

Bus Interconnect & Input Output

CIT 595
Spring 2007

Computer System

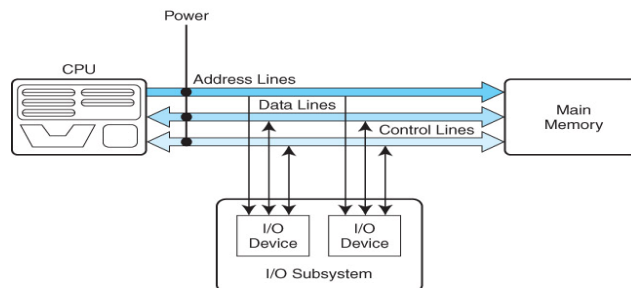
- Computer System consists of the following sub-systems:
 1. Process or CPU
 2. Memory
 3. I/O
 4. Bus
- Focus:
 - *Bus* allows CPU, Memory and I/O to communicate with each other
 - *I/O* allows medium through which the computer system communicates with the outside world

CIT 595

15 - 2

Bus Structure

- A bus is a set of electrical wires that conveys a single bit of information along each line
- Two types: *point-to-point*, and *multipoint* buses.



CIT 595

Multipoint

15 - 3

Bus Structure (lines)

Bus lines are usually classified into:

- *Address*: Identify the source or destination of data e.g. memory location generated by CPU
- *Data*: carry data information e.g. data from a location in memory
- *Control*: transmit command and timing information
 - Command e.g. Memory Write vs. I/O device Write
 - Timing enforces validity of address and data values
 - E.g. clock
- *Power*: provide electrical voltages
- Form the *bus protocol*

CIT 595

15 - 4

Bus Operation

- If one component wants to send/receive data to/from another component, it must:
 1. Obtain the bus i.e. *request* to use the bus
 - If devices go randomly then we have chaos with two components trying to send data on the bus
 2. Once access is *granted*, transfer the information on the bus
 - Address, Data, Control (command)

CIT 595

15 - 5

Problem with Multiple-Access Point Bus

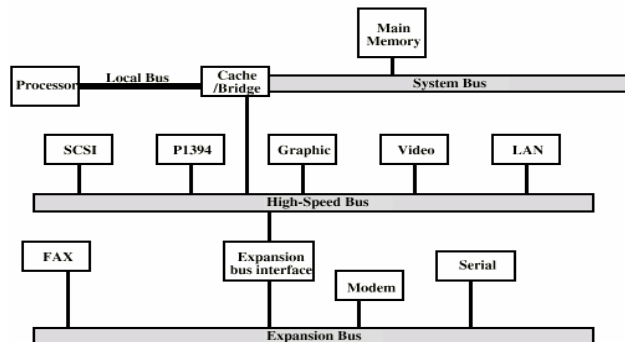
- Although multiple-access point bus is less expensive than point-to-point
- With too many sharing one bus, starts to become a bottleneck
- Increases propagation delay i.e. time it take for components/devices to co-ordinate transfer (bus operation)
- This is because bus is now congested and processing many requests

CIT 595

15 - 6

Multiple-Bus Hierarchy

- Place components with similar data rates
- Join the different levels with bridge



CIT 595

15 - 7

Bus Arbitration

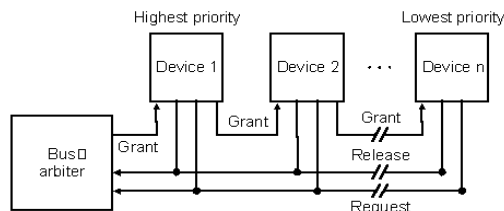
- Who grants requests?
 - Assign a *arbitrator*
- Arbitration goals
 - Prevent more than one bus user at a time
 - Give all devices a fair chance to access the bus
- Bus Arbitration Schemes
 - Single Arbitrators
 - Distributed Arbitrators

