



Power.org™ Embedded Bus Architecture Report

Presented by the Bus Architecture TSC

Version 1.0 – 11 April 2008

© Copyright 2008 Power.org. All rights reserved.

The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

Implementation of certain elements of this document may require licenses under third party intellectual property rights, including without limitation, patent rights. Power.org and its Members are not, and shall not be held, responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

THIS POWER.ORG SPECIFICATION PROVIDED “AS IS” AND WITHOUT ANY WARRANTY OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL POWER.ORG OR ANY MEMBER OF POWER.ORG BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, PUNITIVE, OR CONSEQUENTIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Questions pertaining to this document, or the terms or conditions of its provision, should be addressed to:

IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854
Attn: Power.org Board Secretary

Introduction

Power.org's mission is to develop, enable and promote Power Architecture™ technology as the preferred open standard hardware development platform for the electronics industry and to administer qualification programs that optimize interoperability and accelerate innovation for a positive user experience. Power.org seeks to solicit the participation of all interested parties on a fair, equitable and open basis.

Power.org's output includes:

- Open standards and specifications
- Business guidelines documents
- Best practices and education
- Certifications to validate implementations and drive adoption

Power.org's specifications will enable:

- Interoperability between community members
- Sustainability built on driving open standards and convergence

Revision History

Version	Date	Editor	Description
0.1	07/13/2007	Jason Hopp	Outline
0.2	08/02/2007	Jason Hopp	Added APB Content
0.3	01/14/2008	Jason Hopp	Added OPB and AHB content
0.4	01/30/2008	Jason Hopp	Added PLB4 content
0.5	02/12/2008	Jason Hopp	Added AXI content
0.6	03/04/2008	Jason Hopp	Added OCP content
0.7	03/17/2008	Jason Hopp	Added PLB6 content
0.8	03/24/2008	Jason Hopp	Incorporated feedback
0.9	04/02/2008	Jason Hopp	Added Summary
0.95	04/08/2008	Jason Hopp	Final review
1.0	04/11/2008	Jason Hopp	Release

Table of Contents

Introduction	2
Revision History	3
Table of Contents	4
1 Overview	8
1.1 Problem	8
1.2 Scope	8
1.3 Hierarchies.....	9
1.4 Purpose	10
1.5 Criteria.....	10
1.6 Buses Evaluated.....	10
1.6.1 CoreConnect Bus Family.....	10
1.6.2 AMBA Bus Family.....	11
1.6.3 OCP-IP Protocol.....	11
2 APB	13
2.1 APB description	13
2.2 APB strengths.....	13
2.3 APB Weaknesses.....	14
2.4 APB Core IP.....	15
2.5 APB Verification IP	16
2.6 APB Conclusion.....	16
3 OPB	17
3.1 OPB description	17
3.2 OPB strengths.....	17
3.3 OPB weaknesses.....	19

3.4	OPB Core IP	20
3.5	OPB Verification IP	21
3.6	OPB conclusion	22
4	AMBA AHB.....	23
4.1	AHB description.....	23
4.2	AHB strengths	24
4.3	AHB weaknesses	26
4.4	AHB Core IP	28
4.5	AHB Verification IP.....	30
4.6	AHB conclusion	30
5	CoreConnect - PLB4	31
5.1	PLB4 description	31
5.2	PLB4 strengths	33
5.3	PLB4 weaknesses	35
5.4	PLB4 Core IP	38
5.5	PLB4 Verification IP.....	39
5.6	PLB4 conclusion	39
6	AMBA - AXI.....	40
6.1	AXI description	40
6.2	AXI strengths	41
6.3	AXI weaknesses	44
6.4	AXI Core IP.....	46
6.5	AXI Verification IP	47
6.6	AXI conclusion.....	48
7	Open Core Protocol (OCP).....	49
7.1	OCP description	49
7.2	OCP strengths.....	50

7.3	OCP weaknesses.....	53
7.4	OCP Core IP	55
7.5	OCP Verification IP	58
7.6	OCP conclusion.....	58
8	CoreConnect - PLB6	60
8.1	PLB6 description	60
8.2	PLB6 strengths	61
8.3	PLB6 weaknesses	65
8.4	PLB6 Core IP	65
8.5	PLB6 Verification IP.....	65
8.6	PLB6 conclusion	66
9	Summary	67
9.1	Low-performance Bus Comparison.....	67
9.2	Mid-performance Bus Comparison	68
9.3	High-performance Bus Comparison.....	71
9.4	Dual Family Architecture	74

Power.org™ Embedded Bus Architecture Report

Executive Summary

The Power.org Bus Architecture Technical Subcommittee seeks to define internal bus architectures for use in System on a Chip designs based on Power Architecture processors. The overall bus architecture chosen by the Bus Architecture TSC is a bus hierarchy having three levels: a low-performance bus, a mid-performance bus, and a high-performance bus.

The purpose of this report is to describe how different buses occupy and span the three levels. Potential Power Architecture customers will understand what buses are recommended, what features are available on each, how they compare and connect to one another, and what types of IP cores and verification enablement are available for each.

The Bus Architecture TSC recommends the following buses for new core IP and SoC designs based on Power Architecture:

High-Performance (> 10GB/s)

- PLB6

Mid-Performance (2 to 10GB/s)

- AXI
- PLB4
- AHB

Low-Performance (< 2GB/s)

- AHB
- APB

The Bus Architecture TSC also recommends that new Power Architecture based SoC designs use existing OPB core IP in the low-performance range.

OCP may be used in a controlled design environment where all features and configuration option requirements are understood by the core IP developers.

Finally, the Bus Architecture TSC recommends the following be developed to enable more design-wins for Power:

- Power Architecture processor gasket to AXI

1 Overview

The Power.org Bus Architecture Technical Subcommittee (TSC) seeks to adopt internal bus architectures for use in System on a Chip (SoC) designs based on Power Architecture processors. It is in the best interest of the Power.org participants to increase the availability of Intellectual Property (IP) cores from which to design complete SoC products. The availability of a common internal bus architecture would foster this goal, and the propagation of Power Architecture within the marketplace.

It is expected that the adoption of a common bus architecture would result in lower development costs, increased compatibility, rapid reuse, reduced time to market, increased IP licensing opportunities, more effective allocation of ecosystem resources, and a stronger and more effective ecosystem.

1.1 Problem

There are many companies that develop core IP for SoC products. The interfaces to these cores can differ from company to company and can sometimes be proprietary in nature. The SoC developer then must expend time, effort, and money to create “bridge” or “glue” logic that allows all of the cores inside the SoC to communicate properly with each other. Incompatible interfaces are thus barriers to both IP developers and SoC developers. SoC integrated circuits envisioned by this subcommittee span a wide breadth of applications, target system costs, and levels of performance and integration.

1.2 Scope

The scope of the Bus Architectures TSC is the set of licensed buses and bus architectures that will enable the embedded SoC application space. Embedded SoCs are used in a wide variety of applications and performance points. The Bus Architectures TSC will define a set of buses and bus architecture topologies that enable applications spanning the low end to the high end of the embedded SoC space. At the high end, these buses should support embedded-class processors (allowing for multiple processors on the bus), accelerators, DDR2/DDR3 memory controllers, PCI Express Gen 2/Gen 3, Gigabit and 10-Gigabit ethernet, and other high speed I/O devices or bridges. At the low end, these buses should support devices like UARTs, I2C, USB, Universal Interrupt Controllers, General Purpose I/O, etc. Buses which support server class multi-processor systems (i.e. many coherent, high bandwidth processors) are not evaluated in this document, however it must be noted that there are a few multi-processor capable buses for Power emerging in the embedded space.

External (to the chip) bus interfaces and architectures were also defined to be beyond the scope of the Bus Architectures TSC for the following reasons: Applications such as accelerators are either being developed as on-chip solutions or are being attached via existing high-performance I/O interconnects, such as PCI Express or Hypertransport. An external bus that is kept coherent with the internal processors and caches is difficult to do in the general case and is therefore typically done only in the case of homogeneous multiprocessors, i.e. allowing a chip to attach externally to a clone of itself. However, this type of bus interconnect is not the kind of interface that would be useful to standardize. Bus standardization creates great leverage for things like memory interfaces, I/O interfaces, and on-chip SoC interfaces, because it provides third party IP developers the ability to develop standard devices that will “plug and play” in many systems. However, all of that leverage is lost in the case of a homogeneous coherent external interface.

1.3 Hierarchies

The Bus Architectures TSC recognizes that a bus hierarchy, rather than a single bus architecture, is needed to efficiently cover the above range of applications and products and their associated performance points, cost points and power points. Bus Hierarchies serve a number of purposes: 1) they enable a larger number of total devices by putting different devices on different levels, 2) they allow simple and slow devices to be designed to simple and slow protocols which use fewer wires and less gates, and 3) they provide greater aggregate system bandwidth by offloading slower transactions to lower levels, thus freeing up bandwidth on the higher levels, which are closer to the processor, and where performance is more critical.

The Power.org Bus Architectures TSC has defined three levels of bus hierarchy and has identified them as the “low-performance bus”, the “mid-performance bus” and the “high-performance bus”. The low-performance bus level supports standard low bandwidth SoC peripheral devices such as I2C, GPIO, UARTs, Timers, and UICs. The mid-performance bus level should have approximately 2 to 10 GB per second (using 90 nm technology as the reference point) of aggregate bandwidth per bus segment. (A bus segment is an instance of the bus where slaves are attached; buses can be architected using more than one bus segment so that different masters can access different slaves simultaneously. In general a given master can access a slave on any bus segment.) The mid-performance bus level should support low to mid-range embedded Power processors, and higher bandwidth I/O devices and bridges such as PCI, PCI-X, Gigabit Ethernet, and DDR memory. The high-performance bus level should have more than 10GB per second of aggregate bandwidth per bus segment. The high-performance bus level should support high end embedded Power processors, and the highest bandwidth I/O devices and bridges, such as PCI Express, Hypertransport, 10 Gigabit Ethernet and DDR3 memory.

1.4 Purpose

The purpose of this report is to evaluate each bus with respect to protocol, performance, technical features, and availability of core and verification IP. Customers of Power Architecture will understand what buses are recommended, what the differences are between them, and what types of cores and verification enablement are available for each.

1.5 Criteria

The Power.org Bus Architectures TSC studied a number of buses in order to determine the hierarchy of buses that will become standardized for Power.org. The three main criteria used to select buses for consideration were:

1. The level of acceptance and penetration in the industry: It would hinder acceptance of the bus hierarchy if an obscure set of buses were chosen that did not have an existing industry support structure and library of useful available IP.
2. The technical ability to support the wide range of applications found in typical SoCs: It would be a mistake to specify a set of buses that lacked the features or sufficient bandwidth to support common SoC applications.
3. The ease of interfacing to Power Architecture processors: Given that the main goal of Power.org is to promote Power Architecture processors, it would make little sense for the Power.org Bus Architectures TSC to specify a set of buses that were difficult or cumbersome to attach to Embedded Power Processors or would add unnecessary bridges in the main transaction paths.
4. The ability to easily obtain a license for the bus architecture to create verification and core IP. Buses that have been restricted to internal development only were not considered in this report.

1.6 Buses Evaluated

The following buses were considered for this report, based on their industry acceptance, technical merits, and enablement of Power processors.

1.6.1 CoreConnect Bus Family

The logical starting point was the set of buses that are native to the existing Power Architecture embedded 4xx processors. These buses are collectively called “CoreConnect” and are comprised of OPB and PLB4, with PLB6 currently

under development. The OPB (On-chip Peripheral Bus) is considered a low-performance, peripheral bus. The PLB4 bus (Processor Local Bus 4) is considered a mid-performance bus, and the PLB6 bus (Processor Local Bus 6) is considered a high-performance bus.

1.6.2 AMBA Bus Family

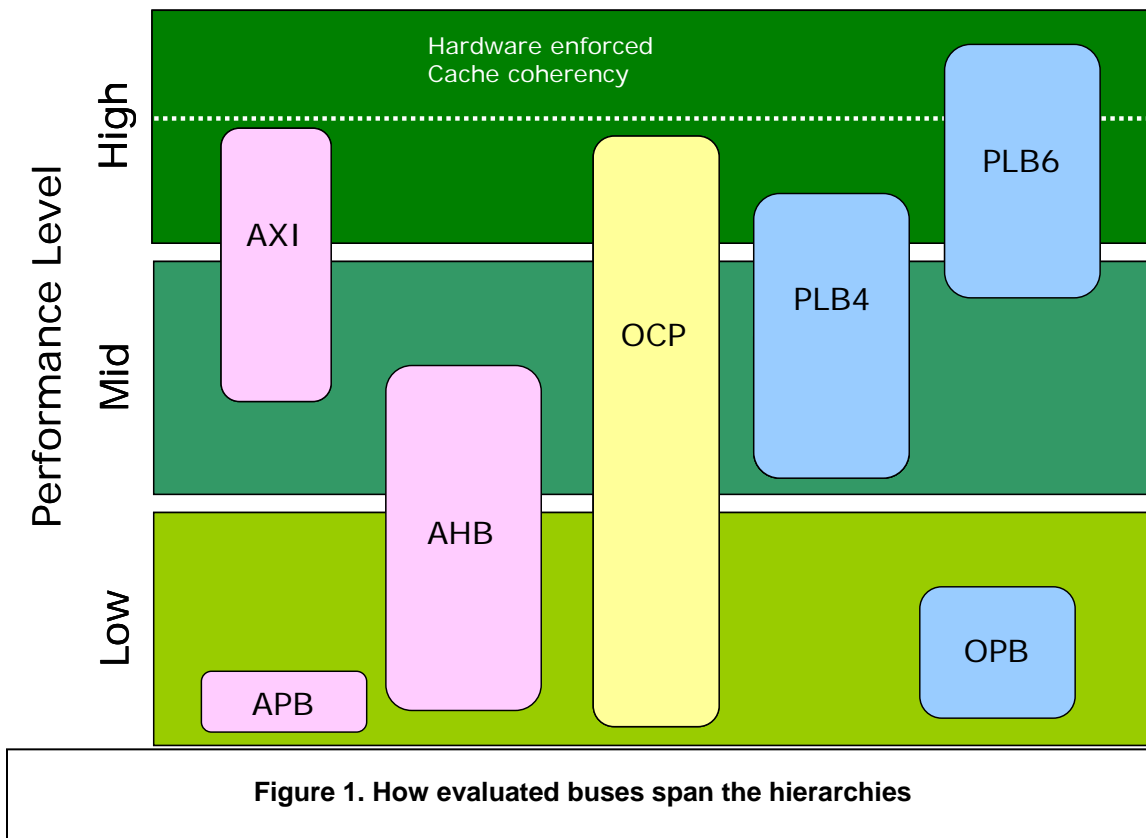
Another family of buses that the Power.org Bus Architectures TSC decided to investigate is the ARM AMBA (Advanced Microcontroller Bus Architecture) set of buses: APB, AHB and AXI. The APB (Advanced Peripheral Bus) is considered a low-performance, peripheral level bus. The AHB (Advanced High-performance Bus) is considered (despite the bus name) a mid-performance bus and the AXI (Advanced eXtensible Interface) bus is considered a high-performance bus. The AMBA family of buses was chosen for consideration because of their widespread acceptance in the industry and large amount of existing IP cores.

We note in passing that the APB bus is not always used as a standalone bus. Some AMBA based IP cores will use a higher performing bus like AHB or AXI for the primary function, but use an APB port for access to configuration registers. This gets configuration accesses off of the main bus which can be reserved for higher bandwidth operations. OPB is never used in this way because CoreConnect provides an additional bus called DCR whose purpose is to offload configuration accesses. All of the APB IP cores discussed here are standalone devices so that this report will be an apples to apples comparison of the two buses.

