

## Key Design Features

- Synthesizable, technology independent VHDL Core
- Supports up to 32 independent read/write ports
- Single memory interface port
- Configurable data and address widths
- Configurable read/write burst size
- Continuous operation between port swaps
- Choice of Round-robin or Fixed-priority arbitration schemes
- Page-break optimization mode reduces the incidence of memory page-breaks and enhances performance
- Master override allows the current master to hold the bus
- Suitable for interfacing to all synchronous memory types including SDRAM, DDR and DDR2

## Applications

- Processor architectures
- Synchronous memory designs
- Any application where the memory bandwidth must be shared efficiently between master devices

## Generic Parameters

Generic name	Description	Type	Valid range
arb_mode	Arbitration mode	integer	0 = Round-robin 1 = Fixed-priority
opt_mode_pb	Enable/Disable Page-break optimization	boolean	True, False
num_ports	Number of ports	integer	2, 4, 8, 16 or 32
num_ports_log2	Log2 no. of ports	integer	Log2 (num_ports)
addr_width	Memory address width	integer	≥ 2
word_width	Memory word width	integer	≥ 2
burst_size	Maximum size of read/write burst (in words)	integer	1 to 256
fifo_depth	Bypass FIFO depth	integer	≥ 4
fifo_depth_log2	Log2 FIFO depth	integer	Log2 (fifo_depth)

## Block Diagram

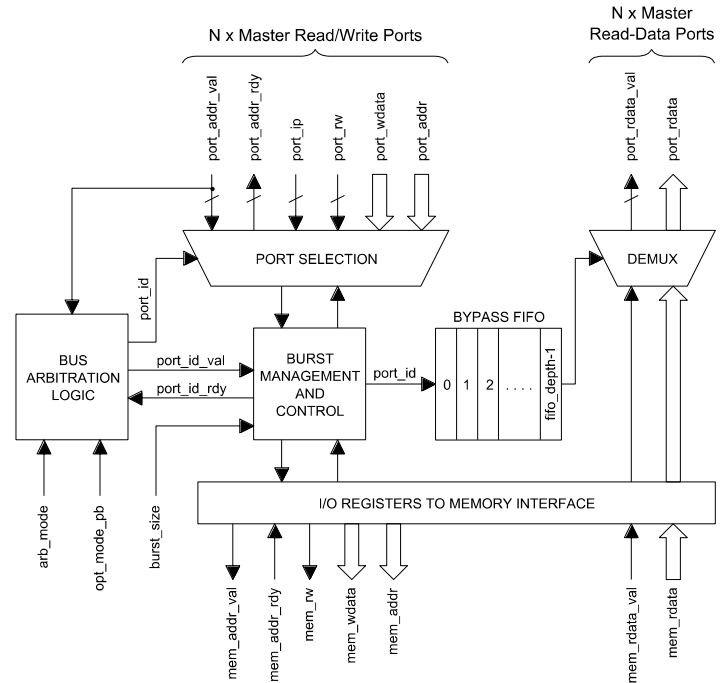


Figure 1: Multi-ported Memory Controller Architecture

## Pin-out Description

### MASTER INTERFACE

Pin name	I/O	Description	Active state
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
port_ip [num_ports - 1 : 0]	in	Address 'in-page' flag / Master override	high
port_rw [num_ports - 1 : 0]	in	Master read/write flag	write low read high
port_wdata [num_ports * word_width - 1 : 0]	in	Master write data	data
port_addr [num_ports * addr_width - 1 : 0]	in	Master address	address
port_addr_val [num_ports - 1 : 0]	in	Indicates that the Master has a valid request	high
port_addr_rdy [num_ports - 1 : 0]	out	Ready handshake signal grants access to the Master	high
port_rdata [word_width - 1 : 0]	out	Master read data	high
port_rdata_val [num_ports - 1 : 0]	out	Read data valid signal for the respective Master	high

