UNIT-V. - Processing and Operating Systems

Introduction:

⇒ Microprocessor execute simple user application program. But some application requires large number of line of code. writing and executing large program on microprocessor is complex task.

⇒ To study the writing complex program for microprocessor, we must understand concept of process and OS.

⇒ The process difines the state of an executing program, while the OS provides the mechanism for switching execution between the processes.

⇒ These two mechanisms together let us build applications with more complex functionality and much greater flexibility to saying satisfy timing requirements.

Multiple Tasks and Multiple Processes:

⇒ Task are units of sequential code implementing the system actions and executed concurrently by an OS.

Scanned with ACE Scanner

⇒ Real time systems requires that tasks be performed within a particular time fuction. Task is related to the performance of the real time systems.

⇒ A task, also called a thread, is a simple program that thinks it has the CPU all to itself. The design process for a real-time application involves splitting the work to be done tasks responsible for a real-time application i.e. a portion of the problem.

⇒ Each task is assigned a priority, its own set of CPU registers and its own stack area.

⇒ In the specified time constraint, system must produce its correct output. If system fail to meet the specified output, then the system is fail or quality decreases.

⇒ Real time systems are used for space flights, air traffic control, high speed aircraft, telephone switching electricity distribution, industrail process etc.

⇒ Real time system must be 100% responsive 100% of the time. Response time is measured in fab fractions of second, but this is an ideal not often achieved in the field.

Scanned with ACE Scanner

⇒ Real time database is updated continuously. In aircraft example, flight data is continuously changing so it is necessary to update. It include speed, direction, location, height etc.

⇒ A process is a sequential program in exection Terms like job and task are also used to denote a process.

⇒ A process is a dynamic entity that executes a program on a particular set of data. Multiple processes may be associated with one program.

⇒ Task is a single instance of an execlable program.

⇒ In a multiprogramming environment, usually more programs to be executed than could possibly be run at one time. In CPU scheduling it switches from one process to another process. CPU resource management is commonly known as scheduling.

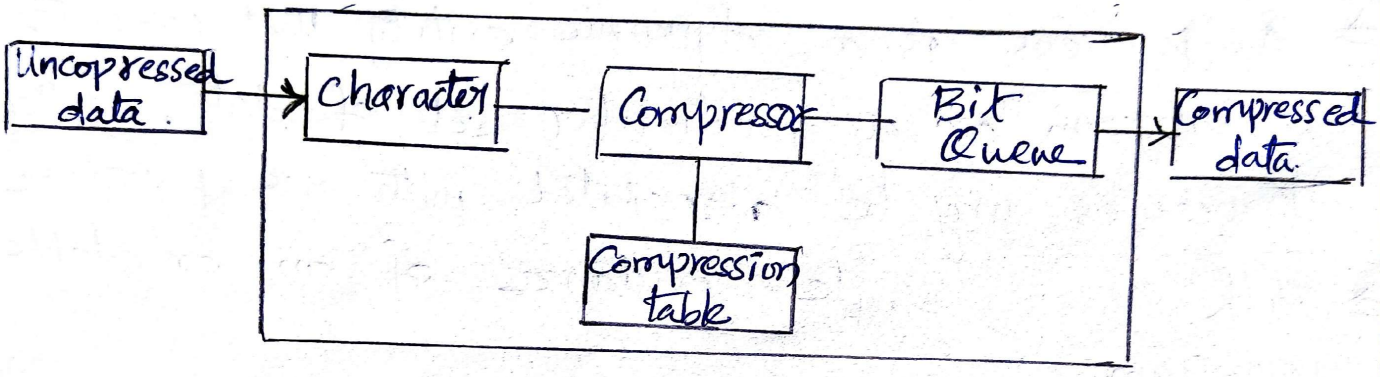
⇒ Objective of the multiprogramming is to increases the CPU utilization. CPU scheduling is one kind of fundemental operating system functions.

⇒ Each process has an execution state which indicate what process is currently doing. The process descriptor is the basic data structure used to represent the specific state for each process.

Scanned with ACE Scanner

⇒ Input and output of the compressor box is serial ports. It takes uncompressed data and processes it. Output of the box is compressed data. Given data is compressed using predefined compression table. Modem is used such type of box.

⇒



⇒ The program's need to receive and send data at different rates. It is an example of rate control problems. It uses asynchronous input. You can provide a button for compressed mode and uncompressed mode.

⇒ when user press uncompressed mode, the input data is passed through unchanged.

⇒ Sampling the button's state too slowly can cause the machine to miss a button depression entirely, but samplying it too frequently and duplicating a data value can cause the machine to incorrectly compress data.

⇒ This problem is solved by maintaining counter.

Scanned with ACE Scanner

Multirate Systems:

⇒ More complicated control systems have multiple sensors and actuators and must support control loops of different rates. Multirate embedded computing systems includes automobile engines, printers and cell phones.

⇒ Tasks may be synchronous or asynchronous. Synchronous tasks may recur at different rates. Processors run at different rates based on computational needs of the tasks.

⇒ Automotive engine control is an example of multirate system. Tasks in automotive engine control are spark control, crankshaft sensing fuel/air mixture, oxygen sensors and Kalman filter.

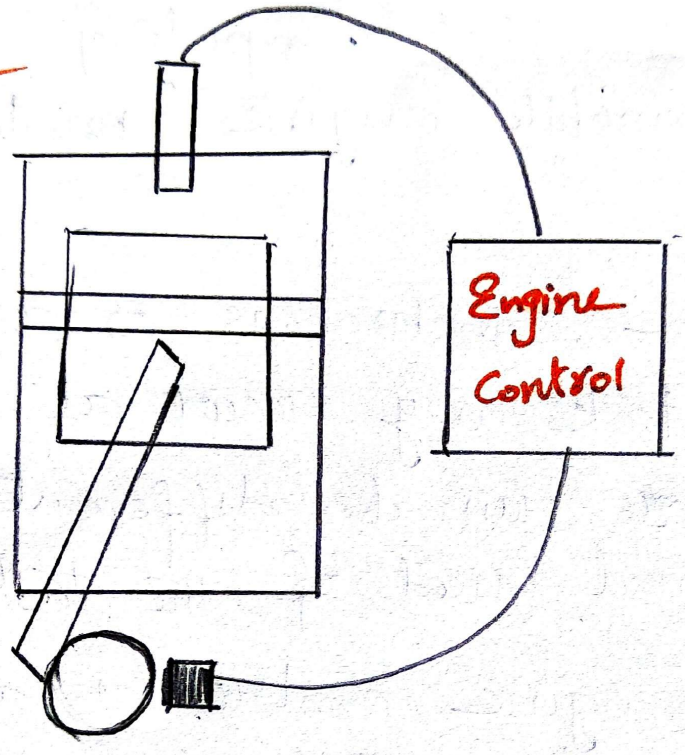
⇒ The figure shows automotive engine control.

⇒ The spark plug must be fired at a certain point in the combustion cycle, but to obtain better performance, the phase relationship between the piston's movement and the spark should change as a function of engine speed.

Scanned with ACE Scanner

⇒ Using a microcontroller that senses the engine crankshaft position allows the spark timing to vary with engine speed.

Engine Control



* Automobile engine controllers use additional sensors, including the gas pedal position and an oxygen sensor used to control emissions. They also use a multimode control scheme.

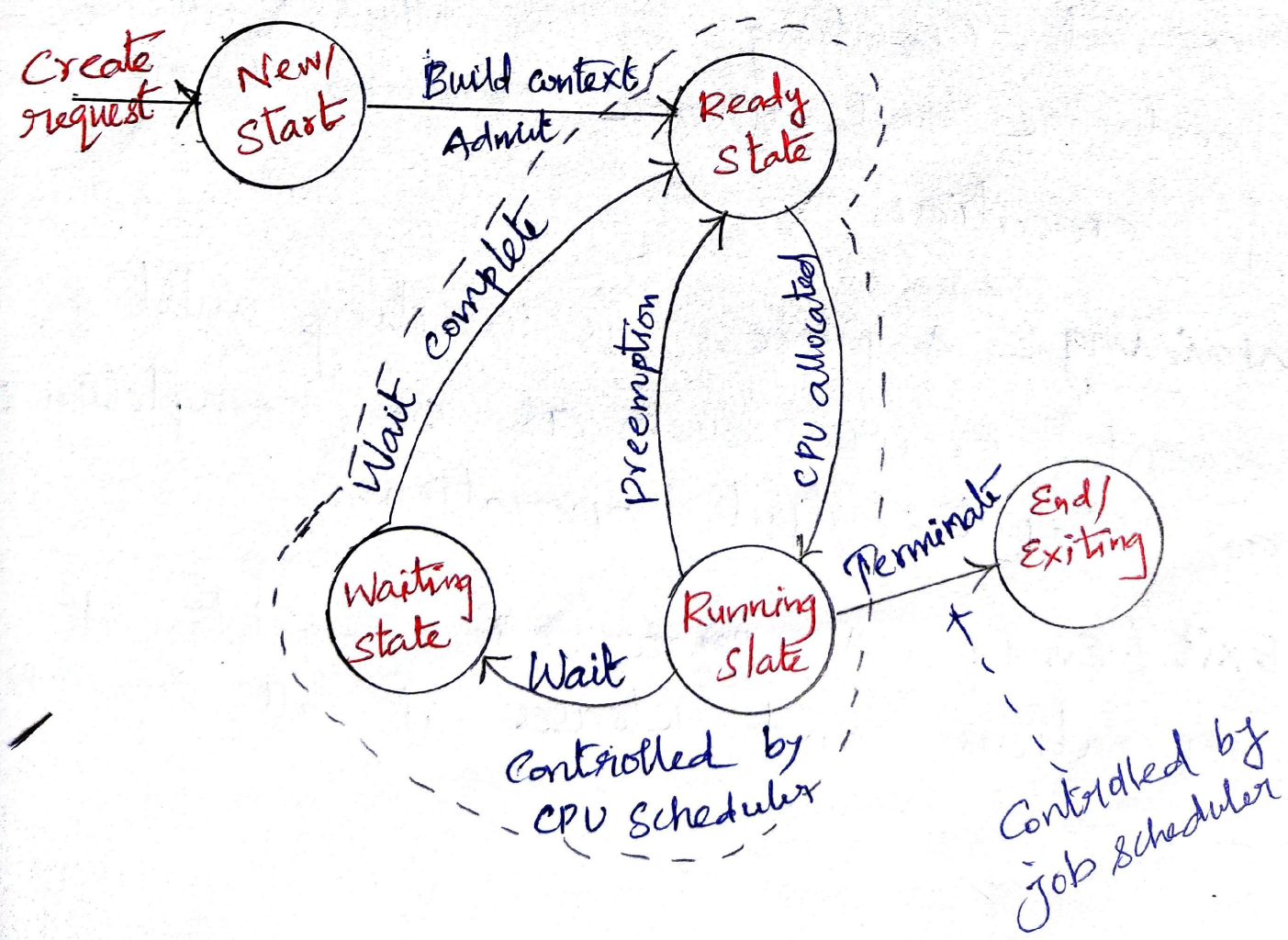
⇒ The larger number of sensors and modes increase the number of discrete tasks that must be performed.

Scanned with ACE Scanner

Process State and Scheduling:

⇒ Each process has an execution state which indicates what process is currently doing. The process descriptor is the basic data structure used to represent the specific state for each process.

⇒ A state diagram is composed of a set of state and transition between states.



Process State Diagram

⇨ State diagram is used by process manager to determine the type of service to provide to the process. The process states are as follows: " New, ready, **running**, **waiting** and **end**.

New: operating system creates new process by using fork() system call. These process are newly created process and resources are not allocated.

Ready: The process is competing for the CPU. Process reaches to the head of the list (queue)

Running: The process that a currently being executed. Operating system allocates all the hardware and software resources to the process for execution.

Waiting: A process is waiting until some event occurs such as the completion of an input-output operation.

Exit / End: A process is completes its operations and releases it all resources.

Scanned with ACE Scanner

⇒ Five types of inter-process communication as follows:

1. shared memory permits processes to communicate by simply reading and writing to a specified memory location.
2. Mapped memory is similar to shared memory except that it is associated with a file in the file system.
3. Pipes permit sequential communication from one process to a related process.
4. FIFOs are similar to pipe, except that unrelated processes can communicate because the pipe is given a name in the file system
5. Sockets supports communication between unrelated processes even on different computers.

Purposes of IPC:

1. Data Transfer: one process may wish to send data to another process.
2. Sharing Data: Multiple processes may wish to operate on shared data, such that if a process modifies the data, that change will be immediately visible to other processes sharing it.

Scanned with ACE Scanner

INTERPROCESS COMMUNICATION MACHANISM:

⇒ Exchange of data between two or more separate, independent processes / threads is possible using IPC. Operating Systems provides facilities / resources for Inter-Process Communication (IPC), such as **message queues, semaphores, and shared memory.**

⇒ A complex programming environment often uses multiple cooperating ~~progress~~ processes to perform related operations. These processes must communicate with each other and share resources and information. The Kernel must provide mechanisms that make this possible. These mechanisms are collectively referred to as **interprocess communication.**

⇒ Distributed computing systems make use of these facilities / resources to provide Application Programming Interface (API) which allows IPC to be programmed at a higher level of abstraction (e.g. send & receive)

Scanned with ACE Scanner

Five types of inter-process communication are as follows:

1. Shared memory permits processes to communi--cate by simply reading and writing to a specified memory location.

2. Mapped memory is similar to shared memory, except that it is associated with a file in the file system.

3. Pipes permit sequential communication from one process to a related process.

4. FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the file system.

5. Sockets support communication between unrelated processes even on different computers.

Purposes of IPC:

1. Data transfer: One process may wish to send data to another process.

Scanned with ACE Scanner

2. Sharing data: Multiple processes may wish to operate on shared data, such that to ~~operate~~ if a process modifies that data, that change will be immediately visible to other processes sharing it.

3. Event modification: A process may wish to notify another process or set of processes that some event has occurred.

4. Resource sharing: The Kernel provides default semantics for resource allocation; they are not suitable for all application.

5. Process control: A process such as a debugger may wish to assume complete control over the execution of another process.

IPC has two forms: IPC on same host
IPC on different hosts.

IPC is used for 2 functions:

1. Synchronization
2. Message passing.

Scanned with ACE Scanner

Features of Message Passing:

1. **Simplicity**: It should be possible to communicate with old and new applications.

2. **Uniform semantics**: Message passing is used for two types of IPC.

   a. **local communication**: Communicating processes are on the same node.

   b. **Remote communication**: Communicating processes are on the different nodes.

3. **Efficiency**: IPC become so expensive if message passing system is not effective

4. **Reliability**: Distributed systems are prone to different catastrophic event such as node crashes or physical link failure. loss of message because of communication link fails. To handle the loss message, we required acknowledgement and retransmission policy.

5. **Correctness**: It is a feature to IPC provides for group communication. Issues related to correctness to a group of receivers will be delivered to either all of them or none of them

Scanned with ACE Scanner

(i) Automicity : Every message sent to a group of receivers will be delivered to either all of them or none of them.

(ii) Ordered delivery : Message arrive to all receivers in an order acceptable to the application.

(iii) Survivability : Message will be correctly delivered despite partial failures of processes, machine or communication links.

6) Security :

Message passing system must provide a secure end to end communication.

7) Portability : Message passing system should itself be portable.

IPC Message passing :

⇒ Message passing system requires the synchronization and communication between the two processes

⇒ The actual function of message passing provided in the form of a pair of primitives.

(a) send (destination_name, message)

(b) Receive (source_name, message)

Scanned with ACE Scanner

Design characteristics of message system for IPC

1) Synchronization between the process.
2). Addressing
3) Format of the message
4) Queueing discipline.

Message structure

Actual data or pointer to the data	Structural information		Sequence number or message ID	Addresses	
	Number of bytes/ elements	Type		Receiving Process address	Sending Process address

← Variable → ← Fixed-length header. →

Size collection of typed data

⇒ The header block of a message may have the following elements:

1. Address: A set of character that uniquely identify both the sender and receiver.
2. Sequence number: It is the message identifier to identify duplicate and lost message in case of system failure.
3. Structural information: It has two part.
   1) The type part that specifies whether the data to be sent the receiver is included within the message
   2) ~~length of the Variable-Size~~ message.

