, will not have a Configuration ROM (CROM) entry. 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 and there is no CROM entry for the OPB Arbiter.

Parameter - Port Dependencies

The width of many of the OPB Arbiter 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 design parameters and I/O signals are shown in [Table 3](#).

Table 3: Parameter-Port Dependencies

	Name	Affects	Depends	Relationship Description
Generic or Port	Design Parameters			
G1	C_NUM_MASTERS	G2-G8 P1,P2 P6-P11 P17,P18	G4, G5, G7	<p>The width of the request and grant buses are set by the number of masters in the design.</p> <p>The only logic present in the OPB Arbiter design when C_NUM_MASTERS=1 is the watchdog timer, therefore, many parameters and I/O signals are unconnected in this case.</p> <p>If C_OPB_DWIDTH=8 and C_PARK=1 and C_PROC_INTRFCE=1 then C_NUM_MASTERS must be <=4 because there is only 2 bits for the park master id in the Control Register.</p>
G2	C_DYNAM_PRIORITY		G1	Unconnected if C_NUM_MASTERS=1.
G3	C_REG_GRANTS		G1	Unconnected if C_NUM_MASTERS=1.
G4	C_PARK	G1	G1	Unconnected if C_NUM_MASTERS=1.
G5	C_PROC_INTRFCE	G1,G6-G8, G11 P1, P2 P6-P10	G1	Unconnected if C_NUM_MASTERS=1.
G6	C_BASEADDR		G1,G5, G11	Unconnected if C_PROC_INTRFCE=0 or C_NUM_MASTERS=1. Range specified by C_BASEADDR and C_HIGHADDR must be at least 0x1FF.
G7	C_OPB_DWIDTH	G1, P1, P8	G1,G5	Unconnected if C_PROC_INTRFCE=0 or C_NUM_MASTERS=1.
G8	C_OPB_AWIDTH	P6,P7	G1,G5	Unconnected if C_PROC_INTRFCE=0 or C_NUM_MASTERS=1.
G9	C_DEV_BLK_ID		G1,G5	Unconnected if C_PROC_INTRFCE=0 or C_NUM_MASTERS=1.
G10	C_DEV_MIR_ENABLE		G1,G5	Unconnected if C_PROC_INTRFCE=0 or C_NUM_MASTERS=1.
G11	C_HIGHADDR		G1,G5, G6	Unconnected if C_PROC_INTRFCE=0 or C_NUM_MASTERS=1. Range specified by C_BASEADDR and C_HIGHADDR must be at least 0x1FF.
I/O Signals				
P1	ARB_DBus(0:C_OPB_DWIDTH-1)		G1,G5, G7	<p>Width varies with the size of the OPB Data bus.</p> <p>This output is grounded if C_PROC_INTRFCE = 0 or C_NUM_MASTERS=1.</p>

Table 3: Parameter-Port Dependencies (Contd)

	Name	Affects	Depends	Relationship Description
P2	ARB_xferAck		G1,G5	This output is grounded if C_PROC_INTRFCE = 0 or C_NUM_MASTERS=1.
P3	ARB_Retry			This output is always grounded.
P4	ARB_ToutSup			This output is always grounded.
P5	ARB_ErrAck			This output is always grounded.
P6	OPB_ABus(0:C_OPB_AWIDTH-1)		G1,G5, G8	Width varies with the size of the OPB Address bus. This input is unconnected if C_PROC_INTRFCE = 0 or C_NUM_MASTERS = 1.
P7	OPB_BE(0:C_OPB_DWIDTH/8-1)		G1,G5, G8	Width varies with the size of the OPB Address bus. This input is unconnected if C_PROC_INTRFCE = 0 or C_NUM_MASTERS = 1.
P8	OPB_DBUS(0:C_OPB_DWIDTH-1)		G1,G5, G7	Width varies with the size of the OPB Data bus. This input is unconnected if C_PROC_INTRFCE = 0 or C_NUM_MASTERS = 1.
P9	OPB_RNW		G1,G5	This input is unconnected if C_PROC_INTRFCE = 0 or C_NUM_MASTERS = 1.
P10	OPB_seqAddr		G1,G5	This input is unconnected if C_PROC_INTRFCE = 0 or C_NUM_MASTERS = 1.
P11	M_request(0:C_NUM_MASTERS-1)		G1	Width varies with the number of OPB masters. This input is unconnected if C_NUM_MASTERS=1.
P12	OPB_xferAck			
P13	OPB_select			
P14	OPB_retry			
P15	OPB_toutSup			
P16	OPB_timeout			
P17	OPB_busLock		G1	This input is unconnected if C_NUM_MASTERS=1.

