Topic 10
Bus Architecture & Interconnects

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

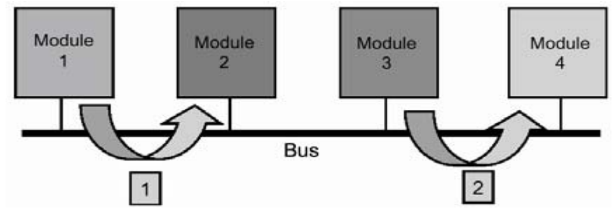
URL: www.ee.imperial.ac.uk/pcheung/
E-mail: p.cheung@imperial.ac.uk

Introductions & Sources

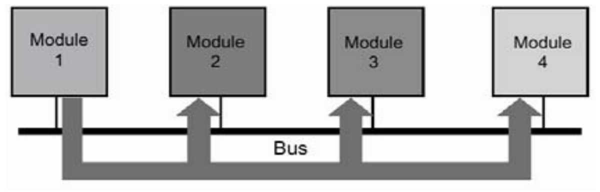
- We will consider a number of issues related to bus architectures in digital systems.
- Useful references:
  - "Bus Architecture of a System on a Chip with User-Configurable System Logic", Steven Winegarde*n, IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 35, NO. 3, MARCH 2000, p425-433.*
  - *"AMBA: ENABLING REUSABLE ON-CHIP DESIGNS", David Flynn, IEEE Micro, July/August 1997.*
  - *AMBA™ Specification (Rev 2.0), ARM Ltd., 1999*
  - *The CoreConnect Bus Architecture, IBM, http://www.chips.ibm.com/products/coreconnect*
  - *VSI Alliance Architecture Document, version 1.0, 1997.*
  - Draft Chapter, *"System-on-Chip",* Flynn & Luk

Basic concepts: Bus basics: order and broadcast properties

- Communications on buses must be in strict order: serial nature of bus



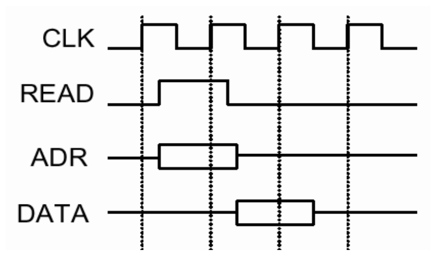
- It can broadcast a transaction – sending to multiple components simultaneously

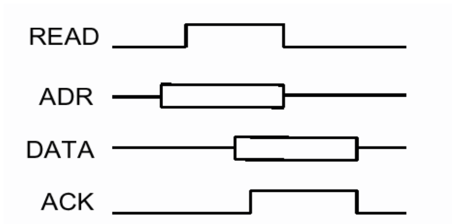


Basic concepts: Cycles, messages and transactions

- Buses operate in units of *cycle*s, *Messages* and *transaction*s.
  - *Cycle*s: A message requires a number of clock cycles to be sent from sender to receiver over the bus.
  - *Message*: These are logical unit of information. For example, a write message contains an address, control signals and the write data.
  - *Transactio*n: A transaction consists of a sequence of messages which together form a transaction. For example, a memory read requires a memory read message and a reply with the requested data.

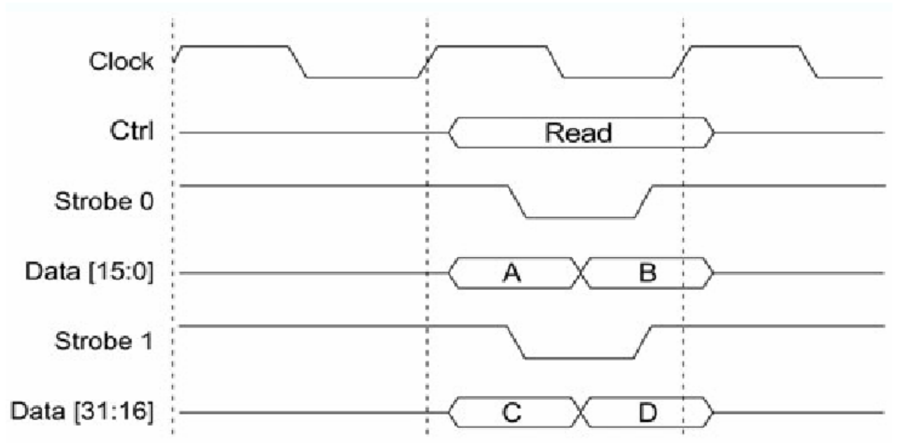
Synchronous vs Asynchronous





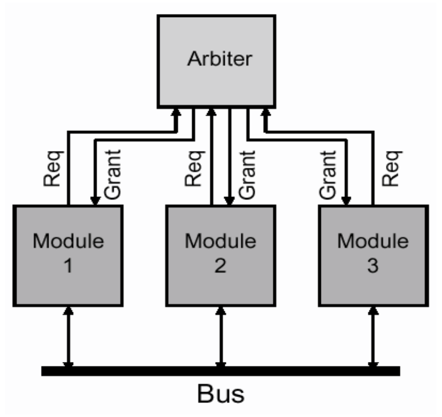
1. Master puts address on bus and asserts READ when address is stable
2. Memory puts data on bus and asserts ACK when data is stable
3. Master deasserts READ when data is read
4. Memory deasserts ACK

Basic concepts: Typical Source Synchronous Data Transfer



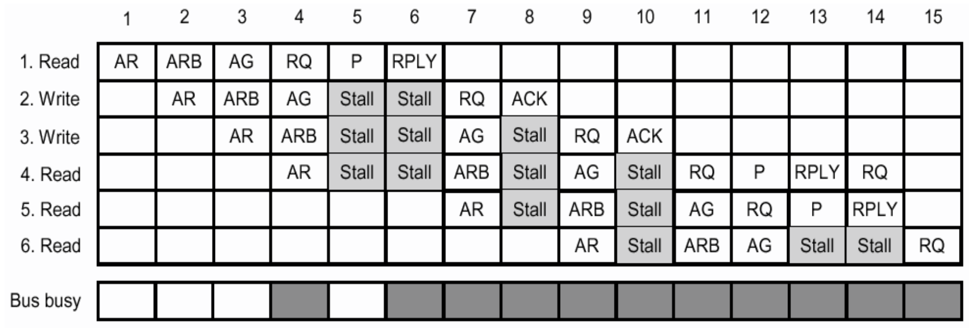
Basic concepts: Bus arbitration

- Only one bus master can control the us.
- Need some way of deciding who is master – may use a bus arbiter:



Basic concepts: Bus pipelining

- A transaction may take multiple cycles
- Overlap multiple transaction through pipelining:

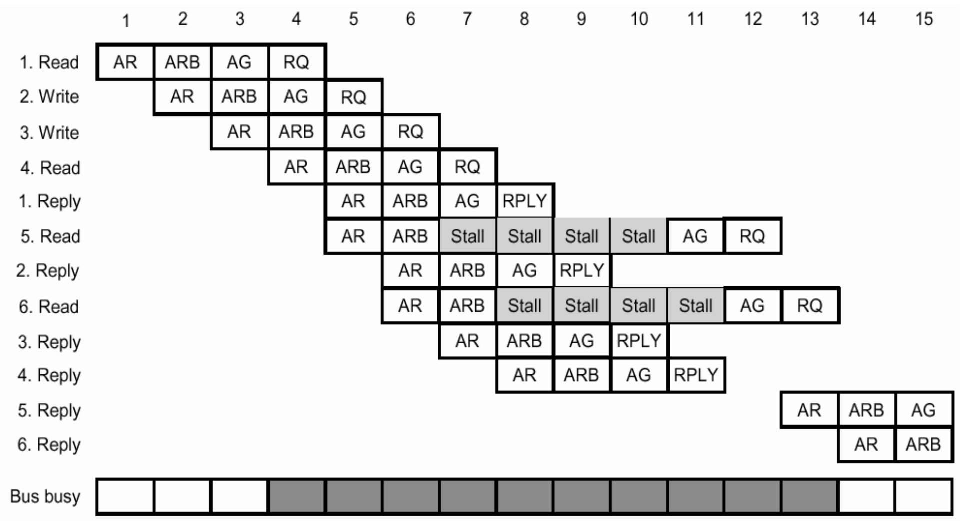


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1. Read	AR	ARB	AG	RQ	P	RPLY									
2. Write		AR	ARB	AG	Stall	Stall	RQ	ACK							
3. Write			AR	ARB	Stall	Stall	AG	Stall	RQ	ACK					
4. Read				AR	Stall	Stall	ARB	Stall	AG	Stall	RQ	P	RPLY	RQ	
5. Read							AR	Stall	ARB	Stall	AG	RQ	P	RPLY	
6. Read									AR	Stall	ARB	AG	Stall	Stall	RQ
Bus busy				▨		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

Basic concepts: Split-transaction bus

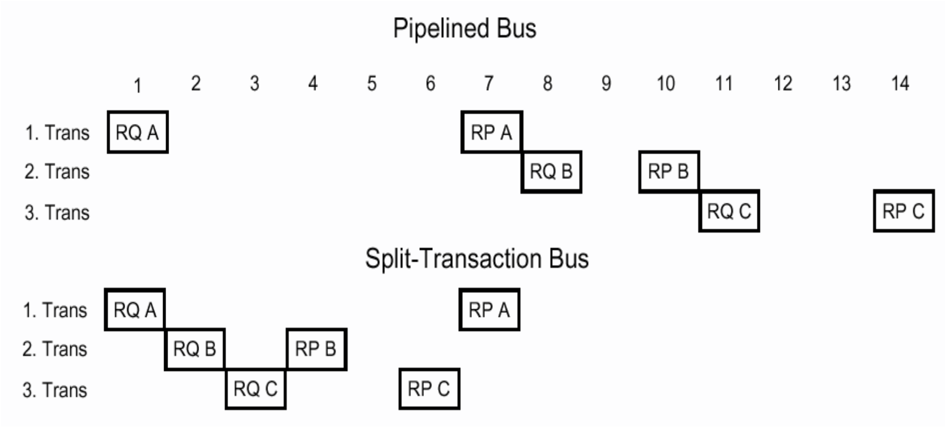
- A bus transaction can be divided into two or more phases, e.g.
  - "Request" phase
  - "Reply" phase
- These can be split into two separate sub-transactions, which may or may not happen consecutively. If split, these must compete for the bus by arbitration.

Basic concepts: Split-transaction bus



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1. Read	AR	ARB	AG	RQ											
2. Write		AR	ARB	AG	RQ										
3. Write			AR	ARB	AG	RQ									
4. Read				AR	ARB	AG	RQ								
1. Reply					AR	ARB	AG	RPLY							
5. Read					AR	ARB	Stall	Stall	Stall	Stall	AG	RQ			
2. Reply						AR	ARB	AG	RPLY						
6. Read						AR	ARB	Stall	Stall	Stall	Stall	AG	RQ		
3. Reply							AR	ARB	AG	RPLY					
4. Reply								AR	ARB	AG	RPLY				
5. Reply													AR	ARB	AG
6. Reply														AR	ARB
Bus busy				▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		

Basic concepts: Pipelined only bus vs split-transaction bus



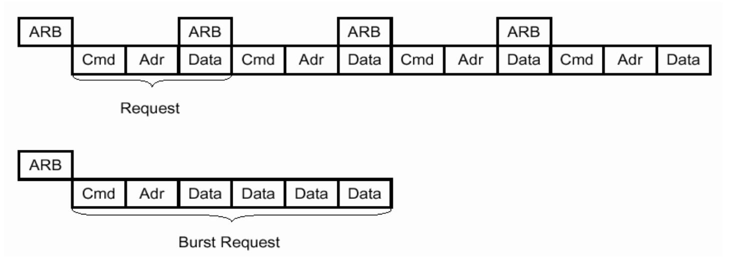
Pipelined Bus

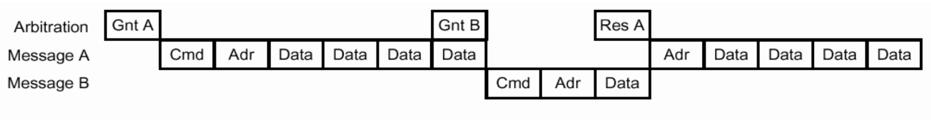
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. Trans	RQ A						RP A							
2. Trans								RQ B		RP B				
3. Trans											RQ C			RP C

Split-Transaction Bus

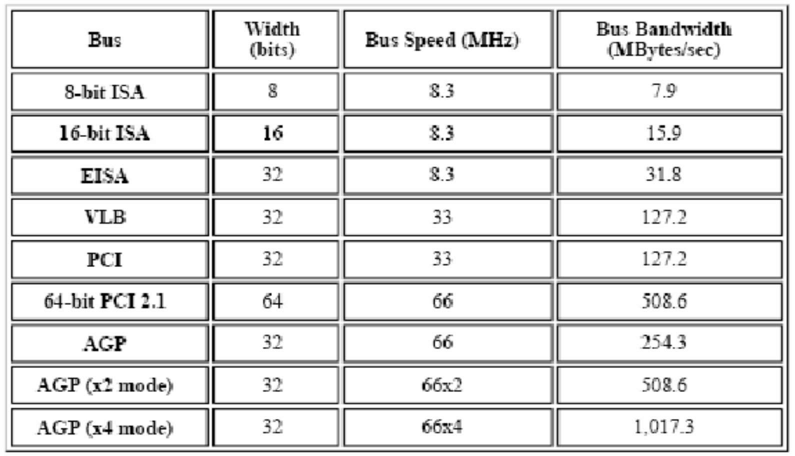
	1	2	3	4	5	6	7
1. Trans	RQ A						RP A
2. Trans		RQ B		RP B			
3. Trans			RQ C			RP C	

Basic concepts: Burst transfer mode



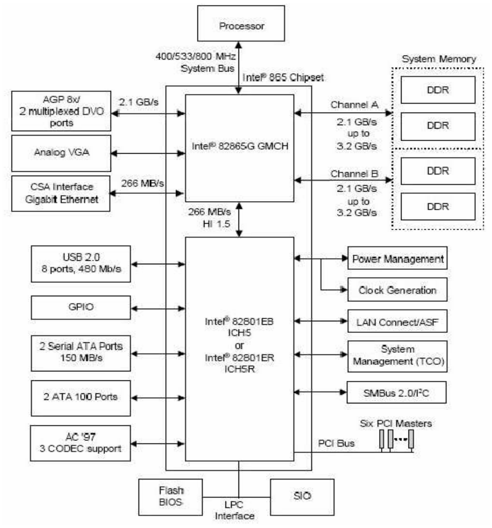


Bus bandwidth



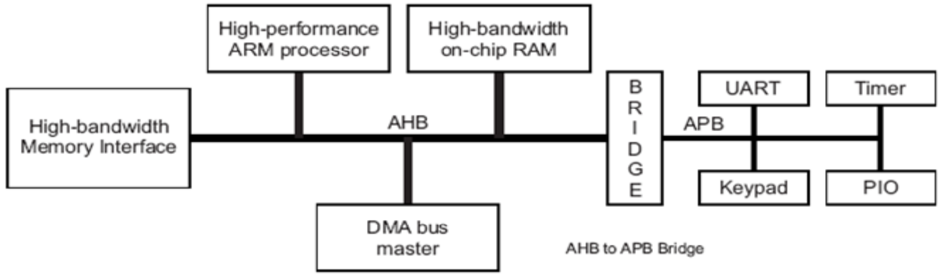
Bus	Width (bits)	Bus Speed (MHz)	Bus Bandwidth (MBytes/sec)
8-bit ISA	8	8.3	7.9
16-bit ISA	16	8.3	15.9
EISA	32	8.3	31.8
VLB	32	33	127.2
PCI	32	33	127.2
64-bit PCI 2.1	64	66	508.6
AGP	32	66	254.3
AGP (x2 mode)	32	66x2	508.6
AGP (x4 mode)	32	66x4	1,017.3