Scanned with ACE Scanner

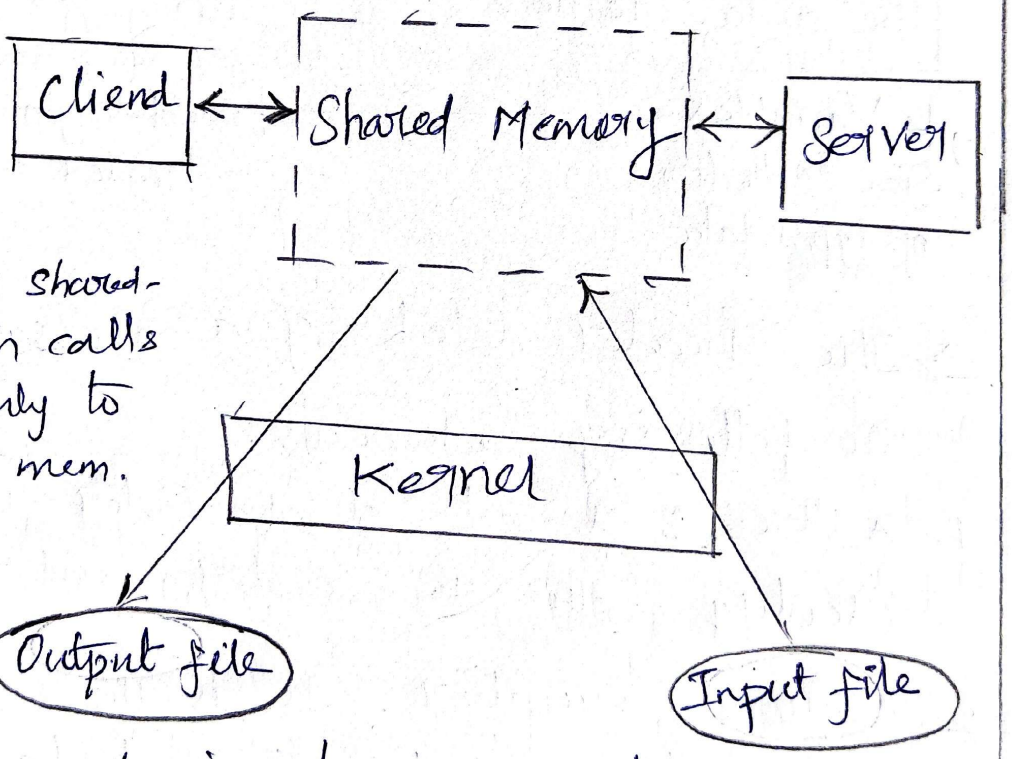
Shared Memory:

⇒ A region of memory that is shared by co-operating processes is established. Processes can then exchanged information by reading and writing data to the shared region.

⇒ Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer.

⇒ Shared memory is faster than message passing

⇒ In contrast, in shared-memory, the system calls are required only to establish shared mem. regions.



Advantages:

1. Good for sharing large amount of data.
2. Very fast.

Limitations:

⇒ No synchronization provided - application must creat their own.

Scanned with ACE Scanner

Evaluating Operating System Performance.

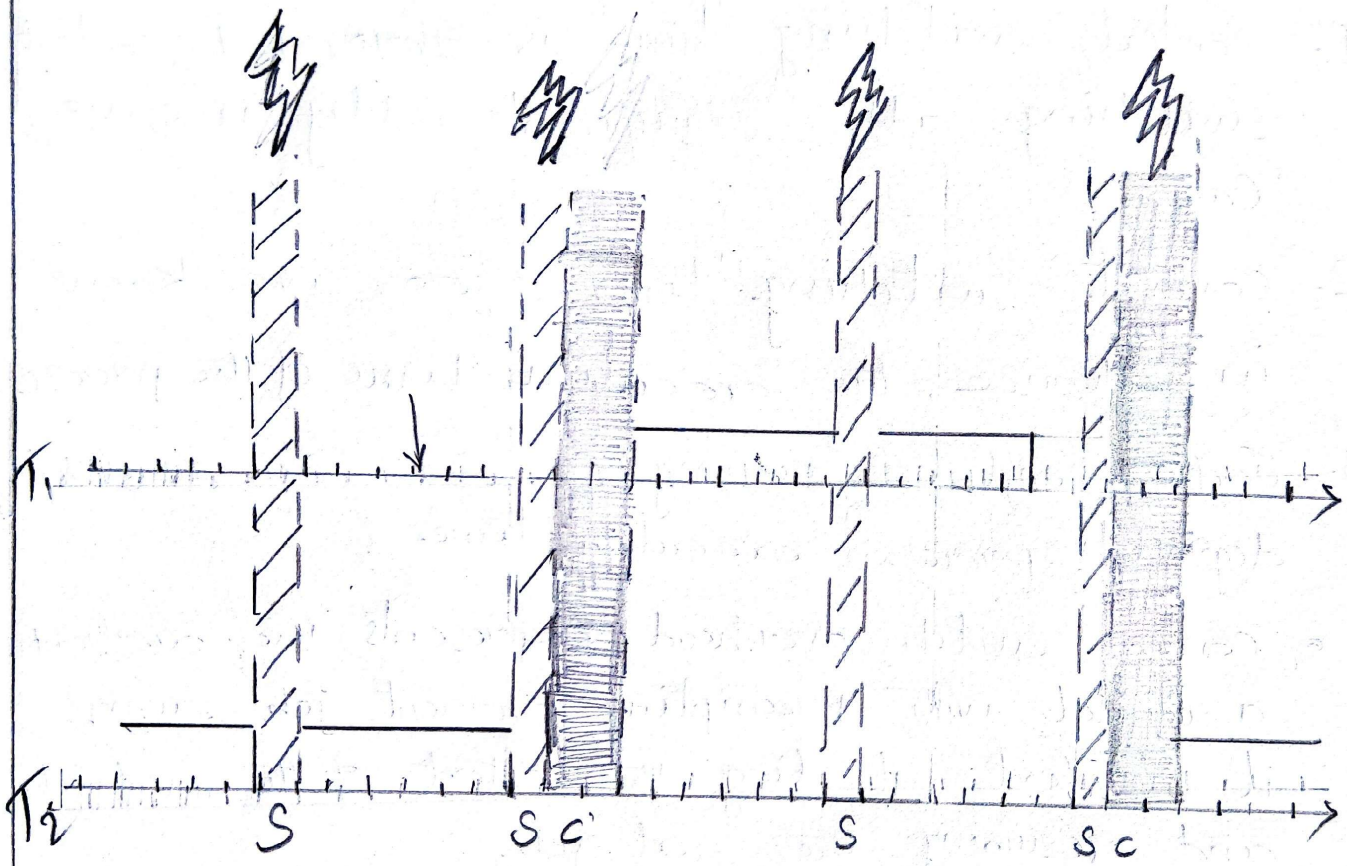
⇨ Scheduling algorithm is not used for calculating performance of the real system running processes. Assumption about analysis of scheduling policy are as follows:

1. Context switching time is zero, But context-switching adds significant delay in some cases.
2. context switching time is zero, we know in advance, the execution time of the processes.
3. Cache conflicts among processes can drastically degrade process execution time.

- context switch overhead represents the overhead associated with preempting current job, saving its context, loading the cotext of the next job and resuming the next job.
- The context switch into task (T_i) is performed at the kernel priority level, but only when task (T_i) is ready to execute.
- Figure next shows the scheduling the context switch overheads in a time-triggered preemptive system.

Scanned with ACE Scanner

- Consider two tasks T_1 with $C_1 = 10$ and T_2 with $C_2 > 17$. clock interrupt handler is invoked with period $T_0 = 8$ and computation time $C_0 = 1$.
- The execution of the Task T_2 is regularly preempted by the clock handler. Somewhere before the second clock tick T_1 is released



Clock tick ↓ Event release Scheduling overhead Context Switch overhead

Scheduling (S) and context switch (C) overheads in a time-triggered preemptive system.

⇒ After executing for 10 time units T_1 is finished and released the processor until the next clock tick, During the next scheduler invocation the process is switched back to T_2.

Scanned with ACE Scanner

Power optimization strategies for Processes.

⇨ RTOS and system architecture can use static and dynamic power management mechanisms to help manage the system's power consumption.

⇨ Power management policy in general examines the state of the system to determine when to take actions.

⇨ Power management policy is a strategy for determining when to perform certain power mangement operations.

⇨ **Dynamic Power management (DPM)** is a design methodology for dynamically reconfiguring system to provide the requesting services & performance levels with a minimum number of active components or a minimum load on such components.

⇨ Going into a low-power mode takes time; generally, the more that is shut off, the longer the delay incurred during restart. Because power-down and power-up are not free, modes should be changed carefully.

Scanned with ACE Scanner

⇨ In most real-time operating systems, a context switch requires only a few hundred instruction, which with only slightly more overhead for a simple real-time scheduler like RMS. When the overhead time is very small relative to the task periods, then the zero-time context switch assumption is often a reasonable approximation.

⇨ Process execution time is not contant. Extra CPU time can be good. Extra CPU time can also be bad because next process runs earlier, causing new preemption.

⇨ Processes can cause additional caching problems. Even if individual processes are well-behaved, processes may interfere with each other.

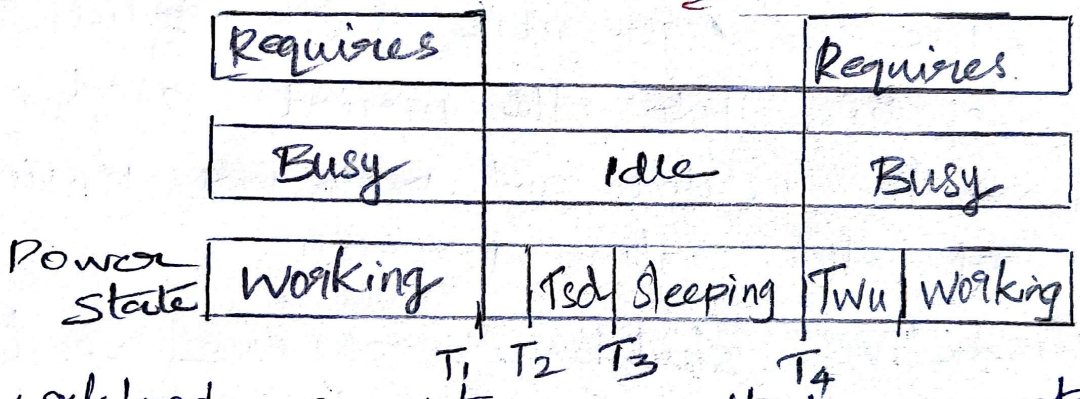
⇨ Worst-execution time with bad behavior is usually much more worse than execution time with good cache behavior.

Scanned with ACE Scanner

⇨ Determining when to switch into and out of a power-up mode requires an analysis of the overall system activity.

⇨ System-level power management saves power of subsystems. Examples of devices include I/O controllers, hard disk drives, network interface cards and displays. Shutting down hard disks and displays is the most widely adopted system-level power management on PCs.

Power management concept



⇨ A workload consists of multiple requests. For hard disks, requests are read or write commands; for network cards, requests are packets to send or to be received.

⇨ When there are requests, the device is busy; otherwise, it is idle. Here, the device is idle between T1 & T4. When the device is idle, it can be shut down to enter a low-power sleeping state.

⇨ The device is shutdown at T2 and woken up at T4 when requests arrive again. Changing power states takes time: Tsd and Twu are the shutdown and wake-up delays.

Scanned with ACE Scanner

⇨ In the example of hard disks and displays, it takes several seconds to wake-up these devices. Furthermore, waking up a sleeping device may take extra energy.

Predictive shutdown: shutdown as soon as a new idle period starts based on history. Avoid wasting energy before reaching timeout threshold.

Predictive wakeup: wake up when predicted idle time expires, even if no new activity has occurred.

- Use a regression ~~of~~ of predict the length of this ~~hop~~ idle period based on preceding active period and ~~posi~~ ~~pre~~ previous n pairs of idle / active periods.

Predictive shutdown: Observe short active period tends to be followed by long idle period in some applications. If the preceding active period is less than as threshold, the idle period is predicted to be longer than break-even time.

Advanced Configuration and Power Interface

⇨ Advanced Configuration and Power Interface (ACPI) is an open industry standard for power management services. It is designed to be compatible with a wide variety of OSs.

Scanned with ACE Scanner

⇨ Power management states and required functionality are defined for multiple levels of the system.

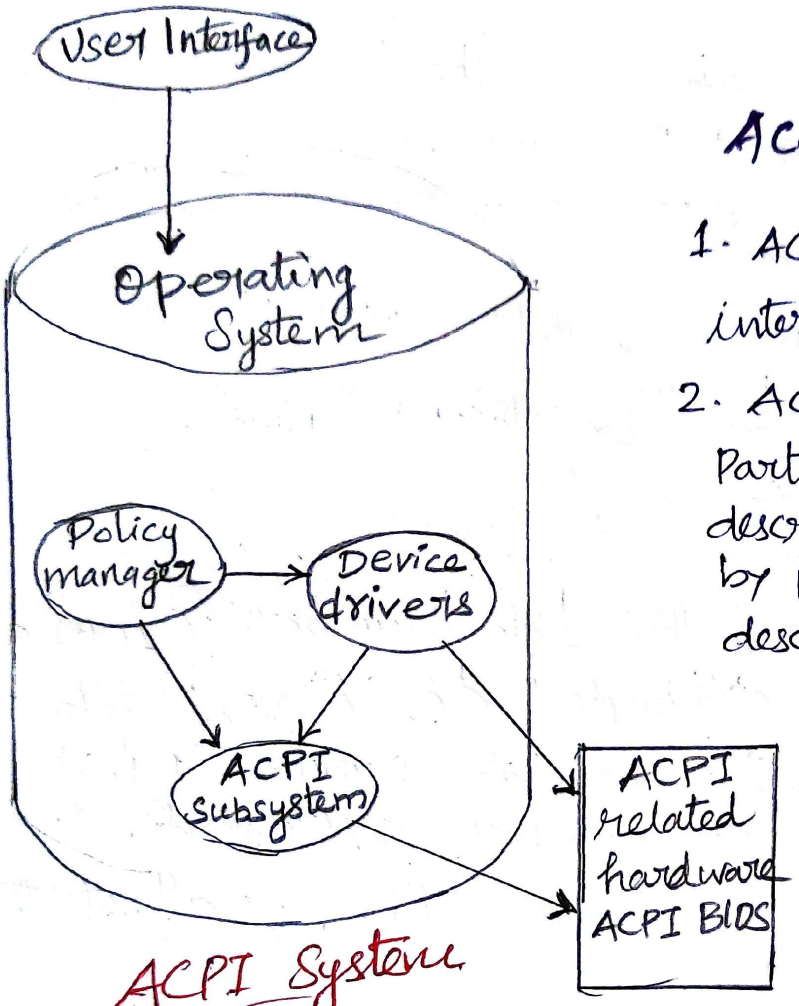
1. Global View : Gx states
2. System : Sx states
3. Processor : Cx states
4. PCI / PCI-X bus : Bx states
5. PCI / express links : Lx states
6. Devices : Dx states

- General trend of the state numbers : (Zero) 0 is the active state (example : G0, S0, D0). System is available to user. The state number 1 to n are sleep states. Higher number corresponds to lower power. User perception is OFF for all of these sleep state.

- ACPI functions are as follows :
  1. System power management
  2. Device power management
  3. Processor power management
  4. Plug and play
  5. System events
  6. Battery managements.
  7. Thermal management
  8. Embedded controller.

Scanned with ACE Scanner

⇨ The following figure ACPI system was designed to enable the operating system to set-up and control the individual hardware components.



ACPI System

ACPI run-time components

1. ACPI tables: Describe the interface to the hardward.
2. ACPI registers: The constrained part of the hardware interface described (at least in location) by the ACPI subsystem description tables.
3. ACPI BIOS: ACPI system firmware. The firmware boots the machine and compatible with ACPI spec.

