Overview

- Basics of a Bus and SoC/On-chip Busses
- AMBA 2.0 and 3.0 - AHB, APB and AXI Protocols
- IBM CoreConnect Bus – PLB and OPB
- Avalon Bus
- StBus (STMicroelectronics)
SoC Integration and Interconnect Architectures

• **SoC Integration is the most important part of SoC design.**
  - Integration of IP cores.
  - The method connect the IP cores.
  - Maximize the reuse of design to lower cost.

• **SoC Interconnect Architectures**
  - Bus-based Interconnection.
  - NoC: Network on Chip that hides the physical interconnects from the designer.
Busses: Basic Architecture

PCB Busses – VME, Multibus-II, ISA, EISA, PCI and PCI Express

• Bus is made of wires shared by multiple units with logic to provide an orderly use of the bus.
• Devices can be Masters or Slaves.
• Arbiter determines - which device will control the bus.
• Bus protocol is a set of rules for transmitting information between two or more devices over a bus.
• Bus bridge connects two buses, which are not of the same type having different protocols.
• Buses may be unified or split type (address and data).
Bus: The Basic Architecture

Decoder determines the target for any transfer initiated by a master
Bus Signals

Typically a bus has three types of signal lines

**Address**
- Carry address of destination for which transfer is initiated
- Can be shared or separate for read, write data

**Data**
- Transfer information between source and destination devices
- Can be shared or separate for read, write data

**Control**
- Requests and acknowledgements
- Specify more information about type of data transfer e.g. Byte enable, burst size, cacheable/bufferable, ...
Bus Interconnection Architectures

IP blocks need to communicate among each other

• **System level issues and specifications of an SoC Interconnect:**
  - Communication Bandwidth – Rate of Information Transfer
  - Communication Latency – Delay between a module requesting the data and receiving a response to its request.
  - Master and Slave – Initiate (Master) or response (Slave) to communication requests
  - Concurrency Requirements – Simultaneous Comm. Channels
  - Packet or Bus Transaction – Information size per transaction
  - Multiple Clock Domains – IP module operate at different clocks
Bus Basics

Bus Master: has ability to control the bus, initiates transaction

Bus Slave: module activated by the transaction

Bus Communication Protocol: specification of sequence of events and timing requirements in transferring information.

Asynchronous Bus Transfers: control lines (req, ack) serve to orchestrate sequencing.

Synchronous Bus Transfers: sequence relative to common clock.
Embedded Systems busses
Actel SmartFusion system/bus
SoC Bus Architectures

<table>
<thead>
<tr>
<th>Technology</th>
<th>AMBA</th>
<th>AXI (AMBA 3)</th>
<th>CoreConnect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>ARM</td>
<td>ARM</td>
<td>IBM</td>
</tr>
<tr>
<td>Core type</td>
<td>Soft/hard</td>
<td>Soft/hard</td>
<td>Soft</td>
</tr>
<tr>
<td>Architecture</td>
<td>Bus</td>
<td>Unidirectional channels</td>
<td>Bus</td>
</tr>
<tr>
<td>Bus width</td>
<td>8–1024</td>
<td>8–1024</td>
<td>32/64/128</td>
</tr>
<tr>
<td>Frequency</td>
<td>200 MHz</td>
<td>400 MHz*</td>
<td>100–400 MHz</td>
</tr>
<tr>
<td>Maximum BW (GB/s)</td>
<td>3</td>
<td>6.4*</td>
<td>2.5–24</td>
</tr>
<tr>
<td>Minimum latency (ns)</td>
<td>5</td>
<td>2.5*</td>
<td>15</td>
</tr>
</tbody>
</table>

*As implemented in the ARM PL330 high-speed controller.

BW, bandwidth.

HW Area for a Slave

<table>
<thead>
<tr>
<th>Standard</th>
<th>Speed (MHz)</th>
<th>Area (rbe*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBA (implementation dependent)</td>
<td>166–400</td>
<td>175,000</td>
</tr>
<tr>
<td>CoreConnect</td>
<td>66/133/183</td>
<td>160,000</td>
</tr>
</tbody>
</table>

*rbe = register bit equivalent; estimates are approximate and vary by implementation.
On-Chip Busses

• AMBA 2.0, 3.0 (ARM)
• CoreConnect (IBM)
• Avalon (Altera)
• STBus (STMicroelectronics)
• Sonics Smart Interconnect (Sonics)
• Wishbone (Opencores)
• PI Bus (OMI)
• MARBLE (Univ. of Manchester)
• CoreFrame (PalmChip)
AMBA 2.0
Advance Microcontroller Bus Architecture