Bus hierarchy



AMBA bus

- Based around ARM processor
  - AHB – Advanced High-Performance Bus
    - ➤ Pipelining of Address / Data
    - ➤ Split Transactions
    - ➤ Multiple Masters
  - APB – Advanced Peripheral Bus
    - ➤ Low Power / Bandwidth Peripheral Bus

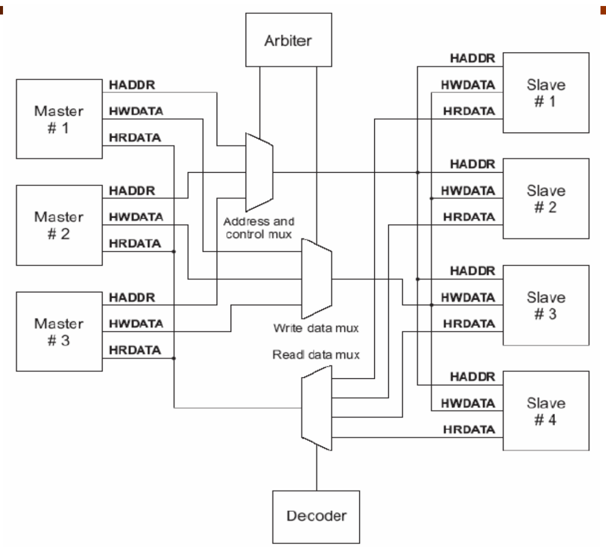

High-performance ARM processor
High-bandwidth on-chip RAM
High-bandwidth Memory Interface
AHB
DMA bus master
BRIDGE
APB
UART
Timer
Keypad
PIO
AHB to APB Bridge

AMBA Bus Design Goals

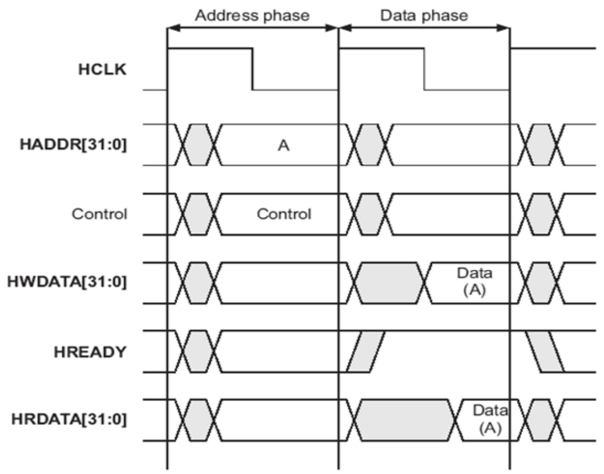
- Encourages modular design and design reuse
- Well defined interface protocol, clocking and reset
- Low-power support (helped by two-level partitioning)
- On-chip test access – built-in structure for testing modules connected on the bus

- Transactions on AHB
  - Bus master obtain access to the bus
  - Bus master initiates transfer
  - Bus slave provides response

AMBA bus arbitration

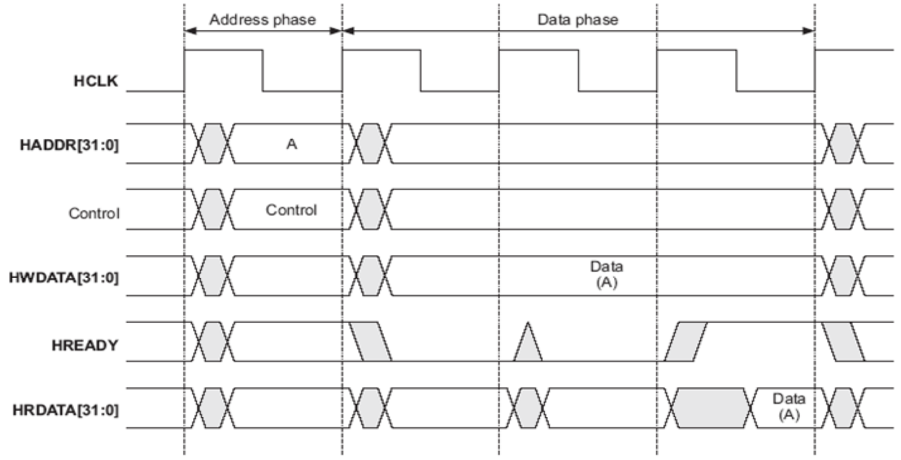


Simple AHB Transfer



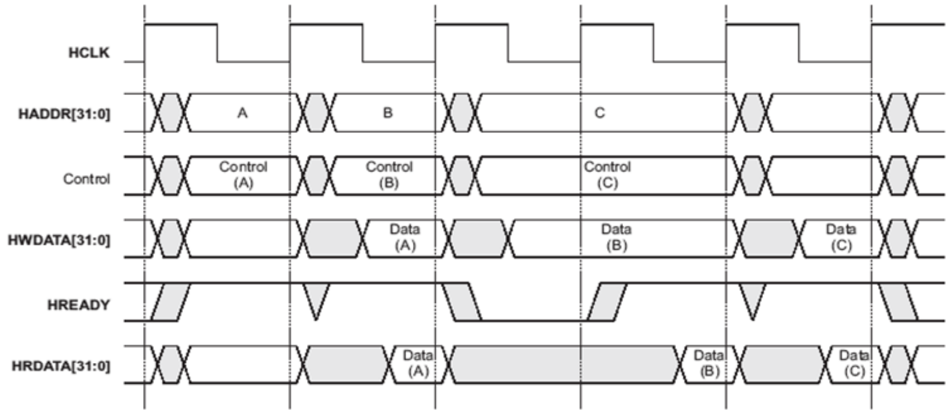
a) transfer without wait state

AHB Transfer with wait states

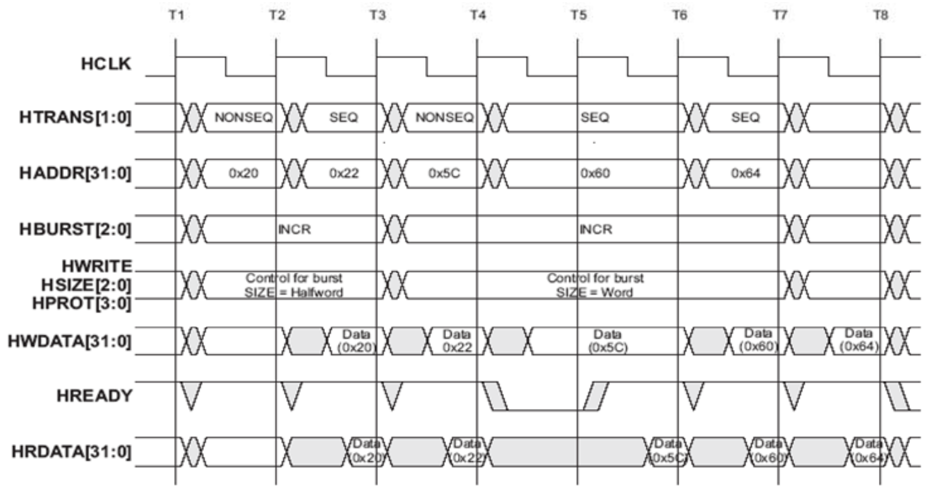


a) transfer with wait states

Multiple transfers with Pipelining



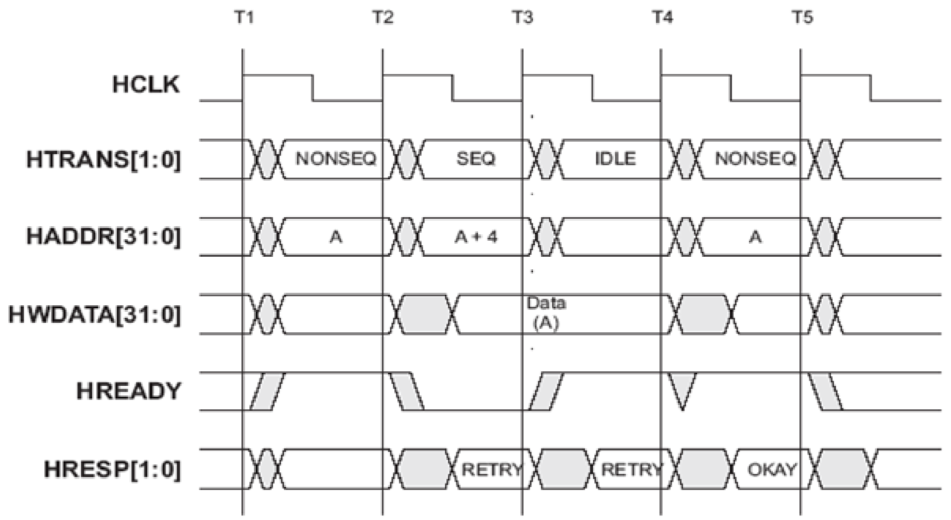
Burst mode transfer (undefined length)



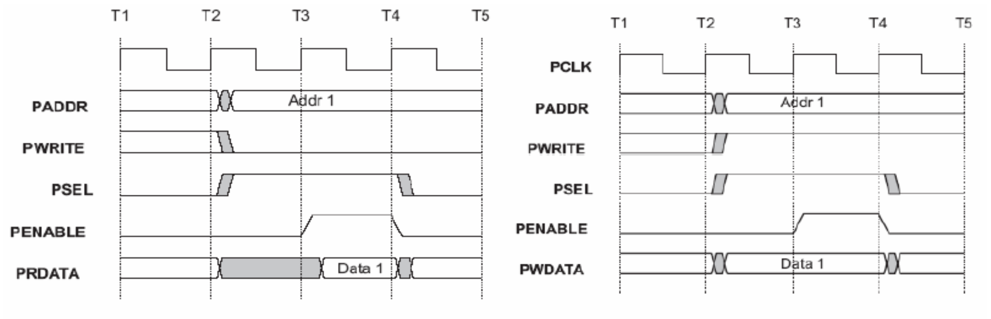
Slave Transfer Responses

- 1. Complete transfer immediately (single cycle transfer)
- 2. Insert one or more wait states to allow completion
- 3. Signal error to indicate transfer failed
- 4. Back of from the bus, try later (RE-TRY or SPLIT responses)

Retry Respnses on the AHB bus

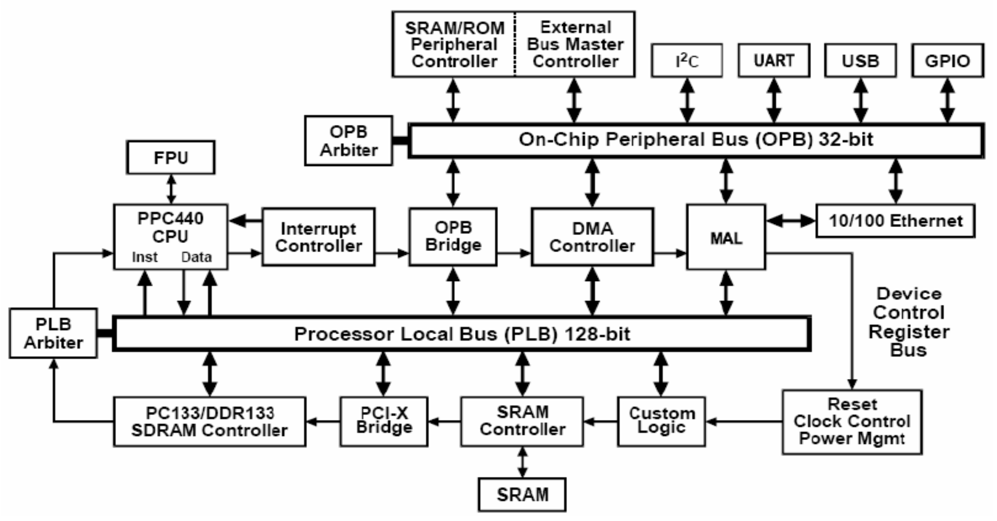


Advanced Peripheral Bus (APB)

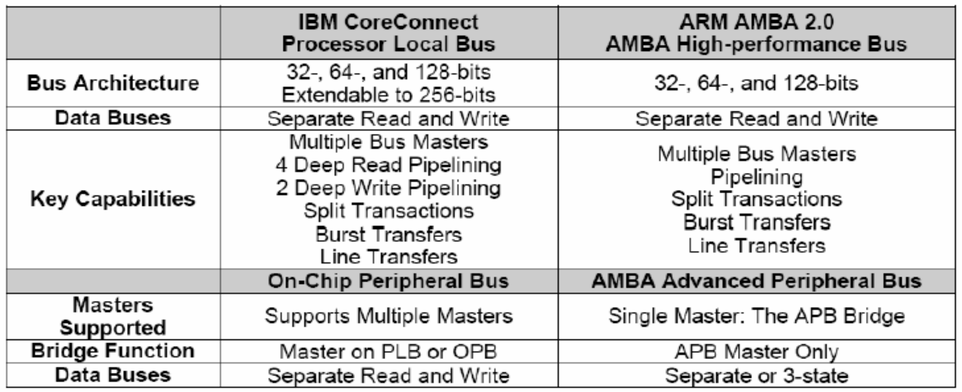


a) Read operation b) Write operation

IBM CoreConnect Bus


SRAM/ROM Peripheral Controller
External Bus Master Controller
I²C
UART
USB
GPIO
OPB Arbiter
On-Chip Peripheral Bus (OPB) 32-bit
FPU
PPC440 CPU
Inst
Data
Interrupt Controller
OPB Bridge
DMA Controller
MAL
10/100 Ethernet
Device Control Register Bus
PLB Arbiter
Processor Local Bus (PLB) 128-bit
PC133/DDR133 SDRAM Controller
PCI-X Bridge
SRAM Controller
Custom Logic
Reset Clock Control Power Mgmt
SRAM

