PCI Arbiter (v1.00a)

Introduction

The PCI Arbiter provides arbitration for two to eight PCI master agents. Parametric selection determines the number of masters competing for PCI bus control. Arbitration follows a rotating scheme to provide fair access to the PCI bus for all master agents. Bus parking occurs when no master requests PCI control. The last active master – the most recent master to request, control, and relinquish the bus – is designated as the park master agent. When multiple masters request control of the PCI bus, the highest priority master request will be granted for a minimum of two PCI clock cycles, then the grant will be removed. This allows the granted agent to control the bus for the duration of its latency timer. Upon expiration of its latency timer, this agent must relinquish the bus and allow the PCI Arbiter to grant control to the next highest priority requesting master agent. Transitions between distinct grant signal assertions are always separated by one PCI clock cycle when all grant signals are deasserted.

Formerly known as the OPB_PCI_Arbiter, this new core does not provide an option for the inclusion of an OPB interface or programmable registers. Loss of the marginal utility of an OPB interface and the program registers is compensated by a more robust implementation with more clearly defined operations.

The PCI_Arbiter follows guidelines defined in the PCI Local Bus Specification, Version 3.0, February 3, 2004. The PCI_Arbiter is designed for operation in 33 and 66 MHz PCI systems.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Spartan®-3, Spartan-3E, Virtex®-4 and Virtex-5	
Version of core	pci_arbiter	v1.00a
Resources Used		
	Min	Max
LUTs	See Table 5 and Table 6	
FFs		
Block RAMs		
Provided with Core		
Documentation	Product Specification	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs & application notes	N/A	
Additional Items		
Design Tool Requirements		
Xilinx® Implementation Tools	ISE® v11.1 or later	
Verification	Mentor Graphics ModelSim v6.4b and above	
Simulation	Mentor Graphics ModelSim v6.4b and above	
Synthesis	XST 11.1 or later	
Support		
Provided by Xilinx, Inc.		

Features

- Variable number of PCI masters set by a parameter
- Rotating arbitration
- Bus parking
- Grant deasserts to invoke the granted master latency timer for cases of multiple requests
- Minimum two clock cycle grant assertion
- Grants deasserts one cycle between grant transitions
- The last active master is the default for bus parking

Functional Description

The top-level block diagram of the PCI Arbiter is shown in Figure 1.

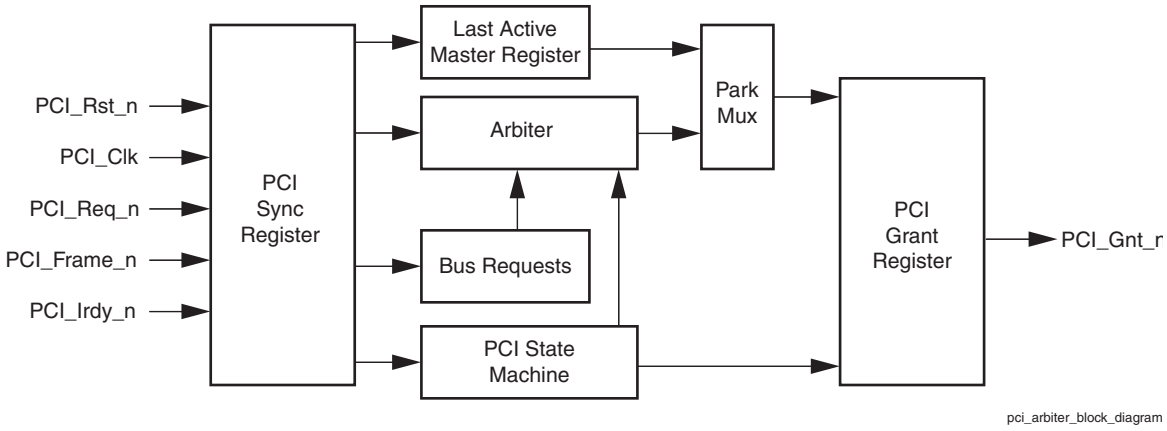

Figure 1: **PCI Arbiter Block Diagram**

PCI_Arbiter I/O Signals

The I/O signals for the PCI_Arbiter are listed in Table 1 and are shown in Figure 1, the PCI_Arbiter block diagram.

