QUAZAR FPGA RTL Memory QPR Controller Selector

PRODUCT BRIEF

QUAZAR Serial Accelerator Devices

This product brief presents a comprehensive overview of the integration and implementation of the MoSys family of QUAZAR accelerator engines and shows how applications can be accelerated by utilization of the readily available MoSys IP.

The MoSys RTL controller operates similar to a QDR type memory.

Key Features / Product Options

- Highlight simple user RTL interface that controls the serial memory interface, requiring no user RTL design effort
- Interfaces between Memory Controller and User Application
- Read and Write controls
- Interface Signals
- Memory Controller selected by user to operate memory in DEEP Mode (4 SRAMS) or WIDE Mode (8 SRAMS)

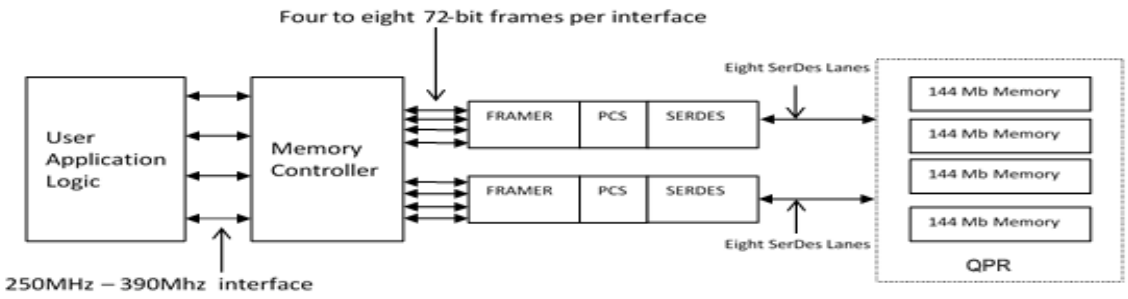
Overview

The MoSys memory controllers are designed to simplify the integration of the accelerator engines into a design.

- The controllers are built with all the high-speed SerDes control and implementation of the GCI protocol essentially “hidden away” from your design effort.
- MoSys controllers which have been deployed in the field since 2004 have been proven to be robust and reliable.
- The interface which is presented to the user application interface is a straightforward Address, Data, Command bus structures compatible with and easily adapted to an AXI interface.
- Multiple versions are available to support different access patterns and for different hosts (Xilinx, Intel, ASIC etc.)

This write-up has been presented to allow a user to realize that integration and implementation of the MoSys QUAZAR family of accelerator engines is not a long process and can be accelerated by utilization of the readily available MoSys IP.

QUAZAR Accelerators



RTL Memory QUAZAR QPR Controller Selector

Introduction

The MoSys QUAZAR memory controllers are designed and offered with a few variations of memory access patterns. The most common access patterns are:

- Balanced Read/Write (similar to QDR SRAM)
- Native (higher read access than write – for table access applications)
- This is two separate controllers)

The RTL that is supplied by MoSys provides an interface between the User Application Logic and the MoSys QUAZAR Accelerator Engine device (could be MSQ220 or MSQ230 devices). The Memory Controller also implements all the required signaling and handshaking defined in the GCI Interface protocol, Framer logic.

The signals interface at the User Application provides Quad Partition Rate (QPR) devices with a simple SRAM memory read/write operation. This simple interface shields the users from designing scheduling logic for QPR memory partitions wheel.

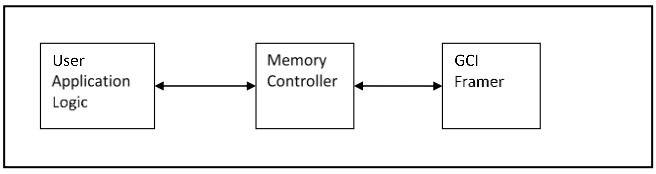


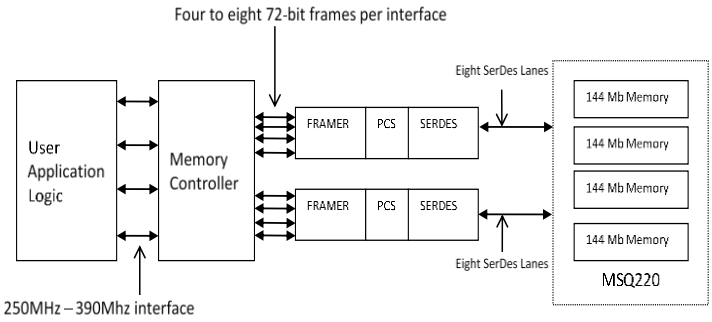
Figure 2: Memory Controller Interface

The goal of each of the Memory Controller designs is to balance the bandwidth between the User Application Interface and the QPR Interface. For many applications there will be four read/write interfaces from the User Application running at the host core clock frequency. This is to balance the bandwidth of the application logic (assumed to be running at FPGA speeds vs. the GCI I/O and core frequency of the MoSys accelerator engines)

In many of the FPGA applications that has been between 250MHz and 390MHz clock rate. Each interface can accommodate one memory read and one memory write on each clock cycle. These result in memory accesses per interface that can saturate the access bus to the memory.