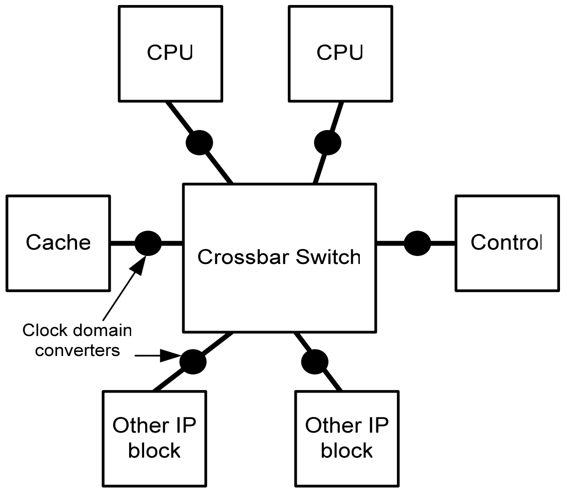
CoreConnect vs AMBA



	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

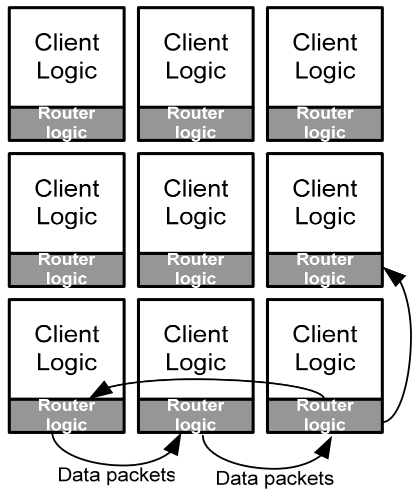
Crossbar Switch Approach

- Uses asynchronous channels
- Different modules can run at different clock frequency
- Globally Asychronous, Locally Synchronous (GALS) system



Network-on-chip approach

- Array of tiles
- Each tile contains client logic and router logic
- 2-D mesh topology
- Uses data packets, not wires, for communication
- Predictable delay, and noise



Chapter 5

Interconnect Architectures

5.1 Introduction

SOC designs usually involve the integration of intellectual property (IP) cores, each separately designed and verified. System integrators can maximize the re-use of design efforts by providing a common backbone for SOC modules, to reduce costs and to lower risks. Frequently the most important issue confronting an SOC integrator is the method by which the IP cores are connected together.

SOC interconnect alternatives extend well beyond conventional computer buses. This chapter provides an overview of three SOC interconnect architectures: bus, switch, and Network-on-Chip. A number of bus architectures developed specifically for SOC technology are then described and compared. Switch-based alternatives to bus based interconnects are considered, especially recent trends in SOC interconnects such as Network-on-Chip technology.

As we shall see, Network-on-Chip (NOC) is more than an alternative to bus or switch technology. It is often described at a higher level of abstraction, hiding the underlying physical interconnects from the designer. It is usually implemented by switch technology, although in principle it can also be a bus. To avoid confusion, we follow current SOC usage and refer to interconnect as a bus (usually no confusion), as a switch when there is a crossbar visible to the designer, and as a NOC when implemented by a switch. In the NOC case the switch is often more than a crossbar and could consist of a distributed interconnect or a multistage switching network.

There is a great deal of bus and computer interconnect literature. The units being connected are sometimes referred to as agents (in buses) or nodes (in general interconnect literature); we simply use the term *units*. Given that, at least at the current time, SOCs only require a small number of units to be interconnected, the chapter provides a simplified view of the interconnect alternatives.

5.2 Overview: Interconnect Architectures

Figure 5.1 depicts a system which includes a SOC module. The SOC module typically contains a number of IP blocks, one or more of which are processors. In addition, there

are various types of on-chip memory serving as cache, data or instruction storage. Other IP blocks serving application-specific functions, such as graphics processors, video codecs and network control units, are integrated in the SOC.

The IP blocks in the SOC module need to communicate with each other. External to the SOC module are off-chip memories, off-chip peripheral devices and mass storage devices. The cost and performance of the system therefore depends on both on-chip and off-chip interconnect structure.

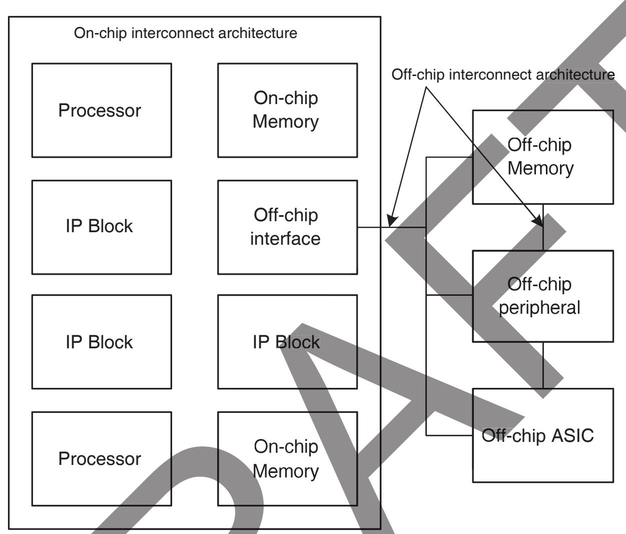


Figure 5.1 A simplified block diagram of an SOC module in a system context.

Choosing a suitable interconnect architecture requires the understanding of a number of system level issues and specifications. These are:

1. **Communication bandwidth:** the rate of information transfer between a module and the surrounding environment in which it operates. Usually measured in bytes/second, the bandwidth requirement of a module dictates to a large extent the type of interconnection required in order to achieve the overall system throughput specification.

2. **Communication latency:** the time delay between a module requesting data and receiving a response to the request. Latency may or may not be important in terms of overall system performance. For example, long latency in a video streaming application usually has little or no effect on the user's experience. Watching a movie that is a couple of seconds later than when it is actually broadcast is of no consequence. In contrast, even a short latency in a two-way mobile communication system can make it almost impossible to carry out a conversation.

3. **Master and slave:** these concern whether a unit can initiate or react to communication requests. A master, such as a processor, controls transactions between itself and other modules. A slave, such as memory, responds to requests from the master. A SOC design typically would have more than one master and numerous slaves.

4. **Concurrency requirement:** the number of independent simultaneous communication channels operating in parallel. Obviously the higher the concurrency, the higher the system throughput.

5. **Multiple clock domains:** different IP modules may operate at different clock and data rates. For example, a video camera captures pixel data at a rate governed by the video standard used, while a processor's clock rate is usually determined by the technology and architectural design. As a result, IP blocks inside an SOC often need to operate at different clock frequencies, creating separate timing regions known as clock domains. Crossing between clock domains can cause metastability, deadlock and synchronization problems.

Technology	AMBA	CoreConnect	Smart InterconnectIP	Nexus
Company	ARM	IBM	Sonics	Fulcrum
Core Type	Soft / Hard	Soft	Soft	Hard
Architecture	Multiple bus	Multiple bus with switch fabric	Multiple bus	switch fabric
Bus width	8–1024	32 / 64 / 128	16	8–128
Frequency	200MHz	100–400MHz	300MHz	1GHz
Max. BW	3GB/s	2.5–24GB/s	4.8GB/s	72GB/s
Min. latency	5ns	15ns	n/a	2ns

Table 5.1 Examples of interconnect architectures [26].

Given a set of communication specifications, a designer can explore the different bandwidth, latency, concurrency and clock domain requirements of different architectures such as bus, switch, and Network-on-Chip. Some examples of these are given in Table 5.1.

Designing the interconnect architecture for an SOC requires careful consideration of all the factors listed above. Figure 5.2 shows a brief outline for interconnect design. The rest of this section will provide an overview of three interconnect scheme, which will be explored in greater detail later in this chapter.

5.2.1 Bus

The performance of a computer system is heavily dependent on the characteristics of its interconnect architecture. A poorly designed system bus can throttle the transfer of instructions and data between memory and processor, or between peripheral devices and memory. This communication bottleneck is the focus of attention among many microprocessor and system manufacturers who, over the last three decades, have

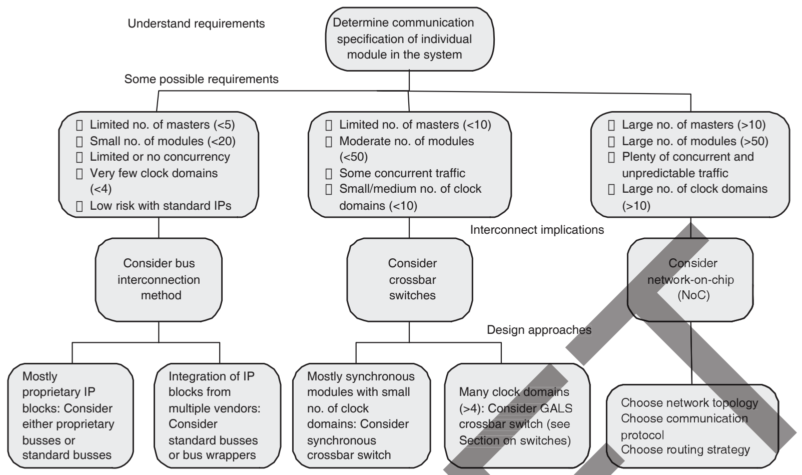

Understand requirements
Determine communication specification of individual module in the system
Some possible requirements
□ Limited no. of masters (<5)
□ Small no. of modules (<20)
□ Limited or no concurrency
□ Very few clock domains (<4)
□ Low risk with standard IPs
□ Limited no. of masters (<10)
□ Moderate no. of modules (<50)
□ Some concurrent traffic
□ Small/medium no. of clock domains (<10)
□ Large no. of masters (>10)
□ Large no. of modules (>50)
□ Plenty of concurrent and unpredictable traffic
□ Large no. of clock domains (>10)
Interconnect implications
Consider bus interconnection method
Consider crossbar switches
Consider network-on-chip (NoC)
Design approaches
Mostly proprietary IP blocks: Consider either proprietary busses or standard busses
Integration of IP blocks from multiple vendors: Consider standard busses or bus wrappers
Mostly synchronous modules with small no. of clock domains: Consider synchronous crossbar switch
Many clock domains (>4): Consider GALS crossbar switch (see Section on switches)
Choose network topology
Choose communication protocol
Choose routing strategy

Figure 5.2 An outline for interconnect design.

adopted a number of bus standards. These include the popular VME bus and the Intel Multibus-II. For personal computers, the evolution includes the ISA bus, the EISA bus and the now prevalent PCI and PCI Express (PCI-X) buses. All these bus standards are designed to connect together integrated circuits on a printed circuit board (PCB) or PCBs in a system.

While these bus standards have served the computing community well, they are not particularly suited for SOC technology. For example, all such system level buses are designed to drive a backplane, either in a rack-mounted system or on a computer motherboard. This imposes numerous constraints on the bus architecture. For a start, the number of signals available is generally restricted by the limited pin count on an IC package or the number of pins on the PCB connector. Adding an extra pin on a package or a connector is expensive. Furthermore, the speed at which the bus can operate is often limited by the high capacitive load on each bus signal, the resistance of the contacts on the connector, and the electromagnetic noise produced by such fast switching signals traveling down a PCB track.

Table 5.2 gives a comparison of a number of different bus interconnect architectures together with size and speed statistics for a typical bus slave.

Standard	Speed (MHz)	Area (**rbe**)
AMBA	100	172900
CoreConnect	80	158900

Table 5.2 Comparison of bus interconnect architectures [30].

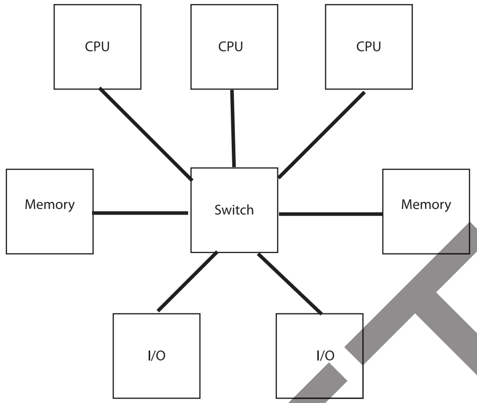


Figure 5.3 A switch-based interconnect scheme forming a Globally Asynchronous/Locally Synchronous (GALS) system [9].

5.2.2 Switch

While bus interconnect has been the predominant architecture for SOC interconnections, it suffers from a number of drawbacks. Even a well-designed bus-based system may suffer from data transfer bottlenecks, limiting the performance of the entire system. It is also not inherently scalable. As more and more modules are added to a bus, not only does data congestion increase, but power consumption also rises due to the increased load presented to the bus driver circuits. Switch based interconnection schemes have the potential to avoid some of these limitations.

A switch-based interconnection scheme uses either a centralized crossbar switch or a number of distributed switches [8] to connect together SOC modules. Apart from the advantage of avoiding traffic congestion, a switch-based scheme may allow modules to operate at different clock frequencies as well as alleviating the bus loading problem.

Figure 5.3 shows a crossbar-based interconnect which connects some locally synchronous blocks on the same chip [9]. The crossbar switch is fully asynchronous. Inside the chip, clock domain converters are used to bridge the asynchronous interconnect to the synchronous blocks, forming a globally asynchronous/locally synchronous (GALS) system.

5.2.3 Network-on-Chip

In the Network-on-Chip (NOC) interconnection scheme, units are connected via a homogeneous and scalable switching system. Communications between the units are through data transfers sometimes called packets. In a network there can be more than one path for communication between blocks. Such a network must therefore include schedules or dynamic decision making on the routing of communication traffic. Figure 5.4 shows an example of a NOC where IP blocks are connected together via a network of switches. One key feature of a NOC interconnect architecture is the use of

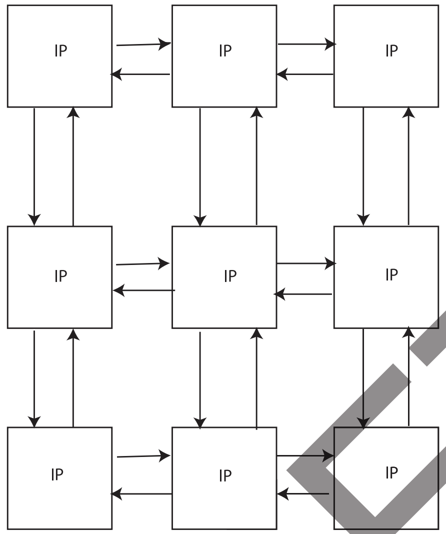


Figure 5.4 A NOC and Switch example.

a layered communication scheme, separating the transaction layer (which is specific to the application), the transport layer (which handles the packets) and the physical layer (which deals with wires and clocks). A NOC can be scalable, and the layered approach also ensures that it can easily be adapted to the latest silicon process technology. A NOC system can easily migrate to a newer and faster technology by changing the design of the physical layer alone.

