Wishbone Interconnection
for Portable IP Cores
Specification Revision A

Silicore Corporation
3525 E. 27th Street, No. 301. Minneapolis, MN (USA) 55406
TEL: 612.722.3815 FAX: 612.722.5841
www.silicore.net

Electronic Design
Sensors · IP Cores

Preliminary Specification

This is a preliminary specification which is made available for public comment. The contents are subject to change. Do not specify or claim conformance to this document.

Stewardship

Stewardship for this specification is maintained by Silicore Corporation. Questions, comments and suggestions about this document are welcome and should be directed to:

Wade D. Peterson, Silicore Corporation
3525 E. 27th Street, Suite 301; Minneapolis, MN USA 55406
TEL: (612) 722-3815; FAX: (612) 722-5841
E-MAIL: peter299@maroon.tc.umn.edu URL: http://www.silicore.net

Silicore Corporation provides this document as a public service to its customers and to the IP core industry as a whole. The intent of this specification is to improve the quality of Silicore products, as well as to foster cooperation among the users and suppliers of IP cores. If you share this interest, and would be willing to participate in a recognized standards activity (ANSI, IEEE etc.), then please contact the steward.

Limited Copyright Release / Royalty Release / Patent Notice

This document is copyrighted by Silicore Corporation. However, this copyright is maintained so as to preserve the integrity of the specification, and is not an attempt to prevent the dissemination of the document nor to collect royalty fees for its use. It is the intention of Silicore Corporation to act as the steward for the specification until it can be transferred to an approved standards organization.

Notice is hereby given that this document may be freely copied and distributed by any means. However, this copyright release is limited to exact duplications of the document (including this page), with no revisions or changes of any kind. All other copyrights are reserved.

This specification may be used for product design and the production of parts or IP cores without any royalty obligations to Silicore Corporation.

The authors of this specification are not aware that the information contained herein, nor of products designed to the specification, cause infringement on the patent, copyright, trademark or trade secret rights of others. However, the possibility exists that such infringement may exist without their knowledge. The user of this document assumes all responsibility for determining if products designed to this specification infringe on the intellectual property rights of others.

Disclaimer

In no event shall Silicore Corporation be liable for incidental, consequential, indirect, or special damages resulting from the use of this specification. By adopting this specification, the user assumes all responsibility for its use or misuse.

Revision: A (preliminary); released: June 16, 1999

Table of Contents

Chapter 1 - Introduction

The Wishbone[1] interconnection is a portable and flexible interface for use with semiconductor IP cores. Its purpose is to foster design reuse by alleviating system-on-a-chip integration problems. This is accomplished by creating a common interface between IP cores. This improves the portability and reliability of the system, and results in faster time-to-market for the end user.

Previously, IP cores used non-standard interconnection schemes that made them difficult to integrate. This required the creation of custom glue logic to connect each of the cores together. By adopting a standard interconnection scheme, the cores can be integrated more quickly and easily by the end user.

This specification can be used for soft core, firm core or hard core IP. Since firm and hard cores are generally conceived as soft cores, the specification is written from that standpoint.

This specification does not require the use of specific development tools or target hardware. Furthermore, it is fully compliant with virtually all logic synthesis tools. However, some of the examples presented in the specification do use the VHDL hardware description language. These are presented only as a convenience to the reader, and should be readily understood by users of other hardware description languages (such as Verilog®). Schematic based entry tools can also be used.

The Wishbone interconnect is intended as a general purpose interface. As such, it defines the standard data exchange between IP core modules. It does not attempt to regulate the application-specific functions of the IP core.

The Wishbone architects were strongly influenced by two factors. First, there was a need for a good, reliable system-on-a-chip integration solution. Second, they were impressed by the traditional system integration solutions afforded by microcomputer buses such as PCI bus and VMEbus.

In fact, the Wishbone architecture is analogous to a microcomputer bus in that that they both: (a) offer a flexible integration solution that can be easily tailored to a specific application; (b) offer a variety of bus cycles and data path widths to solve various system problems; and (c) allow products to be designed by a variety of suppliers (thereby driving down price while improving performance and quality).

However, traditional microcomputer buses are fundamentally handicapped for use as a system-on-a-chip interconnection. That's because they are designed to drive long signal traces and connector systems which are highly inductive and capacitive. In this regard, system-on-a-chip is much simpler and faster. Furthermore, the system-on-a-chip solutions have a rich set of interconnection resources. These do not exist in microcomputer buses because they are limited by IC packaging and mechanical connectors.

The Wishbone architects have attempted to create a specification that is robust enough to insure complete compatibility between IP cores. However, it has not been over specified so as to unduly constrain the creativity of the core developer or the end user. It is believed that these two goals have been accomplished with the publication of this document.

The terminology used throughout this specification is defined in the Glossary at the end of this document.

---

[1] Webster's dictionary defines a wishbone as "the forked clavicle in front of the breastbone of most birds." The term 'Wishbone interconnect' was coined by Wade Peterson of Silicore Corporation. During the initial definition of the bus he was attempting to find a name that was descriptive of a bi-directional data bus that used either multiplexors or three-state logic. This was solved by forming an interface with separate input and output paths. When these paths were connected to three-state logic it formed a 'Y' shaped configuration that resembled a wishbone. The actual name was conceived during a Thanksgiving Day dinner that included roast turkey. Thanksgiving Day is a national holiday in the United States, and is observed on the third Thursday in November. It is generally celebrated with a traditional turkey dinner.

1.1 Wishbone Features

The Wishbone interconnection makes system-on-a-chip and design reuse easy by creating a standard data exchange protocol. Features of this technology include:

- Simple, compact, logical IP core hardware interfaces require very few logic gates.
- Full set of popular data transfer bus protocols including:

  - Single READ / WRITE cycle
  - BLOCK transfer cycle
  - RMW cycle
  - EVENT cycle

- Data bus widths[2] and operand sizes from 8 to 64-bits.
- Supports both BIG ENDIAN and LITTLE ENDIAN data ordering.
- Flexible interface supports memory mapped, FIFO memory and crossbar interconnections.
- Handshaking protocol allows each core to throttle data transfer speed.
- Up to one data transfer per clock cycle.
- Supports normal cycle termination, retry termination and termination due to error.
- Address widths[3] up to 64-bits.
- User-defined tag support. This is useful for identifying data transfers such as:

  - Data transfers
  - Interrupt vectors
  - Cache control operations

- MASTER / SLAVE architecture for very flexible system designs.
- Multiprocessing (multi-MASTER) capabilities. This allows for a wide variety of system-on-a-chip configurations, including:

  - Single MASTER / single SLAVE
  - Multiple MASTER / single SLAVE
  - Single MASTER / multiple SLAVE
  - Multiple MASTER / multiple SLAVE
  - Crossbar switches

- Arbitration methodology is defined by the end user (priority arbiter, round-robin arbiter, etc.).

[2] Specifications are given for data port and operand sizes up to 64-bits. However, the basic architecture can theoretically support any data width (e.g. 128-bit, 256-bit etc.).

[3] Specifications are given for address widths between zero (non-existent) and 64-bits. However, the basic architecture can theoretically support any address width.

- Supports various IP core interconnection means, including:

  - Unidirectional bus
  - Bi-directional bus
  - Multiplexor based interconnections
  - Three-state based interconnections
  - Off chip I/O

- Synchronous design assures portability, simple design and ease of test.

- Very simple timing specification.

- Documentation requirements allow the end user to quickly evaluate interface needs.

- Independent of hardware technology (FPGA, ASIC, etc.).

- Independent of delivery method (soft, firm or hard core).

- Independent of synthesis tool, router and layout tool technology.

1.2 Wishbone Objectives

The main objective of the specification is to create a flexible interconnection means for use with semiconductor IP cores. This allows various IP core modules to be connected together to form a system-on-a-chip.

A further objective of the specification is to enforce good compatibility between IP core modules. This enhances design reuse.

A further objective of the specification is to create a robust standard, but one that does not unduly constrain the creativity of the core developer or the end user.

A further objective of the specification is to make it easy to understand by both the core developer and the end user.

A further objective of the specification is create a portable interface that is independent of the underlying semiconductor technology. For example, the interconnect must be capable of working with both FPGA and ASIC hardware target devices.

A further objective of the specification is to make the interface independent of logic signaling levels.

A further objective of the specification is to create a flexible interconnection scheme that is independent of the IP core delivery method. For example, it may be used with ‘soft core’, ‘firm core’ or ‘hard core’ delivery methods.

A further objective of the specification is to be independent of the underlying hardware description. For example, soft cores may be written and synthesized in VHDL, Verilog® or some other hardware description language. Schematic entry may also be used.

A further objective of the specification is to require a minimum standard for documentation. This allows IP core users to quickly evaluate and integrate new cores.

A further objective of the specification is to eliminate extensive interface documentation on the part of the IP core developer. In most cases, this specification along with the WISHBONE DATASHEET is sufficient to completely document an IP core interface.

A further objective is to create an architecture that has a smooth transition path to support new technologies. This increases the longevity of the specification as it can adapt to new, and as yet un-thought-of, requirements.

A further objective is to create an architecture that allows various interconnection means between IP core modules. This insures that the end user can tailor the system-on-a-chip to his/her own needs.

A further objective is to create an architecture that requires a minimum of glue logic by the end user. In some cases the system-on-a-chip needs no glue logic whatsoever. However, in other cases the end user may choose to use a more sophisticated interconnection method (for example with FIFO memories or crossbar switches) that requires additional glue logic.

A further objective is to create an architecture with variable address and data path widths to meet a wide variety of system requirements.

A further objective is to create an architecture that supports both BIG ENDIAN and LITTLE ENDIAN data transfer organizations.

A further objective is to create an architecture that supports one data transfer per clock cycle.

A further objective is to create an architecture that allows data to be tagged. This allows the purpose for each bus cycle to be identified by a SLAVE. For example, in microprocessor based systems it is often necessary to discriminate between data transfer, interrupt acknowledge and caching operations.

A further objective is to create an architecture with a MASTER/SLAVE topology. Furthermore, the system must be capable of supporting multiple MASTERs and multiple SLAVEs with an efficient arbitration mechanism.

A further objective is to create an architecture that supports crossbar switches.

A further objective is to create a synchronous protocol to insure ease of use, good reliability and easy testing. Furthermore, all transactions can be coordinated by a single clock.

A further objective is to create a synchronous protocol that works over a wide range of interface clock speeds. The effects of this are: (a) that the Wishbone interface can work synchronously with all attached IP cores, (b) that the interface can be used on a large range of target devices, (c) that the timing specification is much simpler and (d) that the resulting semiconductor device is much more testable.

A further objective is to create a synchronous protocol that provides a simple timing specification. This makes the interface very easy to integrate.

A further objective is to create a synchronous protocol where each MASTER and SLAVE can throttle the data transfer rate with a handshaking mechanism.

A further objective is to create a synchronous protocol where data may be transferred through memory mapped, FIFO memory or crossbar switch interconnections.

A further objective is to create a synchronous protocol that is optimized for system-on-a-chip, but that is also suitable for off-chip I/O routing. Generally, the off-chip Wishbone interconnect will operate at slower speeds.

1.3 Specification Terminology

To avoid confusion, and to clarify the requirements for compliance, this specification makes use of five keywords to define the operation of the Wishbone interconnect. The keywords are:

- **RULE**
- **RECOMMENDATION**
- **SUGGESTION**
- **PERMISSION**
- **OBSERVATION**

Any text not labeled with one of these keywords describes the operation in a narrative style. The keywords are defined as follows:

RULE
Rules form the basic framework of the specification. They are sometimes expressed in text form and sometimes in the form of figures, tables or drawings. All rules MUST be followed to ensure compatibility between interfaces. Rules are characterized by an imperative style. The upper-case words MUST and MUST NOT are reserved exclusively for stating rules in this document, and are not used for any other purpose.

RECOMMENDATION
Whenever a recommendation appears, designers would be wise to take the advice given. Doing otherwise might result in some awkward problems or poor performance. While this specification has been designed to support high performance systems, it is possible to create an interconnection that complies with all the rules, but has very poor performance. In many cases a designer needs a certain level of experience with the system architecture in order to design interfaces that deliver top performance. Recommendations found in this document are based on this kind of experience and are provided as guidance for the user.

SUGGESTION
A suggestion contains advice which is helpful but not vital. The reader is encouraged to consider the advice before discarding it. Some design decisions are difficult until experience has been gained. Suggestions help a designer who has not yet gained this experience. Some suggestions have to do with designing compatible interconnections, or with making system integration easier.

PERMISSION
In some cases a rule does not specifically prohibit a certain design approach, but the reader might be left wondering whether that approach might violate the spirit of the rule, or whether it might lead to some subtle problem. Permissions reassure the reader that a certain approach is acceptable and will not cause problems. The upper-case word MAY is reserved exclusively for stating a permission and is not used for any other purpose.

OBSERVATION
Observations do not offer any specific advice. They usually clarify what has just been discussed. They spell out the implications of certain rules and bring attention to things that might otherwise be overlooked. They also give the rationale behind certain rules, so that the reader understands why the rule must be followed.

