Basics of SoC Bus Interconnection

COE838/EE8221: Systems-on-Chip Design
http://www.ecb.torontomu.ca/~courses/coe838/

Dr. Gul N. Khan
http://www.ecb.torontomu.ca/~gnkhan
Electrical, Computer & Biomedical Engineering
Toronto Metropolitan University

Bus Interconnection Overview

- Physical structure
- Clocking
- Arbitration and decoding
- Topology types
- Data transfer modes

Outline

Bus-based Communication Preliminaries

- Terminology
- Physical structure
- Clocking
- Arbitration and decoding
- Topology types
- Data transfer modes
- Physical implementation issues

Communication-centric Design

- Communication is the most critical aspect affecting system performance
- Communication architecture consumes upto 50% of total on-chip power
- Ever increasing number of wires, repeaters, bus components (arbiters, bridges, decoders etc.) increases system cost
- Communication architecture design, customization, exploration, verification and implementation takes up the largest chunk of a design cycle

Communication Architectures in complex systems and SoCs **significantly** affects **performance, power, cost & time-to-market!**

Interconnection Strategies

Tradeoffs between interconnection complexity/parallelism.
How to interconnect hardware modules?
Consider 4 modules (registers) capable of exchanging their contents. Methods of interconnection.

Notation for data swap: **SWAP(Ri, Rj)**

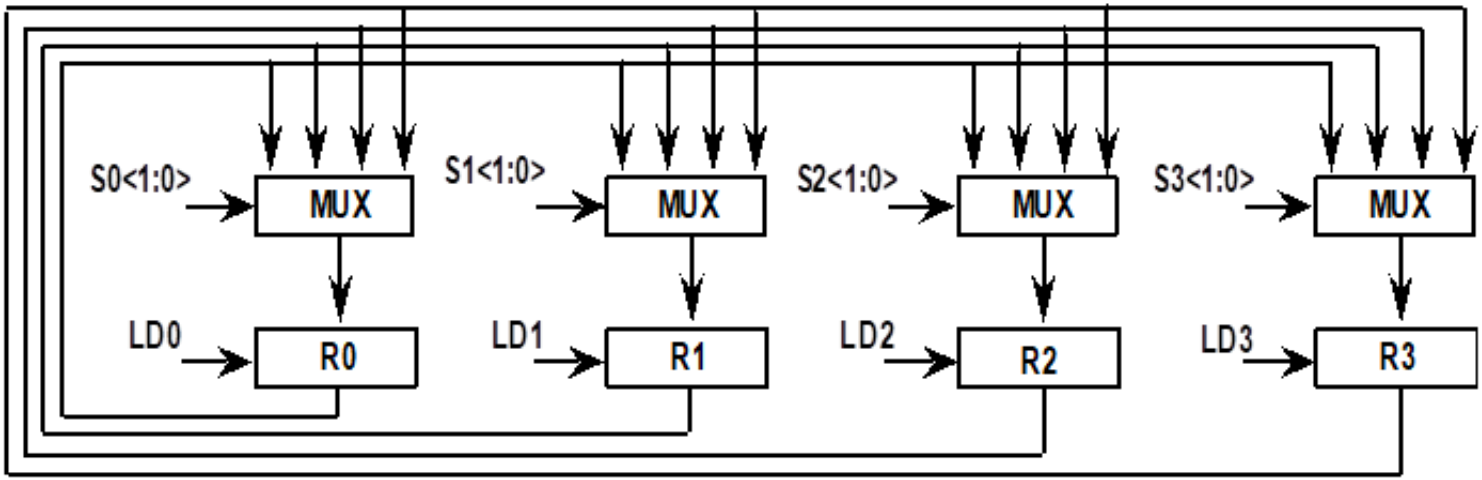
Ri <= Rj; **temp <= Ri;**
Rj <= Ri; **Rj <= temp;**

Module-to-Module Communication

- **Point-to-point**
- **Single shared bus**
- **Multiple special purpose buses**

Point-to-Point Connection

- Four Modules interconnected via 4:1 MUXes and point-to-point connections.



Modules have edge-triggered N-bit registers to transfer, controlled by LDi signals.

Nx4:1 Multiplexers per module (register) controlled by Si<1:0> control signals.

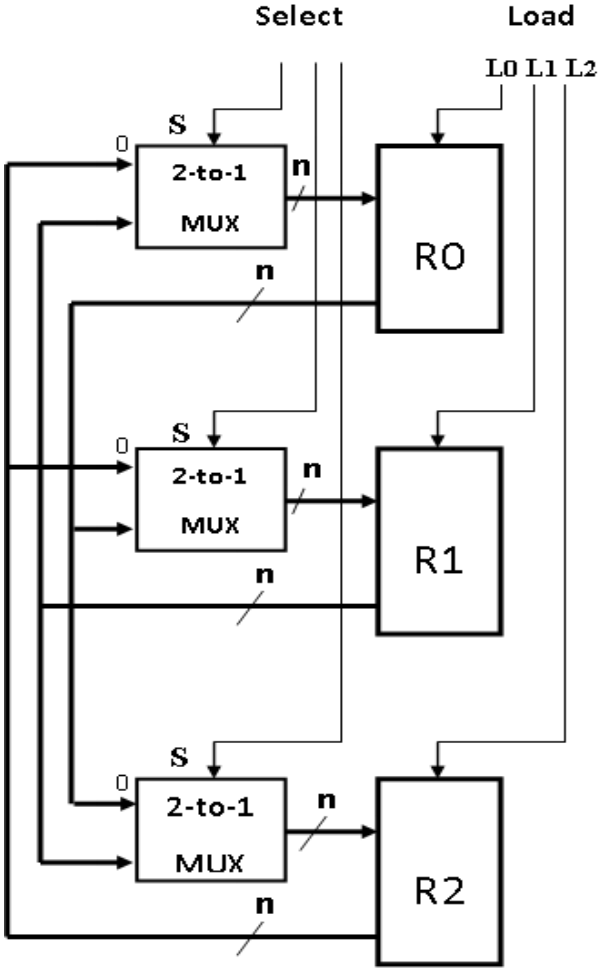
Control of SWAP Operation, SWAP(R1, R2) Control Signals

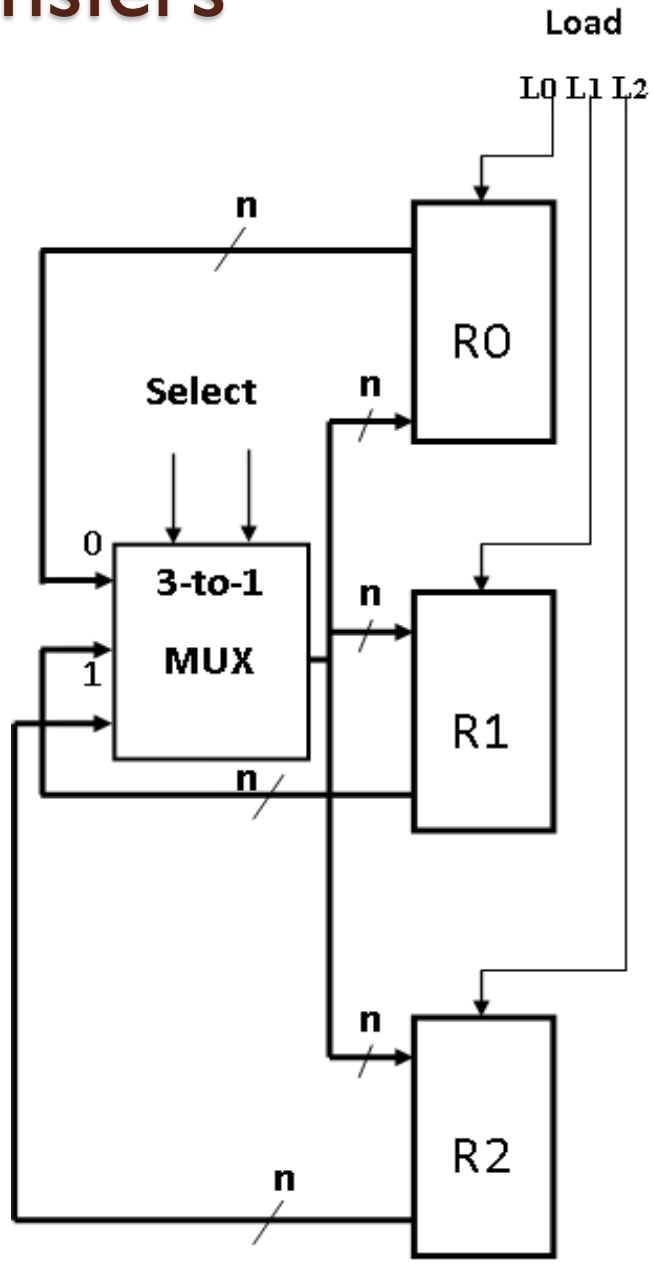
01 → S2<1:0>; Establish

10 → S1<1:0>; connection paths

1 → LD2; 1 → LD1; Swap takes place at next active clock

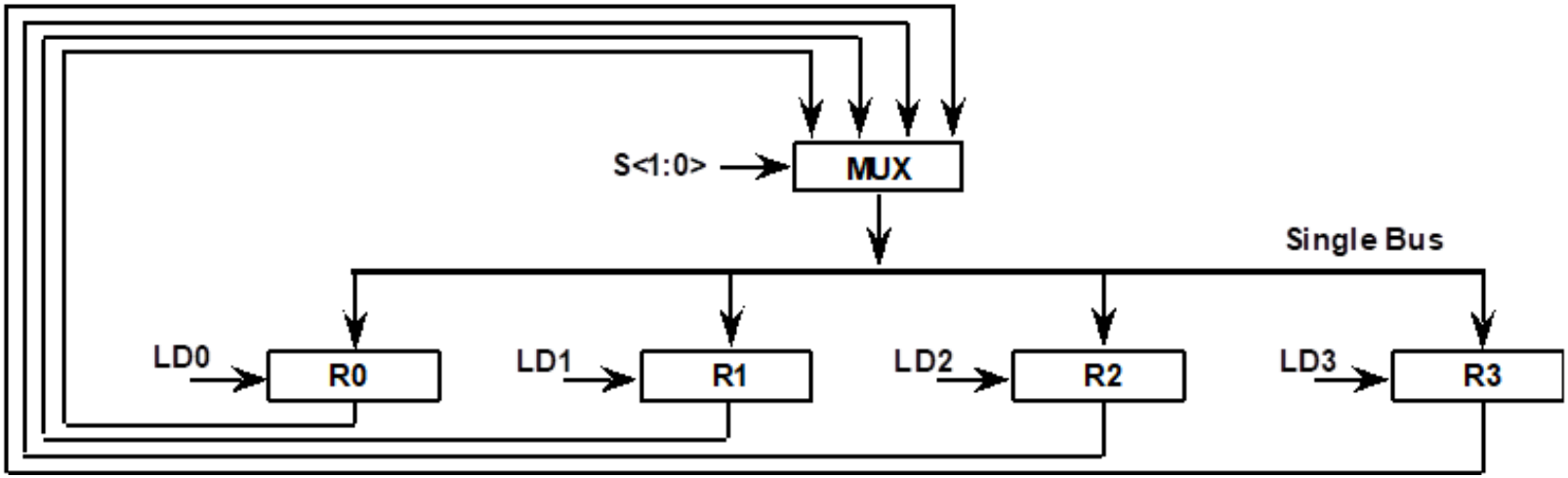
MUX and Bus Transfers





Transfer	S_1	S_0	L_2	L_1	L_0
R0 <= R2	1	0	0	0	1
R0 <= R1, R2 <= R1	0	1	1	0	1
R0 <= R1, R1 <= R0	*Impossible*				

Single Bus Interconnection



- Module MUX are replaced by a single MUX block.
- 25% hardware cost of the previous alternative.
- Shared set of inter-connection is called a BUS.