CIT 595

15 - 8

Single Arbitrator: Daisy Chain Scheme

- **Request Line** is a wired OR, hence arbitrator does not know who requested for bus access
- **Grant** bus control line by the arbitrator is passed down the bus from high to lowest priority I/O device
- I/O device gives up the bus after using it, who gets control next?
- Disadvantage: Can starve low-priority devices



CIT 595

15 - 9

Single Arbitrator: Centralized Scheme

- Receives request (via request line) from one or more components
 - Request lines are distinguished for the arbitrator
- Grants bus access (via grant line) to requester
 - If more than one request, then higher priority gets the bus
 - Does not starve lower priority devices unless a simultaneous request
- Disadvantage
 - Arbitrator is more complex now – needs to know priority of devices
 - Bus size increases due to multiple request lines

CIT 595

15 - 10

Single Arbitration in general

- Arbitrator is also known as master
- Arbitrator can be either:
 - **One** of the components that shares the bus
 - E.g. local bus between CPU and Cache, CPU is the arbitrator
 - Can be a **separate** piece of hardware
 - Common with buses solely for I/O or peripheral devices
- Does not scale well as number components sharing the bus increases
- Can be a single point of failure

CIT 595

15 - 11

Distributed Arbitrator: Decentralized w/ self-section

- No particular arbitrator
- Each device has a request line to access the bus
- Each device can read other request lines
- Each device determines who has the highest priority
- Expensive
 - Each device now needs a priority detection mechanism

CIT 595

15 - 12

Distributed Arbitrator: Decentralized w/ Collision Detection

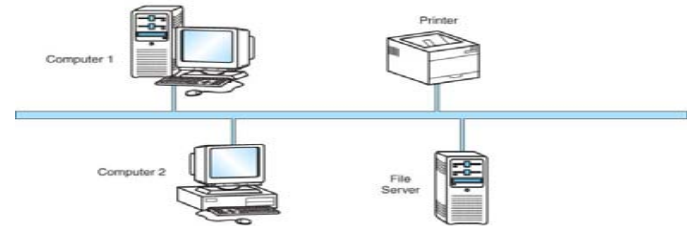
- Devices independently request the bus, then wait listen for a *collision* i.e. one or more devices are trying to access the bus at the same time
- In case of collision, some or all requesters have to ask for another request
- One Implementation of collision detection
 - Each device sends out its priority number on the bus request line serially
 - The bus request line is either ANDed (dominant bit 0) or ORed
 - If requestor does not see its number being displayed on the bus, that means it has to try again

CIT 595

15 - 13

Collision Detection in general

- Collision detection is feature of a bus protocol
- Found in buses that are used in networking
- E.g. Computer, printers, disks on local area network communicate via Ethernet networking protocol



CIT 595

15 - 14

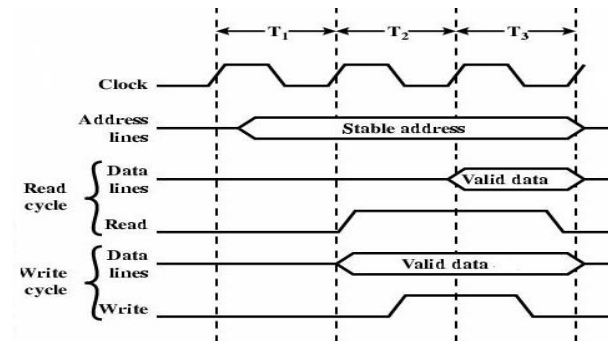
Synchronous Buses

- Events determined by *clock* signal
- Clock period should not be shorter than length of time it takes for the information to traverse the bus
- All devices can read clock line (part of the control lines)
- Synchronize on positive or negative edge
- E.g. processor and memory bus
 - CPU reads/writes every X cycles
 - E.g. 400 MHz SDRAM (Synchronous DRAM)