5.3 Bus: Basic Architecture

5.3.1 Arbitration and Protocols

Conceptually the bus is just wires shared by multiple units. In practice, some logic provides an orderly use of the bus; otherwise two units may send signals at the same time causing conflict. When a unit has exclusive use of the bus, the unit is said to own the bus. Units can be either potentially master units that can request ownership or slave units that are passive and only respond to requests. A bus master is the unit that initiates communication on a computer bus or input/output paths. In an SOC, a bus master is a component within the chip, such as a processor. Other units connected to an on-chip bus, such as I/O devices and memory components, are the “slaves”. The bus master controls the bus paths using specific slave addresses and control signals. Moreover, the bus master also controls the flow of data signals directly between the master and the slaves.

A process called arbitration determines ownership. A simple implementation has a centralized arbitration unit with an input from each potential requesting unit. The arbitration unit then grsnts bus ownership to one requesting unit, as determined by the bus protocol.

A bus protocol is an agreed set of rules for transmitting information between two or more devices over a bus. The protocol determines the following:

- the type and order of data being sent;
- how the sending device indicates that it has finished sending the information;
- the data compression method used, if any;
- how the receiving device acknowledges successful reception of the information;
- how arbitration is performed to resolve contention on the bus and in what priority, and the type of error checking to be used.

5.3.2 Bus Bridge

A bus bridge is a module that connects together two buses, which are not necessarily of the same type. A typical bridge can serve three functions:

1. If the two buses use different protocols, a bus bridge provides the necessary format and standard conversion.
2. A bridge is inserted between two buses to segment them, and keep traffic contained within the segments. This improves concurrency: both buses can operate at the same time.
3. A bridge often contains memory buffers and the associated control circuits that allow write posting. When a master on one bus initiates a data transfer to a slave module on another bus through the bridge, the data is temporary stored in the buffer, allowing the master to proceed to the next transaction before the data is actually written to the slave. By allowing transactions to complete quickly, a bus bridge can significantly improve system performance.

5.4 Analytical Bus Models

The nature of the bus transaction depends on the bus structure. Multiple bus users must be arbitrated for access to the bus in any given cycle. Thus, arbitration can be part of the bus transaction, such as having the request cycle followed by the acknowledge cycle, or it can be performed by adding bus control lines and associated logic.

5.4.1 Bus Varieties

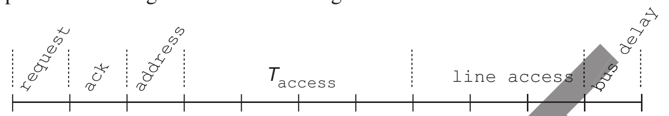
Buses may be *unified* or *split* (address and data). The unified bus is occupied with both address and data; the split bus has separate buses for each function.

Moreover, the buses may be *tenured*. This refers to buses that are occupied only while delivering addresses or data. Such buses assume that the receivers buffer the messages and create separate address and data transactions.

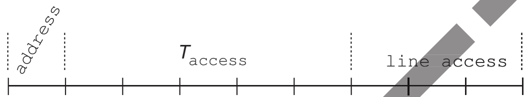
EXAMPLE 5.1 BUS EXAMPLES

Suppose we have a bus with transmission delay of one processor cycle and memory with 4 cycle access. Memory requires an additional 3 cycles to transmit a line. ($m = 1$ with page mode.)

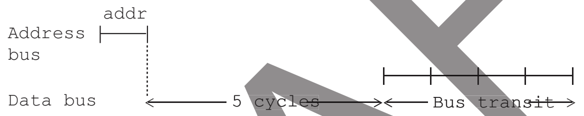
(a) Simple bus. This might have the following bus transaction time:



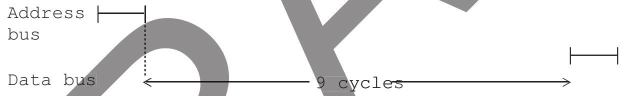
(b) Bus with arbitration support:



(c) Tenured split bus:



(d) Tenured split bus (width 16B), with a 1 cycle bus transaction time:



Cases (c) and (d) are interesting, since the current bus capacity exceeds the memory bandwidth; for instance in case (d), 4w in 8 cycles for memory, and 4w in one cycle for the bus. In both of these cases, the "bus"-memory situation is memory limited since that is where the contention will develop. In these cases, the bus time is added to the memory access time, and bus contention is ignored.

◇

5.4.2 Shared Bus

Whether we need to analyze the bus as a source of contention depends on its offered bandwidth (or offered occupancy) relative to the memory bandwidth. As contention and queues develop at the "bottleneck" in the system, we treat the most limiting resource as the source of the contention, and other parts of the system simply as delay elements. Thus buses must be analyzed for contention when they are more restrictive (have less available bandwidth) than memory.

Buses usually have no buffering (queues), and access delays cause immediate system slowdown. The analysis on the effects of bus congestion is made based on the access patterns.

Generally there are two types of access patterns:

1. Requests without immediate resubmissions. The denied request returns with the same arrival distribution as the original request. Once a request is denied, “something else” happens to delay the resubmission of the request.

2. Requests are immediately resubmitted. This is a more typical case, when multiple independent processors access a common bus. A denied request “sits on” the bus. It is immediately resubmitted. The processor is idle until the request is honored and serviced.

5.4.3 Simple Bus Model: without Resubmission

In the following, we assume that each request occupies the bus for the same service time (e.g. $T_{\text{line access}}$). Even if we have two different types of bus users (e.g. word requests and line requests on a single line, or (dirty) double line requests), most cases are reasonably approximated by simple computation of the per-processor average (offered) bus occupancy, ρ, given by:

$$\rho = \frac{\text{bus transaction time}}{\text{processor time + bus transaction time}}$$

The processor time is the mean time the processor needs to compute before making a bus request. Of course, it is possible for the processor to overlap some of its compute time with the bus time. In this case, the processor time is the net non-overlapped time between bus requests. In any event, $\rho \leq 1$.

The simplest model for n processors accessing a bus is given by:

$$\begin{aligned} \text{Prob}\,(\text{processor does not access bus}) &= 1-\rho \\ \text{Prob}\,(n \text{ processors do not access bus}) &= (1-\rho)^n \\ \text{Prob}\,(\text{bus is busy}) &= 1-(1-\rho)^n \\ &= \text{Bus bandwidth} = \text{Bus } B\,(\rho, n) \end{aligned}$$

and

$$Bw = \frac{\text{Bus } B\,(\rho, n)}{T_{\text{bus}}}$$

and the achieved bandwidth per processor (ρ_a) is given by:

$$\begin{aligned} n\rho_a &= B\,(\rho, n) \\ \rho_a &= \frac{B\,(\rho, n)}{n} \end{aligned}$$

5.4.4 Bus Model with Request Resubmission

A model that supports request resubmission involves a more complex analysis, and requires an iterative solution. There are several solutions, each providing similar results. The solution provided by Hwang and Briggs [16] is an iterative pair of equations:

$$a = \frac{\rho}{\rho + (\rho_a/\rho)\,(1-\rho)}$$

and

$$n\rho_a = 1-(1-a)^n$$

where a is the actual offered request rate. To find a final ρ_a, initially set $a = \rho$ to begin the iteration. Convergence usually occurs within four iterations.

5.4.5 Using the Bus Model: Computing the Offered Occupancy

The model in the preceding section does not distinguish among types of transactions. It just requires the mean bus transaction time, which is the average number of cycles that the bus is busy managing a transaction. Then the issue is finding the offered occupancy, ρ.

The offered occupancy is the fraction of the time that the bus would be busy if there were no contention among transactions (bounded by 1.00). In order to find this, we need to determine the mean time for a bus transaction and the compute time between transactions.

The nature of the processor initiating the transaction is another factor. Simple processors make *blocking* transactions. In this case the processor is idle after the bus request is made and resumes computation only after the bus transaction is complete. The alternative for more complex processors is a *buffered* (or non blocking) transaction. In this case the processor continues processing after making a request, and may indeed make several requests before completion of an initial request. Depending on the system configuration there are three common cases:

1. A single bus master with blocking transactions. In this case there is no bus contention as the processor waits for the transaction to complete. Here the achieved occupancy, ρ_a, is the same as the offered occupancy, and $\rho = \rho_a =$ (bus transaction time)/(compute time + bus transaction time).

2. Multiple (n) bus masters with blocking transactions. In this case the offered occupancy is simply $n\rho$ where ρ is as in case 1. Now contention can develop so we use our bus model to determine the achieved occupancy, ρ_a.

Example. Suppose a processor has bus transactions that consist of cache line transfers. Assume that 80% of the transactions move a single line and occupy the bus for 20 cycles and 20% of the transactions move a double line (as in dirty line replacement) which takes 36 cycles. The mean bus transaction time is 23.2 cycles. Now assume that a cache miss (transaction) occurs each 200 cycles.

In case (1) the bus is occupied: $\rho = \rho_a = 23.2/223.2 = 0.10$; there is no contention, but the bus causes a system slow down, as discussed below.

In case (2) suppose we have 4 processors. Now the offered occupancy is $\rho = 0.104$ and we use our model to find the contention time. Initially we set $a = \rho = .104$, $n\rho_a = 1-(1-a)^n = 1-(1-.104)^4$; now we find ρ_a and substitute the value of ρ_a for a and continue.

So initially $\rho_a = 0.089$; after the next iteration $\rho_a = 0.010$; and after several iterations $\rho_a = 0.095$. We always achieve less than what is offered and the difference is delay due to contention. So:

$$\rho_a = 0.095 = \frac{\text{bus transaction time}}{\text{compute time + bus transaction time + contention time}}$$

Solving for the contention time, we get about 21 cycles.

5.4.6 Effect of Bus Transactions and Contention Time

There are two separate effects of bus delays on overall system performance. The first is the obvious case of blocking which simply inserts a transaction delay into the program execution. The second effect is due to contention. Contention reduces the rate of transaction flow into the bus and memory. This reduces performance proportionally.

In the case of blocking the processor simply slows down by the amount of the bus transaction. So the relative performance compared to an ideal processor with no bus transactions is,

$$\text{Relative Performance} = \frac{\text{compute time}}{\text{compute time + bus transaction time}}$$

In the case (1) example the processor slows down by $200/223.3 = 0.896$.

Contention, when present, adds additional delay. In case (2) the individual processor slows down by $200/(223.2 + 21) = 0.819$ The result of contention is the simply slow down the system (without contention) by the ratio of ρ_a/ρ . The supply of transactions is reduced by this ratio.

5.5 SOC Standard Buses

The two most commonly used SOC bus standards are the AMBA bus developed by ARM, and the CoreConnect bus developed by IBM. The latter has been adopted in Xilinx's Virtex platform FPGA families.

5.5.1 AMBA

The Advanced Microcontroller Bus Architecture (AMBA), introduced in 1997 had its origin from the ARM processor, one of the most successful SOC processors used in industry. The AMBA bus is based on traditional bus architecture employing two levels of hierarchy. Two buses are defined in the AMBA specification [4]:

- The Advanced High-performance Bus (AHB) is designed to connect embedded processors, such as an ARM processor core, to high-performance peripherals, DMA controllers, on-chip memory and interfaces. It is a high-speed, high-bandwidth bus architecture that uses separate address, read and write buses. A minimum of 32 bit data operation is recommended in the standard, and data widths are extendable to 1024 bits. Concurrent multiple master/slave operations are supported. It also supports burst mode data transfers and split transactions. All transactions on the AHB bus are referenced to a single clock edge, making system level design easy to understand.
- The Advanced Peripheral Bus (APB) has lower performance than the AHB bus, but is optimized for minimal power consumption and has reduced interface complexity. It is designed for interfacing to slower peripheral modules.

A third bus, the Advanced System Bus (ASB), is an earlier incarnation of the AHB, designed for lower performance systems using 16/32 bit microcontrollers. It is used where cost, performance and complexity of the AHB is not justified.

The AMBA bus was designed to address a number of issues exposed by users of the ARM processor bus in SOC integration. The goals achieved by its design are [13]:

1. *Modular design and design reuse*. Since the ARM processor bus interface is extremely flexible, inexperienced designers could inadvertently create inefficient or even unworkable designs by using ad hoc bus and control logic. The AMBA specification encourages a modular design methodology that supports better design partitioning and design reuse.
2. *Well-defined interface protocol, clocking and reset*. AMBA specifies a low-overhead bus interface and clocking structure that is simple yet flexible. The performance of the AMBA bus is enhanced by its multi-master, split transaction and burst mode operations.
3. *Low-power support*. One of the attractions of the ARM processor when compared with other embedded processor cores is its power efficiency. The two-level partitioning of the AMBA buses ensures energy-efficient designs in the peripheral modules which fits well with the low-power CPU core.
4. *On-chip test access*. AMBA has an optional on-chip test access methodology that reuses the basic bus infrastructure for testing modules that are connected to the bus.

The Advanced High-Performance Bus (AHB)

Figure 5.5 depicts a typical system using the AMBA bus architecture. The AHB forms the system backbone bus on which the ARM processor, the high-bandwidth memory interface and RAM, and the Direct Memory Access devices reside. The interface between the AHB bus and the slower APB bus is through a bus bridge module.

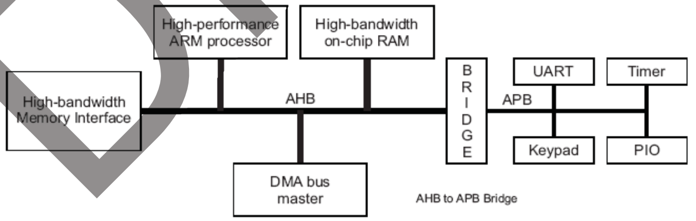

High-performance ARM processor
High-bandwidth on-chip RAM
High-bandwidth Memory Interface
AHB
BRIDGE
APB
UART
Timer
Keypad
PIO
DMA bus master
AHB to APB Bridge

Figure 5.5 A typical AMBA bus based system [13].

The AMBA AHB bus protocol is designed to implement a multi-master system. Unlike most bus architectures designed for PCB based systems, the AMBA AHB bus avoids tristate implementation by employing a central multiplexer interconnect scheme. This method of interconnect provides higher performance and lower power than using tristate buffers. All bus masters assert the address and control signals, indicating the type of transfer each master requires. A central arbiter determines which master has its address and control signal routed to all the slaves. A central decoder circuit selects the appropriate read data and response acknowledge signal from the slave that is involved in the transaction. Figure 5.6 depicts such a multiplexer interconnect scheme for a system with three masters and four slaves.

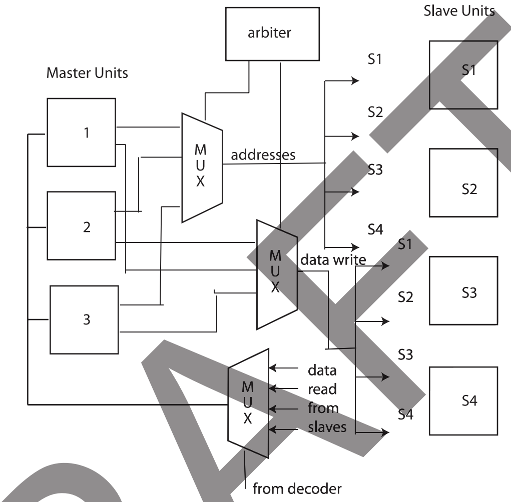


Figure 5.6 Multiplexor interconnection for a 3-masters/4-slaves system [4].