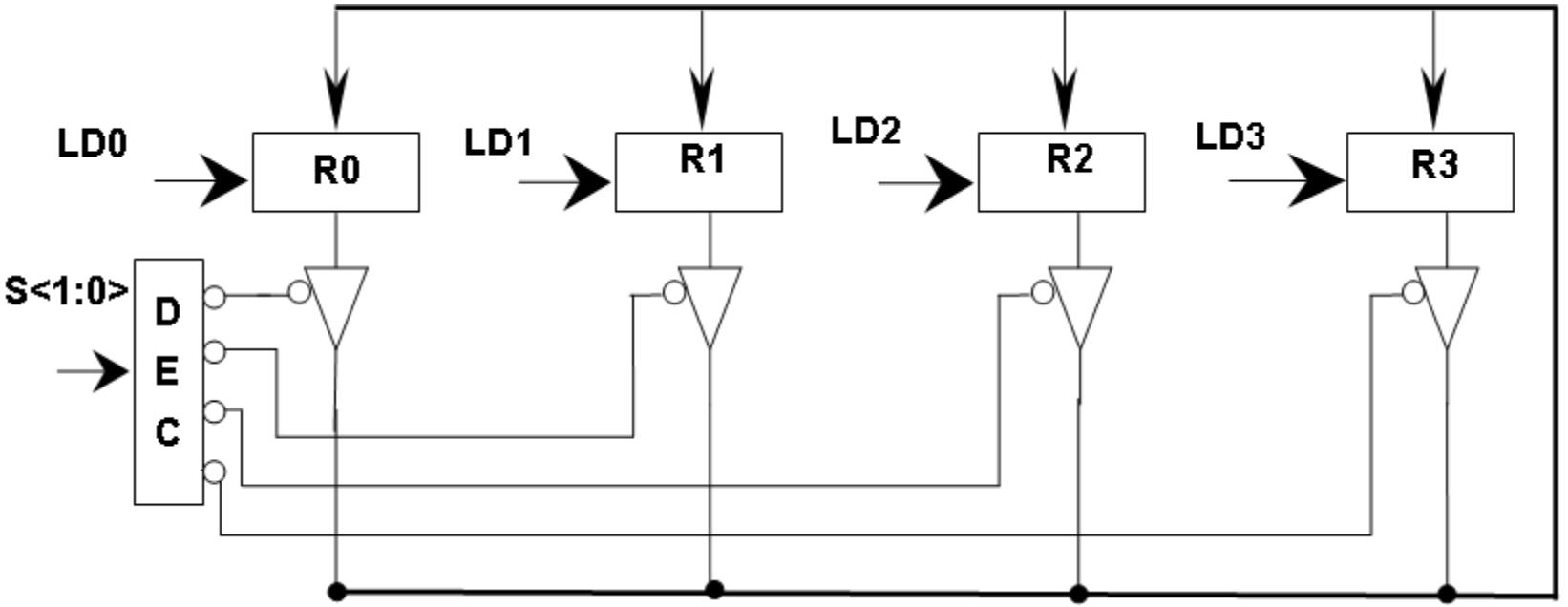
Multiple Transfers R0 <= R1 and R3 <= R2

State X: (R0 <= R1)
01 → S<1:0>;
1 → LD0;

State Y: (R3 <= R2)
10 → S<1:0>;
1 → LD3;

Two control states are required for transfers.

Alternatives to Multiplexers

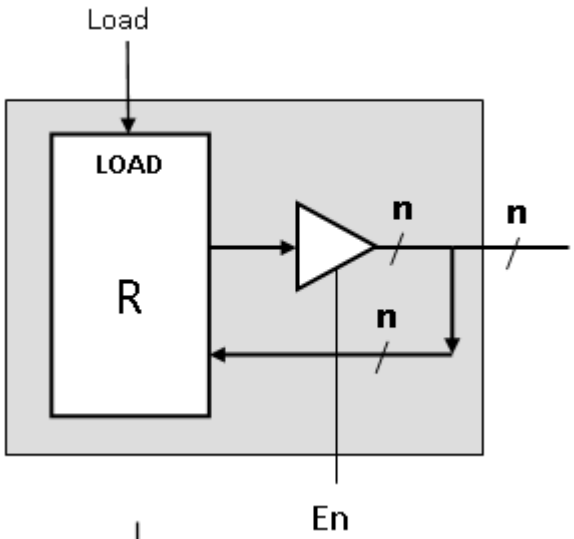


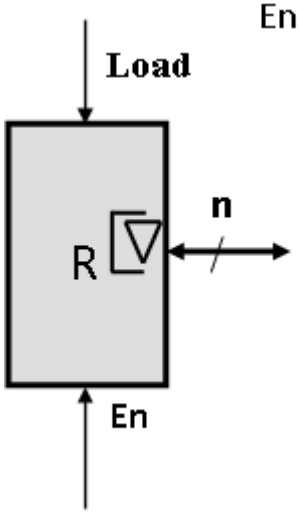
- Tri-state buffers as an interconnection scheme
- Reduces Interconnection Physical Wires

Only one contents gated to shared bus at a time.

Decoder decodes the input control lines S<1:0> and generates one select signal to enable only one tri-state buffer.

Tri-State Bus

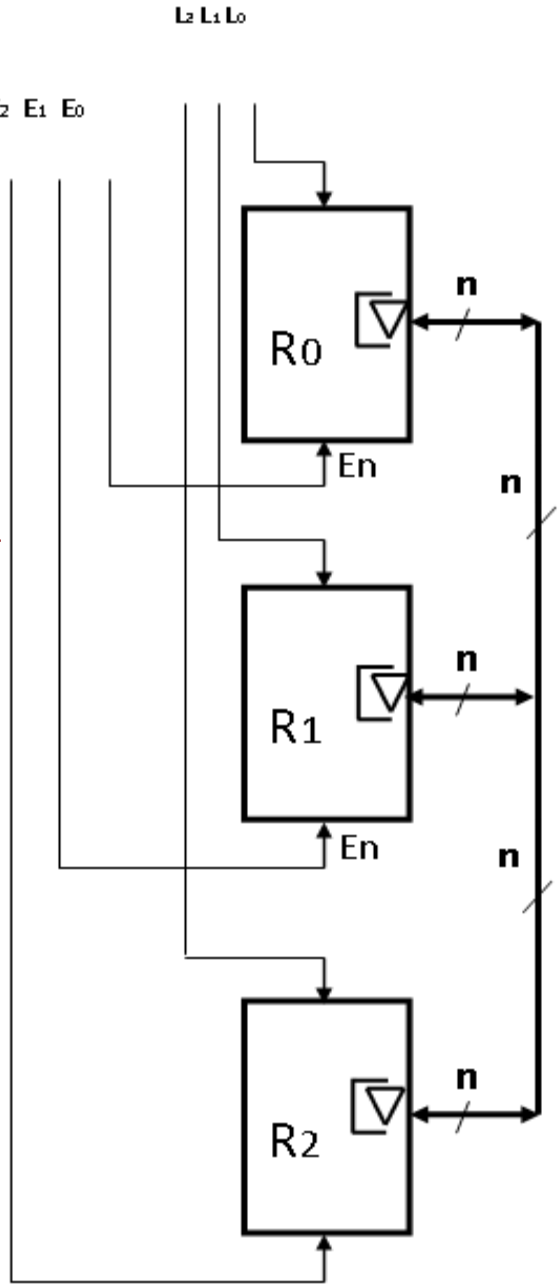




Register with bi-directional input-output lines

Tri-state bus using bi-directional lines.