This allows the total bandwidth at the User Application interfaces to be up to 2.5 billion memory accesses per second when using an MSQ220 device and 5 billion memory operations when using an MSQ230 device. (This bandwidth matches the total I/O bandwidth on a QPR device when using all 16 lanes at maximum allowable SerDes rate of the Accelerator engine device) per lane. The following picture illustrates the above memory bandwidth discussion.

2.5 Billion Accesses for 2 - GCI interfaces



RTL Memory QUAZAR QPR Controller Selector

Memory Operating Modes

Each QPR memory has two ports and the memory can operate in two modes.

- Deep Mode
  - 4 independent SRAMs
  - SRAMs capacity is the device total capacity divided by 4.
  - QPR4 has 4 memories of 2M x 72b
  - QPR8 has 4 memories of 4M x 72b
- Wide Mode
  - 8 independent SRAMS
  - SRAMs capacity is the device total capacity divided by 8
  - QPR4 has 8 memories of 1M x 72b
  - QPR8 has 8 memories of 2M x 72b

The following description of the RTL controller described the User RTL interface to one SRAM. This interface is duplicated for each SRAM on the device.

Diagram shows the Memory DEEP Mode and WIDE Mode for QPR4 and QPR9

Balanced Read/Write, Native

Interfaces between Memory Controller and User Application

This section describes the interface signals and the interface protocols between the Memory Controller and the User Application.

Interface Signals

The interface between Memory Controller and User Application consists of four sets of similar interface signals. However, just one of the interfaces will be described to avoid redundancy. All signals will be appended by the string "_pi" at the end of their names where i can be 0, 1, 2, or 3 depending on the interface set.

The following table describes all the signals in one of the interface signals set between the User Application and the Memory Controller. There are up to four of these interface signals sets in an implementation.

Signal Name	Width	Dir	Description
Read Interface			
rd_pi	1	In	Assertion of this signal to indicate that this is a read transaction.
rd_addr_pi	32	In	Read address of the memory for this transaction. For burst transaction, this is the first address of the burst memory block. Please refer to the Address section of this specification to see the detail of this address field.
rd_data_pi	72	Out	Returned data from BE1 memory. This data is qualified by the “rd_datav_pi” signal.
rd_wait_rq_pi	1	Out	The Memory controller asserts “rd_wait_rq_pi” to indicate that it cannot accept the current read request. The User Application should hold all the request signals (rd_pi, rd_addr_pi …) until the de-assertion of this signal.
rd_datav_pi	1	Out	The Memory Controller asserts this signal to indicate the current data in the “rd_data_pi” bus is valid.
rd_burstcount_pi	5	In	These signals show the number of 72-bits words in this read burst transaction.
rd_burst_pi	1	In	Assertion of this signal to indicate this transaction is the read burst transaction.
rd_flush_pi	1	In	Assertion of this signal will flush all the pending read transactions and their associated read data. The flush action is effective for the next cycle after the assertion of “flush” signal. It is the responsibility of the User Application to ignore or to accept the valid “rd_data_pi” in the current cycle.
Write Interface			
wr_pi			
wr_addr_pi	32	In	Write address of the memory for this transaction. For burst transaction, this is the first address of the burst memory block. Please refer to the Address section of this specification to see the detail of this address field.
wr_data_pi	72	In	Write data from the User Application logic.
wr_wait_rq_pi	1	Out	The Memory controller asserts “wr_wait_rq_pi” to indicate that it cannot accept the current write request. The User Application should hold all the request signals (wr_pi, wr_addr_pi …) until the de-assertion of this signal.
wr_burstcount_pi	5	In	These signals show the number of 72-bits words in this write burst transaction.
wr_burst_pi	1	In	Assertion of this signal to indicate this transaction is the write burst transaction

Interface Protocol

This section illustrates the interface protocols between the User Application and the Memory Controller.