1.6.3 OCP-IP Protocol

The Bus Architectures TSC also investigated the Open Core Protocol (OCP), due to its openness, its definition being driven by participation from its members, its wide range of supported bus features, and its acceptance within several large companies that design their own IP. OCP-IP spans all three levels of performance hierarchy using one architecture specification, but low, mid, and high performance IP are still tailored to suit their individual performance requirements. OCP-IP has been defined since 2001, and has gained acceptance in markets such as consumer electronics and gaming.

The AMBA, CoreConnect, and OCP buses evaluated are shown below in Figure 1. Note that the features and functions of each bus are usually not neatly contained within one performance level. There is often enough flexibility within the architecture that allows implementations to span across performance levels. Note that the upper half of the high performance level is reserved for buses that support hardware-enforced cache coherency, and that only one of the buses evaluated, PLB6, will support this feature.



2 APB

2.1 APB description

The APB Bus is the lowest performance bus in the AMBA family. There are separate address (PADDR), write data (PWRITE), and read data (PRDATA) buses, up to 32-bits each. With 7 additional control signals, there can be up to 103 I/O for each APB slave.

There is one APB master, usually the bridge from a higher performance bus, that begins a transfer by asserting the appropriate PSELn signal with PADDR. PWRITE is active for a write and inactive on a read. PENABLE is asserted in the second clock, and is held active until PREADY is returned by the slave. The minimum transfer, read or write, is two clocks. APB slaves also have the option of inserting wait states for reads or writes by withholding PREADY. There is an optional PSLVERR signal used by the slave to report an error on a read or write with PREADY.

Assuming a frequency of 133MHz, (which should be achievable in a 90nm or smaller process) and the maximum data bus widths of 32-bits, the overall APB bandwidth could be as high as 267MB/s, but that assumes that none of the APB slaves would insert any wait states. Also, the total APB bandwidth must be shared between all of the APB slaves.

2.2 APB strengths

Of the buses reviewed, the APB offers a very low cost (based on area), very low power (based on I/O), and the least amount of complexity. It is intended for use with simple slaves that do not require much bandwidth.

1. Interface Simplicity: The APB interface and protocol are very simple and easy to learn. Relatively little effort needs to be spent on APB design and verification, leaving more time to focus on the device logic.
2. Core IP: There are more than 75 APB cores available from at least 20 different companies. The cores cover a broad range of small bandwidth applications. There is also verification enablement in models and checkers/monitors available from at least 6 different companies with support for 5 different languages.

3. Verification IP: There is verification enablement for APB in models and checkers/monitors available from at least 6 different companies with support for 5 different languages
4. Bridges Exist: APB cores are accessed by the AHB-to-APB or AXI-to-APB Bridge. Masters on PLB4, including Power Architecture processors on PLB4 can use the PLB4-to-AHB Bridge to access APB slaves via the AHB-to-APB Bridge.
5. IP Core Portability: Most of the IP cores designed with an APB interface are available as “synthesizable” cores. The source code, written in Verilog or VHDL, is provided and synthesized into the target technology. This IP delivery method typically presents few timing challenges at the 66-133MHz range expected for APB.
6. Error Reporting: APB defines an optional signal, PSLVERR, driven by APB slaves on reads or writes with PREADY to indicate an error with the transfer.

2.3 APB Weaknesses

APB is a low performance bus, and, as such, offers no advanced features available in mid- and high-performance buses, such as bursting and data pipelining. However, APB also has a few other shortcomings.

1. Slaves Only: The only master on APB is a bridge from a higher level bus, so only slaves may be designed to the APB. This prevents external bus masters from using this simple interface, and forces them to a higher level bus, increasing their complexity.
2. One Bus Size: All APB slaves must be the same bus width. This is considered a strength in high-performance core IP, since it simplifies master designs. However, for APB, this may force some slaves to be larger than intended, and others to add complexity to have the ability to reside on APB's of different bus widths.
3. No Retry: There is no mechanism for the APB slave to inform the Bridge that it is not ready now, so the Bridge can move on to another request, and try again later. This means that all transfers are sequential through the Bridge, and that a long latency in any APB slave can delay transfers to/from other devices.
4. No Timing Specifications: There are no timing requirements that APB cores developers must meet, either in an absolute sense or relative to a clock cycle. This may not present a major problem at low frequencies, however, since there is no gauge of how much

combinatorial logic between clocks is appropriate, closing timing at an SoC level at higher frequencies will be more challenging.

5. No Byte Enables: Although APB can only have a maximum of 32-bit read and write data buses, it would still be useful to have byte enables defined. This is because three-byte reads and writes require two transfers. This could be accomplished in a single transfer if byte enables were used.

2.4 APB Core IP

There are many different types of relatively low-performance cores that attach to APB. The following table was compiled using information gathered from <http://www.design-reuse.com>, but does not reflect a complete list of APB core IP.

No. of Cores	Type	No. of Suppliers
7	AHB to APB Bridge	7
1	AXI to APB Bridge	1
1	Codec	1
1	Compact Flash (CF)	1
1	DMA Controller	1
1	Elliptical Curve Cryptology (ECC)	1
6	General Purpose I/O (GPIO)	6
6	General Purpose Timer	6
6	I2C	5
1	I2S	1
4	Interrupt Controller	4
2	IrDA	2
1	Keyboard/Mouse Controller	1
1	Memory Stick	1
1	NAND Flash	1
3	Real Time Clock (RTC)	3
1	Reset Controller	1
1	SDLC	1
2	Secure Digital (SD)	2
3	Smart Card	3

No. of Cores	Type	No. of Suppliers
4	SPI	4
2	SPI/Microwire/SSP	2
5	UART	5
2	Watchdog Timer	2

2.5 APB Verification IP

There are several choices in verification bus functional models, monitors/checkers, assertions, and coverage/scoreboards for APB. The following information was compiled using information gathered from <http://www.design-reuse.com>, but does not reflect a complete list of verification IP available for APB.

APB has a large amount of verification support available in the industry. There are master and slave BFMs available from 5 different suppliers written in ‘e’, SystemVerilog, Verilog, or VHDL. There are monitors, checkers, or assertions available from 9 suppliers.

2.6 APB Conclusion

The APB fits squarely within the low-performance bus definition. It follows a simple protocol with no burst transfers, data pipelining, or byte enables. There is only one master – the bridge from a higher level bus – and only one bus width per implementation. However, even with these restrictions, there are many cores available with low bandwidth requirements that reside on APB and perform their own functions well. There is also an impressive amount of verification IP available in the industry, supporting several languages and test environments.

3 OPB

3.1 OPB description

OPB is the lowest performance bus in the CoreConnect family. The original specification defined separate upper address (UABus), lower address (ABus), and data (DBus), buses, up to 32-bits each. With 19 additional control signals, the OPB, as originally defined, can have up to 115 I/O for each OPB master.

An enhanced definition of OPB later included byte enables, and extended the data bus to 64-bits, along with corresponding controls, for a total of 160 I/O for each enhanced OPB master.

In SoCs where only OPB slaves are required, the PLB-to-OPB Bridge acts as the only master. In other systems, there may be multiple OPB masters that make requests and are granted access to the OPB by an arbiter. There are no architectural restrictions which limit the number of OPB masters, but the current implementation of the OPB arbiter supports up to four. An OPB master begins a transfer by asserting its Mn_request. When the OPB arbiter asserts OPB_MnGrant, the master asserts Mn_select along with the appropriate transfer qualifiers Mn_UABus, Mn_ABUS, Mn_RNW, and others. Mn_DBus is also driven if the transfer is a write. The OPB slave indicates that it accepts the transfer by asserting Sln_xferAck, and the master either deasserts its Mn_select or provides new transfer qualifiers while leaving Mn_select active, if OPB_MnGrant is active. Read data is driven by the slave when Sln_xferAck is asserted. Write data is captured by the slave when Sln_xferAck is sampled active. The slave has the option of inserting wait states in read or write transfers by delaying the assertion of Sln_xferAck. There is an optional Sln_errAck signal asserted by the slave with Sln_xferAck to report an error on a read or write.

Assuming a frequency of 166MHz, which should be achievable in a 90nm or smaller process, and the widest data bus width of 64-bits, the overall OPB bandwidth could be as high as 1.33 GB/s, but that assumes that none of the OPB slaves would insert any wait states. The more common bus width of 32-bits would yield a bandwidth as high as 667 MB/s. Note that the total OPB bandwidth must be shared across all of the OPB slaves.

3.2 OPB strengths

OPB offers a low cost (in area), low power (based on I/O) bus with a moderate amount of additional complexity over APB. It may be used with both with simple slaves that do not require much bandwidth and with more sophisticated masters

and slaves that require more bandwidth and a request per data beat burst capability.

1. **Bus Master Capability:** OPB can support multiple bus masters, and the current OPB arbiter implementation supports up to four. The primary use of this feature is to enable devices outside the SoC to access internal SoC memory using a simple interface. OPB also offers bus parking and overlapped arbitration to reduce any performance impacts of arbitrating between multiple masters. Bus locking is also supported to support atomic operations.
2. **Dynamic Bus Sizing:** This is a useful feature that allows masters and slaves of different bus widths to interoperate with one another. For example, a 32-bit master may attempt a four byte write to an 8-bit slave. When the master receives the first `_xferAck`, it will increment its address by one byte and make another request. Four write requests will be made by the master to complete the write transfer.
3. **Retry Capability:** OPB allows slaves that cannot currently respond to retry transfers via the `SI_retry` output. This enables better bus utilization, since the alternative would be for the slave to simply hold the bus and insert wait states until it was ready to respond with `SI_xferAck`. The retry may also be used by high latency slaves to perform “delayed reads”, where the read is queued and performed, but responded with `SI_retry` until the read data is available.
4. **Timing Specifications:** OPB defines timing requirements relative to a clock cycle for all inputs and outputs of masters and slaves on the bus. This allows SoC developers to judge how compatible masters and slaves will be.
5. **Core IP Support:** There are more than 50 OPB cores available from at least 2 different companies. The cores cover a broad range of low bandwidth applications.
6. **Interface Simplicity:** OPB slaves may be designed with a simple subset of the architecture with a data bus width as small as 8-bits. However, OPB masters must be more complex than APB slaves, since they must support OPB slaves of differing bus widths.
7. **Bridges Exist:** Power Architecture processors, and other masters on PLB4, can use the PLB4-to-OPB Bridge to access OPB slaves. In the other direction, OPB masters may access PLB4 slaves via the OPB-to-PLB4 Bridge. AMBA AHB masters can communicate with OPB slaves via the AHB-to-OPB Bridge.

8. Sequential Address Burst: OPB offers a type of bursting via a signal driven by the master; Mn_seqAddr. Slaves that decode this input are aware that the next address will be sequential with respect to the current address. This allows at least some prefetched read data to be sourced in a single clock without an address decode, and enables data packing on writes. There is, however, no indication of the overall length of the burst.
9. Byte Enables: OPB defines byte enables, although many of the core implementations do not take advantage of this, making use of the dynamic bus sizing feature instead.
10. Error reporting: OPB defines a signal, SI_errAck, that is driven by slaves on reads or write transfers with SI_xferAck to indicate an error.
11. IP Core Portability: Most of the IP cores designed with an OPB interface are available as “synthesizable” cores. The source Verilog or VHDL is provided and synthesized into the target technology. This typically presents few timing challenges at the 66-166MHz range expected for OPB.

3.3 OPB weaknesses

OPB is a low performance bus, although it has more bandwidth than APB. It does offer a simple sequential address burst capability, but none of the other advanced features available in mid and high-performance buses, such as data pipelining. However, OPB also has one significant shortcoming.

1. Single-Sourced Verification IP: There is verification enablement available for OPB master and slave designs for IBM customers, including an OPB master BFM, slave BFM, and monitor. However, there is no third party toolkit available using a modern, constrained random verification methodology. While the toolkit available from IBM is capable enough to ensure a good design in a stand-alone environment, it is difficult to incorporate with models written to a more up-to-date methodology.

3.4 OPB Core IP

There are many different types of low-performance cores that attach to OPB. The following table was compiled using information gathered from both <http://www.design-reuse.com> and from IBM's IP Development Library, but does not reflect a complete list of OPB core IP.

No. of Cores	Type	No. of Suppliers
2	OPB Arbiter	2
2	PLB to OPB Bridge	2
2	OPB to PLB Bridge	2
1	AHB to OPB Bridge	1
1	OPB to OPB Bridge	1
1	OPB to DCR Bridge	1
2	DMA Controller	2
1	OPB to/from PCI Bridge	1
1	External Bus Controller (for slaves)	1
1	External Bus Controller (for masters)	1
2	SDRAM Controller	1
2	DDR Controller	1
1	DDR2 Controller	1
1	SRAM Controller	1
1	BRAM (Block RAM) Controller	1
1	IDE Host Controller	1
1	USB 2.0 Device Controller	1
3	Ethernet 10/100 EMAC	2
1	Ethernet 100/1G/10G XEMAC	1
1	Ethernet RGMII	1
1	Ethernet ZMII	1
1	Ethernet MIB	1
1	Audio Codec	1
1	NAND Flash Controller (via EBC)	1
1	PCMCIA Controller (via EBC)	1

No. of Cores	Type	No. of Suppliers
4	UART	2
1	Smart Card	1
1	Parallel Port	1
1	Serial Communication Port	1
1	Touch Panel Controller	1
1	HDLC	1
2	EMC Controller	1
1	System ACE Interface Controller	1
1	Delta-Sigma DAC	1
1	ATM Utopia Level 2	1
2	General Purpose I/O (GPIO)	2
2	I2C	2
1	Interrupt Controller	1
2	General Purpose Timer	2
1	Watchdog Timer	1

3.5 OPB Verification IP

There is an OPB Toolkit available to IBM customers, which includes master and slave BFM, an OPB monitor, and user's guide describing the Bus Functional Language (BFL) used to write test cases. All components are available in VHDL and Verilog.

The OPB Toolkit available from IBM gives customers the ability to verify OPB master and slave IP using deterministic test cases written in BFL, and then compiled into Verilog or VHDL using a provided compiler. It is also possible to place the master and slave BFM in a mode where they automatically generate and respond to requests, but there is a limited amount of control placed on the types of requests and responses. Due to the limited random control, the fact that the BFM are controlled through HDL that must be compiled each time a test case is run, and the lack of functional coverage points, it is difficult to make use of the OPB Toolkit in a modern, constrained random verification environment.

3.6 OPB conclusion

OPB has many features associated with a low-performance bus, plus a few others, such as bus mastering and sequential address bursting. The protocol is flexible enough to support very simple 8-bit slaves on the low end, up to 64-bit masters with byte enables on the high end. There is also only one source of verification IP for OPB, and it is most beneficial when used to write deterministic test cases, and as such does not fit in well with other verification environments using constrained random control with functional coverage points. However, even with these restrictions, there are many cores available for OPB that perform well, and, most importantly, have been proven to work in dozens of SoCs. There is no need for customers of these SoCs to incur additional risk by porting these proven cores to another bus.

4 AMBA AHB

4.1 AHB description

AHB is the mid-performance bus in the AMBA family. The protocol defines a 32-bit address bus (HADDR), but this has been extended in some implementations. The read and write data buses (HRDATA and HWDATA) may be defined under the specification as 2^n bits wide, from 8-bit to 1024-bit, but the most common implementation has been 32-bit. With 27 additional control signals, and assuming the most common address and data bus width of 32-bit, AHB can have up to 123 I/O for each AHB master.

Up to 16 AHB masters may be connected to a central interconnect which arbitrates between master requests, and multiplexes the winning request and corresponding transfer qualifiers to the slaves. Slave read data is multiplexed back to the masters.

