System Busses / Networks-on-Chip

EECE 579 - Advanced Topics in VLSI Design
Spring 2009
Brad Quinton
Outline

1. Simple systems busses
   - Overview
   - AMBA APB
   - Advantages/Limitations

2. Complex systems busses
   - Overview
   - AMBA AHB
   - Advantages/Limitations

3. Networks-on-Chip (NoC)
   - Overview
   - AMBA AXI
   - Research Topics: Topology, Protocol, VLSI Implementation...
Simple System Busses
Simple System Busses

• The primary goal of a simple system bus is to allow *software* (running on a processor) to *communicate* with other *hardware* in the SoC.

• There are many different implementations ... but they are all *very similar*.
Embedded Processor I/O

- RISC-based embedded processors communicate with external hardware using two simple instructions:
Embedded Processor I/O

- RISC-based embedded processors communicate with external hardware using two simple instructions:
  - **Load Operation**: Copies a word of data from a specific address to a *local register*
  - **Store Operation**: Copies a word of data from a *local register* to a specific address
Embedded Processor I/O

- RISC-based embedded processors communicate with external hardware using two simple instructions:
  - **Load Operation:** Copies a word of data from a specific address to a *local register*
  - **Store Operation:** Copies a word of data from a *local register* to a specific address

- The simple system bus is just a direct extension of this model
Embedded Processor I/O
Embedded Processor I/O

Software sets up the register with the address and data ...
Embedded Processor I/O

Software sets up the register with the address and data ...

Blocks decode addresses to see if they are the targets...
Embedded Processor I/O

Software sets up the register with the address and data ...

Blocks decode addresses to see if they are the targets...

Data transferred between register and hardware
AMBA Specification

- **AMBA**: Advanced Microcontroller Bus Architecture

- Created by ARM to enable standardized interfaces to their embedded processors

- Actually three standards: APB, AHB, and AXI

- Very commonly used for commercial IP cores
AMBA Specification

- **AMBA**: Advanced Microcontroller Bus Architecture

- Created by ARM to enable standardized interfaces to their embedded processors

- Actually three standards: APB, AHB, and AXI

- Very commonly used for commercial IP cores
AMBA Specification

• **AMBA**: Advanced Microcontroller Bus Architecture

• Created by ARM to enable standardized interfaces to their embedded processors

  Simple Bus  Complex Bus

• Actually three standards **APB**, **AHB**, and **AXI**

• Very commonly used for commercial IP cores
AMBA Specification

- **AMBA**: Advanced Microcontroller Bus Architecture
- Created by ARM to enable standardized interfaces to their embedded processors
  - Simple Bus  Complex Bus  NoC
- Actually three standards **APB**, **AHB**, and **AXI**
- Very commonly used for commercial IP cores
AMBA APB: Read Operation
AMBA APB: Read Operation

Target Address

- **PADDR**
- **PWRITE**
- **PSEL**
- **PENABLE**
- **PRDATA**

Diagram showing the timing of read operation with labels for PADDR, PWRITE, PSEL, PENABLE, and PRDATA.
AMBA APB: Read Operation

Target Address

Transaction Type

Diagram showing timing signals for read operations.
AMBA APB: Read Operation

Target Address

Transaction Type

Address Decode
AMBA APB: Read Operation

Target Address

Transaction Type

Address Decode

Optional (for asynchronous implementations ...)

Diagram showing the timing of PADDR, PWRITE, PSEL, PENABLE, and PRDATA signals with time stamps T1 to T5.
AMBA APB: Read Operation

Target Address

Transaction Type

Address Decode

Optional (for asynchronous implementations ...)

Read Data
AMBA APB: Write Operation
AMBA APB: Write Operation

Common Signals Between Read and Write

T1 T2 T3 T4 T5

PCLK
PADDR Addr 1
PWRITE
PSEL
PENABLE
PWDATA Data 1
AMBA APB: Write Operation

Common Signals Between Read and Write

Write Data
Remember Our Case Study

Simple generic processor interface:

- data width: 16 bits
- address width: 16 bits
- read cycle time: 50 ns
- write cycle time: 50 ns
Remember Our Case Study

Simple generic processor interface:

- data width: 16 bits
- address width: 16 bits
- read cycle time: 50 ns
- write cycle time: 50 ns
Simple Bus Advantages

- Simple to implement
- Easy to understand
- Simple programming model
- Easy to add new hardware blocks
- Minimal hardware requirements (most of the signals are shared)
Simple Bus Limitations

- Single Master - limits parallelism
- Scalability - performance suffers as bus is loaded...
- Single outstanding request - poor throughput and multi-threading performance bottleneck
Case Study: Single Master

- Imagine a new partition:
  - *APS Bit Error Monitor* communicates directly with *Switch*

- Simple bus doesn’t work...
Case Study: Single Master

- Imagine a new partition:
  - APS Bit Error Monitor communicates directly with Switch

- Simple bus doesn’t work...
Case Study: Single Master

- Imagine a new partition:
  - *APS Bit Error Monitor* communicates directly with *Switch*

- Simple bus doesn’t work...