Transactions on the AHB bus involve the following steps:

- Bus master obtains access to the bus - this process begins with the master asserting a request signal to the arbiter. If more than one master simultaneously requests the control of the bus, the arbiter determines which of the requesting masters will be granted the use of the bus.
- Bus master initiates transfer - a granted bus master drives the address and control signals with the address, direction and width of the transfer. It also indicates whether the transaction is part of a burst in the case of burst mode operation. A write data bus operation moves data from the master to a slave, while a read data bus operation moves data from a slave to the master.
- Bus slave provides a response - a slave signals to the master the status of the transfer such as whether it was successful, if it needs to be delayed, or that an error occurred.

Figure 5.7a depicts a basic AHB transfer cycle. An AHB transfer consists of two distinct phases: the address phase and the data phase. The master asserts the address and

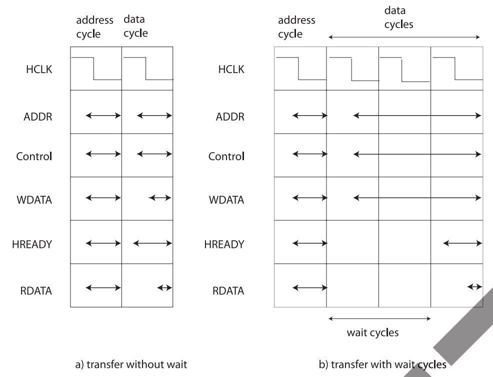


Figure 5.7 A simple AHB transfer [4].

control signals on the rising edge of **HCLK** during the address phase, which always lasts for a single cycle. The slave then samples the address and control signals and responds accordingly during the data phase to a data read or write operation, and indicates its completion with the **HREADY** signal. A slave may insert wait states into any transfer by delaying the assertion of **HREADY** as shown in Figure 5.7b. For a write operation, the bus master holds the data stable throughout the extended data cycles. For a read transfer the slave does not provide valid data until the last cycle of the data phase.

The AHB bus is a pipelined (tenured) bus. Therefore the address phase of any transfer can occur during the data phase of a previous transfer. This overlapping pipeline feature allows for high performance operation.

The Advanced Peripheral Bus (APB)

The Advanced Peripheral Bus (APB) is optimized for minimal power and low complexity instead of performance. It is used to interface to any peripherals which are low bandwidth.

The operation of the APB is straightforward, controlled by a three-state finite-state machine.

5.5.2 CoreConnect

As in the case of AMBA bus, IBM's CoreConnect Bus is an SOC bus standard designed around a specific processor core, the PowerPC, but it is also adaptable to other processors. The CoreConnect Bus and the AMBA bus share many common features. Both have a bus hierarchy to support different levels of bus performance and complexity. Both have advanced bus features such as multiple master, separate read/write ports, pipelining, split transaction, burst mode transfer and extendable bus width.

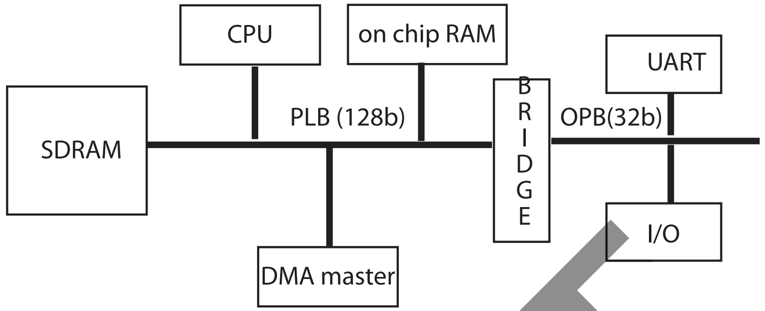


Figure 5.8 A CoreConnect based SOC [17].

The 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 5.8 illustrates how the CoreConnect architecture can be used in an SOC system built around a PowerPC. High performance, high bandwidth blocks such as the PowerPC 440 CPU core, PCI-X bus bridge and PC133/DDR133 SDRAM Controller are connected together using the PLB, while the OPB hosts lower data rate on-chip 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 modules.

The Processor Local Bus (PLB)

The PLB is used for high bandwidth, high performance and low latency interconnections between the processors, memory and DMA controllers [17]. The fully synchronous, split transaction bus with separate address, read and write data buses, allows two simultaneous transfers per clock cycle.

PLB transactions, as in the AMBA AHB, consist of multiple phases which may last for one or more clock cycles, and involve the address and data buses separately. Transactions involving the address bus 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 5.9 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

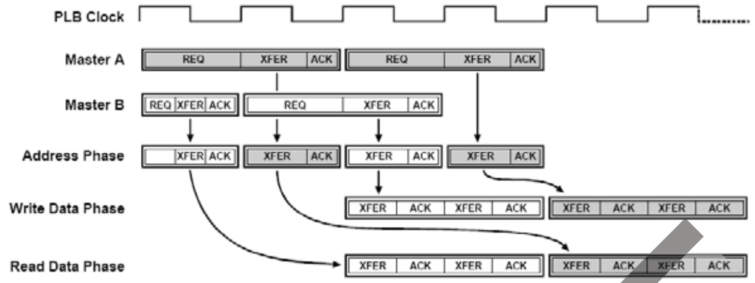


Figure 5.9 PLB Transfer Protocol [17].

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.

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 5.9, the first (or only) data beat of a write transfer coincides with the address transfer phase.

Split Transaction

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 5.9.

The On-Chip Peripheral Bus (OPB)

The On-Chip Peripheral Bus (OPB) is a secondary bus architected to alleviate system performance bottlenecks by reducing capacitive loading on the PLB[18]. Peripherals suitable for attachment to the OPB include serial ports, parallel ports, UARTs, GPIO, timers and other low-bandwidth devices. The OPB is far more sophisticated than the AMBA APB. It 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 peripherals to be added to a custom core logic design without changing the I/O on either the OPB arbiter or existing peripherals. Figure 5.10 shows one method of structuring the OPB address and data buses. Both masters and slaves provide enable control signals for their outbound buses. By requiring that each unit provide this signal, the associated bus combining logic can be strategically placed throughout the chip. As shown in the figure, either of the masters is capable of pro-

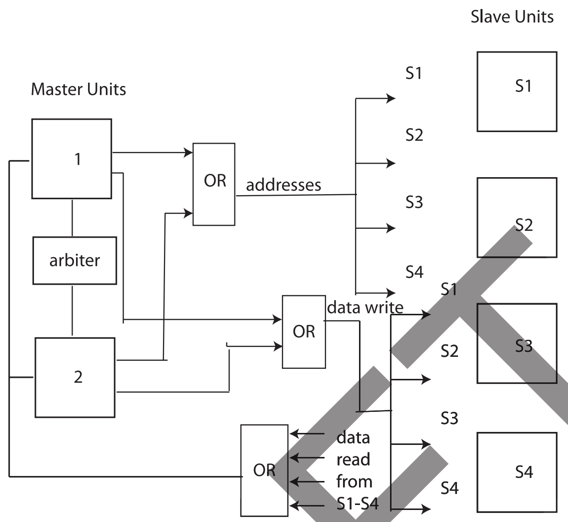
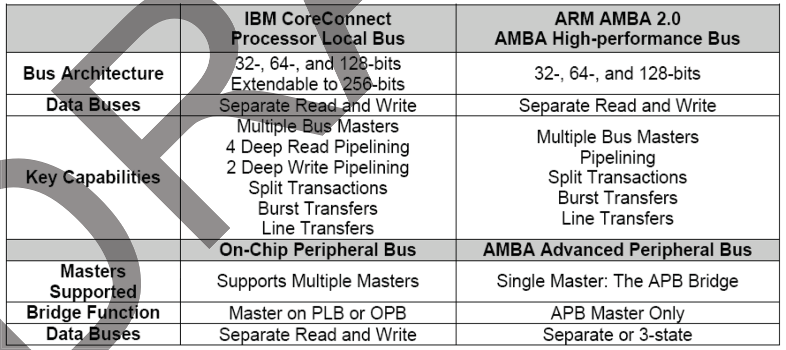


Figure 5.10 The On-chip Peripheral Bus (OPB) [18].



	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

Figure 5.11 Comparison between CoreConnect and AMBA architectures [30].

viding an address to the slaves, whereas both masters and slaves are capable of driving and receiving the distributed data bus.

Figure 5.11 shows a comparison between the AMBA and CoreConnect bus standards.

5.5.3 Bus Sockets and Bus Wrappers

Using a standard SOC bus for the integration of different reusable IP blocks has one major drawback. Since standard buses specify protocols over wired connections, an IP block that complies with one bus standard cannot be reused with another block using a different bus standard. One approach to alleviate this is to employ a hardware "socket" (also called a bus wrapper) to separate the interconnect logic from the IP core using a well-defined IP core protocol which is independent of the physical bus protocol. Core-to-core communication is therefore handled by the hardware wrapper, not by the core. This approach is taken by Virtual Socket Interface Alliance (VSIA) [7] with their Virtual Component Interface (VCI) [34] and by Sonics Inc. employing the Open Core Protocol (OCP) and Silicon Backplane μNetwork [33].

VSIA proposes a set of standards and interfaces known as Virtual Socket Interface (VSI) that enables system level interaction on a chip using pre-designed blocks (called Virtual Components) [34]. This encourages integrated circuits to be designed using a component paradigm. The Virtual Components (VCs), which are effectively IP blocks that conform to the VSI specifications, can be one of three varieties. *Hard* VCs consist of placed and routed gates with all silicon layers defined. It has predictable performance, area usage and power consumption, but offers no flexibility. *Soft* VCs are designed in some hardware description language representation, which are mapped to physical design through synthesis, placement and routing. They can be easily modified, but generally take more effort to integrate and verify in the SOC design as well as having less predictable performance. Finally, *firm* VCs offer a compromise between the two. They come in the form of generators or partially placed library blocks that require final routing and/or placement adjustment. This form of VCs provide more predictable performance than soft VCs, but still offer some degree of flexibility in aspect ratio and configuration.

In order to connect these different VCs together, VSIA has developed a Virtual Component Interface (VCI) specification to which other proprietary buses can interface. By following the VCI specification, a designer can take a virtual component and integrate it with any of several buses in order to meet system performance requirements. The VCI standard specifies a family of protocols. Currently three protocols are defined: the Peripheral VCI (PVCI), the Basic VCI (BVCI) and the Advanced VCI (AVCI) [34]. The PVCI is a low performance protocol where the request and the response data transfer occur during a single control handshake transaction. It is therefore not a split-transaction protocol. The BVCI employs a split-transaction protocol, but responses must arrive in order. In other words, the response data must be supplied in the same order in which the initiator generated the requests. The AVCI is similar to the BVCI, but out of order transactions are allowed. Requests are tagged and transactions can be interleaved and re-ordered.

In addition to the specification of the Virtual Component Interface, VSIA also specifies a number of abstraction layers as depicted in Figure 5.12 to define the representation views required to integrate a virtual component into an SOC design [7]. The idea is that if both the IP block provider (VC provider) and the system integrator (VC integrator) conform to the Virtual Socket Interface Specifications (VSI) at all levels of abstraction, SOC designs using a IP component paradigm can proceed with less risk of errors.

An alternative to VCI is the Open Core Protocol (OCP) promoted by the Open Core Protocol International Partnership (OCP-IP) [27]. The OCP defines a point-to-point in-

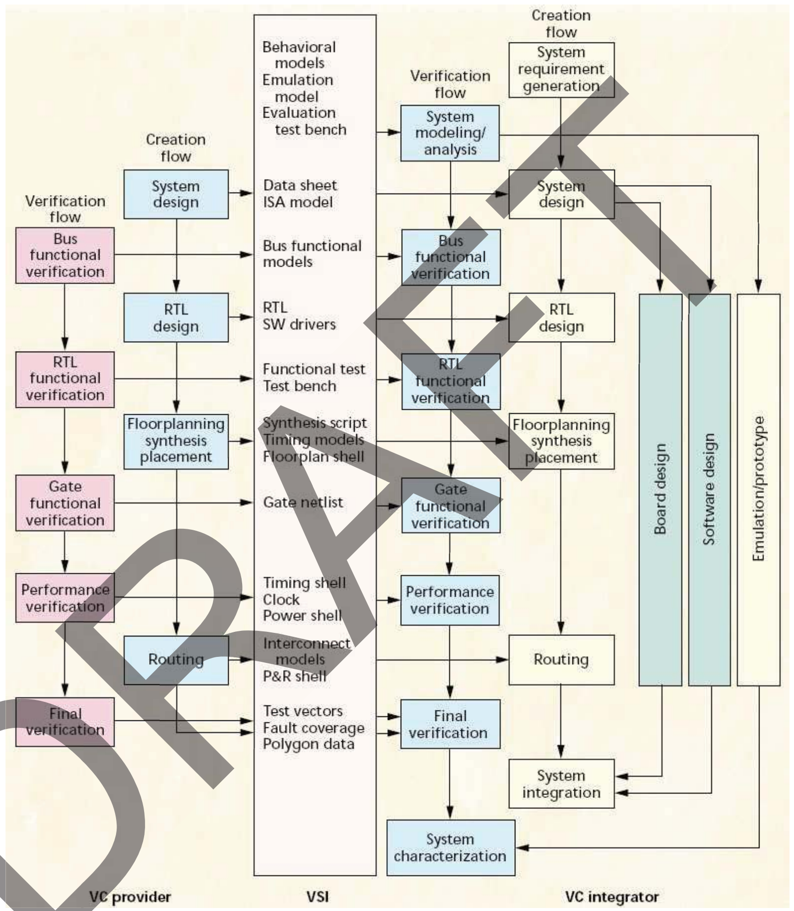


Figure 5.12 VSIA representation views for integrating a virtual component into an SOC [34].

