The CoreConnect™ Bus Architecture

Recent advances in silicon densities now allow for the integration of numerous functions onto a single silicon chip. With this increased density, peripherals formerly attached to the processor at the card level are integrated onto the same die as the processor. As a result, chip designers must now address issues traditionally handled by the system designer. In particular, the on-chip buses used in such system-on-a-chip designs must be sufficiently flexible and robust in order to support a wide variety of embedded system needs.

The IBM Blue Logic™ cores program provides the framework to efficiently realize complex system-on-a-chip (SOC) designs. Typically, a SOC contains numerous functional blocks representing a very large number of logic gates. Designs such as these are best realized through a macro-based approach. Macro-based design provides numerous benefits during logic entry and verification, but the ability to reuse intellectual property is often the most significant. From generic serial ports to complex memory controllers and processor cores, each SOC generally requires the use of common macros.

Many single chip solutions used in applications today are designed as custom chips, each with its own internal architecture. Logical units within such a chip are often difficult to extract and re-use in different applications. As a result, many times the same function is redesigned from one application to another. Promoting reuse by ensuring macro interconnectivity is accomplished by using common buses for inter-macro communications. To that end, the IBM CoreConnect architecture provides three buses for interconnecting cores, library macros, and custom logic:

- Processor Local Bus (PLB)
- On-Chip Peripheral Bus (OPB)
- Device Control Register (DCR) Bus

Figure 1 illustrates how the CoreConnect architecture can be used to interconnect macros in a PowerPC 440 based SOC. High performance, high bandwidth blocks such as the PowerPC 440 CPU core, PCI-X Bridge and PC133/DDR133 SDRAM Controller reside on the PLB, while the OPB hosts lower data rate peripherals. The daisy-chained DCR bus provides a relatively low-speed data path for passing configuration and status information between the PowerPC 440 CPU core and other on-chip macros.

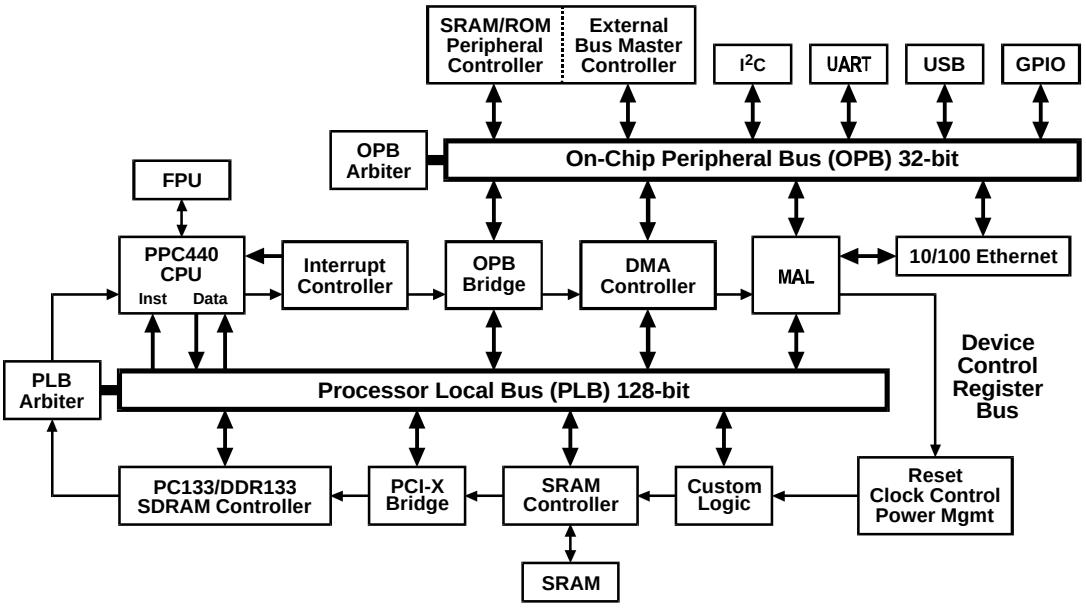

Figure 1: CoreConnect Based System-On-a-Chip.

	IBM CoreConnect Processor Local Bus	ARM AMBA 2.0 AMBA High-performance Bus
Bus Architecture	32-, 64-, and 128-bits Extendable to 256-bits	32-, 64-, and 128-bits
Data Buses	Separate Read and Write	Separate Read and Write
Key Capabilities	Multiple Bus Masters 4 Deep Read Pipelining 2 Deep Write Pipelining Split Transactions Burst Transfers Line Transfers	Multiple Bus Masters Pipelining Split Transactions Burst Transfers Line Transfers
	On-Chip Peripheral Bus	**AMBA Advanced Peripheral Bus**
Masters Supported	Supports Multiple Masters	Single Master: The APB Bridge
Bridge Function	Master on PLB or OPB	APB Master Only
Data Buses	Separate Read and Write	Separate or 3-state

Table 1: Comparison of IBM CoreConnect and ARM AMBA 2.0 Architectures.

The CoreConnect architecture shares many similarities with the Advanced Microcontroller Bus Architecture (AMBA™) from ARM Ltd. As shown in Table 1, the recently announced AMBA 2.0[1] includes the specification of many high performance features that have been available in the CoreConnect architecture for over three years. Both architectures support data bus widths of 32-bits and higher, utilize separate read and write data paths and allow multiple masters. CoreConnect and AMBA 2.0 now both provide high performance features including pipelining, split transactions and burst transfers. Many custom designs utilizing the high performance features of the CoreConnect architecture are available in the marketplace today.

Open specifications for the CoreConnect architecture are available on the IBM Microelectronics web site. In addition, IBM offers a no-fee, royalty-free CoreConnect architectural license. Licensees receive the PLB arbiter, OPB arbiter and PLB/OPB bridge designs along with bus model toolkits and bus functional compilers for the PLB, OPB and DCR buses. In the future, IBM intends to include compliance test suites for each of the three buses.

Processor Local Bus

The PLB and OPB buses provide the primary means of data flow among macro elements. Because these two buses have different structures and control signals, individual macros are designed to interface to either the PLB or the OPB. Usually the PLB interconnects high-bandwidth devices such as processor cores, external memory interfaces and DMA controllers.

The PLB addresses the high performance, low latency and design flexibility issues needed in a highly integrated SOC through:

- Decoupled address, read data, and write data buses with split transaction capability
- Concurrent read and write transfers yielding a maximum bus utilization of two data transfers per clock
- Address pipelining that reduces bus latency by overlapping a new write request with an ongoing write transfer and up to three read requests with an ongoing read transfer.
- Ability to overlap the bus request/grant protocol with an ongoing transfer

[1] ARM AMBA data obtained from the ARM web pages.

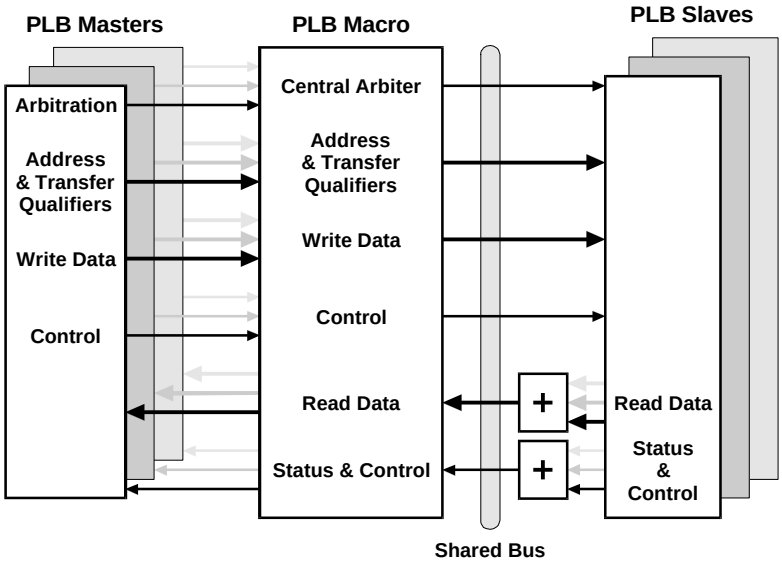

Figure 2: Example of PLB Interconnection.