1.4 Use of Timing Diagrams

Figure 1-1 shows some of the key features of the timing diagrams in this specification. Unless otherwise noted, the MASTER signal names referenced in the timing diagrams. In some cases the MASTER and SLAVE signal names are different. For example, in the single MASTER / single SLAVE configuration, the [ADR_O] and [ADR_I] signals are connected together. Furthermore, the actual waveforms at the SLAVE may vary from those at the MASTER. That's because the MASTER and SLAVE interfaces can be connected together in different ways. In all cases, the timing diagrams show how the two interfaces are connected together.

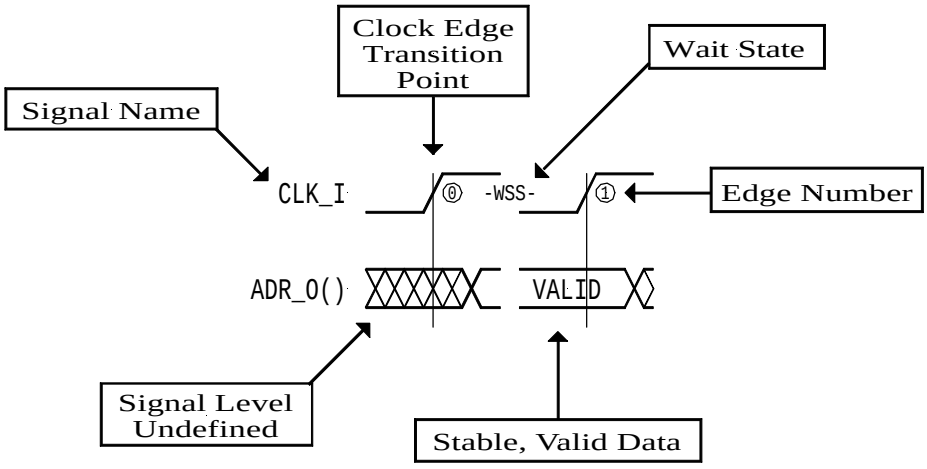


Figure 1-1. Use of timing diagrams.

Individual signals may or may not be present on an specific interface. That's because most of the signals are optional.

Two symbols are also presented in relation to the [CLK_I] signal. These include the positive going clock edge transition point and the clock edge number. In most diagrams a vertical guideline is shown at the positive-going edge of each [CLK_I] transition. This represents the theoretical transition point at which flip-flops register their input value, and transfer it to their output. The exact level of this transition point varies depending upon the technology used in the target device. The clock edge number is included as a convenience so that specific points in the timing diagram may be referenced in the text. The clock edge number in one timing diagram is not related to the clock edge number in another diagram.

Gaps in the timing waveforms may be shown. These indicate either: (a) a wait state or (b) a portion of the waveform that is not of interest (in the context of the diagram). When the gap indicates a wait state, the symbols '-WSM-' or '-WSS-' are placed in the gap along the [CLK_I] waveform. These correspond to wait states inserted by the MASTER or SLAVE interfaces.

Undefined signal levels are indicated by a hatched region. In MASTER interfaces, this region indicates that the signal level is undefined, and may take any state. In SLAVE interfaces, this region indicates that the current state is undefined, and should not be relied upon. When signal arrays are used, stable and predictable signal levels are indicated with the word 'VALID'. Non-array signals show a steady high or low state.

1.5 Signal Naming Conventions

All signal names used in this specification have the '_I' or '_O' characters attached to them. These indicate if the signals are an input (to the core) or an output (from the core). For example, [ACK_I] is an input and [ACK_O] is an output. This convention is used to clearly identify the direction of each signal.

Signal arrays are identified by a signal name followed by the array boundaries in parenthesis. For example, [DAT_I(63..0)] is a signal array with upper array boundary number sixty-three, and lower array boundary number zero. Furthermore, the array boundaries indicate the full range of the permissible array size. The array size on any particular core may vary. In many cases the array boundaries are omitted if they are irrelevant to the context of the description.

When used as part of a sentence, signal names are enclosed in brackets '[ ]'. This helps to discriminate signal names from the words in the sentence.

Chapter 2 – Interface Specification

This chapter describes the signaling method between MASTER and SLAVE modules. This includes numerous options which may or may not be present on a particular interface. Furthermore, it describes a minimum level of required documentation that must be created for each IP core.

2.1 Required Documentation

Documentation must be provided for each IP core with a Wishbone interconnect. This helps the end user understand the operation of the core, and how to connect it to other cores. The documentation takes the form of a WISHBONE DATASHEET. This can be included in a technical reference manual for the IP core.

RULE 2.10
Each Wishbone compatible IP core MUST include a WISHBONE DATASHEET as part of the IP core documentation.

RULE 2.20
The WISHBONE DATASHEET MUST include the signal names that are defined for the IP core. Furthermore, each core-specific signal name MUST be cross-referenced to the signal name used in this specification.

PERMISSION 2.10
Any signal name MAY be used to describe each of the Wishbone signals.

RULE 2.30
Signals MUST be named in accordance with the rules of the native tool in which the IP core was designed.

OBSERVATION 2.10
Most hardware description languages (such as VHDL or Verilog®) have naming conventions. For example, the VHDL hardware description language defines the alphanumeric symbols which may be used. Furthermore, it states that UPPERCASE and LOWERCASE characters may be used in a signal name.

SUGGESTION 2.10
It is recommended that the interface use the signal names that are defined in this document.

OBSERVATION 2.20
Core integration is simplified if the signal names match those given in this specification. However, in some cases (such as IP cores with multiple Wishbone interconnects) they cannot be used. The use of non-standard signal names will not result in any serious integration problems since all hardware description tools allow signals to be renamed.

RULE 2.40
All Wishbone interface signals MUST use active high logic.

2.2 Wishbone Signal Description

This section describes the signals used in the Wishbone interconnect. Some of these signals are optional, and may or may not be present on a specific interface.

2.3.1 Signals Common to MASTER and SLAVE Interfaces

CLK_I
The clock input [CLK_I] coordinates all activities for the internal logic within the Wishbone interconnect. All Wishbone output signals are registered at the rising edge of [CLK_I]. All Wishbone input signals must be stable before the rising edge of [CLK_I].

RST_I
The reset input [RST_I] forces the Wishbone interface to restart. Furthermore, all internal self-starting state machines will be forced into an initial state.

2.3.2 MASTER Signals

ACK_I
The acknowledge input [ACK_I], when asserted, indicates the termination of a normal bus cycle. Also see the [ERR_I] and [RTY_I] signal descriptions.

ADR_O(63..0)
The address output array [ADR_O(63..0)] is used to pass a binary address, with the most significant address bit at the higher numbered end of the signal array. The lower array boundary is specific to the data port size. The higher array boundary is core-specific. In some cases (such as FIFO interfaces) the array may not be present on the interface.

CYC_O
The cycle output [CYC_O], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_O] signal is asserted during the first data transfer, and remains asserted until the last data transfer. The [CYC_O] signal is useful for interfaces with multi-port interfaces (such as dual port memories). In these cases, the [CYC_O] signal requests use of a common bus from an arbiter. Once the arbiter grants the bus to the MASTER, it is held until [CYC_O] is negated.

DAT_I(63..0)
The data input array [DAT_I(63..0)] is used to pass binary data. The array boundaries are determined by the port size. Also see the [DAT_O(63..0)] and [SEL_O(7..0)] signal descriptions.

DAT_O(63..0)
The data output array [DAT_O(63..0)] is used to pass binary data. The array boundaries are determined by the port size. Also see the [DAT_I(63..0)] and [SEL_O(7..0)] signal descriptions.

ERR_I
The error input [ERR_I] indicates an abnormal cycle termination. The source of the error, and the response generated by the MASTER is defined by the IP core supplier in the WISHBONE DATASHEET. Also see the [ACK_I] and [RTY_I] signal descriptions.

RTY_I
The retry input [RTY_I] indicates that the indicates that the interface is not ready to accept or send data, and that the cycle should be retried. When and how the cycle is retried is defined by the IP core supplier in the WISHBONE DATASHEET. Also see the [ERR_I] and [RTY_I] signal descriptions.

SEL_O(7..0)
The select output array [SEL_O(7..0)] indicates where valid data is expected on the [DAT_I(63..0)] signal array during READ cycles, and where it is placed on the [DAT_O(63..0)] signal array during WRITE cycles. Also see the [DAT_I(63..0)], [DAT_O(63..0)] and [STB_O] signal descriptions.

STB_O
The strobe output [STB_O] indicates a valid data transfer cycle. It is used to qualify various other signals on the interface such as [SEL_O(7..0)]. The SLAVE must assert either the [ACK_I], [ERR_I] or [RTY_I] signals in response to every assertion of the [STB_O] signal.

TAGN_O
The tag output(s) [TAGN_O] can be used to indicate the type of data transfer in progress. Furthermore, 'N' in this signal name refers to a tag number because multiple tags may be used. For example, [TAG1_O] may indicate a valid data transfer cycle, [TAG2_O] may indicate an interrupt acknowledge cycle and so on. The exact meaning of each tag is defined by the IP core provider in the WISHBONE DATASHEET.

WE_O
The write enable output [WE_O] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

2.3.3 SLAVE Signals

ACK_O
The acknowledge output [ACK_O], when asserted, indicates the termination of a normal bus cycle. Also see the [ERR_O] and [RTY_O] signal descriptions.

ADR_I(63..0)
The address input array [ADR_I(63..0)] is used to pass a binary address, with the most significant address bit at the higher numbered end of the signal array. The lower array boundary is specific to the data port size. The higher array boundary is core-specific. In some cases (such as FIFO interfaces) the array may not be present on the interface.

CYC_I
The cycle input [CYC_I], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_I] signal is asserted during the first data transfer, and remains asserted until the last data transfer.

DAT_I(63..0)
The data input array [DAT_I(63..0)] is used to pass binary data. The array boundaries are determined by the port size. Also see the [DAT_O(63..0)] and [SEL_O(7..0)] signal descriptions.

DAT_O(63..0)
The data output array [DAT_O(63..0)] is used to pass binary data. The array boundaries are determined by the port size. Also see the [DAT_I(63..0)] and [SEL_O(7..0)] signal descriptions.

ERR_O
The error output [ERR_O] indicates an abnormal cycle termination. The source of the error, and the response generated by the MASTER is defined by the IP core supplier in the WISHBONE DATASHEET. Also see the [ACK_O] and [RTY_O] signal descriptions.

RTY_O
The retry output [RTY_O] indicates that the indicates that the interface is not ready to accept or send data, and that the cycle should be retried. When and how the cycle is retried is defined by the IP core supplier in the WISHBONE DATASHEET. Also see the [ERR_O] and [RTY_O] signal descriptions.

SEL_I(7..0)
The select input array [SEL_I(7..0)] indicates where valid data is placed on the [DAT_I(63..0)] signal array during WRITE cycles, and where it should be present on the [DAT_O(63..0)] signal array during READ cycles. Also see the [DAT_I(63..0)], [DAT_O(63..0)] and [STB_I] signal descriptions.

STB_I
The strobe input [STB_I] indicates a valid data transfer cycle. It is used to qualify various other signals on the interface such as [SEL_I(7..0)]. The SLAVE must assert either the [ACK_O], [ERR_O] or [RTY_O] signals in response to every assertion of the [STB_I] signal.

WE_I
The write enable input [WE_I] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

Chapter 3 – Bus Interface

The bus interface is described in terms of its general operation, reset operation, handshaking protocol, bus cycles and the data organization during transfers. Additional requirements for the bus interface (especially those relating to [CLK_I]) can be found in the timing specifications in Chapter 4.

3.1 General Operation

Each MASTER and SLAVE are interconnected with a set of signals that permit them to exchange data. For descriptive purposes this interconnection is called a *bus*. Address, data and other information is impressed upon this bus in the form of *bus cycles*.

3.1.1 Reset Operation

All hardware must be initialized to a pre-defined state. This is accomplished with the reset signal [RST_I]. This signal can be asserted at anytime. It is also used for test simulation purposes by initializing all self-starting state machines and counters which may be used in the interface.

RULE 3.10
MASTER and SLAVE interfaces MUST initialize themselves after the assertion of [RST_I].

RULE 3.20
[RST_I] MUST be asserted for at least one [CLK_I] cycle.

RULE 3.30
The interface MUST be capable of reacting to [RST_I] at any time.

RULE 3.40
Self-starting state machines and counters on the Wishbone interface MUST initialize themselves to a pre-defined state after the assertion of [RST_I].

OBSERVATION 3.10
In general, a self-starting state machine does not need to be initialized. However, this may cause problems because some simulators may not be sophisticated enough to find an initial starting point for the state machine. The initialization rule prevents this problem by forcing the state machine to a pre-defined state.

RULE 3.50
The following MASTER and SLAVE signals MUST be negated after the assertion of [RST_I]: [STB_O], [CYC_O], [ACK_O], [ERR_O] and [RTY_O]. The state of all other signals is undefined.

3.1.2 Handshaking Protocol

All bus cycles use a handshaking protocol between the MASTER and SLAVE interfaces. As shown in Figure 3-1, the MASTER asserts [STB_O] when it is ready to transfer data. [STB_O] remains asserted until the SLAVE asserts one of the cycle terminating signals [ACK_I], [ERR_I] or [RTY_I]. At every rising edge of [CLK_I] the terminating signal is sampled. If it is asserted, then [STB_O] is negated. Both

sides of the interface can then completely control the rate at which data is transferred. If the SLAVE always operates at the maximum speed of the core, and if the [ERR_I] and [RTY_I] signals are not used, then the [ACK_I] signal may be tied 'high'. The interface will function normally under these circumstances.

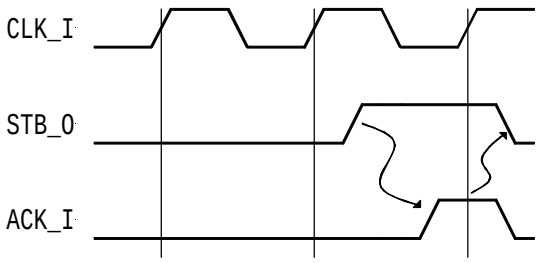


Figure 3-1. Local bus handshaking protocol.

Most of the examples in the specification describe the use of [ACK_I] to terminate a local bus cycle. However, the SLAVE can optionally terminate the cycle with an error [ERR_O], or request that the cycle be retried [RTY_O].

All interfaces include the [ACK_I] terminator signal. Asserting this signal during a bus cycle causes it to terminate normally.

Asserting the [ERR_I] signal during a bus cycle will terminate the cycle. It also serves to notify the MASTER that an error occurred during the cycle. This signal is generally used if an error was detected by SLAVE logic circuitry. For example, if the SLAVE is a parity-protected memory, then the [ERR_I] signal can be asserted if a parity fault is detected. This specification does not dictate what the MASTER will do in response to [ERR_I].

Asserting the optional [RTY_I] signal during a bus cycle will terminate the cycle. It also serves to notify the MASTER that the current cycle should be aborted, and retried at a later time. This signal is generally used for shared memory and bus bridges. In these cases SLAVE circuitry would assert [RTY_I] if the local resource is busy. This specification does not dictate when or how the MASTER will respond to [RTY_I].

The simplest form of the Wishbone interconnect is the EVENT cycle. In this case only the handshaking signals are present. To indicate an event the MASTER asserts [STB_O], and the slave reacts by asserting [ACK_O].

RULE 3.60
As a minimum, the MASTER interface MUST include the following signals: [ACK_I], [CLK_I], [CYC_O], [RST_I] and [STB_O]. As a minimum, the SLAVE interface MUST include the following signals: [ACK_O], [CLK_I] and [RST_I]. All other signals are optional.

PERMISSION 3.10
MASTER and SLAVE interfaces MAY be designed to support the [ERR_I] and [ERR_O] signals. In these cases, the SLAVE asserts [ERR_O] to indicate that a pre-defined error has occurred during the bus cycle. This specification does not dictate what the MASTER will do in response to [ERR_I].

RULE 3.70
If a MASTER supports the [ERR_I] signal, then the WISHBONE DATASHEET MUST describe how it reacts in response to the signal. If a SLAVE supports the [ERR_O] signal, then the WISHBONE DATASHEET MUST describe the conditions under which the signal is generated.

PERMISSION 3.20
MASTER and SLAVE interfaces MAY be designed to support the [RTY_I] and [RTY_O] signals. In these cases, the SLAVE asserts [RTY_O] to indicate that the interface is busy, and that the bus cycle should be retried at a later time. This specification does not dictate what the MASTER will do in response to [RTY_I].

RULE 3.80
If a MASTER supports the [RTY_I] signal, then the WISHBONE DATASHEET MUST describe how it reacts in response to the signal.

RULE 3.90
If a SLAVE supports the [ERR_O] or [RTY_O] signals, then the SLAVE MUST NOT assert more than one of the following signals at any time: [ACK_O], [ERR_O] or [RTY_O]

RULE 3.100
MASTER and SLAVE interfaces MUST be designed so that there are no intermediate logic gates between a registered flip-flop and the signal outputs on: [STB_O] and [CYC_O].

OBSERVATION 3.20
The Wishbone interface can be designed so that there are no intermediate logic gates between a registered flip-flop and the signal output. This rule prevents sloppy design practices from slowing down the interconnect.

RULE 3.110
SLAVE interfaces MUST be designed so that the [ACK_O], [ERR_O] and [RTY_O] signals are asserted and negated in response to the assertion and negation of [STB_O]. Furthermore, this activity MUST occur asynchronous to the [CLK_I] signal.

OBSERVATION 3.30
The asynchronous logic requirement assures that the interface can accomplish one data transfer per clock cycle. Furthermore, it simplifies the design of arbiters in multi-MASTER applications.

PERMISSION 3.30
Under certain circumstances SLAVE interfaces MAY be designed to hold [ACK_O] in the asserted state. This situation occurs when there is a single SLAVE on the interface, and that SLAVE always operates without wait states. In this case, the MASTER will assert the [STB_O] signal for one clock cycle.

RULE 3.130
MASTER interfaces MUST be designed to operate normally when SLAVE interface holds [ACK_I] in the asserted state.

3.1.3 Use of [STB_O]

RULE 3.140
MASTER interfaces MUST qualify the following signals with [STB_O]: [ADR_O], [DAT_O()], [SEL_O()], [WE_O], [SEL_O] and [TAGN_O].

RULE 3.150
MASTER interfaces MUST assert [CYC_O] for the duration of SINGLE READ / WRITE, BLOCK and RMW cycles. [CYC_O] MUST be asserted no later than the rising [CLK_I] edge that qualifies the assertion of [STB_O]. [CYC_O] MUST be negated no earlier than the rising [CLK_I] edge that qualifies the negation of [STB_O].

3.1.4 Use of [ACK_O], [ERR_O] and [RTY_O]

RULE 3.160
SLAVE interfaces MUST qualify the following signals with [ACK_O], [ERR_O] or [RTY_O]: [DAT_O()].

3.1.5 Use of [TAGN_O] Signals

Each Wishbone MASTER can drive one or more optional [TAGN_O] signals. These are user-defined tags that accompany each data transfer, and allows one type of data to be discriminated from another. This allows each bus cycle to be identified as to it's purpose. For example, in microprocessor based systems it is often necessary to discriminate between data transfer, interrupt acknowledge and caching operations.

RULE 3.180
MASTER interfaces that support the [TAGN_O] signal(s) MUST describe their use in the WISHBONE DATASHEET.

3.2 SINGLE READ / WRITE Cycles

The SINGLE READ / WRITE cycles perform one data transfer at a time. These are the basic cycles used to perform data transfers on the Wishbone interconnect.

RULE 3.190
All MASTER and SLAVE interfaces that support SINGLE READ or SINGLE WRITE cycles MUST conform to the timing requirements given in sections 3.2.1 and 3.2.2.

PERMISSION 3.40
MASTER and SLAVE interfaces MAY be designed so that they do not support the SINGLE READ or SINGLE WRITE cycles.

3.2.1 SINGLE READ Cycle

Figure 3-2 shows a SINGLE READ cycle. The bus protocol works as follows:

CLOCK EDGE 0: MASTER presents [ADR_O()] and [TAGN_O].
MASTER negates [WE_O] to indicate a READ cycle.
MASTER presents bank select [SEL_O()] to indicate where it expects data.
MASTER asserts [CYC_O] to indicate the start of the cycle.
MASTER asserts [STB_O] to qualify [ADR_O()], [SEL_O()] and [WE_O].

SETUP, EDGE 1: SLAVE decides inputs, and responds by asserting [ACK_I].
SLAVE presents valid data on [DAT_I()].
SLAVE asserts [ACK_I] in response to [STB_O] to indicate valid data is present.
MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()].

Note: SLAVE may insert wait states (-WSS-) before asserting [ACK_I], thereby allowing it to throttle the cycle speed. Any number of wait states may be added.

CLOCK EDGE 1: MASTER latches data on [DAT_I()].
MASTER negates [STB_O] and [CYC_O] to indicate the end of the cycle.

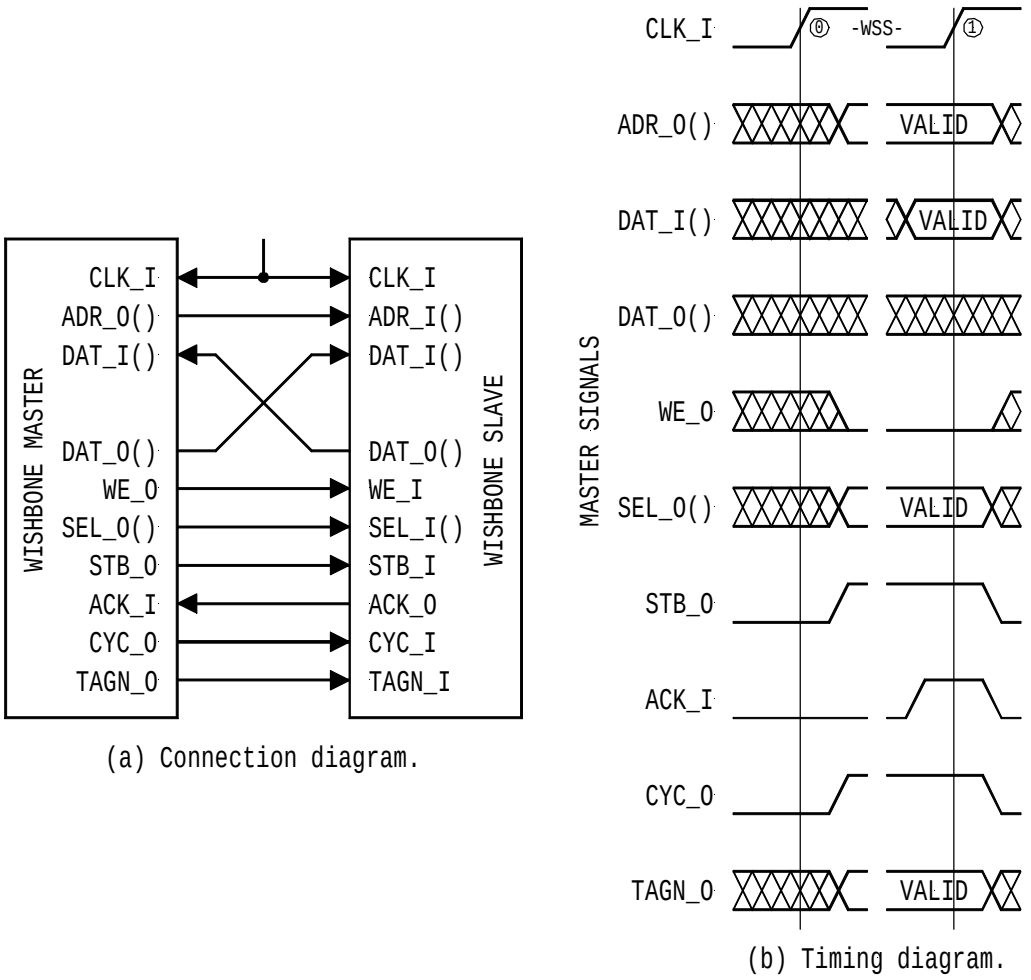


Figure 3-2. SINGLE READ cycle.

3.2.2 SINGLE WRITE Cycle

Figure 3-3 shows a SINGLE WRITE cycle. The bus protocol works as follows:

CLOCK EDGE 0: MASTER presents [ADR_O()] and [TAGN_O].
MASTER asserts [WE_O] to indicate a WRITE cycle.
MASTER presents bank select [SEL_O()] to indicate where it will place data.
MASTER asserts [CYC_O] to indicate the start of the cycle.
MASTER asserts [STB_O] to qualify [ADR_O()], [SEL_O()] and [WE_O].

SETUP, EDGE 1: SLAVE decides inputs, and responds by asserting [ACK_I].
SLAVE presents prepares to latch data on [DAT_O()].
SLAVE asserts [ACK_I] in response to [STB_O] to indicate that it will latch data.
MASTER monitors [ACK_I], and prepares to terminate the cycle.

Note: SLAVE may insert wait states (-WSS-) before asserting [ACK_I], thereby allowing it to throttle the cycle speed. Any number of wait states may be added.

CLOCK EDGE 1: SLAVE latches data on [DAT_O()].
MASTER negates [STB_O] and [CYC_O] to indicate the end of the cycle.