Table 3: Parameter-Port Dependencies (Contd)

	Name	Affects	Depends	Relationship Description
P18	OPB_MGrant(0:C_NUM_MASTERS-1)		G1	Width varies with the number of OPB masters. This output is set to 1 if C_NUM_MASTERS = 1.
P19	OPB_Clk			
P20	OPB_Rst			

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 deasserted, 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.

The fixed OPB arbitration protocol with combinational grant outputs is shown in [Figure 17](#), while [Figure 18](#) shows the fixed OPB arbitration protocol when the OPB Arbiter has been parameterized to have registered grant outputs.

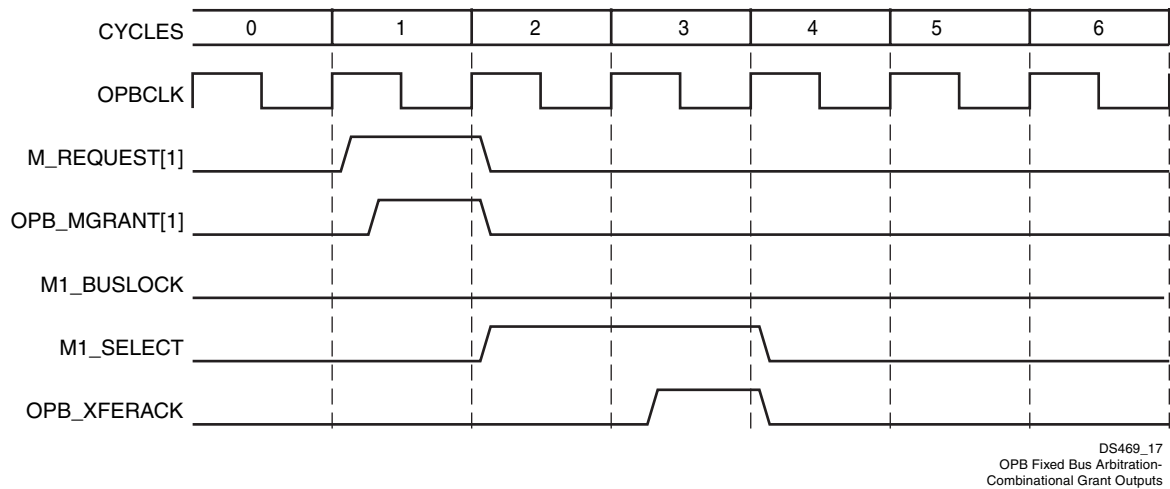


Figure 17: OPB Fixed Bus Arbitration - Combinational Grant Outputs

