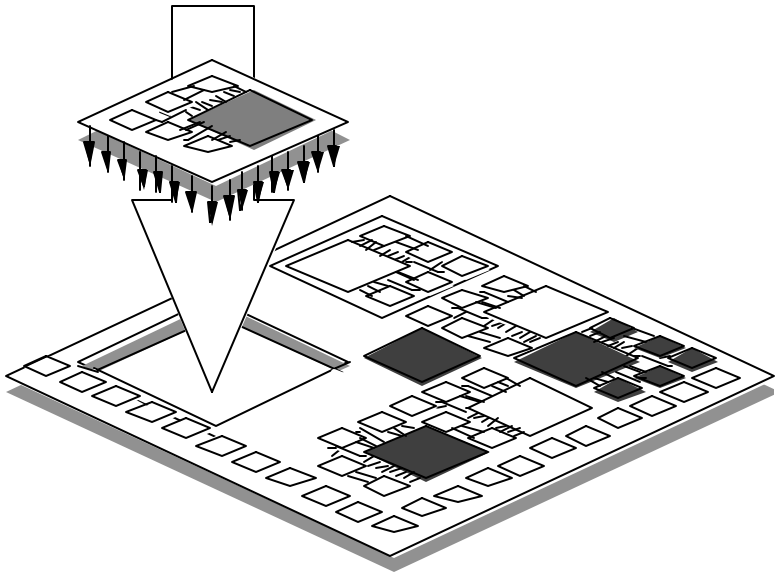
VSI Alliance™
Virtual Component Interface Standard
Version 2
(OCB 2 2.0)

On-Chip Bus
Development Working Group

April 2001



HOW TO OBTAIN LICENSE RIGHTS
FOR THE VSI ALLIANCE DOCUMENT:

On-Chip Bus Development Working Group
Virtual Component Interface Standard
Version 2.0
(OCB 2 2.0)

VSI ALLIANCE (VSIA) COPYRIGHT LICENSE

The VSI Alliance is the copyright owner of the document identified above.

The VSI Alliance will make royalty-free copyright licenses for this document available to VSI Alliance Members. Non-members must pay a fee for the copyright license.

Use of the document by members and non-members of the VSI Alliance is subject to the terms of the license. You are not entitled to use the document unless you agree to the terms of the license (and, if applicable, pay the fee).

The license terms are set forth on the Web site of the VSI Alliance at http://www.vsi.org.

THE DOCUMENT IS PROVIDED BY VSIA ON AN “AS-IS” BASIS, AND VSIA HAS NO OBLIGATION TO PROVIDE ANY LEGAL OR TECHNICAL ASSISTANCE IN RESPECT THERETO, TO IMPROVE, ENHANCE, MAINTAIN OR MODIFY THE DOCUMENT, OR TO CORRECT ANY ERRORS THEREIN. VSIA SHALL HAVE NO OBLIGATION FOR LOSS OF DATA OR FOR ANY OTHER DAMAGES, INCLUDING SPECIAL OR CONSEQUENTIAL DAMAGES, IN CONNECTION WITH THE USE OF THE DOCUMENT BY SUBSCRIBER. VSIA MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY AS TO INFRINGEMENT, OR THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SUBSCRIBER SHOULD BE AWARE THAT IMPLEMENTATION OF THE DOCUMENT MAY REQUIRE USE OF SUBJECT MATTER COVERED BY PATENT OR OTHER INTELLECTUAL PROPERTY RIGHTS OF THIRD PARTIES. NO LICENSE, IMMUNITY, OR OTHER RIGHT IS GRANTED BY THIS LICENSE IN ANY SUCH THIRD-PARTY RIGHTS. NEITHER VSIA NOR ITS MEMBERS TAKE ANY POSITION WITH RESPECT TO THE EXISTENCE OR VALIDITY OF ANY SUCH RIGHTS.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

On-Chip Bus
Development Working Group

Members of the Development Working Group:

Agilent Technologies	ARM
Cadence Design Systems	Co-Design Automation
Easics NV	ECSI
Fujitsu Limited	Hitachi Semiconductor
InSilicon	Infineon Technologies
LSI Logic	Lucent Technologies
Nokia Mobile Phones	OKI Electric IndustryCo.
Palmchip	Philips Semiconductors
Sonics	STMicroelectronics
Synopsys	Toshiba
Xilinx	

Active Contributors:

Bruce Mathewson ..... ARM
Larry Cooke ..... Cadence Design Systems
Mani Gopalakrishnan ..... Fujitsu Microelectronics
J. Sukarno Mertoguno (Vice Chair) ..... Individual Member
Prakash Bare ..... Individual Member
Mason Weems ..... Individual Member
Rick Born ..... LSI Logic
Anssi Haverinen (Chair) ..... Nokia Mobile Phones
Frits Zandveld ..... Philips Semiconductors
John Carey ..... STMicroelectronics
Graham Matthew ..... STMicroelectronics
Drew Wingard ..... Sonics
Greg Tanaka ..... Synopsys
Glen Baxter ..... Xilinx

Other Contributors

Won Rhee ..... Agilent Technologies
John Cornish ..... ARM
Larry Rosenberg ..... Cadence Design Systems
Peter Flake ..... Co-Design Automation
Tony Gore ..... ECSI
Glen Vinogradov ..... Individual Member
Prof. Ganesh Gopalakrishnan ..... Individual Member
Frank Vahid ..... Individual Member
David Jennings ..... Infineon Technologies
Yuji Hatano ..... Hitachi Semiconductor
Osamu Yamashiro ..... Hitachi Semiconductor
Charles Minter ..... Toshiba

Technical Editors

Wilsa Schroers and Sybil Sommer

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Revision History

Version 1.0	Mar00	Wilsa Schroers	Copy edited
Version 1.0	08Mar00	David Huddleston	Formatted figures
Version 1.0	10Mar00	Editorial Staff	Formatted document in FrameMaker, inserted figures for Final Release, added license
Version 1.0	23Mar00	Editorial Staff	Edited figures, updated headers
Version 2.0	13Feb01	Editorial Staff	Copy edited and formatted
Version 2.0	30Mar01	Editorial Staff	Final copy edits and formatting

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table of Contents

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

VSIA CONFIDENTIAL LICENSED DOCUMENT

Appendices

List of Tables

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

List of Figures

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

1. Virtual Component Interface (VCI) Background

This standard is the result of the work of the On-Chip Bus (OCB) Development Working Group (DWG) of the Virtual Socket Interface Alliance (VSIA). The charter defined for the OCB DWG was to define an on-chip bus-related specification for the design, integration, and test of multiple functional blocks on a single piece of silicon.

Two generic audiences are targeted: providers of functional blocks and users or integrators of these blocks. For the providers of functional blocks, the standard defines the protocols and interfaces the users require for effective reuse of the blocks in an integrated product design. For the users of functional blocks in compliance with this standard, it depicts the information the users will need to properly evaluate, integrate, and verify one or more functional blocks in a design.

The overall objective is to obtain a general interface, such that Intellectual Property (IP), in the shape of Virtual Components (VCs) of any origin, can be connected to Systems-on-Chips (SoC) of any chip integrator. In this manner, VCs are not limited to one-time usage by their designers. They can be *re-used* over and over. Such re-use of VCs also applies to VC providers or to external system integrators.

Early in the existence of this DWG, it became clear that picking an existing bus or defining a new one would not be the right way to go. First of all, system integrators stick to their own bus for a relatively long time, if only to allow for connection of existing VCs without modification. Connecting new VCs via a bus-to-bus bridge is expensive in timing and in silicon footprint. Furthermore, all existing buses are well designed for their particular usage. Inventing yet another bus does not make much sense, because there would be little chance of such a bus being accepted.

For these reasons, the DWG decided to define an interface rather than a bus. This interface can be used as a point-to-point connection if needed and also as an interface to a bus connector.

This standard does not define Virtual Component Interface (VCI) compliance. It is defined by the OCB Attributes Specification (OCB 1 2.0). To the best of our knowledge, as of the public release of this standard, verification (beyond review by the experts within the OCB DWG and the VSI membership at large) has been limited to the PVCI section and parts of the BVCI section of the standard.

1.1 Goals

The following primary goals were considered in defining the VCI.

1. The VCI must enable maximum portability of a VC.
    - VCI-compliant VCs should inter-operate with OCBs of varying protocols and performance levels.
    - The VCI should not dictate the integration methodology. That is, the VCI can be used as a point-to-point interface without an OCB, and can be connected to different OCBs with automated methods, or with handcrafted wrappers.
2. The VCI should not require modification of VCI-compliant VCs in order to connect to a different VCI-compliant VC or OCB.
    - Some combinations of VCI-compliant VCs and OCBs may result in reduced features or performance, but must still function correctly and reliably.
3. The VCI should be simple and efficient to implement, with clear and easy to understand protocol.
    - This is necessary for wide acceptance.
4. The VCI must include designs for a compatible family of VC Interface, that is, Peripheral VCI (PVCI), Basic VCI (BVCI), and Advanced VCI (AVCI).
    - This enhances inter-operability choices for the system chip designer, and design opportunities for the VC designer.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5. The protocol must be fully defined to include full enumeration of possible interactions with statements on allowed and disallowed behavior. However, it is essential for wide adoption that there be some mechanism for extending the protocol.

6. The VCI must minimize the number of truly optional signals and thus minimize the complexity of VCI compliance checking. However, signals such as data and address lines and supported transfer widths may be parameterizable and/or scalable.

1.2 Key Assumptions

This section describes the assumptions that were made in defining the VCI.

1. Initiator and target connections are point-to-point and unidirectional.

   Both multiplexed and tri-state OCBs can be supported by allowing the OCB wrappers to implement the OCB transceivers. Separate unidirectional nets are simpler to handle, and this circumvents the requirement for arbitration in the VCI protocol.

2. The initiator can only present requests, and the target can only respond.

   If a VC needs both capabilities, implement parallel initiator and target interfaces.
   - A combined (peer-to-peer) protocol is much more complex to define and implement, due to interface arbitration requirements.
   - Some OCB architectures can take advantage of the separate initiator and target interfaces by connecting the VC Initiator interface to a system OCB, whereas the separate VC target interface might connect to a peripheral or configurable OCB.
   - Finally, the parallel interface is simpler to model and offers possible performance advantages using simultaneous transfers.

3. The VCI must include limited fundamental Read and Write requests.
   - The PVCI is limited to read and write.
   - The BVCI includes Nop and read lock.
   - The AVCI contains features to support better control over system interconnect.

4. Address and data widths are determined by VC requirements.
   - The OCB target initiator should scale its address and data widths to match target.

5. The standard should ensure that any required data and address storage in the wrappers is minimal.
   - Anything more than minimal timing overhead is deemed unacceptable and negatively impacts acceptance.

6. Clock domain crossing will not be visible at the interface.
   - The interface is fully synchronous. Clock domain crossing is considered a design guideline for wrapper or VC implementation.

1.3 Document Conventions

This section gives the conventions used in defining the VCI.

b - Number of bytes in a cell

n - Most significant bit of address field

m - Least significant bit of cell address

k - Number of bits in the packet length field

q - Number of bits in the packet-chain length field

E - Number of bits in the error extension field

Timing constraints:

Early - Signal is valid within 20% of the clock cycle from the rise of the Clock signal

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Middle - Signal is valid within 50% of the clock cycle from the rise of the Clock signal

Late - Signal is valid within 80% of the clock cycle from the rise of the Clock signal

1.4 Document Organization

This standard begins with defining the basic characteristics of the VCI. The different complexity interfaces are described in detail next, starting with the simplest one, the PVCI. The PVCI and BVCI chapters are designed so that they can be read independently. That is, if a reader wants to know how to use the PVCI, the reader does not need to be familiar with the BVCI. The AVCI contains a set of optional features that can be added to the BVCI. Only the differences from the BVCI are explained in the AVCI chapter. After the interface descriptions, some design guidelines are given. The VCI Transaction Language section is described in Appendix A.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

2. VCI Characteristics

2.1 VCI Definition

The Virtual Component Interface (VCI) is an interface rather than a bus. Thus the VCI specifies:

- A request-response protocol
- A protocol for the transfer of requests and responses
- The contents and coding of these requests and responses

The VCI does not touch areas as bus allocation schemes, competing for a bus, and so forth.

There are three complexity levels for the VCI: Peripheral (PVCI), Basic VCI (BVCI), and Advanced VCI (AVCI). The PVCI provides a simple, easily implementable interface for applications that do not need all the features of the BVCI. The BVCI defines an interface that is suitable for most applications. It has a powerful, but not overly complex protocol. The AVCI adds more sophisticated features, such as threads, to support high-performance applications. The PVCI is a subset of the BVCI, and the AVCI is a superset of the BVCI. All these interfaces are designed to be compatible with each other.

When this document refers to VCI as opposed to PVCI, BVCI, or AVCI, it is referring to either the entire standard, or any non-level specific implementation of this standard.

2.2 Point-to-Point Usage

As an interface, the VCI can be used as a point-to-point connection between two units called the initiator and the target, where the initiator issues a request and the target responds. The VCI can also be used as the interface to a "wrapper," a connection to a bus. This is how the VCI allows the VC to be connected to any bus. (See Section 3.4, "PVCI Protocol.") Basic point-to-point usage is depicted in Figure 1.

This interface is very simple where protocol and coding are concerned. Nevertheless, the interface contains many sophisticated features, as explained in further chapters of this standard. Its simplicity is the reason for the small footprint and the high bandwidth.

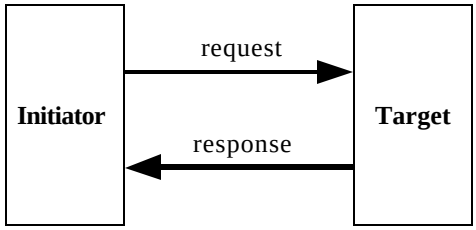


Figure 1: VCI is a Point-to-Point Connection

2.3 Split Protocol

BVCI and AVCI make use of a "split protocol." That is, the timing of the request and the response are fully separate. The initiator can issue as many requests as needed, without waiting for the response. The protocol does not prescribe any connection between issuing of requests and arrival of the corresponding responses. The only thing specified is that the order of responses corresponds to the order of requests.

In the AVCI, requests may be tagged with identifiers, which allows such requests and request threads to be interleaved and responses to arrive in a different order. Responses bear the same tags issued with the corresponding requests, such that the relation can be restored upon the reception of a response. Additional details are provided in Chapter 5, "Advanced VCI."

In the PVCI, no split protocol is used, and each request must be followed by a response before the initiator can issue a new request. This simplification is used to support simple peripheral buses.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

2.4 VCI Usage with a Bus

The VCI can be used as the interface to a wrapper, which means a connection to a bus. This is how the VCI allows the VC to be connected to any bus. An initiator is connected to that bus using a bus initiator wrapper. A target is connected to that bus using a bus target wrapper. Once the wrappers for the bus have been designed, any VC can be connected to that bus, as depicted in Figure 2. The wrapper and the VCI boxes (the non-shaded area in the figure) together correspond to the traditional bus interface within both the initiator and the target. The whole interface functions as a point-to-point connection. Of course, the use of VCI does not prevent using native bus components at the same time. In fact, the presented use of VCI is completely transparent to the bus system.

It is intended that OCB suppliers provide VCI wrappers to their proprietary bus, or that EDA vendors provide tools to create such wrappers automatically. This will free the IP provider from having to understand the details of many buses.

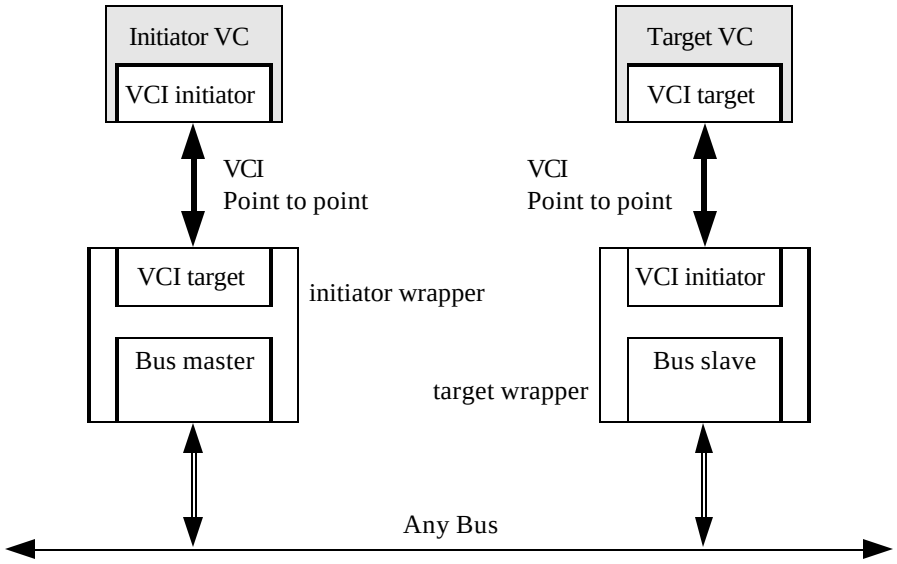


Figure 2: Two VCI Connections Used to Realize a Bus Connection

2.5 Notes

- The initiator and target are simpler than their equivalents directly connected to a bus. The VCI initiator and the VCI target are much simpler than the bus initiator and target interfaces.
- The bus interface is not removed. It is in the wrapper. The wrapper does not contain much more than the bus interface, which is needed anyway when connecting to the bus and to the VC interface block.
- The VCI initiator and VCI target blocks shown in Figure 2 are very simple. They do not add much to the total footprint or timing, since some logic is often needed anyway to connect the actual functionality of initiator and target to their bus interfaces. *Only the vci_initiator – vci_target interface is standardized in the VCI.*

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

3. Peripheral VCI

This chapter defines the first member of the VCI family, the Peripheral Virtual Component Interface (PVCI). It can be used in conjunction with peripheral on-chip buses, as defined in the VSIA OCB Specification 1 1.0, and for point-to-point connections between VCs. The PVCI is a subset of the Basic VCI (BVCI), just as the BVCI is a subset of the Advanced Virtual Component Interface (AVCI).

3.1 Organization

This chapter contains the following sections:

Section 3.1: Describes the organization.

Section 3.2: Gives a technical introduction to PVCI.

Section 3.3: Provides a detailed description of PVCI signals.

Section 3.4: Defines the PVCI protocol.

3.2 Technical Introduction to PVCI

The following sections give a technical introduction to the PVCI.

3.2.1 Initiator – Target Connection

As shown in Figure 3, the request contents and the response contents are transferred under control of the protocol, a simple two-wire handshake.

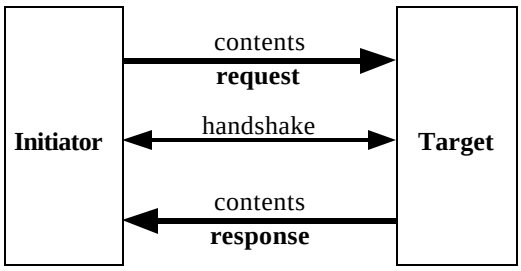


Figure 3: PVCI ‘- Request and Response Contents Controlled by a Simple Handshake

3.2.2 The Handshake

Note that in Figure 3, the handshake arrow is double-pointed, showing that signals in both directions are involved. (Signal here is synonymous to net, wire, or port.) The handshake protocol is aimed at synchronizing two blocks by transferring control information in both directions. The contents arrow points one way only. The PVCI handshake signals are called **VAL** and **ACK**, which stand for Valid and Acknowledge.

Request contents flow from the initiator to the target. Response contents flow from the target to the initiator. The handshake protocol is shown in Figure 4 and Figure 5. Note that the delays are exaggerated in the figures. The triggering events for each transition are expressed with arcing arrows.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

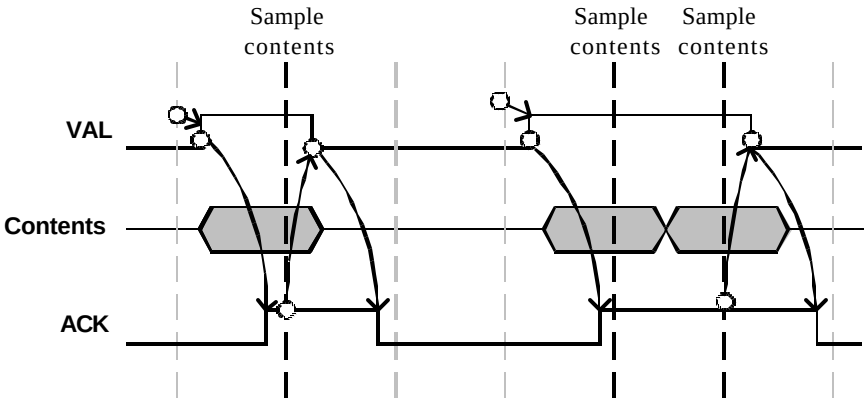


Figure 4: Control Handshake (Asynchronous ACK)

Notes for Figure 4:

- Vertical dashed lines show rising clock edges.
- VAL shows "at the next rising edge, contents can be read." No actual timing is specified here.
- While VAL is active at the rising edge of the clock, ACK shows that the contents will be read by the target, or that the contents are available to be read from the target. This signifies the end of the transaction.

VAL and contents must be maintained until ACK has become asserted and there is a rising edge of the clock. Maintaining the VAL signal in asserted mode for another clock cycle after ACK = 1 means that another request is waiting, as shown on the right side of Figure 4.

The ACK can be either generated asynchronously of the VAL, as shown in Figure 4, or set synchronously at the rising edge of the clock, as shown in Figure 5. The synchronous ACK can be set at the next clock cycle after Valid, or later (a slow reaction, or late ACK). If asynchronous ACK is used, special design considerations are needed to make sure that the ACK is stable at the rising clock edge. This means that if the ACK is not driven directly by a flip-flop, it must be generated from the VAL signal in the target with a minimum amount of logic and wire delay.

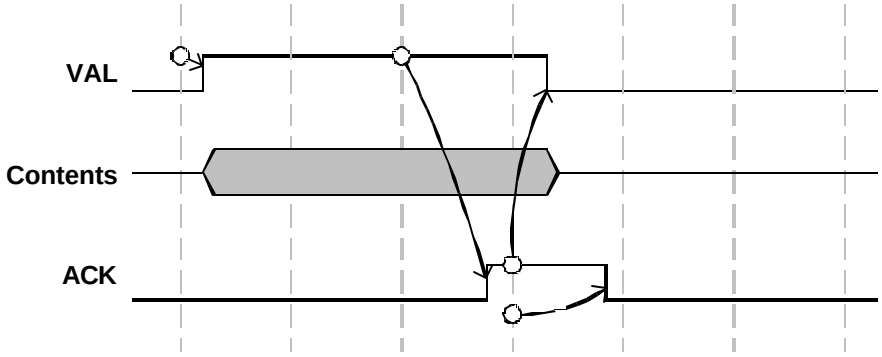


Figure 5: Fully Synchronous Control Handshake (ACK Late by Two Cycles)

Default Acknowledge

A "default acknowledge" behavior is permitted on top of this protocol. Since the acknowledge-signal is only sampled when VAL = 1, it can be asserted long before it is needed. Such an early ACK has no influence on the protocol behavior. The early ACK means "don't care" (that is, the signal is not being considered, if VAL is not active). The acknowledge signal can even be tied permanently active, if the target is always able to serve the request in one clock cycle.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

VSIA CONFIDENTIAL LICENSED DOCUMENT

3.2.3 Request and Response Contents

Each handshake is used to transfer a *cell* across the interface. *Cell size* is the width of the data passing across a VCI. It is typically 1, 2, or 4 bytes. The cell is synonymous to a data word, and the cell size to word length. The cell size is always the same as the size of the particular VC Interface. Figure 6 shows the signal groups of the PVCI belonging to the request and response. The request contents include the following: Valid, cell-address, byte-enables within the cell, data to be written, command signal indicating whether you are reading or writing data, and end-of-packet, indicating the end of a burst of several cells. (End-of-packet is described later in this document.) The response contents include the following: acknowledge, response error, and the data read from the target. The initiator interface is similar, except that the signal directions are opposite.

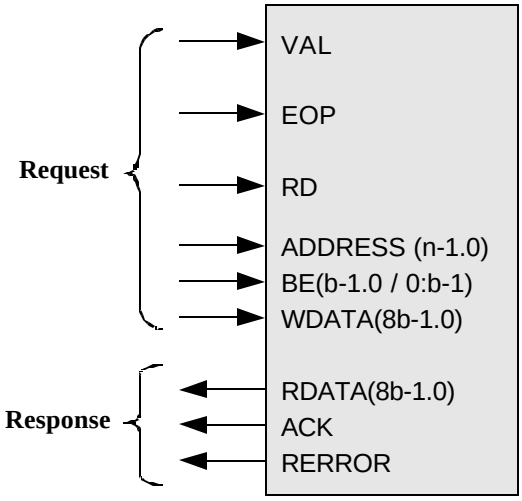


Figure 6: Peripheral VC Interface – Target

3.2.4 Main PVCI Features:

- Up to 64-bit Address
- Up to 32-bit Read Data
- Up to 32-bit Write Data
- Synchronous
- Allows for 8-bit, 16-bit, and 32-bit devices
- 8-bit, 16-bit, and 32-bit Transfers
- Simple packet, or 'burst' transfer
- Optional Free-BE mode allows transfer of any combination of bytes of a word
- Least Significant Bit is bit "0"
- PVCI has no explicit endianness
- Natural byte alignment

3.3 Signal Definitions

This section defines the PVCI signals. First, the signals are summarized in table format. Next, a textual definition of each signal is given. All signals included in Table 1 are assumed to be active-high signals unless indicated otherwise. It is recommended that all signal outputs become stable before Early. It can be assumed that all inputs are stable before Late.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 1: PVCI Signals

Signal Name	Driver	Receiver	Width	Comments
System Signals				
CLOCK	System	Initiator/ Target	1	Supplied by the system. Interface is synchronous to the rising edge only.
RESETN	System	Initiator/ Target	1	Supplied by the system. Active low reset. De-asserts VAL and ACK.
Handshake, Flow Control, and Shaping				
VAL	Initiator	Target	1	This pair of signals provides flow control for a cell transferring across the VC Interface, and validates all signals associated with that cell. VAL==1: indicates that the initiator has a cell available. ACK==1: indicates that the target can complete the operation on the cell. The cell is therefore transferred when VAL==ACK==1
ACK	Target	Initiator	1	
EOP	Initiator	Target	1	EOP==1 indicates that the current cell is the last one in a series of cells accessed at contiguous addresses. Can be interpreted as an inverted burst signal.
Operation Information				
ADDRESS [N-1:0]	Initiator	Target	N	N is a parameter based upon the capabilities of the target, and is defined and fixed at the time of component instantiation. The most significant bit of the address is carried by bit N-1, and the least significant bit is carried by bit 0. See Section 3.3.3, “Address and Data Signals,” for use of low-order address bits.
RD	Initiator	Target	1	This is a single bit code giving the operation type: RD=1: Read data from the target peripheral RD=0: Write data to the target peripheral
BE [b-1:0 | 0:b-1]	Initiator	Target	b	This is a field with b-bits, one for each byte, which indicates which bytes of the word being transferred are enabled. The usage restrictions of BE are listed in Section 3.4.3, “Data Formatting and Alignment.”
WDATA [8b-1:0]	Initiator	Target	8b	This is the data that is transferred with write operations to the target. For the PVCI, the allowed values of b are 4, 2, and 1. Bit 8b-1 is the most significant bit, and bit 0 is the least significant bit. Byte [8b-1:8b-8] represents the most significant byte. For VCs supporting a data size that is not a power of two, the next larger supported b will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide PVCI with the 4 Most Significant bits tied to logic zero.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 1: PVCI Signals (Continued)

Signal Name	Driver	Receiver	Width	Comments
RDATA [8b-1:0]	Target	Initiator	8b	This is the data that is returned from the target with read operations. Since the Peripheral Interface has no pipelining, RDATA is validated by the target when it asserts ACK. It is defined in the same way as WDATA above.
RERROR [E:0]	Target	Initiator	E+1	Indicates error in transfer. Optional error extension bits indicate nature of the error. E is defined by parameter ERRLEN. It is zero or more.

3.3.1 System-Level Signals

3.3.1.1 Clock

Signal Name: Clock
Signal Abbreviation: CLOCK
Polarity: Active at Positive edge
Driven By: System
Received By: VCI initiator, VCI target

This signal provides the timing for the VCI and is an input to the initiator and the target that are connected by the PVCI. All initiator and target output signals are asserted and de-asserted relative to the rising edge of CLOCK, and all initiator and target inputs are sampled relative to this edge.

3.3.1.2 Reset

Signal Name: Reset
Signal Abbreviation: RESETN
Polarity: Asserted Negative
Driven By: System
Received By: VCI initiator, VCI target
Timing: Asserted > 8 clock cycles

This signal is used during power-on reset and is used to bring the PVCI to an idle or quiescent state. This idle state is defined as the PVCI state in which:

1. The VAL signal is de-asserted.

2. The ACK signal is de-asserted.

The system must guarantee that RESETN is asserted for at least eight cycles of CLOCK (unless the RESETLEN parameter is set).

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

3.3.2 Control Signals

3.3.2.1 Valid

Signal Name:	Valid
Signal Abbreviation:	VAL
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 and next rising edge of CLOCK

The VAL signal is driven by a VCI initiator to indicate that there is a valid address, data, and command on the PVCI. All of the initiator control signals are qualified by VAL. The initiator keeps VAL asserted, and all of its control signals valid and stable, until it receives the ACK signal from the target. The initiator should not assert VAL unless the current transaction is intended for the target. Thus, the target wrapper may need to perform address decoding on its on-chip bus side to generate VAL for the target, thereby accomplishing device selection.

3.3.2.2 Acknowledge

Signal Name:	Acknowledge
Signal Abbreviation:	ACK
Polarity:	Asserted Positive
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted and de-asserted at rising edge of CLOCK. The signal must change before Late.

The ACK signal is asserted by the target to indicate the completion of a transfer between the initiator and target. In the case of write operations, this means that the target has accepted the data that is on the write data bus, or will do so at the end of the current clock cycle. In the case of read operations, the assertion of the ACK by the target indicates that the target has placed data to be transferred to the initiator on the read data bus. The transaction completes as soon as the rising edge of CLOCK samples ACK. The target may de-assert the ACK by the next rising edge of CLOCK unless a new command has been initiated by the initiator, or default acknowledge is used. Similarly, the initiator de-asserts VAL by the next rising edge of CLOCK unless it is presenting a new command.

3.3.2.3 Read / Not Write

Signal Name:	Read / Not Write
Signal Abbreviation:	RD
Polarity:	Read at positive, write at zero
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK

This signal is a one-bit command asserted by the initiator and indicates the direction of the requested transfer. RD indicates that the requested transfer is a read if it is asserted high, and a write if it is asserted low. This signal must be valid any time that the VAL signal is asserted. A read transfer is a request for the target to supply data on RDATA to be read into the initiator. A write transfer is a request for the target to accept data on WDATA from the initiator.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

3.3.2.4 End-of-Packet

Signal Name:	End-of-Packet
Signal Abbreviation:	EOP
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until ACK = 1 at rising edge of CLOCK

The EOP signal is de-asserted by the initiator to indicate that the transfer being performed will be followed with a transfer by the initiator to the next higher cell address. This signal is used by the target device to preset address in order to improve the data transfer performance. Thus this signal can be interpreted as an inverted burst signal. The burst is similar to a packet in the BVCI, except that rather than prescribing a strict atomicity, it merely indicates a contiguous address behavior. The burst transfer is completed once a cell is transferred with the EOP signal asserted, or when the target signals an Abort error. For the legal address behavior during a burst, see Section 3.3.3.1, “Address.”

A BVCI packet with CONTIG signal asserted high and WRAP signal asserted low can be mapped to a PVCI burst.

3.3.3 Address and Data Signals

Address

Signal Name:	Address
Signal Abbreviation:	ADDRESS[n-1:0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until ACK = 1 at rising edge of CLOCK

The PVCI initiator uses ADDRESS to identify which target resource the current transaction acts upon. The N-lines of address bus form a binary number that represents an address. N is a parameter based upon the capabilities of the target and is defined and fixed at the time of component instantiation. The most significant bit of the address bus will be carried by bit N-1 and the least significant bit of the address will be carried by bit 0. ADDRESS[n-1:m] provides a cell address where the cell size is specified by the total data width of the PVCI. M equals 0 for cell size 1, 1 for cell size 2, and 2 for cell size 4. Sub-cell addressing is handled by the byte-enable signals.

It is permissible to supply the low-order address bits to allow the original full intention of the transfer to be maintained. This relies on knowledge of the endianness of the initiator. This additional information is not required to complete the operation, which is completely defined by a cell-aligned address and byte enables (that is, the low-order address bits may be tied to logical zero). However, in some systems, some efficiency gains may be possible by taking advantage of this information.

When the EOP signal is low, the ADDRESS of the next cell must be CELLSIZE + ADDRESS of the current cell, and the ADDRESS of the current cell must be aligned with the cell boundary.

3.3.3.1 Byte Enable

Signal Name:	Byte Enable
Signal Abbreviation:	BE[**b**-1:0|0:**b**-1]
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK

BE is a **b**-bit field that indicates which bytes of the cell being transferred are enabled. The **b** equals the total data width of the PVCI/8. These signals must be valid any time that the VAL signal is asserted. The usage of byte enables is described in Section 3.4.3, “Data Formatting and Alignment.”

BE[3:0] is used for little endian VCs where the LSB (Least Significant Byte) is labeled Address 0.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

BE[0:3] is used for big endian VCs where the MSB (Most Significant Byte) is labeled Address 0.

3.3.3.2 Write Data

Signal Name:	Write Data
Signal Abbreviation:	WDATA[8**b**-1 : 0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK

The write data lines are driven by the VCI initiator, and are used to transfer write data from an initiator to a target device. Write data consists of **b** logical byte lanes, based upon the capabilities of the target, and is defined and fixed at the time of component instantiation. Allowed values of b are 1, 2, and 4, allowing 8-, 16- or 32-bit interfaces. Bit 8***b**-1 is the most significant bit of the most significant byte and bit 0 is the least significant bit of the least significant byte. The write data lines must contain valid write data while the VAL signal is asserted and the RD is indicating a write transfer.

For VCs supporting a data size that is not an eight-bit increment, the next larger supported bus size will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide PVCI with the four most significant bits tied to logic zero.

3.3.3.3 Read Data

Signal Name:	Read Data
Signal Abbreviation:	RDATA[8b-1: 0]
Polarity:	N/A
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK

The read data lines are driven by the VCI initiator, and are used to transfer read data from a target to an initiator device. Read data consists of **b** logical byte lanes, based upon the capabilities of the target, and is defined and fixed at the time of component instantiation. Allowed values of b are 1, 2 and 4, allowing 8-, 16- or 32-bit interfaces. Bit 8***b**-1 is the most significant bit of the most significant byte, and bit 0 is the least significant bit of the least significant byte. The read data lines must contain valid read data while the ACK signal is asserted and the RD is indicating a read transfer.

For VCs supporting a data size that is not an eight-bit increment, the next larger supported bus size will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide PVCI with the four most significant bits tied to logic zero.

3.3.4 Error Signals

3.3.4.1 Response Error

Signal Name:	Response Error
Signal Abbreviation:	RERROR[E:0]
Polarity:	Positive
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted at rising edge of CLOCK when ACK == 1 at rising edge of CLOCK

Error signal is valid only when ACK=1, with the following meaning:
For ERRLEN = 0 (E=0)
RERROR=0: Normal (no error)
RERROR=1: General data error. The entire packet is considered bad.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

For ERRLEN = 1 (E=1)
RERROR = 00: Normal (no error)
RERROR = 01: General data error. The entire packet is considered bad
RERROR = 10: Reserved
RERROR = 11: Abort Disconnect

For ERRLEN = 2 (E=2)
RERROR =000: Normal (no error)
RERROR = xx0: Reserved
RERROR = 001: General data error. The entire packet is considered bad
RERROR = 011: Reserved
RERROR = 101: Bad data (retry)
RERROR = 111: Abort Disconnect

After receiving an error, the initiator may or may not continue with the current packet. If it chooses to end the packet prematurely, it can do so by asserting EOP. After this, it can choose not to try a transfer anymore (Abort) or Retry part or all of the transfer. For any error, the target must process the subsequent pending cells and packets with the normal protocol, i.e. it must continue sending responses until it has processed the EOP with or without further errors signaled. In general, the RERROR is more informative than prescriptive, and the target may not assume any special behavior from the initiator. The initiator is anyhow encouraged to act responsibly, when it meets an error.

While sending an error, the target should continue with the burst, with or without further errors until it receives a cell with EOP. The initiator can choose to continue with the burst or set the EOP on the next cell. In the case of bad data, just the data of the error may be considered bad. If ERRLEN=2 is supported, the target must be capable of accepting a retried request to the same address where it sets Retry-error, but the initiator can choose to retry a cell, burst, or nothing at all. It is strongly recommended that all PVCI components support at least one bit error. The PVCI target is strongly recommended to signal an error when it receives a request it does not support.

3.4 PVCI Protocol

3.4.1 Operation Types

The operation types listed here define the legal PVCI requests and responses. Any other operations are outside the PVCI specification.

3.4.1.1 Transfer Request

The following is a list of the default transaction types that may be initiated by an initiator across the PVCI. An initiator should not initiate an operation of a width that the target does not support.

- Read8: Read one byte. The byte may be on any byte lane expressed with the BE field. This is supported by all PVCI components.
- Read16: Read two bytes. The bytes must be on contiguous byte lanes, and are expressed with the BE field. The operations must be aligned to 16-bit sub-word boundaries, that is, transfer with BE=0110 is not allowed. This is supported by 2- and 4-byte PVCI components.
- Read32: Read four bytes. This is supported by 4-byte PVCI components.
- Read N Cells: Read a packet of N cells. The addresses of consecutive cells must be ascending, cell aligned, and consecutive. The byte enables of individual cells may have any legal values. The last cell is indicated with the EOP signal. Each cell of a packet is individually requested and handshaken. Only one pending request for a cell is allowed at a time.

Write operations are similar to read operations, except for the data direction.

- Write8: Write one byte.
- Write16: Write two bytes.
- Write32: Write four bytes.
- Write N Cells: Write a packet of N cells.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

All transfers are packet transfers of one or more cells. A packet transfer of length 1 is indistinguishable from a single transfer. The initiator and the target do not need support packets that cross address selection boundaries.

The optional Free-BE mode operations are:

- Read: Read any combination of the bytes in the cell as expressed by the BE field.
- Write: Write any combination of the bytes in the cell as expressed by the BE field.
- Read N: Similar to Read N Cells, but with any combination of bytes in the cells enabled with BE field
- Write N: Similar to Write N Cells, but with any combination of bytes in the cells enabled with BE field.