Global System states

- G0: Working (System operational)
- G1: Sleeping - no user threads, system looks off
- G2/S5: Soft off
- G3: Mechanical off

Processor power states:

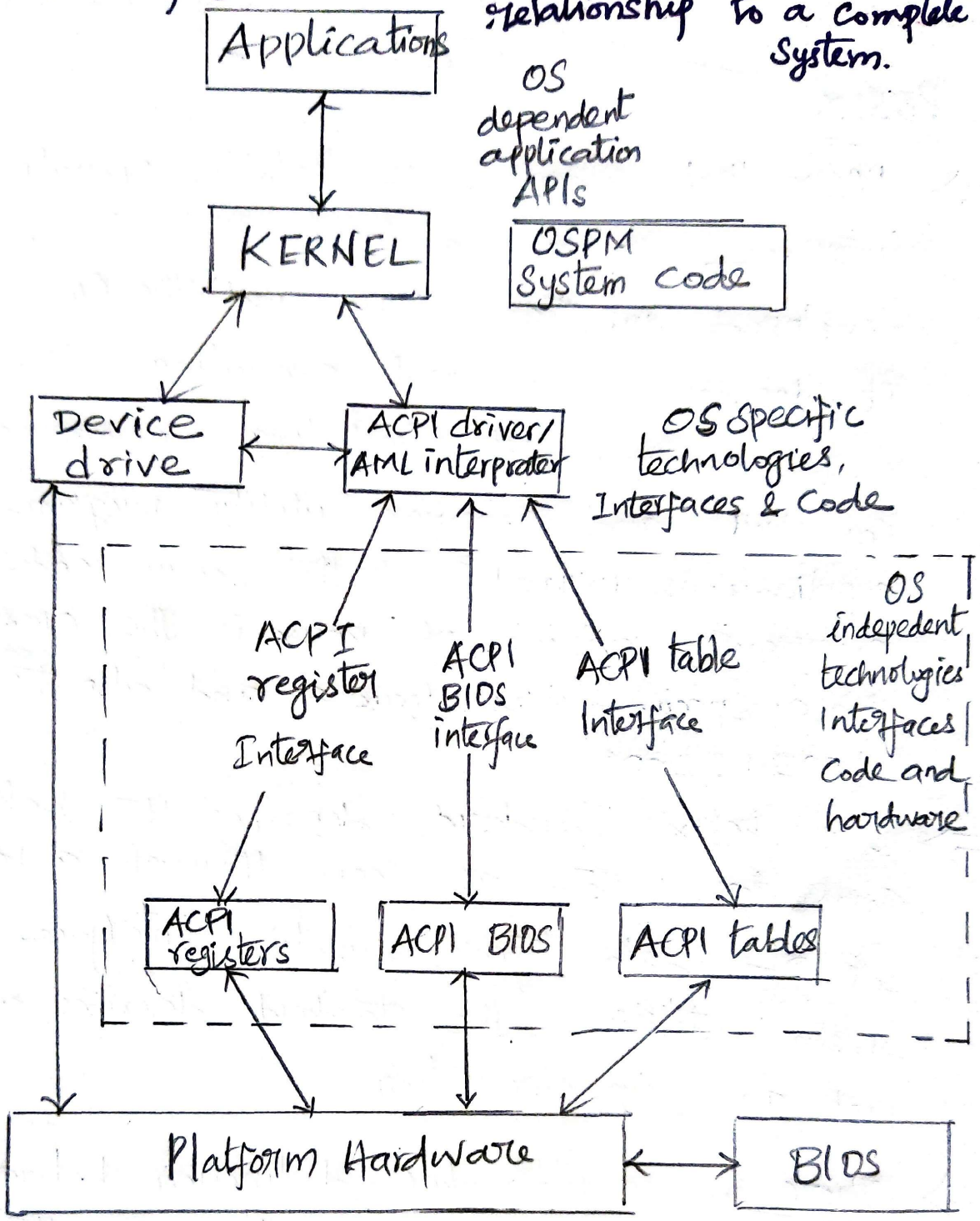
- C0: Full power, instruction execute
- C1: Processor stopped
- C2: Processor stopped, less power than C1
- C3: Processor stopped, cache ignore snoops

Device power states

D3: Off - Power off to device
D2: Less power than D1
D1: Less power than D0
D0: Fully-on.

Scanned with ACE Scanner

Advanced Configuration & Power Interface & its relationship to a complete System.



Sleeping States:

S0: System working

S1: low latency sleeping State Processor context ~~not~~ maintained

S2: low latency sleeping State Processor context not maintained.

S3: low latency sleeping state DRAM still maintained

S4: lowest power longest wake-up DRAM not maintained

S5: Soft off state

Scanned with ACE Scanner

Example Real Time Operating Systems.

POSIX

⇨ POSIX that stands for Portable Operating System interface is a standard that is being jointly developed by the IEEE and The Open Group. It defines a standard operating system interface and environment, including a command interpreter (or shell), and common utility program to support applications portability at the source code level. The current revision of POSIX is The Open Group Base Specifications Issue 6 and also the IEEE std. 1003.1 : 2001

⇨ The POSIX standard describes the behavior of a computer system as seen through a particular set of data types, function interface and system utilities. The standard describes an interface not an implementation.

⇨ There is no particular distinction between system functions and library functions, as the C99 libraries are included with the POSIX libraries.

⇨ The basic goal was to promote portability of applications programs across UNIX system environment by developing a clear, consistent and unambiguous standard for the interface specification of a portable

Scanned with ACE Scanner

operating system based on the UNIX system documentation. The standard codifies the common, existing definition of the UNIX system.

- The standard is composed by four major components:

1. **Base Definitions**: This includes general terms, concept and interfaces common to entire standard.

2. **System Interfaces**: This comprises the definitions for system service functions for the C programming language, function and portability issues, error handling and recovery.

3. **Shell and Utilities**: It contains the definitions for a standard source code-level interface to command interpretation services.

4. **Rationale**: It contains information that does not fit well into the rest of the document structure.

Posix.1: IEEE 1003.1-1990 adapted by ISO. As ISO/IEC 9945:1:1990 standard gives standard for base operating system API.

Posix.1b: IEEE 1003.4:1993 Gives standard APIs for real time OS interface including inter-process comm.

Posix.1c: Specifies multi thread programming interface

Scanned with ACE Scanner

⇨ To ensure program confirms to POSIX.1 standard user should define _POSIX_SOURCE as

1. #define _POSIX_SOURCE OR
2. Specify -D_POSIX_SOURCE to a C++ compiler.

Windows CE:

⇨ Windows CE is based on windows 95 with the usual interface, adapted for small devices. The development for this operating system under the code name Pegasus began in 1995.

Specially designed for micro-computers, the abbreviation CE stands informal for "Compact Edition".

⇨ The first version of windows CE requires as a minimum 4MB of ROM, 2MB of RAM and a processor of the SuperH3, MIPS 3000 or MIPS4000 architecture.

⇨ Windows CE 2.0 came in October 1997 with the first devices manufacturers itself. TrueType fonts improving now the appearance by the device manufacturers with a display of 640×480 pixel full VGA resolution and 24-bit color depth.

⇨ The manageable memory can now be up to 4MB. The software "Handled PC Explore" is renamed to ActiveSync.

Scanned with ACE Scanner

⇒ The update windows CE 2.10 in July 1998 allows the use of TCP/IP and the file system FAT 32. With the modular file wrapper can be incorporated up to 256 different file systems. The RAM can now be up to 16MB.

⇒ The new command line processor allows in this release for the first time the use of commands without a graphical user interface. An infrared port and USB controller increases the scope.

- Windows CE 3.0 is only available for ARM CPUs. As new feature the Bluetooth support was introduced.

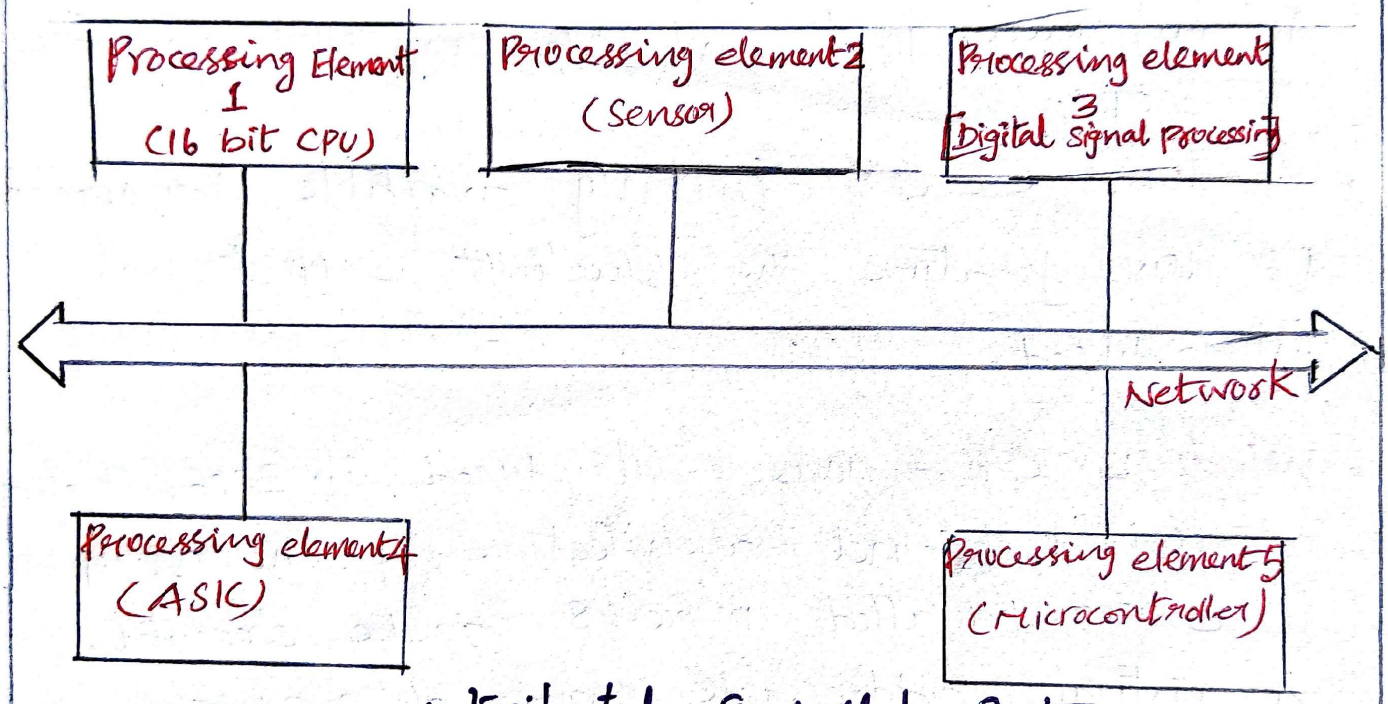
- windows CE6 was introduced in 2006. It offers a revised kernel architecture of the OS. up to 32,000 parallel processes can be executed. A virtual addressable range of 2 Gbyte is possible for every process

⇒ The multimedia capabilities have been expanded and now support HD-DVD, DVD-UDF 2.5, multichannel audio and much more. The compatibility to existing Windows CE applications and drivers are kept.

Scanned with ACE Scanner

Distributed Embedded Systems.

- In a distributed embedded system, several Processing Elements (PEs) are connected by a network that allows them to communicate. below figure shows an example of a distributed embedded system.



Distributed Embedded System.

- Processing element may includes DSP, CPU or microcontroller. Nonprogrammable unit such as the ASICs is also to implement as PE.

- By using this entire processing element, it forms bus topology. It is also possible to form other topology also. It is also possible that the system can use more than one network, such as when relatively independent functions require relatively little communication among them.

Scanned with ACE Scanner

~~UNDERLINE~~ DISTRIBUTED EMBEDDED ARCHITECTURE

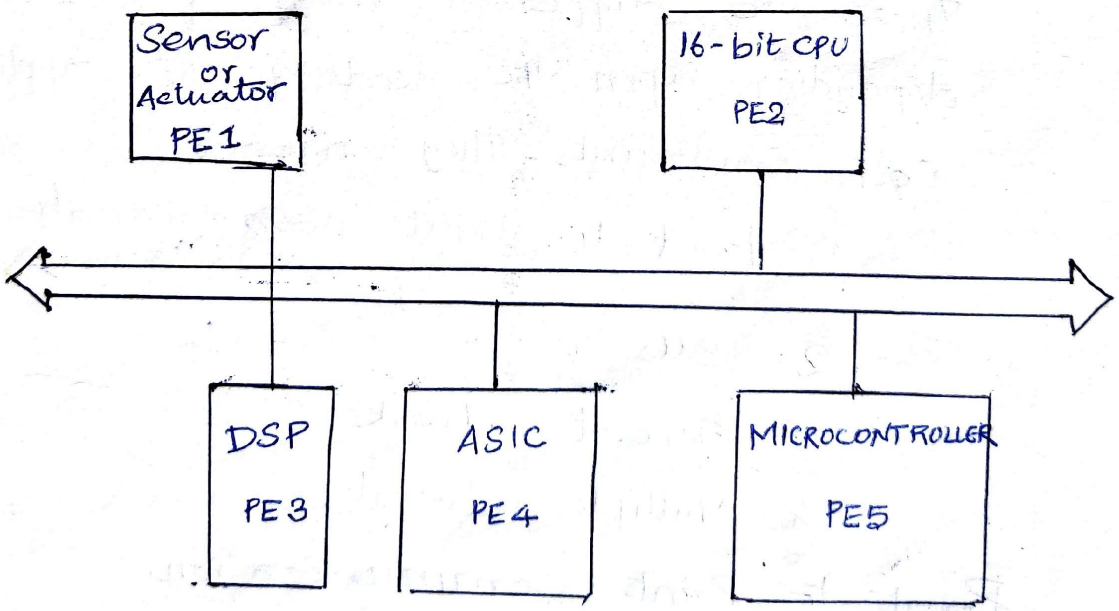
→ A distributted embedded system can be organized in many different ways.

→ The basic element or units
- Processing Elements - DSP, CPU, Sensor/actuator ASIC & microcontroller.
- Network

→ The network can be any topology, normally bus topology is used. More than one network also can be implemented.

→ The PEs are connected using communication link.

→ The system of processing elements and networks forms the hardware platform on which the application run.



An example distributed Embedded System

→ When analysing network performance - the speed at which PEs can communicate over the bus would be difficult when allowed bus arbitration.

Scanned with ACE Scanner

⇒ The distributed systems are necessary because the devices that the PEs communicate with are physically seperated.

⇒ If the deadlines for processing the data are short, then it may be more cost effective.

⇒ An important advantage of a distributed system is that if several CPUs are in the system, you can use one to generate inputs for another and watch its output.

i.e good isolation makes easy to diagnose the problem in a part of the system while other parts are working.

Hardware and Software Architecture

There are different ways of distributed system depending upon the needs of the application & cost constraints. They are

1) Point-to-Point communication
2) BUS
3) General networks.
4) Multiple networks.

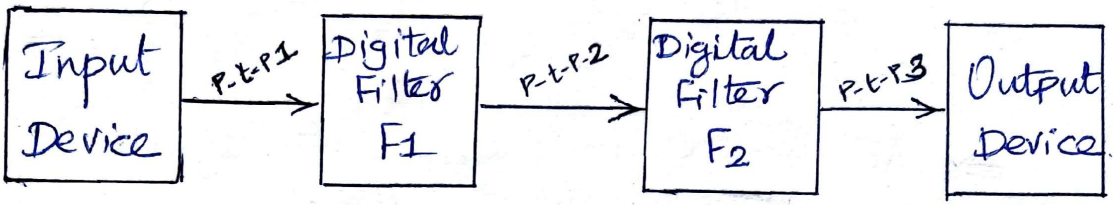
Point-to-Point communication

→ A point-to-point link establishes a connection between exactly two processing elements (PE).

→ Point-to-point links are simple to design precisely because they deal with only two components.

Scanned with ACE Scanner

⇒ The figure below shows a simple example of a distributed embedded system built from point to point.



A simple processing system built from Point-to-Point links

→ The input signal is sample by the input device and passed to the digital filter F1, over a point-to-point link (P-t-P link 1).

→ The result of the filter is sent through a second point to point link to the digital filter F2.

→ The result is then sent to output device over a third point-to-point link (P-t-P3)

→ Using point-to-point connection allows both F1 & F2 to receive a new sample and send a new output at the same time no worry about collisions on the communication network.

Buses

⇒ It is possible to build a full-deplex, point-to-point connection that can be used for simultaneous communication in both directions between the two PEs.

⇒ In the bus topology, multiple devices to be connected to the like.

