Renaissance
DATA SHEET

Renaissance™ Product Family mRnW Sequential Memory Cores

This datasheet provides functional information for the multiport Read-Write (mRnW) Algorithmic Memory® cores.

Release Date: Oct 6th, 2013

Contents

1. Introduction

This datasheet describes the functioning of Memoir's Renaissance product family memory cores. The cores are created by wrapping RTL-IP around physical memories in a target library. The RTL implements the algorithm specified memory core type. The memory cores are generated by the user using the MemoGen^{TM} software made available by Memoir Systems. The Renaissance family currently offers memory cores which provide 2X, 4X performance acceleration.

This datasheet gives an overview of a generic 'mRnW' memory core, a multi-port read-write core with 'm' read-ports and 'n' write-ports. This datasheet describes a sample memory core with four read and four write ports. Functionality of cores with different values of 'm' and 'n' can be inferred similarly from this description.

Specific information about a memory core - address/data bus-widths, clock-frequency, timing parameters, area, power, underlying physical memory components, refresh-scheme details (if applicable), etc. - is made available to the user in a separate datasheet which is auto-generated by the MemoGen^{TM} software.

Each mRnW memory core supports the following:

- Parameterized number of address-words, data widths
- Clock-synchronous, SRAM-like interface with pipelined random memory access for all ports
- Pass-through interface for diagnostic and testability (incl. BIST) support

2. Block Diagram and Interface Description

Figure 2-1 illustrates the block diagram of a 4 read port and 4 write port Renaissance memory core. For compactness, subscripts are appended to the port signal names to imply four independent ports. Subscripts {0, 1, 2, 3} imply four read ports numbered port-0, port-1, port-2, and port-3. Similarly subscripts {4, 5, 6, 7} imply four write ports numbered port-4, port-5, port-6, and port-7. These subscripts should NOT be confused as bit-numbers of a bus.

