MoSys Technical Brief

Title: MoSys FPGA RTL Memory Controller Selector
Author: M. Baumann
Date: 2020, 04, 14

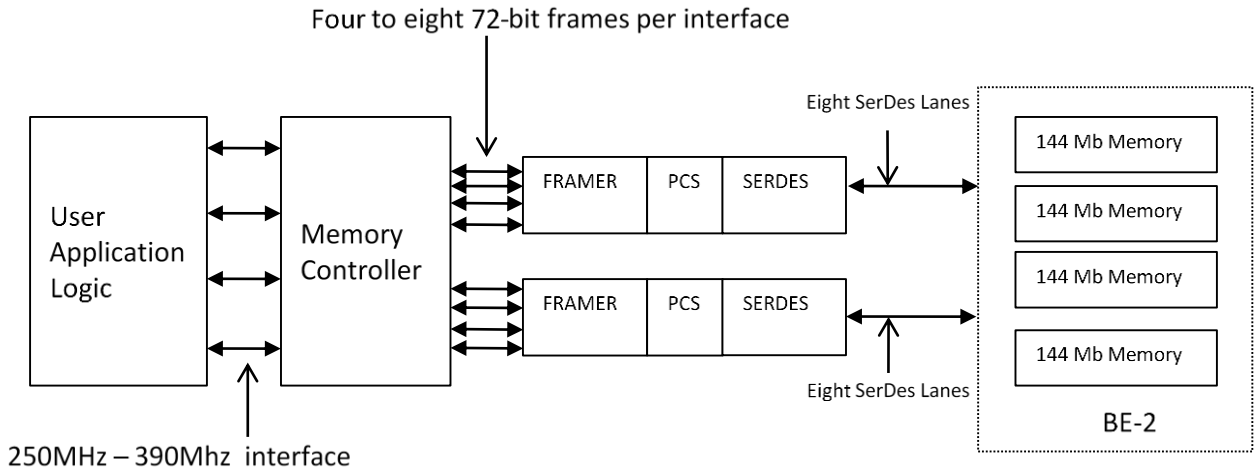
Document Number:
TB_AD_FPGA RTL MEM CONTROLER SELECTOR_200413

SUMMARY

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

Key Points

- Interfaces between Memory Controller and User Application
- RMW Statistics
- Interface Signals



MoSys FPGA RTL Memory Controller Selector

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 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.

Introduction

The MoSys 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)
- Burst (Allows one command to access 2, 4 or 8 locations)
- Statistics (Takes advantage of the Accelerator Engines on board ALU to keep data statistics)

The RTL that is supplied by MoSys provides an interface between the User Application Logic and the MoSys Accelerator Engine device (could be BE-2 or BE-3 families). 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 Bandwidth Engine User a simple SRAM memory read/write operation with burst capability. This simple interface shields the users from the BE-X commands and the scheduling logic for Bandwidth Engine memory partitions wheel.

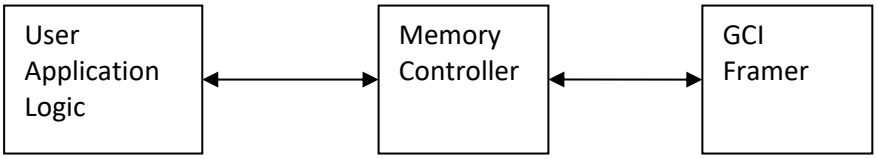


Figure 1: Memory Controller Interface

The goal of each of the Memory Controller designs is to balance the bandwidth between the User Application Interface and the Bandwidth Engine 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 billion memory accesses per second when using a BE-2 device and 6 billion memory operations when using a BE-3 device. (This bandwidth matches the total I/O bandwidth on Bandwidth Engine 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 Billion Accesses for 4 interfaces