3.4.1.2 Transfer Response

The following is a list of the responses that are allowable by a target across the PVCI:

- Not Ready
- Transfer Acknowledged
- Error (with possible extension specifiers)

3.4.2 Handshake Protocol

The two-wire handshake was introduced in Section 3.2.2, “The Handshake.” The PVCI handshake protocol is a subset of the BVCI handshake protocol. (For more information, see Section 4.2.2, “Technical Introduction to the BVCI.”) Connecting a BVCI target to a PVCI initiator is trivial; the extra signals are tied into constants. The performance achieved is similar to native Peripheral connection. It is also possible to connect a BVCI initiator to a PVCI target, if the initiator knows the limitations of the target.

The VAL-ACK pair of signals provides a flow control for a cell transferring across the VC Interface, and validates all signals associated with that cell. See Figures 7 through 10 for examples.

- VAL==1: Indicates that the initiator is presenting a request.
- ACK==1: Indicates that the target is ready to present a response.

3.4.2.1 Handshake Rules:

R1: **not changeable**: Once an initiator has asserted VAL, it is not permissible to de-assert it or to change any request field, until acknowledged. This rule may be overridden by system timeout function, in a case where prior knowledge exists that the target has stalled and will not respond.

R2: **default_ack**: ACK is permitted to be asserted when VAL is low.

R3: **valid response**: The target must be presenting stable response fields at the rising edge of the clock while both ACK and VAL are asserted.

The initiator must keep VAL asserted, and all of its control signals valid and stable until it receives the ACK signal from the target. The initiator should not assert VAL unless the current transaction is intended for the target.

3.4.2.2 Packet Transfer

The packet (burst) transfer makes it more efficient to transfer a block of cells with consecutive addresses. While the EOP signal is de-asserted during a request, the address of the next request will be ADDRESS+cell_size. The ADDRESS must be aligned to the cell boundary. The packet transfer is completed once a cell is transferred with the EOP signal asserted. In terms of the BVCI operations, the PVCI burst equals to a packet transfer with CONTIG signal asserted and WRAP signal de-asserted. (For more information, see Section 4.3, “BVCI Signal Descriptions.”) The byte-enable signals may have any legal combination in each cell of the packet.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

3.4.2.3 Handshake Examples:

Figure 7 shows waveforms of read and write operations to a target, which needs two clock cycles to complete the read, and one cycle to complete the write. If the RERROR signal is not drawn, it is 0 (“Normal”) for all the examples. Notice that while mixing asynchronous and synchronous ACK behavior is legal, and must be supported by all initiators, it is not recommended for targets.

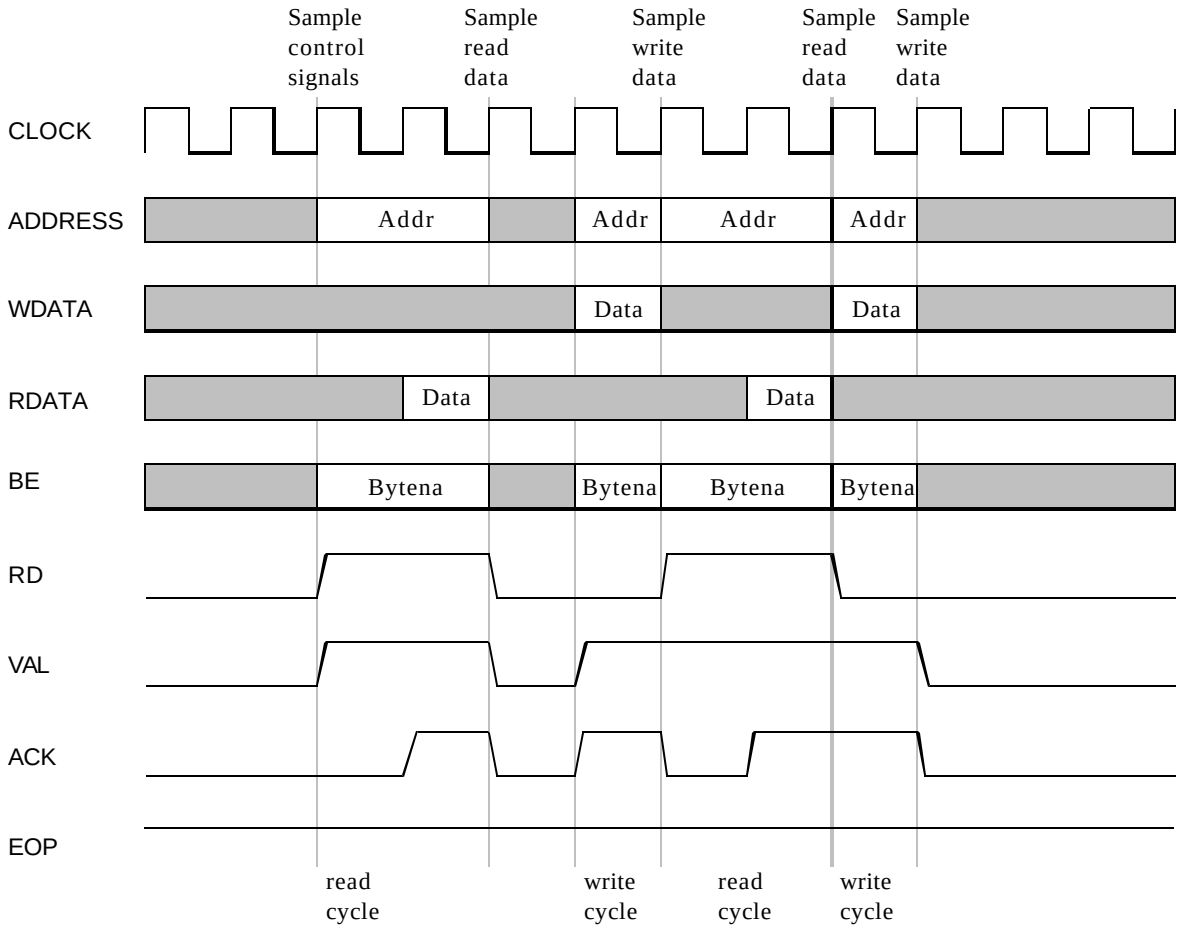


Figure 7: PVCI Read and Write

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

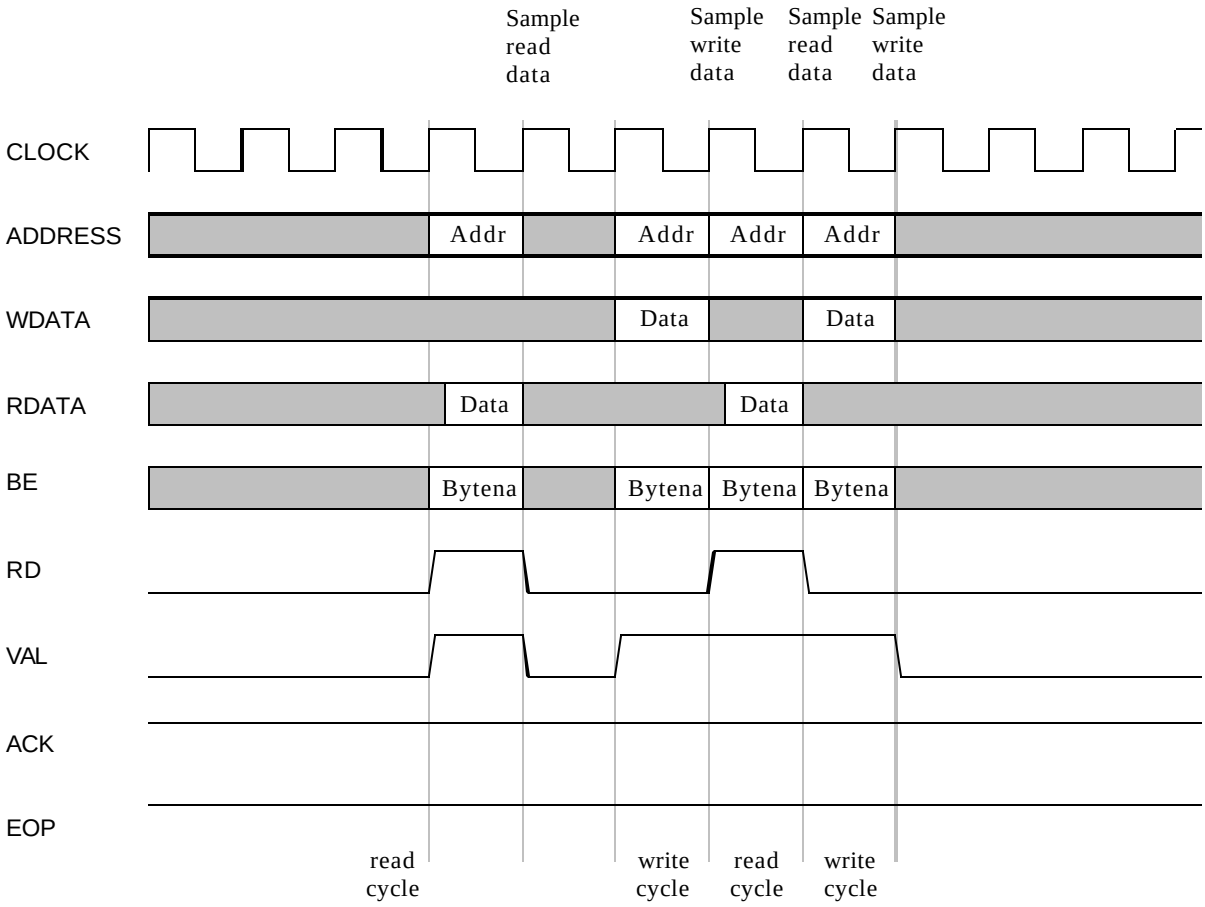


Figure 8: PVCI Read and Write with Default Acknowledge

Figure 8 shows waveforms of read and write operations to a target, which has default acknowledge. That is, a single cycle is required to complete the read and write operations.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

VSIA CONFIDENTIAL LICENSED DOCUMENT

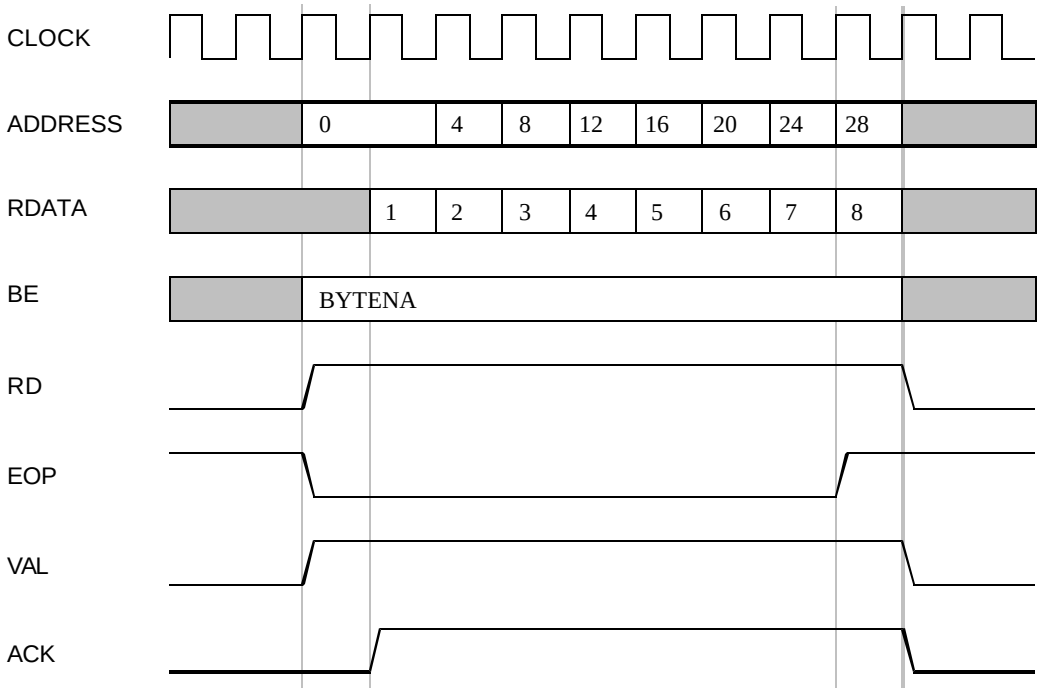


Figure 9: PVCI Burst Read

Figure 9 shows how the EOP signal can be used to indicate address predictability to a similar target, which has an internal address counter to support pre-fetching read data. Notice that the cell size is 4 in the example, resulting in the ADDRESS being incremented by 4 at each read. In this example, the target can respond to the read in two clock cycles in single transfer, and in one cycle in burst transfer. The ACK signal can be generated from the VAL, RD, and EOP signals.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

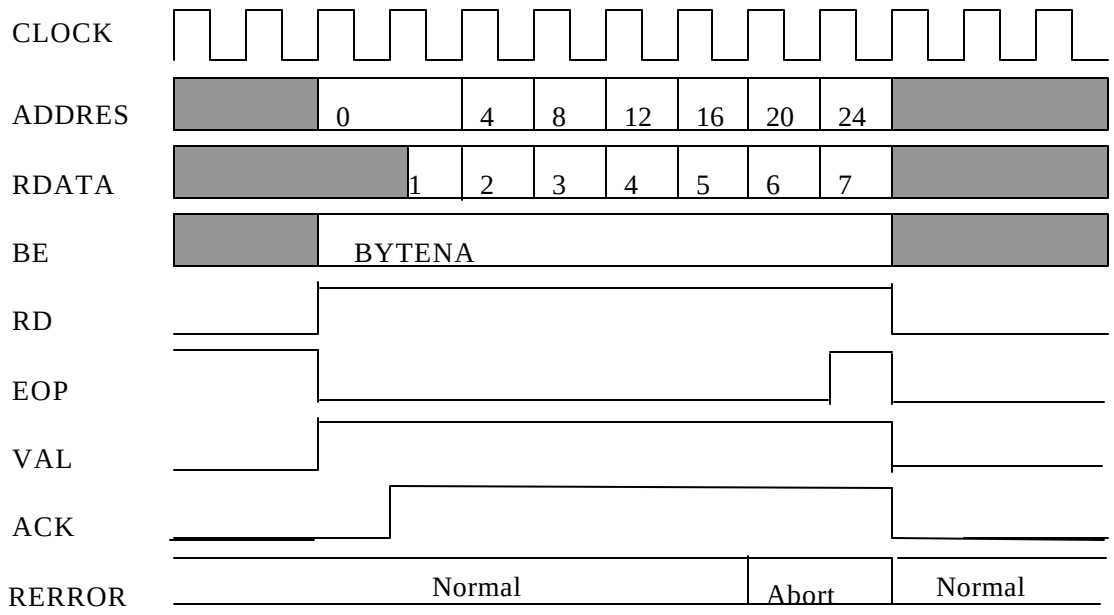


Figure 10: Aborting a Burst

Figure 10 shows how an "Abort" error response terminates a burst transfer.

3.4.3 Data Formatting and Alignment

This section describes the PVCI conventions used for bit and byte ordering and alignment. The PVCI defines bit "0" as the least significant bit of a vectored field, such as WDATA, RDATA, or ADDRESS. It defines bit "8b-1" as the most significant data bit of each data bus, and bit "N-1" as the most significant address bit. N is defined to be the number of physical ports or wires associated with a particular instantiation of a PVCI.

The PVCI is endian-independent at the interface. Even though the naming convention may imply a little endian interface, the DATA[7:0] means address = 0 for little endian, and address = 3 for big endian (in the case of a 32-bit interface). On the other hand, the PVCI component must choose an endianness and declare this to the system (unless the component is a memory, in which case it does not matter). When any component interprets a byte address, it makes a decision about which byte lane to use.

The PVCI uses natural alignment to support peripherals and VCs that handle multiple size transfers (for example, byte and word transfers). This means that transfer sizes that are smaller than the physical width of the data lines (WDATA or RDATA) will occur in their natural byte lanes and not be right or left justified. The byte enable lines (BE) indicate which byte lane(s) contain the desired data. Table 2 shows the various data alignments defined for the PVCI data transfers. The entries labeled "XX" are "don't cares," which provide flexibility and support for byte replication if desired.

The PVCI standard has two operational modes related to byte enables: Default mode and Free-BE mode.

3.4.3.1 Byte Enables in Default Mode

Every PVCI component must support usage of the byte enable lines that are restricted to the contiguous cases. Referring to the following table and the 32-bit example, the allowable patterns for the byte enables are 0000, 0001, 0010, 0100, 1000, 0011, 1100, and 1111. Patterns such as 1011 or 1101 are not allowed.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 2: PVCI Data Alignment

VC Data bus size	Xfer size	BE [3:0|0:3]	Data[31:24]	Data[23:16]	Data[15:8]	Data[7:0]
32	32	Others	Undefined			
		0000	XX	XX	XX	XX
		1111	Byte	Byte	Byte	Byte
	16	0011	XX	XX	Byte	Byte
		1100	Byte	Byte	XX	XX
	8	0001	XX	XX	XX	Byte
		0010	XX	XX	Byte	XX
		0100	XX	Byte	XX	XX
		1000	Byte	XX	XX	XX
VC Data bus size	Xfer size	BE [1:0|0:1]	Data[31:24]	Data[23:16]	Data[15:8]	Data[7:0]
16	16	00			XX	XX
		11			Byte	Byte
	8	01			XX	Byte
		10			Byte	XX
VC Data bus size	Xfer size	BE[0]	Data[31:24]	Data[23:16]	Data[15:8]	Data[7:0]
8	8	0				XX
		1				Byte

In both big or little endian mode, the BE[0] always corresponds to the byte address 0 in the cell.

3.4.3.2 Byte Enables in Free-BE Mode

The Free-BE mode is an optional operation mode for PVCI components. It is legal in this case to support any BE pattern. This mode can be used to make connections to some buses more efficient (for example, a bus that support 3-byte wide transfers). A target that supports the Free-BE mode automatically supports the default mode. A PVCI initiator must not require that the target supports Free-BE mode, but it can take advantage of it if the target does support this mode. The operation mode is selected in component instantiation; it is not a runtime parameter.

For more information, see Section 6.2, “VCI Parameters.”

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4. Basic VCI

This chapter defines a Basic Virtual Component Interface (BVCI) to be used in conjunction with on-chip system buses, and for point-to-point connections between VCs. The BVCI is a subset of the Advanced VCI (AVCI). The BVCI is designed to fulfill most on-chip interfacing protocol needs.

4.1 Organization

This chapter contains the following sections:

- Section 4.1: Describes the organization.
- Section 4.2: Gives a technical introduction to BVCI.
- Section 4.3: Provides a detailed description of BVCI signals.
- Section 4.4: Defines the BVCI protocol.

4.2 Technical Introduction to the BVCI

The following sections give a technical introduction to the BVCI.

4.2.1 Initiator – Target Connection

As shown in Figure 11, the request contents and the response contents are separately transferred under control of the protocol, a simple handshake.

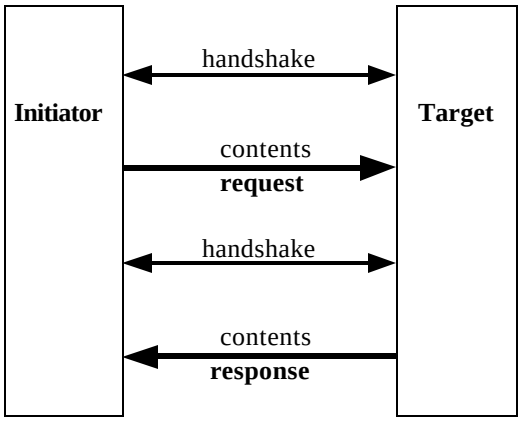


Figure 11: BVCI Request and Response – Handshake and Contents

4.2.2 The Handshake

The BVCI handshake is different from the Peripheral VCI (PVCI) handshake, in that the request and response handshakes are completely independent of each other. The handshake protocol is aimed at synchronizing two blocks by transferring control information in both directions. In the request, the handshake signals are called CMDVAL and CMDACK. In the response, they are called RSPVAL and RSPACK. (These signal names stand for Command Valid, Command Acknowledge, Response Valid, and Response Acknowledge, respectively.) BVCI handshakes can be reduced to PVCI handshakes using the rules presented in Section 6.1.5, “VCI-to-VCI Conversions.”

Request contents flow from initiator to target. Response contents flow from target to initiator. Figure 12 shows the handshake protocol.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

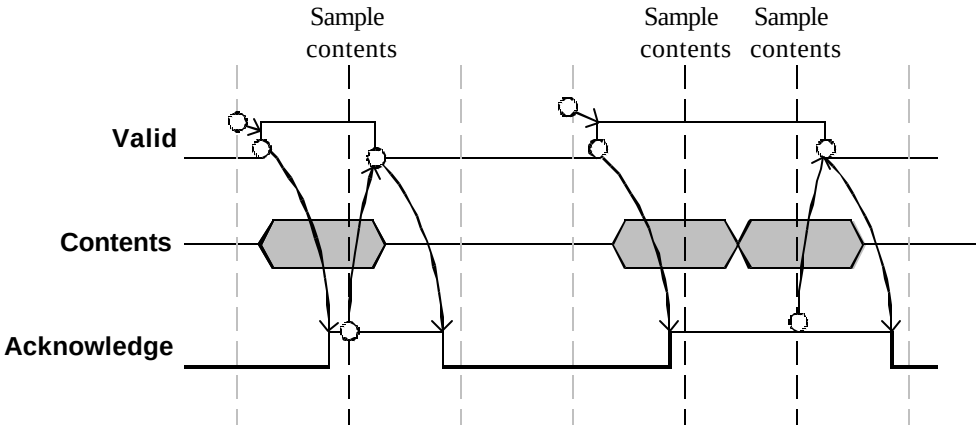


Figure 12: Control Handshake for Both Request and Response (Asynchronous ACK)

Notes for Figure 12:

- Vertical dashed lines show rising clock edges.
- VAL shows "at the next rising edge, contents can be read." No actual timing is specified here.
- ACK, where there is a coinciding VAL, shows "contents will be read." No actual timing is specified here.

As indicated on the right-hand side of Figure 12, maintaining the VAL signal in asserted state for another clock cycle after ACK = 1 means that another request or response is ready for reading. VAL and contents must be maintained until the next rising clock edge after the ACK has become asserted.

The ACK can be either generated asynchronously of the VAL as shown in Figure 12, or set synchronously at the rising edge of the clock as shown in Figure 13 (which shows a slow reaction, or late ACK). If asynchronous ACK is used, special design considerations are needed to make sure that the ACK is stable at the rising clock edge. For this reason, it is not recommended to use acknowledge, which is not generated synchronously except with "default acknowledge" behavior, as shown in Figure 13.

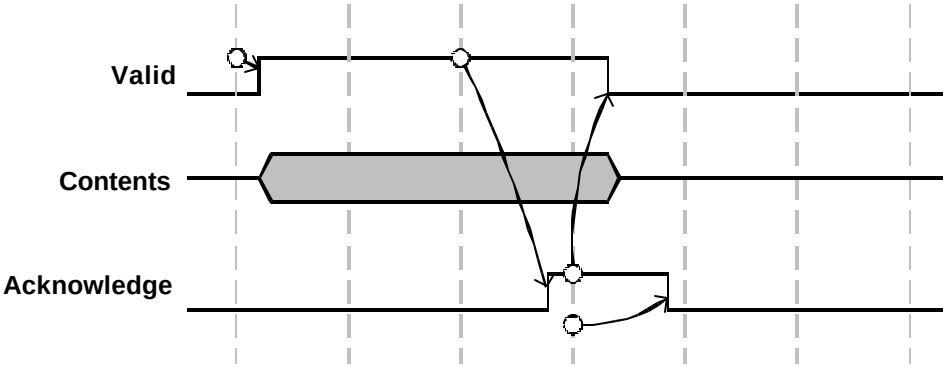


Figure 13: Fully Synchronous Control Handshake (ACK Late by Two Cycles)

4.2.3 The Default Acknowledge

A "default acknowledge" is permitted on top of the protocol. Since the Acknowledge signal is only sampled when VAL = 1, the Acknowledge signal can be asserted long before it is needed. Such an early ACK has no influence on the protocol behavior. The early ACK is merely "don't care," meaning that the signal is not being considered unless VAL is active at a clock edge. This makes it possible to tie ACK permanently active.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.2.4 Cells, Packets, and Packet Chains

4.2.4.1 Cell

Each handshake is used to transfer a cell across the interface. The cell size is the width of the data passing across a VCI. It will typically be 1, 2, 4, 8, or 16 bytes.

4.2.4.2 Packet

Cell transfers can be combined into packets, which may map onto a burst on a bus. A VCI operation consists of a request packet and a response packet. As noted in Chapter 2, although the responses in a packet arrive in the same order as their requests, there is no further relation between the timing of the series of requests and the series of responses. The protocol is a split protocol. Packets have been introduced for three reasons:

1. When connecting the BVCI to a bus, if the underlying bus system is aware that more operations are to follow, no bus arbitration is required between operations of one packet. This may gain valuable bus capacity and speed.
2. Packets are atomic. That is, the point-to-point connection is maintained over the packet, no matter what is between initiator and target. This complex interconnection consists of one or more buses, or "nothing at all." This allows for monopolizing a buffer. Note that long packets block any other use of the connection system. **It is strongly advised to keep packets as short as possible to prevent degradation of system performance.**

Packets are similar in concept to "frames" in Peripheral Component Interconnect (PCI), where a connection is established to enable data to be transferred until the initiator indicates the termination by closing the frame.

4.2.4.3 Packet Chain

Packets can be combined into chains. This allows longer chains of operations to go uninterrupted, as long as no higher priority operation claims part of the connection, such as a bus. Packet chains thus produce the same bus efficiency that packets provide, without actually excluding any other usage of the connection system. The VCI initiator may merely "notice" that the CMDACK to the first operation of a new packet is withheld for a long time. After this pause, the accepting of requests proceeds as if no other usage of the connection system had been served. To obtain this behavior, CMDVAL should be held asserted between the packets of a chain. There is no upper limit to the length of a packet chain.

4.2.5 Request Contents

Request contents are partitioned into three signal groups:

1. Opcode to specify the nature of the request (as read or write)
2. Packet Length and Chaining to control packets
3. Address and Data to further detail a read or write action.

Request contents are validated by the CMDVAL signal.

4.2.5.1 Opcode

This subset of the request contents is constant for all the operations in a packet.

- **Command**: The two-bit field CMD can specify no operation, read operation, write operation, or read locked.
- **Flags** (address algorithm indicators): When none of the flags is asserted, there is no predefined relationship among the addresses of subsequent operations in a packet or in a packet chain. Three flags, though, can indicate a predefined algorithm among subsequent addresses. This allows targets to compute addresses before they actually arrive on the bus and thus, in some cases, to gain valuable clock ticks. The

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

address, even when obeying a predefined algorithm, is sent with each operation, allowing cost effective targets to function without address prediction. The three flags are:

1. Contiguous: Indicates that the addresses within a packet are increasing in a contiguous manner.

2. Wrap (only valid with contiguous addressing): The increase is done modulo the packet length, and only when packet length is a power of two. This allows for a cache line refill starting with the missing word rather than with the word at the lowest address.

3. Constant (no change in address): This mode is useful for the case when a series of FIFOs have contiguous addresses and a packet or a chain is used to empty or fill just one of these FIFOs.

(In the AVCI, more such flags are defined.)

4.2.5.2 Packet Length and Chaining

This subset of the request contents is constant for all the operations in a packet, apart from the indicator "End of Packet," which is only asserted in the last cell of a packet. Contents include:

Packet Length: The length of the packet expressed in bytes.

End of Packet: This bit is asserted in the last operation of a packet. The bit could be considered redundant in th presence of a defined packet length, but it allows a target to be designed without its own explicit remaining length counter. Furthermore, end-of-packet is a necessity with an undefined packet length.

Chain Length: Presents the amount of packets yet to come in a packet chain. Chain length = 0 thus represents a one-packet chain. There is no need for an "eoc" (end of chain).

Chain Fixed: This indicates that the opcode fields (see Section 4.2.5.1) and packet length are equal for all packets of the chain. Otherwise each packet of a chain has its own such fields and flags.

4.2.5.3 Address and Data

This subset of the request contents is issued anew for each cell within a packet. The contents include the following fields:

Address: The address field designates the target if several targets can be reached through the VCI, and the detailed location within the target to which the request is made.

- This standard does not prescribe how target addresses are allocated to address space. For example, four FIFOs may be assigned consecutive word addresses, while memories may occupy Megabytes.
- The address is updated for every operation in a packet even if the address algorithm is specified by means of the flags.
- The address is specified as the lowest byte address of the concerned data.

Byte enable The main role of byte enable is in write operations. Each asserted bit in this field designates a byte in the cell to be actually written into the target. Each non-asserted bit marks a byte not to be overwritten. In read operations, byte enable is needed in those wrappers or bridges where cell length or data alignment changes.

There are two separate naming conventions for byte enable, corresponding to a natural endianness:

- **BE**[3:0] is used for little-endian VCs where the LSB (Least Significant Byte) is labeled Address 0.
- **BE**[0:3] is used for big-endian VCs where the MSB (Most Significant Byte) is labeled Address 0.

Write data: The write data field is only used in write operations. The data lines carry the bytes to be copied to the target. Data are "naturally aligned," so the byte corresponding with byte address modulo cell size = 0 is at byte line 0 on the bus.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.2.6 Response Contents

Each request has its response. The response contents are validated with the RSPVAL signal. Contents include:

Response Error: This field indicates whether the request could be handled correctly.

Read Data: The data returned as a result of a read request. This field has no meaning in write operations. Data are "naturally aligned," so the byte corresponding with byte address modulo cell size = 0 is at byte line 0 on the bus.

4.3 BVCI Signal Descriptions

This section contains a detailed technical description of the BVCI. Descriptions of the signals used between the initiator and target over the BVCI are provided in the following sections.

4.3.1 Signal Type Definition

Table 3 specifies the signal types that are used in Section 4.3. They are defined from the point of view of the devices rather than the wrapper or arbiter.

Table 3: Signal Type Definition

Type	Description
IA	Input to all devices
IT	Generated by the Initiator and sampled by the Target
TI	Generated by the Target and sampled by the Initiator
MA	Mandatory signal for both Initiator and Target
MI	Mandatory signal for the Initiator but an optional signal for the Target
MC	Mandatory signal for supporting chaining function for both Initiator and Target

Signals that do not have one of the mandatory descriptors listed above are optional and do not support the indicated function.

4.3.2 Signal Parameters

Table 4 specifies parameters that are used in Section 4.3.

Table 4: Signal Parameters

Parameter	Description
B	Number of bytes in a cell (must be a power of 2)
K	Number of bits in the **PLEN** field (maximum value is 9)
N	Number of bits in the **ADDR** field (maximum value is 64)
E	Number of bits in the RERROR extension (maximum value is 2)
Q	Number of bits in the **CLEN** field (maximum value is 9)

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.3.3 Signal Directions

Figure 14 diagrams the signal directions between the initiator and the target.

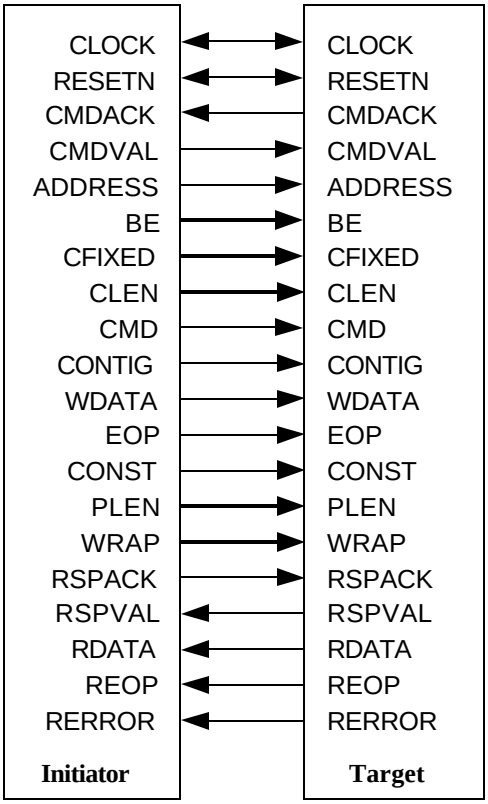


Figure 14: Diagram of VCI Signal Directions

4.3.4 Signal List

All signals included in Table are assumed to be active-high signals unless indicated otherwise. It is recommended that all signal outputs are stable before Early. It can be assumed that all inputs are stable before Late. A detailed signal description follows the summary table.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 5: Signal List

Name	Type	Description
Global Signals		
CLOCK Clock	IA MA	CLOCK provides the timing for all transactions. All signals are sampled on the rising CLOCK edge. All timing parameters are initiated with respect to this edge.
RESETN Reset	IA MA	Reset is used to bring all devices up on a common signal. RESETN is an active low signal and must be asserted for a minimum of eight CLOCK cycles. The rising edge of RESETN is synchronous to the rising edge of CLOCK.
Request Handshake		
CMDACK Command Acknowledge	TI MI	Acknowledge is used by the target to indicate to the initiator that a given cell can be transferred. Hence, a cell is transferred from the initiator when both CMDVAL and CMDACK signals are asserted.
CMDVAL Command Valid	IT MI	Valid indicates that the initiator wishes to perform a cell transfer to the target. The cell is transferred when both the CMDVAL and CMDACK signals are asserted.
Request Content		
ADDRESS[n-1:0] Address	IT MA	ADDRESS is the address of the request generated by the initiator and received by the target. The address updates for every cell transferred within a packet and must remain within the address space of a single target. The pattern of permissable addresses are defined by the flags (CONTIG, WRAP, and CONST signals). ADDRESS contains the lowest byte address for the first transfer in the packet. For all cells after the first transfer, ADDRESS is aligned to a cell boundary. The combination of a cell-aligned address and byte enables is sufficient to perform the transfer correctly. However, the addition of extra information in the first address may allow performance advantage in some systems. Note that a non cell-aligned address is endian dependent.
BE [b-1:0|0:b-1] Byte Enable	IT MA	Byte Enable indicates which bytes of the cell being transferred or requested by the initiator are enabled.
CFIXED Chain Fixed	IT MC	Chain Fixed indicates that the opcode (CMD, CONTIG, WRAP, and CONST) and PLEN fields will be constant across the chain, and that the address field behavior is the same among packets within a chain.
CLEN[q-1:0] Chain Length	IT MC	Chain Length indicates the number of packets remaining in a chain. The last packet transferred in a chain has a zero CLEN value. The CLEN value can also be tied off to zero if packet chaining is not required.
CMD[1:0] Command	IT MA	Command is a 2-bit code defining the operation type being attempted by the Initiator to the Target and is encoded as follows: 00b = NOP, no data is actually transferred (optional) 01b = READ, data is requested by the initiator from the target 10b = WRITE, data is transferred from the initiator to the target 11b = LOCKED READ, data is requested by the initiator from the target with the added function of locking out access to at least the particular cell until a WRITE packet is transferred to the same cell (optional)

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 5: Signal List (Continued)

Name	Type	Description
Request Content (continued)		
CONTIG Contiguous	IT	CONTIG indicates that the sequence of addresses that will be performed within the packet is contiguous. When CONTIG is asserted, the address is normally increased by the cell size in bytes. If a packet does not start or end at a cell boundary, the first or last transfers contain less bytes than a cell. In such cases, the address is increased not by the length of a cell but by the amount of bytes actually transferred in such a first or last cell.
WDATA[8b-1:0] Data	IT MA	WDATA is the data transferred by the initiator to the target during a WRITE command. The actual data bits that are enabled during a given cell transfer are defined by the byte enables. The most significant bit is always on the LHS – bit 8b-1. The least significant bit is always on the RHS – bit 0.
EOP End Of Packet	IT MI	End Of Packet is asserted on the last cell of a packet, indicating that all cells associated with the given packet have been transferred.
CONST Constant	IT	CONST indicates that the address will remain constant throughout the entire packet. When CONST is asserted, CONTIG and WRAP are ignored.
PLEN[k-1:0] Packet Length	IT	Packet Length indicates the length of the packet in bytes. The valid range for PLEN is 1 to 2^k-1 (1 to 511 given that k is limited to 9). A value of 0 for PLEN can be used to indicate that the packet length is undefined (no implied packet length).
WRAP Wrap	IT	WRAP is used in conjunction with CONTIG to specify how to handle addresses that increment past the boundary indicated by PLEN. If WRAP is asserted and PLEN has only a single bit set (indicating a power of 2 packet size), the address will wrap around. It is illegal to set WRAP otherwise.
Response Handshake		
RSPACK Response Acknowledge	IT MA	Response Acknowledge is used by the initiator to indicate to the target that a given cell will be transferred. Hence, a response cell is transferred from the target when both RSPVAL and RSPACK signals are asserted.
RSPVAL Response Valid	TI MA	Response valid indicates that the target wishes to perform a response cell transfer to the initiator. The response cell is transferred when both the RSPVAL and RSPACK signals are asserted.
Response Content		
RDATA [8b-1:0] Response Data	TI MA	Response data is the data that is returned by the target to the initiator with read operations.
REOP Response End Of Packet	TI MA	Response End of Packet is asserted on the last cell of the response packet, indicating that all cells associated with the given response packet have been transferred.
RERROR Response Error	TI	Response Error is asserted by the target during a response packet to indicate that an error has occurred during the current packet.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.3.5 System-Level Signals

4.3.5.1 Clock

Signal Name:	Clock
Signal Abbreviation:	CLOCK
Polarity:	Active at Positive edge
Driven By:	System
Received By:	VCI initiator, VCI target

This signal provides the timing for the VCI and is an input to both the initiator and target that are connected via the BVCI. All initiator and target output signals are asserted/de-asserted relative to the rising edge of CLOCK and all initiator and target inputs are sampled relative to this edge.

4.3.5.2 Reset

Signal Name:	Reset
Signal Abbreviation:	RESETN
Polarity:	Asserted Negative
Driven By:	System
Received By:	VCI initiator. VCI target
Timing:	Asserted > 8 or RESETLN clock cycles

This signal is used during power-on reset, and is used to bring the BVCI to an idle or quiescent state. This idle state is defined as the BVCI state in which:

1. The [CMD|RSP]VAL signals are de-asserted.

2. The [CMD|RSP]ACK signals are de-asserted.

The system must guarantee that RESETN is asserted for at least eight cycles of CLOCK (unless the RESETLEN parameter is set).

4.3.6 Request Signals

4.3.6.1 Command Valid

Signal Name:	CommandValid
Signal Abbreviation:	CMDVAL
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK == 1 and next rising edge of CLOCK.