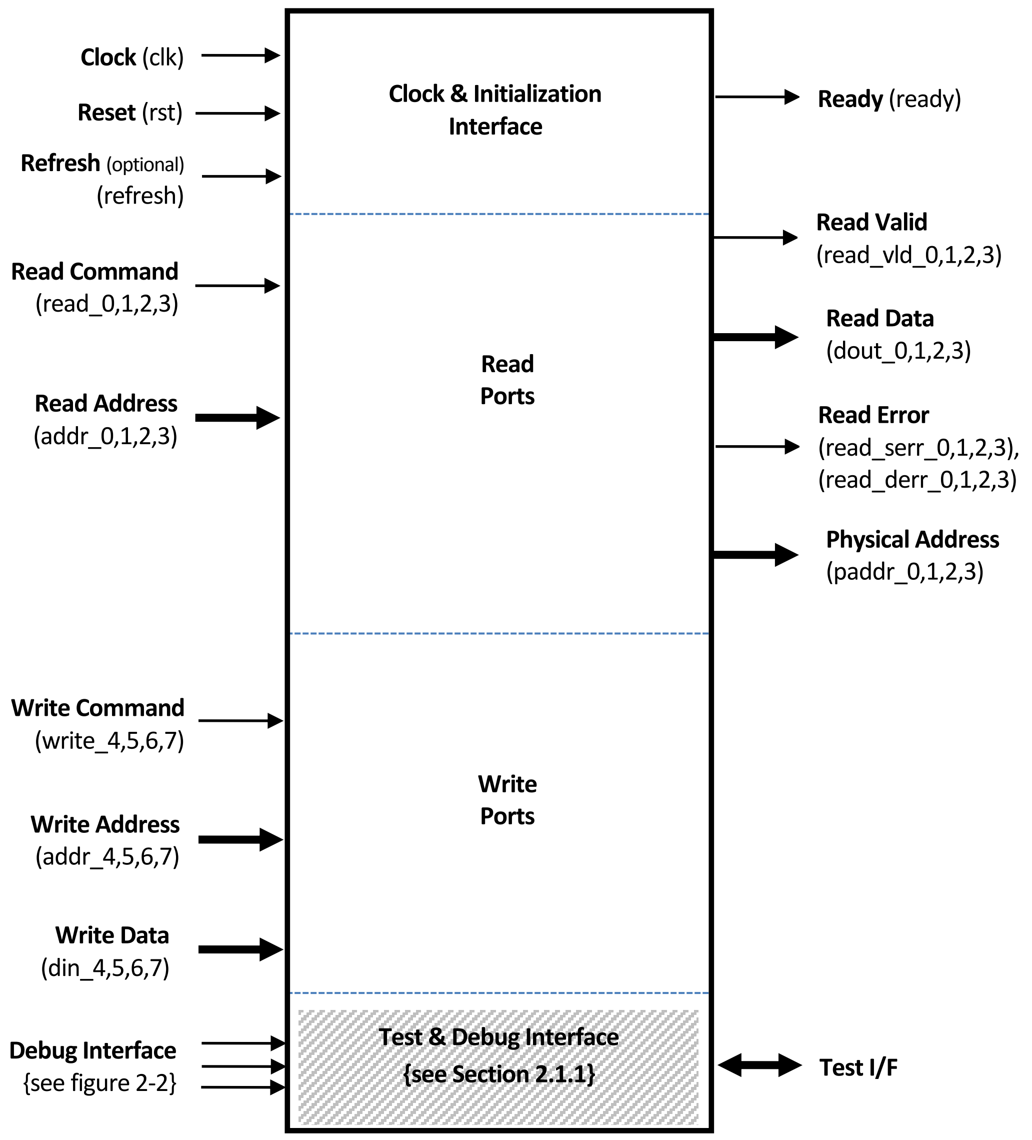

Clock (clk)
Reset (rst)
Refresh (optional) (refresh)
Read Command (read_0,1,2,3)
Read Address (addr_0,1,2,3)
Write Command (write_4,5,6,7)
Write Address (addr_4,5,6,7)
Write Data (din_4,5,6,7)
Debug Interface {see figure 2-2}
Clock & Initialization Interface
Read Ports
Write Ports
Test & Debug Interface {see Section 2.1.1}
Ready (ready)
Read Valid (read_vld_0,1,2,3)
Read Data (dout_0,1,2,3)
Read Error (read_serr_0,1,2,3), (read_derr_0,1,2,3)
Physical Address (paddr_0,1,2,3)
Test I/F

Figure 2-1: Memory Core Block Diagram

2.1. Interface Description

Table 2-1 describes the interface pins of a memory core shown in Figure 2-1.

Table 2-1: Interface Pins

Pin	Pin Type	Description	Notes
Clock and Initialization			
clk	Input	Clock	Clock input pin
lclk	Input	Slow Clock	Slow Clock input pin
rst	Input	Reset	Reset pin
refresh	Input	Refresh	Signal to the controller that eDRAM can be refreshed in this cycle. Only needed when IP uses eDRAM. (Optional)
ready	Output	Ready	Assertion implies memory core ready for functional operation
Read Ports			
addr_{0,1,2,3}[1][2]	Input	Read Address	Address bits (width N bits) to denote up to 2^N addresses; N is parameter. Same value for all ports
read_{0,1,2,3}	Input	Read Valid	Denotes that a Read command is valid
dout_{0,1,2,3}	Output	Read Data	Data (width W) associated with a read address; W is parameter. Same value for all ports
read_vld_{0,1,2,3}	Output	Read Data Valid	Denotes valid read data
read_serr_{0,1,2,3}, read_derr_{0,1,2,3}	Output	Read Error	Denotes read data error. Data on *dout* pins is invalid
paddr_{0,1,2,3}	Output	Physical Address	Physical Address (width P bits) associated with the read. Parameter P depends on the physical address space of the memory core
Write Ports			
addr_{4,5,6,7,8}[3]	Input	Write Address	Address bits (width N bits). Parameter N is the same as that used for Read address.
write_{4,5,6,7,8}	Input	Write Valid	Denotes that a write command is valid
din_{4,5,6,7,8}	Input	Write Data	Data (width W bits) associated with a write command; Parameter W is same that for Read data.

[1] For compactness, port signal names are appended with '_{w, x, y, z}' . **Each appended signal name implies four independent port signal names.** This nomenclature is followed throughout the document.
[2]For 'n' RW cores (n=2,3,4), the port numbering is same for read and write ports. Further, the address busses are common. E.g. for a 2RW core, only *addr_0*, *addr_1* are offered. The other signals are *read_0,1, write_0,1, dout_0,1*, *paddr_0,1* and *din_0,1*.
[3]For cores with less than 4 read ports, the write port numbering begins where the read port numbering ends. E.g. for a core with 2 read ports and 1 write port, the numbering scheme would be read_0, read_1, write_2.

The different interfaces are described next.

- **Clock and Initialization pins:** This set of signals is common to all ports. Clock is provided to the memory core via the *clk* input pin. The frequency is user determined. The clock pin is common to all the ports indicating full synchronous operation across the memory core (Multi-clock domain operation across ports is currently not supported).

  The memory core is initialized by asserting the *rst* pin for a minimum of ten clocks. The completion of reset process is indicated by assertion of the *ready* pin. The mRnW memory core is ready for normal operations on the next clock after this assertion.

- **Read Interface:** Each read port consists of read command (read_{0,1,2,3}) and a corresponding read address (*addr*_{0,1,2,3}). The read data (*dout*_{0,1,2,3}) is available after a fixed latency along with the read valid signal (*read_vld*_{0,1,2,3}). The latency of a read access is fixed irrespective of operations on other ports. The *read_vld* signal is provided as a timing signal to enable the host chip to accept valid data. This way the host chip does not have to hardwire the latency value in its logic.

  Read Error: The assertion of read error signal indicates detection of ECC error in the internal data structures implemented in the core. Their occurrences are reported with the assertion of *read_serr* (single-bit error)*,* or *read_derr* (double-bit error) outputs. The read physical address (described next) points to the physical memory location where the error occurred. The user is required to reset the memory core and initiate the reset process

  Read error events happen only for certain memory cores, depending on the underlying algorithm. The core-specific datasheet states if the Read error outputs are used for that core. If not, these outputs are held inactive (low) throughout the course of operation.