AMBA AHB
- High performance
- Pipelined operation
- Multiple bus masters
- Burst transfers
- Split transactions

AMBA ASB
- High performance
- Pipelined operation
- Multiple bus masters

AMBA APB
- Low power
- Latched address and control
- Simple interface
- Suitable for many peripherals
AMBA Busses

Advanced Microcontroller Bus Architecture

**Simple Bus**

Actually 3 standards: APB, AHB, and AXI

AHB – Advanced High-Performance Bus

- Pipelining of Address / Data
- Split Transactions
- Multiple Masters

APB – Advanced Peripheral Bus

- Low Power / Bandwidth Peripheral Bus

Very commonly used for commercial IP cores
APB Bus

- A simple bus that is easy to work with
- Low-cost
- Low-power
- Low-complexity
- Low-bandwidth
- Non-pipelined
- Ideal for peripherals
APB State Machine

• **IDLE**
  - Default APB state

• **SETUP**
  - When transfer required
  - $PSELx$ is asserted
  - Only one cycle

• **ACCESS**
  - $PENABLE$ is asserted
  - Addr, write, select, and write data remain stable
  - Stay if $PREADY = L$
  - Goto IDLE if $PREADY = H$ and no more data
  - Goto SETUP if $PREADY = H$ and more data pending
Notations

- Clock
- HIGH to LOW
- Transient
- HIGH/LOW to HIGH
- Bus stable
- Bus to high impedance
- Bus change
- High impedance to stable bus
APB Bus States

- **IDLE**
  - Default APB state

- **SETUP**
  - When transfer required
  - PSELx is asserted
  - Only one cycle

- **ACCESS**
  - PENABLE is asserted
  - Addr, write, select, and write data remain stable
  - Stay if PREADY = L
  - Goto IDLE if PREADY = H and no more data
  - Goto SETUP is PREADY = H and more data pending

Setup phase begins with this rising edge
APB Signals

• PCLK: the bus clock source (rising-edge triggered)
• PRESETn: the bus (and typically system) reset signal (active low)
• PADDR: the APB address bus (can be up to 32-bits wide)
• PSELx: the select line for each slave device
• PENABLE: indicates the 2\textsuperscript{nd} and subsequent cycles of an APB xfer
• PWRITE: indicates transfer direction (Write=H, Read=L)
• PWDATA: the write data bus (can be up to 32-bits wide)
• PREADY: used to extend a transfer
• PRDATA: the read data bus (can be up to 32-bits wide)
• PSLVERR: indicates a transfer error (OKAY=L, ERROR=H)
APB bus Signals

- **PCLK**
  - Clock
- **PAADDR**
  - Address on bus
- **PWRITE**
  - 1=Write, 0=Read
- **PWDATA**
  - Data written to the I/O device. Supplied by the bus master/processor.
APB Bus Signals

• **PSEL**
  - Asserted if the current bus transaction is targeted to *this* device

• **PENABLE**
  - High during entire transaction *other than* the first cycle.

• **PREADY**
  - Driven by target. Indicates if the target is *ready* to do the transaction.
  - *Each target has its own PREADY*
A Write Transfer - No Wait States

Setup phase begins with this rising edge.

- **PCLK**
- **PADDR**
- **PWRITE**
- **PSEL**
- **PENABLE**
- **PWDATA**
- **PREADY**

**Setup Phase**

**Access Phase**

G. Khan

SoC Interconnection  Bus Structures

Page: 21
A Write Transfer with Wait States

Setup phase begins with this rising edge

T0 T1 T2 T3 T4 T5 T6

PCLK
PADDR
PWRITE
PSEL
PENABLE
PWDATA
PREADY

Setup Phase
Wait State
Wait State
Access Phase

Addr 1
Data 1

G. Khan
SoC Interconnection Bus Structures
A Read Transfer - No Wait States

Setup phase begins with this rising edge

Setup Phase

Access Phase

Setup phase begins with this rising edge
A Read Transfer with Wait States

Setup phase begins with this rising edge

Setup Phase    Wait State    Wait State    Access Phase
AHB Bus

- 1 unidirectional address bus (HADDR)
- 2 unidirectional data buses (HWDATA, HRDATA)
- At any time only 1 active data bus
Simple AHB Transfer

no wait state
AHB-Lite Bus Master/Slave Interface

Global signals
- HCLK
- HRESETn

Master out/slave in
- HADDR (address)
- HWDATA (write data)
- Control
  - HWRITE
  - HSIZE
  - HBURST
  - HPROT
  - HTRANS
  - HMASTLOCK

Slave out/master in
- HRDATA (read data)
- HREADY
- HRESP
AHB-Lite Signals

Global Signals
- HCLK: the bus clock source (rising-edge triggered)
- HRESETn: the bus (and system) reset signal (active low)