The CMDVAL signal is driven by a VCI initiator to indicate that there is a valid request cell on the BVCI. All of the initiator request signals are qualified by CMDVAL. The initiator keeps CMDVAL asserted, and all of its control signals valid and stable, until it receives the CMDACK signal from the target. The initiator should not assert CMDVAL unless the current transaction is intended for the target. Thus, the initiator may need to perform address decoding on its on-chip bus side to generate CMDVAL for the target, thereby accomplishing device selection.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.3.6.2 Command Acknowledge

Signal Name:	CommandAcknowledge
Signal Abbreviation:	CMDACK
Polarity:	Asserted Positive
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted and negated at rising edge of CLOCK

The CMDACK signal is asserted by the target to indicate the completion of a request between the initiator and target. In the case of write operations, this means that the target has accepted the data that is on the write data bus, or will do so at the end of the current clock cycle. In the case of read operations, the assertion of the CMDACK by the target indicates that the target has accepted the request and has started processing it. The request completes as soon as the rising edge of CLOCK samples CMDACK. The target may de-assert the CMDACK by the next rising edge of CLOCK unless a new command has been initiated by the initiator, or default acknowledge is used. Similarly, the initiator de-asserts CMDVAL by the next rising edge of CLOCK unless it is presenting a new request.

4.3.6.3 Command

Signal Name:	Command
Signal Abbreviation:	CMD
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK == 1 at rising edge of CLOCK

This signal is a two-bit command field asserted by the initiator, and indicates the nature of the requested transfer. It is encoded as follows:

- 00b = NOP, no data is actually transferred (optional)
- 01b = READ, data is requested by the initiator from the target
- 10b = WRITE, data is transferred from the initiator to the target
- 11b = LOCKED READ, data is requested by the initiator from the target with the added function of locking out access to at least the particular cell until a write operation is performed to the same cell-address. In contiguous address mode, all packed addresses are locked. In other address modes, only the first accessed cell address is locked.

This signal must be valid any time that the CMDVAL signal is asserted.

4.3.6.4 End-of-Packet

Signal Name:	End-of-Packet
Signal Abbreviation:	EOP
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

The EOP signal is de-asserted by the initiator to indicate that the transfer being performed will be followed with a transfer by the initiator to the next higher cell address. This signal is used by the target device to pre-calculate the address in order to improve the data transfer performance. The packet transfer is completed once a cell is transferred with the EOP signal asserted.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.3.6.5 Chain Fixed

Signal Name:	ChainFixed
Signal Abbreviation:	CFIXED
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

Chain Fixed indicates that the opcode (CMD, CONTIG, WRAP, and CONST) and PLEN fields will be constant across the chain and the address field behavior will be identical among packets within a chain.

4.3.6.6 Chain Length

Signal Name:	ChainLength
Signal Abbreviation:	CLEN
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

Chain Length indicates the number of packets remaining in a chain. The last packet transferred in a chain will have a zero CLEN value. The CLEN value can also be tied off to zero if packet chaining is not required.

4.3.6.7 Contiguous

Signal Name:	Contiguous
Signal Abbreviation:	CONTIG
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

Contiguous-signal indicates that the sequence of addresses that will be accessed within the packet is contiguous. When CONTIG is asserted, the address is normally increased by the cell size in bytes. If a packet does not start or end at a cell boundary, the first or last transfer contains less bytes than a cell. In such cases, the address is increased not by the length of a cell but by the amount of bytes actually transferred in the first or the last cell. In the middle part of the packet, the address always increases by cell size when CONTIG is active.

4.3.6.8 Packet Length

Signal Name:	PacketLength
Signal Abbreviation:	PLEN
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

Packet Length indicates the length of the packet in bytes. The valid range for PLEN is 1 to 2^k-1 (1 to 511, given that k is limited to 9). A value of 0 for PLEN can be used to indicate that the packet length is undefined (there is no implied packet length). In this case, the packet ends as a cell is transferred with EOP active. PLEN is held constant over the packet. Thus, PLEN does not indicate “remaining length.”

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

PLEN does not indicate the actual transferred bytes (the number of enable bytes in a packet). The PLEN=CELLSIZE*(transferred cells-2)+index of first active BE bit in the first cell + index of the last active BE bit in the last cell. For example, if CELLSIZE=4, PLEN=12, cells transferred=4. If BE=x100 in the first cell, it must be 001x in the last cell. The first and last active bytes indicate the start and end of the packet. If BE=xx10 in the first cell, it must be xxx1 in the last cell, and so on. The middle cells of the packet can have any combination of the BE. This behavior facilitates connecting VCIs with different CELLSIZE together, and not breaking packet addressing.

4.3.6.9 Constant

Signal Name:	Constant
Signal Abbreviation:	CONST
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

Constant indicates that the address will remain constant throughout the entire packet. When CONST is asserted, CONTIG and WRAP are ignored.

4.3.6.10 Wrap

Signal Name:	Wrap
Signal Abbreviation:	WRAP
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

Wrap is used in conjunction with CONTIG to indicate how addresses that increment past the boundary indicated by PLEN are handled. If WRAP is asserted and PLEN has only a single bit set (indicating a power of 2 packet size), the address will wraparound. If the packet length is not power of two or if CONTIG is not active, the WRAP signal is "don't care." The wrap address equals (ADDR and not [PLEN-1]) + PLEN. The next address after wrapping equals (ADDRESS AND NOT [PLEN-1]).

4.3.6.11 Address

Signal Name:	Address
Signal Abbreviation:	ADDRESS[n-1:0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

ADDRESS is the address of the request generated by the initiator and received by the target. The address updates for every cell transferred within a packet and must remain within the address space of a single target. The pattern of addresses that are permissible is defined by the operation type (CONTIG, WRAP, and CONST signals). ADDRESS contains the lowest byte address for the first transfer in the packet. For all cells after the first transfer, ADDRESS is aligned to a cell boundary. The combination of a cell-aligned address and byte enables is sufficient to perform the transfer correctly. However, the addition of extra information in the first address may allow performance advantage in some systems. Note that a non cell-aligned address is endian dependent.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.3.6.12 Byte Enable

Signal Name:	Byte Enable
Signal Abbreviation:	BE[**b**-1:0|0:**b**-1]
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

BE is a b-bit field that indicates which bytes of the cell being transferred are enabled. The b equals to total data width of the BVCI/8. These signals must be valid any time that the CMDVAL signal is asserted. In write transfers, the disabled bytes are not overwritten. In read transfers, the target may or may not present the disabled bytes in the RDATA bus. They are ignored by the initiator. The direction of the BE-signal range numbering depends upon the endiannes s. See also PLEN.

4.3.6.13 Write Data

Signal Name:	Write Data
Signal Abbreviation:	WDATA[8**b**-1:0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK

The write data lines are driven by the VCI initiator, and are used to transfer write data from an initiator to a target device. Write data consists of **b** logical byte lanes, based upon the capabilities of the target, and is defined and fixed at the time of component instantiation. Allowed values of **b** are powers of two. Bit 8***b**-1 is the most significant bit of the most significant byte, and bit 0 is the least significant bit of the least significant byte. The write data lines must contain valid write data while the CMDVAL signal is asserted and the CMD is indicating a write transfer.

For VCs supporting a data size that is not an 8-bit increment, the next larger supported bus size will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide VCI with the four most significant bits tied to logic zero.

4.3.7 Response Signals

4.3.7.1 Response Valid

Signal Name:	ResponseValid
Signal Abbreviation:	RSPVAL
Polarity:	Asserted Positive
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted at rising edge of CLOCK until RSPACK = 1 at rising edge of CLOCK

The RSPVAL signal is driven by a VCI target to indicate that there is a valid response on the BVCI. All of the target response signals are qualified by RSPVAL. The target keeps RSPVAL asserted, and all of its control signals valid and stable, until it receives the RSPACK signal from the target.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.3.7.2 Response Acknowledge

Signal Name:	ResponseAcknowledge
Signal Abbreviation:	RSPACK
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted and negated at rising edge of CLOCK

The RSPACK signal is asserted by the target to indicate the completion of a response transfer between the initiator and target. This means that the initiator has accepted the response data, which is on the target's response signals, or will do so at the end of the current clock cycle. The response completes as soon as the rising edge of CLOCK samples RSPACK. The initiator may de-assert the RSPACK by the next rising edge of CLOCK unless a new response has been initiated by the target, or default acknowledge is used.

4.3.7.3 Read Data

Signal Name:	Read Data
Signal Abbreviation:	RDATA[8b-1: 0]
Polarity:	N/A
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted at rising edge of CLOCK until RSPACK = 1 at rising edge of CLOCK

The read data lines are driven by the VCI initiator, and are used to transfer read data from a target to an initiator device. Read data consists of **b** logical byte lanes, based upon the capabilities of the target, and is defined and fixed at the time of component instantiation. Allowed values of **b** are powers of 2 bytes. Bit 8*__b__-1 is the most significant bit of the most significant byte, and bit 0 is the least significant bit of the least significant byte. The read data lines must contain valid read data while the RSPVAL signal is asserted and the response is to a read request.

For VCs supporting a data size that is not an 8-bit increment, the next larger supported bus size will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide BVCI with the four most significant bits tied to logic zero.

4.3.7.4 Response End-of-Packet

Signal Name:	ResponseEndofPacket
Signal Abbreviation:	REOP
Polarity:	Asserted Positive
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted at rising edge of CLOCK until RSPACK = 1 at rising edge of CLOCK

Response End-of-Packet is asserted on the last cell of the response packet, indicating that all cells associated with the given response packet have been transferred.

4.3.7.5 Response Error

Signal Name:	Response Error
Signal Abbreviation:	RERROR[E:0]
Polarity:	Positive
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted at rising edge of CLOCK when RSPVAL = 1, until RSPACK = 1 at rising edge of CLOCK

Error signal is valid only when RSPVAL=1, with the following meaning:

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

For ERRLEN = 0 (E=0)
RERROR=0: Normal (no error)
RERROR=1: General data error. The entire packet is considered bad.

For ERRLEN = 1 (E=1)
RERROR = 00: Normal (no error)
RERROR = 01: General data error. The entire packet is considered bad.
RERROR = 10: Reserved
RERROR = 11: Abort Disconnect

For ERRLEN = 2 (E=2)
RERROR =000: Normal (no error)
RERROR = xx0: Reserved
RERROR = 001: General data error. The entire packet is considered bad.
RERROR = 011: Reserved
RERROR = 101: Bad data (retry)
RERROR = 111: Abort Disconnect

After receiving an error, the initiator may or may not continue with the current packet. If it chooses to end the packet prematurely, it can do so by asserting EOP regardless of the PLEN value. After this, it can choose not to try a transfer anymore (to Abort), or to Retry part or all of the transfer. For any error, the target must process the subsequent pending cells and packets with the normal protocol. That is, it must continue sending responses until it has processed the EOP, with or without further errors signaled. In general, the RERROR is more informative than prescriptive, and the target may not assume any special behavior from the initiator. The initiator is encouraged to act responsibly when it meets an error.

It is strongly recommended that all BVCI components support at least one bit error. It is strongly recommended that the BVCI target signal an error when it receives a request it does not support. Obviously, VCI provides a possibility to get error records through normal Read-requests from the target, but this belongs to domain of a particular VC implementation.

4.4 BVCI Protocol

The VCI protocol is described as a set of three stacked layers: the transaction layer, the packet layer, and the cell layer.

4.4.1 Transaction Layer

The transaction layer is above the concerns of hardware implementation. It defines the system as a series of communicating objects that can be either hardware or software modules. The information exchanged between initiator and target nodes is in the form of a request-response pair, as illustrated in Figure 15.

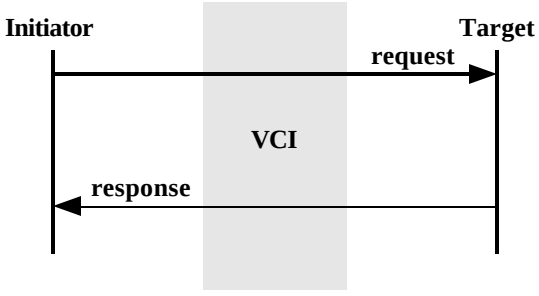


Figure 15: System Transaction Layer View of Information Transfer over the VCI

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.4.1.1 Transaction

A pair of request and response transfers is called a *transaction*. Typically, the basic unit of information exchanged is some form of data structure. As the communication is decomposed to lower layers of abstractions, the basic transfer unit becomes smaller.

4.4.2 Packet Layer

The packet layer adds generic hardware constraints to the system model. In this layer, VCI is a bus-independent interface that supports point-to-point physically address-mapped split transactions between initiators and targets in unit time. There is not yet a commitment to a particular interface width. In this layer, the request and response information to be transferred across the interface is split into more manageable chunks, on the basis of generic hardware constraints. This is illustrated in Figure 16.

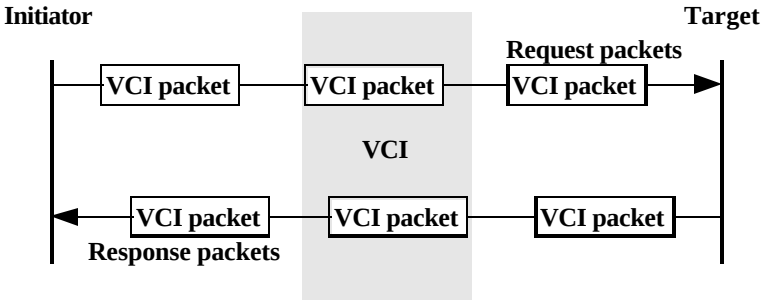


Figure 16: Packet Layer View of Information Transfer over the VCI

4.4.2.1 Operation

A transaction is called a VCI *operation* if the information is exchanged using atomic request and response transfers across the interface. The information exchanged during an operation is in the form of a VCI packet, a packet layer transfer unit defined in the following section. In a packet layer, a VCI transaction decomposes into one or more operations.

4.4.2.2 Packet

A *packet* is the basic unit of information that can be exchanged over the VCI in an atomic manner. The point-to-point connection between initiator and target is maintained throughout the transfer of a packet.

The request and response transfer units of a VCI transaction are decomposed into one or more request-response packet pairs. Multiple packets can be combined to form larger, non-atomic transfer units called packet chains, as explained in the next section. A VCI operation is a single request-response packet pair. For each request packet transferred from initiator to target, there is one corresponding response packet transferred from target to initiator. The targets return response packets in the same order as request packets are issued.

The content of a packet depends on whether it is a request or response packet and the type of operation being carried out, such as read or write. The data field of request packets is relevant only for write operations. The field, RDATA, in response packets contains valid values only for read and locked_read operations. The request packet header contains information on packet chaining. More details on other packet fields appear in subsequent sections of this document.

Packet length is the number of bytes transferred during a read, write, or locked_read operation. This field is irrelevant for NOP operations. A zero value specified for PLEN in read, write, or locked_read operations indicates that the length of the packet is undefined. Long packets can result in reduced arbitration overhead and some optimization in certain areas, like read pre-fetching and SDRAM access. This can improve the data transfer rate for that particular thread. However, very long packets transferred over a shared interconnect system lock out all other agents, causing degradation of the system performance. Hence, it is highly recommended to use short packets to optimize the overall system performance and to use the packet chaining mechanism, described in the next section, to create long, but non-atomic, transfers.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.4.2.3 Packet Chain

The *packet chain* mechanism allows a VCI initiator to describe linkage between related packets to the system. The system can combine chained packets into larger, more efficient, higher performance transfers that satisfy the constraints of the system application. Packet chains produce the same transfer efficiency that long packets provide without requiring the entire transfer to be atomic.

Two fields in the request packet header, *CLEN* and *CFIXED*, specify the packet chaining information. *CLEN* describes the number of packets remaining in the chain (not including the current packet). This field is normally decremented with every transmitted packet.

The flag, *CFIXED*, provides information about the linkage between the header fields among the packets in a packet chain. If set, *CFIXED* guarantees that the opcode and packet length fields are constant across the chain.

The packet chain mechanism allows the VCI initiators to minimize their packet length to that which is required for proper operation (that is, functionality). This mechanism concurrently provides the interconnect logic and the target with enough information for optimizing the chained transfers to meet application performance and efficiency requirements.

4.4.3 Cell Layer

The cell layer adds more hardware details such as interface width, handshake scheme, wiring constraints, and a clock to the system model. This layer is not concerned about shared interconnects and arbitration for such services. In the cell layer, VCI is viewed as a bus-independent, point-to-point interface that supports physically address-mapped split transactions between initiators and targets, using a cycle-based handshake protocol. Introduction of interface width information enables decomposing of packets (the basic transfer unit defined in the packet layer) into cells that are handshaken under a cycle-based protocol across the definitive sized interface.

4.4.3.1 Cell

A cell is the basic unit of information, transferred on rising CLOCK edges under the VAL-ACK handshake protocol, defined by the cell layer. Multiple cells constitute a packet. Both request and response packets are transferred as series of cells on the VCI. The number of cells in a packet depends on the packet length and the interface width.

The structure of a cell is very similar to that of a packet, except for the decomposing of WDATA and byte enable fields and the introduction of end-of-packet fields. The opcode fields, chaining information, and PLEN in a request cell are the same as the corresponding request packet fields.

The request cell structure and response cell structure are detailed in Table 6 and Table 7.

Table 6: Request Cell Structure

Comment		Field	Description
Packet chain information		CLEN	Number of remaining packets in the chain
		CFIXED	Indicates if opcode and PLEN are the same across all packets in the chain
		ADDR	Address of each cell being transferred
		PLEN	Packet length: Total number of enabled and disabled data bytes transferred in the operation
Opcode		CMD	Command: read, write, nop, or locked_read
	Flags	CONTIG	Contiguous address mode
		WRAP	Wrapped address mode
		CONST	Constant address mode
		BE	Byte Enable: indicates which bytes are involved in the operation

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 6: Request Cell Structure (Continued)

Comment	Field	Description
Optional field	WDATA	Data in write requests
	EOP	Indicates if the cell is the last one in the request packet

Table 7: Response Cell Structure

Comment	Field	Description
	RERROR	Error status for the cell
Optional field	RDATA	Data in read, locked_read responses
	REOP	Indicates if the cell is the last one in the response packet

4.4.3.2 VAL- ACK Handshake

The VAL-ACK handshake provides unambiguous synchronization of initiator and target modules for transferring cells over the interface. It is a simple handshake protocol based on two control signals. Two separate sets of handshake signals are used in the VCI interface: one for transferring request cells from initiator to target, and the other for transferring response cells from target to initiator. The transfer of request and response cells over the interface are completely de-coupled, concurrent events. The handshake fundamentals explained below apply to both request and response channels. Generic names, VAL and ACK, are used in the discussion for the control signals. These names represent CMDVAL and CMDACK signals of the request channel and RSPVAL and RSPACK signals of the response channel.

Cell transfer on the channel is solely controlled by the following two signals, as shown in Table 8.

Table 8: Handshake Signals

Signal	Description
VAL	Driven by an initiator module to indicate that a cell was placed on the interface and is ready for transfer
ACK	Driven by a target module to indicate that it has received the cell, if any was present as announced by VAL

Table 9 summarizes different encoding for the VAL-ACK signals and the corresponding channel states.

Table 9: VAL-ACK Encoding and Channel States

VAL	ACK	State	Description
0	0	IDLE	The channel is in idle state.
1	0	VALID	Valid is asserted. Waiting for Acknowledge.
0	1	DEFAULT_ACK	Ready to Grant
1	1	SYNC	Handshake synchronization

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

The channel is in IDLE state when both VAL and ACK are de-asserted. The initiator module unconditionally asserts VAL when the cell information is placed on the interface. The target module may delay the assertion of ACK if it is not ready to accept the cell. The channel is said to be in REQUEST state if ACK is not asserted for a VAL on the rising edge of the CLOCK. The channel state transitions to SYNC when both VAL and ACK are asserted. The handshake synchronization and cell transfer occur on each CLOCK edge when the VCI is in SYNC state. (See Figure 17.)

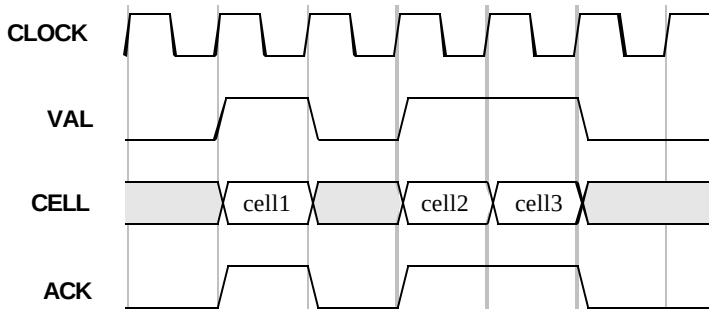


Figure 17: VAL-ACK Handshake for Single-Cell Transfer

While transferring a multi-cell packet, either module can insert wait cycles by de-asserting VAL or ACK. The receiving module can assert ACK, even when VAL is not present, to indicate that it is ready to ACK the next VAL. The corresponding channel state is DEFAULT_ACK. In this case, cell transfer will occur on the first CLOCK edge on which VAL is asserted.

Once the initiator module asserts VAL, it cannot change that signal until the requested cell is transferred, regardless of the state of ACK. Similarly, once the receiving module asserts ACK, it should not de-assert that signal until a cell is transferred. Generally, neither module shall change its mind once it has committed to the cell transfer.

However, de-committing the ACK signal may be necessary in certain scenarios, such as a default ACK from a bus wrapper module to an initiator, where this bus is “parked.” Such deviations from the recommended interface behavior should be clearly documented when VCs are implemented. Table 10 summarizes all the permitted state transitions.

Table 10: Permitted State Transactions

From	To	Validity	Description
IDLE	IDLE	Valid	Continues in idle
IDLE	VAL	Valid	New Valid asserted
IDLE	DEFAULT_ACK	Valid	Default Acknowledge asserted
IDLE	SYNC	Valid	New Valid acknowledged on same CLOCK
VAL	IDLE	Invalid	Valid de-committing not allowed
VAL	VALID	Valid	Valid maintained
VAL	DEFAULT_ACK	Invalid	Valid de-committing not allowed
VAL	SYNC	Valid	Valid Acknowledged
DEFAULT_ACK	IDLE	Invalid	Acknowledge de-committing not allowed
DEFAULT_ACK	VAL	Invalid	Acknowledge de-committing not allowed
DEFAULT_ACK	DEFAULT_ACK	Valid	Acknowledge maintained

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 10: Permitted State Transactions (Continued)

From	To	Validity	Description
DEFAULT_ACK	SYNC	Valid	Valid Acknowledged
SYNC	IDLE	Valid	Back to idle
SYNC	VAL	Valid	New Valid asserted
SYNC	DEFAULT_ACK	Valid	Default Acknowledge asserted
SYNC	SYNC	Valid	Consecutive Valid-Acknowledge

4.4.3.3 VAL Time and ACK Time

VAL_time and ACK_time are introduced as a way of classifying the handshakes. The definitions are:

- VAL_time = number of cycles where VAL is de-asserted between cell transfers.
- ACK_time = number of cycles where VAL is asserted before ACK is asserted.

Figures 18 through 20 illustrate different VAL-ACK handshakes.

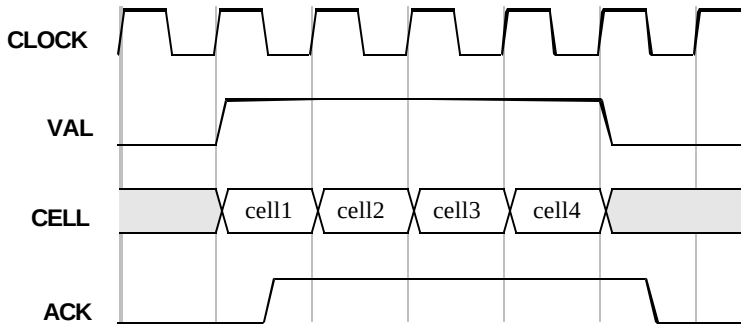


Figure 18: VAL-ACK Handshake with VAL Time = 0, ACK Time = 0

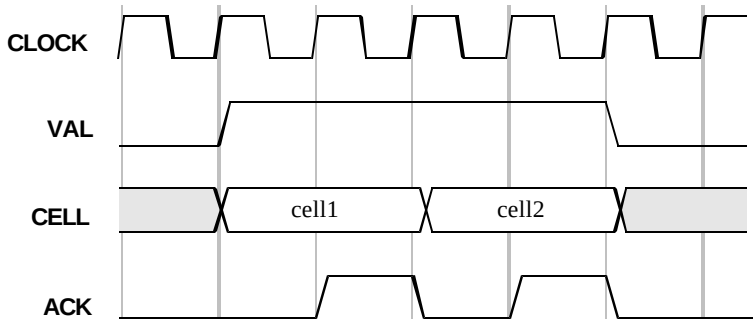


Figure 19: VAL-ACK Handshake with VAL Time = 0, ACK Time = 1

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

VSIA CONFIDENTIAL LICENSED DOCUMENT

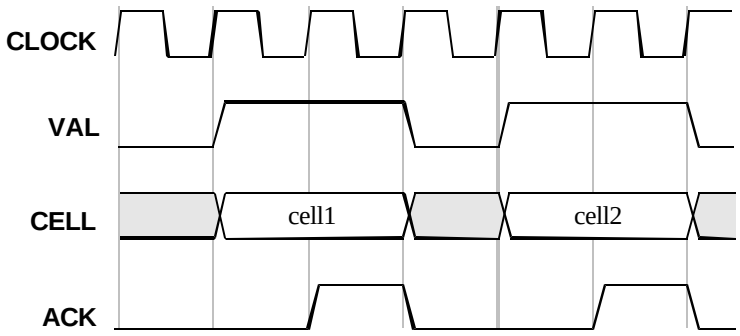


Figure 20: VAL-ACK Handshake with VAL Time = 1, ACK Time = 1

4.4.4 BVCI Operations

The basic transfer mechanism in BVCI is packet transfer. Every operation on the VCI consists of a request packet transfer from the initiator to the target and a response packet in return. Both initiator and target modules are responsible for transferring the VCI request and response packets across the interface in an atomic manner. A packet is sent as a series of cells with the EOP (end of packet) field in the last cell set to value 1. Each cell is individually handshaken across the interface under the VAL-ACK handshake. Either the initiator or the target can insert wait cycles between cell transfers by de-asserting VAL or ACK. The order of packets and the number and order of cells within packets should be maintained the same between request and response transfers.

The transfer of request and response cells over the interface is a completely de-coupled concurrent event. The signals used for encoding cell information, handshake signals, and the timing of the request and response transfers are totally separate.

All VCI signals are sampled on the rising edge of the CLOCK. The handshake signals CMDVAL, CMDACK, RSPVAL, and RSPACK are sampled on every rising CLOCK edge. The rest of those signals that encode the request and response cells are sampled on rising CLOCK edges, qualified by the corresponding VAL signal. The timing diagrams in this section show the relationship of relevant signals involved in illustrated operations.

4.4.4.1 Transfer Requests

The following is a complete list of operation types that may be initiated across the BVCI. See the VCI Parameters section of this document for definition of the parameters indicated in bold text type.

- *Read a cell* with any or all of bytes active.
- *Read and lock a cell* with any or all of bytes active.
- *Write a cell* with any or all of bytes active (and release lock).
- *Read a packet of plen bytes from random addresses*, with any combination of bytes enabled in each cell.
- *Read a packet of plen bytes from random addresses*, with any combination of bytes enabled in each cell, and lock the first accessed address.
- *Write a packet of plen bytes to random addresses*, with any combination of bytes enabled in each cell, and release the lock of the first accessed address.
- *Read a packet of plen bytes from contiguous addresses*, with any combination of bytes enabled in each cell.
- *Read a packet of plen bytes from contiguous addresses*, with any combination of bytes enabled in each cell, and lock the accessed addresses.
- *Write a packet of plen bytes to contiguous addresses*, with any combination of bytes enabled in each cell, and release locks.
- *Read a packet of plen bytes from contiguous addresses*, wrapping the address at boundary indicated by PLEN, with any combination of bytes enabled in each cell. Packet length must be a power of 2.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

- *Read a packet of plen bytes from contiguous addresses*, wrapping the address at boundary indicated by PLEN, with any combination of bytes enabled in each cell, and lock the accessed addresses. Packet length must be a power of 2.
- *Write a packet of plen bytes to contiguous addresses*, wrapping the address at boundary indicated by PLEN, with any combination of bytes enabled in each cell, and release locks. Packet length must be a power of 2.
- *Read a packet of plen bytes from one address*, with any combination of bytes enabled in each cell.
- Read a packet of plen bytes from one address, with any combination of bytes enabled in each cell, and lock the address.
- *Write a packet of plen bytes to one address*, with any combination of bytes enabled in each cell, and release lock.
- When Packet Chaining is supported (CHAINING = True):
  - *Issue a chain of packets* (chain any of the above transactions), so that all the other packet header fields are identical over the chain but the address and chain length counter (chain fixed).
  - *Issue a chain of packets* (chain any of the above transactions), so that all packets may have different header fields (chain not fixed).

4.4.4.2 Transfer Responses

The following is a list of responses that are allowable by a target across the BVCI.

- *Read of cell/packet successful*, return read data.
- *Write of cell/packet successful*.
- *Read packet general error*, the entire packet bad, return data invalid.
- *Write packet general error*, the entire packet bad.
- *Read bad data error*, suggestion to retry transfer of packet or cell, return data invalid.
- *Write bad data error*, suggestion to retry transfer of packet or cell.
- *Read/Write Abort disconnect*, suggestion for not to try the transfer again.

4.4.5 Read Operation

Figure 21 illustrates a 32-byte read operation on a 32-bit wide VCI. The operation starts with the initiator placing the first cell of a request packet on the interface and asserting CMDVAL on CLOCK 2. The cell information is encoded in the signals: CMD, CONTIG, WRAP, CONST, PLEN, EOP, ADDRESS, and BE. The fields on chaining information, CLEN and CFIXED, are both set to value 0 for this operation and are not shown in the diagram.

As shown in Figure 21:

- The CMD and addressing mode flags, CONTIG, WRAP, and CONST, specify that the packet is part of a read operation with contiguous cell addresses.
- The field PLEN indicates that the packet length is 32 bytes.
- The address field contains the address of the target and the byte address of the location from where the data is requested.
- BE indicates which byte lanes are involved in the first cell transfer.
- EOP is set to 0 to indicate that the current cell is not the last one in the packet.

The earliest the cell transfer can complete is in CLOCK period 2 itself, if the signal CMDACK is asserted combinatorially or if the channel is in default acknowledge state.

The initiator module updates the ADDR and BE signals to reflect the address and byte enable information for the second cell of the packet being transferred. Note that the address(n-1:m) field does not change. (It is illegal to address more than one target within the same packet.) The signals: CMD, CONTIG, WRAP, CONST, and PLEN (also CLEN and CFIXED, which are not shown) are common to all the cells in the request packet.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

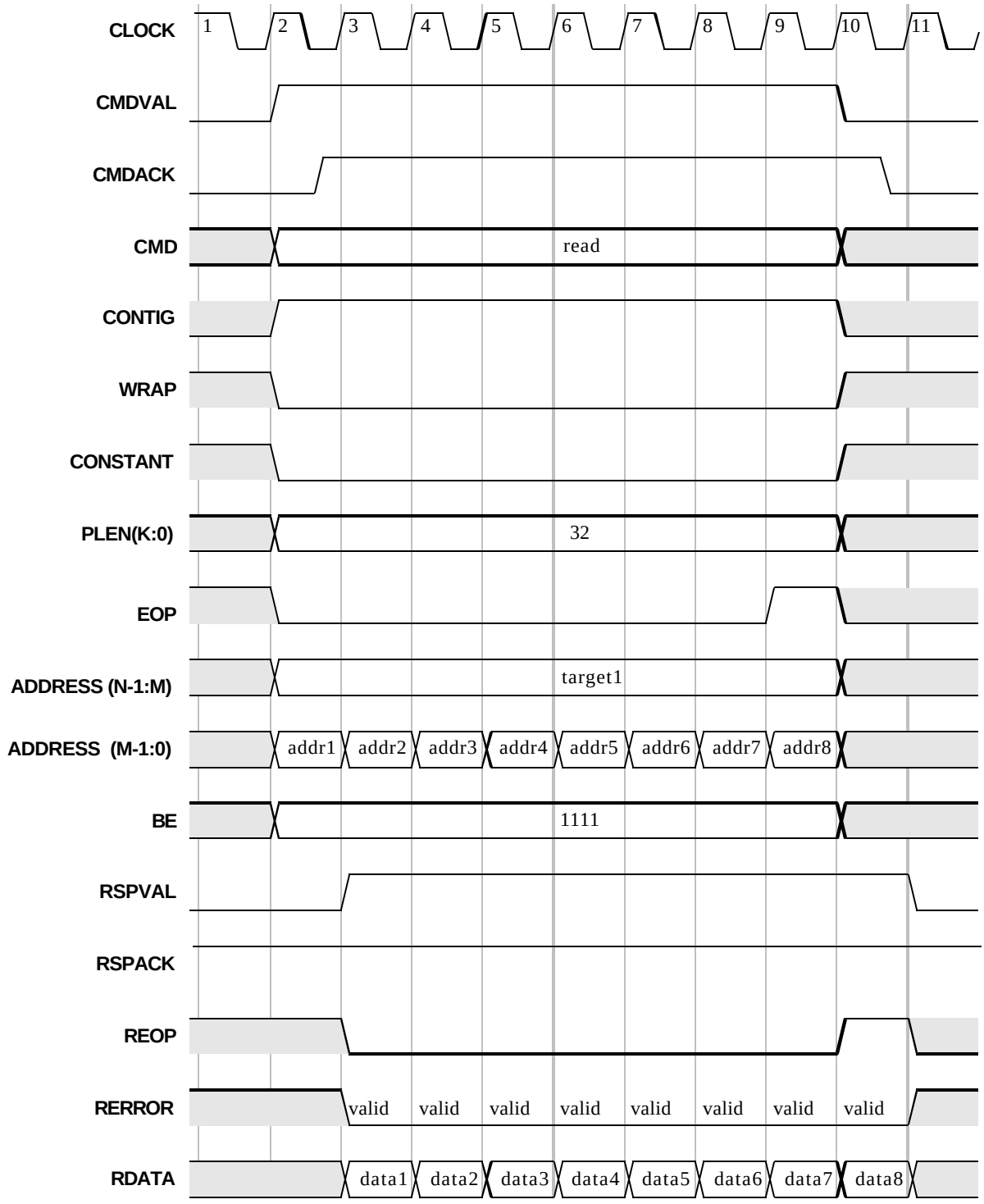


Figure 21: 32-byte Read Operation on a 32-bit VCI

All eight cells in the request packet are transferred in eight CLOCK cycles with CMDVAL and CMDACK asserted corresponding to VAL time = 0 and ACK time = 0. The CMDVAL for the last cell is asserted at CLOCK period 9, and the cell transfer completes on the same cycle. Note that the initiator asserted the signal EOP at CLOCK 9 to indicate that the cell being transferred is the last one in the packet.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

In the response channel, the initiator continuously asserts the signal, RSPACK, indicating its readiness to accept the response cell with ACK time = 0. This is a default acknowledge condition, as the CMDACK is asserted when CMDVAL is not present. All eight response cells are transferred in eight CLOCK cycles, as both VAL time and ACK time are 0. The first cell is transferred in CLOCK 3 when the Target module asserts RSPVAL. The target continues to assert RSPVAL until CLOCK 10, transferring one cell on every CLOCK edge. At CLOCK 10, the target also indicates to the initiator through REOP that the current cell is the last one in the response packet. Each response cell contains the read data (on RDATA) and the error flag (on RERROR).

4.4.6 Write Operation

Figure 22 illustrates a 32-byte write operation on a 32-bit VCI. A write operation is similar to a read operation except that:

- The initiator provides the write data on signal WDATA as part of each request cell.
- The field RDATA is not significant.

The operation starts with the initiator placing the first cell of a request packet on the interface and asserting CMDVAL on CLOCK 2. The cell information is encoded in the signals: CMD, CONTIG, WRAP, CONST, PLEN, EOP, ADDRESS, BE, and WDATA.

As shown in Figure 22:

- The signal, CMD, indicates that the packet is part of a write operation.
- The addressing mode flags, CONTIG, WRAP, and CONST, specify that the cell addresses are contiguous.
- BE indicates which byte lanes are valid in the write data.

The initiator module updates the ADDRESS, BE, and WDATA signals upon every cell transfer to reflect the byte address, enable information, and the data, until all four cells in the packet are transferred. Note that the address is updated for every cell, even though the opcode field indicates that the cell addresses are contiguous.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

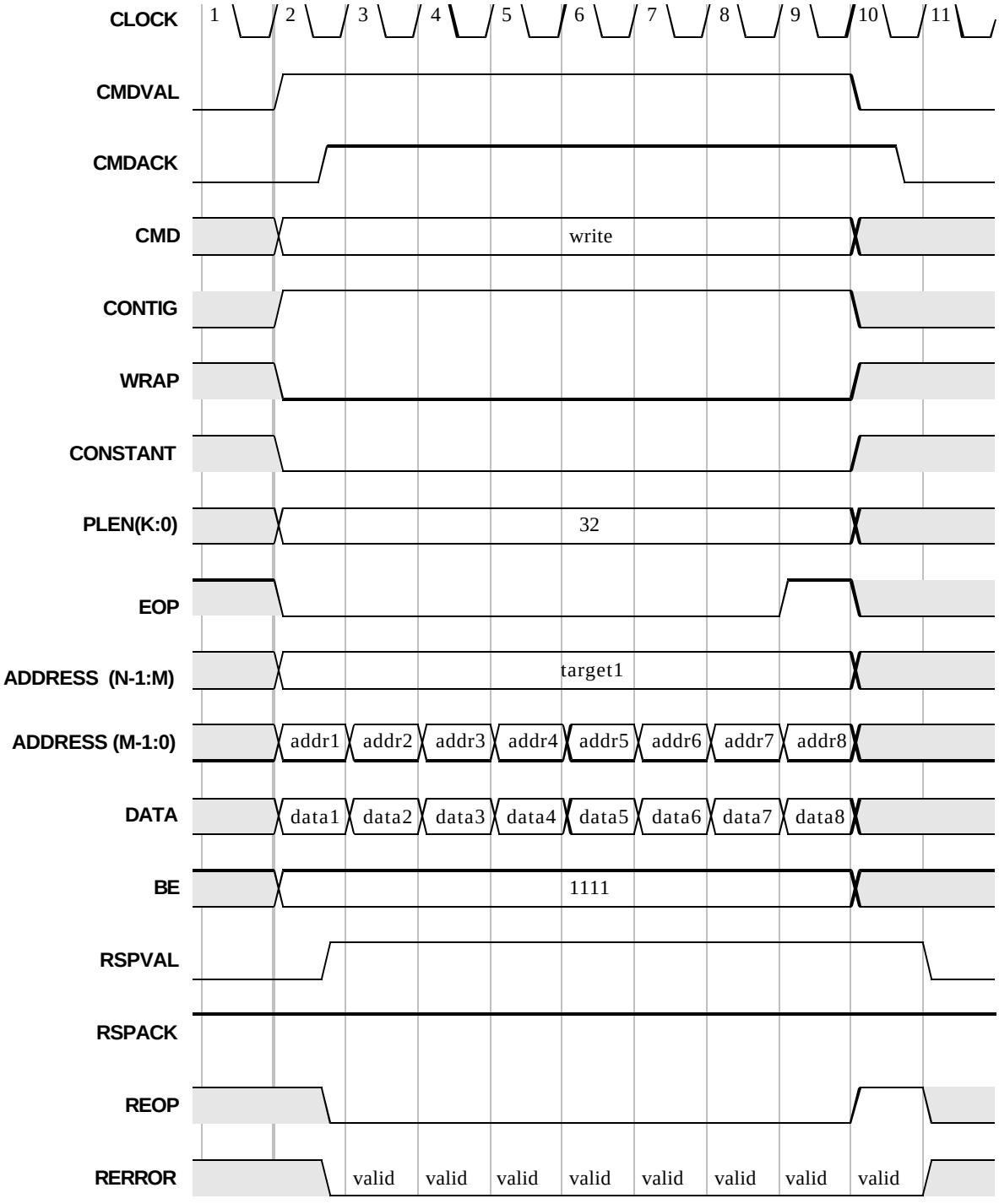


Figure 22: 32-byte Write Operation on a 32-bit VCI

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.
VSIA CONFIDENTIAL LICENSED DOCUMENT

All eight cells in the request packet are transferred in eight CLOCK cycles with CMDVAL and CMDACK asserted corresponding to VAL time = 0 and ACK time = 0. The response cells are also transferred in eight CLOCK cycles. The only information contained in write response cells is the error flag (on RERROR).