An AHB master begins a transfer by asserting its HBUSREQx signal. When the AHB arbiter asserts HGRANTx and HREADY is active, the master performs the address phase of the current transfer by asserting HADDR with HTRANS[0:1] and other transfer qualifiers. The address phase is valid for one clock cycle, and HWDATA is also driven if the transfer is a write. The HADDR and other transfer qualifiers for the next transfer are driven by master in the next clock, and held until the AHB slave completes the data phase for the current transfer by asserting HREADY. Read data is driven by the slave in the same clock that HREADY is asserted. Write data is captured by the slave when HREADY is sampled active. The slave has the option of inserting wait states in read or write transfers by delaying the assertion of HREADY. The slave may respond with OKAY, ERROR, SPLIT or RETRY.

AHB always has one data phase HREADY that corresponds to the preceding address phase. So, there are no bursts where one address phase requests multiple data beats. Also, since the address can only be used by the slave when HREADY is active, AHB is limited to performing one transfer at a time, with only a one clock overlap between the previous data phase and current address phase. This is a significant drag on performance with respect to a more sophisticated bus that supports data pipelining. For example, an initial read from an AHB-based memory controller may take 60ns, due to the latency of the DRAM. If the subsequent read also takes 60ns, and can only be requested from memory when the initial read data is available with HREADY, then the AHB master must wait another 60ns (minus one clock cycle) for the subsequent read data. This is only a slight improvement over OPB, since the subsequent address on AHB does not have to be sequential to the initial address. AHB slaves with

longer latency may be designed to perform “delayed reads” by responding with either SPLIT or RETRY instead of OKAY. This allows the AHB to be utilized by other devices while the data is read data is being retrieved.

Assuming a frequency of 266MHz, which should be achievable in a 90nm or smaller process, and the most common data bus width of 32-bits, the overall AHB bandwidth would be as high as 1.07 GB/s, but that assumes that none of the AHB slaves would insert any wait states. At the same frequency, an AHB configured to 64-bit would yield a bandwidth of 2.13 GB/s. Note that, even though the AHB has separate read and write data buses, the bandwidth is limited by the single HREADY that completes the data phase.

4.2 AHB strengths

AHB offers a fairly low cost (in area), low power (based on I/O) bus with a moderate amount of complexity. It may be used with both with simple slaves that do not require much bandwidth and with more sophisticated masters and slaves that require more bandwidth and require a one request per data beat burst capability. AHB can achieve higher frequencies than OPB since the protocol separates the address and data phases, where OPB allows a single cycle master request and slave response. AHB can use the higher frequency along with separate data buses that can be defined to 128-bit and above to achieve the bandwidth required for mid-performance bus applications.

1. Wealth of IP: AHB, by far, boasts the largest number of IP cores available in the industry. There are choices from multiple vendors for the most sought after cores, and a large variety of specialized cores as well.
2. Excellent Verification Support: AHB also offers a large selection of verification IP from several different suppliers. The solutions offered support several different languages and run in a choice of environments.
3. Direct or Logic “Gasket” Attachment for Power: Freescale offers a licensable Power processor that attaches directly to AHB. IBM 4xx PowerPC processors natively attach to PLB4, and require some interface logic to translate from PLB4 to AHB. This logic has been implemented with a very small number of gates to connect the PPC405 to AHB. Further analysis is required to determine if attaching higher performance Power processors to AHB makes sense, as AHB does not support read and write data pipelining.
4. Bridges Exist: Power Architecture processors, and other masters on PLB4, can use the PLB4XAHB Bridge to access AHB slaves.

The same bridge may also be configured to be bi-directional, allowing AHB masters to access PLB4 slaves. Also, AHB masters may access OPB slaves via the AHB-to-OPB Bridge.

5. Moderate Complexity: While AHB masters and slaves may be more difficult to design than OPB masters and slaves, the increase may be somewhat offset with the availability of sophisticated verification support.
6. Delayed Read Capability: AHB allows slaves with significant latency to respond to read with an HRESP of “SPLIT.” The slave will then request the bus on behalf of the master when the read data is available. This enables better bus utilization, since the alternative would be for the slave to simply stall the bus and insert wait states by holding HREADY inactive until it was ready to respond with the read data.
7. One Request Per Data Beat Burst: AHB offers burst capability by defining incrementing bursts of unspecified length, as well as 4, 8, and 16 beat bursts, both incrementing and wrapping. Although AHB requires that an address phase be provided for each beat of data, the slave can still use the burst information to make the proper request on the other side. This helps to mask the latency of the slave.
8. Configurable Data Bus Width: AHB is defined with a choice of several bus widths, from 8-bit to 1024-bit. The most common implementation has historically been 32-bit, but higher bandwidth requirements may be satisfied by using 64 or 128-bit buses.
9. Protection Control: AHB masters use the HPROT signals to indicate whether or not a transfer is cacheable, protected, bufferable, or data vs. opcode.
10. Error Reporting: AHB used the HRESP signals driven by the slaves to indicate when an error has occurred.
11. IP Core Portability: The vast majority of the IP cores designed with an AHB interface are available as “synthesizable” cores. The source Verilog or VHDL is provided and synthesized into the target technology.

4.3 AHB weaknesses

While AHB can reach the bandwidth consistent with a mid-performance bus, and offers one request per data beat burst capability, it does not offer other advanced features available in mid and high-performance buses.

1. **No Data Pipelining:** AHB cannot achieve full data bus utilization and bandwidth if some slaves have a relatively high latency. For example, on a 200MHz AHB, if memory accesses take 60ns, there will be 12 “dead” clocks in between the memory read data beats. This is due to the fact that a subsequent request will only be accepted by the memory controller when the last data beat of the initial request is presented on the bus with HREADY.
2. **No Byte Enables:** Given a 128-bit bus, AHB defines transfer sizes of 1, 2, 4, 8, and 16 bytes. Because byte enables are not defined, there are cases where multiple transfers must be made inside a single quadword. For example, 4 transfers (doubleword, word, halfword, and byte) would have to be requested to obtain data for 15 of the 16 bytes on a 128-bit AHB. This could be accomplished in a single transfer if byte enables were defined and used.
3. **Data Bus Width Complexity:** An AHB slave design must support several transfers of several widths, since the architecture allows masters to request transfers of any of a number of sizes, up to and including the size of the bus. Additional steering and muxing logic is required in the slave design to handle this. There is further complication due to the fact that the data bus width itself is configurable, and any legal sizes up to the configured size must be supported.
4. **No Slave Burst Termination:** AHB allows for bursts of an undefined length, although AHB masters are responsible for ensuring that their requests do not cross 1K address boundaries. So, a 32-bit AHB master could tie up the data bus for 256 data beats, plus insert wait states each time the slave data buffer is required to be refilled. It is highly unlikely that slaves will be designed with 1K data buffers. AHB slaves are not able to stop masters from making more requests; only to insert wait states. AHB specifies that the arbiter is responsible for preventing excessive access to the bus from a single master. The arbiter is allowed to grant the bus to another master, even if the current master is in the middle of a burst. It is then up to the master to re-request the transfer, starting at the appropriate address with the update number of beats. However, the responsibility should lie with the slave, since it inherently understands its own data buffer limitations.

5. No “Cut Through” Writes: When a bridge from AHB to another bus begins to receive write data from a request specifying 4, 8 or 16 beats, it must wait see how many beats of write data will be received before making a request on the other side of the bridge. This is because AHB allows masters to either terminate a burst early or insert wait states in between the data beats. This results in a longer latency on these write transfers.
6. Timing Specifications: AHB defines timing parameters for many of the relationships between signals on the bus. However, these are not associated with requirements relative to a clock cycle. Therefore, SoC developers must integrate AHB cores and run chip level static timing analysis to judge how compatible AHB masters and slaves are with one another.
7. Parity Undefined: While there are most likely AHB implementations out there that have added parity, there is no substitute for including this information in the specification for interoperability. Left undefined, AHB masters may check read data for odd parity, while AHB slaves generate even parity.
8. Address Space: Interfaces such as PCI-X define a 64-bit address space. Although AHB is only architected as 32-bit, many in the industry have logically extended the bus to 64-bit when required.
9. Single Cycle Address Response: AHB requires slaves to respond to HTRANS with an HRESP of OKAY in one clock. Of course, this is more difficult to do at higher frequencies, especially in more complex slaves with more queues. Unfortunately, slaves with higher performance requirements are the ones most likely to have more queues, and the ones that will have the most difficulty closing timing at higher frequencies.
10. Central Address Decode: Power-based SoCs cover a wide range of applications, and there is a corresponding wide range of address map requirements. Having the address decodes for all AHB slaves reside within the interconnect means having to support the most complex split address ranges, even for the simplest of slaves. The number, size, and granularity of address ranges can also be made configurable, but this also adds complexity to the interconnect. Address range decodes are more appropriately placed in the slaves themselves, so that multiple split address ranges are only implemented in slaves that require such complexity.

4.4 AHB Core IP

There is a vast library of IP cores available on AHB. Some could be classified as low-performance, and others as mid-performance. Many are synthesizable, offering configurable data bus widths, which allow the cores to span a few different performance points. The following table was compiled using information gathered from both <http://www.design-reuse.com>, but does not reflect a complete list of AHB core IP.

No. of Cores	Type	No. of Suppliers
5	AHB Arbiter, Decode, and Control	4
1	Clockless Multilayer AHB	1
1	AHB to AXI Bridge	1
1	AHB Master to AXI Master Gasket	1
2	AHB to AHB Bridge	2
7	AHB to APB Bridge	7
1	AHB to OPB Bridge	1
2	Generic AHB Master Template	2
2	Generic AHB Slave Template	2
10	Microprocessor	6
2	CPU Gasket	2
1	L2 Cache Controller	1
8	DMA Controller	7
3	LCD Controller	3
2	I2S/SPDIF Audio Controller	1
2	DSP Core	1
2	JPEG Codec	2
3	MPEG-4 Codec	2
6	10/100 Ethernet Controller	6
3	10/100/1G Ethernet Controller	3
2	External Bus/Memory Interface	2
4	IEEE 1394a - Firewire	2
5	IDE Controller	5
2	SATA Controller	2

No. of Cores	Type	No. of Suppliers
1	Interrupt Controller	1
17	SDRAM Controller	7
8	DDR Controller	4
3	DDR2 Controller	1
10	SRAM Controller	5
6	NAND Flash Controller	5
4	NOR Flash Controller	3
8	Flash Controller	5
2	Compact Flash Controller	2
1	USB 1.1 Host Controller	1
4	USB 1.1 Device Controller	2
4	USB 2.0 Host Controller	3
6	USB 2.0 OTG Controller	4
4	USB 2.0 Device Controller	3
7	Secure Digital (SD) Controller	4
3	Multi-Media Card (MMC) Controller	2
1	XD Controller	1
1	HPNA Core	1
1	Spacewire Codec	1
1	Utopia 1/2/2+/3/3+	1
6	PCI Controller	4
1	AHB Master to PCI Master Gasket	1
3	AHB to PCI Express Bridge	3
3	AES/Triple DES Security	3
1	High Speed SPI	1
1	Serial RapidIO	1
1	Xtensa PIF to AHB Bridge	1
6	AHB-based SoC Platform	4

4.5 AHB Verification IP

There are many choices in verification bus functional models, monitors/checkers, assertions, and coverage/scoreboards available for AHB. The following information was compiled using information gathered from <http://www.design-reuse.com>, but does not reflect a complete list of verification IP available for AHB.

AHB has an impressive amount of verification support available in the industry. There are nine master and seven slave BFM's available from nine different suppliers written in 'e', SystemVerilog, Verilog, PureSpec, or OpenVera. There are monitors, checkers, or assertions available from 10 suppliers, written in 'e', PSL, SystemVerilog, Verilog, VHDL, PureSpec, or OpenVera.

4.6 AHB conclusion

AHB is similar to OPB in features, but does offer the ability for slaves to act on a declared burst length, supports delayed reads, and can reach mid-performance bandwidth through increases in frequency and an expandable data bus width. However, AHB is the most impressive due to its very large selection of IP cores available for attachment. In fact, AHB boasts the largest IP core library in the industry. AHB also offers a wide range of verification IP that supports many different languages and environments.

The limitations of AHB are technical in nature, and prevent it from being considered as a high-performance bus. Notably absent from AHB are data pipelining, simultaneous read and write data, and byte enables.

5 CoreConnect - PLB4

5.1 PLB4 description

PLB4 is the mid-performance bus in the CoreConnect family. The protocol defines a 64-bit address bus (PLB_UABus and PLB_ABus), but this has been implemented as 36-bit in some implementations with neither PCI-X nor PCI Express. The read and write data buses (PLB_MnRdDBus and PLB_wrDBus) may be defined under the PLB specification as 32, 64, or 128-bit. However, a de facto standard of 128-bit was set with the original PLB4 core implementations, and this has been re-enforced through the PLB4 Interoperability Specification released through Power.org. There are 77 additional control signals defined in the architecture specification, but 27 of these are optional. Assuming the full 64-bit address bus and all control signals are implemented, PLB4 can have up to 397 I/O for each PLB4 master. However, assuming a more common implementation of 50 control signals, there would be 370 I/O for each PLB4 master.

There are two arbiters available for PLB4. The first arbitrates among up to eight masters to route the winning requests to slaves on single slave segment. In the second, up to 12 PLB4 masters may be connected to a 2-way crossbar which arbitrates between master requests, and multiplexes the winning request and corresponding transfer qualifiers to one of the two slave segments. There is a mechanism in the crossbar that prevents read requests from being made on a segment if there is a read in progress from the same master on the other segment. This prevents returning read data collisions, but restricts data pipelining from a single master to one segment at a time. However, reads from two different masters can make read requests on the two slave segments simultaneously.

A PLB4 master begins a transfer by asserting its Mn_request signal with all of its transfer qualifiers, including Mn_ABus, Mn_RNW, and Mn_size. Mn_wrDBus must also be valid if the request is a write. The crossbar contains an arbiter per slave segment that arbitrates between master requests. The winning master's transfer qualifiers are multiplexed to the slave segment with PAVvalid (if the request is primary – first in the pipeline) or SAVvalid (if the request is secondary – not first in the pipeline). Most PLB4 slaves implement a single address decode input, which requires the SoC integrator to implement external address decode logic per slave. However, this provides greater flexibility in the overall address map, allowing a slave to decode a split address range, for example. Once presented on the segment with AValid, the slave responds with SI_addrAck or SI_retry. If there is a software problem, and the request is made to an address that is not decoded by any slave, the master receives PLB_MnTimeout. If the master finds that it no longer needs the request, it can cancel the transfer by

asserting Mn_abort. The transfer will be aborted by the slave unless SI_addrAck is driven a clock before the Mn_abort. Due to the complexity required in the PLB4 slave, write transfers are not permitted under the PLB4 Interoperability Specification. Any one of the previously mentioned four signals, SI_addrAck, SI_rearbitrate, PLB_MnTimeout, or Mn_abort, completes the address phase of a PLB4 transfer.

For single and line transfers, the data phase is controlled by the slave, which must respond with one or more SI_rdDacks for reads or SI_wrDacks for writes. For single transfers, the master is responsible for making additional requests to slaves that are smaller than the master. This is referred to as a “conversion cycle”. For multi-beat transfers, the slave is responsible for adjusting the number of data beats required after calculating the smaller size between the master and slave. Neither conversion cycles nor increasing the number of data beats is required if masters and slaves adhere to the PLB4 Interoperability Specification released via Power.org.

For burst transfers, the data phase is controlled by both the master and slave. The end of the burst is defined a SI_rd/wrDack sampled with PLB_rd/wrBurst inactive. The master may deassert its Mn_rdBurst earlier than indicated by the length of fixed-length burst originally requested. Early terminated fixed-length burst writes, or write underflows, are not allowed under the PLB4 Interoperability Specification. However, the slave may terminate a read or write burst early by asserting SI_rdBTerm or SI_wrBTerm. This allows PLB4 slaves with smaller data buffers to make progress on larger requests.