**MEMORY INTERFACE**

Pin name	I/O	Description	Active state
mem_rw	out	Memory read/write flag	write low read high
mem_wdata [word_width - 1 : 0]	out	Memory write data	data
mem_addr [addr_width - 1 : 0]	out	Memory address	address
mem_addr_val	out	Indicates a valid memory request	high
mem_addr_rdy	in	Memory ready handshake	high
mem_rdata [word_width - 1 : 0]	in	Memory read data	high
mem_rdata_val	in	Memory read data valid signal	high

**General Description**

MEM\_ARBITER is a multi-ported memory controller that allows up to 32 master devices to share a single memory port. The controller uses a simple valid-ready protocol to request and grant access to the individual bus masters.

A choice of either round-robin or fixed-priority arbitration schemes is available with additional functionality to minimize the frequency of memory page-breaks. Memory accesses are granted in unique bursts - either read or write or both. Once a Master has completed a burst then access is given to the next Master on the bus. The overall objective is to share the memory bandwidth as efficiently as possible between contending devices.

In addition to the fixed and round-robin arbitration modes, an individual Master has the capacity to override normal port arbitration and hold the bus for consecutive bursts if required. This can be useful to allow high priority devices to hold the bus for longer periods without breaking the general arbitration scheme.

The interface with memory is a single read/write port. All data transfers to memory use the valid-ready protocol, meaning that a transfer occurs on the rising edge of *clk* when *mem\_addr\_val* and *mem\_addr\_rdy* are both active high. The read data return path has a single valid flag with no ready handshake.

Master inputs are packed into continuous bit-vectors with Master device '0' positioned at the least significant position, Master '1' at position 1 etc. up to the maximum number of devices defined in *num\_ports*. As an example, Master '0' valid/ready inputs would be signals *port\_addr\_val[0]* and *port\_addr\_rdy[0]* respectively. Master '1' valid/ready inputs would be *port\_addr\_val[1]* and *port\_addr\_rdy[1]*. As a further example, consider a controller configuration with a 16-bit word width. Master '0' write data would be specified in the signal bits *port\_wdata[15 downto 0]* and Master '1' write data would be specified in *port\_wdata[31 downto 16]*.

**Bus Arbitration Unit**

The arbitration unit decides the order in which the master ports are serviced. At the end of a burst request, the burst management unit requests a new port id from the arbitration unit which, in turn, is used to multiplex the next master request into the controller.

The type of arbitration algorithm used is determined by the generic parameter *arb\_mode*. Setting this parameter to '0' will select a round-robin algorithm while setting it to '1' will use a fixed-priority scheme.

**Round-Robin mode**

Round-robin arbitration services each *valid* port in numerical order. For instance if Masters 2, 3, 5 and 7 are valid and the current Master being serviced is 3, then the next Master to be serviced will be 5, then 7, then 2 etc. Round-robin is probably the fairest scheme as it guarantees an equal share of the memory bandwidth between all valid Masters. The maximum time a Master could be expected to wait (in system clock cycles) before bus access is granted is determined by the formula:

$$\text{Waiting time} = (\text{num ports} - 1) * \text{burstsize}$$

For example, for a controller with 8 ports and a burst size of 8 words then the longest time a Master would be expected to wait for bus access would be  $7 * 8 = 56$  system clock cycles<sup>1</sup>.

**Fixed-priority mode**

Fixed-priority arbitration gives absolute priority to Master '0' followed by Master '1', then Master '2' etc. This arbitration mode is particularly suited to system architectures where there is a strict hierarchy in the Master devices accessing the bus. Careful system design must be employed when using fixed-priority arbitration as it may lead to some of the lower priority ports being 'starved' of bandwidth.

**Page-break optimization (or Master override) flag**

Page-break optimization mode may be enabled by setting the generic parameter *opt\_mode\_pb* = true. When enabled the arbitration scheme takes into account the state of the *port\_ip* input flags. When a master device asserts this flag high with its corresponding *port\_addr\_val* signal, the arbiter will assume that the next burst of addresses will be 'in-page'.

