CSELT S.p.A

Via G. Reiss Romoli, 274
I-10148 Torino, Italy
Phone: +39 011 228 7165
Fax: +39 011 228 7003
E-mail: viplibrary@cselt.it
URL: www.cselt.it

Features

- Supports Spartan, Spartan™-II, Virtex™, and Virtex™-E devices
- The Arbiter core controls the access of different masters to a shared resource. The arbitration algorithm is based on request priorities and on request arrival times
- Request priorities can change dynamically
- Two priority classes supported (strong priority/weak priority)
- Strong priority requests can interrupt any lower priority request being serviced
- Weak priority requests cannot interrupt any other request being serviced
- Access counters available to evaluate the number of resource accesses accomplished by a master
- Customizable VHDL source code available, allowing generation of different netlist versions
- Customized testbench for pre- and post-synthesis verification supplied with the module
- Core customization:
  - Number of masters requesting access to shared resource
  - Number of request priorities
  - Number of strong priorities
  - Number of bits for each access counter

Applications

General purpose bus arbitration

AllianceCORE™ Facts

Core Specifics[1]		
Supported Family	Spartan	Virtex
Device Tested	S10-3	V50-6
CLBs[2]	154	182
Clock IOBs	1	1
IOBs[3]	20	20
Performance (MHz)	16	33
Xilinx Tools	M1.5i/M2.1i	M1.5i/M2.1i
Special Features	None	None
Provided with Core		
Documentation		User Manual
Design File Formats		EDIF netlist, XNF netlist, VHDL source available extra
Constraints File		TOP_ARBITER_nl.ncf
Verification		VHDL testbench
Instantiation Templates		VHDL, Verilog
Reference Designs & Application Notes		None
Additional Items		None
Simulation Tool Used		
Synopsys VSS		
Support		
Design and customization support provided by CSELT		

Notes:
1. Data refer to the following customisation:
   - 3 masters
   - 3 request priorities
   - 2 strong priorities, 1 weak priority
   - 4-bit access counters
2. Utilization numbers for Virtex are in CLB slices
3. Assuming all core I/Os are routed off-chip

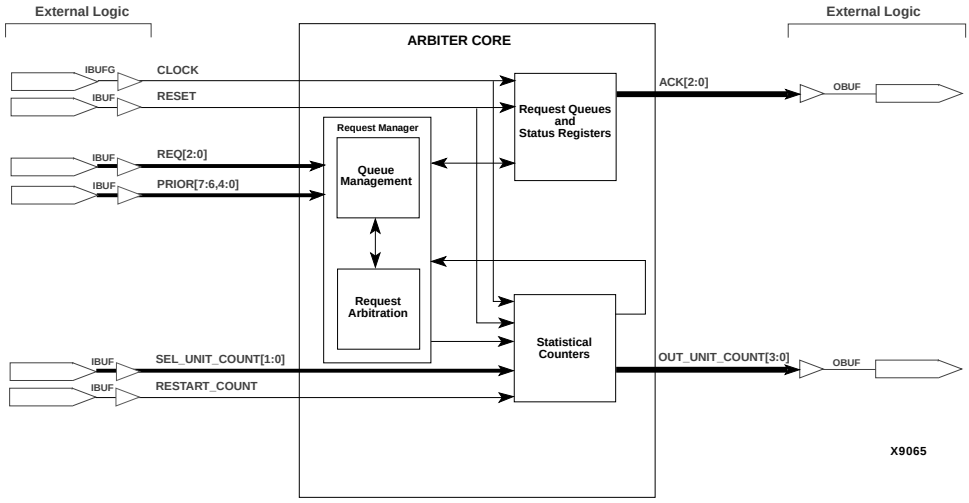

Figure 1: Arbiter Block Diagram

General Description

The Arbiter core manages the access to a resource shared by different masters in a fast and flexible way.