PLB4 offers a significant performance increase over the previously evaluated buses, by offering read and write address to data pipelining. That is, more than one address phase may be accepted before the first data phase is completed. Specifically, PLB4 supports four deep read, and two deep write data pipelining.

Data pipelining is a very effective mechanism to maintain bandwidth with slaves that have relatively high latency. Taking the DRAM example again, an initial read from a PLB4-based memory controller may take 60ns. If three subsequent reads also take 60ns, and can be requested before the read data is returned from the first read request, then the read data from all four read requests will be available in consecutive clocks, assuming a PLB4 frequency of 200MHz and reads of 3 or more beats. This means that full bus utilization can be achieved even though there is a relatively high latency to deal with.

Assuming a frequency of 200MHz, which should be achievable in a 90nm or smaller process, and the most common data bus width of 128-bits, the overall PLB4 bandwidth would be as high as 6.4 GB/s. However, that assumes full utilization of both the read and write data buses, which requires data pipelining of multi-beat transfers large enough to overcome the latency of slaves involved.

5.2 PLB4 strengths

PLB4 offers significantly more bandwidth than the buses previously evaluated. There is a cost for this performance in number of I/O required. Area consumed is dependent on the performance required by each IP core. PLB4 can achieve fairly high frequencies, but is limited by paths driven from the master/slave, through the arbiter, and to the slave/master, without being latched. PLB4 can combine its advanced features with a reasonably high frequency to meet the bandwidth requirements of mid-performance bus applications.

1. **Data Pipelining:** Buses that do not pipeline cannot achieve full bus utilization and peak bandwidth given that there will be transfer latency to and from the slaves. PLB4 supports 4 deep read and 2 deep write data pipelining. This allows for full bus utilization with typical DRAM page miss access times. For example, on a 200MHz PLB4, if memory accesses take 60ns, there will be no “holes” in the read data bus if 4 read deep data pipelining is enabled, and each read is at least 4 beats. So, even though the latency is not low, the maximum read data bandwidth of 3.2GB/s can be achieved.
2. **Separate Data Buses:** PLB4 doubles the peak bandwidth at a given frequency by using separate buses for read and write data. These buses are used in conjunction with data pipelining command to data phases to increase performance.
3. **Double Data Rate Support:** PLB4 supports a mechanism where masters and slaves that are both capable can exchange read or write burst data at twice the base PLB4 clock frequency. This is referred to as “128-bit DDR” in the PLB4 Architecture Specification. This is a somewhat limited feature, since only bursts of an even number of quadwords are supported. However, it is very useful for doubling the performance of long bursts, especially when dealing with interfaces such as PCI Express.
4. **Multiple Data Beats Per Burst Request:** PLB4 address and data phases are independent from one another, so a burst request that is accepted by a slave with `SI_addrAck` can consist of the number of data beats the master requested (or until the slave buffer is filled), with no interaction with the address or transfer qualifiers.
5. **Byte Enables:** PLB4 defines single and contiguous byte enables such that any number of bytes between 1 and 16 can be transferred in a single beat on the read or write data buses. Buses that do not support byte enables can require that multiple transfers be made for bytes inside a single quadword.

6. Native Attach: PLB4 is the native interface for all IBM PowerPC 4xx processors. There is no additional glue logic required to attach any of these CPUs to PLB4.
7. Bridges Exist: Masters on PLB4 can use the PLB4XAHB Bridge to access AHB slaves, and the same bridge can be configured to allow AHB masters to access PLB4 slaves. There are also bi-directional bridges available to allow PLB4 masters and slaves to communicate with slaves and masters on OPB or slaves and masters on another PLB4.
8. Delayed Reads: Some PLB4 slaves with significant latency have been designed to accept reads with SI_rearbitrate instead of SI_addrAck. This frees up the read data bus for other PLB4 traffic while the long latency read occurs. Once the read data is stored in the slave, the next time the read request is placed on the bus, it is accepted with SI_addrAck, and the read data is provided immediately. This mechanism is not explicitly stated in the specification, but is allowed under the PLB4 architecture. PLB4 masters may assert Mn_abort when the arbiter has granted the bus to another request, so slaves that implement delayed reads must monitor the individual Mn_abort signals along with Mn_UABus, Mn_ABus, and Mn_RNW; not just PLB_abort. Delayed reads allows other slaves with shorter latencies access to the rdDBus, and make progress while the long latency delay read is being performed.
9. “Cut Through” Writes: When a bridge from PLB4 to another bus begins to receive write data from a fixed-length burst request, it can make a request on the other side of the bridge. Note that this is not guaranteed under the PLB4 Architecture, but is when PLB4 masters adhere to the PLB4 Interoperability Specification, which does not permit early termination (by the master) of fixed-length burst writes.
10. Slave Burst Termination: Slaves are allowed to terminate a read/write variable or fixed-length burst by asserting SI_rd/wrBTerm. This allows slaves to accept (with SI_addrAck) bursts, but then discontinue them when the slave fills its data buffer. This enables slaves to be designed with smaller data buffers to save area. Slave burst termination is only required in buses that have either open-ended burst or long fixed-length burst requests, and PLB4 defines fixed-length bursts up to 256 beats.
11. Availability of Core IP: There are more than 60 PLB4 cores available from at least two different companies. The cores cover a range of mid-performance bandwidth applications, and include cores such as Gigabit Ethernet, PCI Express, and DDR2 memory controllers.

12. **Verification Support:** A complete package of PLB4 verification IP is available. The package includes master and slave BFMs, protocol checkers, and monitors that can be run in a constrained random environment with functional coverage. The only drawback is that the verification IP is only available from one supplier.
13. **Timing Specifications:** PLB4 defines timing requirements for all of the inputs and outputs defined. The timing is specified as a percentage of the clock cycle with respect to the rising edge of the PLB4 clock.
14. **Data Bus Width:** PLB4 is defined with a choice of 32, 64, and 128-bit data bus widths. The most common implementation of 128-bit has become the de facto standard, and the one specified in the PLB4 Interoperability Specification.
15. **Error Reporting:** PLB4 defines errors in the address and data phases of transfers. A PLB_MnTimeout will occur during the address phase if a slave does not respond to a request in time. There is no data phase in this case. Slaves may assert an error per data beat on reads/writes with SI_rd/wrErr, and latent write errors, or other errors detected by the slave cause SI_MIRQ to assert.
16. **Command Attributes:** PLB4 masters may present different attributes with each request to inform the slave or get the slave to behave the proper way.
17. **Address Space:** Interfaces such as PCI Express define a 64-bit address space, and PLB4 is architected as 64-bit.
18. **IP Core Portability:** Many of the IP cores designed with a PLB4 interface are available as “synthesizable” cores. The source Verilog or VHDL is provided and synthesized into the target technology.

5.3 PLB4 weaknesses

PLB4 can achieve the bandwidth consistent with mid- and high-performance buses through data pipelining, separate 128-bit read and write data buses, and double data-rate burst data, but the performance is paid for with the price of complexity. PLB4 is still lacking in a few key features available in high-performance buses.

1. **Limited Frequency:** Due to the PLB4 protocol definition, there are timing paths that go from a driving latch in the slave, through combinatorial steering logic in the PLB4 arbiter, and to a receiving latch in the master. These paths limit the frequency of a PLB4

implementation to about 200MHz in 90nm technology. This can be pushed to around 266MHz with careful placement in physical design.

2. No Out of Order Read Data. For PLB4, the order of the SI_addrAck's on read requests dictates the order of returning read data. Slaves that accept reads with PLB_SAVValid must wait for the arbiter to assert PLB_rdPrim before sending their read data. Unfortunately, this allows a slave with long latency that happens to assert SI_addrAck first to delay the read data from a slave with short latency that asserts SI_addrAck after the first.
3. No Command Pipelining. A PLB4 master request must be held active until acknowledged by the slave accepting the transfer. Under the three-cycle acknowledge timing guidelines supported by the PLB4 Interoperability Specification, one master request can be made every three clocks.
4. Pipelined Read Burst Complexity: As repeatedly mentioned before, data pipelining is an important, performance-enhancing feature. However, pipelining bursts, particularly read bursts, can complicate the design and verification of PLB4 IP. The combination of early and delayed rdComp, rdBurst inactive with rdDack defining the end of the burst, and the possibility of bursts of one complicate the logic required to support pipelining read bursts.
5. Slave Burst Termination: This was previously listed as a strength of PLB4, since it allows slaves to be designed with smaller buffers without tying up the data bus; inserting wait states while the buffer is being refilled. However, masters are burdened with additional complexity in dealing with read and write bursts that may be terminated early, sometimes repeatedly. Masters that pipeline bursts must deal with even more complexity, as they must determine where to make the "repeat" request for the slave burst terminated transfer with respect to new requests.
6. No Unaligned Burst Support: PLB4 bursts must begin and end on an address boundary that matches the master data bus width. So, 128-bit master must request quadword bursts. For transfers not aligned on quadword boundaries, this means that either multiple requests must be made. For example, on a 128-bit PLB4, a burst write of 64 bytes beginning at an address of x5 would require 3 transfers; a single write, followed by a fixed-length quadword burst of 3, followed by another single write. Given different protocol rules, this could be accomplished in a single transfer.

7. **Data Bus Width Complexity:** PLB4 master and slave designs that support the full architecture must support transfers of 32, 64, and 128-bits. Masters that support smaller slaves must contain conversion cycle logic that makes an additional request after an initial single request. Slaves that support smaller masters must contain logic that multiplies the number of data beats on multi-beat line and burst reads. Fortunately, the originally de facto standard of 64-bit and 128-bit masters and 128-bit slaves, is now officially specified in the PLB4 Interoperability Specification.
8. **Missing IP:** While there is a very good selection of IP available on PLB4, there are a few key pieces of IP that are unavailable that would fit within its performance range, i.e. USB 2.0 and SATA controllers.
9. **Master Abort Complexity:** PLB4 allows masters to assert the Mn_abort up to and including the clock in which PLB_MnAddrAck is asserted. This creates complications in the slave, such as when a bridge must terminate a transaction on another interface based on the incoming PLB_abort. Masters that adhere to the PLB Interoperability Specification are not allowed to abort write transfers.
10. **Limited Pipeline Depth:** PLB4 is capable of data pipelining 4 deep reads and 2 deep writes. This provides an excellent performance increase when interfacing to interfaces with a moderate amount of latency, such as DDR2/3. However, 4 reads and 2 writes becomes a performance limitation for interfaces with even larger latencies, such as PCI Express.

5.4 PLB4 Core IP

There is a substantial library of IP cores available on PLB4. Most are mid- to high-performance, and many are synthesizable. A handful would be a more appropriate fit on a low-performance bus, i.e. UARTs and GPIO. The following table was compiled using information gathered from both <http://www.design-reuse.com>, and from IBM's IP Development Library, but does not reflect a complete list of PLB4 core IP.

No. of Cores	Type	No. of Suppliers
3	PLB Arbiter	2
1	PLB4 Crossbar Arbiter	1
1	PLB4 to PLB4 Bridge	1
1	PLB4 to AHB Bridge	1
2	PLB4 to OPB Bridge	2
2	OPB to PLB4 Bridge	2
1	AHB to PLB Master Gasket	1
2	Generic Interface to/from PLB	2
4	Microprocessor	1
1	L2 Cache Controller	1
1	DMA Controller	1
2	10/100/1G Ethernet Controller	2
1	10/100/1G/10G Ethernet Controller	1
1	ATM with Utopia Level 2/3 Interface	1
2	Bus Analyzer/Performance Monitor	2
1	IDE Controller	5
1	SDRAM Controller	1
6	DDR Memory Controller	3
1	Mobile DDR Memory Controller	1
5	DDR2 Memory Controller	3
1	DDR3 Memory Controller	1
1	GDDR3 Memory Controller	1
1	SRAM Controller	1

No. of Cores	Type	No. of Suppliers
1	NAND Flash Controller	1
1	BRAM Controller	1
1	Security Processor	1
3	PCI Controller	2
1	PCI-X Controller	1
2	PLB4 to PCI Express Bridge	1
1	I2O Messaging	1
1	16450 UART	1
1	16550 UART	1
1	GPIO	1

5.5 PLB4 Verification IP

There is only one choice in verification IP for PLB4. Fortunately, the sole supplier provides bus functional models, monitors/checkers, and functional coverage/scoreboards as a complete, packaged solution. Although the environment is PureSpec, there is support for test benches and test cases written in 'e', C/C++, SystemC, SystemVerilog, Verilog, VHDL, PureSpec, or OpenVera.

5.6 PLB4 conclusion

PLB4 surpasses the buses previously evaluated in performance, largely due to its support of data pipelining with separate read and write data buses. Its data bandwidth can also be doubled if both master and slave support the 128-bit DDR mode. PLB4 is the native attach point for all IBM PowerPC 4xx processors currently available, so there is no glue logic required which may cost a cycle or two of latency. PLB4 splits the address and data phases, and supports fixed-length bursts of multiple data beats from a single request.

PLB4 has a solid library of IP, but there are a few key cores missing, such as USB and SATA controllers. There is a complete package of verification IP available for PLB4, although it is single-sourced. The technical limitations of PLB4 include the lack of support for out-of-order read data and combinatorial logic in the arbiter restricting the maximum achievable frequency.

6 AMBA - AXI

6.1 AXI description

AXI is the high-performance bus in the AMBA family. The architecture defines three write channels and two read channels. The write channels are address, write data, and response. The read channels are address and read data. The address channels include 32-bit address buses, AWADDR and ARADDR, but this could be extended in some implementations. The write and read data buses (WDATA and RDATA) may be defined under the specification as any 2^n number, from 8-bit to 1024-bit. With the assumption that both the address and data buses are 32-bit, and that the data buses are 128-bit, the write address, write data, and write response channels would require 56, 139, and 8 I/O, respectively. The read address and read data channels would require 56 and 137 I/O, respectively. Thus, each 128-bit AXI master has 396 I/O total.

AXI masters and slaves are connected together through a central interconnect, which routes master requests and write data to the proper slave, and returning read data to the requesting master. The interconnect also maintains ordering based on tags if, for example, a single master pipelines read requests to different slaves.

AXI uses a handshake between VALID and READY signals. VALID is driven by the source, and READY is driven by the destination. Transfer of information, either address and control or data, occurs when both VALID and READY are sampled high.

An AXI master begins a read transfer by driving an address, ARADDR, and other transfer qualifiers with ARVALID. In high frequency implementations, the interconnect latches and drives the same signals to the slave, which responds with ARREADY. The slave drives read data, RDATA, with RVALID, and the transfer is made when RVALID and RREADY are sampled active. Again, in high frequency implementations, it is common for the interconnect to latch and drive the read data back to the requesting master. The last beat of read data is driven with RLAST.

For a write transfer, the AXI master begins by driving an address, AWADDR, and other transfer qualifiers with AWVALID. An interconnect latches and drives the same signals to the slave in high frequency implementations, and the slave responds with AWREADY. When WVALID is active, the first beat of write data is valid. WVALID may be driven with valid data even before or after the address that relates to it. The transfer is made when WVALID and WREADY are sampled active. The AXI master drives the last beat of write data with WLAST. The slave

then drives a write response, BRESP, with BVALID back to the master to indicate when the write transfer is complete, along with any applicable error information.

Like PLB4, AXI splits the address and data phases, so data pipelining is supported. Thus, AXI can achieve full bus utilization, even when slaves with relatively high latency are attached.

Assuming a frequency of 400MHz, which should be achievable in a 90nm or smaller process, and a read and write data bus width of 128-bits, the overall AXI bandwidth would be as high as 12.8 GB/s. Since AXI supports data pipelining, there is a much greater chance of achieving this peak bandwidth in a practical application. At the same frequency, an AXI configured to 64-bit would yield a bandwidth of 6.4 GB/s.