A page is normally defined as an open row in an SDRAM memory architecture. Asserting the *port\_ip* flag will force the arbiter to stay with the current Master and in doing so, it will be able to take advantage of the currently open page. In practice, this is much more efficient than opening, closing, then re-opening the same page when multiple Masters are in contention.

Obviously, the *port\_ip* flag must be used sensibly, as over-use may also lead to the starvation of other Masters. As with fixed-priority schemes, use of the *port\_ip* flag will require careful modelling an testing to ensure port starvation does not occur.

In addition, the *port\_ip* flag may be used as a general master override flag for high-priority accesses (irrespective of whether the address is in page or not). If one or two Masters have higher priority, then they can use this flag to force preference on the bus. In some cases, a combination of round-robin and the use of the *port\_ip* flag may be a better compromise than a full fixed-priority scheme.

**Bypass FIFO**

The bypass FIFO buffers the read port id's between the front-end and back-end of the of the controller. The id's are used to de-multiplex the returning read data from the external memory interface and present it to the correct Master. It's important that the depth of the bypass FIFO is sufficient to hide the latency of a memory access otherwise the performance of the controller will be degraded.

<sup>1</sup> Assuming an 'ideal' external memory and the *port\_ip* flag is not used by one of the other devices on the bus.

If the depth of this FIFO is less than the number of clock cycles taken for an external memory read, then the arbiter will introduce extra stalling on the bus when reading memory. In practice it may not be possible to use a FIFO deep enough to hide the maximum memory latency. However, the depth of this FIFO must be set to at least the average latency for the best system performance.

### Burst Management and Control

The burst management unit accepts a port id from the bus arbitration unit and grants the Master with this port id access to the bus. Memory accesses are granted in bursts - a burst being a sequence of consecutive accesses on the bus. The burst size is defined in the generic parameter *burst\_size* and can be anything from 1 to 256 in length<sup>2</sup>.

Bursts can be a write-only, read-only or mixture of both reads and writes in the same operation. Once the maximum burst size has been reached, a new port id is requested and arbitration is given to the next Master. If the current Master drops its valid signal in the middle of a burst, then the Master will loose arbitration on the bus and access will be granted to the next Master.

### Performance considerations

In summary, in order to use the available memory bandwidth as efficiently as possible, and to avoid 'dead time' on the bus, the following basic rules must be adhered to.

1. The burst size of the master device and the controller must be the same.
2. The Master must not drop its valid signal in the middle of a burst.
3. For SDRAM architectures, the address sequence within a burst must be in the same memory page (open row).
4. If a Master wants to access a different page, then the current burst must be terminated and the new page access must be started in the next burst.
5. The Master should use the *port\_ip* flag when it has a number of sequential bursts that are in the same page.
6. The Master should restrict each burst to either a read or a write operation as some SDRAM architectures incur a performance penalty when switching between reads and writes in the same burst.
7. The maximum latency of a memory read should be matched in the bypass FIFO.

Note that these rules are guidelines only. Breaking these rules will not break the functionality of the controller in any way. Sticking to these rules, however, will ensure optimum performance and best use of the available memory bandwidth.

### Functional Timing

Figure 2 demonstrates the arbitration between four separate Masters on the bus using a round-robin scheme<sup>3</sup>

All four Masters are continually requesting access - characterized by their *port\_addr\_val* signals going high. The round-robin algorithm grants access on the bus in numerical order with the respective *port\_addr\_rdy* signal granting access to each Master in sequence.

Note that the burst size in this example has been set to 8, each Master being granted access for 8 cycles before arbitration is passed to the next Master on the bus.