Table 1: **PCI_Arbitier I/O Signals**

Port	Signal Name	Interface	I/O	Description
PCI Interface				
P1	PCI_Clk	PCI	I	PCI Clock
P2	PCI_Rst_n	PCI	I	PCI Reset, active low
P3	PCI_Req_n[0: C_NUM_PCI_MSTRS - 1]	PCI	I	PCI Request, active low
P4	PCI_Gnt_n[0: C_NUM_PCI_MSTRS - 1]	PCI	O	PCI Grant, active low
P5	PCI_Frame_n	PCI	I	PCI Frame, active low
P6	PCI_Irdy_n	PCI	I	PCI Ready, active low

PCI_Arbiter Design Parameters

The PCI_Arbiter may be configured for particular systems by appropriately selecting features through parameter values. This allows the user to craft a design with the minimum necessary resources that operates at the best possible performance. The parameter features available in the Xilinx PCI_Arbiter are shown in Table 2.

Table 2: **PCI_Arbiter Design Parameters**

Generic	Feature Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Top Level					
G1	Device family	C_FAMILY	virtex4, virtex5, spartan3	virtex4	string
PCI Interface					
G2	Number of PCI Masters	C_NUM_PCI_MSTRS	2 - 8	4	integer

Allowable Parameter Combinations and Considerations

The allowed values for C_NUM_PCI_MSTRS are 2 through 8. The numbering of PCI Masters, Requests, and Grants range from 0 to C_NUM_PCI_MSTRS - 1. Therefore, the default of value of 4 provides 4 PCI master agents numbered 0 through 3. Any number of PCI master agents from 2 through 8 work in all of the FPGA family types listed in Table 2.

Parameter/Port Dependencies

The width of two PCI_Arbiter signals depend upon the selected value C_NUM_PCI_MSTRS as illustrated in Table 3.

Table 3: **Parameter/Port Dependencies**

Generic or Port	Name	Affects	Depends	Relationship Description
Design Parameters				
G2	C_NUM_PCI_MSTRS	P3, P4		C_NUM_PCI_MSTRS determines the number of PCI requests and grants.
I/O Signals				
P3	PCI_Req_n		G2	The number of PCI requests depends on C_NUM_PCI_MSTRS.
P4	PCI_Gnt_n		G2	The number of PCI grants depends on C_NUM_PCI_MSTRS.

Rotating Arbitration Scheme

The rotating arbitration scheme shifts the priority level for every PCI master following each bus arbitration cycle. Immediately after reset, PCI master zero has the highest priority level, level zero. From the highest priority, the priority level decreases as the PCI master number increases. Each arbitration cycle causes all of the priority levels to shift. The PCI master agent with the highest priority, zero, receives the lowest priority, C_NUM_PCI_MSTRS-1, while the remaining PCI master agents receive higher level priorities. Over time, this provides equal bus access for all PCI master agents. Table 4 illustrates the rotating arbitration scheme with eight masters, C_NUM_PCI_MSTRS = 8. Each column denotes the appropriate priority level for the PCI master in a given row and time is increasing from left to right. These priorities shift in this nature regardless of which master agent wins access to the bus. At any point when multiple masters agents compete for bus access, the competing master with the highest priority shall be granted access and all masters will receive new priority levels. Any agent failing to obtain bus access will receive progressively higher priority levels for each arbitration cycle and eventually obtain bus access.

Table 4: **Rotating Arbitration Scheme for 8 PCI Master Agents (C_NUM_PCI_MSTRS = 8).**

PCI Mstr #	Priority Level at Tn	Priority Level at Tn+1	Priority Level at Tn+2	Priority Level at Tn+3	Priority Level at Tn+4	Priority Level at Tn+5	Priority Level at Tn+6	Priority Level at Tn+7	Priority Level at Tn+8	Priority Level at Tn+9
0	0	7	6	5	4	3	2	1	0	7
1	1	0	7	6	5	4	3	2	1	0
2	2	1	0	7	6	5	4	3	2	1
3	3	2	1	0	7	6	5	4	3	2
4	4	3	2	1	0	7	6	5	4	3
5	5	4	3	2	1	0	7	6	5	4
6	6	5	4	3	2	1	0	7	6	5
7	7	6	5	4	3	2	1	0	7	6

Behavioral Waveforms

These waveforms detail the operation of the PCI_Arbiter and depict the sequence of events for various scenarios as PCI master agents compete for access to the PCI bus. They provide a consistent description for the development of documentation, design, simulation, verification, and tests.

The first case, illustrated in Figure 2, represents the most simple operation for the PCI_Arbiter. In this case, the PCI bus starts from an idle situation with the PCI_Arbiter providing grant to the last active PCI master, agent number four in this example, for bus parking. The last active master is defined as the most recent PCI master agent that correctly responded to grant by appropriately asserting PCI_Frame_n for a bus transaction. The PCI master agent connected to PCI_Req_n(3) then requests access to the PCI bus. After two PCI_Clk cycles, the PCI_Arbiter grants access by asserting PCI_Gnt_n(3). Master agent number three responds with a bus transaction, initiated by asserting of Frame_n and Irdy_n and terminated by deasserting PCI_Frame_n and PCI_Irdy_n. Notice that PCI_Req_n(3) deasserts in the same clock cycle that PCI_Frame_n asserts. This is the minimum duration for a master to assert a request and be granted the bus. Master agent three becomes the last active master and the PCI_Arbiter parks on this agent while the PCI bus goes idle. This is indicated by the continued assertion of PCI_Gnt_n(3).

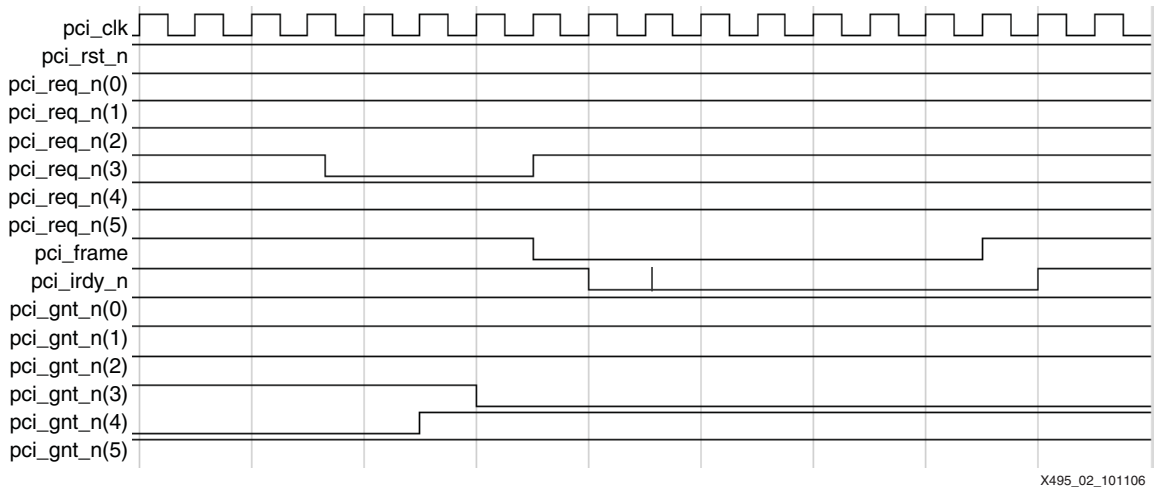

Figure 2: **Simple, Single Request and Grant**

Figure 3 shows the PCI_Arbiter operation when a requesting PCI master agent fails to respond. Once grant is asserted, the requesting PCI master agent must assert PCI_Frame_n within 16 clock cycles to obtain control of the bus. Failing this, the requesting PCI master agent will lose its bus privilege and cannot be recorded as the last active master. In this case master agent number four requests the bus by asserting PCI_Req_n(4) and fails to assert PCI_Frame_n within its window of opportunity. This causes the PCI_Arbiter to remove the PCI_Gnt_n(4), retain the prior last active master, and return to park mode. If PCI_Req_n(4) remained asserted, the PCI_Arbiter would continue asserting PCI_Gnt_n(4) for 16 clock cycles for another time-out unless a different master agent requests the bus. This behavior arises from the uncertainty of the PCI_Arbiter "knowing" if a request is valid, as all requests must be treated as valid. To prevent this behavior, master agents must respond to grant.