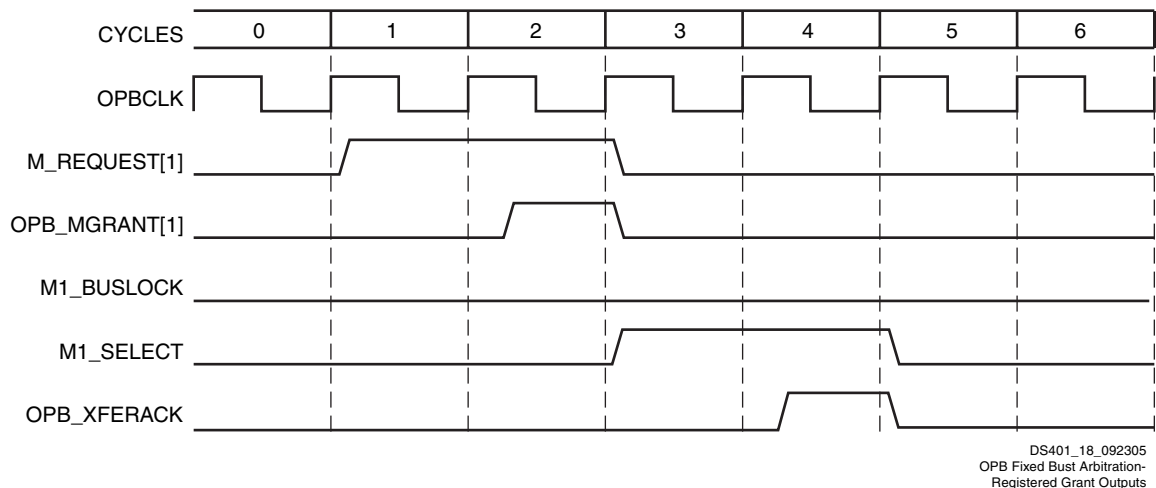


Figure 18: OPB Fixed Bus Arbitration -Registered Grant Outputs

An OPB master device need not deassert its request upon receipt of a bus grant signal if it has multiple bus transfer cycles to perform. **Figure 19** 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, therefore bus grant is asserted combinational during valid arbitration cycles.

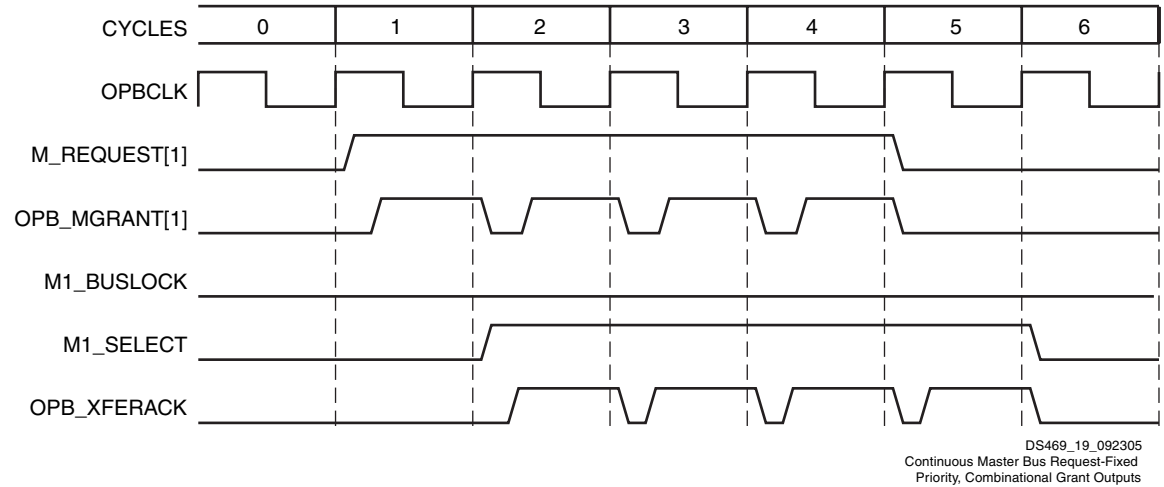


Figure 19: Continuous Master Bus Request - Fixed priority, Combinational Grant Outputs

When the OPB Arbiter has been parameterized for registered grant outputs and fixed priority, the bus grants are registered as shown in Figure 20.