Instead of separate Input and Output Lines



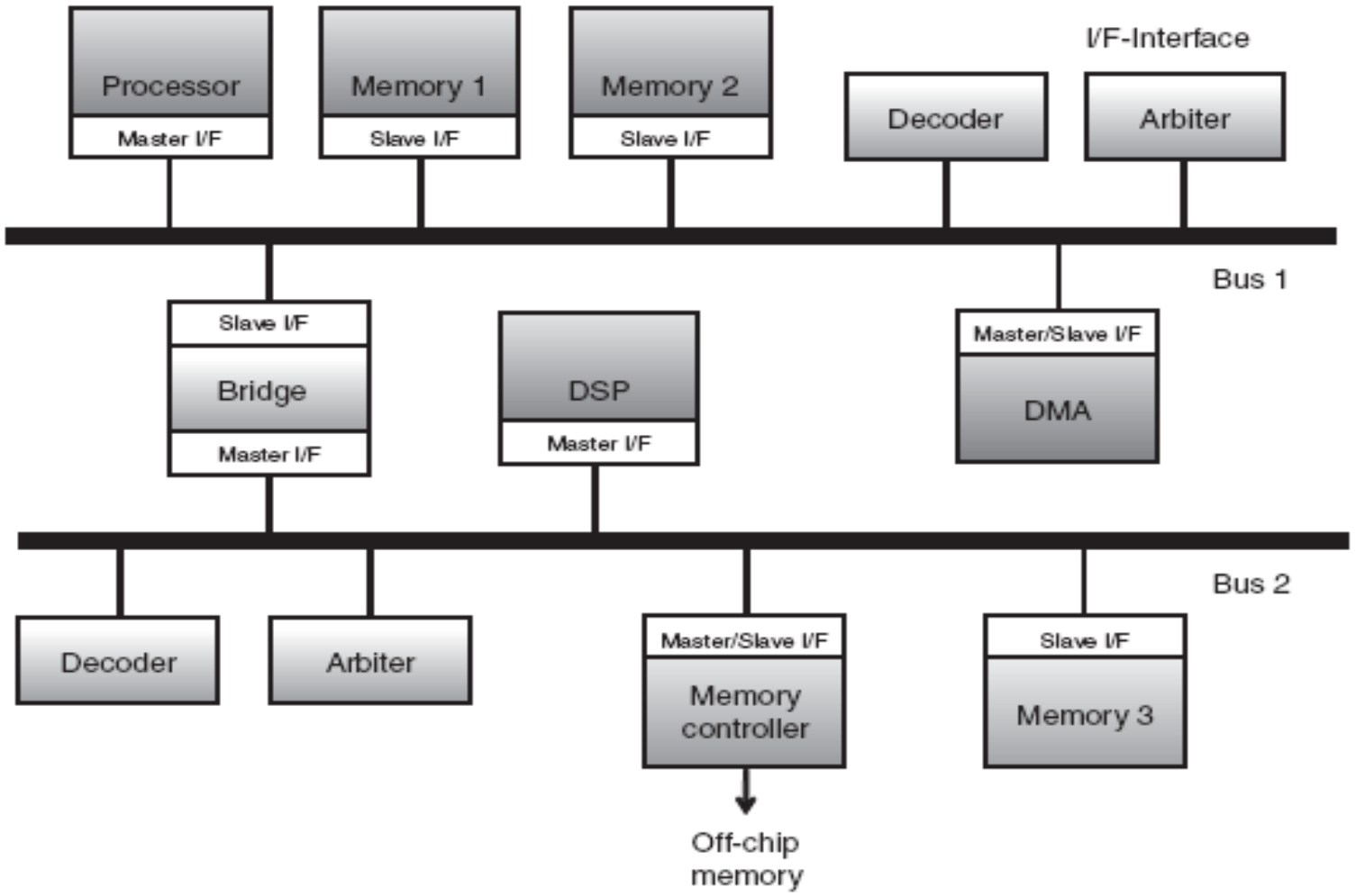
Bus: Basic Architecture

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

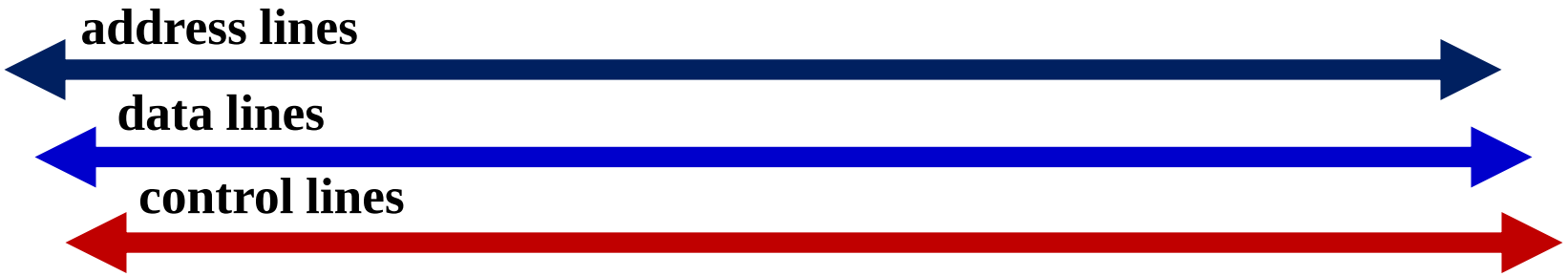
- Bus is made of wires shared by multiple units with logic to provide an orderly use of the bus.
- Devices can be Masters or Slaves.
- **Arbiter** determines - which device will control the bus.
- **Bus protocol** is a set of rules for transmitting information between two or more devices over a bus.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

Bus: Basic Architecture

Decoder determines the target for any transfer initiated by a master



Bus Signals



Typically a bus has three types of signal lines

Address

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

Data

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

Control

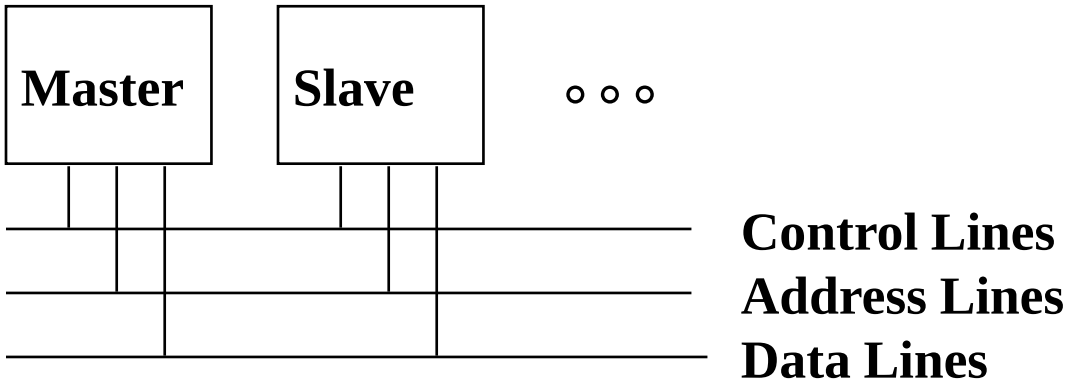
- Requests and acknowledgements
- Specify more information about type of data transfer e.g. Byte enable, burst size, cacheable/bufferable, …

Interconnect Architectures

IP blocks need to communicate among each other

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

Bus Basics



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

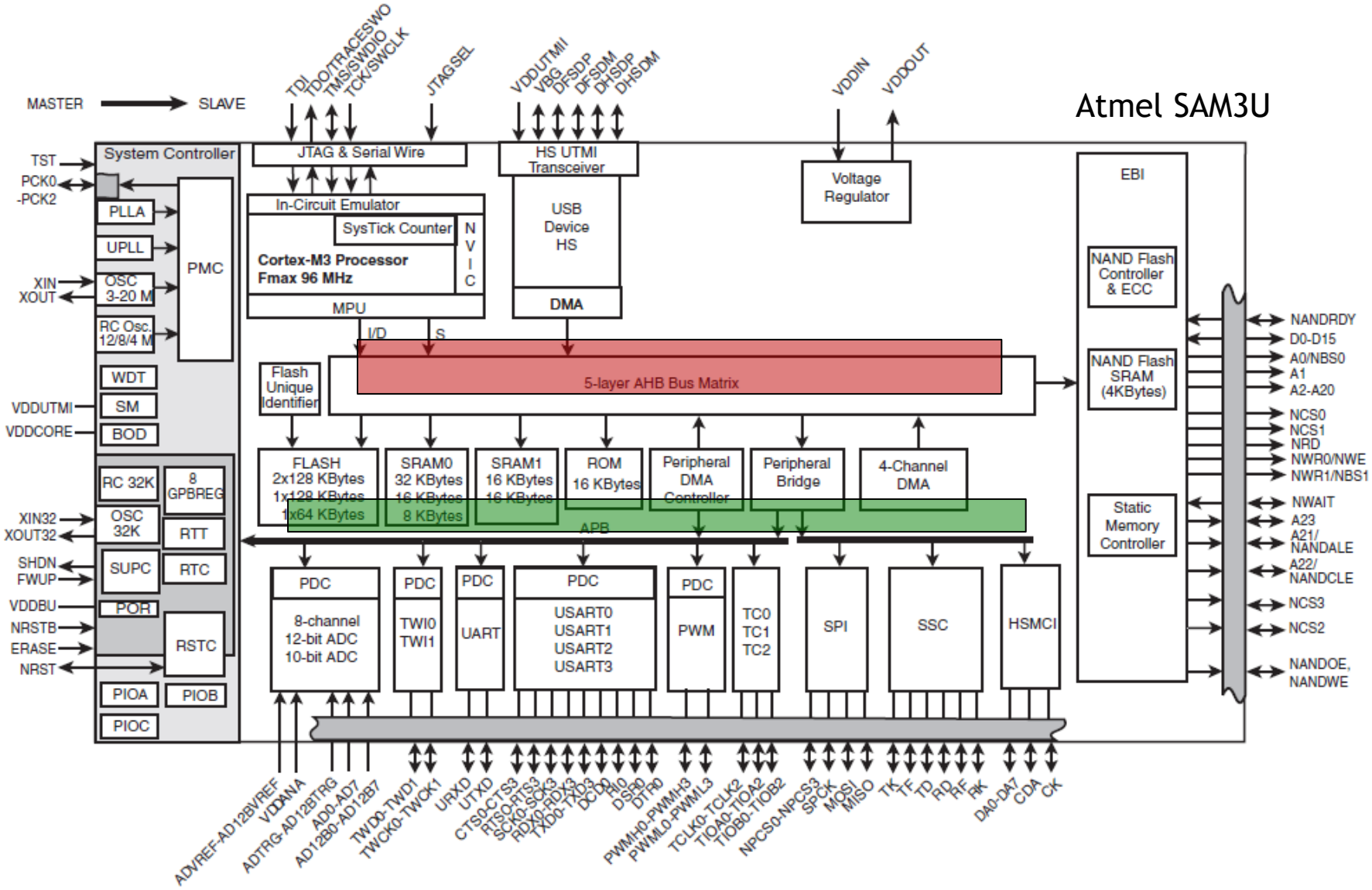
Bus Slave: module activated by the transaction

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

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

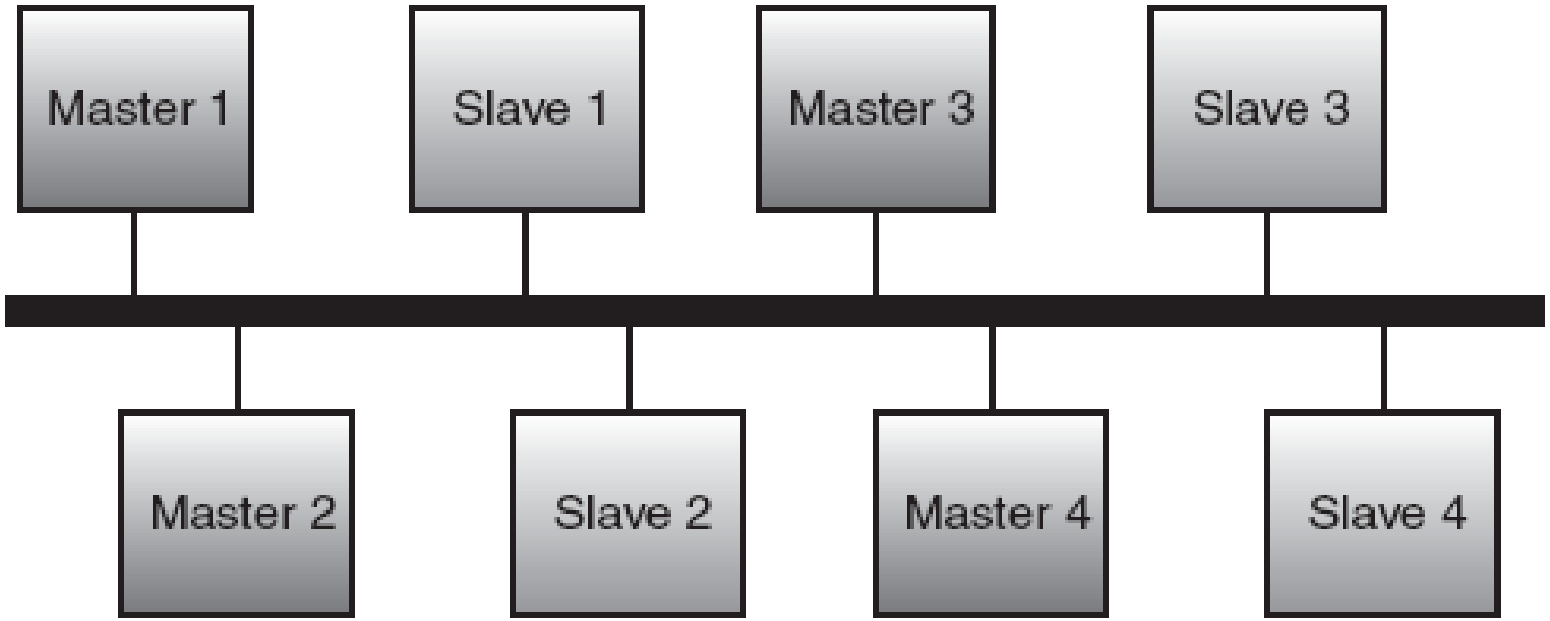
Synchronous Bus Transfers: sequence relative to common clock.