4.4.7 Other Operations

Two other operation types supported on VCI are NOP and Locked Read. The NOP operation is similar to a read or write operation, except that the CMD field of the request packet contains value “00” (NOP), and there is no data transferred along with request and response cells.

The locked read operation is similar to a read operation. The only difference is in the CMD signal encoding. The target module or system locks out the memory locations addressed by the locked read request until another operation is issued to the same locations by the same initiator. For more information on the exact behavior, see Section 4.4.4, “BVCI Operations.”

Packet Chain Transfer

Figure 23 illustrates a read transaction on the VCI composed of two operations grouped through the packet chaining mechanism. The operations are similar to the normal read operations except that the packet chaining information is provided along with the request packets. This information is encoded on signals CLEN and CFIXED.

CLEN specifies the number of packets remaining in the chain, excluding the current one. In this example it is 2-1 = 1. Note that this information is CONST within the packet and dynamic across packets. CLEN is specified as 0 for the second request packet, indicating that it is the last packet in the chain.

The flag, CFIXED, is asserted for the first packet to indicate that same opcode will be used for all the packets in the chain. CFIXED also guarantees that the address relationship between the last cell of a packet in the chain and the first cell of the following packet will be the same as the one defined by the ADDRESS mode flags for cells within packets. In this example, CFIXED = 1 guarantees that the cell addresses will be contiguous across the packet chain.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

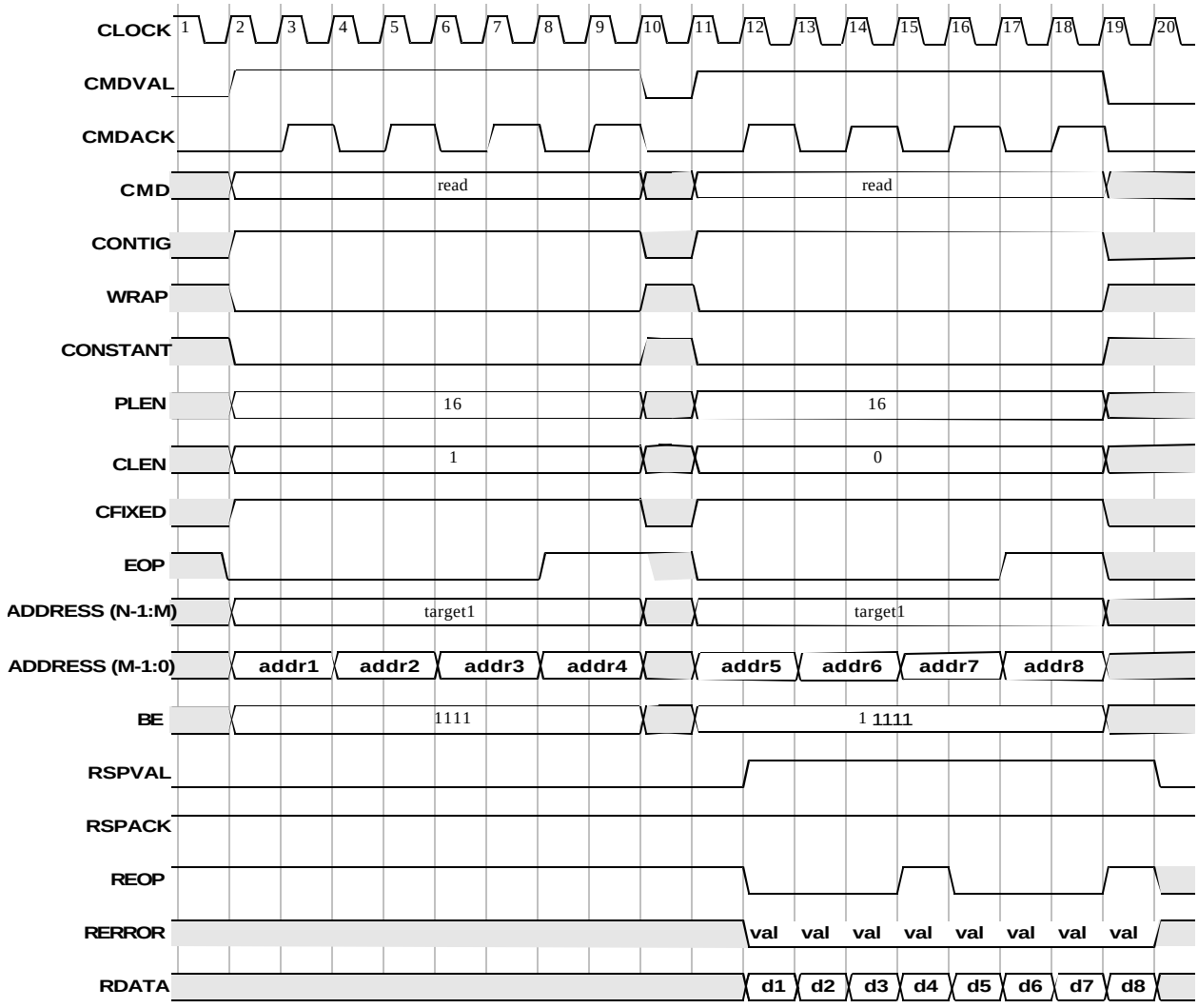


Figure 23: Packet Chain Transfer on the VCI

The chaining information on the first packet, along with the packet header information, conveys to the target module that the initiator module intends to read (CLEN + 1) x PLEN = (1 + 1) x 16 = 32 bytes of data from contiguous address locations starting from ADDR1. This information provides an option to the target module to combine the two request packets to improve the transfer efficiency, if possible. In this example, it is assumed that the target treats the packet chain as a single packet containing eight cells to perform a high efficiency data fetch. Note that the first response packet has a latency of nine CLOCK cycles, whereas the second response packet arrives four CLOCK cycles after the arrival of the corresponding request packet.

4.4.8 Address Modes

The VCI requires the initiator to provide cell addresses along with every request cell transferred to the target module. In addition, the initiator may also specify a predefined algorithm to calculate subsequent cell addresses from the previous cell address using the three addressing mode flags, CONTIG, WRAP, and CONST. When no flags are asserted by the initiator, there is no predefined relationship among cell addresses.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.4.8.1 Random Address Mode

When no opcode flags are asserted, the initiator specifies random cell addressing mode. In this case, there are no predetermined relationships among cell addresses. Figure 24 illustrates a 12-byte read operation with random address mode.

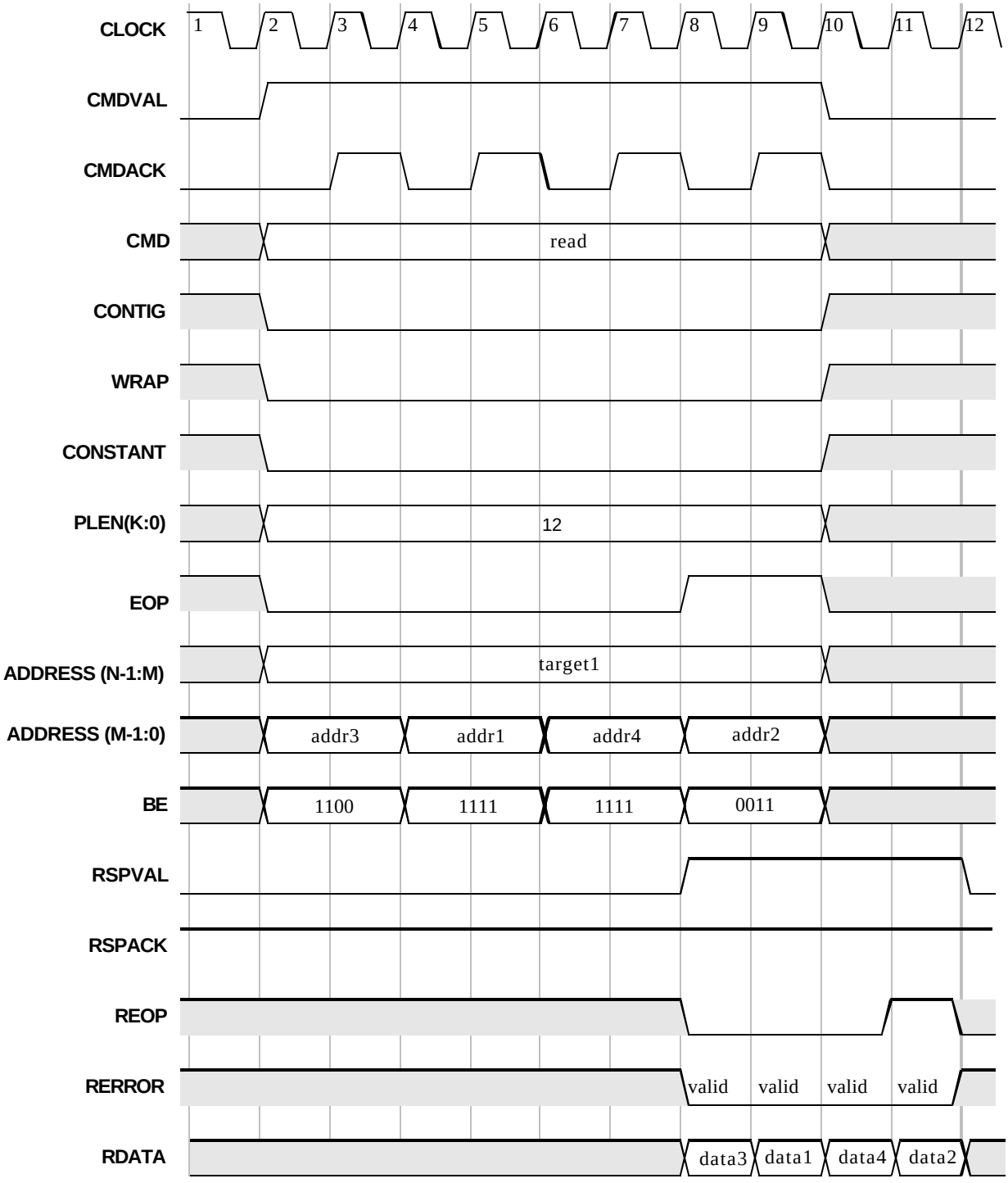


Figure 24: 12-byte Read Operation with Random Address Mode

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

4.4.8.2 Contiguous Address Mode

When CONTIG is asserted and WRAP is not asserted, the cell addresses advance in a contiguous manner. In this case, cell addresses can be calculated by adding the number of bytes transferred in the previous cell to the previous cell address. Figure 25 illustrates a 12-byte read operation with contiguous address mode.

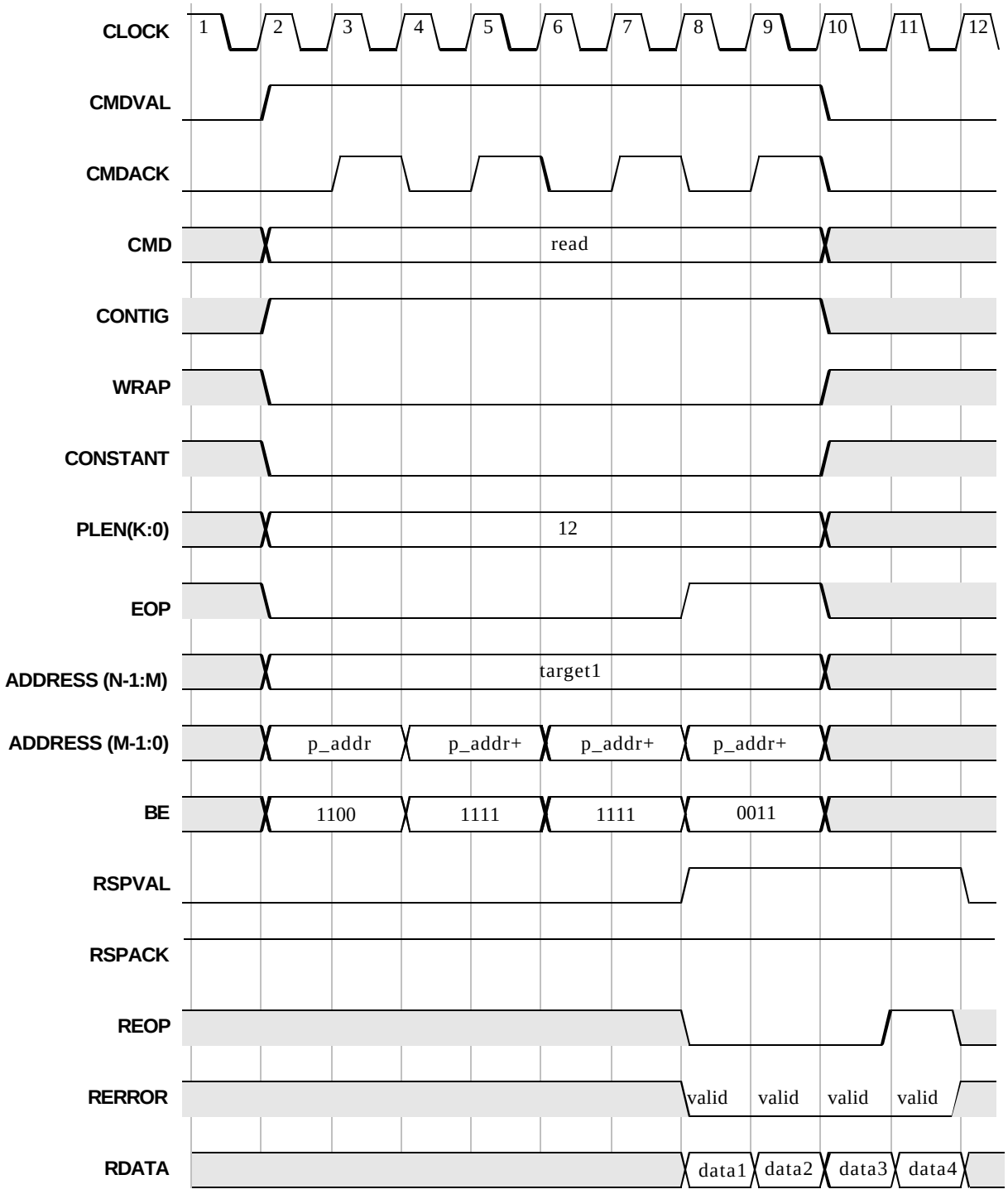


Figure 25: 12-byte Read Operation with Contiguous Address Mode

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

VSIA CONFIDENTIAL LICENSED DOCUMENT

Wrap Address Mode

This addressing mode is specified by assertion of the flags CONTIG, and WRAP. If the WRAP mode is specified and the field PLEN is specified as 2^A, then the following action takes place. The cell address advances in a contiguous manner and wraps around when the sum of the number of bytes transferred and the lower A+1 bits of the cell address are more than the value of PLEN. (The carry bit does not propagate beyond bit location A.) A 16-byte read operation in WRAP address mode is illustrated in Figure 26.

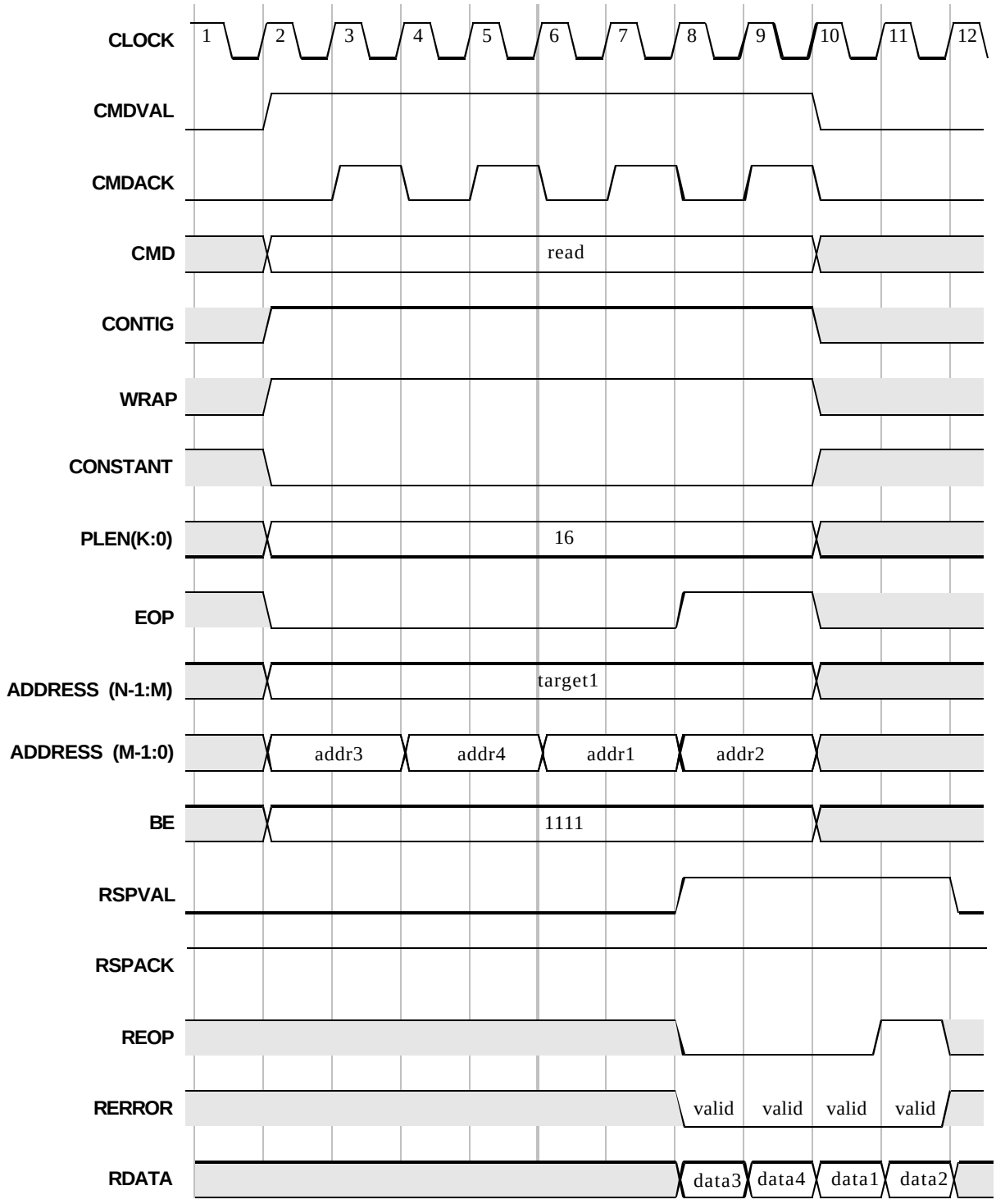


Figure 26: 16-byte Read Operation with Wrap Address Mode

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

Constant Address Mode

The constant address mode is specified using the flag CONST. When this mode is specified for a request packet, the cell address remains the same as the starting cell address across all the cells in the packet or packet chain. Figure 27 illustrates a 16-byte read operation in constant mode. Note that all the bits in the ADDRESS(n-1.0) remain constant for all the cells in the packet.

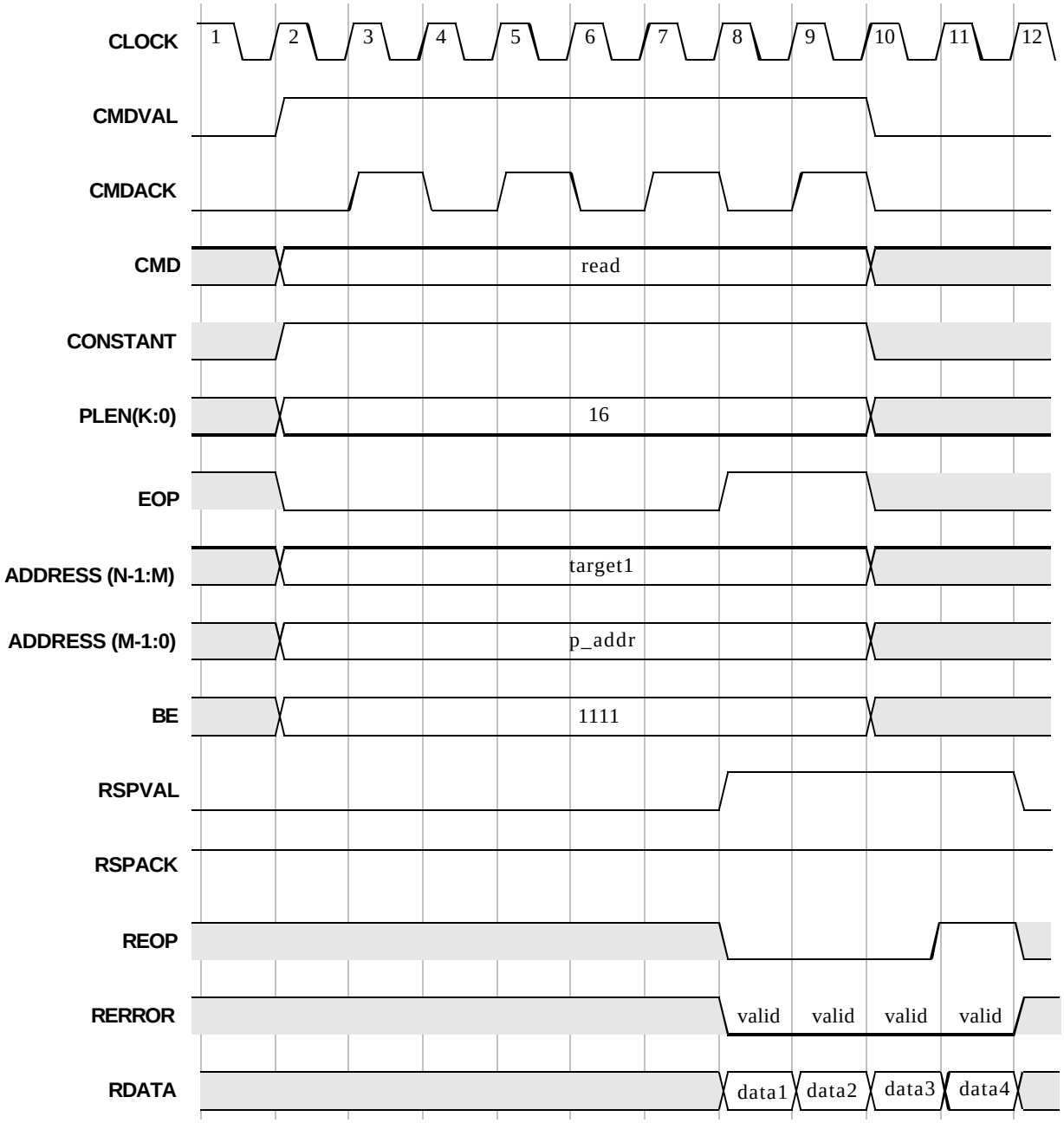


Figure 27: 16-byte Read Operation with Constant Address Mode

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

4.4.9 BVCI Signaling Rules

The following rules apply:

- For each packet transferred on the request channel, there should be exactly one response packet transferred back on the response channel.
- The order in which packets are transferred on the response channel should be same as the order in which the corresponding packets appear on the request channel.
- The number of cells in response packets should be same as in the corresponding request packets regardless of the opcode.
- The order of cells within a response packet should be same as the order in the corresponding request packet.
- Once CMDVAL is asserted, the initiator module cannot change that signal until the requested cell is transferred regardless of the state of CMDACK.
- Once a cell is placed on the interface and CMDVAL is asserted, the initiator module cannot modify the cell content until the requested cell transfer is complete.
- ADDRESS(n-1:m), the top n-m bits of the ADDRESS signal which indicates the target being addressed, should remain constant across all the cells in a packet.
- Opcode signals, CMD(1:0) and the ADDRESS mode flags should remain constant throughout the packet transfer.
- PLEN signal, which indicates the length of the request packet, should remain constant throughout the packet transfer.
- CLEN and CFIXED signals should remain constant across all the cells within a packet. It is recommended that the CLEN field should either stay constant or decrement by one with every packet transfer. It is legal for the initiator to change CLEN in unexpected ways between packets. This behavior is discouraged, as it adds complexity to the target. Initiators that exhibit this discouraged behavior must document it, and targets must document what CLEN behavior they can accept.
- Changing the CMD field between packets in a packet chain is strongly discouraged. Initiators that exhibit this discouraged behavior must document it.
- Asserting the flag WRAP without asserting CONTIG is invalid. Asserting WRAP when PLEN is not a power of 2 value is also invalid.
- The initiator shall not assert byte enable bits for those bytes that are outside the address range indicated by the address and PLEN fields.
- It is recommended that once the receiving module asserts CMDACK, the module cannot change that signal until the next clock cycle. This ensures that the transferred cell is not corrupted.

4.5 Additional Timing Diagrams

The following figures show additional examples of timing diagrams.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

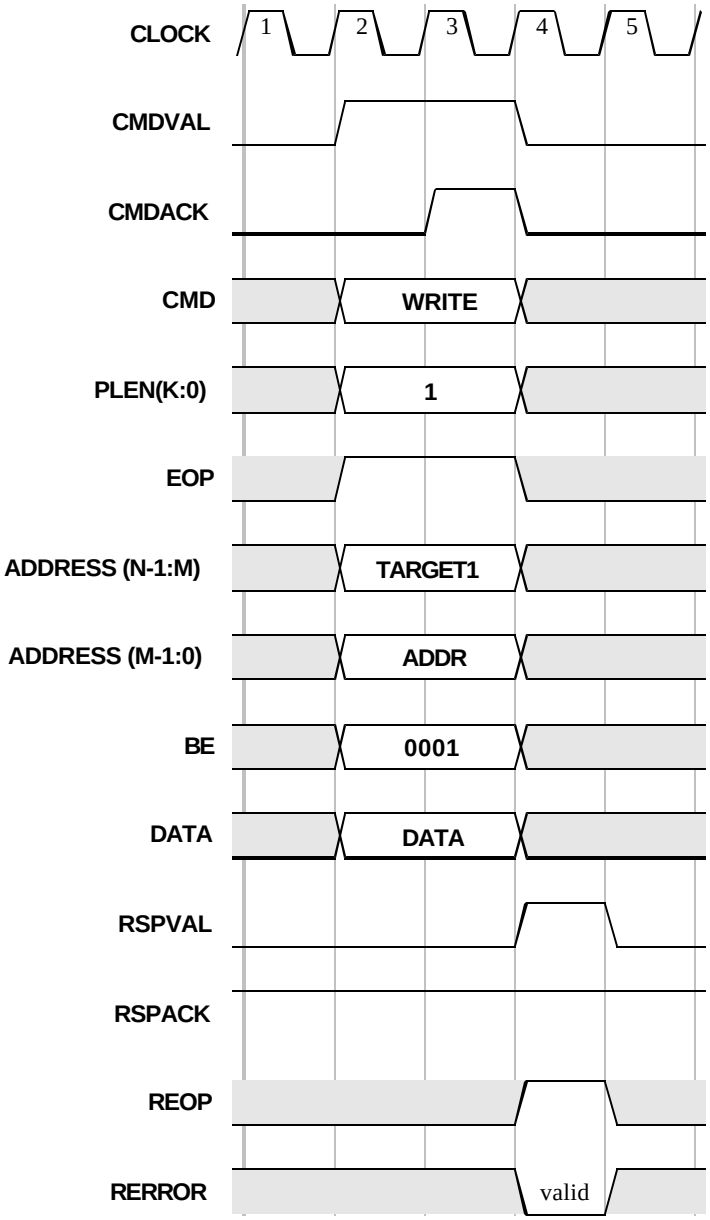


Figure 28: 1-byte Write Operation on a 32-bit VCI

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

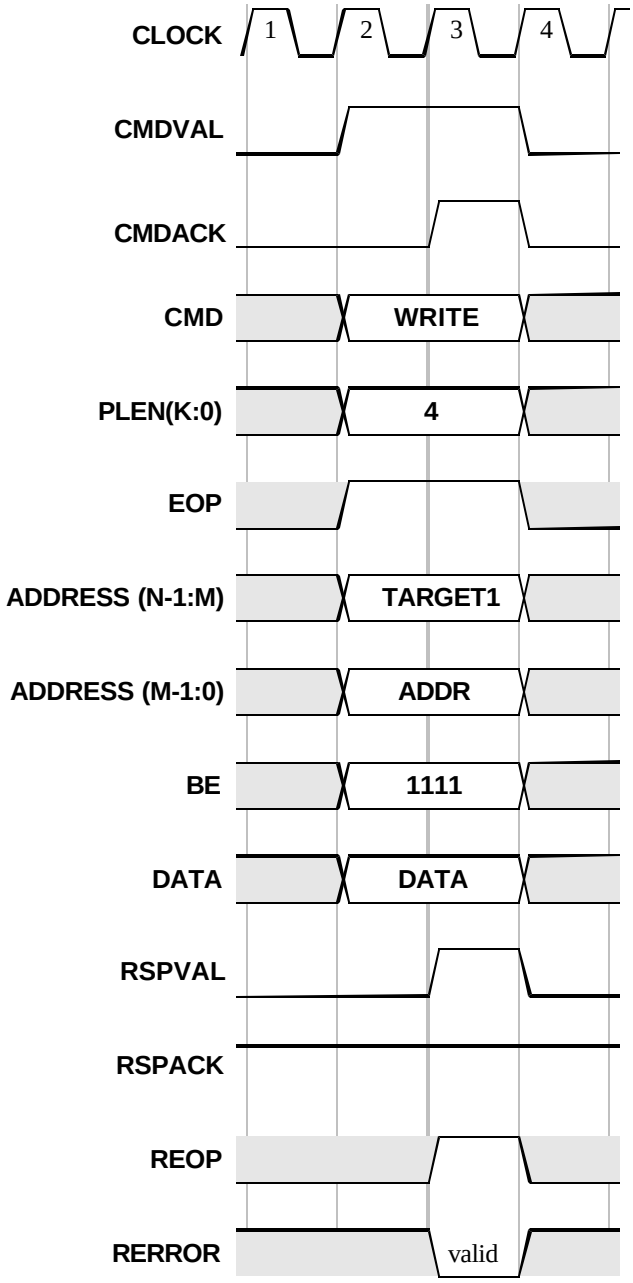


Figure 29: 4-byte Write Operation on a 32-bit VCI

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

5. Advanced VCI

This chapter defines the Advanced Virtual Component Interface (AVCI), a set of optional extensions to the Basic Virtual Component Interface (BVCI). The AVCI is to be used in conjunction with the on-chip system bus or for point-to-point connection between high performance VCs or bus(es). The AVCI is a superset of the BVCI.

The AVCI signals and protocols are optional features that can be added to the BVCI components to enhance performance in certain conditions. Thus it is mostly safe to ignore those signals in interconnections that do not support them, such as connecting an AVCI component to a BVCI component. Usually, only the system performance may suffer, not the functionality.

5.1 Organization

This chapter contains the following sections:

- Section 5.1: Describes the organization.
- Section 5.2: Gives a technical introduction to AVCI.
- Section 5.3: Provides a detailed description of the AVCI signals.
- Section 5.4: Defines the AVCI protocol in detail.
- Section 5.5: Shows additional timing diagrams.

5.2 Technical Introduction to the AVCI

Many system-on-chip (SoC) and multi-processor system-on-chip (MPSOC) applications require the system bus or interconnect to route multiple high bandwidth initiators, such as processor(s), DSP(s), DMA engine, and other high-bandwidth, real-time, special-application VCs. The interconnect of such systems often has several performance-enhancing features that are not supported by BVCI.

The AVCI was designed to close the feature gap described above. AVCI was defined as an optional extension of BVCI. As optional features, the extensions can be safely ignored without resulting in an error condition. A VC with an AVCI can easily be connected to a bus that supports BVCI, albeit with some performance degradation, due to loss of transfer efficiency.

The only major incompatibility between AVCI and BVCI is support for out-of-order transactions, and an advanced packet model. For this reason, the AVCI has parameters BVCIMODE and PUREPACKET indicating whether it is in BVCI compatibility mode or not. In BVCI compatibility mode, out-of-order transactions are not allowed, and addressing and other behavior matches BVCI specification.

The definitions for handshake protocol, cell, and packet layer are similar to the BVCI, introduced in Section 4.2, “Technical Introduction to the BVCI,” and not repeated here. The technical features that AVCI adds to the BVCI are introduced below.

5.2.1 Advanced Packet Model

The AVCI can make use of Advanced Packet Model, where request and response packets do not have the same size. An advanced read request packet consists of one cell, which sets the start address and address behavior signaling. The read response packet consists of as many cells as required for returning the read data. An advanced write request packet consists of as many cells as needed for write data, but the control information (address, command, and so forth) is only needed in the first cell. The packet model that an AVCI component supports is indicated with parameter PUREPACKET. This parameter may be either static or dynamic. In the case of a dynamic parameter, the default packet model is the BVCI model. The dynamic parameter can be implemented with a configuration pin or a configuration register.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5.2.2 Arbitration Hiding

Arbitration hiding signals allow pipelines of both the request and response packets through the system, allowing latency hiding of any arbitration delays. These signals, when presented to the interface, indicate the intent of the presenter (the initiator for Arbitration Command Valid and Arbitration Address, and the target for Arbitration Response Valid and Arbitration Source) to continue sending requests (initiator) or responses (target). The interconnect responds to these signals by presenting Arbitration Command Acknowledge (to an initiator) or Arbitration Response Acknowledge (to a target) on the interface. Upon detecting the acknowledge signals, the initiator or the target understands that it has the right to use the next cycle(s) to send a request or a response. These signals overlap the arbitration of the following packet with the transfer of the current packet. Note that these signals operate at packet-level rather than cell-level. The support for arbitration hiding is indicated with parameters. (For more information, see Chapter 6, “Design Guidelines.”)

5.2.3 Source Identification

The AVCI provides a system with a unique identifier for each initiator. This identifier can be used by the target,. for example, to detect which initiator has used the locking command, and to deny access from other initiators.

5.2.4 Multi-Threading and Out-of-Order Transactions

Packet Identifier, Thread Identifier, and Source Identifier signals can be used together to implement a multi-threaded system. The thread identifier is an extension to the source identifier, which can be used to create virtual initiators. An AVCI target may return response packets out of order with respect to request packets if the BVCIMODE parameter is not set. The packet identifier signal is required to indicate in which packet the response cells belong.

5.3 AVCI Signal Description

This section initiates a detailed AVCI technical description. Descriptions of signals used between the initiator and target over the VCI are provided in the following sections. Only the signals specific to the AVCI are included. The total AVCI signal set consists of BVCI signals and AVCI extensions.

5.3.1 Signal Type Definition

Table 11 specifies the signal types that are used in Section 5.3. The signal types are defined from the point of view of the devices, rather than the wrapper or arbiter.

Table 11: Signal Type Definition

Text	Description
IA	Input to all devices
IT	Generated by the initiator and sampled by the target
TI	Generated by the target and sampled by the initiator
MA	Mandatory signal for both initiator and target
MI	Mandatory signal for initiator but an optional signal for the target
MC	Mandatory signal for supporting chaining function for both initiator and target
MGA	Mandatory signal for general arbitration hiding support
MW	Mandatory signal for advanced wrapping support
MD	Mandatory signal for defined address mode support
MO	Mandatory signal for out-of-order transaction support

Signals that do not have one of the mandatory descriptors listed above are optional.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5.3.2 Signal Parameters

Table 12 specifies parameters that are used in Section 5.3.

Table 12: Signal Parameters

Parameter	Description
B	Number of bytes in a cell (must be a power of 2, maximum value is 32 bytes)
K	Number of bits in the PLEN field (maximum value is 9)
N	Number of bits in the ADDRESS field (maximum value is 64)
E	Number of bits in the RERROR extension field (maximum value is 3)
Q	Number of bits in the CLEN field (maximum value is 8)
F	Number of bits in the RFLAG field (maximum value is system dependent)
S	Number of bits in the SRCID and RSRCID fields (maximum value is 5)
P	Number of bits in the PKTID and RPKTID fields (maximum value is 8)
T	Number of bits in the TRDID and RTRDID (maximum value is system dependent)
W	Number of bits in the WRPLEN field (maximum value is 5)

5.3.3 Signal Directions

Figure 30 diagrams the signal directions between the initiator and the target.