Master out/slave in
- HADDR[31:0]: the 32-bit system address bus
- HWDATA[31:0]: the system write data bus
- Control
  - HWRITE: indicates transfer direction (Write=1, Read=0)
  - HSIZE[2:0]: indicates size of transfer (byte, halfword, or word)
  - HBURST[2:0]: indicates single or burst transfer (1, 4, 8, 16 beats)
  - HPROT[3:0]: provides protection information (e.g. I or D; user or handler)
  - HTRANS: indicates current transfer type (e.g. idle, busy, nonseq, seq)
  - HMASTLOCK: indicates a locked (atomic) transfer sequence

Slave out/master in
- HRDATA[31:0]: the slave read data bus
- HREADY: indicates previous transfer is complete
- HRESP: the transfer response (OKAY=0, ERROR=1)
Basic Read and Write - no Wait States

Pipelined Address & Data Transfer
Read Transfer - 2 Wait States

Two wait states added by slave by asserting HREADY low

Valid data produced
Write Transfer - One Wait State

One wait state added by slave by asserting HREADY low

Valid data held stable
Wait States extend the Address Phase of Next Transfer

Address stage of the next transfer is also extended

One wait state added by slave by asserting HREADY low
Types of Transfers

Four types (HTRANS[1:0])

- **IDLE (00)**
  - No data transfer is required
  - Slave must OKAY w/o waiting
  - Slave must ignore IDLE

- **BUSY (01)**
  - Insert idle cycles in a burst
  - Burst will continue afterward
  - Address/control reflects next transfer in burst
  - Slave must OKAY w/o waiting
  - Slave must ignore BUSY

- **NONSEQ (10)**
  - Indicates single transfer or first transfer of a burst
  - Address/control unrelated to prior transfers

- **SEQ (11)**
  - Remaining transfers in a burst
  - Addr = prior addr + transfer size
4-beat burst - Master busy & Slave Wait

Master busy indicated by HTRANS[1:0]

One wait state added by slave by asserting HREADY low
Size (width) of a Transfer

- HSIZE[2:0] encodes the size
- It cannot exceed the data bus width (e.g. 32-bits)
- HSIZE + HBURST is determines wrapping boundary for wrapping bursts
- HSIZE must remain constant throughout a burst transfer

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>Byte</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>16</td>
<td>Halfword</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>32</td>
<td>Word</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>64</td>
<td>Doubleword</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td>4-word line</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>256</td>
<td>8-word line</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>512</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1024</td>
<td>-</td>
</tr>
</tbody>
</table>
AHB Burst Types

<table>
<thead>
<tr>
<th>HBURST[2:0]</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>SINGLE</td>
<td>Single transfer</td>
</tr>
<tr>
<td>001</td>
<td>INCR</td>
<td>Incrementing burst of unspecified length</td>
</tr>
<tr>
<td>010</td>
<td>WRAP4</td>
<td>4-beat wrapping burst</td>
</tr>
<tr>
<td>011</td>
<td>INCR4</td>
<td>4-beat incrementing burst</td>
</tr>
<tr>
<td>100</td>
<td>WRAP8</td>
<td>8-beat wrapping burst</td>
</tr>
<tr>
<td>101</td>
<td>INCR8</td>
<td>8-beat incrementing burst</td>
</tr>
<tr>
<td>110</td>
<td>WRAP16</td>
<td>16-beat wrapping burst</td>
</tr>
<tr>
<td>111</td>
<td>INCR16</td>
<td>16-beat incrementing burst</td>
</tr>
</tbody>
</table>

- Burst of 1, 4, 8, 16, and undef
- Wrapping bursts: (1) 4 beats x 4-byte words wrapping
  (2) Wraps at 16 byte boundary (e.g. 0x34, 0x38, 0x3c, 0x30,...)
- Bursts must not cross 1KB address boundaries.
Four beat Wrapping Burst (WRAP4)
Four-beat Incrementing Burst (INCR4)
An Eight-beat Wrapping Burst (WRAP8)
An Eight-beat Incrementing burst (INCR8) using Half-word Transfers
An Undefined Length Incrementing Burst (INCR)
Multi-master AHB-Lite Requires a Multi-layer Interconnect

• AHB-Lite is single-master
• Multi-master operation
  ▪ Must isolate masters
  ▪ Each master assigned to layer
  ▪ Interconnect arbitrates slave accesses
• Full crossbar switch often not needed
  ▪ Slaves 1, 2, 3 are shared
  ▪ Slaves 4, 5 are local to Master 1
AHB Control signals

