



ECTM Interface Specification

Document Number: MD00052

Revision 1.03

December 18, 2000

**MIPS Technologies, Inc.
1225 Charleston Road
Mountain View, CA 94043-1353**

Copyright © 2000 MIPS Technologies, Inc. All rights reserved.

Unpublished rights reserved under the Copyright Laws of the United States of America.

This document contains information that is proprietary to MIPS Technologies, Inc. (“MIPS Technologies”). Any copying, modifying of this information (in whole or in part) which is not expressly permitted in writing by MIPS Technologies is strictly prohibited. At a minimum, this information is protected under unfair competition laws and the expression of the information contained herein is protected under federal copyright laws. Violations thereof may result in criminal penalties and fines.

MIPS Technologies reserves the right to change the information contained in this document to improve function, design or otherwise. MIPS Technologies does not assume any liability arising out of the application or use of this information. Any license under patent rights or any other intellectual property rights owned by MIPS Technologies or third parties shall be conveyed by MIPS Technologies in a separate license agreement between.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government (“Government”), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies.

MIPS, R3000, R4000, R5000, R8000 and R10000 are among the registered trademarks of MIPS Technologies, Inc., and R4300, R20K, MIPS16, MIPS32, MIPS64, MIPS-3D, MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MDMX, SmartMIPS, 4K, 4Kc, 4Km, 4Kp, 5K, 5Kc, 20K, 20Kc, EC, MGB, SOC-it, SEAD, YAMON, ATLAS, JALGO, CoreLV and MIPS-based are among the trademarks of MIPS Technologies, Inc.

All other trademarks referred to herein are the property of their respective owners.

Table of Contents

1	Introduction	5
1.1	Features	5
1.2	Basic Operation	6
2	Signal List	7
3	Timing Diagrams	10
3.1	Single Read Transactions	10
3.2	Single Write Transactions	14
3.3	Back-to-back Read Transactions	18
3.4	Back-to-back Write Transactions	20
3.5	Read Transaction Followed by a Write Transaction	21
3.6	Write Transaction Followed by a Read Transaction	23
3.7	Burst Transactions	26
4	External Write Buffers	31
A	Endianess	31
B	Lower Address Bit Generation	33
C	Revision History	33

EC™ Interface Specification

1 Introduction

This document describes the EC™ interface designed for microprocessor cores. All MIPS cores implementing the EC interface comply to this specification. Implementation-specific details can be found in the documentation accompanying the core.

Use the EC interface to attach memory controllers, memory-mapped I/Os, etc. A bus controller must be included in cases where multiple slaves connect to the EC interface. [Figure 1](#) shows an example of the EC interface placement within a system.

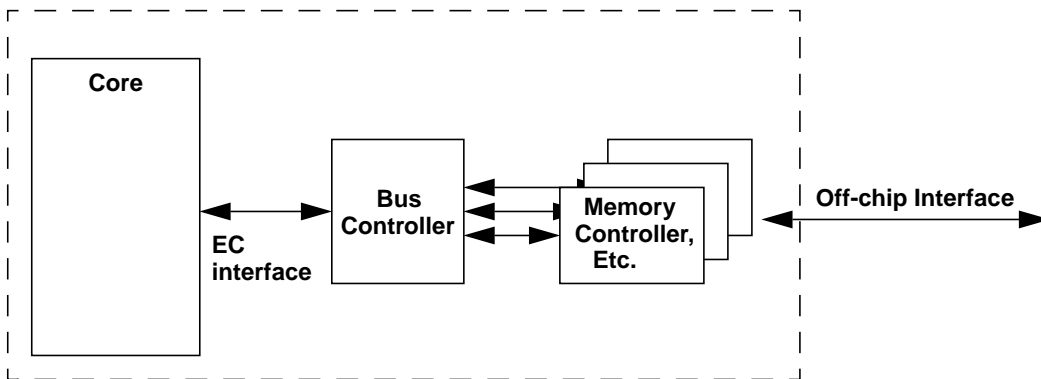


Figure 1 Example of the EC Interface in a System

1.1 Features

The EC interface has the following features:

- 32 or 64-bit data buses
- 36-bit addressing
- Separate read and write data buses
- All signals are unidirectional—no bidirectional or 3-state buses
- Fully registered, synchronous interface to the master
- Separate read and write bus error indications
- Separate address and data phases allow pipelining on the interface
- No limit on the number of outstanding transactions
- Number of outstanding transactions can be limited by the slave
- Support for variable burst length
- Sequential or sub-block ordering burst address sequences
- Indication of first and last address phase of a burst
- Request for emptying external write buffers and indication of external write buffers being empty
- Byte enable indication

- Indication of instruction read (fetch)
- Address and data phases can complete the same cycle they are initiated (zero wait states)
- No limit on the number of wait states in address and data phases
- Independent read and write data phases. A read transaction can overtake a write transaction and vice versa.
- Only one master and one slave

1.2 Basic Operation

All inputs to the master are sampled at the rising edge of the Clock signal. Further the outputs from the master change with respect to a rising edge of the Clock signal.

The EC interface does not include a signal to indicate reset. Therefore to reset the EC interface, reset the master and the slave simultaneously. Whenever the EC interface is reset, all transactions are aborted and the bus returns to the idle state. EB_ARdy, EB_AValid, EB_WDRdy, EB_RdVal, EB_Burst, EB_BFirst, EB_BLast, EB_RBErr, and EB_WBErr must be driven inactive during reset.

Each transaction on the EC interface has an *address phase* and a *data phase*, which can have a number of wait states.

A wait state in the address phase is named an *address wait state* and is defined as a clock cycle where EB_AValid is asserted and EB_ARdy was sampled deasserted in the beginning of the cycle.

An address phase begins in the clock cycle where the master asserts EB_AValid. An address phase ends on the positive clock edge following an asserted sample of EB_ARdy. For maximum speed (no address wait states), EB_ARdy has to be sampled asserted on the positive clock edge preceding the beginning of the address phase. During an address phase, all signals driven by the master are unchanged and stable (except from the write data bus, EB_WData).

Due to the separate read and write data buses, two types of data phases exist: the read data phase and the write data phase.

A wait state in a data phase is named a *data wait state*. It is defined as a clock cycle where the corresponding address phase has been started (and possibly ended) and:

- For a write data phase, EB_WDRdy is sampled deasserted at the beginning of the cycle
- For a read data phase, EB_RdVal is sampled deasserted at the end of the cycle

A read data phase begins in the clock cycle where the master starts the corresponding read address phase. However, if there are outstanding read data phases when the read address phase begins, the corresponding read data phase does not start until all of the preceding read data phases have ended. The read data phase ends at the positive clock edge where EB_RdVal is sampled asserted. It can not end earlier than when the corresponding address phase ends.

A write data phase begins in the clock cycle where the master starts the corresponding write address phase. However, if there are outstanding write data phases when the write address phase begins, the corresponding write data phase does not start until all of the preceding write data phases have ended. The write data phase ends at the positive clock edge following the positive clock edge where EB_WDRdy is sampled asserted. For maximum speed (no data wait states), EB_WDRdy must be asserted on the positive clock edge preceding the beginning of the corresponding address phase. It cannot end earlier than the corresponding address phase ends.

From these definitions, for a given transaction the number of data wait states must be greater than or equal to the number of address wait states.

2 Signal List

Table 2 lists all of the EC interface signals in alphabetical order. Table 1 defines the signal directions.

Table 1 Signal Direction Key

Dir	Description
I	Input to the master. Unless otherwise noted, input signals are sampled on the rising edge of the appropriate CLK signal.
O	Output from the master. Unless otherwise noted, output signals are driven on the rising edge of the appropriate CLK signal.
SI	Static input to the master. These signals are normally tied to either power or ground and should not change state while RESET is deasserted.

Table 2 EC Interface Signals

Signal Name	Dir	Description
EB_A[35:2]	O	Address bus. Only valid when EB_AValid is asserted. Note that only EB_A[35:3] address lines are used in 64-bit implementations.
EB_ARdy	I	Assertion of this signal indicates whether the slave is ready for a new address. The master does not complete the address phase until the clock cycle after EB_ARdy is sampled asserted.
EB_AValid	O	Assertion of this signal indicates that the values on the address bus and access type lines are valid (signifying an address phase is ongoing). EB_AValid is always valid and cannot be deasserted between address phases within a burst.

Table 2 EC Interface Signals (Continued)