  Read Physical Address: ECC implementation on the user data is the responsibility of the user. The user can choose to do this as part of data itself. Due to address mapping schemes deployed within the core, the physical location of data for a given user read address may change from access to access. The physical address of the read location (*read_padr*_{0,1,2,3}) is provided to help with address logging in case the user chooses to maintain statistics on ECC error events. The mapping between the physical address and physical memories (reference-designators) is detailed in the memory core-specific datasheet.

  Note: The read Physical address is mainly used during normal operation to help the user track any ECC error events seen in the data. But for those cores where a Read error event can happen, the physical address also points to the erring memory location during such an event.

- **Write Interface:** The interface accepts a write command (*write*_{4,5,6,7,8}), a write address (*addr*_{4,5,6,7,8}) and the corresponding write data (*din*_{4,5,6,7,8}).

2.1.1. Debug and Test Interface with description

Figure 2-2 shows the Test & Debug interface of the Renaissance family memory IP.

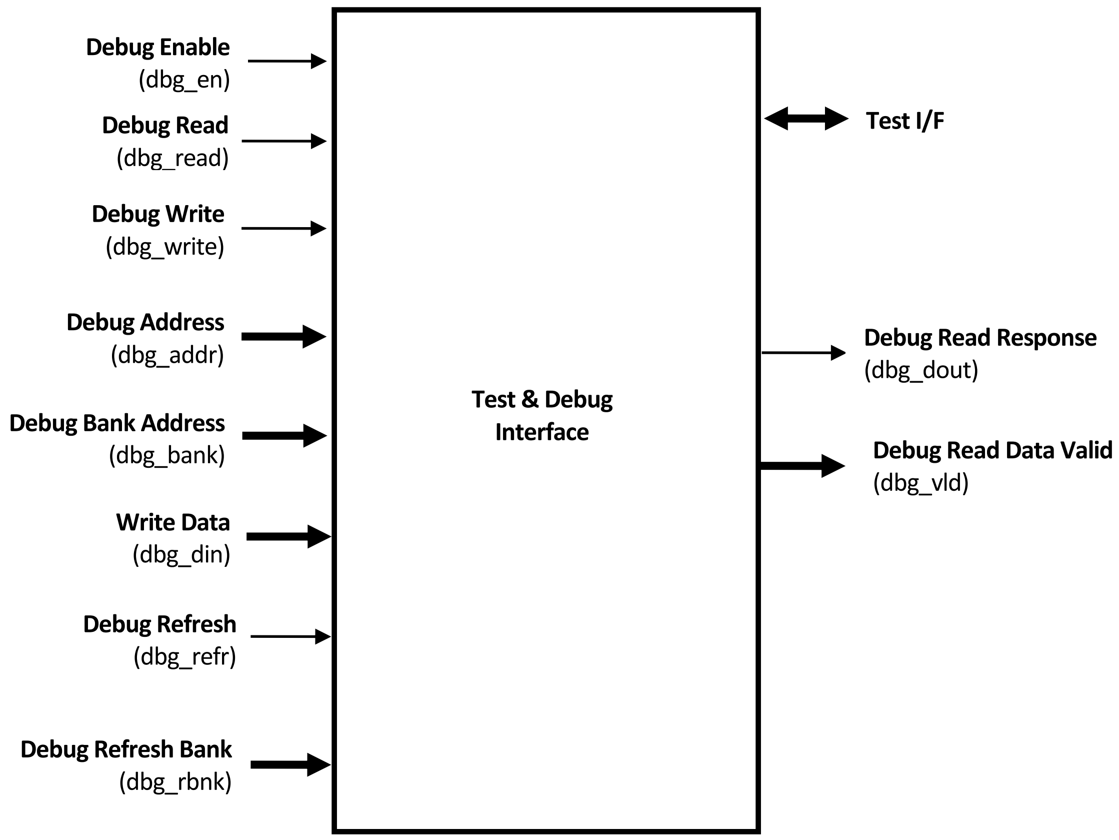


Figure 2.2: Debug & Test Interface

- **Test I/F:** This set of signals is common to all ports. Memoir memory cores support all Test and Diagnostics features (including M-BIST) offered by the vendor of the physical library. All the associated pins are direct passed-thru to the physical memories from the Test Interface.

- **Debug Interface:** This interface is used to directly access the physical memory macros. The Memoir logic is bypassed during these accesses. This interface could be used to perform direct read and write operation to the physical memory macros. If a bank is put in debug mode it cannot be used in functional mode. Some of the interface pins can be shared with the DFT (BIST) pins as needed.

  In order to enter the debug mode the application needs to assert the dbg_en signal. Along with the dbg_en signal a bank number corresponding to the physical macro being debugged is also provided on the dbg_bank bus. There can be different number of banks in different IPs. The IP specific datasheet has details on the number of banks present in specific IP. When a physical macro is put in debug mode the remaining physical macros can still be in operational in functional mode. (Refresh logic for the non-debug mode macros would still continue to operate in regular mode)

  The debug interface consists of a read command interface {dbg_read, dbg_addr} along with a read response interface {dbg _valid, dbg_dout}. The assertion of dbg_read signal implies a valid read operation request. The address of the read is split in two portions – read bank address (dbg_bank) and the read word address (dbg_addr). The dbg_bank corresponds to the physical bank being read and the dbg_addr corresponds to the word within the bank that is being read.

The debug interface also has a write command interface {dbg_write, dbg_addr, dbg_din}. The address signals area shared between the debug read and write commands i.e. only one of the two operations (debug read or debug write) can occur in one cycle. The debug write address is split in two portions – write bank address (dbg_bank) and the write word address (dbg_addr). The dbg_bank corresponds to the physical macro being written to and the dbg_addr corresponds to the word within the physical macro that is being written. The physical bank architecture is tied to the algorithmic memory being used. The detailed organization of physical memory macros is captured in the memory core-specific datasheet.

Debug interface also has a refresh command interface {dbg_refr, dbg_rbnk}. Refresh command is only used for the eDRAM banks used in the algorithmic memory IP. If an eDRAM bank is in debug mode than the user logic is responsible to for generating refresh commands to the eDRAM bank being debugged. The refresh command is launched using the dbg_refr signal. The dbg_rbnk bus identifies the sub-bank within the dbg_bank to be refreshed.

The application has complete control of all the debug data bits in this mode and the IP does not generate or check ECC bits during debug mode operations.

Table 2-12 describes the debug interface pins of a memory core.

Table 2-2: Interface Pins (cont.)

Pin	Pin Type	Description	Notes
Debug & Test Interface			
Test I/F	I/O	Test Interface pins	Test/Diagnostic pins associated with base memories
dbg_en	Input	Debug Enable	Denotes that the debug logic is enabled
dbg_read	Input	Debug Read	Denotes that read command (for the debug port) is valid
dbg_write	Input	Debug Write	Denotes that write command (for the debug port) is valid
dbg_addr	Input	Debug Address	Address (word) of the debug operation
dbg_bank	Input	Debug Bank Address	Address (bank) of the debug operation. Actual number of banks in different IP's can vary.
dbg_din	Input	Debug Write Data	Data for the debug port write operation. The width of this bus dependent on the IP and the type of bank being accessed.
dbg_dout	Output	Debug Read Response	Carries back the data for the debug read operation.
dbg_vld	Output	Debug Read Data Valid	Denotes valid read data on the *dbg_dout* port
dbg_refr	Input	Debug Refresh	Denotes a valid refresh command being issued to the bank under debug.
dbg_rbnk	Input	Debug Refresh Bank	Denotes the sub-bank within the dbg_bank to be refreshed.

3. Interface Timing

Timing information for the various operations is presented in this section.

3.1. Initialization

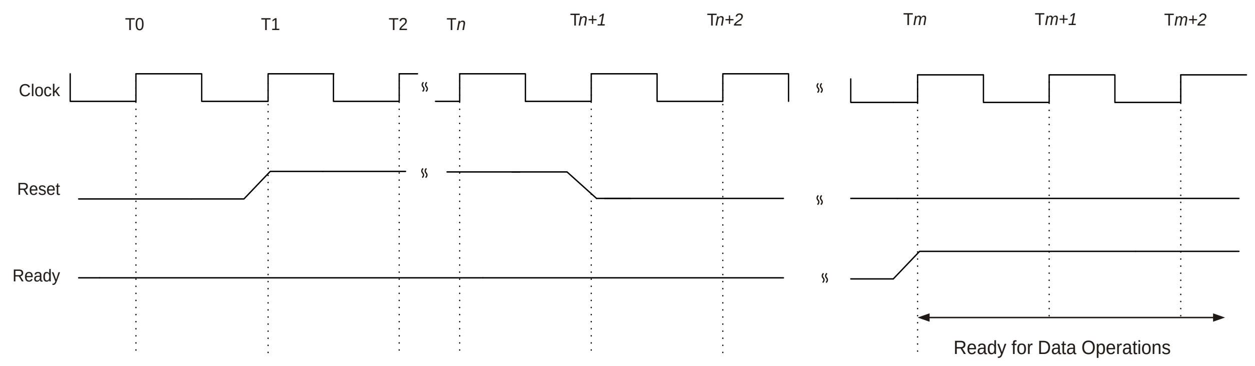


Figure 3-1: Reset Sequence

The reset sequence presented in Figure 3-1 brings the mRnW memory core out of reset. The *reset* pin is asserted for a minimum of 10 clock cycles.

The internal reset process begins after the de-assertion of reset. It completes a fixed time delay later. The completion is indicated by the assertion of *ready* pin. The *reset* to *ready* delay will depend on the size of memory core and the algorithm implemented. Its value is specified in the memory core-specific datasheet (auto-generated by the MemoGen^{TM} software). The host logic should wait until *ready* is asserted. The mRnW memory core is ready to accept data operations on the next clock after *ready* is asserted. The *ready* pin is kept asserted throughout the operation of the memory core. This is shown in the timing diagrams for the read and write operations in the following sections.

