Table of Contents

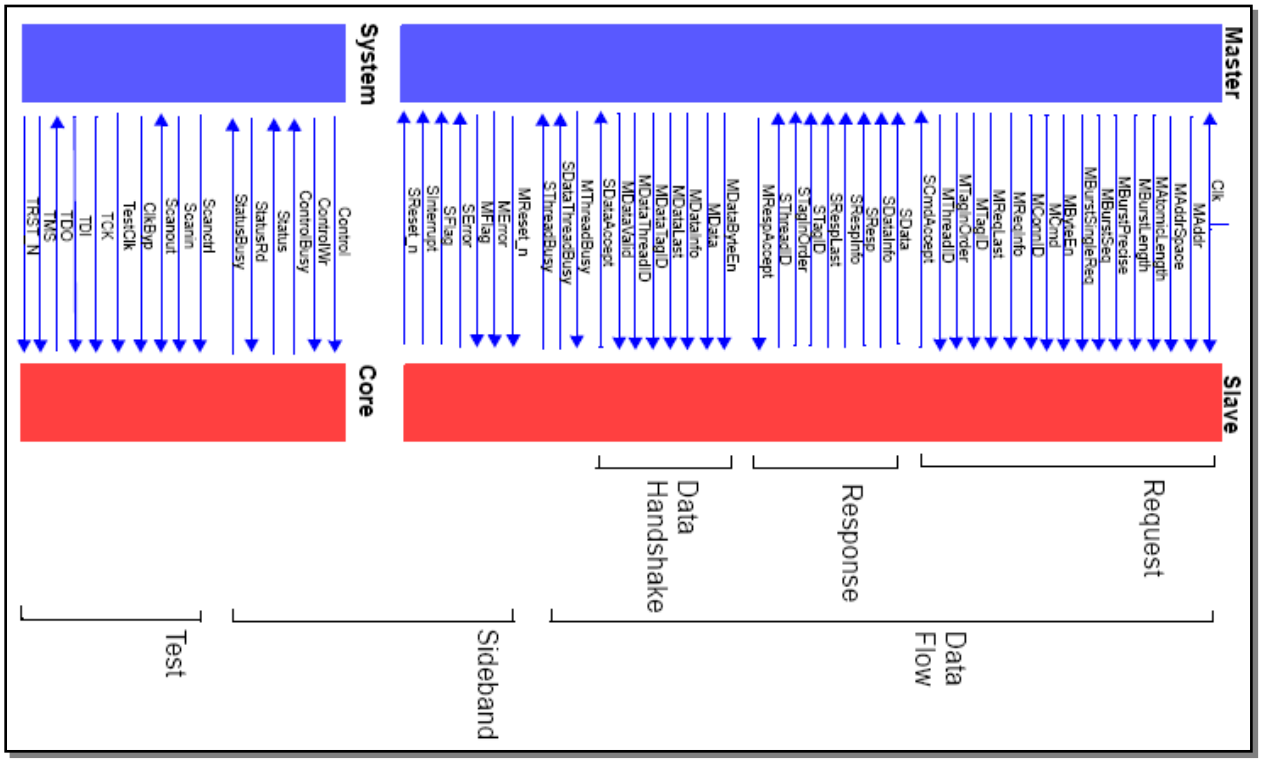
1 OPEN CORE PROTOCOL – OCP 2.1

1.1 About OCP-IP®

OCP International Partnership Association, Inc. (OCP-IP) is an independent, non-profit semiconductor industry consortium formed to administer the support, promotion and enhancement of the Open Core Protocol (OCP). OCP is the first fully supported, openly licensed, comprehensive interface socket for semiconductor intellectual property (IP) cores. The mission of OCP-IP is to address problems relating to design, verification and testing which are common to IP core reuse in System-on-Chip (SoC) products. OCP-IP is funded through the annual membership dues of its members: intellectual property, companies, integrated device manufacturers, system companies, EDA providers and design houses. VSIA endorses the OCP socket, and OCP-IP is an Adoption Group of the VSI Alliance.

The OCP captures all core characteristics without restricting system arbitration, address map, etc. The main features are following:

- Small set of mandatory signals, with a wide range of optional signals
- Synchronous, unidirectional signaling allows simplified implementation, integration and timing analysis
- Configurable address and data word width
- Structured method for inclusion of sideband signals: high-level flow control, interrupts, power control, device configuration registers, test modes, etc.
- Transfers may be *pipelined* to any depth for increased throughput
- Optional burst transfers for higher efficiency
- Multiple concurrent transfers use *thread identifiers* for out-of-order completion
- *Connection identifiers* provide end-to-end traffic identification for differential quality of service, etc.
- Synchronization primitives include atomic test-set, lazy synchronization, non-posted write commands
- OCP is a functional superset of the VSIA's Virtual Component Interface (VCI) adding protocol options that include configurable sideband signaling and test harness signals



1.1.1 Basic OCP interoperability

- Master/slave interface with unidirectional signals
- Driven and sampled by the rising edge of the OCP clock
- Fully synchronous, no multi-cycle timing paths

- All signals are strictly point-to-point (except clock and reset)
- Simple request/acknowledge protocol
- Supports data transfer on every clock cycle
- Allows master or slave to control transfer rate
- Core-specific data and address bus definition including:
- Byte and non-byte-oriented data busses
- Read-only and write-only interfaces
- In-band data tagging (parity, EDC, etc.)
- In-band command tagging (protocol extensions, etc.)
- Pipelined or blocking commands including non-posted write
- Security access permissions can be part of any request
- Well-defined linguistic format for core characteristics, interfaces (signals, timing and configuration) and performance
- Timing category specifications
- Level 2 - tightest, highest performance interface timing
- Level 1 - conservative timing for effortless integration
- Level 0 - protocol without timing specified (especially useful for simulation/verification tools)

1.1.2 Simple Extensions performance

- Bursts group related transfers into a complete transaction
- Burst transactions supported:
  - Sequential (precise or indefinite length)
  - Streaming (e.g. FIFO)
  - Core-specific (e.g. cache lines)
  - Controlled atomicity in breaking up long bursts
  - Two-dimensional block sequences
- Pipelined (cmd/add ahead of data) writes
- Aligned or random byte enable orders
- Read and write data flow control
- Address space definition for multi-address-segment targets
- Single-request/multiple-data or one command per data phase

1.1.3 Sideband Extensions custom signaling

- Core-specific, user defined signals:
  - System event signals (e.g. interrupts, error notification)
  - Two synchronous resets defined: master-to-slave and/or slave-to-master
  - Data transfer coordination (e.g. high-level flow control)
- Debug and Test Interface Extensions
  - Support structured full or partial scan test environments
  - Scan pertains to internal scan techniques for a predesigned hard-core or end user-inserted into a soft-core
  - Clock Controls are used for scan testing and debug, including multiple clock domains
  - IEEE 1149 supports cores with a JTAG Test Access Port
  - JTAG- and Enhanced-JTAG-based debug for MIPS®, ARM®, TI® DSP, SPARC™ and others

1.1.4 Complex Extensions concurrency support

- Thread identifiers enable:
  - Interleaved burst transactions
  - Out-of-order transaction completion
  - Differential quality of service
- Rigorous thread flow control definition guarantees non-blocking
- Connection identifiers enable:
  - End-to-end system initiator identification
  - Service priority management by system targets
- Tagging provides shared flow control for out-of-order transactions

1.2 Theory of Operation

This chapter provides an overview of operations describing the OCP Protocol release 2.1.

1.2.1 Point-to-Point Synchronous Interface

To simplify timing analysis, physical design, and general comprehension, the OCP is composed of uni-directional signals driven with respect to, and sampled by the rising edge of the OCP clock. The OCP is fully synchronous and contains no multi-cycle timing paths. All signals other than the clock are strictly point-to-point.

1.2.2 Bus Independence

A core utilizing the OCP can be interfaced to any bus. A test of any bus-independent interface is to connect a master to a slave without an intervening onchip bus. This test not only drives the specification towards a fully symmetric interface but helps to clarify other issues. For instance, device selection techniques vary greatly among on-chip buses. Some use address decoders. Others generate independent device select signals (analogous to a board level chip select). This complexity should be hidden from IP cores, especially since in the directly-connected case there is no decode/selection logic. OCP-compliant slaves receive device selection information integrated into the basic command field.

Arbitration schemes vary widely. Since there is virtually no arbitration in the directly-connected case, arbitration for any shared resource is the sole responsibility of the logic on the bus side of the OCP. This permits OCPcompliant masters to pass a command field across the OCP that the bus interface logic converts into an arbitration request sequence.

1.2.3 Commands

There are two basic commands, Read and Write and five command extensions. The WriteNonPost and Broadcast commands have semantics that are similar to the Write command. A WriteNonPost explicitly instructs the slave not to post a write. For the Broadcast command, the master indicates that it is attempting to write to several or all remote target devices that are connected on the other side of the slave. As such, Broadcast is typically useful only for slaves that are in turn a master on another communication medium (such as an attached bus).

The other command extensions, ReadExclusive, ReadLinked and WriteConditional, are used for synchronization between system initiators. ReadExclusive is paired with Write or WriteNonPost, and has blocking semantics. ReadLinked, used in conjunction with WriteConditional has non-blocking (lazy) semantics. These synchronization primitives correspond to those available natively in the instruction sets of different processors.

1.2.4 Address/Data

Wide widths, characteristic of shared on-chip address and data buses, make tuning the OCP address and data widths essential for area-efficient implementation. Only those address bits that are significant to the IP core should cross the OCP to the slave. The OCP address space is flat and composed of 8-bit bytes (octets).

To increase transfer efficiencies, many IP cores have data field widths significantly greater than an octet. The OCP supports a configurable data width to allow multiple bytes to be transferred simultaneously. The OCP refers to the chosen data field width as the word size of the OCP. The term word is used in the traditional computer system context; that is, a word is the natural transfer unit of the block. OCP supports word sizes of power-of-two and nonpower-of-two as would be needed for a 12-bit DSP core. The OCP address is a byte address that is word aligned.

Transfers of less than a full word of data are supported by providing byte enable information that specifies which octets are to be transferred. Byte enables are linked to specific data bits (byte lanes). Byte lanes are not associated with particular byte addresses. This makes the OCP endian-neutral, able to support both big and little-endian cores.

1.2.5 Pipelining

The OCP allows pipelining of transfers. To support this feature, the return of read data and the provision of write data may be delayed after the presentation of the associated request.

1.2.6 Response

The OCP separates requests from responses. A slave can accept a command request from a master on one cycle and respond in a later cycle. The division of request from response permits pipelining. The OCP provides the option of having responses for Write commands, or completing them immediately without an explicit response.

1.2.7 Burst

To provide high transfer efficiency, burst support is essential for many IP cores. The extended OCP supports annotation of transfers with burst information. Bursts can either include addressing information for each successive command (which simplifies the requirements for address sequencing/burst count processing in the slave), or include addressing information only once for the entire burst.