Signal Name	Dir	Description																																																																											
EB_BE[3:0] EB_BE[7:4] ^a	O	<p>Indicates which bytes of the EB_RData or EB_WData buses are involved in the data phase corresponding to the current address phase. If an EB_BE signal is asserted, the associated byte is being read or written. EB_BE lines are only valid while EB_AValid is asserted.</p> <p>During bursts all lines must be asserted.</p> <p>During single transactions, if the master supports EB_BE patterns other than the default ones listed in the two tables below, it must have an input signal that makes it possible to force it into using the default patterns only.</p> <table border="1"> <thead> <tr> <th colspan="4">Byte enables supported by default in 64-bit implementations</th> </tr> </thead> <tbody> <tr> <td>00000001</td> <td>00000010</td> <td>00000100</td> <td>00001000</td> </tr> <tr> <td>00010000</td> <td>00100000</td> <td>01000000</td> <td>10000000</td> </tr> <tr> <td>11000000</td> <td>00110000</td> <td>00001100</td> <td>00000011</td> </tr> <tr> <td>11100000</td> <td>01110000</td> <td>00001110</td> <td>00000111</td> </tr> <tr> <td>11110000</td> <td>00001111</td> <td>11111000</td> <td>00011111</td> </tr> <tr> <td>11111100</td> <td>00111111</td> <td>11111110</td> <td>01111111</td> </tr> <tr> <td>11111111</td> <td></td> <td></td> <td></td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th colspan="4">Byte enables supported by default in 32-bit implementations</th> </tr> </thead> <tbody> <tr> <td>0001</td> <td>0010</td> <td>0100</td> <td>1000</td> </tr> <tr> <td>1100</td> <td>0011</td> <td>0111</td> <td>1110</td> </tr> <tr> <td>1111</td> <td></td> <td></td> <td></td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>EB_BE Signal</th> <th>Read Data Bits Sampled</th> <th>Write Data Bits Driven Valid</th> </tr> </thead> <tbody> <tr> <td>EB_BE[0]</td> <td>EB_RData[7:0]</td> <td>EB_WData[7:0]</td> </tr> <tr> <td>EB_BE[1]</td> <td>EB_RData[15:8]</td> <td>EB_WData[15:8]</td> </tr> <tr> <td>EB_BE[2]</td> <td>EB_RData[23:16]</td> <td>EB_WData[23:16]</td> </tr> <tr> <td>EB_BE[3]</td> <td>EB_RData[31:24]</td> <td>EB_WData[31:24]</td> </tr> <tr> <td>EB_BE[4]^a</td> <td>EB_RData[39:32]</td> <td>EB_WData[39:32]</td> </tr> <tr> <td>EB_BE[5]^a</td> <td>EB_RData[47:40]</td> <td>EB_WData[47:40]</td> </tr> <tr> <td>EB_BE[6]^a</td> <td>EB_RData[55:48]</td> <td>EB_WData[55:48]</td> </tr> <tr> <td>EB_BE[7]^a</td> <td>EB_RData[63:56]</td> <td>EB_WData[63:56]</td> </tr> </tbody> </table>	Byte enables supported by default in 64-bit implementations				00000001	00000010	00000100	00001000	00010000	00100000	01000000	10000000	11000000	00110000	00001100	00000011	11100000	01110000	00001110	00000111	11110000	00001111	11111000	00011111	11111100	00111111	11111110	01111111	11111111				Byte enables supported by default in 32-bit implementations				0001	0010	0100	1000	1100	0011	0111	1110	1111				EB_BE Signal	Read Data Bits Sampled	Write Data Bits Driven Valid	EB_BE[0]	EB_RData[7:0]	EB_WData[7:0]	EB_BE[1]	EB_RData[15:8]	EB_WData[15:8]	EB_BE[2]	EB_RData[23:16]	EB_WData[23:16]	EB_BE[3]	EB_RData[31:24]	EB_WData[31:24]	EB_BE[4] ^a	EB_RData[39:32]	EB_WData[39:32]	EB_BE[5] ^a	EB_RData[47:40]	EB_WData[47:40]	EB_BE[6] ^a	EB_RData[55:48]	EB_WData[55:48]	EB_BE[7] ^a	EB_RData[63:56]	EB_WData[63:56]
Byte enables supported by default in 64-bit implementations																																																																													
00000001	00000010	00000100	00001000																																																																										
00010000	00100000	01000000	10000000																																																																										
11000000	00110000	00001100	00000011																																																																										
11100000	01110000	00001110	00000111																																																																										
11110000	00001111	11111000	00011111																																																																										
11111100	00111111	11111110	01111111																																																																										
11111111																																																																													
Byte enables supported by default in 32-bit implementations																																																																													
0001	0010	0100	1000																																																																										
1100	0011	0111	1110																																																																										
1111																																																																													
EB_BE Signal	Read Data Bits Sampled	Write Data Bits Driven Valid																																																																											
EB_BE[0]	EB_RData[7:0]	EB_WData[7:0]																																																																											
EB_BE[1]	EB_RData[15:8]	EB_WData[15:8]																																																																											
EB_BE[2]	EB_RData[23:16]	EB_WData[23:16]																																																																											
EB_BE[3]	EB_RData[31:24]	EB_WData[31:24]																																																																											
EB_BE[4] ^a	EB_RData[39:32]	EB_WData[39:32]																																																																											
EB_BE[5] ^a	EB_RData[47:40]	EB_WData[47:40]																																																																											
EB_BE[6] ^a	EB_RData[55:48]	EB_WData[55:48]																																																																											
EB_BE[7] ^a	EB_RData[63:56]	EB_WData[63:56]																																																																											
EB_BFirst	O	Assertion of this signal indicates the address phase is the first address phase of a burst. EB_BFirst is always valid.																																																																											
EB_BLast	O	Assertion of this signal indicates the address phase is the last address phase of a burst. Note that the time for assertion of EB_BLast is determined by use of EB_Burst, EB_BFirst, and EB_BLen. EB_BLast is always valid.																																																																											
EB_BLen[1:0]	O	<p>EB_BLen[1:0] indicate the length (number of address/data phases) of the burst. This signal is an implementation-specific static output.</p> <table border="1"> <thead> <tr> <th>EB_BLength[1:0]</th> <th>Burst Length</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>reserved</td> </tr> <tr> <td>1</td> <td>4</td> </tr> <tr> <td>2</td> <td>8</td> </tr> <tr> <td>3</td> <td>reserved</td> </tr> </tbody> </table>	EB_BLength[1:0]	Burst Length	0	reserved	1	4	2	8	3	reserved																																																																	
EB_BLength[1:0]	Burst Length																																																																												
0	reserved																																																																												
1	4																																																																												
2	8																																																																												
3	reserved																																																																												

Table 2 EC Interface Signals (Continued)

Signal Name	Dir	Description
EB_Burst	O	Assertion of this signal indicates that the current address phase is for a cache fill or a write burst. EB_Burst is always valid.
EB_EWBE	I	Indicates that all external write buffers are empty. The external write buffers must deassert EB_EWBE in the cycle following the assertion of the corresponding EB_WDRdy and keep EB_EWBE deasserted until the external write buffers are empty. See Section 4, "External Write Buffers" on page 31 for more details.
EB_Instr	O	Assertion of this signal indicates that the address is for an instruction fetch as opposed to a data read. EB_Instr is only valid when EB_AValid is asserted.
EB_RBErr	I	Bus error indicator for read transactions. EB_RBErr is always valid. Only assert it in the same cycle that the corresponding EB_RdVal is asserted.
EB_RData[31:0] EB_RData[63:32] ^a	I	Read data bus. Valid at the end of a read data phase (on the rising clock edge where EB_RdVal is sampled asserted).
EB_RdVal	I	Assertion of this signal indicates that the slave is driving read data on EB_RData (it ends a read data phase). EB_RdVal must always be valid. EB_RdVal must never be asserted until after the corresponding EB_ARdy is sampled asserted.
EB_SBlock	SI	When this signal is asserted, sub-block ordering is used for addressing bursts. When this signal is deasserted, sequential addressing is used. See Section 3.7, "Burst Transactions" on page 26 for details.
EB_WBErr	I	Bus error indicator for write transactions. EB_WBErr is always valid. Only assert it in the cycle following an asserted sample of the corresponding EB_WDRdy.
EB_WData[31:0] EB_WData[63:32] ^a	O	Write data bus. Kept unchanged and stable during a write data phase until the write data phase ends (the positive clock edge following an asserted sample of EB_WDRdy).
EB_WDRdy	I	Assertion of this signal indicates that the slave is ready to process a write; it ends a write data phase and the EB_WData can change after the positive clock edge that follows the positive clock edge where EB_WDRdy is sampled asserted. EB_WDRdy is not sampled until the rising edge where the corresponding EB_ARdy is sampled asserted.
EB_Write	O	Assertion of this signal indicates that the address phase is for a write. Deassertion of this signal indicates that the address phase is for a read. This signal is only valid when EB_AValid is asserted.
EB_WWBE	O	Assertion of this signal indicates that the master is waiting for external write buffers to empty. EB_WWBE can be asserted when EB_EWBE is asserted, but if EB_EWBE is deasserted and EB_WWBE is asserted, EB_EWBE must be asserted eventually. See Section 4, "External Write Buffers" on page 31 for more details.

a. Optional. Only used in 64-bit implementations.

3 Timing Diagrams

The following sections provide examples of typical EC interface timing.

3.1 Single Read Transactions

Figure 2 shows the basic timing relationships between signals during a simple (fastest) read transaction. When the master is ready to begin a bus transaction (cycle 3), the address is driven on EB_A, the associated control information is driven on EB_Instr, EB_Burst, EB_BFirst, EB_BLast, EB_BLen, EB_Write, and EB_BE, and EB_AValid is asserted. On the same rising clock edge that these signals are driven out (end of cycle 2), the EB_ARdy signal state is sampled. If EB_ARdy is sampled deasserted, the master maintains the transaction values on the previously mentioned signals. The master continues driving valid and stable values on these interface signals until the rising clock edge following the one that the EB_ARdy signal is sampled asserted.

Starting in the same cycle as the read transaction is initiated, the master samples EB_RdVal, EB_RData, and EB_RBErr. These signals are sampled on each rising clock edge until the EB_RdVal signal is sampled asserted. The data values sampled with this asserted EB_RdVal are considered valid. However, if EB_RBErr was sampled asserted in same cycle, the transaction is considered failed.

Note that the data phase cannot end earlier than the corresponding address phase. EB_ARdy must be sampled asserted at least one clock cycle before the corresponding EB_RdVal is sampled asserted.