Embedded Systems busses



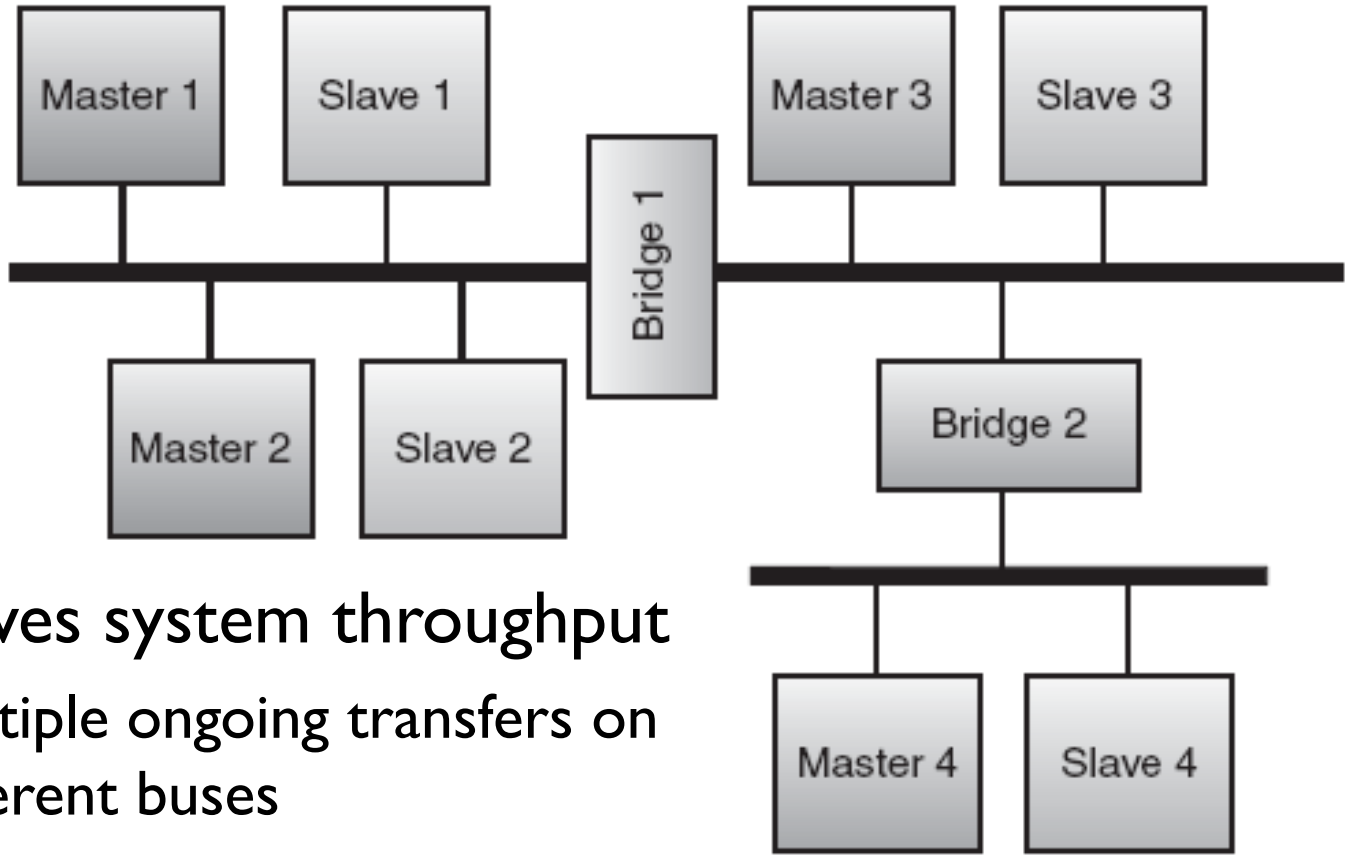
Types of Bus Topologies

Shared bus



Types of Bus Topologies

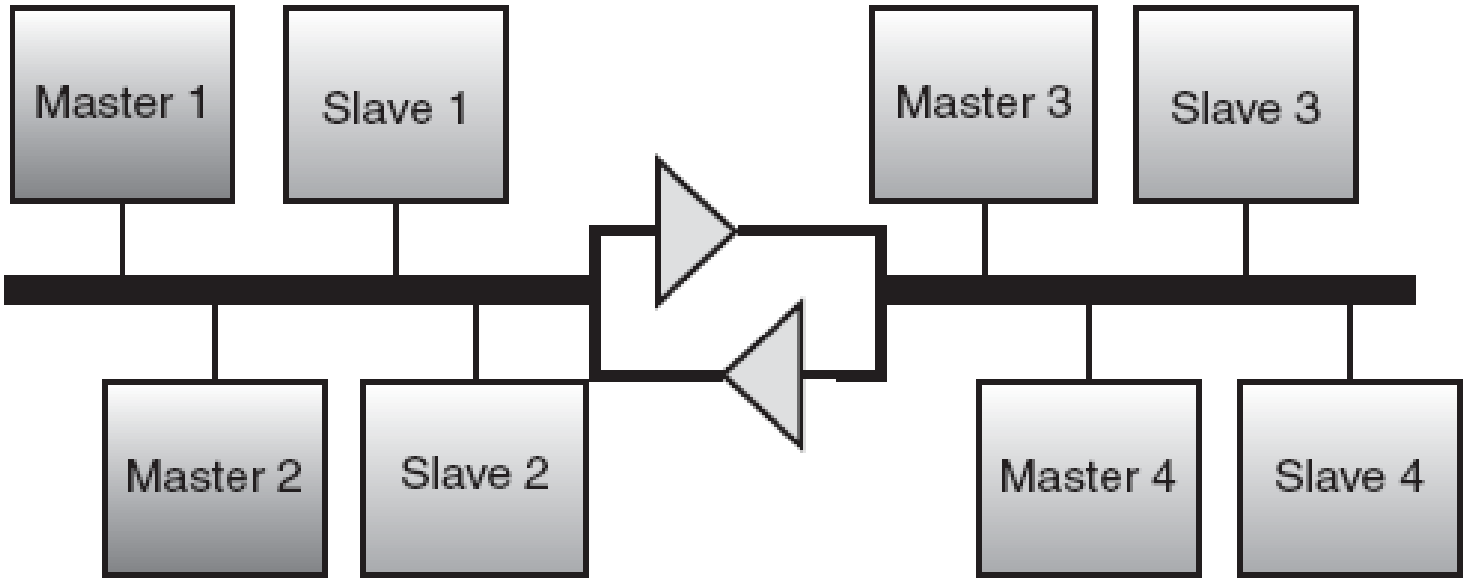
- Hierarchical shared bus



- Improves system throughput
  - Multiple ongoing transfers on different buses

Types of Bus Topologies

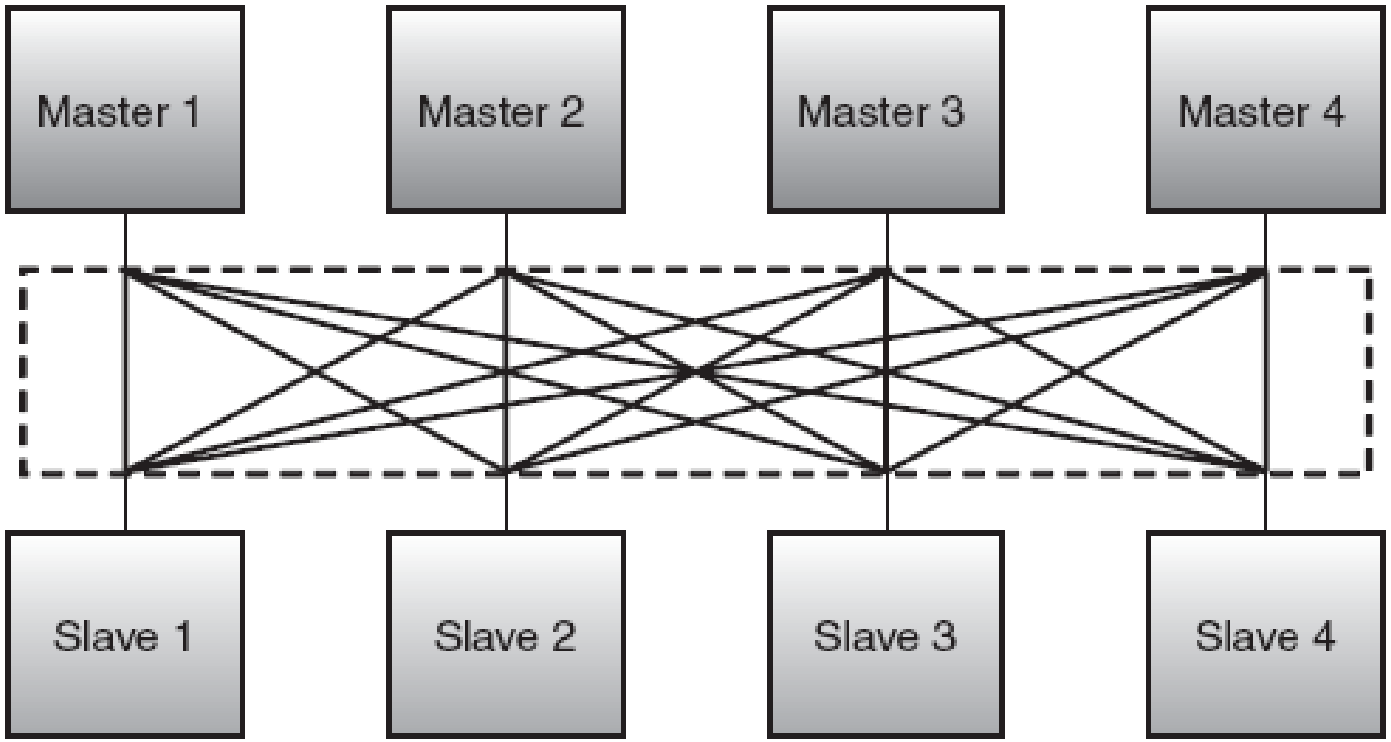
- Split bus



- Reduces impact of capacitance across two segments
- Reduces contention and energy