1.2.8 In-band Information

Cores can pass core-specific information in-band in company with the other information being exchanged. In-band extensions exist for requests and responses, as well as read and write data. A typical use of in-band extensions is to pass cacheable information or data parity.

1.2.9 Tags

Tags are available in the OCP interface to control the ordering of responses. Without tags, a slave must return responses in the order that the requests were issued by the master. Similarly, writes must be committed in order. With the addition of tags, responses can be returned out-of-order, and write data can be committed out-of-order with respect to requests, as long as the transactions target different addresses. The tag links the response back to the original request.

Tagging is useful when a master core such as a processor can handle out-of-order return, because it allows a slave core such as a DRAM controller to service requests in the order that is most convenient, rather than the order in which requests were sent by the master.

Out-of-order request and response delivery can also be enabled using multiple threads. The major differences between threads and tags are that threads can have independent flow control for each thread and have no ordering rules for transactions on different threads. Tags, on the other hand, exist within a single thread and are restricted to shared flow control. Tagged transactions cannot be re-ordered with respect to overlapping addresses. Implementing independent flow control requires independent buffering for each thread, leading to more complex implementations. Tags enable lower overhead implementations for out-of-order return of responses at the expense of some concurrency.

1.2.10 Threads and Connections

To support concurrency and out-of-order processing of transfers, the extended OCP supports the notion of multiple threads. Transactions within different threads have no ordering requirements, and independent flow control from one another. Within a single thread of data flow, all OCP transfers must remain ordered unless tags are in use. Transfers within a single thread must remain ordered unless tags are in use. The concepts of threads and tags are hierarchical: each thread has its own flow control, and ordering within a thread either follows the request order strictly, or is governed by tags.

While the notion of a thread is a local concept between a master and a slave communicating over an OCP, it is possible to globally pass thread information from initiator to target using connection identifiers. Connection information helps to identify the initiator and determine priorities or access permissions at the target.

1.2.11 Interrupts, Errors, and other Sideband Signaling

While moving data between devices is a central requirement of on-chip communication systems, other types of communications are also important. Different types of control signaling are required to coordinate data transfers (for instance, high-level flow control) or signal system events (such as interrupts). Dedicated point-to-point data communication is sometimes required. Many devices also require the ability to notify the system of errors that may be unrelated to address/data transfers.

The OCP refers to all such communication as sideband (or out-of-band) signaling, since it is not directly related to the protocol state machines of the dataflow portion of the OCP. The OCP provides support for such signals through sideband signaling extensions.

Errors are reported across the OCP using two mechanisms. The error response code in the response field describes errors resulting from OCP transfers that provide responses. Write-type commands without responses cannot use the in-band reporting mechanism. The second method for reporting errors across the OCP uses out-of band error fields. These signals report more generic sideband errors, including those associated with posted write commands.

1.3 OCP Profiles

This chapter provides profiles that capture the OCP features associated with standard communication functions. The pre-defined profiles help map the requirements of a new core to OCP configuration guidelines.

The expected benefits include:

- Reduced risk of incompatibility when integrating OCP based cores originating from different providers
- Reduced learning curve in applying OCP for standard purposes
- Simplified circuitry needed to bridge an OCP based core to another communication interface standard
- Improved core maintenance
- Simplified creation of reusable core test benches

Profiles address only the OCP interface, with each profile consisting of OCP interface signals, specific protocol features, and application guidelines. For cores natively equipped with OCP interfaces, profiles minimize the number of interface and protocol options that need to be considered.

Two sets of OCP profiles are provided: profiles for new IP cores implementing native OCP interfaces and profiles that are targeted at designers of bridges between OCP and other bus protocols.

Since the other bus protocols may have several implementation flavors that require custom OCP parameter sets, the bridging profiles are incomplete. The bridging profiles can be used with OCP serving as either a master or a slave.

The initial set of profiles cover the most common core communication requirements. Each OCP profile defines one or more applications. The profiles available for release 2.1 are:

- Native OCP Profiles – The native OCP profiles are designed for new IP cores implementing native OCP interfaces.
  - Block data flow
  - Sequential undefined length data flow
  - Register access
- Bridging Profiles – These profiles are designed to simplify or automate the creation of bridges to other interface protocols. The bridge can have an OCP master or slave port. There are two types:
  - The simple H-bus profile is intended to provide a connection through an external bridge, for example to a CPU with an AMBA AHB protocol.
  - The X-bus interfaces support cacheable and non-cacheable instruction and data traffic between a CPU and the memories and register interfaces of other targets. The X-bus profiles might be used with a CPU core that internally used the AMBA AXI protocols, and were externally bridged to OCP.
    - X-bus packet write
    - X-bus packet read

Each profile addresses distinct OCP interfaces, though most systems constructed using OCP will have a mix of functional elements. As different cores and subsystems have diverse communication characteristics and constraints, various profiles will prove useful at different interfaces within the system.

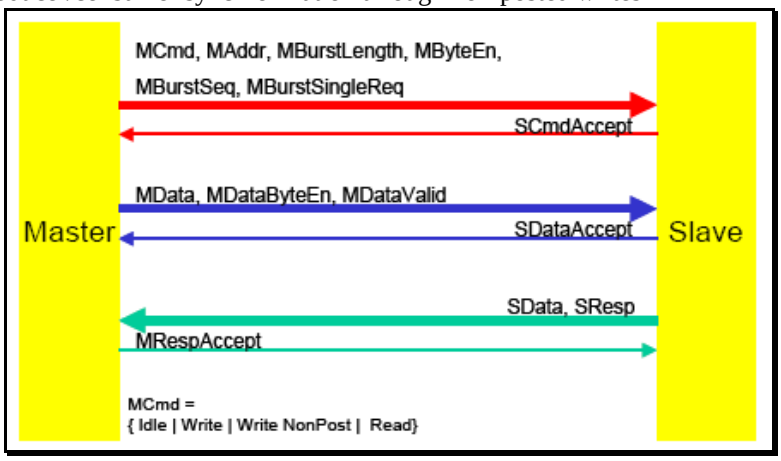
1.3.1 Block Data Flow Profile

The -block data flow profile is designed for master type (read/write, read-only, or write-only) interfaces of cores exchanging data blocks with memory. This profile is particularly effective for managing pipelined access of defined-length traffic (for example, MPEG macroblocks) to and from memory.

Cores with the following characteristics would benefit from using this profile:

- Block-based communication (includes a single data element block)

- A single request/multiple data burst model using incremental or a stream burst sequence
- De-coupled, pipelined command and data flows
- 32 bit address
- Natural data width and block size
- Use of byte enables
- Single threaded
- Support for producer/consumer synchronization through non-posted writes

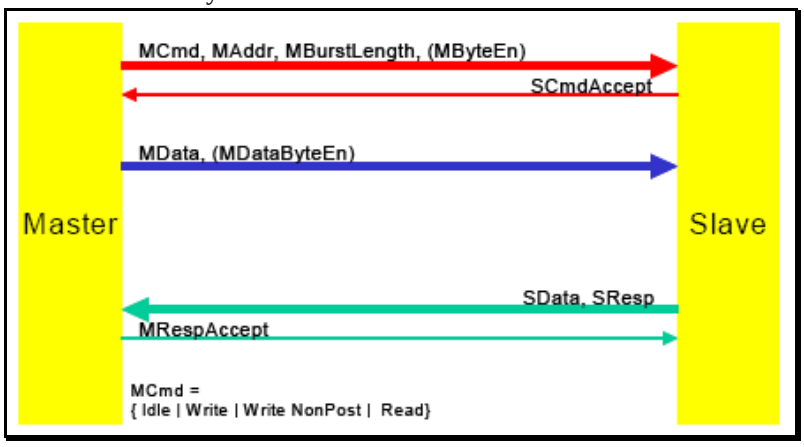


1.3.2 Sequential Undefined Length Data Flow Profile

This profile is a master type (read/write or read-only, or write-only) interface for cores that communicate data streams with memory.

Cores with the following characteristics would benefit from using this profile:

- Communication of an undefined amount of data elements to consecutive addresses
- Imprecise burst model
- Aligned command/write data flows, decoupled read data flow
- 32 bit address
- Natural data width
- Optional use of byte enables
- Single thread
- Support for producer/consumer synchronization

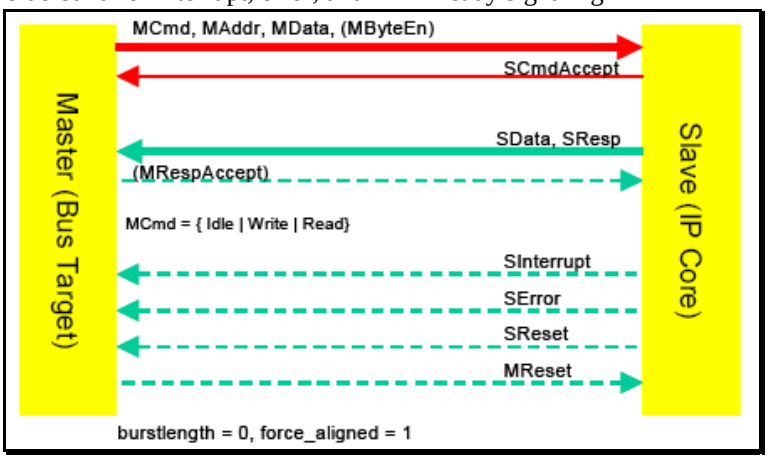


1.3.3 Register Access Profile

The register access profile offers a control processor the ability to program the operation of an attached core. This profile supports programmable register interfaces across a wide range of IP cores, such as simple peripherals, DMA engines, or register-controlled processing engines. The IP core would be an OCP slave on this interface, connected to a master that is a target addressed by a control processor.

Cores with the following characteristics would benefit from using this profile:

- Address mapped communication, but target decoding is handled upstream
- Natural data width (unconstrained, but 32b is most common)
- Natural address width (based on the number of internal registers X data width)
- No bursting
- Precise write responses indicate completion of write side-effects
- Single threaded
- Optional aligned byte enables for sub-word CPU access
- Optional response flow control
- Optional use of side-band for interrupt, error, and DMA ready signaling

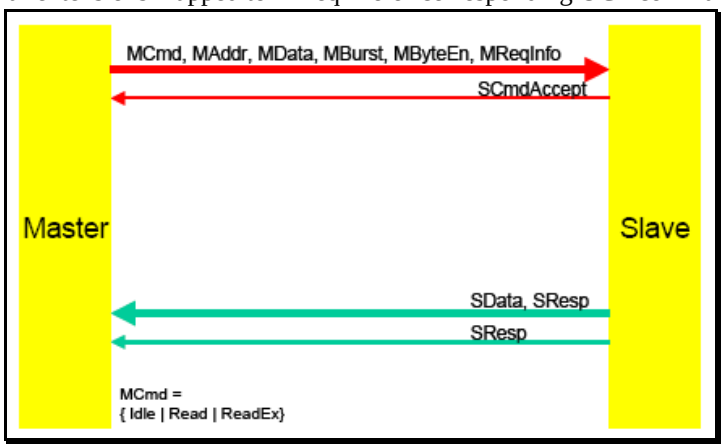


1.3.4 Simple H-bus Profile

This profile allows you to create OCP master wrappers to native interfaces of simple CPU type initiators with multiple-request/multiple-data, read and write transactions.