Communication between the Arbiter core and the resource masters is based on a request-acknowledge: each master asking for access to the arbitrated resource sends a request with associated priority information to the Arbiter core. Arbiter grants access to a specific master by setting the corresponding acknowledge output.

Priorities can be divided into two classes, strong priorities and weak priorities. The difference between these two classes lies in the capability of interrupting the service of lower priority requests. Strong priority requests can, and do, interrupt lower priority requests, while weak priority requests cannot. The threshold between strong and weak priorities is customizable.

For statistical purposes, the Arbiter can count how many times a specific master has been granted access to the arbitrated resource, independently from the number of interruptions during each access.

The flexibility of Arbiter is shown, for example, when the Unit set consists of a CPU and other units, all sharing the same bus: assigning Strong priority to CPU and Weak Priorities to other units, the bus arbitration is optimized.

The Absolute Maximum ratings, Operating Conditions, DC Electrical Specifications and Capacitances depend on the Xilinx device selected for implementation and can be retrieved from the corresponding Xilinx datasheet.

Functional Description

The internal architecture of the Arbiter core is shown in Figure 1. A brief description of the operation of each module follows.

Requests Manager

The Requests Manager Block receives as input Requests and Priorities from all the connected masters (REQ and PRIOR signals) and elaborates these signals in two phases.

The first phase is Queue Management: each request is checked and sorted according to its priority. The priority value is read only when the request becomes active or inactive and it should be stable throughout the time the request is presented to the Arbiter (REQ signal set to ‘1’).

The Queue Management block sorts and updates the active requests in the priority queues (insertion of new pending requests and deletion of the inactive ones) and accordingly updates the core status registers.

The second phase is Request Arbitration: it implements the algorithm that manages the active Requests according to their priority first, then to their arrival time and, finally, to a static priority order depending on the request port number.

The Request Arbitration reads the priority queues, updates the core status registers, and selects the active unit; it also properly manages interruptions to have a fast and safe access to the arbitrated resource.

Queues and Status Registers

The Queues and Status Registers Block is the process that allows to read and write the registers (Request Queues and

Status Registers) used by the Arbiter to accomplish its operations.

The Request Queues store Active and Pending Requests, sorted by priority and arrival time. The queues are implemented using a set of register FIFOs, accessible for read and write from the Request Manager in a single clock cycle.

The Queue Status Registers provide, the following information for all the Priority Queues:

- The Start pointers, used to select the first data to read from the FIFO;
- The End pointers, used to select the FIFO register to write data in;
- Flags, used to discriminate an empty queue from a queue full of data.;
- The Pending Request Status, that defines the units that have sent a Request signals, and detects a transition of their Request signal from Active to Inactive;
- The Weak-Priority Request Status, that keeps all the information used by the Arbiter to manage interruptions of weak-priority requests;
- The Acknowledge Status, used to set Active at any time no more than one Acknowledge signal, and never before a minimum delay of 1 clock cycle, starting from any active input request (to avoid conflicts on the shared resource).

Statistical counters

This block can be used for test or statistical analysis of the Arbiter behavior; it consists of a set of counters, each of which is used to keep track of the number of times a unit has been successfully served, independently from transitory interruptions.

Access to the counters is available at any time and does not affect the other Arbiter operations.

Counters have a size defined from the C_COUNTER_BITS generic; when this value is set to zero, the counters disappear from the Arbiter.

A synchronous reset, active high, can be used to reset all the counters to zero.

Pinout

The pinout of this core has not been fixed to a specific FPGA I/O allowing flexibility with a user's application. Signal names are shown in the block diagram in Figure 1 and described in Table 1.

Core Modifications

CSELT provides netlist customized to user's requirements. The Arbiter core source code is parametric. Parameters shown in Table 2 are implemented as a set of generics in the synthesizable VHDL source code of the core. Parameters allow the user to specify some architectural and functional features of the synthesized core netlist, so as to adapt it to a specific design or application.