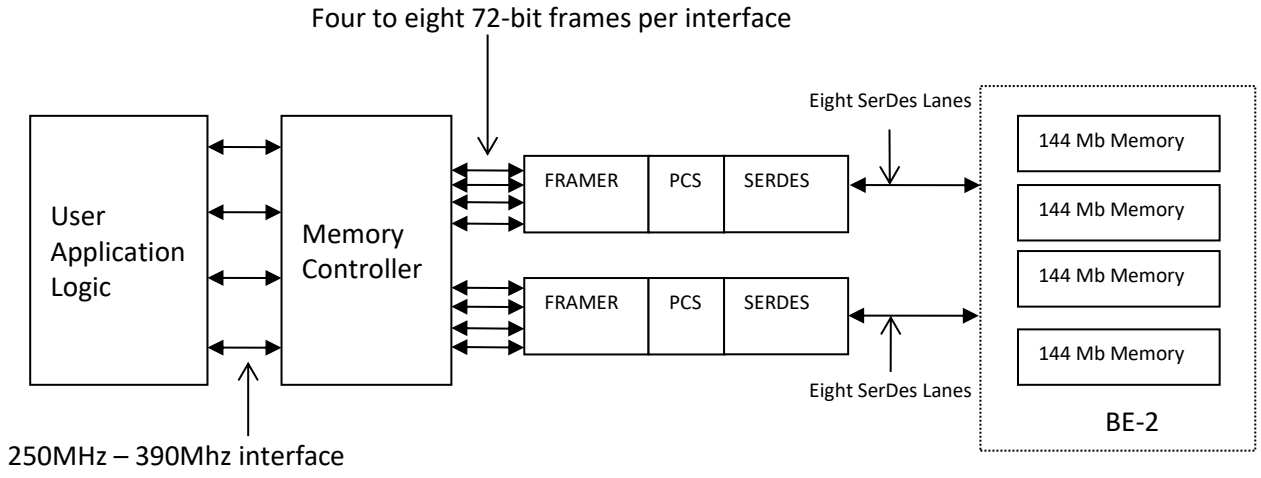


Figure 2: Memory Bandwidth between Memory Controller and Bandwidth Engine.

Balanced Read/Write, Native and BURST

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 four of these interface signals sets in the current 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.

Signal Name	Width	Dir	Description
Write Interface			
wr_pi			1
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

Table 1: Interface Signals Set between Memory Controller and User Application logic