Scanned with ACE Scanner

⇒ Like a microprocessor bus, PEs connected to the bus have addresses.

⇒ Communication on the bus uses a form of packets. It contain destination address & data to be delivered & includes error detection/correction information as parity.

⇒ The header in the packet signals to other PEs that the bus is in use.

⇒ It is the responsibility of the transmitting PE to devide its data into packets; the receiving PE must reassemble the complete data message from the packets.

Header	Address	Data	Error correction

Format of a typical message on a Bus

⇒ Distributed buses must be used the arbitration to control simultaneous access. they are defined as follows

* Fixed-priority arbitration
* Fair arbitration

Fixed priority arbitration -

→ Priority assigned to the devices from High to low

→ The low-priority device will not able to transmit anything until the high-periority device has sent all its data packets

Fair arbitration -

→ Round-Robin arbitration is most commonly used of the fair arbitration schemes.

→ It requires PCI bus because most implementation of PCI uses round-robin arbitration.

Scanned with ACE Scanner

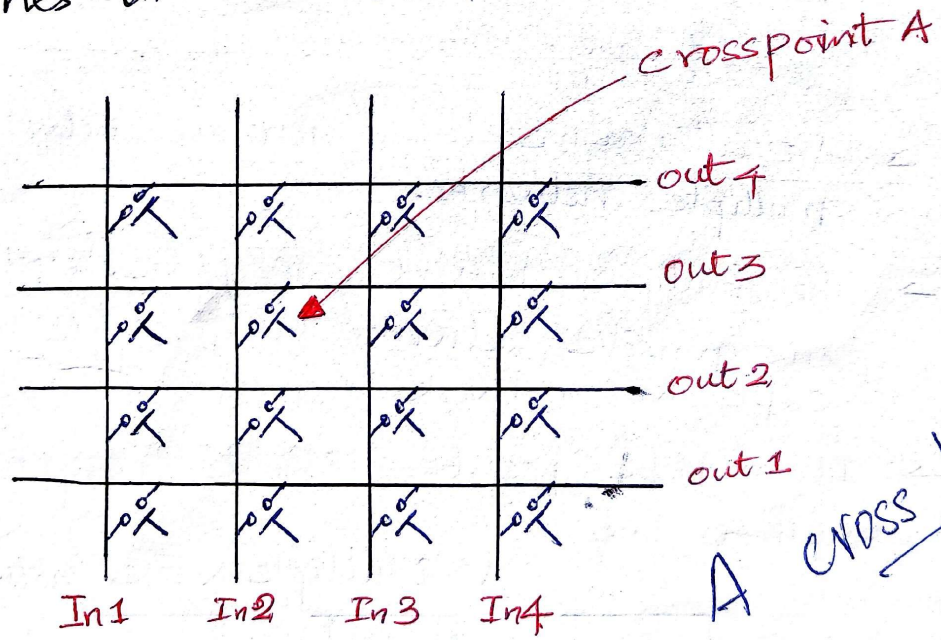
General Networks

→ A **Bus** has limited bandwidth. Since all devices connect to the bus, communication can interfere with each other.

→ Usually crossbar network is used, because It allows all combinations of input/output connections to be made.

→ A crosspoint is a switch that connects an input to an output.

→ To connect an input to an output, we activate the crosspoint at the intersection between the corresponding input & output lines in the crossbar



A cross bar network

→ In the above example, to connect In2 & Out3, crossbar 'A' should be activated.

→ The major drawback of crossbar network is expensive.

→ The size of the network grows as the square of number of inputs (if inputs = outputs)

Scanned with ACE Scanner

Multiple Networks

→ The crossbar is a direct network in which messages go from source to destination without going through any memory elements.

→ Multistage networks have immediate routing nodes to guide the data packets.

→ Most networks are blocking, particularly A bus is maximally blocking network since any message on the bus blocks message from any other node.

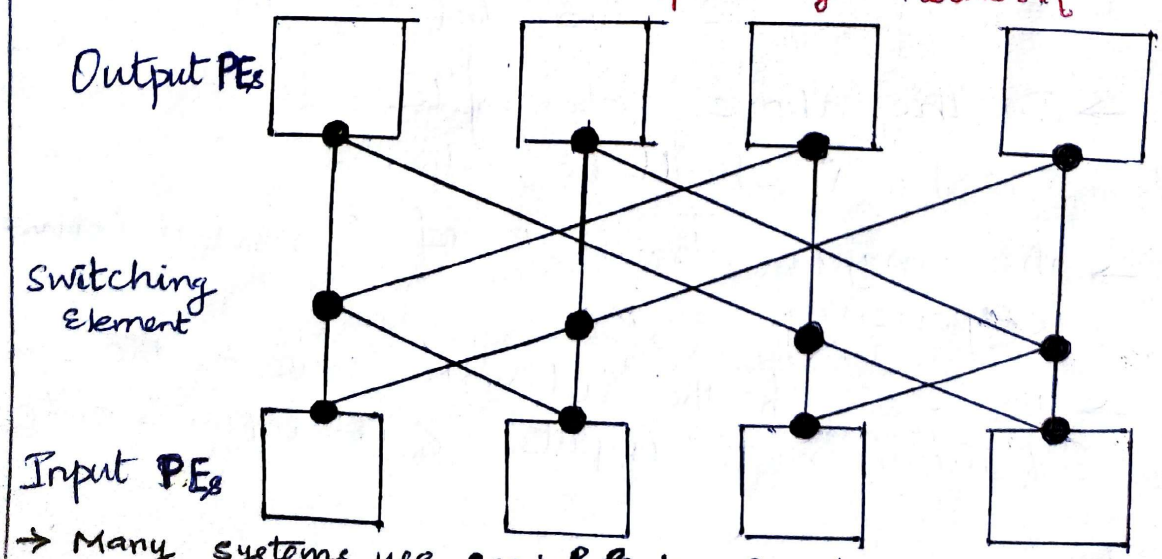
→ The microprocessor embedded network implement the basic communication functions in hardware & sofware.

→ An alternative to a non-bus network is to use **multiple networks.**

→ The multiple network may be cheaper to use two slow, inexpensive networks than single high performance, expensive networks.

→ It may be possible to use simpler topologies such as buses.

A Multiple stage network



→ Many systems use **serial links** for low-speed communication & **CPU** for higher speed and volume data transfer.

Scanned with ACE Scanner

Networks for Embedded Systems

* There are several system buses have been used to to build distributed embedded systems.

* Multibus and VME were developed by Intel & Motorola, respectively for multicard computer systems and have been widely used in industrial applications.

* The ISA bus has been used to support many I/O cards for PC-based embedded systems.

* The I^2C bus is used in microcontroller based systems.

* The Echelon LON network was developed for home & industrial automations.

* Many DSPs supply their own interconnect structures for multiprocessing.

* Many networks designed for embedded networks & general purpose computing.

The I^2C [Inter Integerated circuits]

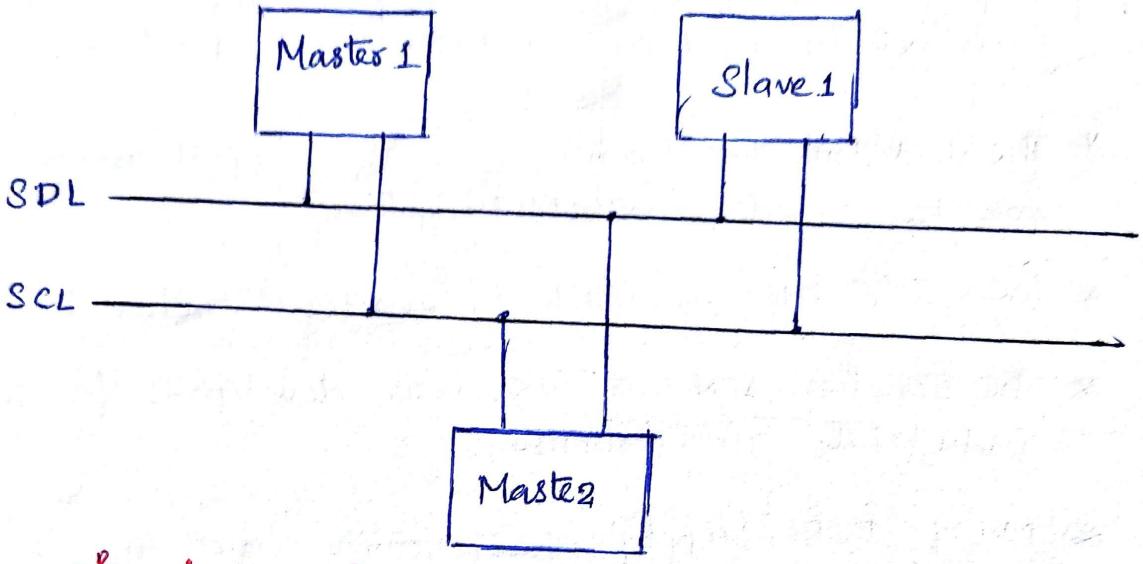
- It is commonly used to link microcontrollers to systems.
- It has even be used for the command interface in an MPEG-2 video chip; a seperate bus was used for High speed video & setup information is transmitted using I^2C.

Physical Layer

- Advantages of I^2C are
  - low cost
  - simple implementation
  - moderate speed [upto 100 kilobits/sec – standed bus
    upto 400 kilobits/sec – extented bus]
  - It uses only two lines

Scanned with ACE Scanner

* The below figure shows a typical I^2C bus system.
* Every node in the network is connected to SDL & SCL.
* One or more than one nodes may be act as **master** and other nodes are act as **slave**.



Structure of I^2C bus system

Electrical Interface

→ The bus does not define particular voltages to be used for high or low so that either bipolar or MOS circuits can be connected to the bus.

→ A pull-up resistor keeps the default state of the signal high and transistor are used in each bus device to pull-down the signal when 0s & 1s to be transmitted.

→ Open collector/open drain signaling allows several devices to simultaneously write the bus without causing electrical demage.

→ The I^2C bus is designed as a multi-master bus on various times. As a result, there is no global master.

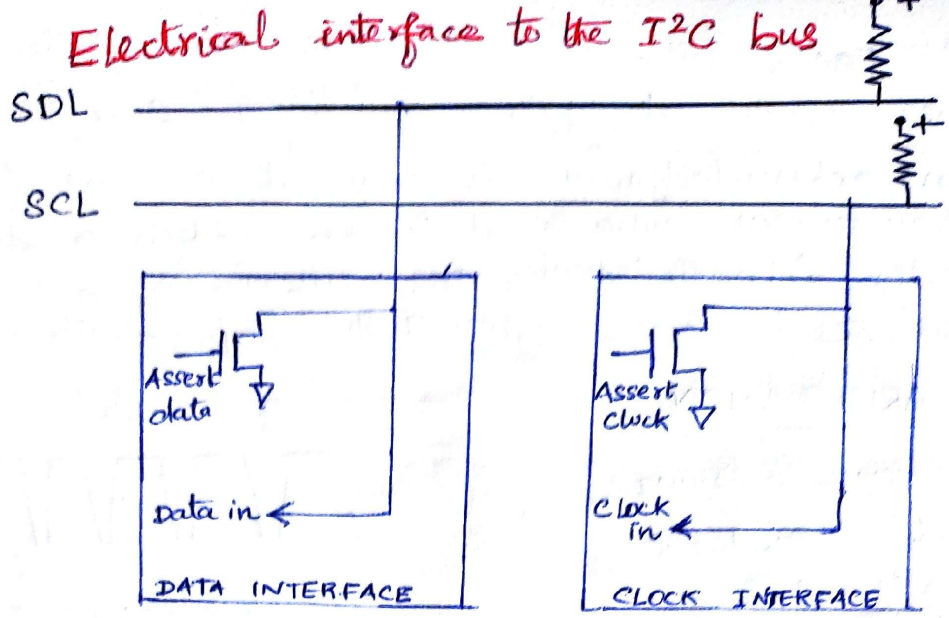
→ A master drives both SCL & SDL when it is sending data.

→ When the bus is idle, both SCL & SDL remain high.

→ Each master device must listen to the bus while transmitting to be sure that it is not interfering with another message - if the device receives a different value, it means it is interferring with another message.

Scanned with ACE Scanner

Electrical interface to the I²C bus



DATA LINK LAYER:

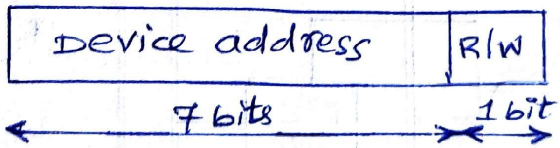
→ Every I²C device has an address & no two devices in the system have the same address.

→ A bus transaction is comprised of a series of one-byte transmissions and an address followed by one or more data bytes.

→ when the master wants to write slave, it transmits the slave's address followed by the data.

→ An address transmission includes

7-bit - address
1-bit - data direction — [0 for writing from the master to slave
1 for reading from the slave to master.



→ A bus transaction is initiated by a start signal & completed with an end signal as follows:

* A start is signaled by leaving the SCL high & sending a 1 to 0 transition on SDL

* A stop is signaled by setting the SCL high & sending a 0 to 1 transition on SDL

BYTE FORMAT

→ The transmission starts when SDL is pulled low while SCL remain high.

→ After this start condition, the clock line is pulled low to intiate data transfer.

Scanned with ACE Scanner

⇒ At each bit, the clock line goes while the data line assumes its proper value of 0 or 1.

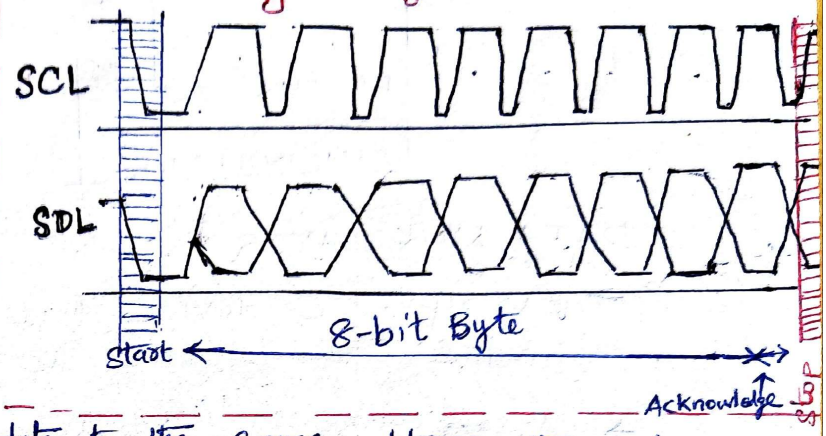
→ An acknowledgment is sent at the end of every 8-bit transmission, whether it is an address or data

→ After acknowledgment, the SDL goes 0 to 1 while the SCL is high, signaling the stop condition.

BUS ARBITRATION:

If a device is trying to send a logic 1. but hears a logic 0, it immediately stops txg. & gives the other sender priority.

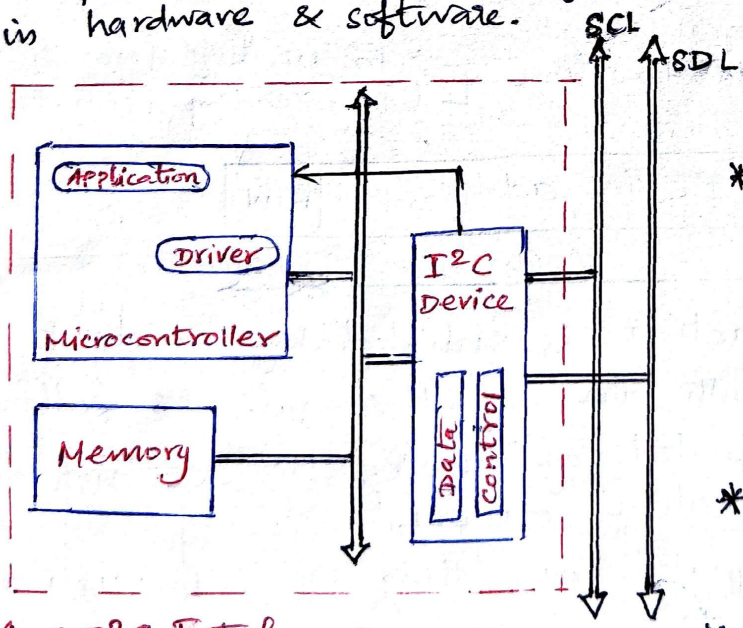
Transmitting a byte on the I^2C bus



→ If two devices are trying to send identical data to the same address, then of course they never interfere & both succeed in sending their message.