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>Opcode fetch</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>Data access</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>User access</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>Privileged access</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Not bufferable</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>Bufferable</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Not cacheable</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Cacheable</td>
</tr>
</tbody>
</table>

- Protection control
  HPROT[3:0], provide additional information about a bus access
AHB Pipelining with Burst

Address and data of consecutive transfers are transmitted in the same clock cycle
AHB Pipelined Transactions

Transaction A Starts

Transaction B Starts

Transaction A Completes

Note backpressure
AHB Pipelined Burst Transfers

Bursts cut down arbitration, handshaking time, improve performance
AHB Split Transfers

- Improves bus utilization
- May cause deadlocks if not carefully implemented
AHB Bus Matrix

AHB can be employed and implemented as a bus matrix.
AHB-APB Bridge

AHB signals

- System bus slave interface
- Read data
- Reset
- Clock

APB bridge

- PSEL1
- PSEL2
- \ldots
- PSEL_n
- PENABLE
- PADDR
- PWRITE
- PWDATA

Selects
Strobe
Address and control
Write data

High performance
Low power (& performance)
AMBA Bus Arbitration

• Several masters and slaves are connected to AHB.
• An arbiter decides which master will transfer data.
• Data is transferred from a master to a slave in bursts.
• Any burst involves read/write of a sequence of addresses.
• The slave to service a burst is chosen depending on the addresses (decided by a decoder).
• AHB is connected to APB via a bus bridge.

Let us study the transfer features of AHB protocol.
AHB Arbitration

![Diagram of AHB Arbitration with arbitration, masters, and HADDR outputs.](image-url)
Request Grant Protocol

Before a transaction a master makes a request to the central arbiter. Eventually the request is granted. The transaction proceeds.

Performance Impact
**Arbitration Cost**

<table>
<thead>
<tr>
<th>Time for arbitration</th>
<th>Time for handshaking</th>
</tr>
</thead>
</table>

### Timelines:

- **T1**: Master asserts request
- **T2**: A number of cycles later, arbiter asserts grant
- **T3**: Master drives address after both HGRANT and HREADY are high
- **T4**: Address sampled and data starts when HREADY high

### Signals:

- **HCLK**
- **HBUSREQx**
- **HGRANTx**
- **HMASTER[3:0]**
- **HADDR[31:0]**
- **HWDATA[31:0]**
- **HREADY**
AMBA 3.0

Introduces AXI high performance protocol

- Support for separate read address, write address, read data, write data, write response channels
- Out of order transaction completion
- Fixed mode burst support
  - Useful for I/O peripherals
- Advanced system cache support
  - Specify if transaction is cacheable and buffer-able
  - Specify attributes such as write-back/write-through
- Enhanced protection support
  - Secure/non-secure transaction specification
- Exclusive access (for semaphore operations)
- Register slice support for high frequency operation
AMBA AXI Read Channels

Independent

Give me some data

Master interface

Read address channel

Address and control

Read data channel

Read data

Read data

Read data

Read data

Slave interface

Here it is
AMBA AXI Write Channels

I’m sending data. Please store it.

Here is the data.

I received that data correctly.
AMBA AXI Write Channels

Sending data, store it.

The data is here.

I received that data correctly.

channels synchronized with ID # or “tags”
AMBA AXI Flow-Control

- Information moves only when:
  - Source is Valid, and
  - Destination is Ready

- On each channel the master or slave can limit the flow

- Very flexible
AMBA AXI Read

Read Address Channel

Read Data Channel
AMBA AXI Write

Write Address Channel

Write Data Channel

Write Response Channel
AHB vs. AXI Burst

AHB Burst

- Address and Data are locked together (a single pipeline stage).
- *HREADY* controls intervals of address and data.

AXI Burst: One Address for entire burst
AHB vs. AXI Burst

AXI Burst

- Simultaneous read, write transactions
- Better bus utilization
AXI Out of Order Completion

With AHB

- If one slave is very slow, all data is held up
- SPLIT transactions provide very limited improvement

With AXI Burst

- Multiple outstanding addresses, out of order (OO) completion allowed
- Fast slaves may return data ahead of slow slaves
# AHB vs. AXI - Summary