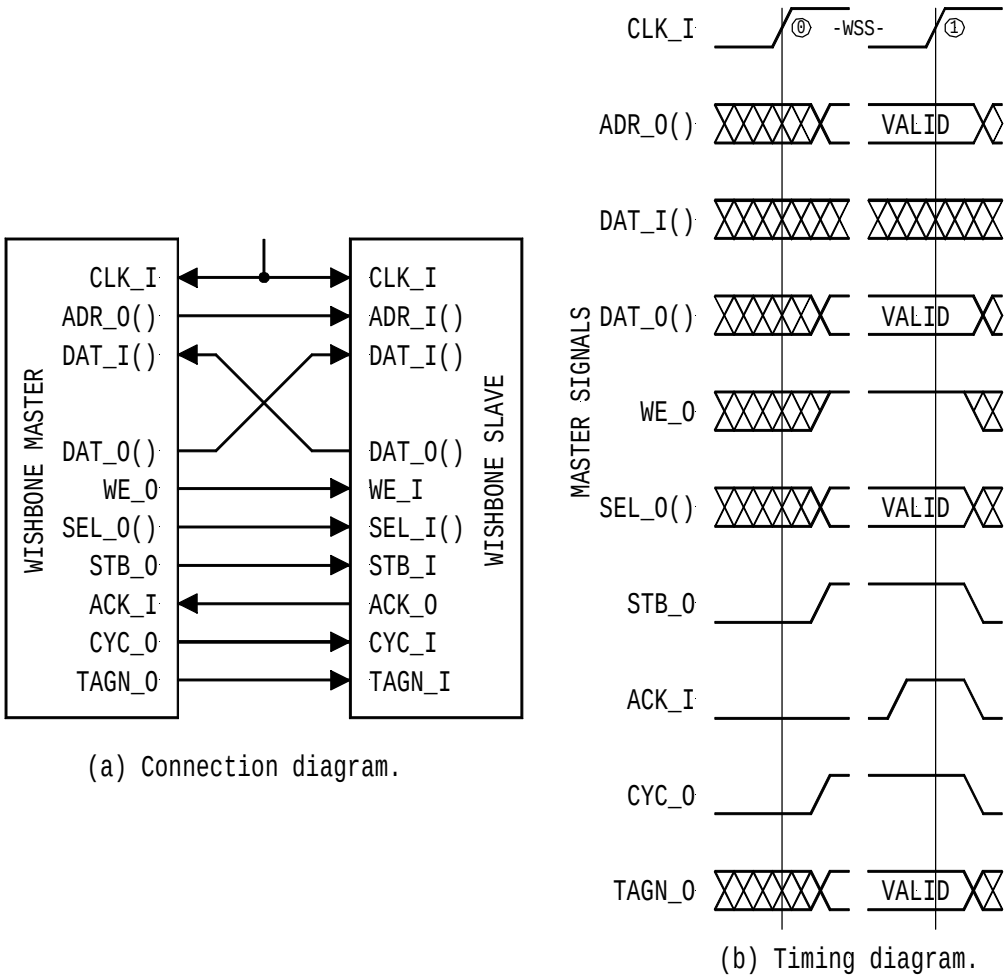


Figure 3-3. SINGLE WRITE cycle.

3.3 BLOCK READ / WRITE Cycles

The BLOCK transfer cycles perform multiple data transfers. They are very similar to single READ and WRITE cycles, but have a few special modifications to support multiple transfers.

During BLOCK cycles, the interface basically performs SINGLE READ/WRITE cycles as described above. However, the BLOCK cycles are modified somewhat so that these individual cycles are combined together to form a single BLOCK cycle. This function is most useful when multiple MASTERs are used on the interconnect. For example, if the SLAVE is a shared (dual port) memory, then an arbiter for that memory can determine when one MASTER is done with it so that another can gain access to the memory.

As shown in Figure 3-4, the [CYC_O] signal is asserted for the duration of a BLOCK cycle. This signal can be used to request permission to access from a shared resource from a local arbiter, and hold the access until the end of the current cycle. During each of the data transfers (within the block transfer), the normal handshaking protocol between [STB_O] and [ACK_I] is maintained.

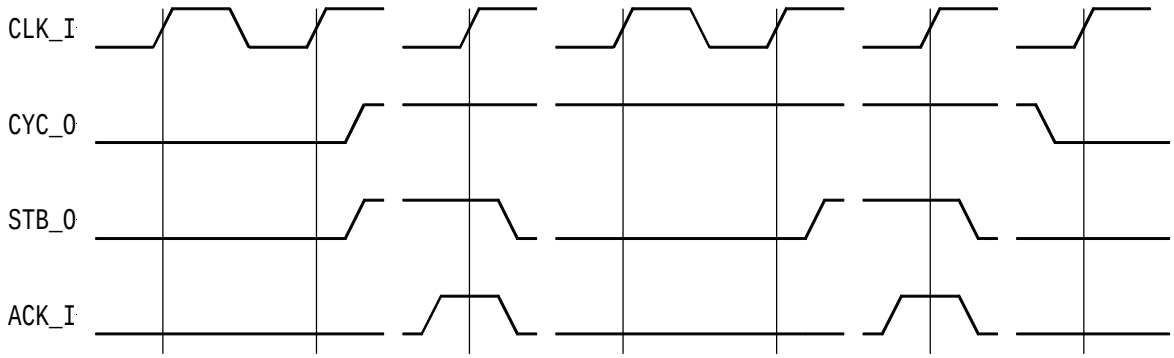


Figure 3-4. Use of [CYC_O] signal during BLOCK cycles.

It should be noted that the [CYC_O] signal does not necessarily rise and fall at the same time as [STB_O]. [CYC_O] may be asserted at the same time as [CYC_O], or one or more [CLK_I] edges before [CYC_O]. Similarly, [CYC_O] may be negated at the same time as [STB_O], or after an indeterminate number of clock cycles.

RULE 3.200
All MASTER and SLAVE interfaces that support BLOCK cycles MUST conform to the timing requirements given in sections 3.3.1 and 3.3.2.

PERMISSION 3.50
MASTER and SLAVE interfaces MAY be designed so that they do not support the BLOCK cycles.

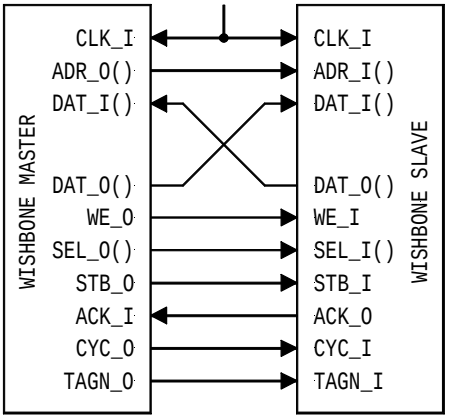
3.3.1 BLOCK READ Cycle

Figure 3-5 shows a BLOCK READ cycle. The BLOCK cycle is capable of a data transfer on every clock cycle. However, this example also shows how the MASTER and the SLAVE can both throttle the bus transfer rate by inserting wait states. A total of five transfers are shown. After the second transfer the MASTER inserts a wait state. After the fourth transfer the SLAVE inserts a wait state. The cycle is terminated after the fifth transfer. The protocol for this transfer works as follows:

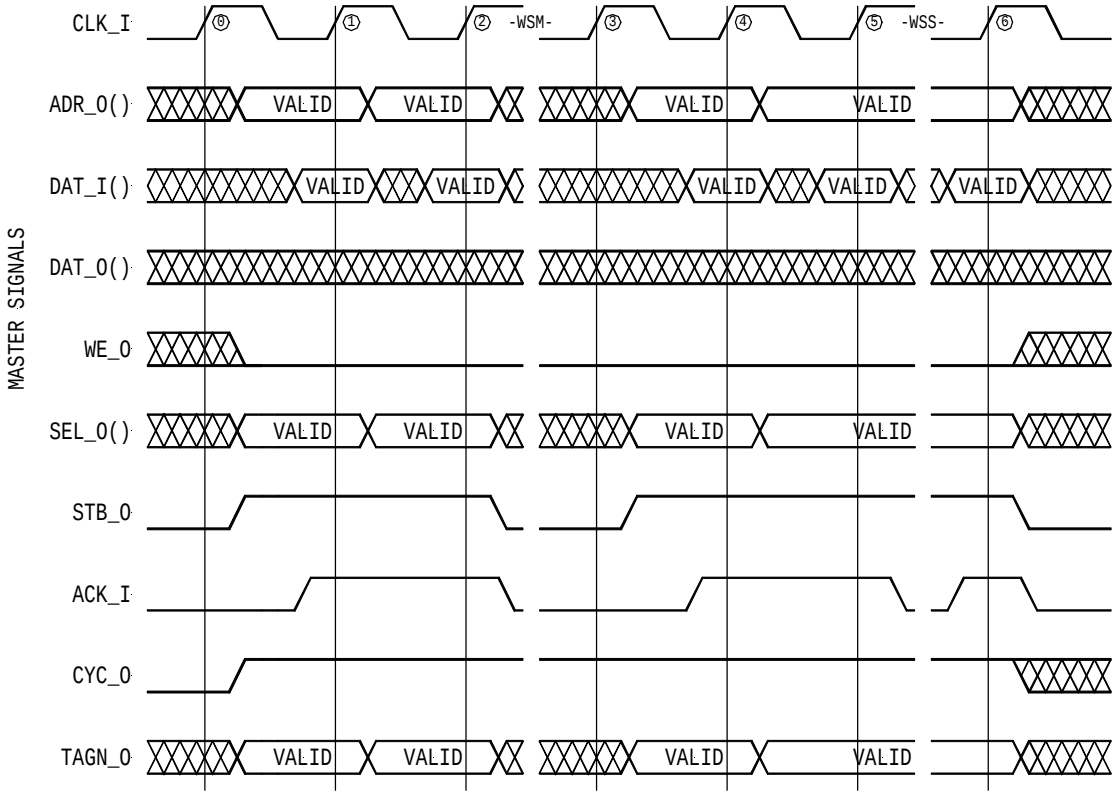
CLOCK EDGE 0:	MASTER presents [ADR_O()] and [TAGN_O]. MASTER negates [WE_O] to indicate a READ cycle. MASTER presents bank select [SEL_O()] to indicate where it expects data. MASTER asserts [CYC_O] to indicate the start of the cycle. MASTER asserts [STB_O]. Note: the MASTER must assert [CYC_O] and/or [TAGN_O] at, or anytime before, clock edge 1. The use of [TAGN_O] is optional.
SETUP, EDGE 1:	SLAVE decodes inputs, and responds by asserting [ACK_I]. SLAVE presents valid data on [DAT_I]. MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()].
CLOCK EDGE 1:	MASTER latches data on [DAT_I()]. MASTER presents new [ADR_O()] and [TAGN_O]. MASTER presents new bank select [SEL_O()] to indicate where it expects data.
SETUP, EDGE 2:	SLAVE decodes inputs, and responds by asserting [ACK_I]. SLAVE presents valid data on [DAT_I]. MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()].
CLOCK EDGE 2:	MASTER latches data on [DAT_I()]. MASTER negates [STB_O] to introduce a wait state (-WSM-).
SETUP, EDGE 3:	SLAVE negates [ACK_I] in response to [STB_O]. Note: any number of wait states can be inserted by the MASTER at this point.
CLOCK EDGE 3:	MASTER presents new [ADR_O()] and [TAGN_O]. MASTER presents new bank select [SEL_O()] to indicate where it expects data. MASTER asserts [STB_O].
SETUP, EDGE 4:	SLAVE decodes inputs, and responds by asserting [ACK_I]. SLAVE presents valid data on [DAT_I]. MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()].
CLOCK EDGE 4:	MASTER latches data on [DAT_I()]. MASTER presents [ADR_O()] and [TAGN_O]. MASTER presents new bank select [SEL_O()] to indicate where it expects data.
SETUP, EDGE 5:	SLAVE decodes inputs, and responds by asserting [ACK_I]. SLAVE presents valid data on [DAT_I]. MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()].
CLOCK EDGE 5:	MASTER latches data on [DAT_I()]. SLAVE negates [ACK_I] to introduce a wait state. Note: any number of wait states can be inserted by the SLAVE at this point.

SETUP, EDGE 6: SLAVE decodes inputs, and responds by asserting [ACK_I].
SLAVE presents valid data on [DAT_I].
MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()].

CLOCK EDGE 6: MASTER latches data on [DAT_I()].
MASTER terminates cycle by negating [STB_O] and [CYC_O].



(a) Connection diagram.



(b) Timing diagram.

Figure 3-5. BLOCK READ cycle.

3.3.2 BLOCK WRITE Cycle

Figure 3-6 shows a BLOCK WRITE cycle. The BLOCK cycle is capable of a data transfer on every clock cycle. However, this example also shows how the MASTER and the SLAVE can both throttle the bus transfer rate by inserting wait states. A total of five transfers are shown. After the second transfer the MASTER inserts a wait state. After the fourth transfer the SLAVE inserts a wait state. The cycle is terminated after the fifth transfer. The protocol for this transfer works as follows:

CLOCK EDGE 0: MASTER presents [ADR_O()] and [TAGN_O].
MASTER asserts [WE_O] to indicate a WRITE cycle.
MASTER presents bank select [SEL_O()] to indicate where it expects data.
MASTER asserts [CYC_O] and [TAGN_O] to indicate the start of the cycle.
MASTER asserts [STB_O].

Note: the MASTER must assert [CYC_O] and/or [TAGN_O] at, or anytime before, clock edge 1. The use of [TAGN_O] is optional.

SETUP, EDGE 1: SLAVE decodes inputs, and responds by asserting [ACK_I].
SLAVE prepares to latch data on [DAT_O].
MASTER monitors [ACK_I], and prepares to terminate the current data phase.

CLOCK EDGE 1: SLAVE latches data on [DAT_O()].
MASTER presents [ADR_O()] and [TAGN_O].
MASTER presents new bank select [SEL_O()] to indicate where it expects data.

SETUP, EDGE 2: SLAVE decodes inputs, and responds by asserting [ACK_I].
SLAVE prepares to latch data on [DAT_O].
MASTER monitors [ACK_I], and prepares to terminate the current data phase.

CLOCK EDGE 2: SLAVE latches data on [DAT_O()].
MASTER negates [STB_O] to introduce a wait state (-WSM-).

SETUP, EDGE 3: SLAVE negates [ACK_I] in response to [STB_O].

Note: any number of wait states can be inserted by the MASTER at this point.

CLOCK EDGE 3: MASTER presents [ADR_O()] and [TAGN_O].
MASTER presents bank select [SEL_O()] to indicate where it expects data.
MASTER asserts [STB_O].