Interface Protocol

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

Single Read/Write transactions

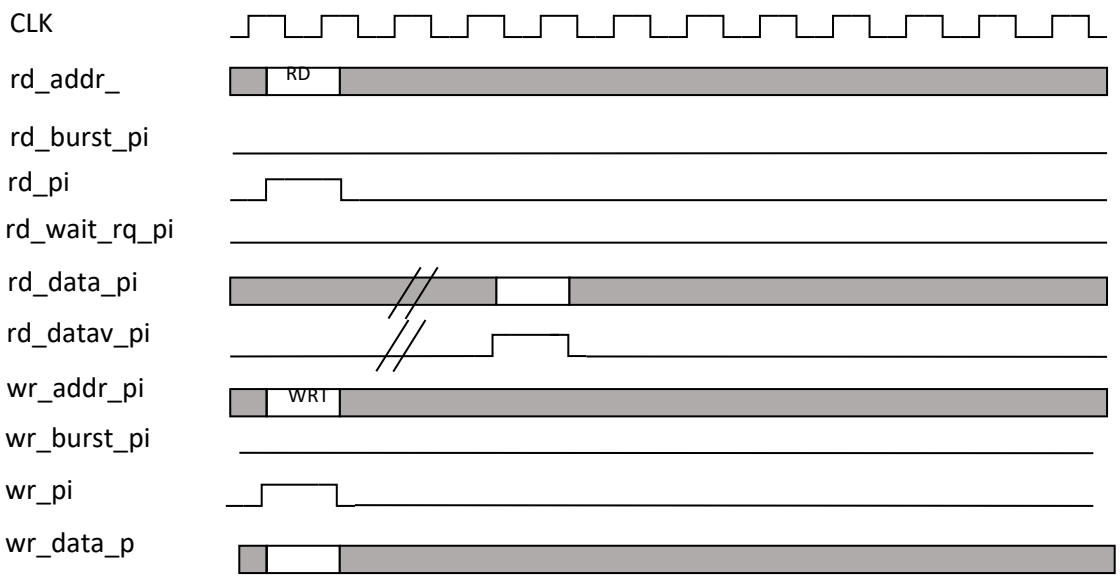


Figure 3: Single Read/Write Transaction

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

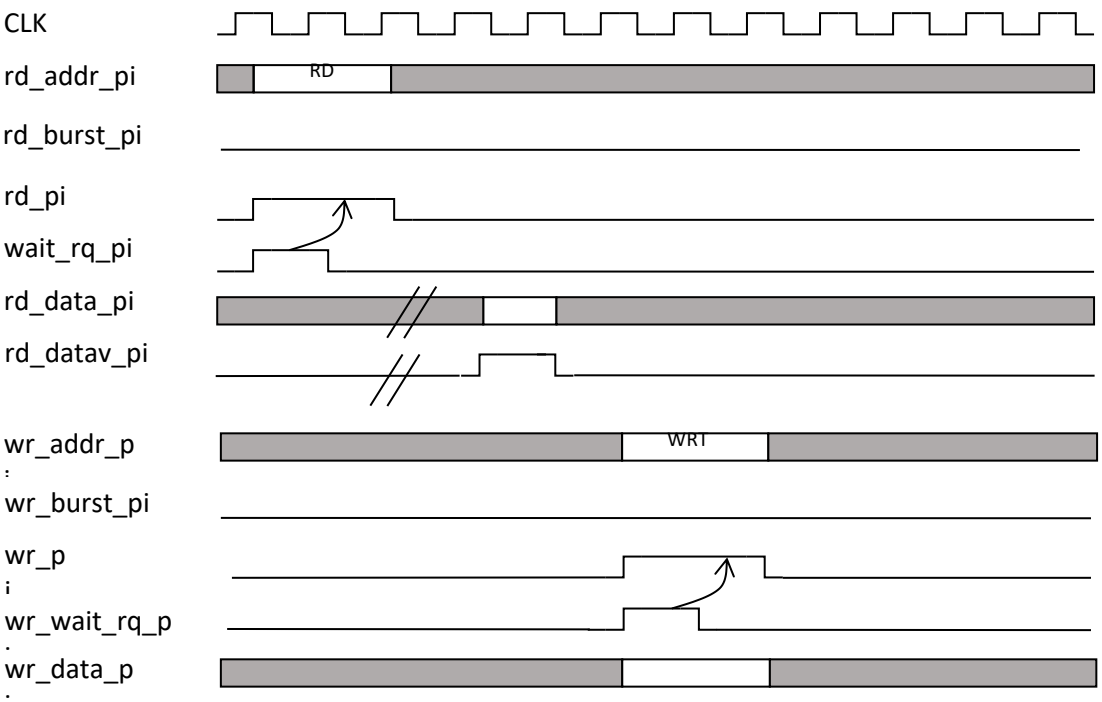


Figure 4: Single Read/Write Transaction with Wait cycles

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

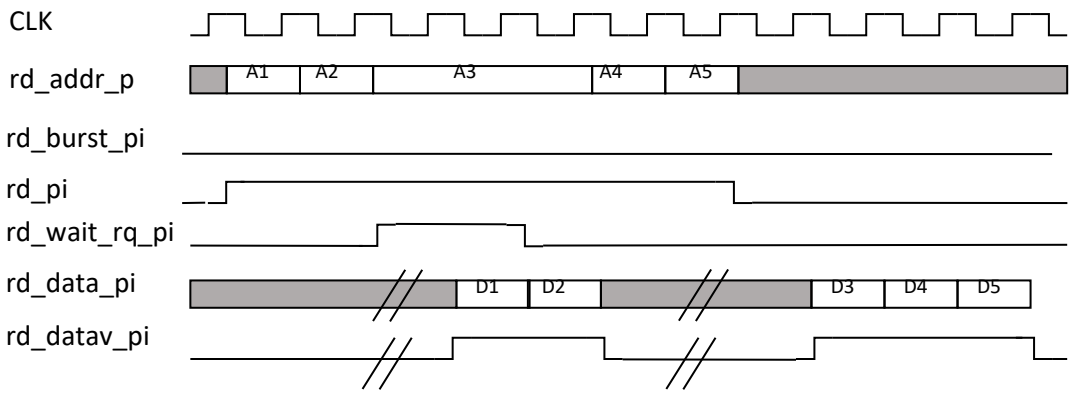


Figure 5: Pipelined Read Transactions

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.