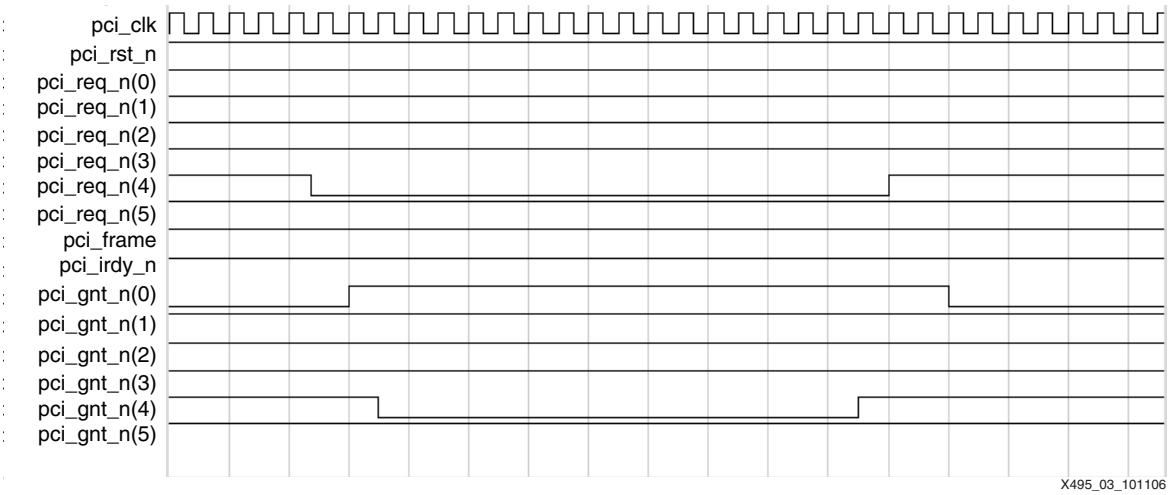

Figure 3: **Grant Removed After 18 Clocks if Master Does Not Respond**

The PCI Specification allows a requesting PCI master to assert PCI_Frame_n in the same clock cycle that the PCI_Arbiter is removing PCI_Gnt_n. This is defined as a valid response by the requesting master and the PCI Arbiter must accommodate this transaction. This requesting master would then be recorded as the last active master. Figure 4 illustrates this case by the deassertion of PCI_Gnt_n(3) just as PCI_Frame_n is asserted. The PCI_Arbiter reasserts PCI_Gnt_n(3) to acknowledge and allow the bus transaction, then records master agent number three as the last active master as seen in the park mode when the PCI bus goes idle.

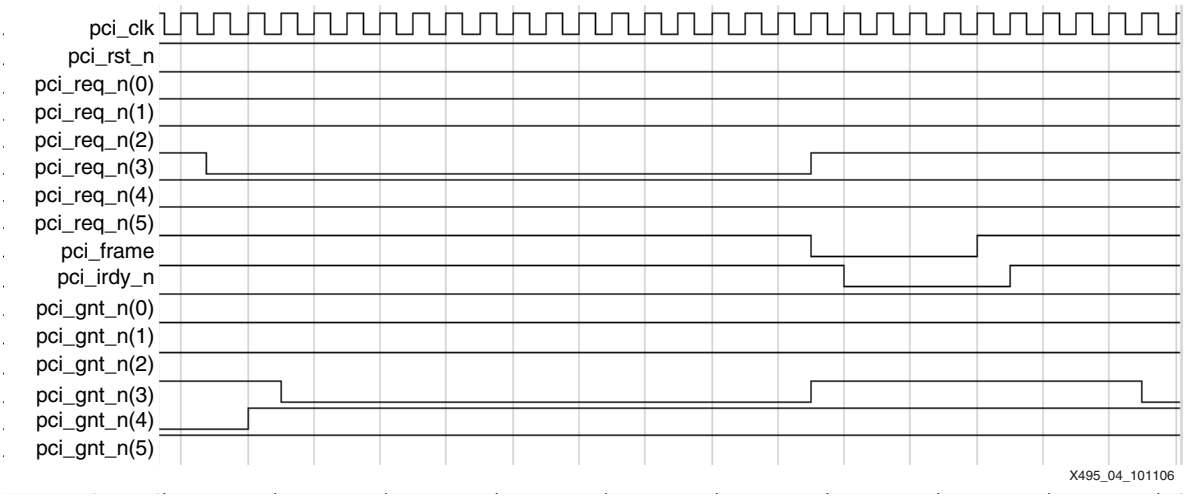

Figure 4: **PCI_Frame_n Response in the Clock Cycle That PCI_Gnt_n is Removed**

Figure 5 shows that the last active master, while in park mode, may request and be granted control of the PCI bus. In this case, the PCI_Arbiter must recognize and grant the request. This PCI master will retain its status as the last active master. Figure 5 starts with master agent number one as the park master but transitions the park master to agent number two following the request and transaction of master agent number two. The last transaction illustrates the park master requesting and obtaining PCI bus access.