3.2. Read Timing

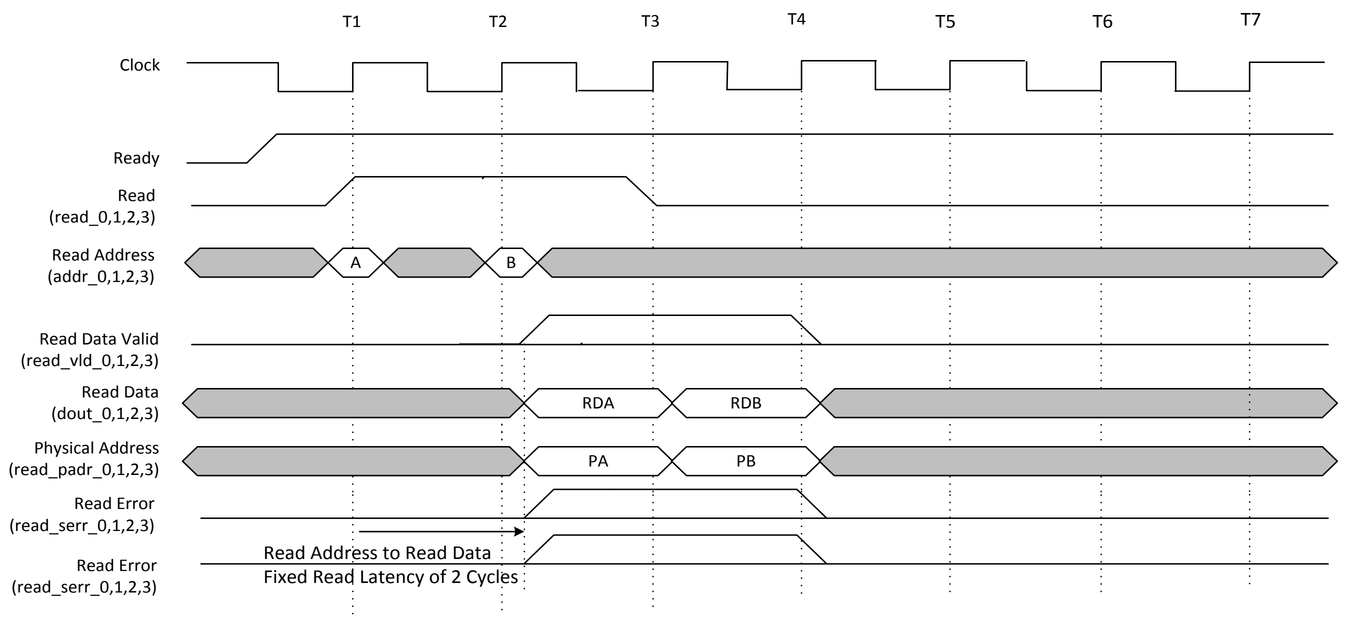


Figure 3-2: Read Interface Timing Diagram

The timing diagram in Figure 3-2 shows two read requests to addresses – A and B respectively. The first read operation reads address A in clock cycle 1. Data for A is returned in clock cycle 3. Similar operations happen for address B in clock cycle 2. Data for B is returned in clock cycle 4. The two read ports operate independently of each other. The signal names in the figure represent signals for ALL the read ports of the memory core.

A read latency of ‘2’ clocks is assumed in the illustrations above. The latency of a given memory core is always fixed, irrespective of port operations on other ports. It depends on the inherent latencies of the base-memories and the algorithm type implemented. The validity of data for requests A and B is indicated by the assertion of *read_vld* in clock cycles 3 and 4.

The mRnW sequential memory core does NOT provide error-protection for user data. The user is responsible for incorporating any error-detection/correction schemes as part of the data itself. The physical address (*read_padr_{0,1,2,3}*) output is provided to help the user with any address/data logging for ECC statistics maintenance on user data. The mapping between physical address and the underlying physical memories is provided in the memory core-specific datasheet. The physical addresses (PA and PB) for each read request are returned alongside their data output – in cycles 3 and 4 respectively.