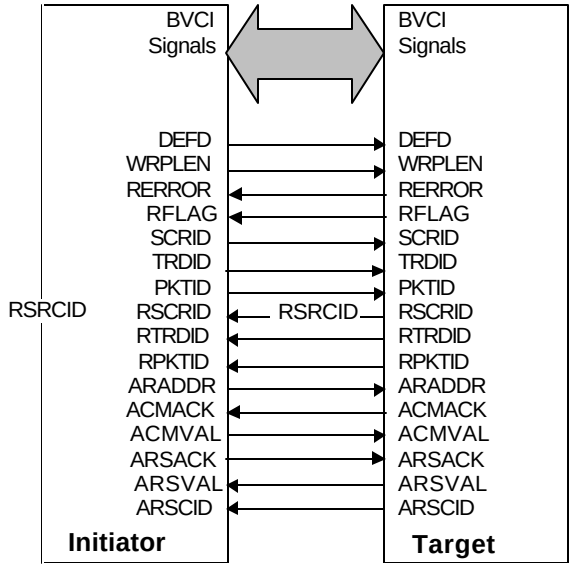


Figure 30: Diagram of VCI Signal Directions

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5.3.4 Signal List

All signals included in Table 13 are assumed to be active-high signals unless explicitly indicated otherwise. It is recommended that all signal outputs are stable before Early. It can be assumed that all inputs are stable before Late. A detailed signal description follows the summary table.

Table 13: Signal List

Name	Type	Description
Request Content		
DEFD Defined	IT MD	If "1," indicates the relationship among ADDRESS and BYTE ENABLE of each cell in the transaction follows a predefined pattern, understood by both initiator and target.
WRPLEN[W-1:0] Wraplen	IT MW	Address larger or equal to 2^WRAPLEN will be mapped to Address modulo (2^WRAPLEN). Maximum wrap length: 2^ (2^ W)
Response Content		
RERROR[E:0] Response Error	TI	Response Error is asserted by the target during a response packet to indicate that an error has occurred during the current packet. The RERROR consists of error indicator bit number 0, and an extension field. The extension field is wider in AVCI than in BVCI.
RFLAG[F-1:0] Response Flag	TI	Optional user-defined response code.
Request Threading Signals		
SRCID[S-1:0] Source Identifier	IT	Unique description for each initiator in the system.
TRDID[T-1:0] Thread Identifier	IT MO	Non-unique description of a thread.
PKTID[P-1:0] Packet Identifier	IT MO	Identifies packets in a transaction for out-of-order systems. Together with SCRID identifies virtual or logical identifiers.
Response Threading Signals		
RSRCID[S-1:0] Response Source Identifier	TI	Copy of SRCID that is returned with a response to a request.
RTRDID[T-1:0] Response Thread Identifier	TI MO	Copy of TRDID that is returned with a response to a request.
RPKTID[P-1:0] Response Packet Identifier	TI MO	Copy of PKTID that is returned with a response packet.
Arbitration Hiding Signals		
ARADDR[N-1:0] Arbitration Address	IT MGA	Next/early version of ADDRESS for general arbitration hiding mode. The source is VC initiator. The target is the arbiter or the initiator wrapper, which connects to the underlying bus system arbiter. This signal is intended to the interconnect, not to the VC target. The signal exists only in VC targets that are initiator wrappers.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 13: Signal List (Continued)

Name	Type	Description
ACMVAL Arbitration Command Valid	IT MGA	Arbitration hiding request valid. Valid indicates that the initiator wishes to indicate the next packet's start address to the interconnect. The source is the VC initiator. The target is the arbiter or the initiator wrapper, which connects to underlying bus system arbiter. This signal is intended for the interconnect, not the VC target. The signal exists only in VC targets that are initiator wrappers.
ACMACK Arbitration Command Acknowledge	TI MGA	Arbitration hiding request-acknowledge. Acknowledge is used by the interconnect to indicate to the initiator that a given arbitration address can be transferred. Hence, ARADDR is transferred from the initiator when both ACMVAL and ACMACK are asserted at the rising edge of the clock. The source is the arbiter or the initiator wrapper, which connects to underlying bus system arbiter. The target is the VC initiator. This signal comes from the interconnect, not the VC target. The signal exists only in VC targets that are initiator wrappers.
ARSVAL Arbitration Response Valid	TI	Arbitration hiding response valid. Similar to the ACMVAL, but for the response handshake. The source is the VC target, which connects to the underlying bus system arbiter. The target is an arbiter or a target wrapper. This is intended for the interconnect, not the VC initiator. The signal exists only in VC initiators that are target wrappers.
ARSACK Arbitration Response Acknowledge	IT	Arbitration hiding response-acknowledge. Similar to the ACMACK, but for the response handshake. The source is the VCI target, which connects to the underlying bus system arbiter. The target is an arbiter or a target wrapper. This is intended for the interconnect, not the VC initiator. The signal exists only in VC initiators that are target wrappers.
ARSCID[S-1:0] Arbitration Source Identifier	TI	Next/early version of RSRCID given with ARSVAL.

5.3.5 Request Signals

5.3.5.1 Defined

Signal Name:	Defined
Signal Abbreviation	DEFD
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Same as CMDVAL (See Chapter 4, "Basic VCI.")

The BVCI supports a 2-bit cmd field, and three individual flags (CONTIG, WRAP, and CONST). AVCI adds one more flag called DEFINED. The definition of this flag is: while asserted, the relationship among address and byte enable of each cell in the transaction follows a predefined pattern, understood by both initiator and target.

The "defined" flag can be used to move a specific data structure. It assumes that its target is fully aware of the structure and capable of sending the appropriate data when the starting ADDRESS and BE have been presented with CMD and CMDVAL. The "defined" flag is intended to support an algorithmic address increment (such as striding or double striding), that is commonly found in 2D and 3D graphic applications and in imaging and scientific computing. Since there is only one bit for the defined field for each initiator-target pair, there is exactly one predefined operation supported. In a case where there are more supported predefined operations, virtual target and initiator can be defined with a thread identifier to support more predefined operations.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5.3.5.2 Wrap Length

Signal Name:	Wraplen
Signal Abbreviation:	WRPLEN[W-1:0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Same as CMDVAL (See Chapter 4, “Basic VCI.”)

BVCI uses *plen* as the wrap boundary when WRAP flag is active. AVCI adds WRAPLEN as a new parameter, which indicates that the wrap operation should use the WRAPLEN wrap boundary instead of PLEN. Enabling this option, wrap operation is not limited anymore to the size of the packet.

- Address larger or equal to 2 ^ WRAPLEN will be mapped to (Address modulo (2 ^ WRAPLEN)).
- Maximum wrap length = 2^(2 ^ W)

One of the applications of WRAPLEN is to prevent a packet from crossing a device boundary. In this case, WRAPLEN is set to have the value of the size of a device address range. It can also be used for optimizing cache line fill of a CPU.

5.3.5.3 Source Identifier

Signal Name:	SourceID
Signal Abbreviation:	SRCID[S-1:0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Same as CMDVAL (See Chapter 4, “Basic VCI.”)

Each initiator in a system is assigned a unique identifier, called SRCID. This identifier can be used by the target to detect which initiator is using lock.

5.3.5.4 Thread Identifier

Signal Name:	ThreadID
Signal Abbreviation:	TRDID[T-1:0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Same as CMDVAL (See Chapter 4, “Basic VCI.”)

A device (an initiator and/or a target) may behave as more than one logical device. This kind of device is considered to have multiple virtual devices. TRDID can be used as an extension to the SRCID to create logical, or virtual devices. In a system where the number space of the SRCID runs out, the space can be also extended with TRDID.

5.3.5.5 Packet Identifier

Signal Name:	PacketID
Signal Abbreviation:	PKTID[P-1:0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Same as CMDVAL (See Chapter 4, “Basic VCI.”)

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

In a system where out-of-order transfer completion is supported, PKTID and RPKTID are required to identify which response packet matches which request packet. If there is confusion with packets coming from different initiators, SRCID can be used to make the packets unique.

5.3.6 Response Signals

5.3.6.1 Response Error

Signal Name:	ResponseError
Signal Abbreviation:	RERROR[E:0]
Polarity:	N/A
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Same as RSPVAL (See Chapter 4, “Basic VCI.”)

The bit 0 of the RERROR is the bit 0 of the BVCI RERROR. The extension bits are encoded differently. The error signal is valid only when RSPVAL=1, with the following meaning:
bit [3:0]
RERROR = 0001: No info
RERROR = 1001: Transaction supported and serviced, but not completed
RERROR = *101: Transaction supported, but not serviced, retry later
RERROR = 0011: Transaction not supported, but not serviced
RERROR = 1011: Transaction not supported, degraded into basic read or write, or other, indicated by RFLAG (without RFLAG, it is degraded to basic read or write)
RERROR = *111: Fatal error, target disconnect (itself)

5.3.6.2 Response Flag

Signal Name:	ResponseFlag
Signal Abbreviation:	RFLAG[F-1:0]
Polarity:	N/A
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Same as RSPVAL (See Chapter 4, “Basic VCI.”)

RFLAG is defined for supporting optional user-defined response code. A specific response status can be returned to the initiator directly using one of these flags. In the system where there are two different interpretations of user defined flags, they will be mapped onto different bits in the RFLAG. This mapping is system specific. For example, RFLAG[1:0] = (RFLAG1, RFLAG0).

5.3.6.3 Response Source Identifier

Signal Name:	ResponseSourceID
Signal Abbreviation:	RSRCID[S-1:0]
Polarity:	N/A
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Same as RSPVAL (See Chapter 4, “Basic VCI.”)

This is a copy of SRCID that is returned along with the response. It helps identify the destination of the response.

5.3.6.4 Response Thread Identifier

Signal Name:	ResponseThreadID
Signal Abbreviation:	RTRDID[T-1:0]
Polarity:	N/A

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Driven By:	VCI target
Received By:	VCI initiator
Timing:	Same as RSPVAL (See Chapter 4, “Basic VCI.”)

This is a copy of TRDID that is returned along with the response. It helps identify the thread of destination of the response if the target is multi-threaded.

5.3.6.5 Response Packet Identifier

Signal Name:	ResponsePacketID
Signal Abbreviation:	RPKTID[P-1:0]
Polarity:	N/A
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Same as RSPVAL (See Chapter 4, “Basic VCI.”)

This is a copy of PKTID that is returned along with the response. It helps identify the originating request, and reorders the packet if the system supports out-of-order transactions.

5.3.7 Side Band Signals

These signals are not for VCI initiator-target communication, but for initiator-bus arbiter and target-bus arbiter signaling. Thus the target for ARADDR, ACMVAL, and ARSACK is not a VCI target device (such as a peripheral), but the VCI target acting as a wrapper, which connects the VCI initiator to the bus interconnect. Similarly, the source for ACMACK, ARSACK, and ARSCID is not the VCI initiator device (such as a processor), but the VCI initiator acting as a target wrapper, which connects VCI target to the bus interconnect.

5.3.7.1 Arbitration Address

Signal Name:	ArbitrationAddress
Signal Abbreviation:	ARADDR[N-1:0]
Polarity:	N/A
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Same as ACMVAL (See Chapter 4, “Basic VCI.”)

Provides an early version of the address of the first cell of the next packet for arbiter. In some cases this may contain only the higher address bits used for device selection.

5.3.7.2 Arbitration Command Valid

Signal Name:	Arbitration Command Valid
Signal Abbreviation:	ACMVAL
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK, until ACMACK == 1 and next rising edge of CLOCK

This signal is activated at exactly two cycles before the first CMDVAL of the packet to which the arbitration hiding applies. CMDVAL must be issued at latest one cycle after the initiator samples ACMACK at the rising clock edge. Arbitration hiding is done only once in a packet. The signal validates the AADDR in arbitration hiding mode.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5.3.7.3 Arbitration Command Acknowledge

Signal Name:	ArbitrationCommandAcknowledge
Signal Abbreviation:	ACMACK
Polarity:	Asserted Positive
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted at rising edge of CLOCK, when ACMVAL == 1 until next rising edge of CLOCK.

Arbitration hiding command acknowledge. Acknowledge is used by the interconnect to indicate to the initiator that a given arbitration address can be transferred. Hence, AADDR is transferred from the initiator when both ACMVAL and ACMACK are asserted at rising clock edge.

5.3.7.4 Arbitration Response Valid

Signal Name:	ArbitrationResponseValid
Signal Abbreviation:	ARSVAL
Polarity:	Asserted Positive
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Asserted at rising edge of CLOCK, until ARSACK == 1 and rising edge of CLOCK.

Arbitration hiding response valid. VAL-handshake signal for arbitration response. This signal is activated exactly two cycles before the first RSPVAL of the response packet the arbitration hiding applies to. RSPVAL must be issued at latest on the cycle after the target samples ARSACK at the rising clock edge.

5.3.7.5 Arbitration Response Acknowledge

Signal Name:	ArbitrationResponseAcknowledge
Signal Abbreviation:	ARSACK
Polarity:	Asserted Positive
Driven By:	VCI initiator
Received By:	VCI target
Timing:	Asserted at rising edge of CLOCK, when ARSVAL == 1 until next rising edge of CLOCK.

Arbitration hiding response acknowledge. ARCID is transferred from the target when both ARSVAL and ARSACK are asserted at rising clock edge. This signal can be generated synchronously or asynchronously of the ARSVAL.

5.3.7.6 Arbitration Source Identifier

Signal Name:	ArbitrationSourceID
Signal Abbreviation:	ARSCID[S-1:0]
Polarity:	N/A
Driven By:	VCI target
Received By:	VCI initiator
Timing:	Same as ACMVAL.

The next or early version of RSRCID is returned with ARSVAL instead of RSPVAL.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5.4 AVCI Protocol

The AVCI protocol is described similarly to the BVCI with three layers: the transaction layer, the packet layer, and the cell layer. There are no differences between the protocols in the transaction layer, but the packet and cell layers differ slightly. See Section 4.4, “BVCI Protocol,” for an introduction to BVCI protocols.

5.4.1 AVCI Packet Layer

The AVCI packet layer, and thus packet, operation, and packet chain, are the same as in the BVCI, with AVCI having a new, optional, advanced packet model. In the advanced packet model, the request and response packets do not have the same number of cells, as shown in Figure 31. Since most of the difference is in the cell-level behavior of the packet, the advanced packet model is explained in Section 5.4.3, “AVCI Operations.” The packet model that a VC uses is indicated with a parameter. See Appendix A for message sequence charts of packet communication in PVCI, BVCI, and AVCI.

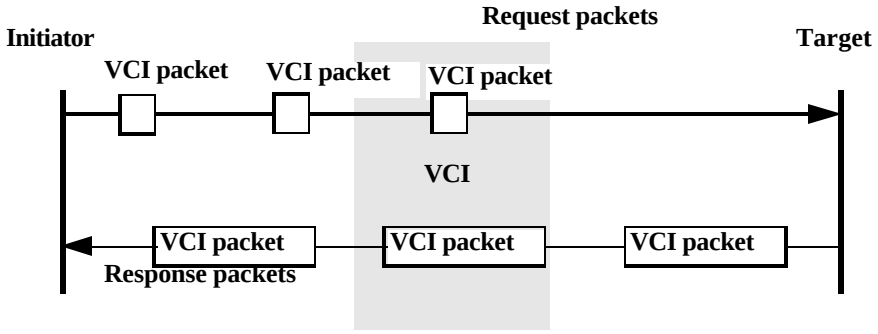


Figure 31: Advanced Packet Model

5.4.2 Cell Layer

The AVCI cell layer differs from the BVCI with a couple of additional fields, and with side band signals for arbitration hiding. The arbitration hiding signals are not part of cells; they are separately handshaken. The AVCI cell handshake is similar to the BVCI handshake.

The request cell structure and the response cell structure are detailed in Table 14 and Table 15. AVCI signals are shown on a grey background.

Table 14: Request Cell Structure

Comment	Field	Description
Packet chain Information	CLEN	Number of remaining packets in the chain
	CFIXED	Indicates if opcode and PLEN are same across all packets in the chain
	ADDRESS	Address of each cell being transferred
	PLEN	Packet length: Total number of data bytes transferred in the operation
Optional fields	WRPLEN	Wrap length: Address wrapping boundary
	SRCID	Source identifier
	PKTID	Packet identifier
	TRDID	Thread identifier

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 14: Request Cell Structure (Continued)

Comment		Field	Description
Opcode	Flags	CMD	Command: read, write, nop, or locked_read
		CONTIG	Contiguous address mode
		WRAP	Wrap address mode
		CONST	Constant address mode
		DEFD	Defined address mode
		BE	Byte Enable: indicates which bytes are involved in the operation
Optional field		WDATA	Data in write requests
		EOP	Indicates if the cell is the last one in the request packet

Table 15: Response Cell Structure

Comment	Field	Description
Optional fields	RERROR	Error status for the cell
	RFLAG	User specific response flag
	RSRCID	Returned source identifier
	RPKTID	Returned packet identifier
	RTRTID	Returned thread identifier
	RDATA	Data in read and locked_read responses
	REOP	Indicates if the cell is the last one in the response packet

5.4.2.1 Extended Error Reporting

Currently RERROR is defined as a 4-bit field, with bit 0 identical to bit 0 of RERROR of the BVCI. The optional bits 1 to 3 provide additional information of the transaction's status. Thus, it does not break the system should it be ignored.

Error condition is associated with a packet. However, error reporting should occur along with every response cell, to indicate errors as early as possible. Error code provides additional status information of the transfer, and the state of the target (in case of "abort").

The transaction may be degraded to generic read or write instruction, which is indicated by the RERROR signal. The target may also degrade the transaction into a transaction other than generic read or write operation. In this case, RFLAG can be used to accompany RERROR to indicate the operation performed.

5.4.2.2 Response Flag for Optional Status Reporting

RFLAG provides the user with a way to return status information that cannot be encoded in RERROR. The encoding is system specific.

5.4.2.3 Arbitration Hiding Fields

Table 16: Arbitration Hiding Signals

Comment	Field	Description
Optional fields	**ARADDR**	Arbitration address
	ARSCID	Returned source identifier

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5.4.2.4 Arbitration Hiding Handshake

The arbitration hiding is indicated with handshake signals. The arbitration hiding handshake protocol is similar to the BVCI and AVCI cell handshake. The acknowledge signals can be synchronous or asynchronous, or tied permanently high. Figure 32 and Figure 33 show the handshake protocols. Dashed lines indicate active clock edges.

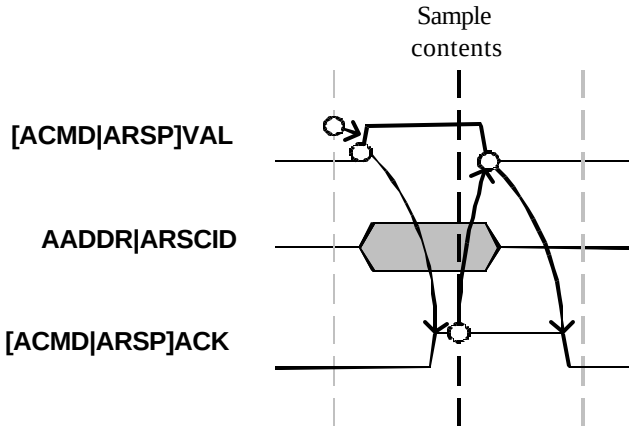


Figure 32: Arbitration Hiding Handshake (Asynchronous)

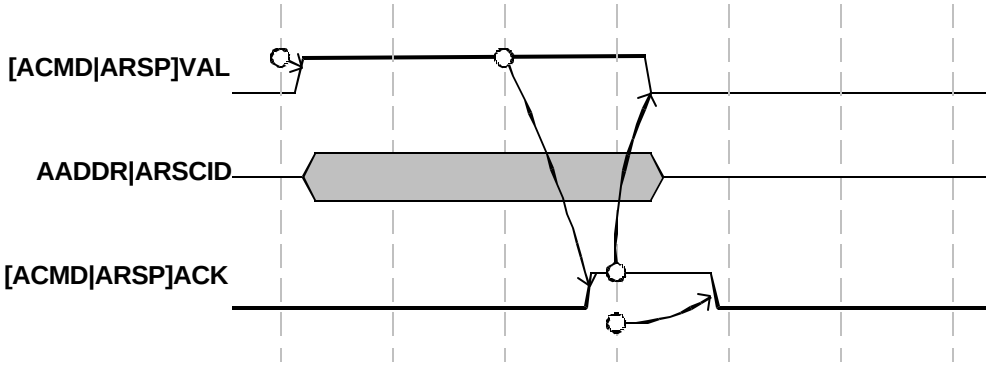


Figure 33: Arbitration Handshake (Synchronous)

5.4.3 AVCI Operations

5.4.3.1 Transfer Requests

The following is a complete list of the default transaction types that may be initiated across the AVCI. See Section 6.2, “VCI Parameters,” for the meaning of the parameters indicated in bold text.

Standard Transactions (**BVCIMODE = True**, **PUREPACKET = False**):

- *Read a cell* with any or all bytes active.
- *Read and lock a cell* with any or all bytes active.
- *Write a cell* with any or all bytes active (and release lock).
- *Read a packet of plen bytes from random addresses*, with any combination of bytes enabled in each cell.
- *Read a packet of plen bytes from random addresses*, with any combination of bytes enabled in each cell, and lock the first accessed cell address.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

- *Write a packet of plen bytes to random addresses*, with any combination of bytes enabled in each cell, and release lock of the first accessed cell address.
- *Read a packet of plen bytes from contiguous addresses*, with any combination of bytes enabled in each cell.
- *Read a packet of plen bytes from contiguous addresses*, with any combination of bytes enabled in each cell, and lock the accessed addresses.
- *Write a packet of plen bytes to contiguous addresses*, with any combination of bytes enabled in each cell, and release locks.
- *Read a packet of plen bytes from contiguous addresses*, wrapping the address at boundary indicated by PLEN, with any combination of bytes enabled in each cell. Packet length must be a power of 2.
- *Read a packet of plen bytes from contiguous addresses*, wrapping the address at boundary indicated by PLEN, with any combination of bytes enabled in each cell, and lock the accessed addresses. Packet length must be a a power of 2.
- *Write a packet of plen bytes to contiguous addresses*, wrapping the address at boundary indicated by PLEN, with any combination of bytes enabled in each cell, and release locks. Packet length must be a power of 2.
- *Read a packet of plen bytes from one address*, with any combination of bytes enabled in each cell.
- *Read a packet of plen bytes from one address*, with any combination of bytes enabled in each cell, and lock the address.
- *Write a packet of plen bytes to one address*, with any combination of bytes enabled in each cell, and release lock.

When Defined Address mode is supported (**DEFINED = True**):

- *Read a packet of plen bytes from addresses known* to both initiator and target, with any combination of bytes enabled in each cell.
- *Read a packet of plen bytes from addresses known* to both initiator and target, with any combination of bytes enabled in each cell, and lock the accessed addresses.
- *Write a packet of plen bytes to addresses known* to both initiator and target, with any combination of bytes enabled in each cell, and release locks.

When Advanced Wrapping is supported (**WRPLENSIZE > 0**):

- *Read a packet of plen bytes from contiguous addresses*, wrapping the address at boundary indicated by WRPLEN, with any combination of bytes enabled in each cell.
- *Read a packet of plen bytes from contiguous addresses*, wrapping the address at boundary indicated by WRPLEN, with any combination of bytes enabled in each cell, and lock the accessed addresses.
- *Write a packet of plen bytes to contiguous addresses*, wrapping the address at boundary indicated by WRPLEN, with any combination of bytes enabled in each cell, and release locks.

When Packet Chaining is supported (**CHAINING = True**):

- *Issue a chain of packets* (chain any of the above transactions), so that all the other packet header fields are identical over the chain except for the address and chain length counter (chain fixed). Responses must be in order.
- *Issue a chain of packets* (chain any of the above transactions), so that all packets may have different header fields (chain not fixed). Responses can be out of order.

With Advanced Packet mode (**PUREPACKET = True**):

- Read transaction requires only the first cell of the request packet.

5.4.3.2 Transfer Responses

The following is a list of the responses that are allowable by a target across the AVCI:

- *Read of cell/packet successful*, return read data
- *Write of cell/packet successful*
- *Read packet general error*, the entire packet bad, return data invalid
- *Write packet general error*, the entire packet bad

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

- *Read bad data error*, suggestion to retry transfer of packet or cell, return data invalid
- *Write bad data error*, suggestion to retry transfer of packet or cell
- *Read/Write Abort* disconnect, suggestion to not retry the transfer

When the Response Flag is supported (RFLAGSIZE > 0):

- All the previous responses apply, together with *user defined response code*.
- When Out-of-Order Transactions are supported, (PKTIDSIZE > 0, BVCIMODE = False)

All the previous responses apply, but response packets are allowed to come in a different order than the request packets. Each packet has a set of identifiers by which they can be identified.

5.4.3.3 Arbitration Hiding Note

The arbitration hiding signals are not part of the transaction, but they give additional information that helps build the system interconnect.

5.4.4 Incrementing Address (BVCI Packet Model)

This packet model requires that addresses be sent with each cell requested. The entire packet has as many cells as addresses. The following are properties of an incrementing packet model:

- A request cycle is as long as the number of cells. A request of 16 bytes on a 4-byte interface is asserted for four (16/4) cycles (not necessarily consecutive).
- The address changes every request cycle. A request of 16 bytes on a 4-byte interface --> A0, A1, A2, A3.
- The response is expected to be as long as the request. A request of 16 bytes on a 4-byte interface --> R0, R1, R2, R3. Thus, respond and request are balanced. Note that R0, R1, R2, and R3 can be non-consecutive, interface-cycle-wise.
- The system is very error tolerant. The way the cmd (command) field is defined, it will degrade gracefully, since the only real command (that can cause error) is the first 2 bits in the cmd field. The remaining 4 bits are performance enhancement bits. Since valid address (along with byte enable) is present for every cell (cycle) requested, ignoring the last 4 bits will not produce any error. In this model all of the unsupported commands will degrade gracefully (completed correctly even if slower).

5.4.5 Non-Incrementing Address (Advanced Packet Model)

This packet model only requires a single address, and a request for the entire packet. The following are properties of a non-incrementing packet model:

- A read request packet has always one cell. A request of 16 bytes on a 4-byte interface is asserted for 1 cell.
- Only the first address is sent. A request of 16 bytes on 4 bytes interface --> A0.
- The response is expected to be as long as response for the BVCI model above. A request of 16 bytes on a 4-byte interface --> R0, R1, R2, R3. Thus, response and request are not balanced. Note that R0, R1, R2, and R3 can be non-consecutive , interface-cycle-wise.
- Shorter request time means that the initiator can start another request sooner. This can potentially lead to higher performance (especially in the case of a burst read followed by a burst write).
- Potentially better clustered transactions lead to better, less fragmented utility of the bandwidth.
- It may indicate unnecessary errors.

5.4.5.1 Comparison between Incrementing and Non-Incrementing Packet Models

The following are the similarities and differences of the packet models described above:

- A non-incrementing packet model with contig off is similar to an incrementing packet model.
- The bandwidth of the bus remains the same for burst write. In case of burst read followed by burst write, the non-incrementing packet model has a better bandwidth. Overall, non-incrementing models will have a little better performance (for latency and bandwidth).

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Figure 34 shows the advanced write of two packets. The transfer is not faster than a normal packet transfer. Notice that the constant, contiguous, wrapped, and defined address modes could each be used. In contiguous address mode, the byte enable of the first cell is indicated by the BE signal. The byte enable of the last cell is calculated of the PLEN and the first BE. For example, BE1 = "0011," PLEN = 7 => last BE becomes "1000." The byte enables from the cells in the middle of the packet are all ones ("1111"). The byte enable of the first cell must be contiguously high, starting from the byte corresponding to the lowest active byte address. For example, "0001," "0011," "0111," and "1111" (the leftmost bit is the lowest byte address) are legal first-cell byte enables for a 32-bit interface. In constant address mode, the byte enable of the first cell is replicated over the packet. In the defined address mode, the byte enable depends on the agreement between the initiator and the target for the address pattern.

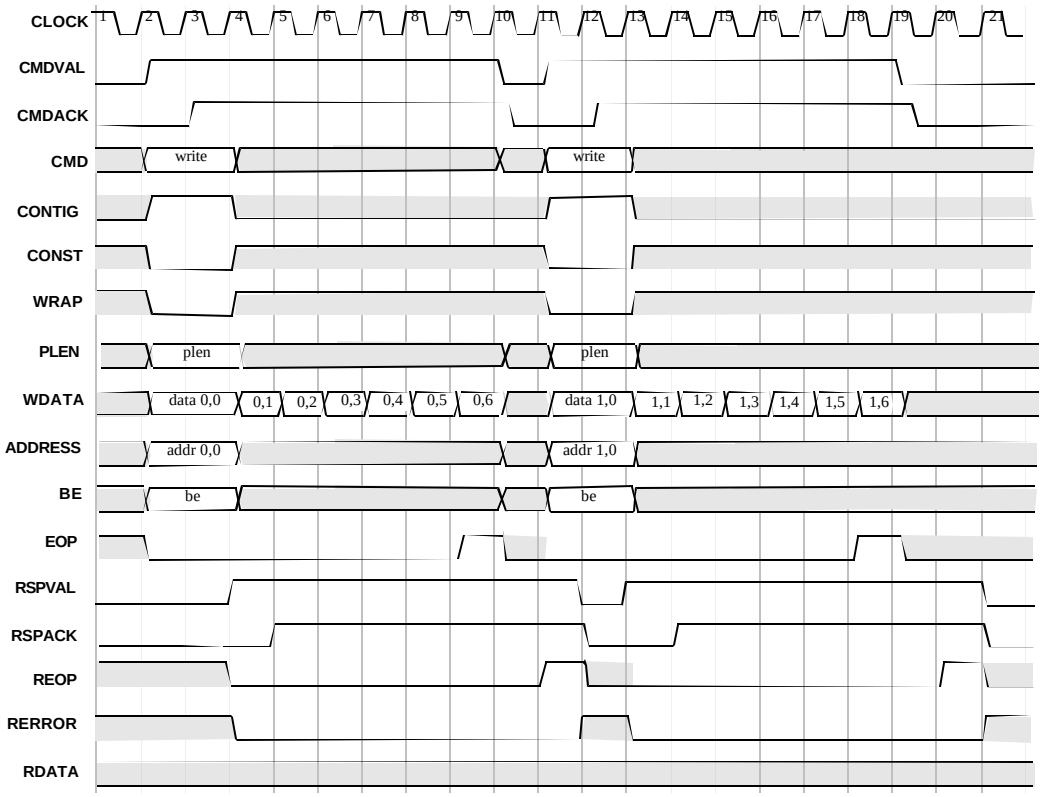


Figure 34: Advanced Write of Two Packets

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

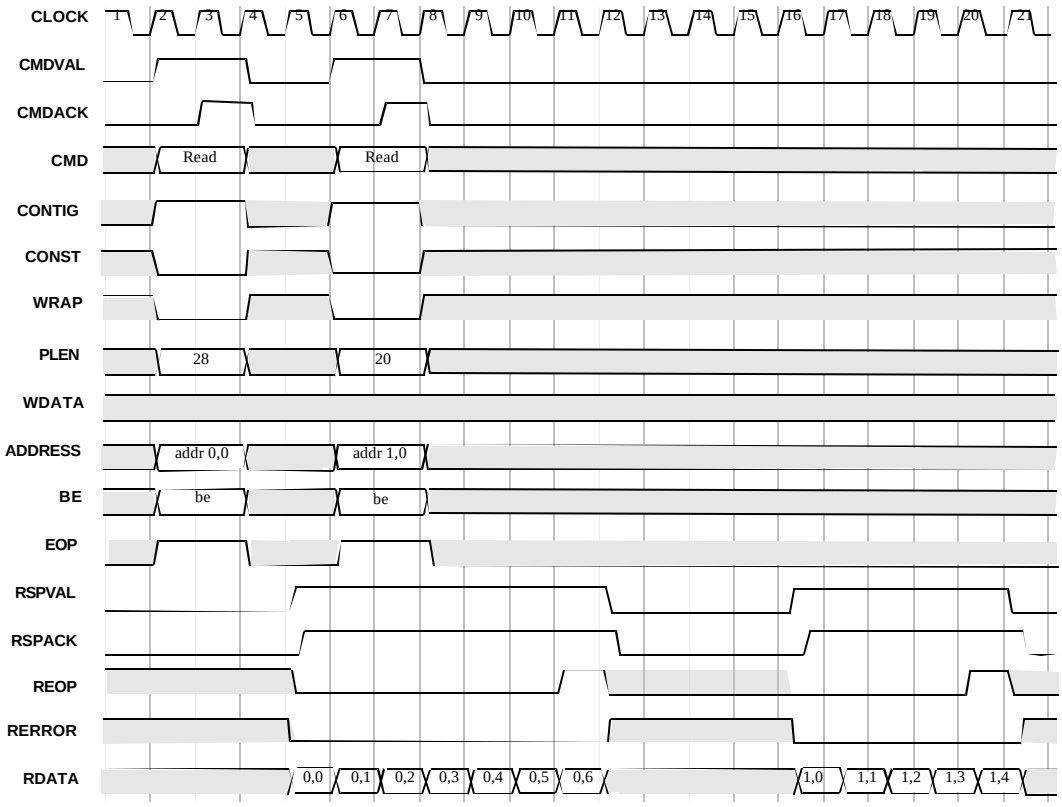


Figure 35: Advanced Read of Two Packets Over a 32-Bit Interface

Figure 35 shows an advanced write of two packets of 28 and 20 bytes over a 32-bit interface. The latencies between the packets are for example only. The packets can be requested and responded back-to-back if the system allows that.

5.4.6 Multi-Threaded Transactions

Initiators in a multi-threaded system are allowed to initiate multiple outstanding transactions. Hence, additional identification fields are required to properly differentiate each packet flowing in the system. These fields are transferred with each packet (and cell), and their values are constant over a packet. These fields are SRCID, TRDID, and PKTID. This information is sent with a request (CMDVAL) and returned with a response (RSPVAL). These signals act as labels (tags) for the initiator (or the interconnect) to identify and reorder the response packets in a multi-threaded, out-of-order completion interconnect.

In general, the SRCID and PKTID fields allow a transaction and its parts (packets) to be uniquely identified in the system. The TRDID extends SRCID, in case an initiator supports multiple threads. Figure 36 shows advanced packet transfer of two out-of-order packets, which have different source, packet, and thread identifiers. In the response, the packets come in reverse order. Some mandatory signals have not been drawn for readability. Notice that a packet is atomic, and the cells within a packet cannot be out of order. The RSPACK is in default acknowledge mode.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

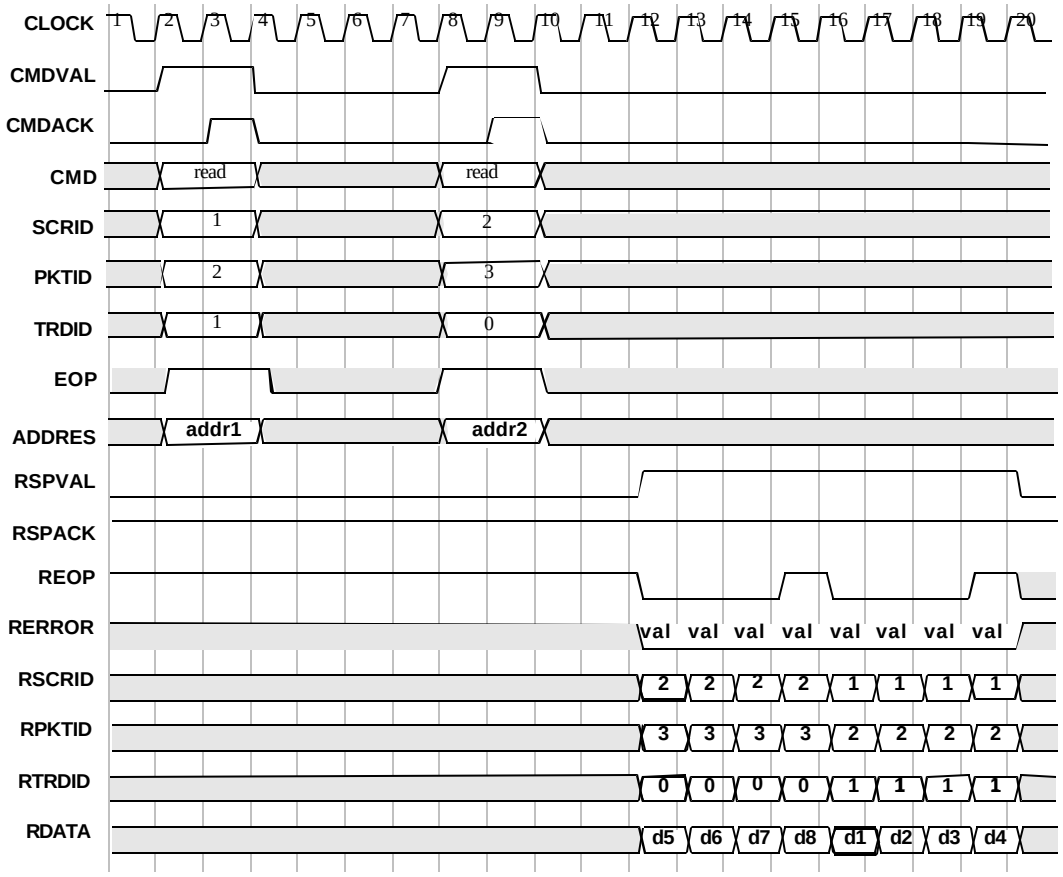


Figure 36: Advanced Read of Two Packets with Different Source, Packet, and Thread Identifiers

5.4.7 Arbitration Hiding Mode

A full-featured general arbitration hiding mode, where the request and response are arbitrated separately, may require all of the signals ACMVAL, AADDR, ACMACK, ARSPVAL, ARSCID, and ARSPACK. A lesser system will only use a subset of these signals. A parameter is required to indicate this matter.