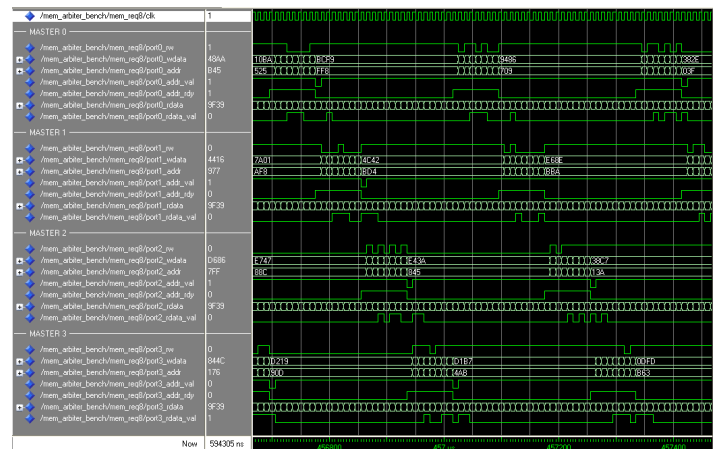


Figure 2: Round-robin arbitration of 4 separate Masters

The next timing waveform is an example of page-break optimization in use. Again, as in the previous example, a nominal burst size of 8 has been chosen. Access is granted to Master '0' first for a normal burst of 8 cycles. Access is then granted to Master '1', but notice that during the first burst, Master '1' asserts the *port\_ip* flag high. This flag overrides normal arbitration and holds the bus for a further 8 cycles. In this way, Master '1' holds the bus for a full 16 cycles and avoids breaking page between consecutive bursts.

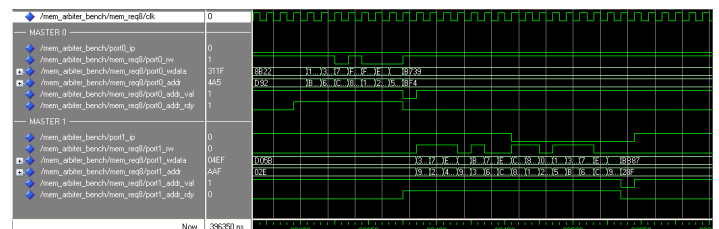


Figure 3: Use of the in-page optimization flag

<sup>2</sup> Maximum burst size may be extended on request. Please contact ZIPcores.

<sup>3</sup> The top-level signal names have been modified slightly to make them easier to read. For instance the signal *port\_wdata[15:0]* is labelled *port0\_wdata* in the timing waveform.

Figure 4 demonstrates the memory interface signalling. All accesses use the valid-ready protocol in the same manner as the inputs to the controller. A transfer occurs on a rising clock-edge whenever valid and ready are both high. The blue circle in the timing waveform is an example of a pipeline stall. In this instance, the signal *mem\_addr\_val* is high and *mem\_addr\_rdy* is low. Notice that the address and data are stalled for one cycle until ready is asserted high and normal transfers resume.

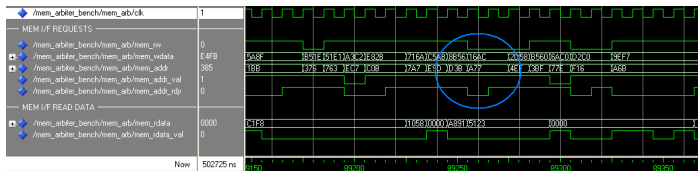


Figure 4: Memory interface signalling showing pipeline stalls

## Source File Description

All source files are provided as text files coded in VHDL. The following table gives a brief description of each file.

