WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores

Release B3.1

Oct 20, 2019

CONTENTS:

Stewardship

This specification is maintained by the Free and Open Source Silicon Foundation. Questions, comments and suggestions about this document are welcome and should be directed to the FOSSi Foundation Specification Committee (specs@fossi-foundation.org, https://www.fossi-foundation.org). Steward for this specification is Richard Herveille (rherveille@opencores.org).

FOSSi Foundation maintains this document to provide an open, freely useable interconnect architecture for IP-cores hosted on OpenCores.org, LibreCores.org and others' IP Cores.

These specifications are intended to guarantee compatibility between compliant IP-cores and to improve cooperation among different users and suppliers.

Copyright & Trademark Release / Royalty Release / Patent Notice

Disclaimers

Acknowledgements

Like any great technical project, the WISHBONE specification could not have been completed without the help of many people. The Steward wishes to thank the following for their ideas, suggestions and contributions:

- Ray Alderman

- Yair Amitay
- Danny Cohan
- Marc Delvaux
- Miha Dolenc
- Volker Hetzer
- Magnus Homann
- Brian Hurt
- Linus Kirk
- Damjan Lampret
- Wade D. Peterson[1]
- Barry Rice
- John Rynearson
- Avi Shamli
- Rudolf Usselmann

Revision History

This specification is managed at the following repository: https://github.com/fossi-foundation/wishbone

[1] Wade D. Peterson from Silicore Corporation is the original author and steward. Without his dedication this specification would have never been where it is now.

CHAPTER

ONE

INTRODUCTION

The WISHBONE[1] System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores is a flexible design methodology for use with semiconductor IP cores. Its purpose is to foster design reuse by alleviating System-on-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, 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 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 three factors. First, there was a need for a good, reliable System-on-Chip integration solution. Second, there was a need for a common interface specification to facilitate structured design methodologies on large project teams. Third, they were impressed by the traditional system integration solutions afforded by micro-computer buses such as PCI bus and VMEbus.

[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 scheme he was attempting to find a name that was descriptive of a bi-directional data bus that used either multiplexers or three-state logic. This was solved by forming an interface with separate input and output paths. When these paths are connected to three-state logic it forms a 'Y' shaped configuration that resembles 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.

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 improv- ing performance and quality).

However, traditional microcomputer buses are fundamentally handicapped for use as a System-on-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-Chip is much simpler and faster. Furthermore, the System-on-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.

1.1 WISHBONE Features

The WISHBONE interconnection makes System-on-Chip and design reuse easy by creating a standard data exchange protocol. Features of this technology include:

- Simple, compact, logical IP core hardware interfaces that require very few logic gates.
- Supports structured design methodologies used by large project teams.
- Full set of popular data transfer bus protocols including:
  - READ/WRITE cycle
  - BLOCK transfer cycle
  - RMW cycle
- Modular data bus widths and operand sizes.
- Supports both BIG ENDIAN and LITTLE ENDIAN data ordering.
- Variable core interconnection methods support point-to-point, shared bus, crossbar switch, and switched fabric interconnections.
- Handshaking protocol allows each IP core to throttle its data transfer speed.
- Supports single clock data transfers.
- Supports normal cycle termination, retry termination and termination due to error.
- Modular address widths

- Partial address decoding scheme for SLAVEs. This facilitates high speed address decoding, uses less redundant logic and supports variable address sizing and interconnection means.
- User-defined tags. These are useful for applying information to an address bus, a data bus or a bus cycle. They are especially helpful when modifying a bus cycle to identify information such as:
  - Data transfers
  - Parity or error correction bits
  - 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-Chip configurations.
- Arbitration methodology is defined by the end user (priority arbiter, round-robin arbiter, etc.).
- Supports various IP core interconnection means, including:
  - Point-to-point
  - Shared bus
  - Crossbar switch
  - Data flow interconnection
  - Off chip
- Synchronous design assures portability, simplicity and ease of use.
- Very simple, variable timing specification.
- Documentation standards simplify IP core reference manuals.
- 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.
- Independent of FPGA and ASIC test methodologies.
- Seamless design progression between FPGA prototypes and ASIC production chips.

1.2 WISHBONE Objectives

The main objectives of this specification are

- to create a flexible interconnection means for use with semiconductor IP cores. This allows various IP cores to be connected together to form a System-on-Chip.
- to enforce compatibility between IP cores. This enhances design reuse.
- to create a robust standard, but one that does not unduly constrain the creativity of the core developer or the end user.
- to make it easy to understand by both the core developer and the end user.
- to facilitate structured design methodologies on large project teams. With structured design, individual team members can build and test small parts of the design. Each member of the design team can interface to the common, well-defined WISHBONE specification. When all of the sub-assemblies have been completed, the full system can be integrated.
- to create a portable interface that is independent of the underlying semiconductor technology. For example, WISHBONE interconnections can be made that support both FPGA and ASIC target devices.
- to make WISHBONE interfaces independent of logic signaling levels.
- 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.
- 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.
- to require a minimum standard for documentation. This takes the form of the WISHBONE DATASHEET, and allows IP core users to quickly evaluate and integrate new cores.
- 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 data interface.
- to allow users to create SoC components without infringing on the patent rights of others. While the use of WISHBONE technology does not necessarily prevent patent infringement, it does provide a reasonable safe haven where users can design around the patent claims of others. The specification also provides *cited patent references*, which describes the field of search used by the WISHBONE architects.
- to identify critical System-on-Chip interconnection technologies, and to place them into the public domain at the earliest possible date. This makes it more difficult for individuals and organizations to create proprietary technologies through the use of patent, trademark, copyright and trade secret protection mechanisms.
- to support a business model whereby IP Core suppliers can cooperate at a technical standards level, but can also compete in the commercial marketplace. This improves the overall quality and value of products through market forces such as price, service, delivery, performance and time-to-market. This business model also allows open source IP cores to be offered as well.

- 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 unthought-of, requirements.
- to create an architecture that allows various interconnection means between IP core modules. This insures that the end user can tailor the System-on-Chip to his/her own needs. For example, the entire interconnection system (which is analogous to a backplane on a standard microcomputer bus like VMEbus or cPCI) can be created by the system integrator. This allows the interconnection to be tailored to the final target device.
- to create an architecture that requires a minimum of glue logic. In some cases the System-on-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.
- to create an architecture with variable address and data path widths to meet a wide variety of system requirements.
- to create an architecture that fully supports the automatic generation of interconnection and IP Core systems. This allows components to be generated with parametric core generators.
- to create an architecture that supports both BIG ENDIAN and LITTLE ENDIAN data transfer organizations.
- to create an architecture that supports one data transfer per clock cycle.
- to create a flexible architecture that allows address, data and bus cycles to be tagged. Tags are user defined signals that allow users to modify a bus cycle with additional information. They are especially useful when novel or unusual control signals (such as parity, cache control or interrupt acknowledge) are needed on an interface.
- 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.
- to create an architecture that supports point-to-point interconnections between IP cores.
- to create an architecture that supports shared bus interconnections between IP cores.
- to create an architecture that supports crossbar switches between IP cores.
- to create an architecture that supports switched fabrics.
- 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.
- 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.

- to create a variable timing mechanism whereby the system clock frequency can be adjusted so as to control the power consumption of the integrated circuit.
- to create a synchronous protocol that provides a simple timing specification. This makes the interface very easy to integrate.
- to create a synchronous protocol where each MASTER and SLAVE can throttle the data transfer rate with a handshaking mechanism.
- to create a synchronous protocol that is optimized for System-on-Chip, but that is also suitable for off-chip I/O routing. Generally, the off-chip WISHBONE interconnect will operate at slower speeds.
- to create a backward compatible registered feedback high performance burst bus.

1.3 Specification Terminology

To avoid confusion, and to clarify the requirements for compliance, this specification uses five keywords. They 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 uppercase 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 are referenced in the timing diagrams. In some cases the MASTER and SLAVE signal names are different. For example, in the point-to-point interconnections 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. Unless otherwise noted, the timing diagrams refer to the connection diagram shown in Figure 1.2.

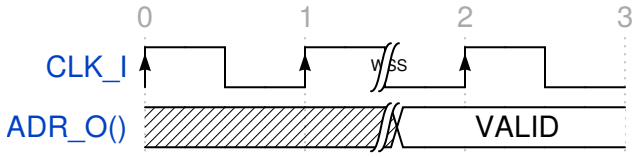


Figure 1.1: Use of timing diagrams.

Some signals may or may not be present on a specific interface. That's because many 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.

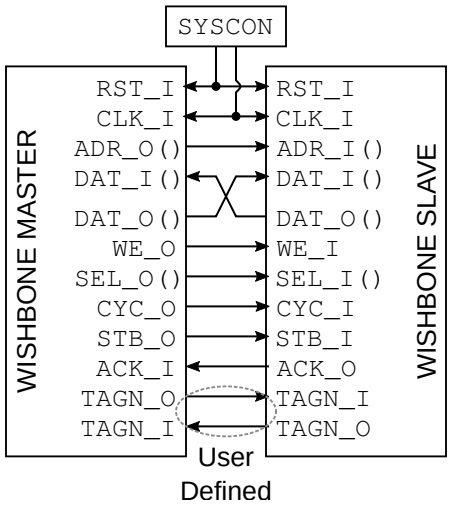


Figure 1.2: Standard connection for timing diagrams.

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 respectively. They also indicate that the signals (with the exception of clock transitions and hatched regions) will remain in a steady state during that time.

Undefined signal levels are indicated by a hatched region. This region indicates that the signal level is undefined, and may take any state. It also 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'.

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 name followed by a set of parenthesis. For example, [DAT_I()] is a signal array. Array limits may also be shown within the parenthesis. In this case the first number of the array limit indicates the most significant bit, and the second number indicates the least significant bit. For example, [DAT_I(63..0)] is a signal array with upper array boundary number sixty-three (the most significant bit), and lower array boundary number zero (the least significant bit). 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.

Special user defined signals, called *tags*, can also be used. Tags are assigned a *tag type* that

indicates the exact timing to which the signal must adhere. For example, if a parity bit such as [PAR_O] is added to a data bus, it would probably be assigned a tag type of TAG TYPE: TGD_O(). This indicates that the signal will adhere to the timing diagrams shown for [TGD_O()], which are shown in the timing diagrams for each bus cycle. Also note that, while all tag types are specified as arrays (with parenthesis '()'), the actual tag does not have to be a signal array. It can also be non-arrayed signal. 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.

1.6 WISHBONE Logo

The WISHBONE logo can be affixed to SoC documents that are compatible with this standard. *logo* shows the logo.

Figure 1.3: WISHBONE Logo.

PERMISSION 1.00 Documents describing a WISHBONE compatible SoC component that are 100% compliant with this specification MAY use the WISHBONE logo.

1.7 Glossary of Terms

0x (numerical prefix) The '0x' prefix indicates a hexadecimal number. It is the same nomenclature as commonly used in the 'C' programming language.

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

Address Tag One or more user defined signals that modify a WISHBONE address. For example, they can be used create a parity bit on an address bus, to indicate an address width (16-bit, 24-bit etc.) or can be used by memory management hardware to indicate a protected address space. All address tags must be assigned a tag type of [TGA_I()] or [TGA_O()]. Also see *tag*, *tag type*, *data tag* and *cycle tag*.

ASIC Acronym for: Application Specific Integrated Circuit. A general term which describes a generic array of logic gates or analog building blocks which are programmed by a metaliza-

tion 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

1. A verb indicating that a logic state has switched from the inactive to the active state. When active high logic is used it means that a signal has switched from a logic low level to a logic high level.
2. *Assert*: to cause a signal line to make a transition from its logically false (inactive) state to its logically true (active) state. Opposite of *negated*.

Bit A single binary (base 2) digit.

Bridge An interconnection system that allows data exchange between two or more buses. The buses may have similar or different electrical, mechanical and logical structures.

Bus

1. A common group of signals.
2. A signal line or a set of lines used by a data transfer system to connect a number of devices.

Bus Interface An electronic circuit that drives or receives data or power from a bus.

Bus Cycle The process whereby digital signals effect the transfer of data across a bus by means of an inter-locked sequence of control signals. Also see: *Phase (bus cycle)*.

BYTE A unit of data that is 8-bits wide. Also see: *WORD*, *DWORD* and *QWORD*.

Crossbar Interconnection (Crossbar Switch) Crossbar switches are mechanisms that allow modules 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 configuration. This creates a configurable and reliable network system. Most crossbar architectures are also scalable, meaning that families of crossbars can be added as the needs arise. A crossbar interconnection is shown in Figure 1.4.

Cycle Tag One or more user defined signals that modify a WISHBONE bus cycle. For example, they can be used to discriminate between WISHBONE SINGLE, BLOCK and RMW cycles. All cycle tags must be assigned a tag type of [TGC_I()] or [TGC_O()]. Also see *tag type*, *address tag* and *data tag*.

Data Flow Interconnection An interconnection where data flows through a prearranged set of IP cores in a sequential order. Data flow architectures often have the advantage of parallelism, whereby two or more functions are executed at the same time. Figure 1.5 shows a data flow interconnection between IP cores.

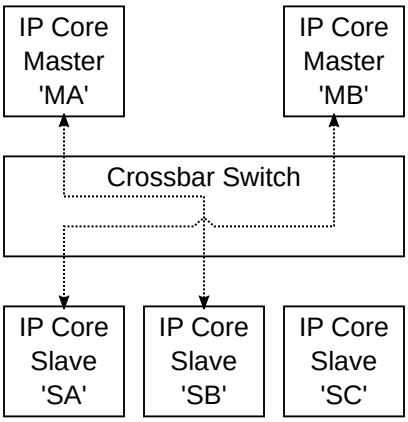


Figure 1.4: Crossbar (switch) interconnection. (Note: Dotted lines indicate one possible connection option).

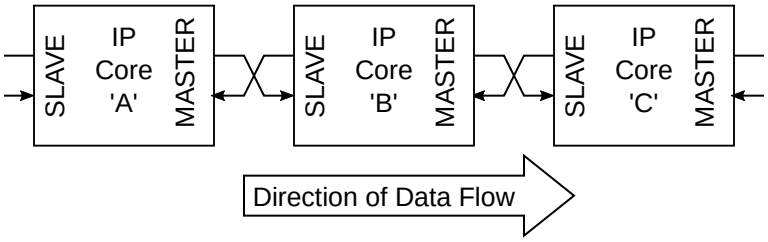


Figure 1.5: Data flow interconnection.

Data Organization The ordering of data during a transfer. Generally, 8-bit (byte) data can be stored with the most significant byte of a mult-byte transfer at the higher or the lower address. These two methods are generally called BIG ENDIAN and LITTLE ENDIAN, respectively. 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 terms BIG ENDIAN and LITTLE ENDIAN for data organization was coined by Danny Cohen of the Information Sciences Institute, and was derived from the book Gulliver's Travels by Jonathan Swift.