Cores with the following characteristics would benefit from using this profile:

- Address mapped communication
- Natural address width
- Byte enable
- Natural data width
- Constrained burst size
- Single thread
- Caching and similar extensions mapped to MReqInfo or corresponding OCP commands



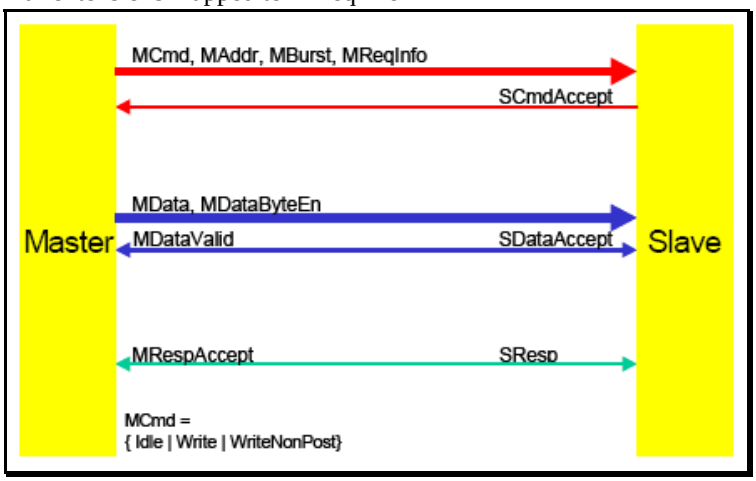
1.3.5 X-Bus Packet Write Profile

This profile is designed to create OCP master wrappers to native interfaces of CPU type initiators with single-request/multiple-data, write-only transactions.

Cores with the following characteristics would benefit from using this profile:

- Packet type communication

- Natural address width
- Separate command/write data handshake
- Byte enable
- Natural data width
- Multi-thread (blocking)
- Caching and similar extensions mapped to MReqInfo

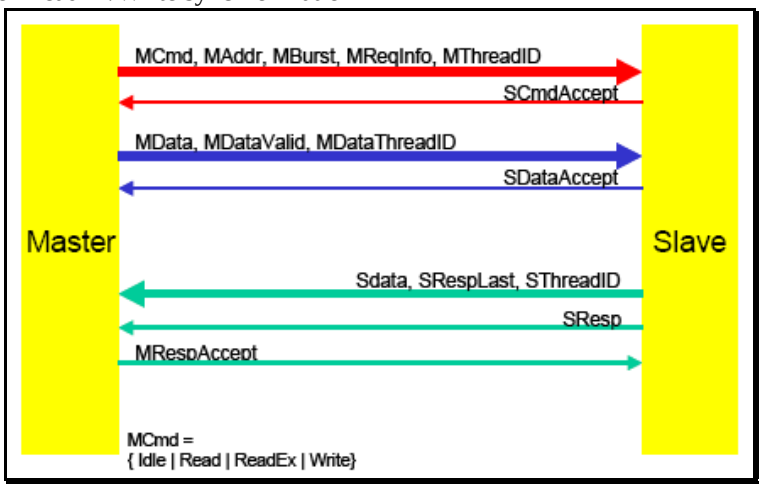


1.3.6 X-Bus Packet Read Profile

This profile helps you create OCP master wrappers for native interfaces of CPU type initiators with single-request multiple-data read-only transactions.

Cores with the following characteristics would benefit from using this profile:

- Packet type communication
- Natural address width
- Single-request multiple-data read
- Byte enable
- Natural data width
- Multi-thread (blocking)
- Caching and similar extensions mapped to MReqInfo
- Write support for ReadEx/Write synchronization



1.4 Tools

OCP-IP offers its members the use of an EDA tool, CoreCreator, to automate the tasks of building, simulating, verifying and packaging OCP-compatible cores. CoreCreator also supplies a protocol checker to ensure compliance with the OCP specification. IP core products can be fully componentized by consolidating core models, timing parameters, synthesis scripts, verification suites and test vectors.

2 AMBA BUS

The AMBA family introduces two bus protocols – AMBA 2 and AMBA 3 frequently named AHB Bus and AXI Bus respectively.

2.1 About The AMBA 2 AHB Protocol

AHB is intended to address the requirements of high-performance synthesizable designs. AMBA AHB implements the features required for high-performance, high clock frequency systems including:

- burst transfers
- split transactions
- single cycle bus master handover
- single clock edge operation
- non-tristate implementation
- wider data bus configurations (64/128 bits)

An AMBA AHB design may contain one or more bus masters, typically a system would contain at least the processor and test interface. However, it would also be common for a Direct Memory Access (DMA) or Digital Signal Processor (DSP) to be included as bus masters.
The external memory interface, APB bridge and any internal memory are the most common AHB slaves. Any other peripheral in the system could also be included as an AHB slave. However, low-bandwidth peripherals typically reside on the APB.
A typical AMBA AHB system design contains the following components:

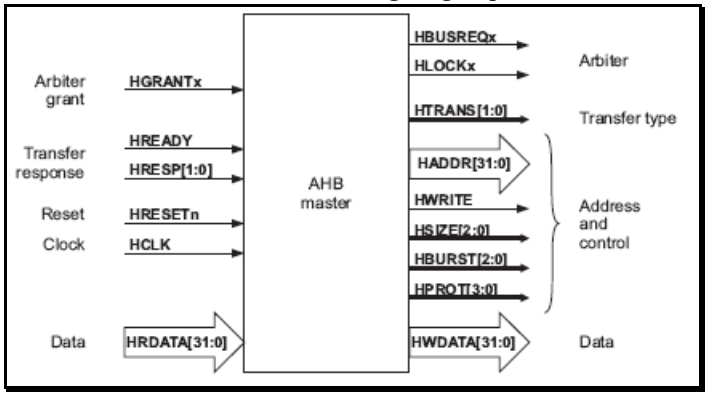
AHB master A bus master is able to initiate read and write operations by providing an address and control information. Only one bus master is allowed to actively use the bus at any one time.

AHB slave A bus slave responds to a read or write operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.

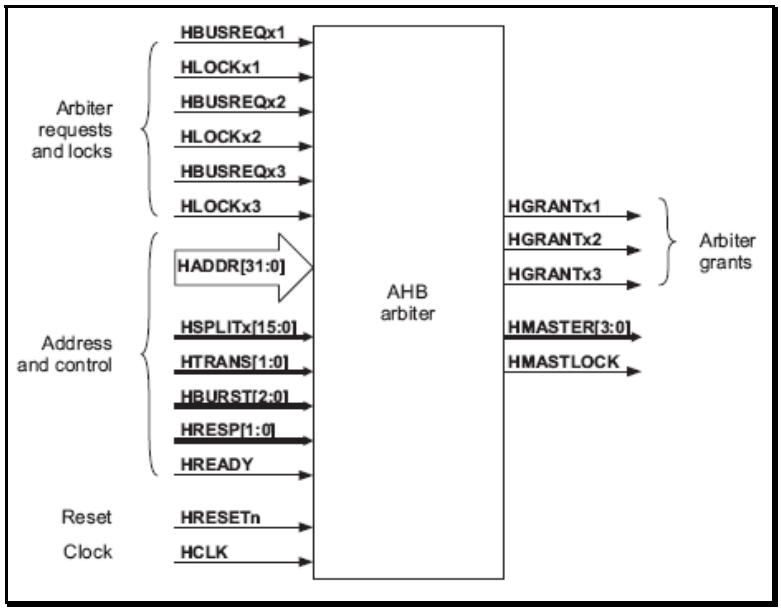
AHB arbiter The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as highest priority or fair access can be implemented depending on the application requirements. An AHB would include only one arbiter, although this would be trivial in single bus master systems.

AHB decoder The AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer. A single centralized decoder is required in all AHB implementations.

The interface diagram of an AHB bus master shows the main signal groups.

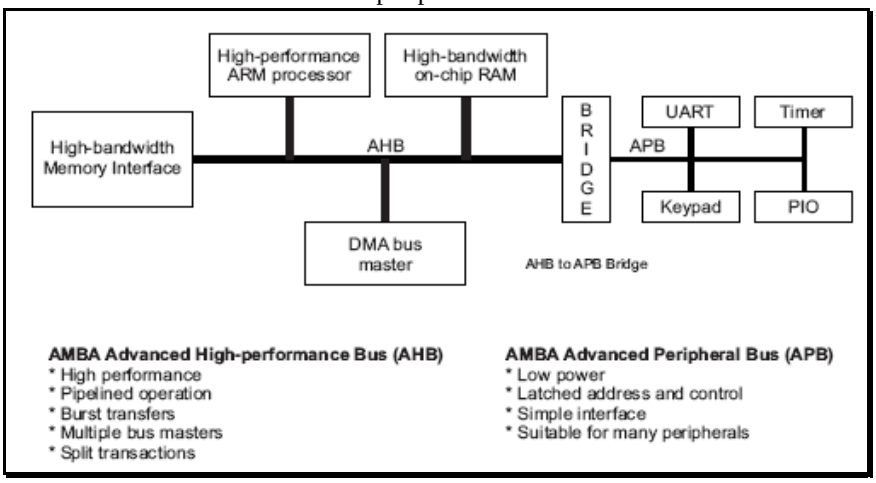


Next figure shows the signal interface of an AHB arbiter.



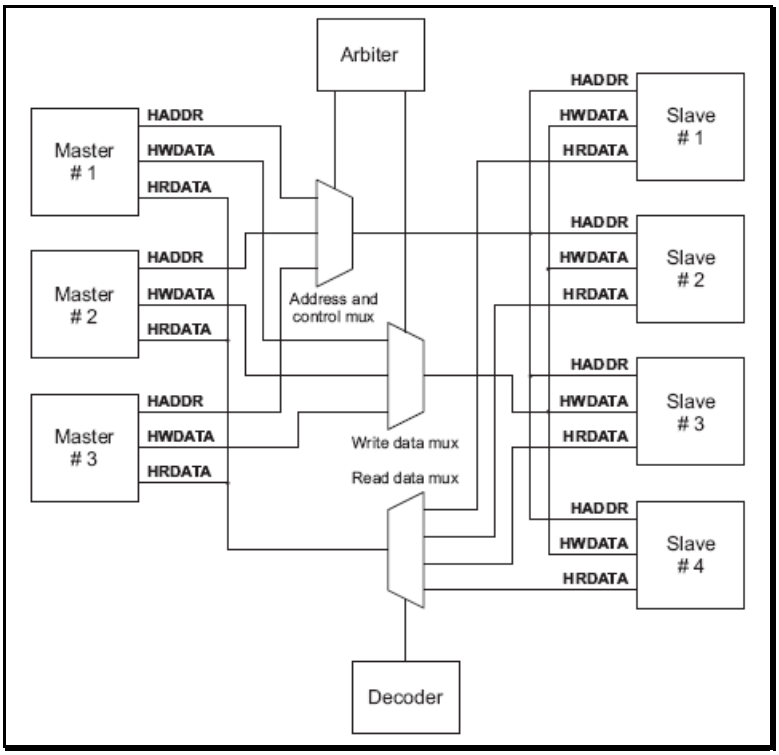
2.1.1 A Typical AMBA AHB-based Microcontroller

An AMBA-based microcontroller typically consists of a high-performance system backbone bus, able to sustain the external memory bandwidth, on which the CPU and other Direct Memory Access (DMA) devices reside, plus a bridge to a narrower APB bus on which the lower bandwidth peripheral devices are located.