<table>
<thead>
<tr>
<th>AMBA 3.0 AXI</th>
<th>AMBA 2.0 AHB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel-based specification, with five separate channels for read address,</td>
<td>Explicit bus-based specification, with single shared address bus and separate</td>
</tr>
<tr>
<td>read data, write address, write data, and write response enabling flexibility</td>
<td>read and write data buses.</td>
</tr>
<tr>
<td>in implementation.</td>
<td></td>
</tr>
<tr>
<td>Burst mode requires transmitting address of only first data item on the bus.</td>
<td>Requires transmitting address of every data item transmitted on the bus.</td>
</tr>
<tr>
<td>OO transaction completion provides native support for multiple, outstanding</td>
<td>Simpler SPLIT transaction scheme provides limited and rudimentary</td>
</tr>
<tr>
<td>transactions.</td>
<td>outstanding transaction completion.</td>
</tr>
<tr>
<td>Fixed burst mode for memory mapped I/O peripherals.</td>
<td>No fixed burst mode.</td>
</tr>
<tr>
<td>Exclusive data access (semaphore operation) support.</td>
<td>No exclusive access support.</td>
</tr>
<tr>
<td>Advanced security and cache hint support.</td>
<td>Simple protection and cache hint support.</td>
</tr>
<tr>
<td>Register slice support for timing isolation.</td>
<td>No inherent support for timing isolation.</td>
</tr>
<tr>
<td>Native low-power clock control interface.</td>
<td>No low-power interface.</td>
</tr>
<tr>
<td>Default bus matrix topology support.</td>
<td>Default hierarchical bus topology support.</td>
</tr>
</tbody>
</table>
IBM CoreConnect On-Chip Bus

CoreConnect is an SOC Bus proposed by IBM having:

- **PLB**: Processor Local Bus, PLB Arbiter, PLB to OPB Bridge
- **OPB**: On-Chip Peripheral Bus, OPB Arbiter
- **DCR**: Device Control Register Bus and a Bridge
CoreConnect Advance Features

IBM CoreConnect Bus with 32-, 64-, and 128-bit versions to support a variety of applications

PLB: Fully synchronous, supports up to 8 masters
- Separate read/write data buses
- Burst transfers, variable and fixed-length, Pipelining
- DMA transfers and No on-chip tri-states required
- Overlapped arbitration, programmable priority fairness

OPB: Fully synchronous, 32-bit address and data buses
- Support 1-cycle data transfers between master and slaves
- Arbitration for up to 4 OPB master peripherals
- Bridge function can be master on PLB or OPB

DCR: Provides fully synchronous movement of GPR data between CPU and slave logic
CoreConnect Bus based SoC

[Diagram of CoreConnect Bus based SoC]

- SRAM/ROM Peripheral Controller
- External Bus Master Controller
- I²C
- UART
- USB
- GPIO
- On-Chip Peripheral Bus (OPB) 32-bit
- OPB Arbiter
- PPC440 CPU
- Int. Controller
- OPB Bridge
- DMA Controller
- MAL
- 10/100 Ethernet
- Device Control Register Bus
- Processor Local Bus (PLB) 128-bit
- PLB Arbiter
- PC133/DDR133 SDRAM Controller
- PCI-X Bridge
- SRAM Controller
- Custom Logic
- Reset Clock Control Power Mgmt
- SRAM
Comparing AMBA and CoreConnect SoC Buses

<table>
<thead>
<tr>
<th></th>
<th>IBM CoreConnect Processor Local Bus</th>
<th>ARM AMBA 2.0 AMBA High-performance Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bus Architecture</strong></td>
<td>32-, 64-, and 128-bits Extendable to 256-bits</td>
<td>32-, 64-, and 128-bits</td>
</tr>
<tr>
<td><strong>Data Buses</strong></td>
<td>Separate Read and Write</td>
<td>Separate Read and Write</td>
</tr>
<tr>
<td><strong>Key Capabilities</strong></td>
<td>Multiple Bus Masters</td>
<td>Multiple Bus Masters</td>
</tr>
<tr>
<td></td>
<td>4 Deep Read Pipelining</td>
<td>Pipelining</td>
</tr>
<tr>
<td></td>
<td>2 Deep Write Pipelining</td>
<td>Split Transactions</td>
</tr>
<tr>
<td></td>
<td>Split Transactions</td>
<td>Burst Transfers</td>
</tr>
<tr>
<td></td>
<td>Burst Transfers</td>
<td>Line Transfers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>On-Chip Peripheral Bus</th>
<th>AMBA Advanced Peripheral Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Masters Supported</strong></td>
<td>Supports Multiple Masters</td>
<td>Single Master: The APB Bridge</td>
</tr>
<tr>
<td><strong>Bridge Function</strong></td>
<td>Master on PLB or OPB</td>
<td>APB Master Only</td>
</tr>
<tr>
<td><strong>Data Buses</strong></td>
<td>Separate Read and Write</td>
<td>Separate or 3-state</td>
</tr>
</tbody>
</table>
IBM CoreConnect Bus