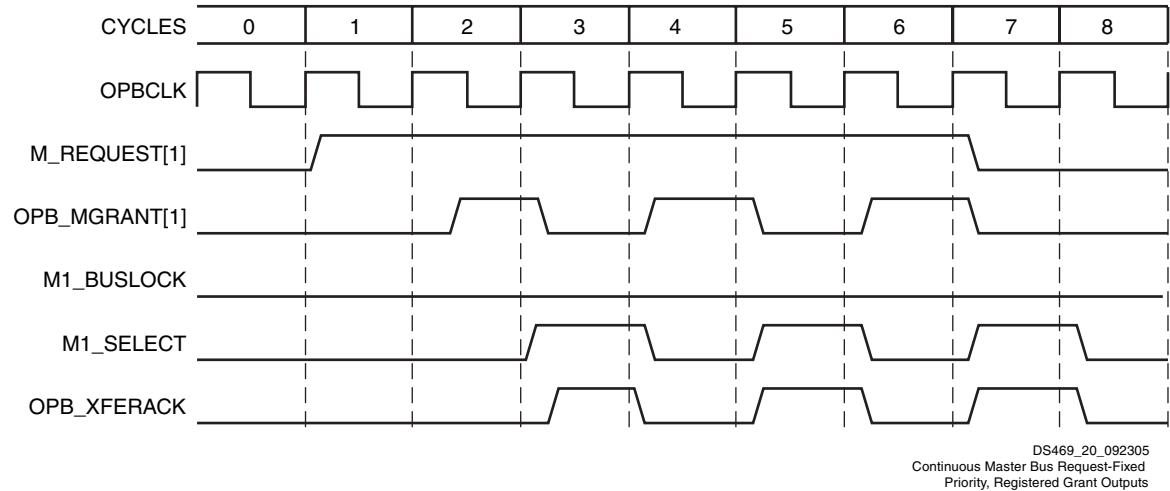
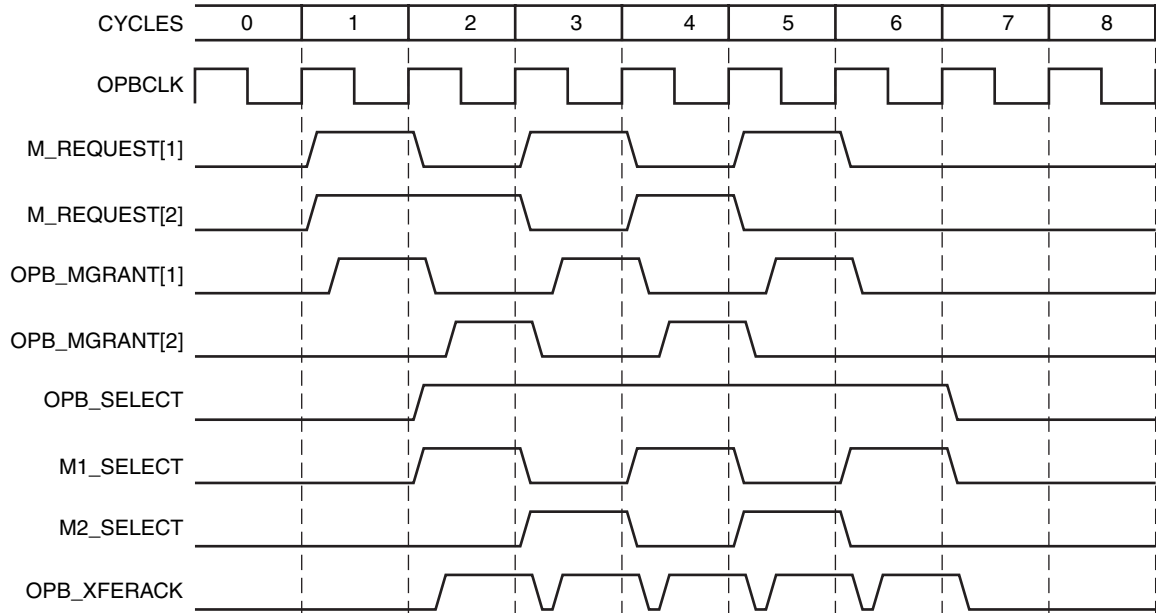


Figure 20: Continuous Master Bus Request - Fixed priority, Registered Grant Outputs