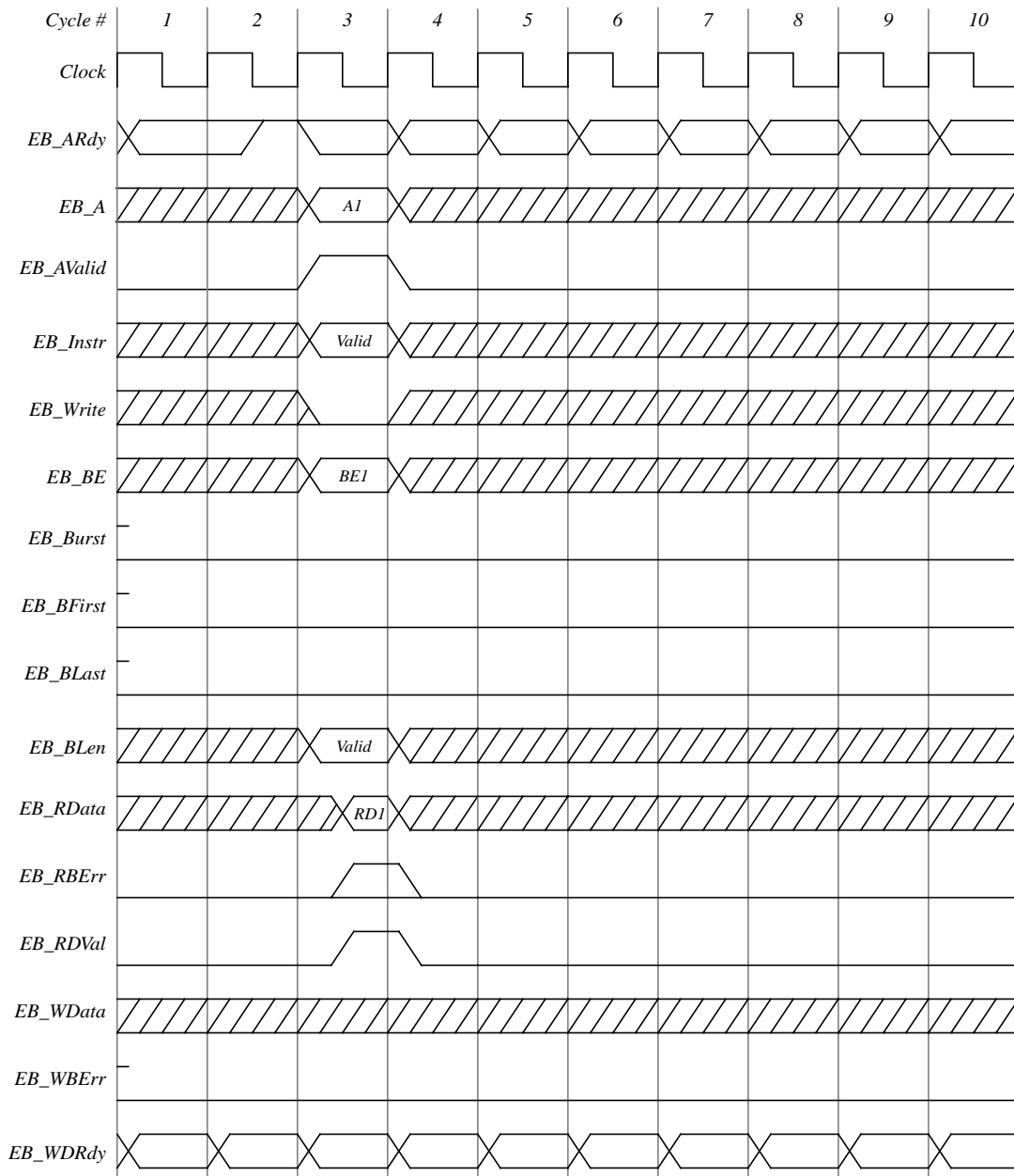


Figure 2 Fastest Single Read Transaction Timing

Figure 3 shows an example of a read transaction with three wait states in the data phase (indicated by the deassertion of EB_RdVal for three clock cycles). EB_RdVal is sampled deasserted on the rising edges at the beginning of cycles 4, 5, and 6 and then is asserted on cycle 7.

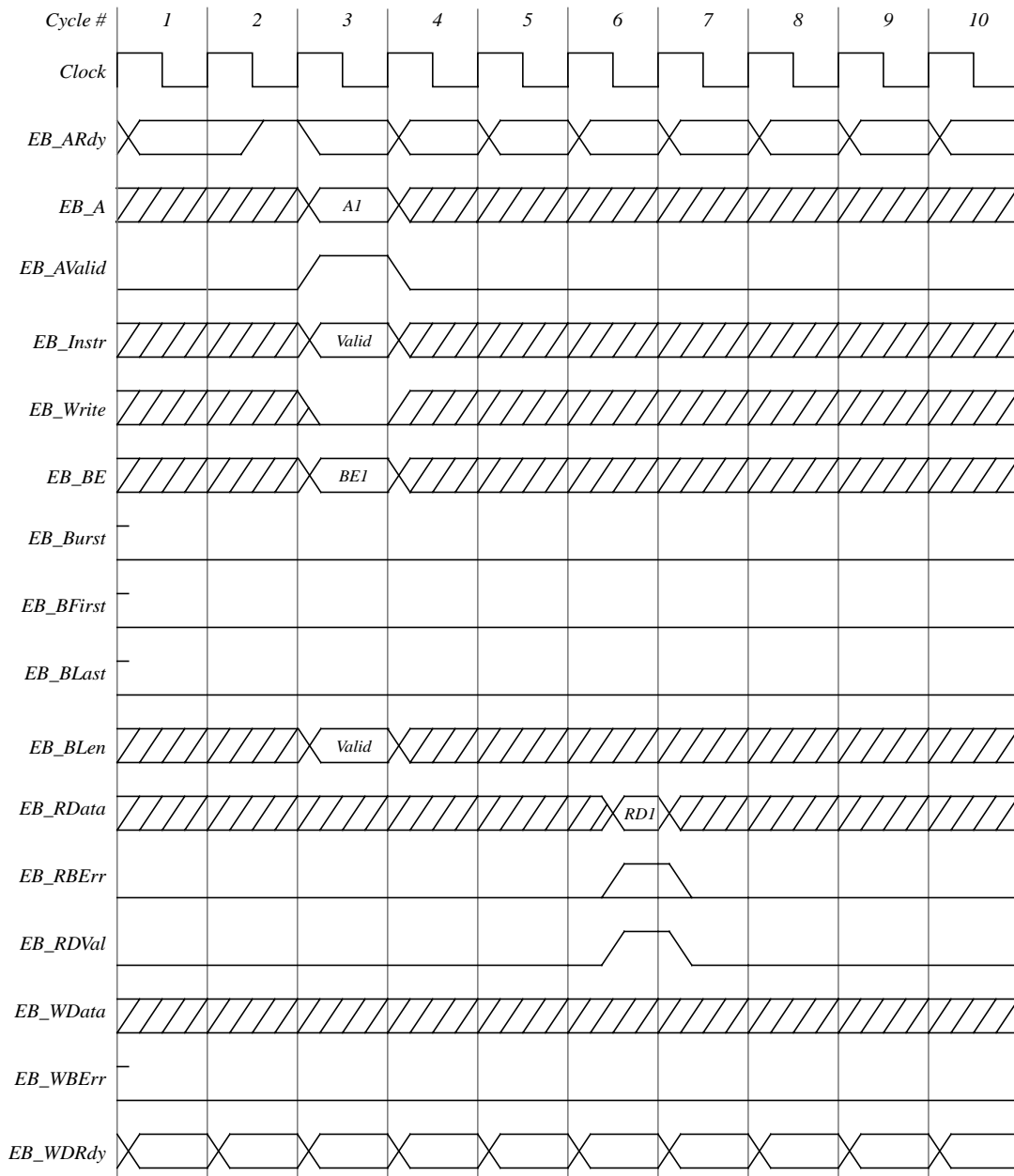


Figure 3 Single Read Transaction Timing (3 Data Wait States)

3.2 Single Write Transactions

Figure 4 shows a zero wait state (fastest) write transaction. Like the read transaction when a write request is issued (cycle 3), the address and control information for the transaction are driven on EB_A, EB_Instr, EB_Burst, EB_BFirst, EB_BLast, EB_BLen, EB_Write, and EB_BE. These signals remain unchanged until the rising clock edge after the EB_ARdy signal is sampled asserted.

The write data is driven on the write data bus, EB_WData, in same cycle as the address is driven on EB_A. The write data is held on the bus until the rising clock edge after EB_WDRdy is sampled asserted.

EB_WBErr is sampled on the first rising clock edge after the rising clock edge that EB_WDRdy is sampled asserted. If EB_WBErr is asserted at this time, the bus transaction is considered failed.

Note that the data phase cannot end earlier than the corresponding address phase. EB_WDRdy must be sampled asserted on the same clock edge or later than the clock edge where the corresponding EB_ARdy is sampled asserted.