2.1.2 Bus Interconnection

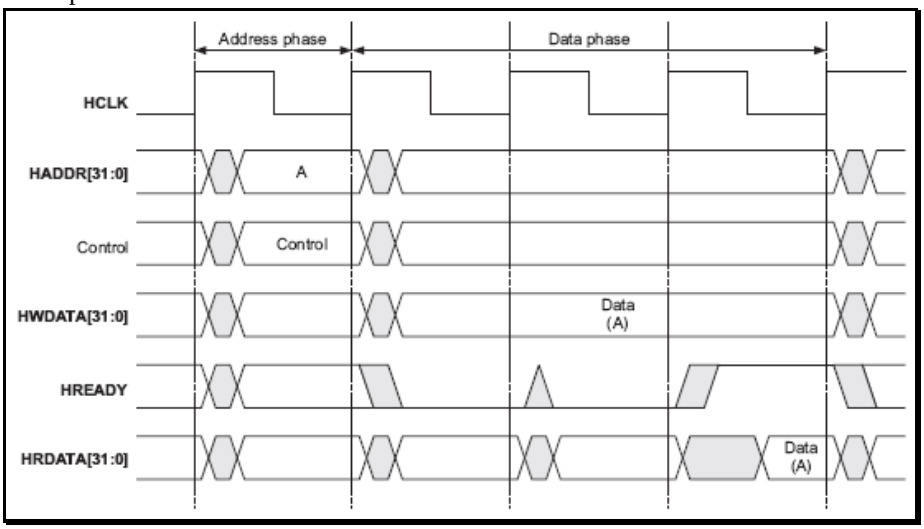
The AMBA AHB bus protocol is designed to be used with a central multiplexor interconnection scheme. Using this scheme all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and response signal multiplexor, which selects the appropriate signals from the slave that is involved in the transfer.



2.1.3 AHB Bus Transfer Examples

First simple example demonstrates how the address and data phases of the transfer occur during different clock periods. In fact, the address phase of any transfer occurs during the data phase of the previous transfer. This overlapping of address and data is fundamental to the pipelined nature of the bus and allows for high performance operation, while still providing adequate time for a slave to provide the response to a transfer.

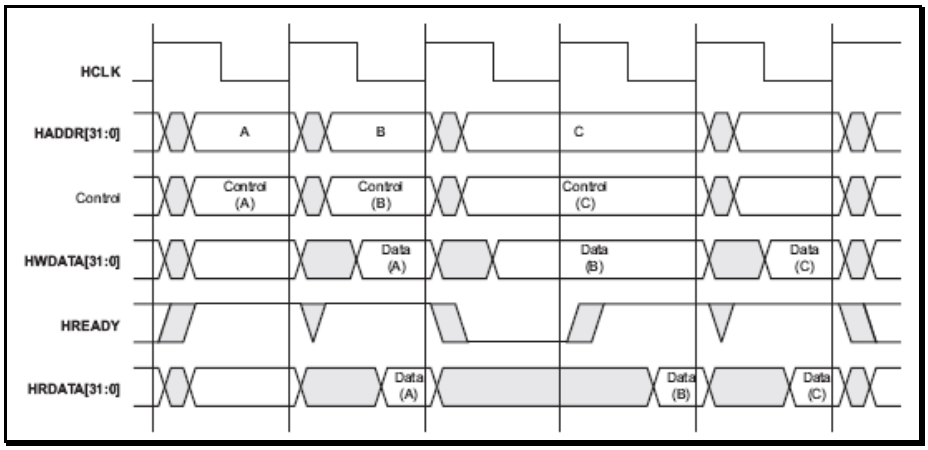
A slave may insert wait states into any transfer, as shown in figure below, which extends the transfer allowing additional time for completion.



NOTES:

1. For write operations the bus master will hold the data stable throughout the extended cycles.
2. For read transfers the slave does not have to provide valid data until the transfer is about to complete.

When a transfer is extended in this way it will have the side-effect of extending the address phase of the following transfer. This is illustrated in figure below, which shows three transfers to unrelated addresses, A, B & C.



NOTES:

1. The transfers to addresses A and C are both zero wait state
2. The transfer to address B is one wait state
3. Extending the data phase of the transfer to address B has the effect of extending the address phase of the transfer to address C.

2.2 About The AMBA 2 APB Protocol

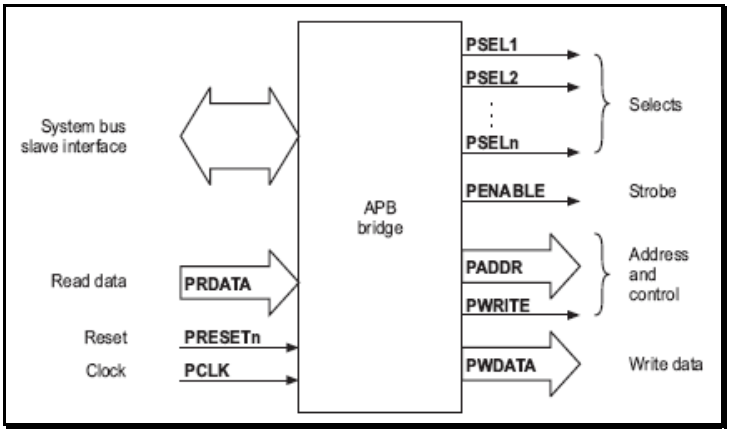
The Advanced Peripheral Bus (APB) is part of the Advanced Microcontroller Bus Architecture (AMBA) hierarchy of buses and is optimized for minimal power consumption and reduced interface complexity.

The AMBA APB should be used to interface to any peripherals which are low bandwidth and do not require the high performance of a pipelined bus interface.

The latest revision of the APB ensures that all signal transitions are only related to the rising edge of the clock. This improvement means the APB peripherals can be integrated easily into any design flow, with the following advantages:

- performance is improved at high-frequency operation
- performance is independent of the mark-space ratio of the clock
- static timing analysis is simplified by the use of a single clock edge
- no special considerations are required for automatic test insertion
- many Application-Specific Integrated Circuit (ASIC) libraries have a better selection of rising edge registers
- easy integration with cycle based simulators

The APB bridge is the only bus master on the AMBA APB. In addition, the APB bridge is also a slave on the higher-level system bus.

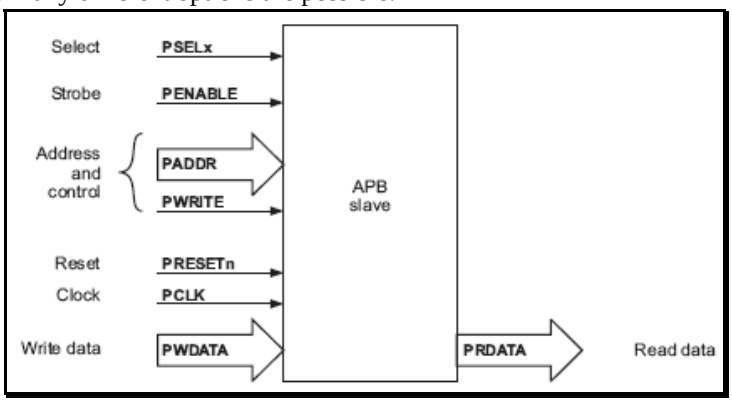


The bridge unit converts system bus transfers into APB transfers and performs the following functions:

- Latches the address and holds it valid throughout the transfer.
- Decodes the address and generates a peripheral select, **PSELx**. Only one select signal can be active during a transfer.

- Drives the data onto the APB for a write transfer.
- Drives the APB data onto the system bus for a read transfer.
- Generates a timing strobe, **PENABLE**, for the transfer.

APB slaves have a simple, yet flexible, interface. The exact implementation of the interface will be dependent on the design style employed and many different options are possible.



The APB slave interface is very flexible.

For a write transfer the data can be latched at the following points:

- on either rising edge of **PCLK**, when **PSEL** is HIGH
- on the rising edge of **PENABLE**, when **PSEL** is HIGH.

The select signal **PSELx**, the address **PADDR** and the write signal **PWRITE** can be combined to determine which register should be updated by the write operation.

For read transfers the data can be driven on to the data bus when **PWRITE** is LOW and both **PSELx** and **PENABLE** are HIGH. While **PADDR** is used to determine which register should be read.

2.3 About The AMBA 3 AXI Protocol

The AMBA AXI protocol is targeted at high-performance, high-frequency system designs and includes a number of features that make it suitable for a high-speed submicron interconnect. The objectives of the latest generation AMBA interface are to:

- be suitable for high-bandwidth and low-latency designs
- enable high-frequency operation without using complex bridges
- meet the interface requirements of a wide range of components
- be suitable for memory controllers with high initial access latency
- provide flexibility in the implementation of interconnect architectures
- be backward-compatible with existing AHB and APB interfaces

The key features of the AXI protocol are:

- separate address/control and data phases
- support for unaligned data transfers using byte strobes
- burst-based transactions with only start address issued
- separate read and write data channels to enable low-cost Direct Memory Access (DMA)
- ability to issue multiple outstanding addresses
- out-of-order transaction completion
- easy addition of register stages to provide timing closure

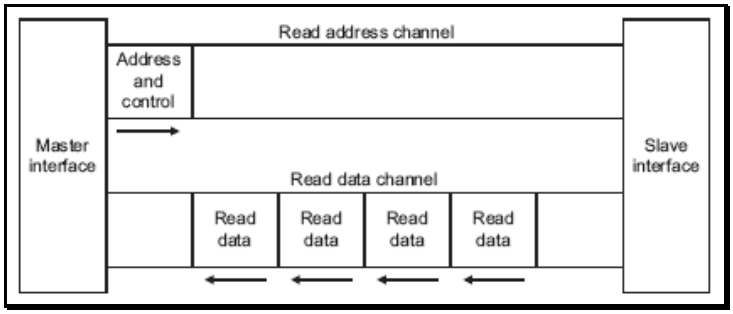
As well as the data transfer protocol, the AXI protocol includes optional extensions that cover signaling for low-power operation.