6.2 AXI strengths

AXI offers several features that allow for high performance. However, achieving the highest performance at a system level requires that all of the AXI masters and slaves are designed to a high performance target, since, for example, a simple slave that does not support out-of-order read data can delay all the other slaves than can. Thus, system integrators should not expect high performance simply because AXI is selected as the bus; the performance achieved is dependent on all the AXI cores design points. AXI can currently achieve higher frequencies than PLB4, since the existing PLB4 implementation has flow-through paths from masters to slaves. AXI can use the higher frequency along with a data bus that can be defined to 128-bit and above to achieve the bandwidth required for mid- and some high-performance bus applications.

1. **Data Pipelining:** Buses that do not pipeline cannot achieve full bus utilization and peak bandwidth given that there will be transfer latency to and from the slaves. AXI supports 16 deep read and 16 deep write data pipelining. This allows for full bus utilization, even with interfaces with extremely long latency, such as PCI express. For example, on a 400MHz AXI, if PCI express accesses take 180ns, there will be no “holes” in the read data bus if the PCIe slave accepts 12 read requests, assuming each read is at least 8 beats. So, even though the latency is large, the maximum (assuming 64-bit) read data bandwidth of 3.2GB/s can be achieved over time.
2. **Multiple Data Beats Per Burst Request:** AXI address and data phases are independent from one another, so a burst request that is accepted by a slave when READY and VALID are active must consist of the number of data beats the master requested, with no further interaction with the address and control signals. The driver of the burst data also sends a LAST signal to mark the last beat of burst data.

3. **Write Strobes:** AXI defines write strobes to enable the transfer of some or all of the bytes on the write data bus in a single beat. Given a 128-bit AXI, this allows any number of bytes between 1 and 16 to be transferred in a single beat on the write data bus. Buses that do not support write strobes can require that multiple transfers be made for bytes inside a single quadword. Having write strobes also enables AXI masters to limit propagation of write data with errors. If the master discovers there are errors in write data that has yet to be sent to AXI, it is allowed to complete the burst from a data beat standpoint, but without enabling any of the write strobes.
4. **Out-of-order Reads:** AXI uses the ARID[3:0] bus on the read address channel to allow slaves to return read data out-of-order with respect to the order of the read requests. Reads made with the same ARID[3:0] must remain in order, but reads made with different ARID[3:0] may be returned out-of-order. This improves read data channel utilization by allowing slaves with shorter latencies to return data as soon as it is available, rather than forcing them to return in the request order, potentially behind a slave with much longer read latency. This is a better mechanism than performing delayed reads, since they require communication between the master and slave once the read data is available, and then a subsequent repeat of the read request from the master. Simple slaves may not choose to implement out-of-order read data, so system integrators must be careful when selecting IP to be placed on the same AXI interconnect. Simple slaves with long latency will negate the benefit of the other slaves performing reads out-of-order, since the interconnect must guarantee that reads with the same ARID from different slaves return read data in the same order in which the requests were made.
5. **Write Data Interleaving:** Multiple AXI masters may attempt to write data to a single slave at the same time. Some masters buffer all of the write data before making the request, but others assemble data to send after the request is made. AXI slaves that support write data interleaving can accept write data from both types of masters in what looks like a single data tenure, with the AWID value switching between the masters sending write data. This allows the interconnect to avoid stalling the write data bus to the slave, waiting for write data to be assembled. Instead, the interconnect sends write data from a “buffered write” master in between beats of write data from the “assemble write” master.
6. **Separate Data Buses:** AXI doubles the peak bandwidth at a given frequency by using separate buses for read and write data. These buses are used in conjunction with data pipelining to increase performance.

7. **Handshaking:** The VALID and READY handshaking mechanism in AXI allows both masters and slaves to control the flow of data. This can be problematic if, for example, a poorly designed master makes a write request while still assembling write data that is coming in from a slower or narrower interface. Hopefully, that is an unlikely design point, but something SoC designers need to be aware of when integrating AXI IP. One real benefit of having the handshake mechanism is that slaves with long latencies can accept more requests than they have buffer space for. So, for example, an AXI PCIe slave may queue eight reads, but only have enough buffer space for four. The slave can do this if the masters are guaranteed to accept the read data when the slave drives it back. This is an area and cost savings, but requires knowledge of specific AXI master capabilities, and should therefore be implemented as a configuration option in a general purpose AXI slave.
8. **Exclusive Access:** AXI supports an exclusive access mechanism that enables semaphore types of operations without requiring the bus to be locked. The process is 1) a master requests an exclusive read from an address location, 2) the same master, some time later, attempts to complete the exclusive operation by attempting an exclusive write to the same address. The exclusive write is only able to complete successfully if no other master has written to that location between the exclusive read and write. This is very similar to the lwarx and stwcx mechanism described in the Power ISA™ document, and this mechanism could be used to improve the performance of Power processors attached to AXI.
9. **No Slave Burst Termination:** Once an AXI slave acknowledges a burst transfer, it is responsible for accepting all of the write data or generating all the read data associated with that burst. This simplifies master designs, since the master does not have to prepare to make a subsequent request if the slave terminated the original request before all of its data was transferred. Giving slaves the ability to terminate bursts is not required when the burst length is declared with the request, and is reasonably short. AXI bursts are 16 beats or fewer.
10. **Excellent Verification Support:** AXI offers a large selection of verification IP from several different suppliers. The solutions offered support several different languages and run in a choice of environments.
11. **Flexible Complexity:** AXI masters and slaves are more difficult to design than AHB masters and slaves. However, there are several optional features that may be left unimplemented in order to reduce the increase in complexity. and may be somewhat offset with the availability of more sophisticated verification support

12. **Data Bus Width:** AXI is defined with a choice of several bus widths, in 2^n increments from 8-bit to 1024-bit. The ability to select the data bus width at the time of synthesis improves the scalability (in peak bandwidth) of the core IP. However, supporting even a few bus widths does add to the complexity of the design, and choices need to interlock with other core IP to maintain interoperability.
13. **Error Reporting:** AXI defines errors in the read and write response channels. The slave can respond with SLVERR if it detects a problem, and the interconnect can respond with DECERR if no slave accepts the transfer. Interrupts from masters are handled outside the AXI architecture.
14. **Command Attributes:** AXI masters may present different attributes with each request to inform the slave or get the slave to behave the proper way.
15. **IP Core Portability:** The vast majority of the IP cores designed with an AXI interface are available as “synthesizable” cores. The source Verilog or VHDL is provided and synthesized into the target technology.

6.3 AXI weaknesses

AXI offers most of the features desired in a mid- to high-performance bus; high bandwidth, multiple data beats per burst request, excellent bus utilization through data pipelining, out-of-order reads, and write interleaving. However, there are features and IP missing that do not make AXI the best fit for Power-based SoCs.

1. **Independent Address Channels:** Having separate read and write address channels seems like a good idea that will simplify designs and increase performance, but problems arise when master ordering requirements are not met. For example, if a master intends to perform writes followed by reads when a write and a read address channel are used, there must be cross-synchronization logic that ensures that the read requests are not made until the writes are complete.
2. **No Native Power Attachment, and No Gasket:** Power processors currently available attach directly, or through a small gasket, to PLB4 or AHB. There is currently no gasket available to interface a Power processor to AXI. This is not an inherent bus limitation, but a matter of development investment that must be made if Power processors are going to communicate directly with IP on AXI. Having a gasket to connect Power processors would enable all of the high performance IP to reside together on AXI.

3. **Lack of Bridges:** AHB is currently the only AMBA bus connected to CoreConnect, both through OPB and PLB4. If the Power processor is connected to PLB4, this places a performance limit on the traffic going between the CPU and AXI. Unless a PLB4-to-AXI Bridge is developed, traffic would have to travel from PLB4 to AHB, and from AHB to AXI. Having a bi-directional bridge between AXI and PLB4 would allow mid-to high-performance PLB4 core IP to work with high-performance AXI IP. However, it would likely not take the place of a gasket specifically made to interface Power processors to AXI, since the gasket could be optimized to be as small as possible with low latency.
4. **Inefficient Unaligned Transfers:** AXI specifies the number of bytes transferred per data beat to a 2^n number. So, given a 128-bit bus, AXI defines transfer sizes of 1, 2, 4, 8, and 16 bytes. For unaligned transfers, this means that either multiple requests must be made, or that a multi-beat burst of a smaller size must be requested. For example, on a 128-bit AXI, a burst write of four beats, with one byte per beat, could be requested to store data for four bytes beginning at an address of $x1$. The same four bytes could also be written using three requests; a byte to $x1$, then two bytes to $x2$, and another byte to $x4$. Given different protocol rules, this could be accomplished in a single transfer.
5. **Timing Specifications:** AXI does not define timing parameters for and of the interface signals, not even just a percentage of the clock cycle. The specification only states that there must not be any combinatorial paths between inputs and outputs on any AXI master or slave. So, SoC developers must integrate AXI cores and run static timing analysis at the chip level to judge how compatible AXI masters and slaves will be with one another.
6. **Data Bus Width Complexity:** Like AHB, an AXI slave design must support several transfers of several widths, since the architecture allows masters to request transfers of any of a number of sizes, up to and including the size of the bus. Additional steering and muxing logic is required in the slave design to handle this. There is further complication due to the fact that the data bus width itself is configurable, and any legal sizes up to the configured size must be supported. The buffer size must also increase when the data bus width increases, since the slave must support a minimum of 16 data beats worth of data, unless software ensures maximum transfers are not requested from slaves that cannot handle them.
7. **No “Cut Through” Writes:** When a bridge from AXI to another bus begins to receive write data from a request that specified a number of beats, it may have to wait until all of the requested beats of write data

are received before making a request on the other side of the bridge. This is because AXI allows masters to withhold write data by leaving WVALID inactive. If the bus on the other side of the bridge does not allow masters to insert wait states in between write data beats, the bridge must wait until the AXI write is complete. This results in a longer latency on these write transfers.

8. **Central Address Decode:** There is a wide range of applications and requirements for Power-based SoCs, and there is a corresponding wide range of address map requirements. Having the address decodes for all AXI slaves reside within the interconnect means having to support the most complex split address ranges, even for the simplest of slaves. The number, size, and granularity of address ranges can also be made configurable, but this also adds complexity to the interconnect. Address range decodes are more appropriately placed in the slaves themselves, so that multiple split address ranges are only implemented in slaves that require such complexity.
9. **Lack of Parity:** While it is more than likely that there are AXI implementations that have added parity, there is no substitute for including this information in the specification for interoperability. Left undefined, AXI masters may check read data for odd parity, while AXI slaves generate even parity.
10. **Address Space:** Interfaces such as PCI Express define a 64-bit address space. Although some implementations have logically extended the bus to 64-bit when required, the AXI architecture only specifies 32-bit.

6.4 AXI Core IP

Considering that it has been available for just about four years, there is a fairly impressive library of IP cores available on AXI. Most of the cores are geared for high performance, considering the interfaces they connect to, such as PCI Express and DDR2/3. There are a few that seem out of place, based on their performance requirements, such as SRAM and NAND Flash controllers. Most are synthesizable, offering configurable data bus widths, so applications that do not require as much bandwidth can save area by using a narrower bus. The following table was compiled using information gathered from <http://www.design-reuse.com>, but it does not reflect a complete list of AXI core IP.

No. of Cores	Type	No. of Suppliers
1	AXI Interconnect	1
1	AXI Register Slice	1
1	AXI to AXI Bridge	1
1	AHB Master to AXI Master Gasket	1
1	AXI to AHB Bridge	1
1	AXI to APB Bridge	1
1	Generic AXI Master Module	1
2	Generic AXI Slave Module	2
3	PCI Express Controller Gen 1	2
5	PCI Express Controller Gen 2	5
1	L2 Cache Controller	1
3	SRAM Controller	1
1	DRAM Controller	1
1	SDRAM Controller	1
4	DDR1 Controller	3
1	Mobile DDR Controller	1
3	DDR2 Controller	2
2	DDR3 Controller	2
1	GDDR3 Controller	1
2	NAND Controller	1
1	OneNAND Flash Controller	1

6.5 AXI Verification IP

There are many choices in verification bus functional models, monitors/checkers, assertions, and coverage/scoreboards available for AXI. The following was compiled using information gathered from <http://www.design-reuse.com>, but does not reflect a complete list of verification IP available for AXI.

AXI has an impressive amount of verification support available in the industry. There are at least four master and slave BFM's accompanied by monitors available from four different suppliers. There are at least three interconnect BFM's

and three scoreboards available from three different vendors. Languages supported include 'e', SystemVerilog, Verilog, VHDL, PureSpec, OpenVera, SystemC, and C/C++. There are also two formal verification suites available from two suppliers.

6.6 AXI conclusion

AXI is a high-performance bus. The data pipelining, multiple data beats per burst request, separate data channels, out-of-order read, and write data interleaving features combine to ensure that the AXI read and write data buses are fully utilized. Given that 90nm AXI implementations can likely achieve frequencies over 400MHz, the bandwidth available for a 128-bit AXI implementation is 12.8GB/s. AXI does not have the number or variety of IP cores available on AHB, however it is likely that only high performance cores will be designed to AXI, and that low to mid-performance legacy AHB cores will be accessed from AXI through a bridge. AXI is supported by a wide range of verification IP that supports many different languages and environments.

The limitation of AXI for Power Architecture users is the absence of licensable Power processors with a native AXI interface, and the lack of a gasket to interface the licensable IBM PowerPC 4xx cores to AXI. There is also no bridge available from CoreConnect to AXI; only to AHB. However, single Power processors with native AXI attachment have been designed and sold primarily within SoCs.

7 Open Core Protocol (OCP)

7.1 OCP description

OCP is a highly configurable interface specification defined by the Open Core Protocol International Partnership (OCP-IP). The architecture defines a point-to-point interface between a master and a slave. Cores that act as both a master and a slave use two separate interfaces. The dataflow interface is comprised of basic (address, command, and data) signals with simple (byte enables and transfer info), burst, tag, thread, sideband, and test extensions. The address, read data, and write data buses are all defined with configurable widths, and many of the control signals are optional. Only clock and a 3-bit command field are required master signals, so assuming 32-bit address, write data, and read data buses, the smallest OCP master interface requires only 100 I/O. Given a 64-bit address bus, with 128-bit read and write data buses, the most sophisticated OCP interface can contain more than 460 I/O.

OCP masters and slaves are often connected together through a central interconnect. Each master connects to a slave port on the interconnect, and each slave connects to a master port on the interconnect. The interconnect arbitrates between master requests intended for the same slave, and between returning slave data intended for the same master.

An OCP master begins a read transfer by driving an address, MAddr, with a command, MCmd. The slave accepts the command with SCmdAccept, and then drives read data, SData, and SResp with each data beat. The last beat of a read burst may be driven with SRespLast.

For a write transfer, the OCP master again begins by driving an address, MAddr, with a command, MCmd. The first beat of write data may be driven with the command if the reqdata_together parameter is set to '1', or it may be driven later with MDataValid. A simple write transfer is made the slave accepts the command with SCmdAccept. A data handshake write transfer is made with MDataValid and SDataAccept are both sampled active. The OCP master may drive the last beat of write data with MDataLast.

Similar to PLB4 and AXI, OCP can be configured to split the address and data phases, so that data pipelining is supported. Thus, when OCP masters and slaves are configured to high-performance, OCP can achieve full bus utilization, even when slaves with relatively high latency are attached.

Assuming a frequency of 400MHz, which should be achievable in a 90nm or smaller process, and a data bus width of 128-bits, the overall OCP bandwidth

would be as high as 12.8 GB/s. Since OCP supports data pipelining, there is a much greater chance of achieving this peak bandwidth in a practical application.

7.2 OCP strengths

OCP is an open standard that spans almost the entire performance range, since the basic interface may be used for low performance cores, and multiple extensions may be used to design high performance cores. Due to this flexibility, OCP is best suited in environments where all of the core IP designs are controlled “in house” by a single SoC architect.