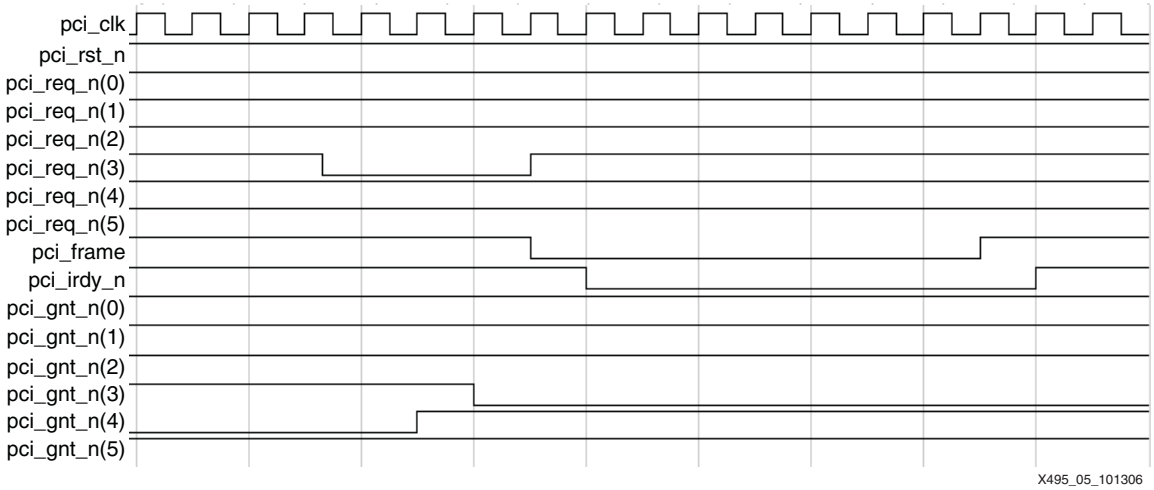

Figure 5: **PCI_Arbiter Response to Request From the Park Master**

The PCI Specification allows a PCI master to start a transaction when its grant is asserted and the bus is idle, without regard to the state of its request signal. That is when both Frame_n and Irdy_n are deasserted the park master agent may initiate a bus transaction without asserting request, Req_n. The PCI Arbiter must recognize and allow this situation. In this case, this master would maintain its status as the last active master. This scenario is shown in Figure 6 with master agent number three as the park master.

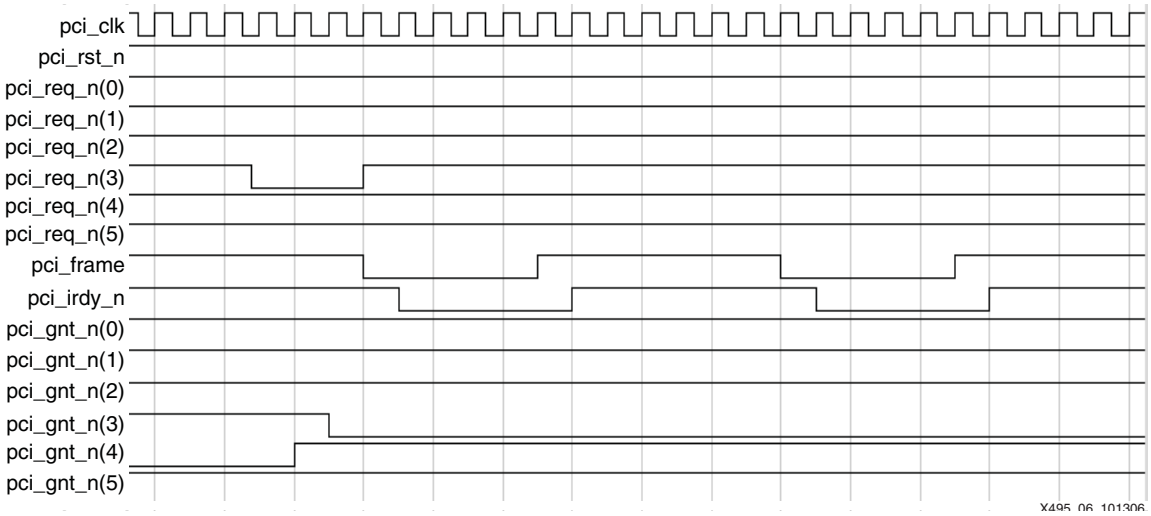

Figure 6: **Park Master Controlling the Bus Without First Requesting Access**

Figure 7 shows the PCI_Arbiter operation for multiple requests. In this case, the park master is number zero and all six connected master agents request access simultaneously. The PCI_Arbiter sequentially grants access to agents zero through five. As master agent five is the last active master, it completes this example as the park master.