Data Tag One or more user defined signals that modify a WISHBONE data transfer. For example, they can be used carry parity information, error correction codes or time stamps. All data tags must be assigned a tag type of [TGD_I()] or [TGD_O()]. Also see *tag type*, *address tag* and *cycle tag*.

DMA Unit Acronym for Direct Memory Access Unit.

1. A device that transfers data from one location in memory to another location in memory.
2. A device for transferring data between a device and memory without interrupting program flow.
3. A device that does not use low-level instructions and is intended for transferring data

between memory and/or I/O locations.

DWORD A unit of data that is 32-bits wide. Also see: *BYTE*, *WORD* and *QWORD*.

ENDIAN See the definition under 'Data Organization'.

FIFO Acronym for: First In First Out. A type of memory used to transfer data between ports on two devices. In FIFO memories, data is removed in the same order that they were added. The FIFO memory is very useful for interconnecting cores of differing speeds.

Firm Core An IP Core that is delivered in a way that allows conversion into an integrated circuit design, but does not allow the design to be easily reverse engineered. It is analogous to a binary or object file in the field of computer software design.

Fixed Interconnection An interconnection system that is fixed, and *cannot* be changed without causing incompatibilities between bus modules (or SoC/IP cores). Also called a *static interconnection*. Examples of fixed interconnection buses include PCI, cPCI and VMEbus. Also see: *variable interconnection*.

Fixed Timing Specification A timing specification that is based upon a fixed set of rules. Generally used in traditional microcomputer buses like PCI and VMEbus. Each bus module must conform to the ridged set of timing specifications. Also see: *variable timing specification*.

Foundry See silicon foundry.

FPGA Acronym for: Field Programmable Gate Array. 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.

Full Address Decoding A method of address decoding where each SLAVE decodes all of the available address space. For example, if a 32-bit address bus is used, then each SLAVE decodes all thirty-two address bits. This technique is used on standard microcomputer buses like PCI and VMEbus. Also see: *partial address decoding*.

Gated Clock A clock that can be stopped and restarted. In WISHBONE, a gated clock generator allows [CLK_O] to be stopped in its low state. This technique is often used to reduce the power consumption of an integrated circuit. Under WISHBONE, the gated clock generator is optional. Also see: *variable clock generator*.

Glue Logic

1. 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.
2. A family of logic circuits consisting of various gates and simple logic elements, each of which serve as an interface between various parts of a computer system.

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*.

Hard Core An IP Core that is delivered in the form of a mask set (i.e. a graphical description of the features and connections in an integrated circuit).

Hardware Description Language (HDL)

1. Acronym for: Hardware Description Language. Examples include VHDL and Verilog®.
2. A general-purpose language used for the design of digital electronic systems.

Interface A combination of signals and data-ports on a module that is capable of either generating or receiving bus cycles. WISHBONE defines these as MASTER and SLAVE interfaces respectively. Also see: *MASTER and SLAVE interfaces*.

INTERCON A WISHBONE module that interconnects MASTER and SLAVE interfaces.

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

Mask Set A graphical description of the features and connections in an integrated circuit.

MASTER A WISHBONE interface that is capable of generating bus cycles. All systems based on the WISHBONE interconnect must have at least one MASTER interface. Also see: *SLAVE*.

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

Minimization (Logic Minimization) A process by which HDL synthesis, router or other software development tools remove unused logic. This is important in WISHBONE because there are optional signals defined on many of the interfaces. If a signal is unused, then the logic minimization tools will remove these signals and their associated logic, thereby making a faster and more efficient design.

Module In the context of this specification, it's another name for an IP core.

Multiplexer Interconnection An interconnection that uses multiplexers to route address, data and control signals. Often used for System-on-Chip (SoC) applications. Also see: *three-state bus interconnection*.

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

Off-Chip Interconnection An off-chip interconnection is used when a WISHBONE interface extends off-chip. See Figure 1.6.

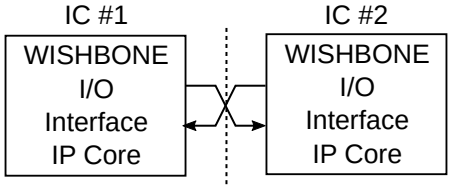


Figure 1.6: Off-chip interconnection.

Operand Size The operand size is the largest single unit of data that is moved through an 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*.

Parametric Core Generator A software tool used for the generation of IP cores based on input parameters. One example of a parametric core generator is a DSP filter generator. These are programs that create lowpass, bandpass and highpass DSP filters. The parameters for the filter are provided by the user, which causes the program to produce the digital filter as a VHDL or Verilog® hardware description. Parametric core generators can also be used create WISHBONE interconnections.

Partial Address Decoding A method of address decoding where each SLAVE decodes only the range of addresses that it requires. For example, if the module needs only four addresses, then it decodes only the two least significant address bits. The remaining address bits are decoded by the interconnection system. This technique is used on SoC buses and has the advantages of less redundant logic in the system. It supports variable address buses, variable interconnection buses, and is relatively fast. Also see: *full address decoding*.

PCI Acronym for: Peripheral Component Interconnect. Generally used as an interconnection scheme between integrated circuits. It also exists as a board level interconnection known as Compact PCI (or cPCI). While this specification is very flexible, it isn't practical for SoC applications.

Phase (Bus Cycle) A periodic portion of a bus cycle. For example, a WISHBONE BLOCK READ cycle could contain ten phases, with each phase transferring a single 32-bit word of data. Collectively, the ten phases form the BLOCK READ cycle.

Point-to-point Interconnection

1. An interconnection system that supports a single WISHBONE MASTER and a single WISHBONE SLAVE interface. It is the simplest way to connect two cores. See Figure 1.7.
2. A connection with only two endpoints.

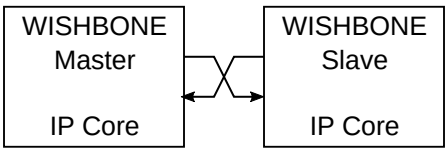


Figure 1.7: Point-to-point interconnection.

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

QWORD A unit of data that is 64-bits wide. Also see: *BYTE*, *WORD* and *DWORD*.

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

RTL

1. Register-transfer logic. A design methodology that 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 where all registers use a common clock.
2. Register-transfer level. A description of computer operations where data transfers from register to register, latch to latch and through logic gates.
3. A level of description of a digital design in which the clocked behavior of the design is expressly described in terms of data transfers between storage elements (which may be implied) and combinatorial logic (which may represent any computing logic or arithmetic-logic-unit). RTL modeling allows design hierarchy that represents a structural description of other RTL models.

Shared Bus Interconnection The shared bus interconnection is a system where a MASTER initiates addressable bus cycles to a target SLAVE. Traditional buses such as VMEbus and PCI bus use this type of interconnection. As a consequence of this architecture, only one MASTER at a time can use the interconnection resource (i.e. bus). Figure Figure 1.8 shows an example of a WISHBONE shared bus inter- connection.

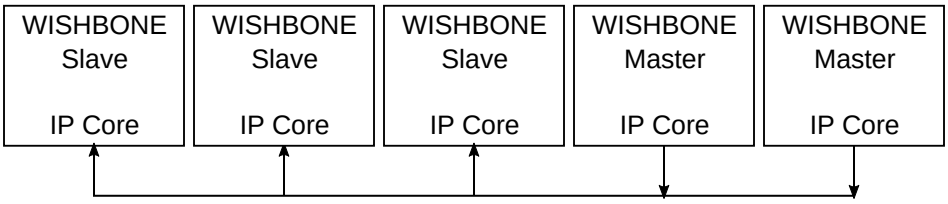


Figure 1.8: Shared bus interconnection.

Silicon Foundry A factory that 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. Also see: *MASTER*.

Soft Core An IP Core that is delivered in the form of a hardware description language or schematic diagram.

SoC Acronym for System-on-Chip. Also see: *System-on-Chip*.

Structured Design

1. A popular method for managing complex projects that is often used with large project teams. When structured design practices are used, individual team members build and test small parts of the design with a common set of tools. Each sub-assembly is designed to a common standard. When all of the sub-assemblies have been completed, the full system can be integrated and tested. This approach makes it much easier to manage the design process.

2. Any disciplined approach to design that adheres to specified rules based on principles such as modularity and top-down design.

Switched Fabric Interconnection A type of interconnection that uses large numbers of crossbar switches. These are organized into arrays that resemble the threads in a fabric. The resulting system is a network of redundant interconnections.

SYSCON A WISHBONE module that drives the system clock [CLK_O] and reset [RST_O] signals.

System-on-Chip (SoC) A method by which whole systems are created on a single integrated circuit chip. In many cases, this requires the use of IP cores which have been designed by multiple IP core providers. System-on-Chip is similar to traditional microcomputer bus systems whereby the individual components are designed, tested and built separately. The components are then integrated to form a finished system.

Tag One or more characters or signals associated with a set of data, containing information about the set. Also see: *tag type*.

Tag Type A special class of signals that is defined to ease user enhancements to the WISHBONE spec. When a user defined signal is specified, it is assigned a tag type that indicates the precise timing to which the signal must conform. This simplifies the creation of new signals. There are three basic tag types. These include address tags, data tags and cycle tags. These allow additional information to be attached to an address transfer, a data transfer or a bus cycle (respectively). The uppercase form TAG TYPE is used when specifying a tag type in the WISHBONE DATASHEET. For example, TAG TYPE: TGA_O() indicates an address tag. Also see: *address tag*, *data tag* and *cycle tag*.

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

Three-State Bus Interconnection A microcomputer bus interconnection that relies upon three-state bus drivers. Often used to reduce the number of interconnecting signal paths through connector and IC pins. Three state buffers can assume a logic low state (‘0’ or ‘L’), a logic high state (‘1’ or ‘H’) or a high impedance state (‘Z’). Three-state buffers are sometimes called Tri-State® buffers. Tri-State® is a registered trademark of National Semiconductor Corporation. Also see: *multiplexer interconnection*.

Variable Clock Generator A type of SYSCON module where the frequency of [CLK_O] can be changed dynamically. The frequency can be changed by way of a programmable phase-lock-loop (PLL) circuit or other control mechanism. Among other things, this technique is used to reduce the power consumption of the circuit. In WISHBONE the variable clock generator capability is optional. Also see: *gated clock generator* and *variable timing specification*.

Variable Interconnection A microcomputer bus interconnection that *can* be changed without causing incompatibilities between bus modules (or SoC/IP cores). Also called a dynamic interconnection. An example of a variable interconnection bus is the WISHBONE SoC architecture. Also see: *fixed interconnection*.

Variable Timing Specification A timing specification that is not fixed. In WISHBONE, variable timing can be achieved in a number of ways. For example, the system integrator can select the frequency of [CLK_O] by enforcing a timing specification during the circuit design. Variable timing can also be achieved during circuit operation with a variable clock generator. Also see: *gated clock generator* and *variable clock generator*.

Verilog® A textual based hardware description language (HDL) intended for use in circuit design. The Verilog® 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, IEEE 1073.3, IEEE 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 SoC applications.

WISHBONE A flexible System-on-Chip (SoC) design methodology. WISHBONE establishes common interface standards for data exchange between modules within an integrated circuit chip. Its purpose is to foster design reuse, portability and reliability of SoC designs. WISHBONE is a public domain standard.

WISHBONE Classic WISHBONE Classic is a high performance System-on-Chip (SoC) interconnect. For zero-wait-state operation it requires that the SLAVE generates an asynchronous cycle termination signal. See chapter 3 for WISHBONE Classic bus cycles. Also see: *WISHBONE Registered Feedback*.

WISHBONE DATASHEET A type of documentation required for WISHBONE compatible IP cores. This helps the end user understand the detailed operation of the core, and how to connect it to other cores. The WISHBONE DATASHEET can be included as part of an IP core technical reference manual, or as part of the IP core hardware description.

WISHBONE Registered Feedback WISHBONE Registered Feedback is a high performace System-on-Chip (SoC) interconnect. It requires that all interface signals are registered. To maintain performance, it introduces a number of novel bus-cycles. See chapter 4 for WISHBONE Registered Feedback bus cycles. Also see: *WISHBONE Classic*.

WISHBONE Signal A signal that is defined as part of the WISHBONE specification. Non-WISHBONE signals can also be used on the IP core, but are not defined as part of this specification. For example, [ACK_O] is a WISHBONE signal, but [CLK100_I] is not.

WISHBONE Logo A logo that, when affixed to a document, indicates that the associated SoC component is compatible with the WISHBONE standard.

Wrapper A circuit element that converts a non-WISHBONE IP Core into a WISHBONE compatible IP Core. For example, consider a 16-byte synchronous memory primitive that is provided by an IC vendor. The memory primitive can be made into a WISHBONE compatible SLAVE by layering a circuit over the memory primitive, thereby creating a WISHBONE compatible SLAVE. A wrapper is analogous to a technique used to convert software written in 'C' to that written in 'C++'.

WORD A unit of data that is 16-bits wide. Also see: *BYTE*, *DWORD* and *QWORD*.

1.8 References

IEEE 100: The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition. IEEE Press 2000.

Feustel, Edward A. "On the Advantages of Tagged Architecture". IEEE Transactions on Computers, Vol. C-22, No. 7, July 1973.

CHAPTER

TWO

INTERFACE SPECIFICATION

This chapter describes the signaling method between the MASTER interface, SLAVE interface, and SYSCON module. 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 for IP Cores

WISHBONE compatible IP cores include documentation that describes the interface. This helps the end user understand the operation of the core, and how to connect it to other cores. This documentation takes the form of a WISHBONE DATASHEET. It can be included as part of the IP core technical reference manual, it can be embedded in source code or it can take other forms as well.

2.1.1 General Requirements for the WISHBONE DATASHEET

RULE 2.00 Each WISHBONE compatible IP core MUST include a WISHBONE DATASHEET as part of the IP core documentation.

RULE 2.15 The WISHBONE DATASHEET for MASTER and SLAVE interfaces MUST include the following information:

1. The revision level of the WISHBONE specification to which is was designed.
2. The type of interface: MASTER or SLAVE.
3. The signal names that are defined for the WISHBONE SoC interface. If a signal name is different than that defined in this specification, then it MUST be cross-referenced to the corresponding signal name which is used in this specification.
4. If a MASTER supports the optional [ERR_I] signal, then the WISHBONE DATASHEET MUST describe how it reacts in response to the signal. If a SLAVE supports the optional [ERR_O] signal, then the WISHBONE DATASHEET MUST describe the conditions under which the signal is generated.