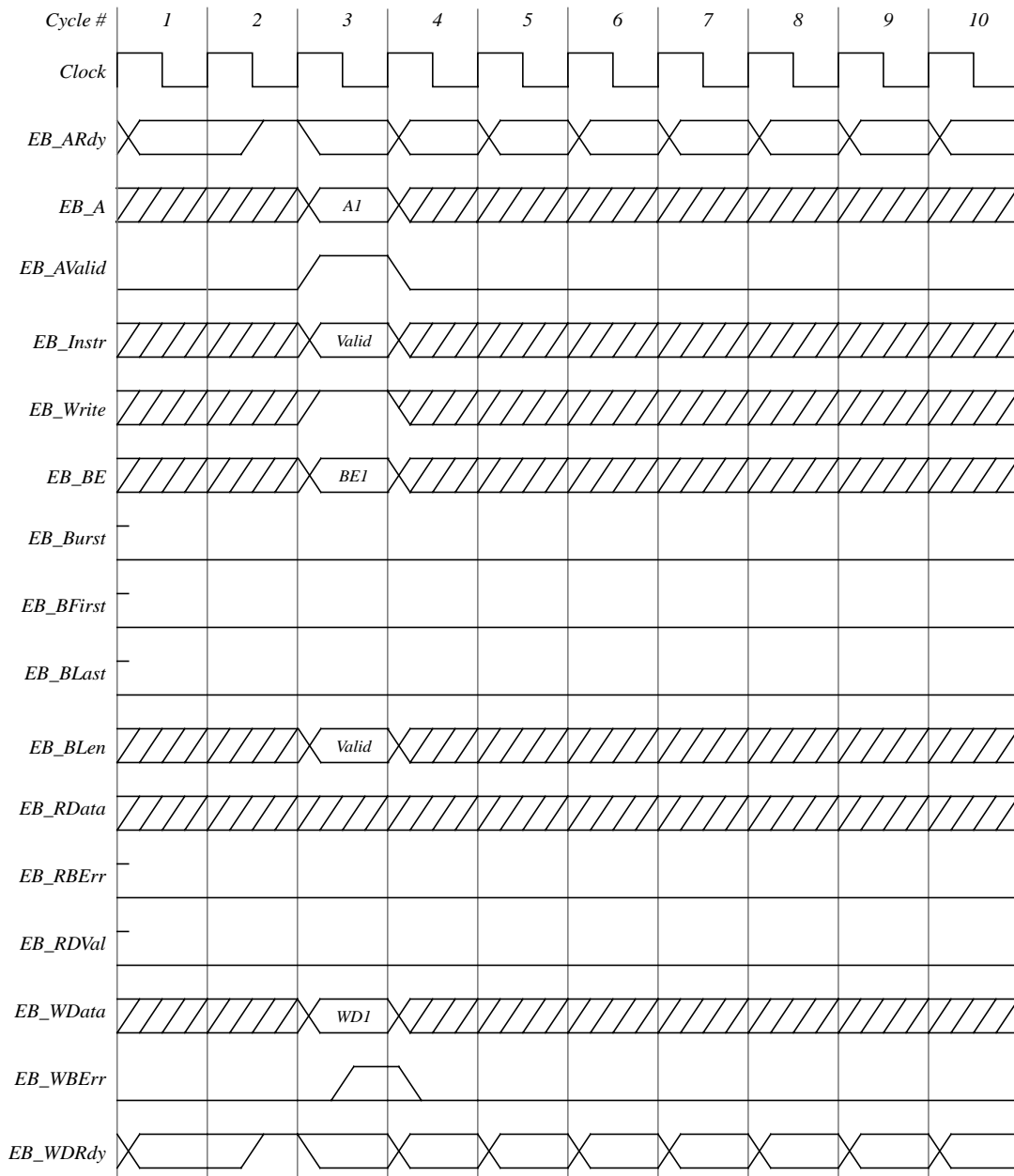


Figure 4 Fastest Single Write Transaction Timing

Figure 5 shows an example of a write transaction with four data wait states, indicated by the deassertion of the EB_WDRdy signal. EB_WDRdy is deasserted for four clock cycles and then asserted. Note that the address phase is prolonged by one clock cycle because EB_ARdy is deasserted for one clock cycle (sampled deasserted at the end of cycle 2).

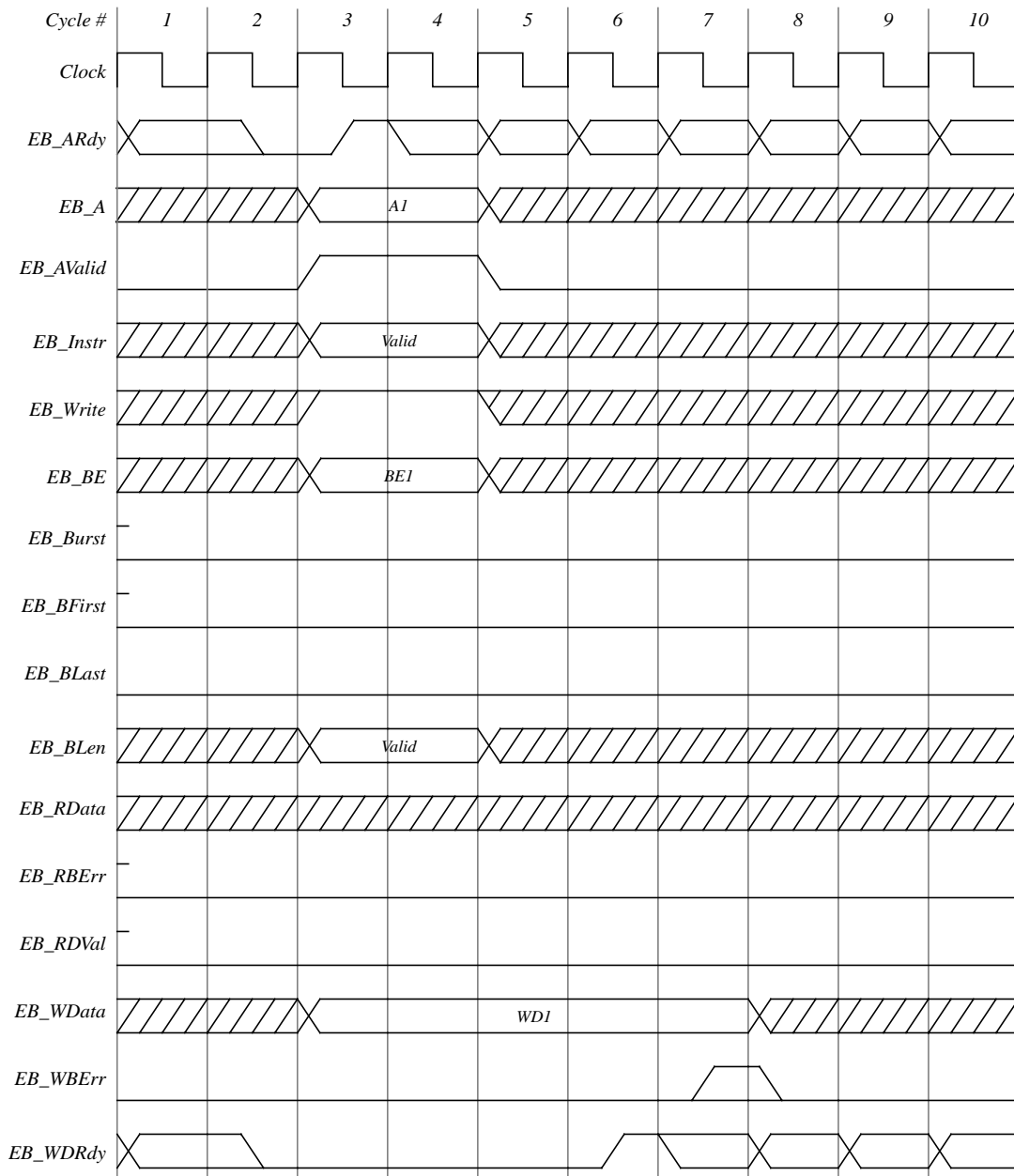


Figure 5 Single Write Transaction Timing (1 Address Wait State and 4 Data Wait States)

3.3 Back-to-back Read Transactions

Figure 6 shows an example of two consecutive read transactions, which shows the ability to pipeline read addresses independent of data wait states. The pipeline depth is implementation specific. Through manipulation of the EB_ARdy signal, the slave can limit the depth of the address pipelining.

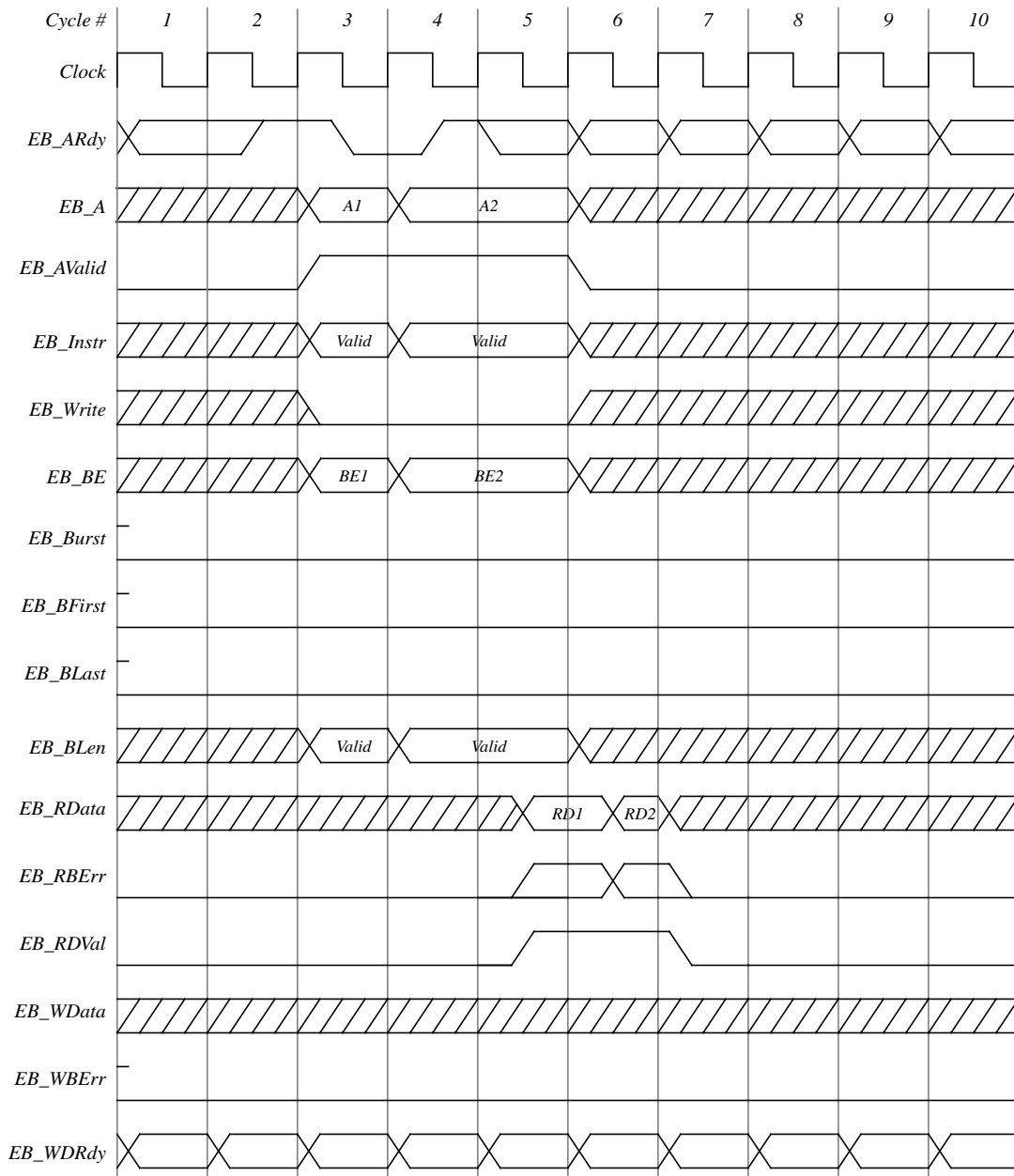


Figure 6 Back-to-back Read Transaction Timing

3.4 Back-to-back Write Transactions

Figure 7 shows an example of two consecutive write transactions. Similar to the read transactions, pipelining of write addresses can occur regardless of data wait states.