**PLB**
- Pipelined
- Burst modes
- Split transactions
- Multiple masters

**OPB**
- Low bandwidth
- Burst mode
- Multiple Masters

**DCR**
- Low throughput
- 1 r/w = 2 cycles
- Ring type data bus
Processor Local Bus (PLB)

**High performance synchronous bus**

- Shared address, separate read and write data buses
- Support for 32-bit address, 16, 32, 64, & 128-bit data bus widths
- Dynamic bus sizing-byte, half-word, word, double-word transfers
- Up to 16 masters and any number of slaves
- AND-OR implementation structure
- Variable or fixed length (16-64 byte) burst transfers
- Pipelined transfers
- SPLIT transfer support
- Overlapped read and write transfers (up to 2 transfers per cycle)
- Centralized arbiter
- Locked transfer support for atomic accesses
# PLB Transfer Phases

<table>
<thead>
<tr>
<th>Address cycle</th>
<th>Request phase</th>
<th>Transfer phase</th>
<th>Address-acknowledge phase</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Data cycle</th>
<th>Transfer phase</th>
<th>Data-acknowledge phase</th>
</tr>
</thead>
</table>

Address and data phases are decoupled
Overlapped PLB Transfers

PLB allows address and data buses to have different masters at the same time
PLB Arbiter

Bus Control Unit
- each master drives a 2-bit signal that encodes 4 priority levels
- in case of a tie, arbiter uses static or RR scheme

Timer
- pre-empts long burst masters
- ensures high priority requests served with low latency
On-chip Peripheral Bus (OPB)

Synchronous bus to connect low performance peripherals and reduce capacitive loading on PLB.

- Shared address bus, multiple data buses.
- Up to a 64-bit address bus width.
- 32- or 64-bit read, write data bus width support.
- Support for multiple masters.
- Bus parking (or locking) for reduced transfer latency.
- Sequential address transfers (burst mode).
- Dynamic bus sizing—byte, half-word, word, double-word transfers.
- MUX-based (or AND–OR) structural implementation.
- Single cycle data transfer between OPB masters and slaves.
- Timeout capability for low-latency for important xfers.
Device Control Register (DCR) Bus

Low speed synchronous bus, used for on-chip device configuration purposes

- meant to off-load the PLB from lower performance status and control read and write transfers
- 10-bit, up to 32-bit address bus
- 32-bit read and write data buses
- 4-cycle minimum read or write transfers
- Slave bus timeout inhibit capability
- Multi-master arbitration
- Privileged and non-privileged transfers
- Daisy-chain (serial) or distributed-OR (parallel) bus topologies
Avalon Bus-based SoC

The diagram illustrates the interconnection of Nios CPU, DMA Controller, Avalon Bus Module, UART, PIO, etc., SRAM, Flash, and Ethernet MAC using Avalon Bus. The diagram shows the flow of data between these components.
Avalon Bus

• Avalon bus is **an active**, on-chip bus architecture that accommodate the SOPC environment.
• The interface to peripherals is synchronous with the Avalon clock. **Therefore, no complex, asynchronous handshaking and acknowledge schemes are necessary.**
• Multiplexers (**not tri-state buffers**) inside the bus determine which signals drive which peripheral. Peripherals are never required to tri-state their outputs. **Even when the peripheral is deselected**
• The address, data and control signals use separate, dedicated ports. **It simplifies the design of peripherals as they don’t need to decode address and data bus cycles as well as disable its outputs when it is not selected.**
Avalon Bus Module Features

Data-Path Multiplexing - Multiplexers transfer data from the selected slave peripheral to the appropriate master peripheral.

Address Decoding - Produces chip-select signals for each peripheral.

Wait-State Generation

Dynamic Bus Sizing

Interrupt-Priority Assignment - When one or more slave peripherals generate interrupts.

Latent Transfer Capabilities

Streaming Read and Write Capabilities - The logic required to allow streaming transfers between master-slave pairs is contained inside the Avalon bus module.
Avalon Bus Module

The Avalon bus module (an Avalon bus) is a unit of active logic that takes the place of passive, metal bus lines on a physical PCB.
System with Master Modules

![Diagram showing a system with master modules](image-url)

- **Library Component**
- **automatically generated “arbitration” Module**
Multi-Master: Avalon Bus Arbitration

Slave (data memory) is shared by two masters (Nios CPU and DMA)
Slave Arbitrator

Avalon bus module contains one slave arbitrator for each shared slave port. Slave arbitrator performs the following.

- Defines control, address, and data paths from multiple master ports to the slave port and specifies the arbitration mechanism to use when multiple masters contend for a slave at the same time.

- At any given time, selects which master port has access to the slave port and forces all other contending masters (if any) to wait, based on the arbitration assignments.