5. If a MASTER supports the optional [RTY_I] signal, then the WISHBONE DATASHEET MUST describe how it reacts in response to the signal. If a SLAVE supports the optional [RTY_O] signal, then the WISHBONE DATASHEET MUST describe the conditions under which the signal is generated.
6. All interfaces that support tag signals MUST describe the name, TAG TYPE and operation of the tag in the WISHBONE DATASHEET.
7. The WISHBONE DATASHEET MUST indicate the port size. MUST be indicated as: 8-bit, 16-bit, 32-bit or 64-bit.
8. The WISHBONE DATASHEET MUST indicate the port granularity. The granularity MUST be indicated as: 8-bit, 16-bit, 32-bit or 64-bit.
9. The WISHBONE DATASHEET MUST indicate the maximum operand size. The maximum operand size MUST be indicated as: 8-bit, 16-bit, 32-bit or 64-bit. If the maximum operand size is unknown, then the maximum operand size shall be the same as the granularity.
10. The WISHBONE DATASHEET MUST indicate the data transfer ordering. The ordering MUST be indicated as BIG ENDIAN or LITTLE ENDIAN. When the port size equals the granularity, then the interface shall be specified as BIG/LITTLE ENDIAN. [When the port size equals the granularity, then BIG ENDIAN and LITTLE ENDIAN transfers are identical].
11. 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 it as UNDEFINED.
12. The WISHBONE DATASHEET MUST indicate if there are any constraints on the [CLK_I] signal. These constraints include (but are not limited to) clock frequency, application specific timing constraints, the use of gated clocks or the use of variable clock generators.

2.1.2 Signal Naming

RULE 2.20 Signal names MUST adhere to the rules of the native tool in which the IP core is designed.

PERMISSION 2.00 Any signal name MAY be used to describe the WISHBONE signals.

OBSERVATION 2.00 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.

RECOMENDATION 2.00 It is recommended that the interface uses the signal names defined in this document.

OBSERVATION 2.05 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.

PERMISSION 2.05 Non-WISHBONE signals MAY be used with IP core interfaces.

OBSERVATION 2.15 Most IP cores will include non-WISHBONE signals. These are outside the scope of this specification, and no attempt is made to govern them. For example, a disk controller IP core could have a WISHBONE interface on one end and a disk interface on the other. In this case the specification does not dictate any technical requirements for the disk interface signals.

2.1.3 Logic Levels

RULE 2.30 All WISHBONE interface signals MUST use active high logic.

OBSERVATION 2.10 In general, the use of active low signals does not present a problem. However, RULE 2.30 is included because some tools (especially schematic entry tools) do not have a standard way of indicating an active low signal. For example, a reset signal could be named [#RST_I], [/RST_I] or [N_RST_I]. This was found to cause confusion among users and incompatibility between modules. This constraint should not create any undue difficulties, as the system integrator can invert any signals before use by the WISHBONE interface.

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.

2.2.1 SYSCON Module Signals

CLK_O

The system clock output [CLK_O] is generated by the SYSCON module. It coordinates all activities for the internal logic within the WISHBONE interconnect. The INTERCON module connects the [CLK_O] output to the [CLK_I] input on MASTER and SLAVE interfaces.

RST_O

The reset output [RST_O] is generated by the SYSCON module. It forces all WISHBONE interfaces to restart. All internal self-starting state machines are forced into an initial state. The

INTERCON connects the [RST_O] output to the [RST_I] input on MASTER and SLAVE interfaces.

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 are stable before the rising edge of [CLK_I].

DAT_I()

The data input array [DAT_I()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]). Also see the [DAT_O()] and [SEL_O()] signal descriptions.

DAT_O()

The data output array [DAT_O()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]). Also see the [DAT_I()] and [SEL_O()] signal descriptions.

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. This signal only resets the WISHBONE interface. It is not required to reset other parts of an IP core (although it may be used that way).

TGD_I()

Data tag type [TGD_I()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data input array [DAT_I()], and is qualified by signal [STB_I]. For example, parity protection, error correction and time stamp information can be attached to the data bus. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification. The name and operation of a data tag must be defined in the WISHBONE DATASHEET.

TGD_O()

Data tag type [TGD_O()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data output array [DAT_O()], and is qualified by signal [STB_O]. For example, parity protection, error correction and time stamp information can be attached to the data bus. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification. The name and operation of a data tag must be defined in the WISHBONE DATASHEET.

2.2.2 MASTER Signals

ACK_I

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

ADR_O()

The address output array [ADR_O()] is used to pass a binary address. The higher array boundary is specific to the address width of the core, and the lower array boundary is determined by the data port size and granularity. For example the array size on a 32-bit data port with BYTE granularity is [ADR_O(n..2)]. 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.

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. Also see the [ACK_I] and [RTY_I] signal descriptions.

LOCK_O

The lock output [LOCK_O] when asserted, indicates that the current bus cycle is uninterruptible. Lock is asserted to request complete ownership of the bus. Once the transfer has started, the INTERCON does not grant the bus to any other MASTER, until the current MASTER negates [LOCK_O] or [CYC_O].

RTY_I

The retry input [RTY_I] 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. Also see the [ERR_I] and [RTY_I] signal descriptions.

SEL_O()

The select output array [SEL_O()] indicates where valid data is expected on the [DAT_I()] signal array during READ cycles, and where it is placed on the [DAT_O()] signal array during WRITE cycles. The array boundaries are determined by the granularity of a port. For example, if 8-bit granularity is used on a 64-bit port, then there would be an array of eight select signals with boundaries of [SEL_O(7..0)]. Each individual select signal correlates to one of eight active bytes on the 64-bit data port. For more information about [SEL_O()], please refer to the data organization

section in Chapter 3 of this specification. Also see the [DAT_I()], [DAT_O()] 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()]. The SLAVE asserts either the [ACK_I], [ERR_I] or [RTY_I] signals in response to every assertion of the [STB_O] signal.

TGA_O()

Address tag type [TGA_O()] contains information associated with address lines [ADR_O()], and is qualified by signal [STB_O]. For example, address size (24-bit, 32-bit etc.) and memory management (protected vs. unprotected) information can be attached to an address. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is defined by this specification. The name and operation of an address tag must be defined in the WISHBONE DATASHEET.

TGC_O()

Cycle tag type [TGC_O()] contains information associated with bus cycles, and is qualified by signal [CYC_O]. For example, data transfer, interrupt acknowledge and cache control cycles can be uniquely identified with the cycle tag. They can also be used to discriminate between WISHBONE SINGLE, BLOCK and RMW cycles. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is defined by this specification. The name and operation of a cycle tag must be defined 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.

2.2.3 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()

The address input array [ADR_I()] is used to pass a binary address. The higher array boundary is specific to the address width of the core, and the lower array boundary is determined by the data port size. For example the array size on a 32-bit data port with BYTE granularity is [ADR_O(n..2)]. 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.

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. Also see the [ACK_O] and [RTY_O] signal descriptions.

LOCK_I

The lock input [LOCK_I], when asserted, indicates that the current bus cycle is uninterruptible. A SLAVE that receives the LOCK [LOCK_I] signal is accessed by a single MASTER only, until either [LOCK_I] or [CYC_I] is negated.

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. Also see the [ERR_O] and [RTY_O] signal descriptions.

SEL_I()

The select input array [SEL_I()] indicates where valid data is placed on the [DAT_I()] signal array during WRITE cycles, and where it should be present on the [DAT_O()] signal array during READ cycles. The array boundaries are determined by the granularity of a port. For example, if 8-bit granularity is used on a 64-bit port, then there would be an array of eight select signals with boundaries of [SEL_I(7..0)]. Each individual select signal correlates to one of eight active bytes on the 64-bit data port. For more information about [SEL_I()], please refer to the data organization section in Chapter 3 of this specification. Also see the [DAT_I(63..0)], [DAT_O(63..0)] and [STB_I] signal descriptions.

STB_I

The strobe input [STB_I], when asserted, indicates that the SLAVE is selected. A SLAVE shall respond to other WISHBONE signals only when this [STB_I] is asserted (except for the [RST_I] signal which should always be responded to). The SLAVE asserts either the [ACK_O], [ERR_O] or [RTY_O] signals in response to every assertion of the [STB_I] signal.

TGA_I

Address tag type [TGA_I()] contains information associated with address lines [ADR_I()], and is qualified by signal [STB_I]. For example, address size (24-bit, 32-bit etc.) and memory management (protected vs. unprotected) information can be attached to an address. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification. The name and operation of an address tag must be defined in the WISHBONE DATASHEET.

TGC_I()

Cycle tag type [TGC_I()] contains information associated with bus cycles, and is qualified by signal [CYC_I]. For example, data transfer, interrupt acknowledge and cache control cycles can be

uniquely identified with the cycle tag. They can also be used to discriminate between WISHBONE SINGLE, BLOCK and RMW cycles. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification. The name and operation of a cycle tag must be defined in the WISHBONE DATASHEET.

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

THREE

WISHBONE CLASSIC BUS CYCLE

WISHBONE Classic bus cycles are described in terms of their general operation, reset operation, handshaking protocol and the data organization during transfers. Additional requirements for bus cycles (especially those relating to the common clock) can be found in the timing specifications in Chapter 5.

3.1 General Operation

MASTER and SLAVE interfaces are interconnected with a set of signals that permit them to exchange data. For descriptive purposes these signals are cumulatively known as a *bus*, and are contained within a functional module called the INTERCON. Address, data and other information is impressed upon this bus in the form of *bus cycles*.

3.1.1 Reset Operation

All hardware interfaces are initialized to a pre-defined state. This is accomplished with the reset signal [RST_O] that can be asserted at any time. It is also used for test simulation purposes by initializing all self-starting state machines and counters which may be used in the design. The reset signal [RST_O] is driven by the SYSCON module. It is connected to the [RST_I] signal on all MASTER and SLAVE interfaces. Figure 3.1 shows the reset cycle.

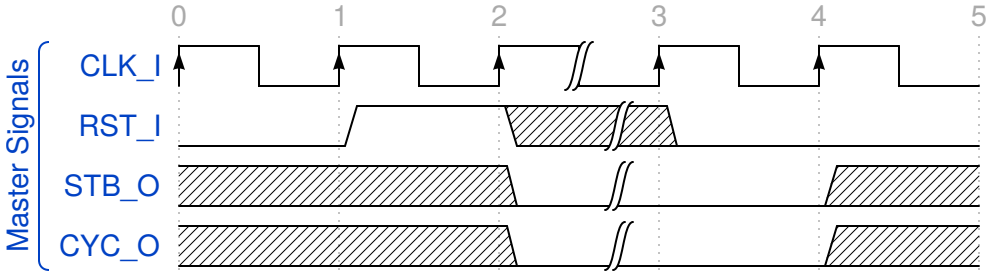


Figure 3.1: Reset cycle.

RULE 3.00 All WISHBONE interfaces MUST initialize themselves at the rising [CLK_I] edge following the assertion of [RST_I]. They MUST stay in the initialized state until the rising [CLK_I] edge that follows the negation of [RST_I].

RULE 3.05 [RST_I] MUST be asserted for at least one complete clock cycle on all WISHBONE interfaces.

PERMISSION 3.00 [RST_I] MAY be asserted for more than one clock cycle, and MAY be asserted indefinitely.

RULE 3.10 All WISHBONE interfaces MUST be capable of reacting to [RST_I] at any time.

RULE 3.15 All self-starting state machines and counters in WISHBONE interfaces MUST initialize themselves at the rising [CLK_I] edge following the assertion of [RST_I]. They MUST stay in the initialized state until the rising [CLK_I] edge that follows the negation of [RST_I].

OBSERVATION 3.00 In general, self-starting state machines do 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. Furthermore, self-starting state machines can go through an indeterminate number of initialization cycles before finding their starting state, thereby making it difficult to predict their behavior at start-up time. The initialization rule prevents both problems by forcing all state machines to a pre-defined state in response to the assertion of [RST_I].

RULE 3.20 The following MASTER signals MUST be negated at the rising [CLK_I] edge following the assertion of [RST_I], and MUST stay in the negated state until the rising [CLK_I] edge that follows the negation of [RST_I]: [STB_O], [CYC_O]. The state of all other MASTER signals are undefined in response to a reset cycle.

OBSERVATION 3.05 On MASTER interfaces [STB_O] and [CYC_O] may be asserted beginning at the rising [CLK_I] edge following the negation of [RST_I].

OBSERVATION 3.10 SLAVE interfaces automatically negate [ACK_O], [ERR_O] and [RTY_O] when their [STB_I] is negated.

RECOMENDATION 3.00 Design SYSCON modules so that they assert [RST_O] during a power-up condition. [RST_O] should remain asserted until all voltage levels and clock frequencies in the system are stabilized. When negating [RST_O], do so in a synchronous manner that conforms to this specification.

OBSERVATION 3.15 If a gated clock generator is used, and if the clock is stopped, then the WISHBONE interface is not capable of responding to its [RST_I] signal.

SUGGESTION 3.00 Some circuits require an *asynchronous* reset capability. If an IP core or other SoC component requires an asynchronous reset, then define it as a non-WISHBONE signal. This prevents confusion with the WISHBONE reset [RST_I] signal that uses a purely synchronous protocol, and needs to be applied to the WISHBONE interface only.

OBSERVATION 3.20 All WISHBONE *interfaces* respond to the reset signal. However, the IP Core connected to a WISHBONE interface does not necessarily need to respond to the reset signal.

3.1.2 Transfer Cycle initiation

MASTER interfaces initiate a transfer cycle by asserting [CYC_O]. When [CYC_O] is negated, all other MASTER signals are invalid. SLAVE interfaces respond to other SLAVE signals only when [CYC_I] is asserted. SYSCON signals and responses to SYSCON signals are not affected.

RULE 3.25 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].

PERMISSION 3.05 MASTER interfaces MAY assert [CYC_O] indefinitely.

RECOMMENDATION 3.05 Arbitration logic often uses [CYC_I] to select between MASTER interfaces. Keeping [CYC_O] asserted may lead to arbitration problems. It is therefore recommended that [CYC_O] is not indefinitely asserted.

RULE 3.30 SLAVE interfaces MAY NOT respond to any SLAVE signals when [CYC_I] is negated. However, SLAVE interfaces MUST always respond to SYSCON signals.

3.1.3 Handshaking Protocol

All bus cycles use a handshaking protocol between the MASTER and SLAVE interfaces. As shown in Figure 3.2, 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. This gives both MASTER and SLAVE interfaces the possibility to control the rate at which data is transferred.

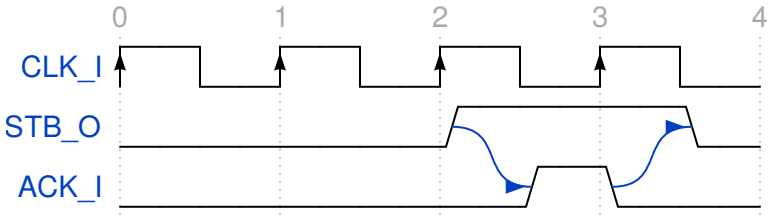


Figure 3.2: Local bus handshaking protocol.

PERMISSION 3.10 If the SLAVE guarantees it can keep pace with all MASTER interfaces and if the [ERR_I] and [RTY_I] signals are not used, then the SLAVE's [ACK_O] signal MAY be tied to the logical AND of the SLAVE's [STB_I] and [CYC_I] inputs. The interface will function normally under these circumstances.