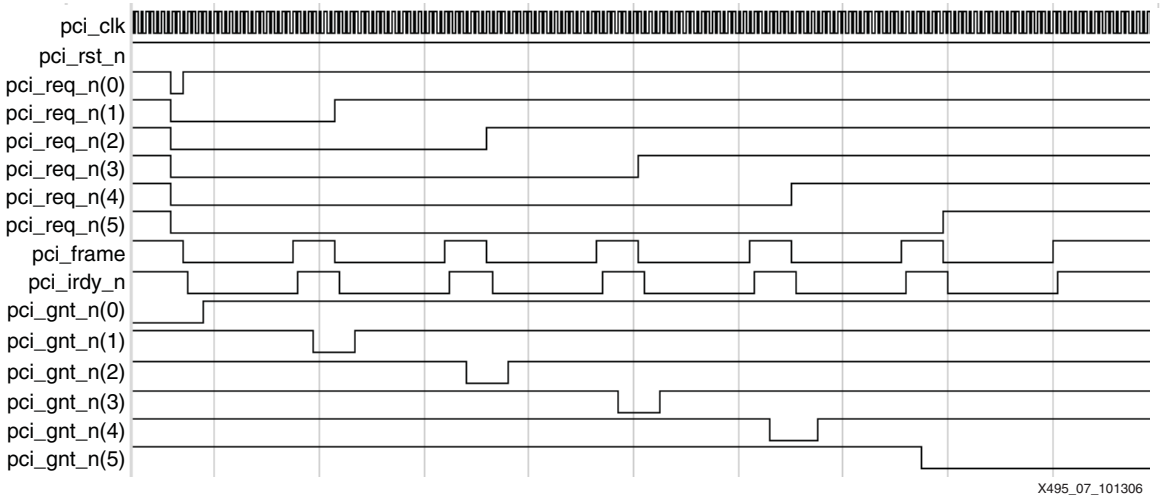

Figure 7: **PCI_Arbiter Response to Multiple Requests**

Figure 8 is another example of the PCI_Arbiter operation for multiple requests. Initially the park master is number two and all six connected master agents requests access simultaneously. The PCI_Arbiter sequentially grants access to agents three through five then provides access to master agents zero, one and two in order. As master agent two is the last active master, it completes this example as the park master.

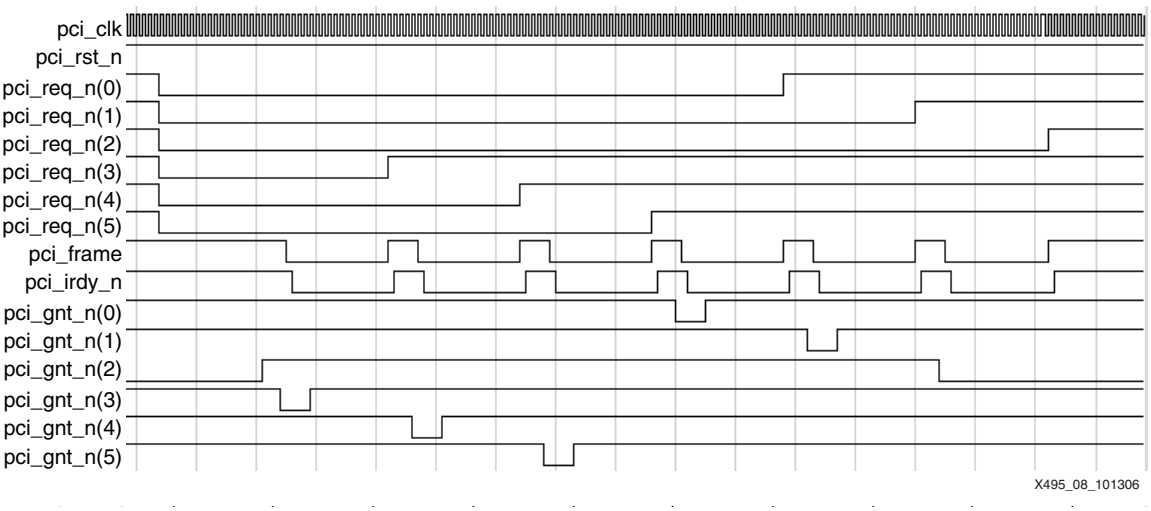

Figure 8: **PCI_Arbiter Response to Multiple Requests**

Figure 9 shows the situation of a master agent, number two, requesting access, as another agent, number three also requests access. After agent number two fails to respond, the PCI_Arbiter grants access to the second requesting agent, number three. Agent number three then becomes the last active master and the park master.