Even if an OPB master asserts request continuously, it will not necessarily receive a valid grant signal. Other OPB masters with higher bus priority may request the OPB and will be 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 21 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,

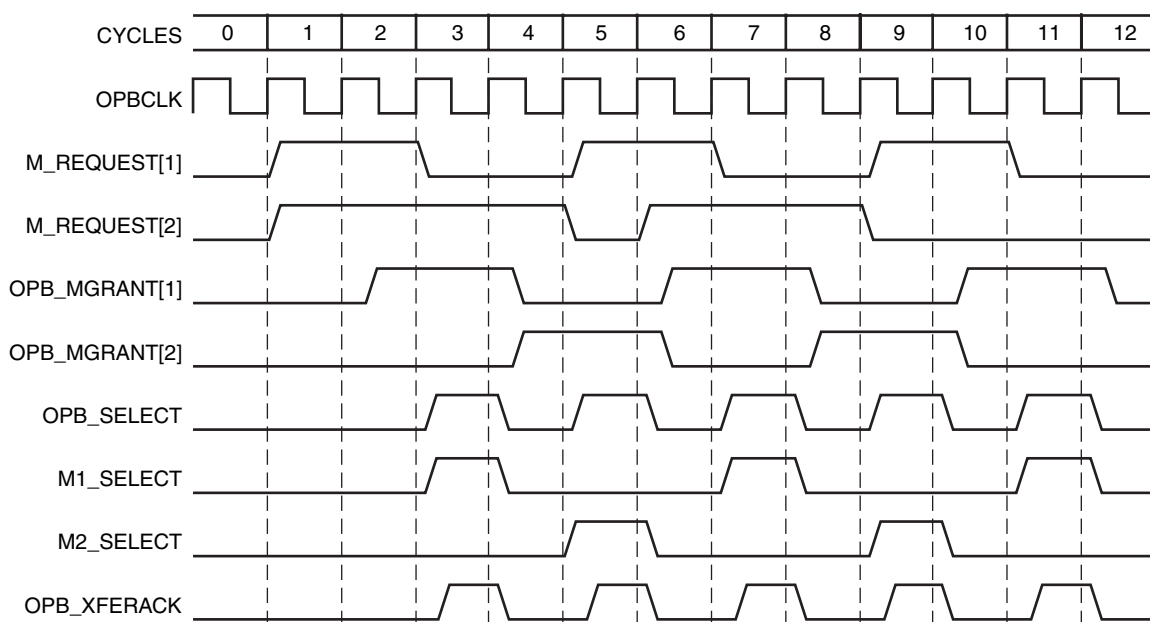
Master 1 completes its first transaction and Master 2 is granted the bus for cycle 3. Thus, during cycle 2, the arbitration for the bus is overlapped with a data transfer. This overlapped bus arbitration improves the bandwidth of the bus.



DS469_21_092305
Multiple Bus Request-Fixed Priority
Arbitration, Combinational Grant Outputs

Figure 21: 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 22. 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.



DS469_22_092305
Multiple Bus Request-Fixed Priority
Arbitration, Registered Grant Outputs

Figure 22: Multiple Bus Requests - Fixed Priority Arbitration, Registered Grant Outputs

OPB 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. 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 hence the number of Priority Registers will vary 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	Access
Control Register	C_BASEADDR + 0x100	Read/Write
LVL0 Priority Register	C_BASEADDR + 0x104	Read/Write
LVL1 Priority Register	C_BASEADDR + 0x108	Read/Write
LVL2 Priority Register	C_BASEADDR + 0x10C	Read/Write
LVL3 Priority Register	C_BASEADDR + 0x110	Read/Write
LVL4 Priority Register	C_BASEADDR + 0x114	Read/Write

Table 4: OPB Arbiter Registers (Contd)