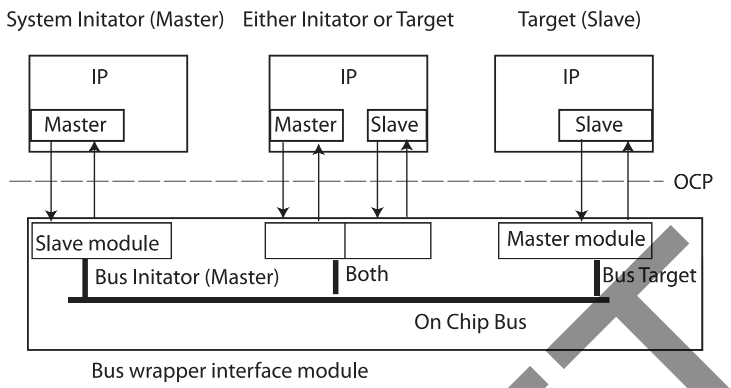


Figure 5.13 A Three Core System using OCP and Bus Wrappers [33].

terface between two communicating entities such as two IP cores using a core-centric protocol. An interface implementing the OCP assumes the attributes of a *socket*, which is effectively a bus wrapper that allows interfacing to the target bus. A system consisting of three IP core modules using the OCP and bus wrappers is shown in Figure 5.13. One modules is a system initiator, one is a system target, and another is both initiator and target.

Another layer of interconnection can be made above the OCP in order to help IP integration further. Sonics Inc. proposes their proprietary SiliconBackplane Protocol that seamlessly glues together IP blocks that uses the OCP. The communication between different blocks takes place over the Silicon Backplane μNetwork, which has a scalable bandwidth of 50–4000 MB/sec. Figure 5.14 depicts how the Sonics μNetwork components are connected together [33].

The wrapper-based approach has been demonstrated to reduce the design time of SOC, but at a cost. Attaching simple wrapper hardware increases the access latencies and incurs a hardware overhead of 3–5 K gates [25]. In addition, first-in-first-out (FIFO) buffers are often embedded in the wrapper hardware in order to improve performance.

5.6 Beyond the Bus: Switching Interconnects

At some point the number of units and the traffic between units force the designer to move beyond the bus and use some form of interconnect switching. Interconnect networks or switches have long been used for interconnecting large processor clusters. This section presents some basic concepts and alternatives in the design of the physical interconnect network. This network consists of a configuration of switches to enable the interconnection of N units. In the context of SOC, these designs are equally applicable to switched interconnects and NOC, which includes a switch at the transaction and physical level. The design efficiency or cost-performance of the interconnection network is determined by:

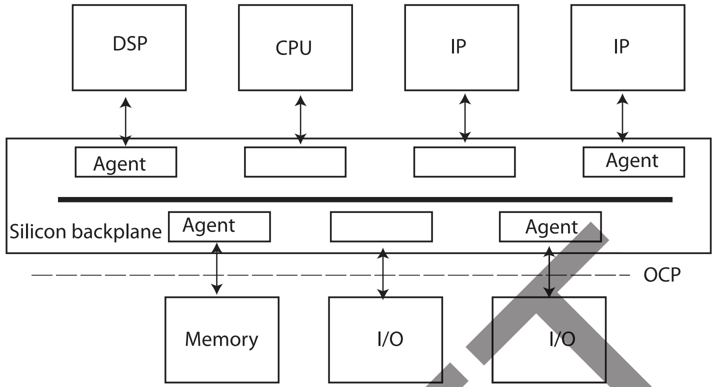


Figure 5.14 Sonics μNetwork Configuration [33].

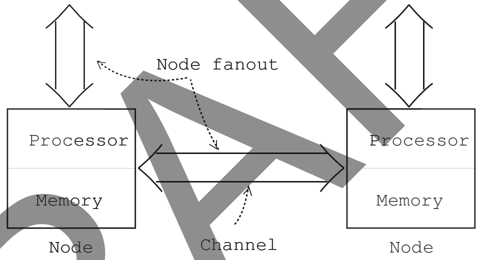


Figure 5.15 Node and channels.

1. The delay in connecting a requesting unit to its destination.
2. The bandwidth between units and the number of connections that can be carried on concurrently.
3. The cost of the network.

SOC interconnect switches. This section is an abstract of some of the basic concepts and results from the general computer interconnect literature. In SOC switching, currently the number of nodes (units) is typically limited to 16–64. Since the units are on chip, the link bandwidth, w, is relatively large: 14–128. In SOC, dynamic networks are dominant so far; either crossbar or MIN static networks, when used, tend to be grid (torus). As the number of SOC units increases, a greater variety of network implementations are expected.

In a network, units communicate with one another via a link or a channel, which can be either unidirectional or bidirectional. Links have bandwidth or the number of bits per unit time that can be transmitted concurrently between units. Units may have multiple

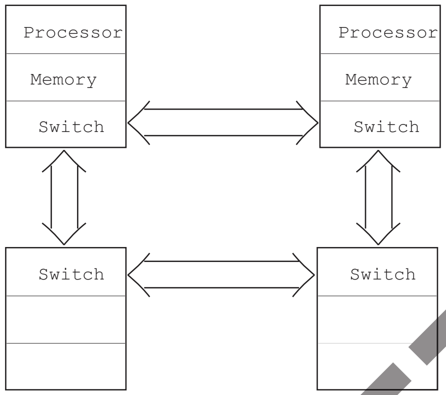


Figure 5.16 Static network (links between units is fixed).

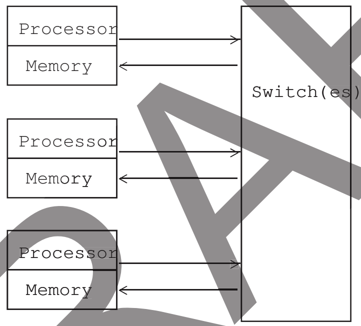


Figure 5.17 Dynamic network (links between units vary to establish connection).

links to other units, this defines their fanout—the number of bidirectional channels that connect a particular node to neighboring nodes.

Networks are said to be static or dynamic. In a *static network*, the topology or the relationship between nodes in the network is fixed (Figure 5.16). The path between two nodes does not change. In a *dynamic network*, the paths between nodes can be altered both to establish connectivity and also to improve network bandwidth (Figure 5.17).

5.6.1 Static Networks

In a static network the *distance* between two units is the smallest number of links or channels (or hops) that must be traversed for establishing communications between them. The *diameter* of the network is the largest distance (without backtracking) between any two units in the network. An example of a static netwok in linear network (Figure 5.18a). We improve its average distance and diameter by converting it into a ring (Figure 5.18b).

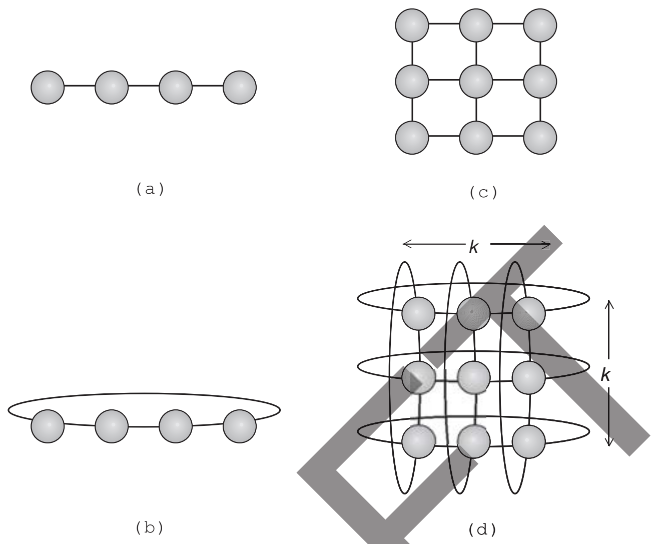


Figure 5.18 Example of static network without preferred sites. (a) Linear array. (b) Linear array with closure (a ring). (c) Grid (2D mesh). (d) $k \times k$ grid with closure (a 2D torus). These are also called (k, d) networks. In (a) and (b), we have $k = 4$, $d = 1$ (one-dimensional). In (c) and (d), we have $k = 3$, $d = 2$.

Assume there are k nodes in a linear array, and we wish to interconnect several such arrays. Instead of simply increasing the number of linear elements, we can increase the dimensionality of the network, creating a grid network of two dimensions (Figure 5.18c). These are called (k,d) networks where $d = n$ is the dimensionality of the network. Figure 5.18(d) represents a torus, commonly refered to as a nearest-neighbor mesh.

We can continue adding nodes to the network beyond the $k \times k$ specified in the grid by using a third dimension. Such an interconnection topology would be referred to as a *k-ary three-cube* [10]; k is simply the number of nodes in each dimension.

When $k = 2$, we have the special case of the binary cube, or hypercube. In the binary hypercube, the dimensionality of the hypercube is determined by the fanout from each node. In general, the number of nodes (N) and the diameter can be determined as follows: for $(2, n)$, the binary n-cube with bidirectional channels has:

$$N = 2^n$$

and for the (2,n) case:

$$\text{Diameter} = n.$$

For general (k, n) with n dimensions and with closure and bidirectional channels, we have

$$N = k^n$$

or

$$n = \log_k N.$$

and

$$\text{Diameter} = \left\lceil \frac{k-1}{2} \right\rceil n.$$

Example. Suppose we have a 4 by 4 grid (torus as in Figure 5.18(d)). In (k,d) terms it is a (4,2) network, $N = 16$ and $n = 2$ and the Diameter is 4.

In general, it is the dimension of the network and its maximum distance that are important to cost and performance.

Links are characterized in three ways:

1. The cycle time of the link, T_{ch}. This corresponds to the time it requires to transmit between neighboring nodes. $1/T_{ch}$ is the bandwidth of a wire in the link or channel.

2. The width of the link, w. This determines the number of bits that may be concurrently transmitted between two nodes.

3. Whether the link is unidirectional or bidirectional.

Associated with the link characterization is the length of the message in bits (l) plus H header bits. The header is simply the address of the destination node. Thus, $T_{ch} \times (l + H)/w$ will be the time required to transmit a message between two adjacent units.

Suppose unit A has a message for unit C, which must be transmitted via unit B. If node B is available, the message is transmitted first from A to B and stored at B. After the message has been completely transmitted, node B accesses node C and transmits the message to C if C is available. Rather than storing the message at B, we can use *wormhole routing*. As the message is received at B, it is buffered only long enough to decode its header and determine its destination. As soon as this minimal amount of information can be determined, the message is retransmitted to C, assuming that C is available. The amount of buffering then required at B is significantly reduced and the overall time of transmission is:

$$T_{\text{wormhole}} = T_{ch}[d \cdot h + l/w],$$

where $k = \lceil H/w \rceil$.

Example. In a 4 by 4 grid, (k,d) = (4,2) and, assuming $T_{ch} = 1$, let $h = 1$, $l = 256$ and $w = 64$. Then $T_{wormhole} = 2 + 4 = 6$ cycles.

Once the header is decoded at an intermediate node, that node can determine whether the message is for it or for another node. The intermediate node selects a minimum distance path to the destination node. If multiple paths have the same distance, then this intermediate node will select the path that is currently unblocked or available to it.

5.6.2 Dynamic Networks

The dynamic indirect network is shown in Figure 5.19a. For ease of representation, the network is usually shown as in Figure 5.19b.

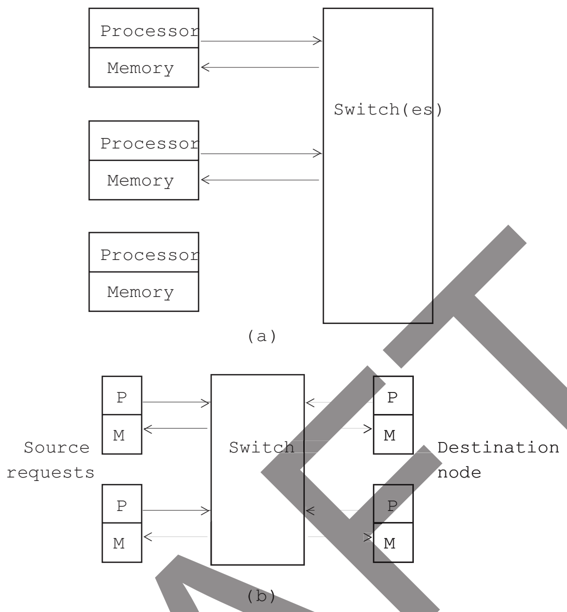


Figure 5.19 A basic dynamic, indirect switching network.

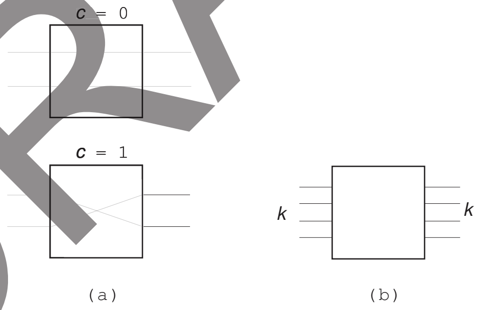


Figure 5.20 (a) A 2 $\times$ 2 crossbar with control c. (b) This can be generalized to a $k \times k$ crossbar switch.

Typically, the basic element in the dynamic network is a crossbar switch (Figure 5.20). The crossbar simply connects one of k points to any of another k points. Multiple messages can be concurrently executed across the crossbar switch, so long as two messages do not have the same destination. The cost of the crossbar switch increases as n^2, so that for larger networks, use of a crossbar switch only becomes prohibitively expensive. In order to contain the cost of the switch, we can use a small crossbar switch as the basis of a multistage network, frequently referred to as a MIN—*multistage in-*

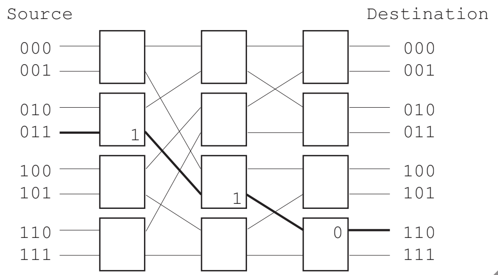


Figure 5.21 Baseline dynamic network topology.

terconnection network [35]. There are many types, including baseline, Benes, Clos, Omega [23], and Banyan networks. The baseline network is among the simplest, and is shown in Figure 5.21. The header causes successive stages of the switch to be set so that the proper connection path is established between two nodes. For example, consider a deterministic "obvious" routing algorithm for these M, N networks. Suppose node 010 sends a message to destination 110. The switch outputs labeled 1, 1, 0 cause the message to be routed to the 110 destination node by setting the control (c) so that either the upper output ("0") or the lower output ("1") of each switch is selected. Similarly, the return path is simply 010. The number of stages between two nodes is:

$$\text{Stages} = \lceil \log_k N \rceil,$$

where k is the number of inputs to the crossbar element ($k \times k$), and therefore the total number of ($k \times k$) switches required for a one-bit wide path is:

$$\frac{N}{k} \times \lceil \log_k N \rceil.$$

Other dynamic networks provide different tradeoffs on achievable message bandwidth, message delay, and fault tolerance. Table 5.3 summarizes some of the attributes of some common dynamic networks.

5.7 Evaluating Interconnect Networks

In recent years, there have been a number of important analyses about the comparative merits of various network configurations [29, 19, 21].

5.7.1 Static versus Dynamic Networks

In this section, we present the results and largely follow the analyses performed by Agarwal in his work on network performance [2].

Table 5.3 Dynamic networks, switching N inputs $\times$ N outputs using $k \times k$ switches (each input is one bit).