1. **Open Standard:** OCP is an openly licensed standard. Members of OCP-IP actively participate in driving new features into the specification, while maintaining backward compatibility with existing OCP cores. Large companies that control all of the core IP designs for SoCs have successfully used OCP as a way to avoid changes in other bus standards.
2. **Threads:** OCP defines optional thread extensions to enable independent concurrent transfer sequences. There are no ordering rules for transfers contained within different threads, so independent flow control is used for each thread. This allows greater concurrency, but the implementation requires independent buffering for each thread.
3. **Timing Parameters:** The OCP Specification does a very good job of defining timing parameters for signal groups. Parameters such as `setuptime`, `c2qtime`, `holdtime`, `drivingcellpin`, `loadcellpin`, and `wireloaddelay` are all defined. A Core Synthesis Configuration File is also described, with `clock`, `area`, `port`, `max delay` (for combinational paths), and `false path` sections defined. Three different timing categories are also specified; `no timing`, `conservative`, and `high performance`. Each of these places a different amount of responsibility for timing on the core developer, and allows SoC integrators to make high-level decisions on the timing compatibility between OCP interfaces. The only problem is that the timing parameters and corresponding file are recommended, but not required.
4. **Verification Support:** OCP offers a large selection of verification IP from several different suppliers. The solutions offered support several different languages and run in a choice of environments. At least part of the reason for this support is because of the Verification Guidelines in the OCP Specification. This section provides direction for writing functional coverage checks. Enabling

parameters for X and Z checks and for illegal switching checks are listed. Various other checks for request, datahandshake, and response phases, and checks between phases are defined.

5. Core Performance Report: The OCP Specification provides a template for a Core Performance Report that is intended to provide SoC architects the information they need to select cores with the desired performance. Master information includes the issue rate, data pipelining support, burst support, and flow control. Slave information includes latency, throughput, data pipelining support, and burst support. These types of reports would be very useful to system integrators, but the report is recommended, not required, for OCP core developers.
6. Configurable Data Pipelining: Buses that do not pipeline cannot achieve full bus utilization and peak bandwidth given that there will be transfer latency to and from the slaves. OCP supports a configurable pipeline depth. Full bus utilization can be achieved if the OCP master and slave can be configured to support a large enough depth, even with interfaces with extremely long latency, such as PCI express.
7. Configurable Multiple Data Beats Per Burst Request: OCP address and data phases can be configured to be independent from one another, so a burst request that is accepted by a slave can be configured to consist of the number of data beats the master requested, with no further interaction with the address and control signals. The driver of the burst data also sends a Last signal to mark the last beat of burst data. OCP defines a variety of bursts, each serving a unique application. Incrementing and wrapping are defined as in other buses, but OCP also offers exclusive-or and streaming modes. Finally, OCP defines an optional signal, MReqLast, used to indicate the last request of a large burst that has been broken into multiple burst requests of smaller lengths.
8. Configurable Byte Enables: OCP defines byte enables to allow the transfer of some or all of the bytes on the read (MByteEn) or write (MDataByteEn) data bus in a single beat. Given a 128-bit OCP, this allows any number of bytes between 1 and 16 to be transferred in a single beat on the data bus. Buses that do not support byte enables can require that multiple transfers be made for bytes inside a single quadword.
9. Configurable Out-of-order Data: OCP defines optional tag extensions to allow slaves to return read data or commit write data out-of-order with respect to the order of the requests. Requests

made with MTagInOrder asserted must remain in order. This improves performance by allowing slaves to return data as soon as it is available, rather than forcing them to return in the request order, which may involve holding read data while a longer latency read takes place. Responses to requests that target overlapping addresses (determined by MAddrSpace, MAddr, and MByteEn) are never returned out-of-order with respect to one another. Simple slaves may not choose to implement out-of-order read data, so system integrators must be careful to make sure that the OCP interconnect will not delay higher performing slave read data when one of these simple slaves is attached.

10. Separate Data Buses: OCP doubles the peak bandwidth at a given frequency by using separate buses for read and write data. These buses are used in conjunction with pipelining command to data phases to increase performance.
11. Configurable Handshaking: OCP defines an optional mechanism to allow both masters and slaves to control the flow of data. SoC designers need to be aware of how this control flow is implemented in masters and slaves, as performance can suffer if, for example, a poorly designed master makes a write request when it will still take a long time to assemble write data that is coming in from a slower or narrower interface.
12. Lazy Synchronization: OCP supports an exclusive access mechanism that enables semaphore types of operations without requiring the bus to be locked. The process is 1) a master requests a ReadLinked, RDL, from an address location, 2) the same master, some time later, attempts to complete the exclusive operation by attempting a WriteConditional, WRC, to the same address. The exclusive write is only able to complete successfully (by the interconnect or the memory controller) if no other master has written to that location between the exclusive read and write. This is nearly identical to the lwarx and stwcx mechanism described in the Power ISA document, and this mechanism could be used to improve the performance of Power processors attached to OCP.
13. Configurable Data Bus Width: OCP is defined with configurable data bus widths to allow area savings for low performance applications, and more bandwidth in high performance applications.
14. Error Reporting: OCP defines errors both with and without corresponding responses.

15. Command Attributes: OCP masters may present different attributes with each request to inform the slave or get the slave to behave the proper way.
16. Address Space: Interfaces such as PCI Express define a 64-bit address space, and OCP may be configured to be any address width.
17. IP Core Portability: Most IP cores designed with an OCP interface are available as “synthesizable” cores. The source Verilog or VHDL is provided and synthesized into the target technology.

7.3 OCP weaknesses

OCP offers almost all of the features desired for low, mid, and high-performance cores. However, because the interface is so configurable, and defines so many optional features, the Bus Architecture TSC cannot recommend OCP for new core designs, where the IP is checked into a library for general use. However, the TSC recognizes that OCP has been used successfully in many cases where core IP is developed to meet a particular design point, defined with a specific application in mind. Thus, there are many examples of OCP core IP that are proven in their target application, and should be re-used in similar applications.

1. Interoperability: OCP offers an almost overwhelming amount of flexibility and configurability in the interface definition. There are at least 80 configuration parameters defined in the OCP Specification; 20 protocol, 3 phase, 40 dataflow, and 17 sideband. On the low performance end, OCP can be defined to resemble APB, best suited for very simple cores with low bandwidth requirements. In the mid-performance range, OCP can be configured to look like AHB, with a command for each corresponding beat of data. OCP can also be configured to a superset of AXI, with the support of threads, tags, and single request per multiple beats of data.

At first glance, this configurability seems to offer one interface for cores to design to. However, it also allows masters and slaves to be designed to adhere to the specification, but then not work with one another. In theory, masters and slaves can be designed to work with any counterpart. Many of the optional features can be placed behind synthesis parameters, so the area of the synthesized core would correspond to the features supported. However, this greatly complicates the design and verification of the cores, increasing time to market. In practice, core designers select the features appropriate for the required amount of performance, and allow the interconnect to

handle translation to other cores with lower or higher performance. However, this also complicates the interconnect.

OCP defines master and slave configuration files to assist with interoperability problems. However, these are most useful in quickly determining that a master and slave are incompatible. There is no easy way of making them work together; either the master or slave design point must change to work with the other.

For this reason, OCP is best suited for use in an environment where all master and slave core designs are tightly controlled to ensure their interoperability.

2. **No Native Power Attachment, and No Gasket:** PLB4 and AHB are the native interfaces for Power processors currently available. At the time of this report, there is no gasket available to interface a Power processor to OCP. This is not an inherent OCP limitation, but a matter of development investment that must be made if Power processors are going to connect to IP on OCP. The TSC recognizes the ability of some OCP-based interconnects to attach directly to AXI and AHB masters, including some Power processors. However, having a PLB4 port on an OCP interconnect, in addition to having AHB or AXI, would enable customers that have invested in developing cores for OCP to use any of the Power processors currently available.
3. **Unaligned Transfers:** OCP specifies the number of bytes transferred per data beat to the value driven on (assuming they are both configured) MByteEn for reads, and on MDataByteEn for writes. For unaligned transfers, this means that either multiple requests must be made, or that a multi-beat burst of a smaller size must be requested. For example, on a 128-bit OCP, a burst write of 44 bytes beginning at an address of x9 would require 3 transfers. Given different protocol rules, this could be accomplished in a single transfer.
4. **Burst Complexity:** OCP offers a wide range of configurability in general, but especially in the types of bursts supported. MBurstSingleReq forces bursts into an AHB-like mode when '0', where there is one request per each data beat. When MBurstSingleReq is '1', bursts closely resemble AXI where there can be multiple data beats per each request. MBurstPrecise controls whether the bursts are fixed length or not, and MDataValid and optionally SDataAccept are used when datahandshake is set to '1' Designing and verifying masters and slaves that are configurable enough to handle all of the possibilities takes more time and effort.

5. **No Slave Burst Termination:** OCP allows for bursts with an undefined or a configurable length, so OCP slaves must either be able to support the maximum burst size, or masters must only request transfers that slaves can handle, or wait states must be inserted by a slave with a smaller buffer. This can be complicated as it must be controlled at an SoC and software level. The maximum number of beats in a burst is configurable with the MBurstLength parameter. If slaves have buffers that cannot accept the maximum number of data beats, either software must ensure that masters do not make large requests to those slaves, or those slaves must insert wait states while the data buffer is being refilled. Slaves that support the largest burst transfer size will consume more area than slaves that are able to accept and then terminate bursts when the data buffer is filled.
6. **Lack of Bridges:** There are currently no bridges available between OCP and CoreConect PLB4 or OPB. It is possible to direct attach AMBA APB, AHB, and AXI IP directly to at least one OCP interconnect currently available. Having a bi-directional bridge between OCP and PLB4 would allow mid- to high-performance PLB4 core IP to work with high-performance OCP IP. However, it would likely not take the place of a gasket specifically made to interface Power Architecture processors to OCP, since the gasket could be optimized to be as small as possible with low latency.
7. **Central Address Decode:** There is a wide range of applications and requirements for Power-based SoCs, and there is a corresponding wide range of address map requirements. Having the address decodes for all OCP slaves reside within the interconnect means having to support the most complex split address ranges, even for the simplest of slaves. The number, size, and granularity of address ranges can also be made configurable, but this also adds complexity to the interconnect. Address range decodes are more appropriately placed in the slaves themselves, so that multiple split address ranges are only implemented in slaves that require such complexity.

7.4 OCP Core IP

There have been many cores designed with an OCP interface. Since the interface is highly configurable, the cores span the low, mid, and some of the high performance range. Some of the cores are geared for high performance, considering the interfaces they connect to, such as DDR2/3, and some are obviously geared for low performance, such as I2C controllers. Thus, to be fair, the total number of OCP cores must be compared against another family of

buses; for example against the total number of APB, AHB, and AXI cores. The total number and breadth of OCP cores, then, falls in between the total number of AMBA and CoreConnect cores. However, many OCP cores have been designed and used “in house”, and have not released for general use as licensable cores. The following table was compiled using information gathered from <http://www.ocpip.org> and <http://www.design-reuse.com>, but it does not reflect a complete list of OCP core IP.

No. of Cores	Type	No. of Suppliers
4	OCP Interconnect	1
1	AHB-to-OCP Bridge	1
1	APB-to-OCP Bridge	1
9	Slice Processors (8 and 16-bit)	1
8	Microcontroller (8-bit)	3
3	DSP Processors (16 and 24-bit)	1
1	DDR3 Controller	1
1	GDDR3 Controller	1
2	DDR2 Controller	1
2	DDR Controller	1
1	Mobile DDR Controller	1
1	SDRAM Controller	1
1	DRAM Scheduler	1
1	L2 Cache Controller	1
1	DMA Controller	1
1	Visual Effects Module	1
1	OneNAND Flash Controller	1
5	PCI	3
6	10/100 Ethernet MAC	5
3	10/100/1G Ethernet MAC	2
1	10G Ethernet MAC	1
1	USB 2.0 Host Controller	1
6	USB 2.0 Device Controller	5
2	USB 2.0 OTG	2
3	USB 1.1 Device Controller	3

No. of Cores	Type	No. of Suppliers
1	USB 1.1 OTG	1
1	SATA Controller	1
3	IDE Controller	2
3	Motion JPEG Encoder/Decoder	1
5	JPEG Still Image Encoder/Decoder	2
7	Color Space Converter	2
2	Discrete Transform (Wavelet/Cosine)	2
2	CAN Bus Controller	2
1	Xtensa PIF Interface	1
2	CRT Controller	2
1	LCD Controller	1
3	MPEG2 Audio Decoder	1
4	MPEG2 Encoder/Decoder	1
4	MPEG4 Encoder/Decoder	1
6	MPEG4/JPEG Encoder/Decoder	3
1	SDLC	1
1	Huffman Encoder/Decoder	1
6	DES/Triple DES Encrypt/Decrypt	3
11	AES Encrypt/Decrypt	2
6	SHA-1/SHA-2	3
4	MD5 Hash Function	3
6	OCB-AES	1
1	AES-CMM Controller	1
1	ARC4 Security	1
1	WLAN Security Processor	1
1	Parallel Port	1
10	UART	4
3	Timer/Counter	2
2	Peripheral Interface	2
2	Keyboard/Display Interface	2
1	Interrupt Controller	1
8	I2C Controller	7

No. of Cores	Type	No. of Suppliers
4	I2S Controller	4
1	SPI	1

7.5 OCP Verification IP

There are several choices in verification bus functional models, monitors/checkers, and assertions available for OCP. The following was compiled using information gathered from <http://www.design-reuse.com>, but does not reflect a complete list of verification IP available for OCP.

OCP has a good amount of verification support available in the industry. There are at least five master and slave BFM's accompanied by monitors available from five different suppliers. There are at least six different vendors that support functional coverage checks. Languages supported include 'e', SystemVerilog, Verilog, and VHDL.

7.6 OCP conclusion

OCP is a very flexible interface that can be configured to fit within the low, mid, or high-performance bus categories. In the low-performance range, it can be configured to be as small and as simple as APB. In the mid-performance range, OCP can be configured to single request/single data to resemble AHB, or single request/multiple data to more closely match AXI. In the high-performance range, the ability to configure data pipelining, single request/multiple data beat bursts, and out-of-order read features combine to ensure that the OCP read and write data buses are fully utilized. Given that 90nm OCP implementations can likely achieve frequencies over 400MHz, the bandwidth available for a 128-bit OCP implementation configured with high-performance features is 12.8GB/s.

OCP is lacking in numbers and variety of IP cores available for re-use. This is even more apparent considering that the OCP cores that are available span cross performance ranges, from low such as I2C to high such as DDR3. However, OCP does have a reasonably good amount of verification IP available from several suppliers that support several different languages and environments.

The biggest concern with OCP is that the enormous amount of flexibility and configurability in the specification inevitably leads to problems with interoperability. For core IP that is designed once, and checked into a library for re-use, it is likely that problems will surface, where a master designed to a certain

configuration will either not work at the intended performance level or not work at all with a slave designed to another configuration. This may be a reason why there are relatively few OCP cores available for general re-use. However, this is a problem that can be solved in a tightly controlled design environment, where an SoC architect communicates with OCP master and slave designers to ensure that the proper features are configured. There are several examples where OCP has been used successfully when all the core IP designs were done as part of constructing an SoC.

A current limitation of OCP for Power Architecture users is the absence of licensable Power processors with a native OCP interface, and the lack of a gasket to interface the licensable IBM PowerPC 4xx cores to OCP. Power processors may be used with OCP-based SoCs if such as gasket is developed or if an OCP interconnect adds support for PLB, as some have added support for AHB and AXI.

8 CoreConnect - PLB6

8.1 PLB6 description

PLB6 is the highest performance bus in the CoreConnect family. It is capable of supporting hardware-enforced cache coherency, but is optimized for a relatively small area with respect to the high bandwidth supported. PLB6 is currently under development, with a targeted availability in 2009, so the information presented here is preliminary, and subject to change.