Table 1: Core Signal Pinout

Signal	Signal Direction	Description
CLOCK	Input	Master clock
RESET	Input	Asynchronous reset
REQ[2:0]	Input	Resource access request; vector range depends on the N_UNIT generic
PRIOR[7:6,4:0]	Input	Priority of the current access request; vector range depends on the N_UNIT generic
SEL_UNIT_ COUNT[1:0]	Input	Access counter selector; vector range depends on the N_UNIT generic
RESTART_COUNT	Input	Synchronous reset of all the access counters
ACK[2:0]	Output	Resource access grant; vector range depends on the N_UNIT generic
OUT_UNIT_ COUNT[3:0]	Output	Access counter value; vector range depends on the C_COUNER_BITS generic

Table 2: Core Parameters (VHDL Generics)

Parameter	Description
N_UNIT	Number of masters accessing the arbitrated resource.
P_PRIORITY	Number of priority values that each request can assume.
S_STRONG	Number of strong priorities.
C_COUNTER _BITS	Number of bits of each resource access counter.

Verification Methods

Extensive functional simulation has been performed for different values of the core parameters, using the Synopsys VSS simulator. Simulation scenarios (including data and command files) and parametric test bench used for design verification are provided with the core.

The parametric test bench is composed of a parametric test vector generator and the Arbiter core. The generated signals test the request-acknowledge protocol using regular patterns to check the correctness of the Arbiter behavior. The time management and the Strong/Weak priority queuing can be easily verified.

Recommended Design Experience

Experience with the Xilinx design flow and logic system design is recommended to the users of the netlist version of the core. For the source code version, users should also be familiar with the Synopsys FPGA synthesis tools (VHDL Compiler, FPGA Compiler) and simulator (VSS).

Ordering Information

The Arbiter core is provided under license by CSELT S.p.A. for use in Xilinx programmable logic devices. Please contact CSELT S.p.A. for information about pricing, terms and conditions of sale.

CSELT S.p.A. reserves the right to change any specification detailed in this document at any time without notice, and assumes no responsibility for any error in this document.

All trademarks, registered trademarks, or servicemarks are property of their respective owners.

Related Information

Xilinx Programmable Logic

For information on Xilinx programmable logic or development system software, contact your local Xilinx sales office, or:

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
Phone: +1 408-559-7778
Fax: +1 408-559-7114
URL: www.xilinx.com

For general Xilinx literature, contact:

Phone: +1 800-231-3386 (inside the US)
+1 408-879-5017 (outside the US)
E-mail: literature@xilinx.com

For AllianceCORE™ specific information, contact:

Phone: +1 408-879-5381
E-mail: alliancecore@xilinx.com
URL: www.xilinx.com/products/logicore/alliance/tblpart.htm

Arbiter
Implementation Request Form

To: CSELT S.p.A.

FAX: +39 011 228 7003

E-mail: viplibrary@cselt.it

CSELT configures and ships Xilinx netlist versions of the Arbiter core customized to your specification. Please fill out and fax this form so that CSELT can respond with an appropriate quotation that includes performance and density metrics for the target Xilinx FPGA.

From: _____

Company:_____

Address:_____

City,State,Zip:_____

Country:_____

Phone:_____

FAX:_____

E-mail:_____

Implementation Issues

1. Number of Masters (2 to 16): _____

2. Number of Priorities (1 to 8)? _____

3. Number of Strong Priorities? _____

4. Statistical Counters Width (0 to 16)? _____

Business Issues

1. Indicate timescales of requirement:
_____ date for decision
_____ date for placing order
_____ date of delivery

2. Indicate your area of responsibility:
_____ decision maker
_____ budget holder
_____ recommender

3. Has a budget been allocated for the purchase?
Yes _____ No _____

4. What volume do you expect to ship of the product that will use this core? _____

5. What major factors will influence your decision?
_____ cost
_____ customization
_____ testing
_____ implementation size

6. Are you considering any other solutions? _____