Application Interface

⇒ The I^2C interface on a microcontroller can be implemented with varying percentage of the functionality in hardware & software.



An I^2C Interface in a microcontroller

* The I^2C device takes care of generating the clock & data.
* The application code routines to send an address & a data byte and so on. which then generates the SCL & SDL, acknowledges, & so forth.
* One of the microcontroller's timer is typically used to control the length of on the bus.
* Interrupt may be used to recognize bits.

* However, when used in master mode, polled I/O may be acceptable if no other pending tasks can be performed, since master initiates their own transfer.

Scanned with ACE Scanner

The CAN Bus [Controller Area Network]

- The CAN bus uses bit-serial transmission.
- CAN can run at rates of 1 Mb/second over a twisted pair connection of 40 meters. An optical link can also be used.
- The bus protocol supports multiple master on the bus.
- CAN provide a good example for a distributed network such as automobiles.
- Each node in the CAN bus has its own electrical drivers and receivers that connect the node to the bus is wired-AND fashion.
- In CAN terminology, a logical 1 on the bus is called recessive & a logical 0 is dominant.
- The driving circuits on the bus cause the bus to be pulled down to '0' if any node on the bus pulls the bus down (making 0 dominant over 1).
- When all nodes are transmitting 1s, the bus is said to be in the recessive state.
- When a node transmits a '0', the bus is in the dominant state.
- Data are sent on the network in packets known as data frames.
- CAN is a synchronous bus - all transmitters must send at the same time for bus arbitration to work.
- Nodes synchronous them selfs to the bus by listing to the bit transitions on the bus.
- The first bit of a data frame provides the first synchronization opportunity in a fram.

Data frame

- ⇒ The data fram starts with a '1' & ends with a string of seven zeroes.
- ⇒ The first field in the packet contains the packets destination address and is known as arbitration field.

Scanned with ACE Scanner

⇒ The destination identifier is 11 bits long & last one bit is RTR bit (Remote Transmission Request).

⇒ If RTR = 0, then read the data from the device specified by the identifier.
RTR = 1 write operation.

⇒ The control field provides an identifer extention & 4-bit length for the data field with a '1' in between.

⇒ The data field is from 0 to 64 bytes, depending on the value in the control field.

⇒ A cyclic redundancy check (CRC) is sent after the data field for error detection.

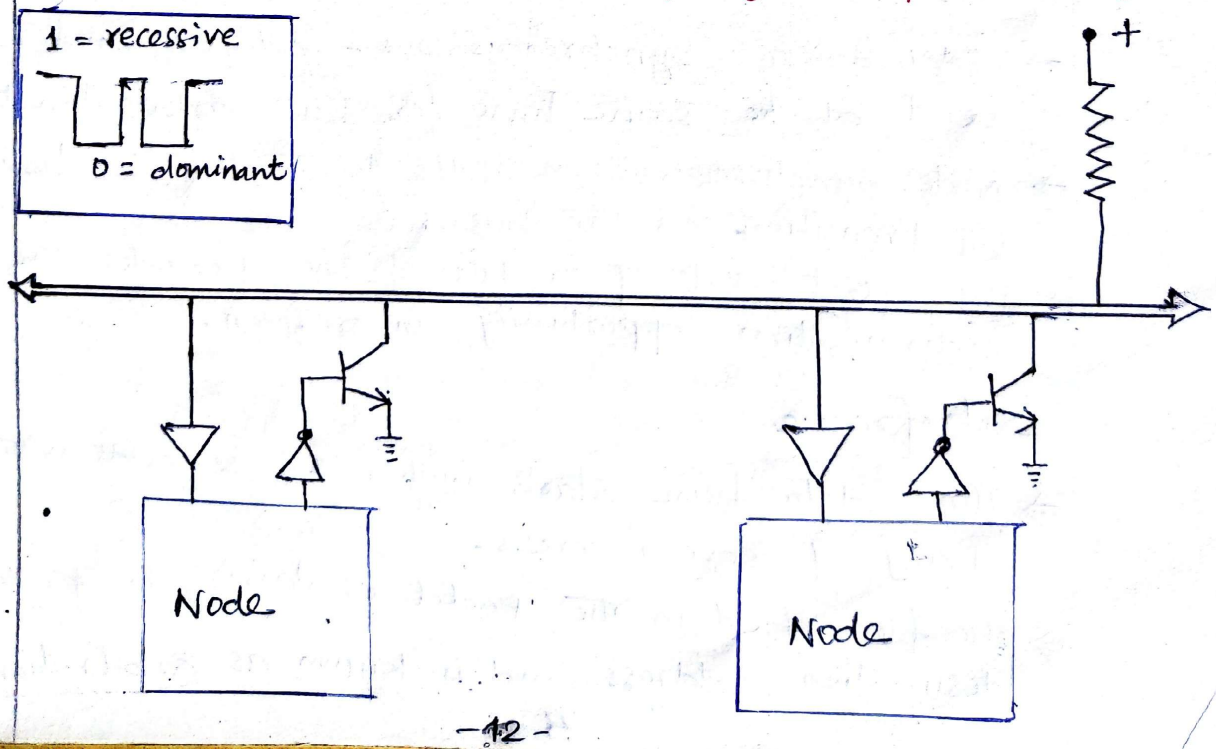
⇒ The acknowledge field is used to inform the reception is correct or error

⇒ When the receiver detects the error, it forces the acknowledgement value to '0'

⇒ If the sender sees a '0' on the bus in ACK slot, it knows that it must retransmit.

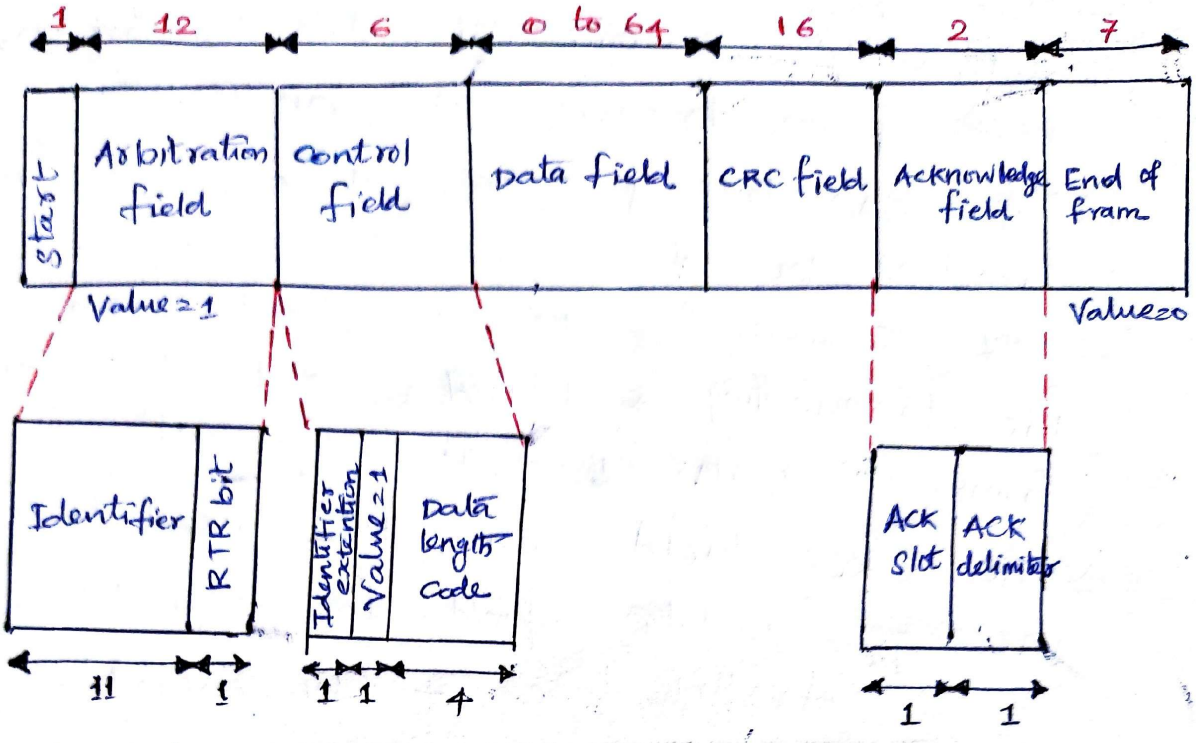
⇒ The ACK slot is followed by a single bit delimiter followed by the end-of fram field.

Physical & Electrical organization of a CAN bus.



Scanned with ACE Scanner

The CAN data fram format



Arbitration

⇒ The CAN network uses an arbitration using Carrier Sense Multiple Acess with Arbitration on Message Priority [CSMA/AMP].

⇒ This method is similar to the I^2C bus's arbitration.

⇒ CAN supports a data push programming style.

⇒ All nodes in the network are simultaneously transmit. When a node sees '0' (dominant bit) in the identifier, it stops transmitting.

⇒ By the end of the arbitration field, only one transmitter will be left.

⇒ The identifier field acts as a priority identifier, with the all-0 identifier having the highest priority.

Error handling

⇒ An error frame can be generated by any node that detect error on the bus.

⇒ When an error is detecting, a node interrupts the current transmission with an error frame which consists of an error flag followed by an error delimiter of 8 recessive bits.

Scanned with ACE Scanner

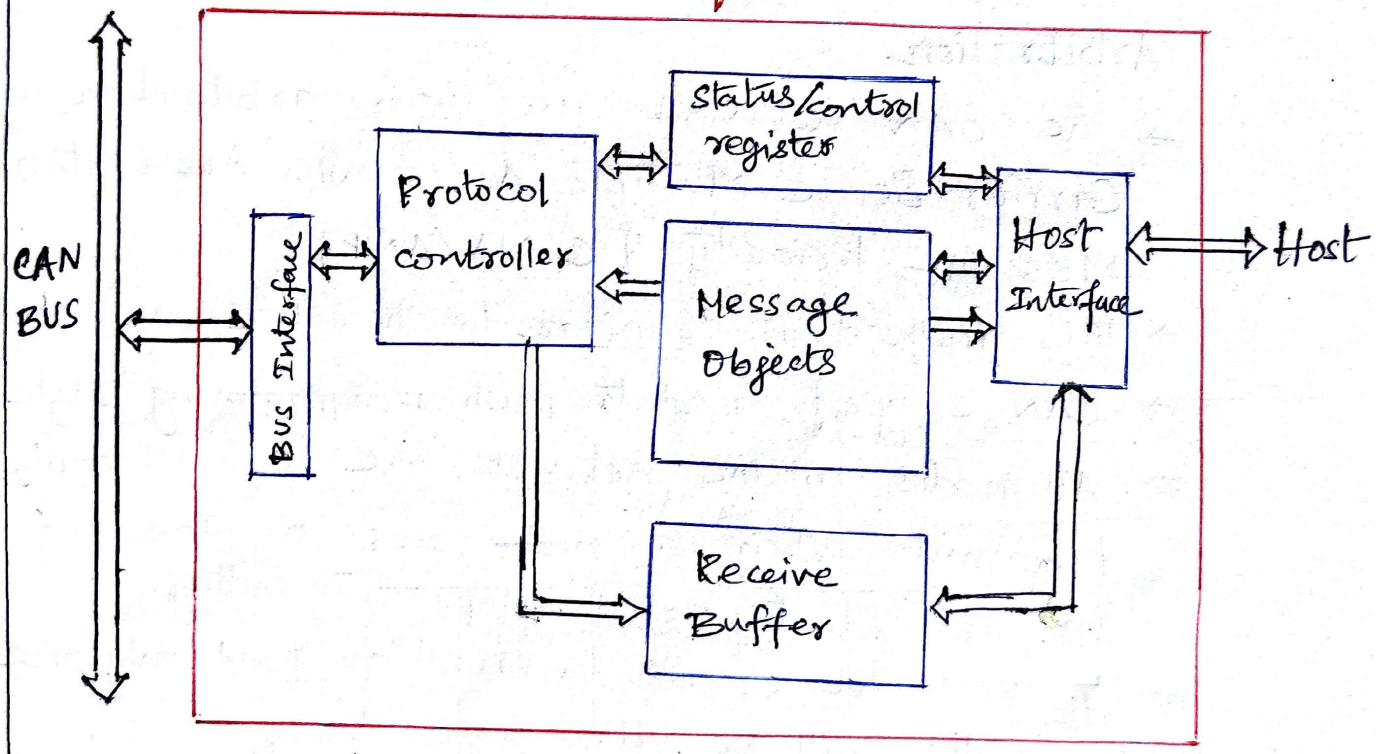
⇒ The error delimiter field allows the bus to return to the quiescent state (inactive state) so that fram transmission can resume.

⇒ The CAN bus provides an overload fram during inactive period.

⇒ If the node sees the overload fram, it stops the transmitting & delayed for two overload frams in a row then it retransmitts.

⇒ The CRC field can be used to check a message's data field for correctness.

Architecture of CAN controller.



⇒ The transmitting node retransmits the data fram until it gets the acknowledgement

⇒ The CAN architecture implements the physical & Data link layer [since CAN is bus, it does not need network layer] to establish end-to-end connection.

⇒ The protocol block responsible for determining when to send messages, when a message must be resent due to arbitration losses, & when a message should be received.

Scanned with ACE Scanner

Classification of Scheduling algorithm

Scheduling Algorithm.

- Offline Scheduling (static, clock driven)
- On-line Scheduling (dynamic)
  - Static-priority Scheduling (RM, WORKS, SCHED-FIFO)
  - Deadline driven Scheduling (EDF, ...)
  - General purpose OS Scheduling (fair, interactive ...)

Commonly used approaches to scheduling real time s/m's.

1) clock driven.

2) Weighted round robin – primarily used for scheduling real time traffic in high speed switched networks

3) Priority driven.

Scanned with ACE Scanner

Clock driven approach

* decisions are only made at a priority chosen time instants

  (i.e) – decisions on what jobs execute at what times are made at specific time instants

* It is Sufficient to have a h/w timer (no need for OS)

* Regularly spaced time instants (periodical)

* Schedule is computed offline and stored for use at run time.

  - All parameters of hard real-time jobs are fixed and known.
  - Scheduling overhead during runtime is minimal
  - Complexity of the scheduling algo is not important
  - Good (optimal) off-line schedules can be found.

Disadv

  - No flexibility.

* Applicable Only when we know all about the System in advance

  - Fixed set of tasks, fixed and known task parameters and resource requirements.
  - met by many Safety-critical applications
  - Easier to certify

Scanned with ACE Scanner

Weighted Round Robin Approach

Round robin Approach:- (processor-sharing algo)

Uses

1) used for scheduling time-shared appln.

* when a job ready for execution, it joins First-In-First-Out (FIFO) Queue.
* The job at the head of the queue executes for atmost one time slice
* If the job does not complete by the end of the time slice, it is preempted and placed at the end of the queue to wait for its next turn.

Exa

* when there are 'n' ready jobs, each job gets one time slice every 'n' time slices,
* Length of time slice is short, the execution of each job starts immediately after it becomes ready.
* Each job gets $1/n^{th}$ share of the processor when there are 'n' jobs ready for execution
* Hence the round robin algo is called the processor sharing algo.

Scanned with ACE Scanner

Weighted round-robin algorithm :-

Uses

1) Used for scheduling real-time traffic in high speed switched n/ws.