3.3. Write Timing

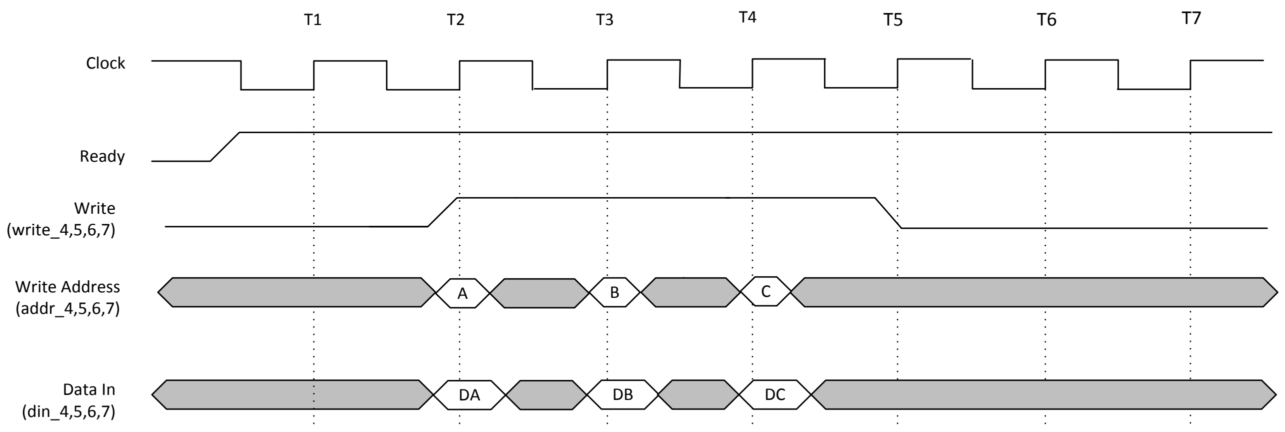


Figure 3-3: Write Interface Timing Diagram

The timing diagram in Figure 3-3 shows three write requests – A, B and C. The first write operation writes to address A in clock cycle 2 with data for A being written in the same cycle. The second write operation writes to address B in clock cycle 3. Similarly, the write operations to addresses C happens in clock cycle 4 with the corresponding data being written in the same cycles.

Like the read ports, the write ports operate independently of each other. The signal names in the figure represent signals for ALL the write ports.

3.4. Read and Write Interaction & Sequential Constraints

All read ports operate independently of all the other read ports and all write ports operate independently of the other write ports. However, there is a read to write delay that the read scheduler should adhere to. This delay is dependent on the specific implementation of the IP. The read scheduler needs to honor this delay.

Also there are specific read and write rate constraints that are defined as Seq (a, w) that the read and write schedulers should honor. The ‘a’ in Seq (a, w) refers to the number of expectable accesses in a moving window size of ‘w’ cycles to a partition of the address space. In other words, the sequential constraints are described as Seq (a, w) for a specific partition of the address space that could be accessed by the read or write port. The address space can be carved into a few partitions, each of them having its own set of constraints.

For example, on a read port there can be two sets of read constraints describing the access rate. Seq (1, 4) for address space A and Seq (1, 6) for address space B. Note that the union of address space A and B would give you the complete address space and there is no overlap in the two address spaces. They are mutually exclusive. With these set of contraints the read scheduler should not launch more than 1 read access to address space A within a moving window size of 4 cycles – Seq (1, 4). For the second constraint Seq (1, 6) the read scheduler should not launch more than 1 accesses to address space B within a moving window size of 6 cycles – Seq (1, 6).

Similar constrains are defined for write ports. For address space A Seq(4,12) and for address space B Seq(4 in 26). The CPU port is the lowest priority port with the constraints Seq(1,26) for both address space A and B.

3.5. Refresh Timing

This section is **optional**. It applies to only those memory cores which are built with eDRAMs as base memories.

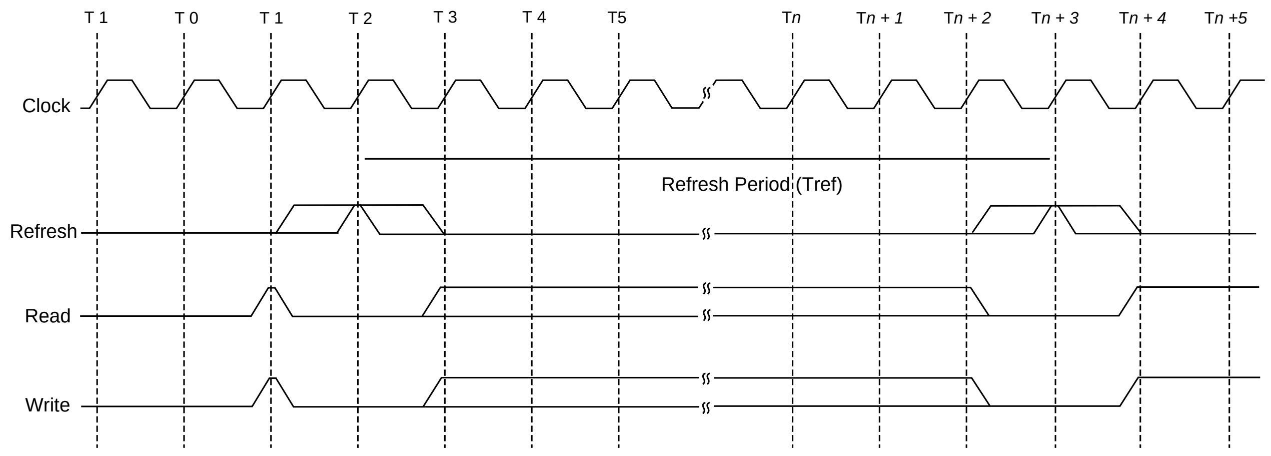


Figure 3-4: One-in-N Refresh Timing Diagram

Figure 3-4 shows the refresh operation for a One-in-N refresh scheme. Other refresh schemes for an eDRAM based memory core are possible. If implemented, they would be specific to a given memory core implementation and will be described in the memory core-specific datasheet.

A *refr* command to the mRnW memory core initiates a refresh operation on the eDRAM instantiated inside. The refresh-controller implemented in the memory core manages the refresh bank address sequencing and issues it internally on the eDRAM's address bus. The user is NOT required to issue any addresses externally at the memory core interface.