CIT 595

15 - 15

Synchronous Timing Diagram



E.g. CPU read/write from/to memory

Note: not all signals shown

CIT 595

15 - 16

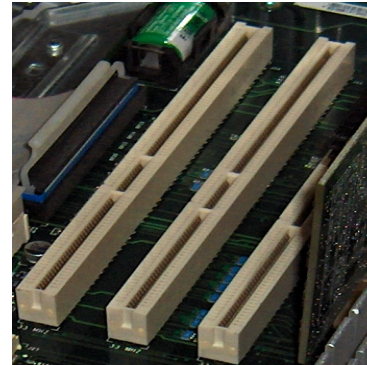
Peripheral Component Interface (PCI)

- Is a bus standard for attaching peripheral (external) devices to a computer motherboard
 - Motherboard – board that houses all electronics
- Processor-independent bus
- Used for high-speed I/O such Ethernet Interface card, Video, Cards, Sound Cards etc.
- Started out with 33.33 MHz clock with synchronous transfers
 - peak transfer rate of 133 MB per second for 32-bit bus width
(33.33 MHz × 32 bits × (1 byte ÷ 8 bits) = 133 MB/s)

CIT 595

15 - 17

PCI (contd..)



PCI Expansion Slot



Ethernet Card

CIT 595

15 - 18

Asynchronous Buses

- Used when data rate less predictable
- Example: waiting for user (keyboard) input
 - E.g. CPU must *synchronize* with keyboard device
 - So that it doesn't miss data or write too quickly
- Need some handshaking signals such as:
 - reqRead: indicate read for address put on the bus
 - Ack: acknowledgment signal to indicate request notified or request received
 - readyData: indicate that data is put on data lines

CIT 595

15 - 19

External or Peripheral or I/O Devices

- Link to the outside world
- Types:
 1. Human readable
 - Monitor, printer
 2. Machine readable
 - Disk, CD ROM, Scanner, Barcodes
 3. Communication
 - Modem
 - Network Interface Card (NIC)

CIT 595

15 - 20

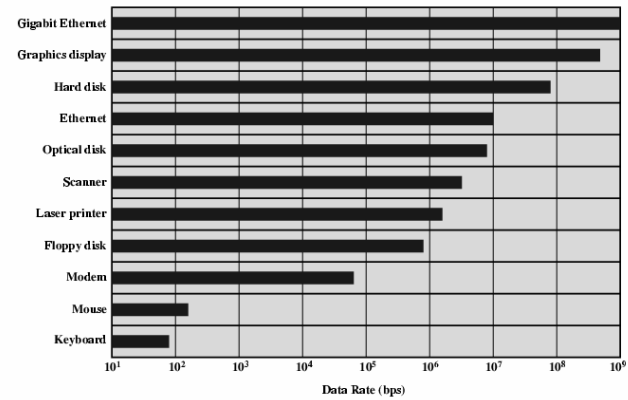
Communication between I/O and rest of the subsystem

- The peripherals (I/O devices) are not directly linked to the (I/O) bus because of *wide variety* of peripherals
 - Delivering different amounts of data
 - At different speeds i.e. data rate
 - In different formats
 - Also makes bus design complicated