Single Read/Write transactions

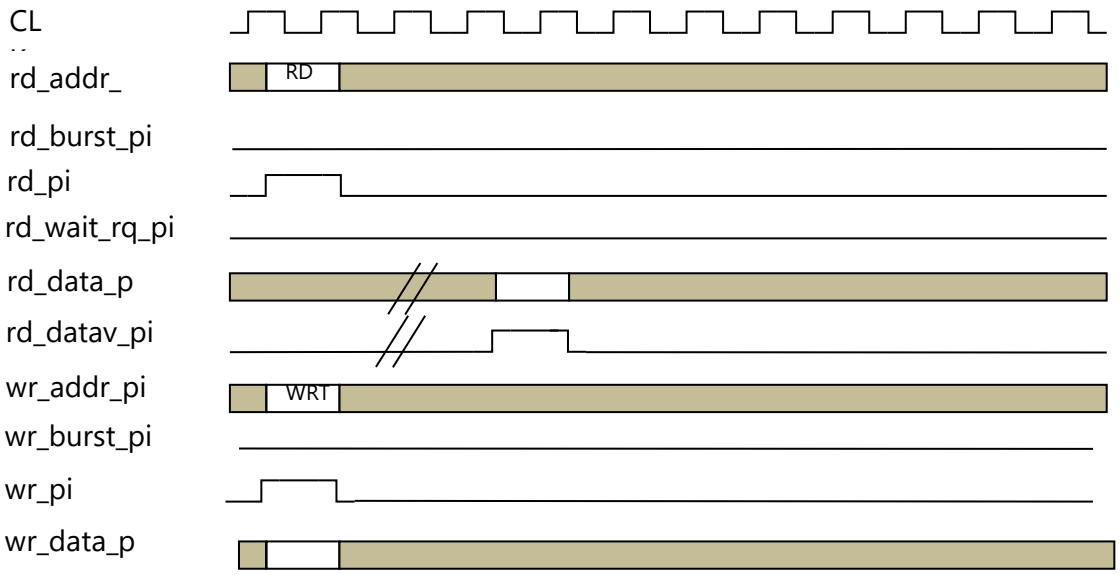


Figure 3: Single Read/Write Transaction

The above picture illustrates the read transaction followed by a write transaction. A read transaction is initiated by the user application on the assertion of the “rd_pi” signal, and the read address “rd_addr_pi”. The Memory Controller returns the “rd_data_pi” along with the “rd_datav_pi” signal to indicate the read data is valid in that cycle.

The write cycle is initiated by the user application on the assertion of the “wr_pi” signal, the write address “wr_addr_pi”, and the “wr_data_pi”.

Single Read/Write Transaction with Wait cycles

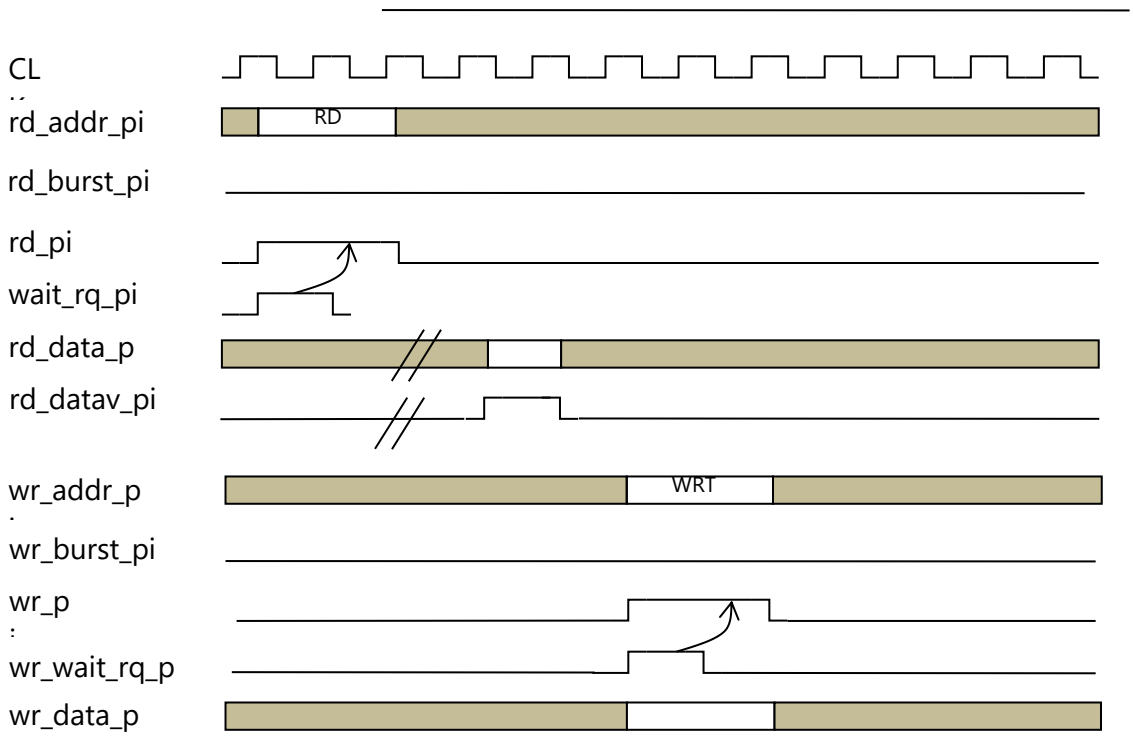


Figure 4: Single Read/Write Transaction with Wait cycles

The above picture illustrates the read transaction followed by the write transaction with wait request from the Memory Controller to stall the User Application.