OBSERVATION 3.25 SLAVE interfaces assert a cycle termination signal in response to [STB_I]. However, [STB_I] is only valid when [CYC_I] is valid. RULE 3.35 The cycle termination signals [ACK_O], [ERR_O], and [RTY_O] must be generated in response to the logical AND of [CYC_I] and [STB_I].

PERMISSION 3.15 Other signals, besides [CYC_I] and [STB_I], MAY be included in the generation of the cycle termination signals.

OBSERVATION 3.30 Internal SLAVE signals also determine what cycle termination signal is asserted and when it is asserted.

Most of the examples in this 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 MASTER 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 asserts [RTY_I] if the local resource is busy. This specification does not dictate when or how the MASTER will respond to [RTY_I].

RULE 3.40 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], [CYC_I], [STB_I], and [RST_I]. All other signals are optional.

PERMISSION 3.20 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 an error has occurred during the bus cycle. This specification does not dictate what the MASTER does in response to [ERR_I].

PERMISSION 3.25 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 3.45 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].

OBSERVATION 3.35 If the SLAVE supports the [ERR_O] or [RTY_O] signals, but the MASTER does not support these signals, deadlock may occur.

RECOMMENDATION 3.10 Design INTERCON modules to prevent deadlock conditions. One solution to this problem is to include a watchdog timer function that monitors the MASTER's [STB_O] signal, and asserts [ERR_I] or [RTY_I] if the cycle exceeds some pre-defined

time limit. INTERCON modules can also be designed to disconnect interfaces from the WISHBONE bus if they constantly generate bus errors and/or watchdog time-outs.

RECOMMENDATION 3.15 Design WISHBONE MASTER interfaces so that there are no intermediate logic gates between a registered flip-flop and the signal outputs on [STB_O] and [CYC_O]. Delay timing for [STB_O] and [CYC_O] are very often the most critical paths in the system. This prevents sloppy design practices from slowing down the interconnect because of added delays on these two signals.

RULE 3.50 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_I].

PERMISSION 3.30 The assertion of [ACK_O], [ERR_O], and [RTY_O] MAY be asynchronous to the [CLK_I] signal (i.e. there is a combinatorial logic path between [STB_I] and [ACK_O]).

OBSERVATION 3.40 The asynchronous assertion of [ACK_O], [ERR_O], and [RTY_O] assures that the interface can accomplish one data transfer per clock cycle. Furthermore, it simplifies the design of arbiters in multi-MASTER applications.

OBSERVATION 3.45 The asynchronous assertion of [ACK_O], [ERR_O], and [RTY_O] could proof impossible to implement. For example slave wait states are easiest implemented using a registered [ACK_O] signal.

OBSERVATION 3.50 In large high speed designs the asynchronous assertion of [ACK_O], [ERR_O], and [RTY_O] could lead to unacceptable delay times, caused by the loopback delay from the MASTER to the SLAVE and back to the MASTER. Using registered [ACK_O], [ERR_O], and [RTY_O] signals significantly reduces this loopback delay, at the cost of one additional wait state per transfer. See WISHBONE Registered Feedback Bus Cycles for methods of eliminating the wait state.

PERMISSION 3.35 Under certain circumstances SLAVE interfaces MAY be designed to hold [ACK_O] in the asserted state. This situation occurs on point-to-point interfaces where there is a single SLAVE on the interface, and that SLAVE always operates without wait states.

RULE 3.55 MASTER interfaces MUST be designed to operate normally when the SLAVE interface holds [ACK_I] in the asserted state.

3.1.4 Use of [STB_O]

RULE 3.60 MASTER interfaces MUST qualify the following signals with [STB_O]: [ADR_O], [DAT_O()], [SEL_O()], [WE_O], and [TAGN_O].

PERMISSION 3.40 If a MASTER doesn't generate wait states, then [STB_O] and [CYC_O] MAY be assigned the same signal.

OBSERVATION 3.55 [CYC_O] needs to be asserted during the entire transfer cycle. A MASTER that doesn't generate wait states doesn't negate [STB_O] during a transfer cycle, i.e.

it is asserted the entire transfer cycle. Therefore it is allowed to use the same signal for [CYC_O] and [STB_O]. Both signals must be present on the interface though.

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

RULE 3.65 SLAVE interfaces MUST qualify the following signals with [ACK_O], [ERR_O] or [RTY_O]: [DAT_O()].

3.1.6 Use of TAG TYPES

The WISHBONE interface can be modified with user defined signals. This is done with a technique known as tagging. Tags are a well known concept in the microcomputer bus industry. They allow user defined information to be associated with an address, a data word or a bus cycle. All tag signals must conform to set of guidelines known as TAG TYPEs. Table 3.1 lists all of the defined TAG TYPEs along with their associated data set and signal waveform. When a tag is added to an interface it is assigned a TAG TYPE from the table. This explicitly defines how the tag operates. This information must also be included in the WISHBONE DATASHEET.

Table 3.1: TAG TYPEs

	MASTER		SLAVE	
Description	TAG TYPE	Associated with	TAG TYPE	Associated with
Address tag	TGA_O()	ADR_O()	TGA_I()	ADR_I()
Data tag, input	TGD_I()	DAT_I()	TGD_I()	DAT_I()
Data tag, output	TGD_O()	DAT_O()	TGD_O()	DAT_O()
Cycle tag	TGC_O()	Bus Cycle	TGC_I()	Bus Cycle

For example, consider a MASTER interface where a parity protection bit named [PAR_O] is generated from an output data word on [DAT_O(15..0)]. It's an 'even' parity bit, meaning that it's asserted whenever there are an even number of '1's in the data word. If this signal were added to the interface, then the following information (in the WISHBONE DATASHEET) would be sufficient to completely define the timing of [PAR_O]:

SIGNAL NAME: PAR_O

DESCRIPTION: Even parity bit

MASTER TAG TYPE: TGD_O()

RULE 3.70 All user defined tags MUST be assigned a TAG TYPE. Furthermore, they MUST adhere to the timing specifications given in this document for the indicated TAG TYPE.

PERMISSION 3.45 While all TAG TYPES are specified as arrays (with parenthesis '()'), the actual tag MAY be a non-arrayed signal.

RECOMMENDATION 3.15 If a MASTER interface supports more than one defined bus cycle over a common set of signal lines, then include a cycle tag to identify each type of bus cycle. This allows INTERCON and SLAVE interface circuits to discriminate between these bus cycles (if needed). Define the signals as TAG TYPE: [TGC_O()], using signal names of [SGL_O], [BLK_O] and [RMW_O] when identifying SINGLE, BLOCK and RMW cycles respectively.

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. Note that the [CYC_O] signal isn't shown here to keep the timing diagrams as simple as possible. It is assumed that [CYC_O] is continuously asserted.

RULE 3.75 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 3.50 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 3.3 shows a SINGLE READ cycle. The bus protocol works as follows:

CLOCK EDGE 0: MASTER presents a valid address on [ADR_O()] and [TGA_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 [TGC_O()] to indicate the start of the cycle.

MASTER asserts [STB_O] to indicate the start of the phase.

SETUP, EDGE 1: SLAVE decodes inputs, and responding SLAVE asserts [ACK_I].

SLAVE presents valid data on [DAT_I()] and [TGD_I()].

SLAVE asserts [ACK_I] in response to [STB_O] to indicate valid data.

MASTER monitors [ACK_I], and prepares to latch data on [DAT_I()] and [TGD_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()] and [TGD_I()].

MASTER negates [STB_O] and [CYC_O] to indicate the end of the cycle.

SLAVE negates [ACK_I] in response to negated [STB_O].

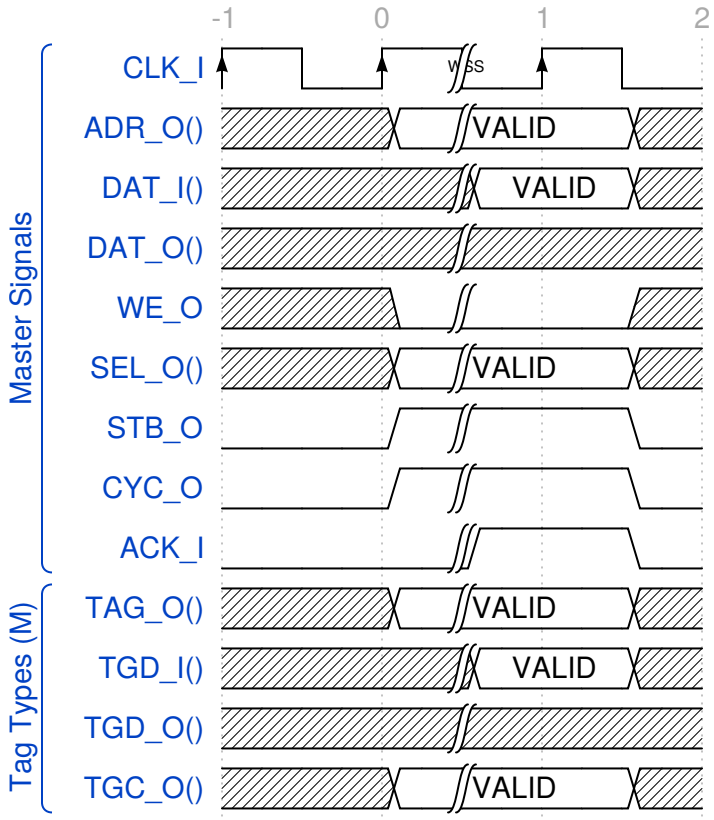


Figure 3.3: SINGLE READ cycle.

3.2.2 SINGLE WRITE Cycle

Figure 3.4 shows a SINGLE WRITE cycle. The bus protocol works as follows:

CLOCK EDGE 0: MASTER presents a valid address on [ADR_O()] and [TGA_O()].

MASTER presents valid data on [DAT_O()] and [TGD_O()].

MASTER asserts [WE_O] to indicate a WRITE cycle.

MASTER presents bank select [SEL_O()] to indicate where it sends data.

MASTER asserts [CYC_O] and [TGC_O()] to indicate the start of the cycle.

MASTER asserts [STB_O] to indicate the start of the phase.

SETUP, EDGE 1: SLAVE decodes inputs, and responding SLAVE asserts [ACK_I].

SLAVE prepares to latch data on [DAT_O()] and [TGD_O()].

SLAVE asserts [ACK_I] in response to [STB_O] to indicate latched 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()] and [TGD_O()].

MASTER negates [STB_O] and [CYC_O] to indicate the end of the cycle.

SLAVE negates [ACK_I] in response to negated [STB_O].

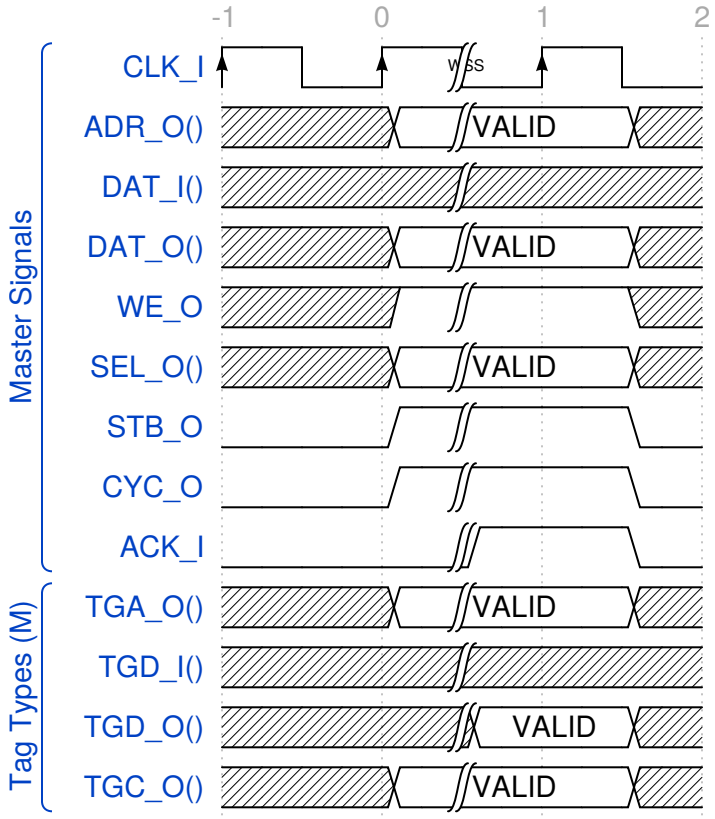


Figure 3.4: SINGLE WRITE cycle.

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 (called *phases*) 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 3.5, the [CYC_O] signal is asserted for the duration of a BLOCK cycle. This signal can be used to request permission to access a shared resource from a local arbiter. To hold the access until the end of the cycle the [LOCK_O] signal must be asserted, as is shown. During each of the data transfer phases (within the block transfer), the normal handshaking protocol between [STB_O] and [ACK_I] is maintained.

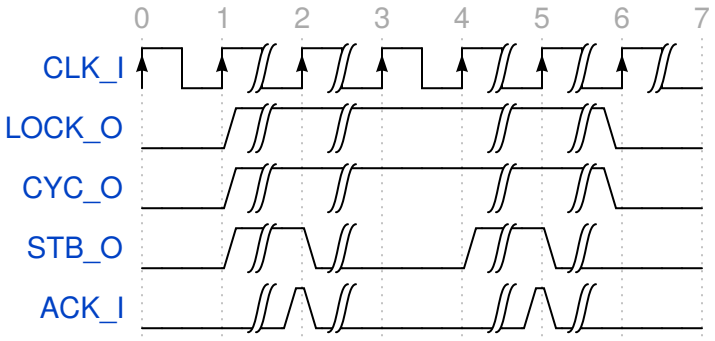


Figure 3.5: Use of [CYC_O] signal during BLOCK cycles.

RULE 3.80 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 3.55 MASTER and SLAVE interfaces MAY be designed so that they do not support the BLOCK cycles.

3.3.1 BLOCK READ Cycle

Figure 3.6 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 interfaces can both throttle the bus transfer rate by inserting wait states. A total of five transfers (phases) 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 a valid address on [ADR_O()] and [TGA_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 [TGC_O()] to indicate the start of the cycle.

MASTER asserts [STB_O] to indicate the start of the first phase.

Note: the MASTER asserts [CYC_O] and/or [TGC_O()] at, or anytime before, clock edge 1.

SETUP, EDGE 1: SLAVE decodes inputs, and responding SLAVE asserts [ACK_I].

SLAVE presents valid data on [DAT_I()] and [TGD_I()].

MASTER monitors [ACK_I], and prepares to latch [DAT_I()] and [TGD_I()].

CLOCK EDGE 1: MASTER latches data on [DAT_I()] and [TGD_I()].

MASTER presents new [ADR_O()] and [TGA_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()] and [TGD_I()].

MASTER monitors [ACK_I], and prepares to latch [DAT_I()] and [TGD_I()].