Burst Read Transaction

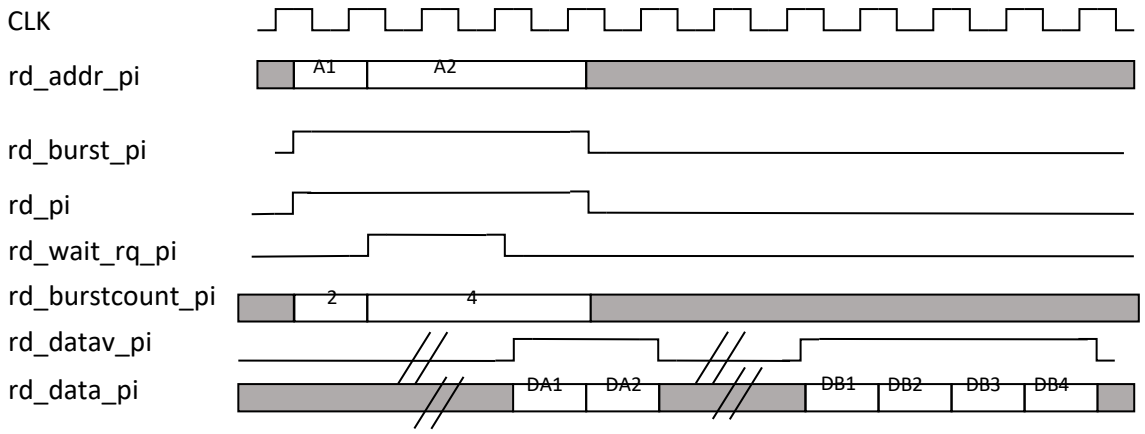


Figure 6: Burst Read Transaction

The Memory Controller supports burst read transactions. The User Application initiates the burst read transaction by asserting signal “rd_burst_pi” along with the “rd_pi” signal, the start address of the burst block (“rd_addr_pi” bus), and the burst count (“rd_burstcount_pi” bus). Similar to the single read request, the Memory Controller stalls the User Application by asserting the “rd_wait_rq_pi” signal. The User Application needs to hold the read signal, the burst indicator signal, the burst count, and the start address in this case. The Memory Controller returns the burst data (“rd_data_pi” bus) along with the “rd_datav_pi” signal to indicate the validity of the data. The Memory Controller uses the start address and increment this address for subsequent words in the burst block. If the address crosses the memory partition address, it will be wrapped around in the current implementation.

The above picture illustrates the burst transactions with 2 words and 4 words burst sizes. The second transaction is stalled by the Memory Controller with the assertion of “rd_wait_rq_pi” signal.

Burst Write Transaction

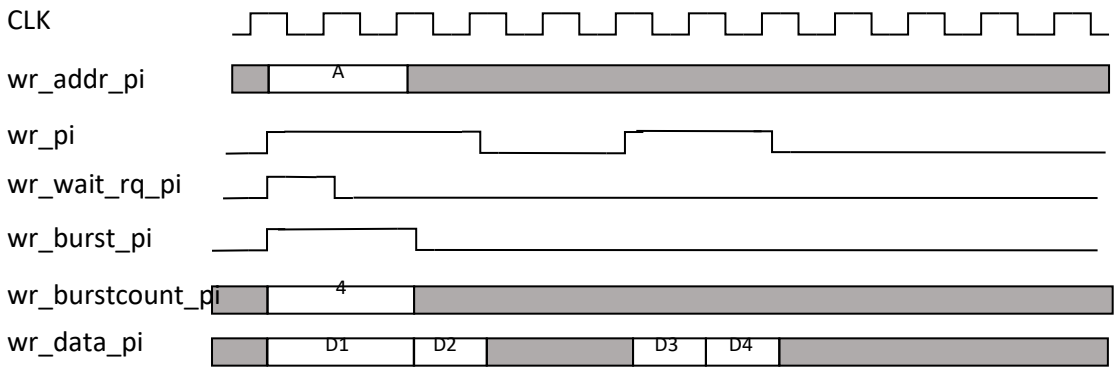


Figure 7: Burst Write Transaction

The Memory Controller supports burst write transactions. The User Application initiates the burst write transaction by asserting the “wr_burst_pi” signal, the “wr_pi” signal, the start address of the burst block (“wr_addr_pi” bus), the burst count (“wr_burstcount_pi” bus), and the write data (“wr_data_pi” bus). Similar to the single write request, the Memory Controller can stall the User Application by asserting the “wr_wait_rq_pi” signal. The User Application needs to hold the write signal, the burst indicator, the burst count, the start address, and the write data in this case. The User Application can throttle the write data within the burst block by using the “wr_pi” signal. The Memory Controller uses the start address and increment this address for subsequent words in the burst block. If the address crosses the memory partition address, it will be wrapped around in the current implementation.

The above picture illustrates the burst write transaction with the burst count of 4. The Memory Controller stalls the request by asserting the “wr_wait_rq_pi”. The User Application throttles the “wr_data_pi” by using the “wr_pi” signal.

Statistics (R-M-W)

1 Introduction

This is the specification for the BE Statistic Controller. This interface performs the same bridge function between the application logic and the bandwidth engine device however it is designed to support R-M-W or read-modify-write operations in order to allow the application to issue ALU operations to perform functions such as maintaining statistics or metering. This design utilizes the large memories found in the BE2 or BE3 devices along with its embedded ALU and its RMW instructions in order to implement the counter or metering.

As an example, with the current BE2 memory size and 8 serial lanes running at 12.8 Gbps, this Statistic Controller has the following capabilities:

- Incrementing up to eight concurrent counters at the rate up to 160 Million counts per second which is higher than typical 100GE at 148.8 Mpps.
- The 576 Mb memory capacity on BE2 can support up to 8 x 256K counters.
- Interface to read the counters. This operation returns the counter content.
- These counters are lifetime counters (which means they are a full 64 bits wide).
- Counters are initialized by the controller after de-assertion of reset.
- Diagnostic interface to read and write BE2 memory for initial memory testing.
- Optional ECC Correction circuitry to correct one-bit error and detect two bits or more error on the counter read data.

2 Micro Architecture

For a statistics and R-M-W operations there are 8 counter interfaces in which four counters are mapped to lower part of the four partitions in BE memory and other four counters are mapped to upper part of the four partitions. The user specifies 18-bit counter index that is mapped to a location in partition. The figure 8 below illustrates the statistic controller interfaced with user at one end and GCI PCS/Framer at the other end.

The user requests are captured at the input of an Asynchronous FIFO at a clock frequency of the application logic. The R-M-W (or statistics) controller then pulls user requests out of this Asynchronous FIFO at its internal clock derived from SerDes clock. The statistic controller operates at this frequency to optimize the interface with GCI running at SerDes speed. The statistic controller creates RMW (Read-Modify-Write) commands from the user requests to implement counter operation using BE memory.

The statistic controller has following major blocks:

1. Eight asynchronous FIFOs corresponding to 8 user increment requests.
2. Eight asynchronous FIFOs corresponding to 8 users read count requests.
3. One Asynchronous FIFO for memory Diagnostic interface user request.
4. A scheduler block which performs RMW command formation and scheduling based on user request.
5. A receive data block that receives the read data from the PCS/Framer block and send it to the user on its respective interface, this being the same interface that requested the action.
6. A debug interface block that tests the BE memory.