The PLB6 architecture supports both coherent and non-coherent (I/O) devices. I/O masters attach to a master interface, while coherent processors attach to both a master and snoop interface. Coherent slaves, such as memory controllers, connect to a single slave segment, and I/O slaves connect to a slave interface on one of seven other segments. The master, snoop, and slave segment interfaces all use 64-bit address buses and separate 128-bit read and write data buses. Each non-coherent I/O master requires on the order of 450 I/O, which is slightly higher than a 128-bit AXI master, and comparable to an OCP master configured for high-performance. Each coherent master uses about an additional 250 I/O, for snooping and intervention data.

PLB6 masters and slaves are connected together through a central Bus Controller, which is responsible for routing commands and write data to the proper slave segment, and returning slave read data back to the requesting master. The Bus Controller also routes read intervention data from the snooper to the requesting master.

The basic protocol used by PLB6 begins with a master making a request with `_mM_req`. The Bus Controller is responsible for routing the request to the proper slave segment, and to the snoopers if the request is directed to coherent memory space. The slave and snoopers provide partial responses using `_sN_pval`, and the Bus Controller uses these responses to drive a combined response back to the requesting master. Slaves or snoopers can provide read data, and masters are responsible for driving write data once the combined response is received.

PLB6 is unique among the buses evaluated in that it supports both command and data pipelining. Command pipelining occurs when a master makes a second request without having received a response for the first request. The ability to support command pipelining is critical to maintaining a high level of performance in implementations with hardware-enforced cache coherency. Data pipelining is also supported to sustain high bandwidth despite having a moderate to high level of latency to memory.

Assuming a target frequency of 400MHz in a 90nm or smaller process, the point-to-point PLB6 bandwidth will reach as high as 12.8 GB/s. Since PLB6 supports both command and data pipelining, there is a much greater chance of achieving this peak bandwidth in a practical application.

8.2 PLB6 strengths

PLB6 offers the highest performance of the buses evaluated in this report. It supports the same high bandwidth as AXI and OCP, given the same 90nm process. In addition, PLB6 includes command and data pipelining, aligned and unaligned burst support, hardware-enforced cache coherency, and other features, to achieve the high level of performance required for some embedded applications.

1. **Multi-core Cache Coherency:** Some high-performance applications require more raw computing power than can be provided by a single processor. PLB6 offers the ability to connect multiple processors as masters to the same Bus Controller. All of these processors can share the same memory, and operate in a cache coherent manner.
2. **I/O Coherency:** Single processor implementations can also benefit from supporting cache coherency. I/O masters making read requests to coherent memory can 'hit' the cache, and the returning read data will be provided with low latency, since it is driven from the cache and not main memory.
3. **Intervention Capability:** PLB6 supports read data intervention for snooping masters. Intervention data is provided by a snooping master that determines that a request 'hit' its cache, when the cache line is in a state that allows intervention. PLB6 defines cache states beyond MESI to facilitate providing intervention data. This improves performance by reducing latency, as the data is sourced from a snooping master's cache rather than from main memory.
4. **Command Pipelining:** Masters on the other buses evaluated in this report must receive an acknowledge for an initial request before making a second request. PLB6 masters can make the subsequent request before receiving an acknowledge for the first. This increases the total command bandwidth, supporting more requests in the same period of time.
5. **Efficient Unaligned Burst Support:** For each burst request, the PLB6 protocol uses the starting address to define the initial data beat alignment, and the byte enables to define the final data beat alignment. So, for an unaligned burst read of 35 bytes from address x3, the byte

enables must be xFC00, and the read can be performed in a single transfer.

6. Data Pipelining: Buses that do not pipeline cannot achieve full bus utilization and peak bandwidth given that there will be transfer latency to and from the slaves. PLB6 supports 32 deep read and 32 deep write pipelining. This allows for full bus utilization, even with interfaces with extremely long latency, such as PCI Express. For example, on a 400MHz PLB6, if PCI Express accesses take 180ns, there will be no “holes” in the read data bus if the PCIe slave accepts 10 read requests, assuming each read is at least eight beats. So, even though the latency is large, the maximum read data bandwidth of 6.4GB/s can be achieved over time.
7. Native Attachment: PLB6 will be the native interface for the upcoming generation of IBM PowerPC 4xx processors. There will be no additional glue logic required to attach any of these Power processors to PLB6.
8. Out-of-order Reads: PLB6 tags master requests using `_mM_tag(0:4)` and returning slave read data using `_sN_rd_mtag(0:4)`. This allows slaves to return read data out-of-order with respect to the order of the read requests. This improves read data bus utilization by allowing slaves with shorter latencies to return data as soon as it is available, rather than forcing them to return in the request order, potentially behind a slave with much longer read latency. Simple slaves may not choose to implement out-of-order read data. However, unlike AXI, having such slaves has no impact on other PLB6 masters and slaves.
9. Out-of-order Write Data: Multiple PLB6 masters may attempt to write data to a single slave at the same time. The Bus Controller is not responsible for ensuring that the write data arrives at the slave in the order that the slave responded to the master requests. Instead, the write data is tagged using `_mM_wr_stag(0:3)` to enable the slave to associate the write data with the original request.
10. “Cut Through” Writes: PLB6 masters must be able to provide all write data indicated by the request in consecutive clocks once the slave provides the response. Knowing that write data will arrive on consecutive clocks allows bridges to other buses to make requests as soon as write data begins to arrive.
11. Multiple Data Beats Per Burst Request: PLB6 address and data phases are independent from one another, so a burst request that is accepted by a slave with `_sN_pval` can consist of the number of data

beats the master requested, with no further interaction with the address or transfer qualifiers.

12. **Byte Enables:** PLB6 defines byte enables to enable the transfer of some or all of the bytes on the read or write data buses in a single beat. Since PLB6 has 128-bit data buses, this allows any number of bytes between 1 and 16 to be transferred in a single beat. Buses that do not support byte enables sometimes force multiple transfers to be made for bytes inside a single quadword.
13. **Bridges Under Development:** PLB6 Masters will be able to access PLB4 slaves using a PLB6-to-PLB4 Bridge. Conversely, PLB4 masters will be able to access PLB6 slaves using a PLB4-to-PLB6 Bridge. Having bi-directional access to PLB4 allows the connection of PLB6 to AHB and/or OPB.
14. **Separate Data Buses:** PLB6 doubles the peak bandwidth at a given frequency by using separate buses for read and write data. These buses are used in conjunction with pipelining command to data phases to increase performance.
15. **Support for Power Architecture Atomic Update:** PLB6 supports atomic updates via the `lwarx` and `stwcx` instructions specified in the Power ISA. The process is: 1) a master performs a load using `lwarx` from an address location, and a reservation for a subsequent `stwcx` is created. 2) the same master, some time later, attempts to perform a store using the `stwcx` instruction. The `stwcx` is only allowed to complete if the reservation created by the `lwarx` has not been overwritten by another processor. PLB6 requires that the Power processor generating the `lwarx` provide this information to the bus in order to maintain cache coherency of the reservation.
16. **No Slave Burst Termination:** A PLB6 slave must only acknowledge a request for a write transfer if it has the buffer space available for accepting the write data. Thus, the smallest buffer size of a PLB6 slave is 128 bytes. If the slave bridges to another bus, a read transfer should not be acknowledged unless the slave has enough buffer space to store the read data internally, before sending the data back to the master with no wait states. The PLB6 Architecture allows the slave to pace the read data, but this is not the recommended implementation, as it does not make the most efficient use of the read data bus.

Not allowing slaves to terminate bursts early simplifies master designs, especially when dealing with pipelining. Giving slaves the ability to terminate bursts is not required when the burst length is declared with

the request, and is reasonably short. PLB6 bursts are eight beats or fewer.

17. Verification Support: There is verification IP in development from an independent supplier that will seamlessly integrate in with a choice of other buses, including AXI, PLB4, and AHB.
18. Range of Complexity: Coherent PLB6 masters are complex, and difficult to design, due to all of the verification required to cover all of the cache state transitions. However, I/O masters are much less complicated to design and verify, followed by coherent slaves, and I/O slaves. It could be argued that PLB6 I/O slaves are less difficult to design and verify than AXI and OCP slaves, since the data bus width is fixed at 128-bit, and there are very few optional features.
19. Fixed Data Bus Width: PLB6 is defined with fixed read and write data bus widths of 128-bit. This limits the ability of PLB6 to push down into the mid-performance range, but it simplifies design and verification, and improves interoperability.
20. Timing Specifications: PLB6 defines timing requirements for all of the inputs and outputs defined. To ensure high frequency designs, PLB6 defines the timing requirements to ensure that devices drive outputs off of a latch and capture inputs directly into a latch. The timing is specified as a percentage of the clock cycle with respect to the rising edge of the PLB6 clock.
21. Error Reporting: PLB6 defines errors for request responses, as well as errors for read and write data, per beat. PLB6 also defines parity for address, byte enables, and read and write data. Interrupts are handled outside the PLB6 architecture.
22. Command Attributes: PLB6 masters may present different attributes with each request to inform the slave or get the slave to behave the proper way. PLB6 defines commands and command attributes that are tightly integrated with the Power ISA.
23. Address Space: Interfaces such as PCI Express define a 64-bit address space, and PLB6 is architected with a 64-bit address bus.
24. IP Core Portability: PLB6 cores currently in development are being designed as “synthesizable” cores. The source Verilog will be provided and synthesized into the target technology.

8.3 PLB6 weaknesses

PLB6 offers all of the features desired in a high-performance bus; optional hardware-enforced cache coherency, high bandwidth, excellent bus utilization through command and data pipelining, out-of-order reads, and write data tagging. However, PLB6 is not currently available, and will be announced with a limited set of high-performance core IP.

1. Availability: PLB6 core and verification IP is currently still being developed, and will be available in 2009.
2. Missing Core IP: PLB6 will be made available with several high-performance cores, including a Power processor, bus controller, DDR3 memory controller, PCI Express controller, DMA controller, and bridges to and from PLB4. Other high-performance core IP required for some SoCs must be designed once PLB6 is made available.

8.4 PLB6 Core IP

The following PLB6 cores are currently in development, with scheduled availability for use in SoC designs in 2009.

No. of Cores	Type	No. of Suppliers
1	PLB6 Bus Controller	1
1	PLB6 to PLB4 Bridge	1
1	PLB4 to PLB6 Bridge	1
1	PPC476 Power processor	1
1	PCI Express Controller Gen 2	1
1	DDR3 Memory Controller	1
1	DMA Controller	1

8.5 PLB6 Verification IP

PLB6 verification IP is being developed in conjunction with PLB6 core IP. There will be bus functional models, monitors/checkers, and coverage/scoreboards available for PLB6.

The languages supported will include 'e', SystemVerilog, Verilog, VHDL, OpenVera, SystemC, and C/C++.

8.6 PLB6 conclusion

PLB6 is a high-performance bus that is capable of supporting single or multi-processor applications. Hardware-enforced cache coherency with multiple processors and I/O coherency with a single processor are both supported with intervention data to reduce latency. Command and data pipelining, multiple data beats per burst request, out-of-order read data, and write data tagging features combine to ensure that the separate PLB6 read and write data buses are fully utilized. Given that 90nm PLB6 implementations can likely achieve frequencies over 400MHz, the bandwidth available for a single PLB6 master or single PLB6 slave segment is 12.8GB/s.

PLB6 will not be available until 2009, and will be released with a handful of high-performance IP cores, as well as verification IP that supports different languages and simulators. While it is not expected that PLB6 will have the number or breadth of cores available on mid or low-performance buses, it is expected that application-specific high-performance cores will be designed to PLB6, once the architecture is released through a licensing program.

9 Summary

The Bus Architecture TSC has attempted to include as much information as possible in the previous sections describing each bus. However, the buses must be compared to one another at a higher level, side by side, in order to form the recommendations for buses going forward.

Desired characteristics of a bus vary between performance levels in the hierarchy. For example, simplicity is always desired. However, while simplicity of the interface is very important in the low-performance range, it is less important than other characteristics in the high-performance range.

The following tables list desirable attributes in each performance range of the hierarchy. Some buses appear in more than one table, since they are capable of spanning across performance ranges.

9.1 Low-performance Bus Comparison

Low-performance buses are geared for simplicity and ease of use. Developers in this space must be able to spend most of their time on the design and verification of the device function, not on the interface logic.

Key: 1 – Excellent 2 – Very Good 3 – Good 4 – Fair 5 – Poor

Description	AHB	APB	OCP	OPB
Architectural complexity	3	1	4	2
Interface simplicity	3	1	2	2
Core IP availability	2	1	4	2
Verification enablement	1	1	1	4
Interoperability	1	1	4	1
Bridges to other buses	1	1	3	1
Master and slave support	1	5	1	1

Description	AHB	APB	OCP	OPB
Accepts devices of different widths	5	5	1	1
Allows requests smaller than bus width	1	5	1	1
Slave retry	1	5	5	1
Byte enables	5	5	1 ¹	3
Timing requirements	5	5	1	2
Burst support	2	5	2 ¹	3
Error handling	2	2	2	2
Free to license	1	1	1	1
Open standard	5	5	1	5
Core IP portability	1	1	1	1

¹OCP can be configured to support this feature.

9.2 Mid-performance Bus Comparison

Mid-performance buses have a broad bandwidth range, and go from the low end to high end of the range through additional features and increases in data bus width. The lack of one key feature, such as data pipelining, can prevent a bus from spanning the entire range.

Key: 1 – Excellent 2 – Very Good 3 – Good 4 – Fair 5 – Poor

Description	AHB	AXI	OCP	PLB4
Architectural complexity	2	2	4	3
Design complexity	1	2	3	3
Core IP availability	1	2	3	2
Verification enablement	1	1	2	3
Interoperability	2	2	5	2
Power processor attachment	3	5	5	1
Bridges to other buses	1	2	3	1
Burst support	3	1	1 ⁴	2
Slave burst terminate	5	1 ¹	5	1
Data pipelining	5	1	1 ⁴	2
Pipelined burst complexity	3 ²	2	2	4
Simultaneous read and write data	5	1	1 ⁴	1
High frequency	2	1	1	3
Timing requirements	5	5	1	2
Delayed read	1	3 ³	3 ³	2
Out-of-order data	5	1	1 ⁴	5
Efficient unaligned transfers	3	3	3	3
Cut through writes	5	5	5	1

Description	AHB	AXI	OCP	PLB4
Slave retry	1	1	5	1
Byte enables	5	2	1 ⁴	1
Power transfer attributes	3	3	1 ⁴	1
Power storage synch (lwarx/stwcx)	4	2	2	4
Addressable space	4	4	1 ⁴	1
Distributed address decode	5	5	5	1
Communicate core IP performance	5	5	1	5
Error handling	2	2	2	2
Parity defined	5	5	4	1
Free to license	1	1	1	1
Open standard	5	5	1	5
Core IP portability	1	1	1	1

¹Buses that define bursts longer than 16 beats should allow slaves to burst terminate. Slave burst termination is not required on buses that limit burst requests to 16 beats or less.

²Does not apply to AHB, since AHB does not support data pipelining.

³Delayed reads are not required on buses that support out-of-order read data return.

⁴OCP can be configured to support this feature.

9.3 High-performance Bus Comparison

High-performance buses make more sacrifices in complexity and area to achieve the best possible performance. However, it is important to remember that time-to-market is always an important factor in the embedded SoC market. A bus that strives for the best performance without regard to the cost will not win in the marketplace. The best high-performance bus in this market offers high-performance features that are not extremely expensive to implement, and pushes complexity into the master, especially coherent master, designs.