Source file	Description
pipeline_reg.vhd	Pipeline register
fifo_sync.vhd	Synchronous FIFO
arb_rr.vhd	Round-robin arbitration unit
arb_pri.vhd	Fixed-priority arbitration unit
mem_arbiter_mux.vhd	Input multiplexer
mem_req_model.vhd	Master request model
mem_req_model8.vhd	Master request model (8 Masters)
ram_sp.vhd	Single port RAM
mem_model_4096x16bit.vhd	External 16-bit memory model
mem_arbiter.vhd	Top-level block
mem_arbiter_bench.vhd	Top-level test bench

## Functional Testing

An example VHDL test bench is provided for use in a suitable VHDL simulator. The compilation order of the source code is as follows:

1. pipeline\_reg.vhd
2. fifo\_sync.vhd
3. arb\_rr.vhd
4. arb\_pri.vhd
5. mem\_arbiter\_mux.vhd
6. mem\_arbiter.vhd
7. mem\_req\_model.vhd
8. mem\_req\_model8.vhd
9. ram\_sp.vhd
10. mem\_model\_4096x16bit.vhd
11. mem\_arbiter\_bench.vhd

The VHDL test bench instantiates the multi-port memory controller together with a model containing 8 bus Masters and an external memory component organized as 4096 x 16-bits.

The generic settings of the controller may be modified by the user to model different test scenarios. In particular, the burst size, arbitration mode and page-break optimization settings may be modified to observe the behaviour under the different arbitration schemes.

In the example provided, the simulation must be run for at least 3 ms during which time the eight master devices will generate random requests to the controller. The burst size is set to 8 with round-robin arbitration selected and page-break optimization disabled. The controller is connected to a 16-bit memory model which is also customizable. The memory model has generic parameters to add randomized stalling at its interface and also, to change the latency of a memory read.

During the course of the simulation, two text files are generated called *mem\_arbiter\_in.txt* and *mem\_arbiter\_out.txt*. These files respectively contain the input requests to the arbiter and the returning read data for each of the master devices. The file *mem\_arbiter\_in.txt* is partitioned into four fields: port id, read/write flag, address and write data. The file *mem\_arbiter\_out.txt* is partitioned into two fields: port id and read data.

The contents of both files may be compared to verify correct operation of the multi-ported memory controller. Scripts to parse and format the test data are available on request.

## Synthesis

The files required for synthesis and the design hierarchy is shown below:

- mem\_arbiter
  - arb\_rr
  - arb\_pri
  - mem\_arbiter\_mux
  - pipeline\_reg
  - fifo\_sync
  - pipeline\_reg

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx Virtex 5 and the Altera Stratix III series of FPGA devices. The lowest and highest speed grade devices have been chosen in both cases for comparison.

The multi-port memory controller supports 2, 4, 8, 16 or 32 master ports. For a smaller number of ports, the unused inputs should be tied to logic '0' which will result in the associated logic being optimized out during synthesis. Further optimizations can also be made if individual ports are made read or write only - i.e. the *port\_rw* flags tied to either '0' or '1'.

In addition, it is important to note that the speed and area of the controller is greatly effected by the number of ports specified and the width of the addresses and data used. As a general rule, the arbitration mode and page-break optimization modes have negligible effect on the overall speed and area figures.

Trial synthesis results are shown with the generic parameters set to: *arb\_mode* = 0, *opt\_mode\_pb* = false, *num\_ports* = 8, *num\_ports\_log2* = 3, *addr\_width* = 20, *word\_width* = 16, *fifo\_depth* = 32, *fifo\_depth\_log2* = 5.

Resource usage is specified after Place and Route.

*VIRTEX 5*

<b>Resource type</b>	<b>Quantity used</b>
Slice register	76
Slice LUT	200
Block RAM	0
DSP48	0
Clock frequency (worst case)	227 MHz
Clock frequency (best case)	301 MHz

*STRATIX III*

<b>Resource type</b>	<b>Quantity used</b>
Register	99
ALUT	209
Block Memory bit	96
DSP block 18	0
Clock frequency (worse case)	265 MHz
Clock frequency (best case)	342 MHz

**Revision History**

<b>Revision</b>	<b>Change description</b>	<b>Date</b>
1.0	Initial revision	22/06/09