Register Name	OPB Address	Access
LVL5 Priority Register	C_BASEADDR + 0x118	Read/Write
LVL6 Priority Register	C_BASEADDR + 0x11C	Read/Write
LVL7 Priority Register	C_BASEADDR + 0x120	Read/Write
LVL8 Priority Register	C_BASEADDR + 0x124	Read/Write
LVL9 Priority Register	C_BASEADDR + 0x128	Read/Write
LVI10 Priority Register	C_BASEADDR + 0x12C	Read/Write
LVL11 Priority Register	C_BASEADDR + 0x130	Read/Write
LVL12 Priority Register	C_BASEADDR + 0x134	Read/Write
LVL13 Priority Register	C_BASEADDR + 0x138	Read/Write
LVL14 Priority Register	C_BASEADDR + 0x13C	Read/Write
LVL15 Priority Register	C_BASEADDR + 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 the [Specification Exceptions](#) section in this document for more information.

OPB Arbiter Control Register

The OPB Arbiter Control Register is shown in [Figure 23](#) 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.

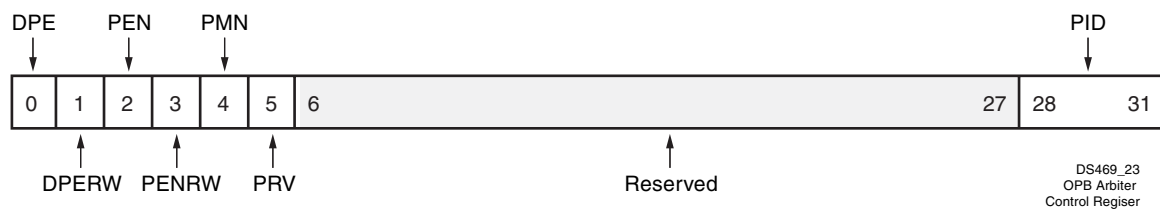


Figure 23: OPB Arbiter Control Register

[Table 5](#) shows the Control Register bit definitions. The Control Register definition deviates from the IBM OPB Arbiter specification. This deviation is necessary due to the fact that the Park Master ID (PID) field will vary 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' ⁽¹⁾ '1' ⁽²⁾	Dynamic Priority Enable. Enables dynamic priority arbitration algorithm and update of the Priority Register. <ul style="list-style-type: none"> • '0' - dynamic priority arbitration disabled • '1' - dynamic priority arbitration enabled
1	DPERW	Read	'0' ⁽¹⁾ '1' ⁽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. <ul style="list-style-type: none"> • '0' - DPE bit is read only • '1' - DPE bit is read/write
2	PEN	Read/Write	'0' ⁽³⁾ '1' ⁽⁴⁾	Park Enable. Enables parking on a master when no other masters have requests asserted. <ul style="list-style-type: none"> • '0' - parking disabled • '1' - parking enabled
3	PENRW	Read	'0' ⁽³⁾ '1' ⁽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. <ul style="list-style-type: none"> • '0' - PEN bit is read only • '1' - PEN bit is read/write
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. <ul style="list-style-type: none"> • '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.

Table 5: OPB Arbiter Control Register Bit Definitions (Contd)

Bit(s)	Name	Core Access	Reset Value	Description
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" ⁽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.

If the OPB Arbiter has been parameterized to only support fixed priority arbitration, the DPE bit is set to '0' and is read-only by the processor core. The DPERW bit is then set to '0' and reflects the fact that the OPB Arbiter only supports fixed priority arbitration. 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. Although the OPB Arbiter has been parameterized to support bus parking, bus parking can be disabled by negating the PEN bit. Also, if the OPB Arbiter has been parameterized to not include a processor or OPB slave interface, the Control Register is not accessible. As a result, the parking mode will be set to park on the last master which is the default value in the Control Register of the PMN bit. In order 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 set up appropriately.

Because the processor will require multiple bus cycles to update the masters' priority levels in the Priority Registers, there will exist some period of time in which the Priority Registers will not contain unique master IDs. Though the arbiter will still function properly in this circumstance, a particular master will not have a priority level associated with it and therefore will never receive a grant if it issues a request. Because this condition could cause the OPB to become unresponsive, 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 will use the IDs of the masters as their priority level until the PRV bit has been asserted.

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).