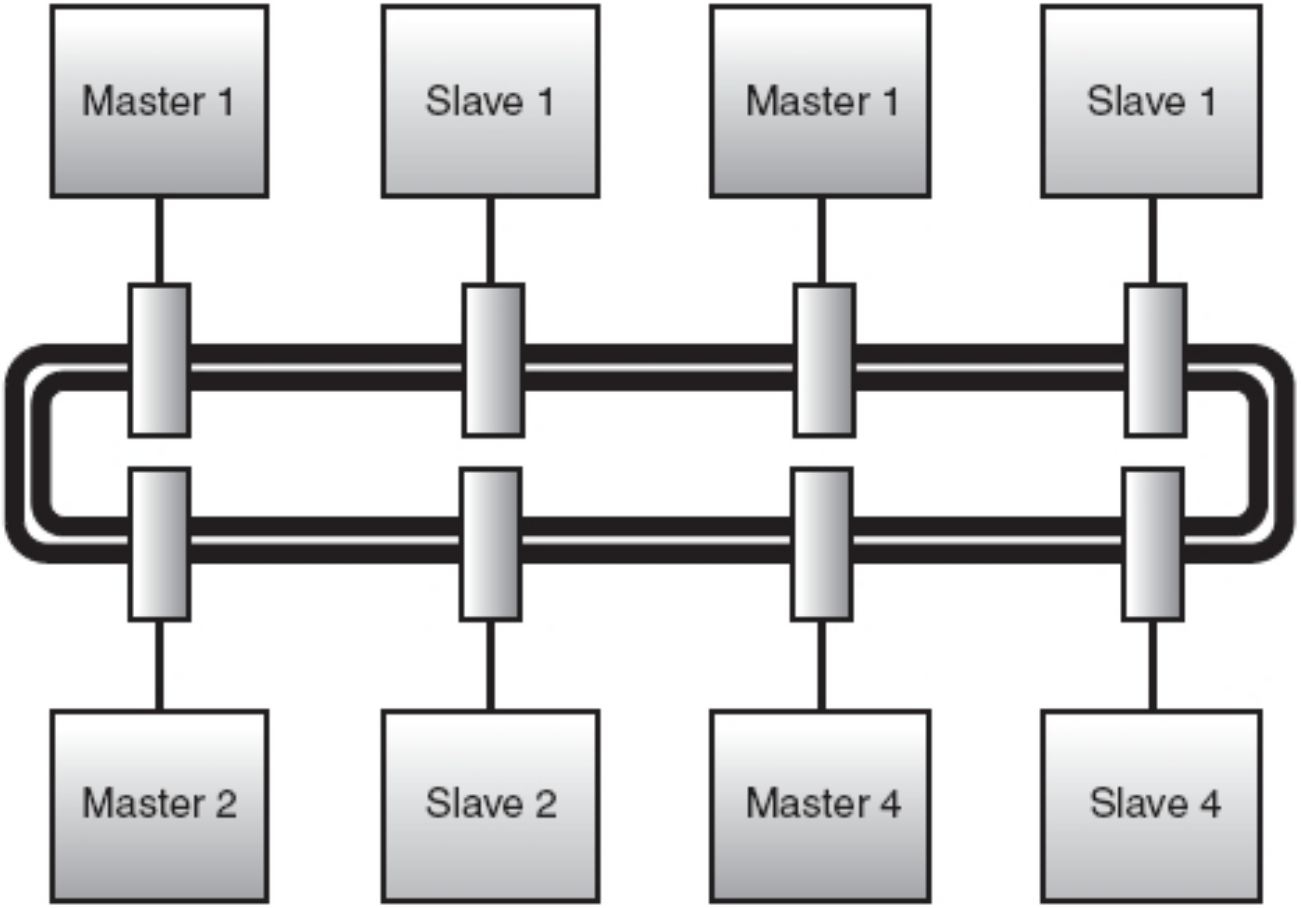
Types of Bus Topologies

- Full crossbar/matrix bus (point to point)



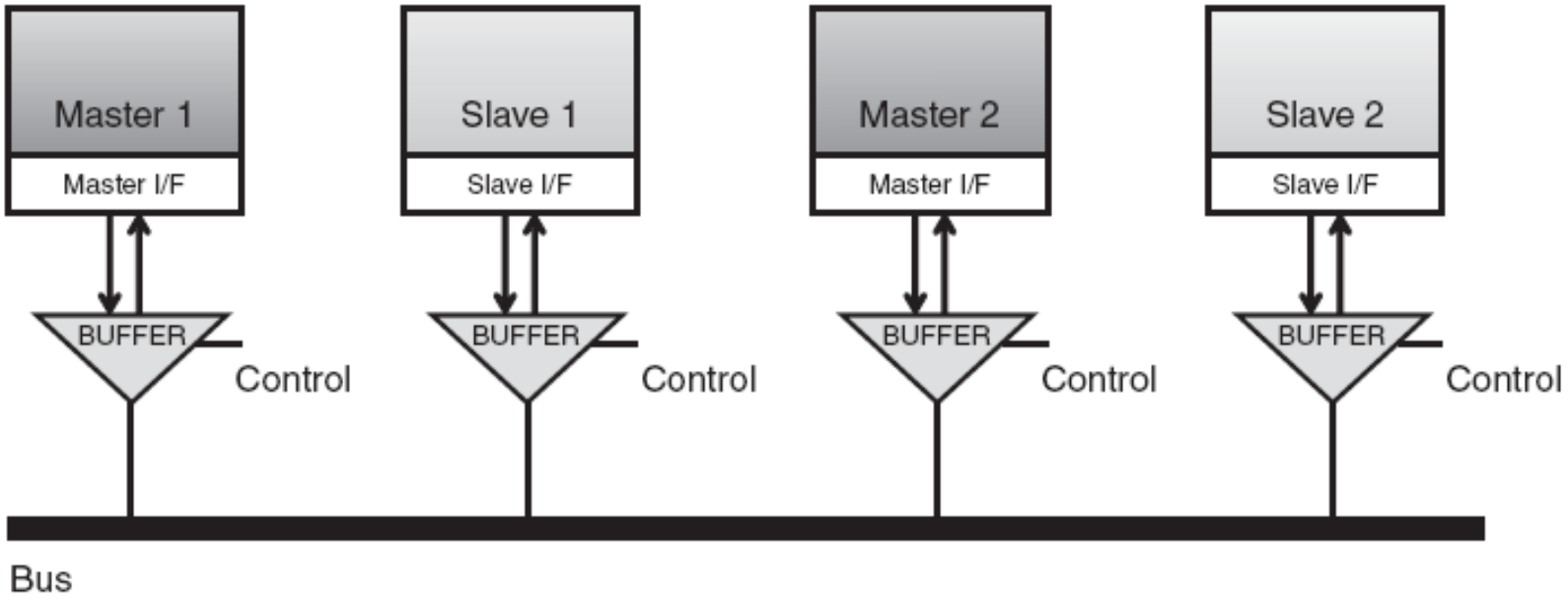
Types of Bus Topologies

Ring bus



Bus Physical Structure

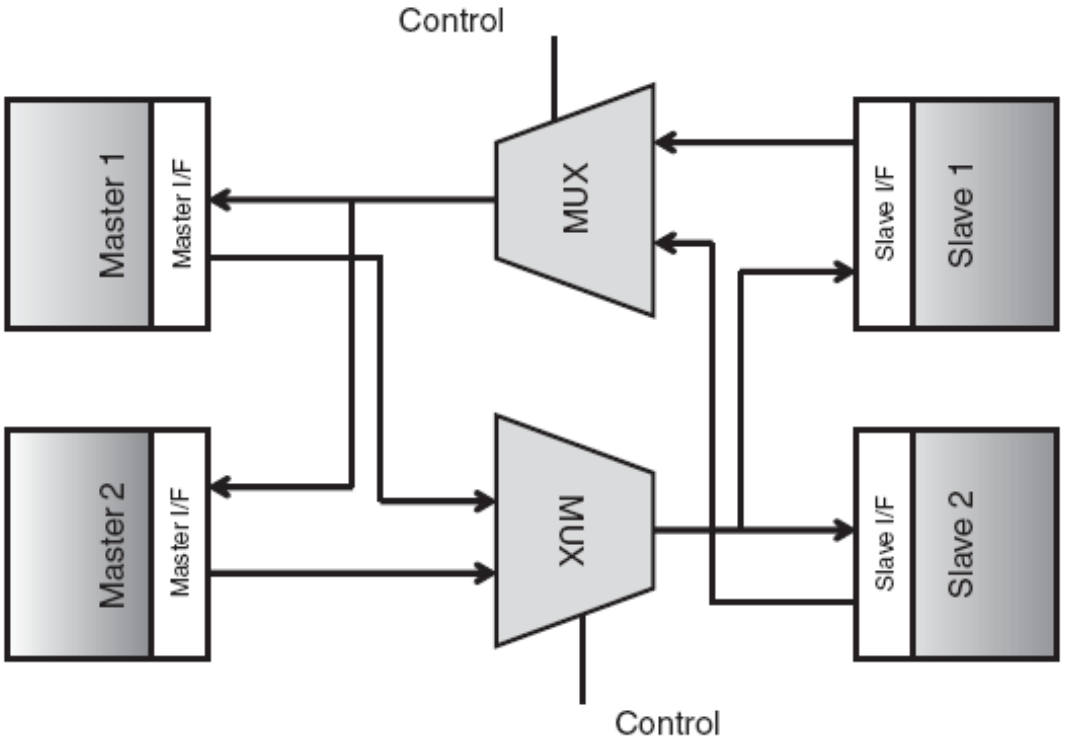
Tri-state buffer based bidirectional signals



- Commonly used in off-chip/backplane buses
  + take up fewer wires, smaller area footprint
  - higher power consumption, higher delay, hard to debug