2.3.1 Architecture

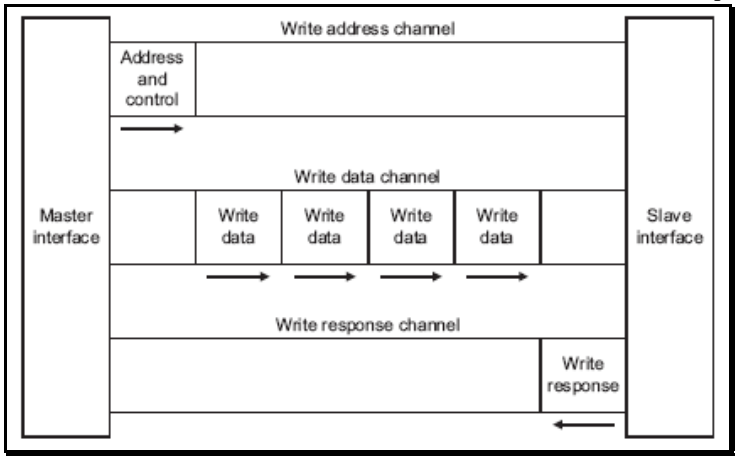
The AXI protocol is burst-based. Every transaction has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. In write transactions, in which all the data flows from the master to the slave, the AXI protocol has an additional write response channel to allow the slave to signal to the master the completion of the write transaction.
The AXI protocol enables:

- address information to be issued ahead of the actual data transfer
- support for multiple outstanding transactions
- support for out-of-order completion of transactions

The 1st figure shows how a read transaction uses the read address and read data channels.



The 2nd figure shows how a write transaction uses the write address, write data, and write response channels.



2.3.2 Channel definition

Each of the five independent channels consists of a set of information signals and uses a two-way VALID and READY handshake mechanism.
The information source uses the VALID signal to show when valid data or control information is available on the channel. The destination uses the READY signal to show when it can accept the data. Both the read data channel and the write data channel also include a LAST signal to indicate when the transfer of the final data item within a transaction takes place.

2.3.2.1 Read and write address channels

Read and write transactions each have their own address channel. The appropriate address channel carries all of the required address and control information for a transaction. The AXI protocol supports the following mechanisms:

- variable-length bursts, from 1 to 16 data transfers per burst
- bursts with a transfer size of 8-1024 bits
- wrapping, incrementing, and non-incrementing bursts
- atomic operations, using exclusive or locked accesses
- system-level caching and buffering control

- secure and privileged access.

2.3.2.2 Read data channel

The read data channel conveys both the read data and any read response information from the slave back to the master. The read data channel includes:

- the data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide
- a read response indicating the completion status of the read transaction.

2.3.2.3 Write data channel

The write data channel conveys the write data from the master to the slave and includes:

- the data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide
- one byte lane strobe for every eight data bits, indicating which bytes of the data bus are valid.

Write data channel information is always treated as buffered, so that the master can perform write transactions without slave acknowledgement of previous write transactions.

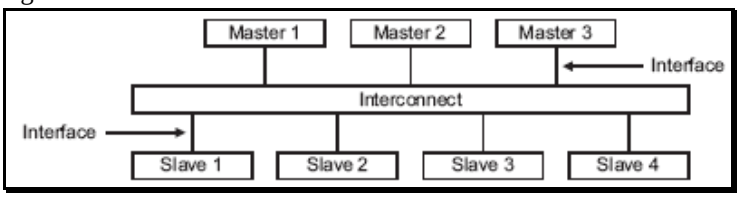
2.3.2.4 Write response channel

The write response channel provides a way for the slave to respond to write transactions. All write transactions use completion signaling.

The completion signal occurs once for each burst, not for each individual data transfer within the burst.

2.3.3 Interface and interconnect

A typical system consists of a number of master and slave devices connected together through some form of interconnect, as shown in figure below.



The AXI protocol provides a single interface definition for describing interfaces:

- between a master and the interconnect
- between a slave and the interconnect
- between a master and a slave

The interface definition enables a variety of different interconnect implementations. The interconnect between devices is equivalent to another device with symmetrical master and slave ports to which real master and slave devices can be connected.

Most systems use one of three interconnect approaches:

- shared address and data buses
- shared address buses and multiple data buses
- multilayer, with multiple address and data buses

In most systems, the address channel bandwidth requirement is significantly less than the data channel bandwidth requirement. Such systems can achieve a good balance between system performance and interconnect complexity by using a shared address bus with multiple data buses to enable parallel data transfers.

2.3.4 Register slices

Each AXI channel transfers information in only one direction, and there is no requirement for a fixed relationship between the various channels. This is important because it enables the insertion of a register slice in any channel, at the cost of an additional cycle of latency. This makes possible a trade-off between cycles of latency and maximum frequency of operation.

It is also possible to use register slices at almost any point within a given interconnect. It can be advantageous to use a direct, fast connection between a processor and high-performance memory, but to use simple register slices to isolate a longer path to less performance-critical peripherals.

2.3.5 Transaction ordering

The AXI protocol enables out-of-order transaction completion. It gives an ID tag to every transaction across the interface. The protocol requires that transactions with the same ID tag are completed in order, but transactions with different ID tags can be completed out of order.

Out-of-order transactions can improve system performance in two ways:

- The interconnect can enable transactions with fast-responding slaves to complete in advance of earlier transactions with slower slaves.
- Complex slaves can return read data out of order. For example, a data item for a later access might be available from an internal buffer before the data for an earlier access is available.

If a master requires that transactions are completed in the same order that they are issued, then they must all have the same ID tag. If, however, a master does not require in-order transaction completion, it can supply the transactions with different ID tags, enabling them to be completed in any order.

In a multimaster system, the interconnect is responsible for appending additional information to the ID tag to ensure that ID tags from all masters are unique. The ID tag is similar to a master number, but with the extension that each master can implement multiple virtual masters within the same port by supplying an ID tag to indicate the virtual master number.

Although complex devices can make use of the out-of-order facility, simple devices are not required to use it. Simple masters can issue every transaction with the same ID tag, and simple slaves can respond to every transaction in order, irrespective of the ID tag.

2.3.6 Additional features

The AXI protocol also supports the following additional features:
Burst types – The AXI protocol supports three different burst types that are suitable for:

- normal memory accesses
- wrapping cache line bursts
- streaming data to peripheral FIFO locations

System cache support:

- The cache-support signal of the AXI protocol enables a master to provide to a system-level cache the bufferable, cacheable, and allocate attributes of a transaction.

Protection unit support:

- To enable both privileged and secure accesses, the AXI protocol provides three levels of protection unit support

Atomic operations:

- The AXI protocol defines mechanisms for both exclusive and locked accesses

Error support:

- The AXI protocol provides error support for both address decode errors and slave-generated errors

Unaligned address:

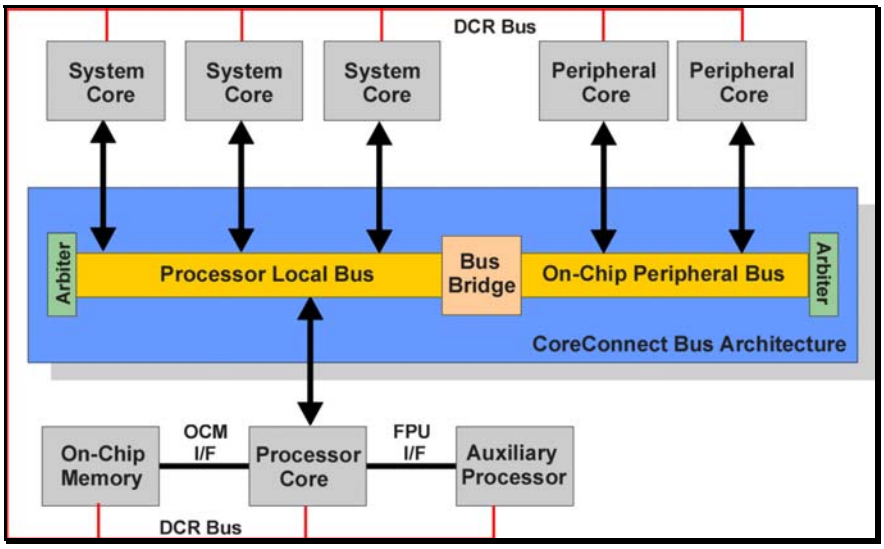
- To enhance the performance of the initial accesses within a burst, the AXI protocol supports unaligned burst start addresses

3 CORECONNECT BUS ARCHITECTURE

The IBM CoreConnect bus architecture eases the integration and reuse of processor, system, and peripheral cores within standard product and custom system-on-a-chip (SOC) designs. The versatility of the CoreConnect bus architecture allows engineers to assemble custom SOC designs using cores designed to CoreConnect specifications. With time-consuming performance, functional, and timing pattern issues resolved, designers can focus on product differentiation – dramatically reducing costs and time-to-market for simple and complex SOC designs.

The CoreConnect bus architecture is a standard SOC design point, and serves as the foundation of IBM Blue LogicTM

Core Library or other non-IBM devices. Elements of this architecture include the processor local bus (PLB), the on-chip peripheral bus (OPB), a bus bridge, and a device control register (DCR) bus. High-performance peripherals connect to the high-bandwidth, low-latency PLB. Slower peripheral cores connect to the OPB, which reduces traffic on the PLB, resulting in greater overall system performance. Bus model toolkits are available to assist in designing custom logic to the PLB/OPB/DCR bus standards.



CoreConnect Advanced Features:

- Open architecture bus standard with published specifications
- 32-, 64-, 128 and 256-bit versions to support a wide variety of applications
- Enables IP reuse in multiple designs
- No-fee, no-royalty licensing

3.1 PLB Overview

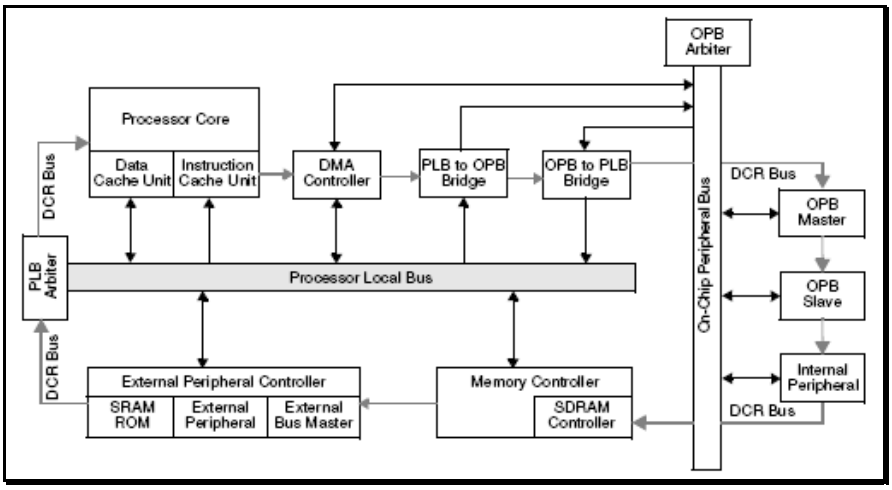
The processor local bus (PLB) is a high performance 64-bit address bus and up to 256-bit data bus which provides a standard interface between the processor cores and integrated bus controllers so that a library of processor cores and bus controllers can be developed for use in Core+ASIC and systemon-a-chip (SOC) designs.