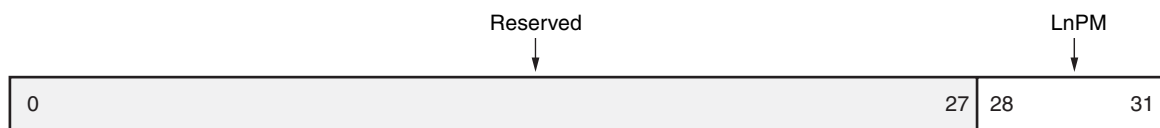
OPB Arbiter Priority Registers

Each Priority Register holds the master ID of the OPB master at that priority level as shown in [Figure 24](#) and described in [Table 6](#). The relative priority of each master is determined by the location of the ID of the register 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 will vary based on the number of masters support by the OPB Arbiter. The master ID will always be aligned to the LSB position within the register so that the master ID will appear as an integer when accessed by software.

Notes:

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



DS469_24
OPB Arbiter LVLn
Priority Register

Figure 24: OPB Arbiter OPB Arbiter LVLn Priority Register

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 ⁽¹⁾	Level n Priority Master ID. This field contains the ID of the master at level n priority. ⁽²⁾

Notes:

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

Each master ID must be contained in one of the Priority Registers, otherwise the request of that master will be ignored by the arbiter (because it has no priority value) and a grant to that master will never be 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. Therefore, 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. Therefore, 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 will insure that whenever the PRV bit is asserted, all master IDs have been assigned a priority. When the PRV bit is negated, the OPB arbiter will assign priority based on the master ID, i.e., Master 0 will have level 0 priority (highest), Master 1 will have level 1 priority, and Master n will have 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.

Design Implementation

Target Technology

The intended target technology is a Virtex-II FPGA.

Device Utilization and Performance Benchmarks

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

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

The OPB Arbiter benchmarks are shown in Table 7 for a Virtex-II -5 FPGA using multi-pass place and route.

Table 7: OPB Arbiter FPGA Performance and Resource Utilization Benchmarks (Virtex-II -5)

Parameter Values						Device Resources			f_{MAX} (MHz)
C_NUM_MASTERS	C_DYNAM_PRIORITY	C_PARK	C_PROC_INTRFCE	C_REG_GRANTS	C_OPB_DWIDTH / C_OPB_AWIDTH	Slices	Slice Flip- Flops	4-input LUTs	f_{MAX}
1	N/A	N/A	N/A	N/A	N/A	4	4	6	290
2	0	0	0	0	32/32	11	7	18	223
2	1	1	1	1	32/32	76	76	87	163

Table 7: OPB Arbiter FPGA Performance and Resource Utilization Benchmarks (Virtex-II -5) (Contd)

4	0	0	0	0	32/32	27	13	39	153
4	1	0	0	0	32/32	42	21	69	167
4	0	1	0	0	32/32	42	17	59	150
4	0	0	1	0	32/32	75	63	95	136
4	0	0	0	1	32/32	30	18	40	173
4	1	1	1	1	32/32	130	97	163	126
8	0	0	0	0	32/32	93	31	132	122
8	1	1	1	1	32/32	278	145	398	100
16	0	0	0	0	32/32	441	84	654	100
16	1	1	1	1	32/32	904	252	1477	82

Notes:

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

Specification Exceptions

Register Definitions and Addressing Exceptions

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 doesn't 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 therefore 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 Exceptions

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	SI_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 Exceptions

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 Exceptions

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 Exceptions

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.

Clock and Power Management Exceptions

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, therefore 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 Exceptions

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. Therefore, 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:

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

Revision History

Date	Version	Revision
12/9/04	1.0	Initial Xilinx release.
4/4/05	1.1	Updated for EDK 7.1.1 SP1; updated trademarks and supported family listing.
7/21/04	1.2	Corrected Allowable values for C_OPB_AWIDTH and C_OPB_DWIDTH
9/23/05	1.3	Converted to new DS template; restructured document layout; updated figures to Xilinx graphic standards.