Key: 1 – Excellent 2 – Very Good 3 – Good 4 – Fair 5 – Poor

Description	AXI	OCP	PLB4	PLB6
Architectural complexity	1	4	3	2
Design complexity ¹	1	3	3	2
Core IP availability	2	3	2	5 ²
Verification enablement	1	1	3	5 ²
Interoperability	2	4	3	1
Power processor attachment	5	5	1	1
Bridges to other buses	2	3	2	1
Multi-core cache coherency	5	5	5	1
I/O coherency	5	5	5	1
Intervention capability	5	5	5	1
Burst support	1	1 ³	2	1
Slave burst terminate ⁴	2	5	1	1
Command pipelining	5	5	5	1

Description	AXI	OCP	PLB4	PLB6
Data pipelining	2	1 ³	3	1
Pipelined burst complexity	1	1 ³	3	1
Data bus width complexity	3	3	1	1
Simultaneous read and write data	1	1 ³	1	1
High frequency	1	1	3	1
Timing requirements	5	1	2	2
Out-of-order data	1	1 ³	4	1
Efficient unaligned transfers	3	3	3	1
Cut through writes	5	1 ³	2	1
Slave retry	5	5	1	1
Byte enables	2	1 ³	1	1
Power transfer attributes	4	1 ³	1	1
Power storage synch (lwarx/stwcx)	2	2	4	1
Power memory barrier (mbar/msync)	4	4	4	1
Addressable space	4	1 ³	1	1
Distributed address decode	5	5	1	1
Communicate core IP performance	5	1	3	2
Error handling	2	1	2	1
Parity defined	5	5	1	1
Free to license	1	1	1	5 ²

Description	AXI	OCP	PLB4	PLB6
Open standard	5	1	5	5
Core IP portability	1	1	1	1

¹Reflects relative complexity of designing a non-coherent slave with pipelining.

²PLB6 is not currently available for use. Targeted availability for PLB6 Core and Verification IP is 2009.

³OCP can be configured for high-performance with respect to this feature.

⁴Buses that define bursts longer than 16 beats should allow slaves to burst terminate. Slave burst termination is not required on buses that limit burst requests to 16 beats or less.

9.4 Dual Family Architecture

As shown in the bus comparison tables, there is no single bus that has every desired feature available, even in one performance category. Many of the performance-enhancing features themselves have trade-offs in complexity; nothing comes for free. It is the responsibility of the SoC architect to make the sometimes difficult decisions required to achieve the right balance between performance and area/cost/power. The Bus Architecture TSC supports the idea of providing SoC design teams with many choices of core and verification IP, and formally supports the concept of a dual family architecture, with cores on both AMBA and CoreConnect buses.

Figure 2 shows an AMBA only configuration, with a low to mid-performance Power processor connected either directly or through a small gasket to AHB, with a bridge to APB for low-performance cores.

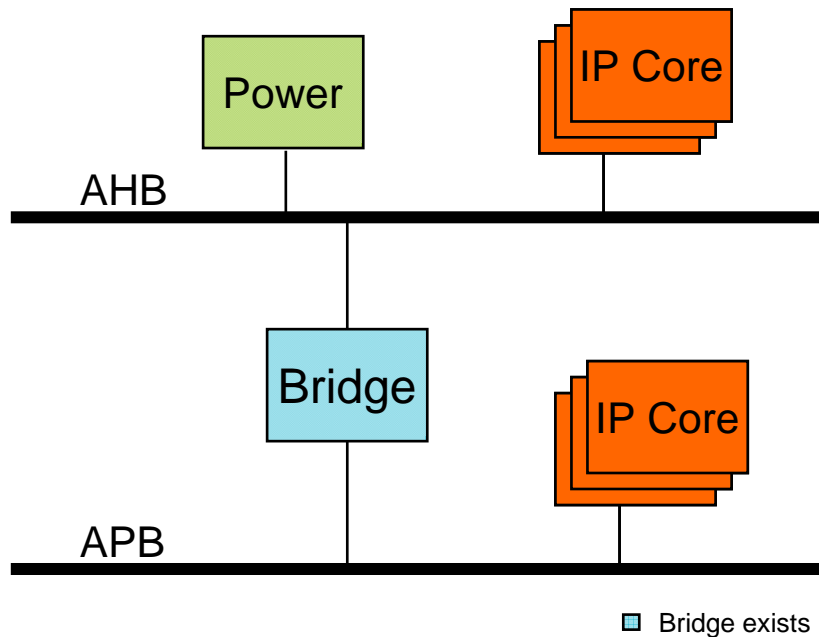


Figure 2. Low/mid-performance, AMBA only

Figure 3 shows a CoreConnect only solution, with a mid-performance Power processor attached to PLB4, with low-performance IP connected to OPB.

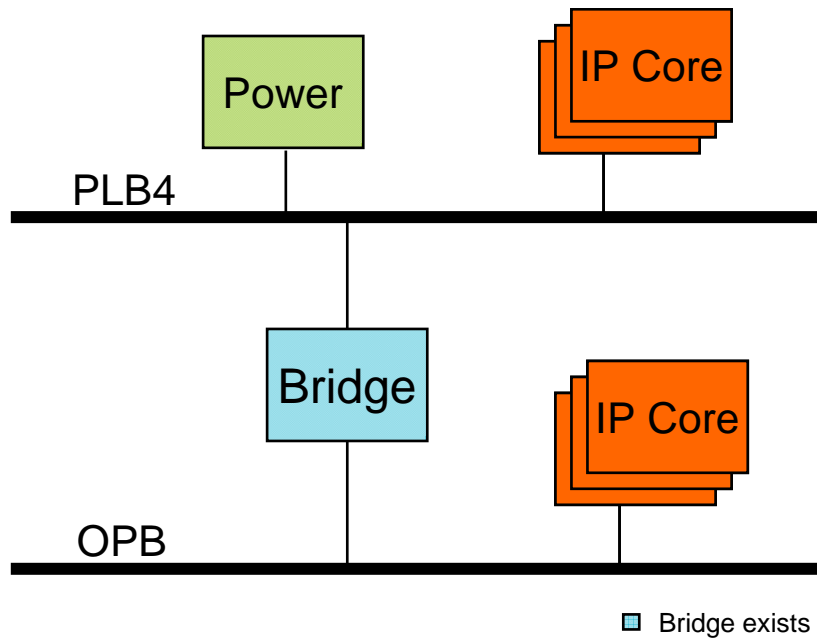


Figure 3. Mid-performance, CoreConnect only

Figure 4 shows a mid-performance Power processor connected to PLB4, with bridges to both AHB and APB.

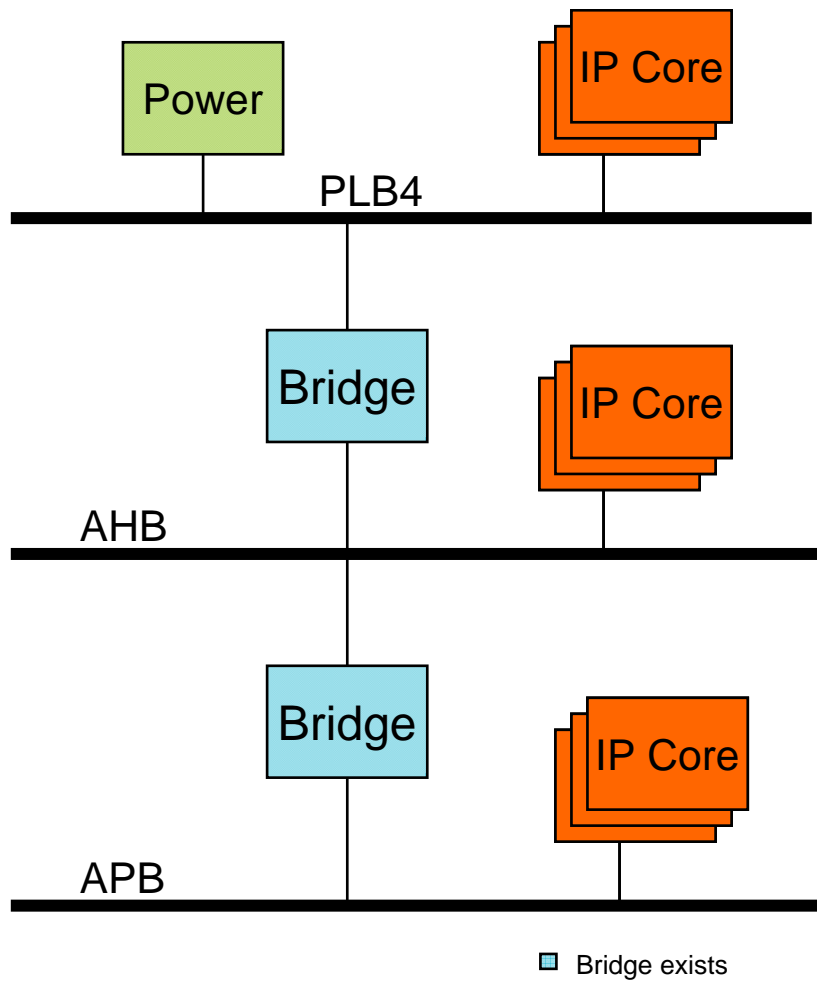


Figure 4. Mid-performance, CoreConnect and AMBA

Figure 5 shows two high-performance Power processors connected to a high-performance, cache coherent bus, with bridges to AXI, AHB, and APB.

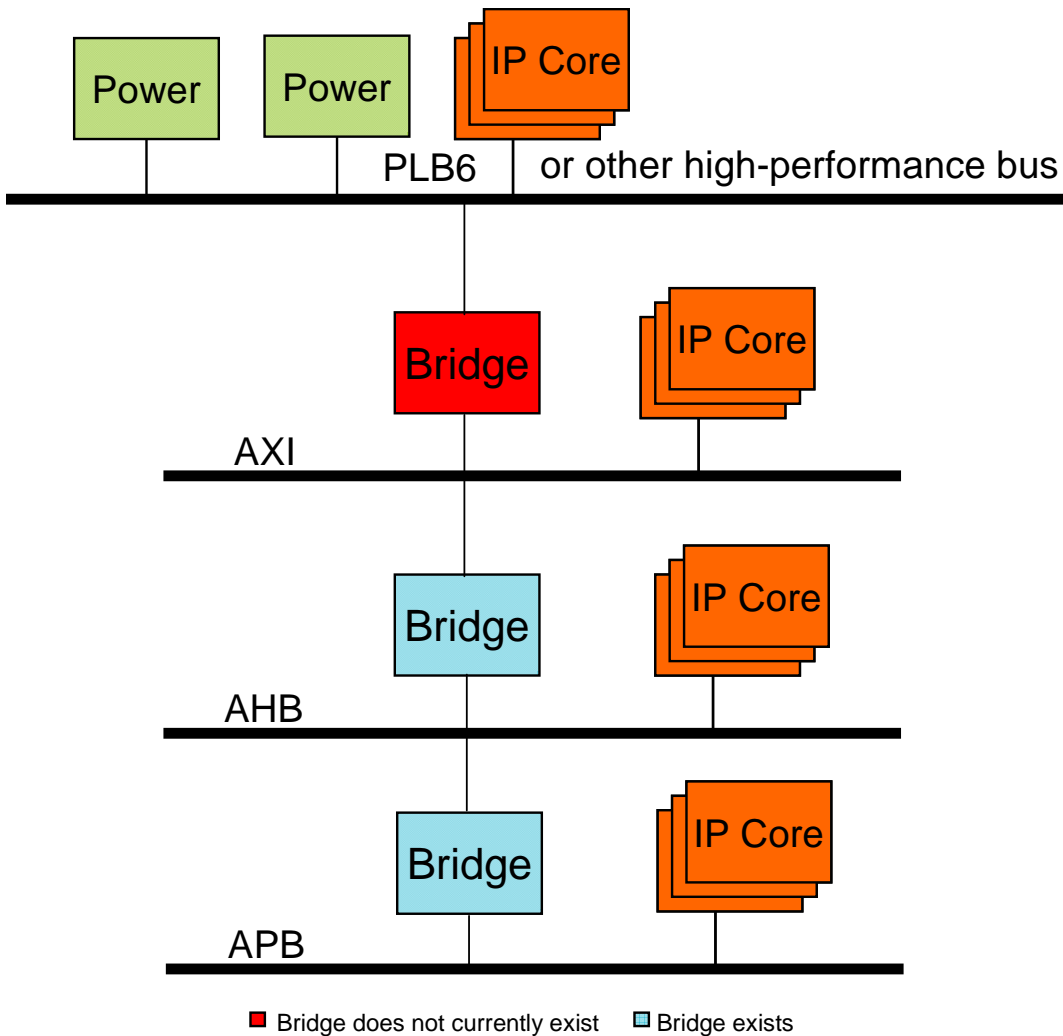


Figure 5. High-performance with AMBA

Figure 6 shows four Power processors connected to a high-performance, cache coherent bus. There are bridges to both AXI and PLB4, and bridges below to AHB and APB, and OPB. Note that either PLB4 or AHB may offer another attachment point for a Power processor operating on memory that is not coherent with the memory used by the Power processors on the high-performance bus.

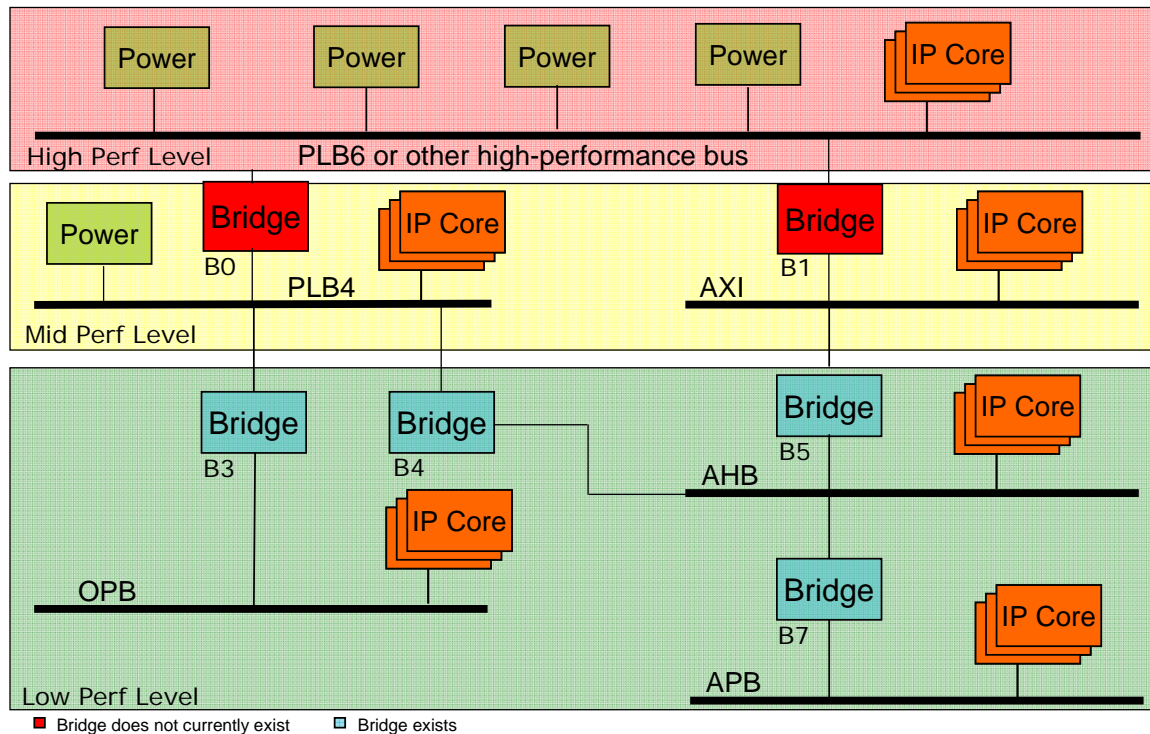


Figure 6. High-performance with CoreConnect and AMBA