CLOCK EDGE 2: MASTER latches data on [DAT_I()] and [TGD_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.

CLOCK EDGE 3: MASTER presents new [ADR_O()] and [TGA_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()] and [TGD_I()].

MASTER monitors [ACK_I], and prepares to latch [DAT_I()] and [TGD_I()].

CLOCK EDGE 4: MASTER latches data on [DAT_I()] and [TGD_I()].

MASTER presents [ADR_O()] and [TGA_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()] and [TGD_I()].

MASTER monitors [ACK_I], and prepares to latch [DAT_I()] and [TGD_I()].

CLOCK EDGE 5: MASTER latches data on [DAT_I()] and [TGD_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()] and [TGD_I()].

MASTER monitors [ACK_I], and prepares to latch [DAT_I()] and [TGD_I()].

CLOCK EDGE 6: MASTER latches data on [DAT_I()] and [TGD_I()].

MASTER terminates cycle by negating [STB_O] and [CYC_O].

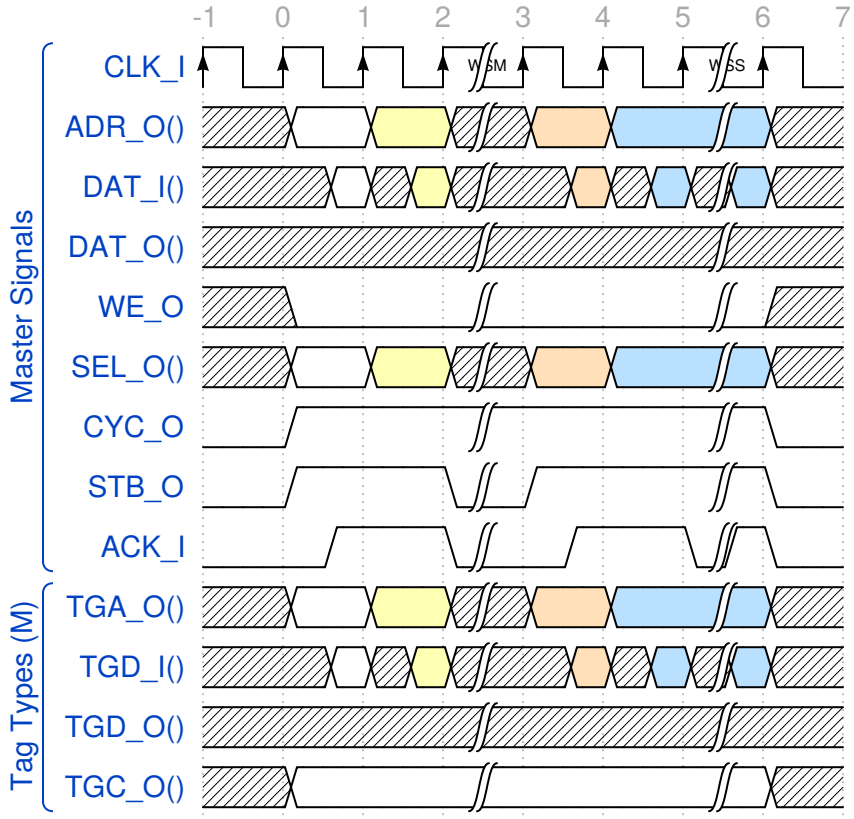


Figure 3.6: BLOCK READ cycle.

3.3.2 BLOCK WRITE Cycle

Figure 3.7 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 interfaces 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 [TGA_O()].

MASTER asserts [WE_O] to indicate a WRITE cycle.

MASTER presents bank select [SEL_O()] to indicate where it sends data.

MASTER asserts [CYC_O] and [TGC_O()] to indicate cycle start.

MASTER asserts [STB_O].

Note: the MASTER asserts [CYC_O] and/or [TGC_O()] at, or anytime before, clock edge 1.

SETUP, EDGE 1: SLAVE decodes inputs, and responds by asserting [ACK_I].

SLAVE prepares to latch data on [DAT_O()] and [TGD_O()].

MASTER monitors [ACK_I], and prepares to terminate current data phase.

CLOCK EDGE 1: SLAVE latches data on [DAT_O()] and [TGD_O()].

MASTER presents [ADR_O()] and [TGA_O()].

MASTER presents new bank select [SEL_O()] to indicate where it sends data.

SETUP, EDGE 2: SLAVE decodes inputs, and responds by asserting [ACK_I].

SLAVE prepares to latch data on [DAT_O()] and [TGD_O()].

MASTER monitors [ACK_I], and prepares to terminate current data phase.

CLOCK EDGE 2: SLAVE latches data on [DAT_O()] and [TGD_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 [TGA_O()].

MASTER presents bank select [SEL_O()] to indicate where it sends 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()] and [TGD_O()].

MASTER monitors [ACK_I], and prepares to terminate data phase.

CLOCK EDGE 4: SLAVE latches data on [DAT_O()] and [TGD_O()].

MASTER presents [ADR_O()] and [TGA_O()].

MASTER presents new bank select [SEL_O()] to indicate where it sends data.

SETUP, EDGE 5: SLAVE decodes inputs, and responds by asserting [ACK_I].

SLAVE prepares to latch data on [DAT_O()] and [TGD_O()].

MASTER monitors [ACK_I], and prepares to terminate data phase.

CLOCK EDGE 5: SLAVE latches data on [DAT_O()] and [TGD_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()] and [TGD_O()].

MASTER monitors [ACK_I], and prepares to terminate data phase.

CLOCK EDGE 6: SLAVE latches data on [DAT_O()] and [TGD_O()].

MASTER terminates cycle by negating [STB_O] and [CYC_O].

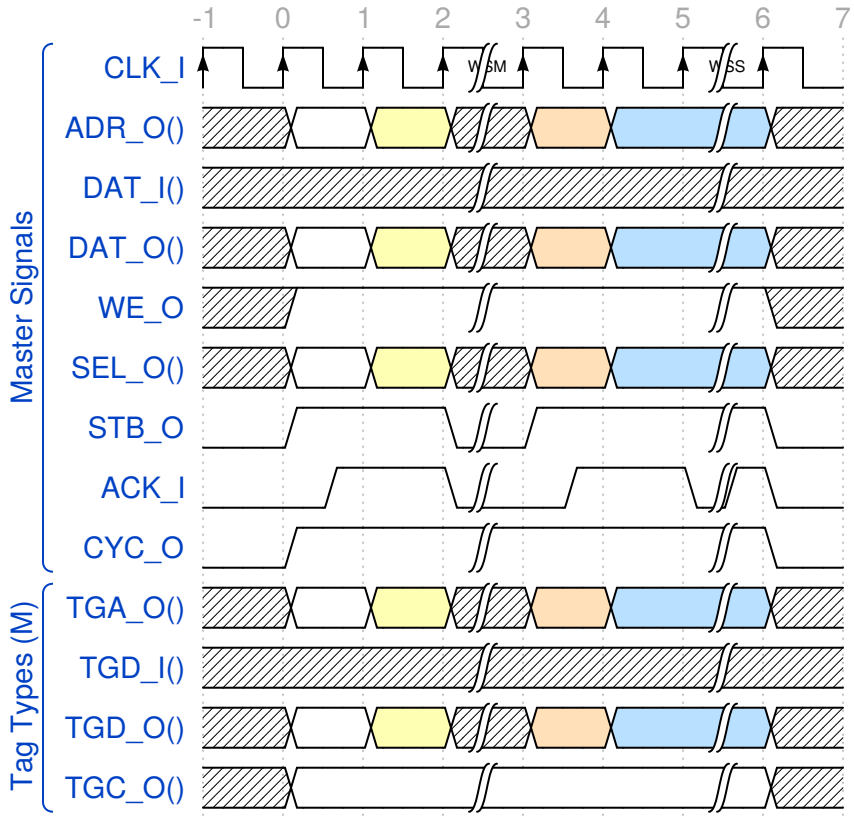


Figure 3.7: BLOCK WRITE cycle.

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 3.85 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 3.8 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 interfaces 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 [TGA_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 [TGC_O()] to indicate the start of cycle.

MASTER asserts [STB_O].

Note: the MASTER asserts [CYC_O] and/or [TGC_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()] and [TGD_I()].

MASTER monitors [ACK_I], and prepares to latch [DAT_I()] and [TGD_I()].

CLOCK EDGE 1: MASTER latches data on [DAT_I()] and [TGD_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 WRITE data on [DAT_O()] and [TGD_O()].

MASTER presents new bank select [SEL_O()] to indicate where it sends data.

MASTER asserts [STB_O].

SETUP, EDGE 3: SLAVE decodes inputs, and responds by asserting [ACK_I].

SLAVE prepares to latch data on [DAT_O()] and [TGD_O()].

MASTER monitors [ACK_I], and prepares to terminate data phase.

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

CLOCK EDGE 3: SLAVE latches data on [DAT_O()] and [TGD_O()].

MASTER negates [STB_O] and [CYC_O] indicating the end of the cycle.

SLAVE negates [ACK_I] in response to negated [STB_O].

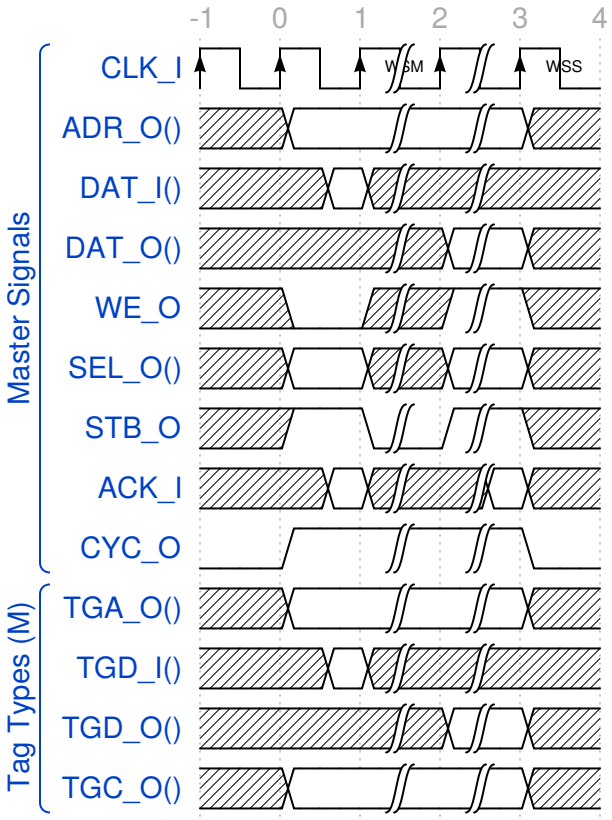


Figure 3.8: RMW cycle.

3.5 Data Organization

Data organization refers to the ordering of data during transfers. There are two general types of ordering. These 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 3.2. Figure Figure 3.9 shows the operand locations for input and output data ports.

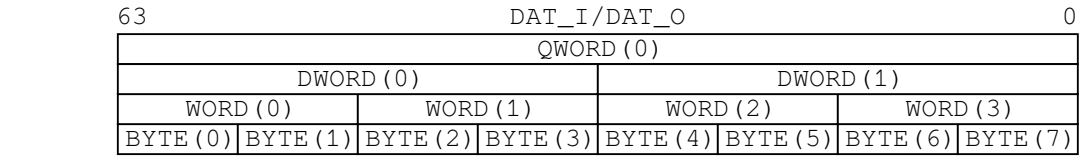
Table 3.2: Data Transfer Nomenclature

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'.

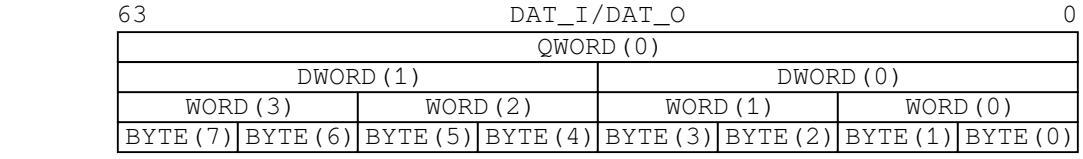
The table also defines the granularity of the interface. This indicates the minimum unit of data transfer that is supported by the interface. For example, the smallest operand that can be passed through a port with 16-bit granularity is a 16-bit WORD. In this case, an 8-bit operand cannot be transferred.

Figure 3.10 shows an example of how the 64-bit value of 0x0123456789ABCDEF 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(63) / DAT_O(63). The least significant bit is at DAT_I(0) / 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.

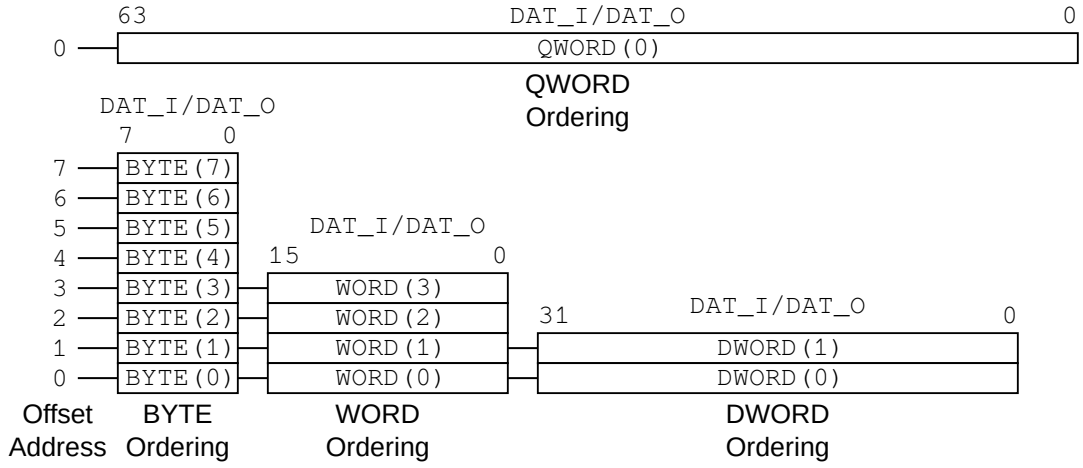
Figure 3.11 shows an example of how the 64-bit value of 0x0123456789ABCDEF is transferred through BYTE, WORD, DWORD and QWORD ports using LITTLE ENDIAN data organization. Through the 64-bit QWORD port the number is directly transferred with the most significant bit at DAT_I(63) / DAT_O(63). The least significant bit is at DAT_I(0) / 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 3.9: Operand locations for input and output data ports.