- Rather than giving all the ready jobs equal share of the processor, different jobs may be given different weights
- Weights of the job refers to the fraction of processor time allocated to the job

* a job with weight wt gets wt time slice every round.
* By adjusting the weights of jobs, we can speed up or retard the progress of each job
* It is not suitable to schedule precedence constraint jobs, since it gives each job a fraction of the processor.
* It is suitable for pipeline jobs

Scanned with ACE Scanner

Example :-

Consider 2 job sets

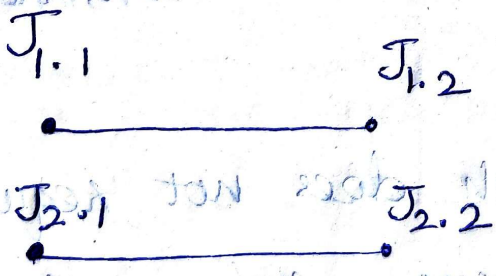
$$J_1 = \{ J_{1.1} , J_{1.2} \}$$

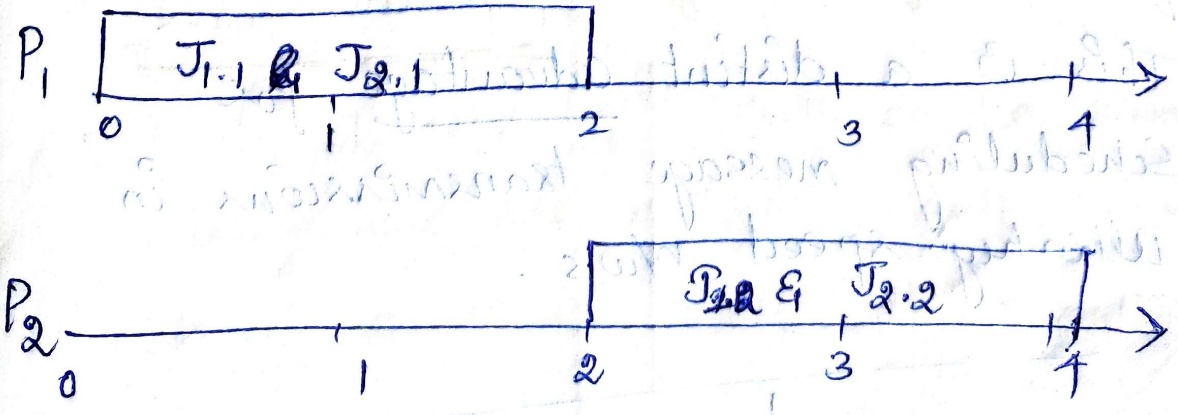
$$J_2 = \{ J_{2.1} , J_{2.2} \}$$

* all jobs release time are 0.
* their execution times are 1.

* $J_{1.1}$ and $J_{2.1}$ execute on processor P_1
* $J_{1.2}$ and $J_{2.2}$ execute on processor P_2.

* $J_{1.1}$ is the predecessor of $J_{1.2}$
* $J_{2.1}$ is the predecessor of $J_{2.2}$





Scanned with ACE Scanner

Priority driven

* never leave any resources idle intentionally
* Scheduling decisions are made when events such as releases and completions of jobs occur.

Hence priority driven algo are event driven

Other names ① Work conserving scheduling.

② greedy scheduling:

* Bcz it tries to make locally optimal decisions
* when a processor or resources available and some job can use it to make progress. Such algo never makes the job wait.

③ List scheduling

* Implemented by assigning priorities to jobs.
* Jobs ready for execution are placed in one or more queues ordered by the priorities of the job.
* At any scheduling time, the job with highest priority are scheduled & executed on available processors

Scanned with ACE Scanner

Scheduling algorithms

— used in Non-real time S/m's are priority driven

FIFO } — these algo assign priorities
LIFO } to jobs according to their release times.

shortest execution time → first

SETF } — these algo assign priorities
LETF } on the basis of job execution times.

Scanned with ACE Scanner

Effective release times and Deadlines

* The gn release times & deadlines of jobs are sometimes inconsistent with the precedence constraints of the job.
* Therefore a set of effective release times & deadlines are derived from these timing constraints
* The derived timing constraints are consistent with the precedence constraints.

Only one processor

Rules

Effective Release Time :-

The Effective release time of a job without predecessors is equal to its gn release time.

$$ERT_{w/o\,pre} = RT_{gn}.$$

The Effective release time of a job with predecessors is equal to the maximum value among its gn release times & the effective release times of all of its precedecessors.

$$ERT_{pre} = \max(RT_{gn}, ERT_{p}).$$

Scanned with ACE Scanner

Effective Deadline

The effective deadline of a job without a successor is equal to its given deadline.

$$\boxed{ED_{ws} = D_{gn}}$$

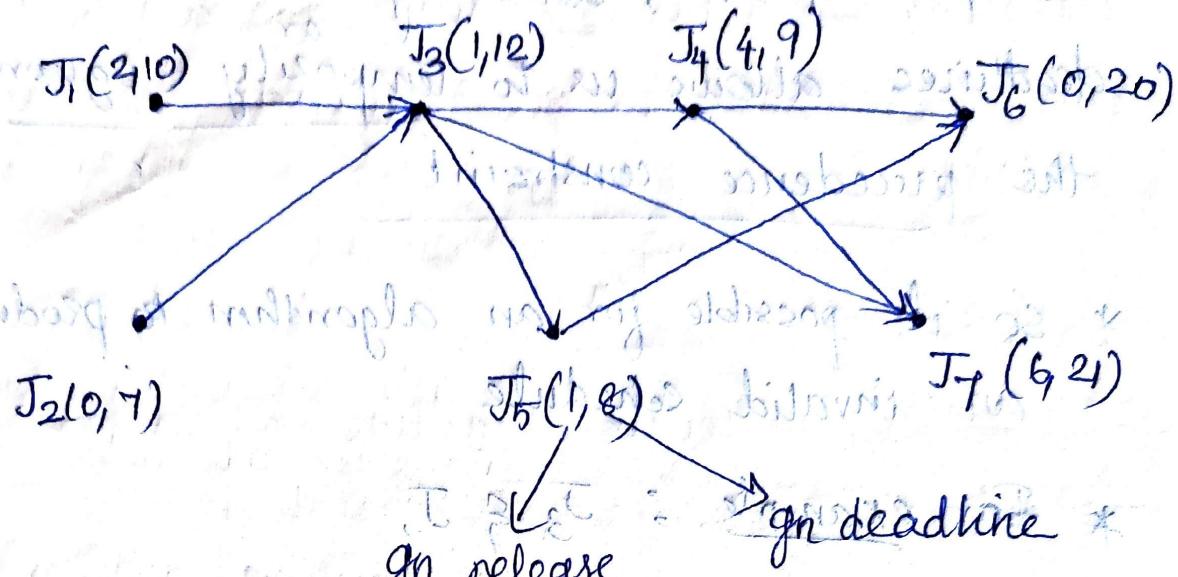
The effective deadline of a job with ~~out~~ a successors is equal to the minimum value among its gn deadline and the effective deadlines of all of its successors.

$$\boxed{ED_s = \min(D_{gn}, ED_s)}$$

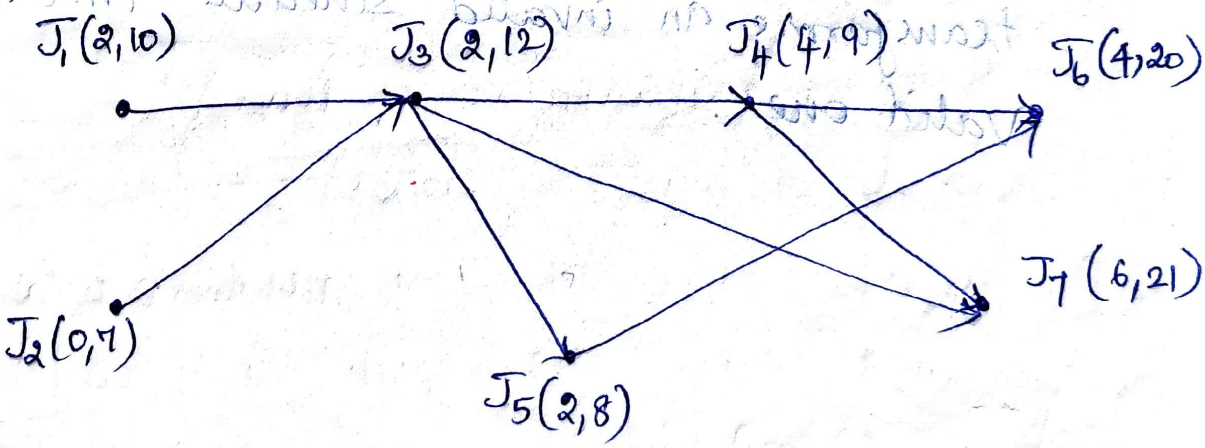
* The effective release times of all jobs can be computed in one pass through the precedence graph in $O(n^2)$ time where 'n' is the no. of jobs.
* The effective deadlines can be computed in $O(n^2)$ time.

Scanned with ACE Scanner

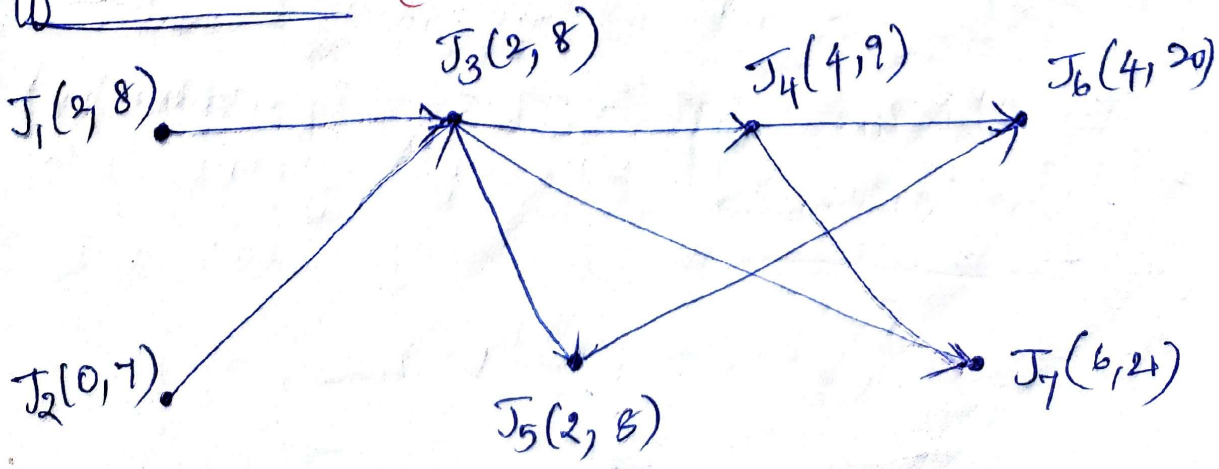
Example :- of Effective timing constraints



Effective RT (Largest)



Effective deadline (Smallest)



Scanned with ACE Scanner

Disadv

* When there is only one processor and jobs are preemptable, working with ERT & deadlines allows us to temporarily ignore the precedence constraint

* So it possible for an algorithm to produce an invalid schedule

* For example : J_3 & J_1

* This pbm is overcome by adding the step to swap the 2 jobs, the swapping transforms an invalid schedule into a valid one.

Scanned with ACE Scanner

Merits of Priority driven

* Easy to Implement.
* simple priority algo & for these priority algo, the run-time overhead (due to maintaining a priority queue of ready jobs) can be made very small.
* does not require the info on the release times & execution times of the jobs a priori. This advantage makes it suitable for applications with varying time & resource requirements.

Demerits

* It has not been widely used in hard real time s/m's (especially safety-critical s/m's).

* Reason :-
  - timing behavior is nondeterministic when job parameters vary.
  - It is difficult to validate that the deadlines of all jobs schedule in a priority driven manner. indeed meet their deadlines

Scanned with ACE Scanner

Predictability of Executions

* The validation pbm is easy whenever the exe behavior of the set J is predictable (i.e no scheduling anomalies).

* **Predictability**

The schedule of J produced by the gn Scheduling algo when the execution time of every job has its maximum value, is the Maximal Schedule of J.

The Schedule of J produced by the gn Scheduling algo when the exe time of every job has its actual minimum value is the minimal schedule.

when the exe time of every job has its actual value, the resultant schedule is the actual schedule of J.

* Since the range of exe time of every job is known, the maximal and minimal schedules of J can be easily be constructed. ~~when the~~

* In contrast, its actual schedule is unknown b'cx the actual values of the execution times are known.

Scanned with ACE Scanner

* The execution of J (under the gn priority-driven scheduling algo) is predictable if the actual start time & actual completion time of every job according to the actual job bounded by its start times & completion times according to the maximal and minimal schedules.

- $s(J_i)$ - actual start time of J_i.
- $s^+(J_i)$ and $s^-(J_i)$ be the start times of J_i according to the maximal & minimal schedules.
- J_i is start time predictable, if

$$\boxed{s^-(J_i) \le s(J_i) \le s^+(J_i)}$$

- $f(J_i)$ be the actual completion time of J_i according to the actual schedule of J.
- $f^+(J_i)$ and $f^-(J_i)$ be the completion times of J_i according to the maximal & minimal schedule of J.
- J_i is completion time predictable, if

$$f^-(J_i) \le f(J_i) \le f^+(J_i)$$

Scanned with ACE Scanner

* The execution of J_i is predictable, if J_i is both start time and completion time predictable.

* The execution behavior of the entire set J is predictable, if every job in J is predictable.

<u>exa</u> J_4 is not completion time predictable, & the s/m is not predictable.

* execution of independent, pre emptable but nonmigratable jobs is not predictable.

Validation Algorithms & their performance

- A validation algo allows us to determine whether all jobs in a s/m indeed meet their timing constraints despite scheduling anomalies.

* The merits (correctness) of validation algo are measured in terms of their

complexity
robustness
Accuracy.

Scanned with ACE Scanner

- At $t=2$, J_7 has the next highest level(3) so it goes next.
- At $t=3$, J_3, J_5, J_8 and J_9 are released. J_5 has the next highest level(2), so it runs.
- At $t=4$, either J_2 or J_8 could run b'cz both have level 1. But At this point J_2 has already missed its deadline
- At $t=5$, either J_2 or J_8 could run.
- At $t=6$, J_3, J_6, J_9 are all eligible to run and are all at level 0.

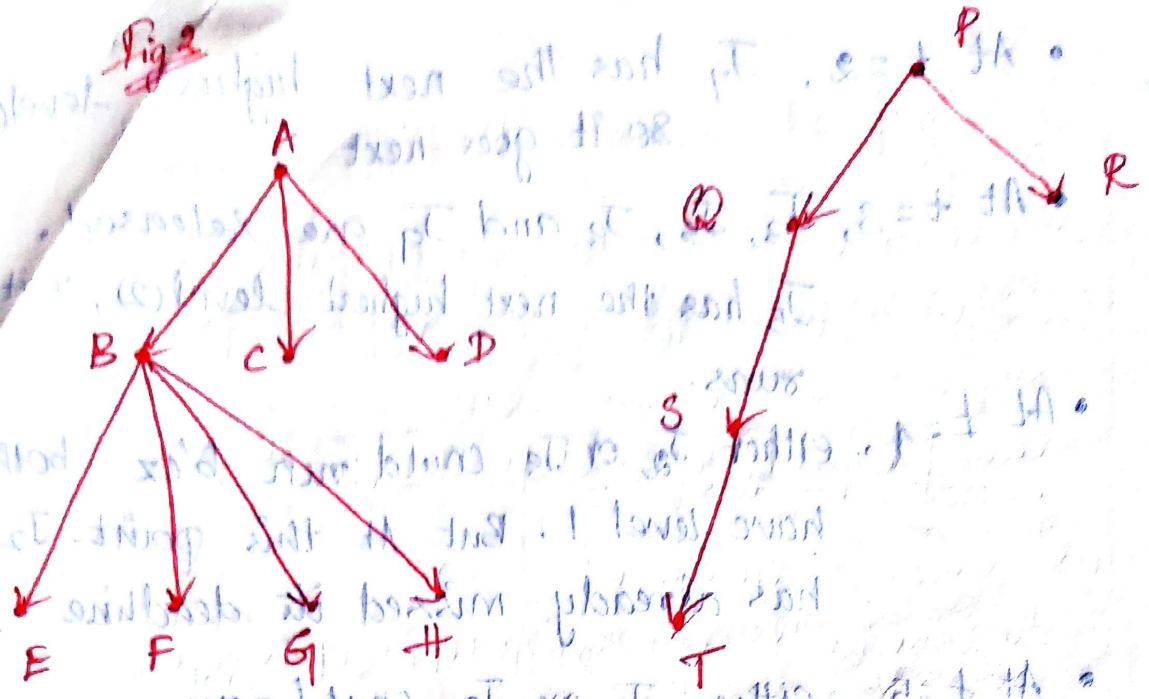
Corollary/ conclusion

Since J_2 and J_3 miss their deadlines. This is not a optimal scheduling algo

2) Problem 2: The Execution times of the jobs in the precedence graph in fig2 are all equal to 1 and their release times are identical. Give a non pre-emptive optimal schedule that minimizes the completion time of all jobs on three processors. Describe briefly the algo you used to find the schedule.