Network	Other Equivalent Networks	Stages of Delay (in units of $k \times k$ switch delay)	Blocking	Approx. Cost ($k \times k$ switches)
Baseline	Delta, Omega SW Banyan	$\lceil \log_k N \rceil$	Yes	$\frac{N}{k}\lceil \log_k N \rceil$
Benes	—	$2\lceil \log_k N \rceil - 1$	Nonblocking if reconfigured	$\frac{2N}{k}\lceil \log_k N \rceil$
Clos	—	$2\lceil \log_k N \rceil - 1$	Strictly non-blocking	$\frac{4N}{k}\lceil \log_k N \rceil$

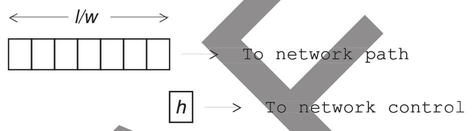


Figure 5.22 Message transmission from node to switch.

Dynamic Networks

Assume we have a dynamic indirect network made up of $k \times k$ switches with wormhole routing. Let us assume this network has n stages and channel width w with message length l. In the indirect network, we assume that the header network path address is transmitted in one cycle just before the message leaves the node (i.e., there is only one cycle of header overhead to set up the interconnect); see Figure 5.22.

Assuming the switches have unit delay ($T_{\text{ch}} =$ one cycle), the total time for a message to transit the network without contention is:

$$T_c = n + \frac{l}{w} + 1 \text{ cycles.}$$

For all our subsequent analysis we assume that $n + l/w \gg 1$, so

$$T_c = n + \frac{l}{w} \text{ cycles.}$$

In a blocking dynamic network, each network switch has a buffer. If a block is detected, a queue develops at the node; so each of N units with occupancy ρ requests service from the network. Since the number of connection lines at each network level is the same (N), then expected occupancy for each is ρ. At each switch, the message transmits experiences a waiting time. Kruskal and Snir [21] have shown that this waiting time is (assume that $T_{\text{ch}} = 1$ cycle and express time in cycles):

$$T_w = \frac{\rho(l/w)(1 - 1/k)}{2(1-\rho)}.$$

The channel occupancy is

$$\rho \;=\; m\frac{l}{w}$$

where m is the probability that a node makes a request in a channel cycle.

The total message transit time, T_{dynamic}, is:

$$\begin{aligned} T_{\text{dynamic}} &= T_c + nT_w \\ &= \left[n + \frac{l}{w} + \frac{n\rho}{2(1-\rho)}(\frac{l}{w})(1 - 1/k)\right] T_{\text{ch}}. \end{aligned}$$

Static Networks

A similar analysis may be performed on a static (k, n) network. Let k_d be the average number of hops required for a message to transit a single dimension. For a unidirectional network with closure $k_d = \frac{(k-1)}{2}$, and for a bidirectional network $k_d = \frac{k}{4}$ (k even), the total time for a message to pass from source to destination is:

$$T_c = \left[h \times n \times k_d + \frac{l}{w}\right] T_{\text{ch}}.$$

Again, we assume that $T_{\text{ch}} = 1$ cycle and perform the remaining computations on a cycle basis. Agarwal [2] computes the waiting time ($M/G/1$) as:

$$T_w = \frac{\rho}{1-\rho}\frac{l}{w}\frac{k_d - 1}{k_d^2}(1 + 1/n).$$

The total transit time for a message to a destination ($h = 1$) is

$$\begin{aligned} T_{\text{static}} &= T_c + nk_dT_w \\ &= nk_d + l/w + \frac{nk_d\rho}{1-\rho}\left(\frac{l}{wk_d}\right)(1 + 1/n). \end{aligned}$$

The preceding cannot be used for low k (i.e., $k = 2, 3, 4$). In this case [1],

$$T_w = \frac{\rho}{2(1-\rho)}\frac{l}{w}$$

and

$$\rho = \frac{mk_dl}{2w} \qquad \text{or, for hypercube,} \quad \frac{mk_dl}{w}.$$

5.7.2 Comparing Networks: Example

In the following example assume that m, the probability that a unit requests service in any channel cycle is 0.1; $h = 1$, $l = 256$, $w = 64$. Compare a 4 by 4 grid (torus) static network with $N = 16, k = 4, n = 2$ and a MIN dynamic network with $N = 16, k = 2$.

For the dynamic network, the number of stages is:

$$n = \log_2 16 = 4$$

while the channel occupany is:

$$\rho = m\frac{l}{w} = 0.1\frac{256}{64} = 0.4$$

The message transit time without contention is:

$$T_c = n + \frac{l}{w} + 1 = 4 + \frac{256}{64} + 1 = 9 \text{ cycles}$$

while the waiting time is:

$$T_w = \frac{\rho(l/w)(1-1/k)}{2(1-\rho)} = \frac{0.4(256/64)(1-1/2)}{2(1-0.4)} = \frac{0.8}{1.2} = 0.67 \text{ cycle}$$

Hence the total message transit time is:

$$T_{\text{dynamic}} = T_c + nT_w = 9 + 4(0.67) = 11.68 \text{ cycles}$$

For the static network, the average number of hops $k_d = k/4 = 1$, and the total message time is:

$$T_c = \left[h \times n \times k_d + \frac{l}{w}\right] T_{\text{ch}} = [1 \times 2 \times 1 + (256/64)] = 6.$$

Since

$$\rho = \frac{mk_d l}{2w} = \frac{0.1 \times 1 \times 256}{2 \times 64} = 0.2,$$

and T_w for low k is given by:

$$T_w = \frac{\rho}{2(1-\rho)}\frac{l}{w} = \frac{0.2}{2(1-0.2)}\frac{256}{64} = 0.5,$$

the waiting time is given by

$$\begin{aligned} T_{\text{static}} &= T_c + nk_dT_w \\ &= 6 + 2(1)(0.5) \\ &= 7 \text{ cycles} \end{aligned}$$

5.8 Switches in SOC

An effective alternative to bus based interconnect is to disperse data traffic over the entire design by connecting the user IP cores through an interconnect fabric. In this way, data transfer bottlenecks are avoided because multiple data transfers can be performed

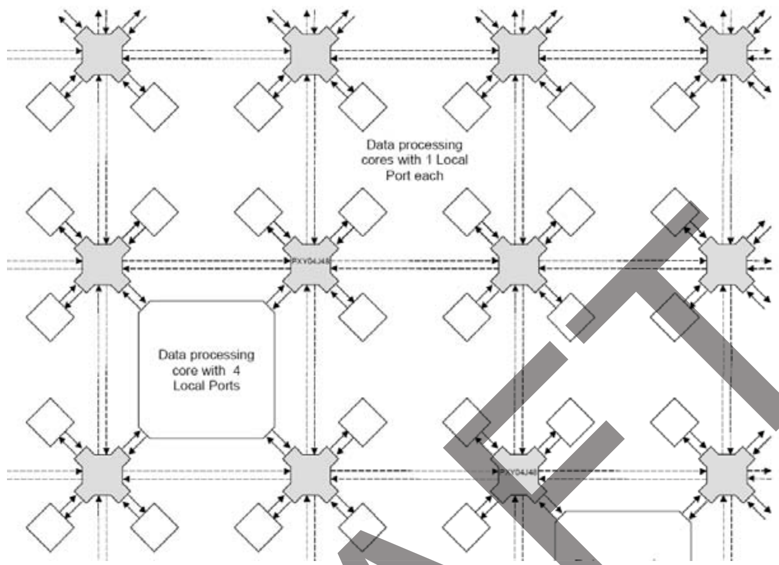

Data processing cores with 1 Local Port each
Data processing core with 4 Local Ports

Figure 5.23 Xfabric connecting Data Processing Core via Junction Components [8].

simultaneously. Xfabric uses this approach to connect user cores on a Xilinx FPGA as shown in Figure 5.23 [8]. Data processing cores with one to four communication ports are interconnected via a fabric of junction components (shown in gray). These data routing junctions manage system data flow autonomously between multiple user cores. Multiple instances of junctions form a 2-dimensional communication grid that can connect together up to 1024 single-port cores. Horizontal and vertical data transport links (shown in dashed lines) between junction components enable efficient and deterministic data communications between cores located anywhere on the chip.

Figure 5.24 shows the functional schematic of a junction component. Each junction consists of four Local Ports (LPORT0 to LPORT3) and four Global Ports (GPORT0 to GPORT3). User cores send 48-bit words and receive 32-bit words via Local Ports, while the 16-bit Global Ports are used to route data to adjacent junctions.

Each junction component performs all the necessary routing and arbitration function to deliver multiple parallel data streams between data sources and destinations with minimum latency, thus avoiding transfer bottlenecks found in bus based systems.

5.8.1 Asynchronous Crossbar Interconnect for Synchronous SOC

Another switch-based interconnect scheme designed specifically for SOC applications is the PivotPoint architecture by Fulcrum [9]. The center of the system is the Nexus crossbar switch (see Figure 5.3) which has a data throughput rate of 1.6Tbps. Nexus uses clockless asynchronous circuits and has the advantages normally associated with

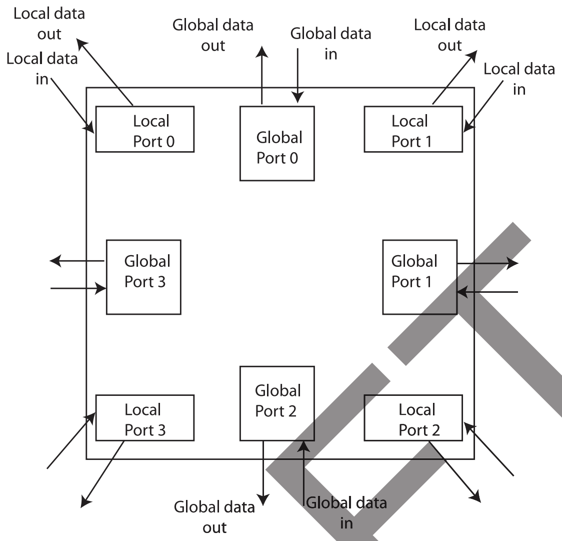


Figure 5.24 Schematic diagram of a Junction component [8].

this design style, including adaptivity to process technology, environmental variations, and lower system power consumption. The choice of asynchronous design style is partly driven by the need for interconnecting multiple clock domain cores. The synchronous cores can run at different frequencies with independent phase relationships to each other. Clock-domain converters are required to interface between the synchronous cores and the asynchronous crossbar. Since the crossbar switch does not use any clock signals, integrating different clock domains require no extra effort. In this way, the system is globally asynchronous, but locally synchronous, which is also known as a GALS system.

Data transfer on Nexus is done through bursts. Each burst contains a variable number of data words (36-bit) and is terminated by a tail signal. A 4-bit control is used to indicate a destination channel (TO), which becomes the source channel (FROM) when the burst leaves the crossbar. The format of the burst is shown in Figure 5.25. Bursts are automatically routed by the crossbar and cannot be dropped, fragmented or duplicated. The crossbar provides the routing through a physical link which is created when the first word of the burst enters the crossbar and is closed when the last word leaves the crossbar.

PivotPoint is a system level architecture that is built on top of the Nexus crossbar switch. Figure 5.26 shows a simplified PivotPoint architecture. In addition to the Nexus crossbar switch, the first-in-first-out (FIFO) buffer provides data buffering function for the transmit (TX) and the receive (RX) channels. The System Packet Interface (SPI-4.2) implements a standard protocol for chip-to-chip communication at data rates of 9.9 to 16 Gbps.

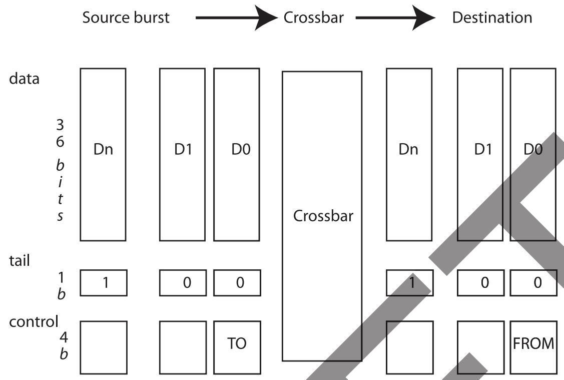


Figure 5.25 Format of Burst used on Nexus [9].

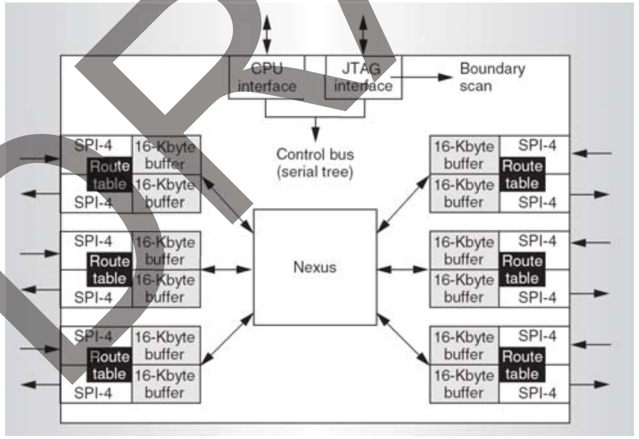


Figure 5.26 PivotPoint Architecture [9].

5.8.2 Blocking vs. Non-blocking

Nexus and PivotPoint are designed to avoid head-of-the-line (HOL) blocking. HOL blocking occurs when one packet failing to progress results in other unrelated packets behind it to be blocked. PivotPoint uses virtual channels (also called ports) to transport separated traffic streams simultaneously. Blocked packets in one channel only blocks packets behind it on the same channel. Packets on other channels are free to progress. In this way communication stalls are minimized.

5.9 Network-On-Chip (NOC)

Network-on-Chip (NOC) is an approach to SOC interconnects that promises to overcome a number of limitations found in the conventional bus-based approach [6]. Although the bus standards discussed earlier provide some degree of portability and reusability of IP cores, they are difficult to adapt to advancements in both process and bus interface technologies in the future. The fundamental weakness of buses is that they do not take a layered approach to interconnection, i.e., there is no explicit separation between the transaction level communication in the application layer and the interconnect signals in the physical layer. In contrast, activities in NOC systems are generally separated into transaction, transport and physical layers as depicted in Figure 5.27. As a result, NOC systems can easily be adapted to the rapid advances in process technology or system architecture.

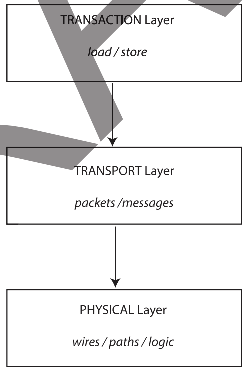


Figure 5.27 The layered architecture of NOC [5].