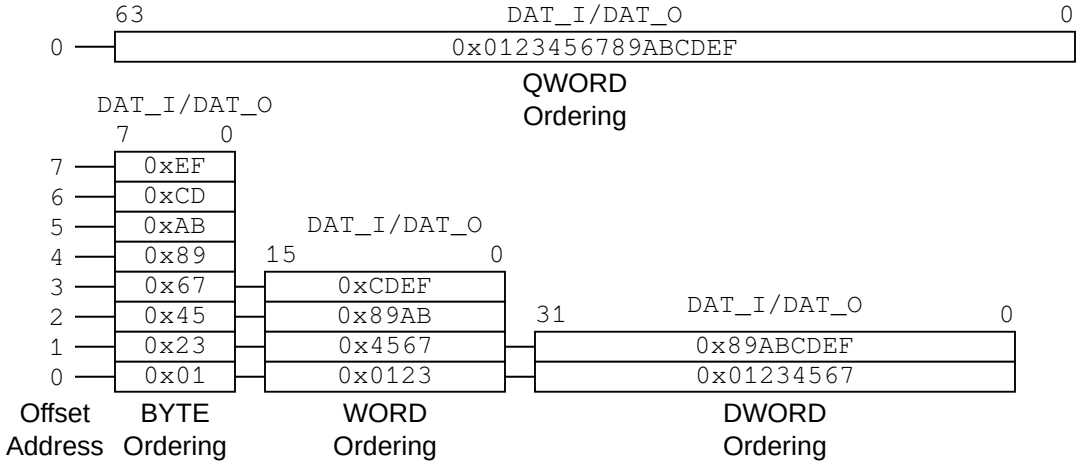


Figure 3.10: Example showing a variety of BIG ENDIAN transfers over various port sizes.

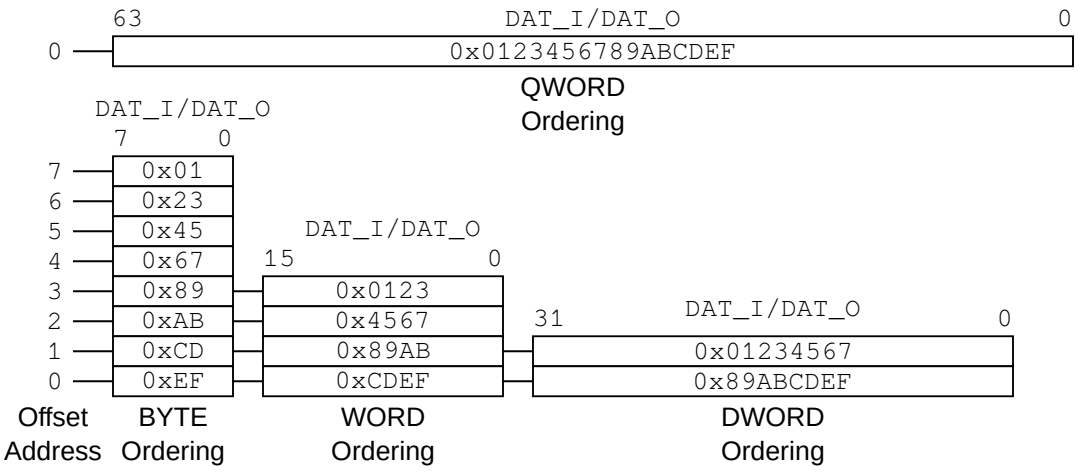


Figure 3.11: Example showing a variety of LITTLE ENDIAN transfers over various port sizes.

RULE 3.90 Data organization MUST conform to the ordering indicated in Figure 3.9.

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 3.20 Design interfaces so that data is transferred sequentially from lower addresses to higher addresses.

OBSERVATION 3.60 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.

3.5.3 Data Organization for 64-bit Ports

RULE 3.95 Data organization on 64-bit ports MUST conform to Figure 3.12.

3.5.4 Data Organization for 32-bit Ports

RULE 3.100 Data organization on 32-bit ports MUST conform to Figure 3.13.

3.5.5 Data Organization for 16-bit Ports

RULE 3.105 Data organization on 16-bit ports MUST conform to Figure 3.14.

64-bit Data Bus With 8-bit (BYTE) Granularity

Address Range ADR_I ADR_O	Active Portion of Data Bus							
	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)
Ordering: Big Endian	BYTE(0)	BYTE(1)	BYTE(2)	BYTE(3)	BYTE(4)	BYTE(5)	BYTE(6)	BYTE(7)
Ordering: 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 ADR_I ADR_O	Active Portion of Data Bus			
	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)
Ordering: Big Endian	WORD(0)	WORD(1)	WORD(2)	WORD(3)
Ordering: Little Endian	WORD(3)	WORD(2)	WORD(1)	WORD(0)

64-bit Data Bus With 32-bit (DWORD) Granularity

Address Range ADR_I ADR_O	Active Portion of Data Bus	
	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)
Ordering: Big Endian	DWORD(0)	DWORD(1)
Ordering: Little Endian	DWORD(1)	DWORD(0)

64-bit Data Bus With 64-bit (QWORD) Granularity

Address Range ADR_I ADR_O	Active Portion of Data Bus
	DAT_I DAT_O (63..00)
Active Select Line	SEL_I(0) SEL_O(0)
Ordering: Big Endian	QWORD(0)
Ordering: Little Endian	QWORD(0)

Figure 3.12: Data organization for 64-bit ports.

32-bit Data Bus With 8-bit (BYTE) Granularity

Address Range ADR_I ADR_O (63..02)		Active Portion of Data Bus			
		DAT_I DAT_O (31..24)	DAT_I DAT_O (23..16)	DAT_I DAT_O (15..00)	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)
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 ADR_I ADR_O (63..01)		Active Portion of Data Bus	
		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)
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 ADR_I ADR_O (63..00)		Active Portion of Data Bus
		DAT_I DAT_O (31..00)
Active Select Line		SEL_I(0) SEL_O(0)
Ordering	Big Endian	DWORD(0) DWORD(1)
	Little Endian	DWORD(1) DWORD(0)

Figure 3.13: Data organization for 32-bit ports.

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)	Active Portion of Data Bus: DAT_I DAT_O (07..00)
	Active Select Line	SEL_I(1) SEL_O(1)	SEL_I(0) SEL_O(0)
Ordering	Big Endian	BYTE(0)	BYTE(1)
		BYTE(2)	BYTE(3)
		BYTE(4)	BYTE(5)
		BYTE(6)	BYTE(7)
	Little Endian	BYTE(1)	BYTE(0)
		BYTE(3)	BYTE(2)
		BYTE(5)	BYTE(4)
		BYTE(7)	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)
Ordering	Big Endian	WORD(0)
		WORD(1)
		WORD(2)
		WORD(3)
	Little Endian	WORD(0)
		WORD(1)
		WORD(2)
		WORD(3)

Figure 3.14: Data organization for 16-bit ports.

3.5.6 Data Organization for 8-bit Ports

RULE 3.110 Data organization on 8-bit ports MUST conform to Figure 3.15.

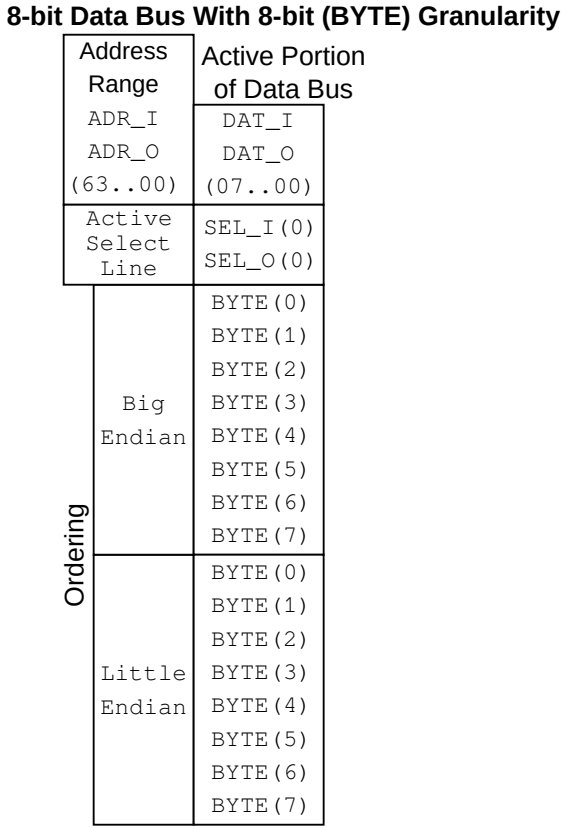


Figure 3.15: Data organization for 8-bit ports.

3.6 References

Cohen, Danny. *On Holy Wars and a Plea for Peace*. IEEE Computer Magazine, October 1981. Pages 49-54. [Description of BIG ENDIAN and LITTLE ENDIAN.]

CHAPTER

FOUR

WISHBONE REGISTERED FEEDBACK BUS CYLCES

4.1 Introduction, Synchronous vs. Asynchronous cycle termination

To achieve the highest possible throughput, WISHBONE Classic requires asynchronous cycle termination signals. This results in an asynchronous loop from the MASTER, through the INTERCONN to the SLAVE, and then from the SLAVE through the INTERCONN back to the MASTER, as shown in Figure 4.1. In large System-on-Chip devices this routing delay between MASTER and SLAVE is the dominant timing factor. This is especially true for deep sub-micron technologies.

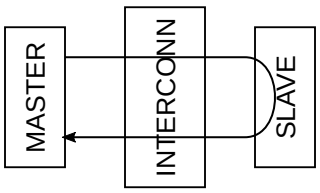


Figure 4.1: Asynchronous cycle termination path

The simplest solution for reducing the delay is to cut the loop, by using synchronous cycle termination signals. However, this introduces a wait state for every transfer, as shown in Figure 4.2.

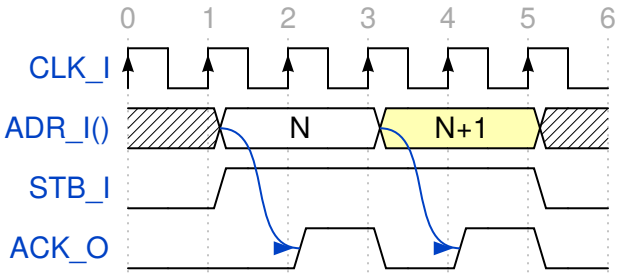


Figure 4.2: WISHBONE Classic synchronous cycle terminated burst

During cycle-1 the MASTER initiates a transfer. The addressed SLAVE responds in the next cycle with the assertion of ACK_O. During cycle-3 the MASTER initiates a second cycle, addressing

the same SLAVE. Because the SLAVE does not know in advance it is being addressed again, it has to negate ACK_O. At the earliest it can respond in cycle-4, after which it has to negate ACK_O again in cycle-5.

Each transfer takes two WISHBONE cycles to complete, thus only half of the available bandwidth is useable. If the SLAVE would know in advance that it is being addressed again, it could already respond in cycle-3. Decreasing the amount of cycles needed to perform the transfers, and thus increasing throughput. The waveforms for that cycle are as shown in Figure 4.3.

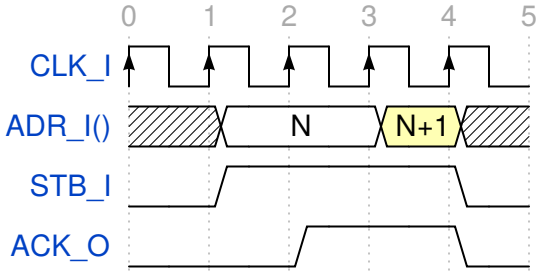


Figure 4.3: Advanced synchronous terminated burst

During cycle-1 the MASTER initiates a transfer. The addressed SLAVE responds in the next cycle with the assertion of ACK_O. The MASTER starts a new transfer in cycle-3. The SLAVE knows in advance it is being addressed again, therefore it keeps ACK_O asserted.

A two cycle burst now takes three cycles to complete, instead of four. This is a throughput increase of 33%. WISHBONE Classic however would require only 2 cycles. An eight cycle burst takes nine cycles to complete, instead of 16. This is a throughput increase of 77%. WISHBONE Classic would require eight cycles. For single transfers there is no performance gain.

Table 4.1: Burst comparison

Burst length	Asynchronous Cycle termination	Synchronous Cycle termination	Advanced Synchronous Cycle termination
1	1 (200%)	2 (100%)	2
2	2 (150%)	4 (75%)	3
4	4 (125%)	8 (62%)	5
8	8 (112%)	16 (56%)	9
16	16 (106%)	32 (53%)	17
32	32 (103%)	64 (51%)	33

Table 4.1 shows a comparison between the discussed cycle termination types, for zero wait state bursts at a given bus-frequency. Asynchronous cycle termination requires only one cycle per transfer, synchronous cycle termination requires two cycles per transfer, and the advanced synchronous cycle termination requires (burst_length+1) cycles. The percentages show the relative throughput for a burst length, where the advanced synchronous cycle termination is set to 100%.

Advanced synchronous cycle termination appears to get the best from both the synchronous and asynchronous termination schemes. For single transfers it performs as well as the normal syn-

chronous termination scheme, for large bursts it performs as well as the asynchronous termination scheme.

NOTE that for a system that already needs wait states, the advanced synchronous scheme provides the same throughput as the asynchronous scheme.

A given system, with an average burst length of 8, is intended to run at over 150MHz. It is shown that moving from asynchronous termination to synchronous termination would improve timing by 1.5ns. Thus allowing a 193MHz clock frequency, instead of the 150MHz.
The asynchronous termination scheme has a theoretical throughput of 150Mcycles per sec. For the given average burst length of 8, the advanced synchronous termination scheme has a 12% lower theoretical throughput than the asynchronous termination scheme. However the increased operating frequency allows it to perform more cycles per second. The theoretical throughput for the advanced synchronous scheme is 193M / 1.12 = 172Mcycles per sec.

System layout requires that all block have registered outputs. The average burst length used in the system is 4.
Moving to the advanced synchronous termination scheme improves performance by 60 %.

4.2 WISHBONE Registered Feedback

WISHBONE Registered Feedback bus cycles use the Cycle Type Identifier [CTI_O()], [CTI_I()] Address Tags to implement the advanced synchronous cycle termination scheme. Both MASTER and SLAVE interfaces must support [CTI_O()], [CTI_I()] in order to provide the improved bandwidth. Additional information about the type of burst is provided by the Burst Type Extension [BTE_O()], [BTE_I()] Address Tags. Because WISHBONE Registered Feedback uses Tag signals to implement the advanced synchronous cycle termination, it is inherently fully compatible with WISHBONE Classic. If only one of the interfaces (i.e. either MASTER or SLAVE) supports WISHBONE Registered Feedback bus cycles, and hence the other supports WISHBONE Classic bus cycles, the cycle terminates as though it were a WISHBONE Classic bus cycle. This eases the integration of WISHBONE Classic and WISHBONE Registered Feedback IP cores.

PERMISSION 4.00 MASTER and SLAVE interfaces MAY be designed to support WISHBONE Registered Feedback bus cycles.

RECOMMENDATION 4.00 Interfaces compatible with WISHBONE Registered Feedback bus cycles support both WISHBONE Classic and WISHBONE Registered Feedback bus cycles. It is recommended to design new IP cores to support WISHBONE Registered Feedback bus cycles, so as to ensure maximum throughput in all systems.