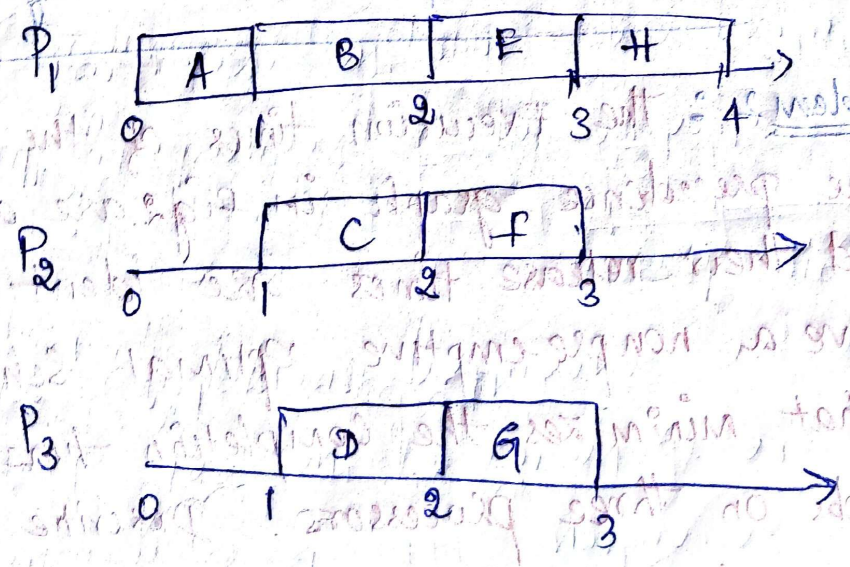
Scanned with ACE Scanner

Fig 2



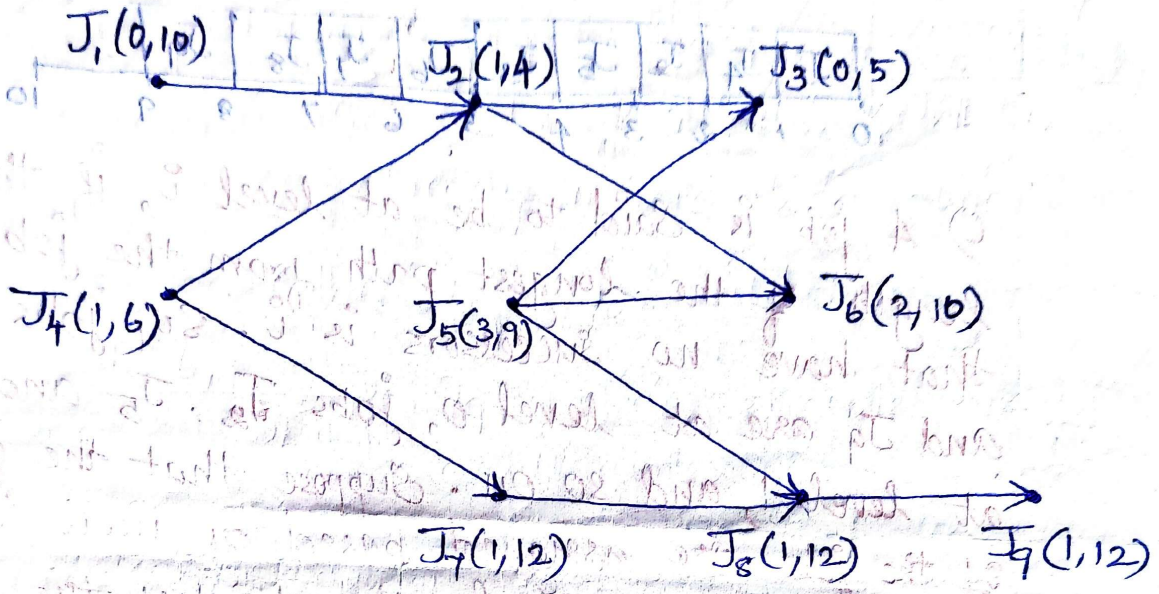
Soln:-

- Execution time of all jobs equal to 1.
  Release times are identical.
  Non pre emptive optimal solution



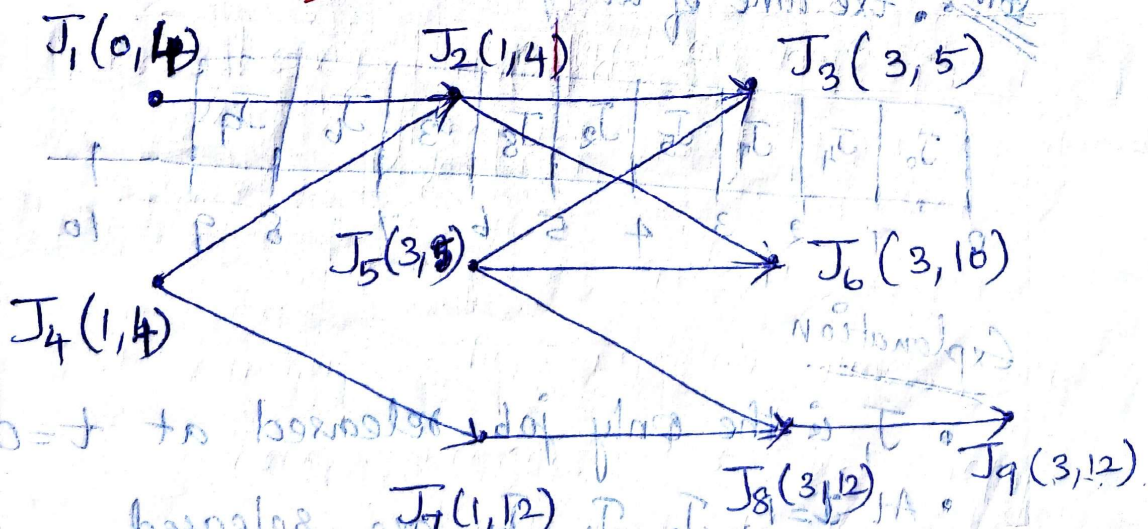
CHAPTER : 4 . Exercises Problem

① The feasible interval of each job in the precedance graph in fig. is given next to its name. The execution time of all jobs are equal to 1.



(a) Find the effective release times and deadlines of the jobs in the precedence graph.

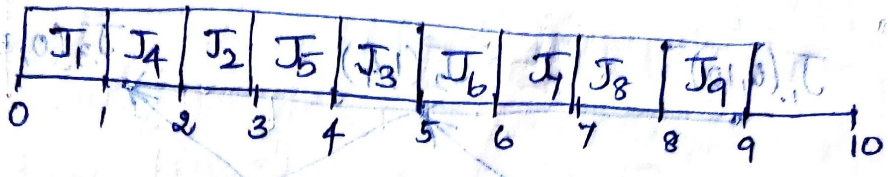
Soln :- ERT & ED



b) Find an EDF schedule of the jobs

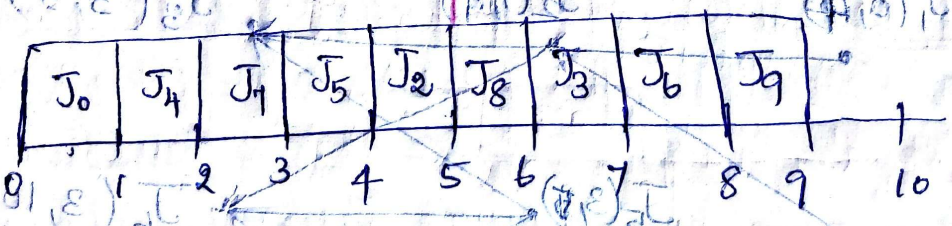
Soln:- execution time of all job = 1

EDF schedule based on ERT & EDL.



c) A job is said to be at level i, if the length of the longest path from the job to jobs that have no successors is i. So, jobs J_3, J_6 and J_9 are at level 0, jobs J_2, J_5 and J_8 are at level 1, and so on. Suppose that the priorities of the jobs are assigned based on their levels: the higher the level, the higher the priority. Find the priority-driven schedule of the jobs in fig 1. according to this priority assignment.

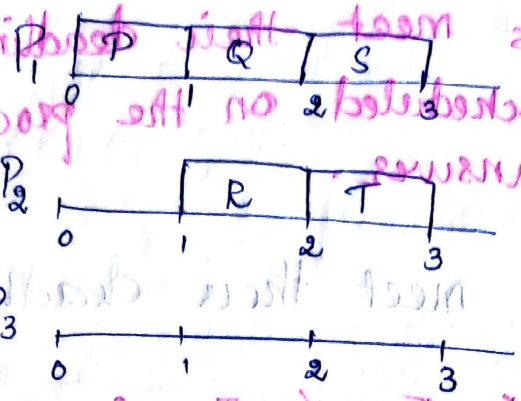
Soln:- • Exe. time of all jobs is 1.



Explanation

- J_1 is the only job released at $t=0$.
- At $t=1$, J_2, J_4, J_7 are released, J_4 has a level of 3, so it goes first.

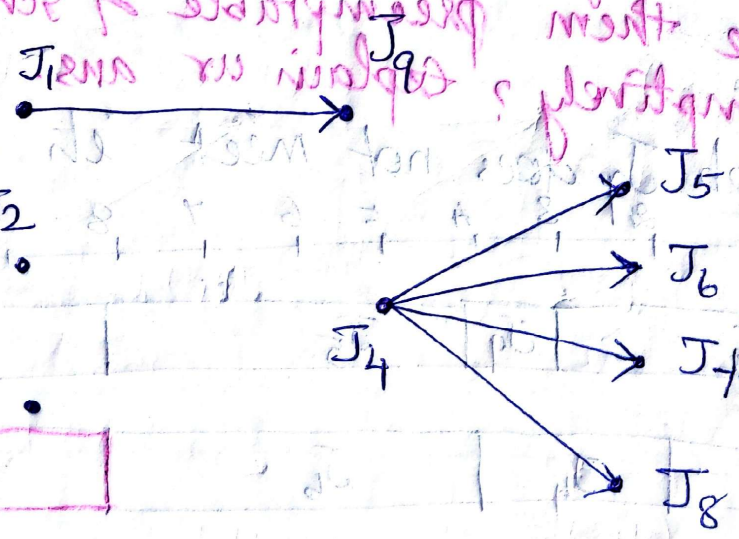
Scanned with ACE Scanner



Problem 3 :- A s/m contains 9 nonpreemptable jobs named J_i for $i = 1, 2 \ldots 9$. Their execution times are 12. J_1 is the immediate predecessor of J_9, and J_4 is the immediate predecessor of J_5, J_6, J_7 and J_8. There is no other precedence constraints. For all the jobs, J_i has a higher priority than J_k if $i < k$.

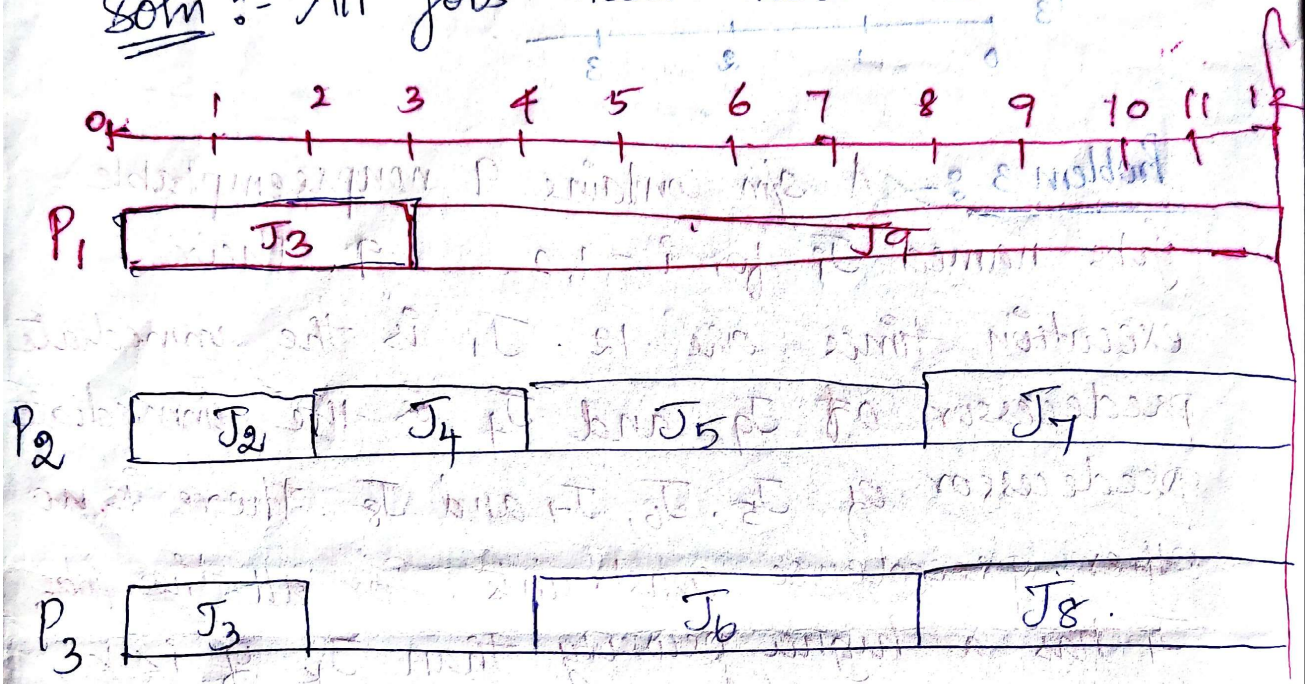
Solu

a) Draw the precedence graph of the jobs



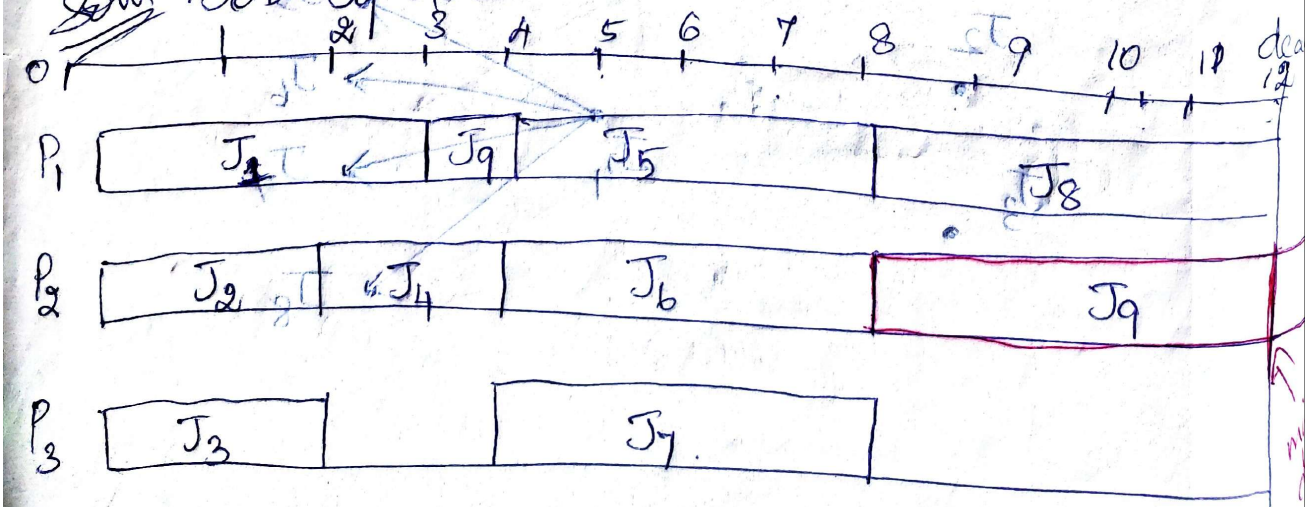
b) Can the jobs meet their deadlines if they are scheduled on the processor? Explain your answer.

Soln :- All jobs meet their deadline



c) Can the jobs meet their deadlines if we make them preemptable & schedule them preemptively? Explain ur ans.