SETUP, EDGE 4: SLAVE decodes inputs, and responds by asserting [ACK_I].
SLAVE prepares to latch data on [DAT_O].
MASTER monitors [ACK_I], and prepares to terminate the current data phase.

CLOCK EDGE 4: SLAVE latches data on [DAT_O()].
MASTER presents [ADR_O()] and [TAGN_O].
MASTER presents new bank select [SEL_O()] to indicate where it expects data.

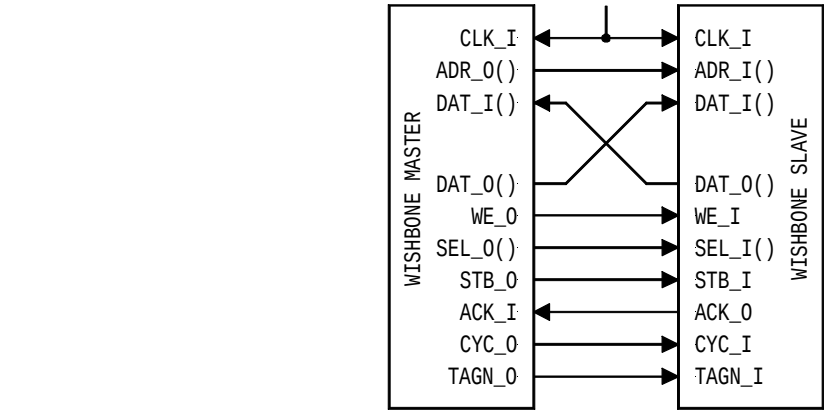
SETUP, EDGE 5: SLAVE decodes inputs, and responds by asserting [ACK_I].
SLAVE prepares to latch data on [DAT_O].
MASTER monitors [ACK_I], and prepares to terminate the current data phase.

CLOCK EDGE 5: SLAVE latches data on [DAT_O()].
SLAVE negates [ACK_I] to introduce a wait state.

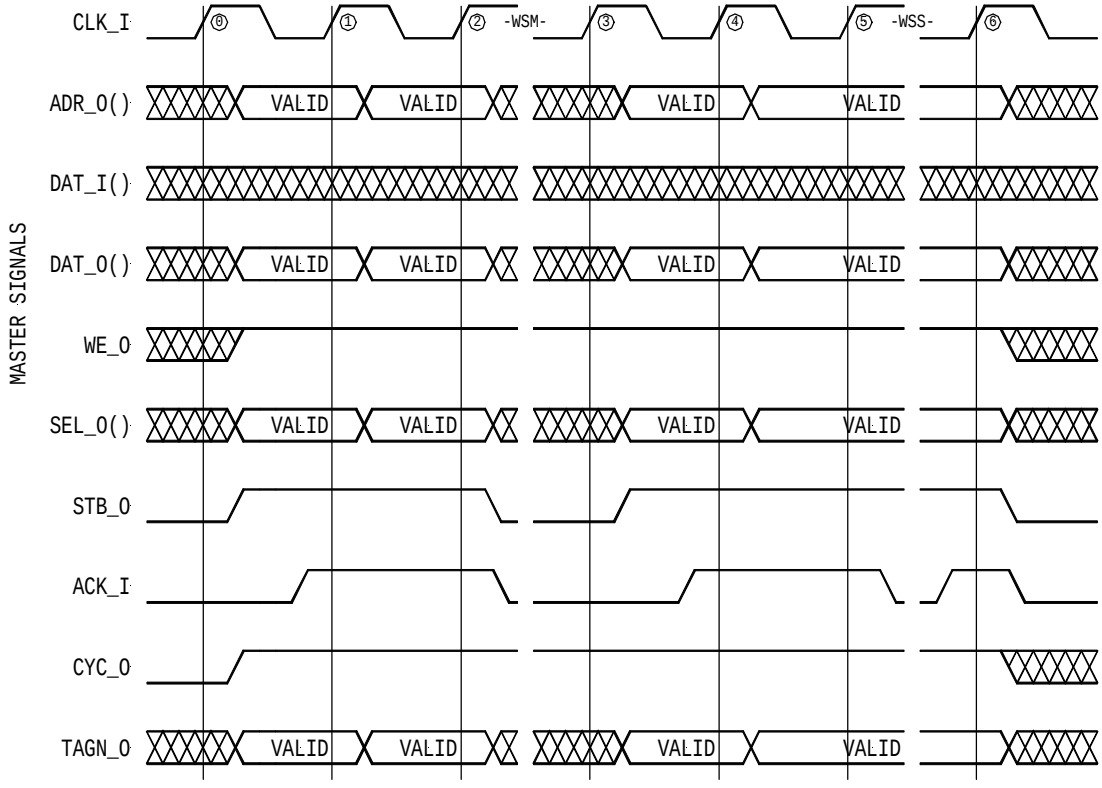
Note: any number of wait states can be inserted by the SLAVE at this point.

SETUP, EDGE 6: SLAVE decodes inputs, and responds by asserting [ACK_I].
SLAVE prepares to latch data on [DAT_O].
MASTER monitors [ACK_I], and prepares to terminate the current data phase.

CLOCK EDGE 6: SLAVE latches data on [DAT_O()].
MASTER terminates cycle by negating [STB_O] and [CYC_O].



(a) Connection diagram.



(b) Timing diagram.

Figure 3-6. BLOCK WRITE cycle.

3.4 RMW Cycle

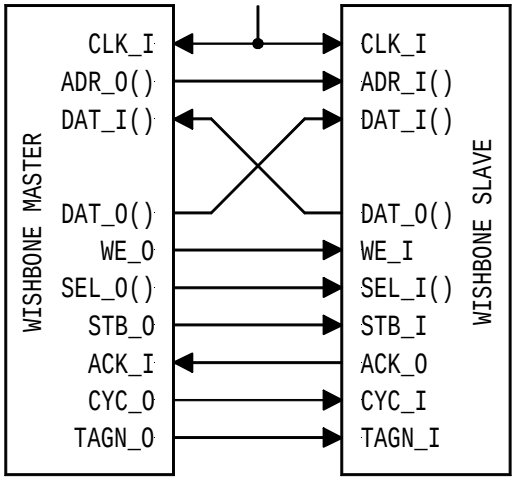
The RMW (read-modify-write) cycle is used for indivisible semaphore operations. During the first half of the cycle a single read data transfer is performed. During the second half of the cycle a write data transfer is performed. The [CYC_O] signal remains asserted during both halves of the cycle.

RULE 3.210
All MASTER and SLAVE interfaces that support RMW cycles MUST conform to the timing requirements given in section 3.4.

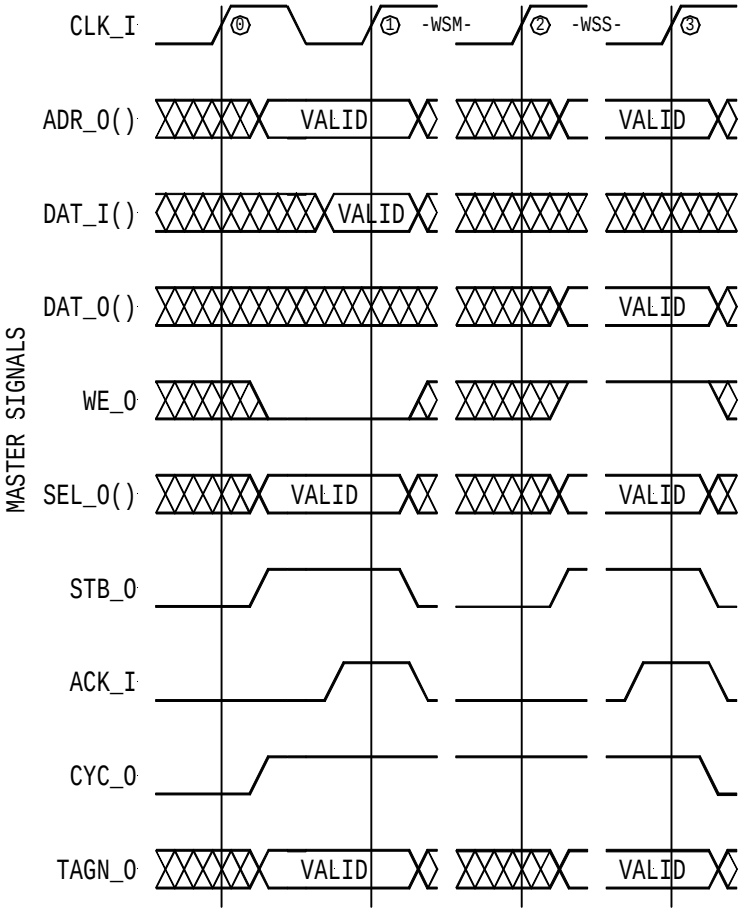
PERMISSION 3.60
MASTER and SLAVE interfaces MAY be designed so that they do not support the RMW cycles.

Figure 3-7 shows a read-modify-write (RMW) cycle. The RMW cycle is capable of a data transfer on every clock cycle. However, this example also shows how the MASTER and the SLAVE can both throttle the bus transfer rate by inserting wait states. Two transfers are shown. After the first (read) transfer, the MASTER inserts a wait state. During the second transfer the SLAVE inserts a wait state. The protocol for this transfer works as follows:

CLOCK EDGE 0:	MASTER presents [ADR_O()] and [TAGN_O]. MASTER negates [WE_O] to indicate a READ cycle. MASTER presents bank select [SEL_O()] to indicate where it expects data. MASTER asserts [CYC_O] and [TAGN_O] to indicate the start of the cycle. MASTER asserts [STB_O]. Note: the MASTER must assert [CYC_O] and/or [TAGN_O] at, or anytime before, clock edge 1. The use of [TAGN_O] is optional.
SETUP, EDGE 1:	SLAVE decodes inputs, and responds by asserting [ACK_I]. SLAVE presents valid data on [DAT_I]. MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()].
CLOCK EDGE 1:	MASTER latches data on [DAT_I()]. MASTER negates [STB_O] to introduce a wait state (-WSM-).
SETUP, EDGE 2:	SLAVE negates [ACK_I] in response to [STB_O]. MASTER asserts [WE_O] to indicate a WRITE cycle. Note: any number of wait states can be inserted by the MASTER at this point.
CLOCK EDGE 2:	MASTER presents the same [ADR_O()] and [TAGN_O] as was on clock 1. MASTER presents WRITE data on [DAT_O()]. MASTER presents new bank select [SEL_O()] to indicate where it expects data. MASTER asserts [STB_O].
SETUP, EDGE 3:	SLAVE decodes inputs, and responds by asserting [ACK_I] (when ready). SLAVE presents valid data on [DAT_I]. MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()]. Note: any number of wait states can be inserted by the SLAVE at this point.
CLOCK EDGE 3:	SLAVE latches data on [DAT_O()]. MASTER negates [STB_O] and [CYC_O] indicating the end of the cycle. SLAVE negates [ACK_I] in response to negated [STB_O].



(a) Connection diagram.



(b) Timing diagram.

Figure 3-7. RMW cycle.

3.5 Data Organization

Data organization refers to the ordering of data during transfers. There are two general types of ordering which are called BIG ENDIAN and LITTLE ENDIAN. BIG ENDIAN refers to data ordering where the most significant portion of an operand is stored at the lower address. LITTLE ENDIAN refers to data ordering where the most significant portion of an operand is stored at the higher address. The Wishbone architecture supports both methods of data ordering.

3.5.1 Nomenclature

A BYTE(N), WORD(N), DWORD(N) and QWORD(N) nomenclature is used to define data ordering. These terms are defined in Table 3-1. Figure 3-8 shows the operand locations for input and output data ports.

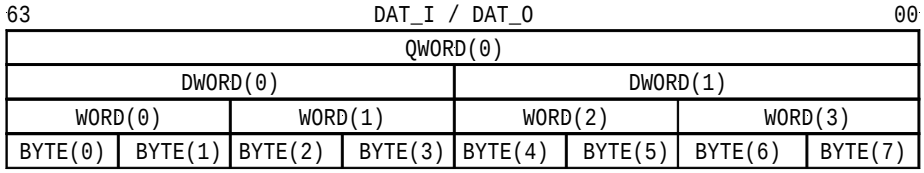
Table 3-1. Data transfer nomenclature.

Data Transfer Nomenclature		
Nomenclature	Granularity	Description
BYTE(N)	8-bit	An 8-bit BYTE transfer at address 'N'.
WORD(N)	16-bit	A 16-bit WORD transfer at address 'N'.
DWORD(N)	32-bit	A 32-bit Double WORD transfer at address 'N'.
QWORD(N)	64-bit	A 64-bit Quadruple WORD transfer at address 'N'.

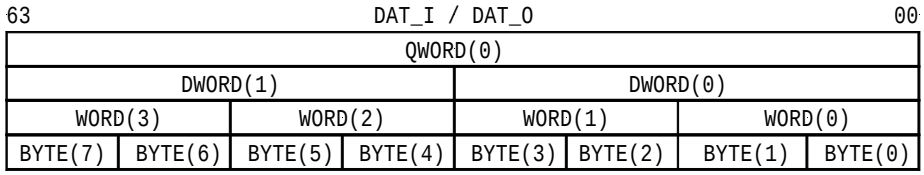
The table also defines the granularity of the interface. This indicates the minimum unit of data transfer that is supported by the interface. For example, the smallest operand that can be passed through a port with 16-bit granularity is a 16-bit WORD. In this case, an 8-bit operand cannot be transferred.