- Controls the slave port, based on the address, data, and control signals presented by the currently selected master port.
Multi-Masters and Slaves

Simultaneous multi-master system that permits bus transfers between two masters and two slaves.

<table>
<thead>
<tr>
<th>Master Request Slave (MRS)</th>
<th>Multiplexer control that connects the wait and data signals from multiple slave ports to a single master port.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Select Granted (MSG)</td>
<td>Multiplexer control that connects the data and control signals from multiple master ports to a single slave port.</td>
</tr>
<tr>
<td>Wait</td>
<td>Input to each master port that indicates that the bus transfer should be held when the desired slave port cannot be accessed immediately.</td>
</tr>
</tbody>
</table>
Standard Bus Architectures

• AMBA 2.0, 3.0 (ARM)
• CoreConnect (IBM)
• Avalon (Altera)
• STBus (STMicroelectronics)
• Sonics Smart Interconnect (Sonics)
• Wishbone (Opencores)
• PI Bus (OMI)
• MARBLE (Univ. of Manchester)
• CoreFrame (PalmChip)
• ...
STBus

- Consists of 3 synchronous bus-based interconnect specifications
  - Type 1
    - Simplest protocol meant for peripheral access
  - Type 2
    - More complex protocol
    - Pipelined, SPLIT transactions
  - Type 3
    - Most advanced protocol
    - OO (out-of-order) transactions, transaction labeling/hints
Type 1 and 3

Type 1

- Simple handshake mechanism
- 32-bit address bus
- Data bus sizes of 8, 16, 32, 64 bits
- Similar to IBM CoreConnect DCR bus

Type 3

- transaction completion
- Requires only single response/ACK Supports all Type 2 functionality
- OO (out-of-order) for multiple data transfers (burst mode)
Type 2

- Supports all Type 1 functionality
- Pipelined transfers
- SPLIT transactions
- Data bus sizes up to 256 bits
- Compound operations
  - READMODWRITE: Returns read data and locks slave till same master writes to location
  - SWAP: Exchanges data value between master and slave
  - FLUSH/PURGE: Ensure coherence between local and main memory
  - USER: Reserved for user defined operations
STBus

All types have

- MUX-based implementation
- Shared, partial or full crossbar implementation
STBus Arbitration

• Static priority
  ▪ Non-preemptive

• Programmable priority

• Latency based
  ▪ Each master has register with max. allowed latency (clock cycles)
  ▪ If value is 0, Each master also has counter loaded with max. latency value when master makes request
  ▪ Master counters are decremented at every subsequent cycle
  ▪ Arbiter grants access to master with lowest counter value
  ▪ In case of a tie, static priority is used
    • Higher priority master must be granted bus access as soon as it requests it.
STBus Arbitration

• **Bandwidth based**
  - Similar to TDMA/RR (Round Robin) scheme

• **STB**
  - Hybrid of latency based and programmable priority schemes
  - In normal mode, programmable priority scheme is used
  - Masters have max. latency registers, counters (latency based)
  - Each master also has an additional \textit{latency-counter-enable} bit
  - If this bit is set, and counter value is 0, master is in “panic state”
  - If one or more masters in panic state, programmable priority scheme is overridden, and panic state masters granted access

• **Message based**
  - Pre-emptive static priority scheme
Socket-based Interface Standards

Defines the interface of components

- Does not define bus architecture implementation
- Shield IP designer from knowledge of interconnection system, and enable same IP to be ported across different systems
- Requires Adaptor components to interface with implementation
Socket-based Interface Standards

• **Must be generic, comprehensive, and configurable**
  - to capture basic functionality and advanced features of a wide array of bus architecture implementations

• **Adaptor (or translational) logic component**
  - Must be created only once for each implementation (e.g. AMBA)
  - adds area, performance penalties, more design time
  - + enhances reuse, speeds up design time across many designs

• **Commonly used socket-based interface standards**
  - Open Core Protocol (OCP) ver 2.0
    - Most popular – used in Sonics Smart Interconnect
  - VSIA Virtual Component Interface (VCI)
    - Subset of OCP
OCP 2.0/3.0
Open Core Protocol