In addition to providing a high bandwidth data path, the PLB offers designers flexibility through the following features:

- Support for both multiple masters and slaves
- Four priority levels for master requests allowing PLB implementations with various arbitration schemes
- Deadlock avoidance through slave forced PLB rearbitration
- Master driven atomic operations through a bus arbitration locking mechanism
- Byte-enable capability, supporting unaligned transfers
- A sequential burst protocol allowing byte, half-word, word and double-word burst transfers
- Support for 16-, 32- and 64-byte line data transfers
- Read word address capability, allowing slaves to return line data either sequentially or target word first
- DMA support for buffered, fly-by, peripheral-to-memory, memory-to-peripheral, and memory-to-memory transfers
- Guarded or unguarded memory transfers allow slaves to individually enable or disable prefetching of instructions or data
- Slave error reporting
- Architecture extendable to 256-bit data buses
- Fully synchronous

The PLB specification describes a system architecture along with a detailed description of the signals and transactions. PLB-based custom logic systems require the use of a PLB macro to interconnect the various master and slave macros.

Figure 2 illustrates the connection of multiple masters and slaves through the PLB macro. Each PLB master is attached to the PLB macro via separate address, read data and write data buses and a plurality of transfer qualifier signals. PLB slaves are attached to the PLB macro via shared, but decoupled, address, read data and write data buses along with transfer control and status signals for each data bus.

The PLB architecture supports up to 16 master devices. Specific PLB macro implementations, however, may support fewer masters. The PLB architecture also supports any number of slave devices. The

number of masters and slaves attached to a PLB macro directly affects the maximum attainable PLB bus clock rate. This is because larger systems tend to have increased bus wireload and a longer delay in arbitrating among multiple masters and slaves.

The PLB macro consists of a bus arbitration control unit and the control logic required to manage the address and data flow through the PLB. The separate address and data buses from the masters allow simultaneous transfer requests. The PLB macro arbitrates among these requests and directs the address, data and control signals from the granted master to the slave bus. The slave response is then routed from the slave bus back to the appropriate master.

PLB Bus Transactions

PLB transactions consist of multiphase address and data tenures. Depending on the level of bus activity and capabilities of the PLB slaves, these tenures may be one or more PLB bus cycles in duration. In addition, address pipelining and separate read and write data buses yield increased bus throughput by way of concurrent tenures. Address tenures have three phases: request, transfer and address acknowledge. A PLB transaction begins when a master drives its address and transfer qualifier signals and requests ownership of the bus during the request phase of the address tenure. Once the PLB arbiter grants bus ownership the master's address and transfer qualifiers are presented to the slave devices during the transfer phase. The address cycle terminates when a slave latches the master's address and transfer qualifiers during the address acknowledge phase.

Figure 3 illustrates two deep read and write address pipelining along with concurrent read and write data tenures. Master A and Master B represent the state of each master's address and transfer qualifiers. The PLB arbitrates between these requests and passes the selected master's request to the PLB slave address bus. The trace labeled Address Phase shows the state of the PLB slave address bus during each PLB clock.

As shown in Figure 3, the PLB specification supports implementations where these three phases can require only a single PLB clock cycle. This occurs when the requesting master is immediately granted access to the slave bus and the slave acknowledges the address in the same cycle. If a master issues a request that cannot be immediately forwarded to the slave bus, the request phase lasts one or more cycles.