The processor local bus is a high performance on-chip bus used in highly integrated Core+ASIC systems. The PLB supports read and write data transfers between master and slave devices equipped with a PLB bus interface and connected through PLB signals.

Each PLB master is attached to the PLB through separate address, read data, and write data buses and a plurality of transfer qualifier signals. PLB slaves are attached to the PLB through shared, but decoupled, address, read data, and write data buses and a plurality of transfer control and status signals for each data bus.

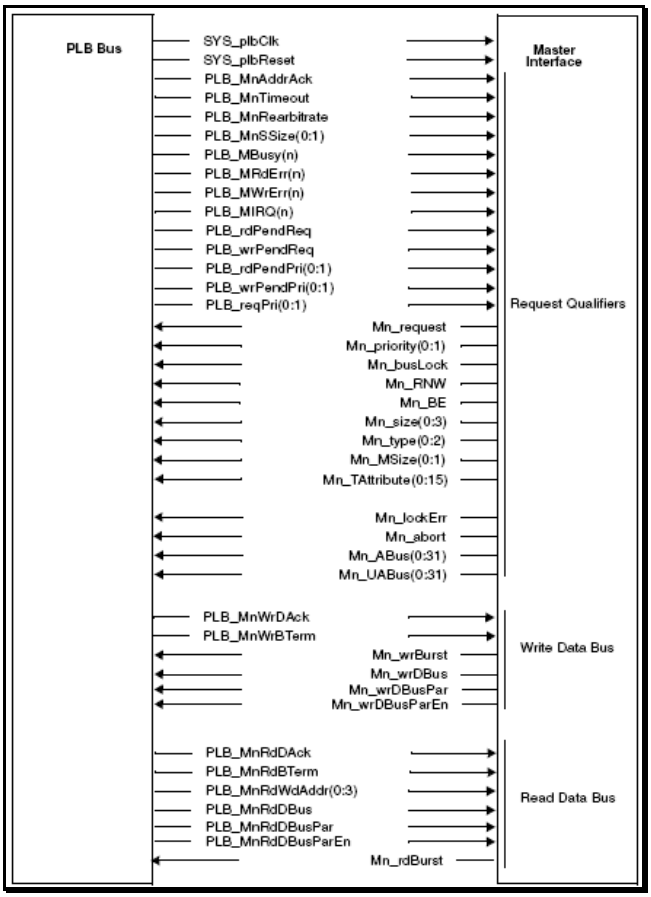
Access to the PLB is granted through a central arbitration mechanism that allows masters to compete for bus ownership. This arbitration mechanism is flexible enough to provide for the implementation of various priority schemes. Additionally, an arbitration locking mechanism is provided to support master-driven atomic operations.

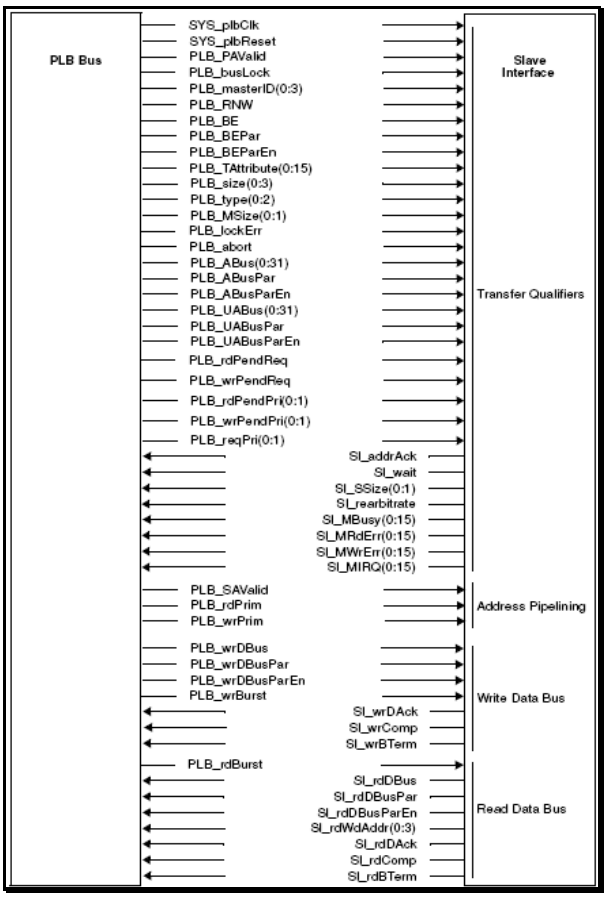
The PLB is a fully-synchronous bus. Timing for all PLB signals is provided by a single clock source which is shared by all masters and slaves attached to the PLB.
The figure below demonstrates how the processor local bus is interconnected for the purpose of Core+ASIC development or system-on-a-chip design.

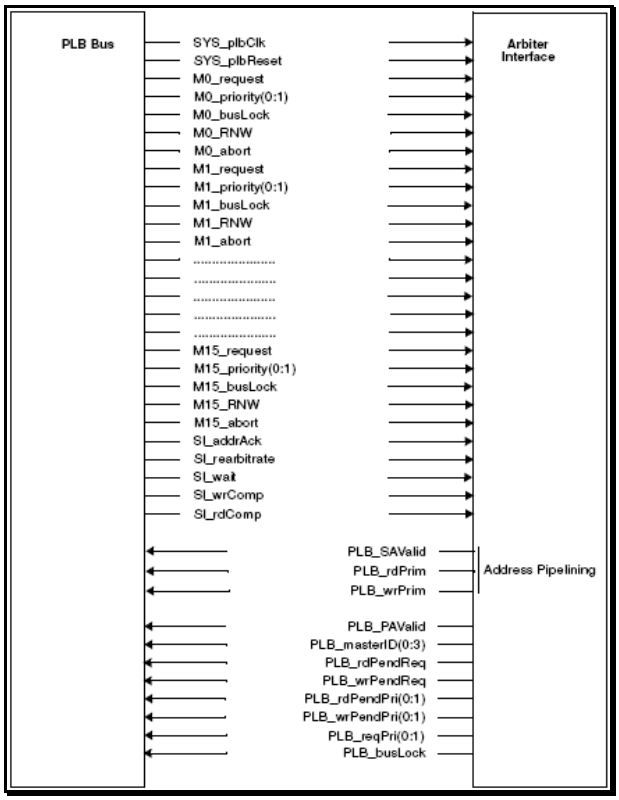


The PLB I/O signals are grouped under the following interface categories depending on their function:

- PLB Master Interface
- PLB Slave Interface
- PLB Arbiter Interface







3.1.1 PLB Features

The PLB addresses the high performance and design flexibility needs of highly integrated Core+ASIC systems.

High Performance:

- Overlapping of read and write transfers allows two data transfers per clock cycle for maximum bus utilization
- Decoupled address and data buses support split-bus transaction capability for improved bandwidth
- Extendable Address pipelining reduces overall bus latency by allowing the latency associated with a new request to be overlapped with an ongoing data transfer in the same direction
- Hidden (overlapped) bus request/grant protocol reduces arbitration latency
- PLB is a fully synchronous bus

System Design Flexibility:

- Bus architecture supports up to sixteen masters and any number of slave devices
- Four levels of request priority for each master allow PLB implementations with various arbitration schemes
- 32, 64, 128, 256-bit data bus implementations
- Bus arbitration-locking mechanism allows for master-driven atomic operations
- Byte-enable capability allows for unaligned transfers and odd-byte transfers
- Support for 16-, 32-, and 64-byte line data transfers
- Read word address capability allows slave devices to fetch line data in any order (that is, targetword- first or sequential)
- Sequential burst protocol allows byte, halfword, and word burst data transfers in either direction
- Guarded and unguarded memory transfers allow a slave device to enable or disable the prefetching of instructions or data
- DMA buffered, flyby, peripheral to memory, memory to peripheral, and DMA memory to memory operations are also supported
- Optional parity support provides enhanced data protection where necessary

3.1.2 PLB Transfer Protocol

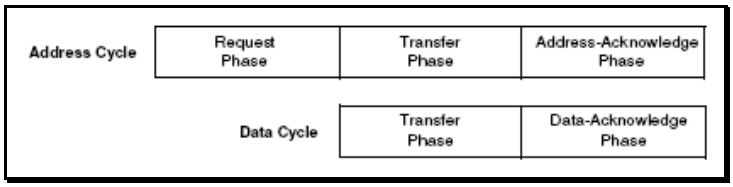
A PLB transaction is grouped under an address cycle and a data cycle.

The address cycle has three phases: request, transfer, and address acknowledge. A PLB transaction begins when a master drives its address and transfer qualifier signals and requests ownership of the bus during the request phase of the address cycle. Once bus ownership has been granted by the PLB arbiter, the master's address and transfer qualifiers are presented to the slave devices during the transfer phase.

During normal operation, the address cycle is terminated by a slave latching the master's address and transfer qualifiers during the address acknowledge phase.

Each data beat in the data cycle has two phases: transfer and data acknowledge. During the transfer phase the master will drive the write data bus for a write transfer or sample the read data bus for a read transfer. Data acknowledge signals are required during the data acknowledge phase for each data beat in a data cycle.

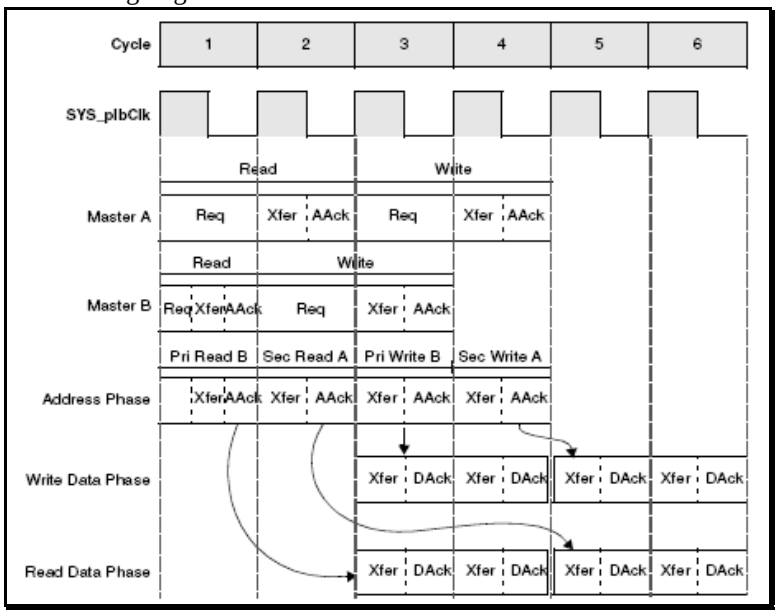
Note: For a single-beat transfer, the data acknowledge signals also indicate the end of the data transfer. For line or burst transfers, the data acknowledge signals apply to each individual beat and indicate the end of the data cycle only after the final beat.