RULE 4.00 All WISHBONE Registered Feedback compatible cores MUST support WISHBONE Classic bus cycles.

4.3 Signal Description

CTI_IO()

The Cycle Type Idenfier [CTI_IO()] Address Tag provides additional information about the current cycle. The MASTER sends this information to the SLAVE. The SLAVE can use this information to prepare the response for the next cycle. Table 4.2 Type Identifiers

Table 4.2: Cycle Type Identifiers

CTI_IO(2:0)	Description
'000'	Classic cycle.
'001'	Constant address burst cycle
'010'	Incrementing burst cycle
'011'	*Reserved*
'100'	*Reserved*
'101	*Reserved*
'110'	*Reserved*
'111'	End-of-Burst

PERMISSION 4.05 MASTER and SLAVE interfaces MAY be designed to support the [CTI_I()] and [CTI_O()] signals. Also MASTER and SLAVE interfaces MAY be designed to support a limited number of burst types.

RULE 4.05 MASTER and SLAVE interfaces that do support the [CTI_I()] and [CTI_O()] signals MUST at least support the Classic cycle [CTI_IO()='000'] and the End-of-Cycle [CTI_IO()='111'].

RULE 4.10 MASTER and SLAVE interfaces that are designed to support a limited number of burst types MUST complete the unsupported cycles as though they were WISHBONE Classic cycle, i.e. [CTI_IO()= '000'].

PERMISSION 4.10 For description languages that allow default values for input ports (like VHDL), [CTI_I()] MAY be assigned a default value of '000'.

PERMISSION 4.15 In addition to the WISHBONE Classic rules for generating cycle termination signals [ACK_O], [RTY_O], and [ERR_O], a SLAVE MAY assert a termination cycle without checking the [STB_I] signal.

OBSERVATION 4.00 To avoid the inherent wait state in synchronous termination schemes, the SLAVE must generate the response as soon as possible (i.e. the next cycle). It can use the [CTI_I()] signals to determine the response for the next cycle. But it cannot determine the state of [STB_I] for the next cycle, therefore it must generate the response independent of [STB_I].

PERMISSION 4.20 [ACK_O], [RTY_O], and [ERR_O] MAY be asserted while [STB_O] is negated.

RULE 4.15 A cycle terminates when both the cycle termination signal and [STB_I], [STB_O] is asserted. Even if [ACK_O], [ACK_I] is asserted, the other signals are only valid when [STB_O], [STB_I] is also asserted.

BTE_IO() The Burst Type Extension [BTE_O()] Address Tag is send by the MASTER to the SLAVE to provides additional information about the current burst. Currently this information is only relevant for incrementing bursts, but future burst types may use these signals.

Table 4.3: Type Extension for Incrementing and Decrementing bursts

BTE_IO(1:0)	Description
'00'	Linear burst
'01'	4-beat wrap burst
'10'	8-beat wrap burst
'11'	16-beat wrap burst

RULE 4.20 MASTER and SLAVE interfaces that support incrementing burst cycles MUST support the [BTE_O()] and [BTE_I()] signals.

PERMISSION 4.25 MASTER and SLAVE interfaces MAY be designed to support a limited number of burst extensions.

RULE 4.25 MASTER and SLAVE interfaces that are designed to support a limited number of burst extensions MUST complete the unsupported cycles as though they were WISHBONE Classic cycle, i.e. [CTI_IO()= 000'].

4.4 Bus Cycles

4.4.1 Classic Cycle

A Classic Cycle indicates that the current cycle is a WISHBONE Classic cycle. The SLAVE terminates the cycle as described in chapter 3. There is no information about what the MASTER will do the next cycle.

PERMISSION 4.30 A MASTER MAY signal Classic Cycle indefinitely.

OBSERVATION 4.05 A MASTER that signals Classic Cycle indefinitely is a pure WISHBONE Classic MASTER. The Cycle Type Identifier [CTI_O()] signals have no effect; all SLAVE interfaces already support WISHBONE Classic cycles. They might as well not be present on the interface at all. In fact, routing them on chip may use up valuable resources. However they might be useful for arbitration logic, or to keep the buses from/to interfaces coherent.

Figure 4.4 shows a Classic read cycle. A total of two transfers are shown. The cycle is terminated after the second transfer. The protocol for this cycle works as follows:

CLOCK EDGE 0: MASTER presents [ADR_O()].

MASTER presents Classic Cycle on [CTI_O()].

MASTER negates [WE_O] to indicate a READ cycle.

MASTER presents select [SEL_O()] to indicate where it expects data.

MASTER asserts [CYC_O] to indicate cycle start.

MASTER asserts [STB_O].

SETUP, EDGE 1: SLAVE decodes inputs.

SLAVE recognizes Classic Cycle and prepares response.

SLAVE prepares to send data.

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 1: SLAVE asserts [ACK_I]

SLAVE presents data on [DAT_I()].

SETUP, EDGE 2: SLAVE does not expect another transfer.

MASTER prepares to latch data on [DAT_I()].

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 2: SLAVE negates [ACK_I].

MASTER latches data on [DAT_I()]

MASTER presents new address on [ADR_O()]

SETUP, EDGE 3: SLAVE decodes inputs.

SLAVE recognizes Classic Cycle and prepares response.

SLAVE prepares to send data.

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 3: SLAVE asserts [ACK_I]

SLAVE presents data on [DAT_I()].

SETUP, EDGE 4: SLAVE does not expect another transfer.

MASTER prepares to latch data on [DAT_I()].

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 4: SLAVE negates [ACK_I].

MASTER latches data on [DAT_I()]

MASTER negates [CYC_O] and [STB_O] ending the cycle

Todo: Does SEL_O really stay constant between accesses?

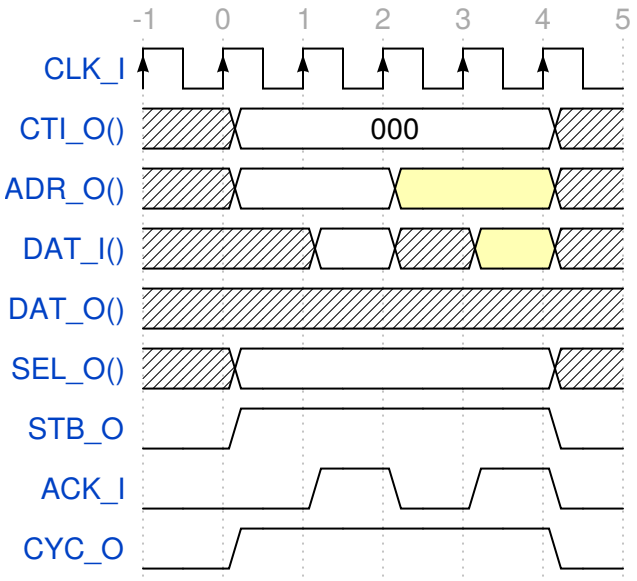


Figure 4.4: Classic Cycle

4.4.2 End-Of-Burst

End-Of-Burst indicates that the current cycle is the last of the current burst. The MASTER signals the slave that the burst ends after this transfer.

RULE 4.30 A MASTER MUST set End-Of-Burst to signal the end of the current burst.

PERMISSION 4.35 The MASTER MAY start a new cycle after the assertion of End-Of-Burst.

PERMISSION 4.40 A MASTER MAY use End-Of-Burst to indicate a single access.

OBSERVATION 4.05 A single access is in fact a burst with a burst length of one.

Figure 4.5 demonstrates the usage of End-Of-Burst. A total of three transfers are shown. The first transfer is part of a WISHBONE Registered Feedback read burst. Transfer two is the last transfer of that burst. The burst is ended when the MASTER sets [CTI_O()] to End-Of-Burst ('111'). The cycle is terminated after the third transfer, a single write transfer. The protocol for this cycle works as follows:

SETUP EDGE 0: WISHBONE Registered Feedback burst read cycle is in progress.

MASTER prepares to latch data on [DAT_I()]

MASTER monitors [ACK_I] and prepares to terminate current data phase.

MASTER prepares to end current burst

SLAVE expects another cycle and prepares response

CLOCK EDGE 0: MASTER latches data on [DAT_I()]

MASTER presents new [ADR_O()]

MASTER presents End-Of-Burst on [CTI_O()]

SLAVE presents new data on [DAT_I()]

SLAVE keeps [ACK_I] asserted to indicate that it is ready to send new data

SETUP EDGE 1: SLAVE decodes inputs.

SLAVE recognizes End-Of-Burst and prepares to terminate burst

SLAVE prepares to send last data.

MASTER prepares to latch data on [DAT_I()]

MASTER monitors [ACK_I] and prepares to terminate current data phase.

MASTER prepares to start a new cycle

CLOCK EDGE 1: MASTER latches data on [DAT_I()]

MASTER starts new cycle by presenting End-Of-Burst on [CTI_O()]

MASTER presents new address on [ADR_O()]

MASTER presents data on [DAT_O()]

MASTER asserts [WE_O] to indicate a WRITE cycle

SLAVE negates [ACK_I]

SETUP, EDGE 2: SLAVE decodes inputs

SLAVE recognizes End-Of-Burst and prepares for a single transfer.

SLAVE prepares response.

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 2: SLAVE asserts [ACK_I].

SETUP, EDGE 3: SLAVE prepares to latch data on [DAT_O()]

SLAVE prepares to end cycle.

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 3: SLAVE latches data on [DAT_O()]

SLAVE negates [ACK_I]

MASTER negates [CYC_O] and [STB_O] ending the cycle.

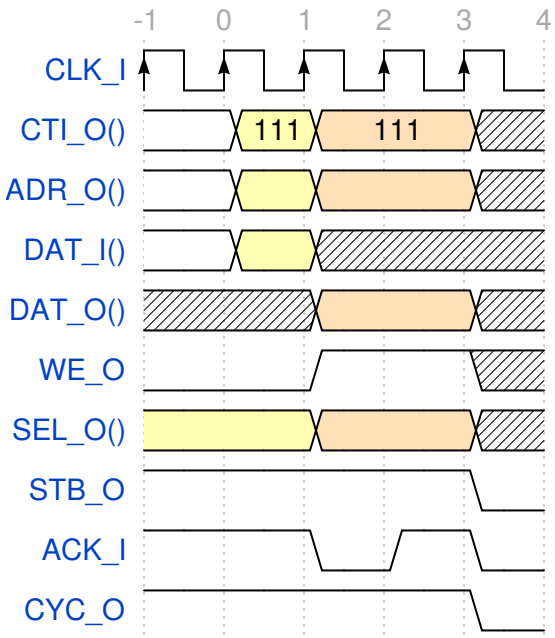


Figure 4.5: End-of-Burst

4.4.3 Constant Address Burst Cycle

A constant address burst is defined as a single cycle with multiple accesses to the same address. Example: A MASTER reading a stream from a FIFO.

RULE 4.35 A MASTER signaling a constant address burst MUST initiate another cycle, the next cycle MUST be the same operation (either read or write), the select lines [SEL_O()] MUST have the same value, and that the address array [ADR_O()] MUST have the same value.

PERMISSION 4.40 When the MASTER signals a constant address burst, the SLAVE MAY assert the termination signal for the next cycle as soon as the current cycle terminates.

Figure 4.6 shows a CONSTANT ADDRESS BURST write cycle. After the initial setup cycle, the Constant Address Burst cycle is capable of a data transfer on every clock cycle. However, this example also shows how the MASTER and the SLAVE interfaces can both throttle the bus transfer rate by inserting wait states. A total of four transfers are shown. After the first transfer the MASTER inserts a wait state. After the second transfer the SLAVE inserts a wait state. The cycle is terminated after the fourth transfer. The protocol for this transfer works as follows:

CLOCK EDGE 0: MASTER presents [ADR_O()].

MASTER presents Constant Address Burst on [CTI_O()].

MASTER asserts [WE_O] to indicate a WRITE cycle.

MASTER presents select [SEL_O()] to indicate where it sends data.

MASTER asserts [CYC_O] to indicate cycle start.

MASTER asserts [STB_O].

SETUP, EDGE 1: SLAVE decodes inputs.

SLAVE recognizes Constant Address Burst and prepares response.

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 1: SLAVE asserts [ACK_I]

SETUP, EDGE 2: SLAVE expects another transfer and prepares response for new transfer.

SLAVE prepares to latch data on [DAT_O()].

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 2: SLAVE latches data on [DAT_O()].

SLAVE keeps [ACK_I] asserted to indicate that it's ready to latch new data.

MASTER inserts wait states by negating [STB_O].

NOTE: any number of wait states can be inserted here.

SETUP, EDGE 3: MASTER is ready to transfer data again.

CLOCK, EDGE 3: MASTER presents [SEL_O].

MASTER presents new [DAT_O()].

MASTER asserts [STB_O].

SETUP, EDGE 4: SLAVE prepares to latch data on [DAT_O()]

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK, EDGE 4: SLAVE latches data on [DAT_O()].

SLAVE inserts wait states by negating [ACK_I].

MASTER presents new [DAT_O()].

NOTE: any number of wait states can be inserted here.

SETUP, EDGE 5: SLAVE is ready to transfer data again.

MASTER monitors [ACK_I] and prepares to terminate current data phase.

MASTER prepares to signal last transfer.

CLOCK, EDGE 5: SLAVE asserts [ACK_I].

SETUP, EDGE 6: SLAVE prepares to latch data on [DAT_O()].

SLAVE expects another transfer and prepares response for new transfer.

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK, EDGE 6: SLAVE latches data on [DAT_O()].

SLAVE keeps [ACK_I] asserted to indicate that it's ready to latch new data.

MASTER presents new [DAT_O()].

MASTER presents End-Of-Burst on [CTI_O()].

SETUP, EDGE 7: SLAVE prepares to latch last data of burst on [DAT_O()]

MASTER monitors [ACK_I] and prepares to terminate current cycle.

CLOCK, EDGE 7: SLAVE latches data on [DAT_O()].

SLAVE ends burst by negating [ACK_I].

MASTER negates [CYC_O] and [STB_O] ending the burst cycle.