CIT 595

15 - 21

Typical I/O Data Rates

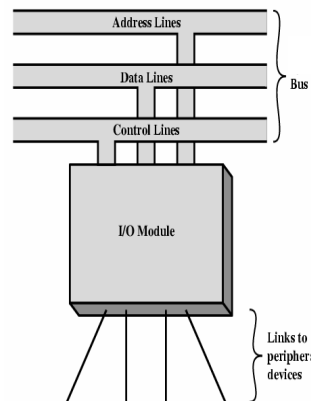


CIT 595

15 - 22

I/O Module

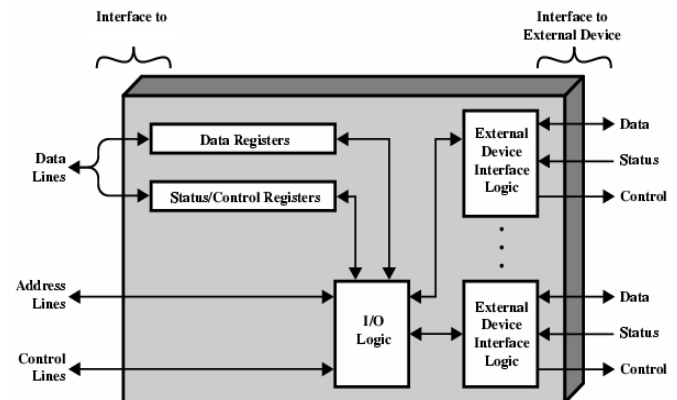
- An *I/O Module* interfaces to the
 - Bus and
 - Controls One or more peripheral devices
- It contains special purpose hardware(s)
- Performs data transfers between the device and the rest of computer system
- Also known as I/O controller



Generic I/O Module 15 - 23

CIT 595

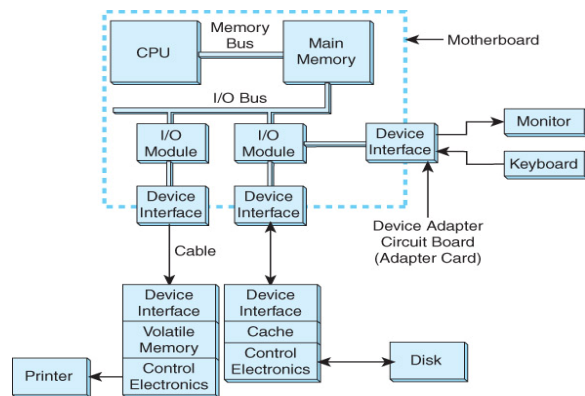
I/O Module Diagram



CIT 595

15 - 24

I/O Configuration with Overall Computer System



CIT 595

15 - 25

I/O Module Function

- Device Communication through the Device Interface
 - Control: determine whether device will send or receive data
 - Data: Actual Data transferred
 - Status: Device is ready, Data is available for transfer or error
- Data Buffering
 - Must handle varying data rates from memory and I/O device
 - Buffer data so that no one gets tied down
- Error Detection
 - Report to CPU Mechanical Failure e.g. paper jam
 - Report Data Transmission Error

CIT 595

15 - 26

I/O Module Function (contd..)

- Control and Signaling
 - Co-ordinates data transfer based on communication method with the processor
 - Communication with CPU
 1. Programmed a.k.a Polling
 2. Interrupt driven
 3. Direct Memory Access (DMA)

CIT 595

15 - 27

Programmed/Polling I/O

- CPU executes a program that communicates with I/O module
 - Read/write commands
 - Sensing status
 - Transferring data
- CPU waits for I/O module to complete operation
 - Disadvantage is that it wastes CPU time

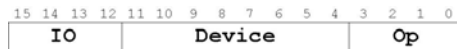
CIT 595

15 - 28

Addressing I/O Devices

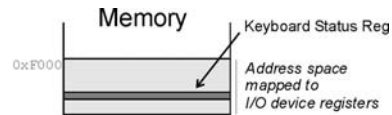
Instructions

- Designate opcode(s) for I/O
- Register and operation encoded in instruction



Memory-mapped

- Assign a memory address to each device register
- Use data movement instructions (LD/ST) for control and data transfer
- No actual memory access performed



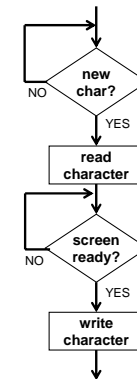
CIT 595

15 - 29

Polling Keyboard Echo Routine

```

POLL1  LDI R0, KBSRPtr
        BRzp POLL1
        LDI R0, KBDRPtr
POLL2  LDI R1, DSRPtr
        BRzp POLL2
        STI R0, DDRPtr
        ...
KBSRPtr .FILL xFE00
KBDRPtr .FILL xFE02
DSRPtr  .FILL xFE04
DDRPtr  .FILL xFE06
    
```