3.1.3 Overlapped PLB Transfers

PLB address, read data, and write data buses are decoupled from one another allowing for address cycles to be overlapped with read or write data cycles, and for read data cycles to be overlapped with write data cycles. The PLB split-bus transaction capability allows the address and data buses to have different masters at the same time.

PLB address pipelining capability allows a new bus transfer to begin before the current transfer has been completed. Address pipelining reduces overall bus latency on the PLB by allowing the latency associated with a new transfer request to be overlapped with an ongoing data transfer in the same direction.



3.2 OPB Overview

The on-chip peripheral bus (OPB) is designed for easy connection of on-chip peripheral devices. It provides a common design point for various on-chip peripherals. The OPB is a fully synchronous bus which functions independently at a separate level of bus hierarchy. It is not intended to connect directly to the processor core. The processor core can access the slave peripherals on this bus through the PLB to OPB bridge unit which is a separate core. See the PLB to OPB bridge core specification for more information. Peripherals which are OPB bus masters can access memory on the

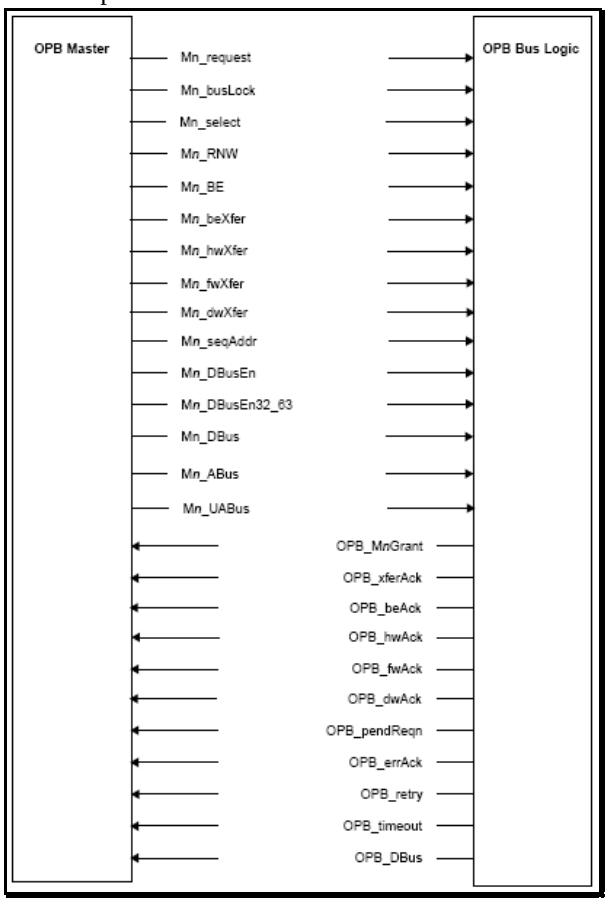
PLB through the OPB to PLB bridge unit which is a separate core. See the OPB to PLB bridge core specification for more information.

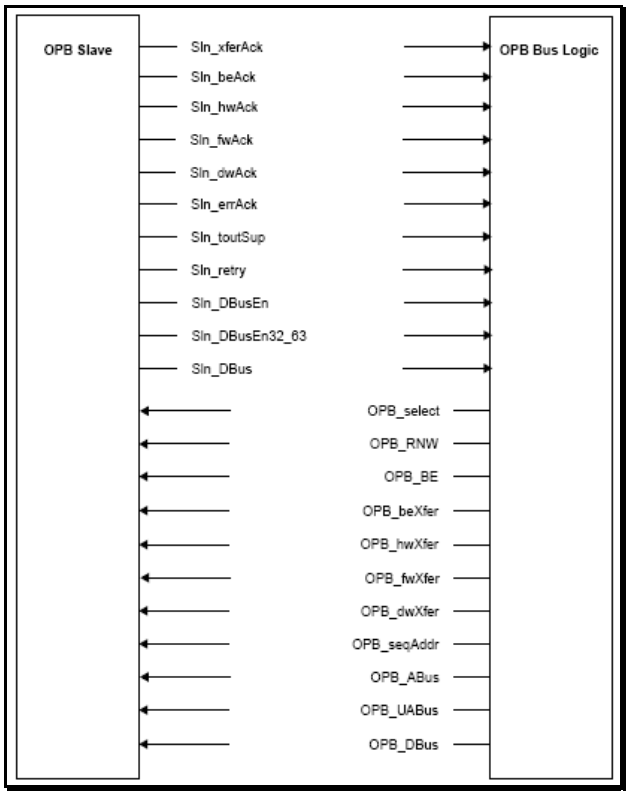
The on-chip peripheral bus has the following features:

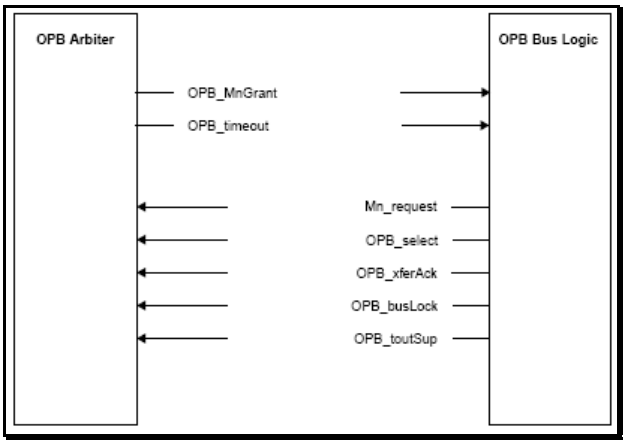
- Up to a 64-bit address bus
- 32-bit or 64-bit data bus implementations
- Fully synchronous
- Provides support for 8-bit, 16-bit, 32-bit, and 64-bit slaves
- Provides support for 32-bit and 64-bit masters
- Dynamic bus sizing; byte, halfword, fullword, and doubleword transfers
- Optional Byte Enable support
- Uses a distributed multiplexer method of attachment instead of threestate drivers, to ease manufacturing test. Address and data buses may be implemented in distributed AND-OR gates or as a dotted bus
- Byte and halfword duplication for byte and halfword transfers
- Single cycle transfer of data between OPB bus master and OPB slaves
- Sequential address protocol support
- Devices on the OPB may be memory mapped, act as DMA peripherals, or support both transfer methods
- A 16-cycle fixed bus timeout provided by the OPB arbiter
- OPB slave is capable of disabling the fixed timeout counter to suspend bus timeout error
- Support for multiple OPB bus masters
- Bus parking for reduced latency
- OPB masters may lock the OPB bus arbitration
- OPB slaves capable of requesting retry to break possible arbitration deadlock
- Bus arbitration overlapped with last cycle of bus transfers

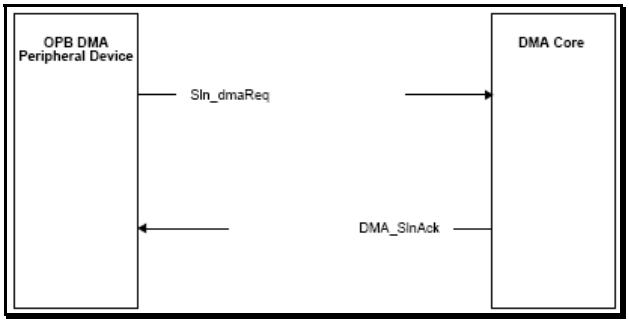
The OPB I/O signals are grouped under the following interface categories depending on their function:

- OPB Master Interface
- OPB Slave Interface
- OPB Arbiter Interface
- Optional DMA Interface



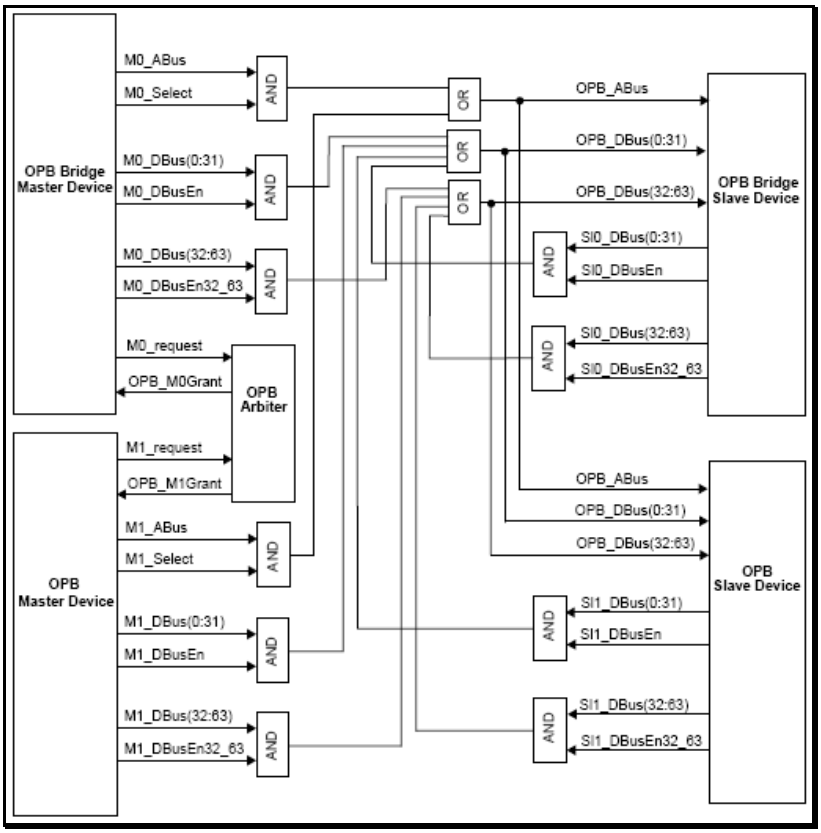






3.2.1 Physical Implementation

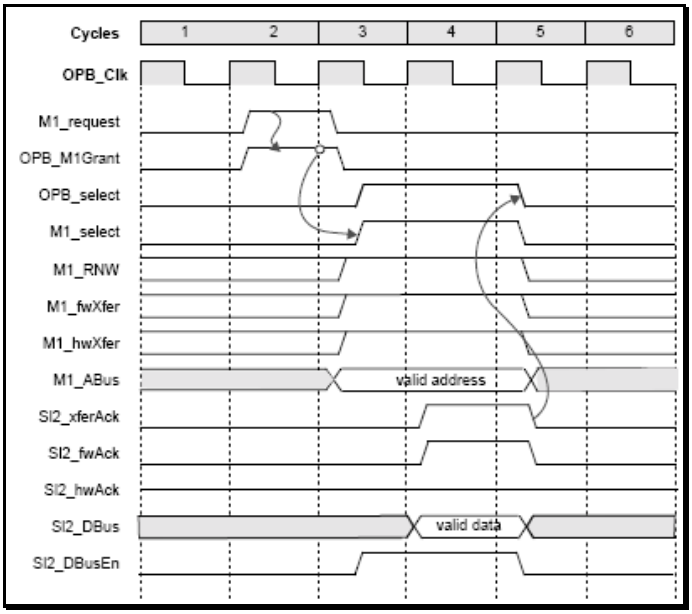
Since the OPB supports multiple master devices, the address bus and data bus are implemented as a distributed multiplexer. This design will enable future peripherals to be added to the chip without changing the I/O on either the OPB arbiter or the other existing peripherals. By specifying the bus qualifiers as I/O for each peripheral (select for the ABus, DBusEn for the DBus), the bus can be implemented in a variety of ways, that is, as a distributed ring mux (shown below), using centralized AND/OR's or multiplexers, using transceiver modules and a dotted bus, etc. The optimal design for each implementation will vary, depending on the number of devices attached to the OPB and timing and routing constraints.