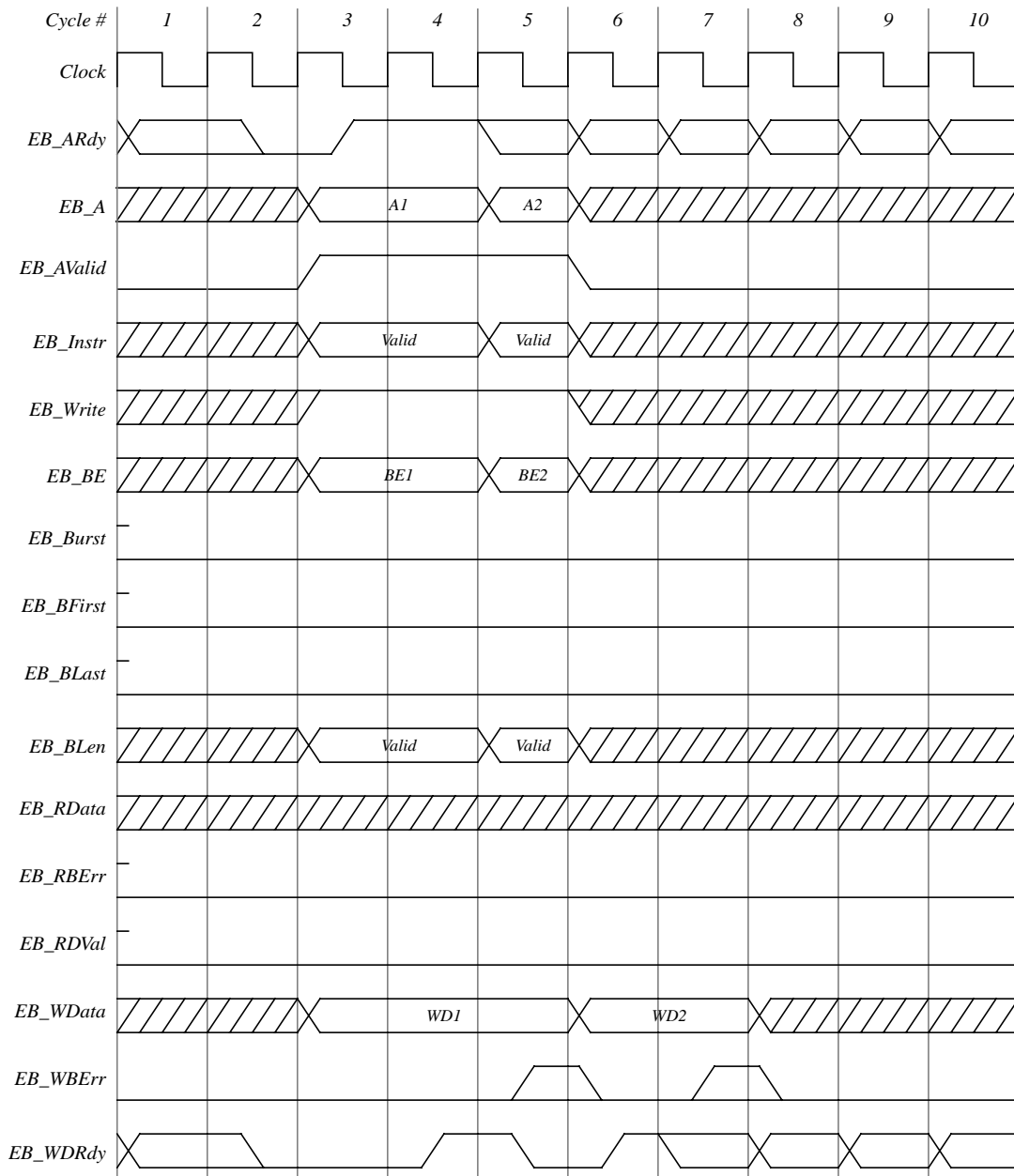


Figure 7 Back-to-back Write Transaction Timing

3.5 Read Transaction Followed by a Write Transaction

Figure 8 shows the relationship between a read transaction and a subsequent write transaction. A write transaction following a read transaction behaves as described for the single write transaction. Completion of these transactions out of order is allowed.

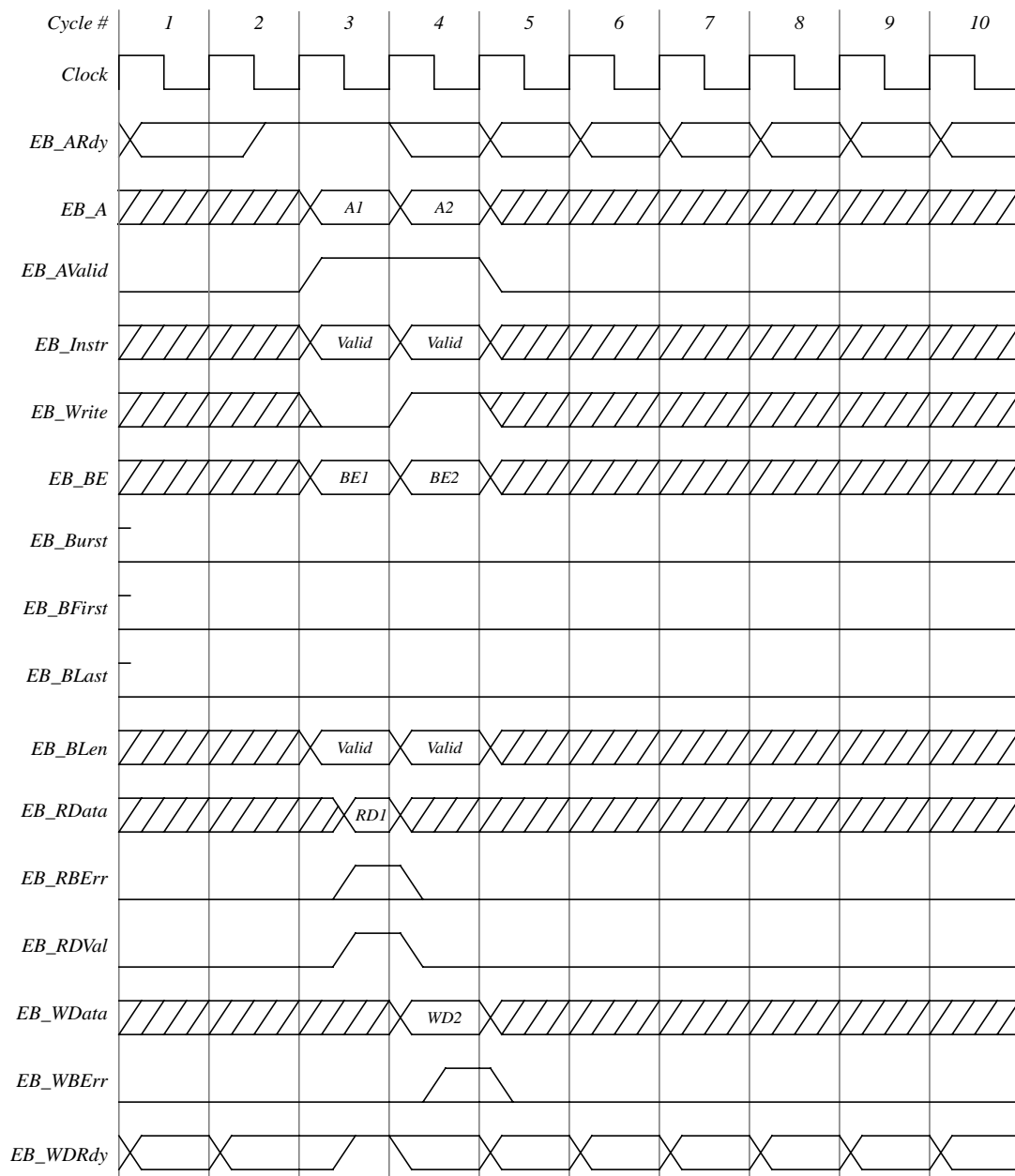


Figure 8 Read Transaction Followed by a Write Transaction

Figure 9 shows an example of a read transaction followed by a write transaction where the write transaction is completed prior to the read transaction (out of order).