Bus Physical Structure

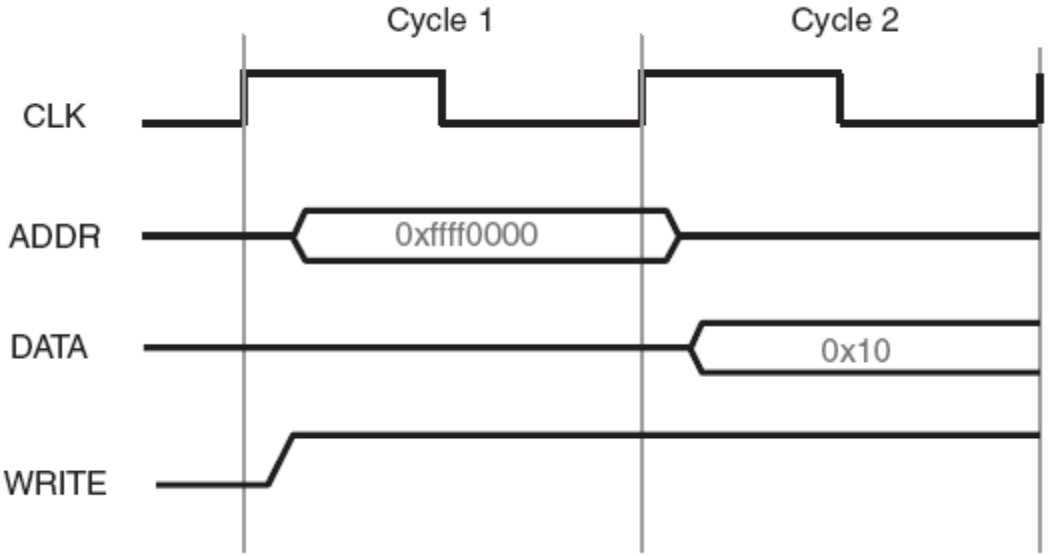
MUX based signals



Separate read, write channels

Bus Clocking

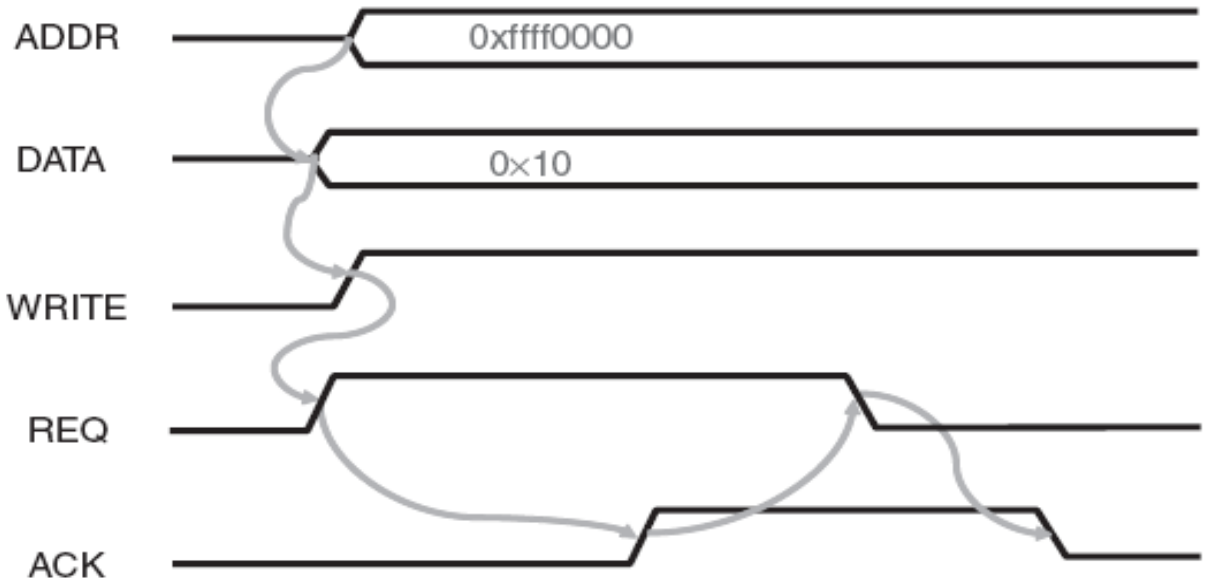
- Synchronous Bus
  - Includes a clock in control lines
  - Fixed protocol for communication that is relative to clock
  - Involves very little logic and can run very fast
  - Require frequency converters across frequency domains



Bus Clocking

Asynchronous Bus

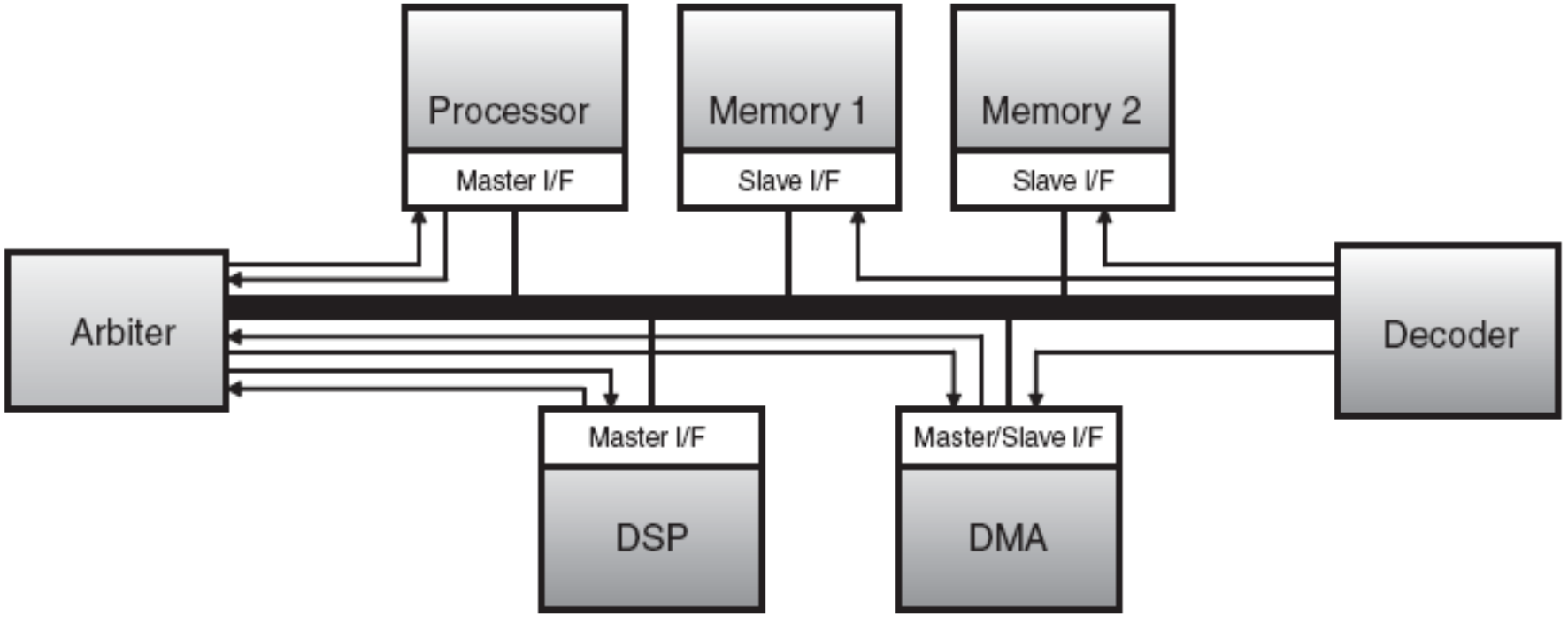
- No clock
- Requires a handshaking protocol
  - performance not as good as that of synchronous bus
  - No need for frequency converters, but does need extra lines
- No clock skew as in the synchronous bus



Decoding and Arbitration

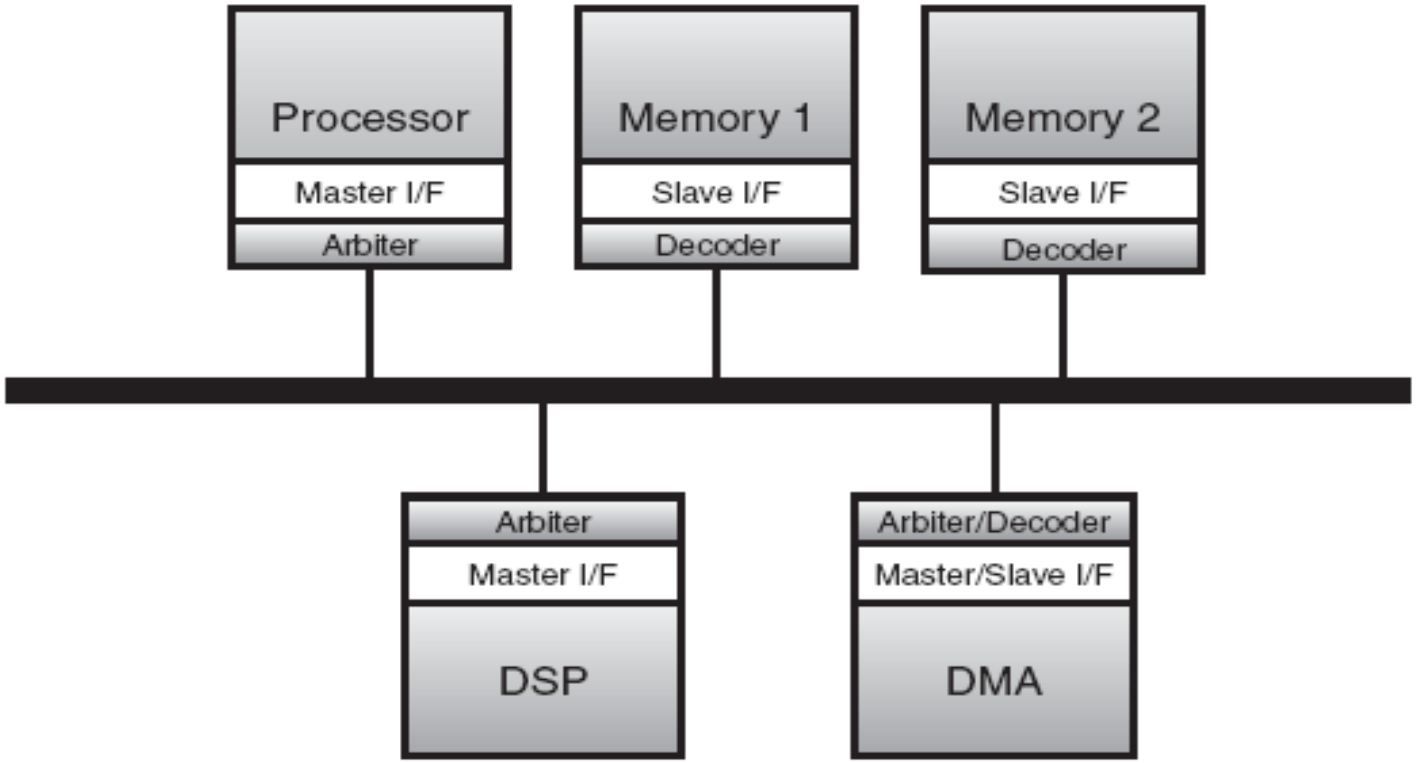
- Decoding
  - determines the target for any transfer initiated by a master
- Arbitration
  - decides which master can use the shared bus if more than one master request bus access simultaneously
- Decoding and Arbitration can either be
  - *centralized*
  - *distributed*