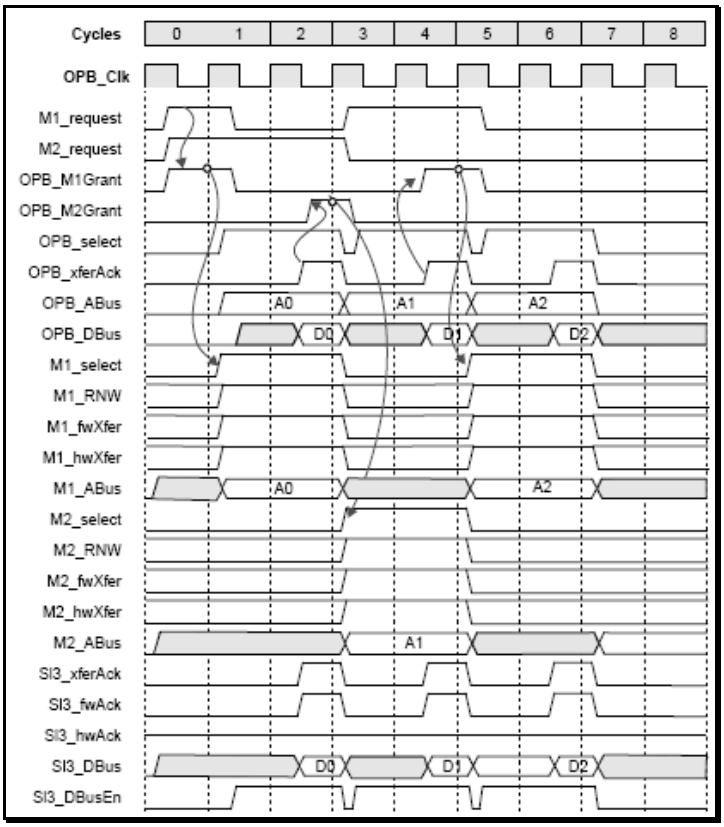
Control signals from OPB masters and slaves to and from the OPB arbiter and the peripherals will be similarly OR'ed together, and then sent to each device. Bus arbitration signals such as Mn_request and OPB_MnGrant are directly connected between the OPB arbiter and each OPB master device.

3.2.2 Bus Transfer Examples

The 1st figure shows typical OPB data transfer cycle. In the following example, fullword master device 1 reads data from fullword slave device 2. Note that slave device 2 has a two-cycle latency. When the OPB master device requires access to OPB, it asserts its request signal. The OPB arbiter will assert the master's grant signal according to bus arbitration protocol, and during a valid bus arbitration cycle. The requesting OPB master device assumes OPB ownership by asserting its select signal, in the cycle following that in which it samples its grant at the rising edge of OPB Clock. The slave completes the transfer by asserting xferAck, which causes the master to latch data from the data bus on read transfers, and deassert select.



The 2nd figure shows a more general case of OPB data transfer. In the following example, two OPB masters, master 1 and 2, require the OPB and assert requests in the same cycle. Master 1 is a fullword device, and requests a read operation from slave 3. Master 2 is a fullword device and also requests a read operation from slave 3. Slave 3 is a fullword device with a two-cycle latency. OPB Master 1 has high bus priority.



In the above example, the data transfer operation proceeds as follows:

- Master 1 and 2 assert requests in cycle 0. OPB arbiter grants to master 1 due to its higher priority by asserting OPB_M1Grant
- Upon sampling OPB_M1Grant on the rising edge, master device 1 initiates a data transfer in cycle 1 by asserting M1_select, and negates M1_request. The OPB arbiter negates OPB_M1Grant. OPB_M2Grant is not asserted since this is not a valid arbitration cycle (bus is busy but not in final transfer cycle)
- Slave 3 acknowledges in cycle 2, indicating completion of the data transfer, by asserting Sl3_xferAck. Master 1 latches data on OPB_DBus at the end of this cycle, and relinquishes control of the bus by deasserting M1_select, which also gates off all its control signals. The OPB arbiter asserts OPB_M2Grant upon the assertion of OPB_xferAck, overlapping arbitration with the data transfer
- Master 2 then samples OPB_M2Grant at the rising edge in cycle 3, and initiates data transfer by asserting M2_select, and negates M2_request. The OPB arbiter negates OPB_M2Grant
- Slave 3 acknowledges in cycle 4, indicating completion of the data transfer, by asserting Sl3_xferAck. Master 2 latches data on OPB_DBus at the end of this cycle, and relinquishes control of the bus by deasserting select, which also gates off all of its control signals. The OPB arbiter asserts OPB_M1Grant upon the assertion of OPB_xferAck, overlapping arbitration with the data transfer
- Upon sampling OPB_M1Grant on the rising edge in cycle 5, master 1 initiates a data transfer by asserting M1_select, and negates M1_request. The OPB arbiter negates OPB_M1Grant
- Finally slave 3 acknowledges in cycle 6, indicating completion of the data transfer, by asserting it's Sl3_xferAck signal. Master 1 latches data on OPB_DBus at the end of this cycle, and relinquishes control of the bus by deasserting M1_select, which also gates off all its control signals

Under this protocol, bus arbitration and data transfer are overlapped. This allows OPB to transfer data efficiently, avoiding dedicated bus arbitration cycles.

3.3 DCR Overview

The device control register (DCR) bus is designed to transfer data between a DCR master, typically a CPU's general purpose registers, and the DCR slave logic's device control registers. The DCR bus removes configuration registers from the memory address map, reduces loading, improving bandwidth of the processor local bus, and reducing latency to registers.

The DCR bus is a rising edge synchronous bus. Therefore, it is assumed that in a system-on-chip (SOC) environment where the DCR master and the DCR slave units are running at different clock speeds, the slower clock's rising edge always corresponds to the faster clock's rising edge. The DCR bus may be implemented in one of two possible configurations. It may be implemented by daisy-chaining the slave devices or by creating a distributed-OR structure out of the slave devices. The daisy chain approach may allow for easier chip-level wiring while the distributed-OR approach allows for easier chip-level timing closure.

The DCRs are on-chip registers that reside architecturally outside of the DCR master. They are accessed by using DCR read and DCR write operations. For PowerPC CPUs performing DCR accesses, these instructions are known as the "move to device control register" (mtdcr) and "move from device control register" (mfdcr) instructions. Additional instructions in the PowerPC instruction set may be implemented for automatic handling of the privileged signal and for 32-bit addressing. DCR transactions may control the configuration of on-chip peripherals, such as memory controllers, DMA controllers, arbiters, bridges... etc.

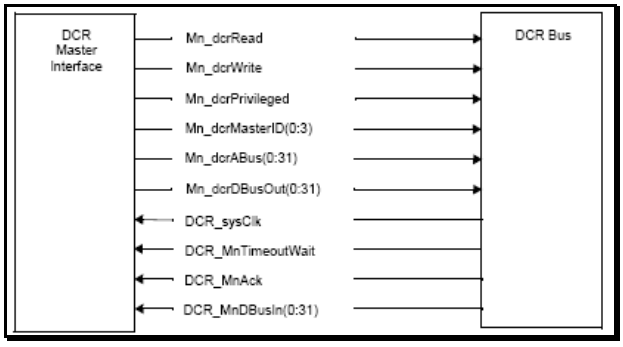
Features of the device control register bus include:

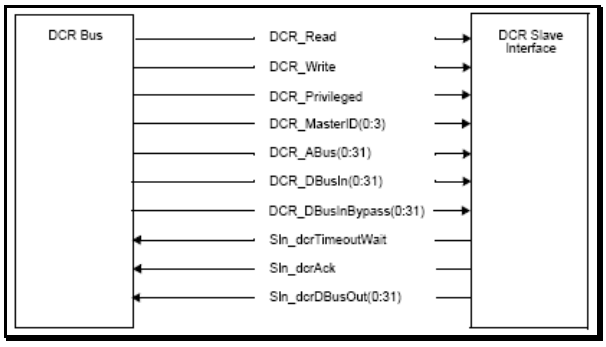
- 10-bit up to 32-bit address bus
- 32-bit data bus
- 4-cycle minimum read or write transfers extendable by slave or master
- Handshaking supports clocked asynchronous transfers
- Multi-master arbitration
- Slave bus timeout inhibit capability
- Privileged and Non-Privileged transfers
- Slaves may be clocked either faster or slower than master
- Daisy-chain (serial) or distributed-OR (parallel) bus topologies
- A simple but flexible interface

The DCR bus I/O signals are grouped under the following interface categories depending on their function.

The master may connect directly to one or more slave or attach to a DCR arbitration unit to allow multiple DCR masters to arbitrate for control of a single slave bus segment. The Mn prefix refers to master number "n" where n can take on value from 0-15. For an implementation with only one master n should be 0.

The Sln prefix refers to slave number "n" where n can take on any integer value.





4 BUS PROTOCOLS COMPARISON AND USABILITY FOR ASIC/SOC DESIGN

The main protocols advantages and disadvantages are summarized in the table below. It is not expected to be an objective comparison – the table reflects author's personal preferences and approaches to complex systems design.

Feature	AMBA	OCP	CoreConnect
High-performance bus	AXI	Profile to be selected	PLB
Peripheral bus	AHB	Profile to be selected	OPB
Configuration bus	APB or AHB	Profile to be selected	DCR
	Comparison between High-performance protocols		

Feature	AMBA	OCP	CoreConnect
Data pipe	Completely decoupled and overlapped Address, Read and Write transfers	Decoupled Address and Data phase is varied in accordance to profile	Completely decoupled and overlapped Address, Read and Write data phase
Out-of-order transfer	Master ID	Two levels: tags and threads	Not supported
Unaligned transfer	Supported	Supported	Supported
Protection Control	Supported	Supported	Supported
Security Control	Supported	Supported	Mapped on User attribute
Data word width	Flexible	Flexible	Flexible
Maximal burst	16, flexible increment	Configurable by design	16 for 32-bit PLB; 256 for wider PLB, flexible increment
Arbitration	External	Interface not specified	Embedded – up to 16 masters
Priority scheme	External	Not specified	Embedded – 4 levels
Integrability	De-facto standard	OCP-AHB bridges	PLB-AHB and OPB-AHB bridges
Concluded usage precedence	The Highest	Medium	The Lowest and only with PPC platform