CIT 595

15 - 30

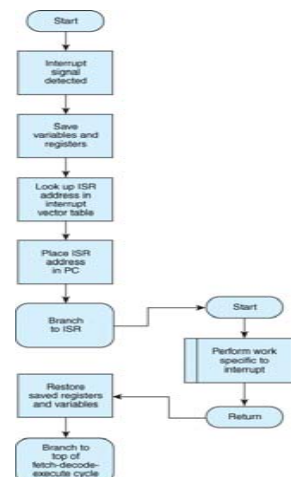
Interrupt Driven I/O

- Overcomes CPU waiting
- No repeated CPU checking of device
- I/O module interrupts when ready
- Basic Operations
 - CPU issues read command
 - I/O module gets data from peripheral whilst CPU does other work
 - I/O module interrupts CPU
 - CPU requests data
 - I/O module transfers data

CIT 595

15 - 31

Single Interrupt Handling



CIT 595

15 - 32

Dealing with Multiple Interrupts (Nested)

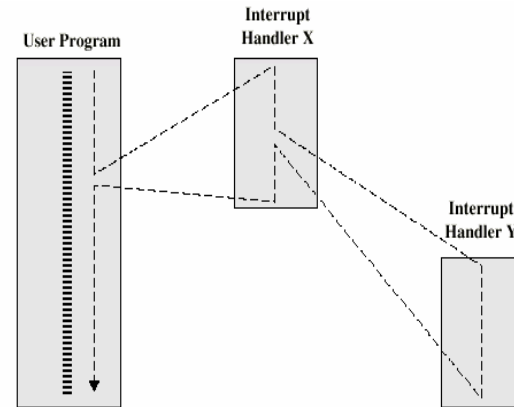
Dealing with interrupts while servicing another:

- 1. Disable Interrupts while servicing another
 - Not good if the interrupt is of critical nature
- 2. Assign priorities to devices
 - If current interrupt has low priority, suspend and service new interrupt
 - Remember to save the state of the previous interrupt – goes on the operating system stack

CIT 595

15 - 33

Multiple Interrupts Handling (Nested)



CIT 595

15 - 34

Simultaneous Interrupts

- Need to determine higher priority among devices who interrupted at the same time
- Use a special hardware called *interrupt controller*

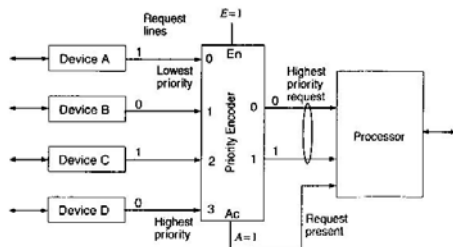


Figure 9.17 Resolving interrupt requests using a priority encoder.

CIT 595

15 - 35

Simultaneous Interrupts (contd..)

- Priority Encoder – opposite of decoder
- Example 4:2 Priority Encoder
 - x3 has highest priority, x2 the next highest...x0 has lowest priority
 - y1y0 are outputs determine the which request goes to processor for servicing

x3	x2	x1	x0	y1	y0
1	x	x	x	1	1
0	1	x	x	1	0
0	0	1	x	0	1
0	0	0	1	0	0

CIT 595

15 - 36

Direct Memory Access (DMA)

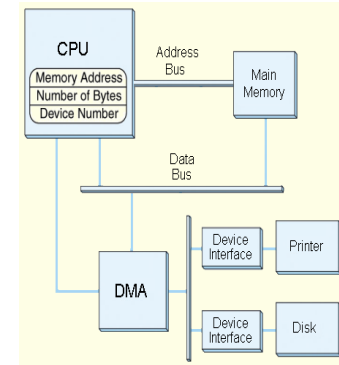
- Interrupt driven and programmed I/O require active CPU intervention
 - CPU is tied up either polling or ISR routine
- Additional Module (hardware) on bus
- DMA controller takes over from CPU for I/O

CIT 595

15 - 37

DMA (contd..)