Figure 37, Figure 38, and Figure 39 show how arbitration hiding works. Notice that the arbitration hiding is shown only for the second packet transfer for simplicity. Nothing prevents applying it to the first packet also. Figure 37 shows a normal packet transfer for comparison. The one-cycle delay between the packets is for example only; it is not a requirement of the packet transfer. The handshake is synchronous, which can be seen as a one-cycle delay in the first cells. The target uses the EOP signal to do an efficient burst transfer, which is seen from the lack of further acknowledge delays; the rest of the cells in the packets are transferred in one cycle each.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

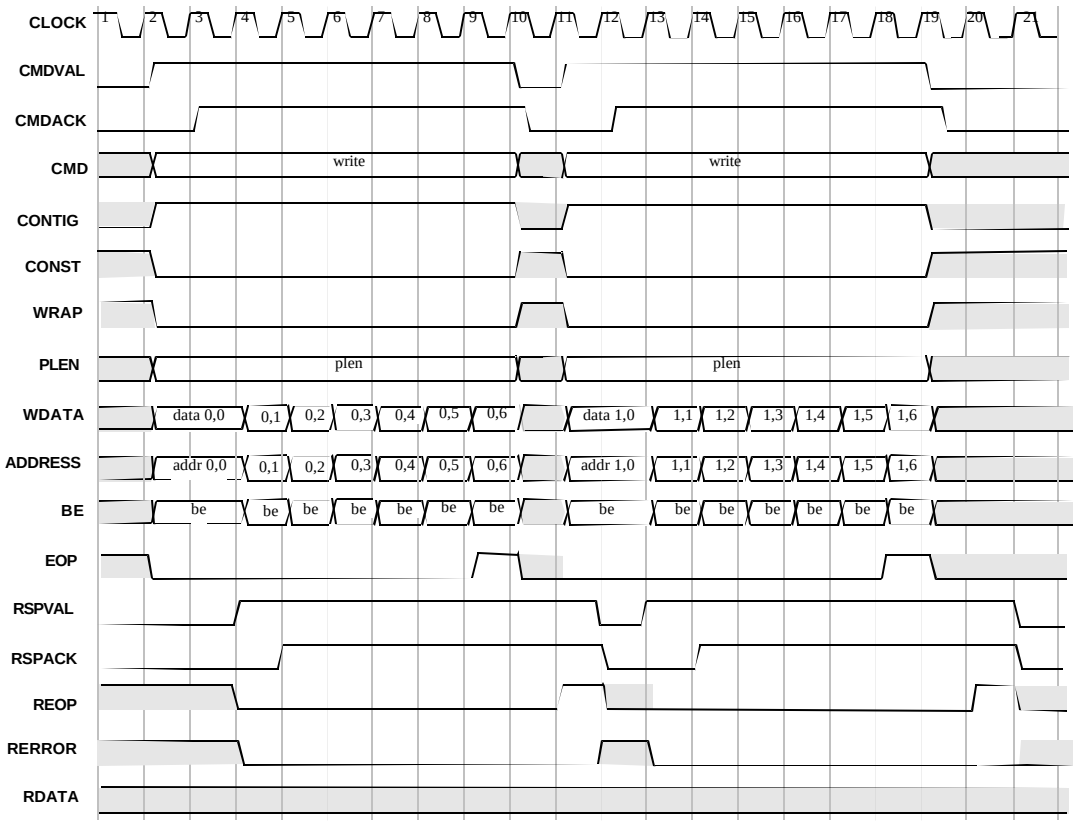


Figure 37: Normal Packet Transfer

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

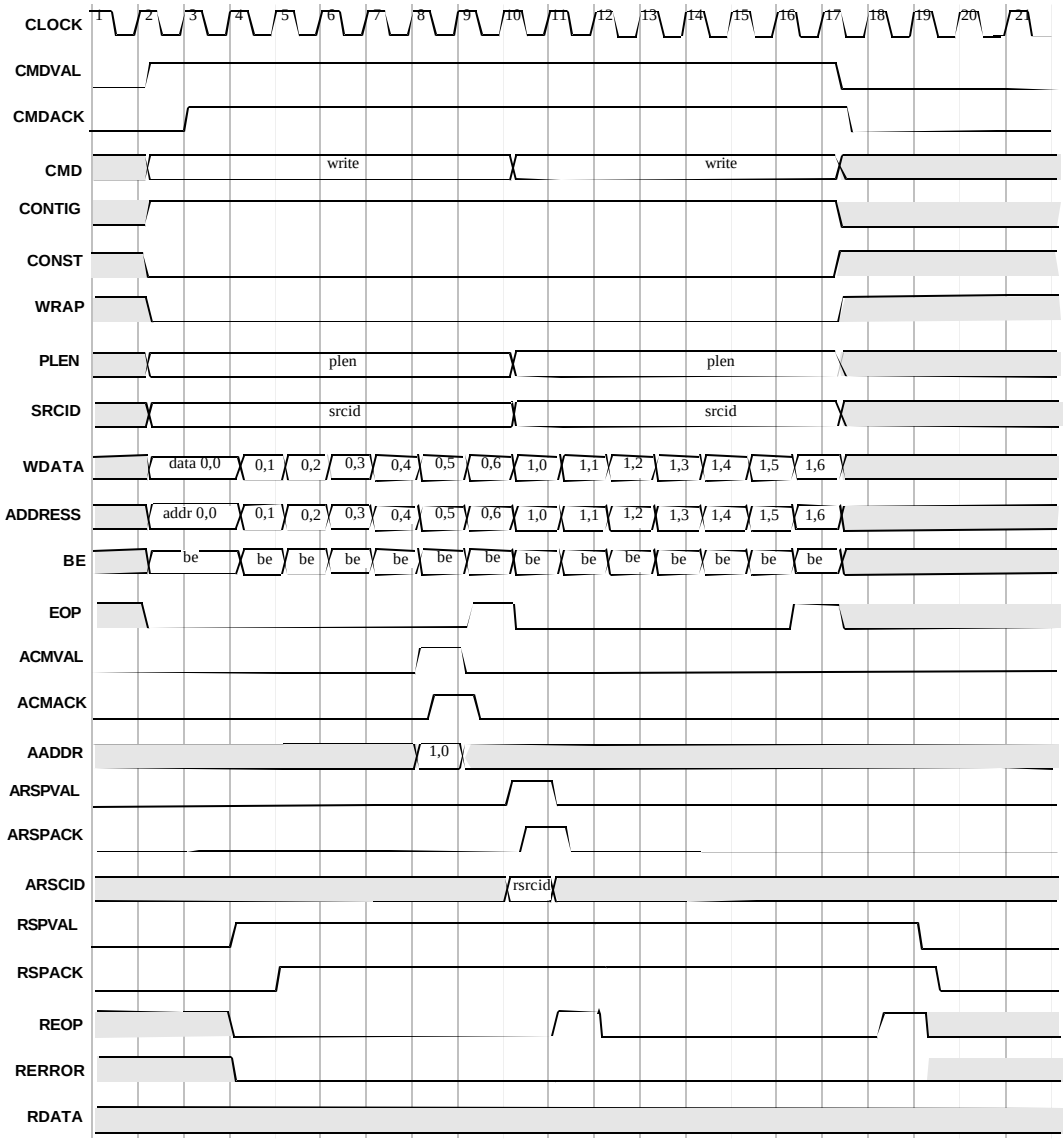


Figure 38: Packet Transfer with Arbitration Hiding

Figure 38 shows the general arbitration hiding. As can be seen, the second packet transfer can start earlier than normally. The arbitration hiding handshake acknowledges are asynchronous in the examples. The arbitration request handshake gives the first address and source identifier of the second packet, and the response handshake returns a copy of the source identifier.

5.4.8 Address Modes

The AVCI addressing is similar to the BVCI with two exceptions: there are two different wrap address modes, and there is a defined address mode. By default, AVCI requires the initiator to provide cell addresses along with every request cell transferred to the target module. In addition, the initiator may also specify a predefined algorithm to calculate subsequent cell addresses from the previous cell address using the four addressing mode flags: CONTIG, WRAP, CONST, and DEFD. When none of the flags is asserted by the initiator, there is no predefined relationship among cell addresses.

The random, contiguous, and constant address modes are the same as in the BVCI. The defined and wrapped modes are described below.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

5.4.8.1 Defined Address Mode

Figure 39 shows an example using the defined address mode. In this example, the initiator and the target have a mutual agreement that if the DEFD signal is active, the address is incremented by 28 in each cell. The address is drawn in the figure. In the example, the target can respond to the defined address pattern as quickly as to an incrementing address burst. Both handshakes are synchronous in the example. (Those signals irrelevant for illustrating the behavior are not shown.) If more address patterns are needed, the source identifier can be utilized to create virtual initiators. Each virtual initiator (with a different TRDED) can have a different address pattern.

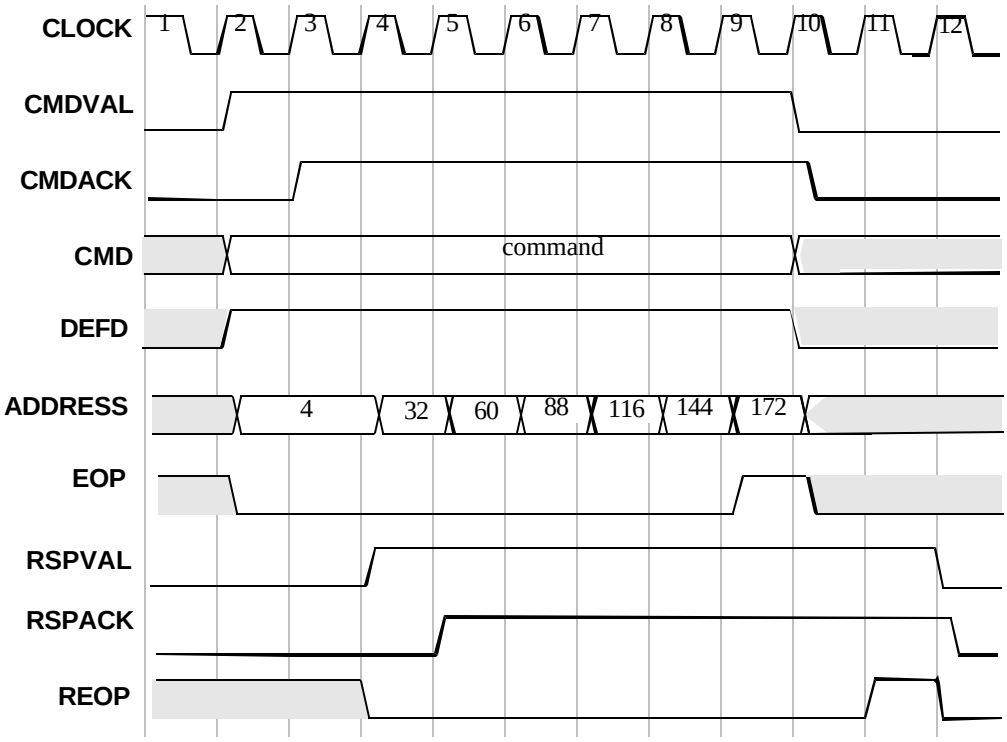


Figure 39: A Packet with Defined Address Mode

5.4.8.2 Wrap Address Modes

AVCI can use two wrapping modes: the BVCI wrapping based on PLEN, and a special mode based on the wrap length specifier. Figure 40 shows how the wrap length specifier is used to indicate a wrapping address different from that indicated by the packet length field. If the value of the WRPLEN signal is > 0, the PLEN is overridden in wrapping. For this reason, this wrapping mode is not compatible with BVCI. When connecting an AVCI initiator with a WRPLEN signal to a BVCI target, this wrapping mode cannot be used. In Figure 40 the RSPACK is tied high.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

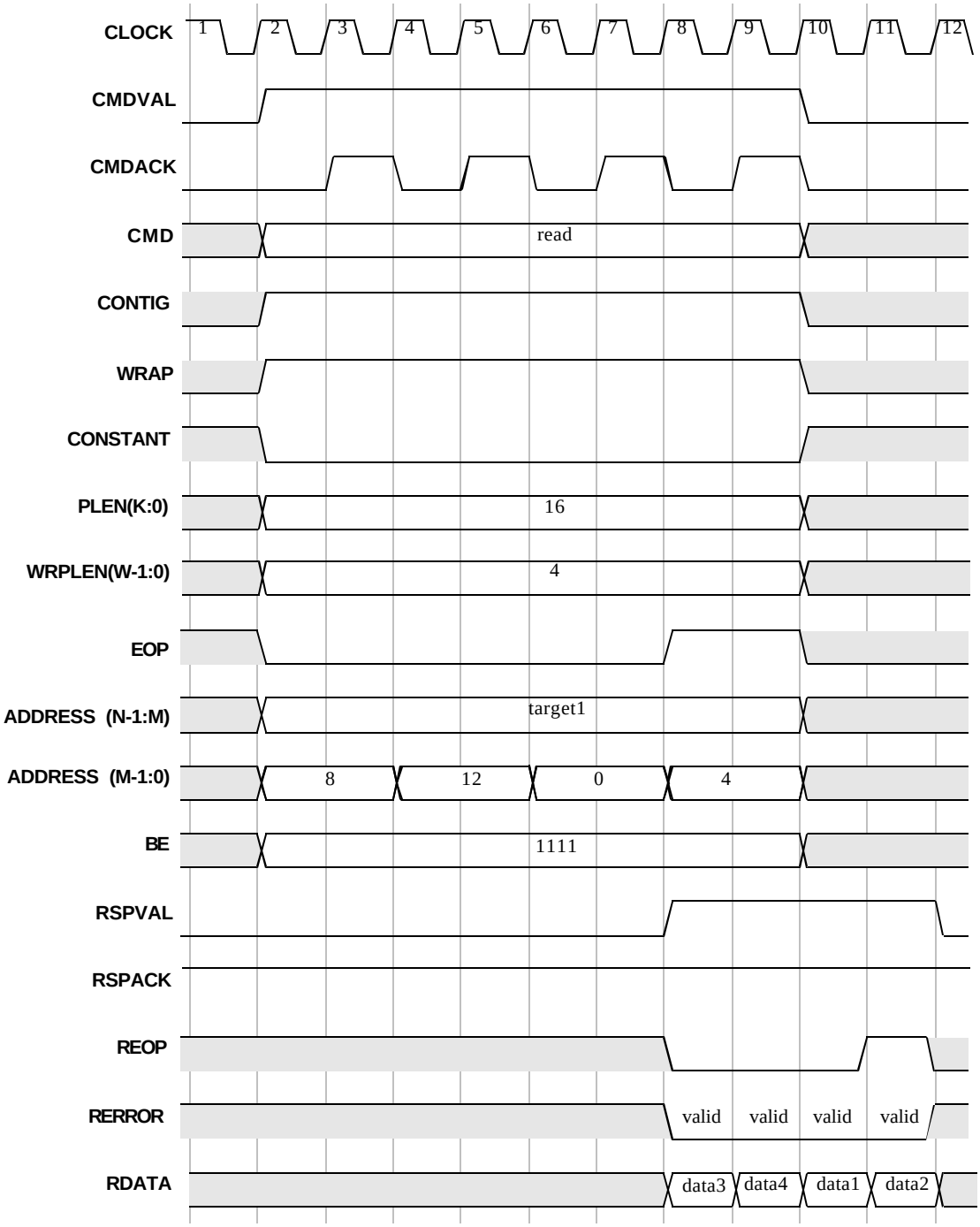


Figure 40: Packet Write with Specified Wrap Length

5.4.9 AVCI Signaling Rules

The BVCI signaling rules apply to AVCI, with the following exceptions:

- For each packet transferred on the request channel, there should be exactly one response packet transferred back on the response channel.
- The order in which packets are transferred on the response channel should be same as the order in which the corresponding packets appear on the request channel.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

- The number of cells in the response packets should be same as in the corresponding request packets, regardless of the opcode.
- The order of cells within a response packet should be same as the order in the corresponding request packet.
- Once a cell is placed on the interface and CMDVAL is asserted, the initiator module cannot modify the cell content until the requested cell transfer is complete, or system defined time-out occurs.
- ADDRESS(n-1:m), the top n-m bits of the ADDRESS signal indicating the target being addressed should remain constant across all the cells in a packet.
- Opcode signals, CMD(1:0), and the ADDRESS mode flags should remain constant throughout the packet transfer.
- The PLEN signal, which indicates the length of the request packet, should remain constant throughout the packet transfer.
- The CLEN and CFIXED signals should remain constant across all the cells within a packet. It is recommended that the field, CLEN, should either stay constant or decrement by one with every packet transfer. It is legal for the initiator to change CLEN in unexpected ways between packets. This behavior is discouraged, as it adds complexity to the target. Initiators that exhibit this discouraged behavior must document it, and targets must document what CLEN behavior they can accept.
- Changing the CMD field between packets in a packet chain is strongly discouraged. Initiators that exhibit this discouraged behavior must document it.
- Asserting the flag WRAP without asserting CONTIG is invalid. Asserting WRAP when PLEN is not a power of 2 value is also invalid.
- The initiator shall not assert byte enable bits for those bytes that are outside the address range indicated by the address and PLEN fields.
- It is recommended that once the receiving module asserts CMDACK, the module cannot change that signal until the next clock cycle. This ensures that the transferred cell is not corrupted.

The order of packets transferred in request and response channels may be different, if the packets can be identified based on source identifier, packet identifier, and thread identifier.

- In advanced packet mode, the number of cells in request and response packets may be different. A read request may have only one cell, and a write response may have only one cell. However, a read response and a write request can have multiple cells, depending on the PLEN field and cell size.
- Asserting CONTIG-flag overrides DEFD-flag.
- Asserting WRPLEN field with a value other than 0 while wrap is high overrides PLEN field for defining the wrapping address. Thus, asserting WRAP for packets with lengths not a power of 2 is legal.
- The arbitration hiding request handshake must be completed at least one clock cycle before the packet to which the arbitration hiding applies.
- The address field is invalid for the current packet during the special arbitration hiding request handshake.
- The special arbitration hiding handshake cannot be performed during the first cell of any packet.

5.5 Additional Timing Diagrams

Figure 41 and Figure 42 are additional timing diagrams. These diagrams show arbitration hiding without response arbitration handshake in AVCI packet write and read operations.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

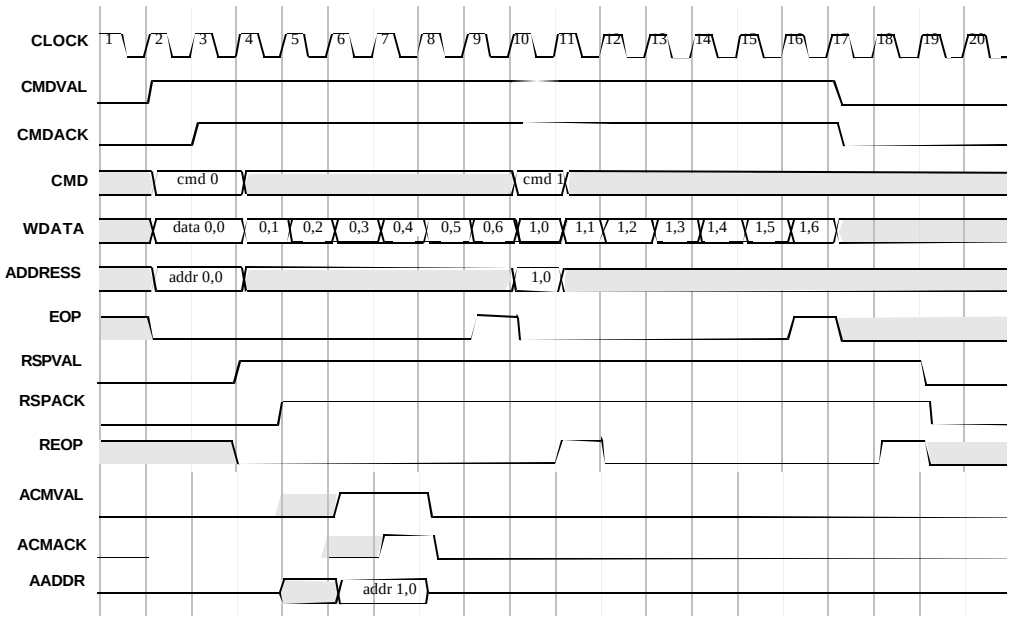


Figure 41: Advanced Packet Write with Arbitration Hiding, Without Arsval Handshake

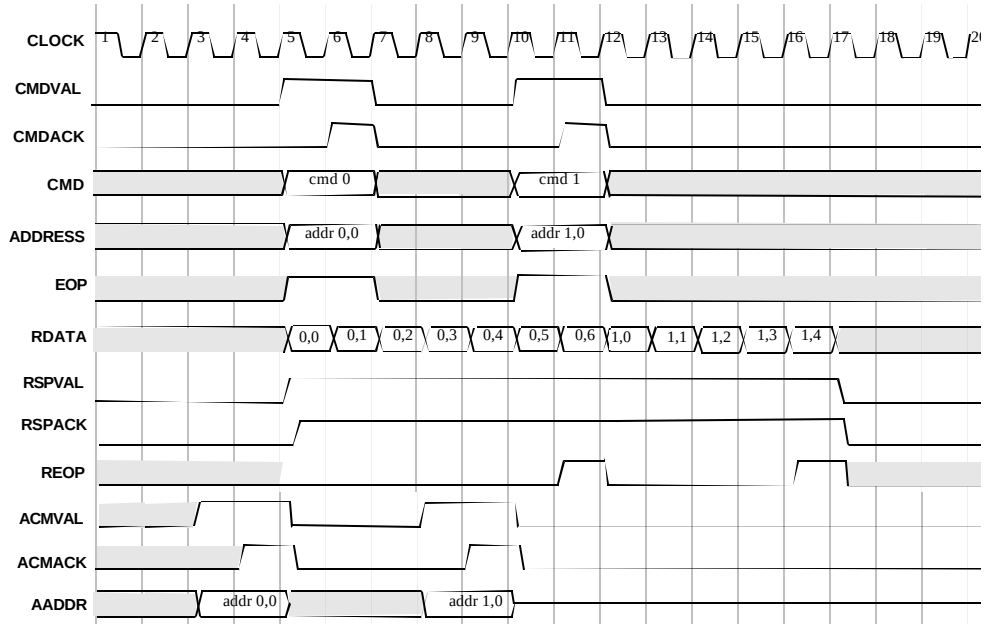


Figure 42: Advanced Packet Read with Arbitration Hiding, Without Arsval Handshake

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

6. Design Guidelines

6.1 User Guide

This section describes the high-level responsibilities of the VC initiator and VC target, together with the associated implementation considerations. It also discusses the differences between Basic VCI (BVCI) and Peripheral VCI (PVCI). The focus is in describing use of the VCI with an on-chip bus.

The VC initiator to OCB initiator wrapper interface is identical in protocol to the OCB target wrapper to VC target interface . The differences between the VC and the OCB show up in the implementation considerations. There are at least two good reasons to keep the VC->OCB interface identical to the OCB->VC interface:

- This facilitates direct VC=> VC connections (that is, without an intervening OCB).
- VC initiators have communication features similar to OCBs. Thus, we can use existing VC initiators as reasonable test cases to ensure that the OCB initiator responsibilities are reasonable. A similar argument may be made for the OCB target responsibilities.

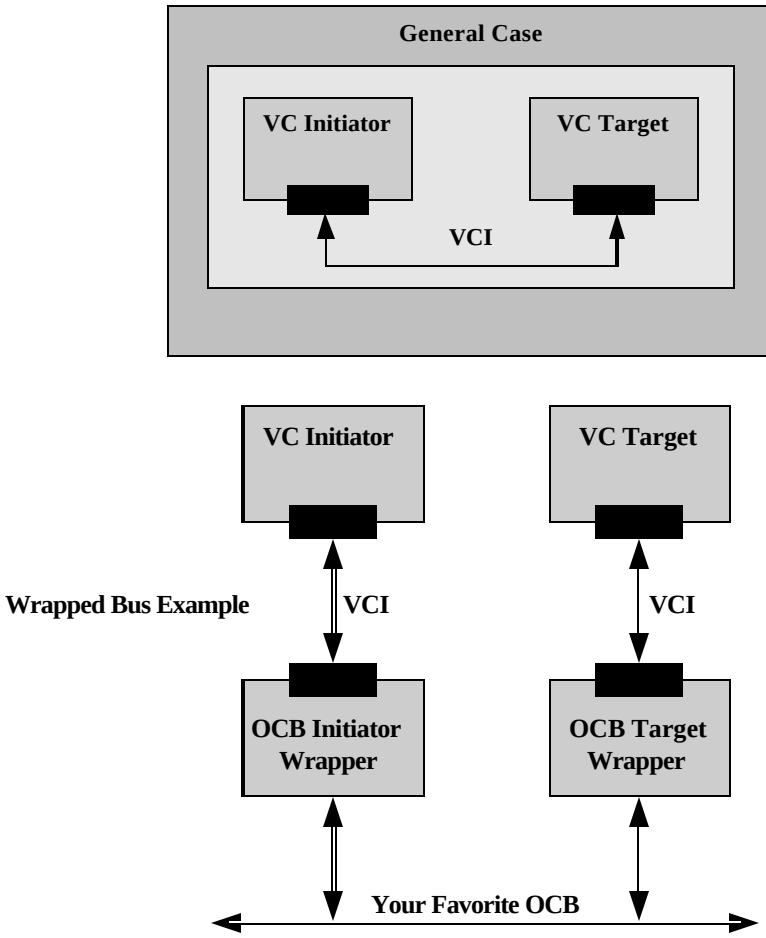


Figure 43: VCI Block Diagram

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

The intended use for the PVCI is in VC-VC communication, and wrapping a VCI target to an on-chip bus. Although it is possible to use the PVCI as a bus initiator, this is an unlikely case, since the peripheral bus initiator is usually a bus bridge from a system bus. The PVCI properties are tuned more towards their use as a peripheral bus target. The BVCI is more versatile, and its features are tuned towards use as either an initiator or a target for an OCB.

Another possible topology is presented in Figure 44. In this case, there is no bus, but all the VCI targets are connected point-to-point to a central OCB-VCI bridge. In this case, the bridge looks like a single target to the OCB in question, and contains a number of VCI initiators. The bridge is more complicated than a bus wrapper, but the electrical design of point-to-point VCI connections may be easier than the design of a shared bus.

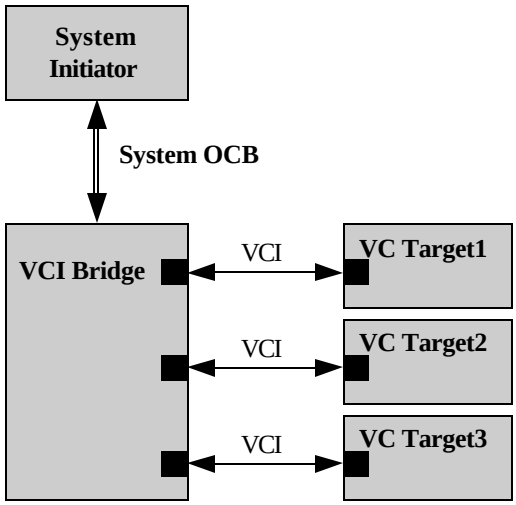


Figure 44: VCI with Star Topology

The following discussion covers the responsibilities of four different entities, as depicted in Figure 43: VC initiator, OCB initiator wrapper, OCB target wrapper, and VC target. It also covers the main differences between the BVCI and the PVCI.

6.1.1 VC Initiator Responsibilities

The VC initiator is in complete control over the presentation of requests on its VC interface; there is *no* arbitration. It asserts the CMDVAL (VAL for PVCI) signal to indicate that a valid request information is present. The size of ADDRESS and [W|R]DATA are dictated by the VC's capabilities, as is the set of supported transfer widths. The initiator must hold the request information stable until it samples CMDACK (ACK for PVCI) asserted at the rising edge of CLOCK. On a read transfer, the PVCI initiator must acquire RDATA after it samples ACK asserted at the rising edge of CLOCK in PVCI. In BVCI, the VC initiator can delay acquiring RDATA by holding RSPACK de-asserted while RSPVAL is asserted. Once the BVCI initiator asserts the RSPACK, it must complete the response transfer after it samples RSPVAL asserted at the rising edge of CLOCK.

The provider of the VC initiator must provide a list of VCI configuration parameters for the VC, including such aspects as address and data bus widths, supported transfer widths, and timing data. These parameters allow the system integrator to configure the OCB initiator wrapper to meet the constraints of both the system and the VC initiator.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

6.1.2 OCB Initiator Wrapper Responsibilities

The OCB initiator wrapper is responsible for accepting the request from the VC initiator, and controlling the OCB (as an OCB initiator) to accomplish the transfer. In particular, the OCB initiator wrapper is responsible for making requests to and accepting responses from the bus arbiter, initiating the transfer on the OCB (inserting any required OCB turn-around cycles), handling any OCB-level handshaking, and getting Read data from the OCB. In most instances, the OCB initiator wrapper should not need to store any request information (particularly address or write data) on behalf of the VC initiator, since the OCB initiator wrapper CMDACK (ACK) signal can be used to force the VC initiator to hold the request information. It may need to store response information (such as RDATA) in the BVCI, if the VC initiator expresses that it cannot accept the read data by not asserting the RSPACK.

However, electrical concerns will typically force the OCB initiator wrapper to buffer the request signals to drive long OCB wires, as well as dealing with OCB topology issues (for example, tri-state and multiplexed buses). The OCB initiator wrapper is responsible for ensuring that the OCB timing is correct, and that VCI RDATA and handshaking signals make the required setup time to the VC initiator.

The OCB initiator wrapper should be configurable enough to adapt to a range of possible VC initiator capabilities. In particular, the OCB initiator wrapper should have variable address and data bus widths, and support for multiple transfer widths. This configurability allows the system integrator to match the OCB initiator wrapper to the VC initiator.

Several issues arise with respect to the configuration of the OCB initiator wrapper. A VC initiator with a 16-bit address may need to connect to an OCB with a 32-bit address. In such a case, the system integrator should be free to synthesize the most significant 16 bits of the OCB address by whatever means are appropriate for the system. One way might be to statically configure the upper bits into the OCB initiator wrapper. Another method might involve placing a writable register somewhere on the OCB to hold the upper bits. The OCB initiator wrapper is also responsible for any required data shifting. Data shifting, which can arise from width mismatches or endianness differences between the VC initiator and the OCB, should not be required. This requirement can generally be lifted by the appropriate connection of the VC byte lanes to the VCI byte lanes, if the endianness is of static nature.

6.1.3 OCB Target Wrapper Responsibilities

The OCB target wrapper is in complete control over the presentation of requests on its VCI; there is *no* arbitration. The OCB target wrapper acts as an OCB target; it must convert the OCB transfer into a VCI-compliant transfer. In particular, it must translate OCB request information into the VCI CMDVAL (VAL) signal, which indicates the presence of a valid VCI request *that is intended for the VC target*. Device selection mechanics are very specific to different OCBs. Traditional approaches to device selection include distributed address decoding and centralized address decoding. For a distributed-decoding OCB, the address decoder to determine VCI CMDVAL (VAL) will likely be inside the OCB target wrapper. For a centralized-decoding OCB, the VCI CMDVAL (VAL) signal is likely just a buffered version of the select signal from the centralized decoder.

The OCB target wrapper presents the request information across the VCI to the VC target. It must hold the request information stable until it samples the VCI CMDACK (ACK) asserted. Since the OCB target wrapper is responsible for ensuring compliance with the OCB transfer protocol, most OCB target wrappers will use OCB target handshaking to force the OCB target wrapper to hold the request information stable (that is, insert wait states). Thus, data and address storage is rarely required in the OCB target wrapper. The OCB target wrapper may need to delay the assertion of CMDVAL so that it and the other request fields satisfy the VC target setup time. It should wait until the VC target returns CMDACK before releasing the OCB resources associated with the transfer. It is the OCB target wrapper's responsibility to ensure that VCI RDATA makes the timing path back across the OCB to the OCB target wrapper. Since the two-wire PVCI handshake does not allow the OCB target wrapper to force the VC target to hold RDATA, the OCB must run slow enough to allow RDATA to make it back across the OCB or else data storage is required inside the OCB initiator to acquire RDATA. The BVCI handshake allows for the OCB target wrapper to stall the VC target. In any case, it is typically the OCB target wrapper's responsibility to buffer RDATA to drive the OCB data wires, thereby isolating the VC target from the loading and topology (tri-state, multiplexed, and so on) of the OCB.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

The OCB target wrapper should be configurable enough to match its VCI address and data bus widths, and transfer widths to the VC target's implementation of the VCI. This configurability is key to allow the system integrator to design working systems using the VCI. When the OCB has a different data bus width than the VC target, it is the OCB target wrapper's responsibility to accomplish any required data multiplexing. The VC target ADDRESS is typically much narrower than the OCB ADDRESS; the VC target should be configurable so that only the required ADDRESS bits are presented to the VC target.

6.1.4 VC Target Responsibilities

The VC target is responsible for accepting the request from the OCB target wrapper and accomplishing the transfer. Since every request presented to the VC target is intended for it, the VC target should be designed to acknowledge *every* transfer. In simpler terms, the ACK response will often be a delayed version of the VAL. The VC target should typically delay assertion of ACK until it has completed the transfer of the request; this forces the OCB target wrapper to hold the request information steady, thus eliminating the need for the VC target to store the request.

The provider of the VC target must provide a list of VCI configuration parameters for the VC, including such aspects as address/data bus widths, supported transfer widths, and timing data. These parameters allow the system integrator to configure the OCB target wrapper to meet the constraints of both the system and the VC target.

6.1.5 VCI-to-VCI Conversions

VCI components with different parameters, such as size, are not supposed to connect together directly. A special wrapper has to be used in this case, as shown in Figure 45. The wrapper is a VC, which takes care of buffering, and other logic needed to map different VCIs together. This situation occurs when using VCI for point-to-point communication. When connecting different VCIs through an OCB, the OCB wrappers take care of size conversions.

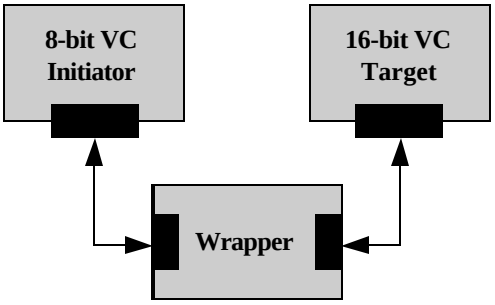


Figure 45: Interconnecting Different-Size VCI Components

6.1.5.1 General VCI-VCI Interoperability

In general, VCI components can be connected with the following conditions:

- An initiator can be connected to a target, and vice versa.
- The address size of the initiator and the target must match. If the initiator's address space is smaller than the target's address space, all the target's addresses cannot be accessed. However, the target's address size can be smaller than the initiator's address size.
- The cell size must match. If the initiator's cell is wider than the target's cell, a wrapper is needed to convert one wide access into two or more narrower ones.
- The endianness must match. If it does not, a wrapper is needed to convert the endianness.
- If the target does not support Free-BE mode, the initiator must honor that.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

6.1.5.2 AVCI-BVCI Interoperability

A BVCI initiator can be connected to an AVCI target with the following conditions:

- General VCI-VCI interoperability rules must be satisfied.
- The signals with the same names in AVCI and BVCI are directly connected.
- DEFT is tied low.
- WRPLEN is tied low.
- Arbitration hiding input signals are tied low.
- TRDID signal is tied low.
- PKTIDsignal is tied low.
- SRCID signal is tied low.
- The rest of the AVCI output signals are not connected.

An AVCI initiator can be connected to a BVCI target with the following conditions:

- General VCI-VCI interoperability rules must be satisfied.
- Only BVCI transactions can be used.
- The signals with the same names in AVCI and BVCI are directly connected.
- PUREPACKET must be False, and BVCIMODE must be True.
- RFLAG signal is tied low.
- Arbitration hiding input signals are tied low.
- RTRDID signal is tied low.
- RPKTID signal is tied low.
- RSRCID signal is tied low.
- The rest of the AVCI output signals are not connected.

6.1.5.3 BVCI-PVCI Interoperability

Connecting a PVCI initiator to a BVCI target is not a likely scenario. It can be done through a wrapper, if necessary. The wrapper must take care of the handshake conversion.

A BVCI initiator can be connected to a PVCI target through a wrapper without compromising BVCI performance. It is the wrapper's responsibility to store pipelined BVCI transactions, and to convert these transactions to non-pipelined and non-split PVCI transactions.

A BVCI initiator can be connected directly to a PVCI target with following conditions:

- General VCI-VCI interoperability rules must be satisfied.
- Only PVCI transactions are allowed. Packets with other than contiguous, unwrapped address modes are converted to single-cell packets in PVCI. Read Lock cannot be used.
- The CMDVAL is connected to VAL. (Notice also the CMD encoding listed below.)
- The ACK is connected to CMDACK and RSPVAL.
- The RSPACK is set to Default ACK mode.
- The EOP signal of the PVCI target is set high, except when CONTIG = 1, WRAP = 0, CONST = 0, EOP signals can be connected.
- The REOP signal is created from the PVCI EOP signal.
- The CMD signal is connected to the RD and CMDVAL signals with the following encoding:
    - CMD RD VAL
    - 00 don't care 0
    - 01 1 CMDVAL
    - 10 0 CMDVAL
    - 11 1 CMDVAL (in addition RERROR[0] = 1)
- ADDRESS signals are connected.
- RDATA signals are connected.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

- RERROR signals are connected.
- BE signals are connected.
- WDATA signals are connected.
- The rest of the BVCI signals are not connected.

6.2 VCI Parameterization

This section deals with parameters that are not part of operations, that is, they are not communicated with VCI signals.

The VCI parameters can control:

- Generation of the VC itself as a soft VC
- Configuration signal tie-offs
- Dynamic configuration of a VC through registers, pins, and so on

Independent of the implementation model abstraction level, language, or format, these parameters must be documented. The implementation of the parameters in the models can be done with generics (for example, in HDL), setup or header files, etc. In case of a configurable hard VC, some of these parameters may be implemented with external pins that can be tied-off. Table 17 presents the notation for the scope where each parameter applies. In addition, each of the parameters can be used to control generation of the VC, for example, by setting the values in a GUI (graphical user interface).

Table 17: Parameter Scopes

Name	Description
DOC	Required in documentation.
HARD	Can be used as a pin parameter in any kind of VC to set the parameter in integration time statically.
SOFT	Can be used as a soft parameter for generating or parameterizing the VC implementation source code.
DYN	Can be used dynamically, either with a dynamically connected pin, or a register that can be accessed through the VCI. In either case, the behavior is customized, and must be documented.

6.2.1 Background for Parameterization

For this discussion, VCs can be broadly divided into three separate categories, two of which are only subtly different. These are:

- Hard VC (for example, a Standard Cell VC with layout)
- Soft VC Generated
- Soft VC Compiled

A Firm VC can fall into either the soft or hard category, depending on the configurability.

Each of these has differing needs when it comes to parameterization. A Hard VC can not be further customized without the need for re-layout, unless the VC developer has the forethought to provide additional configuration pins. One issue that Hard VC developers face is that they do not want to have multiple flavors of their VC lying around. An approach to this issue is to add the small extra logic to configure the component to satisfy the system needs. This does cost silicon, and is not always possible. The chief benefit of Hard VC is that it gives known performance in a given technology.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

The Soft VC Generated and Soft VC Compiled have opposite goals. In these categories, the VC vendor is often specifically seeking to have the VC be technology independent. This is in contrast to the Hard VC, which is painful to migrate from one technology to another. Most VC developers who develop Soft VCs do so using the C method: "Soft VCs Compiled." In this method, the VC model is built once up front with all the options required. Through a means of telling the VC what options are wanted, the VC is compiled and the logic not required is removed during compilation. This method is analogous to *#ifdef* in the C programming language. However, there are some disadvantages to this method. First, there is additional overhead during compilation to remove unneeded features. This means that a lot of CPU time can be used for unneeded features. Second, the compiled code is often not as good as it could be if the feature set were simply instantiated up front. In the other software method, "Soft VC Generated," the HDL code for the desired feature set is instantiated all at the same time. Most important, only the desired features are present. This method leads to the optimal combination of time and performance at the VC user's end, with a modest investment in time at the VC developer's end. With the use of the "Soft VC Generated" method, the VC can be algorithmically generated for each appropriate instance.