Figure 5.28 shows a general-purpose on-chip interconnect network comprising of a number of modules such as processors, memories and IP blocks organized as tiles. These module tiles are connected to the network that routes packets of data between them. All communications between tiles are via the network. The area overhead of the network logic can be as low as 6.6% [11]. The key characteristics of such NOC architecture are: 1) layered architecture which is easily scalable; 2) flexible network topology which can be configured by the user to optimize performance for different

applications; 3) point-to-point communication effectively decouples the IP blocks from each other.

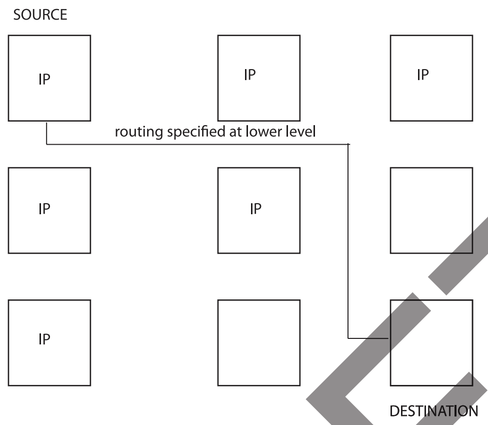


Figure 5.28 A typical NOC architecture [5].

5.9.1 NOC Layered Architecture

Most NOC architectures adopt a 3-layered communication scheme (see Figure 5.29). The **physical layer** specifies how packets are transmitted over the physical interfaces. Any changes in process technology, interconnecting switch structure and clock frequency affect only this layer. Upper layers are not compromised in any way. The **transport layer** defines how packets are routed through the switch fabric. A small header cell in the packet is typically used to specify how routing is to be done. The **transaction layer** defines the communication primitives used to connect the IP blocks to the network. The NOC Interface Unit (NIU), analogous to the Network Interface Card (NIC) in computer networks, provides the transaction level services to the IP block, governing how information is exchanged between NIUs to implement a particular transaction.

The layered architecture of NOC offers a number of benefits [5]:

1. *Physical and transport layers can be independently optimized* - the physical layer is governed mostly by process technology while the transaction layer is dependent on the particular application. The layered approach allows them to be separately optimized without affecting each other.

2. *Inherently scalable* - a properly designed switch fabric in a NOC can be scaled to handle any amount of simultaneous transactions. The distributed nature of the architecture allows the switches to be optimized to match the requirements. At the same time, the NIU responsible for the transaction layer can be designed to satisfy the performance requirement of the IP block that it services with no effect on the configuration and performance of the switch fabric.

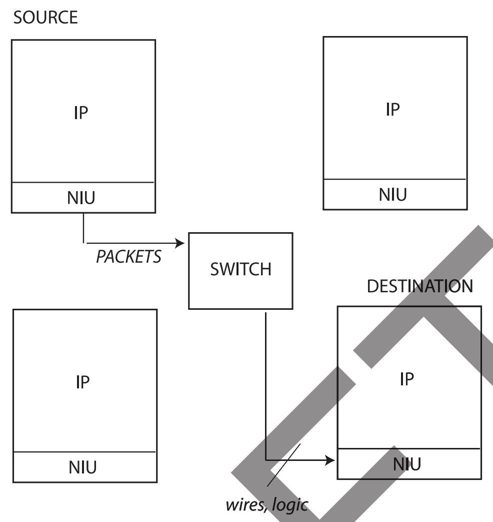


Figure 5.29 The Transaction, Transport and Physical Layers of a NOC [5].

3. *Better control of quality-of-service* - rules defined in the transport layer can be used to distinguish between time-critical and best-effort traffic. Prioritizing packets helps to achieve quality-of-service requirements enabling real-time performance on critical modules.

4. *Flexible throughput* - by allocating multiple physical transport links, throughput can be increased to meet the demand of a system statically or dynamically.

5. *Multiple clock domain operation* - since the notion of a clock only applies to the physical layer and not to the transport and transaction layers, a NOC is particularly suited to a SOC system containing IP blocks that operate at different clock frequencies. Using suitable clock synchronization circuits at the physical layer, modules with independent clock domains can be combined with reduced timing convergence problems.

5.9.2 Bus vs. NOC

When compared with buses, NOC is not without drawbacks. Perhaps the most significant weakness of NOC is the extra latency that it introduces. Unlike data communication networks, where quality of service is governed mainly by bandwidth and throughput, SOC applications usually also have very strict latency constraints. Furthermore, the NIU and the switch fabric add to the area overhead of the system. Therefore direct implementation of a conventional network architecture in SOC generally results in unacceptable area and latency overheads. Figure 5.30 presents the pros and cons between buses and NOC approaches to SOC interconnect qualitatively.

Bus Pros & Cons	**NOC Pros & Cons**
Every unit attached adds parasitic capacitance (-)	Only point-to-point one-way wires are used for all network sizes (+)
Bus timing is difficult in deep sub-micron process (-)	Network wires can be pipelined because the network protocol is globally asynchronous (+)
Bus testability is problematic and slow (-)	Dedicated BIST is fast and complete (+)
Bus arbiter delay grows with the number of masters. The arbiter is also instance-specific (-)	Routing decisions are distributed and the same router is reinstanciated, for all netowrk sizes (+)
Bandwidth is limited and shared by all units attached (-)	Aggregated bandwidth scales with the network size (+)
Bus latency is zero once arbiter has granted control (+)	Internal network contention causes a small latency (-)
The silicon cost of a bus is low for small systems (+)	The network has a significant silicon area (-)
Any bus is almost directly compatible with most available IPs, including software running on CPUs (+)	Bus-oriented IPs need smart wrappers. Software needs clean synchronization in multiprocessor systems (-)
The concepts are simple and well understood (+)	System designers need re-education for new concepts (-)

Figure 5.30 The bus-versus-NOC arguments [14].

5.10 Conclusions

The interconnect subsystem is the backbone of the SOC. The system's performance can be throttled by limitations in the interconnect. Because of its importance, a great deal of attention has been afforded to optimum cost-performance interconnect strategies.

From a cursory view, it appears that there are three distinct approaches to SOC interconnect: bus based, switch based and network based (NOC). A closer examination shows that these are most often complementary approaches. Indeed a NOC will include one or more switches, connecting nodes which can themselves be a bus based cluster of processors or other IPs.

In the past most SOCs were predominantly bus based. The number of nodes to be connected were small (perhaps 4 or 8 IPs), and each node consisted solely of a single IP. This was a tried and tested method of interconnect that was both familiar and easy to use. Even now the use of standard protocols and bus wrappers make the task of IP core integration less error-prone. Also the large number of bus options available allows users to trade-off between complexity, easy of use, performance, and universality.

As the number of interconnected nodes increases, the bandwidth limitations of bus-based approaches becomes more apparent. Switches overcome the bandwidth limitations but with additional cost and, depending on the configuration, additional latency. As switches (whether static or dynamic) are translated into IP and supported with experience and the emergence of tools, they will become the standard SOC interconnect especially for high-performance systems.

Modeling the performance of either bus or switch is an important part of the SOC design. If initial analysis of bus-based interconnection demonstrates insufficient bandwidth and system performance, switch-based design is the alternative. Initial analysis and design selection is usually based on analytic models; but once the selection has been narrowed to a few alternatives, a more thorough simulation should be used to validate the final selection. The performance of the SOC will depend on the configuration and capability of the interconnection scheme.

Network-on-Chip is a promising extension. For a relatively small overhead, it enables a layering of the interconnect implementation. This allows designs to be re-engineered and extended to include new switches, etc. without affecting the upper level SOC implementation. Growth in NOC adoption will facilitate easier SOC development.

5.11 Problem Set

1. A tenured split (address plus bi directional data bus) bus is 32 plus 64 bits wide. A typical bus transaction (read or write) uses a 32 bit memory address and sub sequentially has a 128 bit data transfer. If the memory access time is 12 cycles,
   (a) Show a timing diagram for a read and a write (assuming no contention).
   (b) What is the (data) bus occupancy for a single transaction?
2. If 4 processors use the bus described above and ideally (without contention) each processor generates a transaction every 20 cycles:
   (a) What is the offered bus occupancy?
   (b) Using the bus model without resubmissions, what is the achieved occupancy?
   (c) Using the bus model with resubmissions, what.s the achieved occupancy?
   (d) What is the effect on system performance for the (b) and (c) results?
3. Search for current products that use the AMBA bus; find at least three distinct systems and tabularize their respective parameters (AHB and APB): bus width, bandwidth, maximum number of IP users per bus. Provide additional details as available.
4. Search for current products that use the CoreConnect bus; find at least three distinct systems and tabularize their respective parameters (PLB and OPB): bus width, bandwidth, maximum number of IP users per bus. Provide additional details as available.

5. Discuss some of the problems that you would expect to encounter in creating a bus wrapper to convert from an AMBA bus to a CoreConnect bus.
6. A static switching interconnect is implemented as a 4x4 torus (2D) with worm hole routing. Each path is bidirectional with 32 wires; each wire can be clocked at 400 Mbps. For a message consisting of an 8 bit header and 128 bit "payload".
   (a) What is the expected latency (in cycles) for a message to transit from one node to an adjacent node?
   (b) What is the average distance between nodes and the average message latency (in cycles)?
   (c) If the network has occupancy of 0.4, what is the delay due to congestion (waiting time) for the message?
   (d) What is the total message transit time?
7. A dynamic switching interconnect is to connect 16 nodes using a baseline switching network implemented with 2x2 crossbars. It takes one cycle to transit a 2x2. Each path is bidirectional with 32 wires; each wire can be clocked at 400 Mbps. For a message consisting of an 8 bit header and 128 bit "payload".
   (a) What is the expected latency (in cycles) for a message to transit from one node to any other?
   (b) Draw the network.
   (c) What is the message waiting time, if the network has occupancy of 0.4?
   (d) What is the total message transit time?
8. The bisection bandwidth of a switching interconnect is defined as the maximum available bandwidth across a line dividing the network into two equal parts (number of nodes). What is the bisection bandwidth for the static and dynamic networks outlined above?
9. Search for at least three distinct NOC systems; compare their underlying switches (find at least one dynamic and one static example). Provide details in table form.

Bibliography

[1] S. Abraham and K. Padmanabhan. Performance of direct binary n-cube networks for multiprocessors. *IEEE Transactions on Computers*, 38(7):1000–1111, July 1989.

[2] A. Agarwal. Limits on interconnection network performance. *Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[3] K. Ajo, A. Okamura and M. Motomura. Wrapper-Based Bus Implementation Techniques for Performance Improvement and Cost Reduction. *IEEE Journal of Solid-State Circuits*, 39(5):804-817, May 2004.

[4] ARM. AMBA Specification. Rev 2.0, ARM-IHI-0011A.

[5] Arteris. A comparison of Network-on-Chip and Busses, White Paper, 2005.

[6] L. Benini and G. De Micheli. Networks on chips: A new SOC paradigm. *IEEE Computer*, pp. 70–78, 2002.

[7] M. Birnbaum and H. Sachs. How VSIA Answers the SOC Dilemma. *IEEE Computer*, pp. 42–50, June 1999.

[8] CrossBow Technologies Inc. Xfabric Core Connectivity Junction. Preliminary Product Specification, May 16 2004.

[9] U. Cummings. PivotPoint: Clockless Crossbar Switch for High-Performance Embedded Systems. *IEEE Micro*, pp. 48–59, Mar-Apr 2004.

[10] W.J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6), June 1990.

[11] W.J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *DAC 2001*, 2001.

[12] W.J. Dally and C.L. Seitz. *Technical report 5208:tr:86*, Computer Science Department, Caltech. pp. 1–19, 1986.

[13] D. Flynn. AMBA: Enabling Reusable On-chip Designs. *IEEE Micro*, pp. 20–27, Jul-Aug 1997.

[14] P. Guerrier and A. Grenier. A generic architecture for on-chip packet-switched interconnections. *Proc. IEEE Design Automation and Test in Europe (DATE 2000)*, pp. 250–256, 2000.

[15] J. Hu and R. Marculescu. Application-specific buffer space allocation for networks-on-chip router design. *IEEE/ACM International Conference on Computer Aided Design*, pp. 354–361, 2004.

[16] K. Hwang and F.A. Briggs. *Computer Architecture and Parallel Processing*, McGraw-Hill series in computer organization and architecture, 1984.

[17] IBM. 128-bit Processor Logic Bus – Architecture Specification. Version 4.4, SA-14-2538-02, May 2001.

[18] IBM. On-chip Peripheral Bus - Architecture Specification. Version 2.1, SA-14-2528-02, April 2001.

[19] J. R. Jump and S. Lakshmanamurthy. NETSIM: A general-purpose interconnection network simulator. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 121–125, January 1993.

[20] F. Karim, A. Nguyen and S. Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, 2002.

[21] C. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C-32(12), December 1983.

[22] S. Kumar. A network on chip architecture and design methodology. *Proc. International Symposium in VLSI*, pp. 117–124, 2002.

[23] D. Lawrie. Access and alignment of data in an array processor. *IEEE Transactions on Computers*, C-24(12):1145–1154, December 1975.

[24] A. Lines. Asynchronous Interconnect for Synchronous SoC Design. *IEEE Micro*, pp. 32–41, Jan-Feb, 2004.

[25] D. Lyonnard, S. Yoo, A. Baghdadi and A.A. Jerraya. Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip. *Colloq. CAO*, pp. 15–18, May 2002.

[26] J. McGregor. Interconnects Target SoC Design. *Microprocessor Report*, 2004.

[27] Open Core Protocol International Partners. Open Core Protocol Specification 1.0. *OCP-IP Association*, Document Version 002, 2001.

[28] P.P. Pande, C. Grecu, M. Jones, A. Ivanov and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Transactions on Computers*, 54(8):1025–1040, Aug 2005.

[29] J. H. Patel. Performance of processor–memory interconnections for multiprocessors. *IEEE Transactions on Computers*, C-30(10):771–780, October 1981.

[30] P. Pelgrims. Overview on: AMBA, AVALON, CORECONNECT, WISHBONE. *Evaluation Report*, De Nayer Instituut, version 1.1, 2003.

[31] C.H. Pyoun, C.H. Lin, H.S. Kim and J.W. Chong. The efficient bus arbitration scheme in soc environment. *IEEE International Workshop on System-on-Chip for Real-Time Applications*, pp. 311–315, 2003.

[32] E. Salminen, V. Lahtinen, K. Kuusilinna and T. Hamalainen. Overview of Bus-based System-on-Chip Interconnections *IEEE International Symposium on Circuits and Systems*, 2:372–375, May 2002.

[33] Sonics Inc. Sonics μNetwork Technical Overview. *Sonics Inc.*, Document Revision 1-2002.

[34] Virtual Socket Interface Alliance. On-Chip Bus DWG. Virtual Component Interface (VCI) Specification Version 2. OCB 2 2.0, 2001.

[35] C.-L. Wu and T.-Y. Feng. On a class of multistage interconnection networks. *IEEE Transactions on Computers*, pp. 696–777, August 1980.