The only requirement on the user's side is to provide one refresh command within the Tref window. A variation of this scheme is to provide a minimum number of refresh commands (Nminref) within the Tref window. Nminref depends on the size of the instantiated eDRAM memory core. The values of Nminref and Tref are specified in the memory core-specific datasheet which is auto-generated by the MemoGen software.

The user should NOT assert either the *read* or *write* commands on any port when *refr* is asserted. Doing so will result in erroneous read or write operation.

3.6. Debug Timing

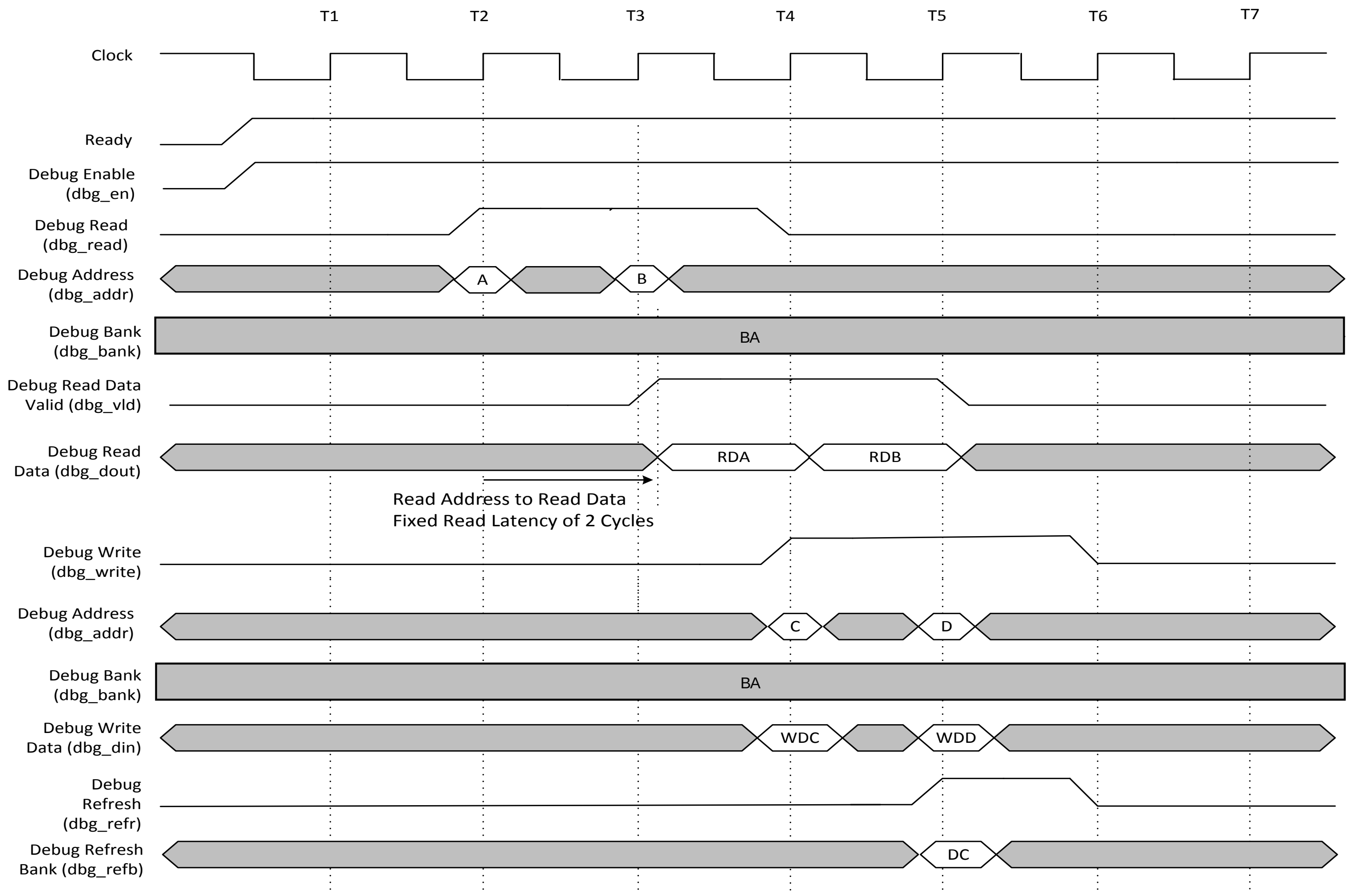


Figure 3-5: Debug Interface Timing Diagram

The above timing diagram shows the Algorithmic IP under debug mode. The assertion of dbg_en signal signifies that the IP is under debug mode. The bank being debugged is the bank BA. This information is carried on the dbg_bank bus.