These different approaches require different parameterization schemes. For example, the Hard VC must inherently utilize pins on the VC to tie options Off or On by tying the pin to a power or ground. On the other hand, the Soft VC options can support their parameterization in a few different ways: by adding hardware pins which get tied to power or ground; by reading in a file or command line and logically or physically applying the values read to the pins; or by specifying the options in a GUI-based software program that either applies values, instantiates code, or does both.

6.2.2 Parameters

In tables 18, 19, 20, and 21, "True" means high signal value, and "False" means low signal value or "0." The default value is "0" for all the size- and width-type parameters. The default value of the binary-type parameters is "True" or "1."

Table 18: Common Parameters for all VC Interfaces

Name	Description	Range	Scope
INITIATOR	True if a VCI initiator	0,1	DOC
ADDRSIZE (N)	Address size, width of address bus in bits	0 to 64	DOC, SOFT
CELLSIZE (B)	Cell size, number of bytes in the data bus. If either RDATA or WDATA is narrower than this, the unused signals must be tied to logic zero in either the initiator or the target interface. If a BVCI is made wider than eight bytes, it becomes an AVCI.	1, 2, 4 for PVCI 1, 2, 4, 8 for BVCI 2^i, $i \in \{0,1,2,3,4,5\}$ for AVCI	DOC, SOFT
BIGENDIAN	True, if the component is big endian, False if it is little endian.	0, 1	DOC, HARD, SOFT, DYN
NOENDIAN	True, if the component does not care about endianness (such as a memory). Overrides the BIGENDIAN parameter.	0, 1	DOC
RESETLEN	Minimum number of clock cycles reset required being active. If 0, use default = 8 clock cycles.	Positive integer	DOC
ERRLEN (E)	Number of error extension bits (defined as E). The default is 0.	0 to 2 for PVCI, BVCI 0 to 3 for AVCI	DOC, SOFT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 19: Parameters Specific to PVCI

Name	Description	Range	Scope
FreeBE	True, if the VCI supports unrestricted combinations of byte enables. The restricted BE combinations, as defined in Section 3.4.3, “Data Formatting and Alignment,” must be supported by all VCI implementations.	0,1	DOC
DefACK	True, if ACK is allowed to be asserted even when VAL is low (de-asserted)	0,1	DOC, HARD, SOFT

Table 20: Parameters Specific to BVCI

Name	Description	Range	Scope
CHAINING	True if the VCI is capable of packet chaining. Chaining VCI contains all signals and fields of type MC shown in Table 5.	0,1	DOC, SOFT
PLENSIZE (K)	Width of PLEN signal in bits	0 to 9	DOC, SOFT
CLENSIZE (Q)	Width of CLEN signal in bits	0 to 8	DOC, SOFT
DefCMDACK	True, if CMDACK is allowed to be asserted even when CMDVAL is low (de-asserted)	0,1	DOC, HARD, SOFT
DefRSPACK	Similar to DefCMDACK, but for the Initiator. True, if the initiator is always ready to acknowledge RSPVAL (the initiator is not allowed to insert a wait state).	0,1	DOC, HARD, SOFT

6.2.2.1 General for BVCI

The initiator contains all the signals and fields of the type MA and MI. The target contains all the signals and fields of type MA. Table 21 shows the default values for command extensions. These are used to document the initiator’s and the target’s behavior while they do not support full functionality of fixed packet, constant packet, contiguous packet, or wrapping address. For the initiator, the following parameters state that it will always send request of a certain behavior, and a target only expects requests with a certain behavior. The signal itself may or may not exist, but the behavior is known through these parameters, and the signals can be connected (tied off) accordingly.

Table 21: BVCI Default Values

Name	Description	Range	Scope
DCFIXED	Signal CFIXED is tied to High/Low	0,1	DOC, HARD
DCONST	Signal CONST is tied to High/Low	0,1	DOC, HARD
DCONTIG	Signal CONTIG is tied to High/Low	0,1	DOC, HARD
DWRAP	Signal WRAP is tied to High/Low	0,1	DOC, HARD

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Table 22: Parameters Specific to AVCI

Name	Description	Range	Scope
DEFINED	True, if support for *defined* transfer mode. (DEFD signal must exist)	0,1	DOC, SOFT
PUREPACKET	True, if non-incrementing packet model.	0,1	DOC, HARD, SOFT, DYN
BVCIMODE	True, if strictly in-order transactions are supported and WRPLEN = 0.	0,1	DOC, HARD, SOFT, DYN
GENERALARI	True, if general arbitration hiding request signals are supported. (ACMVAL, ACMACK, AADDR must exist.)	0,1	DOC, SOFT
GENERALART	True, if general arbitration hiding response signals are supported. (GENERALART must be true + ARSPVAL, and ARSPACK must exist.)	0,1	DOC, SOFT
RFLAGSIZE (F)	Number of bits in the RFLAG field	Positive integer	DOC, SOFT
SRCIDSIZE (S)	Number of bits in the SRCID and RSRCID fields	0 to 5	DOC, SOFT
PKTIDSIZE (P)	Number of bits in the PKTID and RPKTID fields	0 to 8 (greater or equal than CLENSIZE)	DOC, SOFT
TRDIDSIZE (T)	Number of bits in the TRDID and RTRDID fields	Positive integer	DOC, SOFT
WRPLENSIZE (W)	Number of bits in the WRPLEN field	0 to 5	DOC, SOFT
DefACMACK	True, if ACMACK is allowed to be asserted even when ACMVAL is low (de-asserted).	0,1	DOC, HARD, SOFT
DefARSACK	Similar to DefACMACK, but for the initiator. True, if the initiator is always ready to acknowledge ARSVAL (the initiator is not allowed to insert a wait state).	0,1	DOC, HARD, SOFT

6.2.2.2 General for AVCI

The AVCI consists of BVCI signals and parameters added with AVCI extensions. Thus the signals indicated as mandatory in Table 5 and the parameters in Table 20 apply to AVCI.

6.3 Implementation Guidelines

Since the VCI is fully synchronous, and the wiring between the wrapper and a VCI component is supposedly very short, the VCI implementation should be very easy to design. A legal VCI component *must* sample all the signals at the rising edge of the CLOCK. To create a legal VCI bus wrapper for an OCB that samples at the falling edge of the clock, buffering registers must be added.

The VCI standard should not pose any restrictions to making EDA tools that synthesize the bus wrapper automatically. This approach is preferred over manual wrapper design, since it theoretically reduces the need for implementation verification.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

In register transfer level implementation of the VCI, there are few restrictions: The flip-flops inferred must be rising-edge clocked and have active-low reset. The VCI standard does not pose any other electrical constraints to the physical implementation except for those timing constraints given in signal definitions. The VCI component provider must document the physical implementation parameters and constraints, such as voltage swings. The guidelines given in VSIA Implementation/Verification DWG specifications and in On-Chip Bus DWG Attributes specification are to be followed. The manufacturing test constraints and debug structures are not defined in the VCI standard. The guidelines given in the VSIA Manufacturing Test DWG specifications are to be followed.

In the case of a partly or fully combinatorial wrapper, the delay paths starting and ending at the VCI component traverse all the way through the on-chip bus. It is the system integrator's responsibility to ensure the proper system timing in this case.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

7. VCI Glossary of Terms

Advanced Packet Model	A packet model where request and response may have different number of cells.
Arbitration Hiding	Mechanism to give advance information to the arbitration logic to prevent arbitration cycles adding latency to VCI operation.
AVCI	Advanced Virtual Component Interface (VCI)
Bus Wrapper	Logic between the VCI and a bus.
BVCI	Basic Virtual Component Interface (VCI)
Cell	A cell is a grouping of one or more bytes. It contains a number of bytes that is characteristic to the natural width of the VCI implementation. It is the basic unit of information transferred across the VCI in one cycle.
DWG	Development Working Group
Initiator	A VC that sends request packets and receives response packets. It is the agent that initiates transactions, for example, DMA.
IP	Intellectual Property
MPSOC	multi-processor system-on-chip
OCB	On-Chip Bus
Packet	A transport object consisting of an atomic ordered set of cells transferred across the VCI.
Packet Chain	A non-atomic specialized transport object consisting of a set of logically connected packets transferred in the same direction across a VC Interface. The chain of packets is connected because no intervening packets are allowed on the same channel.
PCI	Peripheral Component Interconnect, the industry-standard PCI bus.
PVCI	Peripheral Virtual Component Interface (VCI)
Signal	Synonymous to electrical wire or net.
SoC	System-on-Chip
Target	A VC that receives request packets and sends response packets. It is the agent that responds (with a positive acknowledgment by asserting) to a bus transaction initiated by a initiator, for example, memory.
Transaction	Generic term used for any pair of request and response transfer.
Transaction Layer	This deals with point-to-point transfers between blocks or VCs. It does not define signal names or clock-cycle protocols.
VC	Virtual Component
VCI	Virtual Component Interface, an OCB-standard for communicating between a bus and a virtual component, which is independent of any specific bus or VC protocol.
VCI operation	A transport object consisting of a pair of packets that are transferred in different directions, for example, a single request-response packet pair.
VC wrapper	Logic between an existing VC and the VCI.
VSIA	The Virtual Socket Interface (VSI) Alliance.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

A. Transaction Language

A transaction-based language has been defined by the VSIA to facilitate easier testing and simulation of components and systems that use a VC Interface. The language provides the VC developer and the SoC integrator with a way to use the same test suite when the VC stands alone, and through the bus when the VC is integrated into a system. This can potentially save many hours of translating tests or redeveloping test suites. The language itself consists of a number of commands that represent VC Interface operations. There are formats for these commands for different uses. For example, there is a format for HDL simulator stimulus/response vectors, and a behavioral C function call format for easy writing of these vectors.

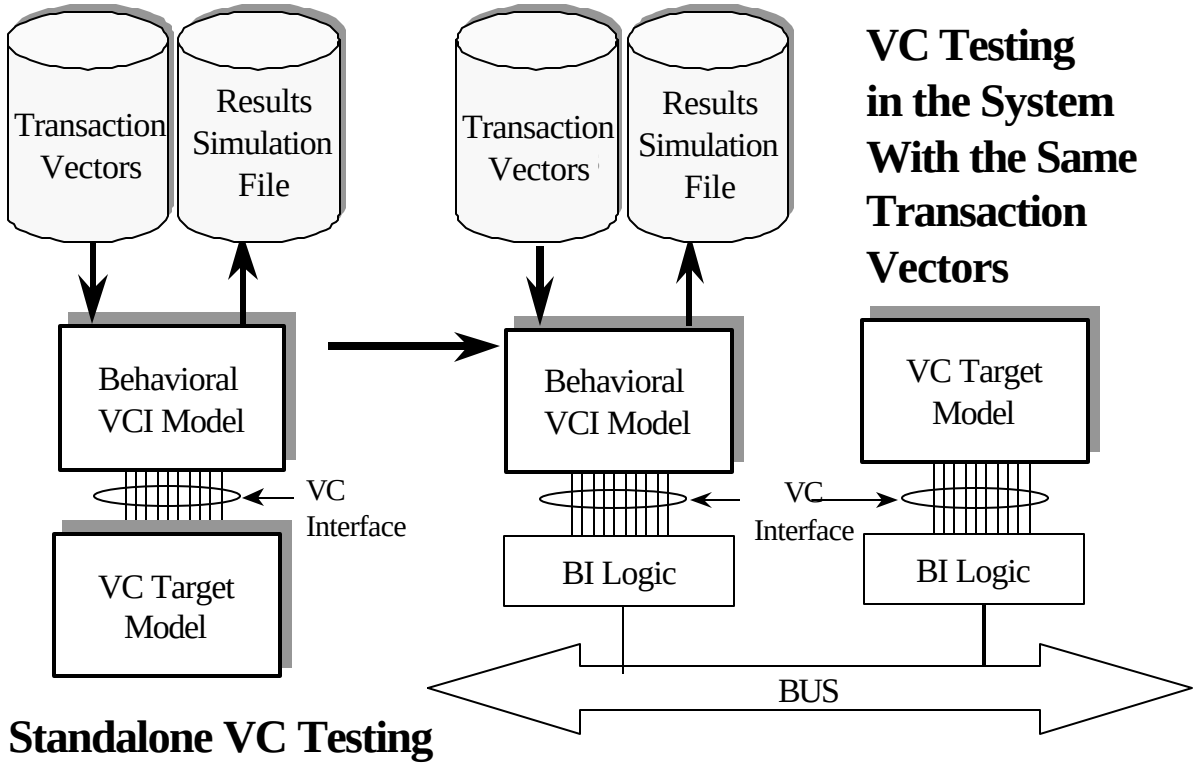


Figure 46: Simulating VCs with VC Interfaces

A.1 Use of VC Interface in VC and System Test

The example in Figure 46 shows how an integrator should be able to test and verify the VC's integration into a bus. In this example the VC provider will provide a set of transaction vectors as a compliance test for their VC. The integrator can verify the correctness of the VC (at the RT level or after synthesis or physical design) by using the behavioral model shown above. It can be expected that a number of VC vendors will offer such a tool to their customers. This tool could be tailored to the specific size and options of interface on the VC. The real productivity gains occur when the same parameterized, behavioral model can be connected to the On-Chip Bus, after the VC has been integrated with the bus. At this point the behavioral model can be retailored to the bus requirements for better system performance, and the compliance vectors can be used to check out the integration of the VC into the system. Furthermore, all the other VCs can also be tested individually through this same interface. It can be expected that a number of other tools like this behavioral model can then be created to generate more sophisticated tests, such as interleaving the various VC compliance vectors to create pseudo system tests, all with very little additional effort from the system integrator.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

A.2 Language Levels

There are two levels to the language as currently defined by the VSIA. At the lowest level the basic language is a direct mapping to the VC interface signals and cell transfers. There are two formats: a simple function call set of statements, and a file (simulation vector) oriented set of commands. The latter format supports simple binary expressions. This allows for assignment of constant values to variables. The file-oriented language is limited to single-pass variable assignments, to simplify the interpreter.

The highest language level abstracts the details of the VC Interface implementation, and concentrates on the essence of transferring arbitrary blocks of data. Single operations can be translated into multiple lower level operations that are mapped to a specific implementation of the interface. This level is structured to be compatible with a subset of the SL-VCI (as described in the *VSI Alliance System-Level Interface Behavioral Documentation Standard*) transactions.

A.2.1 VC Interface Language

The VC Interface Language is targeted to writing portable simulation vectors for VCI components.

There are two syntaxes defined: a function call format (here in C++) and a set of file based commands. The file-based commands are simple in the sense that they do not contain any comparison operations, or other features to help verification. Comparison of return values to expected values can be done by inspection, or by tools external to the language. The purpose of this syntax is to provide a language- and platform-independent vector format. The function call syntax implements the same transactions, but since it is meant to be embedded into a host language, the host language's constructs can be used for building intelligent tests.

This section defines first the abstract transactions and their parameters. Any implementation of the language must implement those.

A.2.1.1 Interface Parameters

The VCI parameters discussed in Chapter 6, "Design Guidelines," must be made visible to the language implementation also. These parameters are used for mapping the Transaction language to VCI language, and VCI language to VC interface signals. Any implementation must set default values for the parameters, and they must be documented. The SLD/DWG defined data types are used for the parameters as follows:

- vsi_bit for binary parameters,
- vsi_int for integer parameters.

`Vsi_int CELLSIZE`:	VCI word length in bytes
`Vsi_int MAXPLEN`:	Maximum allowed packet length for the VCI in bytes
`Vsi_int ADDRSIZE`:	VCI address length in bytes
`Vsi_bit BIGENDIAN`:	Endian translation parameter.
`Vsi_bit NOENDIAN`:	no specified endianness
`char * VCITYPE`:	"PVCI", "BVCI", "AVCI"
`vsi_bit CHAINING`:	True if chaining VCI
`vsi_int CLENSIZE`:	Size of CLEN field
`vsi_int PLENSIZE`:	size of PLEN field
`vsi_int RFLAGSIZE`:	size of RFLAG field
`vsi_int WRLENSIZE`:	size of WRPLEN field
`vsi_bit DCFIXED`:	field cfixed is tied to High/Low
`vsi_bit DCONST`:	field const is tied to High/Low
`vsi_bit DCONTIG`:	field contig is tied to High/Low
`vsi_bit DWRAP`:	field wrap is tied to High/Low
`vsi_bit DEFINED`:	True, if support for defined transfer mode (defaulting command extensions)
`vsi_bit PUREPACKET`:	True, if non-incrementing packet model
`vsi_bit BVCIMODE`:	True, if in-order transactions are supported
`vsi_int TRDIDSIZE`:	Size of ThreadID signal
`vsi_int SRCIDSIZE`:	Size of SourceID signal
`vsi_int PKTIDSIZE`:	Size of PacketID signal

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

A.2.2 Transactions

The transactions are divided into two groups, initiator and target specific. Thus both the initiator and the target communication can be expressed in VCI Language. For example, a VCI monitor can write two trace files of transactions at a particular VC Interface, one for requests and one for responses. This division is there to facilitate expressing split transactions. The requests can be posted without waiting for response. This way, the split transactions can be written with a sequential language implementation. The request transactions also provide a way to wait for the response from the target. Thus a request call completes once the response arrives. To express split transactions with this mechanism requires that the language implementation can use parallel processes or threads.

A.2.2.1 Transaction Configuration

vciConfig, Sets the operation code parameters for the following transactions. If not used, default values are used.

A.2.2.2 Initiator Requests

vciWait, Wait for a specified number of clock cycles, there is no response for this
vciNop, Perform an empty request (No Operation), and wait for response
vciRead, "Read a cell" request
vciReadLock, "Read and lock a cell" request
vciWrite, "Write a cell" request

A.2.2.3 Target Responses

vciReadResp, Respond to a Read request
vciWriteResp, Respond to a Write request
vciNopResp, Respond to a Nop request

The following message sequence charts show examples how cell requests and responses can be ordered when used in the context of different VCI versions. The VCI Language does not have a concept of time, but it has a concept of event order. Each request and response causes an event (except the Wait). The requests can be the same in each of the examples, but the responses are different, depending on the target capabilities (split-transactions, out-of-order, error reporting).

PVCI does not support split transactions. Thus a response always follows a request in the correct order. The vciReadLock becomes vciRead with the PVCI, and vciNop becomes vciWait. See Figure 47.

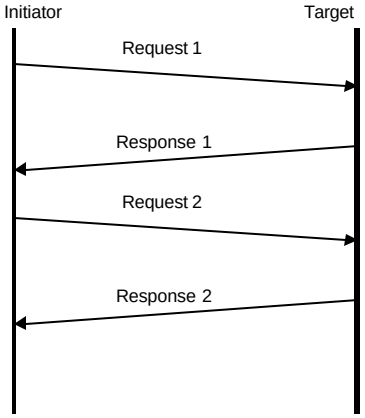


Figure 47: Transactions over PVCI

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

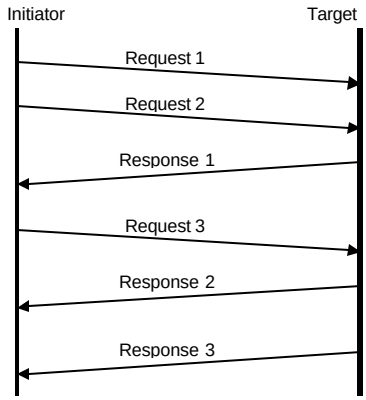


Figure 48: Transactions over BVCI

The BVCI supports split transactions, but not out-of-order responses. Thus multiple request cells, and even packets can be pipelined. See Figure 48.

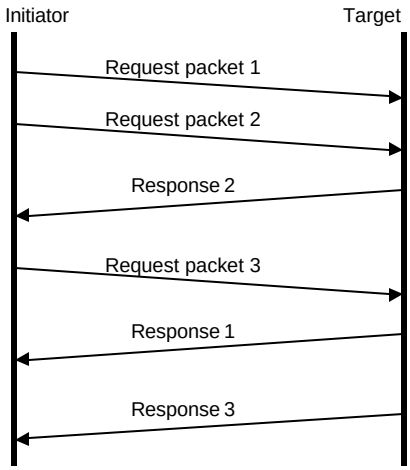


Figure 49: Transactions over AVCI

AVCI adds out-of-order transactions. The response packets may be in any order independent of the requests. The responses can be re-ordered based on source, thread, and packet identifiers. See Figure 49.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

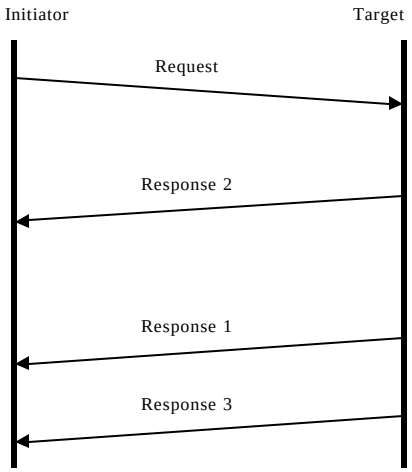


Figure 50: Transactions Over AVCI in Advanced Packet Model (One request per packet)

In the advanced packet model, Read transactions require only one request event per packet. The transaction language is still written with multiple requests, but only the first request is executed with the advanced packet model. See Figure 50.

A.2.3 Transaction Data Fields

Transaction data fields are necessary for performing a transaction. There are two kinds of data fields: fields that control the execution of the transaction, and fields that are sent to the target with a request or to the initiator with a response. The way this is done depends on the implementation. The most straightforward way is to convert the transaction with its data fields into VCI signals in a test bench. Some data fields are specific to different VCI levels, and are ignored in the BVCI and the PVCI.

The default value for each data field is 0. Depending on the language implementation, the unused values may or may not need to be given. (For example, when targeting the code to PVCI, some data fields are obsolete.) When using transactions written for a more complicated VCI with a simpler VCI (such as BVCI transactions with PVCI), only the performance should change.

A.2.3.1 Data Fields Used with vciRead and vciWrite

The description of each field below uses the following format:

name

(applicable interface standard, direction of field)

Description of the data field.

wrap
(AVCI, BVCI, sent to target)

This one-bit value defines how the addresses increment for packets with Contig asserted. WRAP = 0: addresses do not wrap, WRAP = 1: addresses wrap.

wraplen
(AVCI, sent to target)

If WRAP = 1, and the WRAPLEN > 0, the address wraps at the boundary indicated by WRAPLEN:

- Addresses larger or equal to 2 ^ WRAPLEN, will be mapped to Address modulo (2 ^ WRAPLEN).
- Max range: 2 ^ (2 ^ WRPLENSIZE).

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

const
(AVCI, BVCI, sent to target)

This one-bit value defines if the address changes during packet. CONST =1 means that the address is constant over packet.

plen
(AVCI, BVCI, sent to target)

This value gives the length of the packet in bytes. For BVCI, this must be in increments of natural word length in bytes in non-wrapped packets, and a power of two in wrapped packets. If the WRAPLEN = 0 and WRAP= 1, the address wraps as indicated by PLEN. (See Section 4.4, “BVCI Protocol.”)

cfixed
(AVCI, BVCI, sent to target)

This value indicates whether the opcode will change between packets in a chain. If CFIXED = 1 then the opcode does not change. If CFIXED = 0 then the opcode can be changed for each packet.

clen
(AVCI, BVCI, sent to target)

This value indicates how many packets are left in current packet chain. This counts down by each packet.

defined
(AVCI, sent to target)

This value indicates whether there is a relationship among addresses of each cell in the transaction that follows a predefined pattern, and is understood by both initiator and target. 1 = there is a relationship, 0= if there is not.

address
(AVCI, BVCI, PVCI, sent to target)

This 32-bit value indicates the location within the target being accessed. The address is byte-aligned, but notice that the target may ignore the lowest address bits. The byte addressing is encoded in the byte_enable.

byte_enable
(AVCI, BVCI, PVCI, sent to target)

Depending on the VCI size, this is a value up to 32 bits. It contains the byte enabling information for the access. The leftmost bit controls the lowest byte address of data, independent of the endianness.

wdata
(AVCI, BVCI, PVCI, sent to target)

The data to be written in the current cell is given in this field.

eop
(AVCI, BVCI, PVCI, sent to target)

This value is 1 at the last cell of the packet, otherwise it is 0.

edata
(AVCI, BVCI, PVCI, not sent to target)

For read accesses data, the expected data is given in this field. It can be used for comparing expected and returned data.

pktid
(AVCI, sent to target)

In a system where out-of-order transfer completion is supported, PKTID is required to identify which cells belong to a packet. This descriptor is not unique.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

trdid
(AVCI, sent to target)

TRDID is used in the case where a device is allowed to issue multiple outstanding transactions. It is a non-unique descriptor of a thread or a logical device.

srcid
(AVCI, sent to target)

Each initiator in a system is assigned a unique id, called SRCID.

A.2.3.2 Data Fields Used with vciNop

cycles
(AVCI, BVCI, PVCI, not sent to initiator)

This value gives the number of VCI clock cycles for the wait statement.

A.2.3.3 Data Fields Used with vciWriteResp, vciReadResp

reop
(AVCI, BVCI, PVCI, sent to initiator)

This is a single bit value of either 0 or 1. It indicates last cell of response packet.

rerror
(AVCI, BVCI, PVCI, sent to initiator)

The response error signal from target.

rflag
(AVCI, BVCI, PVCI, sent to initiator)

RFLAG is defined for supporting optional, user-defined response code (returned code). A specific response status can be returned to the initiator directly using one of these flags. In the case of a system where there are two different interpretations of user-defined flags, they will be mapped onto two different bits in the RFLAG[1:0]. This mapping is system specific.

rdata
(AVCI, BVCI, PVCI, sent to initiator)

This is the actual read data from the target.

rpktid
(AVCI, sent to initiator)

A copy of PKTID, which is returned along with the response. It helps identify the originating request packet, and reorders the packet if the system supports out-of-order transaction completion.

rtrdid
(AVCI, sent to initiator)

A copy of TRDID that is returned along with the response. It helps identify the thread of destination of the response, if the target is multi-threaded, and the request and response are arbitrated separately.

rsrcid
(AVCI, sent to initiator)

A copy of SRCID that is returned along with the response. It helps identify the originator of the transaction.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

A.2.4 Language Syntaxes

The following sections contain the language syntaxes.

A.2.4.1 File-Based VC Interface Language

This language is meant for generating stimulus for VCI test benches. Therefore, the language is divided into request and response parts. The test bench can read and interpret the request language from a file, and write the response language based on the target's behavior. The language can be generated automatically from the high-level language statements. The file-based VCI languages are case-independent. See the examples below for usage.

```
<syntax> ::= <comment> “\n” | <define> “\n” | <assignment> “\n” | <configure>
“\n>” | <request> “\n”| <response> “\n”
<comment> ::= “//” <string>
<define> := “#define” <unary-expression>[,<unary-expression>]+ …..
<identifier> :=<[R][0-9]+>
<unary-expression> := <“-”|“+”| “ ”><value>
<value> := <hex-val | bit-val | dec-val>
<hex-val> ::= “0x”[0-9a-fA-F]+
<dec-val> ::= [0-9]+
<bit-val> ::= “0” | “1”
<binary-expression> := <identifier|value><“+””-”|“*”|“/”|“&”|“|”><identifi-
er|value>
<assignment> := <identifier> “=” <binary-expression>

<configure> ::= “vciConfig” <opcode> “\n”
<opcode> := <defined> <contig> <const> <wrap> <cfixed> <plen> <clen> <wra-
plen> [<srcid> <trdid>]
<defined> ::= <bit-val> | <binary-expression> | <identifier>
<contig>  ::= <bit-val> | <binary-expression> | <identifier>
<const>  ::= <bit-val> | <binary-expression> | <identifier>
<wrap>  ::= <bit-val> | <binary-expression> | <identifier>
<cfixed>  ::= <bit-val> | <binary-expression> | <identifier>
<plen>  ::= <value> | <binary-expression> | <identifier>
<clen> ::= <value> | <binary-expression> | <identifier>
<wraplen> ::= <value> | <binary-expression> | <identifier>
<trdid>  ::= <value> | <binary-expression> | <identifier>
<srcid>  ::= <value> | <binary-expression> | <identifier>

<request> ::= <nop> “\n”
| <wait> “\n”
| <input> “\n”
| <output> “\n”
<nop> ::= “vciNop” address [<pktid>]
<wait> ::= “vciWait” [cycles]
<inputcmd> ::= “vciRead”| “vciReadLock”
<input> ::= <inputcmd> <address> <be> <eop> [<edata>  [<pktid>]]
<output> ::= “vciWrite" <address> <be> <eop> <wdata> [<pktid>]
<cycles> ::= <value>
<opcode> ::= <value>
<address> ::= <value> | <binary-expression> | <identifier>
<edata> ::= <value>  | <binary-expression> | <identifier>
<wdata> ::= <value>  | <binary-expression> | <identifier>
<be> ::= <value> | <binary-expression> | <identifier>
<eop> ::= <value> | <binary-expression> | <identifier>
<wraplen>  ::= <value> | <binary-expression> | <identifier>
<pktid> ::= <value> | <binary-expression> | <identifier>

<response> ::= <nop-response> “\n”
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

```
|<input-response> “\n”
| <output-response> “\n”
<nop-response> ::= “vciNopResp” [<rsrcid> <rpktid> <rtrdid>]
<input-response> ::= “vciReadResp” <rdata> <rerror> <rflag> <reop> [<rsrcid>
<rpktid> <rtdrid>]
<output-response> ::= “vciWriteResp” <rerror> <rflag> <reop> [<rsrcid> <rp-
ktid> <rtrdid>]
<rerror> ::= <value> | <binary-expression> | <identifier>
<rdata> ::= <value> | <binary-expression> | <identifier>
<rflag> ::= <value> | <binary-expression> | <identifier>
<reop> ::= <value> | <binary-expression> | <identifier>
<rpktid> ::= <value> | <binary-expression> | <identifier>
<rtrdid>  ::= <value> | <binary-expression> | <identifier>
<rsrcid>  ::= <value> | <binary-expression> | <identifier>
```

The interpreter shall have the following limitations:

- The #define statement assigns specific values to variables R0 through Rxxx, by the order of the listed values.
- There is a limit to the number of identifiers, which is not less than 128.
- There is a limit to the number of characters in an identifier, which is not more than 5.
- An identifier may have more than one #define statement.
- Identifiers keep their currently assigned values until reassigned.
- An assignment statement can also change the values of the registers, one register at a time.

This syntax allows the user to set and change address spaces, and calculate byte enables and expected values. This is useful to “relocate” the transactions from a standalone operation to apply them to different instances of the VC in a system.

If binary expressions and parameters are not used, the language becomes trivial to parse and interparty in a test bench, but non-portable. This style fits best for automatically generated code.

A.2.4.2 Embedded VC Interface Language

This syntax is meant for embedding VCI tests into a host language, such as C++, SystemC, or an HDL. The syntax is presented in C++, but is easily modified for other host languages. The parameter names may be abbreviated in case of clashes with reserved words. The function calls may also have other, implementation-specific parameters.

Control Commands

```
int vciWait (vsi_int cycles);
```

Return value: negative value on error
Cycles: VCI clock cycles to idle

Causes the initiator simulator to wait “cycles”-clock cycles. No traffic is generated into the VCI. This is not a NOP.

```
int vciConfig (vsi_bit contig, vsi_bit const, vsi_bit wrap, vsi_bit defined,
vsi_unsigned plen, vsi_bit cfixed, vsi_unsigned clen, vsi_unsigned wraplen[,
vsi_unsigned srcid, vsi_unsigned trdid]);
```

Return value: negative value on error
Contig: 1=address is contiguous, 0=address not contiguous
Const: 1= address is constant, 0=address not constant
Wrap: 1= address wraps, 0=address does not wrap
Plen: length of packet (0 = arbitrary length)
Cfixed: 1= fixed command over chain
Clen: number of packets in chain
Wraplen: Wrap length (0 = plen used for wrapping)
Defined: Predefined address pattern exists
Trdid: thread identifier, non-unique

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Srcid: source identifier, unique in the system

This configures opcode parameters for transactions. It can be called any time before a transaction call. If not called, default values (all 0) are assumed. This should be called before each packet, if the clen field is used, or any other packet header value changes.

Requests

The request calls can return data through the pointer parameters. It is not mandatory to implement return values, especially if the language is used to generate stimulus vectors.

```
int vciNop (vsi_unsigned address);
```
Return value: negative value on error
Address: target address

This is the No-Operation. This creates an empty packet of one cell in the Target Address. An empty packet means that there is no active command in the request, that is, only a handshake is performed. When mapping this call to PVCI, the handshake is not performed, and the command reverts to vciWait(1). This can be used to probe whether the target is alive.

```
int vciWrite(vsi_unsigned address, vsi_bit eop, vsi_bitvector be,
vsi_unsigned data [, vsi_unsigned * rerror, vsi_unsigned * rflag[,
vsi_unsigned pktid]]);
```
Return value: negative value on error in function call (not rerror)
Address: target address
Eop: end of packet signal
Be: byte enables, bit 0 is lowest address
Data: write data
Rerror: response error
Rflag: user-specified response flag
Pktid: packet identifier, non-unique

vciWrite writes one cell to the target.

```
int vciRead(vsi_unsigned address, vsi_bit eop, vsi_bitvector be,
vsi_unsigned edata[, vsi_unsigned * rdata,vsi_unsigned * rerror,
vsi_unsigned * rflag[, vsi_unsigned pktid]]);
```

Return value: negative value on error in function call (not rerror)
Address: target address
Eop: end of packet signal
Be: byte enables
Rdata: actual returned data
Edata: expected response data
Rerror: response error
Rflag: user-specified response flag
Pktid: packet identifier, non-unique

Read one cell from target.

Responses

These response calls provide a way to implement a behavioral interface model of a VCI target.

```
vciWriteResp(vsi_unsigned rerror, vsi_bit reop [, vsi_unsigned rflag[,
vsi_unsigned rsrcid, vsi_unsigned rpktid, vsi_unsigned rtrdid]]);
```
Rerror: response error (this will be returned by the requesting function)
Reop: response end of packet
Rflag: user-specified response flag
Rpktid: returned packet identifier, non-unique
Rtrdid: returned thread identifier, non-unique
Rsrcid: returned source identifier, unique in the system

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

This is the response to a write request. It is intended for the target side of the interface. REOP should be set on the last cell of the packet. Response identifiers can be used by the interconnect to route, and re-order the responses to the correct Initiators.

```
vciReadResp(vsi_unsigned rerror, vsi_bit reop, vsi_unsigned rdata [, vsi_unsigned rflag[, vsi_unsigned rsrcid, vsi_unsigned rpktid, vsi_unsigned rtrdid]]);
```

Rerror:	response error (this will be returned by the requesting function)
Reop:	response end of packet
Rdata:	actual returned data
Rflag:	user-specified response flag
Rpktid:	returned packet identifier, non-unique
Rtrdid:	returned thread identifier, non-unique
Rsrcid:	returned source identifier, unique in the system

The same as above, but for responding to a cell read. Rdata is the data returned to the read request.