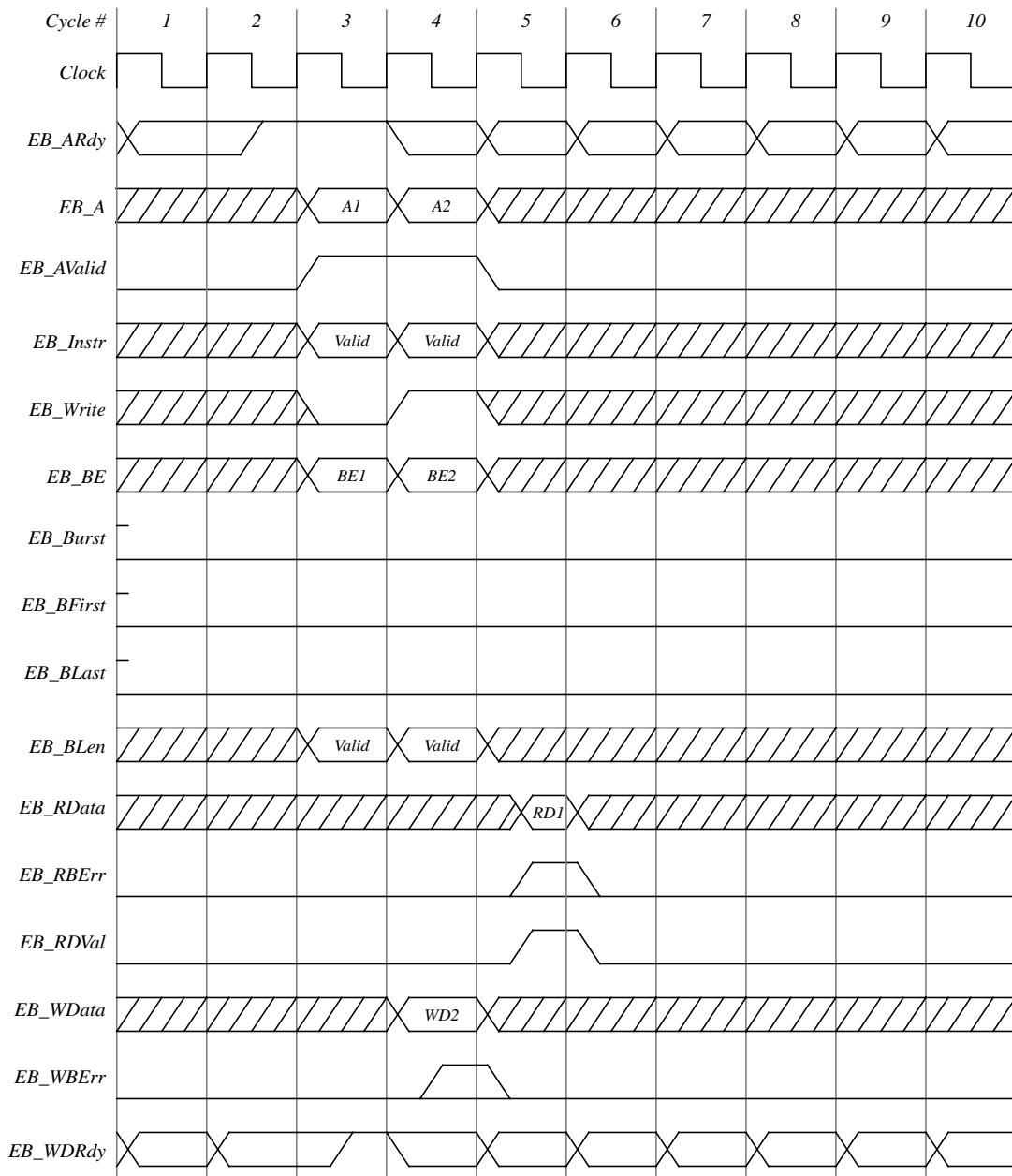


Figure 9 Read Transaction Followed by a Write Transaction with Reordering

3.6 Write Transaction Followed by a Read Transaction

Figure 10 shows an example of a write transaction followed by a read. As in the case of a write following a read, a read transaction following a write transaction is not affected by the behavior of the write transaction. Completion of these transactions out of order is allowed.

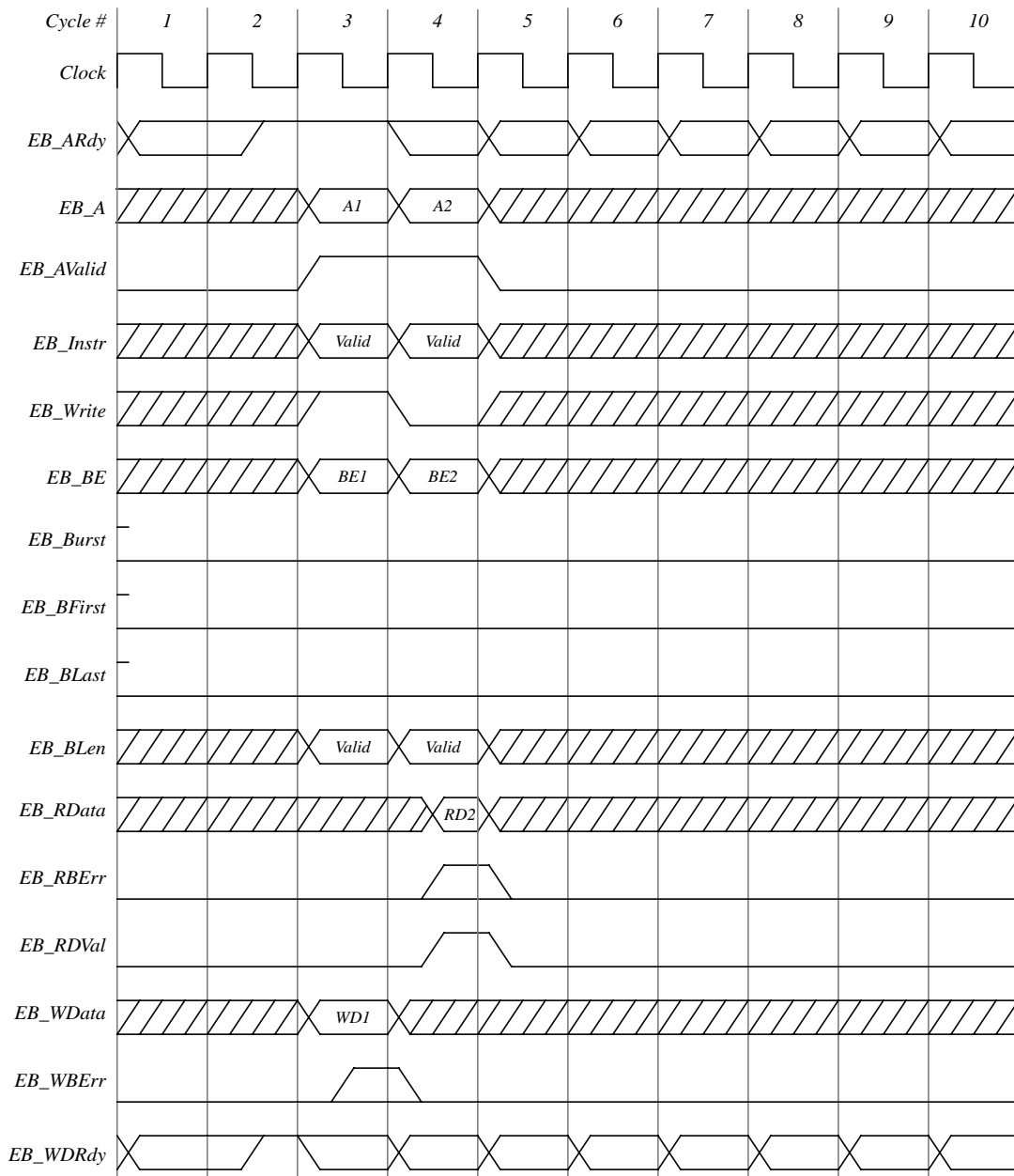


Figure 10 Write Transaction Followed by a Read Transaction

Figure 11 shows an example of a write transaction followed by a read transaction where the read transaction is completed prior to the write transaction (out of order).

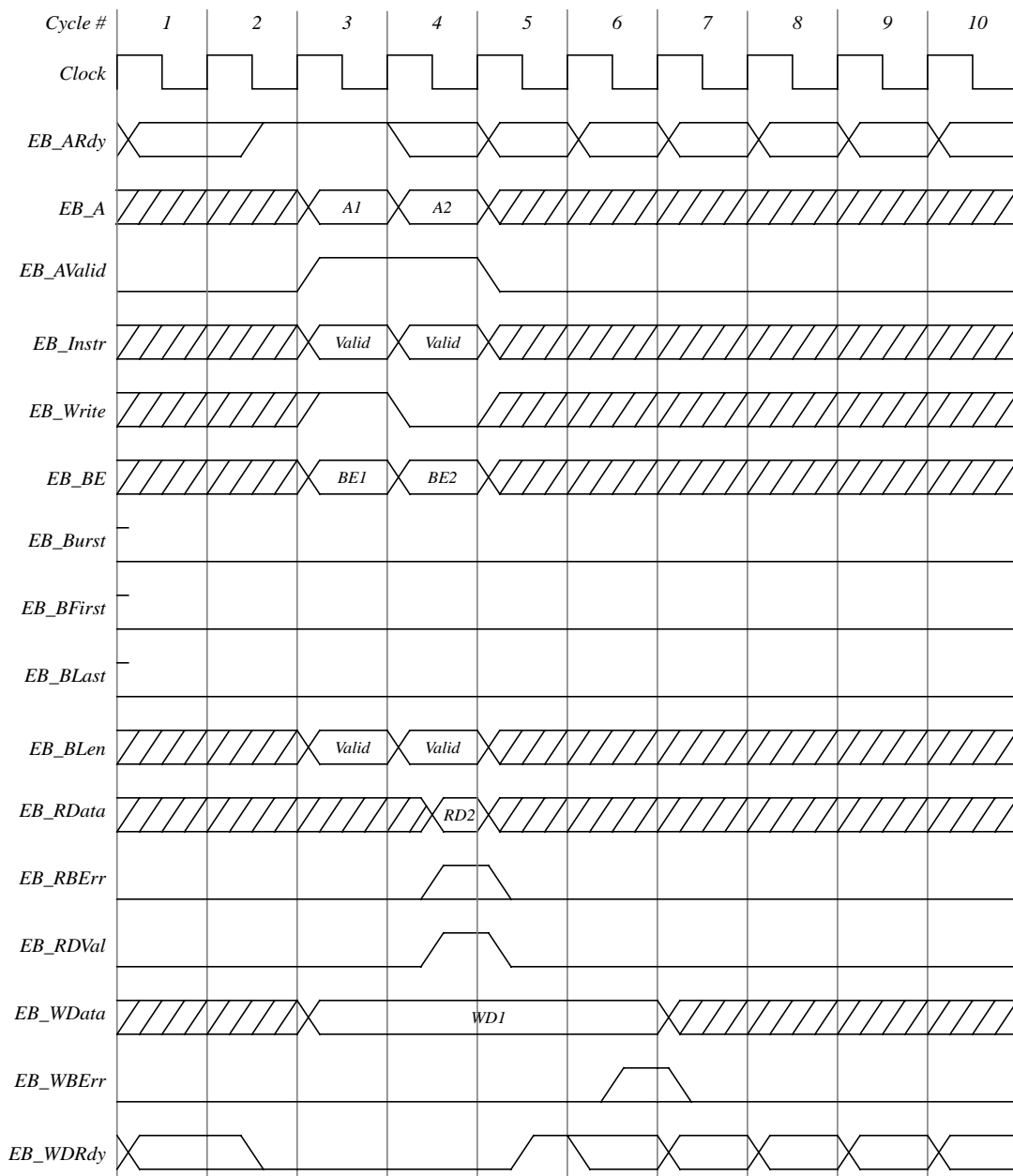


Figure 11 Write Transaction Followed by a Read Transaction with Reordering

3.7 Burst Transactions

A burst transaction initiates the transfer of multiple related transfers. Read bursts are used to read data to be placed in the instruction or data cache. Write bursts are used to empty the contents of the write buffers.

Note that initiated bursts are always completed. The burst transaction cannot be aborted before reaching the burst beat count (indicated by EB_BLen) except in the case where the EC interface is reset.

EB_Burst is asserted during the entire burst address sequence. EB_BFirst is driven asserted during the first address phase of the burst and is deasserted with each of the remaining address phases. EB_BLast is driven asserted during the last address phase and is deasserted with all prior address phases. Apart from EB_Burst, EB_BFirst and EB_BLast behavior, and the deterministic address sequence, the multiple transfers of a burst transaction behave like that of back-to-back single transactions, which simplifies interfacing to systems that do not support burst transactions. Note that it is possible in the presence of data wait states, for all of the burst address phases to complete before the first data phase of the burst (or even of a preceding transaction) has completed. If this behavior is undesirable, EB_ARdy can be used to control the pace at which the addresses are transferred.

Note that EB_AValid cannot be deasserted between address phases within a burst and that all bits in EB_BE must be asserted in all address phases within a burst.

Figure 12 shows an example of a read burst transaction. EB_BLen indicates the length of the burst (see [Section 2, "Signal List" on page 7](#) for further information on EB_BLen). The data requested is always an aligned block according to the EB_BLen signal. The order of the words within the block varies depending on which word in the block is being requested and the value of EB_SBlock (see [Table 3](#) through [Table 6](#) for further information on the refill scheme).

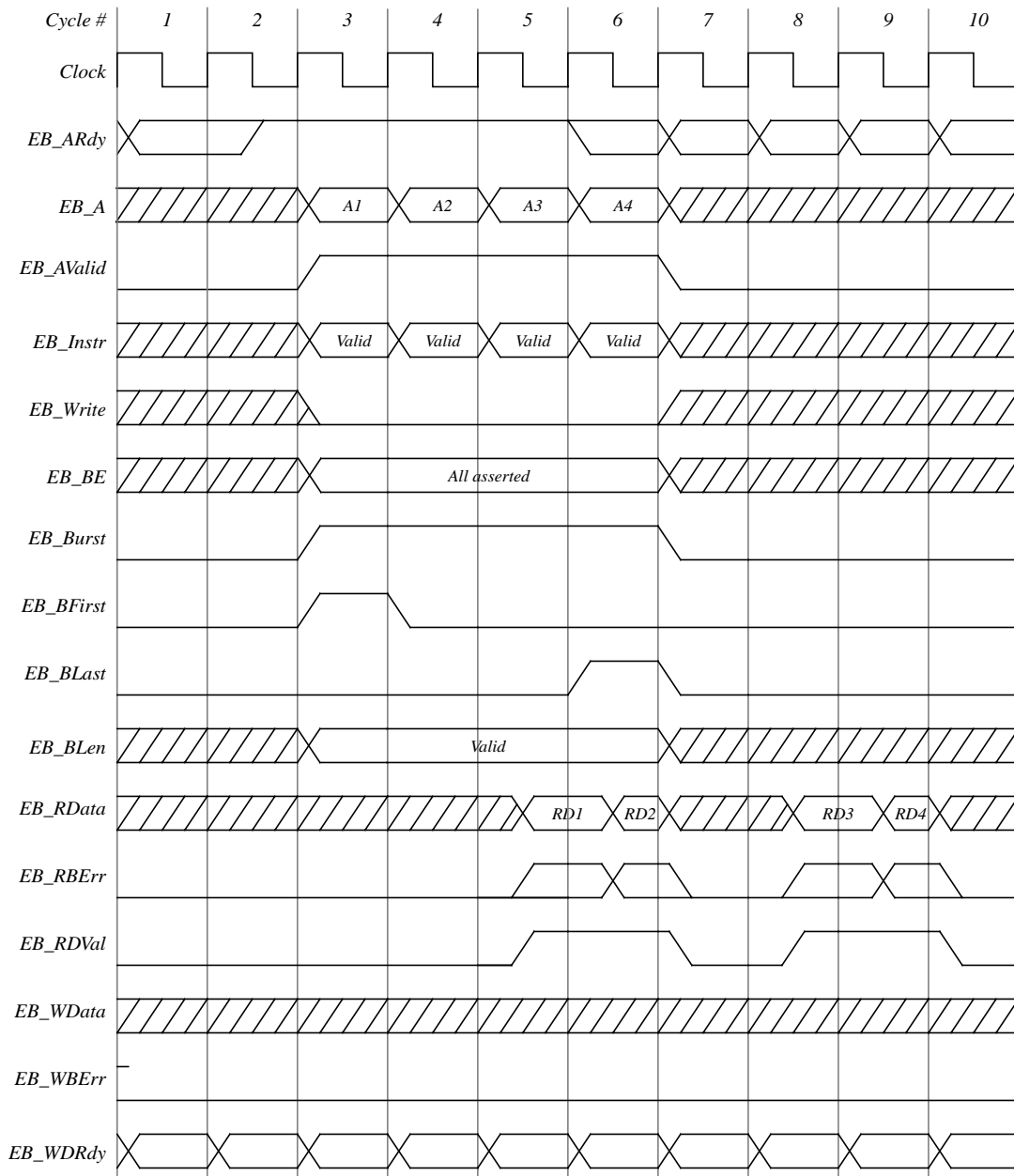


Figure 12 Burst Read Transaction Timing

Table 3 through Table 6 show the possible sequences for the least significant address bits during a burst.

Table 3 Burst Order for Sequential Ordering (4 Beat Bursts)

Req Word (DWord ^a) Address	EB_A[3:2] (EB_A[4:3] ^a) Sequence			
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

a. Optional. Only used in 64-bit implementations.

Table 4 Burst Order for Sub-block Ordering (4 Beat Bursts)

Req Word (DWord ^a) Address	EB_A[3:2] (EB_A[4:3] ^a) Sequence			
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

a. Optional. Only used in 64-bit implementations.

Table 5 Burst Order for Sequential Ordering (8 Beat Bursts)

Req Word (DWord ^a) Address	EB_A[4:2] (EB_A[5:3] ^a) Sequence							
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

a. Optional. Only used in 64-bit implementations.

Table 6 Burst Order for Sub-block Ordering (8 Beat Bursts)

Req Word (DWord ^a) Address	EB_A[4:2] (EB_A[5:3] ^a) Sequence							
	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

a. Optional. Only used in 64-bit implementations.

Figure 13 shows a burst write. Burst write transactions are used to empty write buffers. Write burst addresses always start at the lowest address of an address block according to the EB_BLen indication.

Note that like single transactions, burst read and write transactions can complete out of order. Burst reads can overtake burst writes and vice versa.

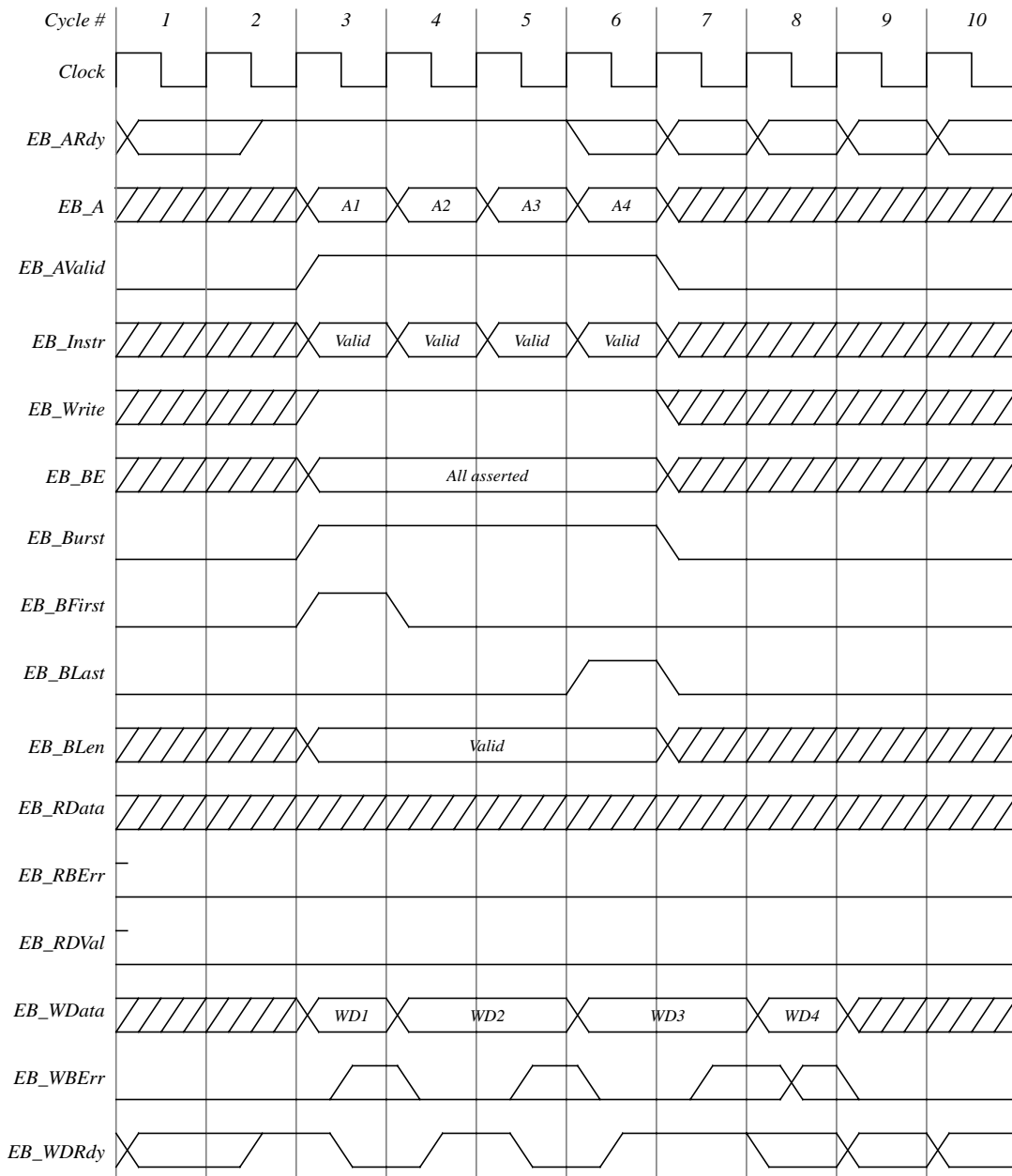


Figure 13 Burst Write Transaction Timing

4 External Write Buffers

Some systems might have external write buffers to increase bus efficiency and system performance. The EC interface has a simple two-signal protocol that allows the master to have some control over the external write buffers. When it asserts EB_WWBE, the master signals that it is waiting for EB_EWBE (External Write Buffers Empty) to be asserted. The master uses EB_EWBE to ensure that all pending writes have completed before it begins a new transaction.