- This can make software the bottleneck in the system....
Single Master Summary

• A bus that is limited to a single master:
  – Makes inter-block communication inefficient
  – Limits parallelism between hardware and software
  – Increases reliance on interrupts
  – Creates software performance bottlenecks
  – Is not compatible with multiple processors
Scalability
Scalability

Blocks are functionally easy to add, but....
Scalability

Blocks are functionally easy to add, but....

Each new block increases the delay on the address and data.
Scalability Summary

• Simple busses are not scaleable because:
  – The address and data “fan-out” to each target
  – Adding a new block increases the load on the bus
  – Increased fanout + greater load = reduce performance
Single Outstanding Request
Single Outstanding Request

Processor is stalled waiting for response...

![Diagram showing signal interactions](image-url)
Single Outstanding Request

Processor is stalled waiting for response...

best-case <= 50% efficiency
Single Outstanding Request Summary

• Busses limited to a single outstanding request:
  
  – Reduce software performance since the software must “stall” on the first transaction
  
  – Are not able to achieve full bus throughput since the data bus is idle during the address phase
Complex System Busses
Complex Systems Busses

• The complex system bus is attempts to address some of the issues with the simple bus:
  – Multi-master
  – Pipelined transactions

• There are many different ways to go about this...
AMBA AHB

- AHB addresses many of the limitations of APB:
  - multi-master
  - multiple outstanding transactions (sort of...)
  - back-to-back transactions

- Unfortunately, this adds **significant complexity**
Bring on the complexity...
Bring on the complexity...

CPU #1

CPU #2

IP Block #1

IP Block #2

IP Block #3

IP Block #4
Bring on the complexity...

Request

CPU #1

CPU #2

IP Block #1

IP Block #2

IP Block #3

IP Block #4
Bring on the complexity...

CPU #1

CPU #2

IP Block #1

IP Block #2

IP Block #3

IP Block #4
Bring on the complexity...
Bus Arbitration

• When multiple masters share a bus there must be some central resource to manage the bus: an **arbiter**

• Once there is competition for the bus, it is possible that it is not ready when you need it: **backpressure**

• Backpressure adds complexity and hurt performance
Request / Grant Protocol
Request / Grant Protocol

Before a transaction a master makes a request to the central arbiter
Request / Grant Protocol

Before a transaction a master makes a request to the central arbiter. Eventually the request is granted.
Request / Grant Protocol

Before a transaction a master makes a request to the central arbiter.

Eventually the request is granted.

Then the transaction proceeds.
Request / Grant Protocol

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

Performance Impact
Pipelined Transactions

• To help improve **bus efficiency** the transactions on the bus can be pipelined

• This is really a simple implementation of **multiple outstanding transactions**

• The address for one transaction can be presented before the data from the **previous transaction** has been completed
Pipelined Transactions
Pipelined Transactions

Transaction A Starts
Pipelined Transactions

Transaction A Starts

Transaction B Starts
Pipelined Transactions

Transaction A Starts

Transaction A Completes

Transaction B Starts
Pipelined Transactions

Notice backpressure

Transaction A Starts
Transaction B Starts
Transaction A Completes
Advantages

- Relatively easy to add new blocks
- Still has the familiar bus structure
- Low hardware cost
- Bus arbitration “solves” many ordering problems
Disadvantages

• Busses that require arbitration:
  – must route signals to the arbitration logic and back
  – must find a “fair” way to share the bus
  – slaves are not always available => backpressure
  – difficult to provide performance guarantees...

• Still potentially a bandwidth bottleneck

• Still doesn’t scale well when blocks are added

• Multiple outstanding transactions not handled well - no ordering information
Networks-on-Chip (NoCs)
Networks-on-Chip

- It is clear that even with significant design effort the bus-style interconnect is not going to sufficient for large SoCs:
  - the physical implementation does not scale: bus fanout, loading, arbitration depth all reduce operating frequency
  - the available bandwidth does not scale: the single bus must be shared by all masters and slaves
Networks-on-Chip

- It is clear that even with significant design effort the bus-style interconnect is not going to sufficient for large SoCs:
  - the **physical implementation** does not scale: bus fanout, loading, arbitration depth all reduce operating frequency
  - the available **bandwidth** does not scale: the single bus must be shared by all masters and slaves

- **Lets start again:** Leverage research from data networking
What do we want?

• The SoCs of the future will:
  – have 100s of hardware blocks,
  – have billions of transistors,
  – have multiple processors,
  – have large wire-to-gate delay ratios,
  – handle large amounts of high-speed data,
  – need to support “plug-and-play” IP blocks

• Our NoC needs to be ready for these SoCs...
The Ideal Network

• What would the ideal network look like?:
  
  – Low area overhead
  – Simple implementation
  – High-speed operation
  – Low-latency
  – High-bandwidth
  – Operate at a constant frequency even with additional blocks
  – Increase available bandwidth as blocks are added
  – Provide performance guarantees
  – Have a “universal” interface
The Ideal Network

• What would the ideal network look like?:

  – Low area overhead
  – Simple implementation
  – High-speed operation
  – Low-latency
  – High-bandwidth
  – Operate at a constant frequency even with additional blocks
  – Increase available bandwidth as blocks are added
  – Provide performance guarantees
  – Have a “universal” interface

These are competing requirements: Design a network that is the “best” fit.
What do we need to decide?