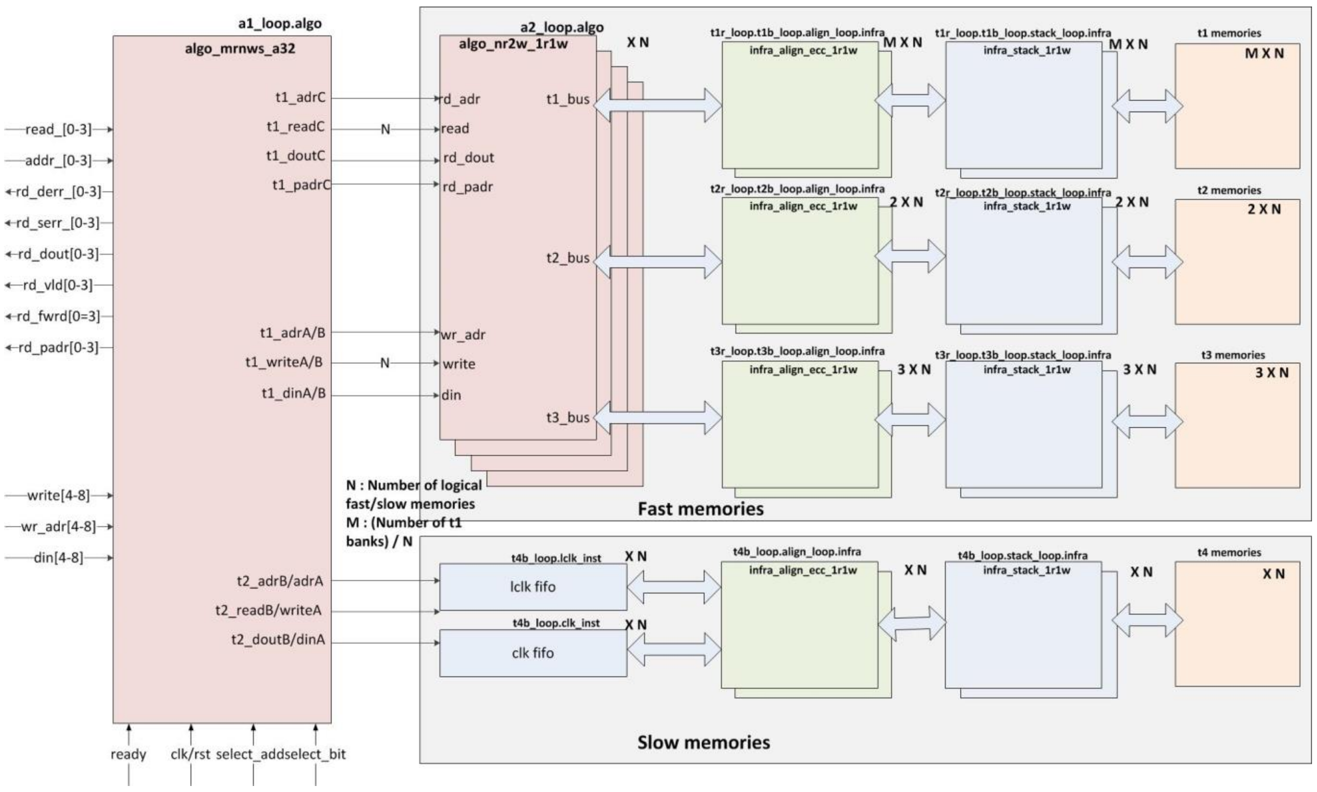
The figure demonstrates two debug read commands to read addresses ‘A’ and ‘B’ at cycle T2 and cycle T3. The read command is launched using the dbg_read signal and the corresponding address is provided on the dbg_addr bus. The bank being read from is ‘BA’ signified by the value on the dbg_bank bus. The data corresponding to these commands is provided back on the dbg_dout bus along with the valid signal dbg_vld.

The figure also demonstrates two debug write commands to write addresses ‘C’ and ‘D’ at cycle T4 and cycle T5. The write command is launched using the dbg_write signal and the corresponding address is provided on the dbg_addr bus. The bank being written to is ‘BA’ signified by the value on the dbg_bank bus. The data being written to is provided on the dbg_din bus.

A refresh command is needed for debug banks if the underlying physical memory used to implement the bank is eDRAM. (User is advised to look at the IP specific datasheet to find out this detail.) The assertion of the dbg_refr signal signifies a valid refresh command. The sub bank being refreshed is provided on the dbg_rbnk bus. In the figure above the sub bank being refreshed is ‘DC’. Note that the refresh command can be launched in a cycle with a valid read or write command (as shown in the figure). It can also be launched in a cycle with no read or write command being launched. The user logic is responsible for making sure that the sub bank being refreshed is not the same address that is being written to or read from. The high bits of the dbg_addr bus signifying the sub bank being accessed. (User is advised to look at the IP specific datasheet to find out this detail.)

4. Block diagram

The f32 provides the functionality of the memory buffer with 4 read ports and 5 write ports. It complies with the constraints described earlier in the document for the read and write traffic. The block diagram for the f32 memory ip is shown below.



5. Revision Change Log

Version	Date of Release	Notes
V1.0	March 17th 2012	Initial Release.
V1.1	Aug 17th 2012	Renaissance 4X, Read_Error functionality addition
V1.2	May 25th 2013	Added the debug interface functionality
V1.3	Apr 30th 2014	Added the block diagram
V1.4	May 1st 2014	Added constraints for inputs and outputs
V1.5	Aug 11th 2014	Fixed some typos in the block diagram

Memoir Systems Inc.

2350 Mission College Blvd. # 1275, Santa Clara, CA 95054

Phone: 1-408-550-2382

Email: support@memoir-systems.com