If no external write buffers exist, tie EB_EWBE HIGH.

Note that EB_WWBE is not used to ensure coherency. If a write transaction is to the external write buffer, the master can generate a read request to the given address without asserting EB_WWBE (because the master has no knowledge of the external write buffers). Therefore any write buffers in the system must maintain coherency with reads.

A Endianness

The EC interface has no signal that indicates little- or big-endian operation. If the slave requires this information, for example, to generate the lower address bits that are not supplied with the EC interface, consult the documentation that comes with the core.

To help understand the use of endianness, [Table 7](#) and [Table 8](#) show some cases of how stores appear on the EC interface in little-endian and big-endian mode in a 32-bit and a 64-bit implementation of the EC interface.

Table 7 Endian Examples, 32-bit Implementation

	Internal Addr[1:0]	Big-endian		Little-endian	
		EB_D[31:0]	EB_BE[3:0]	EB_D[31:0]	EB_BE[3:0]
<pre>lui t0, 0x789a ori t0, t0, 0xbcde</pre>					
sb t0, 0x0(r0)	0	0xdeXXXXXX	1000	0xXXXXXXde	0001
sb t0, 0x1(r0)	1	0xXXdeXXXX	0100	0xXXXXdeXX	0010
sb t0, 0x2(r0)	2	0xXXXXdeXX	0010	0xXXdeXXXX	0100
sb t0, 0x3(r0)	3	0XXXXXXXXde	0001	0xdeXXXXXX	1000
sh t0, 0x0(r0)	0	0xbcdeXXXX	1100	0XXXXbcde	0011
sh t0, 0x2(r0)	2	0XXXXbcde	0011	0bcdeXXXX	1100
swl t0, 0x1(r0)	1	0xXX789abc	0111	0XXXX789a	0011
swl t0, 0x2(r0)	2	0XXXX789a	0011	0xXX789abc	0111
swr t0, 0x1(r0)	1	0bcdeXXXX	1100	0x9abcdeXX	1110
swr t0, 0x2(r0)	2	0x9abcdeXX	1110	0bcdeXXXX	1100
sw t0, 0x0(r0)	0	0x789abcde	1111	0x789abcde	1111

Table 8 Endian Examples, 64-bit Implementation

	Internal Addr[2:0]	Big-endian		Little-endian	
		EB_D[63:0]	EB_BE [7:0]	EB_D[63:0]	EB_BE [7:0]
<pre> lui t0, 0x0123 ori t0, t0, 0x4567 dsll t0, t0, 16 ori t0, t0, 0x89ab dsll t0, t0, 16 ori t0, t0, 0xcdef </pre>					
sb t0, 0x0(r0)	0	0xefXXXXXXXXXXXXXXX	10000000	0XXXXXXXXXXXXXXXXXef	00000001
sb t0, 0x1(r0)	1	0XXefXXXXXXXXXXXXXXX	01000000	0XXXXXXXXXXXXXXXXXefXX	00000010
sb t0, 0x2(r0)	2	0XXXXefXXXXXXXXXXXXXXX	00100000	0XXXXXXXXXXXXXXXXXefXXXX	00000100
sb t0, 0x3(r0)	3	0XXXXXXXXefXXXXXXXXXXX	00010000	0XXXXXXXXXefXXXXXXXXXX	00001000
sb t0, 0x4(r0)	4	0XXXXXXXXXefXXXXXXXXXX	00001000	0XXXXXXXXefXXXXXXXXXXX	00010000
sb t0, 0x5(r0)	5	0XXXXXXXXXXXXefXXXXX	00000100	0XXXXefXXXXXXXXXXXXXXX	00100000
sb t0, 0x6(r0)	6	0XXXXXXXXXXXXXXXXXefXX	00000010	0XXefXXXXXXXXXXXXXXXXXX	01000000
sb t0, 0x7(r0)	7	0XXXXXXXXXXXXXXXXXef	00000001	0efXXXXXXXXXXXXXXXXXXX	10000000
sh t0, 0x0(r0)	0	0xcdefXXXXXXXXXXXXXXX	11000000	0XXXXXXXXXXXXXXXXXcdef	00000011
sh t0, 0x2(r0)	2	0XXXXcdefXXXXXXXXXXX	00110000	0XXXXXXXXXcdefXXXXX	00001100
sh t0, 0x4(r0)	4	0XXXXXXXXXcdefXXXXX	00001100	0XXXXcdefXXXXXXXXXXX	00110000
sh t0, 0x6(r0)	6	0XXXXXXXXXXXXXXXXXcdef	00000011	0xcdefXXXXXXXXXXXXXXXXXX	11000000
swl t0, 0x1(r0)	1	0XX89abcdXXXXXXXXXXX	01110000	0XXXXXXXXXXXXXXXXX89ab	00000011
swl t0, 0x2(r0)	2	0XXXX89abXXXXXXXXXXX	00110000	0XXXXXXXXXXXX89abcd	00000111
swl t0, 0x5(r0)	5	0XXXXXXXXXXXX89abcd	00000111	0XXXX89abXXXXXXXXXXX	00110000
swl t0, 0x6(r0)	6	0XXXXXXXXXXXX89ab	00000011	0XX89abcdXXXXXXXXXXX	01110000
swr t0, 0x1(r0)	1	0xcdefXXXXXXXXXXXXXXX	11000000	0XXXXXXXXXabcdefXX	00001110
swr t0, 0x2(r0)	2	0abcdefXXXXXXXXXXXXXXX	11100000	0XXXXXXXXXcdefXXXXX	00001100
swr t0, 0x5(r0)	5	0XXXXXXXXXcdefXXXXX	00001100	0abcdefXXXXXXXXXXXXXXX	11100000
swr t0, 0x6(r0)	6	0XXXXXXXXXabcdefXX	00001110	0xcdefXXXXXXXXXXXXXXX	11000000
sw t0, 0x0(r0)	0	0x89abcdefXXXXXXXXXXX	11110000	0XXXXXXXXX89abcdef	00001111
sw t0, 0x4(r0)	4	0XXXXXXXXX89abcdef	00001111	0x89abcdefXXXXXXXXXXX	11110000
sdl t0, 0x1(r0)	1	0XX0123456789abcd	01111111	0XXXXXXXXXXXXXXXXX0123	00000011
sdl t0, 0x2(r0)	2	0XXXX0123456789ab	00111111	0XXXXXXXXXXXX012345	00000111
sdl t0, 0x3(r0)	3	0XXXXXXXX0123456789	00011111	0XXXXXXXXXX01234567	00001111
sdl t0, 0x4(r0)	4	0XXXXXXXXX01234567	00001111	0XXXXXX0123456789	00011111
sdl t0, 0x5(r0)	5	0XXXXXXXXXXXX012345	00000111	0XXXX0123456789ab	00111111
sdl t0, 0x6(r0)	6	0XXXXXXXXXXXX0123	00000011	0XX0123456789abcd	01111111

Table 8 Endian Examples, 64-bit Implementation (Continued)

	Internal Addr[2:0]	Big-endian		Little-endian	
		EB_D[63:0]	EB_BE [7:0]	EB_D[63:0]	EB_BE [7:0]
sdr t0, 0x1(r0)	1	0xcdefXXXXXXXXXXXX	11000000	0x23456789abcdefXX	11111110
sdr t0, 0x2(r0)	2	0xabcdefXXXXXXXXXX	11100000	0x456789abcdefXXXX	11111100
sdr t0, 0x3(r0)	3	0x89abcdefXXXXXXXX	11110000	0x6789abcdefXXXXXX	11111000
sdr t0, 0x4(r0)	4	0x6789abcdefXXXXXX	11111000	0x89abcdefXXXXXXXX	11110000
sdr t0, 0x5(r0)	5	0x456789abcdefXXXX	11111100	0xabcdefXXXXXXXXXX	11100000
sdr t0, 0x6(r0)	6	0x23456789abcdefXX	11111110	0xcdefXXXXXXXXXXXX	11000000
sd t0, 0x0(r0)	0	0x0123456789abcdef	11111111	0x0123456789abcdef	11111111

B Lower Address Bit Generation

Figure 14 shows a Verilog example of how the lower address bits can be generated for use with a SysAD interface. Note that this case requires that only the default EB_BE patterns are used.

```

// Low address bit generation
wire [1:0] my_a_1_0 = (BigEndian == 1'b1
    ?
        // big endian
        (EB_BE[3] ? 2'd0 :
         EB_BE[2] ? 2'd1 :
         EB_BE[1] ? 2'd2 :
         2'd3)
    :
        // little endian
        (EB_BE[0] ? 2'd0 :
         EB_BE[1] ? 2'd1 :
         EB_BE[2] ? 2'd2 :
         2'd3)
    ;

```

Figure 14 Example of Generating Low Address Bit

C Revision History

Revision	Date	Comments
01.00	00/03/01	First official release.
01.01	00/07/04	Added revision history and changed page 2, footer and the setup for conversion to pdf format.
01.02	00/10/09	Removed copyright notice from footer.
01.03	00/12/18	Converted document to new template. Editorial changes only.