Figure 3-9 shows an example of how the 64-bit value of 0x0123456789ABC is transferred through BYTE, WORD, DWORD and QWORD ports using BIG ENDIAN data organization. Through the 64-bit QWORD port the number is directly transferred with the most significant bit at DAT_I / DAT_O(63). The least significant bit is at DAT_I / DAT_O(0). However, when the same operand is transferred through a 32-bit DWORD port, it is split into two bus cycles. The two bus cycles are each 32-bits in length, with the most significant DWORD transferred at the lower address, and the least significant DWORD transferred at the upper address. A similar situation applies to the WORD and BYTE cases.

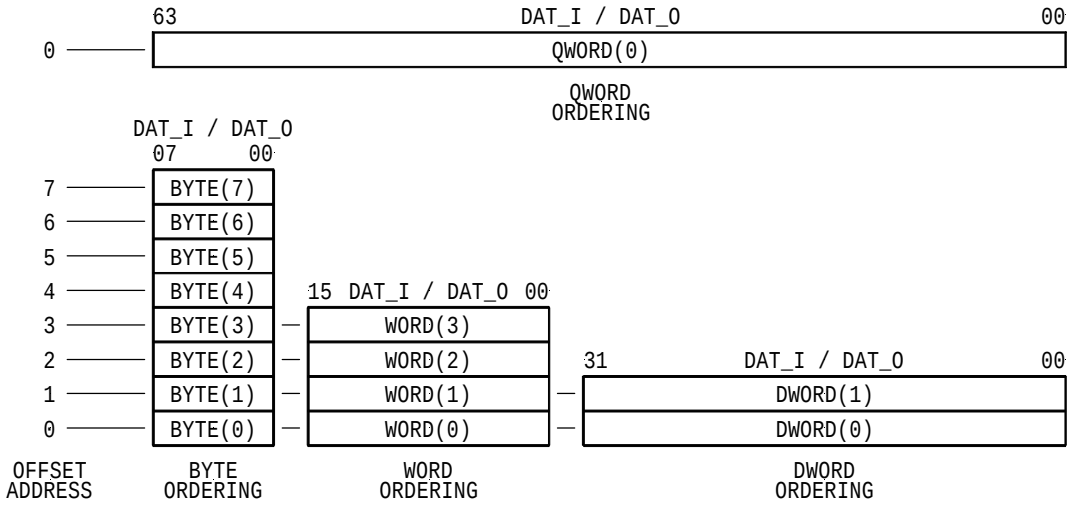
Figure 3-10 shows an example of how the 64-bit value of 0x0123456789ABC is transferred through BYTE, WORD, DWORD and QWORD ports using LITTLE ENDIAN data organization. Through the 64-bit QWORD port the number is directly transferred with the most significant bit at DAT_I / DAT_O(63). The least significant bit is at DAT_I / DAT_O(0). However, when the same operand is transferred through a 32-bit DWORD port, it is split into two bus cycles. The two bus cycles are each 32-bits in length, with the least significant DWORD transferred at the lower address, and the most significant DWORD transferred at the upper address. A similar situation applies to the WORD and BYTE cases.



(a) BIG ENDIAN BYTE, WORD, DWORD and QWORD positioning in a 64-bit operand.



(b) LITTLE ENDIAN BYTE, WORD, DWORD and QWORD positioning in a 64-bit operand.



(c) Address nomenclature.

Figure 3-8. Operand locations for input and output data ports.

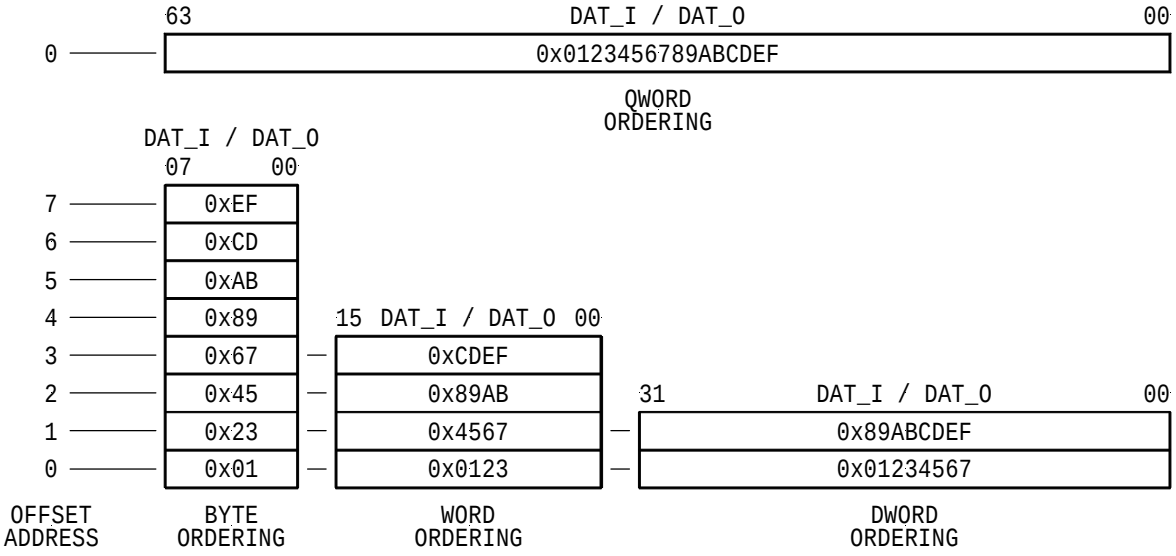


Figure 3-9. Example showing a variety of BIG ENDIAN transfers over various port sizes.

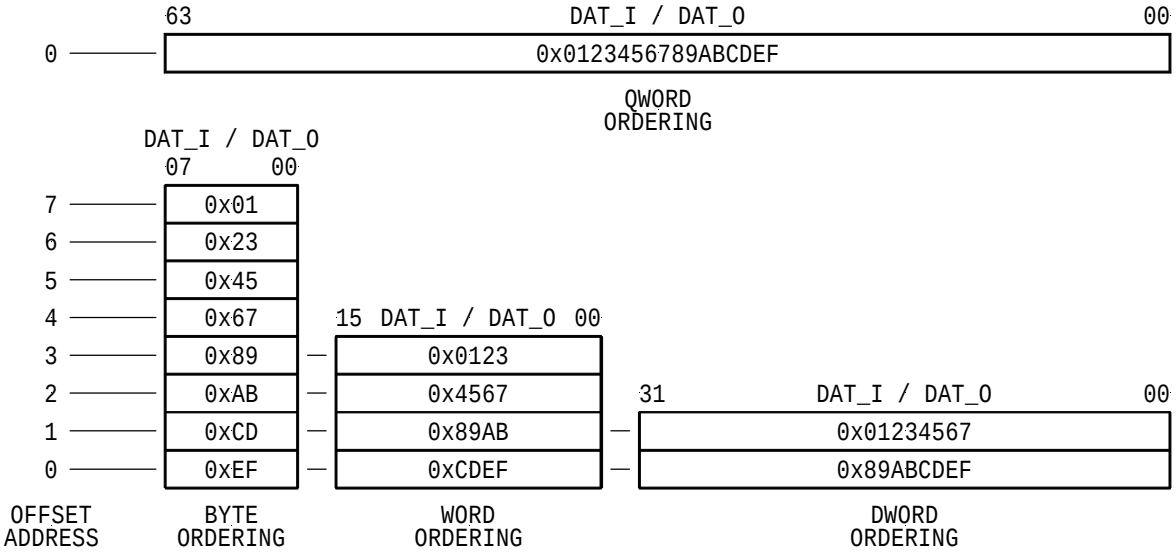


Figure 3-10. Example showing a variety of LITTLE ENDIAN transfers over various port sizes.

RULE 3.220
Data organization MUST conform to the ordering indicated in Figure 3-8.

RULE 3.230
The WISHBONE DATASHEET MUST indicate the port size. The port size shall be indicated as 8-bit, 16-bit, 32-bit or 64-bit.

RULE 3.235
The WISHBONE DATASHEET MUST indicate the port granularity. The granularity shall be indicated as 8-bit, 16-bit, 32-bit or 64-bit.

RULE 3.237
The WISHBONE DATASHEET MUST indicate the maximum operand size. The maximum operand size shall be indicated as 8-bit, 16-bit, 32-bit or 64-bit.

PERMISSION 3.35
In some cases the maximum operand size is unknown. In those cases, the maximum operand size shall be the same as the granularity.

RULE 3.240
The WISHBONE DATASHEET MUST indicate the data transfer ordering. The ordering shall be indicated as BIG ENDIAN or LITTLE ENDIAN.

PERMISSION 3.70
When the port size equals the granularity, then the interface may be specified as BIG ENDIAN and/or LITTLE ENDIAN.

OBSERVATION 3.40
When the port size equals the granularity, then BIG ENDIAN and LITTLE ENDIAN transfers are identical.

3.5.2 Transfer Sequencing

The sequence in which data is transferred through a port is not regulated by this specification. For example, a 64-bit operand through a 32-bit port will take two bus cycles. However, the specification does not require that the lower or upper DWORD be transferred first.

RECOMMENDATION 3.05
Design interfaces so that data is transferred sequentially from lower addresses to a higher addresses.

OBSERVATION 3.50
The sequence in which an operand is transferred through a data port is not highly regulated by the specification. That is because different IP cores may produce the data in different ways. The sequence is therefore application-specific.

RULE 2.45
The WISHBONE DATASHEET MUST indicate the sequence of data transfer through the port. If the sequence of data transfer is not known, then the datasheet must indicate that it is undefined.

3.5.3 Data Organization for 64-bit Ports

RULE 3.250
Data organization on 64-bit ports MUST conform to Figure 3-11.

64-bit Data Bus With 8-bit (BYTE) Granularity									
	Address Range:	Active Portion of Data Bus							
	ADR_I ADR_O (63..03)	DAT_I DAT_O (63..56)	DAT_I DAT_O (55..48)	DAT_I DAT_O (47..40)	DAT_I DAT_O (39..32)	DAT_I DAT_O (31..24)	DAT_I DAT_O (23..16)	DAT_I DAT_O (15..08)	DAT_I DAT_O (07..00)
	Active Select Line	SEL_I(7) SEL_O(7)	SEL_I(6) SEL_O(6)	SEL_I(5) SEL_O(5)	SEL_I(4) SEL_O(4)	SEL_I(3) SEL_O(3)	SEL_I(2) SEL_O(2)	SEL_I(1) SEL_O(1)	SEL_I(0) SEL_O(0)
BYTE Ordering	BIG ENDIAN	BYTE(0)	BYTE(1)	BYTE(2)	BYTE(3)	BYTE(4)	BYTE(5)	BYTE(6)	BYTE(7)
	LITTLE ENDIAN	BYTE(7)	BYTE(6)	BYTE(5)	BYTE(4)	BYTE(3)	BYTE(2)	BYTE(1)	BYTE(0)

64-bit Data Bus With 16-bit (WORD) Granularity					
	Address Range	Active Portion of Data Bus			
	ADR_I ADR_O (63..02)	DAT_I DAT_O (63..48)	DAT_I DAT_O (47..32)	DAT_I DAT_O (31..16)	DAT_I DAT_O (15..00)
	Active Select Line	SEL_I(3) SEL_O(3)	SEL_I(2) SEL_O(2)	SEL_I(1) SEL_O(1)	SEL_I(0) SEL_O(0)
WORD Ordering	BIG ENDIAN	WORD(0)	WORD(1)	WORD(2)	WORD(3)
	LITTLE ENDIAN	WORD(3)	WORD(2)	WORD(1)	WORD(0)

64-bit Data Bus With 32-bit (DWORD) Granularity			
	Address Range	Active Portion of Data Bus	
	ADR_I ADR_O (63..01)	DAT_I DAT_O (63..32)	DAT_I DAT_O (31..00)
	Active Select Line	SEL_I(1) SEL_O(1)	SEL_I(0) SEL_O(0)
DWORD Ordering	BIG ENDIAN	DWORD(0)	DWORD(1)
	LITTLE ENDIAN	DWORD(1)	DWORD(0)

64-bit Data Bus With 64-bit (QWORD) Granularity		
	Address Range	Active Portion of Data Bus
	ADR_I ADR_O (63..00)	DAT_I DAT_O (63..00)
	Active Select Line	SEL_I(0) SEL_O(0)
QWORD Ordering	BIG ENDIAN	QWORD(0)
	LITTLE ENDIAN	QWORD(0)

Figure 3-11. Data organization for 64-bit ports.

3.5.4 Data Organization for 32-bit Ports

RULE 3.260
Data organization on 32-bit ports MUST conform to Figure 3-12.