   On each wheel (or FPGA Cycle time), four counter operations are performed that is four R-M-W commands corresponding to counter increments, are scheduled. On one-wheel cycle, four counters located at lower part of four partitions in BE memory, are incremented. In next wheel cycle, another four counters located at upper part of four partitions, are incremented. Out of every 15 accesses of the BE memory, first 14 accesses are RMW commands corresponding to counter increments and 15^{th} access is the memory read operation. The RMW and read command scheduling is shown in Figure 9 below.

The following pictures illustrate a BE Statistic Application.

Figure 8: BE Statistic Controller

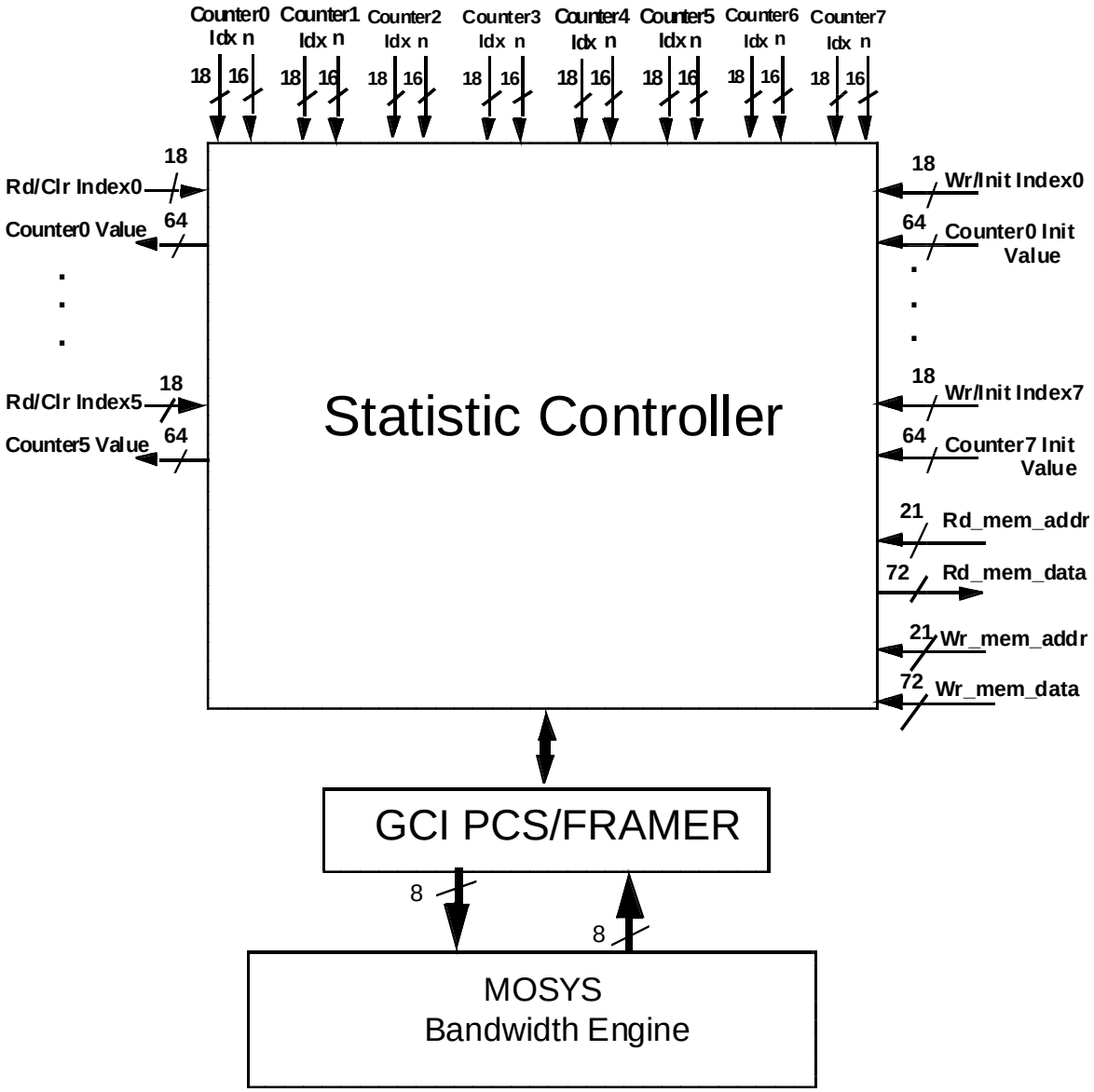


Figure 9: BE Statistic Controller issuance scheduling

		PW-0		PW-1		PW-2		PW-3	
Clock Cycle	CMD_SCH_WHEEL	MC_CMD0_SCH_0	MC_CMD0_SCH_1	MC_CMD1_SCH_0	MC_CMD1_SCH_1	MC_CMD2_SCH_0	MC_CMD2_SCH_1	MC_CMD3_SCH_0	MC_CMD3_SCH_1
0			LDBASE		LDBASE		LDBASE		LDBASE
1	0	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
2	1	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
3	2	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
4	3	CNT_4	LDBASE	CNT_5	LDBASE	RD_6	LDBASE	CNT_7	LDBASE
5	0	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
6	1	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
7	2	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
8	3	CNT_4	LDBASE	RD_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
9	0	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
10	1	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
11	2	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
12	3	RD_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
13	0	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
14	1	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
15	2	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	RD_3	LDBASE
16	3	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
17	0	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
18	1	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
19	2	CNT_0	LDBASE	CNT_1	LDBASE	RD_2	LDBASE	CNT_3	LDBASE
20	3	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
21	0	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
22	1	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
23	2	CNT_0	LDBASE	RD_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
24	3	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
25	0	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
26	1	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
27	2	RD_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
28	3	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	CNT_7	LDBASE
29	0	CNT_0	LDBASE	CNT_1	LDBASE	CNT_2	LDBASE	CNT_3	LDBASE
30	1	CNT_4	LDBASE	CNT_5	LDBASE	CNT_6	LDBASE	RD_7	LDBASE

3.1 Interface Signals

The following table(s) describes all the interface signals between the Statistic Controller and the User Application.

Table 2 Interface Signals between Statistic Controller and User Application

Signal Name	Width	Dir	Description
Counters Increment Interface.			
inc_cntr_#	1	IN	Enable incrementing counter_# (Here, # refers to counters 0 to 7).
inc_index_#[17:0]	18	IN	The index (out of 256K) for counter #.
inc_value_#[15:0]	16	IN	The increment value for counter #.
inc_ready_#	1	OUT	Back pressure signal to counter #. The Statistic Controller only accepts the counter increment input from the user application when this signal is asserted.
Counters Read Interface.			
rd_cntr_#	1	IN	This signal indicates the read counter # request.
rd_cntr_index_#[17:0]	18	IN	The index for read~~/clear~~ counter #.
rd_cntr_ready_#	1	OUT	Back pressure signal to the user application. The Statistic Controller only accepts the counter # read~~/clear~~ request from the user application when this signal is asserted.
rd_cntr_#_data[63:0]	64	OUT	The value of counter #.
rd_cntr_#_valid	1	OUT	This signal indicates that the rd_cntr_#_data bus is valid. If the optional ECC correction circuitry is implemented, this signal also indicates the validity of the two signals rd_cntr_0_corr_err and rd_cntr_0_uncorr_err.
Diagnostic Memory Write Read Interface.			
debug_rd_mem	1	IN	Debug Read memory request
debug_rd_mem_data_valid	1	OUT	This signal indicates that the rd_mem_data bus is valid.
debug_wr_mem	1	IN	Debug memory request
debug_mem_addr[20:0]	21	IN	Debug memory address
debug_mem_partition[1:0]	2	IN	Partition number for the memory debug command
debug_wr_mem_data[71:0]	72	IN	Debug memory write data.
debug_mem_ready	1	OUT	Back pressure signal to the user application. The Statistic Controller only accept memory write request from the user application when this signal is asserted.

Summary

In each of the above cases 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 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. For additional information on any of the controllers that mentioned bove or just to speak with MoSys on the use of their Accelerator Engine Family of devices please contact MoSys at: www.MoSys.com.