Centralized Decoding and Arbitration



- Minimal change is required if new components are added to the system

Distributed Decoding and Arbitration



+ requires fewer signals as compared to centralized method

- more hardware duplication and logic/ chip-area

Arbitration Schemes

- **Random:** Randomly select master to grant the bus access
- **Static priority**
  - Masters assigned static priorities
  - Higher priority master request always serviced first
  - Can be pre-emptive (AMBA-2) or non-preemptive (AMBA-3)
  - May lead to starvation of low priority masters
- **Round-Robin (RR)**
  - Masters allowed to access bus in a round-robin manner
  - No starvation – every master guaranteed bus access
  - Inefficient if masters have vastly different data injection rates
  - High latency for critical data streams

Arbitration Schemes

- **TDMA**
  - Time division multiple access
  - Assign slots to masters based on BW requirements
  - If a master does not have anything to read/write during its time slots, leads to low performance
  - Choice of time slot length and number critical
- **TDMA/RR**
  - Two-level scheme
  - If master does not need to utilize its time slot, second level RR scheme grants access to another waiting master
  - Better bus utilization
  - Higher implementation cost for scheme (more logic, area)

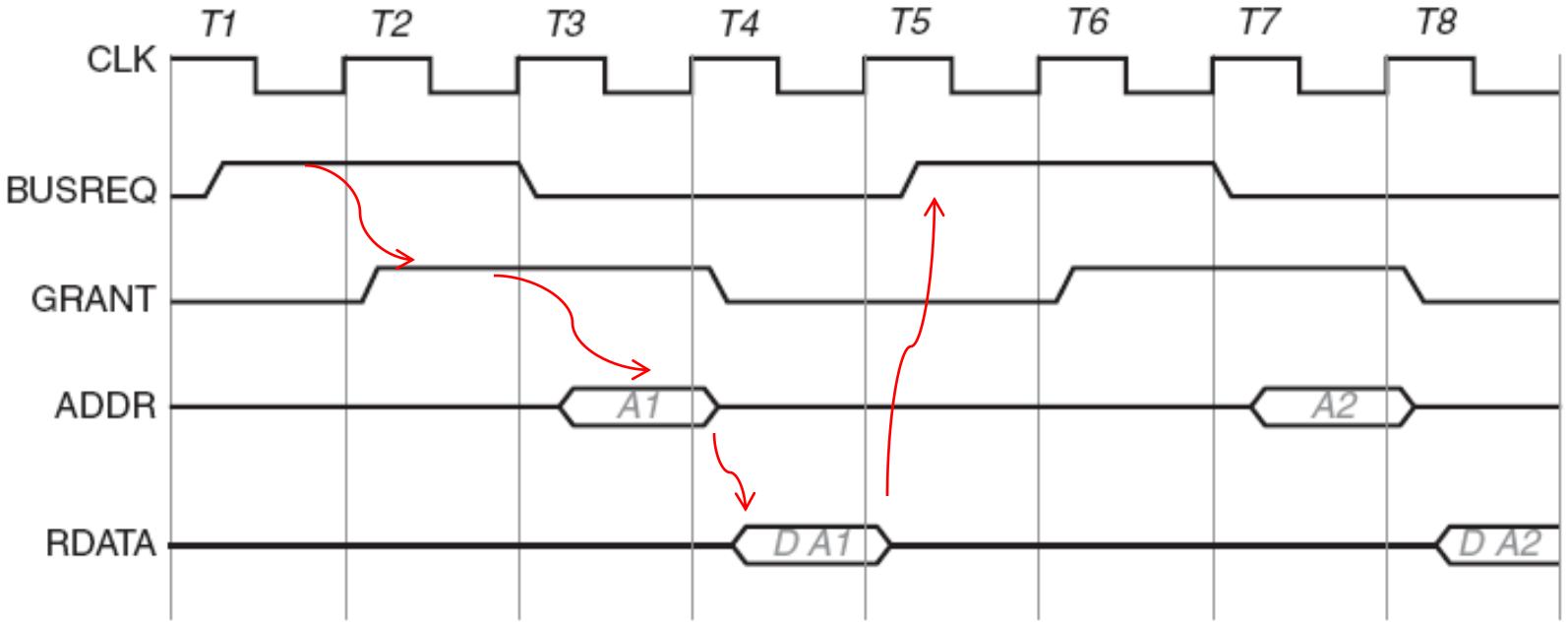
Arbitration Schemes

- **Dynamic priority**
  - Dynamically vary priority of master during application execution
  - Gives masters with higher injection rates a higher priority
  - Requires additional logic to analyze traffic at runtime
  - Adapts to changing data traffic profiles
  - High implementation cost
    (several registers to track priorities and traffic profiles)
- **Programmable priority**
  - Simpler variant of dynamic priority scheme
  - Programmable register in arbiter allows software to change priority

Bus Data Transfer Modes

Single Non-pipelined Transfer

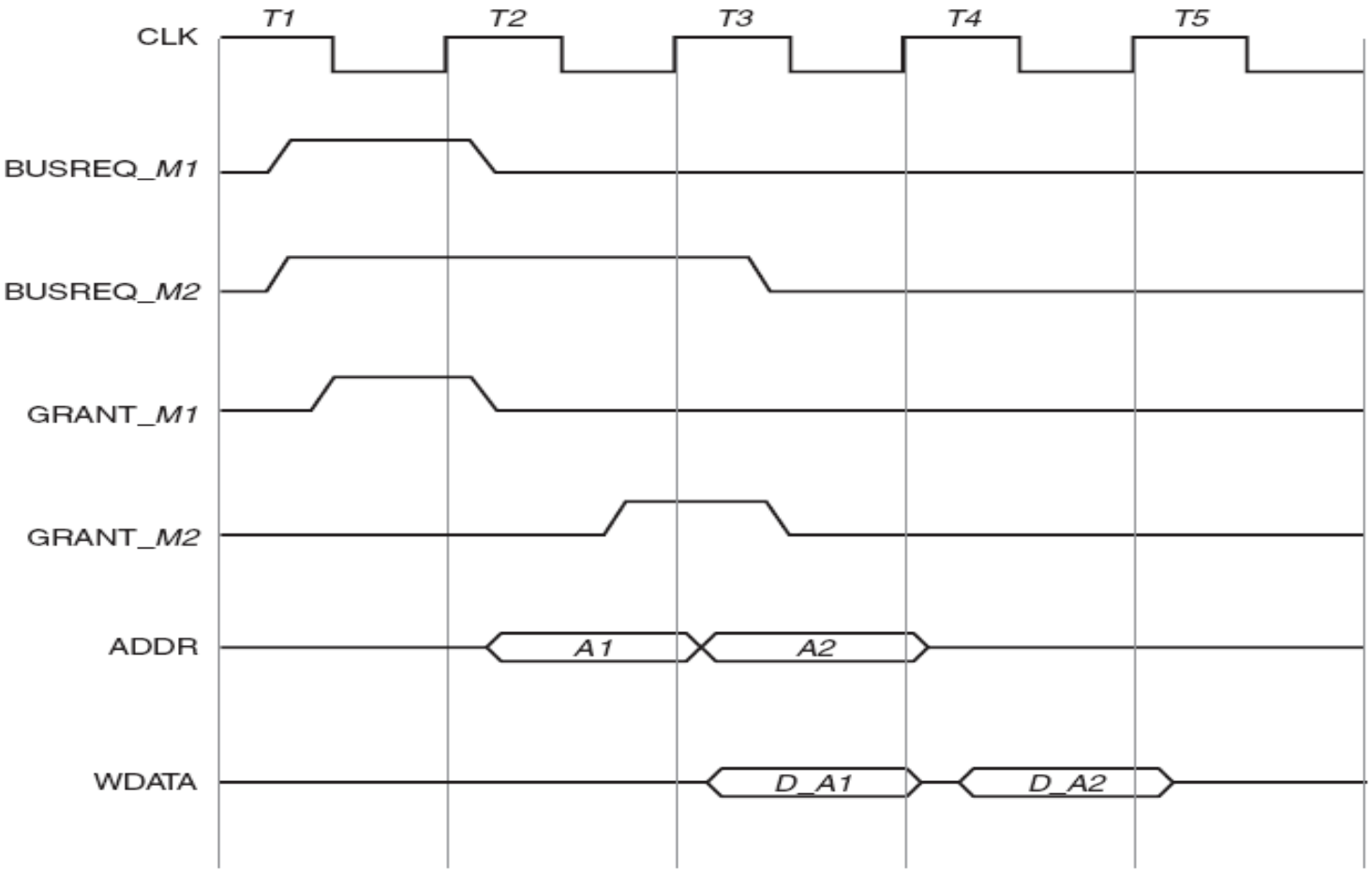
- The Simplest Transfer Mode
- first request for access to bus from arbiter
- on being granted access, set address and control signals
- Send/receive data **in subsequent cycles**