The read transaction is initiated by the assertion of signal “rd_pi” along with the read address “rd_addr_pi”. The Memory Controller asserts the “rd_wait_rq_pi” signal in the same cycle to request the User Application to hold the “rd_addr_pi” bus and the “rd_pi” signal until the de-assertion of “rd_wait_rq_pi”. This is the mechanism for the Memory Controller to asserts back-pressure the User Application logic in the case its FIFOs are full. As in the non-stalled case, the Memory Controller returns the “rd_data_pi” along with the “rd_datav_pi” signal to indicate the validity of the data in that cycle.

The write transaction is initiated by the assertion of signal “wr_pi” along with the write address “wr_addr_pi”, and the “wr_data_pi”. The Memory Controller asserts the “wr_wait_rq_pi” signal in the same cycle to request the User Application to hold the “wr_addr_pi” bus, the “wr_pi” signal, and the “wr_data_pi” bus until the de-assertion of “wait_rq_pi” signal.

Pipelined Read Transactions

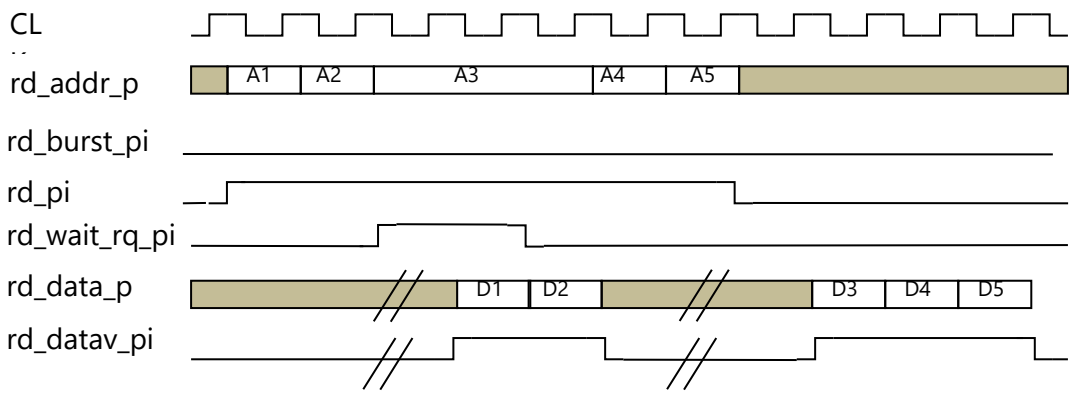


Figure 5: Pipelined Read Transactions

`

The Memory Controller supports pipelined read transactions. The User Application can issue back to back read requests until the Memory Controller issues the back-pressure signal “rd_wait_rq_pi”. All the return “rd_data_pi” are delivered along with the associated “rd_datav_pi” in the order of the read requests.

The above picture illustrates the pipelined read transactions. The User Application issues five back to back read requests with the stall happen on the third request. The Memory Controller returns the five “rd_data_pi” in the order of the requests.

Summary

In each of the above cases MoSys memory controllers are designed to simplify the integration of the QPR accelerator engines into a design using the DEEP Mode or WIDE Mode, whichever is the best solution*. And, to change from one mode too another only requires an FPGA RTL controller change. No hardware changes.*

The controllers are built with all the high-speed SerDes control and implementation of the GCI protocol essentially “hidden away” from your design effort.

MoSys controllers which have been deployed in the field since 2004 have been proven to be robust and reliable.

The interface which is presented to the user application interface is a straightforward Address, Data, Command bus structure, that is compatible with and easily adapted to an AXI interface. Multiple versions are available to support different access patterns and for different hosts (Xilinx, Intel, ASIC etc.)

This write-up has been presented to allow a user to realize that integration and implementation of the MoSys family of accelerator engines is not a long process and can be accelerated by utilization of the readily available MoSys IP.

If, however your desire is to develop your own interface to the Accelerator Engine Family, the GCI specification is readily available and free to implement and use.

.

2309 Bering Drive, San Jose, CA
95131 Tel: 408-418-7500 Fax:
408-418-7501