• Point-to-point synchronous interface
• Bus architecture independent
• Configurable data flow (address, data, control) signals for area-efficient implementation
• Configurable sideband signals to support additional communication requirements
• Pipelined transfer support
• Burst transfer support
• OO (out-of-order) transaction completion support
• Multiple threads
# OCP 3.0 Basic Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Width</th>
<th>Driver</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clk</td>
<td>1</td>
<td>varies</td>
<td>Clock input</td>
</tr>
<tr>
<td>EnableClk</td>
<td>1</td>
<td>varies</td>
<td>Enable OCP clock</td>
</tr>
<tr>
<td>MAddr</td>
<td>configurable</td>
<td>master</td>
<td>Transfer address</td>
</tr>
<tr>
<td>MCmd</td>
<td>3</td>
<td>master</td>
<td>Transfer command</td>
</tr>
<tr>
<td>MData</td>
<td>configurable</td>
<td>master</td>
<td>Write data</td>
</tr>
<tr>
<td>MDataValid</td>
<td>1</td>
<td>master</td>
<td>Write data valid</td>
</tr>
<tr>
<td>MRespAccept</td>
<td>1</td>
<td>master</td>
<td>Master accepts response</td>
</tr>
<tr>
<td>SCmdAccept</td>
<td>1</td>
<td>slave</td>
<td>Slave accepts transfer</td>
</tr>
<tr>
<td>SData</td>
<td>configurable</td>
<td>slave</td>
<td>Read data</td>
</tr>
<tr>
<td>SDataAccept</td>
<td>1</td>
<td>slave</td>
<td>Slave accepts write data</td>
</tr>
<tr>
<td>SResp</td>
<td>2</td>
<td>slave</td>
<td>Transfer response</td>
</tr>
</tbody>
</table>
# ComMand and Response Encoding

<table>
<thead>
<tr>
<th>MCmd[2:0]</th>
<th>Command</th>
<th>Mnemonic</th>
<th>Request Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>Idle</td>
<td>IDLE</td>
<td>(none)</td>
</tr>
<tr>
<td>0 0 1</td>
<td>Write</td>
<td>WR</td>
<td>write</td>
</tr>
<tr>
<td>0 1 0</td>
<td>Read</td>
<td>RD</td>
<td>read</td>
</tr>
<tr>
<td>0 1 1</td>
<td>ReadEx</td>
<td>RDEX</td>
<td>read</td>
</tr>
<tr>
<td>1 0 0</td>
<td>ReadLinked</td>
<td>RDL</td>
<td>read</td>
</tr>
<tr>
<td>1 0 1</td>
<td>WriteNonPost</td>
<td>WRNP</td>
<td>write</td>
</tr>
<tr>
<td>1 1 0</td>
<td>WriteConditional</td>
<td>WRC</td>
<td>write</td>
</tr>
<tr>
<td>1 1 1</td>
<td>Broadcast</td>
<td>BCST</td>
<td>write</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SResp[1:0]</th>
<th>Response</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>No response</td>
<td>NULL</td>
</tr>
<tr>
<td>0 1</td>
<td>Data valid / accept</td>
<td>DVA</td>
</tr>
<tr>
<td>1 0</td>
<td>Request failed</td>
<td>FAIL</td>
</tr>
<tr>
<td>1 1</td>
<td>Response error</td>
<td>ERR</td>
</tr>
</tbody>
</table>
OCP 2.0 Signal Details

• **Dataflow**
  ◦ Basic signals
  ◦ Simple extensions
    • e.g. byte enables, data byte parity, error correction codes, etc.
  ◦ Burst extensions
    • e.g. length, type (WRAP/INCR), pack/unpack, ACK requirements etc.
  ◦ Tag extensions
    • Assign IDs to transactions for reordering support
  ◦ Thread extensions
    • Assign IDs to threads for multi-threading support

• **Sideband (optional)**
  ◦ Not part of the dataflow process
  ◦ Convey control and status information such as reset, interrupt, error, and core-specific flags

• **Test (optional)**
  ◦ add support for scan, clock control, and IEEE 1149.1 (JTAG)
OCP 2.0 Protocol Hierarchy

- Data flow signals combined into groups of request signals, response signals and data handshake signals
- Groups map one-on-one to their corresponding protocol phases (request, response, handshaking)
- Different combinations of protocol phases are used by different types of transfers (e.g. ‘single request/multiple data burst’)
- Burst transactions are comprised of a set of transfers linked together having a defined address sequence and no. of transfers
Example: SoC with Mixed Profiles

[Diagram showing interconnection between CPU, MPEG2 decoder, DMA, OCP-Based interconnect, CPU bus subsystem, UART, USB, PCI, and DRAM controller.]
Summary

• Standards important for seamless integration of SoC IPs
  ▪ avoid costly integration mismatches
• Two categories of standards for SoC communication:
  ▪ Standard bus architectures
    • define interface between IPs and bus architecture
    • define (at least some) specifics of bus architecture that implements data transfer protocol
    • e.g. AMBA 2.0/3.0, IBM Coreconnect, Avalon, STBus, Sonics, Smart Interconnect,
  ▪ Socket based bus interface standards (e.g. OCP 2.0
    • define interface between IPs and bus architecture
    • do not define bus architecture implementation specifics