- Network Interface
- Network Protocol / Transaction Format
- Network Topology
- VLSI Implementation
Network Interface

• We want our network to be “plug-and-play” so **industry standardization** is key

• However the standard be **universal** enough to address many different needs

• AMBA AXI is an example of an attempt at this
AMBA AXI

- ARM added the AXI specification to Version 3.0 of the AMBA standard.
- **New approach:** define the interface and leave the interconnect up to the designers.
- Good plan since a specific bus implementation is no longer required.
- It is possible to use AXI to build many different NoCs.
AMBA AXI

• Interface divided into 5 channels:
  – Write Address
  – Write Data
  – Write Response
  – Read Address
  – Read Data/Response

• Each channel is independent and use two-way flow control
AMBA AXI Read Channels
AMBA AXI Read Channels

Independent
AMBA AXI Read Channels

Give me some data

Independent
AMBA AXI Read Channels

Give me some data

Independent

Here you go
AMBA AXI Read Channels

channels synchronized with ID # or “tags”

Independent

Give me some data

Here you go
AMBA AXI Write Channels
AMBA AXI Write Channels
AMBA AXI Write Channels

I’m sending data. Please store it.
I’m sending data. Please store it.

Here is the data.

AMBA AXI Write Channels
AMBA AXI Write Channels

I’m sending data. Please store it.

Here is the data.

I received that data correctly.
AMBA AXI Write Channels

I’m sending data. Please store it.

Here is the data.

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 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 Flow-Control

- This definition of very independent, fully flow-controlled channels is very useful

- However, there is a potential problem:
AMBA AXI Flow-Control

• This definition of very independent, fully flow-controlled channels is very useful

• However, there is a potential problem: DEADLOCK
AMBA AXI Flow-Control

• This definition of very independent, fully flow-controlled channels is very useful

• However, there is a potential problem: DEADLOCK

• On a write transaction the master **must not wait** for AWREADY before asserting WVALID
AMBA AXI Read
AMBA AXI Read

Read Address Channel

Read Data Channel
AMBA AXI Write
AMBA AXI Write

Write Address Channel

Write Data Channel

Write Response Channel
A True Interface Specification

- Because of the **channel independence** and the two-way **flow-control** the interface does not dictate the network protocol, transaction format, network topology, or VLSI implementation.

- For example:
  - if you want to build a packet-based network, you can “backpressure” the data channel while you build the packet header from the address channel information,
  - you can use store-and-forward, or cut-through,
  - etc.
Network Protocol / Transaction Format

- There are many choices for network protocols and transactions formats:
  - **circuit-switched**: plan and provision a connection before communication starts
  - **packet-switched**: issues packets which compete for network resources
  - **hybrids**: schedule connectivity (dynamic or static)
Network Protocol / Transaction Format

• There are many choice for network protocols and transactions formats:
  
  – **circuit-switched**: plan and provision a connection before communication starts
  
  – **packet-switched**: issues packets which compete for network resources
  
  – **hybrids**: schedule connectivity (dynamic or static)

• There is still lots of research here....
Network Topology

• How should your network elements be interconnected:
  
  – **Fully Connected** ($N^2$): high area cost, high performance
  – **Mesh**: low area cost, potential poor performance
  – **Hypercube**: medium area, traffic dependent performance
  – **Fat-tree**: medium area, traffic dependent performance
  – **Torus**: medium area, traffic dependent performance
Network Topology

- There is lots of research here....
Network Topology - Caveat

- There has been a lot of research on topologies for NoCs, however it is important to realize that the performance of a topology is highly dependent on the traffic patterns!

- Traffic patterns in an SoC that you are designing yourself are NOT random, therefore much of the topology research is not applicable to most SoCs!
VLSI Implementation

• Once you have a topology there is still the mater of implementing it on your SoC

• There are many considerations:
  
  – **Clocking**: Synchronous, Asynchronous
  
  – **Buffer Insertion**: Trade-off power, area, performance
  
  – **Register Insertion / Pipelining**: Trade-off clock frequency, area, and latency
  
  – **Packet Buffers**: Trade-off area, latency and throughput

• Again, lots of research on-going...
Bluetooth “Platform” SoC

Processor

ARM7TDMI

ARBITER

DECODER

TIC

Bluetooth SoC

MCU

SPEECH I/F

RADIO I/F

SHARED MEMORY CONTROLLER

ADC

PLL CLOCKS

PHASE LOCKED LOOP

AHB

APB

DMA

POWER & CLOCK CONTROL

SHARED MEMORY

BRIDGE

SMC

Watchdog

GPIO

PIC

TIMERS

UART

Low-speed I/O and Support Logic

System Bus / Hardware I/F

Application Specific Logic

Watchdog

GPIO

PIC

TIMERS

UART

UART

ACI USB
Research Paper

• Lets look at: