PCI Arbiter Core

**Features**
- Support for up to Five PCI Bus Masters
- Support for Two Arbitration Schemes
  - Pure Rotation
  - Fair Rotation
- Support for Bus Parking
- Hidden Bus Arbitration
- Interface with 33 MHz and 66 MHz PCI Systems
- Implementation in Actel’s SX, SX-A, ProASIC and ProASICPLUS Families
- Synthesizable VHDL Source Code
- Device Utilization
  - SX/SX-A 100-150 Modules
  - ProASIC/ProASICPLUS 124-135 Tiles

**General Description**
The Arbiter core is used to efficiently manage access to a PCI bus that is shared by several masters. Access to the PCI bus is automatically determined by the individual priorities of each master.

In the most common application, customers use an embedded processor as the master with highest priority and a pure-rotation arbitration scheme among other PCI bus masters. The networking and telecom markets are the targets for this macro.

**Implementation**
At any given time, more than one PCI bus initiator (Master) device may request use of the PCI bus by asserting its specific request signal (REQn). The Arbiter determines which PCI bus initiator controls the PCI bus by asserting that device’s specific grant signal (GNTn). Figure 1 shows the PCI Arbiter Core interface signals and Figure 2 illustrates the relationship of the PCI bus initiator devices with the Arbiter.

**Figure 1 • PCI Arbiter Core Interface Signals**

**Figure 2 • Top-Level Interface of the PCI Bus Initiators with the Arbiter**
**Arbitration Schemes**

Pure rotation is a turn-based method that allows each bus master one transaction in turn if multiple masters are requesting the bus simultaneously. If only one master requests the bus, that master will immediately get the grant. **Figure 3** illustrates the pure rotation scheme.

Fair rotation is employed if an embedded processor is required to initialize the system but will not be used after that point. By giving it highest priority, the fair rotation scheme allows the embedded processor access to the PCI bus on every other transaction when it is requesting the bus continuously. The other masters use a pure rotation scheme. The fair rotation scheme is illustrated in **Figure 4**.

**Figure 3** • Pure Rotation Arbitration

**Figure 4** • Fair Rotation Arbitration
**Bus Parking**

The arbiter has been designed to implement bus parking; i.e., it asserts the grant to a default device when none of the request lines are active (there are no devices are requesting the bus). This ensures that a requesting device will received an almost immediate grant. The default case allows the bus to be parked on the device that last acquired the bus. However, by utilizing the constants BUS_PARK, BUS_DEVICE, and BUS_GNTN the user can specify the bus is parked on a device other than the default.

**Hidden Bus Arbitration**

The PCI specification allows bus arbitration to take place while the currently granted device is performing a data transfer. This feature greatly reduces arbitration overhead and improves bus utilization.

**Maximum Latency**

The device granted the bus must initiate a transaction (drop FRAMEn signal) within 16 PCI clock cycles. If the time expires and the device has not initiated a transaction, the arbiter removes the grant from the device and the bus to the device with the next highest priority.

**Behavioral Simulation**

The following procedures is used to simulate the VHDL version of the Arbiter Macro:

### Simulating the Pure Rotation Arbiter Scheme

1. Invoke the V-System simulator.
2. Change to the “/vhdl/tbench/mti_arb” directory.
3. Create a “work” library.
   Type the following command at the prompt:
   ```
   vlib work
   ```
   The test bench will be compiled into this directory.
4. Open the file “arb_wrp.vhd” in the “/vhdl/src” directory and set the constant ALGORITHM to a one.
5. Compile the macro and the test bench.
   Type the following command at the prompt:
   ```
   do compall.do
   ```
6. Simulate the test bench.
   Type the following command at the prompt:
   ```
   do run_fair.do
   ```

### Simulating the Fair Rotation Arbiter Scheme

1. Invoke the V-System simulator.
2. Change to the “/vhdl/tbench/mti_arb” directory.
3. Create a “work” library.
   Type the following command at the prompt:
   ```
   vlib work
   ```
   The test bench will be compiled into this directory.
4. Open the file “arb_wrp.vhd” in the “/vhdl/src” directory and set the constant ALGORITHM to a zero.
5. Compile the macro and the test bench.
   Type the following command at the prompt:
   ```
   do compall.do
   ```
6. Simulate the test bench.
   Type the following command at the prompt:
   ```
   do run_pure.do
   ```

**User Code Customization**

The code is currently written to arbit between 5 master devices. The devices are implemented as states in a finite state machine (FSM). The end user can modify the implementation by adding or subtracting states to meet the number of devices required by the particular application.

**Estimated Performance and Device Utilization**

The expected performance and utilization statistics for the 66 MHz PCI Arbiter using the 54SX16-2 and APA750 devices are shown in Table 1. These numbers are based on post layout results using automatic place and route.

![Table 1](image)

**Table 1 • Utilization and Performance Statistics**

<table>
<thead>
<tr>
<th></th>
<th>Utilization (Estimated)</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Combinatorial Modules</td>
<td>Sequential</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modules</td>
</tr>
<tr>
<td>A54SX16-2</td>
<td>84</td>
<td>32</td>
</tr>
<tr>
<td>APA750</td>
<td>135</td>
<td>N/A</td>
</tr>
</tbody>
</table>