- I/O devices that usually transfer block of memory, use the DMA technique
 - a.k.a *block I/O*
- E.g. disk drive



CIT 595

15 - 38

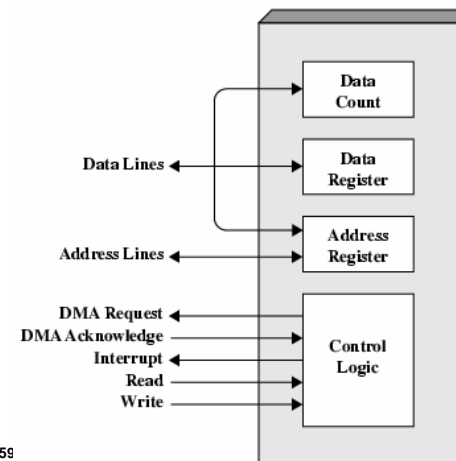
DMA Operation

- CPU/Processor tells DMA controller
 - Read/Write
 - Device address
 - Starting address of memory block for data
 - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished indicating transfer done

CIT 595

15 - 39

DMA Module Diagram



CIT 59

15 - 40

DMA Transfer

- DMA controller and CPU share the bus which allows access to Memory Modules
- DMA controller takes over bus for a cycle when transferring data to memory
 - Coined as *cycle stealing*
- Not an interrupt
 - CPU does not switch context
- CPU suspended just before it accesses bus for memory transaction
 - i.e. before an operand or data fetch or a data write
- Slows down execution of current program but not as much as CPU doing the transfer

CIT 595

15 - 41

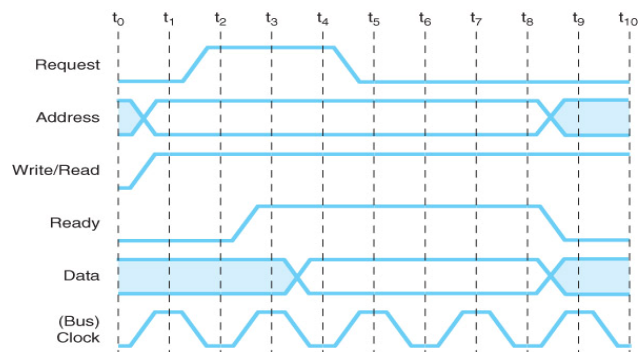
DMA Cycle Stealing Solution

- Perform DMA transfers independently from the system bus by having separate bus for transferring data
- Require memory to be dual-ported i.e. two access paths as CPU and DMA now both send and request data
 - Writes to same location are avoided by having separating memory use for user-space work and I/O data

CIT 595

15 - 42

Bus Timing Diagram: DMA Module and Disk Drive



CIT 595

15 - 43

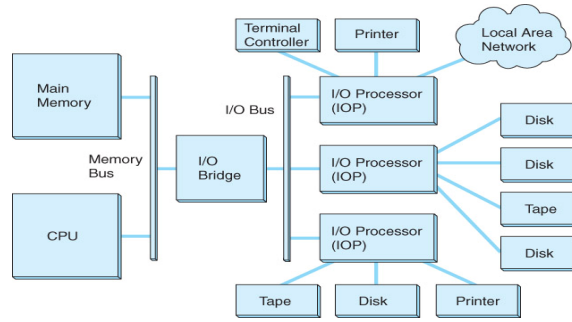
I/O Channels

- All techniques so far work well for single-user system
- In multi-user system more sophisticated method called I/O channels is used
 - E.g. File-server or storage systems
- I/O channel contain dedicated *I/O Processors (IOP)*
 - More than a controller
- One or more I/O processors control various I/O pathways called *channel paths*
- Each channel controls I/O devices that have similar data rate
 - Advantage: slow and fast devices on separate pathways
 - Each Channel multiplexes between devices that it control

CIT 595

15 - 44

I/O Channel Configuration



- IOP use system bus only to get instructions from host CPU
- Data transfer is done using local I/O buses

CIT 595

15 - 45