Each data beat in the data tenure has two phases: transfer and acknowledge. During the transfer phase the master drives the write data bus for a write transfer or samples the read data bus for a read transfer. As shown in Figure 3, the first (or only) data beat of a write transfer coincides with the address transfer

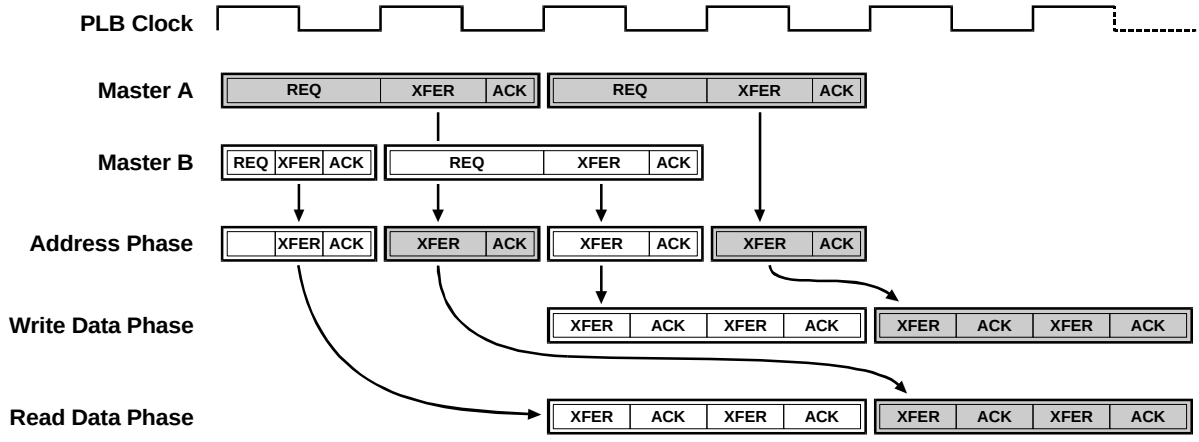

Figure 3: PLB Transfer Protocol Example.

phase. Data acknowledge cycles are required during the data acknowledge phase for each data beat in a data cycle. In the case of a single-beat transfer, the data acknowledge signals also indicate the end of the data transfer. For line or burst transfers, the data acknowledge signals apply to each individual beat and indicate the end of the data cycle only after the final beat. The highest data throughput occurs when data is transferred between master and slave in a single PLB clock cycle. In this case the data transfer and data acknowledge phases are coincident. During multi-cycle accesses there is a wait-state either before or between the data transfer and data acknowledge phases.

The PLB address, read data, and write data buses are decoupled from one another, allowing for address cycles to be overlapped with read or write data cycles, and for read data cycles to be overlapped with write data cycles. The PLB split bus transaction capability allows the address and data buses to have different masters at the same time. Additionally, a second master may request ownership of the PLB, via address pipelining, in parallel with the data cycle of another master's bus transfer. This is shown in Figure 3. Overlapped read and write data transfers and split-bus transactions allow the PLB to operate at a very high bandwidth by fully utilizing the read and write data buses.

Allowing PLB devices to move data using long burst transfers can further enhance bus throughput. However, to control the maximum latency in a particular application, master latency timers are required. All masters able to issue burst operations must contain a latency timer that increments at the PLB clock rate and a latency count register. The latency count register is an example of a configuration register that is accessed via the DCR bus. During a burst operation, the latency timer begins counting after an address acknowledge is received from a slave. When the latency timer exceeds the value programmed into the latency count register, the master can either immediately terminate its burst, continue until another master requests the bus or continue until another master requests the bus with a higher priority.

PLB Cross-Bar Switch

In some PLB-based systems multiple masters may cause the aggregate data bandwidth to exceed that which can be satisfied with a single PLB. With such a system it may be possible to place the high data rate masters and their target slaves on separate PLB buses. An example is a multiprocessor system using separate memory controllers. A macro known as the PLB Cross-Bar Switch (CBS) can be utilized to allow communication between masters on one PLB and slaves on the other. As shown in Figure 4, the CBS is placed between the PLB arbiters and their slave buses. When a master begins a transaction, the CBS uses the associated address to select the appropriate slave bus. The CBS supports simultaneous data transfers on both PLB buses along with a prioritization scheme to handle multiple requests to a common slave port. In addition, a high priority request can interrupt a lower priority transaction.