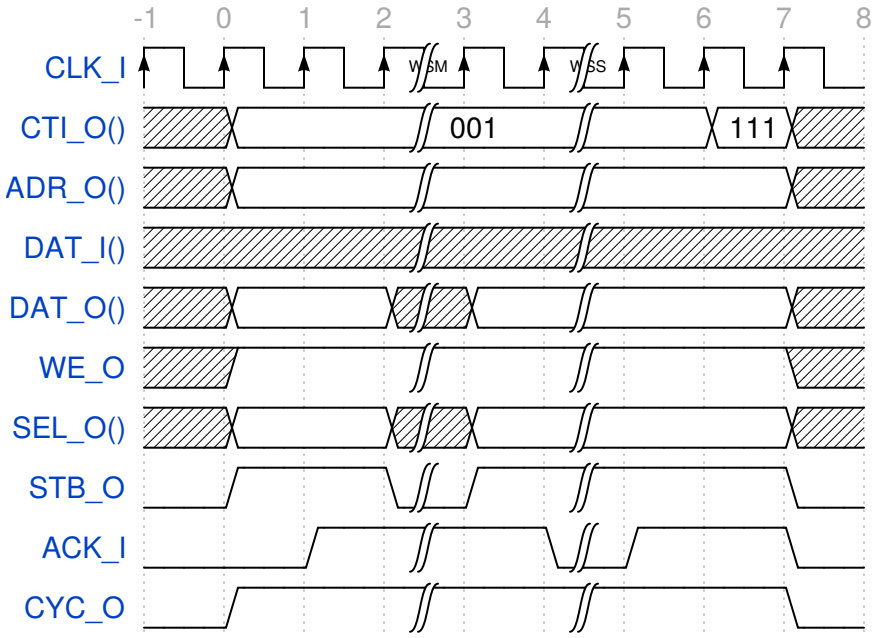


Figure 4.6: Constant address burst

4.4.4 Incrementing Burst Cycle

An incrementing burst is defined as multiple accesses to consecutive addresses. Each transfer the address is incremented. The increment is dependent on the data array [DAT_O()], [DAT_I()] size; for an 8bit data array the increment is 1, for a 16bit data array the increment is 2, for a 32bit data array the increment is 4, etc.

Increments can be linear or wrapped. Linear increments means the next address is one increment more than the current address. Wrapped increments means that the address increments one, but that the addresses' LSBs are modulo the wrap size.

Table 4.4: Wrap Size address increments

Starting address' LSBs	Linear	Wrap-4	Wrap-8
000	0-1-2-3-4-5-6-7	0-1-2-3-4-5-6-7	0-1-2-3-4-5-6-7
001	1-2-3-4-5-6-7-8	1-2-3-0-5-6-7-4	1-2-3-4-5-6-7-0
010	2-3-4-5-6-7-8-9	2-3-0-1-6-7-4-5	2-3-4-5-6-7-0-1
011	3-4-5-6-7-8-9-A	3-0-1-2-7-4-5-6	3-4-5-6-7-0-1-2
100	4-5-6-7-8-9-A-B	4-5-6-7-8-9-A-B	4-5-6-7-0-1-2-3
101	5-6-7-8-9-A-B-C	5-6-7-4-9-A-B-8	5-6-7-0-1-2-3-4
110	6-7-8-9-A-B-C-D	6-7-4-5-A-B-8-9	6-7-0-1-2-3-4-5
111	7-8-9-A-B-C-D-E	7-4-5-6-B-8-9-A	7-0-1-2-3-4-5-6

Example: Processor cache line read

RULE 4.40 A MASTER signaling an incrementing burst MUST initiate another cycle, the next cycle MUST be the same operation (either read or write), the select lines [SEL_O()] MUST have the same value, the address array [ADR_O()] MUST be incremented, and the wrap size MUST be set by the burst type extension [BTE_O()] signals.

PERMISSION 4.45 When the MASTER signals an incrementing burst, the SLAVE MAY assert the termination signal for the next cycle as soon as the current cycle terminates.

Figure 4.7 shows a 4-beat wrapped INCREMENTING BURST read cycle. A total of four transfers are shown. The protocol for this cycle works as follows:

CLOCK EDGE 0: MASTER presents [ADR_O()]

MASTER presents Incrementing Burst on [CTI_O()]

MASTER present 4-beat wrap on [BTE_O()]

MASTER negates [WE_O] to indicate a READ cycle

MASTER presents select [SEL_O()] to indicate where it expects data

MASTER asserts [CYC_O] to indicate cycle start

MASTER asserts [STB_O]

SETUP, EDGE 1: SLAVE decodes inputs.

SLAVE recognizes Incrementing Burst and prepares response.

MASTER prepares to latch data on [DAT_I()]

MASTER monitors [ACK_I] and prepares to terminate current data phase.

CLOCK EDGE 1: SLAVE asserts [ACK_I]

SLAVE present data on [DAT_I()]

SETUP, EDGE 2: MASTER prepares to latch data on [DAT_I()]

MASTER monitors [ACK_I] and prepares to terminate current data phase.

SLAVE expects another transfer and prepares response.

CLOCK EDGE 2: MASTER latches data on [DAT_I()]

MASTER presents new address on [ADR_O()]

SLAVE presents new data on [DAT_I()]

SLAVE keeps [ACK_I] asserted to indicate that it's ready to send new data.

SETUP, EDGE 3: MASTER prepares to latch data on [DAT_I()]

MASTER monitors [ACK_I] and prepares to terminate current data phase.

SLAVE expects another transfer and prepares response.

CLOCK, EDGE 3: MASTER latches data on [DAT_I()].

MASTER presents new address on [ADR_O()]

SLAVE presents new data on [DAT_I()].

SLAVE keeps [ACK_I] asserted to indicate that it's ready to send new data.

SETUP, EDGE 4: MASTER prepares to latch data on [DAT_I()]

MASTER monitors [ACK_I] and prepares to terminate current data phase.

SLAVE expects another transfer and prepares response.

CLOCK, EDGE 4: MASTER latches data on [DAT_I()].

MASTER presents new address on [ADR_O()]

MASTER presents End-Of-Burst on [CTI_O()].

SLAVE presents new data on [DAT_I()].

SLAVE keeps [ACK_I] asserted to indicate that it's ready to send new data.

SETUP, EDGE 5: MASTER prepares to latch data on [DAT_I()]

MASTER monitors [ACK_I] and prepares to terminate current data phase.

SLAVE recognizes End-Of-Burst and prepares to terminate burst.

CLOCK, EDGE 5: MASTER latches data on [DAT_I()].

MASTER negates [CYC_O] and [STB_O] ending burst cycle

SLAVE ends burst by negates [ACK_I]

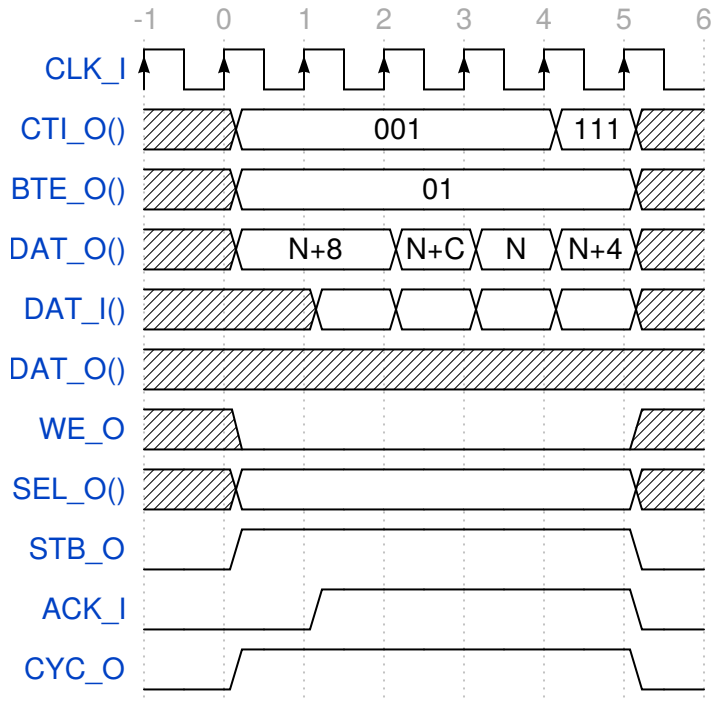


Figure 4.7: 4-beat wrapped incrementing burst for a 32bit data array

CHAPTER

FIVE

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 5.1, and is defined as Tpd,clk-su.

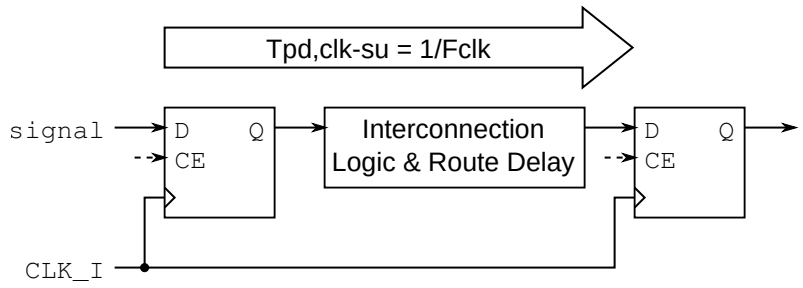


Figure 5.1: Definition for Tpd,clk-su.

RULE 5.00 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 5.00 The user's place and route tool MAY be used to enforce RULE 5.00.

OBSERVATION 5.00 Most place and route tools can be easily configured to enforce RULE 5.00. Generally, it only requires a single timing specification for Tpd,clk-su.

RULE 5.05 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 5.05 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 5.10 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 5.05 The IP core connected to a WISHBONE interface MAY include application specific timing requirements.

RULE 5.10 The clock input [CLK_I] MUST have a duty cycle that is no less than 40%, and no greater than 60%.

PERMISSION 5.10 The SYSCON module MAY use a variable clock generator. In these cases the clock frequency can be changed by the SYSCON module so long as the clock edges remain clean and monotonic, and if the clock does not violate the duty cycle requirements.

PERMISSION 5.15 The SYSCON module MAY use a gated clock generator. In these cases the clock shall be stopped in the low logic state. When the gated clock is stopped and started the clock edges are required to remain clean and monotonic.

SUGGESTION 5.00 When using a gated clock generator, turn the clock off when the WISHBONE interconnection is not busy. One way of doing this is to create a MASTER interface whose sole purpose is to acquire the WISHBONE interconnection and turn the clock off. This assures that the WISHBONE interconnection is not busy when gating the clock off. When the clock signal is restored the MASTER then releases the WISHBONE interconnection.

OBSERVATION 5.15 This specification does not attempt to govern the design of gated or variable clock generators.

SUGGESTION 5.10 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

SIX

CITED PATENT REFERENCES

This chapter contains a partial list of the patents that have been reviewed by the WISHBONE steward and others. In the opinion of the steward, the WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores does not infringe on any of these patents. However, the possibility exists that an integrated circuit device designed to the WISHBONE specification could infringe on the intellectual property rights of others. The user assumes all responsibility for determining if their WISHBONE design infringes on the rights of others. All of these documents contain information relevant to SoC design and integration. Noteworthy patents are marked with a '(*)'.

This list is maintained for three reasons. First, a public domain specification can't depend on patented ideas (unless permission to use the patent is obtained). This is the list of documents that have been reviewed to see if WISHBONE infringes on any known patents. Second, it provides a starting point for IC designers and other researchers who need to know if a specific SoC design infringes on the rights of others. Third, the patent database is a wonderful place to learn how other people have solved similar SoC problems. Patent documents are very good in this regard, as they must fully describe how to reproduce an invention.

6.1 General Methods Relating to SoC

Hartmann, Alfred C. - US Patent No. 6,096,091 DYNAMICALLY RECONFIGURABLE LOGIC NETWORKS INTERCONNECTED BY FALL-THROUGH FIFOS FOR FLEXIBLE PIPELINE PROCESSING IN A SYSTEM-ON-A- CHIP

Luk et al. - US Patent No. 5,790,839 SYSTEM INTEGRATION OF DRAM MACROS AND LOGIC CORES IN A SINGLE CHIP ARCHITECTURE

Luk et al. - US Patent No. 5,883,814 SYSTEM-ON-CHIP LAYOUT COMPILATION

Wingard et al. - US Patent No. 5,948,089 FULLY-PIPELINED FIXED-LATENCY COMMUNICATIONS SYSTEM WITH A REAL TYPE DYNAMIC BANDWIDTH ALLOCATION.

Wingard et al. - US Patent No. 6,182,183 B1 COMMUNICATION SYSTEM AND METHOD WITH MULTILEVEL CONNECTION IDENTIFICATION

6.2 Methods Relating to SoC Testability

Edwards, et al.- US Patent No. 6,298,394 B1 (*) SYSTEM AND METHOD FOR CAPTURING INFORMATION ON AN INTERCONNECT IN AN INTEGRATED CIRCUIT

Flynn, David W. - US Patent No. 5,525,971 INTEGRATED CIRCUIT

6.3 Methods Relating to Variable Clock Frequency

Gandhi et al. - US Patent No. 6,185,691 B1 CLOCK GENERATION

Kardach et al. - US Patent No. 5,473,767 METHOD AND APPARATUS FOR ASYNCHRONOUSLY STOPPING THE CLOCK ON A PROCESSOR.

Kardach et al. - US Patent No. 5,918,043 METHOD AND APPARATUS FOR ASYNCHRONOUSLY STOPPING THE CLOCK ON A PROCESSOR.

Maitra, Amit K. - US Patent No. 5,623,647 APPLICATION SPECIFIC CLOCK THROTTLING

Orton et al. - US Patent No. 6,118,306 (*) CHANGING CLOCK FREQUENCY

Poplingher et al. - US Patent No. 6,173,379 B1 MEMORY DEVICE FOR A MICROPROCESSOR REGISTER FILE HAVING A POWER MANAGEMENT SCHEME AND METHOD FOR COPYING INFORMATION BETWEEN MEMORY SUB-CELLS IN A SINGLE CLOCK CYCLE

Stinson et al. - US Patent No. 6,127,858 METHOD AND APPARATUS FOR VARYING A CLOCK FREQUENCY ON A PHASE BY PHASE BASIS

Thomas, Thomas P. - US Patent No. 6,140,883 TUNABLE, ENERGY EFFICIENT CLOCKING SCHEME

Wong et al. - US Patent No. 5,586,307 METHOD AND APPARATUS SUPPLYING SYNCHRONOUS CLOCK SIGNALS TO CIRCUIT COMPONENTS

Young, Bruce - US Patent No. 6,079,022 METHOD AND APPARATUS FOR DYNAMICALLY ADJUSTING THE CLOCK SPEED OF A BUS DEPENDING ON BUS ACTIVITY

6.4 Methods Relating to Selection of IP Cores

Lee et al. - US Patent No. 6,102,961 METHOD AND APPARATUS FOR SELECTING IP BLOCKS

6.5 Methods Relating to Data Flow Architectures

Cismas, Sorin C. - US Patent No. 6,145,073 DATA FLOW INTEGRATED CIRCUIT ARCHITECTURE

6.6 Methods Relating to Crossbar Switch Architectures

Brewer et al. - US Patent No. 5,577,204 PARALLEL PROCESSING COMPUTER SYSTEM INTERCONNECTIONS UTILIZING UNIDIRECTIONAL COMMUNICATION LINKS WITH SEPARATE REQUEST AND RESPONSE LINES FOR DIRECT COMMUNICATION OR USING A CROSSBAR SWITCHING DEVICE

Nelson et al. - US Patent No. 6,138,185 HIGH PERFORMANCE CROSSBAR SWITCH

Van Krevelen et al. - US Patent No. 6,230,229 B1 METHOD AND SYSTEM FOR ARBITRATING PATH CONTENTION IN A CROSSBAR INTERCONNECT NETWORK