```
vciNopResp (vsi_unsigned rerror [, vsi_unsigned rsrcid, vsi_unsigned rtrdid]);
```

Rerror:	response error (this will be returned by the requesting function)
Rtrdid:	returned thread identifier, non-unique
Rsrcid:	returned source identifier, unique in the system

This is the response from the target for a NOP. This creates an empty response packet of one cell back to the initiator side of the VCI. The rerror is returned to the requesting NOP.

A.2.5 VCI Language Examples

Examples with hard-coded parameters, no thread identifiers used:

STIM.TXT (The Request language file):

```
vciWait 10
vciConfig 0 0 0 0 0 0 0 0 0 0 // one cell, undefined packet length; this is the default
vciWrite 0x00000004 3  1 0x12345678
vciRead 0x00000004 3  1 0x12340000
vciNop 0x00000004
vciConfig 0 0 1 0 0 32 0 0 0 0 // constant address, 32-byte packet
vciWrite 0x00000004 F  0 0x03020100
vciWrite 0x00000004 F  0 0x07060504
vciWrite 0x00000004 F  0 0x0B0A0908
vciWrite 0x00000004 F  0 0x0F0E0D0C
vciWrite 0x00000004 F  0 0x13121110
vciWrite 0x00000004 F  0 0x17161514
vciWrite 0x00000004 F  0 0x1B1A1918
vciWrite 0x00000004 F  1 0x1F1E1D1C
vciRead 0x00000004 F  0 0x03020100
vciRead 0x00000004 F  0 0x07060504
vciRead 0x00000004 F  0 0x0B0A0908
vciRead 0x00000004 F  0 0x0F0E0D0C
vciRead 0x00000004 F  0 0x13121110
vciRead 0x00000004 F  0 0x17161514
vciRead 0x00000004 F  0 0x1B1A1918
vciRead 0x00000004 F  1 0x1F1E1D1C
vciConfig 0 1 0 0 0 32 0 0 0 0 // contiguous address, 32-byte packet
vciWrite 0x00000000 F  0 0x03020100
vciWrite 0x00000004 F  0 0x07060504
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

```
vciWrite 0x00000008 F  0 0x0B0A0908
vciWrite 0x0000000C F  0 0x0F0E0D0C
vciWrite 0x00000010 F  0 0x13121110
vciWrite 0x00000014 F  0 0x17161514
vciWrite 0x00000018 F  0 0x1B1A1918
vciWrite 0x0000001C F  1 0x1F1E1D1C
vciRead 0x00000000 F  0 0x03020100
vciRead 0x00000004 F  0 0x07060504
vciRead 0x00000008 F  0 0x0B0A0908
vciRead 0x0000000C F  0 0x0F0E0D0C
vciRead 0x00000010 F  0 0x13121110
vciRead 0x00000014 F  0 0x17161514
vciRead 0x00000018 F  0 0x1B1A1918
vciRead 0x0000001C F  1 0x1F1E1D1C
vciConfig 0 1 0 1 0 32 0 0 0 0 // contiguous address, wraps at plen, 32-byte
packet
vciWrite 0x00000110 F  0 0x03020100
vciWrite 0x00000114 F  0 0x07060504
vciWrite 0x00000118 F  0 0x0B0A0908
vciWrite 0x0000011C F  0 0x0F0E0D0C
vciWrite 0x00000100 F  0 0x13121110
vciWrite 0x00000104 F  0 0x17161514
vciWrite 0x00000108 F  0 0x1B1A1918
vciWrite 0x0000010C F  1 0x1F1E1D1C
vciRead 0x00000110 F  0 0x03020100
vciRead 0x00000114 F  0 0x07060504
vciRead 0x00000118 F  0 0x0B0A0908
vciRead 0x0000011C F  0 0x0F0E0D0C
vciRead 0x00000100 F  0 0x13121110
vciRead 0x00000104 F  0 0x17161514
vciRead 0x00000108 F  0 0x1B1A1918
vciRead 0x0000010C F  1 0x1F1E1D1C
```

The simulation results in the following response file:
RESP.TXT:

```
vciWriteResp  0 1
vciReadResp 0x12340000 0 1
vciNopResp
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 1
vciReadResp 0x03020100 0 0
vciReadResp 0x07060504 0 0
vciReadResp 0x0B0A0908 0 0
vciReadResp 0x0F0E0D0C 0 0
vciReadResp 0x13121110 0 0
vciReadResp 0x17161514 0 0
vciReadResp 0x1B1A1918 0 0
vciReadResp 0x1F1E1D1C 0 1
vciWriteResp  0 0
vciWriteResp  0 0
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

```
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 1
vciReadResp 0x03020100 0 0
vciReadResp 0x07060504 0 0
vciReadResp 0x0B0A0908 0 0
vciReadResp 0x0F0E0D0C 0 0
vciReadResp 0x13121110 0 0
vciReadResp 0x17161514 0 0
vciReadResp 0x1B1A1918 0 0
vciReadResp 0x1F1E1D1C 0 1
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 1
vciReadResp 0x03020100 0 0
vciReadResp 0x07060504 0 0
vciReadResp 0x0B0A0908 0 0
vciReadResp 0x0F0E0D0C 0 0
vciReadResp 0x13121110 0 0
vciReadResp 0x17161514 0 0
vciReadResp 0x1B1A1918 0 0
vciReadResp 0x1F1E1D1C 0 1
```

Following is a part of the previous example when used with the AVCI advanced packet model. While the parameter PUREPACKET is true, the interpretation of the request file is different. In Read requests, only the first line of the packet is interpreted, and the rest are skipped. (The EOP is also set in the interface.) In Write requests, only the control information in the first line of the packet is interpreted. The control information of the rest of the packet is ignored. The write data is used for each line. Notice that the code itself is usable with BVCI and PVCI. Only the interpretation with AVCI pure packet model is optimized.

STIM.TXT (The Request language file):

```
vciConfig 0 0 1 0 0 32 0 0 0 0 // constant address, 32-byte packet
vciWrite 0x00000004 F  0 0x03020100 //Use control info of this, set EOP = 1
vciWrite 0x00000004 F  0 0x07060504 //Use only data field of this
vciWrite 0x00000004 F  0 0x0B0A0908 //Use only data field of this
vciWrite 0x00000004 F  0 0x0F0E0D0C //Use only data field of this
vciWrite 0x00000004 F  0 0x13121110 //Use only data field of this
vciWrite 0x00000004 F  0 0x17161514 //Use only data field of this
vciWrite 0x00000004 F  0 0x1B1A1918 //Use only data field of this
vciWrite 0x00000004 F  1 0x1F1E1D1C //Use only data field of this
vciRead 0x00000004 F  0 0x03020100 // Use this, set EOP = 1
vciRead 0x00000004 F  0 0x07060504 // Skip this line
vciRead 0x00000004 F  0 0x0B0A0908 // Skip this line
vciRead 0x00000004 F  0 0x0F0E0D0C // Skip this line
vciRead 0x00000004 F  0 0x13121110 // Skip this line
vciRead 0x00000004 F  0 0x17161514 // Skip this line
vciRead 0x00000004 F  0 0x1B1A1918 // Skip this line
vciRead 0x00000004 F  1 0x1F1E1D1C // Skip this line
```

The simulation results in the following response file. The stimulus and response files are thus asymmetric.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

RESP.TXT:

```
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 0
vciWriteResp  0 1
vciReadResp 0x03020100 0 0
vciReadResp 0x07060504 0 0
vciReadResp 0x0B0A0908 0 0
vciReadResp 0x0F0E0D0C 0 0
vciReadResp 0x13121110 0 0
vciReadResp 0x17161514 0 0
vciReadResp 0x1B1A1918 0 0
vciReadResp 0x1F1E1D1C 0 1
```

A.2.6 Examples with Soft Parameters

```
#define 0x4 0x0 0x0 0x1F 0x110 0xFFC // assign values to variables R1 through
R6
vciConfig 0 0 0 0 0 0 0 0 0 0// one cell, undefined packet length; this is
the default
vciWait 10
vciWrite R0 3 1 R5
vciRead R0 3 1 R5
vciNop R0
vciConfig 0 0 1 0 0 32 0 0 0 0// constant address, 32-byte packet
vciWrite R0 F  0 0x03020100
vciWrite R0 F  0 0x07060504
vciWrite R0 F  0 0x0B0A0908
vciWrite R0 F  0 0x0F0E0D0C
vciWrite R0 F  0 0x13121110
vciWrite R0 F  0 0x17161514
vciWrite R0 F  0 0x1B1A1918
vciWrite R0 F  1 0x1F1E1D1C
vciRead R0 F  0 0x03020100
vciRead R0 F  0 0x07060504
vciRead R0 F  0 0x0B0A0908
vciRead R0 F  0 0x0F0E0D0C
vciRead R0 F  0 0x13121110
vciRead R0 F  0 0x17161514
vciRead R0 F  0 0x1B1A1918
vciRead R0 F  1 1F1E1D1C
vciConfig 0 1 0 0 0 32 0 0 0 0 // contiguous address, 32-byte packet
vciWrite R1 F  0 0x03020100
R1 = R1 + R0
vciWrite R1 F  0 0x07060504
R1 = R1 + R0
vciWrite R1 F  0 0x0B0A0908
R1 = R1 + R0
vciWrite R1 F  0 0x0F0E0D0C
R1 = R1 + R0
vciWrite R1 F  0 0x13121110
R1 = R1 + R0
vciWrite R1 F  0 0x17161514
R1 = R1 + R0
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

```
vciWrite R1 F  0 0x1B1A1918
R1 = R1 + R0
vciWrite R1 F  1 0x1F1E1D1C
R1 = R2
vciRead R1 F  0 0x03020100
R1 = R1 + R0
vciRead R1 F  0 0x07060504
R1 = R1 + R0
vciRead R1 F  0 0x0B0A0908
R1 = R1 + R0
vciRead R1 F  0 0x0F0E0D0C
R1 = R1 + R0
vciRead R1 F  0 0x13121110
R1 = R1 + R0
vciRead R1 F  0 0x17161514
R1 = R1 + R0
vciRead R1 F  0 0x1B1A1918
R1 = R1 + R0
vciRead R1 F  1 0x1F1E1D1C
R1 = R4 & R3
R4 = R4 | R3
vciConfig 0 0 0 0 0 32 0 0 0 0 // random address, 32-byte packet
vciWrite R1&R4 F  0 0x03020100
R1 = R1 + R0
vciWrite R1&R4 F  0 0x07060504
R1 = R1 + R0
vciWrite R1&R4 F  0 0x0B0A0908
R1 = R1 + R0
vciWrite R1&R4 F  0 0x0F0E0D0C
R1 = R1 + R0
vciWrite R1&R4 F  0 0x13121110
R1 = R1 + R0
vciWrite R1&R4 F  0 0x17161514
R1 = R1 + R0
vciWrite R1&R4 F  0 0x1B1A1918
R1 = R1 + R0
vciWrite R1&R4 F  1 0x1F1E1D1C
R1 = R1 + R0
vciRead R1&R4 F  0 0x03020100
R1 = R1 + R0
vciRead R1&R4 F  0 0x07060504
R1 = R1 + R0
vciRead R1&R4 F  0 0x0B0A0908
R1 = R1 + R0
vciRead R1&R4 F  0 0x0F0E0D0C
R1 = R1 + R0
vciRead R1&R4 F  0 0x13121110
R1 = R1 + R0
vciRead R1&R4 F  0 0x17161514
R1 = R1 + R0
vciRead R1&R4 F  0 0x1B1A1918
R1 = R1 + R0
vciRead R1&R4 F  1 0x1F1E1D1C
R1 = R5 & R3
R5 = R5 | R3
vciWrite R1&R5 F  0 0x03020100
R1 = R1 + R0
vciWrite R1&R5 F  0 0x07060504
R1 = R1 + R0
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

```
vciWrite R1&R5 F  0 0x0B0A0908
R1 = R1 + R0
vciWrite R1&R5 F  0 0x0F0E0D0C
R1 = R1 + R0
vciWrite R1&R5 F  0 0x13121110
R1 = R1 + R0
vciWrite R1&R5 F  0 0x17161514
R1 = R1 + R0
vciWrite R1&R5 F  0 0x1B1A1918
R1 = R1 + R0
vciWrite R1&R5 F  1 0x1F1E1D1C
R1 = R1 + R0
vciRead R1&R5 F  0 0x03020100
R1 = R1 + R0
vciRead R1&R5 F  0 0x07060504
R1 = R1 + R0
vciRead R1&R5 F  0 0x0B0A0908
R1 = R1 + R0
vciRead R1&R5 F  0 0x0F0E0D0C
R1 = R1 + R0
vciRead R1&R5 F  0 0x13121110
R1 = R1 + R0
vciRead R1&R5 F  0 0x17161514
R1 = R1 + R0
vciRead R1&R5 F  0 0x1B1A1918
R1 = R1 + R0
vciRead R1&R5 F  1 0x1F1E1D1C
```

A.3 High-Level Transaction Language

In the low-level language, the fields for all the signal groups in the VC Interface specification are included. As a result, there is one call for each of the cells of data being transferred. Furthermore, the size of the VC interface and the size of the packet that can be transferred must be known. Thus, the low-level language maps directly into a specific VCI implementation. In the High-Level Transaction Language, the transaction is not limited to the size of the cell or packet, or other parameters of a particular VC Interface implementation.

This language assumes that the users have minimal information about the VC interface. The user wants to read and write chunks of data (arbitrary-size data structures) to and from a target's address space. It is the language implementation's responsibility to know the parameters and limitations of the VCI implementation, and split the data into chains of packets of cells that fit the particular interface. This takes a lot of control away from the user, but adds great portability. The same sequence of transaction calls can be used with all the VCI levels (PVCI, BVCI, and AVCI). Only the efficiency of the physical transfer varies as the physical channel is changed.

The primary use for the language is in writing portable simulation stimulus. In stimulus use, the language implementation can either translate the High-Level Language into low-level language that the VCI test bench reads in (this makes implementing bi-directional difficult), or controls test bench's VCI signals directly. The latter means that the language implementation is embedded into the test-bench simulation model.

The language has two kinds of statements: channel control and data transfer. The channel control calls open a logical channel, or thread, in which the data transfer calls operate. This is introduced into the language to separate configuring physical parameters from actual data transfer, and to support better error checking and especially threads (introduced in Chapter 5, "Advanced VCI").

Only the prototypes of the calls are defined here. The implementation of these calls is not specified, but the input and output behavior is shown in the examples.

A.3.1 Interface Parameters

The parameters of the VCI that are implementation-constant must also be made visible to the language implementation. These parameters are not necessary at the transaction layer, but are used for mapping the transaction language to the VCI language.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

```
Vsi_int CELLSIZE: VCI word length in bytes
Vsi_int MAXPLEN:  maximum allowed packet length for the VCI in bytes
Vsi_int ADDRSIZE: VCI address length in bytes
Vsi_bit BIGENDIAN:endian translation parameter.
Vsi_bit NOENDIAN: no specified endianness
char * VCITYPE:   “PVCI,” “BVCI,” “AVCI”
vsi_bit CHAINING: true if chaining VCI
vsi_int CLENSIZE: size of CLEN field
vsi_int PLENSIZE: size of PLEN field
vsi_int RFLAGSIZE: size of RFLAG field
vsi_int WRLENSIZE: size of WRLEN field
vsi_bit DCFIXED:  field cfixed is tied to High/Low
vsi_bit DCONST:   field const is tied to High/Low
vsi_bit DCONTIG:  field contig is tied to High/Low
vsi_bit DWRAP:    field wrap is tied to High/Low
vsi_bit DEFD:     true, if support for defined transfer mode (defaulting com-
mand extensions)
vsi_bit EXERROR:  true, if support for extended error reporting
vsi_bit PUREPACKET: true, if non-incrementing packet model
vsi_bit BVCIMODE: true, if in-order transactions are supported
vsi_int TRDIDSIZE: size of ThreadID signal
vsi_int SRCIDSIZE: size of SourceID signal
vsi_int PKTIDSIZE: size of PacketID signal
```

A.3.2 Transactions

In contrast to the VCI language, there is no response language.

A.3.2.1 Transaction Configuration:

vciOpenChannel: Sets the VCI parameters for the following transactions in the opened channel. A channel corresponds to a target address space and a thread identifier.

vciCloseChannel: Closes a channel.

Data transfer:
vciTRead: “Read data”
vciTWrite: “Write data”

Figure 51 illustrates how higher level transactions relate to lower levels of timing and data abstraction. The high level transactions are non-atomic, as packet chains, whereas packets are atomic. In the figure, the two transactions overlap in time, since the packets are interleaved. VciTRead #1 is split into a chain of two packets in the packet layer. The transaction is completed as the last packet of a chain is processed. It is assumed that both of the transactions have the same priority in the implementation. The implementation schedules the first packet of the second transaction after the first packet of the first transaction. This interleaving can only happen at the packet level, since packets are atomic. The packets are mapped into cells in the cell layer. The cells can be out-of-order within a packet, as illustrated in the previous section. The transaction languages reside at the Transaction Layer and Cell Layer.

If the time order of transaction completion is modeled (for example, for modeling traffic in a bridge), the transactions must be parallel processes. If the transactions are unidirectional, this is not necessary to generate stimulus for simulation.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

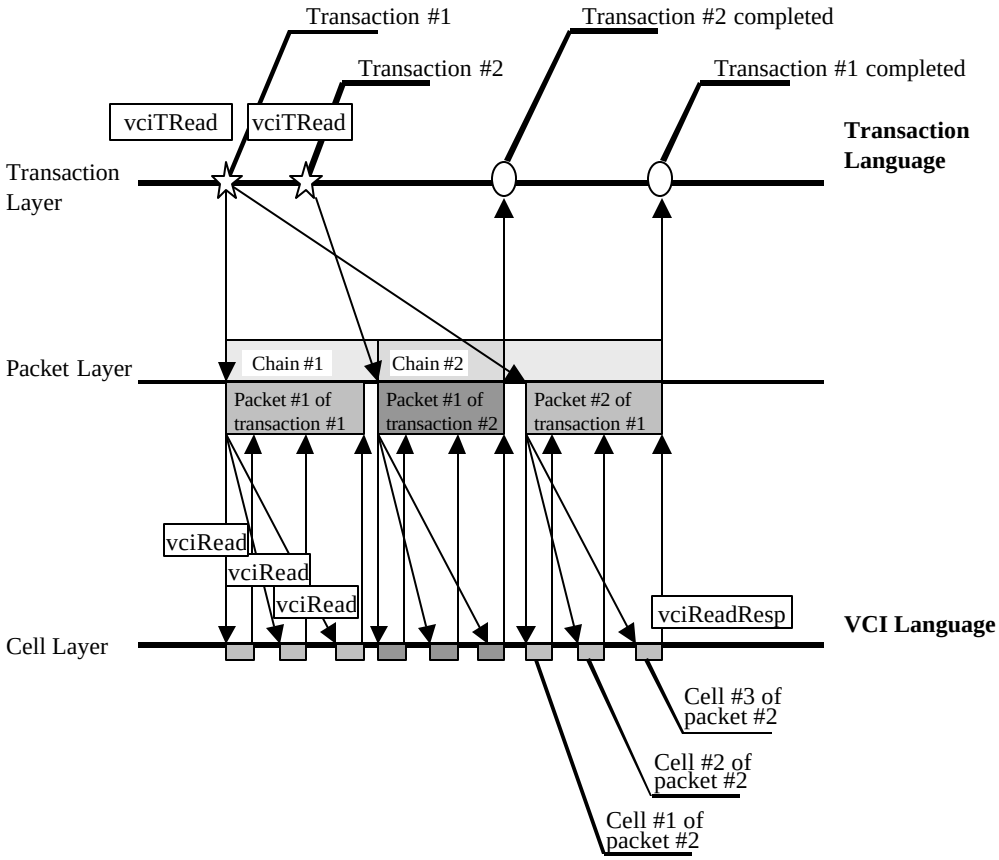


Figure 51: Normal Packet Model

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

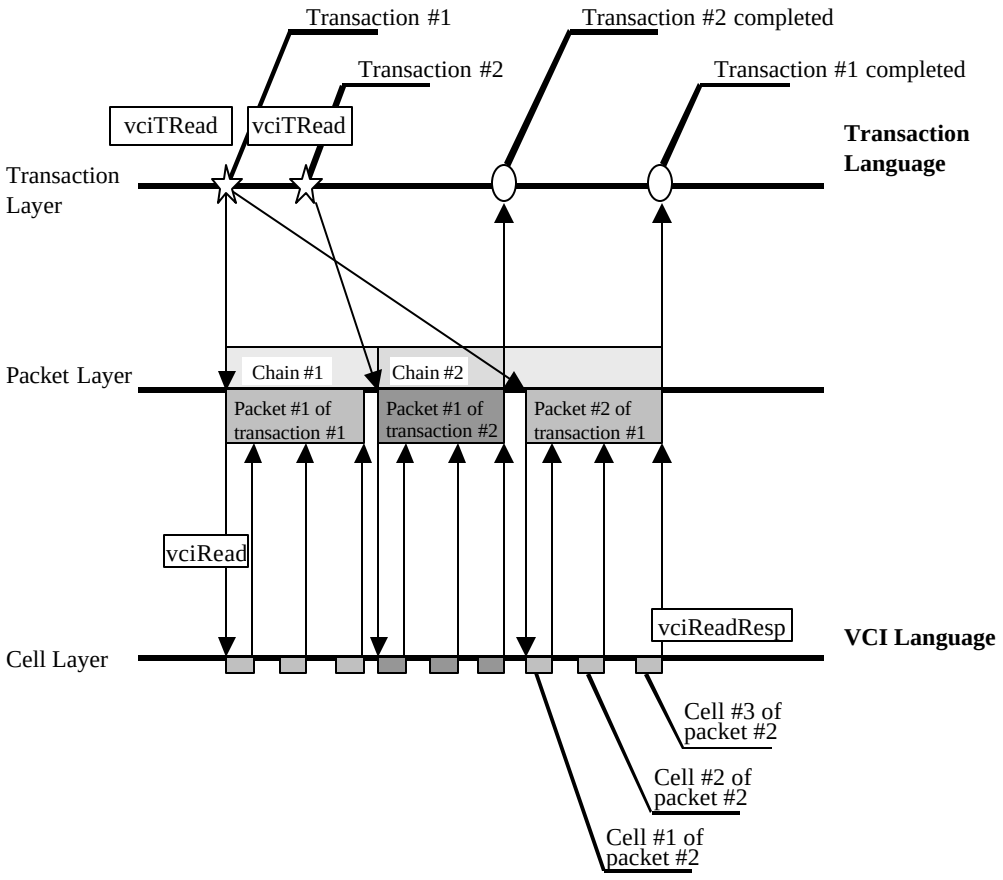


Figure 52: Advanced Packet Model

A.3.3 Transaction Parameters

These parameters are needed to perform a transaction. There are two kind of parameters: those that control the execution of the transaction, and those that are sent to the target with a request or to the initiator with a response. The way this is done depends on the implementation. The most straightforward way is to convert the transaction with its parameters into VCI signals in a test bench. Notice that in the transaction layer, most VCI parameters are not relevant, because they are used only for enhancing cell-layer performance. Therefore, the only parameters that are transported between the initiator and the target in the transaction layer are data, address, transaction identifier, and response. The rest are attributes to the communication channel, and are transferred only when mapping the transactions to the packet and cell layers

The default value for each parameter is 0. Depending on the language implementation, the unused values may or may not need to be given. (For example, when targeting the code to PVCI, some parameters are obsolete.) When using transactions written for a more complicated VCI with a simpler VCI (such as BVCI transactions with PVCI), only the performance should change, not the functionality.

Some parameters are set with the vciOpenChannel command, and some are given with the transaction calls.

A.3.3.1 Read/Write

wrap

This one-bit value defines if the address wraps or not.

Wrap = 0: addresses do not wrap, wrap = 1: addresses wrap.

wraplen

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

- Addresses larger or equal to 2 ^ wraplen are mapped to Address modulo (2 ^ wraplen).
- Max range: 2 ^ (2 ^ WRPLENSIZE).

If wraplen = 0, the wrapping boundary is calculated from the current packet length and the address, as explained in Section 4 of this document, “Basic VCI.”

const

This one-bit value defines if the address changes during packet. Const =1 means the address does not change. Notice that the contig is not visible at this level, since the target parameters are not known. It is set by the implementation.

defined

This value indicates whether there is relationship among addresses of each cell in the transaction that follows a predefined pattern, and is understood by both the initiator and the target. 1 = there is a relationship, 0= there is not.

plen

This value gives the maximum allowed length of the packet in bytes.

cmd

This character string indicates the mode of the opened channel: “R” = read mode, “W” = write mode, “RW” = read-write mode, “RLW” = read-locket write mode. For example, an attempt to write to a channel that has been opened read-only results in an error.

vlen

The number of bytes of valid data to read and write in an address stride. For example, Stride = 3, Vlen = 2 equals two active bytes followed by three inactive bytes.

stride

The number of bytes between strides. A stride means that the address and byte enable have a defined stepping pattern. This can be used with a defined or a random address mode.

baddress

Indicates the lowest address of the channel’s address space. This is a subspace within a target.

eaddress

Indicates the highest address of the channel’s address space. These addresses are for error detection only. In BVCI and PVCI, the channels cannot overlap. In AVCI, the channels may overlap if separated with a thread identifier.

address

(Sent to target)

This value indicates the location within the target being accessed. The address is aligned to words defined by number of data bytes in the interface. The byte addressing is encoded in the byte_enable, which is generated from the lowest address bits, vlen, stride, plen, wraplen, and defined fields.

wdata

(Sent to target)

The data to be written is given in this field.

trdid

(Sent to target)

The thread identifier. This enumerates the virtual Initiators. It is the implementation’s responsibility to assign correct source identifiers to each real initiator.

rerror

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

(Returned to initiator)

The response error signal from the target.

rdata

(Returned to the initiator)

This is the actual read data from the target.

A.3.4 Channel Control Calls

There can be a number of channels open at a time (positive vsi_int range), but they may not have any overlapping address spaces unless they are used with the AVCI. Channels are identified with an integer value, trdid. This can be assigned dynamically, or read from the configuration of the component. Storing the channel information is left to the implementation. The storage can be a dynamic linked list of data structures. Getting a pointer to this structure based on the integer identifier must be implemented within the function bodies. An example of the dynamic data structure to store the channel parameters follows:

```
typedef struct ch {
  vsi_int trdid;    // Thread identifier identifies the channel
  char cmd[4];      // “R”, “W”, “RW”, RLW”
  vsi_int plen;     // Maximum plen for channel. Must be < MAXPLEN
  vsi_bit wrap;     // address wraps
  vsi_bit wraplen;  // Wrap length
  vsi_bit const;    // Constant address
  vsi_bit defined;  // Defined address
  vsi_unsigned vlen;// The number of bytes to the next block of data to read/
write
  vsi_unsigned stride;// The number of bytes of valid data to read/write in
a stride
  vsi_int baddress; // Begin address of allocated channel
  vsi_int eaddress; // End address of allocated channel
  struct ch * next; // Pointer to next element in the linked list
} channel;
// This type stores pointers to the linked list
typedef struct {
  channel *start;
  channel *end;
} channel_list;
```

The Channel configuration calls are as follows:

VCI Open Channel

```
int vciOpen (vsi_int *command, vsi_bit const, vsi_bit defined, vsi_bit wrap,
vsi_unsigned plen, vsi_unsigned wraplen, vsi_unsigned baddress, vsi_unsigned
eaddress[, vsi_unsigned stride, vsi_unsigned Vlen ]).
```

Return value:	positive integer trdid, or negative error value.
Command:	null terminated array of char “R”,”W”, “RW”, “RLW”
Const:	1 = Address is constant, 0 = address is not constant
Defined:	1 = Address is defined, 0 = address is not defined
Wrap:	1 = address wraps, 0 = address does not wrap
Plen:	maximum packet length over the channel
Baddress:	start address of reserved address space
Eaddress:	End address of reserved address space
Wraplen:	wrap length (0 = plen used for wrapping)
Stride:	the number of bytes to the next block of data to read/write
Vlen:	the number of bytes of valid data to read/write in a stride

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

This command opens the channel. The first parameter is the type of transactions on the channel. R is read only, W is write only, RW is read/write, and RLW is read-lock-write. Baddress is the base address, and eaddress is the end or upper address for the channel. Vlen and stride determine the repeating be operation. Plen must be a multiple of stride, and Vlen is the number of bytes that are valid within a stride. The difference between stride and vlen is the number of bytes skipped at the end of a stride. Vlen bytes are valid starting at the first byte, and control the byte_enable signal and address sequence. If wrap = 1, then plen must be a power of 2, or wraplen > 0. The rest of the parameters are the same as in the VCI language. This returns the trdid or error for unable to open a channel. Possible errors include wrong plen, reserved address space, and so on.

VCI Close Channel

```
int vciClose(vsi_int trdid)
```

Return value: 0 on success in closing, negative on unsuccessful
trdid: channel to be closed

This command closes the channel. It returns the last error in the channel (positive value), or 0 if no error. Note that a negative error will occur if the channel is not opened for the type of operation being done.

A.3.5 Data Transfer Calls

```
int vciTWrite (vsi_int trdid, vsi_unsigned address vsi_unsigned tlen,
vsi_int8 *data)
```

Return value: last rerror
trdid: channel to be used
Address: start address of transfer
Tlen: size of data in bytes
Data: array of data to be written

This command writes a full transaction. It stops on the first error and returns the last rerror. Tlen is the amount of data to be transferred (sizeof(*data)). The implementation splits the transaction into a chain of packets. The address behavior in the resulting packets depends on the channel parameters. Access to random addresses must be done with separate calls. The data is packed.

```
int vciTRead (vsi_int trdid, vsi_unsigned address, vsi_int tlen,[vsi_int8
*rdata[,vsi_int8 *edata]])
```

Return value: last rerror
trdid: channel to be used
Address: start address of transfer
Tlen: size of data
Rdata: array of data read
Edata: array of expected data for debugging

This command reads a full transaction. It stops on the first error and returns the last rerror. Tlen is the amount of data to be transferred. If edata is provided, the count is to the first bad cell of data.

When invoked, these read and write commands may in turn call the simpler packet load and stores multiple times to complete the required transactions. The implementations select whether to use fixed or non-fixed packet chains based on tlen, CELLSIZE, address, and wrap, and how these parameter fit together. The primary objective is to lay the data transfer into the address space as the user intends it, based on address control channel parameters. As the other parameters allow, the transfer is then mapped to packet chains in the most efficient manner.

A.3.6 Transaction Language Examples

Example 1:

```
// In this case, the channel parameters are stored in a struct cmdstruct
channel cmdstruct;
int trdid1;
// Define a variable for a 32-byte data vector
vsi_int8 [32] datavec;
// Temporary variable for return values
int err;
/* Open a channel with max packet size 16 bytes, no wrapping, and allocated
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

```
address range from 0x100 to 0x1FF. */
trdid1 = vciOpen (cmdstruct, 0, 0, 0, 16, 0, 0x100, 0x1FF, 0, 0);
/* Write 32 bytes starting from address 0x100, in channel trdid1 */
err = vciTWrite (trdid1, 0x100, sizeof(datavec), datavec);
/* Close the channel */
err = vciClose(trdid1);
```

Same in VCI Language, mapped to a 32-bit AVCI

The source identifier given by the language implementation is “1” and the thread identifier is “0.” In a 32-bit AVCI, the transaction results in a packet chain of two 16-byte, contiguous, non-wrapped packets identified with the packet identifier. The thread identifier defaults to “0,” since there is only one thread. In a BVCI, the identifiers are ignored.

```
vciConfig 0 1 0 0 0 16 1 0 1 0// Configure first packet in the chain
vciWrite 0x00000100 0x15  0 0x12345678 0
vciWrite 0x00000104 0x15  0 0x9abcdef0 0
vciWrite 0x00000108 0x15  0 0x12345678 0
vciWrite 0x0000010C 0x15  1 0x9abcdef0 0
vciConfig 0 1 0 0 0 16 0 0 1 0// Configure the second packet in the chain
vciWrite 0x00000110 0x15  0 0x12345678 1
vciWrite 0x00000114 0x15  0 0x9abcdef0 1
vciWrite 0x00000118 0x15  0 0x12345678 1
vciWrite 0x0000011C 0x15  1 0x9abcdef0 1
```

Example 2:

```
// The channel parameters are again stored in a struct cmdstruct
channel cmdstruct;
int trdid1;
// Define a variable for a 16-byte data vector
vsi_int8 [16] datavec;
// Define a variable for a 16-byte read data vector
vsi_int8 [16] rdatavec;
// Temporary variable for return values
int err;
/* Open a channel with max packet size 16 bytes, no wrapping, and allocated
address range from 0x100 to 0x200. */
trdid1 = vciOpen (cmdstruct, 0, 0, 0, 16, 0, 0x100, 0x1FF, 0, 0);
/* Read 16 bytes starting from address 0x100, in channel trdid1 */
err = vciTRead (trdid1, 0x100, 16, rdatavec);
/* Write the same 16 bytes starting from address 0x120, in channel trdid1 */
err = vciTWrite (trdid1, 0x120, 16, datavec);
/* Close the channel */
err = vciClose(trdid1);
```

Same in VCI Language, mapped to a 32-bit AVCI

The source identifier given by the language implementation is “1” and the thread identifier is “0.” In a 32-bit AVCI, the transaction results in one 16-byte, contiguous, non-wrapped read packet and one write packet. The expected data is “0” since it is unknown, but the field is needed as a placeholder. The thread identifier defaults to “0”, since there is only one thread. In a BVCI, the identifiers are ignored.

```
vciConfig 0 1 0 0 0 16 0 0 1 0// Since the clen is 0 for both packets, this
call suffices
vciRead 0x00000100 0x15  0 0 0
vciRead 0x00000104 0x15  0 0 0
vciRead 0x00000108 0x15  0 0 0
vciRead 0x0000010C 0x15  1 0 0
vciWrite 0x00000120 0x15  0 0x12345678 1
vciWrite 0x00000124 0x15  0 0x9abcdef0 1
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

```
vciWrite 0x00000128 0x15  0 0x12345678 1
vciWrite 0x0000012C 0x15  1 0x9abcdef0 1
```

Example 3:

```
// The channel parameters are again stored in a struct cmdstruct
channel cmdstruct;
int trdid1;
// Define a variable for a 16-byte data vector
vsi_int8 [16] datavec;
// Temporary variable for return values
int err;
/* Open a channel with stride address, max packet size 16 bytes, no wrapping,
and allocated address range from 0x100 to 0x1FF. The stride is “write one
byte, skip three bytes”. Notice that the stride applies only to the target
address, not to the source datavec (which is “packed”) */
trdid1 = vciOpen (cmdstruct, 0, 0, 0, 16, 0, 0x100, 0x1FF, 3, 1);
/* Write the same 16 bytes starting from address 0x120, in channel trdid1 */
err = vciTWrite (trdid1, 0x120, 16, datavec);
/* Close the channel */
err = vciClose(trdid1);
```

Same in VCI Language, mapped to a 32-bit AVCI

The source identifier given by the language implementation is “1” and the thread identifier is “0.” In a 32-bit AVCI, the transaction results in a packet chain of four 16-byte, contiguous, non-wrapped write packets with byte enables adjusted to the stride. This is because the stride step is shorter than a cell. The packet size contains also “punctured” bytes.

```
vciConfig 0 1 0 0 0 16 3 0 1 0
vciWrite 0x00000100 0x1  0 0x12345678 1
vciWrite 0x00000104 0x1  0 0x9abcdef0 1
vciWrite 0x00000108 0x1  0 0x12345678 1
vciWrite 0x0000010C 0x1  1 0x9abcdef0 1
vciConfig 0 1 0 0 0 16 2 0 1 0
vciWrite 0x00000110 0x1  0 0x12345678 2
vciWrite 0x00000114 0x1  0 0x9abcdef0 2
vciWrite 0x00000118 0x1  0 0x12345678 2
vciWrite 0x0000011C 0x1  1 0x9abcdef0 2
vciConfig 0 1 0 0 0 16 1 0 1 0
vciWrite 0x00000120 0x1  0 0x12345678 3
vciWrite 0x00000124 0x1  0 0x9abcdef0 3
vciWrite 0x00000128 0x1  0 0x12345678 3
vciWrite 0x0000012C 0x1  1 0x9abcdef0 3
vciConfig 0 1 0 0 0 16 0 0
vciWrite 0x00000130 0x1  0 0x12345678 1 4 0
vciWrite 0x00000134 0x1  0 0x9abcdef0 1 4 0
vciWrite 0x00000138 0x1  0 0x12345678 1 4 0
vciWrite 0x0000013C 0x1  1 0x9abcdef0 1 4 0
```

Same in VCI Language, mapped to a 16-bit AVCI

The source identifier given by the language implementation is “1” and the thread identifier is “0”. In a 16-bit AVCI, the transaction results in a packet chain of four 16-byte, non-contiguous, non-wrapped write packets with byte enables and address adjusted to the stride. This is because the stride step is longer than a cell. The address pattern could also be defined, if the initiator and the target understand this stride.

```
vciConfig 0 0 0 0 0 16 3 0 1 0
vciWrite 0x00000100 0x1  0 0x12345678 1
vciWrite 0x00000104 0x1  0 0x9abcdef0 1
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved.

```
vciWrite 0x00000108 0x1  0 0x12345678 1
vciWrite 0x0000010C 0x1  1 0x9abcdef0 1
vciConfig 0 1 0 0 0 16 2 0 1 0
vciWrite 0x00000110 0x1  0 0x12345678 2
vciWrite 0x00000114 0x1  0 0x9abcdef0 2
vciWrite 0x00000118 0x1  0 0x12345678 2
vciWrite 0x0000011C 0x1  1 0x9abcdef0 2
vciConfig 0 1 0 0 0 16 1 0 1 0
vciWrite 0x00000120 0x1  0 0x12345678 3
vciWrite 0x00000124 0x1  0 0x9abcdef0 3
vciWrite 0x00000128 0x1  0 0x12345678 3
vciWrite 0x0000012C 0x1  1 0x9abcdef0 3
vciConfig 0 1 0 0 0 16 0 0 1 0
vciWrite 0x00000130 0x1  0 0x12345678 4
vciWrite 0x00000134 0x1  0 0x9abcdef0 4
vciWrite 0x00000138 0x1  0 0x12345678 4
vciWrite 0x0000013C 0x1  1 0x9abcdef0 4
```

Example 4:

```
// The channel parameters are again stored in a struct cmdstruct
channel cmdstruct;
int trdid1, trdid2;
// Define a variable for a 32-byte data vector
vsi_int8 [32] datavec;
// Temporary variable for return values
int err;
/* Open a channel with max packet size 16 bytes, no wrapping, and allocated
address range from 0x100 to 0x1FF. */
trdid1 = vciOpen (cmdstruct, 0, 0, 0, 16, 0, 0x100, 0x1FF, 0, 0);
/* Open a second channel with max packet size 16 bytes, no wrapping, and
allocated address range from 0x200 to 0x2FF. */
trdid2 = vciOpen (cmdstruct, 0, 0, 0, 16, 0, 0x200, 0x2FF, 0, 0);
/* Write 16 bytes starting from address 0x120, in channel trdid1 */
err = vciTWrite (trdid1, 0x120, 16, datavec);
/* Write 16 bytes starting from address 0x136, in channel trdid1 */
err = vciTWrite (trdid1, 0x136, 16, datavec);
/* Write 16 bytes starting from address 0x220, in channel trdid2 */
err = vciTWrite (trdid2, 0x220, 16, datavec);
/* Close the channels */
err = vciClose(trdid1);
err = vciClose(trdid2);
```

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT

B. VCI Frequently Asked Questions

Question: What's the difference between VSI and VCI? Aren't they the same things?

Answer: VSI is a set of all the necessary specifications needed to standardize the technical attributes that make a VC reusable. VCI is a logical interface for connecting VCs in a system. It is one of the standards, which together make up the VSI.

Question: Is it mandatory to support the Default ACK, especially about supporting the single cycle read/write operation?

Answer: The Default ACK behavior is optional, and indicated with the parameter DefACK.

Question: Can any signal be narrower than its defined size, with the upper bits tied to 0 (as explained for WDATA/RDATA in Section 3.3 of this document, "Signal Definitions")? Or is it restricted to data?

Answer: Any output signal can be less than the low-limit of signal width (if one exists) and tied to 0 or 1, depending on the required functionality.

Question: Will there be a time-out condition in any of these interfaces?

Answer: Time-out is a system feature. That is, if a target does not acknowledge a request within the time-out period, the initiator may remove the request.

Question: From timing diagrams, it seems like it is legal for transactions to complete in 2/3/4 cycles?

Answer: It is legal to complete in 1 to n cycles. One-cycle completion means either asynchronous acknowledge or default acknowledge. VCI does not give an upper limit to the number of transaction cycles.

Question: Non-contiguous bytes cannot be sent across the bus in one data packet. They have to be broken into chunks of contiguous bytes per data packet in a transaction. Is there anything wrong in this understanding?

Answer: Non-contiguous bytes can be sent in one packet, if non-existing bytes are "punctured" with byte enables.

Question: If we add a signal, and VCI-compliant initiators and targets are not expecting that signal, can we continue to be VCI compliant because we have the mandatory signals and documentation?

Answer: If your interface works in a VCI-compliant manner without using those extra signals, you would be compliant. That is, you are missing some performance by not using a signal that you have in the interface, but your system would nevertheless work.

Question: What if we want to increase the number of bits in the PVCI interface, such as using ADDR[40:0], DATA[127:0]?

Answer: Using signals that are wider than specified would not be compliant with the current VCI. Nothing prevents using proprietary versions of VCI, since you can map from a 256-bit interface to a 128-bit interface with a wrapper, or use an address decoder to access smaller address spaces.

Copyright 2000 - 2001 by the VSI Alliance, Inc.
All Rights Reserved. VSIA CONFIDENTIAL LICENSED DOCUMENT