Bus Data Transfer Modes

Pipelined Transfer - Overlap address and data phases

Only works if separate address & data busses are present

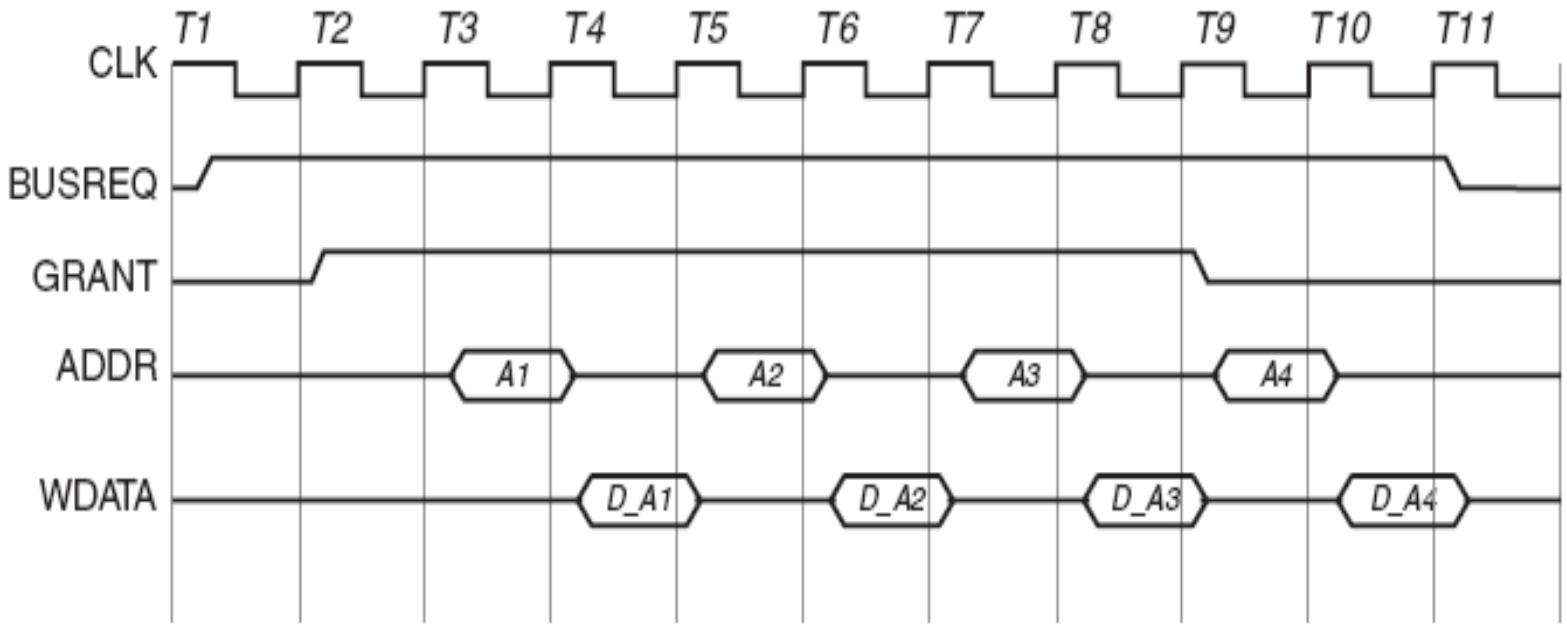


Bus Data Transfer Modes

Non-pipelined Burst Transfer

Send multiple data items, with only a single arbitration for entire transaction

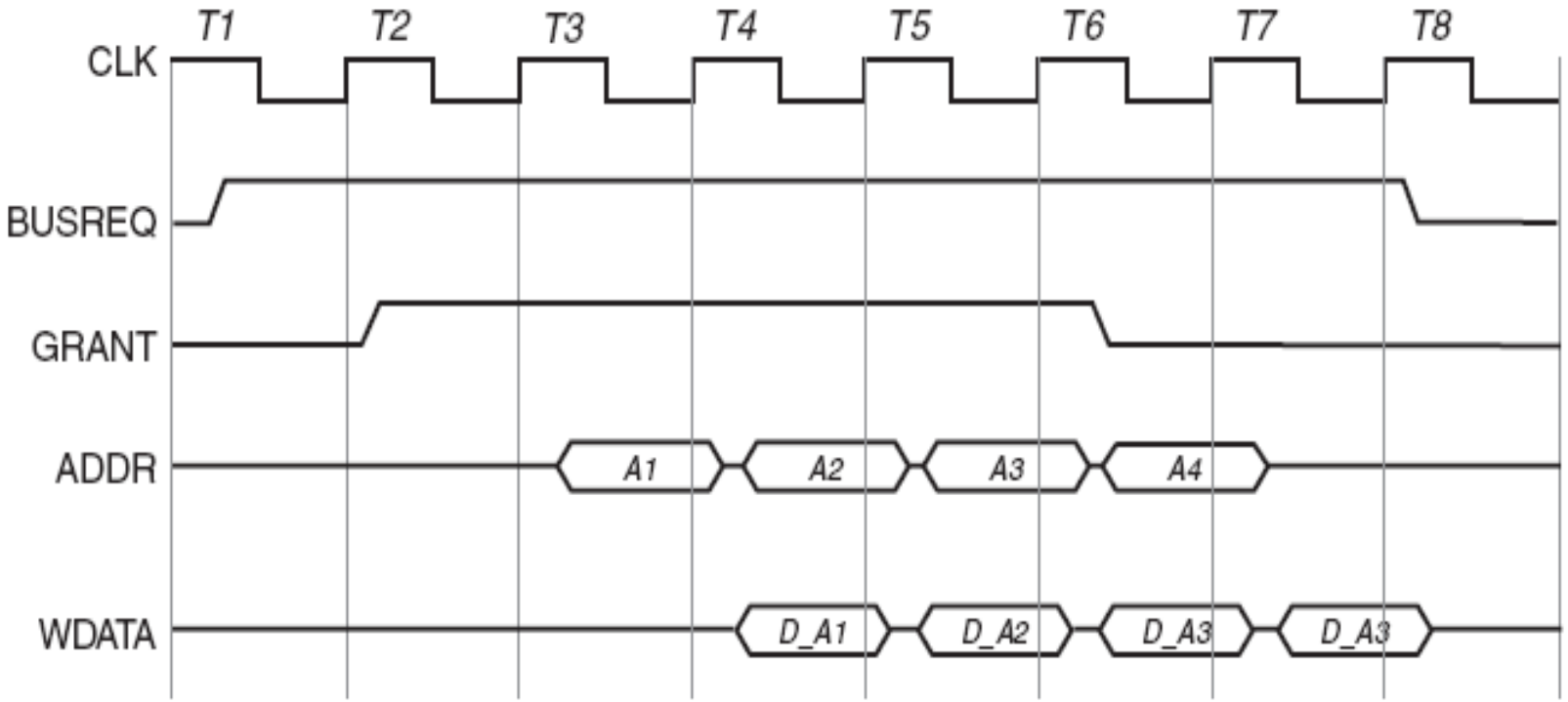
- master must indicate to the arbiter it intends to perform a burst transfer
- Saves time spent requesting for arbitration



Bus Data Transfer Modes

Pipelined Burst Transfer

- Useful when separate address and data buses available
- **Reduces data transfer latency**



Bus Data Transfer Modes

Split Transfer

- If slaves take a long time to read/write data, it can prevent other masters from using the bus
- Split transfers improve performance by 'splitting' a transaction
  - **Master sends read request to slave**
  - **Slave relinquishes control of bus as it prepares data**
    - Arbiter can grant bus access to another waiting master
    - Allows utilizing otherwise idle cycles on the bus
  - **When slave is ready, it requests bus access from arbiter**
  - **On being granted access, it sends data to master**
- Explicit support for split transfers required from slaves and arbiters **(additional signals, logic)**

Bus Data Transfer Modes

Out-of-Order Transfer

- **Allows multiple transfers from different masters, or same master, to be SPLIT by a slave and be in progress simultaneously on a single bus**
- **Masters can initiate data transfers without waiting for earlier data transfers to complete**
- **Allows better parallelism, performance in buses**
- **Additional signals are needed to transmit IDs for every data transfer in the system**
- **Master interfaces need to be extended to handle data transfer IDs and be able to reorder the received data**
- **Slave interfaces have out-of-order buffers for reads, writes, to keep track of pending transactions, plus logic for processing IDs**
  - ***Any application typically has a limited buffer size beyond which performance doesn't increase.***

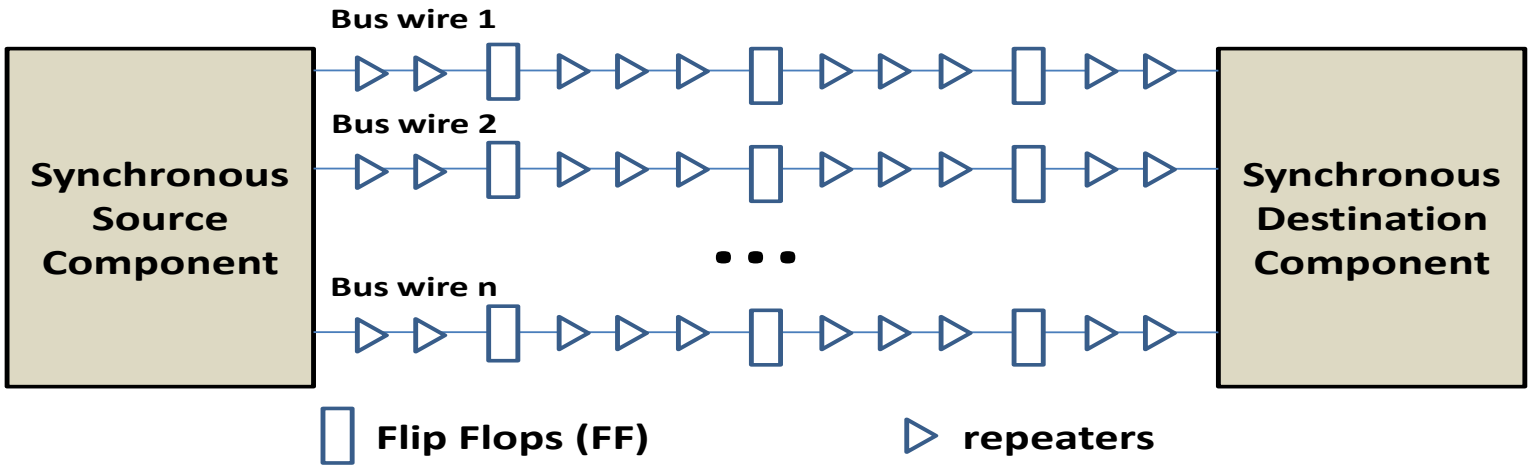
Physical implementation - Bus wires

- Bus wires are implemented as long metal lines on silicon transmitting data using electromagnetic waves
  (finite speed limit)
- As application performance requirements increase, clock frequencies are also increasing
  - Greater bus clock frequency = shorter bus clock period
    100 MHz = 10 ns ; 500 MHz = 2 ns
- Time allowed for a signal on a bus to travel from source-to-destination in a single bus clock=cycle is decreasing
- Can take multiple cycles to send a signal across a chip
  6-10 bus clock cycles @ 50 nm
  unpredictability in signal propagation time has serious consequences for performance and correct functioning of synchronous digital circuits **(such as busses)**

Physical implementation Issues - Bus wires

Partition long bus wires into shorter ones

- Hierarchical or split bus communication architectures
- Register slices or buffers to pipeline long bus wires enable signal to traverse a segment in one clock cycle



- Asynchronous buses: **No clock signal**
- Low level techniques: **add repeaters or using fat wires**

On-Chip Bus Interconnection

For highly connected multi-core system

- **Communication bottleneck**

For multi-master buses

- **Arbitration will become a complex problem**

Power grows for each communication event as more units attached will increase the capacitive load.

A crossbar switch can overcome some of these problems and limitations of the buses

- **Crossbar is not scalable**

Network-on-Chip *vs.* Bus Interconnection

- Total bandwidth grows
- Link speed unaffected
- Concurrent spatial reuse
- Pipelining is built-in
- Distributed arbitration
- Separate abstraction layers

However

- No performance guarantee
- Extra delay in routers
- Area and power overhead?
- Modules need NI
- Unfamiliar methodology

BUS inter-connection is fairly simple and familiar

However

- Bandwidth is limited, shared
- Speed goes down as # of Masters grows
- No concurrency
- Pipelining is tough
- Central arbitration
- No layers of abstraction (communication and computation are coupled)

Summary

- On-chip communication architectures are critical components in SoC designs
  - Power, performance, cost, reliability constraints
  - Rapidly increasing in complexity with the no. of cores
- Review of basic concepts of (widely used) bus-based communication architectures

Open Problems

- Designing communication architectures to satisfy diverse and complex application constraints