32-bit Data Bus With 8-bit (BYTE) Granularity					
	Address Range	Active Portion of Data Bus			
	ADR_I ADR_O (63..02)	DAT_I DAT_O (31..24)	DAT_I DAT_O (23..16)	DAT_I DAT_O (15..08)	DAT_I DAT_O (07..00)
	Active Select Line	SEL_I(3) SEL_O(3)	SEL_I(2) SEL_O(2)	SEL_I(1) SEL_O(1)	SEL_I(0) SEL_O(0)
BYTE Ordering	BIG ENDIAN	BYTE(0) BYTE(4)	BYTE(1) BYTE(5)	BYTE(2) BYTE(6)	BYTE(3) BYTE(7)
	LITTLE ENDIAN	BYTE(3) BYTE(7)	BYTE(2) BYTE(6)	BYTE(1) BYTE(5)	BYTE(0) BYTE(4)

32-bit Data Bus With 16-bit (WORD) Granularity			
	Address Range	Active Portion of Data Bus	
	ADR_I ADR_O (63..01)	DAT_I DAT_O (31..16)	DAT_I DAT_O (15..00)
	Active Select Line	SEL_I(1) SEL_O(1)	SEL_I(0) SEL_O(0)
WORD Ordering	BIG ENDIAN	WORD(0) WORD(2)	WORD(1) WORD(3)
	LITTLE ENDIAN	WORD(1) WORD(3)	WORD(0) WORD(2)

32-bit Data Bus With 32-bit (DWORD) Granularity		
	Address Range	Active Portion of Data Bus
	ADR_I ADR_O (63..00)	DAT_I DAT_O (31..00)
	Active Select Line	SEL_I(0) SEL_O(0)
DWORD Ordering	BIG ENDIAN	DWORD(0) DWORD(1)
	LITTLE ENDIAN	DWORD(0) DWORD(1)

Figure 3-12. Data organization for 32-bit ports.

3.5.5 Data Organization for 16-bit Ports

RULE 3.270
Data organization on 16-bit ports MUST conform to Figure 3-13.

16-bit Data Bus With 8-bit (BYTE) Granularity			
	Address Range ADR_I ADR_O (63..01)	Active Portion of Data Bus	
		DAT_I DAT_O (15..08)	DAT_I DAT_O (07..00)
	Active Select Line	SEL_I(1) SEL_O(1)	SEL_I(0) SEL_O(0)
BYTE Ordering	BIG ENDIAN	BYTE(0) BYTE(2) BYTE(4) BYTE(6)	BYTE(1) BYTE(3) BYTE(5) BYTE(7)
	LITTLE ENDIAN	BYTE(1) BYTE(3) BYTE(5) BYTE(7)	BYTE(0) BYTE(2) BYTE(4) BYTE(6)

16-bit Data Bus With 16-bit (WORD) Granularity		
	Address Range ADR_I ADR_O (63..00)	Active Portion of Data Bus DAT_I DAT_O (15..00)
	Active Select Line	SEL_I(0) SEL_O(0)
WORD Ordering	BIG ENDIAN	WORD(0) WORD(1) WORD(2) WORD(3)
	LITTLE ENDIAN	WORD(0) WORD(1) WORD(2) WORD(3)

Figure 3-13. Data organization for 16-bit ports.

3.5.6 Data Organization for 8-bit Ports

RULE 3.280
Data organization on 8-bit ports MUST conform to Figure 3-14.

8-bit Data Bus With 8-bit (BYTE) Granularity		
	Address Range	Active Portion of Data Bus
	ADR_I ADR_O (63..00)	DAT_I DAT_O (07..00)
	Active Select Line	SEL_I(0) SEL_O(0)
BYTE Ordering	BIG ENDIAN	BYTE(0) BYTE(1) BYTE(2) BYTE(3) BYTE(4) BYTE(5) BYTE(6) BYTE(7)
	LITTLE ENDIAN	BYTE(0) BYTE(1) BYTE(2) BYTE(3) BYTE(4) BYTE(5) BYTE(6) BYTE(7)

Figure 3-14. Data organization for 8-bit ports.

Chapter 4 – Timing Specification

The Wishbone specification is designed to provide the end user with very simple timing constraints. Although the application specific circuit(s) will vary in this regard, the interface itself is designed to work without the need for detailed timing specifications. In all cases, the only timing information that is needed by the end user is the maximum clock frequency (for [CLK_I]) that is passed to a place & route tool. The maximum clock frequency is dictated by the time delay between a positive clock edge on [CLK_I] to the setup on a stage further down the logical signal path. This delay is shown graphically in Figure 4-1, and is defined as Tpd,clk-su.

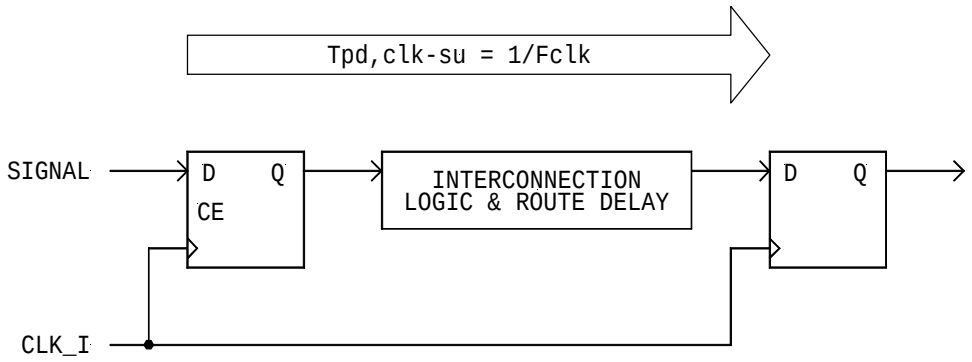


Figure 4-1. Definition for Tpd,clk-su.

RULE 4.10
The clock input [CLK_I] to each IP core MUST coordinate all activities for the internal logic within the Wishbone interface. All Wishbone output signals are registered at the rising edge of [CLK_I]. All Wishbone input signals must be stable before the rising edge of [CLK_I].

PERMISSION 4.10
The user's place and route tool MAY be used to enforce this rule.

OBSERVATION 4.10
Most place and route tools can be easily configured to enforce this rule. Generally, it only requires a single timing specification for Tpd,clk-su.

RULE 4.20
The Wishbone interface MUST use synchronous, RTL design methodologies that, given nearly infinitely fast gate delays, will operate over a nearly infinite range of clock frequencies on [CLK_I].

OBSERVATION 4.20
Realistically, the Wishbone interface will never be expected to operate over a nearly infinite frequency range. However this requirement eliminates the need for non-portable timing constraints (that may work only on certain target devices).

OBSERVATION 4.30
The Wishbone interface logic assumes that a low-skew clock distribution scheme is used on the target device, and that the clock-skew shall be low enough to permit reliable operation over the environmental conditions.

PERMISSION 4.20
The IP core connected to a Wishbone interface MAY include application specific timing requirements.

RULE 4.30
The clock input [CLK_I] MUST have a duty cycle that is no less than 40%, and no greater than 60%.

SUGGESTION 4.1
Design an IP core so that all of the circuits (including the Wishbone interconnect) follow the aforementioned RULEs, as this will make the core portable across a wide range of target devices and technologies.

Chapter 5 – Application Interface

This chapter describes various methods and issues for Wishbone interconnection. In some cases, the interconnection requires no glue logic. In more sophisticated systems, the user may need to add this logic. However, the common interface between the system-on-a-chip components will make this task much simpler.

5.1 Interconnection Methods

There are four general ways to interconnect MASTER and SLAVE IP cores. These include:

- Single MASTER / Single SLAVE interconnection
- Single MASTER / Multiple SLAVE interconnection
- Multiple MASTER interconnection
- Crossbar interconnection

5.1.1 Single MASTER / Single SLAVE Interconnection

The application interface may be operated with a single MASTER and a single SLAVE. This interconnection is generally the simplest and the fastest way to integrate IP cores. It is simplest because (in many cases) the MASTER and SLAVE can simply be connected together. Figure 5-1 shows an example of this interconnection. It should be noted that this example assumes that the port widths, port granularities, operand sizes and address widths are equal.

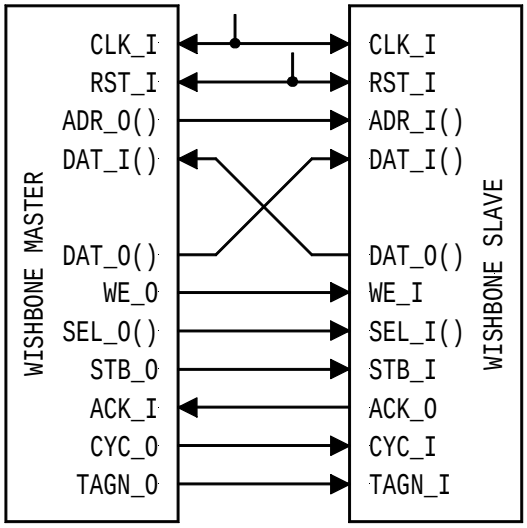


Figure 5-1. Single MASTER / Single SLAVE interconnection example.

5.1.2 Single MASTER / Multiple SLAVE Interconnection

The Wishbone application interface may be operated with a single MASTER and multiple SLAVEs. Refer to the multiple MASTER configuration (below) for more information.

5.1.3 Multiple MASTER Interconnection

The Wishbone application interface may be operated with multiple MASTER interconnections. Figure 5-2 shows a sample application interface with two Wishbone MASTERs, and two Wishbone SLAVEs. For purposes of clarity, only the [CYC_O], [STB_O] and [ACK_I] signals are shown. Figure 5-3 shows a sample timing diagram for the same circuit.

At the beginning of a bus cycle a MASTER asserts its [CYC_O] signal. The [CYC_O] signals from the two MASTERs are routed to an arbiter that determines which gets possession of the interconnection. The winning MASTER is identified by the arbiter by the assertion of [GNT1] or [GNT2], which correspond to the first and second masters respectively. The grant signals are then used determine which of the MASTERs will drive the common cycle [COMCYC] and strobe [COMSTB] signals.

The acknowledge signals from the SLAVEs are 'or'ed together to form a common acknowledge signal [COMACK]. In this case, each slave will decode the common address (not shown), and will respond if it is the participating slave in a bus cycle.

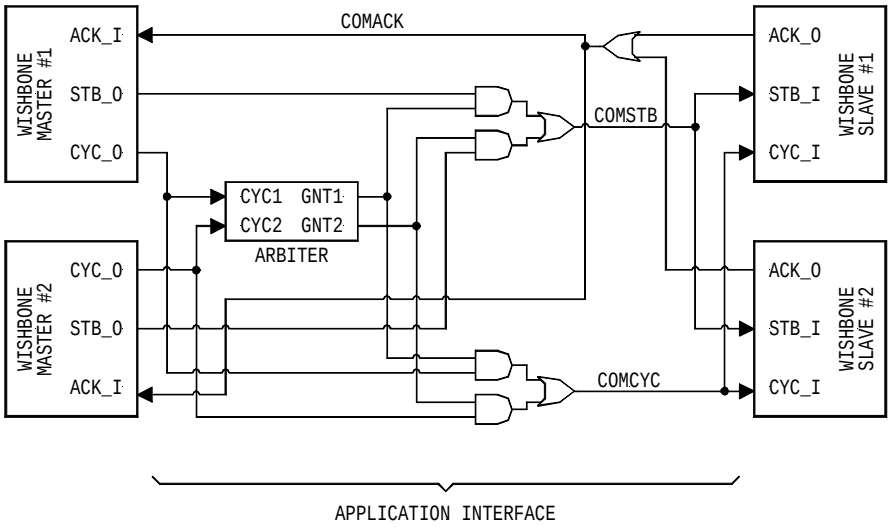
The other interconnect signals (e.g. [DAT_I] and [DAT_O]) are not shown in this example. These can be interconnected using multiplexors or three-state buses.

5.1.4 Crossbar Interconnection

The Wishbone MASTERs and SLAVEs may be interconnected with a crossbar switch. Crossbar switches are systems that usually have multiple MASTERs and multiple SLAVEs.

Crossbar switches are mechanisms that allow individual pairs of MASTERs and SLAVEs to connect and communicate. Each connection channel can be operated in parallel to other connection channels. This increases the data transfer rate of the entire system by employing parallelism. Stated another way, two 100 Mbyte/second channels can operate in parallel, thereby providing a 200 Mbyte/second transfer rate. This makes the crossbar switches inherently faster than traditional bus schemes.

Crossbar routing mechanisms generally support dynamic configurability. This essentially creates a reconfigurable and reliable network system. Most crossbar architectures are also scalable, meaning that families of crossbars can be added as the needs arise.




```
ARBITER EQUATIONS:

GNT1 <= ( not(RST_I) and not(GNT2)              and CYC1 )
     or ( not(RST_I) and not(GNT1) and not(CYC2) and CYC1 );

GNT2 <= ( not(RST_I) and not(GNT2)              and CYC2 and not(CYC1) )
     or ( not(RST_I) and     GNT2  and not(GNT1) and CYC2               );
```

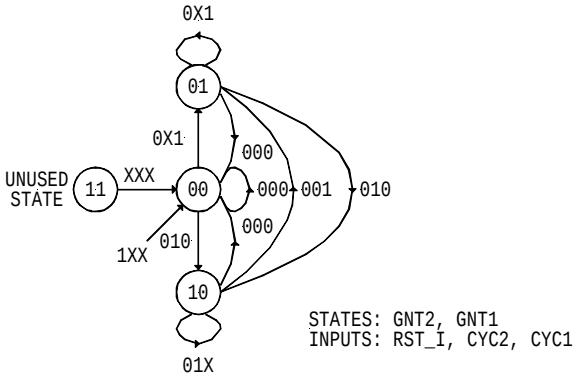


Figure 5-2. Sample application with two Wishbone MASTERs, and two Wishbone SLAVEs.