On-Chip Peripheral Bus

The On-Chip Peripheral Bus (OPB) is a secondary bus architected to alleviate system performance bottlenecks by reducing capacitive loading on the PLB. Peripherals suitable for attachment to the OPB include serial ports, parallel ports, UARTs, GPIO, timers and other low-bandwidth devices. As part of the

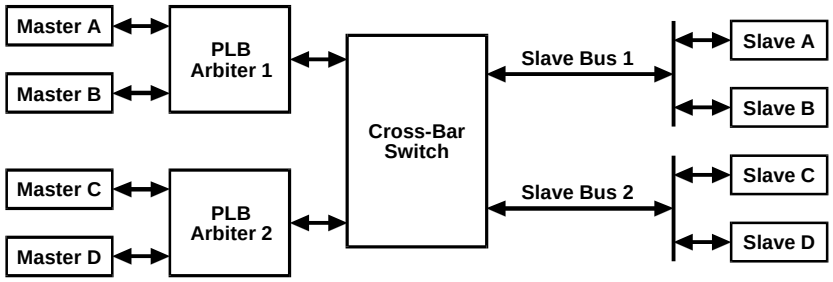

Figure 4: PLB Crossbar Switch.

IBM Blue Logic cores program, all OPB core peripherals directly attach to OPB. This common design point accelerates the design cycle time by allowing system designers to easily integrate complex peripherals into an ASIC.

The OPB provides the following features:

- A fully synchronous protocol with separate 32-bit address and data buses
- Dynamic bus sizing to support byte, half-word and word transfers
- Byte and half-word duplication for byte and half-word transfers
- A sequential address (burst) protocol
- Support for multiple OPB bus masters
- Bus parking for reduced-latency transfers

OPB Bridge

PLB masters gain access to the peripherals on the OPB bus through the OPB bridge macro. The OPB bridge acts as a slave device on the PLB and a master on the OPB. It supports word (32-bit), half-word (16-bit) and byte read and write transfers on the 32-bit OPB data bus, bursts and has the capability to perform target word first line read accesses. The OPB bridge performs dynamic bus sizing, allowing devices with different data widths to efficiently communicate. When the OPB bridge master performs an operation wider than the selected OPB slave the bridge splits the operation into two or more smaller transfers.

OPB Implementation

The OPB supports multiple masters and slaves by implementing the address and data buses as a distributed multiplexer. This type of structure is suitable for the less data intensive OPB bus and allows adding peripherals to a custom core logic design without changing the I/O on either the OPB arbiter or existing peripherals. Figure 5 shows one method of structuring the OPB address and data buses. Observe that both masters and slaves provide enable control signals for their outbound buses. By requiring that each macro provide this signal, the associated bus combining logic can be strategically