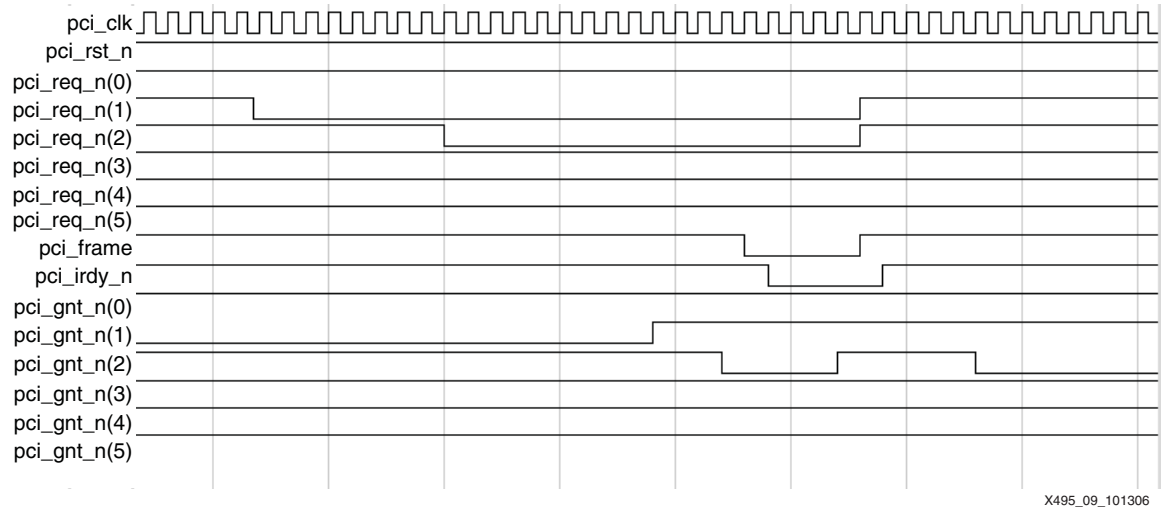

Figure 9: **PCI_Arbiter Grant to a Second Master After First Fails to Respond**

Figure 10 shows the situation of a master agent, number one, requesting access, as another agent, number two also requests access. After agent number one fails to respond and deasserts request, the PCI_Arbiter grants access to the second requesting agent, number two. Agent number two then becomes the last active master and the park master

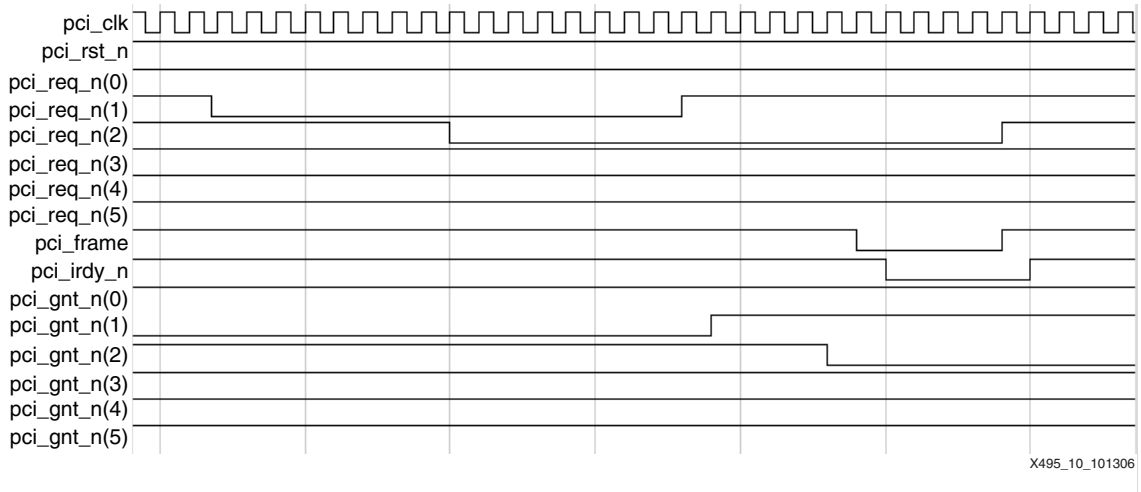

Figure 10: **PCI_Arbiter Grant to a Second Master After First Fails to Respond and Goes Away**

Figure 11 shows the situation of a master agent, number zero, requesting and gaining access, then going away. During the bus transaction for master agent zero, a second master, number one, requests access. The PCI Arbiter detects this as a multiple request situation even though the first request expired. The PCI Arbiter responds by deasserting grant zero forcing the master agent zero's latency timer to activate and eventually terminate the bus transaction for master zero. Following this, the PCI Arbiter grants access to master agent one which then proceeds with its own transaction. In this case master one is the last active master and becomes the parking agent of choice for the PCI Arbiter.

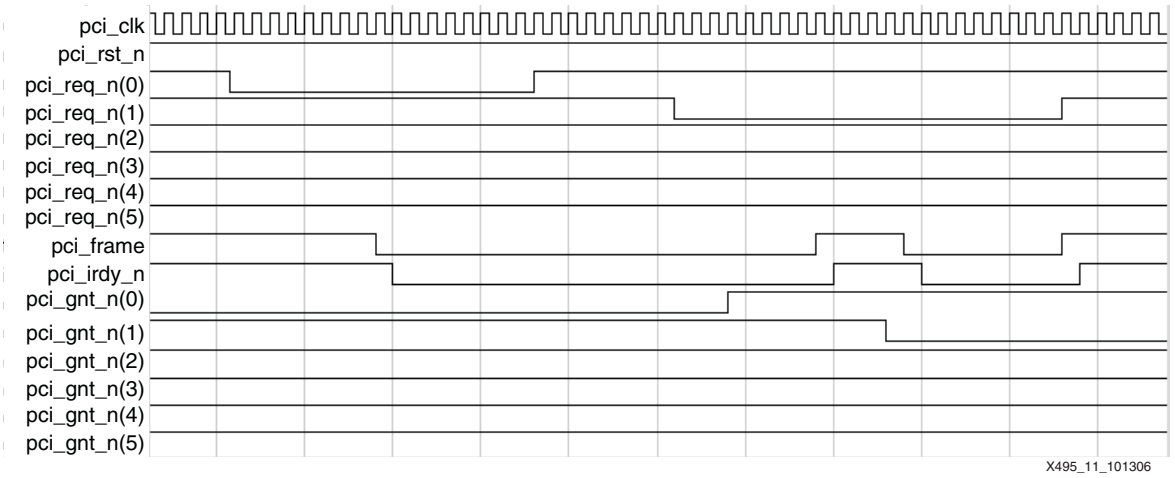

Figure 11: **PCI_Arbiter Detects Multiple Request Even After First Request is Removed**

Design Implementation

Design Tools

The PCI_Arbiter was designed and implemented using VHDL. Logic simulations, both functional and post implementation verification, were performed with Model Technology ModelSim.

Target Technologies

The intended target technologies are Xilinx Spartan-3, Virtex-4 and Virtex-5 FPGAs.

Device Utilization and Timing

Because the PCI_Arbiter will be used with other design modules in the FPGA, the utilization and timing numbers reported in this section are estimates only. As the PCI_Arbiter is combined with other portions of the FPGA design, the utilization of FPGA resources and timing of the PCI_Arbiter design will vary from the results shown in Table 5 and in Table 6.

The PCI_Arbiter resource utilization measured with the Virtex-4 FPGA as the target device are detailed in Table 5.

Table 5: **Performance and Resource Utilization Benchmarks on Virtex-4 FPGA (xc4vlx60-11-ff1148)**

Sr. No.	Parameter Values	Device Resources			$f_{MAX(MHz)}$
	C_NUM_PCI_MSTRS	Slices	Slice Flip - Flops	4 - Input LUTs	
1	2	44	39	68	144.76
2	3	57	50	81	114.68
3	4	63	57	89	126.31
4	5	85	74	111	105.33
5	6	104	83	143	100.25
6	7	111	92	147	100.47
7	8	129	100	165	100.01

The PCI_Arbiter resource utilization measured with the Virtex-5 FPGA as the target device are detailed in Table 6.

Table 6: **Performance and Resource Utilization Benchmarks on Virtex-5 FPGA (xc5vlx50-2-ff1153)**

Sr. No.	Parameter Values	Device Resources		$f_{MAX(MHz)}$
	C_NUM_PCI_MSTRS	Slice Flip - Flops	LUTs	
1	2	14	53	177.49
2	3	24	65	168.92
3	4	30	72	159.67
4	5	46	99	120.24
5	6	54	104	133.62
6	7	62	123	114.13
7	8	70	122	121.07

Reference Documents

The following document contains reference information important to understanding the PCI_Arbiter:

- *PCI Local Bus Specification*, Revision 3.0, February 03, 2004.

Revision History

Date	Version	Revision
11/12/04	1.0	Initial Xilinx release
7/14/06	1.1	Updated for Virtex-5 support
10/16/06	1.2	Converted to new DS template; images updated to graphic standards
04/24/09	1.3	Updated tool support and legal information.

Notice of Disclaimer