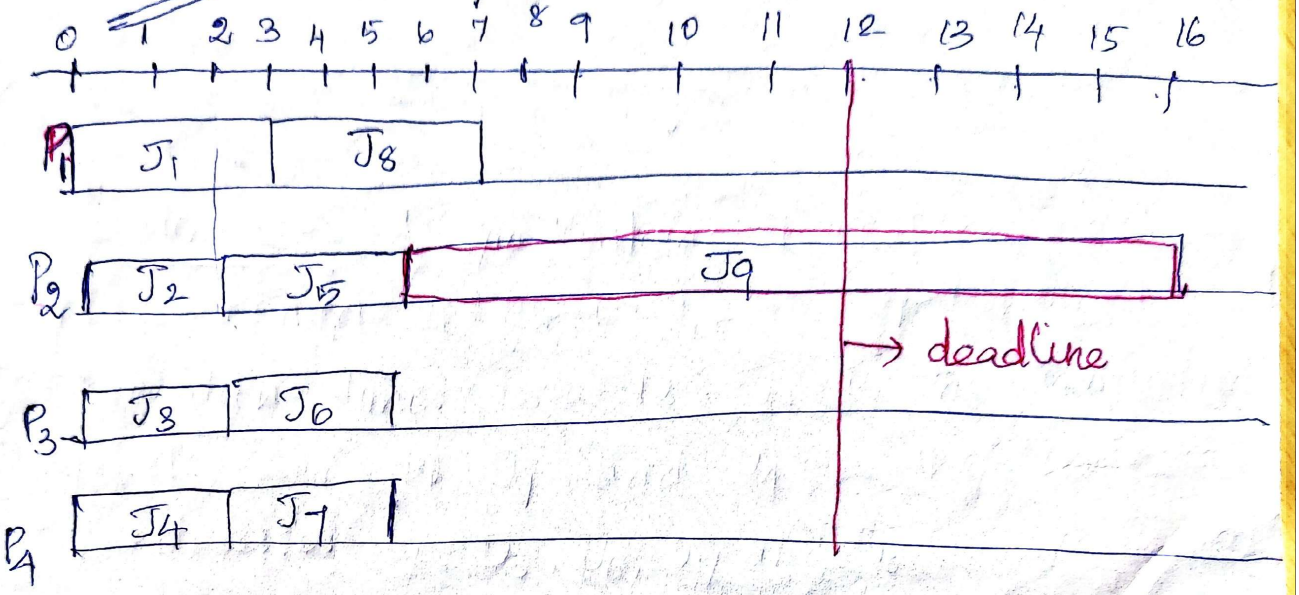
Soln : Job J_9 does not meet its deadline



Scanned with ACE Scanner

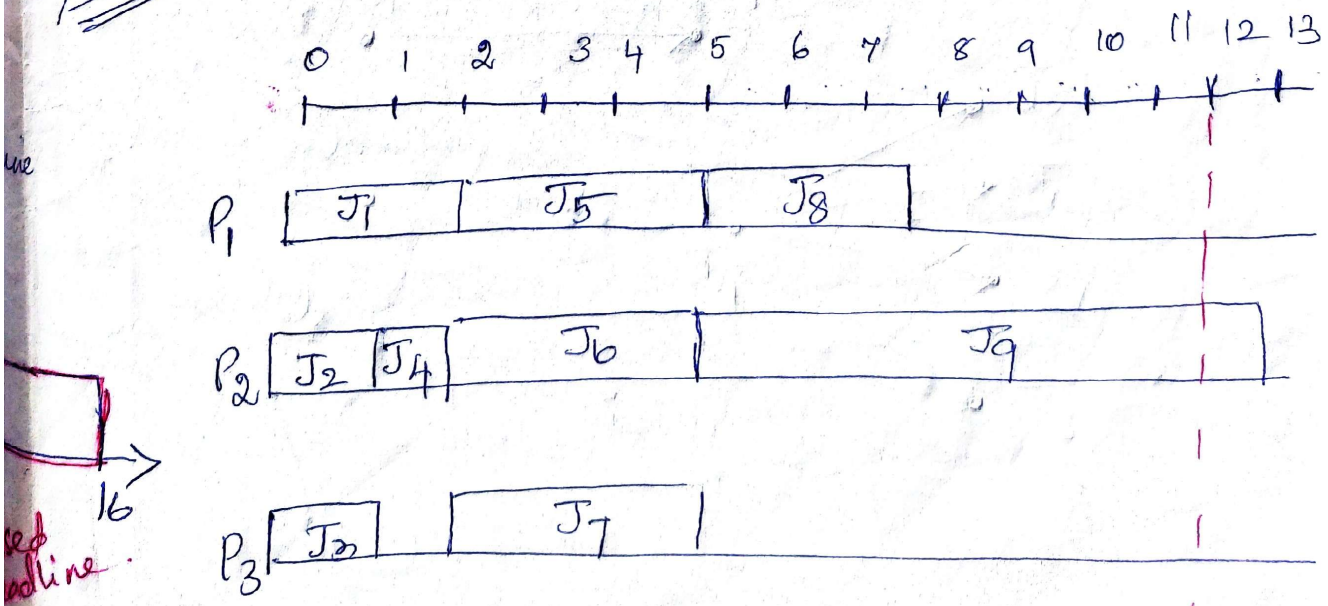
d) Can the jobs meet their deadline, if they are scheduled non-pre-emptively on 4 processors? Explain ur answer.

Soln Job J_9 does not meet its deadline



e) Suppose that due to an improvement of the three processors, the execution time of every job is reduced by 1. Can the jobs meet their deadlines? Explain your answer?

Soln Job J_9 does not meet its deadline



I/o Intensive S/m Design :-

* I/o devices.
* processing the data locally by shipping the data over the n/w.

* Inventory the required I/o devices.
* I/o devices that do not require local processing may be attached to the n/w with simplest available Interface.
* Determine which devices can share a PE or n/w interface.
* Analyse Communication times to determine whether critical comm may interfere with each other.
* Allocate minimum PE with each I/o devices.
* Design rest using Computation intensive s/m's procedure.

Commputation Intensive s/m design :-

* Consider the process and their deadlines and communication.

* tasks with shortest deadline require own PE element.
* Analyse communication times
* Allocate lower-priority tasks to shared PEs where possible.

Scanned with ACE Scanner

- Power consumption & other requirements.
- reallocating processes.
- Load balancing.

Elevator controller

→ distributed system design exa.

- The components are physically distributed among the elevators and floors of the building & the s/m must meet both hard & soft deadlines

(making sure the elevator stops @ the right pt)

(~~requesting~~ responding to requests for elevator)

Theory

1. Elevator car
   * carries passenger
   * runs up & down the hoistway

'N' - no. of hoistway

'F' - no. of floors

Scanned with ACE Scanner

Summing up the worst-case execution times of the processes.

$$\boxed{t_c = \sum_{\text{process } i} \frac{T_l}{T_i} t_{pi}},$$

$t_{pi} \Rightarrow$ execution time of process P_i.

$T_l \Rightarrow$ LCM of all periods T_i

* We can compute the communication volume over the LCM of all periods.

$$\boxed{n_c = \sum_{\text{process } i} l_i}$$

- This formula computes the total no. of bytes transmitted

Two strategies (efficient s/m for our design)

* I/O intensive s/ms:
  - start with the I/O devices & their associated processing.
* Computation-intensive s/ms:
  - start the process.

Scanned with ACE Scanner

Hardware platform Design, Allocation & Scheduling

hardware

Schedule → allocation of processes to PE.

Hardware platform

* No. of PEs required
* types of all PEs
* No. of networks required
* types (and data rates) of the n/ws.

* To evaluate the platform, allocation & Schedule for processes are need to be constructed.

* allocation & scheduling are driven by System performance Analysis

Computation & communication need of the s/m.

Computational needs

* A lower bound on the computational needs of the s/m can be obtained by.

Scanned with ACE Scanner

MPSoC's and Shared Memory Multiprocessors

* Multiprocessor is parallel processors with a single shared memory.
* MPSoC are used to build complex integrated system.
* Multiprocessors have the highest absolute performance - faster than the fastest uniprocessor.
* Parallel processing system is a single program that runs on multiple processors simultaneously.
* Cluster is a set of computers connected over a LAN that functions as a single large multiprocessor.
* Typical MPSoC is a heterogenous multiprocessor.

1. Memory controller : interfaces with the onboard RAM.
2. DMA : Handles automated transfer of data.
3. USB controller : manages the hardware side.
4. DSP core : Provide hardware acceleration
5. Display : Enables the SoC to drive various display types.
6. Camera : Allows the SoC to interface with a camera.
7. Storage : manages I/O with the various types of storage.
8. Debug : Enables the SoC to be connected to hardware debugging tools through various mechanisms, such as JTAG.

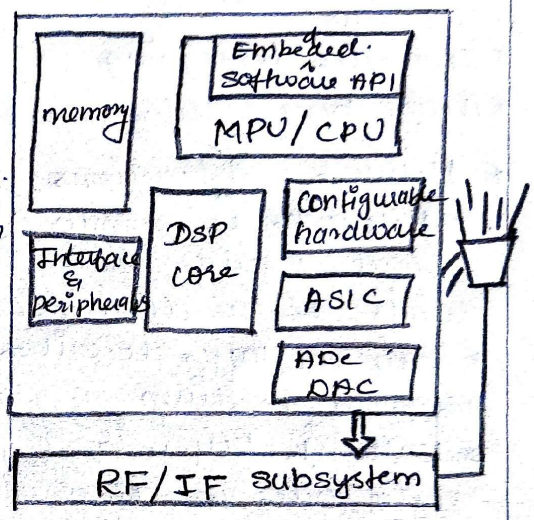


Fig: System on chip

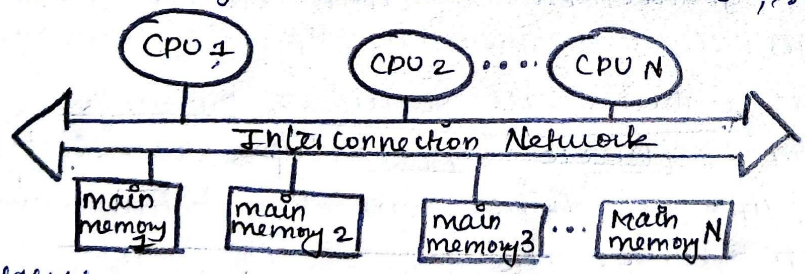


Fig: Shared Memory

* Signal address: offer the programmer a single memory address space that all processors share.
* ~~Sig~~ Message passing: Communicating between multipee processors by explicitly sending and receiving message.
* Heterogenous memory system some memory blocks are accessible by few processors only one or
* Irregular memory structure are often necessary in MPSoCs.

Challenges and opportunities:

* MPSoCs combine the difficult of building complex hardware & software systems.
* Methodology is critical to MPSoC design.
* configurable processors with customized instruction set are one way to improve characteristics.

Design Example : (1) Audio Player

* Audio players are often called MP3 players after the popular audio data format. An MP3 player performs three basic functions : audio storage, audio decompression, and user interface.

* The MP3 player is mainly made up of four parts, which are embedded processor, interface module, storage and external device.

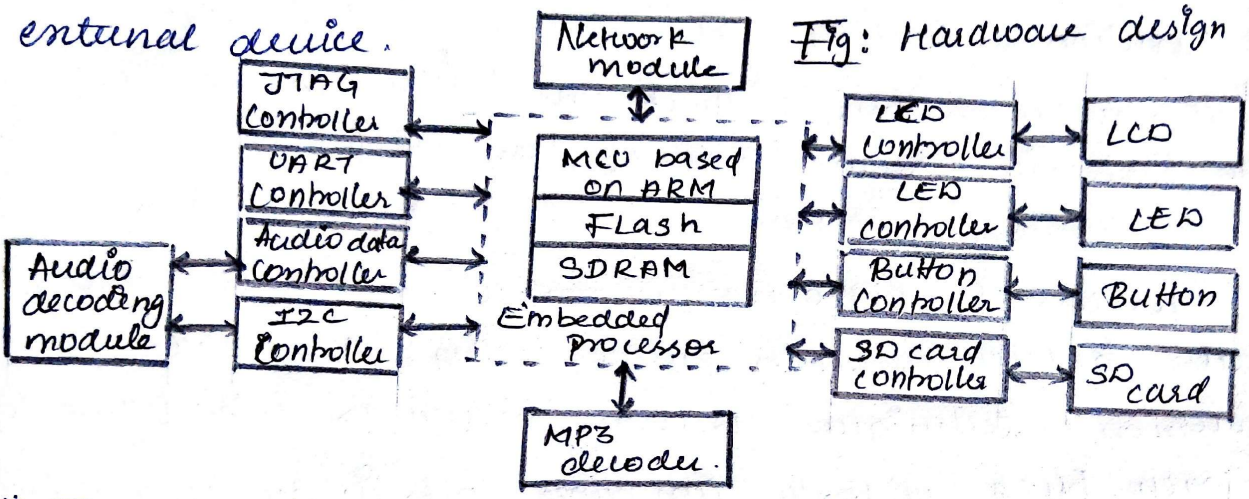


Fig: Hardware design of MP3 player

* The most important hardware module in MP3 player is the decoder module.

* With the rapid development of processor design and related technology, the capability of digital signal processing of RISC approaches DSP level in the last few years.

* Therefore, it is of vital importance to implement MP3 decoder based RISC core. On the other hand, a RISC core can be used as both audio decoding unit and control unit, it is useful for resource constrained system on chip design.

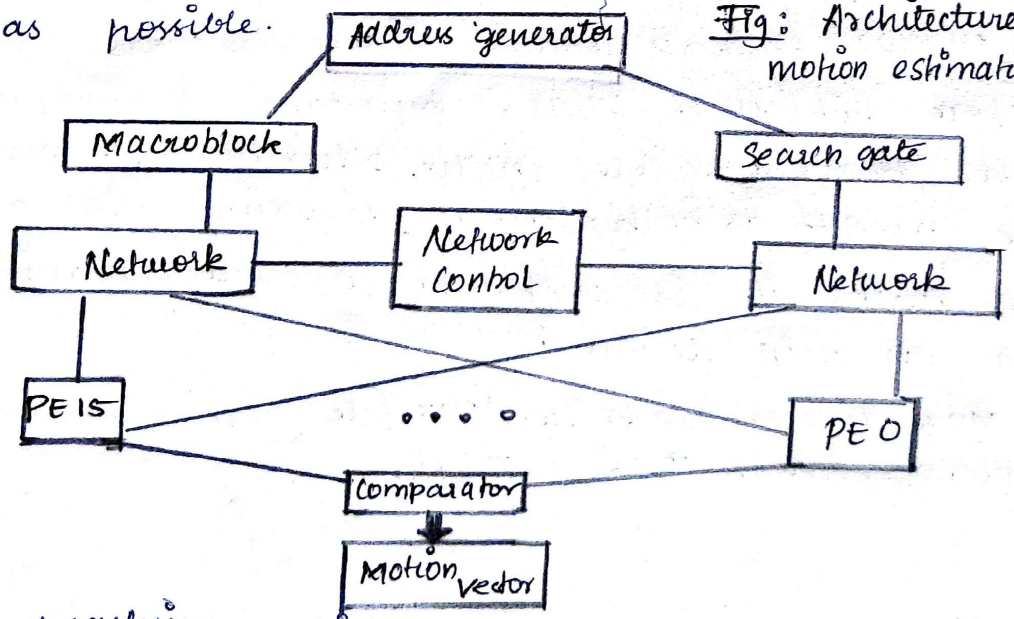
* Our MP3 decoder reads the MP3 file and sends the samples through I2S interface.

Scanned with ACE Scanner

Video Accelerator

* In block based motion estimation, motion estimation is performed on a set of pixels, every frame is divided into blocks of equal size and for each block in the current frame a search is performed in the reference time to find the block resembling the current block the most.
* Block motion estimation is used in digital video compression algorithms so that one frame in the video can be described in terms of the difference between it and another frame.
* The process of video compression using motion estimation is also known as Inter-frame coding. A second compression technique is used, known as intra-frame coding.
* ~~For~~ each block of 16x16 luminance pels in the current frame, a motion vector is computed.
* The motion estimation creates a model by modifying one or more reference frame to match the current frame as closely as possible.

Fig: Architecture for the motion estimation accelerator



* The machine consists of two memory: macroblock memory and search memory.
* It has 16 PEs that perform the difference calculation on a pair of pixels.