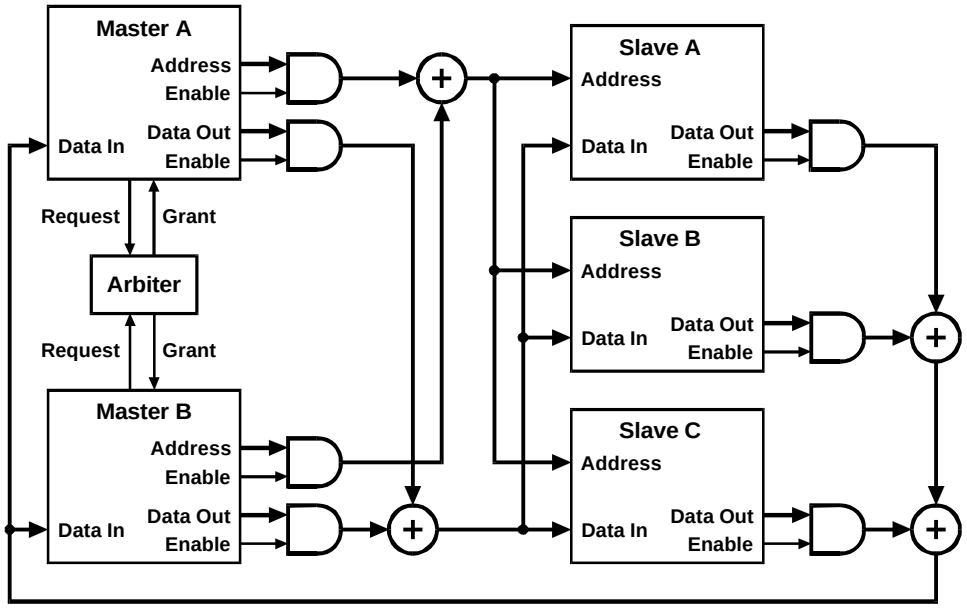

Figure 5: OPB Physical Implementation.

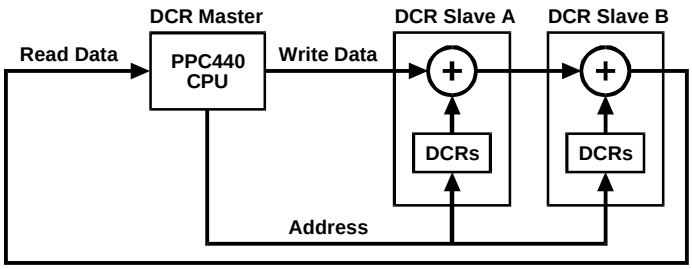

Figure 6: DCR Bus Structure.

placed throughout the chip. As shown in the figure, either of the masters is capable of providing an address to the slaves, whereas both masters and slaves are capable of driving and receiving the distributed data bus.

DCR Bus

Lower performance status and configuration registers are typically read and written through the Device Control Register (DCR) Bus. The DCR provides a maximum throughput of one read or write transfer every two cycles and is a fully synchronous bus typically implemented as a distributed multiplexer. Figure 6 illustrates the address and data flow between a processor core master and the DCR slaves. Observe that the relatively slow DCR bus utilizes a ring-type data bus. This provides the required connectivity while minimizing silicon usage.

Design Toolkits

Design toolkits are available for each of the on-chip buses. These toolkits contain master, slave and arbiter Bus Functional Models (BFM). Also provided is a Bus Functional Compiler used to translate testcases written in a bus functional language into simulator commands executable by the master and slave models.

These toolkits aid in unit and subsystem simulation of macros designed for attachment to any of the three on-chip buses. The toolkits are available in VHDL and Verilog such that they can be used in any simulation environment. The Bus Functional Language (BFL) command definition is unique for each bus and allows the user to execute and respond to all allowable transactions on the particular bus. BFL is processed by the bus functional compiler into command sequences executable by a BFM running on an event driven simulator such as VerilogXL, MTI or VSS.

Each bus toolkit provides a bus monitor that automatically performs architectural protocol checking on all masters and slaves attached to the bus. These checks verify that the masters and slaves under test adhere to the defined bus protocol and help to ensure compatibility when the macros are interconnected in a system environment. In addition to protocol checking, the master and slave models also perform read and write data checking. For example, when a master is programmed to perform a read transfer, the model checks the read data with the expected data as programmed in the BFL.

The toolkits also support concurrently executing multiple master models. A model intercommunication scheme is provided to allow transaction synchronization between the masters. One use of this feature is creating bus contention testcases often necessary to verify a macro.

Summary

The open standard CoreConnect bus provides the primary means of communication between macros in an IBM Blue Logic design. Designers realize several advantages in creating and using compliant macros. First, IBM is continually expanding its extensive library of Blue Logic off-the-shelf core macros. These logic blocks are easily integrated into a customer's SOC solution. Additionally, customers benefit by being able to reuse their own macros in multiple products. Finally, these common interfaces allow for easy integration, providing valuable time savings when implementing complex designs.

Design toolkits are available for the CoreConnect bus and include functional models, monitors, and a bus functional language to drive the models. These toolkits provide an excellent validation environment for engineers designing macros to attach to the PLB, OPB and DCR buses.

Further information on CoreConnect licensing, IBM Blue Logic, available cores, macros, and design toolkits is available on the WWW at:

http://www.chips.ibm.com/products/coreconnect