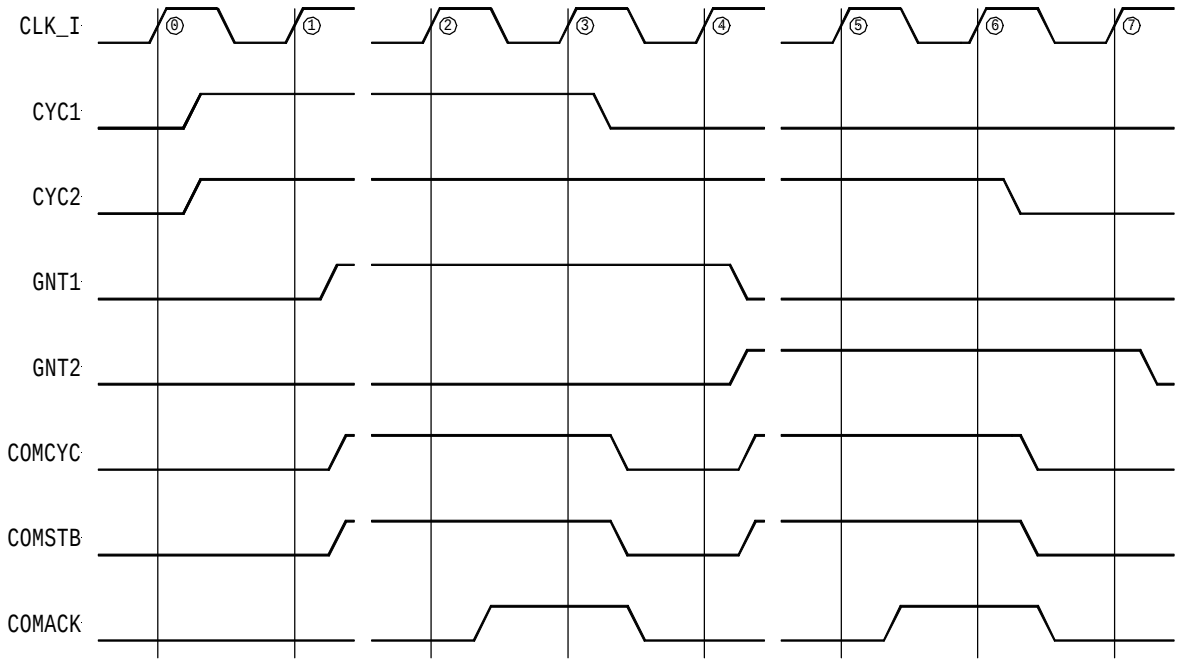


Figure 5-3. Timing diagram for multiple MASTER / multiple SLAVE example.

5.2 Three-State Interconnections

The interconnection can take the form of a three-state bus. Figure 5-4 shows the connection of a MASTER or SLAVE data input and output buses to a three-state data bus. Also note that the resistor/current source listed in the figure can also be a 'pull-down' resistor or current source.

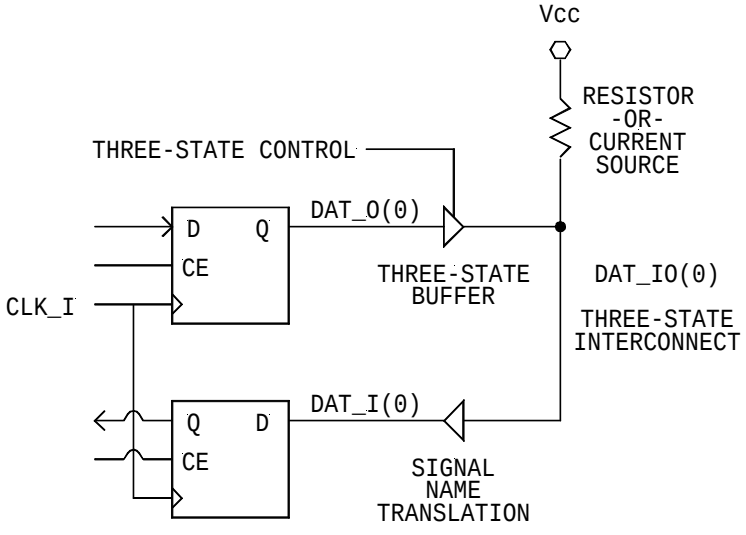


Figure 5-4. Connection of data bus to a three-state interconnection.

5.3 Endian Conversion

In some cases the user may wish to connect a BIG ENDIAN IP core to a LITTLE ENDIAN IP core. In many cases the conversion is quite straightforward, and does not require any exotic conversion logic. Furthermore, the conversion does not create any speed degradation of the interface. In general, the ENDIAN conversion takes place by renaming the data and select I/O signals at the source or destination IP core.

Figure 5-5 shows a simple example where a 32-bit BIG ENDIAN core output (CORE 'A') is connected to a 32-bit LITTLE ENDIAN core input (CORE 'B'). Both cores have 32-bit operand sizes and 8-bit granularity. As can be seen in the diagram, the ENDIAN conversion is accomplished by cross coupling the data and select signal arrays. This is quite simple since the conversion is accomplished by the interconnection wiring between the cores. This is especially simple in soft IP cores (using VHDL or Verilog® hardware description languages), as it only requires the renaming of signals.

In some cases the address lines may also need to be modified between the two cores. For example, if 64-bit operands are transferred between two cores with 8-bit port sizes, then the address lines may need to be modified as well.

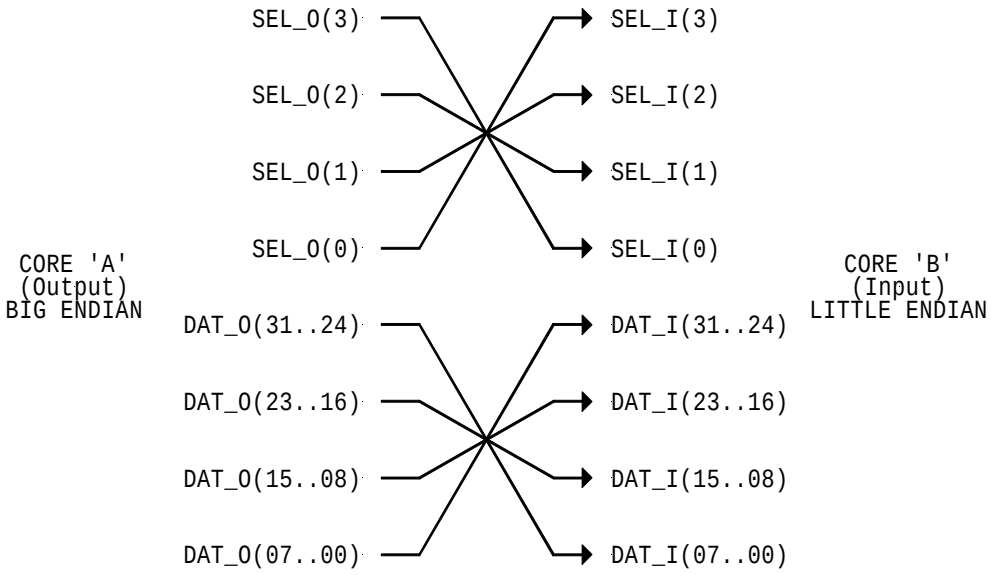


Figure 5-5. Converting a BIG ENDIAN output to a LITTLE ENDIAN input.

Appendix – Glossary of Terms

0x (prefix)
The '0x' prefix indicates a hexadecimal number.

Active High Logic State
A logic state that is 'true' when the logic level is a binary '1'. The high state is at a higher voltage than the low state.

Active Low Logic State
A logic state that is 'true' when the logic level is a binary '0'. The low state is at a lower voltage than the high state.

ASIC
Acronym for: Application Specific Integrated Circuit. General term which describes a generic array of logic gates or analog building blocks which are programmed by a metalization layer at a silicon foundry. High level circuit descriptions are impressed upon the logic gates or analog building blocks in the form of metal interconnects.

Asserted
A verb indicating that a logic state as switched from the inactive state to the active state. When active high logic is used it means that a signal has switched from a low logic level to a high logic level.

Crossbar Interconnect (Switch)
Crossbar switches are mechanisms that allow IP cores connect and communicate. Each connection channel can be operated in parallel to other connection channels. This increases the data transfer rate of the entire system by employing parallelism. Stated another way, two 100 Mbyte/second channels can operate in parallel, thereby providing a 200 Mbyte/second transfer rate. This makes the crossbar switches inherently faster than traditional bus schemes. Crossbar routing mechanisms generally support dynamic configurability. This essentially creates a reconfigurable and reliable network system. Most crossbar architectures are also scalable, meaning that families of crossbars can be added as the needs arise.

Data Organization
The specification ordering of data during transfer. Generally, 8-bit (byte) data can be stored with the most significant byte of a mult-byte transfer at the highest or the lowest address. There are two methods common in the industry which are classified as BIG ENDIAN or LITTLE ENDIAN. However, these informal terms are not well defined and are avoided in this specification. In general, BIG ENDIAN refers to byte lane ordering where the most significant byte is stored at the lower address. LITTLE ENDIAN refers to byte lane ordering where the most significant byte is stored at the higher address. The term ENDIAN was allegedly derived from the book Gulliver's Travels by Jonathan Swift.

ENDIAN
See the definition under 'Data Organization'.

FIFO Memory
Acronym for: First In First Out. A type of memory used to transfer data between ports on two devices. The first data loaded into the FIFO by an output port on one device is the first data read by an input port on another device. The FIFO memory is very useful for interconnecting cores of differing speeds.

FPGA
Acronym for: Field Programmable Gate Array. Generally describes a generic array of logical gates and interconnect paths which are programmed by the end user. High level logic descriptions are impressed upon the gates and interconnect paths, often in the form of IP Cores.

Glue Logic
Logic gates and interconnections required to connect IP cores together. The requirements for glue logic vary greatly depending upon the interface requirements of the IP cores.

Granularity
The smallest unit of data transfer that a port is capable of transferring. For example, a 32-bit port can be broken up into four 8-bit BYTE segments. In this case, the granularity of the interface is 8-bits. Also see *port size* and *operand size*.

HDL
Acronym for: Hardware Description Language. Examples include VHDL and Verilog®.

IP Core
Acronym for: Intellectual Property Core. Also see 'soft core', 'firm core' and 'hard core'.

MASTER
A Wishbone interface that is capable of generating bus cycles. All systems based on the Wishbone interconnect must have at least one SLAVE.

Memory Mapped Addressing
An architecture that allows memory to be stored and recalled at individual, binary addresses.

Negated
A verb indicating that a logic state as switched from the active state to the inactive state. When active high logic is used it means that a signal has switched from a high logic level to a low logic level.

Operand Size
The operand size is the largest single unit of data transfer that will be moved through the interface. For example, a 32-bit DWORD operand can be moved through an 8-bit port with four data transfers. Also see *granularity* and *port size*.

PCI
Acronym for: Peripheral Component Interconnect. Generally used as a specification interconnection scheme between chips. While this specification is very flexible, it isn't practical for system-on-a-chip applications because if it's large size and slow speed.

Port Size
The width of the Wishbone data ports in bits. Also see *granularity* and *operand size*.

Router
A software tool that physically routes interconnection paths between logic gates. Applies to FPGA and ASIC devices.

RTL Design Methodology
A design methodology that uses register-transfer-logic (RTL) concepts. This design methodology moves data between registers. Data is latched in the registers at one or more stages along the path of signal propagation. The Wishbone specification uses a synchronous RTL design methodology which means that each register is clocked with a common clock.

Silicon Foundry
A factory which produces integrated circuits.

SLAVE
A Wishbone interface that is capable of receiving bus cycles. All systems based on the Wishbone interconnect must have at least one SLAVE.

System-on-a-chip
The ability to create whole systems on a single integrated circuit. In many cases, this requires the use of IP cores which have been designed by various IP core providers and individuals. System-on-a-chip is similar to traditional systems whereby the individual components.

Target Device
The semiconductor type (or technology) onto which the IP core design is impressed. Typical examples include FPGA and ASIC devices.

Verilog®
A textual based computer language intended for use in circuit design. The VHDL language is both a synthesis and a simulation tool. Verilog® was originally a proprietary language first conceived in 1983 at Gateway Design Automation (Acton, MA), and was later refined by Cadence Corporation. It has since been greatly expanded and refined, and much of it has been placed into the public domain. Complete descriptions of the language can be found in the IEEE 1364 specification.

VHDL
Acronym for: VHSIC Hardware Description Language. [VHSIC: Very High Speed Integrated Circuit]. A textual based computer language intended for use in circuit design. The VHDL language is both a synthesis and a simulation tool. Early forms of the language emerged from US Dept. of Defense ARPA projects in the 1960's, and have since been greatly expanded and refined. Complete descriptions of the language can be found in the IEEE 1076, 1073.3 and 1164 specifications.

VMEbus
Acronym for: Versa Module Eurocard bus. A popular microcomputer (board) bus. While this specification is very flexible, it isn't practical for system-on-a-chip applications because if it's large size and slow speed.

WISHBONE DATASHEET
Documentation must be provided for each IP core with a Wishbone interconnect. This helps the end user understand the operation of the core, and how to connect it to other cores. The documentation takes the form of a WISHBONE DATASHEET. This can be included in a technical reference manual for the IP core.